Utilisation & Administration

Sommaire général

- Rappels SQL
- "Mouvement" NoSQL
- Différentes implémentations
- Big-Data
- Hadoop
- MongoDB

SGBD-R et SQL

Fonctionnalités d'un SGBD

- Décrire les données et les manipuler
- Contrôler l'intégrité des données
- Contrôler l'accès aux données
- Gérer les accès concurrents (partage)
- Assurer la sécurité de fonctionnement (reprise, après panne)
- Assurer de bonnes performances (index)
- Indépendance physique vis-à-vis des technologies de stockage
- Indépendance logique (vues logiques différentes sur les mêmes données)

SGBD-R

Définition d'un SGBD-R

- Définit par Codd (1970 - IBM)
- Méthode d'organisation des données en matrices (appelées tables ou relations) qui modélisent le monde réel
- Les relations représentent les entités du monde réel ou les associations entre ces entités
- On va pouvoir appliquer les opérations de l'algèbre relationnel

5

J.Delamarche (2002-2015)

SGBD-R

Vocabulaire et termes

- Une **entité** est un ensemble de données cohérents ayant des caractéristiques communes
  - **PERSONNE**, **VEHICULE...**
- Une **relation** (ou **association**) représente un moyen de relier, souvent par une action deux ou plus de deux entités
  - une personne **CONDUIT** un véhicule
- Un **attribut** est une propriété ou caractéristique qualifiant l'entité
  - personne.**SEXE**, vehicule.**IMMATRICULATION**
  - possède un nom et un type de donnée
- Un **domaine** est l'ensemble des valeurs que pourra prendre l'attribut
  - sexe = [**HOMME**, **FEMME**]
- La **cardinalité** est le nombre possible des liens d'une relation
  - une personne conduit **0** ou **1** véhicule
  - une personne possède **0** ou **N** véhicules
- Un **identifiant** est un attribut ou un ensemble d'attribut permettant d'identifier de façon unique une occurrence de l'entité
  - le **n°de sécurité sociale 1600475112335** permet d'identifier de manière unique une personne
- Une **clé étrangère** est un attribut d'une relation qui fait référence à la clé d'une autre relation
  - c'est ainsi que l'on pourra lier plusieurs relations

6

J.Delamarche (2002-2015)

SGBD-R

Motivations pour créer un "schéma"

- En dehors des interrogations, les opérations courantes sur une base de données en fonctionnement sont essentiellement :
  - l'insertion de nouvelles données
  - la modification de données présentes
  - leur suppression.
- Chacun de ces traitements peut provoquer l'apparition d'incohérences dans la base
  - par exemple, une même personne avec deux dates de naissance différentes
  - ou la perte non voulue d'informations
- Ces problèmes révèlent une mauvaise organisation de la base de données
  - =>le risque doit donc être prévenu lors de la conception de cette base

SGBD-R

Conception de la base de données

- Schéma de base = ensemble des tables et des relations entre les tables
- Deux méthodologies sont courantes pour parvenir à un schéma viable de la base de données :
  - concevoir un modèle entités-relations des données, puis le traduire dans le modèle relationnel
  - écrire directement le schéma de la base dans le modèle relationnel puis tester s'il est en forme "normale"

SGBD-R

Traduction d'un schéma E/R

- Pour chaque entité du schéma, on définit une table
- Pour une association *A* -> *B* de type *1-n*, on ajoute un attribut à la table *B* pour recevoir les clefs de *A*
- Pour une association *A* -> *B* de type *n-n*, on crée une nouvelle table reprenant les clés de *A* et de *B*, plus éventuellement d'autres descripteurs de l'association (le couple constitué par les clefs de *A* et de *B* est une clef de la nouvelle table)

↳ Table intermédiaire

J.Delamarche (2002-2015)

SGBD-R

La "normalisation"

- La normalisation correspond au processus d'organiser ses données pour :
  - limiter les redondances en divisant une table en plusieurs, et en les reliant entre elles par des clefs primaires et étrangères
  - limiter les incohérences des données
  - limiter les processus de mise à jour
- L'objectif est d'isoler les données afin que l'ajout, l'effacement ou la modification d'un champ puisse se faire sur une seule table, et se propager au reste de la base par le biais des relations
  - la normalisation introduit en tout 8 formes "normales"

10

J.Delamarche (2002-2015)

SGBD-R

Inconvénients de la normalisation

- Les inconvénients sont :
  - des temps d'accès potentiellement plus longs si les requêtes sont trop complexes
  - une plus grande fragilité des données étant donné la non redondance
  - un manque de flexibilité au niveau de l'utilisation de l'espace disque
- La normalisation des modèles de données a été popularisée principalement par la méthode Merise
- La principale limite de la normalisation est que les données doivent se trouver dans une même base de données (dans un seul schéma).

11

SGBD-R

Les formes "normales"

Forme	Implication
1NF (First Normal Form)	Chaque table doit avoir une clef primaire. Il faut éliminer les colonnes en doublon. Chaque ligne doit contenir une seule valeur.
2NF (Second Normal Form)	Si une table dispose d'une clef, toutes les propriétés doivent en dépendre : les données que l'on retrouve dans plusieurs lignes doivent être sorties dans une table séparée
3NF (Third Normal Form)	Les données d'une table ne dépendent que de la clef, et pas d'une autre colonne de la table : toute colonne dépend non seulement de la clef, mais également d'une autre colonne qui doit être sortie dans sa propre table
BCNF (Boyce-Cod Normal Form)	Aucune propriété faisant partie de la clef d'une relation 3NF, ne doit dépendre d'une propriété ne faisant pas partie de la clef primaire. Il faut donc créer une nouvelle table dont la clef primaire sera la propriété dont provient la relation
4NF (Fourth Normal Form)	Il ne doit y avoir qu'une et une seule dépendance multivaluée élémentaire
5NF (Fifth Normal Form)	Toute dépendance de jointure est impliquée par des clefs candidates de la relation
Domain/Key Normal Form	Il ne doit exister aucune contrainte sinon celles de domaine ou de clef
6NF (Sixth Normal Form)	La sixième forme, encore récente, demande à prendre en compte la dimension temporelle

12

SGBD-R

Choix d'une forme normale

- Pour des petites bases de données, se limiter à la troisième forme normale est généralement une des meilleures solutions d'un point de vue architecture de base de données
- Pour des bases de données plus importantes, cela n'est pas toujours le cas. Il s'agit de choisir l'équilibre entre deux options :
  - la génération dynamique des données via les jointures entre tables
  - l'utilisation statique de données correctement mises à jour

13

J.Delamarche (2002-2015)

SGBD-R

Dénormalisation

- Il convient d'être prudent lorsqu'on renonce à la forme normale :
  - il n'est pas garanti qu'une forme dénormalisée améliore les temps d'accès
  - en effet, la redondance peut entrainer une explosion des volumes de données qui peuvent écrouler les performances ou saturer les disques durs

14

J.Delamarche (2002-2015)

SGBD-R

Exemple de normalisation 1NF

- Soit la table suivante :

Produit	Fournisseur
téléviseur	VIDEO SA, HITEK LTD

- Selon la 1NF, chaque attribut doit contenir une valeur atomique
  - il faut donc décomposer la 2ème colonne

Produit	Fournisseur
téléviseur	VIDEO SA
téléviseur	HITEK LTD

15

J.Delamarche (2002-2015)

SGBD-R

Exemple de normalisation 2NF

- Soit la table suivante :

Produit	Fournisseur	Adresse fournisseur
téléviseur	VIDEO SA	13 rue du cherche-midi
écran plat	VIDEO SA	13 rue du cherche-midi
téléviseur	HITEK LTD	25 Bond Street

redondance => màj difficile

- Chaque attribut qui n'appartient pas à la clé ne doit pas dépendre uniquement d'une partie de la clé

Produit	Fournisseur
téléviseur	VIDEO SA
téléviseur	HITEK LTD
écran plat	VIDEO SA

Fournisseur	Adresse fournisseur
VIDEO SA	13 rue du cherche-midi
HITEK LTD	25 Bond Street

16

J.Delamarche (2002-2015)

SGBD-R

Exemple de normalisation 3NF

- Soit la table suivante :

Fournisseur	Adresse fournisseur	Ville	Pays
VIDEO SA	13 rue du cherche-midi	PARIS	FRANCE
HITEK LTD	25 Bond Street	LONDON	ENGLAND

- Les attributs qui ne font pas partie de la clé ne dépendent pas d'attributs ne faisant pas non plus partie de la clé

Fournisseur	Adresse fournisseur	Ville
VIDEO SA	13 rue du cherche-midi	PARIS
HITEK LTD	25 Bond Street	LONDON

Ville	Pays
PARIS	FRANCE
LONDON	ENGLAND

17

J.Delamarche (2002-2015)

SGBD-R

Erreurs de normalisation en 1NF

- tout attribut contient une valeur atomique

CLIENT_ID*	...
Dupont Paris	
Durand Marseille	

L'attribut CLIENT_ID est composé de 2 attributs atomiques.

CLIENT_ID*	NOM
1	Gérard Dupont
2	Léon Durand

L'attribut NOM est composé de 2 attributs atomiques.

- tous les attributs sont non répétitifs

PRODUIT_ID*	DESCRIPTION	FOURNISSEURS
1	Téléviseur	Sony, Sharp, LG

L'attribut FOURNISSEURS est une liste.

- tous les attributs sont constants dans le temps.

CLIENT_ID*	NOM	PRENOM	AGE
1	Dupont	Gérard	35

L'attribut AGE n'est pas constant dans le temps.

18

J.Delamarche (2002-2015)

Trop de normalisation augmente le temps d'accès.

SGBD-R

Erreurs de normalisation en 2NF/3NF

2FN - deuxième forme normale

- tous les attributs non-clés sont totalement dépendants fonctionnellement de la totalité de la clé primaire.

COMMANDE_ID*	ARTICLE_ID*	DESCRIPTION_ARTICLE
1	15	TV haute définition avec amplificateur dolby 5.1

L'attribut DESCRIPTION_ARTICLE ne dépend que d'une partie de la clef primaire.

3FN - troisième forme normale

- tout attribut n'appartenant pas à une clé ne dépend pas d'un attribut non clé

COMMANDE_ID*	CLIENT_ID	NOM_CLIENT
1	1	Durand

L'attribut NOM_CLIENT dépend de CLIENT_ID.

19

J.Delamarche (2002-2015)

SGBD-R

Transactions

- *Transaction* = groupe de requêtes SQL traitées de manière atomique (tout ou rien)
- Un SGBD transactionnel doit supporter les critères suivants (*ACID transactions*):
  - Atomicité: tout le groupe ou rien
  - Consistence: en cas de crash pendant la transaction, la base est toujours dans un état cohérent
  - Isolation: une transaction ne "voit" pas les autres transactions qui ne sont pas achevées
  - Durabilité: une fois achevée, la transaction est permanente. Un crash du serveur ne doit plus perdre les données (solution matérielle)

20

J.Delamarche (2002-2015)

SQL

Le besoin d'un langage

- Pour décrire les informations stockées dans la base il faut un
  - *DATA DEFINITION LANGUAGE* (DDL)
- Pour manipuler les données, il faut un
  - *DATA MANIPULATION LANGUAGE* (DML)
- Avec les SGBD-R, le langage commun est **SQL**

SQL

Exemple de DDL

- Gestion de table :

```
CREATE TABLE tbl (
    nom VARCHAR(32),
    prenom CHAR(32) NOT NULL DEFAULT '',
    naissance DATE NOT NULL,
);

ALTER TABLE tbl ADD COLUMN enfants INT NOT NULL;
ALTER TABLE tbl MODIFY nom VARCHAR(40);

DROP TABLE tbl;
```

SQL

Exemple de DML

- Insertion, mise à jour et destruction de données :

```
INSERT INTO tbl (nom, prenom, naissance, enfants)
  VALUES
('Dupond', 'Jean', '1980-01-01', 2),
('Durand', 'Jacques', '1970-02-07', 0);

UPDATE tbl SET enfants = 1 WHERE nom = 'Durand';

DELETE FROM tbl WHERE nom = 'Dupond';
```

  - le langage SQL se veut "naturel" (pour un anglophone !)

SQL

Sélection des enregistrements

- On utilise l'instruction **`SELECT`** dont la syntaxe de base est :

```
SELECT liste_selection      // colonnes à sélectionner
FROM   liste_tables         // où trouver les lignes
WHERE  critere_principal    // conditions à satisfaire
GROUP BY colonnes           // comment grouper les rés?
ORDER BY colonnes           // comment trier les rés?
HAVING critere_secondaire   // autre condition
LIMIT  nombre               // limite sur les résultats
```

ü le changé du dernier paramètre !

Syntaxe SQL

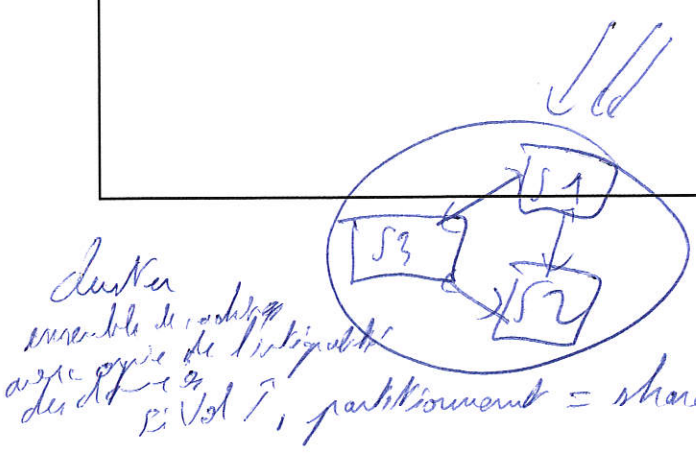
Les Jointures

- Lorsque les requêtes mettent en oeuvre plusieurs tables, on parle de "jointures"
- Jointure complète (produit cartésien ou jointure croisée):
  - `SELECT t1.*,t2.* FROM t1,t2`
- Equi-jointure (test d'égalité entre 2 colonnes):
  - `SELECT t1.*,t2.* FROM t1,t2 WHERE t1.id = t2.id;`
- La jointure "à gauche": produit le contenu de la table de gauche ainsi que les rangées de la table de droite et NULL sinon:
  - `SELECT t1.*,t2.* FROM t1 LEFT JOIN t2 ON t1.id = t2.id`

SGBD-R

La vie d'une base SQL

- La charge augmente :
  - vient la réplication avec lecture sur les réplicats
  - optimisation des requêtes
  - mise en place de cache
  - mise en place de matériel plus performant
  - puis de cluster
  - puis du sharding
  - puis modification du schéma !



No-SQL

Le mouvement NoSQL

- NoSQL = Not-Only-SQL ou No-SQL ?
- Concept initié début 2000 → Système de Stockage
  - BigTable de Google et S3 de Amazon
- Principaux objectifs :
  - résoudre le problème de la montée en charge à faible coût des acteurs majeurs du Web
  - résoudre le problème du Big-Data

ACID = Atomicity Courtenay Isolation Durability.

NoSQL

Le théorème CAP

- Les SGBD-R doivent être ACID → ne sont plus à appliquer
  - a-t-on toujours besoin de cette propriété ?
- Le "BigData" nécessite une architecture distribuée
- Or Eric Brewer a prouvé (puis cela a été démontré) qu'un système distribué ne peut supporter plus de 2 des 3 propriétés suivantes :
  - Consistency :
    - le résultat d'une écriture est-il vu pour tous les lecteurs ?
  - Availability :
    - il doit être toujours possible d'accéder aux données
  - Partition Tolerance :
    - possibilité de continuer à fonctionner avec des noeuds absents

29

J.Delamarche (2002-2015)

NoSQL

Le système doit être "agile"

- Performance et dimensionnement aisé
- Simple à déployer et mettre à jour
- Pas de DBA ni d'Admin système
  - le rêve du développeur !
- Le codage avant tout !
- Le modèle des données est souple et modifiable facilement
  - le stockage en base est secondaire

30

J.Delamarche (2002-2015)

NoSQL

Classification des bases No-SQL

- Bases orientées "colonnes" → HBASE - CASSANDRA
- Bases orientées "documents" → MONGODB
- Bases "clés/valeurs"
- Bases orientées "graphes"
- Bases orientées "objets"
- Bases XML → go name.
- autres...

NoSQL

Classification des bases No-SQL (2)

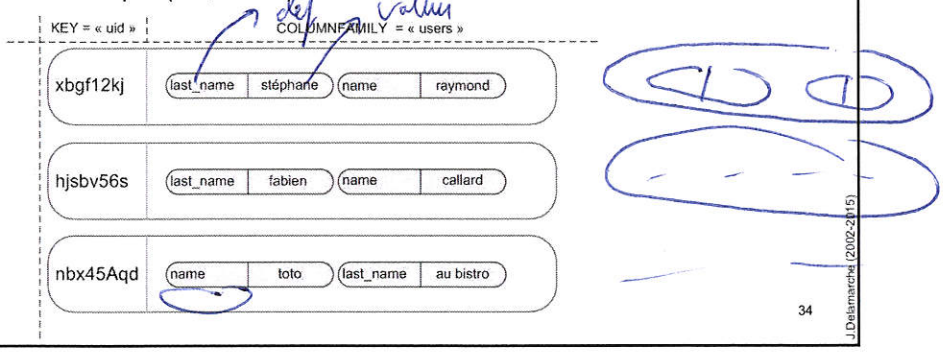
- Liste complète (?) :
  - http://nosql-database.org
- Quels sont les leaders qui vont émergés ?
- Tentative d'unification des langages de requêtages avec UnQL → n'a pas marché !

