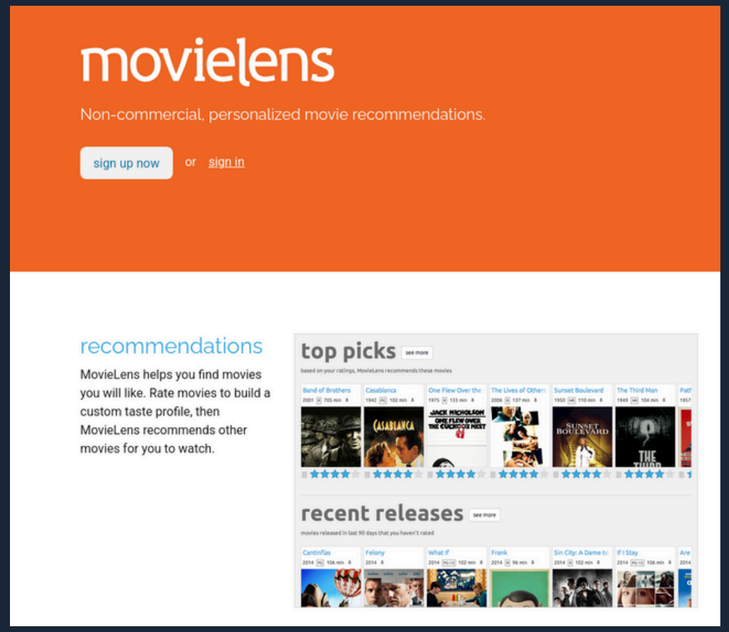
MovieLens

by Laura & Blandine

Introduction

La donnée



Observations

```
├── metadata.json
├── metadata_updated.json
├── ratings.json
├── reviews.json
├── survey_answers.json
├── tag_count.json
└── tags.json
```

```
Affichage des premières lignes et informations de : raw/ratings.json
Taille du fichier : 1.3774 Go
Nombre de lignes dans le fichier : 28490116
============================================================
Premières lignes du fichier :
{"item_id": 5, "user_id": 997206, "rating": 3.0}
{"item_id": 10, "user_id": 997206, "rating": 4.0}
{"item_id": 13, "user_id": 997206, "rating": 4.0}
{"item_id": 17, "user_id": 997206, "rating": 5.0}
{"item_id": 21, "user_id": 997206, "rating": 4.0}
```

```
Affichage des premières lignes et informations de : raw/metadata_updated.json
Taille du fichier : 0.0167 Go
Nombre de lignes dans le fichier : 84661
============================================================

Affichage des premières lignes et informations de : raw/survey_answers.json
Taille du fichier : 0.0036 Go
Nombre de lignes dans le fichier : 58903
============================================================

Affichage des premières lignes et informations de : raw/ratings.json
Taille du fichier : 1.3774 Go
Nombre de lignes dans le fichier : 28490116
============================================================

Affichage des premières lignes et informations de : raw/tag_count.json
Taille du fichier : 0.0091 Go
Nombre de lignes dans le fichier : 212704
============================================================

Affichage des premières lignes et informations de : raw/tags.json
Taille du fichier : 0.0000 Go
Nombre de lignes dans le fichier : 1094
============================================================

Affichage des premières lignes et informations de : raw/metadata.json
Taille du fichier : 0.0194 Go
Nombre de lignes dans le fichier : 84661
============================================================
```

