Tp4 : CouchDB et MapReduce

Malek SLOKOM

1 Introduction à CouchDB

1.1 Qu'est-ce que CouchDB ?

CouchDB est une base de données NoSQL qui stocke les données sous forme de documents JSON. Ces documents peuvent être indexés et traités par des vues utilisant le paradigme MapReduce. CouchDB est conçu pour être flexible et permettre le traitement de grandes quantités de données de manière distribuée. Dans le contexte du TP, nous allons utiliser CouchDB pour stocker des matrices représentant des liens entre pages web et calculer des résultats en utilisant des traitements MapReduce. Ses principales caractéristiques sont :

- **Facile à installer et utiliser.**
- **Open Source** sous licence Apache.
- Utilise une **API REST** :
  - **GET** : Récupérer des ressources.
  - **PUT** : Créer des ressources.
  - **POST** : Envoyer des données.
  - **DELETE** : Supprimer des ressources.

1.2 Installation de CouchDB

Via Docker

Pour exécuter CouchDB avec Docker, utilise la commande suivante :

```
docker run -d --name couchDbDemo -e COUCHDB_USER=abc -e
   COUCHDB_PASSWORD=123 -p 5984:5984 couchdb
```

Listing 1: Installation de CouchDB via Docker

Interface graphique : Accessible à l'adresse suivante :

`http://localhost:5984/_utils`

1.3 Vérification du serveur CouchDB

Pour vérifier si CouchDB est actif :

```
curl -X GET http://abc:123@localhost:5984/
```

Listing 2: Vérification du serveur

2 Manipulation des bases de données

2.1 Création d'une base de données

Pour créer une base de données appelée `films` :

```
curl -X PUT http://abc:123@localhost:5984/films
```

Listing 3: Création d'une base de données

2.2 Insertion de documents

Insertion d'un document simple

```
curl -X PUT http://abc:123@localhost:5984/films/doc1 -d '{"titre
    ":"Inception", "annee":2010}'
```

Listing 4: Insertion d'un document simple

Insertion en masse

Pour insérer plusieurs documents :

```
curl -X POST http://abc:123@localhost:5984/films/_bulk_docs -d
    @films_couchdb.json -H "Content-Type: application/json"
```

Listing 5: Insertion en masse

Exemple de fichier JSON :

```
{
  "docs": [
    {"_id": "movie:1001", "titre": "Inception", "annee": 2010},
    {"_id": "movie:1002", "titre": "Interstellar", "annee": 2014}
  ]
}
```

Listing 6: Exemple de fichier JSON

2.3 Récupération d'un document

Pour récupérer un document avec un identifiant donné :

```
curl -X GET http://abc:123@localhost:5984/films/movie:1001
```

Listing 7: Récupération d'un document

3 MapReduce avec CouchDB

3.1 Qu'est-ce que MapReduce ?

MapReduce est un modèle de programmation conçu pour traiter de grandes quantités de données en parallèle sur un cluster de machines. Il se compose de deux étapes principales :

- **Map** : chaque entrée de données est traitée indépendamment, et une paire clé-valeur est produite.
- **Reduce** :les paires clé-valeur sont agrégées par clé pour produire un résultat final.

Cela permet de répartir le travail de manière efficace et de traiter de grands volumes de données.

3.2 Exemple : Nombre de films par année

3.2.1 Fonction Map

La fonction Map émet l'année comme clé et le titre comme valeur :

```
function (doc) {
  emit(doc.annee, doc.titre);
}
```

Listing 8: Fonction Map

Exemple de résultat intermédiaire :

```
{"key":2010, "value":"Inception"}
{"key":2014, "value":"Interstellar"}
```

Listing 9: Résultat intermédiaire

3.2.2 Fonction Reduce

La fonction Reduce compte le nombre de titres par année :

```
function (keys, values) {
  return values.length;
}
```

