Aide-Mémoire : Commandes d'Exécution

Projet d'Analyse de Données Massives

par Fake News Detection (MapReduce, Pig, Hive, Spark)
26 octobre 2025

Résumé

Ce document résume les commandes d'exécution utilisées pour les quatre technologies principales de ce projet. Les commandes sont exécutées sur l'environnement Cloudera VM (CentOS) ou sur Google Colab, comme spécifié.

1. Mission 2 : MapReduce (Java)

L'implémentation de référence. Le job est lancé par l'utilisateur `cloudera`.

1.1. 1. Compilation du code Java Cette commande compile les 3 fichiers Java en `.class`, en utilisant les bibliothèques Hadoop (classpath).

```
# Devenir l'utilisateur cloudera
sudo su - cloudera

# Aller au dossier de travail (ex: mr_wordcount)
cd ~/mr_wordcount
mkdir classes

# Compiler
javac -cp $(hadoop classpath) -d classes \
    FakeNewsMapper.java FakeNewsReducer.java FakeNewsDriver.java
```

Listing 1: Compilation Java

1.2. 2. Création du fichier JAR Cette commande empaquète les fichiers `.class` compilés dans un seul fichier JAR exécutable.

```
# Créer le JAR
jar -cvf FakeNews.jar -C classes .
```

Listing 2: Création du JAR

1.3. 3. Lancement du Job MapReduce

La commande `hadoop jar` exécute le job sur HDFS.

```
# S'assurer que le dossier de sortie n'existe pas
hdfs dfs -rm -r /user/cloudera/fake_news_project/output_mr

# Lancer le job
hadoop jar FakeNews.jar FakeNewsDriver \
    /user/cloudera/fake_news_project/articles.tsv \
    /user/cloudera/fake_news_project/output_mr
```

Listing 3: Lancement du Job Hadoop

1.4. 4. Vérification des résultats

Afficher les 20 premières lignes du fichier de sortie généré par le Reducer.

```
hdfs dfs -cat /user/cloudera/fake_news_project/output_mr/part-r-00000
    | head -n 20
```

Listing 4: Vérification HDFS

2. Mission 3 : Apache Pig (PigLatin)

Pig abstrait le MapReduce en un script de dataflow.

2.1. 1. Exécution du script Pig La commande `pig` prend le script `.pig` en argument et le convertit en un ou plusieurs jobs MapReduce.

```
# S'assurer d'être l'utilisateur 'cloudera'
sudo su - cloudera

# Lancer le script Pig (ex: analyze.pig)
pig analyze.pig
```

Listing 5: Exécution Pig

3. Mission 3 : Apache Hive (HiveQL)

Hive abstrait le MapReduce en requêtes SQL.

3.1. 1. Préparation (si nécessaire) Nous devons nous assurer que nos données sont dans un dossier HDFS "propre" que Hive peut lire.

```
# Créer un dossier propre pour Hive
hdfs dfs -mkdir -p /user/cloudera/hive_input

# Y déplacer les données (si ce n'est pas déjà fait)
hdfs dfs -mv /user/cloudera/fake_news_project/articles.tsv \
    /user/cloudera/hive_input/
```

Listing 6: Préparation HDFS pour Hive

3.2. 2. Lancement du Shell et Requêtes Nous lançons le shell Hive, puis exécutons nos commandes SQL.

```
# Lancer le shell Hive (en tant que root ou cloudera)
hive

# -- Commandes à l'intérieur du shell 'hive>' --

# 1. (Nettoyage) Supprimer l'ancienne table si elle existe
DROP TABLE IF EXISTS articles;

# 2. Créer la table externe pointant vers nos données HDFS
CREATE EXTERNAL TABLE articles (
    label STRING,
    text_body STRING
```

```
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
LOCATION '/user/cloudera/hive_input/';

# 3. Exécuter la requête d'analyse (WordCount complexe)
SELECT
    label,
    cleaned_word,
    COUNT(1) AS word_count
FROM (
    SELECT
        label,
        LOWER(regexp_replace(word, '[^a-zA-Z]', '')) AS cleaned_word
    FROM
        articles
        LATERAL VIEW explode(split(text_body, ' ')) exploded_table AS
            word
    WHERE
        label = 'TRUE' OR label = 'FAKE'
) subquery
WHERE
    LENGTH(cleaned_word) > 4
GROUP BY
    label, cleaned_word
ORDER BY
    word_count DESC
LIMIT 100;

# 4. Quitter le shell
exit;
```

Listing 7: Commandes HiveQL

4. Mission 4 : Apache Spark (PySpark sur Colab)

Implémentation moderne utilisant les RDD sur Google Colab. L'exécution se fait en cliquant sur "Exécuter la cellule".

4.1. 1. Cellule 1 : Installation

Installation de PySpark dans l'environnement Colab.

```
!pip install pyspark
```

Listing 8: Installation PySpark

4.2. 2. Cellule 2 : Importation des données

Importation du fichier `articles.tsv` dans l'environnement Colab.

```
from google.colab import files
uploaded = files.upload()
print("Fichier importe !")
```

Listing 9: Importation des données

4.3. 3. Cellule 3 : Job Spark (Code PySpark)

C'est le script principal qui définit les RDD et exécute le comptage. L'exécution est lancée via le bouton "Play" de la cellule.

```
import re
from pyspark.sql import SparkSession

# Initialisation
spark = SparkSession.builder.appName("FakeNewsWordCount").getOrCreate
    ()
sc = spark.sparkContext

# RDD 1: Charger les données
baseRDD = sc.textFile("articles.tsv")

# RDD 2: Map (fonction pour mapper les lignes)
def map_line_to_words(line):
    # Logique de parsing et nettoyage
    # Exemple (à compléter selon votre script original) :
    try:
        parts = line.split('\t')
        if len(parts) < 2:
            return [] # Ignorer les lignes mal formées
        label = parts[0]
        text_body = parts[1]

        # Nettoyage et tokenisation des mots
        words = re.findall(r'\b\w+\b', text_body.lower())
```

```
24
25          # Filtrer les mots courts et non-alphabétiques (comme dans
                Hive)
26          cleaned_words = [word for word in words if len(word) > 4 and
                word.isalpha()]
27
28          # Retourne une liste de ((label, word), 1)
29          return [((label, word), 1) for word in cleaned_words if label
                in ['TRUE', 'FAKE']]
30      except:
31          return [] # Gérer les erreurs de ligne
32
33  pairedRDD = baseRDD.flatMap(map_line_to_words)
34
35  # RDD 3: Reduce
36  countsRDD = pairedRDD.reduceByKey(lambda a, b: a + b)
37
38  # RDD 4: Trier
39  sortedRDD = countsRDD.sortBy(lambda pair: pair[1], ascending=False)
40
41  # Action: Lancer le calcul et afficher
42  top_100_words = sortedRDD.take(100)
43  for (key, count) in top_100_words:
44      print(f"{key[0]}\t{key[1]}\t{count}")
45
46  # Arrêter la session Spark
47  spark.stop()
```

Listing 10: Script PySpark Principal