Base orientée "colonnes" : Cassandra

- Ecrit en Java
- Placé en OpenSource par Facebook en 2008
  - géré par la fondation Apache
- Fonctionnalités :
  - tolérance de pannes
  - scalabilité horizontale :
    - lectures / écritures sur n'importe quel noeud
  - tuning du mode d'écriture (sur un noeud, quorum, tous...)
  - réplication multi-sites
  - langage d'interrogations CQL

Cassandra (suite)

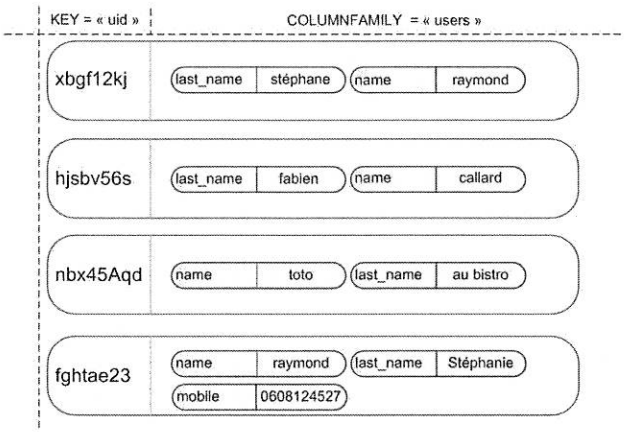
- Structure des données :
  - espace de nommage : *keyspace* (=base)
  - famille de colonnes : *column family* (=table)
    - composée de couples (clés,valeurs)
  - colonne :
    - couple (clé,valeur) avec TIMESTAMP



Cassandra

Cassandra (suite)

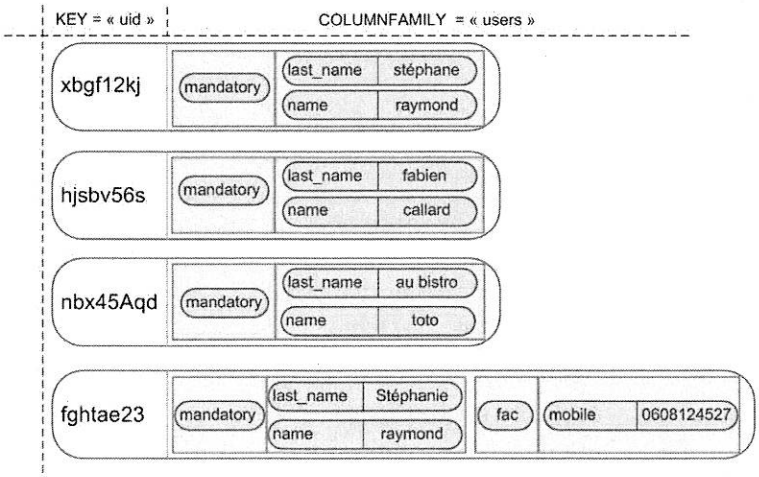
- Le nombre de colonnes peut différer :



Cassandra

Cassandra (suite)

- Possibilité de définir des Super-Columns qui regroupent des colonnes :



=> fait pour stocker des paires clef/valeurs sans type d'information sémantique.
Redis
=> gère des millions de clients simultanément

memcached

Base clé/valeur "volatile" : memcached

- Système de stockage en mémoire de paires (clés/valeurs) de petite taille
- Composé d'un processus serveur *memcached*
- De clients *memcache*
  - le protocole est textuel et simple

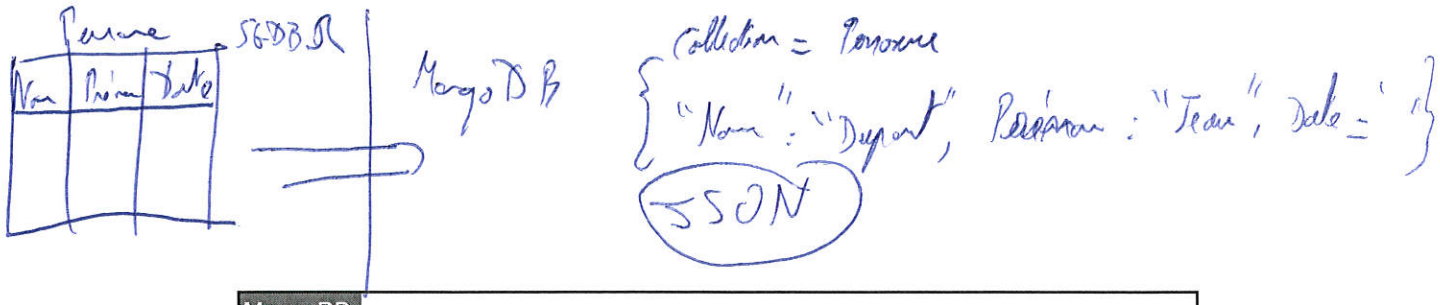
```
$ telnet localhost 11211
stats settings
set var1 0 1000 2
10
STORED
incr var1 2
12
get var1
VALUE var1 0 2
14
END
```


Possible de mettre ce système comme système intermédiaire requêtes

memcached

Memcached (suite)

- Les données possèdent une durée de validité
  - la valeur 0 indique une valeur qui n'expire pas
- L'implémentation `memcached-repcached` permet de mettre en oeuvre de la réplication entre noeuds
- MemcacheDB reprend les mêmes principes mais stocke les données de manière persistante



MongoDB

Base orientée "documents" : MongoDB

- SGBD orienté "document"
  - pas de schéma, pas de relation → pas de jointures
  - plus flexible
- Dimensionnement (*scalability*) plus aisée
  - partitionnement (*sharding*) natif
  - répartition de la charge sur un cluster - réorganisation des documents automatique
- Les fonctions JavaScript remplacent les procédures stockées ⇒ côté client
- Moins d'effort de la part du DBA
- Protocole binaire rapide et efficace

Node.js

39

Autres

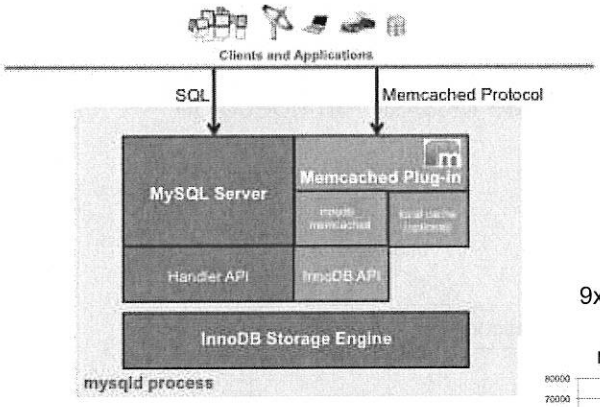
Attitude des acteurs "traditionnels"

- MySQL propose une interface avec le protocole Memcached
- Oracle propose un produit Oracle NoSQL Database !
- Microsoft propose une solution avec son offre de Cloud Computing : SQL Azure (reste du SQL ?)

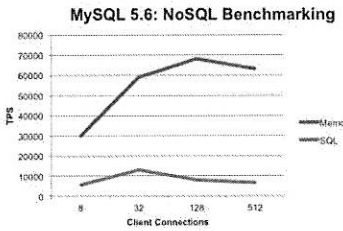
40

MySQL

MySQL et Memcached



9x plus rapide en INSERT



Chaque clé/valeur est stockée comme une ligne d'une table InnoDB
=> supporte le schema-less !

J.Delamarche (2002-2015)

41

Postgres

PostgreSQL 9.3

- Intégration du protocole SQL/MED pour réaliser les écritures sur un serveur distant
  - = description de tables externes
- Bibliothèque d'extensions JavaScript et moteur HStore pour la gestion des paires clés/valeurs
- Opérateurs JSON pour faciliter l'intégration de clients JavaScript
  - support du type JSON pour les colonnes des tables
  - opérateur ->> pour référencer des champs de données JSON :

```
SELECT * FROM products
  WHERE (data->>'in_stock')::integer > 0
  ORDER BY (data->>'price')::decimal;
```

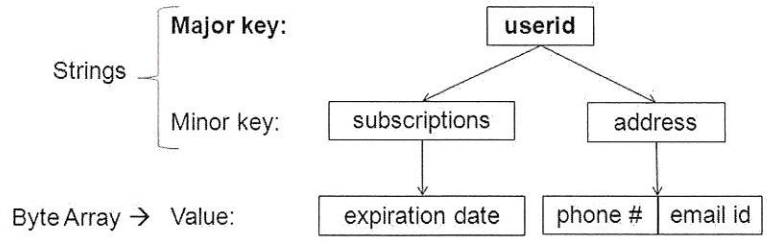
J.Delamarche (2002-2015)

42

Oracle

Oracle NoSQL Database

- Base de données clé/valeur distribuée
  - la clé majeure permet de localiser un noeud de données



43

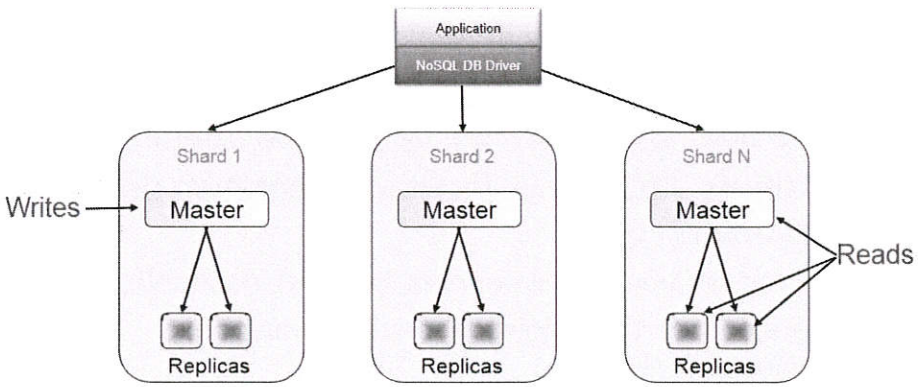
J.Delamarche (2002-2015)

1 fenêtre peut être partagée par plusieurs tâches.

Oracle

Oracle NoSQL Database (2)

- Dimensionnement et réplication :



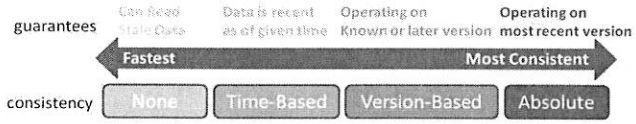
44

J.Delamarche (2002-2015)

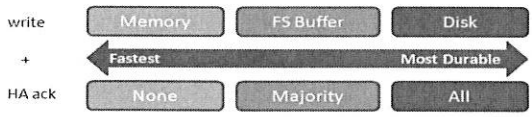
Oracle

Oracle NoSQL Database (3)

- Supporte les transactions ACID
  - différentes politiques de consistance



  - différentes politique de fiabilité en cas de crash



Autres

Autres solutions

- Bases orientées "colonnes" :
  - Hadoop/HBase
  - SimpleDB (Amazon)
- Bases orientées "documents" :
  - MongoDB
  - CouchDB
  - Couchbase Server
- Bases orientées "clés/valeurs" :
  - DynamoDB (Amazon)
    - Voldemort (implémentation OpenSource)
  - Redis
- Bases orientées "graphes" :
  - GraphBase, Neo4J

Big-Data

Qu'est-ce-que le Big-Data ?

- Le coût du stockage baissant, certains métiers collectent des données de plus en plus volumineuses
- Or le traitement de ces données (pout en extraire d'autres données !) ne peut être réalisé dans un temps "acceptable" par une architecture traditionnelle :
  - SGBD-R + machines de traitement
- Il faut déployer des architectures et des algorithmes spécifiques
  - à partir de 10 To de données en-ligne, on parle de Big-Data

47

Big-Data

Solution traditionnelle

- Plus de données implique plus de stockage
- Les périphériques de stockage ont leur performance qui augmentent à coût constant
- Pour traiter des volumes de données importants (supérieur à 1 Eo = 1 million de To), augmenter les perfs des CPU ou les perfs des disques (utiliser des SSD ?) ne suffit pas
- Les goulots d'étranglement sont toujours les I/O disques, voire le réseau

48

Big-Data

Solution "Big-Data"

- Inspirée des travaux de Google, Facebook, Yahoo! ...
- Utilise 3 stratégies complémentaires :
  - la distribution des données
  - la parallélisation des traitements
  - la *commoditization* des infrastructures

Big-Data

Distribution des données

- Si le SAN ne peut héberger les données de manière économique, il faut plusieurs SAN !
- L'ensemble des SAN forme un cluster
- Chaque noeud du cluster reçoit une partie des données
  - partitionnement horizontal (*sharding*)
- Mais les objets peuvent aussi être répartis sur plusieurs noeuds
  - partitionnement vertical
- Pour récupérer un objet, il faut donc identifier les noeuds qui l'hébergent

Big-Data

Parallélisation des traitements

- Les temps de traitement des algorithmes séquentiels sont prohibitifs
- Il faut utiliser le parallélisme
  - mais intelligemment, car il ne faut pas qu'un processus situé sur une machine A requiert les données du noeud B
  - principe de *colocalisation*

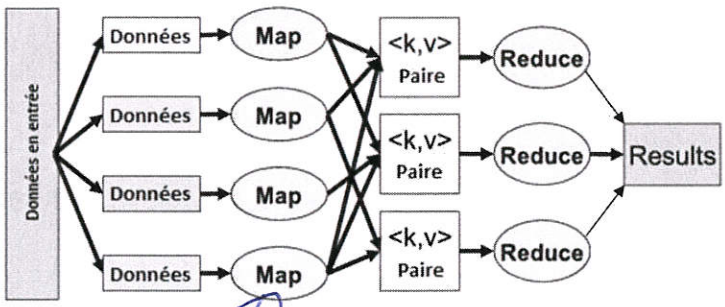
51

J.Delamarche (2002-2015)

Big-Data

L'algorithme map/reduce

- Publié par Google en 2004
- Modèle de programmation parallèle sur une architecture distribuée en cluster
  - Apache Hadoop en est une implémentation
  - les SGBD NoSQL de type "documents" proposent aussi une fonctionnalité similaire



52

J.Delamarche (2002-2015)

Étape de sélection des données.
=> production de paires clé/valeur

Big-Data

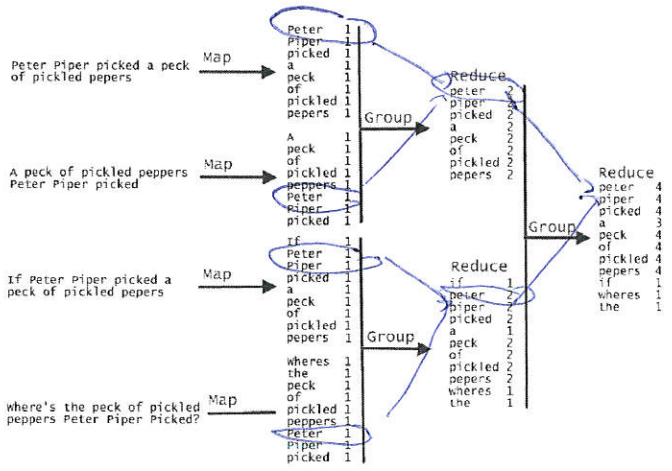
L'algorithme map/reduce (2)

- Input Reader :
  - lit les données stockées et génère des paires clé/valeur transmises aux fonctions de mapping
- Fonctions de mapping :
  - retournent de nouvelles paires clés/valeurs
- Fonctions de partitionnement :
  - attribue les résultats précédents à la fonction de réduction, en fonction des valeurs de clés
- Fonction de comparaison :
  - utilisée pour trier les clés résultant du mapping
- Fonctions de réduction :
  - reçoit une clé et les valeurs et retourne une valeur unique pour la clé
- Output Writer :
  - stocke les résultats sur un système de stockage

Big-Data

Exemple concret

- Compter le nombre d'occurrence de chaque mot dans du texte :



Big-Data

Commoditization de l'infrastructure

- Suit l'idée qu'il est plus économique et plus pratique d'utiliser des serveurs "bas-de-gamme" (*commodity servers*) mais très nombreux que des serveurs "haut-de-gamme" moins nombreux
- Ils servent de noeuds de stockage et de calcul
- Rendent les clusters élastiques
  - il est facile de les changer ou d'en ajouter, d'en supprimer sans gros investissement

Big-Data

Contraintes et défis

- Un système distribué implique :
  - gestion de la cohérence
  - équilibrage de charge
  - tolérance aux pannes
  - distribution multi-datacenters
- Résilience du stockage :
  - les données sont dupliquées entre plusieurs noeuds
    - permet la colocalisation en démarrant les processus sur les noeuds qui disposent des données (le noeud le moins chargé ?)
- Des nouvelles couches applicatives sont nécessaires !

Big-Data

Architecture

- Infrastructure :
  - une topologie de cluster avec forte interconnectivité réseau entre les noeuds
  - souvent, une couche de virtualisation pour ses capacités d'élasticité et provisioning

- Stockage et modélisation des données :
  - besoins décorrélés : le schéma n'est pas dans les données
  - c'est l'application qui donne sens aux données qui sont opaques pour la technologie de stockage

Big-Data

Architecture (2)

- Traitement des données :
  - par l'intermédiaire d'un *pipeline* de traitement
  - peut-être :
    - batch pour des gros volumes
    - interactif pour des unités de volumes faibles avec des temps de réponse garantis
    - événementiel quand on traite les données au fil de l'eau

Big-Data

Architecture (3)

- Etapes possibles du *pipeline* :
  - préparation des données (nettoyage, enrichissement)
  - indexation
    - avec indexation possible par des moteurs de recherche tiers (Lucene, Solr...)
  - extraction de méta-données (multimedia, texte...)
  - construction de modèles statistiques (machine learning)
    - modèles prédictifs
  - création de graphes

= préparation des requêtes pour les agrégations.

