Rapport de Travaux Pratiques :MongoDB & Hadoop

**Expéditeur :**Meimoune Sambe C18615

# Introduction Générale

Ce rapport présente les étapes clés des travaux pratiques et dirigés que nous avons effectués sur deux technologies essentielles pour la gestion et l'analyse des "grandes données" (Big Data) : MongoDB et Apache Hadoop.

L'objectif était de comprendre comment installer ces outils, comment les faire fonctionner et comment les utiliser pour manipuler des données.

# Partie I : MongoDB

## Introduction à MongoDB

MongoDB est une base de données particulière, appelée "NoSQL" et de type "orientée document". Contrairement aux bases de données classiques qui rangent les informations dans des tableaux rigides, MongoDB stocke les données sous forme de "documents" (un peu comme des fichiers JSON). Cela rend la base de données très flexible : on peut ajouter des informations différentes à chaque document sans avoir à tout modifier.

MongoDB permet aussi de gérer des données sur plusieurs serveurs, ce qui est crucial pour la scalabilité des applications d'IA.

## Objectifs du TD

Ce TD avait pour but de nous apprendre à :

* Installer MongoDB sur un ordinateur .
* Démarrer le programme principal de MongoDB (le serveur).
* Utiliser le programme pour parler à la base de données (le client).
* Faire les opérations de base sur les données : créer, lire, modifier et supprimer des informations (CRUD).

## Environnement de Travail

Nous avons travaillé sur un ordinateur utilisant le système d'exploitation **Windows**. Nous avons installé MongoDB en décompressant un fichier zip, et nous avons créé les dossiers nécessaires pour que MongoDB puisse stocker ses données.

## Les Étapes Clés et ce que nous avons observé

* 1. **Installation de MongoDB**
     + Nous avons téléchargé un fichier compressé de MongoDB et l'avons décompressé dans un dossier C:\MongoDB. Cela nous a permis d'installer le programme manuellement.
     + Nous avons créé des dossiers spéciaux C:\data et C:\data\db. Ces dossiers sont comme les tiroirs où MongoDB va ranger toutes les informations de la base de données. Il est essentiel de les créer avant de démarrer MongoDB.

## Démarrage du Serveur MongoDB (mongod.exe)

* + - Nous avons lancé le programme mongod.exe depuis l'invite de commande (la boîte noire).
    - **Observation :** Le serveur a affiché beaucoup de texte, puis a indiqué qu'il était prêt à recevoir des connexions sur le port 27017. Cela signifie qu'il fonctionnait correctement et attendait nos commandes.

## Démarrage du Client MongoDB (mongo.exe)

* + - Nous avons ouvert une deuxième invite de commande et lancé le programme

mongo.exe.

* + - **Observation :** Le client s'est connecté au serveur. L'invite de commande a changé pour un signe > ce qui veut dire qu'on pouvait commencer à taper des requêtes pour interroger la base de données.

## Utiliser les Commandes et l'Aide

* + - Nous avons appris à utiliser la commande help pour voir toutes les options disponibles, et commande.help() pour avoir plus de détails sur une commande spécifique. C'est pratique pour apprendre à utiliser MongoDB sans tout connaître par cœur.

## Exercice 01 : Opérations de base (CRUD)

Cet exercice nous a permis de manipuler concrètement les données.

* + - **a. Créer une base de données (use info) :** Nous avons tapé use info pour créer une base de données appelée "info" et pour commencer à travailler dedans. db nous a confirmé que nous étions bien dans cette base de données.
    - **b. Insérer un premier document (db.produits.insertOne()) :** Nous avons ajouté une information sur un "Macbook Pro" dans une collection appelée produits. Une collection est comme un dossier qui contient plusieurs documents.
    - **c. Insérer d'autres documents :** Nous avons ajouté deux autres informations, un "DELL" et un "Thinkpad X230". L'information sur le "Thinkpad X230" avait un détail en plus (ultrabook: true), ce qui montre bien que MongoDB n'oblige pas toutes les informations à avoir la même structure.

## d. Rechercher des informations (find) :

* + - * db.produits.find() : Pour voir toutes les informations dans le dossier

produits.

