Rapport de LabWork 4 - Installation et Configuration Spark

Réalise par : LAABID ABDESSAMAD

1. Introduction

Ce rapport présente les travaux réalisés dans le cadre du LabWork 4 portant sur l'installation et la configuration de Spark sur une machine Windows. L'objectif principal était d'installer Apache Spark, de vérifier son bon fonctionnement, puis d'implémenter l'exemple du compteur de mots (WordCount) en Python. Une analyse de données météorologiques a également été réalisée, similaire à celle du LabWork 2, mais cette fois en utilisant le framework Spark.

2. Environnement de travail

2.1 Prérequis logiciels

- **Système d'exploitation** : Windows 11
- **Java** : Version 21.0.7
- **Python** : Version 3.12.8
- **Apache Spark** : Version 3.5.5
- **Hadoop** : Version 3 (via la distribution Spark)

2.2 Vérification de l'installation de Java

```
C:\Users\aplus>java -version
java version "21.0.7" 2025-04-15 LTS
Java(TM) SE Runtime Environment (build 21.0.7+8-LTS-245)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.7+8-LTS-245, mixed mode, sharing)
```

2.3 Vérification de l'installation de Python

```
C:\Users\aplus>python --version
Python 3.12.8
```

3. Installation d'Apache Spark

3.1 Téléchargement

Le fichier d'installation de Spark a été téléchargé depuis le site officiel d'Apache Spark (https://dlcdn.apache.org/spark/spark-3.5.5/spark-3.5.5-bin-hadoop3.tgz). La version 3.5.5 pour Hadoop 3 a été sélectionnée.

3.2 Décompression

Après téléchargement, le fichier a été décompressé

4. Configuration de l'environnement

4.1 Configuration des variables d'environnement

Les variables d'environnement suivantes ont été configurées dans le système :

- **JAVA_HOME** : `C:\Program Files\Java\jdk-21`
- **SPARK_HOME** : `C:\Users\aplus\Downloads\spark-3.5.5-bin-hadoop3\spark-3.5.5-bin-hadoop3`
- **HADOOP_HOME** : `C:\Users\aplus\Downloads\spark-3.5.5-bin-hadoop3\spark-3.5.5-bin-hadoop3`
- **PATH** : Ajout des chemins `%JAVA_HOME%\bin`, `%SPARK_HOME%\bin`, et `%HADOOP_HOME%\bin`

4.2 Configuration des Winutils

Pour le bon fonctionnement de Spark sur Windows, les fichiers `winutils.exe` et `hadoop.dll` ont été téléchargés depuis le dépôt GitHub de Steve Loughran (https://github.com/steveloughran/winutils) et placés dans le répertoire `%SPARK_HOME%\bin`.

5. Test de l'installation

5.1 Lancement du shell Spark

Le shell Spark a été lancé via la commande :

```
C:\Users\aplus>spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://aplus:4040
Spark context available as 'sc' (master = local[*], app id = local-1746913747577).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.5
      /_/

Using Scala version 2.12.18 (Java HotSpot(TM) 64-Bit Server VM, Java 21.0.7)
Type in expressions to have them evaluated.
Type :help for more information.
```

.

6. Implémentation du WordCount

6.1 Préparation des données

Un fichier `pg50004.txt` contenant un livre électronique en anglais a été utilisé comme jeu de données d'entrée.

6.2 Implémentation en Scala (via spark-shell)

```
scala> val textFile = sc.textFile("C:/Users/aplus/Documents/tp4/pg50004.txt")
textFile: org.apache.spark.rdd.RDD[String] = C:/Users/aplus/Documents/tp4/pg50004.txt MapPartitionsRDD[1] at textFile at <console>:23

scala> val counts = textFile.flatMap(line => line.split(" ")
     | ).map(word => (word, 1)
     | ).reduceByKey(_+_)
counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:25

scala> counts.take(5).foreach(println)
(adornment,,1)
(intimately,1)
(House,3)
(bone,1)
(nobleman,2)
```

7. Analyse des données météorologiques

7.1 Jeu de données

Le même jeu de données météorologiques que celui utilisé dans le LabWork 2 a été utilisé pour cette analyse.

7.2 Implémentation

```
scala> val tempsFile = sc.textFile("C:/Users/aplus/Documents/sda2/Traitement Paralleles/hadoop_tps/tp2/meteosample.txt")
tempsFile: org.apache.spark.rdd.RDD[String] = C:/Users/aplus/Documents/sda2/Traitement Paralleles/hadoop_tps/tp2/meteosample.txt MapPartitionsRDD[6] at text
File at <console>:23

scala> val tempsResult1 = tempsFile.flatMap(line => line.split("\n")
     | ).map(line => {
     | val fields = line.split(" : ")
     | val year = fields(2).toInt
     | val temp = fields(3).toDouble
     | (year, temp)
     | }).reduceByKey(math.max)
tempsResult1: org.apache.spark.rdd.RDD[(Int, Double)] = ShuffledRDD[9] at reduceByKey at <console>:29

scala> tempsResult.take(5).foreach(println)
<console>:23: error: not found: value tempsResult
       tempsResult.take(5).foreach(println)
       ^

scala> tempsResult1.take(5).foreach(println)
(1940,-17.0)
(1998,-22.0)
(2000,-20.0)
(2010,-1.0)
(2002,-16.0)

scala>
```

8. Conclusion

Ce LabWork a permis de mettre en pratique l'installation et la configuration de Spark sur un environnement Windows. L'utilisation de Spark pour le traitement distribué des données s'est révélée efficace, notamment pour des applications comme le comptage de mots et l'analyse de données météorologiques.

Les principales compétences acquises sont :

- L'installation et la configuration d'un environnement Spark sur Windows

- L'utilisation du shell Spark pour exécuter des commandes interactives
- La mise en œuvre de l'algorithme WordCount avec Spark
- L'application de transformations et d'actions Spark pour l'analyse de données structurées
- La comparaison entre l'approche MapReduce classique et l'API Spark pour le traitement de données

L'utilisation de Spark facilite grandement le développement d'applications de traitement de données distribuées, avec une syntaxe plus intuitive et des performances améliorées par rapport à MapReduce classique.