M1Info

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

Université Paris 8 - Vincennes à Saint-Denis

M1 MIASHS : Big Data et fouille de données

Création d'une architecture distribuée et analyse de données

PANCHALINGAMOORTHY Gajenthran

Organisme d'accueil :	Université Paris 8
Cours :	Cadre logiciel pour le Big Data

Table des matières

1 Introduction **1**

2 Datasets **2**

3 AWS **3**
3.1 Service S3 4
3.2 Service EMR 5

4 Analyse des données (Hive) **9**
4.1 Initialisation des tables 9
4.2 Hive 10
4.3 Hive 12
4.3.1 Simple système de recommendation 12
4.3.2 Système de recommendation basé sur une note mesurée 12
4.3.3 Système de recommendation basé sur la catégorie . . . 13

5 Comparaison de Hive et Python **15**

6 Conclusion **18**

Chapitre 1

Introduction

Avec l'emergence des entreprises de streaming, les utilisateurs consomment de plus de plus de films, de séries ou encore d'oeuvres cinématographiques. La satisfaction des utilisateurs représente donc un enjeu important pour les entreprises telles que Netflix, Amazon Prime ou encore Disney+. Cela passe donc par un système de recommandation de films afin d'avoir un gain de temps conséquent et d'améliorer l'expérience utilisateur.
A l'aide de l'architecture distribuée `Amazon Web Services` (AWS), nous nous pencherons sur ce sujet et analyserons les données à l'aide d'`Hive`. Apache Hive est une infrastructure d'entrepôt de données intégrée sur Hadoop permettant l'analyse, le requétage via un langage proche syntaxiquement de SQL ainsi que la synthèse de données. Bien que initialement développée par Facebook, Apache Hive est maintenant utilisée et développée par d'autres sociétés comme Netflix. Amazon maintient un fork d'Apache Hive qui inclut Amazon Elastic MapReduce dans Amazon Web Services (source : Wikipédia).
Dans ce papier, nous allons d'abord décrire les datasets utilisées pour la phase d'analyse, puis nous créerons l'architecture distribuée AWS, puis nous détaillerons les différents requêtes `Hive`, ensuite nous comparerons l'efficacité de ce dernier avec un autre langage et enfin nous discuterons à propos des avantages proposés par l'architecture distribuée.

Chapitre 2

Datasets

Le système de recommandation se reposera quasiment que sur les films. Afin d'obtenir des résultats fiables et de profiter pleinement de l'architecture distribuée, une base de données d'environ 1Go a été récupérée, en format *.csv*. Elle a été repris de *Kaggle* qui elle-même vient des films listés par *MovieLens*, et contient des films venant de 2018 ou avant. Celle-ci est découpée en plusieurs parties :

- *movies.csv* : regroupe l'identifiant des films (`movieId`), l'identifiant des films sur le site IMDB (`imdb_id`) et le titre des films (`title`).
- *ratings.csv* : regroupe l'identifiant des utilisateurs (`userId`), l'identifiant des films (`movieId`), et la note des utilisateurs (`rating`) sur 5 et la date à laquelle ils ont noté (`timestamp`).
- *votes.csv* : regroupe l'identifiant des utilisateurs (`userId`), l'identifiant des films (`movieId`), l'identifiant des films sur le site IMDB (`imdb_id`) et le vote moyen des films (`vote_average`) et le nombre de votant pour chaque film (`vote_count`).

Ainsi, nous nous appuyons sur une base de données de plus de 45K films et notamment 26 millions de votes dont plus de 250K votants.

Chapitre 3

Création d'une architecture distribuée (AWS)

Suite à la limite de crédit dépassé sur le compte, le projet a donc du être reporté sur un autre compte mais cela n'aura une influence ni sur le fonctionnement de l'architecture, ni sur les explications apportées ci-dessous.
Afin de bénéficier des services proposés par AWS, il fallait tout d'abord se connecter à celui-ci. Pour cela, nous sommes passé par l'interface `RosettaHUB` (fig.3.1) afin de créer de notre compte (déjà réalisé lors des séances "Cadre logiciel pour le Big Data"). Une fois le compte créé, nous devons alimenter notre compte en crédit pour pouvoir utiliser les services AWS. Les détails seront épargnés vu qu'il s'agit ici de manoeuvres déjà exécutées en cours.
Dès que les prépartifis sont terminés, nous pouvons nous attaquer à l'architecture. Tout d'abord, nous devons créer une instance en se dirigeant vers le service `EC2`. Un tutoriel a été mis en ligne sur la plateforme Moodle afin d'expliquer en détails les différentes étapes à suivre pour lancer une instance.
Avant de nous occuper des clusters, nous allons dans un premier temps, récolter les données depuis notre ordinateur et les placer dans le service S3.

