RAPPORT PROJET BIG DATA LABS

Réalisé par: Atika Wamra
Maryame Laouina

Encadré par: El Alaoui Hasna

Table des matières

Introduction

La diffusion en ligne de contenus multimédias a connu une croissance exponentielle ces dernières années, avec des plateformes comme PoliTV émergeant en tant qu'acteurs majeurs de cette révolution numérique. PoliTV, une société internationale de streaming axée sur les films, accueille des millions d'utilisateurs du monde entier. Dans le cadre de son engagement envers l'amélioration continue et la compréhension approfondie de ses données, les gestionnaires de PoliTV ont sollicité le développement d'une application d'analyse de données basée sur Hadoop MapReduce et Spark.

Ce rapport détaille la conception et la mise en œuvre d'une solution complète répondant à des questions spécifiques formulées par les responsables de PoliTV. Les données nécessaires à ces analyses sont fournies à travers trois fichiers volumineux : Users.txt, Movies.txt et WatchedMovies.txt. Ces fichiers contiennent des informations détaillées sur les utilisateurs enregistrés, le catalogue des films disponibles, ainsi que l'historique des films visionnés au cours des années précédentes.

L'objectif principal de ce projet est de fournir des analyses approfondies basées sur ces données, permettant à PoliTV de prendre des décisions éclairées pour améliorer l'expérience utilisateur et d'optimiser la disponibilité de ses contenus. La solution est divisée en deux parties distinctes, utilisant Hadoop MapReduce pour la première partie et Spark pour les deux suivantes.

Ce rapport présente en détail la conception, l'implémentation, et les résultats de chaque composant de l'application, offrant ainsi une compréhension complète de la manière dont les technologies Big Data peuvent être mises en œuvre pour répondre aux défis spécifiques de l'industrie du streaming en ligne.

I. Description de l'environnement de travail

Le succès de ce projet repose sur l'utilisation judicieuse d'outils Big Data bien établis pour traiter, analyser et extraire des informations significatives à partir des données massives fournies par PoliTV. Deux technologies clés ont été choisies pour ce faire : Hadoop MapReduce et Apache Spark, chacune apportant des avantages spécifiques à notre solution.

Hadoop MapReduce :

Hadoop MapReduce a été sélectionné pour sa capacité à traiter de grands ensembles de données de manière distribuée. L'architecture distribuée de Hadoop permet de traiter les fichiers massifs des données de manière parallèle, accélérant ainsi le processus d'analyse. Les étapes de mappage et de réduction offrent une approche efficace pour filtrer, trier et agréger les données afin de répondre à des requêtes spécifiques, telles que l'identification des films visionnés par un seul utilisateur en 2019.

Apache Spark :

Spark a été choisi pour les deux dernières parties du projet en raison de sa puissance de traitement in-memory et de ses fonctionnalités avancées d'analyse de données. Les RDD (Resilient Distributed Datasets) de Spark permettent des opérations de transformation et d'action complexes sur les données, rendant possible l'analyse détaillée des habitudes de visionnage sur plusieurs années. De plus, la facilité d'utilisation de Spark avec le langage Python offre une flexibilité et une lisibilité accrues lors de la conception et du développement de l'application.

L'utilisation de ces deux technologies complémentaires permet de tirer le meilleur parti des avantages spécifiques de chacune, garantissant ainsi une solution robuste et évolutive pour l'analyse des données de PoliTV. Ces choix d'outils ont été motivés par la nécessité de traiter efficacement des volumes massifs de données tout en offrant des performances optimales et une flexibilité d'analyse.

II. Description des données

Les données fournies par PoliTV sont cruciales pour comprendre les comportements des utilisateurs et les tendances de visionnage, formant ainsi la base de nos analyses. Trois fichiers d'entrée principaux structurent ces données : Users.txt, Movies.txt, et WatchedMovies.txt.

Users.txt :

Ce fichier texte massif contient la liste des utilisateurs enregistrés sur la plateforme de streaming PoliTV. Avec des millions d'utilisateurs, chaque ligne de Users.txt représente le profil d'un utilisateur et est formatée comme suit : "Username, Gender, YearOfBirth, Country". Par exemple, "Paolo G76, Male, 1976, Italy" identifie l'utilisateur Paolo G76, de genre masculin, né en 1976 en Italie.

