

# Projet pour validation de l'UE NFE204

de Xuan-Long Pham  
CNAM 2015-2016



## Table des matières

<b>1</b>	<b>Pratique et utilisation de MongoDB .....</b>	<b>3</b>
1.1	Rappel des objectifs.....	3
1.2	Source et récupération des données.....	3
1.3	Importation .....	3
1.3.1	Environnement de MongoDB .....	3
1.3.2	Importation des données.....	4
1.4	Simulation des pannes .....	5
1.4.1	Panne du maître.....	5
1.4.2	Panne d'un esclave.....	7
1.5	Traitement de MapReduce.....	8
1.5.1	MapReduce simple .....	8
1.5.2	MapReduce avec Jointure.....	9
<b>2</b>	<b>Etude des mécanismes de gestion de la cohérence des données de Cassandra .....</b>	<b>11</b>
2.1	Objectif et approche.....	11
2.2	Description générale .....	11
2.2.1	Principales caractéristiques .....	11
2.2.2	Principales fonctions de l'architecture interne .....	11
2.2.3	Partitionnement et réplication des données .....	12
2.3	Gestion de la cohérence des données .....	14
2.3.1	Définition de la cohérence .....	14
2.3.2	Lecture et écriture .....	14
2.3.3	Suppression des données.....	19
2.3.4	Communication inter-nœuds .....	20
2.3.5	Conclusion.....	22

## Liste des figures

Figure 1 – principales caractéristiques de Cassandra.....	11
Figure 2 – principales fonctions de Cassandra. ....	12
Figure 3 – anneau de hachage. ....	13
Figure 4 – nœuds virtuels et physiques. ....	13
Figure 5 – déroulement d'une écriture. ....	15
Figure 6 – déroulement d'une lecture. ....	18
Figure 7 – paramètres du niveau de cohérence. ....	18
Figure 8 – comparaison d'arbres de Merkle.....	20
Figure 9 – échange GOSSIP.....	21
Figure 10 – exemple de DigestMessage. ....	21

Je tiens à adresser mes remerciements à Messieurs Philippe Rigaux et Raphaël Fournier- S'niehotta pour leur clarté et leur pédagogie lors des séances de cours et de TD de l'UE NFE204.

Leurs aides sont précieuses pour ma reconversion en cours au métier de Data Miner.

# 1 Pratique et utilisation de MongoDB

## 1.1 Rappel des objectifs

Ce projet propose d'effectuer une analyse sur le pourcentage des inscriptions des étudiants étrangers à certains établissements de l'enseignement supérieur de France.

L'objectif est de reproduire un cycle de traitement de données complet, depuis la recherche des sources jusqu'à des analyses de type MapReduce.

## 1.2 Source et récupération des données

Le jeu de données est nommé « Effectifs d'étudiants inscrits dans les établissements publics sous tutelle du ministère en charge de l'Enseignement supérieur et de la Recherche ». Il contient 72 variables sur les établissements universitaires, les étudiants et les formations entre 2006 et 2014.

Il est mis à disposition par le ministère de l'Education Nationale, de l'Enseignement Supérieur et de la Recherche en mode Licence Ouverte d'Open Data à l'adresse suivante : <http://data.enseignementsup-recherche.gouv.fr/explore/dataset/fr-esr-sise-effectifs-d-etudiants-inscrits-esr-public/informations/?sort=-rentree>

Il a été envisagé initialement d'importer la totalité des inscriptions dans MongoDB. Mais il s'avère que l'ordinateur (portable) qui fait les calculs n'a pas assez de disque pour supporter les 3 instances de MongoDB en répliquant de données. Les manipulations se limitent donc aux 2703 inscriptions de l'année universitaire 2014-2015 de l'Institut Nationale Polytechnique de Toulouse et des Ecoles normales supérieures de Paris, Lyon, Rennes et Cachan.

Des tests ont été faits d'aspirer les données par API. Mais devant des instabilités, il a été décidé de les télécharger au format JSON. Les fichiers sont visibles aux URL <http://1drv.ms/1QKlzdD> pour l'INP Toulouse et <http://1drv.ms/1Lwq4Oc> pour les ENS.

Ces fichiers sont « propres » et n'ont pas besoin d'aucun nettoyage ni de complétion particulière.

## 1.3 Importation

### 1.3.1 Environnement de MongoDB

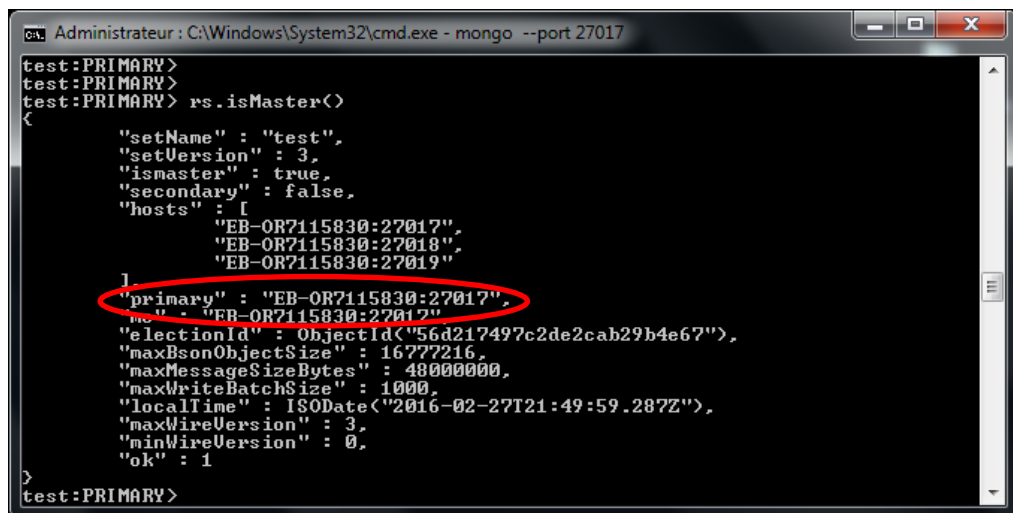
Un réplica set de 3 nœuds est mis en œuvre. Il y a donc 3 copies de chaque donnée.

Tous les trois instances tournent sur un même PC portable sous Windows 7 64 bits, avec 1 CPU quadricore Intel i5-3340@2,7Ghz, 4 Go de RAM et 10 Go disponibles un disque SSD.

Les 3 instances de MongoDB écoutent sur les ports TCP 27017, 27018 et 27019.

C'est la version 3.0.7 64bits de MongoDB qui est utilisée, accompagnée de Robomongo 0.8.5 pour faciliter les manipulations.

La figure suivante présente la sortie de la commande rs.isMaster (Le nœud 2017 y est le maître).



```

Administrator : C:\Windows\System32\cmd.exe - mongo --port 27017
test:PRIMARY>
test:PRIMARY>
test:PRIMARY> rs.isMaster()
{
  "setName" : "test",
  "setVersion" : 3,
  "ismaster" : true,
  "secondary" : false,
  "hosts" : [
    "EB-OR7115830:27017",
    "EB-OR7115830:27018",
    "EB-OR7115830:27019"
  ],
  "primary" : "EB-OR7115830:27017",
  "me" : "EB-OR7115830:27017",
  "electionId" : ObjectId("56d217497c2de2cab29b4e67"),
  "maxBsonObjectSize" : 16777216,
  "maxMessageSizeBytes" : 48000000,
  "maxWriteBatchSize" : 1000,
  "localTime" : ISODate("2016-02-27T21:49:59.287Z"),
  "maxWireVersion" : 3,
  "minWireVersion" : 0,
  "ok" : 1
}
test:PRIMARY>

```

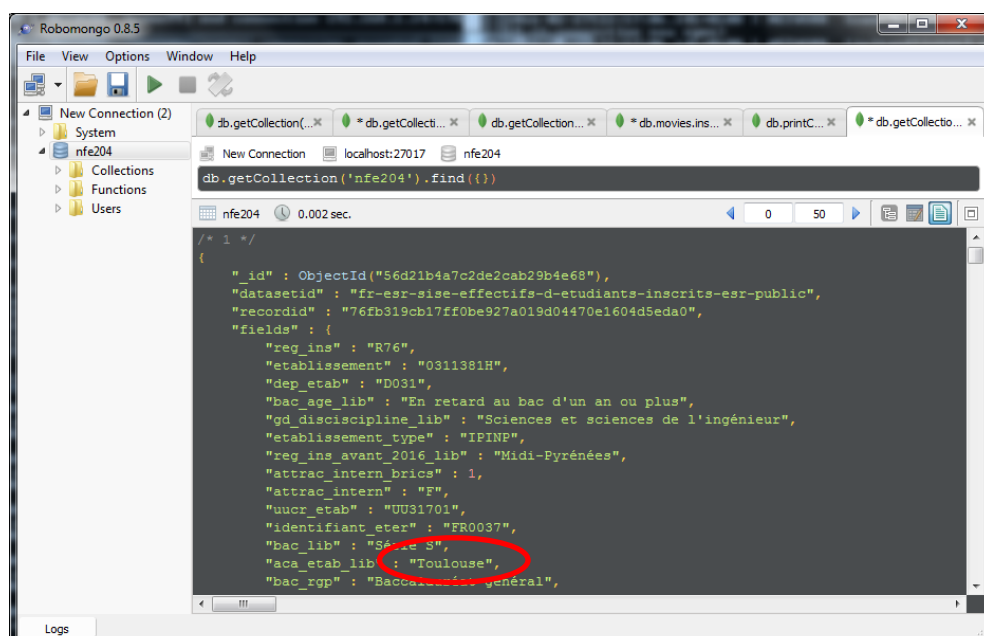
### 1.3.2 Importation des données

Une base de données nommée NFE204 et une collection également nommée NFE204 sont créées.

