Nicolas Travers
Conservatoire National des Arts et Métiers

Introduction

- Hu*mongo*us (monstre / énorme)
- NoSQL : Orienté Documents
  - JSon
  - Documents sérialisés : BSon objects
  - Implémenté en C++
  - Indexation d'attributs (BTree + 2DSphere)
  - Réplication (*Replica Set*) + Distribution (*Sharding*)
  - Stockage : *GridFS*
- Utilisé par :
  - *Doodle, SAP, sourceforge, NY times, bit.ly, github, foursquare, EA games, grooveshark*
  - Licence AGPL (Apache)

Notions

- **Base** de données
  - Commande : > use **maBD** ;
- **Collections** de documents
  - Création : > db.createCollection('***users***');
  - Utilisation : > db.**users**. <commande> ;
    - Commandes : find(), save(), delete(), update(), aggregate(), distinct(), mapReduce()…
    - Equivalent SQL : *FROM*
- **Documents**
  - JSon :

```
{
    "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
    "name": "James Bond", "login": "james", "age": 50,
    "address": {"street": "30 Wellington Square", "city": "London"},
    "job" : ["agent", "MI6"]
}
```

  - Insertion : > db.users.save ( ); //pas de quotes
  - Pas de schéma

Interrogation : Filtrage simple (1/2)

- Requêtes orientées documents
- Commande : > db.users.**find**(<filtre> , <projection>);
- **Filtre** :
  - Motif "JSon" qui doit être contenu dans le document
  - Format "clé/valeur"
  - Peut contenir des valeurs exactes, des opérations, des imbrications, des tableaux
    - Opération : $op (pas de guillemets)
  - Equivalent SQL : *WHERE*
- **Projection** :
  - "Clé/valeur" à retourner (les autres sont supprimées)
  - Equivalent SQL : *SELECT (sans agrégat)*

- Exemple :

> db.users.find({"login" : "james"} , {"name" : 1, "age" : 1});

Interrogation : Filtrage simple (2/2)

- Recherche exacte

```
> db.users.find( { “login” : "james" } , {“name” : 1, “age” : 1});
```

- Dans une imbrication

```
> db.users.find( { “address.city” : "London" } );
```

- Opération

```
> db.users.find( { “age” : { $gt : 40 } } );
//$gt, $gte, $lt, $lte, $ne, $in, $nin, $or, $and, $exists, $type, $size, $cond…
```

- Expression régulière[1]

```
> db.users.find( { ”name” : {$regex : “james”, $options : “i" }} );
```

- Tableau

```
> db.users.find( { ”job” : “MI6”} );          //dans la liste
> db.users.find( { ”job.1” : “MI6”} );        //2° place de la liste
> db.users.find( { ”job” : [“MI6”]} );        //recherche exacte
```

1 - regex : https://docs.mongodb.com/manual/reference/operator/query/regex/

Interrogation : Distinct - Comptage

- Liste des valeurs distinctes d’une clé

```
> db.users.distinct( “name” );
> db.users.distinct( “address.city” );
```

- Compter le nombre de documents

```
> db.users.count();
> db.users.find( { “age” : 50 }).count();
```

Interrogation : pipeline (1/3)

- **aggregate() :** *Séquence* ordonnée d'opérateurs
  pipeline aggregate
- Commande :
  > db.users.**aggregate([{$op1 : {}}, {$op2 : {}}, …])**;
- **Opérateurs** :
  - $match : filtrage simple //équivalent : where
  - $project : Projection de clés //équivalent : select
  - $sort : tri des résultats //équivalent : order by
  - $unwind : normalisation 1NF
  - $group : groupement + fonction agrégation //équivalent : group by + fn
  - $lookup : jointure gauche (depuis 3.2) //équivalent : left outer-join
  - $out : stockage du résultat (depuis 3.2)
  - $geoNear : tri par proximité géographique (lat/long)
  - $redact : élagage conditionnel (documents imbriqués)
  - + $sample, $limit, $skip,

Interrogation : pipeline (2/3)

