École Centrale de Lyon

MSO 3.1 - Technologies informatiques du Big Data
Option Informatique
Rapport

TP2 – Spark

Élèves :
Thibaud MIQUEL

Enseignant :
Stéphane DERRODE

Table des matières

Table des figures

1 Présentation du TP

1.1 Objectif du TP

L'objectif de ce TP est d'utiliser la bibliothèque pyspark pour coder des requêtes et les exécuter directement sur Hadoop en passant par Docker.

1.2 Présentation de Spark

Spark est une API de programmation parallèle sur des données. L'objet principal de Spark est le RDD : Resilient Distributed Dataset. C'est un dispositif pour traiter une collection de données par des algorithmes parallèles robustes. Un RDD ne contient pas vraiment de données, mais seulement un traitement.

Ce traitement n'est effectué que lorsque cela apparaît nécessaire. On appelle cela l'évaluation paresseuse. D'autre part, Spark fait en sorte que le traitement soit distribué sur le cluster, donc calculé rapidement, et n'échoue pas même si des machines tombent en panne. Spark permet donc d'écrire des traitements complexes composés de plusieurs phases map-reduce.

https ://perso.univ-rennes1.fr/pierre.nerzic/Hadoop/tp4.pdf https ://perso.univ-rennes1.fr/pierre.nerzic/Hadoo slide 8 https ://studylibfr.com/doc/3997831/hadoop—semaine-5 https ://www.bestcours.com/documents/0582- outils-hadoop-pour-le-bigdata.pdf page 60

2 Réponse aux questions

2.1 Partie 2 : The MovieLens Database

Ma requête consiste à renvoyer une liste triée des films sur leur notation moyenne par ordre décroissant.

J'ai construit ma requête sans utiliser la fonction moyenne pour faire toute la fonctionnalité moi même. J'ai donc tout d'abord sommé dans un premier dataframe pris de ratings.csv toutes les notations de chaque film en gardant les id des films comme clé. Puis dans un 2eme dataframe j'ai compté le nombre de fois où chaque film est noté en gardant également comme clé l'id des films. Je joins ensuite ces 2 dataframes sur les id puis je divise la somme des notes par le nombre de fois où le film est noté pour obtenir la note moyenne, enfin je trie cette liste par moyenne de notation en ordre décroissant.

Pour obtenir également le nom du film je joins ce dernier dataframe à un dataframe récupéré via movies.csv sur les id de film.

Voici le résultat de cette requête :

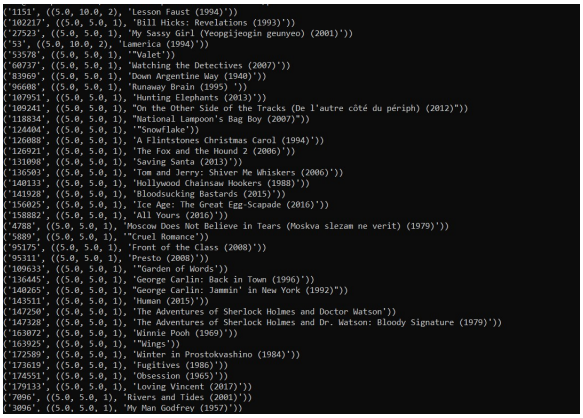

FIGURE 1 – Top résultat de la requête

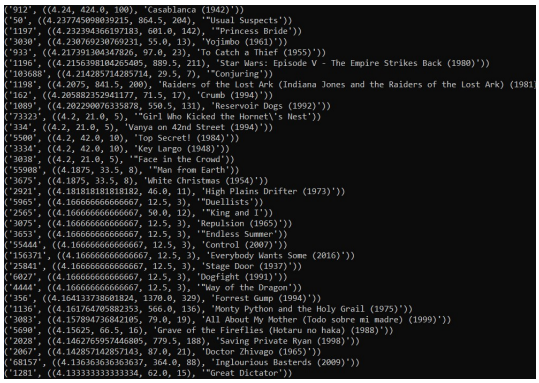

FIGURE 2 – Résultat sous le top malgré le nombre de notation des films

Je me suis rendu compte que ce résultat n'était pas très intéressant car je n'ai pas tenu compte du nombre de fois minimal qu'un film devrait être noté pour apparaître dans ce top. C'est pourquoi dans les résultats ci-dessus il n'y a que très peu de film connu dans les meilleurs résultats.

