

Mini-Projet Big Data

Installation d'un Cluster Hadoop et Analyse du Dataset MovieLens ml-32m



Réalisé par :

Mouad El Abbassi

Ikram El Menhi

Abderrahim Azeroual

Module : Big Data / Hadoop

Année universitaire : 2025–2026

Table des matières

1	Introduction	6
2	Objectifs du projet	7
3	Choix technique : machines virtuelles puis Docker	8
3.1	Première approche avec machines virtuelles	8
3.2	Passage à Docker	8
3.3	Correspondance entre VM et conteneurs Docker	9
4	Architecture du cluster Hadoop	10
4.1	Composants du cluster	10
4.2	Schéma logique de l'architecture	10
4.3	Vérification des conteneurs	11
5	Préparation de l'environnement	12
5.1	Structure du projet	12
5.2	Démarrage du cluster	12
5.3	Vérification du chemin Hadoop	13
6	Configuration Hadoop	14
6.1	Configuration HDFS	14
6.2	Configuration YARN et MapReduce	14
7	Dataset MovieLens ml-32m	16
7.1	Présentation du dataset	16
7.2	Comparaison des fichiers	16
7.3	Vérification locale des fichiers	16
8	Chargement des données dans HDFS	18
8.1	Copie des fichiers vers le conteneur NameNode	18
8.2	Création des dossiers HDFS	18
8.3	Chargement dans HDFS	18
8.4	Vérification dans HDFS	18

9	Programme MapReduce	19
9.1	Présentation du programme Java	19
9.2	Job 1 : note moyenne par film	19
9.3	Job 2 : distribution des notes	19
9.4	Job 3 : tags les plus fréquents	19
9.5	Job 4 : popularité des films	20
9.6	Utilisation du Combiner	20
10	Compilation et exécution	21
10.1	Compilation du programme Java	21
10.2	Test MapReduce WordCount	21
10.3	Exécution du pipeline complet	22
10.4	Commandes des jobs MapReduce	22
11	Résultats obtenus	24
11.1	Fichiers de sortie	24
11.2	Distribution des notes	25
11.3	Films les plus populaires	26
11.4	Tags les plus utilisés	26
11.5	Note moyenne par film	27
12	Visualisation des résultats	28
12.1	Génération du dashboard	28
12.2	Dashboard final	29
13	Interprétation des résultats	30
13.1	Analyse de la distribution des notes	30
13.2	Analyse de la popularité	30
13.3	Analyse des tags	30
13.4	Analyse des meilleures notes moyennes	30
14	Avantages et limites de Hadoop	32
14.1	Avantages	32
14.2	Limites et difficultés rencontrées	32
14.3	Limite de l'approche Docker	32
15	Commandes principales utilisées	33
15.1	Commandes Docker	33
15.2	Vérification de Hadoop	33
15.3	Commandes HDFS	33
15.4	Test WordCount	33
15.5	Commande du pipeline MovieLens	34

15.6 Commande de visualisation	34
16 Captures d'écran à inclure	35
17 Conclusion	36

Table des figures

4.1	Architecture logique du cluster Hadoop avec Docker	10
4.2	Vérification des conteneurs Hadoop avec docker ps	11
11.1	Fichiers de résultats générés par les jobs MapReduce	25
12.1	Dashboard final des résultats MovieLens	29

Liste des tableaux

3.1	Correspondance entre l'approche VM et l'approche Docker	9
7.1	Comparaison des fichiers MovieLens utilisés	16
9.1	Principe du job avg_rating	19
9.2	Principe du job ratings_dist	19
11.1	Distribution des notes MovieLens	26

1 Introduction

Le traitement des données volumineuses nécessite des architectures capables de stocker et de traiter des fichiers de grande taille de manière distribuée. Apache Hadoop est une solution largement utilisée dans ce contexte, car il fournit deux composants fondamentaux : HDFS pour le stockage distribué et MapReduce pour le traitement parallèle.

L’objectif de ce mini-projet est d’installer un cluster Hadoop composé de plusieurs noeuds, de charger un fichier volumineux dans HDFS, puis d’exécuter un programme MapReduce afin d’extraire des informations pertinentes. Le dataset choisi est **MovieLens ml-32m**, qui contient des millions de notes attribuées à des films par des utilisateurs.

Dans ce projet, nous avons analysé les fichiers principaux suivants :

- `ratings.csv` : notes données par les utilisateurs aux films ;
- `tags.csv` : tags associés aux films par les utilisateurs ;
- `movies.csv` : métadonnées des films, notamment les titres et les genres ;
- `links.csv` : identifiants externes vers IMDb et TMDb.

Le fichier `ratings.csv` est le plus volumineux. Il représente donc le meilleur candidat pour démontrer l’intérêt du traitement distribué avec Hadoop.

2 Objectifs du projet

Les objectifs principaux du projet sont les suivants :

1. Installer et configurer un cluster Hadoop multi-noeuds.
2. Configurer les composants HDFS, YARN et MapReduce.
3. Télécharger et préparer le dataset MovieLens ml-32m.
4. Charger les fichiers volumineux dans HDFS.
5. Développer un programme MapReduce en Java.
6. Exécuter plusieurs traitements distribués sur le cluster.
7. Récupérer et visualiser les résultats.
8. Discuter les avantages et les limites de l'utilisation d'un cluster Hadoop.

3 Choix technique : machines virtuelles puis Docker

3.1 Première approche avec machines virtuelles

Initialement, nous avons essayé de réaliser le projet avec trois machines virtuelles, conformément à l'énoncé du mini-projet. L'idée était d'utiliser une machine principale comme **NameNode** et deux autres machines comme **DataNodes**.

Cependant, cette approche a présenté plusieurs difficultés pratiques :

- configuration réseau complexe entre les machines virtuelles ;
- problèmes d'adressage IP et de résolution des noms d'hôtes ;
- difficulté à stabiliser la communication SSH entre les noeuds ;
- consommation élevée de mémoire et de processeur sur la machine locale ;
- temps important nécessaire pour réinstaller et reconfigurer Hadoop sur chaque VM.

