CentraleSupélec

Big Data : Informatique pour les données et calculs massifs

9 – Technologie des moteurs de BdD NoSQL

Stéphane Vialle

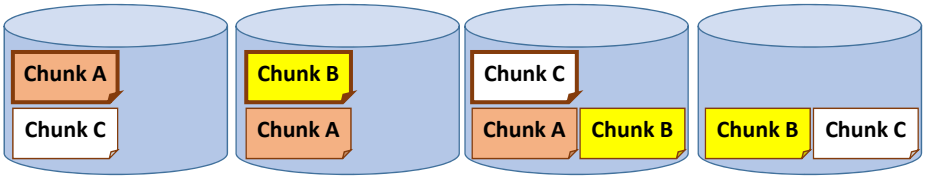
CentraleSupélec université PARIS-SACLAY ÉCOLE DOCTORALE Sciences et technologies de l'information et de la communication (STIC) RISEGrid Grand Est ALSACE CHAMPAGNE-ARDENNE LORRAINE

Stephane.Vialle@centralesupelec.fr
http://www.metz.supelec.fr/~vialle

CentraleSupélec

Concepts de sharding et réplications

Principe de *sharding* et réplication



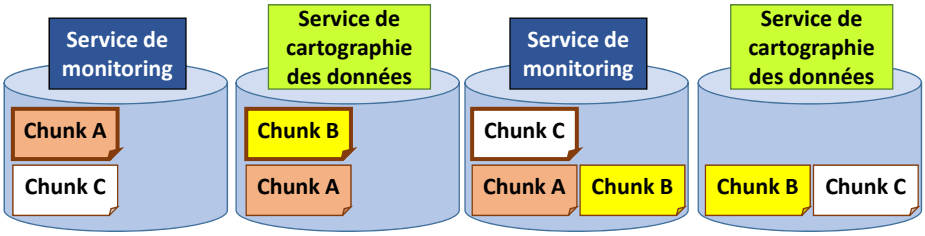
Distribution et réplication:

- Un fichier/une table est découpé en morceaux (*chunks*) distribués pour permettre:
  - des accès parallèles plus rapides (répartir la charge)
  - des stockages plus volumineux (et extensibles)
- Les *chunks* sont répliqués sur des nœuds différents pour la tolérance aux pannes

Concepts d'HDFS

- Le *sharding* est un découpage des différentes lignes d'une table par tranches dans différents *chunks* (au lieu de distribuer les colonnes)

Principe de Sharding et réplication



Des services de :

- Cartographie du stockage
- Monitoring & Hearbeat

Concepts d'HDFS

Réplication des services pour la tolérance aux pannes

CentraleSupélec

Architecture de HBASE
(Microsoft & OpenSrc)

Microsoft Apache

5

CentraleSupélec

Archi. HBase au dessus d'Hadoop

BdD orientée colonnes

100	vente-2010	100000	vente-2011	150000	vente-2014	180000	
200	vente-2012	1000					
211	vente-2010	500000	achat-2016	10000			

- Une ligne est indexée par une clé unique
- Les colonnes sont regroupées en familles
  Une famille est stockée dans un même fichier *Hfile*
- Des options de compression peuvent être associées à une famille de colonnes
- Un *versionning* est disponible pour chaque colonne
- Bâtie au dessus d'Hadoop et d'HDFS

