Rapport : Introduction à CouchDB et au MapReduce

Contexte et objectifs

Nous avons déjà exploré différentes bases de données NoSQL :

- **Redis** : Une base clé-valeur en mémoire, axée sur la rapidité et la simplicité des opérations (listes, ensembles, etc.).
- **MongoDB** : Un système orienté documents (JSON), offrant un langage de requête avancé et des pipelines d'agrégation.

Désormais, nous nous intéressons à **CouchDB**, une base de données NoSQL distribuée qui s'appuie sur :

- Un modèle documentaire (documents JSON).
- Une interface REST (toutes les opérations se font via des requêtes HTTP).
- L'utilisation du paradigme **MapReduce** pour l'analyse et l'agrégation de données.

Au terme de ce document, nous souhaitons :

1. Comprendre les bases de **CouchDB** : installation, interface, interaction via HTTP.
2. Savoir effectuer des opérations CRUD (Create, Read, Update, Delete) sur des documents JSON.
3. Mettre en place des fonctions MapReduce dans CouchDB pour des vues agrégées.
4. Comparer **CouchDB** avec **MongoDB** et **Redis**, en particulier sur la question de l'agrégation de données.

---

1. Présentation de CouchDB

1.1 Qu'est-ce que CouchDB ?

CouchDB est un système de gestion de bases de données **NoSQL** orienté documents. Ses principales caractéristiques sont :

- **Open Source** : Développé par la fondation Apache.
- **API REST** : Toute action de lecture/écriture se fait via des requêtes HTTP (GET, PUT, POST, DELETE).
- **Document Store** : Les données sont stockées sous forme de documents JSON, sans schéma prédéfini.
- **Distribution et résilience** : Conçu pour la réplication et la montée en charge horizontale.

Ainsi, **CouchDB** diffère de :
- **Redis** : Une base clé-valeur en mémoire.
- **MongoDB** : Un document store avec un langage de requête et de puissants pipelines d'agrégation.

1.2 Installation et accès

L'installation peut se faire localement ou via Docker. Par exemple :

```
docker run -d --name couchdbdemo \
    -e COUCHDB_USER=Yacine -e COUCHDB_PASSWORD=Lemouel \
    -p 5984:5984 couchdb
```

Ensuite, on accède à l'interface **Fauxton** via l'URL :

```
http://localhost:5984/_utils
```

L'authentification se fait avec l'utilisateur et le mot de passe spécifiés. On peut aussi gérer la base via des outils comme `curl` ou **Postman**.

---

2. Interactions avec CouchDB via HTTP

Comme mentionné, **CouchDB** s'appuie sur les verbes HTTP :

- **GET** : Récupérer une ressource (liste des bases, un document précis, etc.).
- **PUT** : Créer ou mettre à jour une ressource existante (une base, un document à l'ID donné).
- **POST** : Ajouter une ressource quand on ne connaît pas l'ID à l'avance ou pour insérer plusieurs documents.
- **DELETE** : Supprimer une ressource via son ID et sa révision.

2.1 Créer une base de données

```
curl -X PUT http://Yacine:Lemouel@localhost:5984/films
```

Cette requête crée une base de données nommée `films`.

2.2 Insérer des documents

Insertion avec un ID spécifique (PUT) :

```
curl -X PUT http://Yacine:Lemouel@localhost:5984/films/doc1 \
    -H "Content-Type: application/json" \
    -d '{"titre":"Inception","annee":2010}'
```

Insertion sans ID prédéfini (POST) :

```
curl -X POST http://Yacine:Lemouel@localhost:5984/films \
    -H "Content-Type: application/json" \
    -d '{"titre":"Interstellar","annee":2014}'
```

2.3 Lire un document (GET)

```
curl -X GET http://Yacine:Lemouel@localhost:5984/films/doc1
```

Cette requête renvoie le document doc1 au format JSON.

2.4 Mettre à jour et supprimer (PUT et DELETE)

Pour mettre à jour un document, on doit connaître sa dernière révision (_rev). Pour le supprimer, on doit aussi spécifier la révision :

```
curl -X DELETE "http://Yacine:Lemouel@localhost:5984/films/doc1?
        rev=1-..."
```

2.5 Importer un lot de documents

Pour insérer plusieurs documents en une seule requête :

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

---

3. Introduction à MapReduce dans CouchDB

Le modèle **MapReduce**, rendu populaire par Hadoop, consiste en deux étapes :

1. **Map** : Parcourir chaque document indépendamment, pour produire une ou plusieurs paires (clé, valeur).
2. **Reduce** : Regrouper les valeurs par clé et appliquer une opération d'agrégation (somme, moyenne, etc.).

3.1 Vues MapReduce dans CouchDB

Dans **CouchDB**, les fonctions **map** et **reduce** (en JavaScript) sont enregistrées sous forme de « vues ».
- La map est obligatoire, on peut l'utiliser seule ou l'accompagner d'une reduce.
- La reduce sert à agréger les valeurs issues de la map.

Exemple : Compter le nombre de films par année

- **Fonction map** :

```
function(doc) {
  if (doc.annee && doc.titre) {
    emit(doc.annee, 1);
  }
}
```

Chaque film émet (année, 1).

