Atelier - Hadoop MapReduce vs PySpark

✅ Étape 1 : Installation

```
# Mise à jour
!apt-get update

# Installer Java
!apt-get install openjdk-11-jdk -y

# Télécharger et extraire Hadoop
!wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
!tar -xzvf hadoop-3.3.6.tar.gz

# Télécharger et extraire Spark
!wget https://downloads.apache.org/spark/spark-3.5.6/spark-3.5.6-bin-hadoop3.tgz
!tar -xzvf spark-3.5.6-bin-hadoop3.tgz
```

✅ Étape 2 : Configurer les variables

```
import os

os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["HADOOP_HOME"] = "/content/hadoop-3.3.6"
os.environ["SPARK_HOME"] = "/content/spark-3.5.6-bin-hadoop3"

os.environ["PATH"] += f":{os.environ['HADOOP_HOME']}/bin:
{os.environ['HADOOP_HOME']}/sbin:{os.environ['SPARK_HOME']}/bin"

!java -version
!hadoop version
!spark-shell --version
```

✅ Étape 3 : Créer le fichier de test

```
!wget  https://zenodo.org/records/8196385/files/HDFS_v2.zip
!unzip -o -q HDFS_v2.zip
!cat README.md
```

```
# Exemple simple
!cp node_logs/hadoop-hdfs-datanode-mesos-01.log input.txt

# Vérifier le contenu
!cat input.txt
```

✅ Étape 4 : Scripts MapReduce Hadoop

➤ Mapper

```
%%writefile mapper.py
#!/usr/bin/env python
import sys
import re

for line in sys.stdin:
    # Nettoyer la ligne et la découper en mots
    line = line.strip()
    words = re.split(r'\s+', line)

    # Émettre une paire (mot, 1) pour chaque mot
    for word in words:
        if word: # S'assurer que le mot n'est pas vide
            print(f'{word}\t1')
```

➤ Reducer

```
%%writefile reducer.py
#!/usr/bin/env python
import sys

current_word = None
current_count = 0
word = None

# Lire l'entrée (qui est déjà triée par le framework Hadoop)
for line in sys.stdin:
    line = line.strip()
```

```
    # Tenter de séparer le mot de son compte
    try:
        word, count_str = line.split('\t', 1)
        count = int(count_str)
    except ValueError:
        # Ignorer les lignes mal formées
        continue

    # Si c'est le même mot que le précédent, on incrémente le compteur
    if current_word == word:
        current_count += count
    else:
        # Sinon (si c'est un nouveau mot), on affiche le résultat du mot précédent
        if current_word:
            print(f'{current_word}\t{current_count}')

        # Et on réinitialise les compteurs pour le nouveau mot
        current_count = count
        current_word = word

# Ne pas oublier d'afficher le tout dernier mot après la fin de la boucle !
if current_word == word:
    print(f'{current_word}\t{current_count}')
```

✅ Étape 5 : Exécuter avec Hadoop Streaming

```
!chmod +x mapper.py
!chmod +x reducer.py
```

```
%%sh

# Nettoyer l'ancien output s'il existe
rm -rf output_hadoop

echo "--- DÉBUT DU JOB HADOOP STREAMING ---"

# Enregistrer le temps de début
start=$(date +%s)

# Lancer Hadoop Streaming
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
  -input input.txt \
  -output output_hadoop \
  -mapper mapper.py \
  -reducer reducer.py
```

```
# Enregistrer le temps de fin
end=$(date +%s)

echo "--- FIN DU JOB HADOOP STREAMING ---"
# Calculer et afficher la durée
echo "Durée d'exécution : $(($end - $start)) secondes"


# Afficher les 10 premières lignes du résultat
echo -e "\n--- Résultat Hadoop (10 premières lignes) ---"
head -n 10 output_hadoop/part-00000
```

✅ Étape 6 : Exécuter avec PySpark

```
from pyspark.sql import SparkSession
import time
import re

# Initialisation de la session Spark
spark = SparkSession.builder.appName("WordCountSpark").getOrCreate()
sc = spark.sparkContext

print("--- DÉBUT DU JOB SPARK ---")
start_time_spark = time.time()

# Logique Spark (similaire à la version précédente mais sur input.txt)
word_counts_spark = sc.textFile("input.txt") \
                      .flatMap(lambda line: re.split(r'\s+', line.strip())) \
                      .filter(lambda word: word) \
                      .map(lambda word: (word, 1)) \
                      .reduceByKey(lambda a, b: a + b)

# Collecter les résultats pour forcer l'exécution
results_spark = word_counts_spark.take(10)
# Les transformations (comme split, explode, groupBy) sont paresseuses (lazy) :
elles ne s'exécutent pas tout de suite.
# Elles ne sont matérialisées qu'au moment où une action est demandée (show(),
collect(), count(), take()...).

end_time_spark = time.time()
print(f"--- FIN DU JOB SPARK (terminé en {end_time_spark - start_time_spark:.4f}
secondes) ---")


# Afficher les 10 premiers résultats
```

```
print("\n--- Résultat Spark (10 premiers éléments) ---")
for word, count in results_spark:
    print(f"{word}\t{count}")

# Arrêter la session Spark
spark.stop()
```
