Université Cadi Ayyad

Faculté des Sciences Semlalia
Département d'Informatique

Projet de Big Data Labs

Pour la validation de module en Master Science de données.

L'analyse des films regardés sur PoliTV.

Réalisé par :

Omar BOUDELLAH && Mustapha AHRICHA

Encadré par :

Pr. EL ALAOUI Hasna
Professeur, Département informatique, FSSM Marrakech

Année Universitaire : 2023 / 2024

Table de matière

Table de figures

Chapitre 1: Introduction

1-Objectifs du projet :

Le projet vise à répondre aux besoins d'analyse spécifiques des responsables de PoliTV sur les films regardés. Les principales tâches à accomplir dans le cadre de ce projet consistent à concevoir une application unique, en utilisant à la fois Hadoop MapReduce (pour le premier point) et Spark avec RDD (pour le deuxième point), et à écrire le code Java et Python correspondant. Les objectifs spécifiques sont les suivants :

1. Films Regardés par un Seul Utilisateur au Cours de l'Année 2019 (Hadoop MapReduce).
2. Films Regardés Fréquemment mais Seulement Pendant un An au Cours des Cinq Dernières Années (Spark et RDD) .
3. Extraction des Films les Plus Populaires Depuis au Moins Deux Ans (Spark et RDD) .

2-Importance du projet dans le contexte du Big Data

les analyses que nous faisons permettront aux responsables de PoliTV de mieux comprendre les habitudes de visionnage de leurs utilisateurs, d'identifier les films populaires sur une période prolongée, et d'optimiser leurs recommandations de contenu. En répondant à ces exigences, notre application contribuera à l'amélioration de la plateforme PoliTV en fournissant des informations précieuses sur les préférences des utilisateurs et la popularité des films au fil du temps.

Chapitre 2: Cadre théorique et technologique

1- Big data

1.0 Revue de la littérature sur les technologies Big Data

La revue de la littérature sur les technologies Big Data est une section cruciale qui permet d'explorer les concepts, les défis, et les évolutions significatives dans le domaine du Big Data. Voici un exemple de contenu que vous pourriez inclure dans cette section :

1.1 Introduction au Big Data

Le Big Data, terme émergé au cours des dernières années, représente un changement fondamental dans la manière dont nous collectons, stockons, traitons et analysons les données. Il est caractérisé par le volume massif, la variété de formats, la vitesse de génération et la nécessité d'une vélocité de traitement rapide.

⟹ **Définition Big Data :** Traitement rapide de grands volume de données difficilement gérables avec les outils traditionnels (une base de données classique ou un seul ordinateur).

Le terme Big data focalise sur des données qui contient une plus grand variété, arrivant dans des volumes croissants et une plus de vitesse.

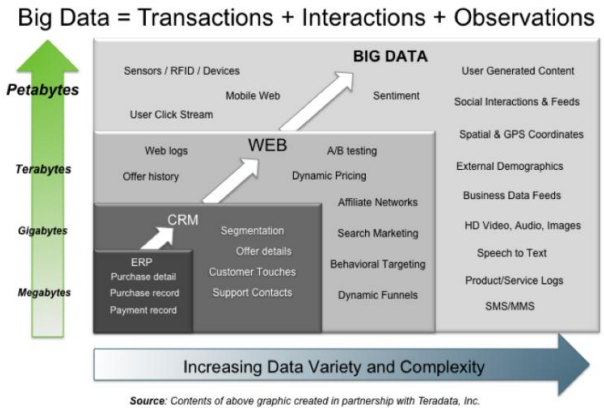


Figure 1 Big data

1.2 Caractéristiques du Big Data

1. **Volume** : Les données sont générées à une échelle exponentielle, dépassant les capacités des systèmes traditionnels de gestion de bases de données.
2. **Variété** : Les données proviennent de diverses sources telles que les réseaux sociaux, les capteurs, les transactions en ligne, etc., et peuvent être structurées, semi-structurées ou non structurées.
3. **Vélocité** : Les données doivent être traitées en temps réel ou quasi-réel, car elles sont générées rapidement.
4. **Véracité** : En raison de la diversité des sources, la qualité des données peut varier, nécessitant des techniques avancées de vérification.
5. **Valeur** : Au-delà de simplement collecter et stocker les données, il est crucial d'extraire des informations significatives et exploitables. La capacité à analyser ces données pour en extraire de la valeur, que ce soit pour prendre des décisions commerciales, améliorer les processus, ou comprendre les modèles, est essentielle. Cette valeur peut être obtenue par le biais d'analyses avancées, de l'apprentissage automatique ou d'autres techniques d'analyse de données.

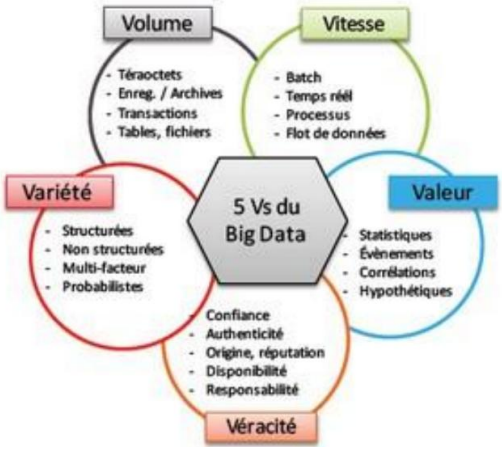


Figure 2 5 V de big data

1.3 Importance du Big Data

L'émergence du Big Data a un impact significatif dans divers domaines :

1. **Recherche Scientifique** : Facilite la gestion et l'analyse de grandes quantités de données expérimentales.
2. **Entreprises** : Permet une prise de décision basée sur les données, l'optimisation des processus et la personnalisation des services.
3. **Santé** : Facilite la recherche médicale, la gestion des dossiers médicaux et la surveillance des épidémies.
4. **Gouvernance** : Aide à l'analyse des tendances socio-économiques, à la prévention de la criminalité, etc.