- **Fonction reduce** (pour compter) :

```
function(keys, values, rereduce) {
  return sum(values);
}
```

Le moteur de CouchDB additionne alors tous les « 1 » émis pour chaque année afin d'obtenir le décompte par année.

Exemple : Compter le nombre de films par acteur

Si chaque document `film` possède un tableau `actors`, nous émettons un 1 pour chaque acteur :

- **Map** :

```
function(doc) {
  if (doc.actors && doc.actors.length > 0) {
    for (var i = 0; i < doc.actors.length; i++) {
      var acteur = doc.actors[i].first_name + " " +
        doc.actors[i].last_name;
      emit(acteur, 1);
    }
  }
}
```

- **Reduce** (même principe) :

```
function(keys, values, rereduce) {
  return sum(values);
}
```

3.2 Comparaison avec MongoDB et Redis

- **MongoDB** : Propose un pipeline d'agrégation complet, un langage de requête riche, assez proche de SQL dans sa philosophie.
- **Redis** : Orienté clé-valeur en mémoire, moins adapté aux agrégations complexes.
- **CouchDB** : Mise sur la **programmation** des agrégations en JavaScript (MapReduce), ce qui offre de la flexibilité, en particulier sur de larges volumes de données distribuées.

---

4. Avantages et inconvénients de CouchDB/ MapReduce

Avantages :
- Écriture flexible des vues (JavaScript).
- Adapté aux environnements distribués et tolérants aux pannes.
- Schéma libre, permettant de gérer des données hétérogènes.

Inconvénients :
- Aucun langage de requête aussi avancé que celui de MongoDB.
- Besoin de bien maîtriser JavaScript et MapReduce.
- Gestion de la cohérence des données reposant beaucoup sur la logique d'application.

Exercice : Représentation d'une matrice et calculs MapReduce

Énoncé :
On considère une matrice M de taille N×N, où chaque entrée M[i][j] est un poids (importance d'un lien entre la page i et la page j).
1. Proposer un modèle (sous forme de documents CouchDB) pour représenter cette matrice.
2. Décrire le MapReduce permettant de calculer la **norme** des vecteurs-lignes ($||V_i|| = \sqrt{(v_1^2 + \ldots + v\_N^2)}$).
3. Décrire le MapReduce pour calculer le produit **M × W** (où W est un vecteur de dimension N).

1. Modélisation de la matrice

Chaque ligne i est représentée par un document :

```
{
  "page_id": i,
  "links": [
    { "target": j1, "weight": w_i_j1 },
    { "target": j2, "weight": w_i_j2 },
    ...
    { "target": jk, "weight": w_i_jk }
  ]
}
```

- `page_id` identifie la page (ligne i).
- `links` liste les liaisons sortantes : la page cible (target) et le poids (weight).

2. Calcul de la norme des lignes

Pour chaque document (ligne i), on calcule la somme des carrés des poids, puis on prend la racine carrée.

- **Map** :

```
function map(doc) {
  var i = doc.page_id;
  var sum_of_squares = 0;
  for (var k = 0; k < doc.links.length; k++) {
    var w = doc.links[k].weight;
    sum_of_squares += w * w;
  }
  emit(i, sum_of_squares);
}
```

- **Reduce** :

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

Interprétation :
- La fonction map émet la somme des carrés pour chaque ligne i.
- La fonction reduce agrège ces sommes (dans ce cas, il n'y a qu'une valeur par i si on regroupe par clé) et calcule la racine carrée.

3. Calcul du produit M × W

Soit un vecteur W = (w_1, w_2, …, w_N) en mémoire (variable globale). Nous voulons pour chaque ligne i :

$\phi[i] = \sum(M[i][j] \times W[j])$.

- **Map** :

```
// On suppose que le vecteur W est accessible globalement
function map(doc) {
  var i = doc.page_id;
  for (var k = 0; k < doc.links.length; k++) {
    var j = doc.links[k].target;
    var w_ij = doc.links[k].weight;
    emit(i, w_ij * W[j]);
```

```
  }
}
```

- **Reduce** :

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

Interprétation :
- La fonction map multiplie chaque poids par l'élément correspondant de W (W[j]) et émet la valeur partielle pour la clé i.
- La fonction reduce additionne toutes les valeurs associées à i.

Conclusion

Avec **CouchDB**, nous avons un aperçu d'une autre approche NoSQL :
- Gestion des documents via **HTTP** (API REST).
- **MapReduce** pour l'agrégation et l'analyse, plutôt que des requêtes déclaratives.

L'exercice sur la matrice M illustre deux usages classiques :
1. Le calcul de la **norme** de chaque ligne.
2. Le calcul du **produit M × W**.

Ces exemples montrent la puissance de MapReduce dans un environnement distribué, tout en soulignant les différences avec Redis (simple clé-valeur en mémoire) et MongoDB (requêtes et agrégations plus déclaratives). Au final, CouchDB nous permet d'étendre notre vision de l'écosystème NoSQL, chaque solution répondant à des besoins spécifiques en termes de performance, de structure, de distribution et de flexibilité.