Les données y sont importées à travers le serveur MongoDB maître et à l'aide de la commande Mongoimport.

Il y a 2 imports successifs dans la collection NFE204, afin de simuler l'initialisation suivie d'une mise à jour.

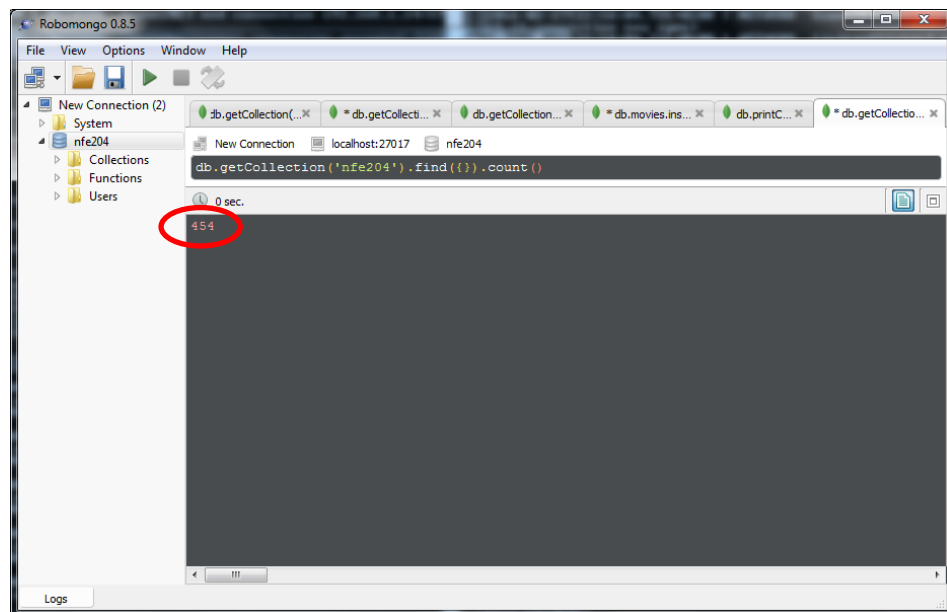
Voici les copies d'écran de l'initialisation avec les 454 documents de l'INP de Toulouse :



```

Robomongo 0.8.5
File View Options Window Help
New Connection (2)
System
nfe204
Collections
Functions
Users
db.getCollection(...)
db.getCollection(...)
db.getCollection(...)
db.movies.ins...
db.printC...
db.getCollection...
New Connection localhost:27017 nfe204
db.getCollection('nfe204').find({})
nfe204 0.002 sec.
/* 1 */
{
  "_id" : ObjectId("56d21b4a7c2de2cab29b4e68"),
  "datasetid" : "fr-esr-sise-effectifs-d-etudiants-inscrits-esr-public",
  "recordid" : "76fb319cb17ff0be927a019d04470e1604d5eda0",
  "fields" : {
    "reg_ins" : "R76",
    "etablissement" : "0311381H",
    "dep_etab" : "D031",
    "bac_age_lib" : "En retard au bac d'un an ou plus",
    "gd discipline_lib" : "Sciences et sciences de l'ingénieur",
    "etablissement_type" : "IPINP",
    "reg_ins_avant_2016_lib" : "Midi-Pyrénées",
    "attrac_intern_brics" : 1,
    "attrac_intern" : "F",
    "uucr_etab" : "UU31701",
    "identifiant_eter" : "FR0037",
    "bac_lib" : "Série S",
    "aca_etab_lib" : "Toulouse",
    "bac_rgp" : "Baccalauréat général",
  }
}

```



Des simulations de pannes sont faites avec ces premières données (voir le chapitre suivant). Une fois les tests de panne passés, un deuxième import ajoute les 2250 inscriptions issues des ENS de Paris, Lyon, Rennes et Cachan. Ce qui fait 2704 documents au total.

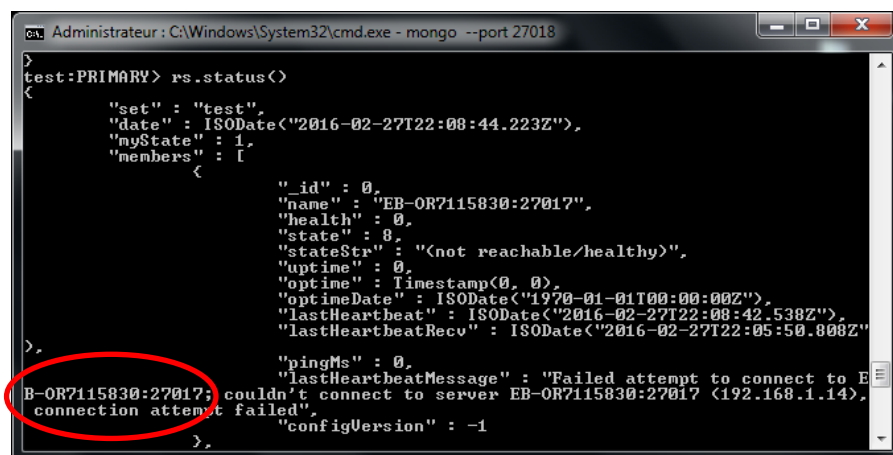
Il est à noter que c'est préférable de désigner les identifiants `_id` avant l'import dans MongoDB, afin d'éviter les problèmes de doublons. En effet, par défaut, MongoDB ajoute un champ `_id` et à tout nouveau document et ne détecte pas les doublons.

## 1.4 Simulation des pannes

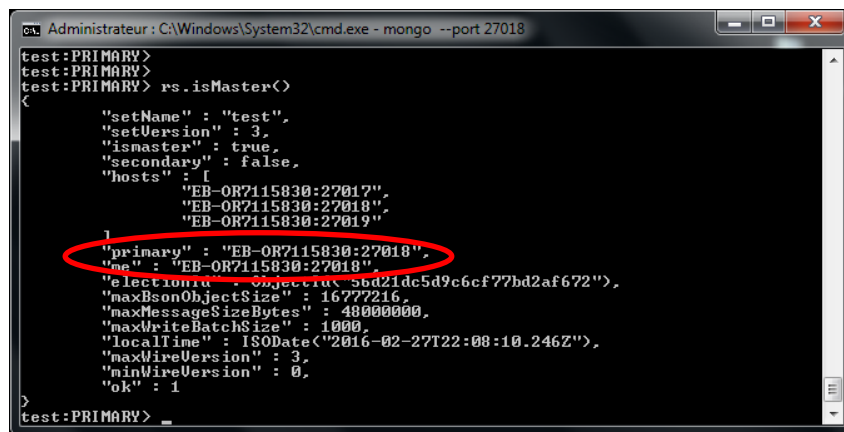
Des pannes ont été simulées afin de vérifier les comportements de MongoDB.