Listing 10: Fonction Reduce

Résultat final :

```
{"key":2010, "value":1}
{"key":2014, "value":1}
```

Listing 11: Résultat final

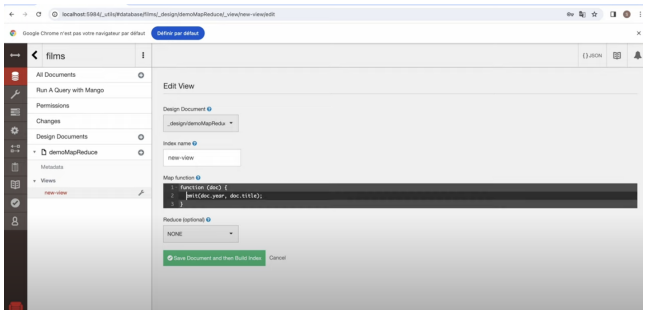

Figure 1: Fonction Map

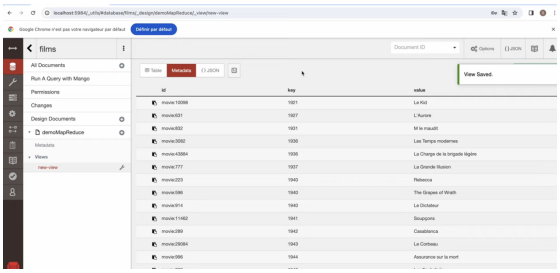

Figure 2: Résultat de Map

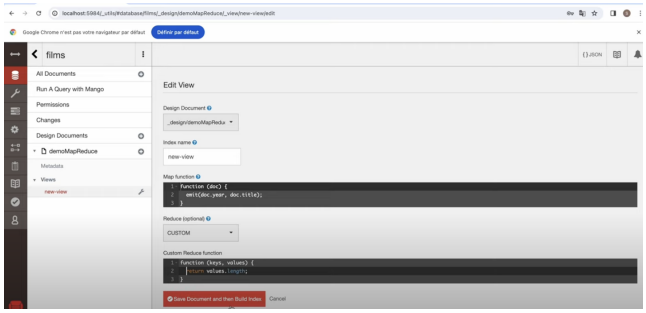

Figure 3: Fonction Reduce

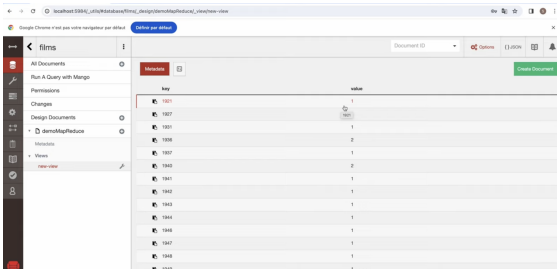

Figure 4: Résultat de Reduce

3.3 Exemple : Nombre de films pour chaque acteur

Fonction Map

Si l'on souhaite compter le nombre de films pour chaque acteur :

```
function (doc) {
  for (i = 0; i < doc.actors.length; i++) {
    emit({"pr nom": doc.actors[i].rst_name, "nom": doc.actors[i].
     last_name}, doc.title);
  }
}
```

Listing 12: Fonction Map pour les acteurs

Fonction Reduce

Pour compter le nombre de films par acteur :

```
function (keys, values) {
  return values.length;
}
```

Listing 13: Fonction Reduce

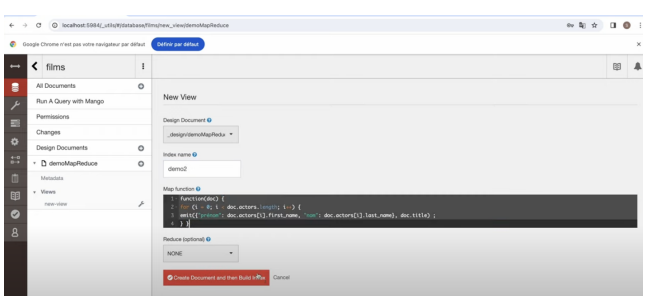

Figure 5: Fonction Map

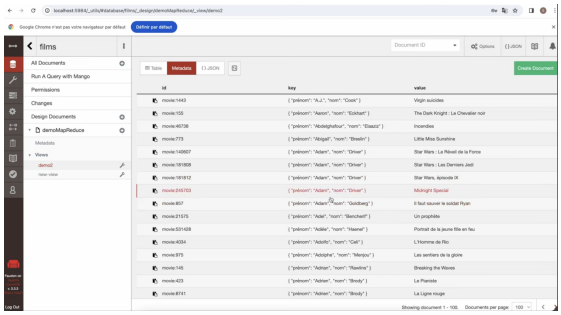

Figure 6: Résultat de Map

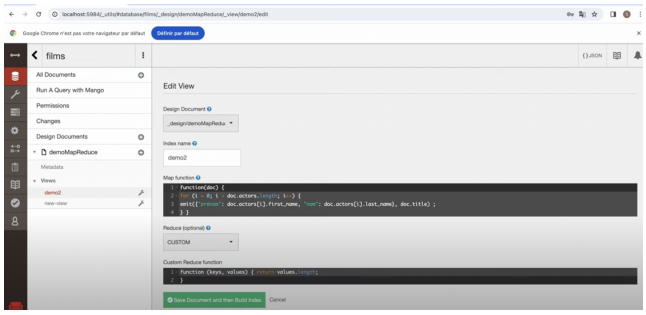

Figure 7: Fonction Reduce

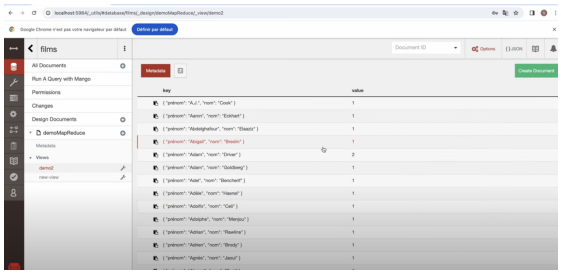

Figure 8: Résultat de Reduce

4 Conclusion

CouchDB, avec son approche simple et son moteur MapReduce, est idéal pour travailler avec des bases de données documentaires et traiter de grandes quantités de données.