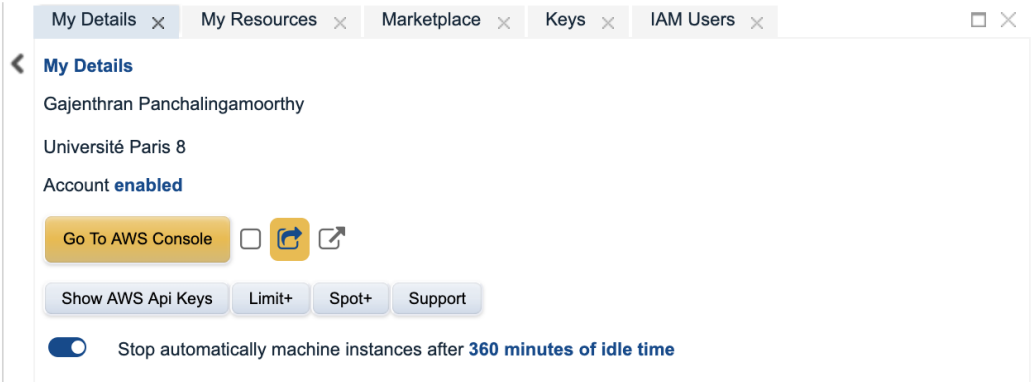

FIGURE 3.1 – Interface `RosettaHUB`

3.1 Service S3

Le service S3 permet de stocker les données en ligne dans le cloud afin qu'on puisse les réutiliser avec d'autres services d'AWS dans la section suivante. Il offre la possibilité de stocker n'importe quelle quantité de données et également d'assurer leurs protections avec une sécurité très fiable. Pour profiter de ce service, dirigeons nous vers la console AWS, et cherchons le service `S3`.
Le service `S3` range les données par compartiments (fig.3.2). Pour ajouter de nouvelles données, il suffit de créer un compartiment en cliquant sur le bouton *Créer un compartiment*. Cela ouvrira une nouvelle fenêtre dans laquelle nous préciserons le nom du compartiment pour pouvoir l'identifier et le récupérer lors de notre connexion (fig.3.3). Une fois le compartiment créé, nous pouvons importer des fichiers à travers le bouton *Charger* et de charger les fichiers que nous souhaitons. Les fichiers sont plutôt lourds donc le temps des transactions risque de durer assez longtemps. A noter que ne nous comptons pas utiliser toutes les données de la base de données, donc une répartition des données de façon judicieuse s'impose [1]. De plus, afin d'avoir une base de données plus organisées, nous pouvons également les ranger dans des dossier avec *Créer un dossier*.

1. Toutes les colonnes des fichiers *.csv* ne vont pas être exploitées, ainsi la décision de supprimer les colonnes inutiles a été prise afin d'alléger la taille de la base.

