

Bases de Données NoSQL - TP

MAP REDUCE COUCHDB et MONGODB

Prénom: Abdelillah

Nom: SELHI

N°Étudiant: 12206456

Groupe: INFOA3

Introduction

Le modèle **MapReduce** est un mécanisme de programmation distribuée conçu pour le traitement de très grands ensembles de données (**Big Data**) sur des clusters de machines. Son architecture permet de paralléliser les calculs, offrant ainsi une solution robuste et évolutive pour l'analyse de données massives et non structurées.

Utilité :

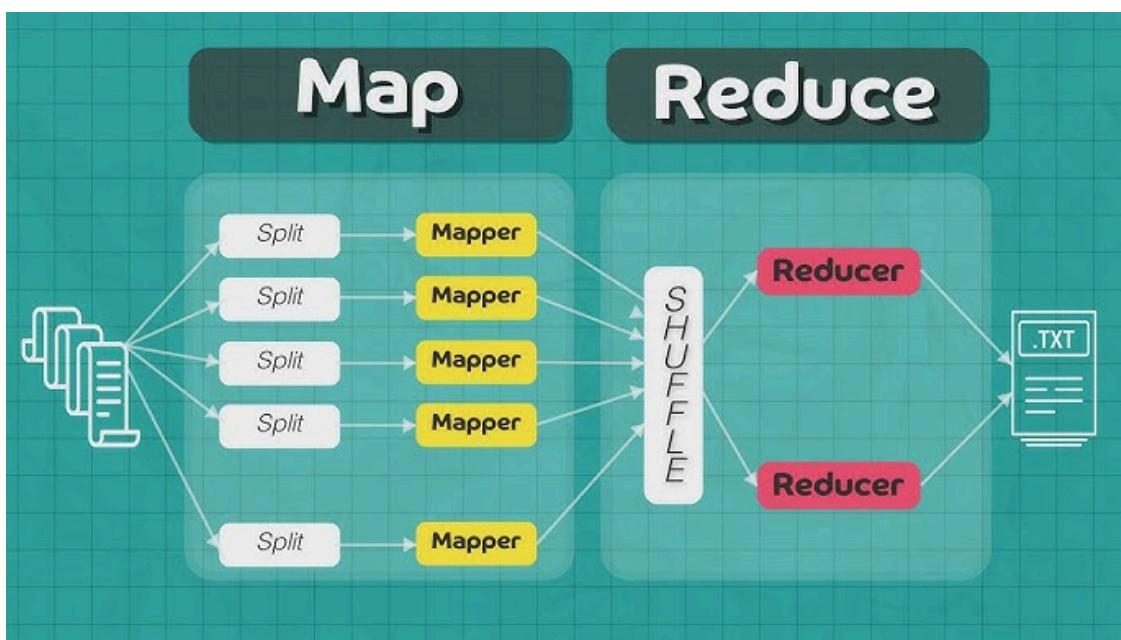
- Il permet une gestion et une analyse efficace des données qui dépassent les capacités des systèmes de bases de données relationnelles traditionnels.
- Il garantit la tolérance aux pannes en distribuant les tâches sur plusieurs nœuds.
- Il est le fondement de nombreux écosystèmes Big Data (comme Hadoop).

Map :

- **Description** : C'est la première étape du processus. Elle prend les données d'entrée, les divise en blocs (split) et les traite pour générer un ensemble de paires clé-valeur intermédiaires.
- **Utilité** : Elle sert principalement à filtrer, organiser et transformer les données brutes en un format adapté à l'agrégation.