Movies.txt :

Ce fichier texte contient le catalogue des films disponibles sur PoliTV, totalisant plus de 100 000 films. Chaque ligne de Movies.txt est associée à un film et suit le format : "MID, Title, Director, ReleaseDate". Par exemple, "MID124, Ghostbusters, Ivan Reitman, 1984/05/01" identifie le film "Ghostbusters" réalisé par Ivan Reitman et sorti le 1er mai 1984.

WatchedMovies.txt :

Ce fichier massif enregistre l'historique des films visionnés, stockant des milliards de lignes de données. Chaque ligne suit le format : "Username, MID, StartTimestamp, End Timestamp". Par exemple, "Paolo G76, MID124, 2010/06/01_14:18, 2010/06/01_16:10" indique que l'utilisateur Paolo G76 a regardé le film associé à "MID124" entre le 1er juin 2010 à 14h18 et le 1er juin 2010 à 16h10.

Ces fichiers constituent une base riche pour l'analyse, offrant des informations détaillées sur les utilisateurs, les films disponibles, et les habitudes de visionnage au fil du temps. La diversité des données permet d'aborder une variété de questions liées aux préférences des utilisateurs, à la popularité des films, et aux tendances de visionnage.

III. Conception et implémentation de la partie Hadoop

Objectif : Films regardés par un seul utilisateur au cours de l'année 2019

1. Conception :

Pour atteindre cet objectif, l'application Hadoop MapReduce a été conçue en deux phases principales : la phase de mapper et la phase de reducer

- Phase de Mapper :
  - Lors de cette phase, chaque ligne de WatchedMovies.txt est analysée pour extraire l'information pertinente, notamment le nom d'utilisateur et la date de début de visionnage.
  - Les données sont filtrées pour inclure uniquement celles de l'année 2019.
  - La clé de sortie est composée du nom d'utilisateur et du MID, avec la valeur associée au timestamp de début de visionnage.
- Phase de Reducer :
  - Les données mappées sont triées par nom d'utilisateur et MID.
  - Un compteur est maintenu pour chaque film (MID) afin de suivre le nombre d'utilisateurs distincts l'ayant visionné en 2019.
  - Les films visionnés par un seul utilisateur sont identifiés en éliminant ceux dont le compteur est supérieur à 1.
  - Les MID des films sélectionnés sont écrits dans le dossier de sortie HDFS "out".

2. Implémentation :

Le code Java correspondant à cette conception utilise les classes Hadoop MapReduce, telles que Mapper et Reducer, pour traiter les données de manière distribuée.

Script:

nous avons créer un dossier projet dans laquelle il y a un dossier src , la commande ls nous a permet de lister le contenu de dossier src .

```
ubuntu@ubuntuL:~/Documents/projet/src$ ls
Movies.txt  UniqueUserMovies2019.java  Users.txt  WatchedMovies.txt
ubuntu@ubuntuL:~/Documents/projet/src$ hdfs dfs -mkdir /in
```

Cette commande copie le fichier watchedmovies.txt depuis votre système de fichiers local vers le système de fichiers distribué Hadoop (HDFS).

```
ubuntu@ubuntuL:~/Documents/projet/src$ hdfs dfs -put WatchedMovies.txt
```

Cette commande **hdfs dfs -put watchedmovies.txt /in** copie simplement le fichier nommé **watchedmovies.txt** depuis votre système de fichiers local vers le répertoire /in du système de fichiers distribué Hadoop (HDFS).

```
ubuntu@ubuntuL:~/Documents/projet/src$ hdfs dfs -put WatchedMovies.txt /in
```

Cette commande compile un fichier Java appelé uniqueuser2019.java en utilisant le classpath spécifié par la commande hadoop classpath. Cela garantit que toutes les dépendances nécessaires pour interagir avec Hadoop sont incluses lors de la compilation du fichier Java.

```
ubuntu@ubuntuL:~/Documents/projet/src$ javac -classpath $(hadoop classpath) UniqueUserMovies2019.java
```

```
ubuntu@ubuntuL:~/Documents/projet/src$ ls
 Movies.txt                                UniqueUserMovies2019.class   WatchedMovies.txt
'UniqueUserMovies2019$IntSumReducer.class'     UniqueUserMovies2019.java
'UniqueUserMovies2019$TokenizerMapper.class'   Users.txt
```