1.4 Exemple use case d'un projet big data

Exemple - Usine 4.0

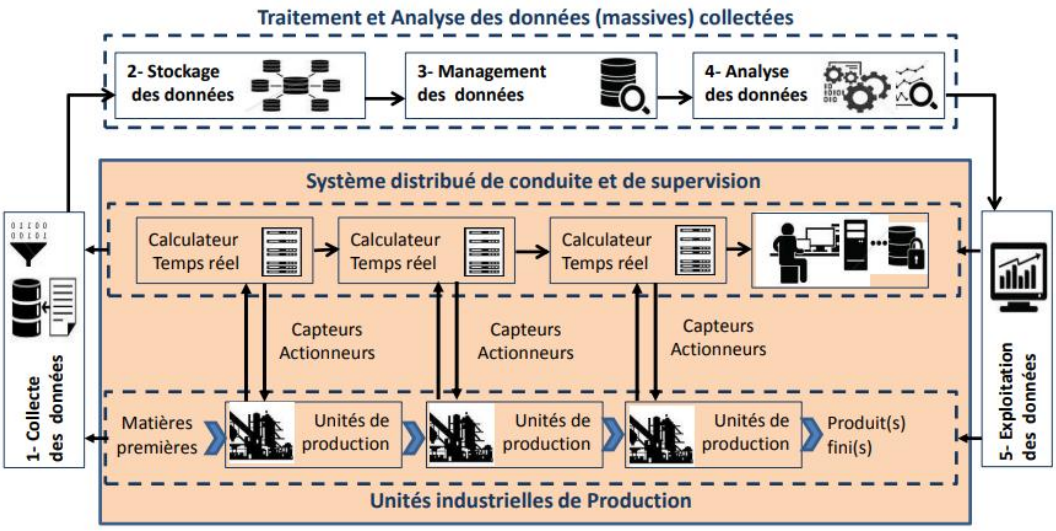


Figure 3 Big data au sein des unités de production

1.5 Sources de Big Data

Les principales sources de données comprennent :

1. Données des Médias Sociaux : Tweets, publications sur Facebook, vidéos sur YouTube, etc.
2. Capteurs et IoT (Internet des Objets) : Données générées par des appareils connectés.
3. Transactions en Ligne : Historique d'achats, transactions financières, etc.
4. Données Générées par l'Utilisateur : Recherches en ligne, commentaires, avis, etc.

1.6 Technologies de Big Data

Les technologies Big Data englobent un éventail d'outils et de techniques utilisés pour collecter, stocker, traiter et analyser ces énormes quantités de données.

Les principaux technologies clés associées au Big Data :

- Hadoop: Un framework open source qui permet de stocker et de traiter de gros volumes de données sur des clusters de serveurs. Il utilise le modèle de programmation MapReduce pour le traitement distribué.

Figure 4 Logo Hadoop

- Spark: Une plateforme de traitement de données rapide et polyvalente qui fonctionne en mémoire et peut traiter des charges de travail plus diverses que Hadoop. Il est souvent utilisé pour le traitement en temps réel et les analyses avancées.

Figure 5 Logo Spark

2- Hadoop

2.1 Introduction

Hadoop a évolué pour devenir l'un des piliers du traitement et du stockage de données massives à l'ère du Big Data. À ses débuts, Hadoop était principalement axé sur deux composants clés : le stockage et le traitement. Alors c’est quoi hadoop? Et quel est son role?

2.1.1 Définition

Hadoop est un framework open source conçu pour le stockage et le traitement distribué de grands ensembles de données sur des clusters d'ordinateurs. Il est principalement utilisé pour gérer et analyser de grandes quantités de données structurées ou non structurées, offrant ainsi une solution évolutive pour le traitement de données massives.

Il est conçu par Doug Cutting en 2004, il est introduit par Google en s'inspirant des solutions Google comme MapReduce, GoogleFS et BigTable.

Déployable et configurable très facilement.

2.1.2 Utilisabilité

L'utilisabilité de Hadoop dépend des besoins spécifiques de chaque projet :

1. **Traitement de gros volumes de données** : Hadoop est particulièrement efficace pour le traitement de grandes quantités de données réparties sur plusieurs nœuds dans un cluster. Si votre cas d'utilisation implique des volumes massifs de données, Hadoop peut être utile.
2. **Évolutivité** : Il offre une grande évolutivité horizontale, ce qui signifie que vous pouvez ajouter de nouveaux nœuds au cluster pour augmenter la capacité de stockage et de traitement.
3. **Tolérance aux pannes** : Hadoop est conçu pour gérer les pannes matérielles. Il réplique automatiquement les données sur plusieurs nœuds, assurant ainsi une certaine tolérance aux pannes.
4. **Adaptabilité** : Il est assez souple pour s'intégrer à d'autres outils et technologies. Cependant, sa configuration et sa gestion peuvent être complexes.
5. **Complexité** : La mise en place et la configuration initiales peuvent être complexes, nécessitant des compétences spécifiques en administration de systèmes distribués.
6. **Performance grâce au parallélisme des données** : Hadoop divise les tâches en petits morceaux et les exécute simultanément sur différents nœuds, exploitant ainsi le parallélisme pour accélérer le traitement des données massives.

2.1.3 Composants fondamentaux

Hadoop est constitué de 2 grandes parties :

HDFS (Hadoop Distributed File Système) : destiné pour le stockage distribué des données.

MapReduce (Distributed Programming Framework) : destine pour le traitement distribué des données.

