Rapport TP4

MapReduce

Addou Aicha Amira

Bombay Marc

3ème Année Ingénieur - INFO - Apprentissage

Introduction

Lors du précédent TP, nous avons étudié plusieurs mécanismes clés des bases de données NoSQL, en particulier ceux qui permettent de passer à l'échelle et d'assurer la tolérance aux pannes : le sharding et la réplication.

Cependant, ces mécanismes introduisent une difficulté majeure : chaque machine ne dispose que d'une partie des données globales. Il devient alors nécessaire de trouver un moyen efficace pour effectuer des calculs globaux (comptages, moyennes, statistiques, agrégations) sur des données distribuées.

C'est précisément pour répondre à ce besoin que le modèle MapReduce a été conçu.

Principe du modèle MapReduce

MapReduce est un modèle de programmation destiné aux traitements distribués sur de grandes quantités de données. Son objectif principal est de permettre l'exécution de calculs parallèles sur plusieurs nœuds, tout en masquant la complexité liée à la distribution des données.
Le fonctionnement de MapReduce repose sur deux étapes fondamentales :

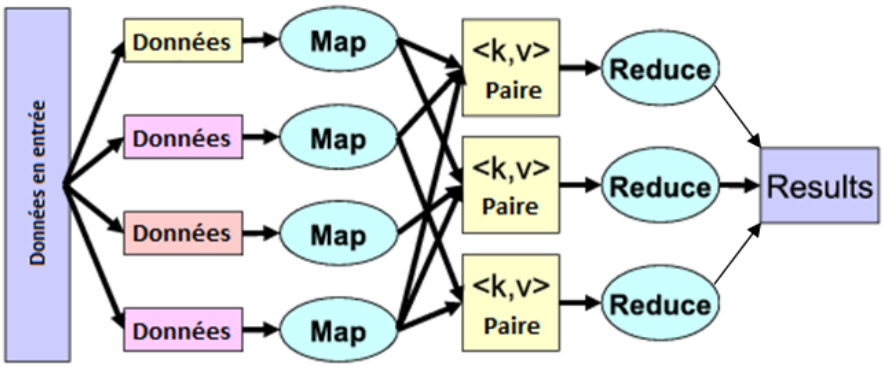
Phase Map

Chaque nœud traite localement les données qu'il héberge. La fonction map prend en entrée un document et produit une ou plusieurs paires clé / valeur.
Exemple : Associer un genre à une série

Phase Reduce

Les paires produites lors de la phase map sont regroupées par clé. La fonction reduce est ensuite appliquée afin d'agréger l'ensemble des valeurs associées à une même clé.
Exemple :Addition de compteurs, Calcul de moyennes

Grâce à cette approche, il est possible d'effectuer des calculs globaux sans jamais centraliser toutes les données sur une seule machine.



CouchDB

Apache CouchDB est une base de données NoSQL orientée documents, pensée dès sa conception pour fonctionner dans des environnements distribués et tolérants aux pannes. Chaque document est identifié par un identifiant unique (_id) et une révision (_rev), ce qui facilite la gestion des conflits lors de la réplication. CouchDB repose sur des principes simples mais efficaces :

- données immuables,
- écritures de type append-only,
- synchronisation par réplication.

MapReduce dans CouchDB

Dans CouchDB, MapReduce constitue le mécanisme central d'interrogation des données. Il n'existe pas de langage de requêtes complexe : les requêtes sont définies sous forme de vues, stockées dans des design documents.
Un aspect fondamental de CouchDB est le caractère persistant des résultats MapReduce. Une fois une vue créée, CouchDB maintient automatiquement son index à jour lors de toute modification des documents.
Cela permet d'obtenir des temps de réponse rapides, sans recalculer l'ensemble des données à chaque requête.