- ***Pipeline*** : résultat d'une opération sert d'entrée pour la suivante

```
> db.users.aggregate([{$match : {"address.city" :  "London"}},
                      {$project : {"login" : 1, "age" : 1}},
                      {$sort : {"age" : 1, "login" : -1}}
    ]);
```

- **$unwind**
  - Créer un document pour chaque instance

```
> db.users.aggregate([ {$unwind :  "$job"} ]);
```

```
{
    "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
    "name": "James Bond", "login": "james", "age": 50,
    "address": {"street": "30 Wellington Square", "city": "London"},
     "job" : ["agent", "MI6"]
}
```

```
{
  "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
  "name": "James Bond", "login": "james", "age": 50,
  "address": {"street": "30 Wellington Square", "city": "London"},
  "job" : "agent"
}
```

```
{
  "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
  "name": "James Bond", "login": "james", "age": 50,
  "address": {"street": "30 Wellington Square", "city": "London"},
  "job" : "MI6"
}
```

Interrogation : pipeline (3/3)

- **$group :** clé (**_id**) + agrégat ($sum / $avg / …)
  Pas de groupement : *valeur unique*
  > db.users.aggregate([{$group : {"_id" : **"age"**, "res": {$sum : 1}}}]);
  Groupement par valeur : ***$clé***
  > db.users.aggregate([{$group:{"_id" : "**$age**", "res": {$sum : 1}}}]);
  Moyenne des valeurs : ***$clé***
  > db.users.aggregate([{$group:{"_id":"$address.city", "moy": {$avg: "**$age**"}}}]);
- **Exemple** de séquence

```
> db.users.aggregate([
        {$match:   {"address.city" : "London"} },
        {$unwind : "$job" },
        {$group :   {"_id" : "$job", "moy": {$avg: "$age"}} },
        {$match :  {"moy" : {$gt : 30}} },
        {$sort :     { "moy" : -1} } ]);
```


Interrogation : Map/Reduce (1/2)

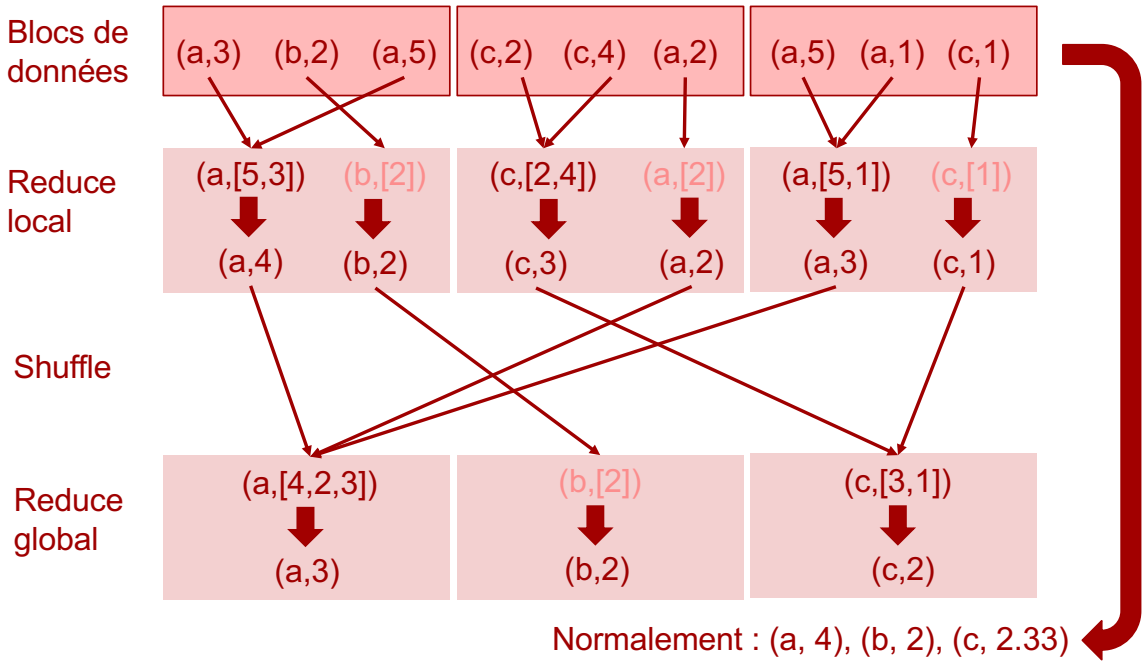
Langage : Javascript

```
var mapFunction = function () {
        if( this.age > 30 && this.job.contains("MI6") )
                emit (this.address.city, this.age);
} //emit : clé, valeur, this => document en cours

var reduceFunction = function (key, values) {
        return Array.sum(values);
} //emit : retourne une valeur pour la liste "values"

var queryParam = {query : {}, out : "result_set"}
        //query : Permet de filtrer les documents AVANT le map, $match
        //out : collection de stockage du résultat

db.users.mapReduce (mapFunction, reduceFunction, queryParam);
db.result_set.find();
        //Consulter le résutlat
```