### 1.4.1 Panne du maître

Le maître 27017 est arrêté en douceur par `db.shutdownServer()`. C'est bien détecté par le cluster :



Et Le nœud 2018 reprend effectivement le rôle du maître :

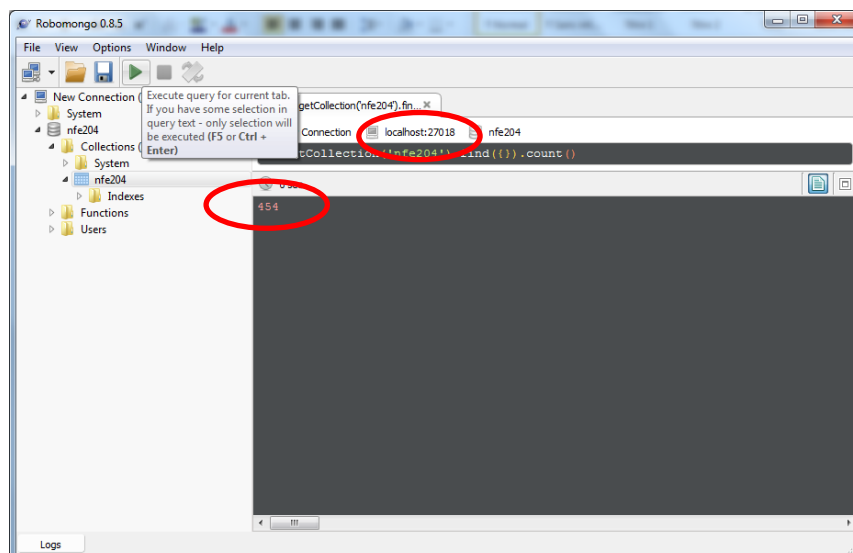


```

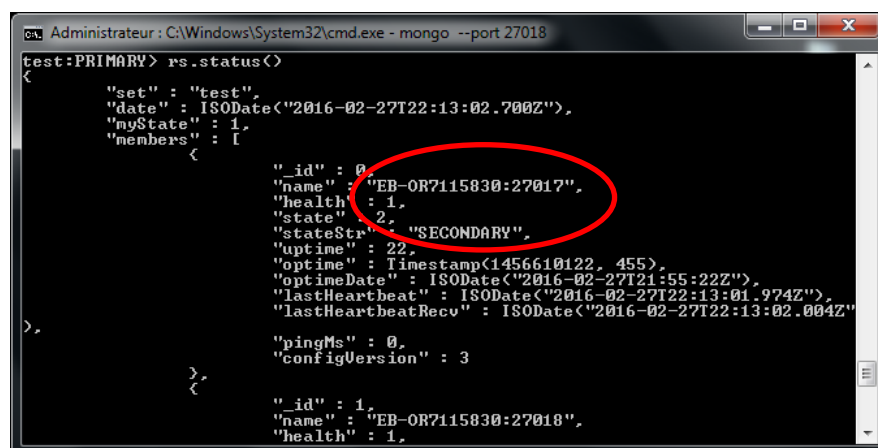
C:\Windows\System32\cmd.exe - mongo --port 27018
test:PRIMARY>
test:PRIMARY>
test:PRIMARY> rs.isMaster()
{
  "setName" : "test",
  "setVersion" : 3,
  "ismaster" : true,
  "secondary" : false,
  "hosts" : [
    "EB-OR7115830:27017",
    "EB-OR7115830:27018",
    "EB-OR7115830:27019"
  ],
  "primary" : "EB-OR7115830:27018",
  "me" : "EB-OR7115830:27018",
  "electionId" : ObjectId("5bd21dc5d9c6cf77bd2af672"),
  "maxBsonObjectSize" : 16777216,
  "maxMessageSizeBytes" : 48000000,
  "maxWriteBatchSize" : 1000,
  "localTime" : ISODate("2016-02-27T22:00:10.246Z"),
  "maxWireVersion" : 3,
  "minWireVersion" : 0,
  "ok" : 1
}
test:PRIMARY>

```

En se reconnectant au nouveau maître, le nœud 27018, Robomongo peut retrouver les données existantes :



Après redémarrage, l'ancien maître nœud 2017 a pu rejoindre le cluster mais en tant qu'esclave :



```

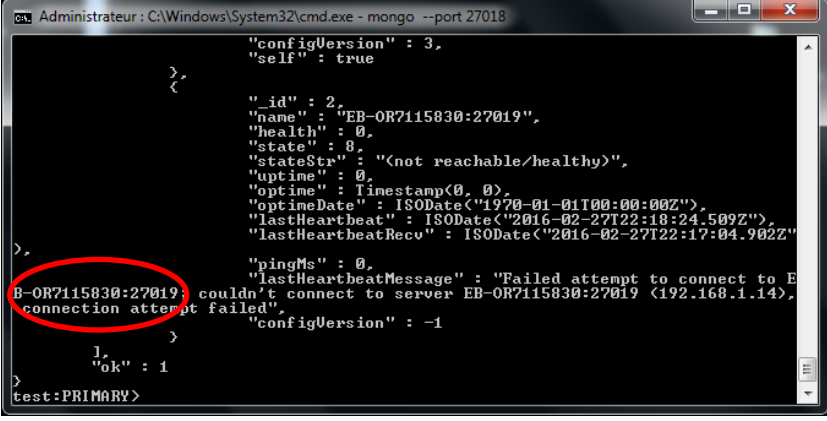
C:\Windows\System32\cmd.exe - mongo --port 27018
test:PRIMARY> rs.status()
{
  "set" : "test",
  "date" : ISODate("2016-02-27T22:13:02.700Z"),
  "myState" : 1,
  "members" : [
    {
      "_id" : 0,
      "name" : "EB-OR7115830:27017",
      "health" : 1,
      "state" : 2,
      "stateStr" : "SECONDARY",
      "uptime" : 22,
      "optime" : Timestamp(1456610122, 455),
      "optimeDate" : ISODate("2016-02-27T21:55:22Z"),
      "lastHeartbeat" : ISODate("2016-02-27T22:13:01.974Z"),
      "lastHeartbeatRecv" : ISODate("2016-02-27T22:13:02.004Z"),
      "pingMs" : 0,
      "configVersion" : 3
    },
    {
      "_id" : 1,
      "name" : "EB-OR7115830:27018",
      "health" : 1,

```



### 1.4.2 Panne d'un esclave

Le nœud esclave 27019 est arrêté brutalement par un signal Ctl-C. C'est bien détecté par le cluster :

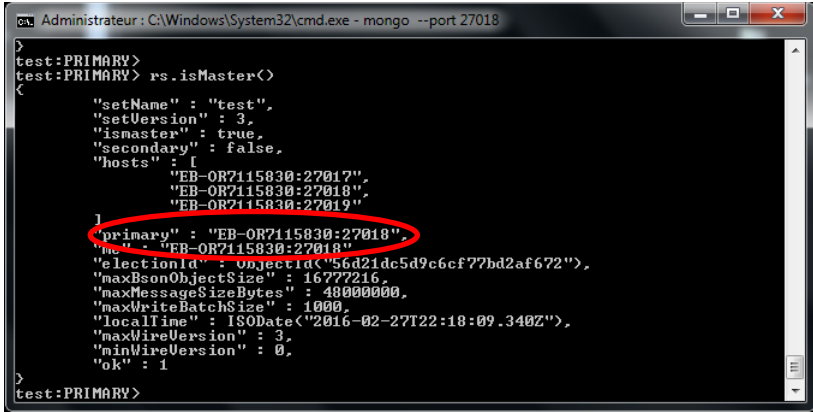


```

Administrator: C:\Windows\System32\cmd.exe - mongo --port 27018
>
{
  "configVersion" : 3,
  "self" : true,
  "_id" : 2,
  "name" : "EB-OR7115830:27019",
  "health" : 0,
  "state" : 8,
  "stateStr" : "<not reachable/healthy>",
  "uptime" : 0,
  "optime" : Timestamp(0, 0),
  "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
  "lastHeartbeat" : ISODate("2016-02-27T22:18:24.509Z"),
  "lastHeartbeatRecv" : ISODate("2016-02-27T22:17:04.902Z"),
  "pingMs" : 0,
  "lastHeartbeatMessage" : "Failed attempt to connect to EB-OR7115830:27019 <192.168.1.14>:27019, couldn't connect to server EB-OR7115830:27019 <192.168.1.14>:27019, connection attempt failed",
  "configVersion" : -1
}
>
1
"ok" : 1
test:PRIMARY>

```

Le nœud 27018 reste maître :

