

Bases de Données NoSQL

## **Compte Rendu TP**

MapReduce avec CouchDB et MongoDB

Rayane Sendjakedine

Année 2025-2026

# 1 Introduction au MapReduce

Le MapReduce est un modèle de programmation qui permet de traiter de gros volumes de données de manière parallèle et distribuée. Le concept est assez simple : on a deux fonctions principales, **map** et **reduce**, qui travaillent ensemble pour transformer et agréger des données.

Le principe de base :

- **Map** : traite chaque document indépendamment et produit des paires (clé, valeur)
- **Shuffle & Sort** : regroupe automatiquement toutes les valeurs ayant la même clé
- **Reduce** : agrège les valeurs pour chaque clé (somme, moyenne, comptage, etc.)

Ce TP se divise en deux parties : d'abord avec CouchDB pour bien comprendre les concepts (car CouchDB permet de voir les résultats intermédiaires), puis avec MongoDB pour des exemples plus pratiques.

## 2 Partie 1 : CouchDB

### 2.1 Présentation de CouchDB

CouchDB est un système de gestion de base de données NoSQL orienté documents, développé par la fondation Apache. Sa particularité est qu'il expose toutes ses fonctionnalités via une API REST, donc on peut tout faire avec les verbes HTTP classiques (GET, POST, PUT, DELETE).

**Installation avec Docker :**

```
1 docker run -d --name couchdb_demo \  
2   -e COUCHDB_USER=admin \  
3   -e COUCHDB_PASSWORD=password \  
4   -p 5984:5984 \  
5   couchdb
```

L'interface web est accessible sur [http://localhost:5984/\\_utils](http://localhost:5984/_utils).

### 2.2 Manipulation de base avec l'API REST

**Créer une base de données :**

```
1 curl -X PUT http://admin:password@localhost:5984/films
```

**Insérer un document :**

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

**Récupérer un document :**

```
1 curl -X GET http://admin:password@localhost:5984/films/doc1
```

**Importer une collection de films :**

```
1 curl -X POST http://admin:password@localhost:5984/films/_bulk_docs \  
2   -H "Content-Type: application/json" \  
3   -d @films_couchdb.json
```

### 2.3 MapReduce avec CouchDB

L'avantage de CouchDB c'est qu'on peut exécuter juste la fonction **map** sans le **reduce**, ce qui permet de voir les résultats intermédiaires. C'est très utile pour comprendre.

### 2.3.1 Exemple 1 : Nombre de films par année

**Objectif :** Calculer combien de films sont sortis chaque année.

**Fonction Map :**

```
1 function(doc) {  
2   emit(doc.annee, doc.titre);  
3 }
```

Cette fonction prend chaque document et émet une paire (année, titre). La clé c'est l'année, et c'est elle qui va servir au regroupement.

**Fonction Reduce :**

```
1 function(keys, values, rereduce) {  
2   return values.length;  
3 }
```

Le reduce compte simplement le nombre de valeurs pour chaque clé (donc chaque année).

**Résultat :** On obtient quelque chose comme :

```
1 {1936: 2, 1950: 1, 1979: 5, ...}
```

C'est exactement comme un `GROUP BY annee` en SQL, sauf que là c'est distribué.

### 2.3.2 Exemple 2 : Nombre de films par acteur

**Objectif :** Compter combien de films a fait chaque acteur.

**Fonction Map :**

```
1 function(doc) {  
2   for (var i = 0; i < doc.actors.length; i++) {  
3     var actor = doc.actors[i];  
4     emit(actor.prenom + " " + actor.nom, doc.titre);  
5   }  
6 }
```

Ici c'est plus intéressant : comme un film peut avoir plusieurs acteurs (c'est un tableau), on parcourt tous les acteurs et on émet une paire pour chacun. Un seul document peut donc générer plusieurs paires clé-valeur.

**Fonction Reduce :**

```
1 function(keys, values, rereduce) {  
2   return values.length;  
3 }
```

Même logique que l'exemple 1 : on compte.

**Résultat :** On peut voir quel acteur a joué dans le plus de films de notre collection.

## 2.4 Avantages de CouchDB pour apprendre

- On peut voir les résultats de `map` seul
- L'interface graphique est claire
- Les résultats intermédiaires sont sauvegardés (on voit bien le shuffle/sort)
- API REST simple à comprendre

## 3 Partie 2 : Exercices théoriques

### 3.1 Modélisation de la matrice de liens

On veut représenter une matrice  $M$  de dimension  $N \times N$  qui contient les liens entre pages web avec leurs poids (importance). Chaque ligne  $i$  représente les liens sortants de la page  $P_i$ .

**Modèle proposé :**

```

1 {
2   "_id": "page_i",
3   "page_id": i,
4   "liens": [
5     {"destination": j, "poids": M_ij},
6     {"destination": k, "poids": M_ik},
7     ...
8   ]
9 }
```

Chaque document représente une page avec tous ses liens sortants. On stocke seulement les liens non-nuls pour économiser de l'espace (matrice creuse).

### 3.2 Calcul de la norme des vecteurs

On veut calculer  $\|V\| = \sqrt{v_1^2 + v_2^2 + \dots + v_N^2}$  pour chaque ligne de la matrice.

**Fonction Map :**

```

1 function(doc) {
2   var somme_carres = 0;
3   for (var i = 0; i < doc.liens.length; i++) {
4     var poids = doc.liens[i].poids;
5     somme_carres += poids * poids;
6   }
7   emit(doc.page_id, somme_carres);
8 }
```

Pour chaque page, on calcule la somme des carrés de ses poids de liens sortants.