Archi. HBase au dessus d'Hadoop

BdD orientée colonnes

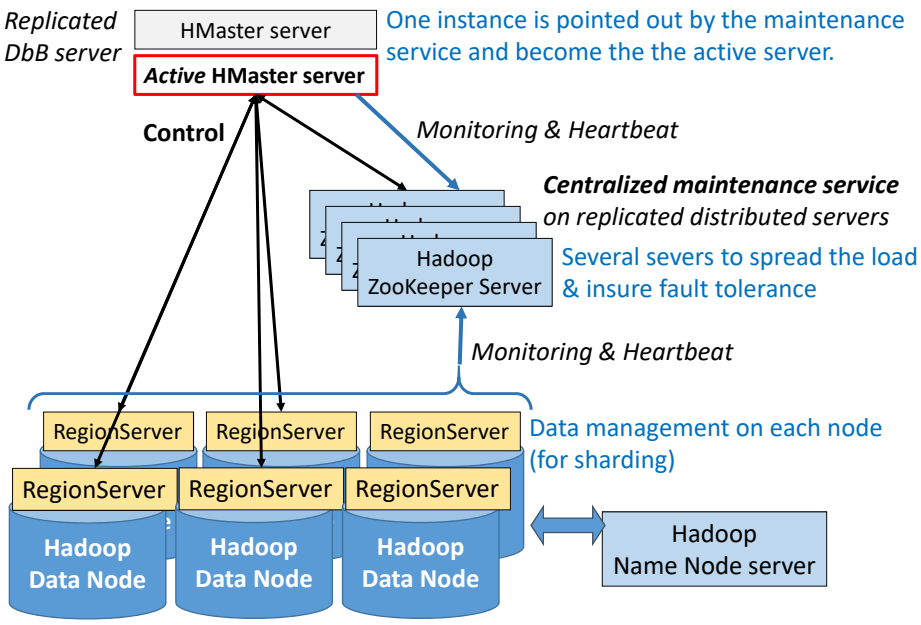
2 niveaux de gestion de la distribution des données :

- *Sharding* des tables de données en *Hfiles*
  → fait par des *Serveurs de Région* de Hbase (1 par nœud)
- Gestion/distribution/réplication des fichiers *Hfiles*
  → fait par *HDFS d'Hadoop*

***Sharding* progressif d'une table :**

1. Au début une table est contenue dans une région (un nœud)
2. Quand elle grossit, une région est scindée en 2 sous-régions sur le même serveur de région (le même nœud)
3. Séparation signalée au maître (le *Hmaster*)
4. Peut demander la migration d'une sous-région sur une autre serveur de région

Archi. HBase au dessus d'Hadoop



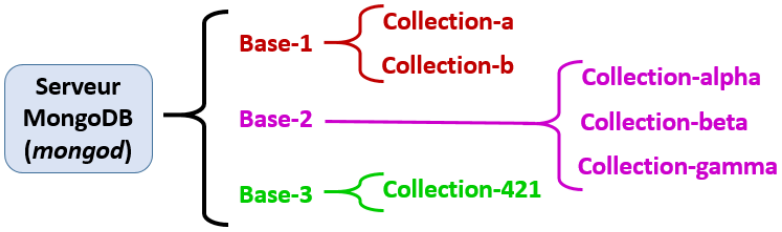
CentraleSupélec

Architecture et *mapReduce* de MongoDB

mongoDB.