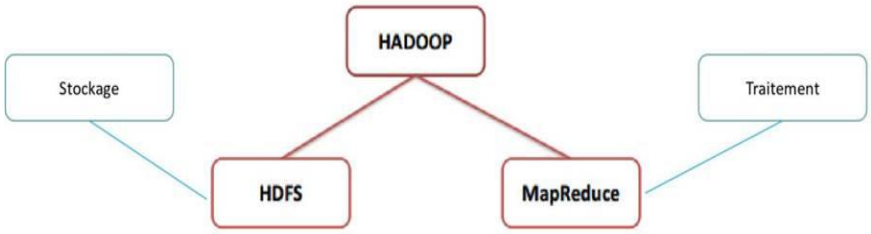


Figure 6 Composants fondamentaux

➡**Version Hadoop**

Il existe deux principales versions de Hadoop : Hadoop 1 et Hadoop 2.

2.2 Hadoop 1

2.2.1 Introduction

Hadoop 1 était la première version majeure du framework. Il comprenait deux composants principaux :

- le système de stockage Hadoop Distributed File System (HDFS) et le framework de traitement MapReduce.
- MapReduce était utilisé pour traiter et analyser les données stockées dans HDFS. Cependant, cette version avait quelques limitations, notamment en termes d'extensibilité et de types de traitement supportés.

2.2.2 Principe de base

Le principe de base de Hadoop 1 repose sur deux composants principaux : Hadoop Distributed File System (HDFS) et le framework de traitement MapReduce.

- ❖ Copier les données vers HDFS: les données sont divisées et stockées sur un ensemble se noeuds.
- ❖ Traiter les données là où elles sont stockées (MAP) et recueillir les réssultats (REDUCE).
- ❖ Copier les résultats vers HDFS.

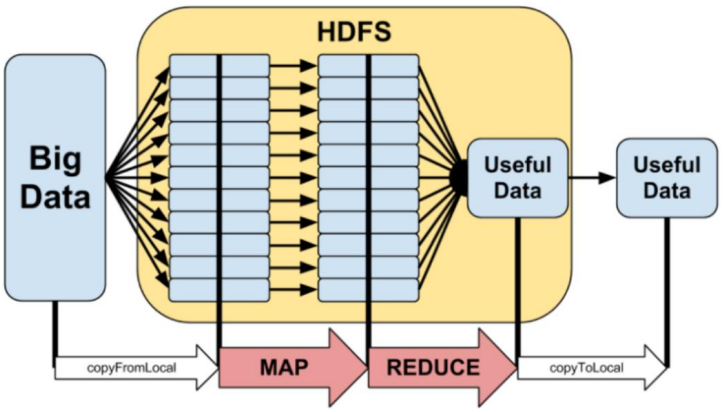


Figure 7 Principe de base

2.2.3 Noyaux de Hadoop

Hadoop Distributed File System (HDFS) :

HDFS est conçu pour stocker de très gros fichiers de données en les répartissant sur plusieurs nœuds dans un cluster. Il divise les fichiers en blocs et les réplique sur différents nœuds pour assurer la tolérance aux pannes et la disponibilité des données.
Il fonctionne selon un modèle maître-esclave, où un nœud maître, le NameNode, gère le système de fichiers en conservant les métadonnées, tandis que les nœuds esclaves, les DataNodes, stockent les données réelles.

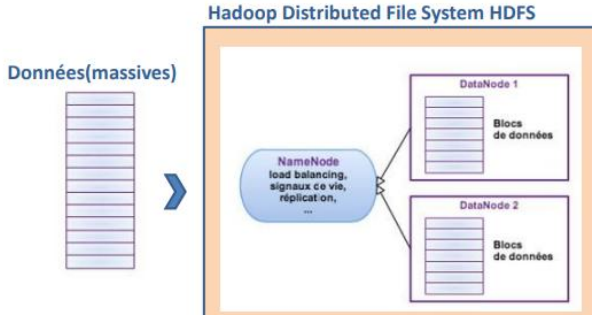
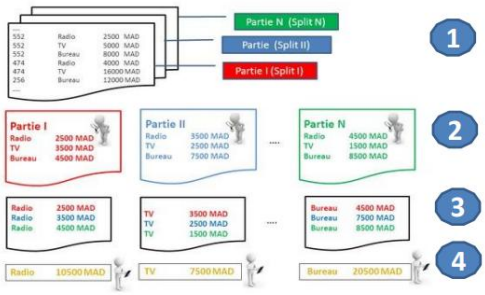


Figure 8 HDFS

MapReduce :
MapReduce est un modèle de programmation pour le traitement parallèle et distribué de grands ensembles de données. Il divise les tâches en deux phases principales : la phase de Map où les données sont traitées et filtrées, puis la phase de Reduce où les résultats intermédiaires sont combinés pour produire le résultat final.
MapReduce fonctionne en distribuant ces tâches sur les nœuds du cluster, ce qui permet un traitement parallèle et efficace des données stockées dans HDFS.

➡ **MapReduce: Exemple d'illustration**



1. Les données initiales sont scindées en lots (**split**)
2. Chaque lot est confié à un *Mapper* qui évalue la **fonction Map** sur chaque enregistrement
3. Les liste obtenues en sortie des *Mappers* sont concaténées, classées par valeur de la clé et scindées en lots par le Framework. C'est la phase dite du **Shuffle**.
4. La **fonction Reduce** exécutée par chaque *Reducer* agrège toutes les valeurs intermédiaires associées à une même clef.

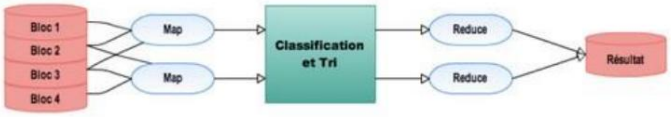


Figure 9 Exemple d'illustration

2.2.4 Infrastructure(s) logicielle(s) d'implémentation du pattern MapReduce

Job MapReduce: Application client qui s'exécute dans Hadoop

• Chaque Job est découpé en tâches **Map et/ou Reduce**.

• **JobTracker** est un processus maitre dont le rôle est de planifier l'exécution des tâches (Map ou Reduce) qu'il attributs aux processus esclaves appelés TaskTracker.