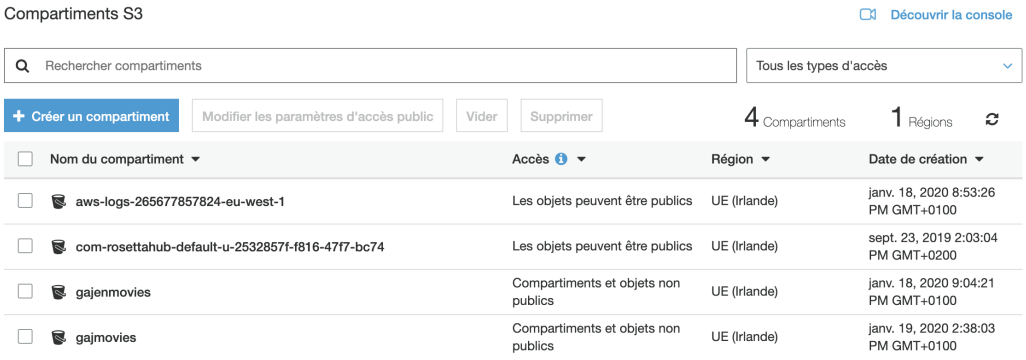

FIGURE 3.2 – Amazon S3 et son rangement par compartiment

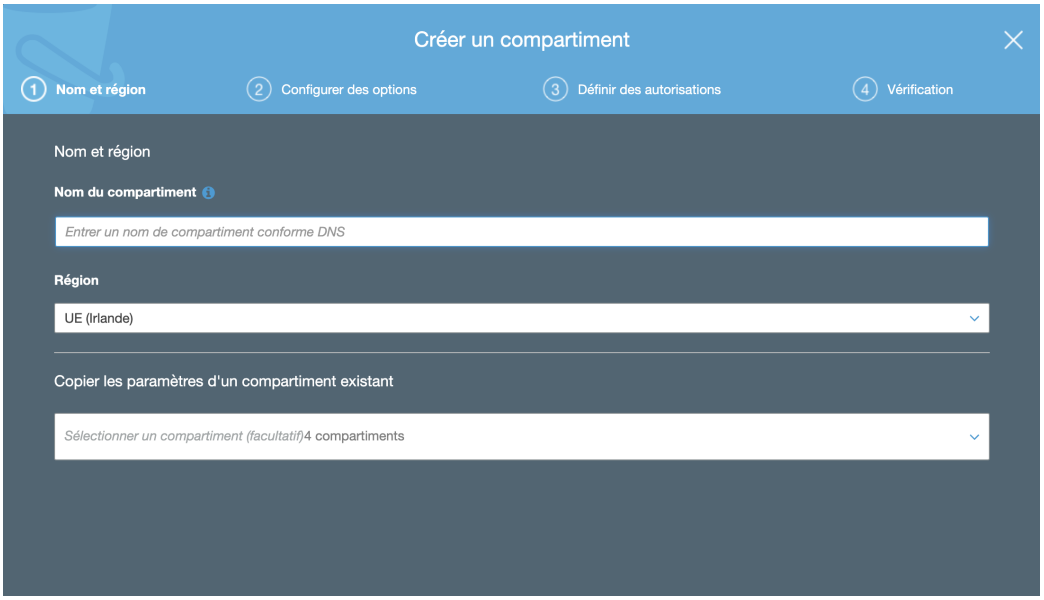

FIGURE 3.3 – Création d'un nouveau compartiment dans Amazon S3

3.2 Service EMR

Désormais, nous pouvons lancer les clusters. Pour cela, nous allons nous rediriger vers la console AWS et chercher le service `EMR`. Le service `EMR` (fig.3.4) est la plateforme de mégadonnées native cloud leader qui traite de grandes quantités de données rapidement et à moindre coût. Utilisant des outils open source tels que `Apache Spark`, `Apache Hive`, `Apache HBase`, `Apache Flink`, `Apache Hudi` (Incubating) et `Presto`, associés à la scalabi-

lité dynamique d'Amazon `EC2` et au stockage évolutif d'Amazon `S3` (source : AWS.Amazon.com).

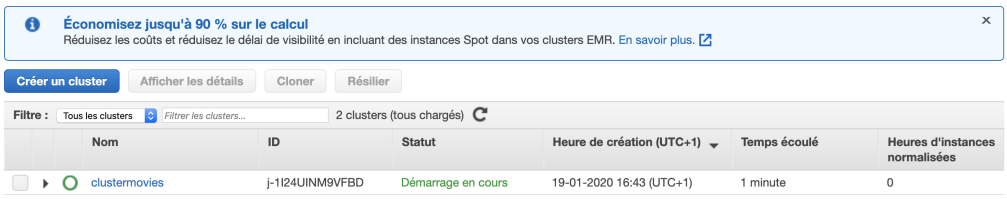

FIGURE 3.4 – Amazon EMR et la liste des clusters

En cliquant sur *Créer un cluster*, nous allons pouvoir créer un nouveau cluster (fig.3.5). Il y a plusieurs paramètres à remplir comme le nom du cluster, les logiciels pouvant être utilisés (en l'occurence, pour ce projet, Coore Hadoop sera utilisé).