Reduce :

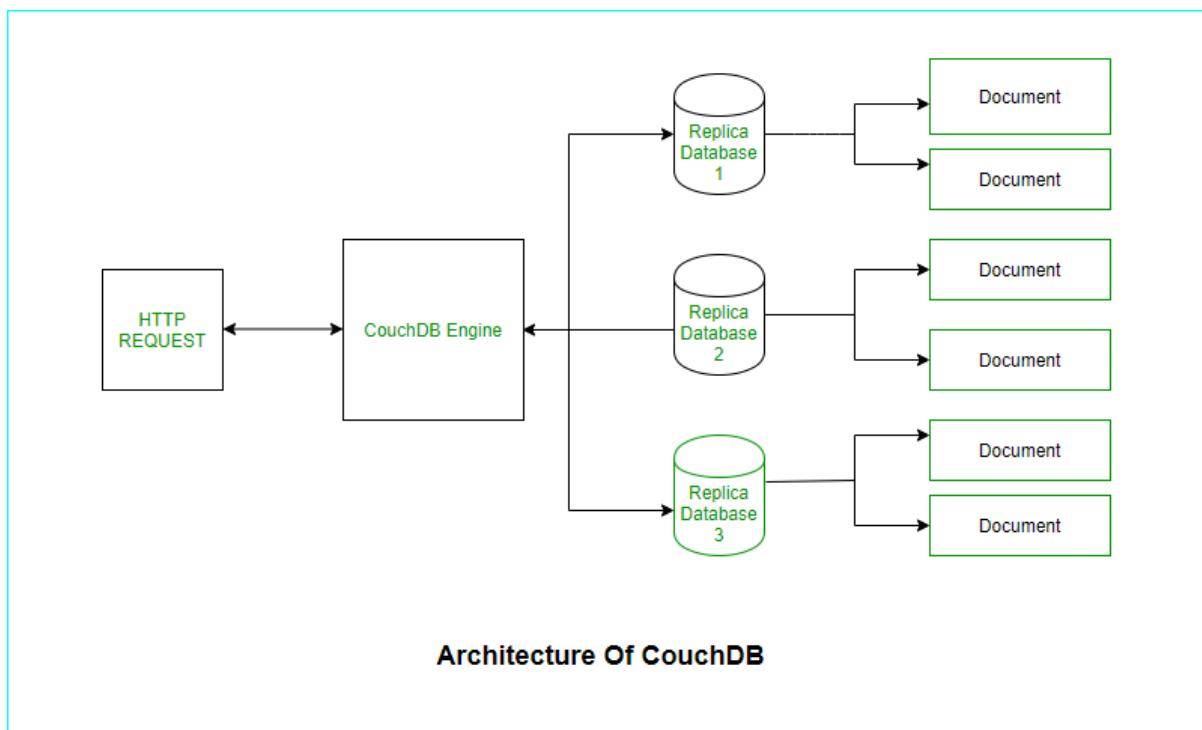
- **Description** : C'est la seconde étape. Elle reçoit toutes les valeurs associées à une même clé intermédiaire et effectue une opération d'agrégation ou de consolidation (somme, comptage, moyenne, etc.).
- **Utilité** : Elle synthétise les données pour fournir le résultat final du traitement.



source: <https://www.youtube.com/watch?v=cHGaQz0E7AU>

CouchDB :

- **Description** : CouchDB est une base de données NoSQL orientée document qui met l'accent sur la simplicité d'accès via HTTP/REST, une idée centrale est de proposer une architecture “web-native” : les opérations CRUD se font via des requêtes HTTP. Elle stocke les données sous forme de documents JSON et se distingue par son modèle de cohérence optimiste, sa réPLICATION facile et son architecture décentralisée.
- **Utilité** :
 - Elle offre une grande flexibilité de schéma grâce à son format JSON.
 - Elle est particulièrement adaptée aux applications mobiles et déconnectées grâce à sa réPLICATION.
 - Elle utilise MapReduce comme mécanisme principal pour créer des vues et exécuter des requêtes d'agrégation sur les données stockées.



source: <https://media.geeksforgeeks.org/wp-content/uploads/20200626162258/CouchDB1.png>