CentraleSupélec

Organisation MongoDB

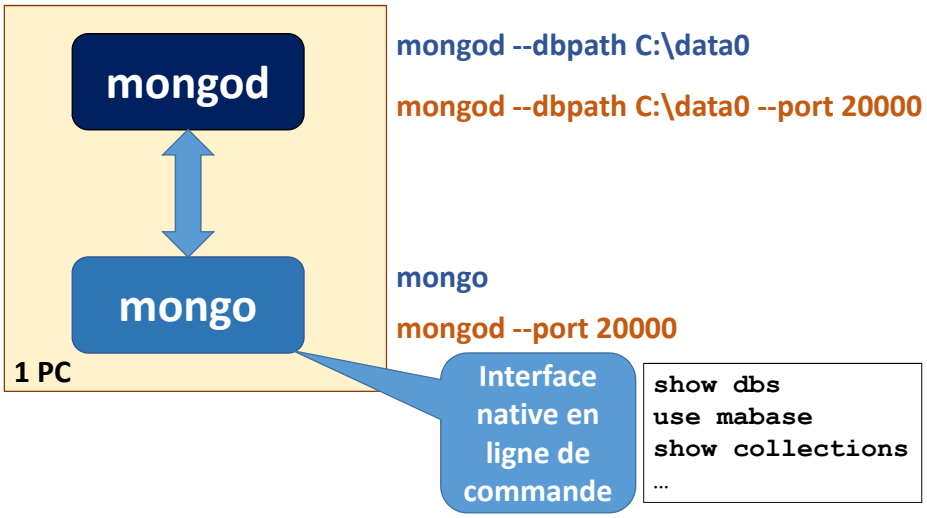


Un serveur × des bases × des collections

L'équivalent des tables/relations en BdD SQL

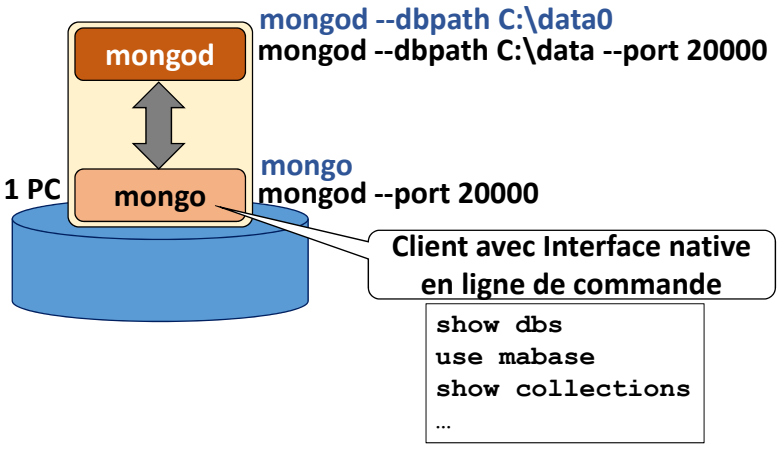
Déploiement local de MongoDB

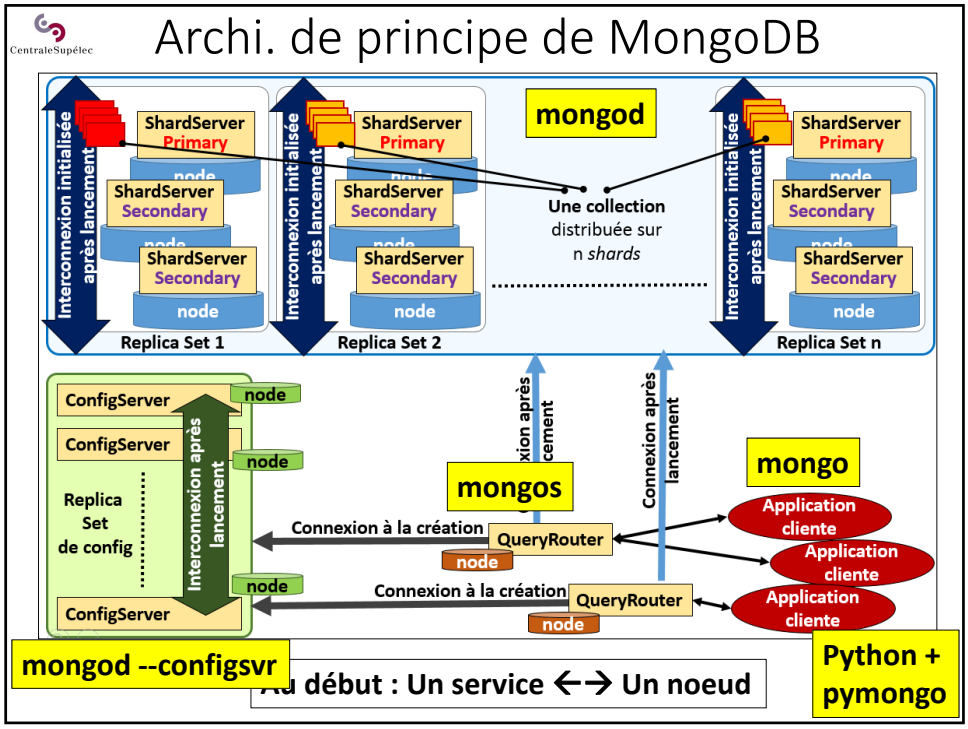
Déploiement sur UN PC, non shardé, non répliqué