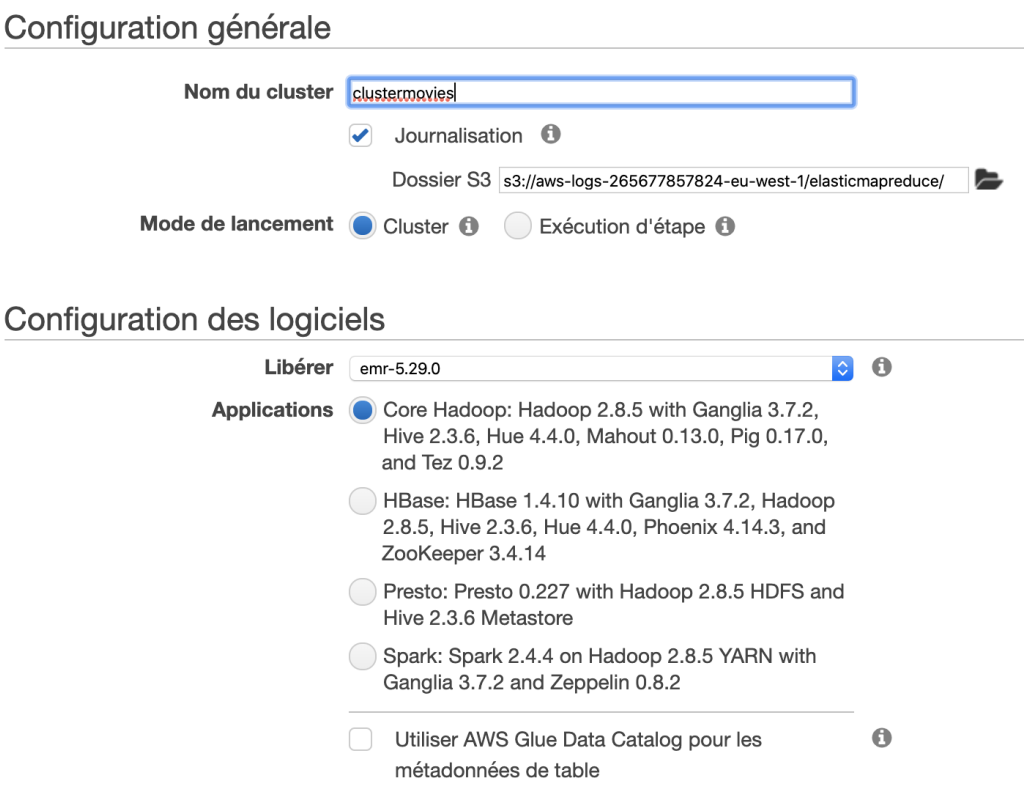

FIGURE 3.5 – Création de clusters dans Amazon `EMR`

De plus, nous préciserons le nom de la clé que nous avons instancié, avec

comme nombre équivalent d'instance à 3 (à savoir 1 noeud maître et 2 noeuds principaux) avec le type d'instance *m3.xlarge* afin d'éviter la surconsommation (fig.3.6)

FIGURE 3.6 – Choix des instances dans Amazon EMR

Puis, pour établir la connexion ssh et profiter des fonctionnalités proposées par EMR, attendez que votre soit totalement prêt (symbolisé par un rond rempli en vert), puis dirigez vous sur le cluster en question afin de trouver la commande utile à la connexion (fig.3.7).

```
MacBook-Pro-de-Gajenthran:~ gajen$ sudo ssh -i ~/gajaws.pem hadoop@ec2-34-253-8-79.eu-west-1.compute.amazonaws.com
Password:
The authenticity of host 'ec2-34-253-8-79.eu-west-1.compute.amazonaws.com (34.253.8.79)' can't be established.
ECDSA key fingerprint is SHA256:lsJXy78aj9Z6mrD88auEjZ6G6XuNd1s3rk0iKDSWPZw.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-34-253-8-79.eu-west-1.compute.amazonaws.com,34.253.8.79' (ECDSA) to the list of known
Last login: Sun Jan 19 16:06:51 2020

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
3 package(s) needed for security, out of 17 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR
```