Installation de CouchDB

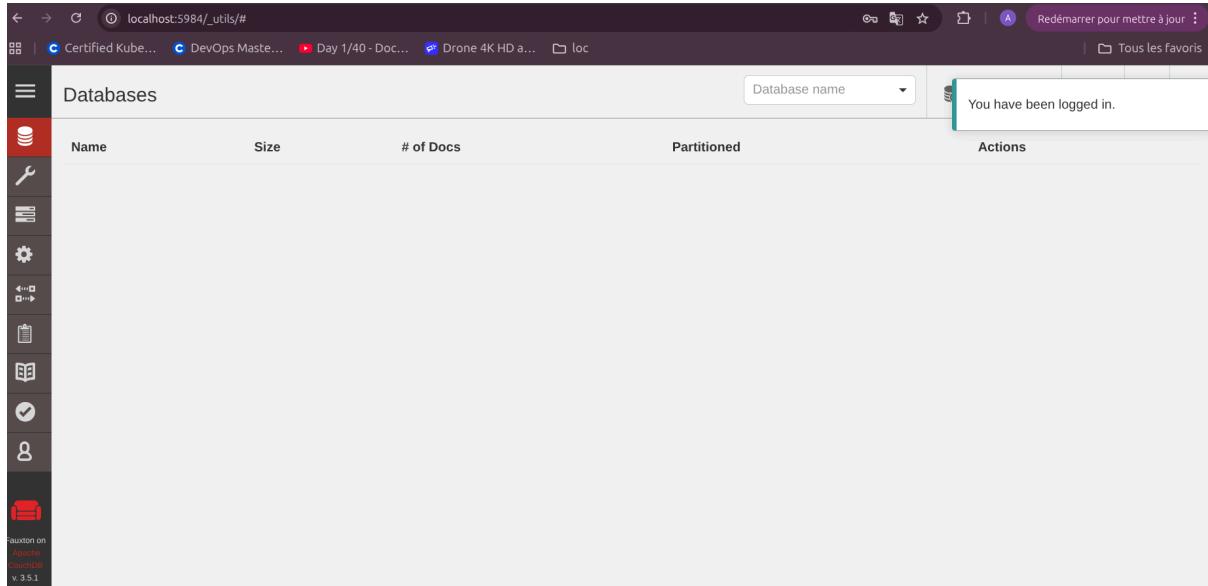
Exécuter la commande:

```
docker run -d \
--name couchdb \
-p 5984:5984 \
-e COUCHDB_USER=admin \
-e COUCHDB_PASSWORD=admin \
couchdb:latest
```

```
adelillah@adelillah-HP-250-G9:~/Documents/INFOA3/NoSQL/NoSQL/tp3$ docker run -d \
--name couchdb \
-p 5984:5984 \
-e COUCHDB_USER=admin \
-e COUCHDB_PASSWORD=admin \
couchdb:latest
Unable to find image 'couchdb:latest' locally
latest: Pulling from library/couchdb
51b27282aaec: Download complete
ae4ce04d0e1c: Extracting 3 s
716c80c273f8: Download complete
7f82cbc9af1f: Download complete
cd726490d7d0: Downloading [=====] 105.5MB/105.5MB
609e5bdfa7b1: Download complete
4b62383183a1: Download complete
87d5a41c1790: Download complete
8856ade7c27a: Download complete
2897b671832f: Download complete
f6c9c88e767d: Download complete
```

```
adelillah@adelillah-HP-250-G9:~/Documents/INFOA3/NoSQL/NoSQL/tp3$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
69f5409045da        couchdb:latest     "tini -- /docker-ent..."   About a minute ago   Up About a minute   0.0.0.0:5984->5984/tcp, [::]:5984->5984/tcp   couchdb
```

CouchDB fournit une interface web d'administration accessible: http://localhost:5984/_utils



Création de la base de données FILMS et insertions des données depuis le fichier JSON

```
adelillah@adelillah-HP-250-G9:~/Documents/INFOA3/NoSQL/NoSQL/tp3$ curl -u admin:admin -X POST "http://localhost:5984/films/_bulk_docs" -H "Content-Type: application/json" -d @films.json
[{"ok":true,"id":"movie:1","rev":"1-c404285f85cc593f351855821ebe5fc7"}, {"ok":true,"id":"movie:24","rev":"1-7f851a642ab7108ad8354952d4c560"}, {"ok":true,"id":"movie:28","rev":"1-5ef74f3b07d597da5c1a41d73e00f308"}, {"ok":true,"id":"movie:33","rev":"1-210992fbbd105dd91ceb02a1f6b181d"}, {"ok":true,"id":"movie:38","rev":"1-902184f7cc63bc4f802f3b33ffd2eb27"}, {"ok":true,"id":"movie:59","rev":"1-2ca4990b59fbbaee2006e0b0ef74481d0"}, {"ok":true,"id":"movie:62","rev":"1-89d7541cf67625fbeda283dadf7294bb"}, {"ok":true,"id":"movie:74","rev":"1-ea1b40608799bd603ebc7f82cf4511ac"}, {"ok":true,"id":"movie:75","rev":"1-522355c47de05179064d6218542bb6ca7"}, {"ok":true,"id":"movie:77","rev":"1-85291b834cfd40e18739d1b37d6def"}, {"ok":true,"id":"movie:78","rev":"1-c2b126bd26e20d4256ea573e5bc5a11a"}, {"ok":true,"id":"movie:85","rev":"1-ae348bef3600f3a445ed329201ccd191"}, {"ok":true,"id":"movie:87","rev":"1-8981cf7f2d13f2d0004ae8c4a8440"}]
```