Big-Data

Architecture (4)

- Requêtage et visualisation :
  - présentation par tableau de bord
  - outils d'exploration de données
  - mode "push" dynamique
  - reporting
  - import/export
- Outils de management
- Outils de supervision
- Outils de sécurisation

Big-Data

Exemple de workflow "Big Data"

- On stocke des informations brutes (logs d'accès Web, interactions utilisateurs, flux...)
- Les logs bruts sont traités par des processus Map/Reduce et les résultats sont stockés dans des dépôts distribués
- D'autres routines de type Map/Reduce calculent les indicateurs métiers (BI)
- Ces données sont stockées en bases
- L'interface Web du BI affiche les graphes calculés à partir de ces données

61

J.Delamarche (2002-2015)

Conclusion

SQL ou NoSQL ?

- Avantages du NoSQL :
  - généralement OpenSource
  - supporte bien la montée en charge
  - solutions variées
  - souplesse par absence de schéma
  - facile à mettre en oeuvre
- Inconvénients du NoSQL :
  - pas ACID
  - restriction CAP
  - encore immature ? Non
  - pauvreté des outils d'administration ? Oui
  - absence de standard

62

J.Delamarche (2002-2015)

Sommaire

Sommaire

- Prise en main
- MapReduce
- Outils connexes
- Administration

Prise en main

Introduction à Hadoop

- Issu des travaux publiés en 2004 par Google :
  - GFS
  - MapReduce
- Géré par la fondation Apache
- Framework (écrit en Java) qui permet de réaliser des tâches en parallèle sur des données stockées dans un cluster
- Utilisé par Twitter, IBM, eBay, Amazon, Facebook; LinkedIn, Yahoo!, Adobe...
- Existe en deux versions : 1.x et 2.x

↳ opérationnelle depuis 1an.

Hadoop

Composants

- Hadoop Common : le noyau
- HDFS (Hadoop Distributed File System)
- Hadoop YARN (Yet Another Resource Negotiator) (v2) : gestion des ressources du cluster et gestion des jobs
- Hadoop MapReduce : (utilise YARN) gestion des jobs en parallèle
- Des applications de plus haut-niveau :
  - HBase, Zookeeper, Oozie, Pig, Hive...

Hadoop

Types de noeuds

- NameNode :
  - contient les méta-données des fichiers HDFS
- DataNode
  - contient les blocs de données HDFS
- JobTracker
  - processus de gestion du MapReduce activé sur le master node
- TaskTracker
  - processus MapReduce activé sur les slaves nodes
  - reçoivent leur part du travail de la part du JobTracker
- ResourceManager (v2 - YARN) :
  - remplace le JobTracker et le TaskTracker

Modes d'exécution

- Mode standalone :
  - configuré par défaut
  - tous les rôles sont pris par le même noeud par une JVM unique
- Mode pseudo-distribué :
  - chaque rôle est pris par une JVM différente qui s'exécute sur la même machine
- Mode distribué :
  - les rôles sont répartis sur des noeuds différents qui peuvent être spécialisés par rôles

Exemple : mode pseudo-distribué

- Il faut modifier les fichiers de configuration :
  - `conf/core-site.xml`
    - le paramètre `dfs.default.name` doit indiquer la valeur `http://localhost:9000`
  - `conf/hdfs-site.xml`
    - le paramètre `dfs.replication` doit avoir la valeur `1`
  - `conf/mapred-site.xml`
    - le paramètre `mapred.job.tracker` doit avoir la valeur `http://localhost:9001`
- Il faut formater le système de fichiers HDFS :
  - `mkdir /var/lib/hadoop`
  - `chmod 777 /var/lib/hadoop`

→ et pour gérer > 16.

Hadoop

HDFS

- Basé sur Google File System (GFS)
- Supporte des Po de données
- Supporte les pannes des noeuds du cluster
- Rapide - permet l'ajout de noeuds quand les clients augmentent
- S'intègre avec le Hadoop MapReduce :
  - les données sont traitées localement quand c'est possible (évite le "*shuffling*")
- Pas adapté pour les accès directs dans les fichiers (mais pour les accès en "streaming")
- Pas de mise à jour des fichiers

71

Hadoop

Structure de HDFS

- Les fichiers sont découpés en blocs (de 64 Mo)
  - prévu pour gérer "peu" de fichiers très volumineux et qui seront lus séquentiellement
- Les blocs sont associés aléatoirement aux *DataNodes*
- Les méta-données sont stockées dans la mémoire du *NameNode* (unique !)
  - ...mais qui est répliqué !
- Les Clients contactent le *NameNode* puis les *DataNodes* (en parallèle !)

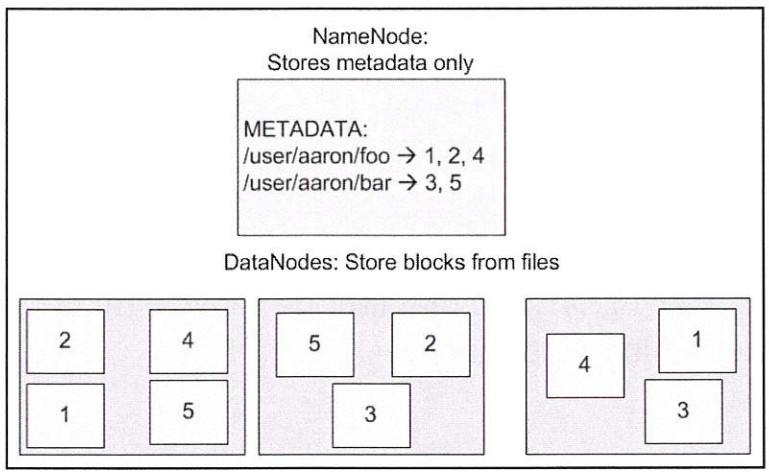
72

Déf de la taille des chunks (64)

Hadoop

Structure de HDFS (2)

- Exemple :



Hadoop

Configuration de HDFS

- Utilise le format XML
- Fichiers de configuration :
  - `conf/hadoop-defaults.xml`
    - contient les valeurs par défaut de tous les paramètres
  - `conf/hadoop-site.xml`
    - contient les valeurs personnalisées
  - les fichiers doivent être identiques sur tous les noeuds du cluster
  - `conf/slaves`
    - liste les *DataNodes* (par leur FQDN)

Hadoop

Paramètres principaux

- `fs.default.name` : URI du *NameNode*
  - ex: `hdfs://monserveur:9000`
- `dfs.data.dir` : chemin de stockage des données dans le FS local
  - ex: `/var/hdfs/data`
- `dfs.name.dir` : chemin de stockage des méta-données
- `dfs.replication` : facteur de réplication de chaque bloc de données
  - ex : `3`

Hadoop

Démarrage et arrêt de HDFS

- Création de l'espace de stockage :

```
$ bin/hadoop hdfs namenode -format
```

- Démarrage de HDFS :

```
$ bin/start-dfs.sh
```

  - démarre les *DataNodes* via SSH
  - lance une interface Web sur le port 50070
- Arrêt de HDFS :

```
$ bin/stop-dfs.sh
```

Hadoop

Utilisation de HDFS

- Afficher le contenu d'un répertoire :

```
$ bin/hadoop|hdfs dfs -ls
$ bin/hadoop|hdfs dfs -ls /
```

- Insertion de données :

```
$ bin/hadoop|hdfs dfs -mkdir /user
$ bin/hadoop|hdfs dfs -put /chemin/fichier
  /user/
$ bin/hadoop|hdfs dfs -ls /user
```

  - la valeur `<r X>` indique le nombre de réplicats

- Affichage et récupération :

```
$ bin/hadoop|hdfs dfs -cat fichier
$ bin/hadoop|hdfs dfs -get fichier /vers/f
```

J.Delamarche (2002-2015)

Hadoop

Administration de HDFS

- Statut global :

```
$ bin/hadoop dfsadmin -report
```

- Statut détaillé :

```
$ bin/hadoop dfsadmin -metasave fichier
```

- Safemode :
  - passage en mode *read-only*

```
$ bin/hadoop dfsadmin -safemode \
                    enter|leave|get|wait
```

- Bonne santé du FS :

```
$ bin/hadoop fsck /
```

- En cas de sortie d'un noeud du cluster (*decommissioning*) :

```
$ bin/hadoop dfsadmin -refreshNodes
```

78

J.Delamarche (2002-2015)

Hadoop

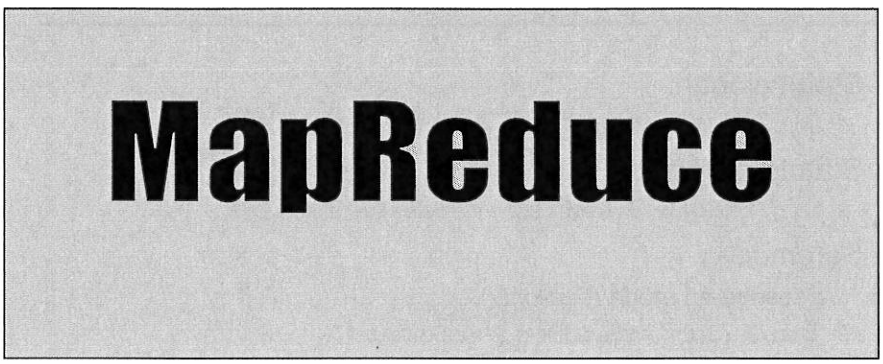
Autres fonctionnalités

- Manipulation de HDFS par des API
- Modèle de sécurité POSIX `-chown` , `-chgrp`, `-chmod`)
  - *superuser* est l'acronyme de l'utilisateur Linux qui a lancé le cluster
  - `dfs.permissions` permet de désactiver les permissions
- Equilibrage des blocs :
  - après ajout d'un noeud, il faut répartir les anciens blocs :

```
$ bin/start-balancer.sh -threshold 5
```

- Copies volumineuses :
  - utilise la commande `distcp` pour le faire via MapReduce
- Procédure (laborieuse) de décommissionnement
- Possibilité de prendre en compte la topologie du réseau et des racks

Hadoop



Exemple de MapReduce

- Pseudo-code de comptage de mots :

```
mapper (filename, file-contents):
  for each word in file-contents:
    emit (word, 1)

reducer (word, values):
  sum = 0
  for each value in values:
    sum = sum + value
  emit (word, sum)
```


API Java - classe Mapper

- Prototype :

```
class Mapper<K1, V1, K2, V2>
{
  void map(K1 key, V1 value, Mapper.Context context)
             throws IOException, InterruptedException
  {..}
}
```

  - `K1` et `V1` représentent la clé et sa valeur, passées en entrée de la fonction
  - `K2` et `V2` représentent la nouvelle clé et la nouvelle valeur en sortie
  - `context` permet à la fonction d'accéder à un environnement et d'y placer le résultat du mapping

MapReduce

API Java - classe Reducer

- Prototype :

```
public class Reducer<K2, V2, K3, V3>
{
  void reduce(K2 key, Iterable<V2> values,
                      Reducer.Context context)
                  throws IOException, InterruptedException
  {..}
}
```

  - `K2` et `V2` représentent la clé et la liste de valeurs, passées en entrée de la fonction
  - `K3` et `V3` représentent la nouvelle clé et la nouvelle valeur en sortie
  - `context` permet à la fonction d'accéder à un environnement et d'y placer le résultat de la réduction

MapReduce

API Java - classe Mapper & Reducer

- Autres méthodes (facultatives) :
  - `void setup(Context context)`
    - appelée avant le 1er appel à `map()`
    - ne fait rien par défaut
  - `void cleanup(Context context)`
    - appelée après le dernier appel à `map()`
    - ne fait rien par défaut
  - `void run(Context context)`
    - appelle `setup()` puis itère sur `map()` puis appelle `cleanup()`

MapReduce

API Java - classe Driver

- Le Driver permet de communiquer avec le framework Hadoop
- C'est la fonction "`main()`"
  - il spécifie les éléments de configuration qui permettent de lancer le job MapReduce :
    - quel Mapper lancer ?
    - quel Reducer lancer ?
    - où sont les données en entrée ?
    - où placer les résultats ?

MapReduce

Autres composants de l'API

- Interfaces `Writable` et `WritableCompare`
  - pour permettre la sérialisation des données sur le réseau ou sur disque
- Wrappers pour les types de base :
  - `IntWritable`, `BooleanWritable`, `ByteWritable`, `DoubleWritable`, `FloatWritable`, `DoubleWritable`, `ArrayWritable`, `MapWritable`...

MapReduce

Autres composants de l'API (2)

- Classes pour lire et écrire les données :
  - `InputFormat` qui définit la méthode `split()` et la méthode `createRecordReader()` qui génère une liste de clés/valeurs à partir du résultat de `split()`
    - classes dérivées :
      - `FileInputFormat`, `TextInputFormat`, `SequenceFileInputFormat` (fichiers binaires)
      - `LineRecordReader`, `SequenceFileRecordReader`
  - `OutputFormat` et `RecordWriter` sont similaires pour l'écriture
    - classes dérivées :
      - `FileOutputFormat`, `NullOutputFormat`, `SequenceFileOutputFormat`, `TextOutputFormat`

MapReduce

Exemple d'implémentation en Java

```
public static class MapClass extends MapReduceBase
  implements Mapper<LongWritable, Text, Text, IntWritable> {

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value,
                  OutputCollector<Text, IntWritable> output,
                  Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      output.collect(word, one);
    }
  }
}

/**
 * A reducer class that just emits the sum of the input values.
 */
public static class Reduce extends MapReduceBase
  implements Reducer<Text, IntWritable, Text, IntWritable> {

  public void reduce(Text key, Iterator<IntWritable> values,
                     OutputCollector<Text, IntWritable> output,
                     Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

MapReduce

Démarrage du job (driver)

```
public void run(String inputPath, String outputPath) throws Exception {
  JobConf conf = new JobConf(WordCount.class);
  conf.setJobName("wordcount");

  // the keys are words (strings)
  conf.setOutputKeyClass(Text.class);
  // the values are counts (ints)
  conf.setOutputValueClass(IntWritable.class);

  conf.setMapperClass(MapClass.class);
  conf.setReducerClass(Reduce.class);

  FileInputFormat.addInputPath(conf, new Path(inputPath));
  FileOutputFormat.setOutputPath(conf, new Path(outputPath));

  JobClient.runJob(conf);
}
```


MapReduce

Autres modèles de programmation

- Hadoop Pipes
  - propose une interface en C++ native
- Hadoop Streaming
  - permet d'invoquer des processus externes via STDIN et STDOUT pour réaliser les opérations de Map/Reduce
  - utile pour écrire dans un autre langage que Java !

MapReduce

MapReduce en Python

- "WordCount" en Python :
  - la fonction de mapping lit les lignes sur STDIN, les découpe en mots et écrit le résultat sur STDOUT

```
#!/usr/bin/python
import sys

for line in sys.stdin:
    line = line.strip()
    words = line.split()

    for word in words:
        print '%s\t%s' % (word, "1")
```


MapReduce

MapReduce en Python (2)

- "WordCount" en Python :
  - la fonction de réduction lit les lignes sur STDIN et compte les occurrences des mots puis écrit le résultat sur STDOUT

```
#!/usr/bin/env python
import sys

# maps words to their counts
word2count = {}

for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    count = int(count)
    try:
        word2count[word] += count
    except:
        word2count[word] = count

for word in word2count.keys():
    print '%s\t%s'% ( word, word2count[word] )
```

MapReduce

MapReduce en Python (3)

- Lancement du traitement sur le cluster :

```
$ hadoop jar /home/hadoop/hadoop/contrib/streaming/hadoop-0.19.2-
streaming.jar -file ./mapper.py -mapper ./mapper.py -file ./reducer.py
-reducer ./reducer.py -input dft -output dft-output
```

- Mais on peut aussi passer des paramètres supplémentaires :