* + - * db.produits.findOne() : Pour voir la première information trouvée.
      * **Recherche par ID :** Nous avons cherché l'identifiant unique du "Thinkpad" et l'avons utilisé pour retrouver son information spécifique. C'est comme le numéro de série d'un produit.
      * **Recherche par prix :** db.produits.find({prix: {$gt: 13723}})

pour trouver les produits dont le prix est plus grand que 13723.

* + - * **Recherche par caractéristique :** db.produits.findOne({ultrabook: true}) pour trouver le premier produit qui a la caractéristique "ultrabook".
      * **Recherche par mot-clé :** db.produits.findOne({nom: /Macbook/i}) pour trouver les produits dont le nom contient "Macbook" (le "i" veut dire qu'on ne fait pas la différence entre majuscules et minuscules).

## Conclusion de la Partie MongoDB

Ce TD a été une bonne introduction à MongoDB. Nous avons vu que c'est une base de données facile à installer et très flexible. Elle permet de stocker des données qui n'ont pas forcément toutes la même forme, ce qui est très pratique en IA où les données sont souvent variées et peuvent évoluer.

# Partie II : TP Hadoop

## Introduction à Hadoop

Apache Hadoop est un outil très important dans le monde du Big Data. C'est un ensemble de programmes qui permet de stocker et de traiter d'énormes quantités de données en les répartissant sur plusieurs ordinateurs. En Intelligence Artificielle, quand on a des données gigantesques (par exemple, des milliards d'images ou des téraoctets de texte), un seul ordinateur ne suffit pas. Hadoop vient à la rescousse avec deux composants principaux : **HDFS** (un système de fichiers spécial pour ranger les données sur plusieurs ordinateurs) et **YARN** (qui gère les ressources et les programmes qui vont travailler sur ces données).

Hadoop est donc essentiel si on veut travailler avec de très grandes bases de données en IA.

## Objectifs du TP

Ce TP avait pour but de nous apprendre à :

* Préparer notre ordinateur (VM Ubuntu) avec les programmes nécessaires comme Java et SSH.
* Installer et configurer Hadoop sur notre machine, en faisant comme si c'était un mini-réseau d'ordinateurs (on appelle ça le mode "pseudo-distribué").
* Comprendre les fichiers qui configurent Hadoop et comment les ajuster.
* Démarrer, vérifier et arrêter les différentes parties de Hadoop.
* Lancer un petit programme de test pour s'assurer que tout fonctionne.

## Environnement de Travail

Nous avons utilisé une **machine virtuelle Ubuntu Desktop 24.04 LTS** .

## Les Étapes Clés et ce que nous avons observé

* 1. **Programmes Prérequis**
     + **Java (OpenJDK 8) :** Nous avons installé Java 8 (sudo apt install

openjdk-8-jdk -y), car Hadoop a besoin de Java pour fonctionner. C'est comme le moteur qui fait tourner Hadoop.

* + - **SSH :** Nous avons installé le programme SSH (sudo apt install openssh-server -y) et l'avons configuré pour qu'il puisse se connecter à

lui-même sur notre machine **sans demander de mot de passe**. C'est très important pour Hadoop, car il a besoin de cette "porte automatique" pour lancer ses propres programmes. Pour cela, nous avons créé des "clés" (ssh-keygen -t rsa) et avons copié la clé publique dans un fichier authorized\_keys, en veillant à ne **jamais mettre de mot de passe** quand on nous l'a demandé.

## Installation d'Apache Hadoop

* + - Nous avons téléchargé le fichier d'installation de Hadoop (hadoop-3.2.1.tar.gz) et l'avons décompressé. Cela a créé un dossier hadoop-3.2.1 contenant tous les fichiers d'Hadoop.

## Configuration d'Hadoop

C'est l'étape où nous avons dit à Hadoop comment fonctionner. Nous avons modifié des fichiers texte.

* + - **workers :** Nous avons mis localhost dedans. Cela dit à Hadoop que tous ses composants vont tourner sur notre seule machine.
    - **hadoop-env.sh :** Nous avons dit à Hadoop où trouver Java sur notre système.
    - **core-site.xml :** Nous avons configuré l'adresse principale de HDFS (le système de fichiers de Hadoop) : hdfs://localhost:9000.
    - **hdfs-site.xml :** Nous avons indiqué à Hadoop où il doit stocker ses données temporaires et combien de copies de chaque donnée il doit faire (une seule copie ici, car nous n'avons qu'une machine). Nous avons aussi créé ces dossiers.
    - **yarn-site.xml et mapred-site.xml :** Ces fichiers ont été configurés pour que Hadoop puisse gérer correctement les tâches et les programmes (MapReduce) qui vont s'exécuter.