FIGURE 3.7 – Etablissement de la connexion ssh avec EMR

La connexion étant établie, il nous reste à transférer les fichiers stockés dans `S3` (fig.3.8, dans notre `EMR` à l'aide de l'argument `-copyToLocal` en indiquant le nom de notre compartiement (fig.3.9. Une fois les fichiers *.csv* importés localement, il faudra envoyer ces fichiers dans `/user/hadoop/` via l'argument `-fromLocal` (fig.3.10).

```
[hadoop@ip-172-31-44-42 ~]$ hadoop dfs -ls s3://gajmovies/
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

Found 2 items
-rw-rw-rw-   1 hadoop hadoop    1500061 2020-01-19 13:39 s3://gajmovies/movies.csv
-rw-rw-rw-   1 hadoop hadoop    1041857 2020-01-19 13:39 s3://gajmovies/votes.csv
```

FIGURE 3.8 – Affichage des compartiments stockés sur `S3` avec `EMR`

```
[hadoop@ip-172-31-44-42 ~]$ hadoop dfs -copyToLocal s3://gajmovies/*
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

20/01/19 16:20:42 INFO s3n.S3NativeFileSystem: Opening 's3://gajmovies/movies.csv' for reading
20/01/19 16:20:42 INFO s3n.S3NativeFileSystem: Opening 's3://gajmovies/votes.csv' for reading
[hadoop@ip-172-31-44-42 ~]$ ls
movies.csv  votes.csv
```

FIGURE 3.9 – Copie des fichiers contenus dans `S3` avec `EMR`

```
[hadoop@ip-172-31-44-42 ~]$ hadoop dfs -copyFromLocal *.csv /user/hadoop/
```

FIGURE 3.10 – Transfert des fichiers locaux dans `/user/hadoop/` avec `EMR`

Désormais, il nous reste plus qu'à exploiter `Hive` afin d'analyser les données que nous venons de récupérer. Pour tester si le logiciel fonctionne correctement, il suffit de taper la commande `hive`, ou `hive -f <fichier.hive>`.

Chapitre 4

Analyse des données (Hive)

Pour l'analyse de données, comme nous l'avons dit, nous nous focaliserons sur le langage `Hive`. Le système de recommandation d'un film peut prendre plusieurs formes, il peut être simple comme assez complexe. Il peut se baser sur les notes, sur le contenu d'un film, sur les goûts de l'utilisateur...

4.1 Initialisation des tables

Avant toute chose, commençons par créer les tables à partir de la base de données. Les créations des tables se trouveront dans *init_ rs.hive*. À l'aide de `CREATE EXTERNAL TABLE IF NOT EXISTS` (fig.4.1) pour créer une table, nous préciserons le même nombre de colonne que les fichiers *.csv*. D'ailleurs, nous préciserons à la fin de l'instruction que les tuples des fichiers *.csv* sont délimités par des virgules comme suit `ROW FORMAT DELIMITED FIELDS TERMINATED BY ','`;. Les types que nous utiliserons principalement concerneront les entiers `INT`, les chaînes de caractères `STRING`, les flottants `FLOAT`. Nous tenons également à mentionner l'ajout de la partition sur l'année de sortie des films qui va avoir un rôle déterminant vu qu'il s'agit de films repris des années 2017 principalement. Pour récupérer les fichiers *.csv* de `/user/hadoop/`, `LOAD DATA INPATH <path> INTO TABLE <table>;` se chargera de mettre en place les table pour pouvoir exploiter la base de données.

```
-- Table rs_movies
CREATE EXTERNAL TABLE IF NOT EXISTS rs_movies
(movieId INT,
imdbId INT,
YEAR INT,
title string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

FIGURE 4.1 – Création des tables sur `Hive`

4.2 Analyse simple des films

Avant d'implémenter des systèmes de recommandations, initions nous à l'analyse simple des données. Les requêtes permettront de vérifier et de s'acclimater avec la base de données.
Par exemple, pour commencer, il serait interessant si nos films possèdent tous des identifiants différents pour pouvoir les exploiter, en mettant en évidence `DISTINCT` pour récupérer un ensemble sans doublons. Puis, cet ensemble sera comparé avec l'ensemble des identifiants pour vérifier si ils sont équivalents, ce qui voudra dire que notre base de données ne comporte pas de doublons (fig.4.2).