```
$ hadoop dfs -rmr dft-output
$ hadoop jar /home/hadoop/hadoop/contrib/streaming/hadoop-0.19.2-
streaming.jar -jobconf mapred.reduce.tasks=16 -file ./mapper.py -
mapper ./mapper.py -file ./reducer.py -reducer ./reducer.py -input dft
-output dft-output
```


Hadoop v2

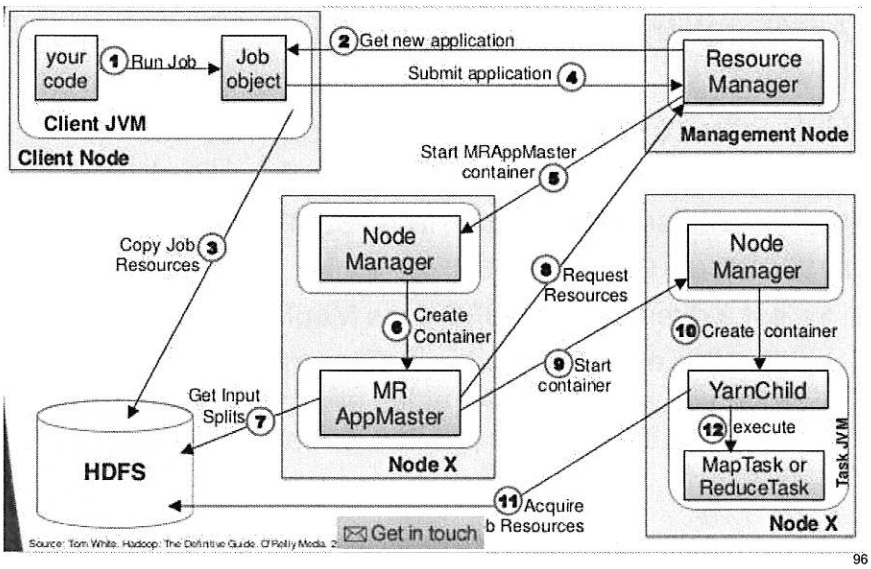
Nouveautés de la v2

- Amélioration des performances (sur HBase et Hive)
- Nouveau composant YARN qui permet de déployer un cluster Hadoop sans MapReduce qui est utilisé en v1 pour la gestion des ressources et des jobs
  - c'est à dire faire du HDFS sans MapReduce
  - nouvelles applications supportées comme le requêtage en temps-réel et les applications de traitement de données par flux (streaming data)

MapReduce avec YARN

- Un Client soumet un Job MapReduce
- Le Resource Manager contrôle l'utilisation des ressources du Cluster Hadoop
- Chaque noeud exécute un Node Manager qui crée le container d'exécution et surveille l'utilisation des ressources
- Le MapReduce Application Master coordonne et gère les jobs MapReduce, négocie avec le RM le scheduling des tâches qui seront démarrées par les NM
- HDFS permet le partage des ressources entre tous les composants de YARN

Diagramme d'interaction



Détail des interactions

1. Client submits MapReduce job by interacting with Job objects; Client runs in it's own JVM
2. Job's code interacts with Resource Manager to acquire application meta-data, such as application id
3. Job's code moves all the job related resources to HDFS to make them available for the rest of the job
4. Job's code submits the application to Resource Manager
5. Resource Manager chooses a Node Manager with available resources and requests a container for MRAppMaster
6. Node Manager allocates container for MRAppMaster; MRAppMaster will execute and coordinate MapReduce job
7. MRAppMaster grabs required resource from HDFS, such as Input Splits; these resources were copied there in step 3

8. MRAppMaster negotiates with Resource Manager for available resources; Resource Manager will select Node Manager that has the most resources
9. MRAppMaster tells selected NodeManager to start Map and Reduce tasks
10. NodeManager creates YarnChild containers that will coordinate and run tasks
11. YarnChild acquires job resources from HDFS that will be required to execute Map and Reduce tasks
12. YarnChild executes Map and Reduce tasks

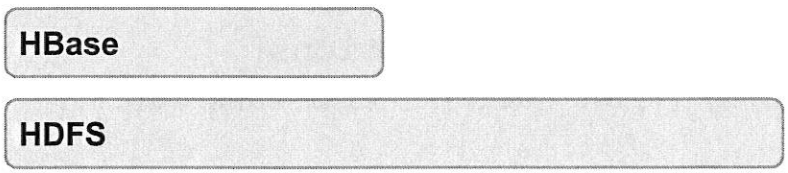
Commande yarn

- `yarn commande [options]`
  - `resourcemanager`
    - lance le Resource Manager
  - `nodemanager`
    - lance un Node Manager sur chaque noeud
  - `jar jarFile [mainClass] args ...`
    - exécute le code Java d'une archive jar
  - `node [-list] [-status Node-Id]`
    - affiche la liste ou l'état d'un noeud
- `mapred commande [options]`
  - `job [-list [all]] [-kill-task id]....`

Outils connexes

Architecture globale

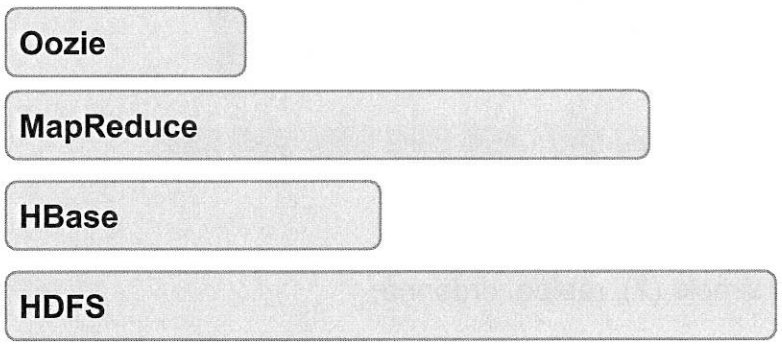
- Pour construire une application, il faut un système de fichiers : **HDFS**
  - qui sera supporté par ext3/4, btrfs...
- Pour stocker et gérer des paires clés/valeurs, il faut une interface de plus haut-niveau : **HBase**



Composants

Architecture globale (2)

- Pour développer des applications parallèles, il faut un framework : le **MapReduce**
- Souvent des jobs multiples sont nécessaires. Il faut un outil de workflow : **Oozie**



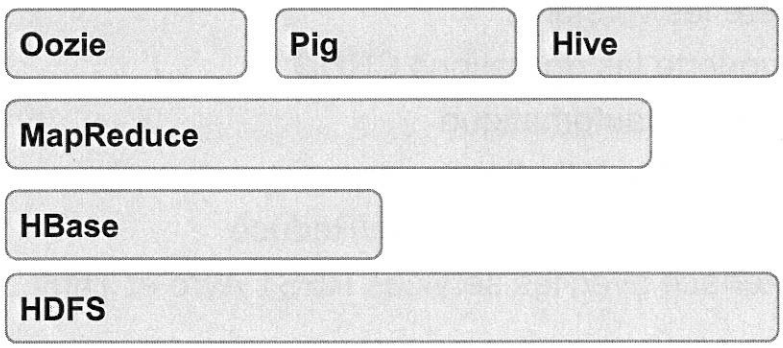
101

J.Delamarche (2002-2015)

Composants

Architecture globale (3)

- Pour les analystes et les scientifiques, il faut un langage de scripting pour le workflow : **Pig**
- Pour les DBA, un langage "à la SQL" est préférable : **Hive**



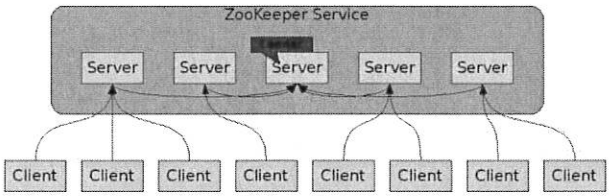
102

J.Delamarche (2002-2015)

Zookeeper

Apache Zookeeper

- Offre des services de coordination (sous forme d'API) pour des applications distribuées
  - gestion de quorum, élection, gestion de groupes



- Propriétés :
  - fonctionnalité répliquée
  - simple (?), rapide, ordonné

HBase

Apache HBase

- Système de stockage de données distribué
- "Orienté colonne"
  - supporte des milliards de lignes et des millions de colonnes
- Gère les versions
- Supporte les opérations CRUD
- Sharding automatique
- Fail-over automatique
- Intégré au framework Map/Reduce
- Interfacé avec les services REST Avro et Thrift

HBase

Quand utiliser HBase ?

- Cas bien connus :
  - des volumes de données importants
  - un nombre très important de clients et de requêtes par clients
- Bonnes performances pour les SELECT aléatoires ou par scan par intervalle de clés
- Bon support pour les schémas variables
  - où les lignes ont des structures différentes
  - où les lignes ont de nombreuses colonnes dont la majorité est NULL

HBase

Quand ne pas utiliser HBase ?

- Pour les applications transactionnelles
- Pour les analyses relationnelles
  - ce qui utilise du GROUP BY, JOIN, LIKE...
- Pour les recherches FULL-TEXT
  - même si des projets essayent de combler cette lacune :
    - HBasene, Lily

HBase

Modèle de données

- Les données sont stockées dans des tables
- Les tables contiennent des lignes
  - les lignes sont référencées par une clé unique
  - les clés sont de type quelconque : chaine, entier, toute structure sérialisable
- Les colonnes sont regroupées en famille de colonnes
- Les valeurs sont stockées dans des cellules
  - les cellules sont identifiées par leurs numéros :
    - de ligne, de familles de colonnes, puis de colonnes
  - les valeurs stockées sont de type quelconque

HBase

Familles de colonnes

- Les lignes sont composées de familles de colonnes
  - syntaxe : `nom_famille:nom_colonne`
- On peut associer des options aux familles :
  - compression
  - localisation en mémoire
  - localisation du stockage
- Les familles sont créées à la création de la table
  - limiter les modifications sur les familles
  - normalement peu de familles
  - mais les colonnes peuvent être créées dynamiquement et peuvent être très nombreuses !

HBase

Gestion des versions

- Les valeurs des cellules sont "versionnées"
  - par défaut, HBase conserve les 3 dernières
  - elles sont classées de manière décroissante
  - l'estampille peut être donnée par le Client

HBase

Structure d'une cellule

- Pour accéder à une valeur de cellule, il faut donner :
  - le nom de la table
  - la clé de la ligne
  - la famille
  - la colonne
  - l'estampille
- Ce qui peut se traduire par :

```
SortedMap< RowKey, List<
   SortedMap<
       Column, List<Value, Timestamp>
   >
>>
```

HBase

Exemple de familles de colonnes

Row Key	Time stamp	Name Family		Address Family	
		first_name	last_name	number	address
row1	t1	**Bob**	**Smith**		
	t5			10	First Lane
	t10			30	Other Lane
	t15			**7**	**Last Street**
row2	t20	**Mary**	Tompson		
	t22			**77**	**One Street**
	t30		**Thompson**		

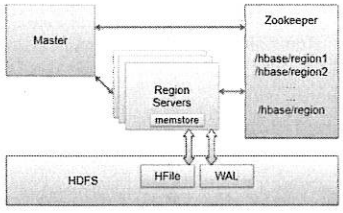
111

J.Delamarche (2002-2015)

HBase

Architecture de HBase

- Les lignes sont regroupées en "Régions"
  - une région est stockée sur un "*Region Server*"
  - un *Region Server* stocke plusieurs régions
  - une région == un shard
- Le Master Server gère les Region Servers
- HBase stocke ses données sur HDFS
- HBase utilise Zookeeper pour coordonner la distribution des données



112

J.Delamarche (2002-2015)

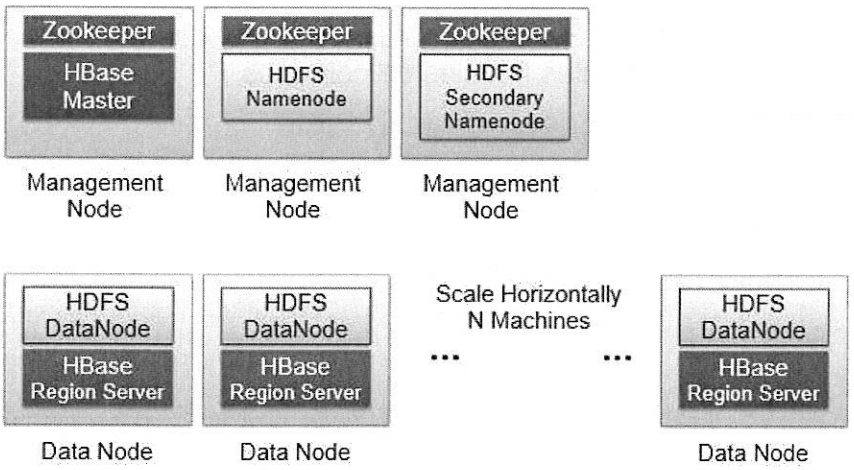
HBase

Stockage des données

- Dans des HFiles usuellement stockés sur HDFS
  - HFile = map clé/valeur
  - chaque famille de colonnes est dans son fichier
    - les colonnes vides ne sont pas stockées
- Les données ajoutées sont écrites dans un log (Write Ahead Log - WAL) et en mémoire
  - quand la mémoire est pleine, les données sont écrites dans le HFile et le WAL est vidé
    - HDFS ne supporte pas les update : les HFiles sont immuables => destruction de clé impossible => utilise des marqueurs de destruction
- Les données sont compactées régulièrement pour réduire le nombre de HFiles et assurer le bon Load-Balancing

HBase

Déploiement de HBase



HBase

Gestion de HBase

- Via l'interface Web du Master Server
  - à l'écoute sur le port 60010
- Via l'interface Web des Region Servers
  - à l'écoute sur le port 60030
- Via le shell HBASE :
  - lancement :

```
$ hbase shell
main> status
main> list
main> exit
```


HBase

Exemples d'opérations

- Création de table
  - avec définition des familles de colonnes
- Insertion de données dans la table
- Accès aux données de la table
  - count, get et parcourt
- Modifier les données
- Supprimer des données
- Supprimer la table

HBase

Création de table

- Soit la table `'Blog'` comprenant deux familles de colonnes :
  - `'info'` : contient 3 colonnes : `'title'`, `'author'`, `'date'`
  - `'content'` : contient 1 colonne : `'post'`
- Commande de création :
  - `hbase> create 'Blog', {NAME=>'info'}, {NAME=>'content'}`
- Différentes options de création :
  - `hbase> create 't1', {NAME => 'f1', VERSIONS => 5}`
  - `hbase> create 't1', {NAME => 'f1', VERSIONS => 1, TTL => 2592000, BLOCKCACHE => true}`
  - `hbase> create 't1', {NAME => 'f1'}, {NAME => 'f2'}, {NAME => 'f3'}`
  - `hbase> create 't1', 'f1', 'f2', 'f3`

HBase

Insertion de données

- Syntaxe :
  - `hbase> put 'table', 'row_id', 'family:column', 'value'`
    - c'est à dire une instruction par cellule
- Exemples :
  - `hbase> put 'Blog', 'Matt-001', 'info:title', 'Elephant'`
  - `hbase> put 'Blog', 'Matt-001', 'info:author', 'Matt'`
  - `hbase> put 'Blog', 'Matt-001', 'info:date', '2009.05.06'`
  - `hbase> put 'Blog', 'Matt-001', 'content:post', 'Do elephants like monkeys?'`

HBase

Comptage des lignes

- Pour obtenir le nombre de lignes :
  - `hbase> count 'Blog'`
    - scanne la table entièrement
    - job MapReduce possible :

      `$ yarn jar hbase.jar rowcount`
- Possibilité d'afficher le comptage en cours :
  - `hbase> count 'Blog', {INTERVAL=>5}`

HBase

Lecture des données

- Syntaxe :
  - `hbase> get 'table', 'row_id' [,options]`
    - en options : `timestamp`, `time-range`, `versions`
- Exemples :
  - `hbase> get 'Blog', 'Matt-001'`
  - `hbase> get 'Blog', 'Matt-001', {COLUMN=>['info:author', 'info:date']}`
  - `hbase> get 'Blog', 'Matt-001', {TIMERANGE=>[ts1, ts2]}`
  - `hbase> get 'Blog', 'Matt-001', {COLUMN=>'info:author', TIMESTAMP=>ts1}`
  - `hbase> get 'Blog', 'Matt-001', {VERSIONS=>4}`

HBase

Scan de tables

- Syntaxe :
  - `hbase> scan 'table' [, options]`
- Exemples :
  - `hbase> scan 'Blog', {LIMIT=>1}`
  - `hbase> scan 'Blog', {STARTROW=>'val1', STOPROW=>'val2'}`
    - `STARTROW` est incluse, `STOPROW` est exclue
  - `hbase> scan 'Blog', {COLUMNS=>['c1','c2']}`
  - `hbase> scan 'Blog', {TIMERANGE=>[ts1, ts2]}`
  - `hbase> scan 'Blog', {FILTER=> org.apache.hadoop.hbase.filter.ColumnPaginationFilter.new(1, 0)}`

HBase

Modifier les données

- La commande `put` insère une ligne si le `row_id` n'existe pas et "met à jour" les données si le `row_id` existe
  - en fait, la valeur de la cellule est ajoutée est versionnée !
  - le nombre de versions est 3 par défaut, mais peut être modifié lors du "`create table`"
- Par défaut, la commande `get` retourne la dernière valeur

HBase

Destruction de données

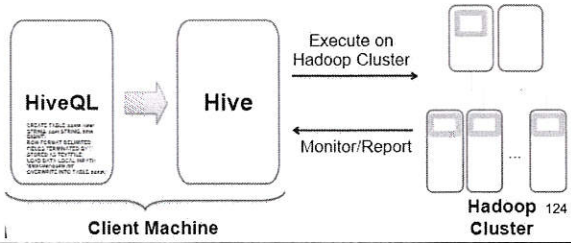
- Destruction de lignes :
  - `hbase> delete 'table', 'rowId', 'column' [,timestamp]`
    - détruit toutes les versions de la cellule
    - avec un `timestamp`, détruit toutes les valeurs antérieures
- Destruction de tables :
  - `hbase> disable 'table'`
    - met la table *offline* pour permettre la modification du schéma
  - `hbase> drop 'table'`
    - prend du temps si la table est volumineuse

Pas de temps réel.

Hive

Apache Hive

- Langage de haut-niveau pour gérer et analyser les données stockées sur HDFS
- Utilise une syntaxe "à la SQL" : **HiveQL**
- Le noyau runtime traduit les requêtes de HiveQL en jobs MapReduce
- Conçu pour les gros volumes
  - pas pour des réponses en temps-réel



Hive

Exemple d'utilisation

- A l'aide du shell :
  - `# hive`
  - `hive> CREATE TABLE posts (users STRING, post STRING, time BIGINT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;`
    - supporte le partionnement
  - `hive> show tables;`
  - `hive> describe posts;`
- Insertion de données à partir d'un fichier texte :
  - `hive> LOAD DATA LOCAL INPATH '/path/data.txt' OVERWRITE INTO TABLE posts;`

Hive

Exemple d'utilisation (2)

- Requêtage :
  - `hive> SELECT COUNT(1) FROM posts;`
    - lance un job MapReduce qui va prendre plusieurs secondes
  - `hive> SELECT * FROM posts WHERE user = "user1" LIMIT 2;`
    - supporte les jointures `INNER`, `LEFT`, `RIGHT`, `OUTER`
- Suppression de la table :
  - `hive> DROP TABLE posts;`

Hadoop

Apache Mahout

- Librairie pour créer des algorithmes de type "machine learning" au-dessus de Hadoop
- Cas d'utilisations :
  - recommendation mining
  - clustering
  - classification
  - frequent item set mining

Hadoop

Apache Pig

- Langage de scripting de haut-niveau qui permet d'effectuer des analyses de données
- Les scripts peuvent être embarqués dans d'autres applications
- Les scripts peuvent invoquer des fonctions utilisateurs codées dans d'autres langages (JRuby, Jython, Java)
- Les scripts sont transformés en opérations MapReduce qui s'exécutent sur le Cluster

Hadoop

Apache Hue

- Interface Web pour :
  - naviguer dans les fichiers de HDFS
  - consulter et gérer les Jobs Map/Reduce et YARN
  - naviguer dans HBase
  - poser des questions à Hive, Pig, Impala et Sqoop2
  - créer et gérer des Workflows pour Oozie
  - naviguer dans Zookeeper

Hadoop

Apache Oozie

- Gestionnaire de jobs
  - de type MapReduce, Streaming ou Pig
- Composants :
  - Oozie Workflow :
    - graphes acycliques (DAG - Directed Acyclical Graph) qui définissent les actions
  - Oozie Coordinator :
    - lance les Workflow selon la présence d'évènements (temps, disponibilité de données...)
- Les workflows sont décrits en XML

Apache Sqoop (Cloudera)

- Outil de migration de données entre Hadoop/HDFS et des datastores comme des bases de données relationnelles
  - fonctionne en mode ligne de commande
- Exemples :