## Optimisations pour les Petits Ordinateurs (VM)

Cette étape est très importante. Hadoop est fait pour de gros serveurs, donc il faut lui dire d'utiliser moins de ressources pour qu'il puisse tourner sur notre machine virtuelle.

* + - Nous avons modifié yarn-site.xml et mapred-site.xml pour **limiter la quantité de mémoire et de puissance de calcul** que Hadoop peut utiliser. Sans

cela, Hadoop pourrait "planter" car il demanderait plus de ressources que notre VM n'en a.

## Démarrage et Tests

* + - **Démarrage :** Nous avons d'abord "préparé" HDFS (bin/hdfs namenode

-format), puis lancé les différents services de Hadoop (sbin/start-dfs.sh et

sbin/start-yarn.sh).

* + - **Vérification (jps) :** La commande jps nous a permis de voir si tous les programmes d'Hadoop fonctionnaient bien. Nous avons dû voir apparaître plusieurs noms comme NameNode, DataNode, ResourceManager, etc.
    - **Interfaces Web :** Nous avons pu vérifier l'état d'Hadoop via des pages web dans le navigateur de la VM (sur <http://localhost:9870/>et [http://localhost:8088/).](http://localhost:8088/)) C'est pratique pour voir les données et les tâches en cours.
    - **Test de calcul (Pi) :** Nous avons lancé un programme de test qui calcule le nombre Pi (bin/hadoop jar ... pi 8 14). Ce programme utilise Hadoop pour faire un calcul distribué, ce qui nous a confirmé que tout l'environnement fonctionnait.
    - **Arrêt :** Nous avons arrêté les services avec sbin/stop-yarn.sh et

sbin/stop-dfs.sh.

* + - **Script de réinitialisation (reset.sh) :** Nous avons créé un petit programme (reset.sh) qui permet d'arrêter, de nettoyer et de redémarrer Hadoop facilement en cas de problème.

## Conclusion de la Partie Hadoop

Ce TP nous a donné une bonne première expérience avec Hadoop. Nous avons appris à l'installer et à le configurer pour qu'il tourne sur une seule machine. Comprendre comment Hadoop gère les données et les tâches est une compétence très utile pour quiconque veut travailler sur des projets d'IA qui nécessitent de traiter des volumes de données importants.

# Conclusion Générale

Ces travaux pratiques nous ont permis de toucher du doigt deux technologies très importantes dans le monde du Big Data et de l'Intelligence Artificielle. MongoDB nous offre une façon flexible de stocker des données variées, tandis que Hadoop nous permet de traiter ces données à une échelle massive. La maîtrise de ces outils est essentielle pour les futurs professionnels de l'IA, car elle leur permettra de construire des systèmes robustes et capables de gérer les quantités de données toujours croissantes nécessaires à l'apprentissage automatique et à l'analyse.

Partie 3

**1. Introduction :**

Ce rapport présente la mise en œuvre d'un programme utilisant Apache Spark (avec PySpark) pour identifier les amis communs entre deux utilisateurs au sein d'un graphe social. Le projet vise à appliquer les concepts de MapReduce, la distribution des fonctionnalités et l'analyse graphique offerts par Spark.

**2. Description du Jeu de Données** :

Le jeu de données d'entrée est un fichier texte où chaque ligne représente un utilisateur et ses amis. Le format est le suivant : <user\_id> <Nom> <friend\_id1>,<friend\_id2>,... Par exemple, 1 2,3,4 signifie que l'utilisateur 1 est ami avec les utilisateurs 2, 3 et 4. Un point crucial est que les amitiés sont mutuelles (si A est ami avec B, alors B est ami avec A).

Pour les tests, le fichier friend.txt a été utilisé, contenant les données suivantes (extraites du notebook fourni) :

* 1 Sidi 2,3,4
* 2 Mohamed 1,3,5
* 3 Aicha 1,2,4,6
* 4 Ahmed 1,3
* 5 Leila 2
* 6 Aisha 3

**3. Environnement de Développement** Le développement et l'exécution du programme ont été réalisés dans un environnement Google Colab. PySpark a été installé et configuré pour permettre l'utilisation des fonctionnalités de Spark, comme indiqué dans la première cellule du notebook.