Cette commande crée un fichier JAR appelé **project.jar** en rassemblant plusieurs fichiers de classes Java nécessaires pour un projet.

```
ubuntu@ubuntuL:~/Documents/projet/src$ jar cf project.jar UniqueUserMovies2019.class 'UniqueUserMovies2019$IntSumReduc
er.class' 'UniqueUserMovies2019$TokenizerMapper.class'
```

```
ubuntu@ubuntuL:~/Documents/projet/src$ ls
 Movies.txt   'UniqueUserMovies2019$IntSumReducer.class'     UniqueUserMovies2019.class   Users.txt
 project.jar  'UniqueUserMovies2019$TokenizerMapper.class'   UniqueUserMovies2019.java    WatchedMovies.txt
```

```
ubuntu@ubuntuL:~/Documents/projet/src$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as ubuntu in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
localhost: namenode is running as process 3450.  Stop it first and ensure /tmp/hadoop-ubuntu-namenode.pid file is empt
y before retry.
Starting datanodes
localhost: datanode is running as process 3579.  Stop it first and ensure /tmp/hadoop-ubuntu-datanode.pid file is empt
y before retry.
Starting secondary namenodes [ubuntuL]
ubuntuL: secondarynamenode is running as process 3745.  Stop it first and ensure /tmp/hadoop-ubuntu-secondarynamenode.
pid file is empty before retry.
Starting resourcemanager
resourcemanager is running as process 3983.  Stop it first and ensure /tmp/hadoop-ubuntu-resourcemanager.pid file is e
mpty before retry.
Starting nodemanagers
localhost: nodemanager is running as process 4102.  Stop it first and ensure /tmp/hadoop-ubuntu-nodemanager.pid file i
s empty before retry.
```

```
ubuntu@ubuntuL:~/Documents/projet/src$ jps
3745 SecondaryNameNode
4102 NodeManager
8262 Jps
3450 NameNode
3579 DataNode
3983 ResourceManager
```

```
ubuntu@ubuntuL:~/Documents/projet/src$ hadoop jar project.jar UniqueUserMovies2019 /in/WatchedMovies.txt /new_out
2023-12-08 12:18:49,299 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-12-08 12:18:50,594 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implemen
t the Tool interface and execute your application with ToolRunner to remedy this.
2023-12-08 12:18:50,677 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/stagin
g/ubuntu/.staging/job_1702032563471_0001
2023-12-08 12:18:51,111 INFO input.FileInputFormat: Total input files to process : 1
2023-12-08 12:18:51,298 INFO mapreduce.JobSubmitter: number of splits:1
2023-12-08 12:18:51,729 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1702032563471_0001
2023-12-08 12:18:51,729 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-12-08 12:18:52,266 INFO conf.Configuration: resource-types.xml not found
2023-12-08 12:18:52,274 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-12-08 12:18:53,522 INFO impl.YarnClientImpl: Submitted application application_1702032563471_0001
2023-12-08 12:18:53,739 INFO mapreduce.Job: The url to track the job: http://ubuntuL:8088/proxy/application_1702032563
471_0001/
2023-12-08 12:18:53,748 INFO mapreduce.Job: Running job: job_1702032563471_0001
2023-12-08 12:19:07,151 INFO mapreduce.Job: Job job_1702032563471_0001 running in uber mode : false
2023-12-08 12:19:07,152 INFO mapreduce.Job:  map 0% reduce 0%
2023-12-08 12:19:12,234 INFO mapreduce.Job:  map 100% reduce 0%
2023-12-08 12:19:20,303 INFO mapreduce.Job:  map 100% reduce 100%
2023-12-08 12:19:20,337 INFO mapreduce.Job: Job job_1702032563471_0001 completed successfully
2023-12-08 12:19:20,472 INFO mapreduce.Job: Counters: 54
```

voila résultat que nous avons obtenu

```
ubuntu@ubuntuL:~/Documents/projet/src$ hadoop fs -cat /new_out/part-r-00000
MID10
MID124
ubuntu@ubuntuL:~/Documents/projet/src$ S
```