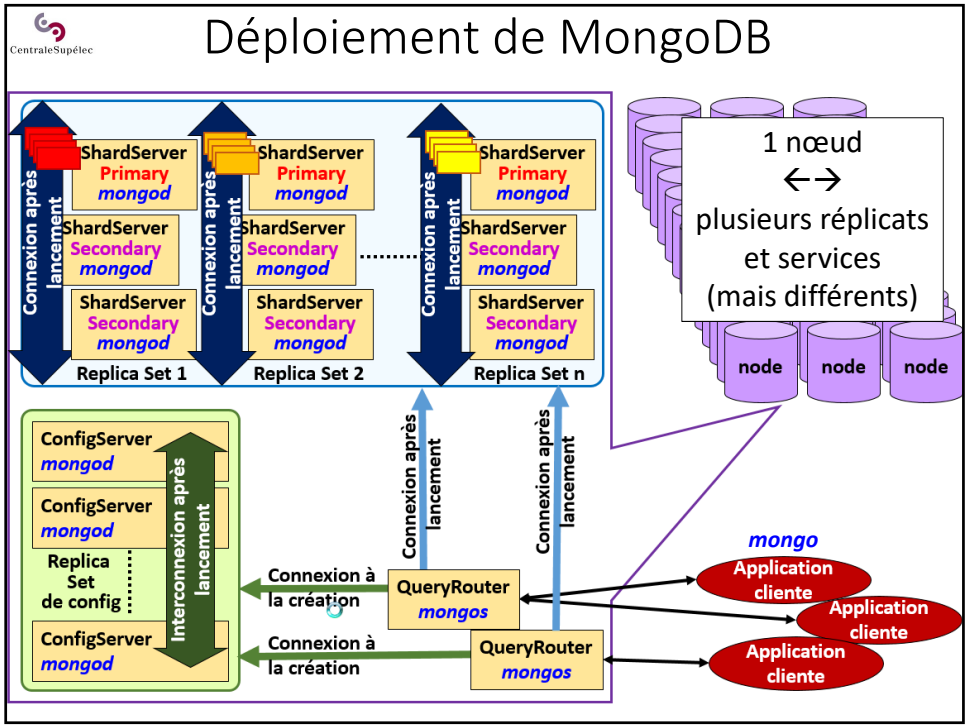


Déploiement local de MongoDB

Déploiement sur UN PC, non shardé, non répliqué




Archi. de principe de MongoDB
CentraleSupélec
mongod
ShardServer Primary
ShardServer Secondary
node
Interconnexion initialisée après lancement
Une collection distribuée sur n shards
Replica Set 1
Replica Set 2
Replica Set n
ConfigServer
Replica Set de config
Interconnexion après lancement
mongos
Connexion à la création
QueryRouter
Connexion après lancement
mongo
Application cliente
mongod --configsvr
Au début : Un service ←→ Un noeud
Python + pymongo


Déploiement de MongoDB
CentraleSupélec
ShardServer Primary mongod
ShardServer Secondary mongod
Connexion après lancement
Replica Set 1
Replica Set 2
Replica Set n
1 nœud
←→
plusieurs réplicats et services (mais différents)
node
ConfigServer mongod
Replica Set de config
Interconnexion après lancement
Connexion à la création
QueryRouter mongos
mongo
Application cliente

mapReduce de MongoDB (1)

MongoDB possède son propre « Map-Reduce » :

- **Ses propres règles de fonctionnement** (un peu différentes d'Hadoop)
- Ses propres règles de déploiement des tâches
- Et son propre middleware sous-jacent (il n'est pas bâti au dessus d'Hadoop)
- **Fonctionne sur des bases distribuées (*sharded*)**