Ces problèmes n'étaient pas liés au principe de Hadoop lui-même, mais surtout à la configuration réseau et à la lourdeur des machines virtuelles dans notre environnement local.

3.2 Passage à Docker

Pour rendre le projet plus stable, reproductible et léger, nous avons finalement choisi Docker. Docker permet de créer plusieurs conteneurs isolés sur la même machine. Chaque conteneur joue le rôle d'un noeud dans le cluster Hadoop.

Le choix de Docker est justifié par les points suivants :

- démarrage plus rapide que les machines virtuelles ;
- consommation plus faible de ressources ;
- configuration plus simple et reproductible grâce à `docker-compose.yml` ;
- possibilité de recréer entièrement le cluster avec une seule commande ;
- isolation des services Hadoop dans des conteneurs séparés.

Ainsi, même si l'énoncé propose VirtualBox ou VMware, l'architecture réalisée avec Docker garde le même objectif pédagogique : simuler un cluster Hadoop composé de plusieurs noeuds.

3.3 Correspondance entre VM et conteneurs Docker

TABLE 3.1 – Correspondance entre l'approche VM et l'approche Docker

Rôle Hadoop	Approche VM	Approche Docker utilisée
NameNode	VM principale	Conteneur <code>namenode</code>
DataNode 1	VM worker 1	Conteneur <code>datanode1</code>
DataNode 2	VM worker 2	Conteneur <code>datanode2</code>
DataNode 3	VM worker 3	Conteneur <code>datanode3</code>
ResourceManager	VM principale	Conteneur <code>resourcemanager</code>
NodeManager	VM worker	Conteneur <code>nodemanager</code>
HistoryServer	Service Hadoop	Conteneur <code>historyserver</code>

4 Architecture du cluster Hadoop

4.1 Composants du cluster

Le cluster Hadoop utilisé contient les services suivants :

- **NameNode** : responsable de la gestion des métadonnées HDFS ;
- **DataNodes** : responsables du stockage réel des blocs de données ;
- **ResourceManager** : responsable de la gestion globale des ressources YARN ;
- **NodeManager** : responsable de l'exécution des tâches sur un noeud ;
- **HistoryServer** : responsable de l'historique des applications MapReduce.

4.2 Schéma logique de l'architecture

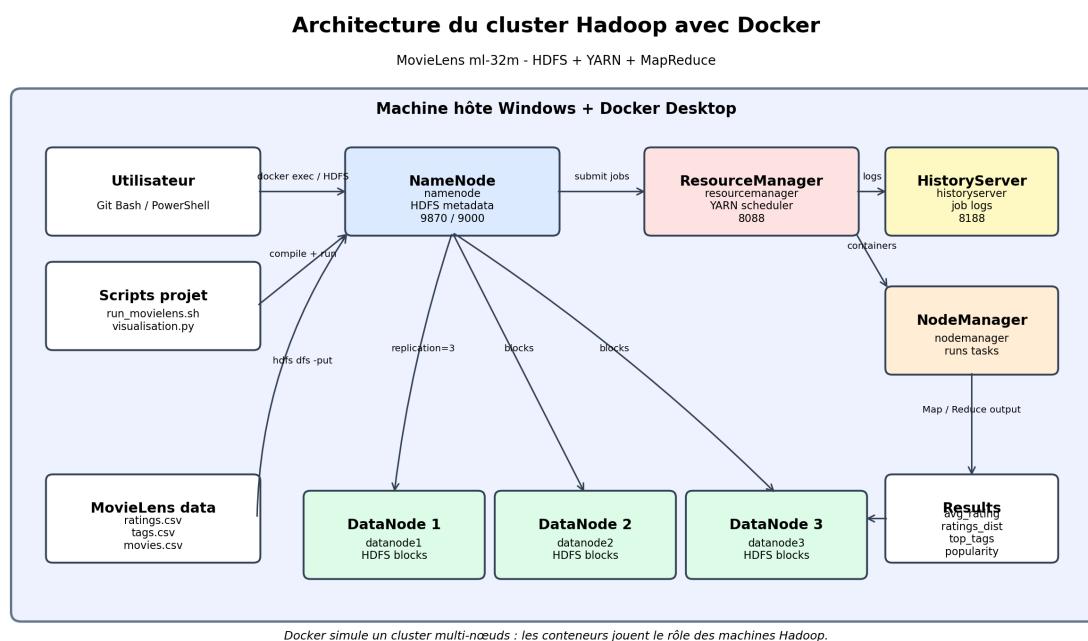


FIGURE 4.1 – Architecture logique du cluster Hadoop avec Docker

Remarque : si vous n'avez pas encore ce schéma, vous pouvez le créer rapidement avec Draw.io ou PowerPoint. Il suffit de représenter le NameNode, les trois DataNodes, le

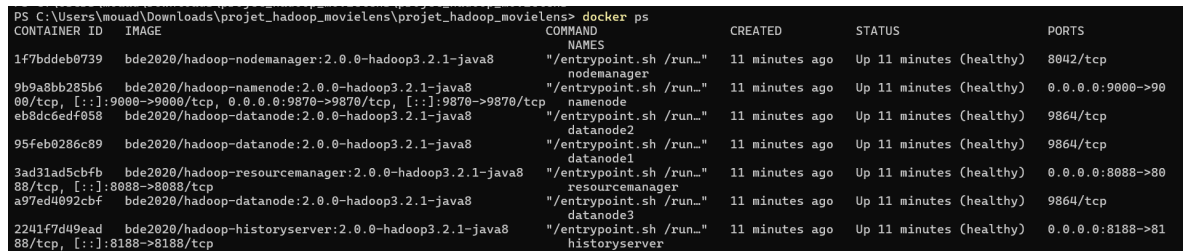
ResourceManager, le NodeManager et le HistoryServer.

4.3 Vérification des conteneurs

La commande suivante permet de vérifier que tous les conteneurs sont démarrés :

```
docker ps
```

Les conteneurs doivent apparaître avec l'état **healthy**.