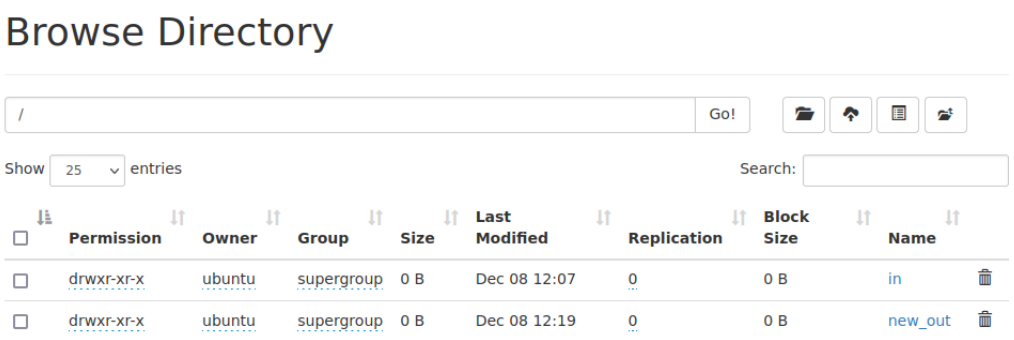
Voila le code java que nous avons exécuté

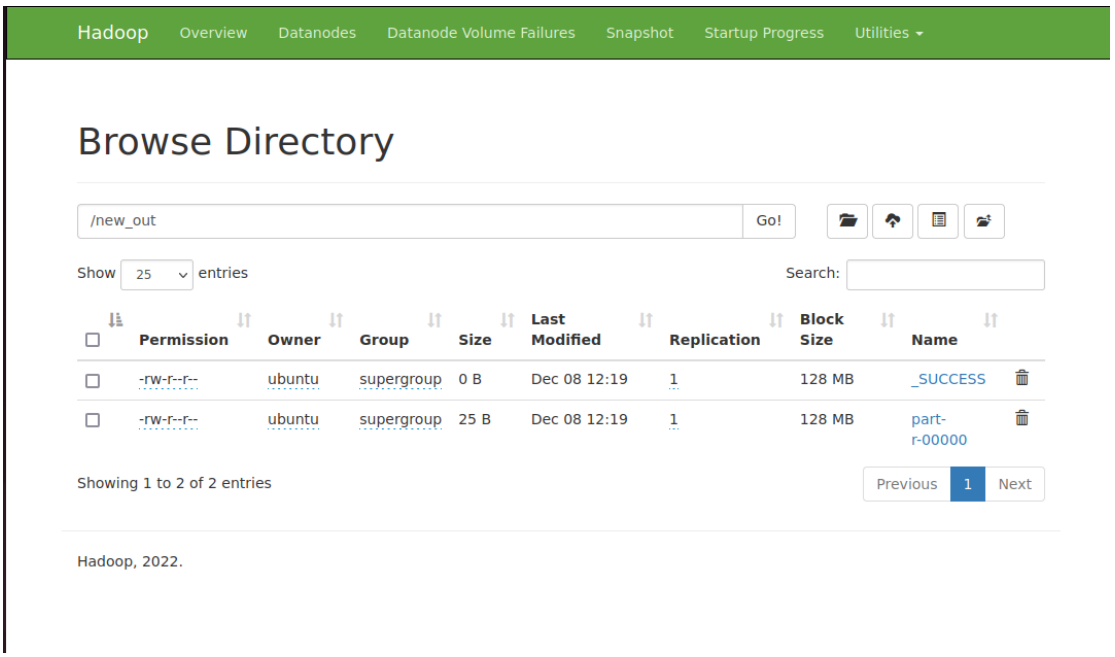
```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

public class UniqueUserMovies2019 {

    public static class TokenizerMapper extends Mapper<Object, Text, Text, Text> {

        private Text movieId = new Text();
        private Text userId = new Text();

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            String[] parts = value.toString().split(",");

            // Assuming line format is UserID, MovieID, Date
            String userID = parts[0];
            String movieID = parts[1];
            String date = parts[2];

            // Check if the date is in 2019
            if (date.contains("2019")) {
                movieId.set(movieID);
                userId.set(userID);
                context.write(movieId, userId);
            }
        }
    }