Les fichiers

metadata.json

{"title": "Toy Story (1995)", "directedBy": "John Lasseter", "starring": "Tim Allen, Tom Hanks, Don Rickles, Jim Varney, John Ratzenberger, Wallace Shawn, Laurie Metcalf, John Morris, R. Lee Ermey, Annie Potts", "dateAdded": null, "avgRating": 3.89146, "imdbId": "0114709", "item_id": 1}

tags.json

{"tag": "secret service", "id": 112}

reviews.json

{"item_id": 7065, "txt": "unbelievable; I cannot understand how anyone can call this one of the greatest movies ever made. It is disgraceful and appalling. I guess it is still high entertainment to see white actors in black face and watch a film loaded with stereotype."}

ratings.json

{"item_id": 5, "user_id": 997206, "rating": 3.0}

Les tâches

1) Attribuer une note aux films

2) Trier les films par notes

3) Ajouter des tags grâce aux reviews

Exploitation du dataset

1

2

Attribuer une note aux films --> Average_rating_per_movie.py

Trier les films par notes --> Best_rated_movies.py

Ajouter des tags grâce aux reviews --> Tags_according_to_reviews.py

Méthodologie

1 Average_rating_per_movie.py

```python
def get_spark() -> SparkSession:
    """
    Return a Spark session
    """
    return SparkSession.builder \
        .appName("Rating Analysis") \
        .getOrCreate()


spark = get_spark()
```

Ouvrir une session Spark :

- initialisation de SparkContext
- interfaces SQL, DataFrames
- driver et executors

Méthodologie

1 Average_rating_per_movie.py

```
schema = StructType([
    StructField("item_id", IntegerType(), True),
    StructField("user_id", IntegerType(), True),
    StructField("rating", FloatType(), True)
])
```

Définir le schéma ; l'utiliser

```
DF_RAW = read_json_with_schema(data_file_paths=DATA_FILE_PATHS, schema=schema)
```

Méthodologie

1 Average_rating_per_movie.py

```python
def compute_average_rating(df: DataFrame) -> DataFrame:
    """
    Compute the average rating for each item_id
    """
    # Group by item_id and compute the average rating
    df_avg = df.groupBy("item_id").avg("rating").withColumnRenamed("avg(rating)", "avg_rating")

    return df_avg
```

ENFIN : Calculer la note globale !

GROUP BY et AVG (équivalent d'un MapReduce)

Méthodologie

2 Best_rated_movies.py

```
DF_AVG_RATING = compute_average_rating(DF_RAW)

DF_RATING_COUNT = DF_RAW.groupBy("item_id").count().withColumnRenamed("count", "count_rating")
DF_AVG_WITH_COUNT = DF_AVG_RATING.join(DF_RATING_COUNT, on="item_id", how="inner")
DF_AVG_WITH_COUNT_FILTERED = DF_AVG_WITH_COUNT.filter(col("count_rating")>50)
```

Comme avant

Nouvelles fonctions : filter, count

SparkSQL : JOIN

Méthodologie

3

Tags_according_to_reviews.py
(and Tags_according_to_reviews_by_chunks.py)

```
reviews_df = spark.read.json("../raw/reviews_bis.json").select("item_id", "txt")
tags_df = spark.read.json("../raw/tags.json").select("tag", "id")

def contains_tag(text, tag):
    return tag.lower() in text.lower() if text else False

contains_tag_udf = udf(contains_tag, BooleanType())
##User-Defined Functions (UDFs) are user-programmable routines that act on one row. Source : Apache Spark

matched_df = reviews_df.crossJoin(tags_df).filter(contains_tag_udf(col("txt"), col("tag")))
```

On ne conserve que ce qui est nécessaire à chaque étape !

Résultats

“cirque”

“chrétien”

“stupide”

Execution distribuée

FICHIERS

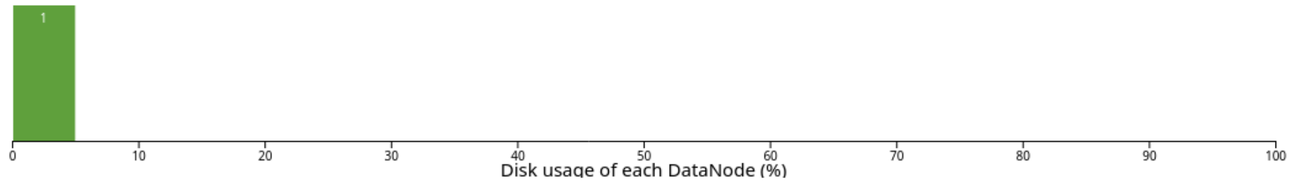
CALCULS

Systeme de fichiers distribués

Datanode Information

In service | Down | Decommissioning | Decommissioned | Decommissioned & dead
Entering Maintenance | In Maintenance | In Maintenance & dead