• Chaque **TaskTracker** doit envoyé des notifications régulières, appelées Heartbeat Call, pour informer le JobTracker de l'état d'avancement des tâches qui lui ont été confiées.

• Le **JobTracker** se charge à la fois: allouer des ressources (CPU, mémoire, … etc.) aux différentes tâches, coordonner les Jobs MapReduce, gérer les fautes en réallouant les slots éventuellement, … etc.

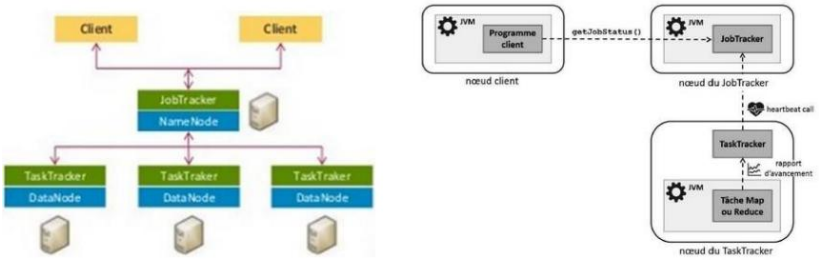


Figure 10 schema MapReduce1

2.3 Hadoop 2

2.3.1 Introduction

Hadoop 2.x a introduit plusieurs améliorations majeures par rapport à la première version. La modification la plus significative était l'ajout du framework **YARN** (Yet Another Resource Negotiator).

- **YARN** (qui signifie "Yet Another Resource Negotiator") a introduit une couche de gestion des ressources plus flexible dans l'écosystème Hadoop, permettant la prise en charge de différents types de frameworks de traitement, au-delà de MapReduce.

Cela a ouvert la porte à d'autres modèles de calcul, comme Apache Spark, Apache Tez, et d'autres, offrant ainsi une plus grande flexibilité dans le traitement des données et une meilleure utilisation des ressources.

2.3.2 Architecture de Hadoop 2

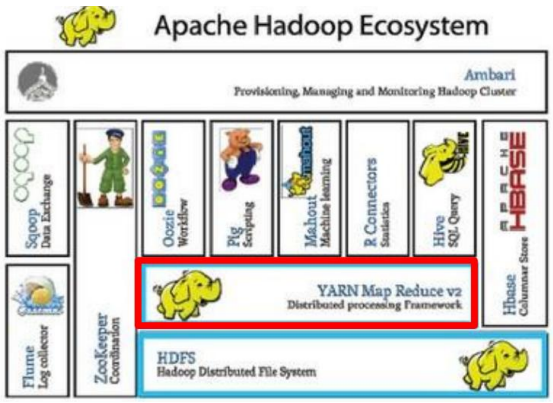


Figure 11 Architecture hadoop 2

YARN signifie Yet Another Resource Negotiator. C'est une plateforme générique de gestion des ressources dans un cluster. YARN a été introduit avec Hadoop 2.0, un framework de traitement distribué open source d'Apache.

YARN sépare le **JobTracker** (Hadoop 1.x) en un **ResourceManager** (RM) au niveau du cluster et un **ApplicationMaster** (AM) spécifique à l'application.

L'architecture de YARN suit un modèle architectural maître-esclave : **ResourceManager** en tant que maître et **NodeManager** spécifique à chaque nœud en tant qu'esclave. NodeManager remplace le **TaskTracker** dans Hadoop 1.x.

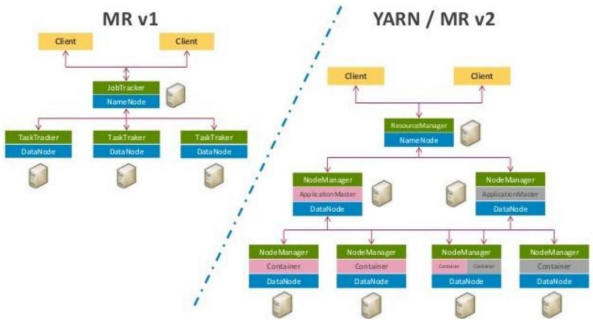


Figure 12 MapReduce1 Vs MapReduce2

2.3.3 Différence Hadoop 1 VS Hadoop 2

En tant que système d'exploitation de données pour Hadoop 2.0, YARN permet à plusieurs applications de coexister dans le même cluster partagé. YARN prend également en compte la scalabilité, une utilisation élevée, prend en charge plusieurs modèles de programmation, propose un modèle de ressources flexible, la sécurité, la fiabilité et la compatibilité.

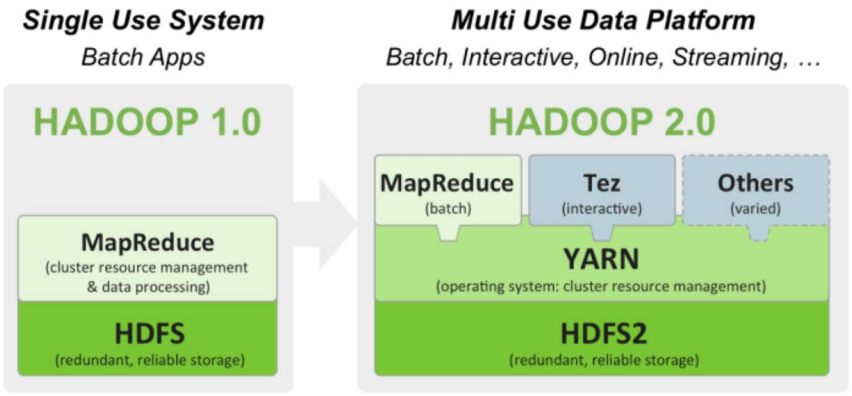
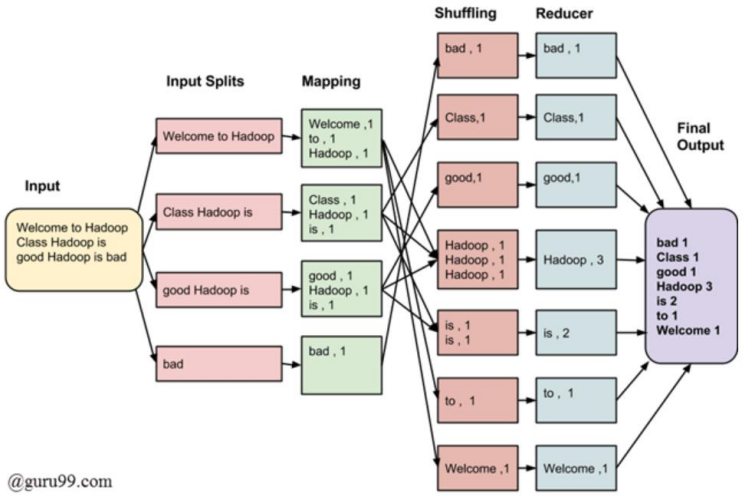