```
$ sqoop import --connect jdbc:mysql://10.0.0.100/hadooptest
        --username hadoopuser --password password --table employees
$ hadoop fs -ls employees
Found 4 items
-rw-r--r-- 3 hadoop grp 0 2012-05-21 04:10 /user/hadoop/employees/_SUCCESS
drwxr-xr-x - hadoop grp 0 2012-05-21 04:10 /user/hadoop/employees/_logs
-rw-r--r-- 3 … /user/hadoop/employees/part-m-00000
-rw-r--r-- 3 … /user/hadoop/employees/part-m-00001
$ sqoop import --connect jdbc:mysql://10.0.0.100/hadooptest --username
   hadoopuser -P --table employees --hive-import --hive-table employees
```


Apache Flume

- Outil de collecte de logs dans un environnement distribué
- Architecture flexible, robuste, tolérante aux pannes qui génère des flux de données depuis des "sources" vers Hadoop
- Exemples de sources :
  - fichiers de logs
  - trafic réseau

Apache Avro

- Système de sérialisation de données pour Hadoop
- Les données sérialisées sont associées à un schéma, ce qui permet de les traiter plus rapidement et efficacement à partir de langage de programmation de haut-niveau

Administration

Administration

Sommaire

- Gestion du cluster HDFS
  - Configurer un SecondaryNameNode
  - Supprimer un DataNode
- Gestion du cluster MapReduce
- Gestion des TaskTrackers

- Replacing a slave node
- Managing MapReduce jobs
- Checking job history from the web UI
- Importing data to HDFS
- Manipulating files on HDFS

Administration

Gestion du cluster HDFS

- Commande : `hadoop fsck`
- Vérification du système de fichiers et des fichiers :
  - `# hadoop fsck /`
  - `# hadoop fsck / -files`
- Localisation des blocs des fichiers :
  - `# hadoop fsck / -files -blocks`
- Suppression des fichiers corrompus ou déplacement vers `/lost+found` :
  - `# hadoop fsck -delete`
  - `# hadoop fsck -move`

Administration

HDFS *Safe Mode*

- Quand le *NameNode* entre dans le *Safe Mode*, HDFS est positionné en mode *Read-Only*
- Quand le *NameNode* démarre, il entre dans le *Safe Mode* et compte le nombre de blocs qui sont "sous-répliqués"
  - la valeur est 3 par défaut
- Si le pourcentage de blocs sous-répliqués > à un seuil, HDFS reste en *Safe Mode*
  - il faut lancer d'autres *Data Nodes*

Administration

Gestion du cluster HDFS (2)

- Commande : `hadoop dfsadmin`
- Obtenir l'état des noeuds :
  - `# hadoop dfsadmin -report`
- Gestion du *Safe Mode* :
  - `# hadoop dfsadmin -safemode get`
  - `# hadoop dfsadmin -safemode enter`
  - `# hadoop dfsadmin -safemode leave`
  - `# hadoop dfsadmin -safemode wait`
- Sauvegarde des méta-données :
  - `# hadoop dfsadmin -metasave meta.log`
    - crée le fichier `$HADOOP_HOME/logs/meta.log`

Administration

Le *SecondaryNameNode (job)*

- Le *NameNode* stocke les méta-données dans un fichier nommé *fsimage*
- Les modifications sur HDFS sont stockées dans un fichier de *log*
- Au boot, le *NameNode* charge le FS et applique les opérations du *log*
- Le *SecondaryNameNode* est un job qui lit périodiquement le *log* et l'applique à *fsimage*
  - permet de redémarrer plus rapidement !

Administration

Configurer le *SecondaryNameNode*

- Ajouter dans le fichier `$HADOOP_HOME/conf/hdfs-site.xml` :

```
<property>
        <name>fs.checkpoint.dir</name>
        <value>/hadoop/data/namesecondary</value>
</property>
```

  - la valeur par défaut est `${hadoop.tmp.dir}/dfs/namesecondary`
- Pour assurer la redondance des méta-données, on peut indiquer une liste de répertoires pour `dfs.name.dir`

Administration

Gestion du cluster MapReduce

- En SafeMode, aucun job ne peut être soumis
- Gestion du SafeMode :
  - `# hadoop mradmin -safemode get`
  - `# hadoop mradmin -safemode enter`
  - `# hadoop mradmin -safemode leave`
  - `# hadoop mradmin -safemode wait`
  - `# hadoop mradmin -refreshQueues`
- Après modification de la configuration des Task Trackers :
  - `# hadoop mradmin -refreshNodes`

Administration

Gestion des TaskTracker

- Le TaskTracker accepte les jobs provenant du JobTracker
  - il crée un processus par demande de job
  - il est responsable de surveille l'état des processus via un *heartbeat*
- Hadoop gère trois listes de TaskTracker :
  - `blacklist` : un TaskTracker qui a un taux d'échec de job > à un seuil est blacklisté
  - `gray list` : idem avec un autre seuil
  - `excluded list` : exclusion manuelle
- Pour obtenir la liste des Task Trackers :
  - `# hadoop job -list-active-trackers`

Administration

Configuration des TaskTrackers

- Via le fichier `$HADOOP_HOME/conf/mapred-site.xml`
  - sur chaque noeud du cluster !

```
<!-- intervalle du heartbeat (en ms) -->
<property>
        <name>mapred.tasktracker.expiry.interval</name>
        <value>60000</value>
</property>

<!-- nb de jobs en erreur qui vont blacklister -->
<property>
        <name>mapred.max.tracker.failures</name>
        <value>10</value>
</property>

<!-- nom du fichier texte qui contient les exclus -->
<property>
        <name>mapred.hosts.exclude</name>
        <value>$HADOOP_HOME/conf/mapred-exclude.txt</value>
</property>
```


Administration

Compléments sur les blacklists

- Un job qui génère des tasks en erreur ne peut blacklister plus de 25% des noeuds
- A tout moment seuls 50% des noeuds peuvent être blacklistés
- Après 24h, un noeud est dé-blacklisté
- Opérations manuelles :
  - `# hadoop job -list-blacklisted-trackers`
  - `# hadoop job -blacklist-tracker <hostname>`
  - `# hadoop job -unblacklist <jobid> <hostname>`
  - `# hadoop job -unblacklist-tracker <hostname>`

Administration

Décommissionner un DataNode

- Objectif :
  - Remplacer un nœud ou bien effectuer une opération de maintenance
- Créer un fichier `$HADOOP_HOME/conf/dfs-exclude.txt` qui contient le nom du nœud
- Ajouter ce fichier dans `$HADOOP_HOME/conf/hdfs-site.xml` :

```
<property>
    <name>dfs.hosts.exclude</name>
    <value>$HADOOP_HOME/conf/dfs-exclude.txt</value>
</property>
```

- Rafraichir le cluster :
  - `hadoop dfsadmin -refreshNodes`

145

Administration

Migrer des données dans HDFS

- Copier à partir du système local :
  - `hadoop fs -mkdir data`
  - `hadoop fs -cp file:///path/file /user/hduser/data`
  - `hadoop fs -put /path/file /user/hduser/data`
  - `hadoop fs -mv file:///path/file /user/hduser/data`
  - `hadoop fs -copyFromLocal src1 src2 data`
  - `hadoop distcp file:///path/bigfile /user/hduser/data`
    - lance un job MapReduce pour copier un gros fichier

146

Administration

Les quotas dans HDFS

- HDFS permet de placer des quotas par utilisateurs et par répertoires
- Pour limiter le nombre de fichiers / répertoire :
  - `hadoop dfsadmin -setQuota 20 /user/hduser`
- Pour limiter l'espace disque :
  - `hadoop dfsadmin -setSpaceQuota 1000000 /user/hduser`
- Vérifier l'état des quotas :
  - `hadoop fs -count -q /user/hduser`
- Supprimer les quotas :
  - `hadoop dfsadmin -clrQuota /user/hduser`
  - `hadoop dfsadmin -clrSpaceQuota /user/hduser`

147

Administration

Les schedulers « enfichables »

- Par défaut, Hadoop n'utilise qu'une seule queue, mais on peut en définir plusieurs
- CapacityScheduler
  - objectif de maximisation de l'utilisation du cluster
  - supporte la prioritisation
  - supporte des queues multiples
- FairScheduler
  - répartiteur équitable des ressources
- Visualisables et gérables via l'UI du JobTracker

148

Administration

Le *CapacityScheduler*

- Modifier le fichier `$HADOOP_HOME/conf/mapred-site.xml`

```
<property>
  <name>mapred.jobtracker.taskScheduler</name>
  <value>org.apache.hadoop.mapred.CapacityTaskScheduler</value>
</property>
```

- Définir une nouvelle queue dans `$HADOOP_HOME/conf/mapred-site.xml`

```
<property>
  <name>mapred.queue.names</name>
  <value>default,hdqueue</value>
</property>
```

149

J.Delamarche (2002-2015)

Administration

Le *CapacityScheduler* (2)

- Configurer la queue dans le fichier `$HADOOP_HOME/conf/capacity-scheduler.xml`

```
<property>
  <name>mapred.capacity-scheduler.queue.hdqueue.capacity</name>
  <value>20</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.default.capacity</name>
  <value>80</value>
</property>

<!- autres paramètres -->
mapred.capacity-scheduler.queue.hdqueue.minimum-userlimit-percent
mapred.capacity-scheduler.maximum-system-jobs
mapred.capacity-scheduler.queue.hdqueue.maximum-initialized-active-tasks
mapred.capacity-scheduler.queue.hdqueue.maximum-initialized-active-tasks-per-user
mapred.capacity-scheduler.queue.hdqueue.supports-priority
```

150

J.Delamarche (2002-2015)

Administration

Configuration des logs

- Utilise `Log4j`
- Configuration :
  - `$HADOOP_HOME/conf/log4j.properties`
  - par défaut, génère les messages `INFO` ou plus dans le fichier `$HADOOP_LOG_DIR/hadoop-$HADOOP_IDENT_STRING-<hostname>.log` et vers la console (STDERR)

Administration

Commandes de gestion des logs

- Connaitre le niveau des logs :
  - `hadoop daemonlog -getlevel master:50030 org.apache.hadoop.mapred.JobTracker`
  - `hadoop daemonlog -getlevel master:50030 org.apache.hadoop.mapred.TaskTracker`
  - `hadoop daemonlog -getlevel master:50070 org.apache.hadoop.dfs.NameNode`
  - `hadoop daemonlog -getlevel master:50070 org.apache.hadoop.dfs.DataNode`
- Modifier le niveau des logs :
  - `hadoop daemonlog -getlevel master:50030 org.apache.hadoop.mapred.JobTracker ERROR`

Administration

Procédure de mise à jour

- Plus ou moins complexe :
  - passage de 1.x à 1.y ou passage de 1.x à 2.y ?
- Etapes principales :
  - arrêter le cluster (`stop-all.sh`) !
  - sauver la localisation des blocks dans HDFS
  - sauver la liste des fichiers de HDFS
  - sauver la description des DataNodes
  - sauver les fichiers de "checkpoints"
  - vérifier que l'arrêt de tous les démons de DataNode
  - mettre à jour les sources de Hadoop
  - recopier les anciennes configuration
  - mettre à jour le lien symbolique de Hadoop
  - mettre à jour les noeuds esclaves, puis le NameNode
  - démarrer le cluster HDFS
  - vérifier le contenu du cluster par rapport à la sauvegarde
  - démarrer le cluster MapReduce

Administration

Sécuriser le cluster

- Mise en oeuvre des ACL pour limiter les accès aux ressources
- Mise en oeuvre de Kerberos pour chiffrer et sécuriser les communications
- ...et pour limiter l'accès aux interfaces d'administration Web

Administration

Mise en oeuvre des ACL

- Possibilité d'activer la SLA (Service Level Authentication) pour autoriser, via des ACL, certains utilisateurs à accéder aux ressources du cluster
- `core-site.xml` permet de gérer l'accès à HDFS, au JobTracker au TaskTracker
- `mapred-site.xml` permet de gérer l'accès aux queues et aux jobs

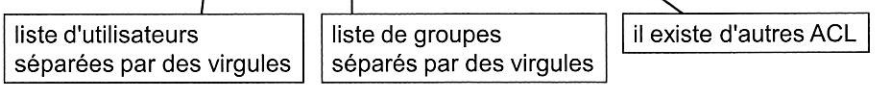
Administration

Mise en oeuvre des ACL

- Exemple de configuration dans `core-site.xml` :

```
<property>
  <name>hadoop.property.authorization</name>
  <value>true</name>

  <name>security.job.submission.protocol.acl</name>
  <value>hduser,bob hadoop</name>
</property>
```



- `hadoop dfsadmin -refreshServiceAcl`
- `hadoop mradmin -refreshServiceAcl`
- `hadoop queue -showacls`

Sommaire

Sommaire

- Prise en main
- Manipulations de documents
- Interrogations
- Indexation
- Agrégation
- Administration
- Réplication
- Sharding et Cluster

Prise en main

Concepts de base

- L'unité de base de stockage est le **document**
  - assimilable à la ligne des tables des SGBD-R
- Les documents sont agrégés dans des **collections**
  - assimilable à une table de SGBD-R
- Chaque document possède un attribut unique dans la collection : **_id**
- Un serveur MongoDB gère une ou plusieurs base de données possédant chacune une ou plusieurs collections
- L'administration s'effectue avec un shell écrit en JavaScript

Prise en main

Le format d'échange JSON

- JavaScript Object Notation (RFC 4627)
- Document texte comprenant :
  - des paires clés / valeurs
  - des listes ordonnées de valeurs
  - les valeurs peuvent être :
    - des scalaires, des tableaux, des objets, des tableaux associatifs
- Exemple :

```
{
  "clé1" : "valeur",
  "clé2" : { "attr1" : 10, "attr2" : [ 1,2,3 ] }
}
```


Prise en main

Le format BSON

- Binary JSON
- Encodage binaire des données JSON
  - plus compact, portable, plus efficace à traiter
- Format d'encodage des documents

Prise en main

Documents

- Assimilés à des tableaux associatifs clés/valeurs dont la représentation en JSON est :
  - `{"cle1": 3, "cle2": "abc", "cle3": 1.2}`
- Les noms de clés contiennent tout caractère UTF-8 exceptés :
  - le nul `\0`
  - le `.` et le `$` qui sont spéciaux
  - et ne débutent pas par `_` qui est réservé
- Les valeurs des clés sont typées

Prise en main

Documents (2)

- Les documents sont sensibles au type (*type-sensitive*) :
  - `{ "cle" : 3 }`
  - `{ "cle" : "3" }`
- Les documents sont sensibles à la casse (*case-sensitive*) :
  - `{ "cle" : 3 }`
  - `{ "Cle" : 3 }`

Prise en main

Types intégrés

- Numbers
  - entiers sur 32 ou 64 bits ou flottants
  - supporte des opérateurs comme `$inc`
  - la conversion 32 vers 64 bits est automatique en cas de débordement
- Dates
  - supporte le format complet avec heure
  - et le format simplifié *yyyy-mm-dd* (pour stocker une date de naissance !)
- Chaines
  - au format UTF-8
- ObjectId
  - référence à un objet
  - ne pas stocker sous forme de chaine !

→ Booléens ?

→ Nulle ?

Prise en main

Collections

- Même si MongoDB est *schema-free*, il est utile de rassembler les documents similaires dans des collections
- Les noms des collections sont des noms UTF-8 qui :
  - ne contiennent pas le caractère nul `\0`
  - ne peut être la chaine vide `""`
  - ne peut être préfixé par `system.` qui est réservé
  - doit éviter de contenir des `$`

Prise en main

Sous-collections

- Il est utile d'organiser les collections dans des espaces de nommage
  - => elles forment alors des sous-collections
- Leur utilisation est recommandée
- Exemple :
  - `blog` est une collection
  - `blog.authors` est une autre collection
- Les noms complets ne doivent pas dépasser 121 caractères (moins de 100 en pratique)

Prise en main

Bases de données

- Les collections sont regroupées en bases de données
- Une instance MongoDB gère plusieurs bases
- Chaque base est stockée dans ses propres fichiers
- Les noms des bases suivent les règles :
  - la chaine vide `""` est invalide
  - les caractères espace, `.`, `$`, `/`, `\` et `\0` sont interdits
  - les noms doivent être en minuscules
  - 64 caractères maximum

Prise en main

Bases de données réservées

- `admin`
  - base "racine" pour la configuration des authentifications
  - utilisée pour lancer certaines commandes (avoir la liste des bases !)
- `local`
  - contient les collections locales qui ne doivent pas être répliquées
- `config`
  - dans un environnement à base de *shards*, contient la configuration des *shards*

169

Prise en main

Exemple d'utilisation

- Lancement du serveur :
  - `# ./mongod`
- Lancement d'un client : le shell Mongo
  - `$ ./mongo`
  - `> help`
- Premières commandes :
  - `> use demo`
  - `> db.personne.insert({nom: "Dupont", prenom: "Jean"})`
  - `> db.personne.find()`
  - `> db.personne.drop()`

170

Manipulation de documents

Insertion de documents

- En utilisant la méthode `insert` des collections :
  - `> db.coll.insert({ "cle": valeur })`
- L'attribut `_id` est ajouté automatiquement
- Il est possible de faire de l'insertion par batch en passant un tableau d'objets
  - attention à la limite de taille des requêtes à 16 Mo
    `> db.isMaster().maxBsonObjectSize`
- La taille d'un document est limitée à 16 Mo
  - à vérifier en fonction des versions
  - la taille d'un document est donnée par la méthode :
    `> Object.bsonsize(doc)`

Manipulations

Suppression de documents

- En utilisant la méthode `remove` des collections :
  - `> db.users.remove()`
    - supprime tous les documents d'une collection, mais ni la collection, ni les index
  - `> db.users.remove({ "ville": "tlse" })`
    - suppression conditionnée par un filtre
- En utilisant la méthode `drop_collection` :
  - `> db.drop_collection('users')`

Manipulations

Mise à jour de documents

- En utilisant la méthode `update` des collections
- Prend deux paramètres :
  - un filtre (une requête) qui sélectionne les documents à modifier
  - la liste des modifications à apporter aux documents

Manipulations

Remplacement de document

- Document initial :