```

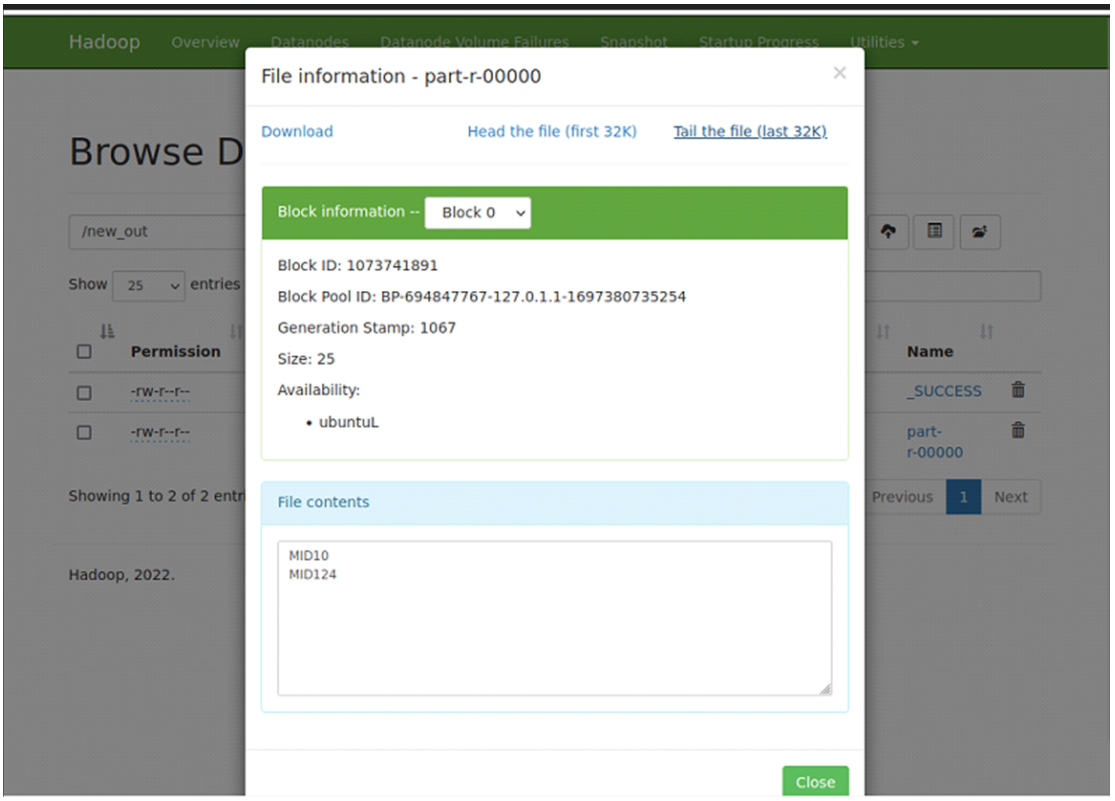
```java
    public static class IntSumReducer extends Reducer<Text, Text, Text, Text> {

        private Text result = new Text();

        public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
  InterruptedException {
            Set<String> uniqueUsers = new HashSet<>();

            for (Text val : values) {
                uniqueUsers.add(val.toString());
            }

            // Write the movie to output only if a single user watched it
            if (uniqueUsers.size() == 1) {
                result.set(uniqueUsers.iterator().next());
                context.write(key, result);
            }
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "single viewer movies 2019");
        job.setJarByClass(UniqueUserMovies2019.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

L'application Hadoop MapReduce est ensuite exécutée sur le cluster, traitant les données de manière parallèle et générant une liste des MID des films visionnés par un seul utilisateur en 2019 dans le dossier de sortie spécifié.

Résultats:




Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities
Browse Directory
/new_out
Go!
Show 25 entries
Search:
Permission
Owner
Group
Size
Last Modified
Replication
Block Size
Name
-rw-r--r--
ubuntu
supergroup
0 B
Dec 08 12:19
1
128 MB
_SUCCESS
-rw-r--r--
ubuntu
supergroup
25 B
Dec 08 12:19
1
128 MB
part-r-00000
Showing 1 to 2 of 2 entries
Previous
1
Next
Hadoop, 2022.


File information - part-r-00000
Download
Head the file (first 32K)
Tail the file (last 32K)
Block information --
Block 0
Block ID: 1073741891
Block Pool ID: BP-694847767-127.0.1.1-1697380735254
Generation Stamp: 1067
Size: 25
Availability:
ubuntuL
File contents
MID10
MID124
Close

IV. Conception et implémentation de la partie Spark

I. Première Partie

Objectif : Films qui ont été regardés fréquemment mais seulement pendant un an au cours des cinq dernières années

1. Conception :

L'application Spark pour cette partie du projet est également divisée en plusieurs étapes, tirant parti des fonctionnalités de traitement distribué des RDD pour effectuer des transformations et des actions complexes.

- Étape 1 - Filtrer les données :
  - Les lignes de WatchedMovies.txt liées aux cinq dernières années sont filtrées.
  - La structure des données est transformée pour isoler le MID, l'année, et le nombre de visionnages.
- Étape 2 - Identifier les films fréquemment visionnés sur une seule année :
  - Les données sont groupées par MID et année.
  - Pour chaque groupe, le nombre total de visionnages est calculé.
  - Seuls les films visionnés au moins 1 000 fois pendant une seule année sont retenus.
- Étape 3 - Émission des résultats :
  - Les résultats, sous la forme d'une paire (MID, année), sont écrits dans le dossier de sortie HDFS "outPart2/".

2. Implémentation :

L'application Spark est mise en œuvre en utilisant le langage Python, profitant de la simplicité et de la lisibilité offertes par PySpark.

Script:

test.py | spark-partie1.py

C: > Users > Maryam > Documents > Mini_Projet_Manssouri_Belemsaggam > 2 SPARK_RDD > spark-partie1.py