Figure 13 Hadoop1 & Hadoop2

3- Présentation de Hadoop MapReduce dans le cas de notre projet



Architecture MapReduce

Figure 14 Use case MapReduce

3.1 Introduction à Hadoop MapReduce

3.1.1 Contexte et Origines

Hadoop MapReduce est un modèle de programmation et un framework open-source développé par Apache pour le traitement distribué de données volumineuses. Il trouve ses racines dans les travaux de Google sur le traitement distribué de données, en particulier le modèle MapReduce.

3.1.2 Objectif de MapReduce

L'objectif principal de MapReduce est de fournir une approche scalable et parallèle pour traiter de grandes quantités de données sur des clusters de machines.

3.2 Fonctionnement de MapReduce

3.2.1 Phases de MapReduce

MapReduce fonctionne en deux phases principales : la phase de Map et la phase de Reduce.

- Phase de Map : Dans cette phase, les données sont divisées en blocs, puis traitées en parallèle par plusieurs mappers. Chaque mapper produit une sortie intermédiaire constituée de paires clé-valeur.
- Phase de Reduce : Les résultats intermédiaires sont triés et regroupés par clé, puis traités par plusieurs reducers qui effectuent une opération de réduction pour générer le résultat final.

3.2.2 Étapes du Processus

- Input Splitting : Les données sont divisées en blocs gérables appelés "splits".
- Mapping : Application de la fonction map à chaque split, générant des paires clé-valeur intermédiaires.
- Shuffling : Tri et regroupement des résultats intermédiaires par clé.
- Reducing : Application de la fonction reduce à chaque groupe de données associées à une clé, produisant le résultat final.

3.3 Écosystème Hadoop

3.3.1 HDFS (Hadoop Distributed File System)

HDFS est le système de fichiers distribué utilisé par Hadoop pour le stockage des données. Il divise les données en blocs et les distribue sur plusieurs nœuds du cluster.

3.3.2 YARN (Yet Another Resource Negotiator)

YARN est le gestionnaire de ressources qui gère et planifie les ressources du cluster, permettant l'exécution de diverses applications, y compris MapReduce, de manière concurrente.

3. 4. Avantages de Hadoop MapReduce

3.4.1 Scalabilité

MapReduce offre une scalabilité horizontale, permettant d'ajouter simplement des nœuds au cluster pour traiter davantage de données.

3.4.2 Tolérance aux Pannes

Le modèle MapReduce est conçu pour être tolérant aux pannes, avec la capacité de récupérer automatiquement à partir d'échecs.

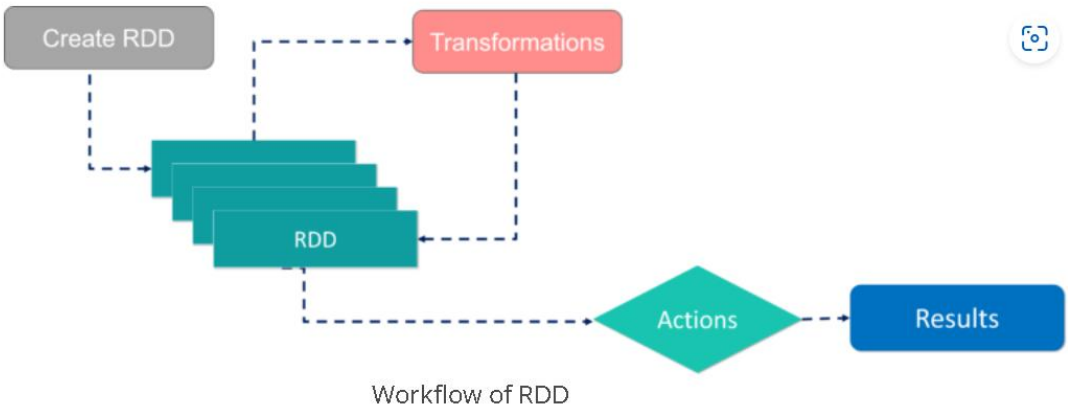
3.4.3 Facilité d'Utilisation

Hadoop MapReduce simplifie le développement en fournissant un modèle de programmation abstrait pour les développeurs.

Conclusion

En conclusion, Hadoop MapReduce est un framework puissant pour le traitement distribué de données massives, offrant une approche scalable et tolérante aux pannes. Bien que son modèle ait été révolutionnaire, l'évolution continue du domaine du Big Data a conduit au développement de frameworks plus modernes, tels que Apache Spark, pour relever de nouveaux défis.

4- Présentation de Spark et RDD



Workflow of RDD

4.1 Introduction à Apache Spark et RDD

4.1.1 Contexte et Origines

Apache Spark est un framework open-source de traitement de données en temps réel et par lots,

émergeant comme une évolution significative dans le domaine du Big Data. Conçu pour surmonter les limitations de MapReduce, Spark a rapidement gagné en popularité grâce à ses performances améliorées et à sa polyvalence.