```
{
  "_id": ObjectId("1234"),
  "nom": "jean",
  "amis": 30,
  "ennemis": 2,
 }
```

- Document final :

```
{
  "_id": ObjectId("1234"),
  "nompersonne": "jean",
  "relations": {
    "amis": 30,
    "ennemis": 2
  }
 }
```


Manipulations

Remplacement de document (2)

```
> var jean = db.users.findOne({"nom": "jean"});
> jean.relations = {"amis": jean.amis, "ennemis": jean.ennemis},
> jean.nompersonne = jean.nom;
> delete jean.amis;
> delete jean.ennemis;
> delete jean.nom;
> db.users.update({"nom": "jean"}, jean}
```

- Mais que se passe-t-il s'il existe plusieurs documents dont le "nom" est égal à "jean" ?
  - le 1er va être modifié et son `_id` va être identique à celui retourné par `findOne()` : est-ce le même ?
    - => duplication de clé détectée
- Par défaut, `update()` modifie un document
  - l'option `{multi: 1}` modifie tous les documents

Manipulations

Remplacement partiel de document

- Parfois, seules des portions du document doivent être modifiées
  - on utilise des *modifiers* atomiques
- Exemple : le modifieur `$set`
  - ajoute un attribut :

```
> db.users.update({"nom": "jean"}, {"$set" : {"age": 30 }})
```

  - modifie un attribut :

```
> db.users.update({"nom": "jean"}, {"$set" : {"age": 40 }})
```

  - modifie le type d'un attribut :

```
> db.users.update({"nom": "jean"}, {"$set" : {"age": [30,40]}})
```

  - `$unset` permet de supprimer un attribut :

```
> db.users.update({"nom": "jean"}, {"$unset" : {"age": 1 }})
```

177

J.Delamarche (2002-2015)

Manipulations

Incrémenter et décrémenter

- Le modifieur `$inc` permet de modifier les valeurs numériques de clés existantes ou bien d'en créer :
  - création de l'attribut :

```
> db.users.update({"nom": "jean"}, {"$inc": {"enfants": 2}})
```

  - incrémentation après une naissance :

```
> db.users.update({"nom": "jean"}, {"$inc": {"enfants": 1}})
```

  - modifieur performant car ne modifie pas la taille d'un document !

178

J.Delamarche (2002-2015)

Manipulations

Modifieurs de tableaux

- Le modifieur `$push` crée ou ajoute un élément à un tableau :
  - création du tableau initial :

```
> db.users.update({"nom": "jean"},
... {"$push": {"livres": {"titre": "tintin"}}})
```

  - ajout d'un élément au tableau

```
> db.users.update({"nom": "jean"},
... {"$push":{"livres": {"titre": "asterix"}}})
> db.users.findOne()
{
  "_id": ObjectId("1234"),
  "nom": "jean",
  "livres": [
    { "titre": "tintin" },
    { "titre": "asterix" },
  ]
}
```

179

Manipulations

Ajout conditionnel d'éléments

- Le modifieur `$addToSet` permet d'ajouter un élémént s'il n'est pas déjà présent :

```
> db.users.update({"nom": "jean"},
... {"$addToSet": {"emails": "jean@free.fr" }})

> db.users.update({"nom": "jean"},
... {"$addToSet": {"emails": "jean@free.fr" }})

> db.users.update({"nom": "jean"},
... {"$addToSet": {"emails": "jean@hotmail.fr" }})

> db.users.findOne()
{
 "_id": ObjectId("1234"),
  "nom": "jean",
  "emails": [
    "jean@free.fr",
    "jean@hotmail.fr"
  ]
}
```

180

Manipulations

Ajout d'une liste à un tableau

- On utilise `$addToSet` avec `$each` :

```
> db.users.update({"nom": "jean"},
... {"$addToSet": {"emails": {"$each": [
...                 "jean@hotmail.fr", "jean@gmail.com" ] }}})

> db.users.findOne()
{
 "_id": ObjectId("1234"),
  "nom": "jean",
  "emails": [
    "jean@free.fr",
    "jean@hotmail.fr",
    "jean@gmail.com"
  ]
}
```


Manipulations

Suppression d'éléments d'un tableau

- Le modifieur `$pop` permet de traiter le tableau comme une liste ou une pile
- Le modifieur `$pull` permet de supprimer des éléments quelconques

```
> db.users.update({"nom": "jean"}, {"$pop": {"emails": 1}})
> db.users.update({"nom": "jean"}, {"$pop": {"emails": -1}})
> db.users.update({"nom": "jean"},
... {"$pull": {"emails": "jean@free.fr"}})

> db.users.findOne()
{
 "_id": ObjectId("1234"),
  "nom": "jean",
  "emails": [
    "jean@hotmail.fr",
  ]
}
```

Manipulations

Modification positionnelle de tableau

- Comment faire quand les élements du tableau sont des sous-documents ?
- On peut accéder aux éléments par leur index (le 1er ayant pour index 0)

```
> db.users.update({"nom": "jean"},
... {"$set": {"livres.0.titre": "Tintin"}})
```


Manipulations

Modification positionnelle de tableau (2)

- Parfois on ne connait pas l'index : on utilise l'opérateur positionnel $:

```
> db.blog.findOne()
{
  "_id": ObjectId("1234"),
  "comments": [
    { "comment": "blabla", "auteur": "jean", "votes": 0 },
    { "comment": "super !", "auteur": "martin", "votes": 3 },
    { "comment": "bof !", "auteur": "alice", "votes": -1 },
  ]
}
> db.blog.update({"comments.auteur": "jean"},
... {"$set": {"comments.$.auteur": "bernard"}})
```

Manipulations

Les *Upserts*

- Type spécial d'*update* :
  - si le document n'existe pas, il est créé
  - évite un test préalable d'existence, qui, en plus, serait sujet à un problème d'accès concurrent, car l'opération *test-and-set* n'est pas atomique !
- Les options de update indique s'il s'agit d'un *upsert* :

```
> db.users.update({"nom": "paul"},
... {"$push": {"livres": {"titre": "tintin"}}}, {upsert: 1})
```

- Avec le shell, la fonction `save()` fait un *upsert* si le document soumis possède un attribut `_id`

185

J.Delamarche (2002-2015)

Manipulations

Les *Upserts* (2)

- Lors d'un upsert, l'opérateur `$setOnInsert` permet de n'affecter des champs uniquement en cas d'insertion :

```
> db.users.update({"nom": "paul"},
... {"$push": {"livres": {"titre": "tintin"}}},
... {"$setOnInsert": {"qte": 1, "annee": 2000}},
... {upsert: true} )
```

186

J.Delamarche (2002-2015)

Manipulations

Mise à jour de documents multiples

- Le 4ème paramètre de `update()` permet de spécifier si tous les documents correspondants aux critères doivent être mis à jour (`true`) ou bien seulement le 1er (`false`)
- La commande **`getLastError`** retourne le nombre de documents mis à jour :

```
> db.user.update({"nom": "jean"}, {"genre": "m"}, false, true);
> db.runCommand({getLastError: 1})
{
  "err": null,
  "updatedExisting": true,
  "n": 5,
  "ok": true
}
```


Manipulations

Atomicité du recherche & mettre à jour

- Faire un `find()` puis un `update()` sur le ou les documents retournés peut poser un problème dans un environnement concurrent
- Pour éviter cela, il faut utiliser l'opération **`findAndModify`**, qui réalise recherche et mise à jour de manière atomique
  - ne fonctionne que sur un document à la fois
- Dans un même registre :
  - pour "isoler" une mise à jour multiple, utiliser l'opérateur `$atomic` avec update afin d'empêcher les écritures sur la même collection

Manipulations

Paramètres de `findAndModify`

- Paramètres :
  - le nom de la collection
  - `query` : le critère de recherche des documents
  - `sort` : critère de tri
  - `update` : modification à apporter
  - `remove` : booléen (exclusif de "update")
  - `new` : booléen qui indique s'il faut retourner le document avant (défaut) ou après sa mise à jour

```
> ps = db.runCommand({"findAndModify": "processes",
... "query": {"status": "READY"},
... "sort": {"priority": -1},
... "update": {"$set": {"status": "RUNNING"}}).value
```

189

Manipulations

Capped Array (v2.4)

- Le modifieur `$each` peut être utilisé avec `$push` pour effectuer un push multiple
- Avec les modifieurs `$sort` et `$slice`, on peut alors limiter la taille d'un tableau :

```
> db.students.update( { name: "joe" },
          { $push: { scores:
                     { $each: [ 90, 92, 85 ] } } } )
> db.students.update( { name: "joe" },
     { $push: { quizzes:
       { $each: [ { id: 3, score: 8 },
                  { id: 4, score: 7 },
                  { id: 5, score: 6 } ],
         $sort: { score: 1 }, $slice: -5 } } } )
```

190

⚠ Dénormaliser les données ~~à~~ ou pas de jointures possibles

Manipulations

Outils de manipulations

- `mongo` :
  - binaire livré en standard
  - offre une interface de type "shell"
- `rockmongo` : → robomongo mieux
  - application PHP qui propose une interface Web
  - facile à installer
  - complet grâce à des plugins
- API clientes (PHP, Python...)

MongoDB

Interrogations

Interrogation

La méthode `find`

- Retourne aucun, un, plusieur ou tous les documents d'une collection :
  - `> db.coll.find()`
  - `> db.coll.find({})`
  - `> db.coll.find({"cle" : valeur})`
- Par défaut, les objets complets sont retournés, mais on peut choisir des paires clés/valeurs :
  - `> db.coll.find({}, {"c1": 1, "c2": 1})`
- ou exclure des paires clés/valeurs :
  - `> db.coll.find({}, {"c1": 0, "c2": 0})`

193

J.Delamarche (2002-2015)

Interrogation

Critères de recherche : les conditions

- Les opérateurs de comparaison sont `$lt`, `$lte`, `$gt`, `$gte` et `$ne` :

```
> db.users.find({"age": {"$gte": 18, "$lte": 30 }})

> start = new Date("01/01/2012")
> db.users.find({"registered": {"$lt": start}})

> db.users.find({"nom": {"$ne": "jean"}})
```

  - il n'y a pas d'opérateur `$eq` !

194

J.Delamarche (2002-2015)

Interrogation

Combinaisons de OR

- Par défaut, les critères multiples sont combinés avec AND
- Les opérateurs `$in`, `$nin` et `$or` permettent de combiner des OR :

```
> db.users.find({"age": {"$in": [18,20,22]}})
> db.users.find({"age": {"$nin": [1,2,3,4]}})
> db.users.find({"$or": [{"nom": "jean"}, {"age": 30}]})
```


Interrogation

Comment faire une négation ?

- ...à l'aide de l'opérateur `$not` qui renverse le test indiqué par tout autre opérateur :

```
> db.users.find({"age": {"$not": {$in": [18,20,22]}}})
```

Interrogation

Cas des valeurs `null` ou inexistantes

- Il est possible de faire des recherches avec la valeur `null`
- Mais si la clé n'existe pas, tous les documents n'ayant pas la clé sont retournés !
- Un contournement possible est de tester que la clé existe ET possède une valeur `null` :
  - `> db.coll.find({"z": {"$exists": true, "$in": [null]}})`
    - rappel : il n'y a pas d'opérateur `$eq` !

Interrogation

Recherche avec expressions régulières

- MongoDB utilise les R.E au format PCRE
- Exemples :

```
> db.users.find({"name": /jean/})
> db.users.find({"name": /^jean/i})

# Si la valeur est une R.E, ça fonctionne aussi,
# c.à.d dans l'exemple que si "name" vaut "/jean/",
# le document correspond à la R.E
```

Interrogation

Recherche dans des tableaux

- Chaque valeur du tableau est testée comme si elle était unique :

```
> db.food.insert({"_id": 1, "fruit": ["apple", "banana", "peach"]})
> db.food.insert({"_id": 2, "fruit": ["apple", "kumquat", "orange"]})
> db.food.insert({"_id": 3, "fruit": ["cherry", "banana", "apple"]})

> db.food.find({"fruit": "banana"})   # retourne _id=1

> db.food.find({"fruit.2": "peach"})  # retourne _id = 1
```

- L'opérateur `$all` permet de tester plusieurs valeurs simultanément :

```
> db.food.find({"fruit": {"$all": ["apple", "banana"]}}) # => 1 et 3

> db.food.find({"fruit": ["apple", "banana", "peach"]}) # => 1

> db.food.find({"fruit": ["peach", "banana", "apple"]}) # => rien !
```

199

J.Delamarche (2002-2015)

Interrogation

Opérateur `$size`

- Permet de tester la taille des tableaux :
  - uniquement un test d'égalité
  - pour des tests d'inégalité, il faut utiliser une clé

```
> db.food.find({"fruit": {"$size": 3}})

> db.food.update({"$push": {"fruit": "fraise"}, {"$inc": 1}})
> db.food.find({"size": {"$gt": 3}})
```

200

J.Delamarche (2002-2015)

Interrogation

Opérateur `$slice`

- Permet d'obtenir une "tranche" de tableau
  - toutes les clés sont retournées

```
> db.blog.posts.findOne(critere, {"comments": {"$slice": 10}})
> db.blog.posts.findOne(critere, {"comments": {"$slice": -10}})
> db.blog.posts.findOne(critere, {"comments": {"$slice": [23,10]}})
```

  - les 10 1ers éléments
  - les 10 derniers
  - les 10 éléments après le 23ème

Interrogation

Requête sur des sous-documents

- Exemple :

```
{
  "name": {
    "first": "Joe",
    "last": "Doe",
  }
}
```

- Mauvais car dépend de l'ordre des clés et doit contenir toutes les clés :

```
> db.people.find({"name": {"first": "Joe", "last": "Doe"}})
```

- Bon car indépendant de l'ordre des clés :

```
> db.people.find({"name.first": "Joe", "name.last": "Doe"})
```

Interrogation

Requête sur des sous-documents (2)

- Exemple :

```
{
  "comments" : [
    { "author": "Joe", "score": 3, "comment": "xyz" },
    { "author": "Mary", "score": 6, "comment": "ab" }
  ]
}
```

- Ne marchent pas :

```
> db.blog.find({"comments": {"author": "Joe", "score": {"$gte": 5}}})
> db.blog.find({"comments.author": "Joe", "comments.score": {"$gte": 5}})
```

- Bon : utiliser `$elemMatch` !