Interrogation : Map/Reduce (2/2)

- **Map**
  - Plusieurs « *emit* » possibles (clé/valeur)
  - Exécuté sur chaque document
    - sauf si : index + queryParam
- **Reduce**
  - Optimisation : calcul local et global
  - Pas de reduce global si un seul doc dans « values »
  - ➢ Production de résultat non homogène
    - Calcul non associatif (moyenne, liste, etc.)
    - Structurer sortie du Map et Reduce
- **Shuffle**
  - Dépend de la clé du « emit »
  - Optimisé si identique au sharding

Problème de Reduce : moyenne (1/2)



Problème de Reduce : moyenne (2/2)

- Calcul de la moyenne non associatif
  - Transformer Map et Reduce en opérations associatives
    - Avg = sum / nb
      - ➢ Produire et agréger la somme et le nombre
      - ➢ {"moy" : sum/nb, "sum" : sum, "nb" : nb}
    - Schéma du map et du reduce identique

Indexation

- **Sharding** par défaut
  - BTree non dense sur « _id »
  - Possibilité de changer la clé
    - créer un index unique

      > db.users.createIndex ({ "login" : 1 }, { "unique" : true }) ;
- Sharding par **hachage**

  > db.users.createIndex({ "_id" : "hashed" }) ;
  - Hachage cohérent : DHT
- Ajout de **BTree**

  > db.users.createIndex({"age":1}) ;
  - Requêtes sur « age » plus rapide
    - Y compris Map/Reduce si intégré à « queryParam »
    - Consultation grâce à « .explain() »
  - Pas de combinaisons d'indexes

Indexation : 2DSphere

Possibilité de faire des requêtes géolocalisées

```
> db.users.ensureIndex( { "address.location" : "2dsphere" } );
```

- Formater la localisation :

```
"location":{"type": "Point", "coordinates" : [51.489220, -0.162866]}
```

- Requêtes sphériques

```
var near = {$near: {$geometry:{"type":"Point","coordinates":[51.489220,-0.162866]},
                    $maxDistance : 10000}};
db.users.find( {"address.location":near}, {"name":1, "_id":0});
```

- Opérateur $geoNear (aggregate)

```
var geoNear = {"near":{"type":"Point", "coordinates" : [51.489220,-0.162866]},
      "maxDistance":10000, "distanceField" : "outputDistance", "spherical":true};
db.users.aggregate([ {"$geoNear": geoNear} ]);
```

- Requêtes « polygonales »

```
var polygon = {$geoWithin : {$geometry : {"type" : "Polygon", "coordinates" : [ [P1], [P2], [P3] ] }}}
```

http://docs.mongodb.org/manual/tutorial/query-a-2d-index/

Mises à jour

- Pas de transactions
- Modifications atomiques de documents
  - $set – Modifie une valeur
  - $unset – Supprime un attribut
  - $inc – Incrément
  - $push – Ajout dans un tableau
  - $pushAll – Plusieurs valeurs dans un tableau
  - $pull – Supprimer une valeur de tableau
  - $pullAll – Supprimer plusieurs valeurs