```python
from pyspark import SparkContext, SparkConf


# Create a Spark configuration and set the application name
LEAST_NUMBER_OCCURRENCES = 4

conf = SparkConf().setAppName("MovieAnalysis")
sc = SparkContext(conf=conf)

# Load the data
watched_movies_rdd = sc.textFile("WatchedMovies.txt")

# Filter data for the last five years
def filter_last_five_years(line):
    try:
        # Extract the year from the StartTimestamp
        year = int(line.split(",")[2].split("/")[0])
        # Check if the year is between 2015 and 2020 (both inclusive)
        return 2015 <= year <= 2020
    except:
        return False

filtered_watched_movies_rdd = watched_movies_rdd.filter(filter_last_five_years)

# Count occurrences of each movie during a specific year
def count_occurrences(line):
    try:
        # Extract MID and year
        username, mid, start_timestamp, end_timestamp = line.split(",")
        year = int(start_timestamp.split("/")[0])
        return ((mid, year), 1)
    except:
        return (("ERROR", 0), 1)  # Placeholder for error handling

# Count total occurrences of each movie per year
total_occurrences_rdd = filtered_watched_movies_rdd.map(count_occurrences).reduceByKey(lambda x, y: x + y)
```

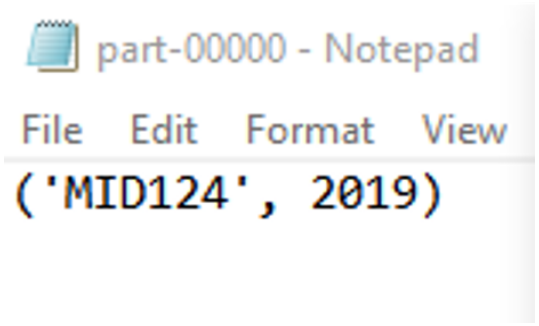
```python
# Filter movies that have been watched more than 4 times in a single year
# and apply 0 to other times
frequent_movies_rdd = total_occurrences_rdd \
    .map(lambda x: (x[0][0], (x[0][1], x[1]))) \
    .groupByKey() \
    .flatMap(lambda x: [(x[0], year) for year, occurrences in x[1] if (occurrences > LEAST_NUMBER_OCCURRENCES and all(other_occurrences == 0 for other_year, other_occurrences in x[1] if

# Save the results to the specified output folder
frequent_movies_rdd.saveAsTextFile("output_folder/")

# Stop the Spark context
sc.stop()
```

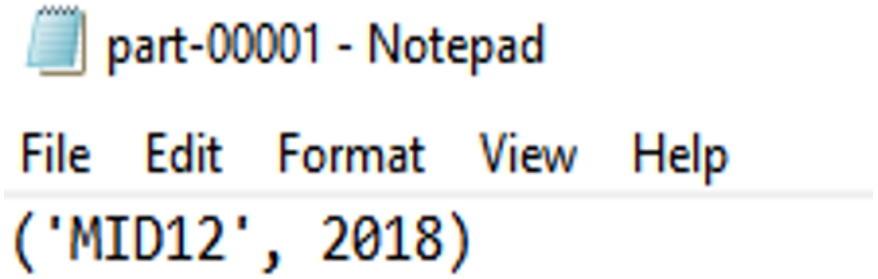
00000 | part-00001 | train_cleane | WatchedMo

File Edit View