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
1f7bddeb0739	bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8	NAMES "/entrypoint.sh /run_"	11 minutes ago	Up 11 minutes (healthy)	8042/tcp
9b9a8bb285b6	bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8	nodemanager "/entrypoint.sh /run_"	11 minutes ago	Up 11 minutes (healthy)	0.0.0.0:9000->9000/tcp, [::]:9000->9000/tcp, 0.0.0.0:9870->9870/tcp, [::]:9870->9870/tcp
eb8dc6edf058	bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8	namenode "/entrypoint.sh /run_"	11 minutes ago	Up 11 minutes (healthy)	9864/tcp
95feb0286c89	bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8	datanode2 "/entrypoint.sh /run_"	11 minutes ago	Up 11 minutes (healthy)	9864/tcp
3ad31ad5cbfb	bde2020/hadoop-resource-manager:2.0.0-hadoop3.2.1-java8	datanode1 "/entrypoint.sh /run_"	11 minutes ago	Up 11 minutes (healthy)	0.0.0.0:8088->8088/tcp, [::]:8088->8088/tcp
a97ed4092cbf	bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8	resource-manager "/entrypoint.sh /run_"	11 minutes ago	Up 11 minutes (healthy)	9864/tcp
2241f7d49ead	bde2020/hadoop-historyserver:2.0.0-hadoop3.2.1-java8	datanode3 "/entrypoint.sh /run_"	11 minutes ago	Up 11 minutes (healthy)	0.0.0.0:8188->8188/tcp, [::]:8188->8188/tcp

FIGURE 4.2 – Vérification des conteneurs Hadoop avec docker ps

5 Préparation de l'environnement

5.1 Structure du projet

La structure du dossier du projet est la suivante :

```
projet_hadoop_movielens/  
|-- docker-compose.yml  
|-- hadoop.env  
|-- java_src/  
|   |-- MovieLensAnalysis.java  
|-- run_movielens.sh  
|-- visualisation_movielens_clean.py  
|-- data/  
|   |-- ratings.csv  
|   |-- movies.csv  
|   |-- tags.csv  
|   |-- links.csv  
|-- results/  
|   |-- avg_rating.txt  
|   |-- ratings_dist.txt  
|   |-- top_tags.txt  
|   |-- popularity.txt  
|-- screenshots/
```

5.2 Démarrage du cluster

Le cluster est lancé avec la commande :

```
docker compose up -d
```

```
C:\Users\woud\Downloads>project_hadoop_movielens\project_hadoop_movielens> docker-compose up -d
time="2026-06-18T19:34:49+01:00" level=warning msg="C:\Users\woud\Downloads\project_hadoop_movielens\project_hadoop_movielens\docker-compose.yml: the a
tribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
time="2026-06-18T19:34:49+01:00" level=warning msg="The \"HADOOP_MAPRED_HOME\" variable is not set. Defaulting to a blank string."
time="2026-06-18T19:34:49+01:00" level=warning msg="The \"HADOOP_MAPRED_HOME\" variable is not set. Defaulting to a blank string."
time="2026-06-18T19:34:49+01:00" level=warning msg="The \"HADOOP_MAPRED_HOME\" variable is not set. Defaulting to a blank string."
time="2026-06-18T19:34:49+01:00" level=warning msg="The \"HADOOP_MAPRED_HOME\" variable is not set. Defaulting to a blank string."
time="2026-06-18T19:34:49+01:00" level=warning msg="The \"HADOOP_MAPRED_HOME\" variable is not set. Defaulting to a blank string."
time="2026-06-18T19:34:49+01:00" level=warning msg="The \"HADOOP_MAPRED_HOME\" variable is not set. Defaulting to a blank string."
time="2026-06-18T19:34:49+01:00" level=warning msg="The \"HADOOP_MAPRED_HOME\" variable is not set. Defaulting to a blank string."
time="2026-06-18T19:34:49+01:00" level=warning msg="The \"HADOOP_MAPRED_HOME\" variable is not set. Defaulting to a blank string."
time="2026-06-18T19:34:49+01:00" level=warning msg="The \"HADOOP_MAPRED_HOME\" variable is not set. Defaulting to a blank string."
time="2026-06-18T19:34:49+01:00" level=warning msg="The \"HADOOP_MAPRED_HOME\" variable is not set. Defaulting to a blank string."
time="2026-06-18T19:34:49+01:00" level=warning msg="The \"HADOOP_MAPRED_HOME\" variable is not set. Defaulting to a blank string."
time="2026-06-18T19:34:49+01:00" level=warning msg="The \"HADOOP_MAPRED_HOME\" variable is not set. Defaulting to a blank string."
time="2026-06-18T19:34:49+01:00" level=warning msg="The \"HADOOP_MAPRED_HOME\" variable is not set. Defaulting to a blank string."
time="2026-06-18T19:34:49+01:00" level=warning msg="The \"HADOOP_MAPRED_HOME\" variable is not set. Defaulting to a blank string."
[+] up 21/28
- Image bde2829/hadoop-resourcemanager:2.0.0-hadoop3.2.1-java8 [*****] 565.1MB / 565.1MB Pulling 106.3s
- Image bde2829/hadoop-namenode:2.0.0-hadoop3.2.1-java8 [*****] 565.1MB / 565.1MB Pulling 106.3s
- Image bde2829/hadoop-datanode:2.0.0-hadoop3.2.1-java8 [*****] 565.1MB / 565.1MB Pulling 106.3s
[+] up 48/112/28/hadoop-datanode:2.0.0-hadoop3.2.1-java8 [*****] 565
✓Image bde2829/hadoop-resourcemanager:2.0.0-hadoop3.2.1-java8 Pulled 275.3s
✓Image bde2829/hadoop-namenode:2.0.0-hadoop3.2.1-java8 Pulled 275.9s
✓Image bde2829/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8 Pulled 275.3s
✓Image bde2829/hadoop-datanode:2.0.0-hadoop3.2.1-java8 Pulled 275.3s
✓Image bde2829/hadoop-historyserver:2.0.0-hadoop3.2.1-java8 Pulled 276.0s
✓Network project_hadoop_movielens_hadoop Created 0.6s
✓Volume projet_hadoop_movielens_hadoop_datanode2 Created 0.8s
✓Volume projet_hadoop_movielens_hadoop_datanode3 Created 0.8s
✓Volume projet_hadoop_movielens_hadoop_historyserver Created 0.1s
✓Volume projet_hadoop_movielens_hadoop_namenode Created 0.2s
✓Volume projet_hadoop_movielens_hadoop_datanode1 Created 0.8s
✓Container resourcemanager Started 13.5s
✓Container datanode3 Started 15.4s
✓Container datanode2 Started 15.2s
✓Container nodemanager Started 12.9s
✓Container namenode Started 15.1s
✓Container datanode1 Started 12.9s
✓Container historyserver Started 13.9s
```