```
SELECT COUNT(DISTINCT(movieId))
FROM movies;
SELECT COUNT(movieId)
FROM movies;
SELECT COUNT(imdb_id)
FROM movies;
SELECT COUNT(DISTINCT(imdb_id))
FROM movies;
```

FIGURE 4.2 – Commande `Hive` pour vérifier si les identifiants des films sont différents

Récupérer le ou les genres d'un film est une tâche assez compliquée avec le modèle de données que nous avons : les genres d'un film sont représentés sous la forme d'un objet JSON. La condition `LIKE` va nous servir à combler ce

problème en indiquant le nom du ou des genres d'un film de la manière : `WHERE genres LIKE '%Animation%'` pour savoir si il s'agit d'un film d'animation (fig.4.3).

```
SELECT *
FROM movies
  WHERE genres
    LIKE '%Animation%'
      AND vote_average > '7.5';
```

FIGURE 4.3 – Commande `Hive` pour récupérer les films d'animations ayant une note $> 7.5/10$

Ensuite, si nous voulons calculer le nombre de films selon le budget des films, il faudra s'attarder sur la commande `GROUP BY` pour regrouper les fims selon le budget. Parmi les différents groupes de budgets, seul les groupes de plus de 5 films seront pris en considération (fig.4.4).

```
SELECT COUNT(movieId), movieId
FROM ratings
  GROUP BY movieId
    ORDER BY COUNT(movieId) DESC;
```

FIGURE 4.4 – Commande `Hive` pour compter le nombre de votes pour chaque films dans *ratings*

Par ailleurs, pour récupérer le nombre de notes de chaque utilisateurs, nous allons une fois de plus bénéficier de la commande `GROUP BY` afin de regrouper les notes par utilisateurs. Seul les utilisateurs ayant votés pour plus de 5 films avec la commande `HAVING <condition>`, et dont la moyenne est supérieure à 4 avec `AVG` (fig.4.5).

```
SELECT userid, AVG(rating)
FROM ratings
  GROUP BY userid
    HAVING COUNT(movieId) > 5
      AND AVG(rating) > 4;
```

FIGURE 4.5 – Commande `Hive` pour récupérer le nombre de notes de chaque utilisateur

4.3 Système de recommandation

Plusieurs systèmes de recommandation seront étudiés afin de tester l'efficacité que cela soit en terme de temps ou encore de la pertinence des résultats obtenus.

4.3.1 Simple système de recommandation

Un simple système de recommandation qui se base uniquement sur les votes moyens des utilisateurs d'un film. Nous nous contentons de classer par ordre décroissant les votes des films et établir une liste des films les plus populaires de cette manière (fig.4.6).
Pour cela, nous devons réaliser une jointure entre les tables `rs_movies` et `vote` afin de mélanger les colonnes `titre` et `vote_average`. La jointure se reposera sur la clé `movieId` (des deux côtés). Pour ranger les enregistrements par ordre décroissant, `ORDER BY ... DESC` nous sera utile. Pour finir, on affichera seulement les 5 films les plus populaire avec `LIMIT 5`.