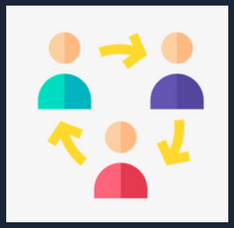
Datanode usage histogram



In operation

DataNode State: All | Show 25 entries | Search:

Node	Http Address	Last contact	Last Block Report	Used	Non DFS Used	Capacity	Blocks	Block pool used	Block pool usage StdDev	Version
/default-rack/blandine-KPL-W0X:9866 (127.0.0.1:9866)	http://blandine-KPL-W0X:9864	2s	1m	1.39 GB	198.83 GB	232.64 GB	12	1.39 GB (0.6%)	0%	3.4.1



Partage

Pannes

Réseau

Caclul distribué

Workers (1)

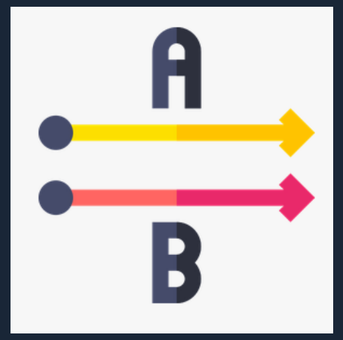
Worker Id	Address	State	Cores	Memory	Resources
worker-20250305173840-147.250.234.244-41397	147.250.234.244:41397	ALIVE	8 (8 Used)	5.7 GiB (1024.0 MiB Used)	

Running Applications (1)

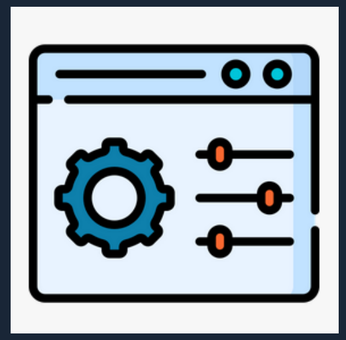
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20250305180206-0001 (kill)	Rating Analysis	8	1024.0 MiB		2025/03/05 18:02:06	blandine	RUNNING	2 s

Completed Applications (1)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20250305174753-0000	Rating Analysis	8	1024.0 MiB		2025/03/05 17:47:53	blandine	FINISHED	47 s



Parallèlisation



Configuration

Réseau

Execution du script

Adaptation du code

```
# Définition des variables d'environnement
SPARK_MASTER = "spark://" + os.getenv("SPARK_MASTER", "spark://129.104.73.59:7077")
HDFS_BASE_PATH = os.getenv("HDFS_BASE_PATH", "hdfs://localhost:9000/user/blandine")
```

```
def get_spark() -> SparkSession:
    """
    Return a Spark session configured with the master URL
    """
    return SparkSession.builder \
        .appName("Rating Analysis") \
        .master(SPARK_MASTER) \
        .getOrCreate()
```

```
# Définition des chemins d'entrée et de sortie sur HDFS
DATA_FILE_PATHS = [f"{HDFS_BASE_PATH}/ratings.json"]
OUTPUT_PATH = f"{HDFS_BASE_PATH}/output/ratings_parquet"
```

Lancement du script

```
export SPARK_MASTER=$(curl -s http://localhost:8080 | grep -oP '(?<=spark://)[^:]+:\d+' | head -n 1)
spark-submit --master spark://$SPARK_MASTER code.py
```

Merci de votre attention

Notre GitHub : https://github.com/Laura-Montagnier/Projet_BigData_FilmReviews/