```
PaoloG76,MID12,2018/06/01_14:18,2018/06/01_16:10
PaoloG76,MID12,2018/07/01_14:18,2018/07/01_16:10
User1,MID12,2018/07/01_14:18,2018/07/01_16:10
User1,MID12,2018/04/01_14:18,2018/04/01_16:10
User2,MID12,2018/08/01_14:18,2018/08/01_16:10
PaoloG76,MID10,2018/06/01_14:18,2018/06/01_16:10
PaoloG76,MID10,2018/07/01_14:18,2018/07/01_16:10
User1,MID10,2018/07/01_14:18,2018/07/01_16:10
User2,MID10,2018/08/01_14:18,2018/08/01_16:10
User2,MID10,2018/08/01_14:18,2018/08/01_16:10
User2,MID10,2019/08/01_14:18,2019/08/01_16:10
User2,MID10,2019/08/01_14:18,2019/08/01_16:10
User2,MID124,2019/08/01_14:18,2018/08/01_16:10
User2,MID124,2019/08/01_14:18,2019/08/01_16:10
User1,MID124,2019/08/01_14:18,2018/08/01_16:10
User2,MID124,2019/08/01_14:18,2019/08/01_16:10
PaoloG76,MID124,2019/07/01_14:18,2019/07/01_16:10
User2,MID124,2019/08/01_14:18,2019/08/01_16:10
```

Résultats





Ce code PySpark réalise une analyse efficace des films qui ont été regardés fréquemment mais seulement pendant un an au cours des cinq dernières années, générant des résultats stockés dans le dossier de sortie spécifié.

II. Deuxième Partie

Objectif : Film le plus populaire depuis au moins deux ans

1. Conception :

Cette partie du projet implique une analyse plus complexe pour identifier les films les plus populaires sur une période de deux ans ou plus. L'approche se divise en plusieurs étapes clés.

- Initialisation de Spark :
  - Initialiser un contexte Spark "MostWatchedMoviesByYear".
- Chargement des données :
  - Charger les données depuis "WatchedMovies.txt" en tant que RDD.
- Analyse et Transformation des données :
  - Appliquer une fonction (parse_line) pour extraire des informations clés.
  - Mapper et réduire pour obtenir la popularité des films par année.
- Collecte des Résultats :
  - Collecter les résultats dans most_popular_movies_data.
- Traitement des Résultats :

        - Créer max_views_by_year pour stocker les films les plus populaires par année.
- Affichage des Résultats :
    - Afficher les films les plus populaires par année.
- Identification des Films sur au Moins Deux Ans :
    - Utiliser une fonction (save_movies_in_multiple_years) pour filtrer les films présents sur au moins deux ans.
- Affichage et Sauvegarde des Résultats :
    - Afficher et sauvegarder les résultats filtrés dans un fichier texte.
- Arrêter le contexte Spark.

2. Implémentation :

L'application Spark est mise en œuvre en utilisant le langage Python avec PySpark.

Script:

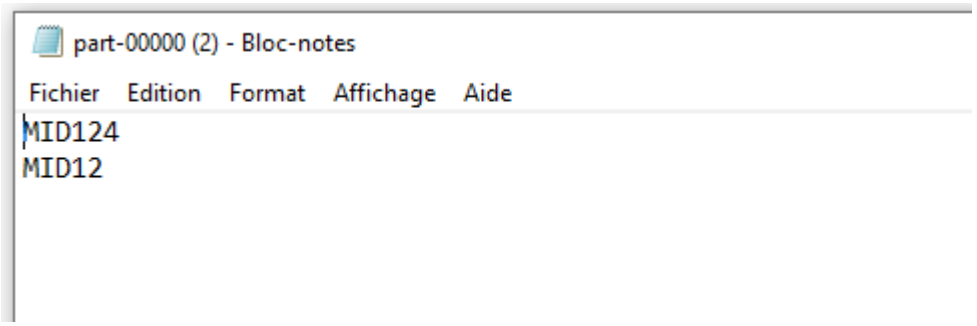
```python
from pyspark import SparkContext
from datetime import datetime