Installation de CouchDB avec Docker

Afin de faciliter l'installation et de garantir un environnement reproductible, CouchDB a été déployé à l'aide de Docker.
Commande de lancement du conteneur :

```
docker run -d \

--name couchdb \

-p 5984:5984 \

-e COUCHDB_USER=admin \

-e COUCHDB_PASSWORD=admin \

couchdb:latest
```

Création de la base et insertion des documents

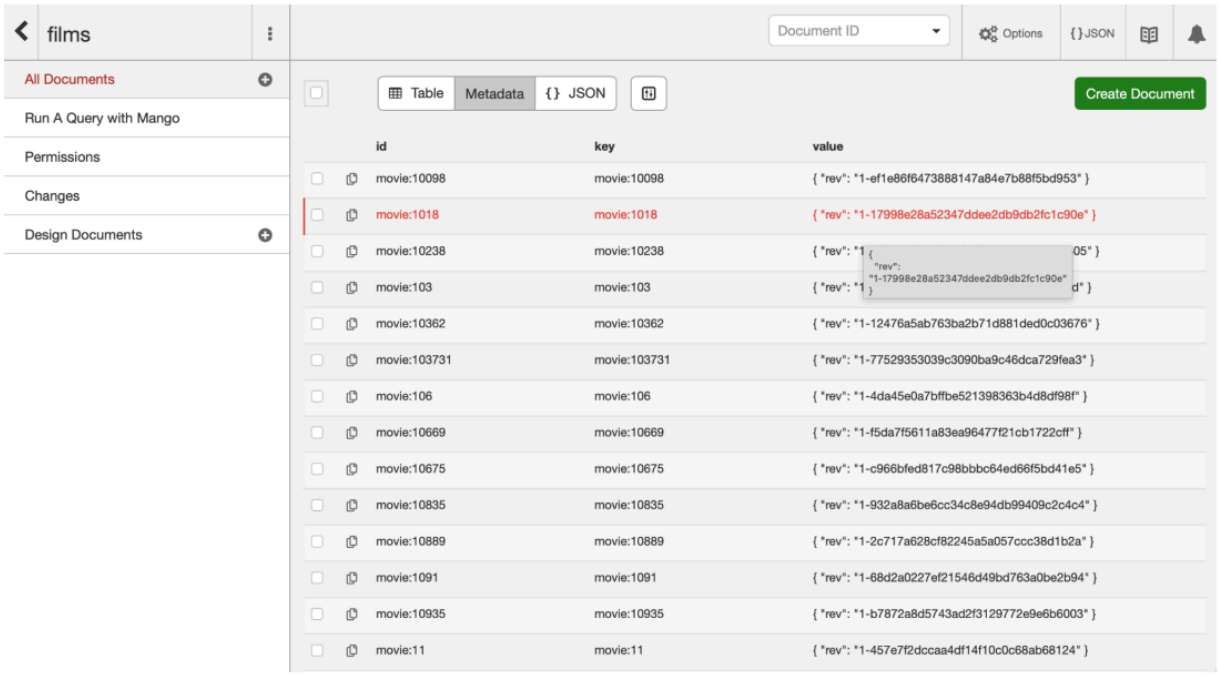
Une base de données contenant des films a été créée, puis alimentée avec plusieurs documents représentant des films, leurs genres, réalisateurs, acteurs et notes.
Cette base a ensuite servi de support pour la réalisation des exercices MapReduce.
La commande suivante permet d'insérer en une seule requête un ensemble de documents JSON dans la base de données films de CouchDB :

```
curl -u admin:admin \

-X POST http://localhost:5984/films/_bulk_docs \

-H "Content-Type: application/json" \

-d @films.json
```