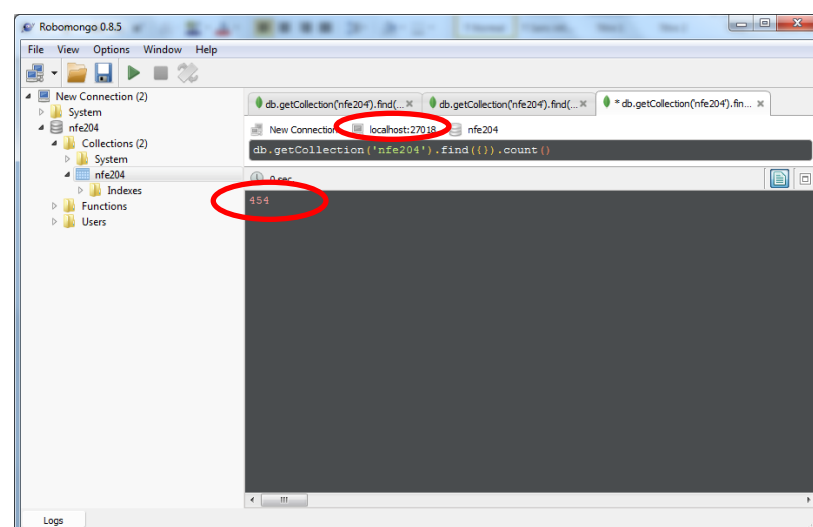


```

Administrator: C:\Windows\System32\cmd.exe - mongo --port 27018
>
test:PRIMARY>
test:PRIMARY> rs.isMaster()
{
  "setName" : "test",
  "setVersion" : 3,
  "ismaster" : true,
  "secondary" : false,
  "hosts" : [
    "EB-OR7115830:27017",
    "EB-OR7115830:27018",
    "EB-OR7115830:27019"
  ],
  "primary" : "EB-OR7115830:27018",
  "me" : "EB-OR7115830:27018",
  "electionId" : ObjectId("56d21dc5d9c6cf77bd2af672"),
  "maxBsonObjectSize" : 16777216,
  "maxMessageSizeBytes" : 48000000,
  "maxWriteBatchSize" : 1000,
  "localTime" : ISODate("2016-02-27T22:18:09.340Z"),
  "maxWireVersion" : 3,
  "minWireVersion" : 0,
  "ok" : 1
}
test:PRIMARY>

```

Et les données du maître restent bien entendu accessibles :



## 1.5 Traitement de MapReduce

### 1.5.1 MapReduce simple

Le cas de MapReduce simple consiste à calculer le pourcentage des étudiants étrangers de l'année 2014-2015 à l'INP de Toulouse et aux ENS.

Les principes sont les suivantes :

- Map envoie des paires intermédiaires composées du code de l'établissement et l'origine de l'inscription, « E » pour Etranger et « F » pour France.
- Reduce reçoit la liste des E et F, fait le comptage et calcule le pourcentage.

La mise en œuvre de Map n'appelle pas de commentaire particulier.

Celle de Reduce est plus complexe. En effet, la fonction Shuffle de MongoDB n'envoie pas les paires intermédiaires à Reduce en une seule fois comme c'est attendu. Elle les envoie en fait en plusieurs séquences. Et une séquence est divisée elle-même de plusieurs envois :

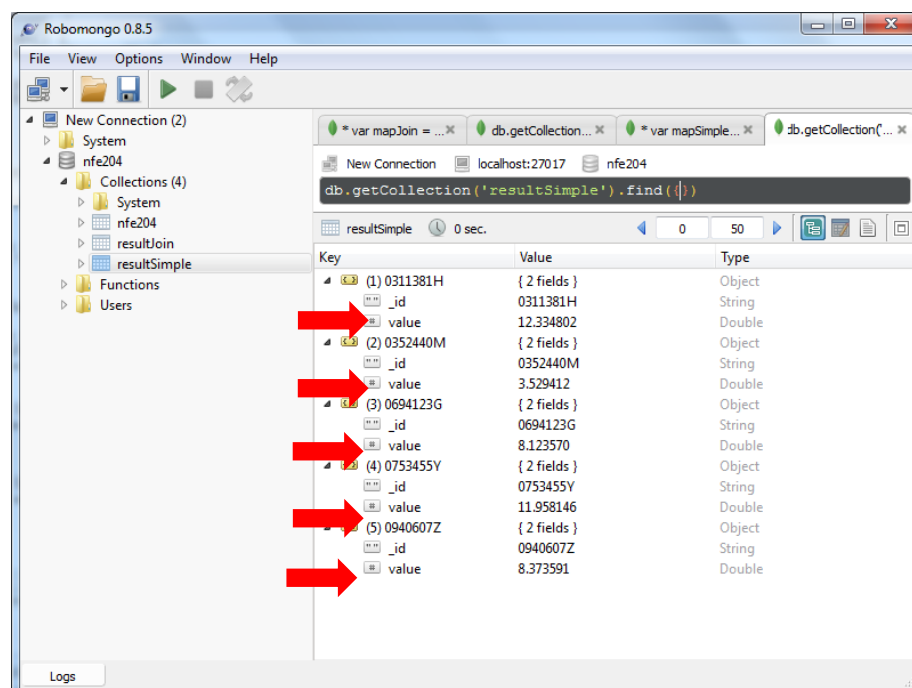
- Un envoi au début de la séquence qui ne se distingue guère d'un envoi « classique » en une seule fois.
- Cet envoi de début de séquence peut être suivi ou pas de plusieurs autres envois qui sont des concaténations des données issues du return du Reduce précédent et des paires intermédiaires de Map.

Les séquences sont suivies d'un envoi final qui est la concaténation des résultats de leur traitement par Reduce.

La complexité supplémentaire vient du fait qu'il faut prendre en considération les séquences et la concaténation finale alors qu'il y a pas d'outil de traçage pour découvrir, comprendre et s'adapter aux comportements du Framework MapReduce de MongoDB en mode distribué.

La fonction Finalize est utilisée pour simplifier le formatage des résultats finaux

Les scripts de Map, Reduce et Finalize sont visibles à l'URL <http://1drv.ms/1TQaiA7>. Les résultats sont concluant :



Les champs « values » donnent le pourcentage des inscriptions d'étudiant étranger par établissement.

La correspondance entre le champ « \_id » qui code les établissements et leur nom est la suivante : 0311381H pour Institut national polytechnique de Toulouse, 0352440M pour l'École normale supérieure de Rennes, 0694123G pour l'École normale supérieure de Lyon, 0753455Y pour l'École normale supérieure de Paris et 0940607Z pour l'École normale supérieure de Cachan.

Dans le chapitre suivant, des noms complets seront ajoutés pour plus de lisibilité, à l'aide d'un MapReduce avec jointure.

### 1.5.2 MapReduce avec Jointure

Le but est d'afficher le nom des établissements au même temps que le pourcentage.

Pour ce faire, 5 nouveaux documents qui associent le code des établissements à leur nom sont créés manuellement et importés dans la collection NFE204. Ils sont typés pour se distinguer des inscriptions dans la même collection NFE204. Ils sont visibles à cette URL <http://1drv.ms/1oOyTs6>.

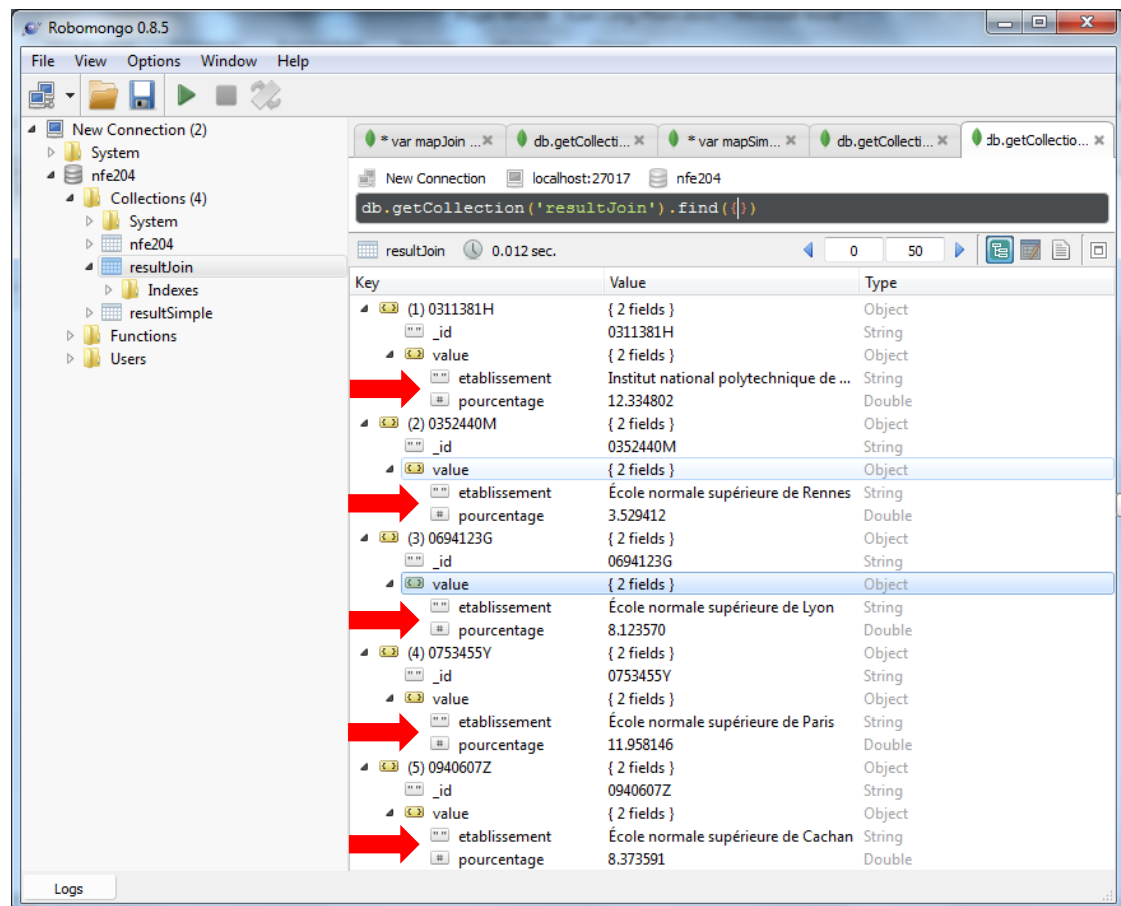
Les principes de la jointure sont les suivants :

- MAP envoie 2 types de paire intermédiaire, l'un contenant le nom des établissements et l'autre l'origine des inscriptions. La clé des 2 types est la même. C'est le code des établissements.
- Reduce reconstitue la jointure à l'aide de cette clé et peut ainsi associer le pourcentage des inscriptions d'étudiant étranger au nom des établissements.