Les données sont maintenant visibles sur l'interface visuelle:

The screenshot shows the Apache CouchDB Futon interface. On the left, there is a sidebar with a back arrow, the word "films", and three vertical dots. Below this are sections for "All Documents", "Run A Query with Mango", "Permissions", "Changes", and "Design Documents", each with a plus sign icon. The main area has tabs for "Table" (which is selected), "Metadata", and "JSON". There is also a "Create Document" button. The table view displays 20 documents, each with a checkbox, an icon, an ID, a key, and a truncated JSON value. The columns are labeled "id", "key", and "value". At the bottom, it says "Showing document 1 - 20. Documents per page: 20" with a dropdown and navigation arrows.

	id	key	value
<input type="checkbox"/>	movie:10098	movie:10098	{ "rev": "1-a9ad1a0a8ec461bac3ce06dee5b3e..."}
<input type="checkbox"/>	movie:1018	movie:1018	{ "rev": "1-d092ae6460979fc8d09e01726ff9..."}
<input type="checkbox"/>	movie:10238	movie:10238	{ "rev": "1-04e18ea88d56259144d7b646b144c..."}
<input type="checkbox"/>	movie:103	movie:103	{ "rev": "1-06e8ddd6bb2f56dacacd39cb970e..."}
<input type="checkbox"/>	movie:10362	movie:10362	{ "rev": "1-9045bb46faeb05af6837d75f365e45..."}
<input type="checkbox"/>	movie:103731	movie:103731	{ "rev": "1-087c137274fb02c1844a882a10d3e..."}
<input type="checkbox"/>	movie:106	movie:106	{ "rev": "1-02252f92774e5f478f0c62e89a7558..."}
<input type="checkbox"/>	movie:10669	movie:10669	{ "rev": "1-e55281901af86aac6e1cfb80e4f2ec..."}
<input type="checkbox"/>	movie:10675	movie:10675	{ "rev": "1-c91ae3faa29bb9b1c2739264cd9e0..."}
<input type="checkbox"/>	movie:10835	movie:10835	{ "rev": "1-901623b2514e8abfd996a2fad7b110..."}
<input type="checkbox"/>	movie:10889	movie:10889	{ "rev": "1-132c0754cd4b6e12ee48d1846af02..."}
<input type="checkbox"/>	movie:1091	movie:1091	{ "rev": "1-53739ea423b09b0f44e1c2f4ba1697..."}

Modèle proposé pour le problème:

Aux origines du MapReduce : 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 documents structurés, pour représenter une telle matrice (s'inspirer du cas Page Rank du moteur de recherche Google, vu en cours). 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 $V(v_1, v_2, \dots, v_N)$ est le scalaire $\|V\| = \sqrt{v_1^2 + v_2^2 + \dots + v_N^2}$
3. Nous voulons calculer le produit de la matrice M avec un vecteur de dimension N , $W(w_1, w_2, \dots, w_N)$. 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.