Réponses aux questions

```
//1. Compter le nombre total de films dans la collection.

var mf = function () {emit("count",1)}

var rf = function (k,v) {return Array.sum(v)}

db.films.mapReduce(mf,rf,{out:"res"})

db.res.find()


//2. Compter le nombre de films par genre.

var mf = function () {emit(this.genre,1)}

var rf = function (k,v) {return Array.sum(v)}
```

```
db.films.mapReduce(mf,rf,{out:"res"})

db.res.find()
```

```
// 3. Compter le nombre de films réalisés par chaque réalisateur.

var mf = function () {emit(this.director._id,1)}

var rf = function (k,v) {return Array.sum(v)}

db.films.mapReduce(mf,rf,{out:"res"})

db.res.find()
```

```
// 4. Compter le nombre d’acteurs uniques apparaissant dans tous les
films.

// L'unicité est faite par le mapReduce. Comme on n'a pas de clé pour les
acteurs, on choisit ici nom+prénom en clé.

var mf=function(){if(!this.actors)return;
this.actors.forEach((a)=>{emit(a.last_name+" "+a.first_name,1)})}

var rf = function (k,v) {return 1}

db.films.mapReduce(mf,rf,{out:"res"})

db.res.find()
```

```
// 5. Lister le nombre de films par année de sortie.

var mf = function () {emit(this.year,1)}

var rf = function (k,v) {return Array.sum(v)}

db.films.mapReduce(mf,rf,{out:"res"})

db.res.find()
```

```
// 6. Calculer la note moyenne par film à partir du tableau grades.

var mf=function(){this.grades.forEach((g)=>(emit(this._id,g.note)))}

var rf=function(k,v){return Array.sum(v)/v.length}

db.films.mapReduce(mf,rf,{out:"res"})

db.res.find()


// 7. Calculer la note moyenne par genre.

var mf=function(){this.grades.forEach((g)=>(emit(this.genre,g.note)))}

var rf=function(k,v){return Array.sum(v)/v.length}

db.films.mapReduce(mf,rf,{out:"res"})

db.res.find()


// 8. Calculer la note moyenne par réalisateur.

var mf =function()
{this.grades.forEach((g)=>(emit(this.director._id,g.note)))}

var rf=function(k,v){return Array.sum(v)/v.length}

db.films.mapReduce(mf,rf,{out:"res"})

db.res.find()


// 9. Trouver le film avec la note maximale la plus élevée.

// En deux temps: max par film (mapreduce) puis max global (.sort
décroissant, suffit le lire le 1er)

var mf=function(){this.grades.forEach((g)=>emit(this._id,g.note))}

var rf=function(k,v){return Math.max.apply(null,v)}

db.films.mapReduce(mf,rf,{out:"res"})
```

```
db.res.find().sort({value:-1})
```

```
// 10. Compter le nombre de notes supérieures à 70 dans tous les films.

var mf=function(){this.grades.forEach((g)=>emit(this._id,g.note))}

var rf=function(k,v){return v.reduce((total,x) => (x > 70 ? total+1 :
total),0)}

db.films.mapReduce(mf,rf,{out:"res"})

db.res.find()
```

```
// 11. Lister tous les acteurs par genre, sans doublons.

var mf=function(){if(!this.actors)return;
this.actors.forEach((a)=>{emit(this.genre,a.last_name+" "+a.first_name)})}

var rf=function(k,v){return [... new Set(v)]} // Supprime les entrées en
double

db.films.mapReduce(mf,rf,{out:"res"})

db.res.find()
```

```
// 12. Trouver les acteurs apparaissant dans le plus grand nombre de
films.

var mf=function(){if(!this.actors)return;
this.actors.forEach((a)=>{emit(a.last_name+" "+a.first_name,1)})}

var rf=function(k,v){return Array.sum(v)}

db.films.mapReduce(mf,rf,{out:"res"})

db.res.find().sort({value:-1})
```

```
// 13. Classer les films par lettre de grade majoritaire ( A , B , C ,
etc.)
```

```
var mf=function(){var gc = {};this.grades.forEach((g)=>(gc[g.grade] = (gc[g.grade] || 0)+1));m=-Infinity;dg="?";for([k,v] of Object.entries(gc)){if(v>m){m=v;dg=k}}emit(dg,this.title)}

var rf=function(k,v){return v}

db.films.mapReduce(mf,rf,{out:"res"})

db.res.find()
```

```
// 14. Calculer la note moyenne par année de sortie des films.

var mf=function(){this.grades.forEach((g)=>emit(this.year,g.note))}

var rf=function(k,v){return Array.sum(v)/v.length}

db.films.mapReduce(mf,rf,{out:"res"})

db.res.find()
```

```
// 15. Identifier les réalisateurs dont la note moyenne de tous leurs films est supérieure à 80.

var mf=function(){this.grades.forEach((g)=>emit(this.director._id,g.note))}

var rf=function(k,v){return Array.sum(v)/v.length}

db.films.mapReduce(mf,rf,{out:"res"})

db.res.find({value:{$gte:80}})
```