Il faut bien entendu gérer le fait que Reduce ne reçoit pas en une fois toutes les paires intermédiaires pour une clé donnée. L'astuce consiste à faire de telle sorte que Reduce retourne la même structure {clé, liste de contenus} que ce qu'il reçoit en entrée de la part de Shuffle.

Les scripts de Map, Reduce et Finalize sont visibles à l'URL <http://1drv.ms/1L57n47>.

Les résultats sont concluant :



## 2 Etude des mécanismes de gestion de la cohérence des données de Cassandra

### 2.1 Objectif et approche

Cassandra est un système de gestion de données de type NoSQL. Elle est historiquement développée par Facebook à partir des travaux de Google sur BigTable et d'Amazon sur Dynamo, avant d'être reversée à la fondation Apache pour le monde d'Open Source.

Ce chapitre a pour but d'étudier les mécanismes mis en œuvre pour gérer la cohérence de données. Il commence par une description rapide de Cassandra afin de situer ces mécanismes dans le contexte général et aborde ensuite leur étude en tant que telle.

### 2.2 Description générale

#### 2.2.1 Principales caractéristiques

La figure suivante positionne Cassandra dans une grille des principales caractéristiques des bases de données NoSQL ; cette grille est extrapolée à partir des leçons du cours NFE204 :

Caratéristique		Valeurs possibles	Cassandra
Généralité	Type de données	Documentaire, Tabulaire et Graphe	Documentaire
	Mode de clustering	Maître/Escave et Multi-maitres	Multi-maitres
	Mode de communication inter-nœuds	Messages asynchrones	Messages asynchrones (Gossip)
	Mode de gestion de panne	Heartbeat et fail-over	Heartbeat et fail-over
	Mode de scalabilité	Horizontale et verticale	Horizontale et verticale (Virtual Node)
	Mode de fonctionnement général	Analytique ou Temps réel	Analytique
	Langages de modélisation et de manipulation des données	Oui et Non	Oui (CQL)
	Framework de traitement intégrée (Ex: MapReduce)	Oui et Non	Non
Réplication des données	Optimisation locale sur le nœud	Ecriture en RAM + fichiers journal	Ecriture en RAM + fichiers journal (Memcache, SSTable, Logs)
	Réplication	Asynchrone	Asynchrone
	Cohérence des données	Forte, Faible et A terme	Forte, Faible et A terme
	Consistency Availability Partition	Availability Partition	Availability Partition
Partitionnement des données	Gestion de la clé de partitionnement	Intervalle et hachage	Hachage
	Dynamacité	Statique ou Dynamique	Dynamique (Virtual Node + Hachage cohérent)

Figure 1 – principales caractéristiques de Cassandra.

La grille n'a pas pour ambition d'être exhaustive. Son seul but est d'ordre pratique : situer (très) sommairement Cassandra par rapport à quelques caractéristiques communes des SGBD NoSQL.

#### 2.2.2 Principales fonctions de l'architecture interne

Plusieurs fonctions de Cassandra opèrent de concert pour contribuer à des degrés divers et directement ou indirectement à la cohérence des données. Il semble donc judicieux d'avoir d'abord une vision globale de l'architecture interne avant d'aborder les mécanismes gestion de la cohérence des données.

La figure suivante présente les principales fonctions de l'architecture interne des nœuds d'un cluster de Cassandra :

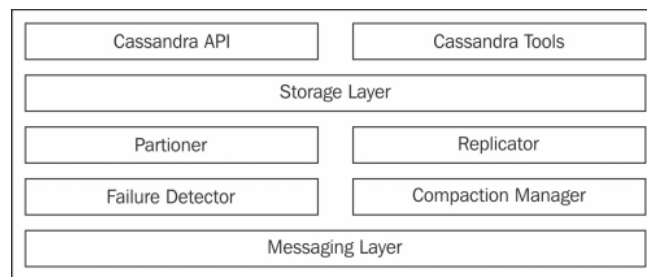


Figure 2 – principales fonctions de Cassandra.

Les lectures et écriture des données s'effectuent à travers les fonctions API et outils.

Elles sont ensuite traitées par la fonction Stockage (Storage Layer) qui joue le rôle de proxy de lecture et d'écriture et, pour ce faire, s'appuie sur les fonctions de partitionnement (Partioner) et de réplication des données (Replicator), de détection de panne (Failure Detector) et de Compactage (Compaction Manager), ainsi que la fonction de communications inter-nœuds à base des échanges asynchrones de message (Messaging Layer).

Les fonctions participant directement à la gestion de la cohérence des données – Stockage, Détection des pannes, Compactage et Communication inter-nœuds - sont décrites au chapitre 2.3 et les autres fonctions – Partitionnement et Réplication - dans la suite de ce chapitre.

### 2.2.3 Partitionnement et réplication des données

#### Hachage cohérent

Le partitionnement des données dans Cassandra est basé sur une fonction de hachage et une répartition des clés dans un anneau.

Les principaux choix pour la fonction de hachage sont Murmur3Partitioner, RandomPartitioner et ByteOrderPartitioner.

- Murmur3Partitioner est conseillé par Cassandra pour plusieurs qualités dont la rapidité et l'uniformité de la distribution des clés l'intervalle  $[-2^{63}, 2^{63} + 1]$  du domaine d'arrivée.
- RandomPartitioner est basé sur un hachage MD5 sur l'intervalle  $[0, 2^{127} + 1]$
- ByteOrderPartitioner est basé sur les octets de la clé et présente des risques de non uniformité de distribution.

Il est à noter qu'il n'est pas possible de changer la fonction de hachage en cours de route car cela rendrait la table de hachage existant incohérente.

Le hachage de Cassandra est cohérent dans le sens où il permet d'adapter les règles d'affectation des données en fonction des ajouts, modifications et suppression des nœuds, tout en conservant la même fonction de hachage.

La figure suivante présente les principes de l'anneau. Pour des raisons pratiques, l'intervalle d'arrivée de la fonction de hachage est présenté sous forme de l'alphabet :

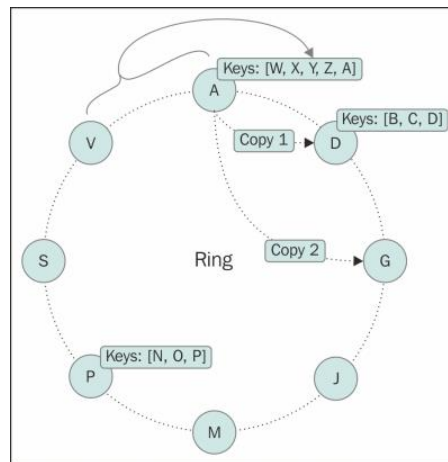


Figure 3 – anneau de hachage.

Chaque nœud gère l'intervalle qui le précède dans le sens de l'aiguille de la montre. Le nœud A gère donc les données dont les clés dans l'intervalle [X, Y, Z, A], D gère l'intervalle [B, C, D] et ainsi de suite jusqu'à V qui gère l'intervalle [T, U, V]. On note aussi que sur la figure, les données de A sont répliquées sur G.

La fonction de hachage est déployée sur tous les nœuds du cluster (voir chapitre 2.2.2).

### Nœud virtuel

Cassandra propose le concept de nœud virtuel afin de faciliter la réorganisation interne du cluster et des données, par exemple lors du processus de traitement de la panne d'un nœud, où il faut recopier les données et les clés de ce nœud vers les autres nœuds ou lors de l'ajout d'un nouveau nœud, où il faut dispatcher une partie des données et des clés des nœuds existants vers ce nouveau nœud.