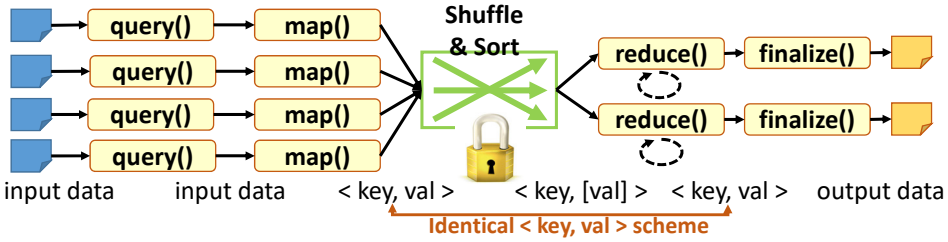
Principes du *mapReduce* de MongoDB :

- Une **query** pour pré-filtrer la collection traitée
- Une fonction **map()**, en Java Script et qui accède à la base
- Une fonction **reduce()**, en Java Script et qui ne doit PAS accéder à la base (seulement aux résultats du map()), qui doit être commutative, associative et idempotente (!!)
- Une fonction **finalize()**, en Java Script et qui ne doit pas accéder à la base
- La possibilité de définir un ensemble de variables globales aux 3 fonctions map(), reduce() et finalize()

15

mapReduce de MongoDB (2)

Principes du *mapReduce* de MongoDB :

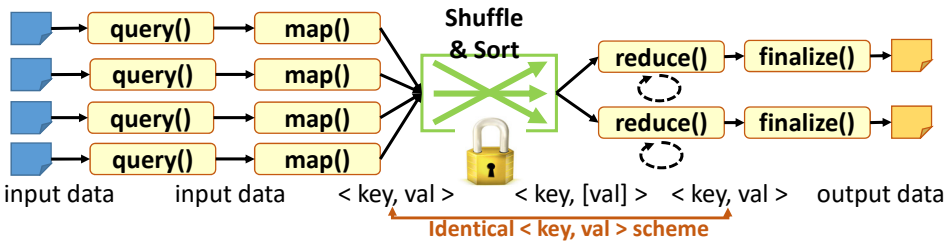


- ***map()*** fonctionne comme en Hadoop, mais **sur une seule collection**
- pour éviter d'implanter un filtrage en JS, une ***query*** exprimée en Mongo shell est applicable en amont (plus pratique et plus rapide)
- ***reduce()*** est **plus contraint qu'en Hadoop** car MongoDB applique *reduce()* sur des < key , [val] > de sortie de map(), et sur des sorties précédentes de *reduce() :*
  - **le format de sortie de *reduce()* doit être celui de sortie de *map()***
  - ***La fonction reduce() doit être commutative, associative et indempotente***
- ***finalize()*** permet de modifier la sortie finale de *reduce()* sans contrainte

16

mapReduce de MongoDB (3)

Principes du *mapReduce* de MongoDB :



- **Pas** de ***Combiner***
- **Pas d'optimisation** possible du ***Shuffle & Sort***

Les contraintes sur la fonction *reduce()* :

- **évitent les débordements mémoire**, en traitant chaque paire < clé, liste de val > en plusieurs passes si la liste de valeurs est « grande »
- **mais…limitent fortement les algorithmes possibles** en une seule opération mapReduce.

17

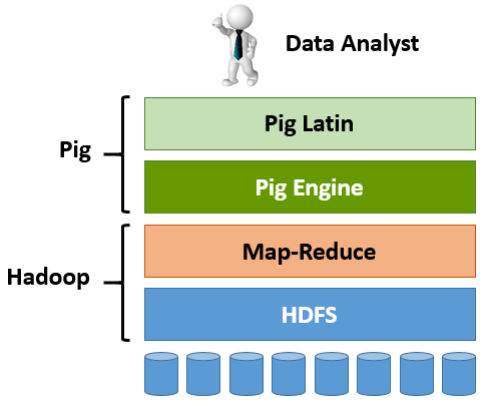
Environnements NoSQL de plus haut niveau