# Initialize a Spark context
spark_context = SparkContext(appName="MostWatchedMoviesByYear")

# Load the data from the file as an RDD
data_file_path = "C:\\Users\\Hp\\Desktop\\WatchedMovies.txt"
data_rdd = spark_context.textFile(data_file_path)

# Define a function to parse each line and convert it to a tuple
def parse_line(line):
    # Split the line into fields
    fields = line.split(',')
    # Extract relevant information
    username, movie_id, start_timestamp, end_timestamp = fields
    start_date = datetime.strptime(start_timestamp, '%Y/%m/%d_%H:%M').date()
    end_date = datetime.strptime(end_timestamp, '%Y/%m/%d_%H:%M').date()
    return username, movie_id, start_date, end_date

# Apply the parse_line function to the RDD
parsed_data_rdd = data_rdd.map(parse_line)

# Use distinct to consider only distinct users for each movie in each year
distinct_data_rdd = parsed_data_rdd.map(lambda x: ((x[1], x[2].year, x[0]), 1)).distinct()

```

```python
# Map each row to a key-value pair where the key is (MovieID, year) and the value is 1
key_value_data_rdd = distinct_data_rdd.map(lambda x: ((x[0][0], x[0][1]), 1))

# Reduce by key to count the occurrences
movie_year_counts_rdd = key_value_data_rdd.reduceByKey(lambda a, b: a + b)

# Map to a structure where the key is the year and the value is a list of tuples (MovieID, count)
year_movie_counts_rdd = movie_year_counts_rdd.map(lambda x: (x[0][1], [(x[0][0], x[1])]))

# Reduce by key to find the most popular movie in each year
most_popular_movies_rdd = year_movie_counts_rdd.reduceByKey(lambda a, b: a + b).map(lambda x: (x[0], sorted(x[1], key=

# Rest of the code...
# Collect the data to the driver program
most_popular_movies_data = most_popular_movies_rdd.collect()

# Create a dictionary to store max views by year
max_views_by_year = {}

# Iterate over the collected data
for year, movie_views_list in most_popular_movies_data:
    max_views = 0
    max_views_movies = []
```

```python
    # Iterate over movies and views in each year
    for movie, views in movie_views_list:
        if views > max_views:
            max_views = views
            max_views_movies = [movie]
        elif views == max_views:
            max_views_movies.append(movie)

    # Store the result in the dictionary
    max_views_by_year[year] = (max_views_movies, max_views)

# Print the result
print(max_views_by_year)

# Function to save movies occurring in multiple years
def save_movies_in_multiple_years(data):
    movie_occurrences = {}

    # Iterate over years and movies
    for year, (movies, _) in max_views_by_year.items():
        for movie in movies:
            if movie in movie_occurrences:
                movie_occurrences[movie].append(year)
            else:
                movie_occurrences[movie] = [year]
```

```
    # Filter movies that occur in at least two years
    movies_in_multiple_years = [movie for movie, years in movie_occurrences.items() if len(years) >= 2]

    return movies_in_multiple_years

# Call the function and print the result
result = save_movies_in_multiple_years(max_views_by_year)
print("Movies that occur in at least two years:", result, type(result))

# Use parallelize to create an RDD from the result
result_rdd = spark_context.parallelize(result)

# Assuming result_rdd is your RDD
coalesced_rdd = result_rdd.coalesce(1)

# Save the coalesced RDD as a text file
coalesced_rdd.saveAsTextFile('C:\\Users\\Hp\\Desktop\\task2_outatik.txt')

# Stop the Spark context
spark_context.stop()
```

Résultats:



Ce code PySpark réalise une analyse approfondie pour identifier les films les plus populaires depuis au moins deux ans, générant des résultats stockés dans le dossier de sortie spécifié.

V. Conclusion

En bref, ce projet d'analyse pour PoliTV a efficacement utilisé Hadoop MapReduce et Spark pour extraire des informations clés. Les analyses ont révélé des insights sur les préférences des utilisateurs et la popularité des films. Ces résultats offrent à PoliTV des conseils précieux pour optimiser son contenu. Ce projet souligne l'impact significatif de l'analyse de données dans l'industrie du streaming en ligne.