Après le démarrage, on vérifie l'état des services :

```
docker ps
```

Les interfaces web disponibles sont :

- NameNode HDFS : <http://localhost:9870>
- YARN ResourceManager : <http://localhost:8088>
- HistoryServer : <http://localhost:8188>

5.3 Vérification du chemin Hadoop

Une erreur rencontrée pendant le projet concernait le chemin Hadoop utilisé par MapReduce. Nous avons donc vérifié le chemin de Hadoop dans le conteneur `namenode` :

```
docker exec namenode printenv HADOOP_HOME
```

Résultat obtenu :

```
/opt/hadoop-3.2.1
```

```

mouad@DESKTOP-K72MJC8 MINGW64 ~/Downloads/projet_hadoop_moielens/projet_hadoop_moielens (master)
$ docker exec namenode bash -c "echo \$HADOOP_HOME"
/opt/hadoop-3.2.1
mouad@DESKTOP-K72MJC8 MINGW64 ~/Downloads/projet_hadoop_moielens/projet_hadoop_moielens (master)

```

6 Configuration Hadoop

6.1 Configuration HDFS

Dans le fichier `hadoop.env`, la configuration HDFS indique que le système de fichiers par défaut est :

```
CORE_CONF_fs_defaultFS=hdfs://namenode:9000
```

La réplication HDFS est configurée à 3 :

```
HDFS_CONF_dfs_replication=3
```

Cela signifie que chaque bloc de données est théoriquement répliqué sur trois DataNodes, ce qui améliore la tolérance aux pannes.

6.2 Configuration YARN et MapReduce

YARN est utilisé comme gestionnaire de ressources :

```
MAPRED_CONF_mapreduce_framework_name=yarn
```

Pendant le projet, une erreur importante est apparue lors du lancement des jobs MapReduce :

```
Error: Could not find or load main class org.apache.hadoop.mapreduce.v2.app.  
MRAppMaster
```

Cette erreur signifie que YARN n'arrivait pas à trouver la classe principale de l'ApplicationMaster MapReduce. La cause était l'absence de la variable `HADOOP_MAPRED_HOME` dans la configuration MapReduce.

La correction appliquée dans `hadoop.env` est la suivante :

```
MAPRED_CONF_yarn_app_mapreduce_am_env=HADOOP_MAPRED_HOME=/opt/hadoop-3.2.1  
MAPRED_CONF_mapreduce_map_env=HADOOP_MAPRED_HOME=/opt/hadoop-3.2.1  
MAPRED_CONF_mapreduce_reduce_env=HADOOP_MAPRED_HOME=/opt/hadoop-3.2.1  
MAPRED_CONF_mapreduce_application_classpath=/opt/hadoop-3.2.1/share/hadoop/  
mapreduce*/,/opt/hadoop-3.2.1/share/hadoop/mapreduce/lib/*
```

```
YARN_CONF_yarn_application_classpath=/opt/hadoop-3.2.1/etc/hadoop,/opt/hadoop-3.2.1/share/hadoop/common/*,/opt/hadoop-3.2.1/share/hadoop/common/lib/*,/opt/hadoop-3.2.1/share/hadoop/hdfs/*,/opt/hadoop-3.2.1/share/hadoop/hdfs/lib/*,/opt/hadoop-3.2.1/share/hadoop/mapreduce/*,/opt/hadoop-3.2.1/share/hadoop/mapreduce/lib/*,/opt/hadoop-3.2.1/share/hadoop/yarn/*,/opt/hadoop-3.2.1/share/hadoop/yarn/lib/*
```

Après cette correction, les conteneurs ont été recréés :

```
docker compose down
docker compose up -d --force-recreate
```

La présence de la configuration a été vérifiée avec :

```
docker exec namenode bash -c "grep -n 'HADOOP_MAPRED_HOME' /etc/hadoop/mapred-site.xml"
docker exec namenode bash -c "grep -n 'mapreduce.application.classpath' /etc/hadoop/mapred-site.xml"
```

```
moaad@DESKTOP-K72MJC8 MINGW64 ~/Downloads/projet_hadoop_movielen/projet_hadoop_movielen (master)
$ cp hadoop.env hadoop.env.bak

moaad@DESKTOP-K72MJC8 MINGW64 ~/Downloads/projet_hadoop_movielen/projet_hadoop_movielen (master)
$ grep -v -E '^(MAPRED_CONF_yarn_app_mapreduce_am_env|MAPRED_CONF_mapreduce_map_env|MAPRED_CONF_mapreduce_reduce_env|MAPRED_CONF_mapreduce_application_classpath|YARN_CONF_yarn_application_classpath)=' hadoop.env.bak > hadoop.env

moaad@DESKTOP-K72MJC8 MINGW64 ~/Downloads/projet_hadoop_movielen/projet_hadoop_movielen (master)
$ cat >> hadoop.env <<'EOF'

# FIX MRAppMaster / MapReduce on YARN
MAPRED_CONF_yarn_app_mapreduce_am_env=HADOOP_MAPRED_HOME=/opt/hadoop-3.2.1
MAPRED_CONF_mapreduce_map_env=HADOOP_MAPRED_HOME=/opt/hadoop-3.2.1
MAPRED_CONF_mapreduce_reduce_env=HADOOP_MAPRED_HOME=/opt/hadoop-3.2.1
MAPRED_CONF_mapreduce_application_classpath=/opt/hadoop-3.2.1/share/hadoop/mapreduce/*,/opt/hadoop-3.2.1/share/hadoop/mapreduce/lib/*
YARN_CONF_yarn_application_classpath=/opt/hadoop-3.2.1/etc/hadoop,/opt/hadoop-3.2.1/share/hadoop/common/*,/opt/hadoop-3.2.1/share/hadoop/common/lib/*,/opt/hadoop-3.2.1/share/hadoop/hdfs/*,/opt/hadoop-3.2.1/share/hadoop/hdfs/lib/*,/opt/hadoop-3.2.1/share/hadoop/mapreduce/*,/opt/hadoop-3.2.1/share/hadoop/mapreduce/lib/*,/opt/hadoop-3.2.1/share/hadoop/yarn/*,/opt/hadoop-3.2.1/share/hadoop/yarn/lib/*
EOF

moaad@DESKTOP-K72MJC8 MINGW64 ~/Downloads/projet_hadoop_movielen/projet_hadoop_movielen (master)
```