```
> db.blog.find({"comments": {"$elemMatch": {"author": "Joe",
                                            "score": {"$gte": 5}}}})
```


Interrogation

Du JavaScript dans les recherches !

- ...en utilisant l'opérateur `$where` :
  - exemple : on cherche les items contenant des fruits en même quantité

```
> db.fruit.insert({"pomme": 1, "banane": 6, "peche": 3})
> db.fruit.insert({"pomme": 8, "epinard": 4, "melon": 4})
```

```
> db.fruit.find({"$where": function() {
    for (var current in this) {
      for (var other in this) {
        if (current != other && this[current] == this[other]) {
          return true;
        }
      }
    }
    return false;
  }});
```

  - mais lent car s'exécute sur le serveur et ne peut utiliser d'index

Interrogation

Du JavaScript dans les recherches ! (2)

- Ecritures plus compactes :

```
> db.fruit.find({"$where": "this.x + this.y == 10"})
> db.fruit.finc({"$where": "function() {
                              return this.x + this.y == 10; }"})
```

- Autre possibilité de réaliser des requêtes complexes : l'algorithme MapReduce
  - (voir plus loin)

Interrogation

Les Curseurs (*cursors*)

- L'appel à la méthode `find()` retourne un *cursor* :

```
> var cursor = db.une_collection.find();
> while (cursor.hasNext()) {
... obj = cursor.next();
... // faire qq chose avec l'objet
... }

> var cursor = db.une_collection.find();
> cursor.forEach(function(x) {
... // faire qq chose
... });
```

- Le shell ne requête pas la base sur `find()`
- On peut ajouter des clauses à `find()`

Interrogation

Limit, skip et sort

- `limit()` permet de limiter le nombre maximum d'objets retournés
- `skip()` permet d'ignorer les "n" premiers objets qui seraient retournés
  - attention : à éviter avec des grandes valeurs (voir plus tard)
- `sort()` permet de spécifier les colonnes qui doivent servir de critère de tri (asc. ou desc.)
- L'ordre de ces fonctions n'a pas d'importance

```
> db.users.find().limit(3).skip(4).sort({username: 1, age: -1})
```


Interrogation

Options avancées

- La requête :

```
> db.coll.find({"foo": "bar"}).sort({"x":1})
```

- est en fait traduite en :

```
> db.coll.find({"$query": {"foo": "bar"},
  "$orderby": {"x": 1}})
```

- Il existe d'autres options :
  - `$maxscan` : nb max de documents à scanner
  - `$min` et `$max` : début et fin de recherche
  - `$hint` : conseil sur l'index (voir plus loin)
  - `$explain` : (voir plus loin)
  - `$snapshot` : pour obtenir un résultat consistent

Interrogation

Utilisation de *snapshots*

- Il y a un risque de modifier les objets parcourus par un curseur :

```
cursor = db.coll.find();
while (cursor.hasNext()) {
        var doc = cursor.next();
        doc = process(doc);
        db.coll.save(doc);
}
```

- Si l'objet devient plus gros après appel à `process()` il peut se retrouver déplacé en fin de collection
  - => le curseur le traitera deux fois !
  - l'option `$snapshot` évite cela !

209

Interrogation

Curseurs internes

- Il y a deux types de curseurs :
  - les curseurs côté "client", gérés par le driver
  - les curseurs "internes", gérés par le serveur
- Les curseurs internes sont détruits :
  - quand il n'y a plus de résultat à retourner
  - quand le client le demande (via le driver)
  - après un timeout d'inactivité alors qu'il reste des résultats !
    - les drivers proposent une fonction "`immortal`" qui permet d'éviter cela, mais le curseur risque d'exister pour l'éternité !

210

MongoDB

Indexation

Index

Besoin d'indexation

- Pour des raisons de performances
  - évite le *table-scan*
- Fonctionnement et intérêt similaires aux SGBD relationnels
- Création d'un index sur un attribut :
  - `> db.coll.ensureIndex({"cle": 1})`
    - la valeur "`1`" indique un tri croissant
    - la valeur "`-1`" indique un tri décroissant

Index

Index multiples (ou composés)

- Les index peuvent porter sur plusieurs colonnes:
  - `> db.coll.ensureIndex({"cle1": 1, "cle2": 1, "cle3": 1})`
- L'index sera utilisé pour toute question mettant en œuvre `cle1`, `cle1` et `cle2` ou `cle1`, `cle2` et `cle3`
- L'optimiseur remet l'ordre des clauses dans l'ordre qui permet d'utiliser l'index :
  - `> db.coll.find({"cle1":123})`
  - `> db.coll.find({"cle1":123,"cle2":123})`
  - `> db.coll.find({"cle2":123,"cle1":123})`

213

Index

Indexer des sous-documents

- Il est possible d'indexer des clés qui composent des sous-documents
  - et aussi de les utiliser dans des index composés
- Exemple :
  - `> db.users.find({"livres.titre": 1})`

214

Index

Index et tri

- L'utilisation de la clause `sort()` implique un tri en mémoire
  - si les documents sont trop nombreux, la mémoire nécessaire peut être insuffisante
- Il faut alors placer un index sur la clé de tri

Index

Nommage des index

- Par défaut, les index créés sont nommés en interne ainsi :
  - `cle1_sens1_cle2_dir2…._cleN_sensN`
  - où `cleX` est le nom de la clé
  - et `sensX` correspond au sens de tri (1 ou -1)
- Il est possible de donner un nom à un index :
  - d'autant plus que le nom d'un index a une taille limitée et donc que le nommage explicite peut être nécessaire…
  -
```
> db.coll.ensureIndex({"a": 1, "b": -1},
                      {"name": "ind"})
```

Index

Index unique

- Un index unique assure l'unicité des valeurs lors de l'insertion
  - attention : il faut donc faire appel à `getLastError` pour vérifier si un `insert()` a réussi
- La définition d'un index unique est :
  - `> db.coll.ensureIndex({"nom": 1},`
    `{"unique": true})`
  - l'attribut `_id` est un index unique
  - si la clé de l'index unique est vide, sa valeur sera `null` et il ne peut y avoir qu'une seule valeur `null` pour un index unique !
  - dans le cas d'un index composé, c'est l'ensemble des clés qui doit avoir une valeur unique

Index

Suppression des doublons

- Comment faire pour créer un index unique alors que certaines valeurs sont doublonnées ?
- Il est possible d'éliminer les documents redondants :
  - `> db.coll.ensureIndex({"nom": 1},`
    `{"unique": true, "dropDups": true})`

outil de debug

Index

Obtenir des conseils sur les index

- La méthode `explain()` permet d'obtenir des informations sur l'utilisation des index :

```
> db.users.find().explain()
{
  "cursor": "BasicCursor",
  "indexBounds": [ ],
  "nscanned": 64,
  "nscannedObjects": 64,
  "n": 64,
  "millis": 0,
  "allPlans" : [
    {
      "cursor": "BasicCursor",
      "indexBounds": [ ]
    }
  ]
}
```


Index

Lecture de `explain()`

- `"cursor" : "BasicCursor"`
  - indique qu'aucun index n'a été utilisé (normal !)
- `"nscanned" : 64`
  - indique le nombre de documents analysés (devrait être aussi proche que possible du "bon" nombre)
- `"n" : 64`
  - nombre de documents retournés
- `"millis" : 0`
  - nombre de millisecondes pour exécuter la requête
- `"allPlans" :`
  - indique les différents "plans d'exécution" possible

Index

Lecture de `explain()` (2)

```
> db.users.find({age: {"$gt": 20, "$lt": 30}}).explain()
{
  "cursor": "BtreeCursor age_1",
  "indexBounds": [
    [ { "age": 20 },
      { "age": 30 } ]
  ],
  "nscanned": 14,
  "nscannedObjects": 12,
  "n": 12,
  "millis": 1,
  "allPlans" : [
    {
      "cursor": " BtreeCursor age_1",
      "indexBounds": [
        [ { "age": 20 },
          { "age": 30 } ]
      ]
    }
  ]
}
```


Index

Lecture de `explain()` (3)

- `"cursor" : "BtreeCursor age_1"`
  - utilise l'index `age_1` dont on peut avoir une description :
    - `> db.system.indexes.find({"name": "age_1"})`
- `"allPlans" :`
  - indique les deux solutions possibles : la "meilleure" a été choisie
- Supposons qu'il existe un index sur `name+age` et un index sur `age+name`...

Index

Lecture de `explain()` (4)

```
> db.users.find({age: {"$gt": 10}, name: "jean"}).explain()
{
  "cursor": "BtreeCursor name_1_age_1",
  "indexBounds": [
    [ { "name": "jean", "age": 10 },
      { "name": "jean": "age": 1.7976e+308} ]
  ],
  "nscanned": 13,
  "nscannedObjects": 13,
  "n": 13,
  "millis": 5,
  "allPlans" : [
    {
      "cursor": " BtreeCursor name_1_age_1",
      "indexBounds": [
        [ { "name": "jean", "age": 10 },
          { "name": "jean": "age": 1.7976e+308} ]
      ]
    }
  ],
  "oldPlan": {
    "cursor": "BtreeCursor name_1_age_1",
    "indexBounds": [
      [ { "name": "jean", "age": 10 },
        { "name": "jean": "age": 1.7976e+308} ]
    ]
  }
}
```


Index

Lecture de `explain()` (5)

- On constate que l'index `name+age` a été utilisé, contrairement à l'ordre de la requête
- La requête suivante utiliserait l'index `age+name`:

```
> db.users.find({"age": 30,
                 "name": /.*/}).explain()
```

- Dans les cas (rares) où on a besoin de préciser le "bon" index, la syntaxe est :

```
> db.users.find({"age": 30,
                 "name": /.*/}).
        hint({"name": 1, "age": 1})
```

Index

Informations sur les index

- Les méta-données sur les index sont stockées dans la collection `system.indexes` de chaque base
  - la collection est manipulée indirectement par `ensureIndex()` et `dropIndexes()`
- La collection `system.namespaces` ne contient que les noms des index
  - => afficher le contenu et déduire pourquoi le nom d'une collection ne peut dépasser 121 octets, sachant que le nom d'un index ne peut dépasser 127 octets !

Index

Changer les index

- Il est possible de changer les index "à chaud"
- La création d'un index est bloquante par défaut, mais il est possible de l'éviter :

```
> db.users.ensureIndex({name": 1},
                       {"background": true})
```

- `dropIndexes()` permet de supprimer un index:

```
> db.runCommand({"dropIndexes": "users",
                 "index": "name"})
```

- `reIndex()` permet de détruire et recréer les index
- On peut supprimer tous les index par :

```
> db.runCommand({"dropIndexes": "users",
                 "index": "*"})
```

Index

Index géospatial

- Utilisé avec 2-dimensions (longitude et latitude par exemple)
- Création avec `"2d"` au lieu de `1` ou `-1` :

```
> db.map.ensureIndex({"gps": "2d"})
```

  - la valeur de l'attribut `gps` est supposée être une paire de valeurs, par défaut comprises entre -180 et 180 :
    - `{ "gps": [0, 100] }`
    - `{ "gps": { "lat": -180, "lon": 45 } }`
  - mais possibilité de donner des limites :
    -
```
...ensureIndex({"pos": "2d"},
                {"min": -1000, "max": 1000})
```

227

Index

Index géospatial (2)

- Le requêtage sur ces attributs utilise les opérateurs `$near`, `$within`, `$box`...

```
> db.map.find({"gps": {"$near":
                          [40, -73]}})
> db.map.find({"gps": {"$within" :
      {"$box": [[10,20], [15,30]]}}})
> db.map.find({"gps": {"$within" :
          {"$center": [[12,5], 5]}}})
```

228

2 dyslexie

Index

Traitement de données spatiales

- L'opérateur `$geoWithin` remplace l'opérateur `$within`
- Permet de faire des recherches d'inclusion de formes :
  - syntaxe de recherche de points situés entre les deux bordures d'un polygone :
    - `db.<collection>.find( { <location field> : { $geoWithin : { $geometry : { type : "Polygon" , coordinates : [ [ [ <lng1>, <lat1> ] , [ <lng2>, <lat2> ] ... ] [ [ <lngA>, <latA> ] , [ <lngB>, <latB> ] ... ] ] } } } } )`
  - les formes possibles sont les boites (`$box`), les polygones (`$polygon`), les cercles (`$center`) et les cercles sur sphères (`$centerSphere`)

229

Index Full-Text

Index Full-Text (v2.4)

- Les index de type `text` permettent d'indexer les mots contenus dans les champs d'une collection
- Un seul index de type `text` par collection
- La commande `text` permet alors d'effectuer des recherches
  - gère les radicaux (singulier, pluriel, conjugaison)
  - 15 langues supportées

230

Index Full-Text

Mise en oeuvre

- Il faut activer le support sur le serveur
  - et tous les noeuds des *replica-sets* et des *clusters* !
  - `textSearchEnabled=true`
    - sur la ligne de commande (`--setParameter ...`)
    - ou dans le fichier de configuration
- Création d'un index full-text sur deux champs :

```
> db.coll.ensureIndex({sujet: "text",
                       contenu: "text"})
```

- Recherche de texte :

```
> db.collection.runCommand( "text",
    { search: <string>, filter: <document>,
      project: <document>, limit: <number>,
      language: <string> } )
```


Index

Collection avec durée de vie (TTL)

- Il est possible d'indexer un champ de type date et de lui associer un TTL (en secondes)
  - `> db.coll.ensureIndex({date_crea: 1},`
    `{expireAfterSeconds: 3600})`
- Un thread se charge de supprimer les éléments obsolètes toutes les minutes

Agrégation

Fonctions d'agrégation simples

- `count()` :
  - retourne le nombre d'éléments d'une collection
  - avec un filtre éventuel
- `distinct` :
  - retourne les valeurs différentes pour une clé
- `group` :
  - permet d'effectuer des traitements sur les élements classés en groupes (come le GROUP BY de SQL)

Agrégation

Fonction `count()`

```
> db.une_collection.count()
10
> db.une_collection.insert({"x": 1})
> db.une_collection.count()
11

> db.une_collection.count({"x": 1})
1
```


Agrégation

Commande `distinct`

```
/* soit la collection "people" suivante :
 * {"name": "Ada", "age": 20 }
 * {"name": "Fred", "age": 35 }
 * {"name": "Susan", "age": 60 }
 * {"name": "Andy", "age": 35 }
 */
> db.runCommand({"distinct" : "people", "key": "age"})
["values": [20, 35, 60], "ok" : 1}
```

Agrégation

Commande `group`

- Exemple :
  - soit la collection suivante

```
{"day": "2012/12/03", "time": "03:57:01", "price": 4.23}
{"day": "2012/12/04", "time": "11:28:39", "price": 4.27}
{"day": "2012/12/03", "time": "05:00:23", "price": 4.10}
{"day": "2012/12/06", "time": "05:27:58", "price": 4.30}
{"day": "2012/12/04", "time": "08:34:50", "price": 4.01}
```

  - nous voulons obtenir la liste de dernières cotations par jour, soit :

```
[
  {"day": "2012/12/03", "time": "05:00:23", "price": 4.10},
  {"day": "2012/12/04", "time": "11:28:39", "price": 4.27},
  {"day": "2012/12/06", "time": "05:27:58", "price": 4.30}
]
```


Agrégation

Commande `group` (2)

- La commande ressemble à :

```
> db.runCommand({"group" : {
... "ns": "stocks",
... "key": "day",
... "initial": {"time": 0},
... "$reduce": function(doc, prev) {
...      if (doc.time > prev.time) {
...           prev.price = doc.price;
...           prev.time = doc.time;
...      }
... }}})
```

Agrégation

Commande `group` (3) - obligatoires

- `"ns"` :
  - nom de la collection sur laquelle travailler
- `"key"` :
  - nom de la clé de regroupement
- `"initial"` :
  - définit le document d' "accumulation" initial qui sera créé et passé à la fonction de réduction pour chaque groupe
- `"$reduce"` :
  - fonction de réduction appelée pour chaque document de la collection
  - reçoit deux paramètres : le document courant et le document "d'accumulation"

Agrégation

Commande `group` (4) - optionnels

- `"keyf"` :
  - permet de définir une fonction qui va retourner la valeur de la clé de regroupement
    - quand la valeur n'est pas la valeur d'une clé (colonne)
- `"condition"` :
  - permet d'ajouter une condition de filtrage sur les objets `"condition": {"day": {"$gt": ... }}`
- `"finalize"` :
  - fonction appelée sur chaque élément retourné et peut les modifier

Agrégation

Utilisation de `finalize`

- Exemple :
  - soit une collection nommée `post` qui correspond aux post d'un blog
  - chaque post possède un ou plusieurs tags
  - nous voulons connaitre les tags les plus populaires pour chaque jour

```
> db.posts.group({
... "key": {"tags": true},
... "initial": {"tags": {}},
... "reduce": function(doc, prev) {
...     for (i in doc.tags) {
...         if (doc.tags[i] in prev.tags) {
...            prev.tags[doc.tags[i]]++;
...         } else {
...            prev.tags[doc.tags[i]] = 1;
...         }
...     }
... })
```


Agrégation

Utilisation de `finalize` (2)

- Exemple :
  - nous voulons connaitre le nom du tag le plus cité pour chaque jour

```
... "finalize": function(prev) {
...   var mostPopular = 0;
...   for (i in prev.tags) {
...      if (prev.tags[i] > mostPopular) {
...         prev.tag = i;
...         mostPopular = prev.tags[i];
...      }
...   }
...   delete prev.tags;
... }})
```

  - les "tags" on été remplacés par un unique "tag"

Agrégation

Utilisation de `keyf`

- Scénario :
  - il faut effectuer le groupement sur une clé mais les valeurs peuvent contenir minuscules et majuscules
  - ou bien, il faut regrouper selon le jour de la semaine

```
... "$keyf": function(x) {
...    return x.category.toLowerCase();
... }

... "$keyf": function (x) {
...    return day_of_week: x.une_date.getDay();
... }
```


Agrégation

Remarques sur la commande `group`

- Le shell propose la méthode `db.coll.group()`
  - les clés `$keyf` et `$reduce` s'appellent `keyf` et `reduce`
- La commande `group` pose un verrou en lecture et bloque les autres threads d'exécuter du JavaScript
- La commande `group` ne fonctionne pas avec les *sharded clusters*
  - il faut utiliser une solution alternative

↳ pas de // → cf MapReduce

Agrégation

Algorithme `MapReduce`

- Sur-ensemble de `distinct`, `count`, `group`
- Utilisé pour les opérations d'agrégation complexes
- Permet l'exécution en parallèle sur plusieurs serveurs
- Travaille selon deux étapes :
  - mapping :
    - appliquée sur tous les objets, permet de générer des clés/valeurs
  - reduction :
    - prend les listes de clés/valeurs et fabrique un seul élément

Agrégation

Fonction d'association (*mapping*)

- La fonction "`map`" qui est invoquée sur chaque objet et qui doit renvoyer un couple clé/valeur à l'aide de la fonction `emit()`

```
> map = function() {
... for (var key in this) {
...     emit(key, {count: 1});
... }};
```

  - `this` permet de référencer l'objet
  - `map()` ne doit pas appeler la base
  - plusieurs appels à `emit()` sont possibles

Agrégation

Fonction de réduction

- La fonction "`reduce`" va être appelée pour chaque valeur de la clé retournée par map
- Le 2ème paramètre sera le tableau des valeurs associées à la clé :

```
> reduce = function(key, emits) {
...    total = 0;
...    for (var i in emits) {
...       total += emits[i].count;
...    }
...    return {"count": total};
... }
```

  - "reduce" doit retourner un élément qui peut être reçu comme 2ème paramètre !
  - ne peut faire appel à la base

247

J.Delamarche (2002-2015)

Agrégation

Commande `mapReduce`

- Syntaxe simplifiée :