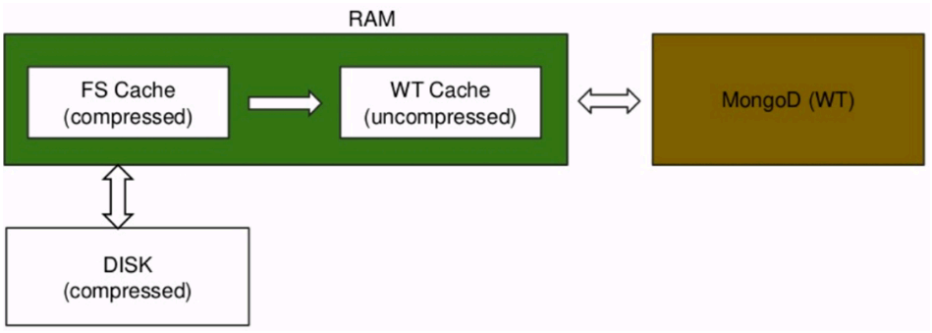
```
> db.users.update (
      { "_id" : ObjectId("4efa8d2b7d284dad101e4bc7") },
      { "$inc" : { "age" : 1 } } );
```

Moteurs de stockage

- **Wired Tiger**
  - Par défaut depuis 3.2
  - Créé pour BerkeleyDB et utilisé dans Oracle NoSQL
  - Concurrence : contrôleur multi-version
- MMAPV1
  - Original
  - Mapping en mémoire des pages
  - Verrouillage au niveau de la collection (écriture lente)
- Encrypted storage
  - Depuis 3.2
  - Basé sur Wired Tiger
  - KMIP (Key Management Interoperability Protocol)
- In Memory
  - Depuis 3.2.6
  - Basé sur Wired Tiger
  - Doit tenir en mémoire (erreur : *WT_CACHE_FULL*)

Wired Tiger – Cache system

- 50% or RAM – 1go
  (ou 256Mo)
- Compression des données en RAM



Moteurs de stockage

	MongoDB WiredTiger	MongoDB In-Memory	MongoDB Encrypted	MongoDB MMAPv1
Write Performance	Excellent Document-Level Concurrency Control	Excellent Document-Level Concurrency Control	Good Document-Level Concurrency Control	Good Collection-Level Concurrency Control
Read Performance	Excellent	Excellent	Good	Good
Predictable Latency (99%)	Good	Excellent	Good	Good
Disk Compression Support	Yes	N/A	Yes	No
MongoDB Query Language Support	Yes	Yes	Yes	Yes
Secondary Index Support	Yes	Yes	Yes	Yes
Replication Support	Yes	Yes	Yes	Yes
Sharding Support	Yes	Yes	Yes	Yes
Ops Manager & Cloud Manager Support	Yes	Yes	Yes	Yes
Native Encryption-At-Rest	No	N/A	Yes	No
Read Concern	Yes	Yes	Yes	No
Security Controls	Yes	Yes	Yes	Yes
Larger than RAM Datasets	Yes	No	Yes	Yes
Platform Availability	Linux, Windows, Mac OS X	Linux, Windows, Mac OS X	Linux, Windows, Mac OS X	Linux, Windows, Mac OS X, Solaris (x86)

Drivers / API

- Drivers : http://docs.mongodb.org/ecosystem/drivers/
  - Python, Ruby, Java, Javascript (Node.js), C++, C#, PHP, Perl, Scala…
  - Syntaxe : http://docs.mongodb.org/ecosystem/drivers/syntax-table/

Driver Java

Connexion

```
ServerAddress server = new
ServerAddress ("localhost", 27017);
Mongo mongo = new Mongo(server);
```

Sélection BD et Collection

```
DB db = mongoClient.getDB( "maBD" );
DBCollection users =
db.getCollection("users");
```

Authentification

```
db.authenticate(login,
passwd.toCharArray());
```

Insertion d'un document (DBObject)

```
DBObject doc = new
BasicDBObject("name", "MongoDB")
 .append("type", "database")
 .append("count", 1)
 .append("info", new BasicDBObject("x",
        203).append("y", 102));
// ou doc = (DBObject) JSON.parse(jsonText) ;
users.insert(doc);
```

Interroger (Document requête + curseur)

```
BasicDBObject query = new
      BasicDBObject("name", "James Bond");
DBCursor cursor = coll.find(query);
try {
  while(cursor.hasNext()) {
       System.out.println(cursor.next());
    }
} finally {
    cursor.close();
}
```