7 Dataset MovieLens ml-32m

7.1 Présentation du dataset

Le dataset utilisé est **MovieLens ml-32m**. Il contient des données de notation de films par des utilisateurs. Les fichiers principaux sont :

- `ratings.csv` : notes des utilisateurs ;
- `movies.csv` : titres et genres des films ;
- `tags.csv` : tags textuels ;
- `links.csv` : liens vers IMDb et TMDb.

7.2 Comparaison des fichiers

TABLE 7.1 – Comparaison des fichiers MovieLens utilisés

Fichier	Taille observée	Rôle	Utilisation
<code>ratings.csv</code>	837 Mo	Notes utilisateurs	Traitement MapReduce principal
<code>tags.csv</code>	70 Mo	Tags utilisateurs	Traitement MapReduce
<code>movies.csv</code>	4.1 Mo	Titres et genres	Enrichissement des résultats
<code>links.csv</code>	quelques Mo	Identifiants externes	Non prioritaire pour l'analyse

Le fichier `ratings.csv` est le plus volumineux. Il contient plus de 32 millions de notes. Il est donc le fichier le plus pertinent pour démontrer la puissance du traitement distribué Hadoop.

7.3 Vérification locale des fichiers

Avant de lancer le traitement, le script vérifie la présence des fichiers dans le dossier `data/` :

```
ratings.csv : 837M
movies.csv  : 4.1M
tags.csv    : 70M
```

OK

```
mouad@DESKTOP-K72MJC8 MINGW64 ~/Downloads/projet_hadoop_movielens/projet_hadoop_movielens (master)
$ bash run_movielens.sh
=====
Mini-Projet Hadoop -- Analyse MovieLens ml-32m
=====
[1/6] Checking data files...
ratings.csv : 837M
movies.csv  : 4.1M
tags.csv    : 70M
OK
```

8 Chargement des données dans HDFS

8.1 Copie des fichiers vers le conteneur NameNode

Le script copie les fichiers locaux vers le conteneur namenode :

```
docker cp ./data/ratings.csv namenode:/tmp/ratings.csv
docker cp ./data/movies.csv namenode:/tmp/movies.csv
docker cp ./data/tags.csv namenode:/tmp/tags.csv
```

8.2 Création des dossiers HDFS

Les dossiers HDFS sont créés avec les commandes suivantes :

```
hdfs dfs -mkdir -p /user/root/movielens/input/ratings
hdfs dfs -mkdir -p /user/root/movielens/input/tags
```

8.3 Chargement dans HDFS

Les fichiers sont ensuite placés dans HDFS :

```
hdfs dfs -put -f /tmp/ratings.csv /user/root/movielens/input/ratings/
hdfs dfs -put -f /tmp/tags.csv /user/root/movielens/input/tags/
```

8.4 Vérification dans HDFS

Résultat attendu :

```
-rw-r--r--  3 root supergroup    836.4 M /user/root/movielens/input/ratings/
ratings.csv
-rw-r--r--  3 root supergroup    69.0 M /user/root/movielens/input/tags/tags.csv
```

```
HDFS verification:
Found 1 items
-rw-r--r--  3 root supergroup    836.4 M 2026-06-10 21:16 /user/root/movielens/input/ratings/ratings.c
sv
Found 1 items
-rw-r--r--  3 root supergroup    69.0 M 2026-06-10 21:16 /user/root/movielens/input/tags/tags.csv
OK
```

9 Programme MapReduce

9.1 Présentation du programme Java

Le programme `MovieLensAnalysis.java` contient quatre jobs MapReduce :

1. `avg_rating` : calcul de la note moyenne par film ;
2. `ratings_dist` : calcul de la distribution des notes ;
3. `top_tags` : calcul des tags les plus utilisés ;
4. `popularity` : calcul du nombre de notes par film.

9.2 Job 1 : note moyenne par film

Ce job lit le fichier `ratings.csv`. Le Mapper extrait le `movieId` et la note. Le Reducer calcule la moyenne des notes pour chaque film.

TABLE 9.1 – Principe du job `avg_rating`

Entrée	Clé intermédiaire	Sortie
<code>userId,movieId,rating,timestamp</code>	<code>movieId</code>	<code>movieId, moyenne</code>

9.3 Job 2 : distribution des notes

Ce job compte le nombre d'apparitions de chaque note : 0.5, 1.0, 1.5, jusqu'à 5.0.

TABLE 9.2 – Principe du job `ratings_dist`

Entrée	Clé intermédiaire	Sortie
<code>rating</code>	<code>rating</code>	<code>rating, nombre</code>

9.4 Job 3 : tags les plus fréquents

Ce job lit `tags.csv`. Le Mapper extrait le tag, le convertit en minuscule, puis le Reducer compte le nombre d'occurrences.