Un serveur physique peut supporter  $n$  nœuds virtuels.  $n$  dépend de ses capacités de calcul et de stockage et de la bande passante disponible.  $n$  est par défaut égal à 256. Les nœuds virtuels fonctionnent de la même manière que les nœuds physiques. La figure suivante présente un anneau composé de 16 nœuds virtuels sur 4 nœuds physiques :

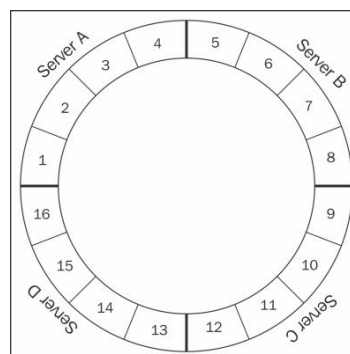


Figure 4 – nœuds virtuels et physiques.

En cas de panne d'un serveur physique, afin de respecter le facteur de réplication, ses nœuds virtuels sont dispatchés à aux serveurs physiques restant et leurs données y sont recopiées en parallèle. Ce parallélisme permet de raccourcir le temps de recopie.

De même, lors de l'ajout d'un nouveau serveur physique, plusieurs nœuds virtuels existants peuvent lui être réaffectés et leurs données recopiées de nouveau en parallèle. Le temps de configuration du nouveau est ainsi raccourci.

### Réplication des données

Afin d'assurer une bonne disponibilité, les données sont répliquées sur plusieurs nœuds différents. Le nombre de réplicas est défini par le facteur de réplication. Ce facteur est paramétrable.

Cassandra support la notion de proximité basée sur le concept de rack et de Datacenter : en fonction du paramétrage et dans la mesure du possible, la réplication se fait en priorité sur d'autres Datacenters puis dans le même Datacenter mais sur les racks différents puis dans le même rack, afin de limiter la portée des pannes. A noter que cela sous-entend que les nœuds et l'anneau peuvent être répartis sur plusieurs Datacenters.

La topologie des Datacenter et racks peuvent être découverte automatiquement à l'aide du protocole Snitch (voir chapitre 2.3.2.2).

## 2.3 Gestion de la cohérence des données

### 2.3.1 Définition de la cohérence

Pour rappel, un système cohérent est un système qui donne la même réponse pour une même requête au même moment, quel que soient les réplicas interrogés.

### 2.3.2 Lecture et écriture

Les principaux composants mis en œuvre sur tous les nœuds pour le support de la lecture et de l'écriture des données sont la table MemTable, les tables SSTable et le journal Commit Log.

La table de hachage MemTable réside en mémoire vive et stocke les données dans des cellules. Les tables SSTable sont sur disque. MemTable flushes y les données notamment lorsque la mémoire vive est saturée. Le fichier Commit log est également sur disque et enregistre les changements des données, dits mutations, qui sont dans MemTable mais pas encore flushés sur disque.

#### 2.3.2.1 Opération d'écriture

Les clients se connectent à n'importe quel nœud qui joue alors le rôle de coordinateur.

Le nœud coordinateur délègue l'opération d'écriture à la fonction Proxy de Stockage qui détermine à son tour en fonction du facteur de réplication la liste des nœuds où l'écriture sera effectuée. Le coordinateur envoie ensuite une demande d'écriture à chaque nœud de la liste, attend les messages de confirmation en retour, les compte et, une fois un seuil fixé par le



paramètre Niveau de cohérence en écriture (ConsistencyLevel) atteint, retourne la confirmation d'écriture aux clients.

Par exemple, avec un facteur de réplication de 3 et un niveau de cohérence en écriture de 2, le coordinateur envoie 3 demandes d'écriture aux 3 nœuds responsables des 3 réplicas et confirme aux clients dès réception des confirmations de 2 de ces 3 nœuds (Le concept de niveau de cohérence sera approfondi au chapitre 2.3.2.3).

La figure suivante présente un exemple de l'écriture sur 7 nœuds dans 3 Datacenters :

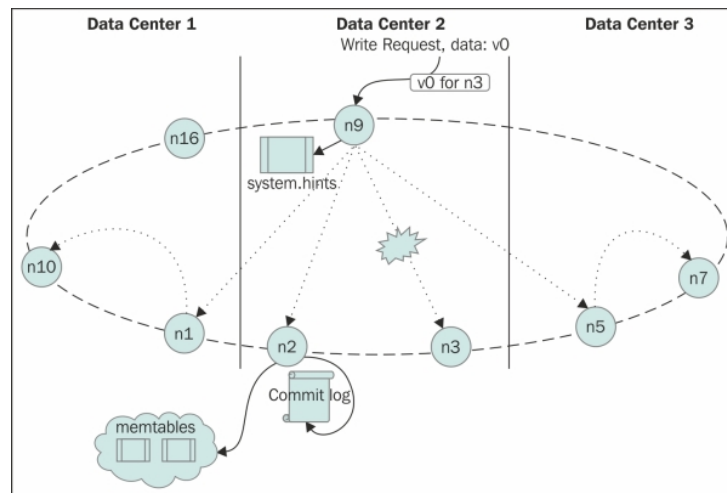


Figure 5 – déroulement d'une écriture.

Lors du déroulement de l'écriture,

- Si des nœuds tombent en panne et le niveau de cohérence en écriture n'est pas conséquent pas atteignable, l'écriture échoue.
- S'il y a suffisamment de nœuds mais des time-outs surviennent à cause d'un problème temporaire comme une surcharge momentanée de l'infrastructure, la fonction Proxy de Stockage écrit les données en local, afin de la rejouer une fois la situation redevenue normale. Ce mécanisme s'appelle Hinted Handoff (voir chapitre 2.3.2.4).
- Si plusieurs réplicas sont dans un même Datacenter distant, la fonction Proxy de Stockage n'envoie la demande d'écriture qu'à un seul nœud; ce nœud la relaie ensuite aux autres nœuds dans le même DC.
- A la réception de la demande d'écriture, chaque nœud journalise la demande dans le fichier de Commit Log puis écrit les données dans les cellules appropriées de la MemTable.

Si la MemTable arrive à saturation, les données sont flushées dans une nouvelle SSTable créée pour l'occasion et le Commit Logs est mis à jour.

Comme une nouvelle SSTable est créée à chaque flush, leur nombre croît au fil du temps et finit par dégrader les performances, en rendant les déplacements des têtes de disques plus fréquents. Afin d'y remédier, un compactage est lancé et périodiquement et quand le nombre de SSTable atteint un seuil paramétrable, pour fusionner plusieurs SSTables dans une table plus grande.

Commit Log permet également d'assurer la pérennité des données en cas de panne d'un nœud. En effet, au redémarrage du nœud en panne :

- La référence du dernier enregistrement du Commit Log à être flushé sur disque est retrouvée à l'aide des métadonnées des SSTable.
- Les opérations d'écriture journalisées sont rejouées dans l'ordre croissant de l'horodatage. MemTable est alors progressivement rempli au fur et à mesure.
- A la fin des exécutions, MemTable est flushé dans une nouvelle SSTable ; les segments du Commit Log deviennent « vides » et sont recyclables pour les nouvelles écritures.

### 2.3.2.2 Opération de lecture

Comme pour la lecture, les clients se connectent à n'importe quel nœud qui joue alors le rôle de coordinateur.

Le coordinateur délègue l'opération de lecture à la fonction Proxy de Stockage qui détermine la liste des nœuds ayant un réplica des données à lire. Ces nœuds sont triés ensuite par l'ordre de proximité décroissant par rapport au coordinateur. Les nœuds les plus proches sont censés d'être les plus rapides pour lui retourner les réponses.

L'estimation de la proximité est effectuée à l'aide de la fonction Snitch. Snitch est paramétrable et peut prendre les valeurs suivantes : SimpleSnitch, PropertyFileSnitch, GossipingPropertyFileSnitch, RackInferringSnitch, EC2Snitch, EC2MultiRegionSnitch et DynamicSnitch.

- SimpleSnitch : le nœud le plus proche est simplement le prochain sur l'anneau,
- PropertyFileSnitch : Snitch détermine la proximité en fonction des déclarations dans le fichier cassandra-topology.properties. La localisation des nœuds y est décrite sous la forme « adresse IP, n° du Datacenter, n° du Rack dans le DC ». Ce fichier est également à modifier à chaque ajout et suppression de nœud et à copier manuellement sur tous les nœuds.
- GossipingPropertyFileSnitch: Snitch détermine la proximité en fonction des déclarations dans le fichier cassandra-rackdc.properties. Le fichier contient les mêmes informations que celui de PropertyFileSnitch et est aussi à modifier à chaque ajout et suppression de nœud. Toutefois sa copie sur tous les nœuds est automatique, à l'aide du protocole GOSSIP.
- RackInferringSnitch : le numéro du Datacenter est le 2<sup>ème</sup> octet de l'adresse IP du nœud et celui du rack le 3<sup>ème</sup>.
- EC2Snitch et EC2MultiRegionSnitch : pour les nœuds hébergés sur EC2 d'Amazon.
- DynamicSnitch : la proximité d'un nœud est calculée à partir de ses derniers temps de réponse.