5 Exercice n°1 : Modèle et traitement MapReduce

Exercice n° 1 : soit une matrice M de dimension $N \times N$ représentant des liens d'un très grand nombre de pages web (soit N). Chaque lien est étiqueté par un poids (son importance).

1. Proposer un modèle, sous forme de ducuments structurés, pour représenter une telle matrice (s'inspirer du cas Page Rank du moteur de recherche Google, vu en cour). Soit C la collection ainsi obtenue.
2. La ligne i peut être vue comme un vecteur à N dimensions décrivant la page P_i. Spécifiez le traitement MapReduce qui calcule la norme de ces vecteurs à partir des documents de la collection C. La norme d'un vecteur $V(v_1, v_2, \ldots, v_N)$ est le scalaire $||V|| = \sqrt{v_1^2 + v_2^2 + \ldots + v_N^2}$
3. Nous voulons calculer le produit de la matrice M avec un vecteur de dimension N, $W(w_1, w_2, \ldots, w_N)$. Le résultat est un vecteur $\phi = \sum_{j=1}^{N} M_{ij} w_j$, On suppose que le vecteur W tient en mémoire RAM et est accessible comme variable statique par toutes les fonctions de Map ou de Reduce. Spécifiez le traitement MapReduce qui implante ce calcul.

Figure 9: Exercice 1

5.1 Modèle de données

La matrice M que nous voulons traiter représente les liens entre N pages web. Chaque élément M_{ij} de la matrice représente le lien entre la page P_i et la page P_j, et chaque lien a un poids représentant son importance.

Chaque ligne de la matrice peut être vue comme un vecteur représentant une page P_i, où chaque composant du vecteur M_{ij} est un poids de lien entre la page P_i et P_j. Pour stocker cela dans CouchDB, chaque document pourrait être structuré comme suit :

```
{
  "_id": "page_1",
  "links": {
    "page_1": 0.1,
    "page_2": 0.3,
    "page_3": 0.5
  }
}
```

Ce document représente la page P_1, avec des liens vers P_1, P_2, et P_3, et leurs poids respectifs.

5.2 Calcul de la norme d'un vecteur

Dans le cas de notre matrice, chaque ligne de la matrice M représente un vecteur. Pour calculer la norme de chaque vecteur dans la collection de documents C, nous utiliserons un traitement MapReduce. L'étape Map consistera à extraire les valeurs des liens (les éléments de chaque ligne de la matrice), et l'étape Reduce agrégra la somme des carrés de ces valeurs.

5.2.1 Étape Map

La fonction Map extraira les valeurs des liens pour chaque ligne.

```
function map(doc) {
    var sum = 0;
    for (var page in doc.links) {
      sum += Math.pow(doc.links[page], 2);
    }
    emit(doc._id, sum);
}
```

5.2.2 Étape Reduce

La fonction Reduce additionnera les valeurs émises par chaque ligne et renverra la norme.

```
function reduce(keys, values, rereduce) {
  var total = 0;
  for (var i = 0; i < values.length; i++) {
    total += values[i];
  }
  return Math.sqrt(total);
}
```

5.3 Calcul du produit de la matrice M avec un vecteur W

Dans ce cas, le vecteur W est stocké en mémoire RAM et est accessible à toutes les fonctions Map et Reduce. Le traitement MapReduce pour ce calcul consistera à multiplier chaque élément de la matrice M par le poids correspondant dans W.

5.3.1 Étape Map

La fonction Map multiplie chaque poids de la matrice par l'élément correspondant du vecteur W.

```
function map(doc) {
  var vectorW ;
    var result = 0;
    for (var page in doc.links) {
      var index = parseInt(page.split('_')[1]);
      result += doc.links[page] * vectorW[index - 1];
    }
    emit(doc._id, result);
}
```

5.3.2 Étape Reduce

La fonction Reduce agrègera les résultats de chaque ligne.

```
function reduce(keys, values, rereduce) {
  var total = 0;
  for (var i = 0; i < values.length; i++) {
    total += values[i];
  }
  return total;
}
```