**4. Implémentation Détaillée**

Les étapes suivantes ont été implémentées en utilisant PySpark :

* **Chargement des Données :** Le fichier texte a été chargé dans un RDD (Resilient Distributed Dataset) Spark. Chaque ligne du fichier est lue comme un enregistrement distinct.

Python

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("AmisCommuns").getOrCreate()

data = spark.read.text("/content/friend.txt") # Lecture du fichier

lines = data.rdd.map(lambda r: r[0]) # Extraction des lignes brutes en RDD

Cette étape correspond à la tâche de "Charger les données à l’aide de PySpark".

* **Génération des Couples d'Amis Possibles :** Une fonction pairs a été définie pour traiter chaque ligne du RDD. Pour chaque utilisateur, elle génère des paires (utilisateur\_courant, ami) en s'assurant que la paire est triée (min, max) pour éviter les doublons comme (1,2) et (2,1). Chaque paire est associée à l'ensemble des amis de l'utilisateur d'origine.

Python

def pairs(line):

parts = line.split(" ")

if len(parts) < 3: return [] # Gestion des lignes mal formées

user = parts[0]

friends = parts[2].split(",")

result = []

for friend in friends:

pair = tuple(sorted([user, friend])) # Tri des utilisateurs pour normalisation

result.append((pair, set(friends))) # Association de la paire avec l'ensemble des amis

return result

pairs\_rdd = lines.flatMap(pairs)

Cette implémentation satisfait la tâche de "Générer tous les couples d’amis possibles, en conservant chaque couple trié (min, max) pour éviter les doublons".

* **Calcul des Amis Communs :** Le pairs\_rdd est ensuite traité pour trouver les amis communs. reduceByKey est utilisé pour regrouper les valeurs (ensembles d'amis) par la même clé (paire d'amis normalisée) et appliquer une opération d'intersection sur ces ensembles. Cela permet de trouver les amis présents dans les listes des deux membres de la paire.

Python

mutual\_friends\_rdd = pairs\_rdd.reduceByKey(lambda s1, s2: s1.intersection(s2))

Cette étape réalise la tâche de "Calculer pour chaque paire d’amis la liste des amis communs".

* **Filtrage et Affichage des Amis Communs Spécifiques :** La dernière tâche consiste à filtrer les résultats pour la paire spécifique "Mohamed (ID 2) et Sidi (ID 1)". La paire cible est normalisée en ('1', '2'). Le résultat est collecté et formaté selon la spécification 1<Nom1>2<Nom2>liste\_amis\_communs.

Python

target\_pair = ('1', '2') # Paire cible normalisée pour Sidi (ID 1) et Mohamed (ID 2)

result = mutual\_friends\_rdd.filter(lambda x: x[0] == target\_pair).collect()

if result:

amis\_communs = ",".join(sorted(list(result[0][1])))

print(f"{target\_pair[0]}<Sidi>{target\_pair[1]}<Mohamed>{amis\_communs}")

else:

print("Aucun ami commun trouvé entre Sidi et Mohamed.")

Cette section répond à l'exigence d'implémenter une fonction ou un script PySpark qui "filtre et affiche la liste des amis communs entre les deux utilisateurs spécifiés : Mohamed (ID 2) et Sidi (ID 1)".

**5. Résultats**

Pour la paire ciblée (Sidi - ID 1 et Mohamed - ID 2), le programme a identifié les amis communs. L'utilisateur 1 (Sidi) est ami avec les utilisateurs 2, 3 et 4. L'utilisateur 2 (Mohamed) est ami avec les utilisateurs 1, 3 et 5. L'ami commun est l'utilisateur 3.

L'affichage final généré par le programme est le suivant : 1<Sidi>2<Mohamed>3

**6. Conclusion**

Ce projet a démontré l'utilisation efficace d'Apache Spark pour l'identification scalable des amis communs dans un graphe social. En tirant parti des capacités de traitement distribué de PySpark, la solution offre une méthode robuste pour extraire des informations pertinentes de vastes ensembles de données relationnelles.