Map/Reduce :

```
MapReduceCommand cmd =
      new MapReduceCommand("users", map,
                reduce, "outputColl",
      MapReduceCommand.
      OutputType.REPLACE, query);
MapReduceOutput out = users.mapReduce(cmd);
for (DBObject o : out.results()) {
      System.out.println(o.toString());
}
//outputType : INLINE, REPLACE, MERGE, REDUCE
```

http://api.mongodb.org/java/current/index.html?_ga=1.177444515.761372013.1398850293

Driver C#

Références/Librairies

```
MongoDB.Bson.dll
      using MongoDB.Bson;
MongoDB.Driver.dll
      using MongoDB.Driver;
```

Connexion

```
var connectionString =
"mongodb://localhost";
var client = new
MongoClient(connectionString);
var server = client.GetServer();
```

Base de données & Collection

```
var db = server.GetDatabase("maBD");
var coll = db.GetCollection<User>("users");
```

Objet « User » à définir

```
public class User{
   public ObjectId Id { get; set; }
   public string name { get; set; }
   public string login { get; set; }
   public int age { get; set; }
   public Address address { get; set; }
}
```

Récupérer un document :

```
var query=Query<User>.EQ(e=>e.login,"james");
var entity = coll.FindOne(query);
```

Résultat de requête : LINQ

```
using MongoDB.Driver.Linq;
```

Requête :

```
var query = coll.AsQueryable<User>()
                .Where(e => e.age > 40)
                .OrderBy(c => c.name);
```

Résultat :

```
foreach (var user in query)
{
   // traitement
}
```

MapReduce :

```
var mr = coll.MapReduce(map, reduce);
foreach (var document in mr.GetResults()) {
   Console.WriteLine(document.ToJson());
}
```

http://www.nuget.org/packages/mongocsharpdriver/

Driver Python

Importation

```
>>> import pymongo
```

Connexion

```
>>> from pymongo import MongoClient
>>> client = MongoClient()
>>> client = MongoClient('localhost', 27017)
```

Base de données & collection

```
>>> db = client.maBD
>>> coll = db.users
```

Récupérer un document

```
coll.find_one ( { "login" : "james"} )
```

Requête

```
>>> for c in coll.find ( {"age": 40} ) :
...       pprint.pprint (c)
```

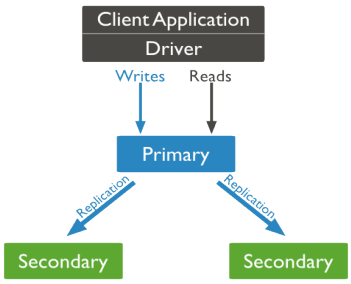
MapReduce

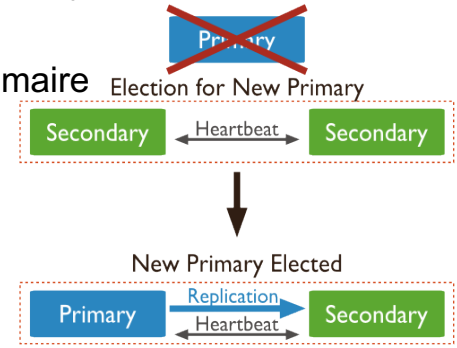
```
>>> from bson.code import Code
>>> map = Code("function () {"
... " this.tags.forEach(function(z) {"
... " emit(z, 1);"
... " });"
... "}")
>>> reduce = Code("function (key, values) {"
... " var total = 0;"
... " for (var i = 0; i < values.length; i++) {"
... " total += values[i];}"
... " return total; } ")
>>> result = coll.map_reduce(map, reduce,
          "myresults")
>>> for doc in myresults.find():
...       print doc
```


Replica Set

- Réplication d'un serveur
  - Asynchrone
    - Primary server : écritures
    - Secondary servers : lectures