La fonction Proxy Stockage du coordinateur envoie la demande de lecture de données au nœud le plus proche et des demandes du digest des mêmes données aux autres nœuds. Lors de la réception des confirmations, elle calcule le digest des données du nœud le plus proche et le compare aux digests des autres nœuds.

- Si un digest diffère des autres, cela veut dire qu'il y a de différentes versions des mêmes données dans le cluster. La fonction Proxy Stockage détermine alors la dernière version, récupère les données correspondantes, confirme aux clients en fonction du paramètre Niveau de cohérence en lecture (voir chapitre 2.3.2.3) et lance la mise à jour des nœuds qui n'ont pas la dernière version en fonction du paramètre `read_repair_chance`. Ce mécanisme est appelé réparation des données incohérentes lors de leur lecture.
- Sinon, le Proxy de Stockage envoie simplement la confirmation aux clients, toujours en fonction du niveau de cohérence en lecture.

A l'intérieur d'un nœud,

- La tentative de lecture est effectuée d'abord dans MemTable, dont l'accès est le plus rapide. Si les données ne s'y trouvent pas, la lecture est tentée dans les SSTables.
- Le Bloom Filter d'une SSTable est alors consulté pour déterminer si les données à lire s'y trouvent. Bloom Filter réside en mémoire vive et il y en a un par SSTable. La réponse de Bloom Filter est probabiliste : elle ne donne jamais de faux négatif (réponse négative alors que la SSTable contient les données recherchées) mais peut donner des faux positifs (réponses positives alors que SSTable ne contiennent pas les données recherchées).
- Une fois la réponse positive d'un Bloom Filter obtenue, l'accès aux données est accéléré à l'aide de l'index PartitionIndexSummary. Il y a un tel index par SSTable. Cet index réside aussi en mémoire vive et contient les pointeurs vers les 1<sup>ère</sup>, (1 + n)<sup>ème</sup>, (1 + 2 x n)<sup>ème</sup> ligne et ainsi de suite de sa SSTable. PartitionIndexSummary permet donc de situer les données recherchées dans un intervalle [k, k+n[, n étant généralement égal à 128.
- Une fois l'intervalle obtenu, l'accès aux données est accéléré à l'aide de l'index PartitionIndex. Il y a un tel index par SSTable. Cet index est sur disque.

La figure suivante résume le déroulement général d'une lecture :

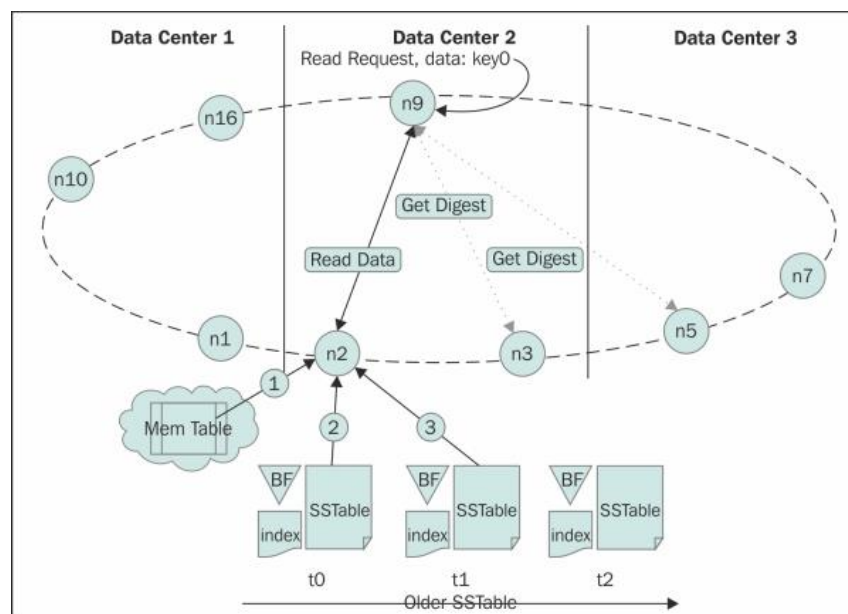


Figure 6 – déroulement d'une lecture.

### 2.3.2.3 Niveaux de cohérence en lecture et en écriture

La cohérence des données est paramétrable en fonction des exigences en termes de cohérence des données et de rapidité des opérations de lecture et d'écriture.

La figure suivante présente les niveaux de cohérence possibles,  $n$  étant le nombre de nœuds du cluster :

Niveau de cohérence	Signification pour les opérations d'écriture	Signification pour les opérations de lecture
Zero	"Fire and forget"	
Any	Réussi sur retour de Hinted Hand Off	
One, Two ...	Réussi si une, deux ou ... confirmations des répliquats reçues	Réussi si une, deux ou ... confirmations des répliquats reçues
Quorum	Réussi si $(n/2 + 1)$ confirmations reçues	Réussi si $(n/2 + 1)$ confirmations reçues
All	Réussi si toutes les confirmations sont reçues	Réussi si toutes les confirmations sont reçues

Figure 7 – paramètres du niveau de cohérence.

Par exemple, pour un cluster dont le facteur de réplication est de 3 :

- Zero : les opérations d'écriture sont réussies sans aucune confirmation du nœud coordinateur,
- Any : les opérations d'écriture sont réussies dès que le nœud coordinateur finit le mécanisme de Hinted Handoff,
- One/Two : les opérations d'écriture et de lecture sont réussies dès réception d'une/deux confirmation(s) des nœuds responsables des répliquats.  
A noter que dans cet exemple, Three est équivalent à All et Four ou plus provoque l'échec systématique des opérations d'écriture ou de lecture.
- Quorum : les opérations d'écriture et de lecture sont réussies dès réception de  $(2/2 + 1) = 2$  confirmations.
- All : les opérations d'écriture et de lecture sont réussies à la réception de la totalité des 3 confirmations.

La rapidité des opérations diminue et la cohérence des données augmente quand on va de Zero à All.

Soit  $R$  est le niveau de cohérence des opérations de lecture et  $W$  celui des opérations d'écriture, en configurant de telle sorte que  $R + W > n$ , on s'assure de la cohérence forte des données. Autrement la cohérence est au mieux à terme.

En choisissant le niveau de cohérence Quorum pour les opérations d'écriture, on peut se prémunir contre le partitionnement du réseau car les opérations d'écriture échoueront de facto pour une des deux partitions.

Les choix de  $R$  et  $W$  dépendent donc des exigences en termes de rapidité des opérations et de cohérence de données.

Un compromis en vue d'une bonne cohérence des données et d'une raisonnable rapidité peut être par exemple  $W = \text{Arrondi\_Supérieur}(n/2 + 1)$  et  $R = n - W + 1$ .

#### 2.3.2.4 Hinted Handoff

Le concept de Hinted Handoff permet au nœud coordinateur de gérer le fait qu'un des nœuds responsables des réplicas peut tomber en panne pendant une opération d'écriture.

Dans ce cas-là, si le niveau de cohérence en écriture est malgré tout atteint, le nœud coordinateur confirme l'écriture au client et stocke les données à écrire dans la table locale `system.hint`, en attendant le retour en service du nœud en panne.

Le nœud coordinateur vérifie le retour en service des nœuds en panne toutes les 10 minutes et, le cas échéant, relance les écritures puis efface les données de la table `system.hint`.

Le Hinted Handoff permet surtout d'accélérer la mise en cohérence des données mais n'est pas lui-même un mécanisme de gestion de la cohérence à proprement parlé car le nœud coordinateur peut tomber aussi en panne et perdre la table `system.hint`.

### 2.3.3 Suppression des données

#### 2.3.3.1 Opération de suppression

Le déroulement de la suppression des données est très similaire à celui de l'écriture (voir chapitre 2.3.2.1.). Néanmoins les colonnes à supprimer de la MemTable ou des SSTable ne sont pas réellement supprimées ; leurs contenus sont simplement remplacés par une valeur spéciale dite Tombstone.

Les données ainsi marquées peuvent être flushées par MemTable comme les autres données dans une nouvelle SSTable ou retournées au nœud coordinateur lors des opérations de lecture (voir chapitre 2.3.2.2) ; elles ne sont toutefois pas renvoyées au client. Elles peuvent aussi participer à la réparation des données incohérentes lors de leur lecture.