```
SELECT m.title, v.vote_average
FROM rs_movies m
    JOIN vote v ON (m.movieId = v.movieId)
ORDER BY vote_average DESC LIMIT 5;
```

FIGURE 4.6 – Requête pour un simple système de recommandation

4.3.2 Système de recommandation basé sur une note mesurée

Pour obtenir un système plus fiable, nous allons effectuer quelques ajustements au premier système. Tout d'abord, pour obtenir des résultats optimaux, les films n'ayant pas dépassé un certain seuil de vote τ ne pourront pas faire partis de cette liste. Ensuite, nous ajouterons à la table `vote`, une nouvelle colonne `wr` qui représentera une note mesurée qui se réfère à la formule suivante : $(\frac{vc}{vc+\tau} * avg) + (\frac{\tau}{\tau+vc} * \mu)$ où vc correspond à nombre de vote d'un film, avg à la note moyenne d'un film, τ au seuil minimal pour accepter un film et μ au vote moyen des films. Une fois toutes les lignes de la colonne `wr` remplies, nous choisirons les 5 premiers films. Pour cela, `ALTER TABLE ... ADD COLUMNS` ajoutera une nouvelle colonne `wr` sur la table `vote`. Nous calculerons μ et τ et les stockeront respectivement dans les variables `avg_votes` et

vote_counts en utilisant la fonction percentile. En ajoutant la colonne, il nous reste simplement à remplir chaque case avec UPDATE ... SET de la colonne grâce la formule vue précédemment. Les variables initialisées au départ seront exploitées à l'aide de la notation suivante : $hiveconf:<variable>. Désormais, il nous suffit de retirer de la table les enregistrements ayant les plus grandes valeurs wr.

```
ALTER TABLE vote ADD columns (wr FLOAT);

-- Calcul de la moyenne des notes moyennes des films
SET avg_votes = percentile(vote.vote_average, 0.5);
SET vote_counts = percentile(vote.vote_count, 0.90);

-- Mis à jour de la nouvelle colonne wr
UPDATE vote
SET wr=(vote.vote_count /
        (vote_count + '${hiveconf:vote_counts}') *
        vote.vote_average) +
        (vote_counts / (vote_counts + vote.vote_count) * '${hiveconf:avg_votes}')

SELECT m.title, v.wr
FROM rs_movies m
    JOIN vote v ON (m.movieId = v.movieId)
ORDER BY wr DESC LIMIT 5;
```

FIGURE 4.7 – Requête pour un système de recommandation plus sophistiqué

4.3.3 Système de recommandation basé sur la catégorie

Par ailleurs, il est également possible de renforcer le système actuel par un filtrage concernant le genre des films (fig.4.8). En effet, il est légèrement compliqué de parser un objet JSON via Hive mais il est possible de contourner le problème d'une manière différente. Cela nécessite, dans un premier lieu, la jointure de 3 tables vote, rs_movies et genres en utilisant des JOIN. Dès que les liaisons faites, il faudra filtrer les tuples en se basant sur la colonne genres. La condition g.genres LIKE '%Romance%' fera l'affaire pour conserver seulement les films romantiques dans notre table. Nous essayerons pas de parser les objets afin de distinguer les différents genres d'un film, nous nous contenterons de vérifier si l'objet (considéré comme une chaîne de caractère) contient le mot demandé (en l'occurence "Romance" dans notre cas).

```
SELECT m.title, v.vote_average FROM rs_movies m
    JOIN vote v ON (m.movieId = v.movieId)
        JOIN genres g ON (v.movieId = g.movieId)
WHERE g.genres LIKE '%Romance%' AND v.vote_count > 500.0
ORDER BY v.vote_average DESC;
```

FIGURE 4.8 – Requête pour un système de recommandation ajoutant les genres

Chapitre 5

Comparaison de Hive et Python

Pour comparer l'importance des clusters et d'`Hive`, nous allons comparer les résultats obtenus lors de la réalisation de `Python`. Le fichier *MR_Popular.py* contient plus ou moins la même fonctionnalité proposées par le système plus sophistiqué d'`Hive` avec une note mesurée (fig.5.1).
Si nous comparons en termes de code les deux systèmes, il y a très peu de différence que l'on peut remarquer (fig.5.2)). La différence se manifeste surtout au niveau de l'exécution des tâches et du temps consacré. `Hive` semble avoir une longueur d'avance notamment grâce à `EMR` qui le rend plus efficace.