9.5 Job 4 : popularité des films

Ce job compte le nombre de notes reçues par chaque film. Un film avec un grand nombre de notes peut être considéré comme très populaire.

9.6 Utilisation du Combiner

Pour les jobs de comptage, un **Combiner** est utilisé afin de réduire la quantité de données envoyée pendant la phase de shuffle. Cela améliore les performances du traitement MapReduce.

10 Compilation et exécution

10.1 Compilation du programme Java

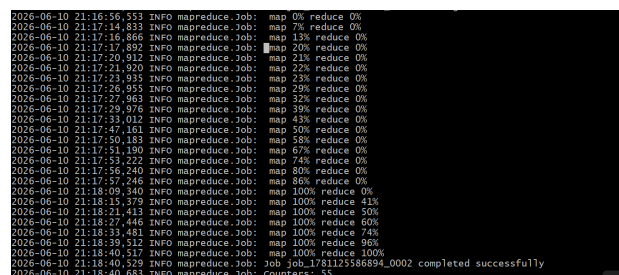
Le script copie le fichier Java dans le conteneur namenode, puis le compile avec le class-path Hadoop :

```
docker cp ./java_src/MovieLensAnalysis.java namenode:/tmp/MovieLensAnalysis.java

docker exec namenode bash -c "
    cd /tmp
    HADOOP_CP=$(hadoop classpath)
    javac -encoding UTF-8 -classpath \"${HADOOP_CP}\" MovieLensAnalysis.java
    jar cf movielens.jar MovieLensAnalysis*.class
"
```

Résultat obtenu :

```
Compilation OK - .class files found
movielens.jar created successfully
```



```
2026-06-10 21:16:56,553 INFO mapreduce.Job: map 0% reduce 0%
2026-06-10 21:17:14,833 INFO mapreduce.Job: map 7% reduce 0%
2026-06-10 21:17:16,866 INFO mapreduce.Job: map 13% reduce 0%
2026-06-10 21:17:17,892 INFO mapreduce.Job: map 20% reduce 0%
2026-06-10 21:17:20,912 INFO mapreduce.Job: map 21% reduce 0%
2026-06-10 21:17:21,920 INFO mapreduce.Job: map 22% reduce 0%
2026-06-10 21:17:23,935 INFO mapreduce.Job: map 23% reduce 0%
2026-06-10 21:17:26,955 INFO mapreduce.Job: map 29% reduce 0%
2026-06-10 21:17:27,963 INFO mapreduce.Job: map 32% reduce 0%
2026-06-10 21:17:29,976 INFO mapreduce.Job: map 39% reduce 0%
2026-06-10 21:17:33,012 INFO mapreduce.Job: map 43% reduce 0%
2026-06-10 21:17:47,161 INFO mapreduce.Job: map 50% reduce 0%
2026-06-10 21:17:50,183 INFO mapreduce.Job: map 58% reduce 0%
2026-06-10 21:17:51,190 INFO mapreduce.Job: map 67% reduce 0%
2026-06-10 21:17:53,222 INFO mapreduce.Job: map 74% reduce 0%
2026-06-10 21:17:56,240 INFO mapreduce.Job: map 80% reduce 0%
2026-06-10 21:17:57,246 INFO mapreduce.Job: map 86% reduce 0%
2026-06-10 21:18:09,340 INFO mapreduce.Job: map 100% reduce 0%
2026-06-10 21:18:15,379 INFO mapreduce.Job: map 100% reduce 41%
2026-06-10 21:18:21,413 INFO mapreduce.Job: map 100% reduce 50%
2026-06-10 21:18:27,446 INFO mapreduce.Job: map 100% reduce 60%
2026-06-10 21:18:33,481 INFO mapreduce.Job: map 100% reduce 74%
2026-06-10 21:18:39,512 INFO mapreduce.Job: map 100% reduce 96%
2026-06-10 21:18:40,517 INFO mapreduce.Job: map 100% reduce 100%
2026-06-10 21:18:40,529 INFO mapreduce.Job: Job job_1781125586894_0002 completed successfully
2026-06-10 21:18:40,683 INFO mapreduce.Job: Context: 51
```

10.2 Test MapReduce WordCount

Avant d'exécuter les traitements MovieLens, un test WordCount a été lancé afin de vérifier que YARN et MapReduce fonctionnent correctement :

```
docker exec namenode /opt/hadoop-3.2.1/bin/hdfs dfs -rm -r -f /tmp/wc /tmp/wc_out
docker exec namenode bash -c "printf 'hello hadoop hello movie\n' > /tmp/wc.txt"
docker exec namenode /opt/hadoop-3.2.1/bin/hdfs dfs -mkdir -p /tmp/wc
```

```

PS C:\Users\mouad\Downloads\projet_hadoop_movielens\projet_hadoop_movielens> docker exec namenode /opt/hadoop-3.2.1/bin/hdfs dfs -cat /tmp/wc_out/part-r-000000
2026-06-10 21:14:04,826 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted = false, remoteHostTrusted = false
hadoop 1
hello 2
movie 1
PS C:\Users\mouad\Downloads\projet_hadoop_movielens\projet_hadoop_movielens>

```

```

docker exec namenode /opt/hadoop-3.2.1/bin/hdfs dfs -put -f /tmp/wc.txt /tmp/wc/
docker exec namenode /opt/hadoop-3.2.1/bin/hadoop jar /opt/hadoop-3.2.1/share/
    hadoop/mapreduce/hadoop-mapreduce-examples-3.2.1.jar wordcount /tmp/wc /tmp/
    wc_out
docker exec namenode /opt/hadoop-3.2.1/bin/hdfs dfs -cat /tmp/wc_out/part-r-000000

```

Résultat obtenu :

```

hadoop 1
hello 2
movie 1

```

10.3 Exécution du pipeline complet

Le pipeline complet est lancé avec :

```
bash run_movielens.sh
```

Ce script réalise automatiquement les étapes suivantes :

1. vérification des fichiers CSV ;
2. copie des fichiers vers le conteneur namenode ;
3. chargement dans HDFS ;
4. compilation du programme Java ;
5. exécution des quatre jobs MapReduce ;
6. récupération des résultats depuis HDFS.