Néanmoins elles sont nettoyées des SSTable lors du compactage, à condition que délai de grâce paramétrable `gc_grace_seconds` soit dépassé.

Elles peuvent donc ressusciter si un des nœuds qui hébergent leurs réplicas tombe en panne puis revient en service et, entre temps, le compactage a nettoyé les autres réplicas. En effet, ce nœud est alors le seul à les avoir et elles peuvent être prises à tort pour des données fraîches qui ne sont pas encore répliquées.

#### 2.3.3.2 Anti-entropie

Le mécanisme d'anti-entropie traite les risques d'incohérence des données ressuscitées du chapitre précédent ainsi que des données écrites mais jamais lues et des données du Hinted Handoff en cas de panne du nœud qui le gère.

Il est lancé lors des compactages dits majeurs des SSTable et consiste à comparer les familles de colonnes contenues dans ces tables avec leurs réplicas et, le cas échéant, à lancer les réparations similaires à celles effectuées lors des opérations de lecture.

Les comparaisons sont effectuées sur des arbres de digests des réplicas, dits arbres de Merkle.

La figure suivant présente 2 arbres de Merkle et leur comparaison :

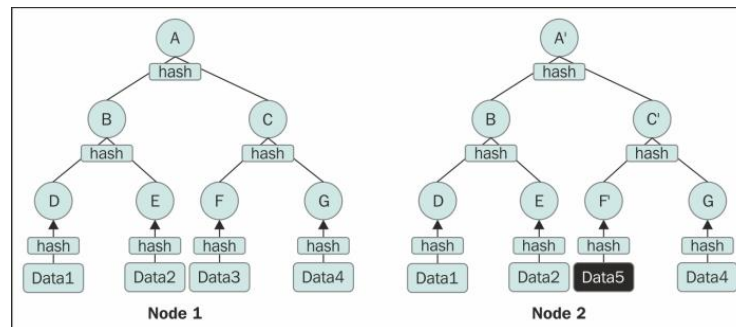


Figure 8 – comparaison d’arbres de Merkle.

Les données d’un réplica sont organisées sous forme arborescente. L’arbre de Merkle en a la même structure mais composé uniquement des hashes. Ainsi comparer 2 réplicas revient à comparer les 2 arbres de Merkle associés en partant des 2 racines, ce qui permet de trouver très rapidement les données qui diffèrent.

### 2.3.4 Communication inter-nœuds

Un nœud ouvre 2 sockets vers chacun des autres nœuds. Il y a donc par exemple 198 sockets par nœud dans un cluster de 100 nœuds.

#### 2.3.4.1 GOSSIP

Un nœud peut héberger un ou plusieurs End-Point et maintient localement une table des états de tout ou partie de l’ensemble des End-Points du cluster.

Le protocole GOSSIP permet à chaque nœud a) de mettre à jour sa table des états en initiant des requêtes d’interrogation vers un ou plusieurs autres nœuds b) de communiquer sa table des états lorsqu’il est interrogé à son tour.

Les requêtes d’interrogation sont lancées toutes les secondes. L’ensemble des tables des états convergent ainsi au fil du temps de manière « virale ».

Voici un échange typique de GOSSIP entre 2 nœuds A et B :

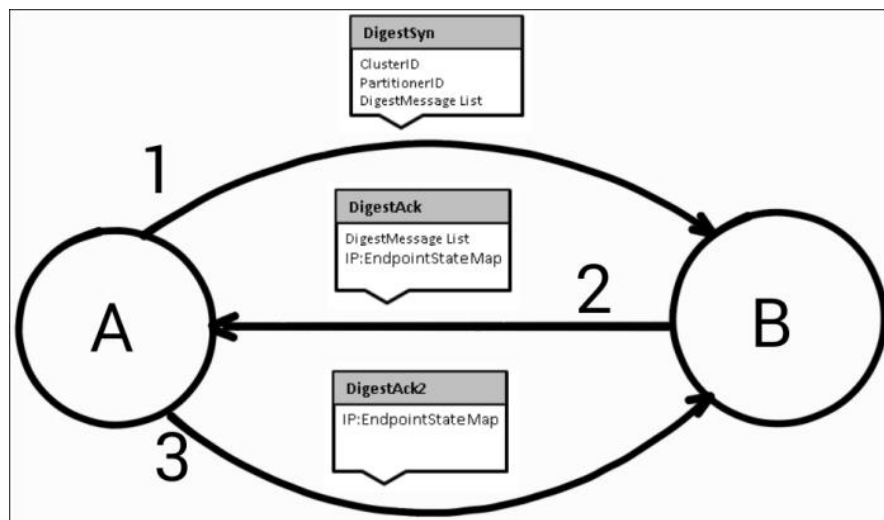


Figure 9 – échange GOSSIP.

Les informations sur l'état des End-Point sont transportées dans la structure DigestMessage : leur adresse IP (Key dans l'exemple suivant), leur état (status) et les numéros de génération (HeartbeatState-154123891) et de version (HeartbeatState-20) de cet état :

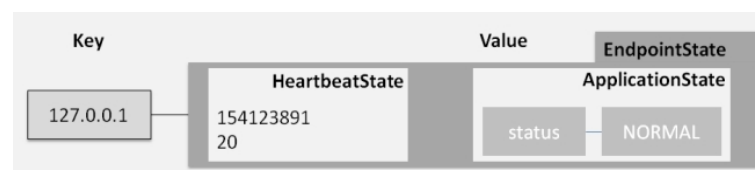


Figure 10 – exemple de DigestMessage.

A transmet à B sa table des états dans les DigestMessage de la requête DigestSync.

B compare les End-Point reçus avec sa propre table des End-Point :

- Si un End-Point de A n'y est pas encore, B demande à A de lui envoyer les données plus fraîches dans sa réponse DigestAck à DigestSync.
- Si un End-Point de A y est mais d'une génération plus récente ou de la même génération mais d'une version plus récente, B demande à A de lui envoyer les données plus fraîches dans la même réponse DigestAck.
- Si un End-point de A y est mais d'une génération plus ancienne ou de la même génération mais d'une version plus ancienne, B transmet ses données à A dans la même réponse DigestAck.

Le cas échéant, A envoie d'un côté les données demandées à B dans un message DigestAck2 et, de l'autre, propage les informations plus fraîches reçues de B aux autres nœuds dans d'autres messages DigestAck2.

De cette façon, GOSSIP contribue à la gestion de la cohérence des données.

#### 2.3.4.2 Seed node

Afin de limiter les échanges lors de l'initialisation, au lieu d'envoyer des messages GOSSIP à tous les nœuds de l'anneau, un nœud nouvellement créé contacte un Seed Node dans de son Datacenter pour récupérer la configuration du cluster.

Il est donc souhaitable que plusieurs nœuds portent la fonction Seed Node dans un même Datacenter, pour des raisons de redondance.

#### 2.3.4.3 Détection des pannes

Chaque nœud maintient une liste horodatée des échanges de GOSSIP avec les autres nœuds.

Afin de limiter les bagotements du cluster lors de l'absence de réponse, le nœud interrogé qui ne répond pas n'est pas immédiatement considéré comme en panne mais seulement au bout d'une temporisation appelée  $\phi$  (Phi) avec  $\phi = -\text{Log}_{10} (P(t_{\text{now}} - t_{\text{last}}))$  où  $t_{\text{now}}$  est le temps courant,  $t_{\text{last}}$  est la date où la dernière réponse GOSSIP est reçue et  $P(t_{\text{now}} - t_{\text{last}})$  la probabilité que la réponse GOSSIP arrive après l'intervalle de temps  $(t_{\text{now}} - t_{\text{last}})$ .

Ainsi, en cas d'absences répétées de réponse, l'intervalle  $(t_{\text{now}} - t_{\text{last}})$  s'allonge,  $\phi$  croît progressivement jusqu'à ce qu'il dépasse un seuil paramétrable et le nœud interrogé est alors considéré en panne (voir <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106.3350> pour plus de détails sur  $\phi$ ).

#### 2.3.5 Conclusion

Plusieurs mécanismes complémentaires les uns des autres contribuent à la gestion de la cohérence des données dans Cassandra.

Ils n'ont pas toutes les mêmes importances. Certains sont indispensables comme le compactage majeur, l'anti-entropie et le Commit Log, d'autres révèlent plus de l'optimisation des performances comme la réparation en lecture, Snitch et les différents niveaux de cohérence en lecture et en écriture et d'autres encore contribuent de façon plus indirecte comme la détection des pannes et GOSSIP.

Dans tous les cas, conformément à ce qui est enseigné dans le cours NFE204, les mécanismes de réconciliation comme le compactage majeur et l'anti-entropie s'avèrent les plus importants dans le cadre de l'architecture multi-maître de Cassandra.