1) Je propose un document json pour chaque case de la matrice:

$M_{12,431}=0.37$ est équivalent à { "i": 12, "j": 431, "value": 0.37 }

2) MapReduce pour la norme de la ligne i

Map

```
function (doc) {
  emit(doc.i, doc.value * doc.value);
}
```

Reduce

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

3) MapReduce pour $\phi_i = \sum M_{ij}W_j$

Map (W accessible via la mémoire)

W est une variable globale, $W = \{431: 0.8, 98: 0.1, \dots\}$

```
function (doc) {
  emit(doc.i, doc.value * W[doc.j]);
}
```

Reduce

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

Exercices MapReduce avec CouchDB:

1 - Nombre total de films

Map:

```
function (doc) {  
    emit("total", 1);  
}
```

Reduce:

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

Ces fonctions sont insérés via l'interface visuelles en créant une vue:

New View

Design Document [?](#) _design/
New document newDesignDoc

Index name [?](#) demo1

Map function [?](#)

```
1 - function (doc) {  
2 |     emit("total", 1);  
3 | }
```

Reduce (optional) [?](#)

CUSTOM

Custom Reduce function

```
1 - function (keys, values, rereduce) {  
2 |     return sum(values);  
3 | }
```

Create Document and then Build Index Cancel

Il faut également cocher reduce sinon seulement le résultat de map est renvoyé:

Le résultat:

key	value
 total	278

2- Nombre de films par genre

Map:

```
function (doc) {  
    emit(doc.genre, 1);  
}
```

Reduce:

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

key	value
Action	36
Adventure	3
Aventure	22
Comédie	25
Comedy	1
Crime	29
Drama	14
Drame	96
Fantastique	4
Fantasy	2
Guerre	1
Histoire	1
Horreur	8

Map Reduce avec mongoDB:

Dans CouchDB, on “crée” MapReduce en enregistrant des fonctions map/reduce dans une view stockée dans la base, puis on interroge cette view via HTTP comme un index réutilisable.

Dans MongoDB, on exécute MapReduce directement sur une collection avec une commande (MapReduce) et le résultat est écrit dans une collection de sortie, sans créer une view persistante de la même manière.

1- Compter le nombre total de films

```
[direct: mongos] mabasefilms> db.films.mapReduce(  
...   function () { emit("total", 1); },  
...   function (key, values) { return Array.sum(values); },  
...   { out: "mr_total_films" }  
... )  
DeprecationWarning: Collection.mapReduce() is deprecated. Use an aggregation instead.  
See https://mongodb.com/docs/manual/core/map-reduce for details.  
{  
  result: 'mr_total_films',  
  ok: 1,  
  '$clusterTime': {  
    clusterTime: Timestamp({ t: 1765912507, i: 61 }),  
    signature: {  
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAA='), 0),  
      keyId: Long('0')  
    }  
  },  
  operationTime: Timestamp({ t: 1765912507, i: 61 })  
}  
[direct: mongos] mabasefilms>  
  
[direct: mongos] mabasefilms> db.mr_total_films.find()  
...  
[ { _id: 'total', value: 1000000 } ]
```

2- Nombre de films par genre

```
[direct: mongos] mabasefilms> db.films.mapReduce(  
...   function () { emit(this.genre, 1); },  
...   function (key, values) { return Array.sum(values); },  
...   { out: "mr_films_par_genre" }  
... )  
... db.mr_films_par_genre.find().sort({ value: -1 })  
...  
[  
  { _id: 'Horreur', value: 167141 },  
  { _id: 'Action', value: 166892 },  
  { _id: 'Aventure', value: 166671 },  
  { _id: 'Drame', value: 166548 },  
  { _id: 'Comédie', value: 166421 },  
  { _id: 'Science-Fiction', value: 166327 }  
]  
[direct: mongos] mabasefilms>
```

3- Nombre de films par réalisateur

```
[direct: mongos] mabasefilms> db.films.mapReduce(  
...   function () { emit(this["réalisateur"], 1); },  
...   function (key, values) { return Array.sum(values); },  
...   { out: "mr_films_par_realisateur" }  
... )  
... db.mr_films_par_realisateur.find().sort({ value: -1 })  
...  
[  
  { _id: 'Francois C.', value: 167059 },  
  { _id: 'Thomas K.', value: 166978 },  
  { _id: 'Pascal U.', value: 166937 },  
  { _id: 'Claude G.', value: 166927 },  
  { _id: 'Nadia Z.', value: 166345 },  
  { _id: 'Samir Y.', value: 165754 }  
]  
[direct: mongos] mabasefilms>
```