4.1.2 Objectif de Spark

Spark vise à offrir une solution plus rapide et flexible pour le traitement distribué des données, en introduisant des améliorations majeures par rapport au modèle MapReduce. Il a été développé pour répondre aux besoins croissants de traitement en temps réel et de diverses applications d'analyse de données.

4.2 Concept de RDD (Resilient Distributed Datasets)

4.2.1 Définition de RDD

Les RDD, ou Resilient Distributed Datasets, constituent le cœur de Spark. Un RDD représente une collection immuable, partitionnée et distribuée d'objets sur un cluster. L'immuabilité garantit la résilience, et la distribution favorise le traitement parallèle sur plusieurs nœuds.

4.2.2 Caractéristiques de RDD

- Résilience : Les RDD sont résilients, ce qui signifie qu'ils peuvent être reconstitués en cas de défaillance d'un nœud du cluster.
- Distribution : Les données sont distribuées de manière transparente, permettant un traitement parallèle efficace.

4.2.3 Opérations sur RDD

- Transformation : Les transformations créent de nouveaux RDD à partir d'un existant. Par exemple, map, filter, reduceByKey.
- Action : Les actions déclenchent le calcul sur le RDD, renvoyant les résultats au programme. Exemples : count, collect, saveAsTextFile.

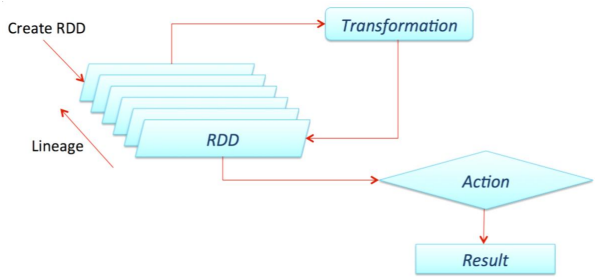


Figure 15 Opérations sur RDD

4.3 Avantages de Spark et RDD

4.3.1 Performances Améliorées

Spark offre des performances supérieures à MapReduce en minimisant les E/S disque et en favorisant le traitement en mémoire, ce qui conduit à des temps d'exécution plus courts.

4.3.2 Prise en Charge du Traitement en Temps Réel

Avec Spark Streaming, Spark excelle dans le traitement en temps réel, permettant des analyses continues sur les flux de données.

4.3.3 Bibliothèques Étendues

Spark propose un écosystème riche de bibliothèques, telles que Spark SQL pour le traitement SQL, MLlib pour l'apprentissage automatique, et GraphX pour le traitement des graphes.

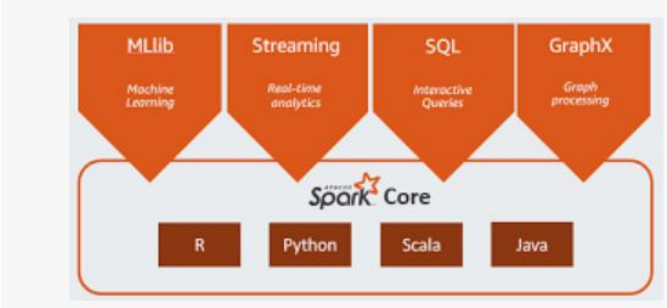


Figure 16 Bibliothèque Etendues

Conclusion

En conclusion, Apache Spark, avec son modèle RDD, représente une avancée significative dans le traitement distribué des données. Ses caractéristiques de résilience, de distribution et de performances en font un choix privilégié dans le domaine du Big Data, y compris notre projet axé sur l'analyse des films regardés.

Chapitre 3: Cas pratique de Projet

1- Les données utilisées

Pour les données, nous utilisons les données fournies dans la Classroom. Cependant, nous considérons que le fichier qui est fréquemment regardé par des utilisateurs 3 fois au lieu de 1000, en réponse à la question 1 de Spark. De plus, nous supprimons une ligne dans le fichier 'watchedMovies' pour répondre à la question 2 de la partie 2. Et voilà le contenu de tous les fichiers de projet:

Fichier User.txt

```
PaoloG76,Male,1976,Italy
User1,Male,1976,Italy
User2,Female,1986,France
```

Fichier Movies.txt

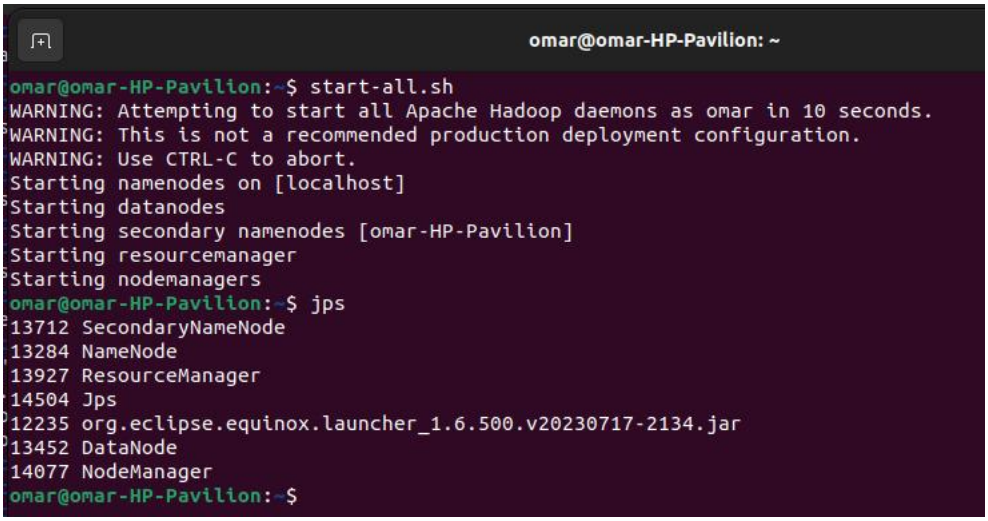
```
MID124,Ghostbusters,Ivan Reitman,1984/05/01
MID12,Title12,Ivan Reitman,1984/05/01
MID10,Title10,Ivan Reitman,1984/05/01
```