10.4 Commandes des jobs MapReduce

Les jobs sont exécutés avec la commande générale suivante :

```
hadoop jar /tmp/movielens.jar MovieLensAnalysis <job> <input> <output>
```

Exemples :

```

hadoop jar /tmp/movielens.jar MovieLensAnalysis avg_rating \
/user/root/movielens/input/ratings \
/user/root/movielens/output/avg_rating

hadoop jar /tmp/movielens.jar MovieLensAnalysis ratings_dist \

```

```
/user/root/movielens/input/ratings \  
/user/root/movielens/output/ratings_dist  
  
hadoop jar /tmp/movielens.jar MovieLensAnalysis top_tags \  
/user/root/movielens/input/tags \  
/user/root/movielens/output/top_tags  
  
hadoop jar /tmp/movielens.jar MovieLensAnalysis popularity \  
/user/root/movielens/input/ratings \  
/user/root/movielens/output/popularity
```

11 Résultats obtenus

11.1 Fichiers de sortie

Après l'exécution, les résultats sont récupérés dans le dossier local **results/** :

```
results/  
|-- avg_rating.txt  
|-- ratings_dist.txt  
|-- top_tags.txt  
|-- popularity.txt
```

Les résultats sont récupérés depuis HDFS avec :

```
hdfs dfs -getmerge /user/root/movielens/output/avg_rating /tmp/avg_rating.txt  
hdfs dfs -getmerge /user/root/movielens/output/ratings_dist /tmp/ratings_dist.txt  
hdfs dfs -getmerge /user/root/movielens/output/top_tags /tmp/top_tags.txt  
hdfs dfs -getmerge /user/root/movielens/output/popularity /tmp/popularity.txt
```



screenshots/09_results_folder.png

FIGURE 11.1 – Fichiers de résultats générés par les jobs MapReduce

11.2 Distribution des notes

La distribution des notes obtenue est la suivante :

TABLE 11.1 – Distribution des notes MovieLens

Note	Nombre de ratings
0.5	525,132
1.0	946,675
1.5	531,063
2.0	2,028,622
2.5	1,685,386
3.0	6,054,990
3.5	4,290,105
4.0	8,367,654
4.5	2,974,000
5.0	4,596,577

Le nombre total de ratings est :

$$525132+946675+531063+2028622+1685386+6054990+4290105+8367654+2974000+4596577 = 32\,000\,200$$

La note la plus fréquente est **4.0**, avec environ **8.4 millions** de ratings.

11.3 Films les plus populaires

Le job `popularity` compte le nombre de notes par film. Les films les plus populaires sont ceux qui ont reçu le plus grand nombre de ratings.

Dans notre résultat, le film le plus noté est :

`The Shawshank Redemption (1994) : 102,929 ratings`

Cela indique que ce film est l'un des plus présents dans les interactions utilisateurs du dataset.

11.4 Tags les plus utilisés

Le job `top_tags` calcule les tags les plus fréquents. Les premiers tags observés sont :

- `sci-fi`
- `atmospheric`
- `action`
- `comedy`
- `funny`

Ces tags donnent une idée des catégories, émotions ou caractéristiques les plus souvent associées aux films.

11.5 Note moyenne par film

Le job `avg_rating` calcule la note moyenne pour chaque film. Pour éviter les films ayant très peu de votes, la visualisation finale affiche les meilleurs films parmi ceux ayant au moins 10 000 ratings.

Le meilleur résultat obtenu est :

The Shawshank Redemption (1994) : 4.40/5
--

12 Visualisation des résultats

12.1 Génération du dashboard

Les résultats MapReduce sont visualisés avec un script Python :

```
python visualisation_movielens_clean.py
```

Ce script lit les fichiers du dossier **results/** et génère l'image :

```
movielens_dashboard_clean.png
```

12.2 Dashboard final



FIGURE 12.1 – Dashboard final des résultats MovieLens

Le dashboard présente quatre visualisations :

1. distribution globale des notes ;
2. top 15 des films les plus notés ;
3. top 15 des tags les plus utilisés ;
4. top 15 des meilleures notes moyennes pour les films avec au moins 10 000 ratings.

13 Interprétation des résultats

13.1 Analyse de la distribution des notes

La distribution montre que les utilisateurs donnent majoritairement des notes positives. La note **4.0** est la plus fréquente, suivie par **3.0** et **5.0**. La moyenne globale est d'environ **3.54/5**.

Cela peut indiquer que les utilisateurs ont tendance à noter les films qu'ils ont appréciés, ou que les films présents dans le dataset sont généralement bien évalués.

13.2 Analyse de la popularité

Les films les plus notés sont principalement des films très connus, comme :

- The Shawshank Redemption ;
- Forrest Gump ;
- Pulp Fiction ;
- Matrix ;
- Silence of the Lambs.

Ces films sont populaires dans le dataset car ils ont reçu un grand nombre de notes. Il faut noter que la popularité ne signifie pas forcément la meilleure qualité, mais plutôt une forte interaction des utilisateurs.

13.3 Analyse des tags

Les tags les plus fréquents montrent les préférences et les descriptions utilisées par les utilisateurs. Les tags comme `sci-fi`, `action`, `comedy` et `atmospheric` montrent que les utilisateurs décrivent souvent les films par genre, ambiance ou émotion.

13.4 Analyse des meilleures notes moyennes

Pour éviter les biais liés aux films ayant très peu de votes, nous avons affiché les films ayant au moins 10 000 ratings. Cela permet d'obtenir un classement plus fiable.

Le film *The Shawshank Redemption* apparaît à la fois comme très populaire et très bien noté, ce qui montre sa forte appréciation dans le dataset.