4- Nombre d'acteurs uniques

```
db.films.mapReduce(  
  function () {  
    if (!this.acteurs) return;  
    this.acteurs.forEach(a => emit(a, 1));  
  },  
  function (key, values) { return 1; },  
  { out: "mr_acteurs_uniques" }  
)  
db.mr_acteurs_uniques.countDocuments({})
```

5- Nombre de films par année

```
[direct: mongos] mabasefilms> db.films.mapReduce(  
...   function () { emit(this["année"], 1); },  
...   function (key, values) { return Array.sum(values); },  
...   { out: "mr_films_par_annee" }  
... )  
... db.mr_films_par_annee.find().sort({_id: 1 })  
[  
  { _id: 1980, value: 21718 },  
  { _id: 1981, value: 22103 },  
  { _id: 1982, value: 21710 },  
  { _id: 1983, value: 21595 },  
  { _id: 1984, value: 21646 },  
  { _id: 1985, value: 21873 },  
  { _id: 1986, value: 21630 },  
  { _id: 1987, value: 21808 },  
  { _id: 1988, value: 21720 },  
  { _id: 1989, value: 21629 },  
  { _id: 1990, value: 21687 },  
  { _id: 1991, value: 21711 },  
  { _id: 1992, value: 21515 },  
  { _id: 1993, value: 21733 },  
  { _id: 1994, value: 21900 },  
  { _id: 1995, value: 21586 },  
  { _id: 1996, value: 21683 },  
  { _id: 1997, value: 21857 },  
  { _id: 1998, value: 21589 },  
  { _id: 1999, value: 21971 }  
]
```

6- Note moyenne par film

```
db.films.mapReduce(  
  function () {  
    for (var i = 0; i < this.grades.length; i++) {  
      emit(this.title, { sum: this.grades[i].note, count: 1 });  
    }  
  },  
  function (key, values) {  
    var res = { sum: 0, count: 0 };  
    values.forEach(function (v) {  
      res.sum += v.sum;  
      res.count += v.count;  
    });  
    return res;  
  },  
  {  
    out: "mr_moy_film",  
    finalize: function (key, res) {  
      res.avg = res.count ? (res.sum / res.count) : null;  
      return res;  
    }  
  }  
)  
db.mr_moy_film.find().limit(10)
```

10. Nombre de notes strictement supérieures à 70

```
function (doc) {  
    doc.grades.forEach(function (g) { if (g.grade > 70) emit(1, 1); });  
}  
function (keys, values, rereduce) {  
    return sum(values);  
}
```

11- Acteurs par genre (sans doublons)

```
db.films.mapReduce(  
function () {  
    if (!this.actors || !this.genre) return;  
    for (var i = 0; i < this.actors.length; i++) {  
        var a = this.actors[i];  
        var full = a.firstname + " " + a.lastname;  
        emit(this.genre, full);  
    }  
},  
  
function (genre, actors) {  
    var seen = {};  
    var out = [];  
    actors.forEach(function (name) {  
        if (!seen[name]) { seen[name] = true; out.push(name); }  
    });  
    return out;  
},  
{ out: "mr_acteurs_par_genre" }  
)  
db.mr_acteurs_par_genre.find().limit(10)
```

12- Acteurs apparaissant dans le plus grand nombre de films

```
db.films.mapReduce(  
  function () {  
    var movieId = this.id;  
    var seen = {};  
    for (var i = 0; i < this.actors.length; i++) {  
      var a = this.actors[i];  
      var actor = a.firstname + " " + a.lastname;  
      if (!seen[actor]) {  
        emit([actor, movieId], 1);  
        seen[actor] = true;  
      }  
    },  
    function (k, vals) { return 1; },  
    { out: "mr_actor_movie" }  
)
```

J'ai eu un souci avec mon docker, ce qui m'a poussé à le réinstaller et donc à supprimer les données MongoDB déjà importées. Faute de temps, il n'a pas été possible de réimporter l'ensemble des films, de reconfigurer complètement l'environnement et de relancer les tests nécessaires pour tester MapReduce.