Manipulation matrice

Modélisation d’une matrice creuse

Pour représenter une très grande matrice N×NN \times NN×N, il est préférable d'adopter une représentation creuse(sparse): seuls les liens existants sont stockés.

1 ligne par document -> le ièeme document Xi va contenir a ligne i et la liste de ses liens sortants avec leur poids Mij

Exemple de document :

```
{
  "_id": "document:Xi",
  "i": 123,
  "outlinks": [
    { "j": 27, "w": 0.42 },
    { "j": 63, "w": 0.10 }
  ]
}
```

Calcul de la norme des vecteurs (une norme par ligne i)

$$\|M_i\| = \sqrt{\sum_j M[i][j]^2}$$

L'objectif est de calculer, pour chaque ligne i de la matrice M, la norme du vecteur correspondant. Étant donné que la matrice est stockée sous forme creuse, seules les valeurs non nulles sont prises en compte.

Phase Map

Pour chaque élément M[i][j], on émet la valeur au carré associée à l'indice de ligne i. Cela permet de préparer le calcul de la somme des carrés pour chaque ligne.

```
function (doc) {
  if (doc.outlinks) {
    doc.outlinks.forEach(function (e) {
```

```
      emit(doc.i, e.w * e.w);
    });
  }
}
```

Phase Reduce

Les valeurs associées à une même ligne i sont additionnées afin d'obtenir la somme des carrés. La racine carrée de cette somme est ensuite calculée pour obtenir la norme du vecteur.

```
function (keys, values, rereduce) {
  let sommeCarree = 0;
  values.forEach(function (v) {
    sommeCarree += v;
  });
  return Math.sqrt(sommeCarree);
}
```

Produit matrice–vecteur

$$(M \times W)_i = \sum_j M[i][j] \times W[j]$$

Cette opération consiste à calculer le produit d'une matrice mpar un vecteur W, en exploitant le modèle MapReduce pour traiter les données distribuées.

Phase Map

Chaque élément M[i][j] est regroupé par ligne i. Pour cela, on conserve à la fois l'indice de colonne j et la valeur correspondante, ce qui permettra par la suite d'accéder facilement à la composante adéquate du vecteur W

```
function (doc) {
  if (doc.outlinks) {
    doc.outlinks.forEach(function (e) {
      emit(doc.i, { j: e.j, v: e.w });
    });
  }
}
```

Phase Reduce

Pour chaque ligne i, on calcule la somme des produits, ce qui correspond à la composante i du vecteur résultat.

```
function (keys, values, rereduce) {
  var s = 0;
  values.forEach(function (e) {
    s += e.v * W[e.j];
  });
  return s;
}
```

Conclusion

Ce TP a permis de mieux comprendre le fonctionnement du modèle MapReduce et son intégration dans CouchDB. À travers différents exercices pratiques, nous avons pu manipuler des vues MapReduce pour effectuer des calculs distribués sur une base de données orientée documents.

Cette approche illustre l'intérêt des bases NoSQL pour le traitement de grands volumes de données réparties, tout en conservant des performances satisfaisantes et une bonne tolérance aux pannes.