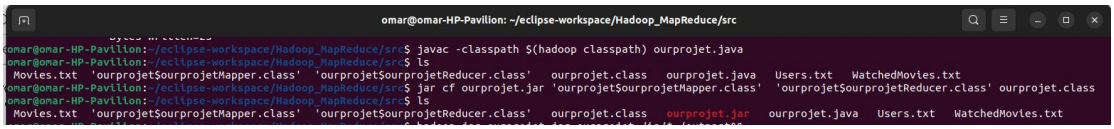
Fichier WatcheMovies.txt

```
PaoloG76,MID124,2018/06/01_14:18,2018/06/01_16:10
PaoloG76,MID124,2018/07/01_14:18,2018/07/01_16:10
User1,MID124,2018/07/01_14:18,2018/07/01_16:10
User2,MID124,2018/08/01_14:18,2018/08/01_16:10
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
User2,MID12,2009/08/01_14:18,2009/08/01_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,MID124,2001/08/01_14:18,2018/08/01_16:10
User2,MID124,2019/08/01_14:18,2019/08/01_16:10
User2,MID124,2019/08/01_14:18,2019/08/01_16:10
User1,MID12,2019/08/01_14:18,2019/08/01_16:10
User2,MID12,2019/08/01_14:18,2019/08/01_16:10
```

Figure 17 Fichiers de travail

2- Code source Hadoop MapReduce





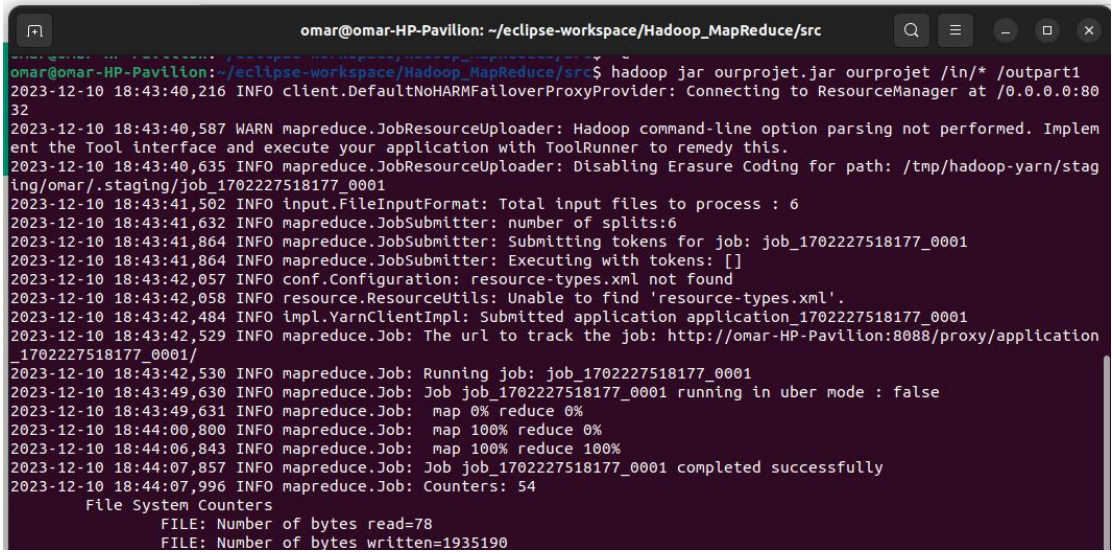


Figure 18 Code source Hadoop MapReduce

=> Mapper class

```java
// Mapper Class
public static class OurProjetMapper extends Mapper<Object, Text, Text, Text> {
    private Text movieId = new Text();
    private Text username = new Text();

    // Map method processes each input record
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        // Split the CSV record into tokens
        String[] tokens = value.toString().split(",");

        // Assuming the format is "Username, MID, StartTimestamp, EndTimestamp"
        if (tokens.length == 4) {
            String startTimestamp = tokens[2].trim();
            // Check if the start timestamp is in the year 2019
            if (isInYear2019(startTimestamp)) {
                movieId.set(tokens[1].trim());
                username.set(tokens[0].trim());
                // Emit movieId as key and username as value
                context.write(movieId, username);
            }
        }
    }
```

Figure 19 Mapper Class

=> Reduce class

```java
// Reducer Class
public static class OurProjetReducer extends Reducer<Text, Text, Text, Text> {
    private Text result = new Text();

    // Reduce method processes the grouped key-value pairs
    public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {
        Set<String> uniqueUsers = new HashSet<>();

        // Iterate through values and add unique usernames to the set
        for (Text value : values) {
            uniqueUsers.add(value.toString());
        }

        // If there is only one unique user, emit the key-value pair
        if (uniqueUsers.size() == 1) {
            result.set(uniqueUsers.iterator().next());
            context.write(key, result);
        }
    }
}
```

Figure 20 Reduce class

Résultat:

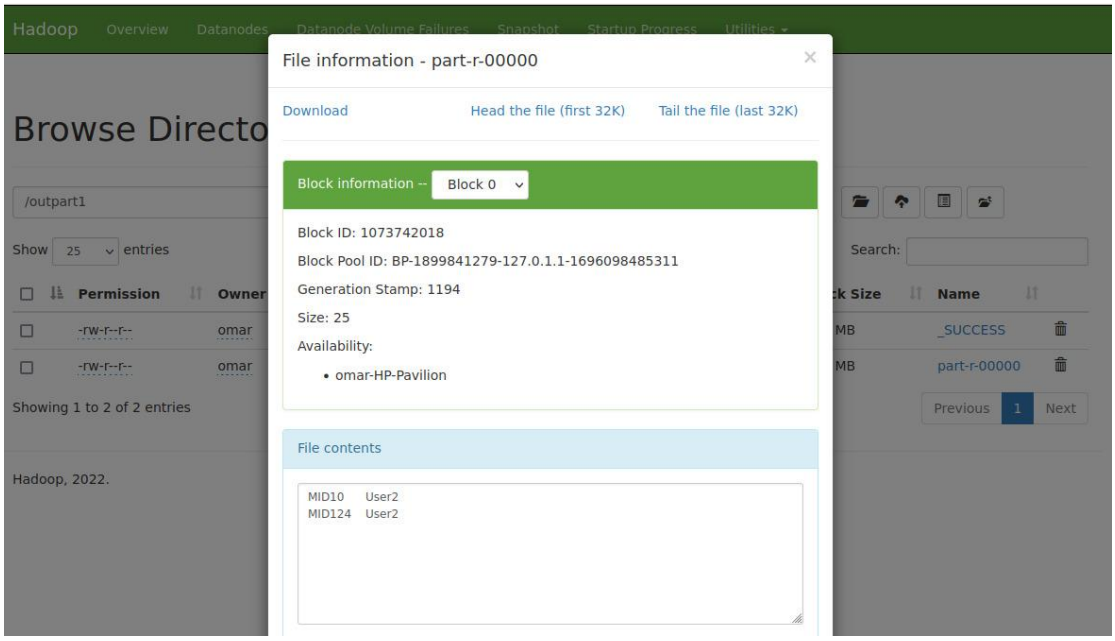


Figure 21 Resultat partie 1

3- Code source Spark et RDD

Question 1:

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

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

# Initialisation du contexte Spark
sc = SparkContext(appName="Films_Frequents_Cinq_Dernieres_Annees")

# Chargement du fichier WatchedMovies.txt en tant que RDD
watched_movies_rdd = sc.textFile("WatchedMovies.txt")

# Filtrage des lignes liées aux cinq dernières années
filtered_movies_rdd = watched_movies_rdd.filter(lambda line: "2015/09/17" <= line.split(",")[2] <= "2020/09/16")

# Transformation en paire clé-valeur avec (MID, (Année, 1)) pour chaque ligne
movies_year_count_rdd = filtered_movies_rdd.map(lambda line: ((line.split(",")[1], line.split(",")[2][:4]), 1))

# Réduction par clé pour compter le nombre de fois que chaque film a été regardé chaque année
movies_year_count_sum_rdd = movies_year_count_rdd.reduceByKey(lambda x, y: x + y)

# Filtrage des films qui ont été regardés au moins 1000 fois au cours d'une année
selected_movies_rdd = movies_year_count_sum_rdd.filter(lambda x: x[1] >= 3)

# Transformation pour obtenir le format (MID, Année)
result_rdd = selected_movies_rdd.map(lambda x: (x[0][0], x[0][1]))

# Affichage des résultats
result_rdd.foreach(print)

# Enregistrement des résultats dans un fichier de sortie (vous pouvez ajuster le chemin selon vos besoins)
result_rdd.saveAsTextFile("out_part2_Q2")

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

Figure 22 Code question1 spark

Figure 23 Output question1 spark

Question2:

Film le plus populaire depuis au moins deux ans.

```python
from pyspark import SparkContext, SparkConf

# Create a Spark configuration and set the application name
conf = SparkConf().setAppName("MostPopularMoviesByYear")
sc = SparkContext(conf=conf)

# Load the WatchedMovies data from the input file
watched_movies = sc.textFile("WatchedMovies.txt")

# Split each line into fields
# Assuming the format is "Username, MID, StartTimestamp, EndTimestamp"
# Adjust the split logic based on your actual data format
watched_movies_data = watched_movies.map(lambda line: line.split(","))

# Map the data to (MID, (Year, User)) pairs
movie_year_user_pairs = watched_movies_data.map(lambda fields: ((fields[1], fields[2].split("/")[0]), fields[0]))

# Remove duplicate users in a specific year for each movie
distinct_users_per_year = movie_year_user_pairs.distinct()

# Map to (MID, 1) to count occurrences of each movie in each year
movie_count = distinct_users_per_year.map(lambda x: ((x[0][0], x[0][1]), 1))
```

```python
# Map to (MID, 1) to count occurrences of each movie in each year
movie_count = distinct_users_per_year.map(lambda x: ((x[0][0], x[0][1]), 1))

# Reduce by key to get the count of distinct users for each movie in each year
movie_user_count = movie_count.reduceByKey(lambda x, y: x + y)

# Map to (Year, (MID, Count)) to find the most popular movie for each year
most_popular_movies = movie_user_count.map(lambda x: (x[0][1], (x[0][0], x[1])))

# Reduce by key to get the final result, selecting the most popular movie for each year
result = most_popular_movies.reduceByKey(lambda x, y: x if x[1] > y[1] else y).map(lambda x: (x[0], x[1][0]))

# Collect the result to the driver program
collected_result = result.collect()

# Count the occurrences of each MID
mid_counts = {}
for _, mid in collected_result:
    mid_counts[mid] = mid_counts.get(mid, 0) + 1

# Filter MIDs that are present in at least 2 years
filtered_mids = [mid for mid, count in mid_counts.items() if count >= 2]

# Filter the collected result based on the selected MIDs
final_result = [(year, mid) for year, mid in collected_result if mid in filtered_mids]

# Save the result to the output directory
sc.parallelize(final_result).map(lambda x: f"{x[1]}").saveAsTextFile("OutPart2_Q2")

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

Figure 24 Code question2 spark

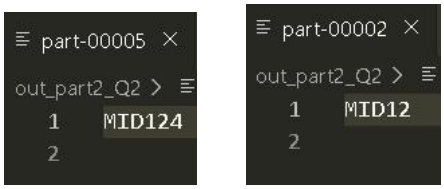


Figure 25 Output question2 spark

FIN