**Fonction Reduce :**

```

1 function(keys, values, rereduce) {
2   if (rereduce) {
3     // Cas ou on re-reduce des resultats deja reduits
4     var somme = 0;
5     for (var i = 0; i < values.length; i++) {
6       somme += values[i];
7     }
8     return somme;
9   } else {
10    // Premier reduce
11    var somme = 0;
12    for (var i = 0; i < values.length; i++) {
13      somme += values[i];
14    }
15    return somme;
16  }
17 }
```

**Finalize (post-traitement) :**

```

1 function(key, value) {
2   return Math.sqrt(value);
}
```

3 }

On prend la racine carrée à la fin pour obtenir la norme.

### 3.3 Produit matrice-vecteur

On veut calculer  $\phi_i = \sum_{j=1}^N M_{ij} \cdot w_j$ .

Le vecteur  $W$  est accessible en mémoire comme variable globale (statique).

**Fonction Map :**

```

1 function(doc) {
2   // W est accessible comme variable globale
3   var resultat = 0;
4   for (var i = 0; i < doc.liens.length; i++) {
5     var j = doc.liens[i].destination;
6     var M_ij = doc.liens[i].poids;
7     resultat += M_ij * W[j]; // W est global
8   }
9   emit(doc.page_id, resultat);
10 }
```

Pour chaque page  $i$ , on calcule sa contribution au produit en parcourant ses liens et en multipliant par les valeurs correspondantes du vecteur  $W$ .

**Fonction Reduce :**

```

1 function(keys, values, rereduce) {
2   // Si la matrice est partitionnee, il faut sommer
3   var somme = 0;
4   for (var i = 0; i < values.length; i++) {
5     somme += values[i];
6   }
7   return somme;
8 }
```

Si plusieurs mappers ont traité des parties de la même ligne (partitionnement par lignes), on somme leurs contributions.

**Résultat :** On obtient le vecteur  $\phi$  où chaque élément est le résultat du produit scalaire de la ligne correspondante avec  $W$ .

## 4 Partie 3 : MapReduce avec MongoDB

MongoDB supporte aussi MapReduce mais la syntaxe est un peu différente. Depuis les versions récentes, MongoDB recommande plutôt d'utiliser l'aggregation pipeline, mais MapReduce reste disponible.

### 4.1 Exemple 1 : Compter le nombre total de films

```

1 // Fonction Map
2 var mapFunction = function() {
3   emit("total", 1);
4 };
5
6 // Fonction Reduce
7 var reduceFunction = function(key, values) {
8   return Array.sum(values);
9 };
10
```

```
11 // Execution
12 db.films.mapReduce(
13     mapFunction,
14     reduceFunction,
15     { out: "resultat_total" }
16 );
17
18 db.resultat_total.find();
```

Tous les documents émettent la même clé "total" avec la valeur 1, puis on somme.

## 4.2 Exemple 2 : Nombre de films par genre

```
1 var mapFunction = function() {
2     emit(this.genre, 1);
3 };
4
5 var reduceFunction = function(key, values) {
6     return Array.sum(values);
7 };
8
9 db.films.mapReduce(
10     mapFunction,
11     reduceFunction,
12     { out: "films_par_genre" }
13 );
```

## 4.3 Exemple 3 : Nombre de films par réalisateur

```
1 var mapFunction = function() {
2     if (this.realisateur) {
3         emit(this.realisateur, 1);
4     }
5 };
6
7 var reduceFunction = function(key, values) {
8     return Array.sum(values);
9 };
10
11 db.films.mapReduce(
12     mapFunction,
13     reduceFunction,
14     { out: "films_par_realisateur" }
15 );
```

## 4.4 Exemple 4 : Note moyenne par film

```
1 var mapFunction = function() {
2     if (this.grades && this.grades.length > 0) {
3         var somme = 0;
4         for (var i = 0; i < this.grades.length; i++) {
5             somme += this.grades[i].score;
6         }
7         emit(this._id, {
8             somme: somme,
```

```

9      count: this.grades.length
10    });
11  }
12 };
13
14 var reduceFunction = function(key, values) {
15   var total = { somme: 0, count: 0 };
16   for (var i = 0; i < values.length; i++) {
17     total.somme += values[i].somme;
18     total.count += values[i].count;
19   }
20   return total;
21 };
22
23 var finalizeFunction = function(key, value) {
24   value.moyenne = value.somme / value.count;
25   return value;
26 };
27
28 db.films.mapReduce(
29   mapFunction,
30   reduceFunction,
31   {
32     out: "notes_moyennes",
33     finalize: finalizeFunction
34   }
35 );

```

Ici on utilise `finalize` pour calculer la moyenne à la fin.

#### 4.5 Exemple 5 : Acteurs uniques

```

1  var mapFunction = function() {
2    if (this.actors) {
3      for (var i = 0; i < this.actors.length; i++) {
4        emit(this.actors[i], 1);
5      }
6    }
7  };
8
9  var reduceFunction = function(key, values) {
10   return 1; // On compte juste la presence
11 };
12
13 db.films.mapReduce(
14   mapFunction,
15   reduceFunction,
16   { out: "acteurs_uniques" }
17 );
18
19 // Compter le nombre total
20 db.acteurs_uniques.count();

```

#### 4.6 Exemple 6 : Films par année de sortie

```

1  var mapFunction = function() {
2    if (this.annee) {

```

```
3     emit(this.annee, 1);
4   }
5 };
6
7 var reduceFunction = function(key, values) {
8   return Array.sum(values);
9 };
10
11 db.films.mapReduce(
12   mapFunction,
13   reduceFunction,
14   { out: "films_par_annee" }
15 );
```