Dans un 2eme temps j'ai donc rajouté comme argument à donner par l'utilisateur un seuil de nombre de notation minimum pour qu'un film apparaisse dans ce top. La requête que j'ai formulé est donc la suivante : **spark-submit –deploy-mode client –master local[2] Q1_movie.py input/ratings.csv input/movies.csv 15** où 15 le nombre de notation minimum qu'un film doit avoir.

```
('3468', ((4.333333333333333, 78.0, 18), '"Hustler'))
('1204', ((4.3, 193.5, 45), 'Lawrence of Arabia (1962)'))
('246', ((4.293103448275862, 124.5, 29), 'Hoop Dreams (1994)'))
('1235', ((4.288461538461538, 111.5, 26), 'Harold and Maude (1971)'))
('168252', ((4.28, 107.0, 25), 'Logan (2017)'))
('2959', ((4.272935779816514, 931.5, 218), 'Fight Club (1999)'))
('1213', ((4.25, 535.5, 126), 'Goodfellas (1990)'))
('930', ((4.25, 85.0, 20), 'Notorious (1946)'))
('3508', ((4.25, 76.5, 18), '"Outlaw Josey Wales'))
('912', ((4.24, 424.0, 100), 'Casablanca (1942)'))
('50', ((4.237745098039215, 864.5, 204), '"Usual Suspects'))
('1197', ((4.232394366197183, 601.0, 142), '"Princess Bride'))
('933', ((4.217391304347826, 97.0, 23), 'To Catch a Thief (1955)'))
('1196', ((4.2156398104265405, 889.5, 211), 'Star Wars: Episode V - The Empire Strikes Back (1980)'))
('1198', ((4.2075, 841.5, 200), 'Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1
('162', ((4.205882352941177, 71.5, 17), 'Crumb (1994)'))
('1089', ((4.202290076335878, 550.5, 131), 'Reservoir Dogs (1992)'))
('356', ((4.164133738601824, 1370.0, 329), 'Forrest Gump (1994)'))
('1136', ((4.161764705882353, 566.0, 136), 'Monty Python and the Holy Grail (1975)'))
('3083', ((4.157894736842105, 79.0, 19), 'All About My Mother (Todo sobre mi madre) (1999)'))
('5690', ((4.15625, 66.5, 16), 'Grave of the Fireflies (Hotaru no haka) (1988)'))
('2028', ((4.1462765957446805, 779.5, 188), 'Saving Private Ryan (1998)'))
('2067', ((4.142857142857143, 87.0, 21), 'Doctor Zhivago (1965)'))
('68157', ((4.136363636363637, 364.0, 88), 'Inglourious Basterds (2009)'))
```

FIGURE 3 – Top résultat de la requête finale

On voit ici un résultat beaucoup plus cohérent avec des noms de films particulièrement connus comme : "Logan", "Fight Club" et "Usual Suspect".

```python
#!/usr/bin/env python3
#-*- coding: utf-8 -*-
import sys
from pyspark import SparkContext

if __name__ == "__main__":

    # Creation d un contexte Spark
    sc = SparkContext(appName="Spark Count")
    sc.setLogLevel("ERROR") # Valid log levels include: ALL, DEBUG, ERROR, FATAL, INFO, OFF, TRACE, WARN

    tableau_ratings = sc.textFile(sys.argv[1]).map(lambda line: line.split(',')) # userId  movieId  rating
    tableau_movies = sc.textFile(sys.argv[2]).map(lambda line: line.split(','))

    paires_ratings = tableau_ratings.map(lambda champs: (champs[1],champs[2])) # movieId rating
    paires_movies = tableau_movies.map(lambda champs: (champs[0],champs[1])) # movieId title

    paires_ratings = paires_ratings.filter(lambda duet: duet[1]!='' and duet[1]!='rating') # on retire les valeurs manquantes et le titre
    paires_movies = paires_movies.filter(lambda duet: duet[0]!='movieId') # on retire le titre

    paires_ratings = paires_ratings.map(lambda champs: (champs[0],float(champs[1]))) # movieId rating

    paires_ratings_sum= paires_ratings.reduceByKey(lambda v1,v2 : v1+v2) # on  somme les notes de tous les films

    paires_ratings_count = tableau_ratings.map(lambda champs: (champs[1],1)) # movieId  1
    paires_ratings_count= paires_ratings_count.reduceByKey(lambda v1,v2 : v1+v2) # on  compte le nombre de fois où chaque film est noté

    threshold = float(sys.argv[3])
    paires_ratings_count = paires_ratings_count.filter(lambda pair: pair[1]>threshold) # on donne un nombre minimale de notation de film

    classement = paires_ratings_sum.join(paires_ratings_count) # movie id, (sum des ratings, count des ratings)
    classement = classement.map(lambda champs: (champs[1][0]/champs[1][1],(champs[0],champs[1][0],champs[1][1]))) # rapport sum/count,(movie id, sum des ratings
    classement = classement.sortByKey(ascending=False) # classement décroissant par rapport sum/count
    classement_join = classement.map(lambda champs: (champs[1][0],(champs[0],champs[1][1],champs[1][2]))) # movie id,  (rapport sum/count,sum des ratings, count

    classement_final = classement_join.join(paires_movies) # on join les 2 dataframes pour obtenir le nom des movies

    classement_final.saveAsTextFile("classement")

    # Arret du contexte Spark
    sc.stop()
```

FIGURE 4 – Code de la requête finale