```
def rating(self, row):
    """
      Calcul de la note mesurée attribuée à un film.
    """
    vc = row['vote_count'];
    va = row['vote_average'];
    return (vc / (vc + self.vote_counts) * va) + \
           (self.vote_counts / (self.vote_counts + vc) *
           self.avg_votes);
```

FIGURE 5.1 – Méthode retournant la note mesurée d'un film

P
```
set_avg_votes();
set_vote_counts();
```
H
```
SET avg_votes = percentile(vote.vote_average, 0.5);
SET vote_counts = percentile(vote.vote_count, 0.90);
```

P
```
UPDATE vote
SET wr=(vote.vote_count /
        (vote_count + '${hiveconf:vote_counts}') *
        vote.vote_average) +
        (vote_counts / (vote_counts + vote.vote_count) * '${hiveconf:avg_votes}')
```
H
```
popular['rating'] = popular.apply(rating, axis=1);
```

P
```
SELECT m.title, v.wr
FROM rs_movies m
    JOIN vote v ON (m.movieId = v.movieId)
ORDER BY wr DESC LIMIT 5;
```
H
```
popular = popular.sort_values('rating', ascending=False);
popular = popular.head(n);
```

FIGURE 5.2 – Comparaison entre le programme `Python` et `Hive`

Cependant, il est possible de rencontrer quelques difficultés lorsque nous souhaitons réaliser un système plus complexe avec `Hive` comme un système basé sur le contenu du film. C'est pourquoi, nous avons décidé de réaliser cela en `Python`. Le but est de matcher les films ayant le plus de ressemblance avec le film passé en paramètre. Le taux de ressemblance sera calculé à l'aide des descriptions, des tags, du titre des films, en utilisant la similarité cosinus $cos\theta = \frac{A*B}{||A|| *||B||}$ (fig.5.4). Puis il suffit de prendre les films pour lesquelles la valeur de la similarité cosinus est la plus élevé par rapport au film choisi pour avoir une bonne recommandation. A noter que nous utiliserons plutôt `linear_kernel` lorsque nous utilisons l'objet `TfidfVectorizer` afin de gagner du temps.

```
if(tfid):
  tfid = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
  tfidm = tfid.fit_transform(self.data['content_based']);
  cs = linear_kernel(tfidm, tfidm)
else:
  cv = CountVectorizer();
  cm = cv.fit_transform(self.data["content_based"]);
  cs = cosine_similarity(cm);
```

FIGURE 5.3 – L'utilisation de la similarité de cosinus pour calculer la similarité entre les films

Par ailleurs, voici un autre modèle se basant sur la méthode des k plus proches voisins (KNN). KNN est une méthode d'apprentissage supervisé. Pour effectuer une prédiction, l'algorithme KNN va se baser sur le jeu de données en entier. En effet, pour une observation, qui ne fait pas parti du jeu de données, qu'on souhaite prédire, l'algorithme va chercher les K instances du jeu de données les plus proches de notre observation. Ensuite pour ces K voisins, l'algorithme se basera sur leurs variables de sortie y pour calculer la valeur de la variable y de l'observation qu'on souhaite prédire (source : mrmint.fr).
Pour la réalisation de KNN, il faudra tout d'abord joindre les bases de données *movies* et *ratings* avec `pd.merge`. Puis, nous prendrons en compte seulement les films ayant un certain nombre de votes (à savoir 20 ou plus dans notre cas). Ensuite, nous utiliserons la classe `NearestNeighbors` afin de calculer k plus proches voisins, notamment avec la méthode `kneighbors`. A noter que nous prendrons les 5 plus proches voisins pour la recommandations des films.

```
data_csr_matrix = csr_matrix(self.data)
knn = NearestNeighbors(metric='cosine',
algorithm='brute')
knn.fit(data_csr_matrix)
dist, idx = knn.kneighbors(self.data.loc['Batman
Returns'].values.reshape(1, -1), n_neighbors=self.K)
```

FIGURE 5.4 – Méthode KNN pour la recommandation des films

Chapitre 6

Conclusion

Pour conclure, l'architecture distribuée représente une conception différente de la programmation dont nous avions pas l'habitude lors des années précédentes. Les services AWS représente de nombreux avantages : ils sont très maniables, faciles à utiliser, flexibles, très sécurisés les rendant ainsi fiables. Par ailleurs, ils peuvent êre exploités dans de nombreux cas, ce qui souligne un peu plus leur flexibilité. Et ce n'est d'ailleurs pas une surprise si ils sont utilisés par les plus grandes entreprises telles que Netflix, Expedia, Sysco ou encore Airbnb.