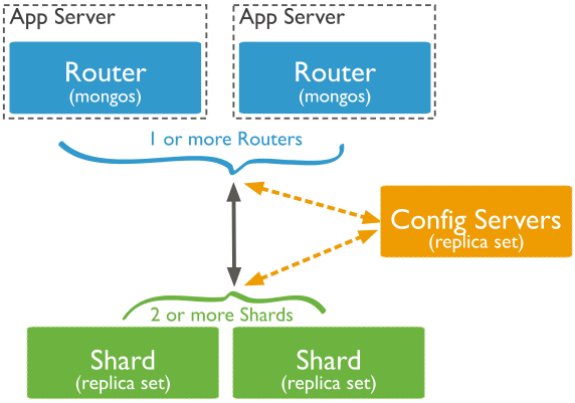
      Mise à jour via oPlog (fichier de log)
  - Distribue la charge de lecture (requêtes)
  - Tolérance aux pannes
    - Élection d'un nouveau serveur primaire
    - Besoin d'un serveur arbitre



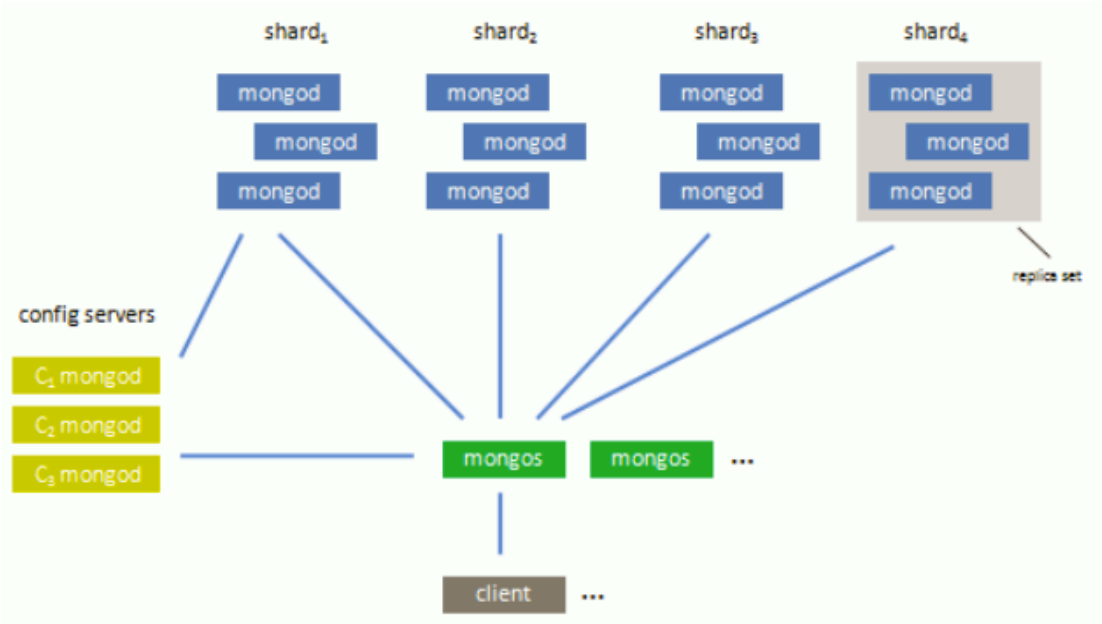


Sharding

- Distribution sur un cluster de machines
  - Equilibrage de charge des données
  - Combinaison avec *Replica Set*
  - Nécessite :
    - *Config Servers*
    - *Mongos* (routeur)
  - Partitionnement sur attribut
    - Ranged-based
    - Hash-based
    - Par zones



Sharding & Replica Sets



Instructions, cf TP : http://chewbii.com/tp-mongodb-replication-et-sharding/

Outils MongoDB

- Répertoire bin/
  - *mongo* : shell pour exécuter les commandes
  - *mongostat* : Status du process en cours (inserts, updates, queries, commands, memory…)
  - *mongotop* : Temps réparti en lecture vs écriture
  - *mongod* : Lancement du serveur (daemon)
  - *mongos* : Service de routage (sharding)
- **Robomongo**
  - *Interface graphique pour exécution*
  - https://robomongo.org

A savoir

- Utiliser la version 64 bits (32 bits limité à 2Go)
- Pas d'authentification par défaut
- 1 document = 16Mo maximum
- Port par défaut : 27017