14 Avantages et limites de Hadoop

14.1 Avantages

L'utilisation de Hadoop présente plusieurs avantages :

- **Scalabilité** : Hadoop peut traiter des fichiers beaucoup plus grands en ajoutant des noeuds au cluster.
- **Tolérance aux pannes** : HDFS réplique les blocs de données sur plusieurs Data-Nodes.
- **Traitement distribué** : MapReduce divise le travail entre plusieurs tâches Map et Reduce.
- **Adapté aux gros volumes** : le fichier `ratings.csv` contient plus de 32 millions de ratings.
- **Automatisation** : le pipeline complet est exécuté par un script shell.

14.2 Limites et difficultés rencontrées

Plusieurs difficultés ont été rencontrées :

- configuration réseau difficile avec les machines virtuelles ;
- erreurs de configuration YARN/MapReduce ;
- problème `MRAppMaster` dû à une variable d'environnement manquante ;
- différence de comportement entre PowerShell et Git Bash pour les commandes contenant des guillemets ou des variables ;
- problèmes d'encodage lors de la génération de la visualisation sous Windows.

14.3 Limite de l'approche Docker

Docker simule un cluster multi-noeuds, mais tous les conteneurs tournent sur la même machine physique. Donc, cette architecture permet de comprendre et démontrer Hadoop, mais elle ne remplace pas totalement un vrai cluster réparti sur plusieurs machines physiques.

15 Commandes principales utilisées

15.1 Commandes Docker

```
docker compose up -d
docker ps
docker compose down
docker compose up -d --force-recreate
```

15.2 Vérification de Hadoop

```
docker exec namenode printenv HADOOP_HOME
docker exec namenode bash -c "grep -n 'HADOOP_MAPRED_HOME' /etc/hadoop/mapred-site.xml"
docker exec namenode bash -c "grep -n 'mapreduce.application.classpath' /etc/hadoop/mapred-site.xml"
```

15.3 Commandes HDFS

```
hdfs dfs -mkdir -p /user/root/movielens/input/ratings
hdfs dfs -mkdir -p /user/root/movielens/input/tags
hdfs dfs -put -f /tmp/ratings.csv /user/root/movielens/input/ratings/
hdfs dfs -put -f /tmp/tags.csv /user/root/movielens/input/tags/
hdfs dfs -ls -h /user/root/movielens/input/ratings/
hdfs dfs -ls -h /user/root/movielens/input/tags/
```

15.4 Test WordCount

```
docker exec namenode /opt/hadoop-3.2.1/bin/hdfs dfs -rm -r -f /tmp/wc /tmp/wc_out
docker exec namenode bash -c "printf 'hello hadoop hello movie\n' > /tmp/wc.txt"
docker exec namenode /opt/hadoop-3.2.1/bin/hdfs dfs -mkdir -p /tmp/wc
```

```
docker exec namenode /opt/hadoop-3.2.1/bin/hdfs dfs -put -f /tmp/wc.txt /tmp/wc/
docker exec namenode /opt/hadoop-3.2.1/bin/hadoop jar /opt/hadoop-3.2.1/share/
    hadoop/mapreduce/hadoop-mapreduce-examples-3.2.1.jar wordcount /tmp/wc /tmp/
    wc_out
docker exec namenode /opt/hadoop-3.2.1/bin/hdfs dfs -cat /tmp/wc_out/part-r-00000
```

15.5 Commande du pipeline MovieLens

```
bash run_movielens.sh
```

15.6 Commande de visualisation

```
python visualisation_movielens_clean.py
```

16 Captures d'écran à inclure

Pour rendre le rapport clair et complet, les captures suivantes doivent être ajoutées dans le dossier `screenshots/` :

N	Nom du fichier	Contenu de la capture
1	01_docker_ps.png	Résultat de <code>docker ps</code> montrant tous les conteneurs en état <code>healthy</code> .
2	02_hadoop_home.png	Résultat de <code>docker exec namenode printenv HADOOP_HOME</code> .
3	03_mapred_fix.png	Vérification de <code>HADOOP_MAPRED_HOME</code> dans <code>mapred-site.xml</code> .
4	04_check_data.png	Étape du script montrant les tailles des fichiers <code>ratings.csv</code> , <code>movies.csv</code> , <code>tags.csv</code> .
5	05_hdfs_files.png	Résultat de <code>hdfs dfs -ls -h</code> montrant les fichiers dans HDFS.
6	06_compilation.png	Compilation Java réussie et création de <code>movielens.jar</code> .
7	07_wordcount_success.png	Test WordCount réussi avec le résultat <code>hadoop 1, hello 2, movie 1</code> .
8	08_yarn_jobs.png	Interface YARN http://localhost:8088 montrant les applications MapReduce.
9	09_results_folder.png	Dossier <code>results/</code> contenant les quatre fichiers de sortie.
10	10_dashboard_clean.png	Dashboard final généré par Python.
11	architecture_cluster.png	Schéma de l'architecture Hadoop Docker.

17 Conclusion

Ce mini-projet a permis d'installer et d'exécuter un cluster Hadoop multi-noeuds à l'aide de Docker. Malgré une première tentative avec des machines virtuelles, les difficultés de configuration réseau nous ont amenés à utiliser Docker comme solution plus légère et reproductible.

Le dataset MovieLens ml-32m a été chargé dans HDFS, puis analysé avec un programme MapReduce en Java. Quatre traitements ont été réalisés : la distribution des notes, la popularité des films, les tags les plus fréquents et la note moyenne par film.

Les résultats montrent que le fichier `ratings.csv`, avec plus de 32 millions de ratings, est le plus adapté pour démontrer la capacité de Hadoop à traiter de grands volumes de données. Les visualisations finales montrent une moyenne globale de **3.54/5**, une forte concentration des notes autour de **4.0**, et la présence de films très populaires comme *The Shawshank Redemption*, *Forrest Gump*, *Pulp Fiction* et *Matrix*.

Ce projet démontre donc l'intérêt de Hadoop pour le stockage distribué, le traitement parallèle et l'analyse de données volumineuses.