- PIG
- HIVE

18

CentraleSupélec

PIG

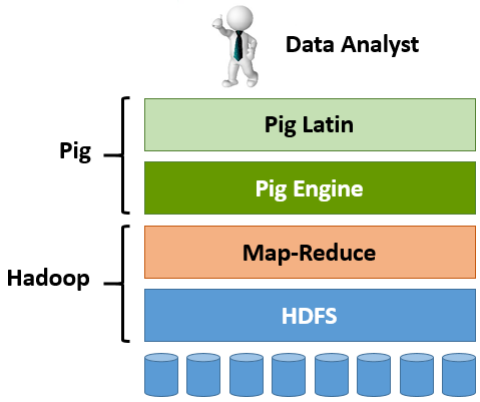


Yahoo! développe en 2006 un outil pour faire « presque du SQL » au dessus d'Hadoop pour simplifier le travail de ses équipes de data-analyst : PIG

Devient OpenSource en 2007

CentraleSupélec

PIG



Langage de requête/programmation : « Pig-Latin »

Ex de commandes : FILTER, GROUP BY, JOIN, SORT …
→ Traduit en opérations Map-Reduce

PIG : PIG-Latin

```
grunt> DUMP ETUDIANT;
(Dupond, Alphonse, 10, Metz)
(Dupond, Grégoire, 9, Rennes)
(Durant, Hubert, 12, Gif)
(Talon, Achille, 15, Gif)

grunt> ETUDIANT2 = FOREACH ETUDIANT GENERATE $0, $1, $2+1 ;

grunt>DUMP ETUDIANT2;
(Dupond, Alphonse, 11)
(Dupond, Grégoire, 10)
(Durant, Hubert, 13)
(Talon, Achille, 16)
```

PIG : PIG-Latin

```
grunt> DUMP VEHICULE;
(AA 123 AB, bleue, Twingo)
(AZ 451 GT, noire, C3)
(BC 634 FY, jaune, Twingo)
(DE 398 AA, blanche, DS4)

grunt>DUMP CONSTRUCTEUR;
(Twingo, Renault)
(C3, Citroen)
(DS4, Citroen)

grunt> VOITURE = JOIN VEHICULE BY $2, CONSTRUCTEUR BY $0;

grunt>DUMP VOITURE;
(AA 123 AB, bleue, Twingo, Twingo, Renault)
(AZ 451 GT, noire, C3, C3, Citroen)
(BC 634 FY, jaune, Twingo, Twingo, Renault)
(DE 398 AA, blanche, DS4, DS4, Citroen)
```

PIG : PIG-Latin

```
grunt> VOITURE2 = FOREACH VOITURE GENERATE $0, $1, $2, $4 ;
grunt> VOITURE3 = ORDER VOITURE2 BY $2, $0 DESC ;
(AZ 451 GT, noire, C3, Citroen)
(DE 398 AA, blanche, DS4, Citroen)
(BC 634 FY, jaune, Twingo, Renault)
(AA 123 AB, bleue, Twingo, Renault)
```

PIG : PIG-Latin

Stratégie :

- Traduit les requêtes en DAG d'opérations Map-Reduce (!!)
- Exprime un maximum de parallélisme entre les opérations du DAG (souci de performance)
- Le « PIG-engine » exécute le DAG, et fixe le nombre de *Reducers au mieux* (mais on peut le guider)

Rmq :

- SQL : langage déclaratif : on dit ce que l'on veut obtenir
- Pig-Latin : langage procédural : on dit quoi faire (pour obtenir le résultat).

CentraleSupélec

Hive

Facebook 2005 : encore plus proche de SQL que PIG

- Hive : langage déclaratif proche de SQL
- Pour des data-analysts qui ne peuvent pas programmer du Map-Reduce (pas programmer en langage procédural)
- Génère du Map-Reduce au dessus d'Hadoop.

CentraleSupélec

Technologie des moteurs de BdD NoSQL