```
> nr = db.runCommand({"mapreduce" : "coll", "map": map, "reduce": reduce})
{
  "result: "tmp.....",
  "timeMillis": 12,
  "counts" : {
    "input": 6,
    "emit": 14,
    "output": 5
  },
  "ok": true
}
```

  - `result` : nom de la collection temporaire créée
  - `input` : nombre de documents envoyés à map
  - `emit` : nombre d'appels à `emit()`
  - `output` : nombre de documents créés

248

J.Delamarche (2002-2015)

Agrégation

Commande `mapReduce` (2)

- Paramètres optionnels :
  - `out` : permet de spécifier une collection qui va recevoir les résultats
    - `{ out : nom_collection }`
    - si la collection existe :
      - `{ out : { action: nom_coll, [ db: dbname] [, sharded: boolean] [, nonAtomic: boolean] }}`
      - l'action peut être : `replace`, `merge` ou `reduce`
    - `{ out : { inline : 1 }}`
      - effectue l'opération de map-reduce en mémoire
  - `query` : filtre de sélection des documents
  - `sort` : critère de tri des documents en entrée
  - `limit` : nombre maximum d'éléments retournés

Agrégation

Commande `mapReduce` (3)

- Paramètres optionnels (suite) :
  - `finalize` : une fonction JavaScript appelée après la réduction, pour chaque clé
    - son prototype est :

```
function(key, reducedValue) {
   ... return modifiedObject;
}
```

  - `scope` : spécifie les variables globales accessibles par `map`, `reduce` et `finalize`
  - `jsMode` : (`false` par défaut)
    - mettre à true si le nb de clés est < 500.000 (évite des conversions JavaScript => BSON => JavaScript)
  - `verbose` : (`true` par défaut) inclut les informations de timing

Agrégation

En cas de problème...

- On peut vérifier les valeurs des paires émises par la fonction `map` en écrivant notre propre fonction `emit` :

```
var emit = function(key, value) {
  print("emit");
  print("key: " + key + " value: " + tojson(value));
}
```

- On vérifie que la fonction de réduction retourne une valeur du même type que celle émise par la fonction map
- Vérifier que la fonction de réduction donne le même résultat quel que soit l'ordre des éléments passés dans le tableau

Agrégation

En cas de problème... (2)

- On vérifie que la fonction de réduction retourne une valeur du même type que celle émise par la fonction map

```
var emit = function(key, value) {
  print("emit");
  print("key: " + key + " value: " + tojson(value));
}
```

Agrégation

Le framework d'agrégation (v2.2+)

- Objectif :
  - simplifier l'utilisation de *mapReduce* pour des calculs d'agrégation simples comme "trouver la moyenne"
- Composants :
  - les pipelines :
    - examinent une collection de documents et les transforme
  - les expressions :
    - produisent des documents en sortie à partir de documents en entrée
- Utilisation :
  - appeler la fonction (ou la commande) `aggregate()` en lui passant une collection, les pipelines et les opérateurs

Agrégation

Exemple d'agrégation

```
{
  title : "this is my title" ,
  author : "bob" ,
  posted : new Date () ,
  pageViews : 5 ,
  tags : [ "fun" , "good" , "fun" ] ,
  comments : [
              { author :"joe" , text : "this is cool" } ,
              { author :"sam" , text : "this is bad" }
            ],
  other : { foo : 5 }
}
```

```
db.articles.aggregate(
  { $project : { author : 1, tags : 1, } },
  { $unwind : "$tags" },
  { $group : {
      _id : { tags : "$tags" },
      authors : { $addToSet : "$author" }
  }
} );
```

Retourne un ensemble de noms d'auteurs groupés par tags.

Liste des opérateurs de calcul

- Opérateurs booléens :
  - `$and`, `$or`, `$not`
- Opérateurs de comparaisons :
  - `$cmp`, `$eq`, `$gt`, `$gte`, `$lt`, `$lte`, `$nte`
- Opérateurs arithmétiques :
  - `$add`, `$divide`, `$mod`, `$multiply`, `$substract`
- Opérateurs de chaine :
  - `$strcasecmp`, `$substr`, `$toLower`, `$toUpper`, `$concat`
- Opérateurs de date :
  - `$dayOfYear`, `$dayOfMonth`, `$dayOfWeek`, `$year`, `$month`, `$week`, `$hour`, `$minute`, `$second`
- Opérateurs conditionnels :
  - `$cond`, `$ifNull`

Administration

Administration

Démarrer MongoDB

- Par invocation du binaire `mongod`
  - `--help`
  - `--dbpath répertoire`
    - indique le répertoire de stockage des données
    - crée un verrou `mongod.lock` pour éviter qu'un autre processus n'utilise les mêmes fichiers
  - `--port numéro`
    - par défaut, c'est le 27017
  - `--fork`
    - pour que `mongod` devienne un démon
  - `--logpath fichier`
    - fichier de log (utilisation de `--logappend` possible)
  - `--config file`
    - nom du fichier contenant les options de configuration
  - `--setParameter`
    - permet de donner une valeur à certains autres paramètres (v2.4)

Administration

Fichier de configuration

- Plutôt que de donner des options sur la ligne de commande, on peut les placer dans un fichier et utiliser l'unique option :
  - `--config nom_fichier`
- La syntaxe est triviale :

```
# Fichier de configuration

logpath = /var/log/mongodb/mongod.log
logappend = true
fork = true
port = 27017
dbpath = /var/lib/mongodb/
```

Administration

Configuration dynamique

- La commande `setParameter` permet de modifier dynamiquement certains paramètres :
  - `> use admin`
  - `> db.runCommand( { setParameter: 1, nom_param: valeur })`
- Les paramètres supportés sont :
  - `journalCommitInterval`
  - `logLevel, logUserIds, traceException`
  - `notableScan, textSearchEnabled`
  - `quiet`
  - `replApplyBatchSize, replIndexPrefetch`
  - `syncDelay`

Administration

Arrêt de MongoDB

- Il suffit d'envoyer un signal SIGINT ou SIGTERM au processus `mongod`
  - l'arrêt est "propre" : les requêtes en cours sont achevées, de même que les allocations de fichiers, etc...
- Autre solution : via la commande `shutdown`

```
> use admin
switched to db admin
> db.shutdownServer()
server should be down...
```

Administration

Surveillance du service

- Via une interface HTTP rudimentaire :
  - `mongod` se met à l'écoute du port de service + 1000
    - soit 28017 par défaut
  - mode REST avec l'option `--rest`
  - désactivation avec l'option `--nohttpinterface`
- Via la commande `serverStatus`
  - `> db.runCommand({"serverStatus": 1})`
- Via la commande `mongostat`

Administration

Informations collectées

- `globalLock`
  - indique le temps global où un verrou en écriture a été posé (les temps sont exprimés en microsecondes)
- `mem`
  - informations sur l'occupation mémoire (en Mo)
- `indexCounters`
  - affiche le taux de réussite d'utilisation des index
- `opcounters`
  - compte le nb d'occurrences de chaque opération
  - si les compteurs débordent : tous sont remis à 0 et "`rollovers`" est incrémenté

Administration

Sécurité et authentification

- MongoDB propose un système d'authentification basé sur des noms d'utilisateurs associés à chaque base
- La base `admin` est traitée spécialement : les utilisateurs de cette base sont des "superusers" et peuvent accéder sans restriction à toutes les bases

Administration

Mise en œuvre de l'authentification

- `mongod` est activé sans authentification
- On crée le 1er compte d'administration :
  - `> use admin`
  - `> db.addUser("root","1234");`
- On crée des comptes par bases :
  - `> use test`
  - `> db.addUser("user1","pwd1");`
  - `> db.addUser("ronly","pwd2",true);`
    - le 3ème paramètre permet de spécifier le mode "*read only*"
    - `addUser()` permet aussi de modifier les mots de passe

Administration

Mise en œuvre de l'authentification (2)

- Activation de `mongod` avec support de l'authentification :
  - `mongod --auth`
- Tests de connexion :
  - `> use test`
  - `> db.test.find()`
  - `error: …`
  - `> db.auth("user1","pwd1");`
  - `> db.test.find()`

269

J.Delamarche (2002-2015)

Administration

Implémentation de l'authentification

- Les utilisateurs d'une base sont stockés dans la collection `system.users` de la base
  - chaque entrée contient 3 attributs : `"user"`, `"readOnly"` et `"pwd"`
- Donc pour supprimer un utilisateur :
  - `> use test`
  - `> db.system.users.remove({"user": "user1"});`
  - `> db.auth("user1","pwd1");`
  - `0`

270

J.Delamarche (2002-2015)

Administration

Cas de "localhost" (v2.2+)

- Si aucun utilisateur n'est défini dans la base `admin`, il est possible de se connecter à partir de *localhost*
- Pour les clusters MongoDB, il faut toujours s'authentifier même si la base `admin` est vide (v2.2+)

Administration

Utilisation des rôles (v2.4)

- Par défaut la v2.4 est compatible avec les versions inférieures
  - possibilité de désactivation au démarrage par :

```
--setParameter
 supportCompatibilityFormPrivilegeDocuments=0
```

- Les rôles donnent aux utilisateurs des privilèges

Administration

Types de rôles

- Database User Roles :
  - `read` : permet le `find()`, `count()`, `distinct()`, `mapReduce()`...
  - `readWrite` : permet `insert()`, `remove()`, `update()`, `drop()`...
- Database Administration Roles :
  - `dbAdmin` : permet `create()`, `drop()`, `ensureIndex()`...
  - `userAdmin` : permet de lire et écrite la collection `system.users` d'une base = administrateur de la base !

Administration

Types de rôles (2)

- Administrative Roles :
  - `clusterAdmin` : ne s'applique qu'à la base `admin` et donne les privilèges de gestion des clusters et des replica-sets
- Any Database Roles :
  - ces privilèges s'appliquent à toutes les bases et doivent être définis dans la base `admin`
  - `readAnyDatabase` = `read` dans toutes les bases
  - `readWriteAnyDatabase` = `readWrite` dans toutes les bases
  - `userAdminAnyDatabase` = `userAdmin` sur toutes les bases
  - `dbAdminAnyDatabase` = `dbAdmin` sur toutes les bases

Administration

Stockage dans `system.users`

- Dans chaque base, la collection `system.users` contient les noms des utilisateurs, leur mot de passe et leurs rôles éventuels :
  - exemples :
    - `{ user: "<username>", pwd: "<hash>", roles: [], }`
    - `{ user: "<username>", userSource: "<database>" roles: [], }`
    - `{ user: "admin", userSource: "$external", roles: [ "clusterAdmin"], otherDBRoles: { config: [ "read" ] records: [ "dbadmin" ] } }`

Administration

Stockage dans `system.users` (2)

- `userSource` indique que les "credentials" de l'utilisateur sont stockés dans une autre base
  - la valeur `$external` permet de référencer des systèmes d'authentification commer Kerberos V (version payante)
  - `userSource` et `pwd` sont exclusifs
- La base `admin` permet de stocker les rôles supplémentaires `otherDBRoles`

Administration

Autres considérations sur la sécurité

- Même avec de l'authentification, le protocole n'est pas chiffré
  - nécessite un tunnel
  - ou bien une extension tierce (payante ?)
- Placer MongoDB derrière un pare-feu
- Limiter les accès sur l'adresse IP local avec l'option `--bindip localhost`
- Supprimer l'interface Web avec l'option `--nohttpinterface`
- Supprimer l'exécution de scripts sur le serveur avec l'option `--noscripting`

277

Administration

Chiffrement des communications

- Non disponible dans la distribution par défaut
  - il faut recompiler les sources
  - ou obtenir la version de "souscription"
- Nouveaux paramètres :
  - `--sslOnNormalPorts`
  - `--sslPEMKeyFile nom_fichier`
  - `--sslPEMKeyPassword mot_de_passe`
- Côté client, il faut activer *ssl* :
  - `--ssl`
  - ou `ssl=true` (selon l'API)

278

Administration

Journalisation

- `mongod` résiste au crash en utilisant un journal :
  - avant de modifier des données, le journal est mis à jour pour indiquer ce qui va être fait
  - la fenêtre de perte en cas de crash est < 100 ms
- Le journal est activé par défaut (sinon, l'activer avec l'option `--journal`)
  - on peut le désactiver avec l'option `--nojournal` !
- Le journal est créé au démarrage si nécessaire (par préallocation)
  - pour éviter le temps d'indisponibilité on peut créer un journal vide en lançant une 2ème instance sur un autre port, puis on déplace le journal ainsi créé

279

Administration

Contrôle du journal

- A l'aide des commandes :
  - `serverStatus`
  - `journalLatencyTest`
    - donne le temps d'écriture sur disque en mode append
    - à faire sur le système oisif puis chargé
    - si le temps est faible, il y a peut-être un cache disque (à désactiver ?)
- L'option `--journalCommitInterval` permet de choisir la valeur maximale (en ms) entre deux opérations d'écriture de groupes d'opérations dans le journal (défaut = 100 ms)
  - cette valeur est temporairement réduite à son tiers quand on effectue une écriture avec le drapeau `"j" : true`

280

Administration

Fonctionnement interne du journal

- Les fichiers du journal sont stockés dans un sous-répertoire du répertoire indiqué par `dbpath`
  - les journaux sont nommés `j._xxxx`
  - `xxxx` est un numéro incrémental stocké dans un fichier du même sous-répertoire
  - quand un journal atteint 1 Go, un nouveau fichier est créé (cette taille est modifiable)
  - les vieux journaux sont automatiquement supprimés
  - si on veut faire des snapshots de `dbpath`, les journaux doivent être sur le même système de fichiers

281

Administration

Backup (format JSON)

- **`mongoexport`**
  - exporte les données au format JSON
  - paramètres :
    - `--db nom_base`
    - `--collection nom_collection` : sans cette option, toutes les collections sont exportées
    - `--out fichier.json` : sans cette option, les données sont affichées sur STDOUT
    - `--dbpath /nom/répertoire` : permet d'accéder aux données sans passer par l'instance `mongod` qui doit être arrêtée
    - `--query '{"champ": 1}'` : permet de filtrer les documents sauvés

282

Administration

Restauration (format JSON)

- `mongoimport`
  - importe les données au format JSON
  - paramètres :
    - `--db nom_base`
    - `--collection nom_collection` : sans cette option, toutes les collections sont importées
    - `--in fichier.json` : sans cette option, les données sont lues sur STDIN
    - `--dbpath /nom/répertoire` : permet de recréer les données sans passer par l'instance `mongod` qui doit être arrêtée
    - `--journal` : (avec `--dbpath`) assure que l'importation enregistre les actions dans le journal
    - `--upsert` : met à jour ou insère

283

Administration

Backup binaire

- `mongodump`
  - exporte les données au format binaire
    - sans option, crée un dump dans le répertoire `dump/` courant
    - attention aux (in)compatibilités entre les versions de Mongo
  - paramètres :
    - `--db nom_base`
    - `--collection nom_collection` : sans cette option, toutes les collections sont exportées
    - `--out répertoire`
    - `--dbpath /nom/répertoire` : permet d'accéder aux données sans passer par l'instance `mongod` qui doit être arrêtée
    - `--oplog` : permet de dumper le log des opérations en cours

284

Administration

Restauration binaire

- `mongorestore`
  - importe les données au format binaire
  - paramètres :
    - `répertoire` ou `fichier` : données à restaurer
    - `--db nom_base`
    - `--collection nom_collection` : sans cette option, toutes les collections sont importées
    - `--dbpath /nom/répertoire` : permet de recréer les données sans passer par l'instance `mongod` qui doit être arrêtée
    - `--journal` : (avec `--dbpath`) assure que l'importation enregistre les actions dans le journal
    - `--oplogReplay` : rejoue l'oplog
    - `--drop` : vide les collections avant importation

285

Administration

Backup au niveau "block"

- En utilisant les mécanismes de "snapshots"
  - comme celui du LVM
- Limitations :
  - sauvegarde totale (pas d'une collection ou d'une base)
  - la base doit être dans un état fiable
  - la journalisation doit être activée, sinon les écritures en cours doivent être "flushées" et la base verrouillée
    - sans journalisation (ou si le journal n'est pas sur le même volume que les données) :
      - `db.fsyncLock()` : vide les caches et verrouille la base
      - `db.fsyncUnlock()` : déverrouille la base

286

Administration

Commandes de gestion

- Nous avons vu les commandes de gestion des documents (creat, read, update, delete)
- Il existe de nombreuses commandes de gestion des bases
  - `getLastError`
  - `ensureIndex`
  - ...

Administration

Fonctionnement des commandes

- Elles s'exécutent ainsi :
  - `> db.runCommand({"nom_cmd" : "parm", "key" : valeur ...})`
- Elles retournent un objet qui contient toujours la clé "`ok`"
  - "`ok`" vaut `true` en cas de succès et `false` sinon
  - en cas d'erreur, la clé "`errmsg`" contient le message d'erreur
- En réalité, `runCommand()` est une fonction "helper" qui équivaut à :
  - `> db.$cmd.findOne(({"nom_cmd" : "parm", "key" : valeur ...})`

Administration

Référence des commandes

- Impossible ! Il y en a autour de 100 !
- La commande `listCommands()` permet de les afficher
- `{"buildInfo": 1}`
  - retourne les infos sur le server MongoDB
- `{"collStats": nom_coll}`
  - stats sur la collection (tailles des données, des index, espace occupé)
- `{"distinct": nom_coll, "key": key, "query": query}`
  - retourne la liste des valeurs de clés distinctes

289

Administration

Référence des commandes (2)

- `{"drop": nom_coll}`
  - supprime une collection
- `{"dropDatabase": 1}`
  - supprime toutes les données de la base courante
- `{"dropIndexes": nom_coll, "index": nom}`
  - supprime l'index ou tous les index si le nom est "*"
- `{"findAndModify"... }` (déjà vu)
- `{"getLastError": 1 [, "w": w [, "wtimeout": timeout]]}`
  - retourne le status de la dernière commande en attendant une synchro de l'esclave ou bien une valeur de timeout (en ms)
- `{"listDatabases": 1}`

290

Référence des commandes (3)

- `{"ping": 1}`
  - teste la connectivité à un serveur
- `{"renameCollection": a, "to": b}`
  - renomme en utilisant des noms complets pour `a` et `b` (ex: `foo.bar`)
- `{"repairDatabase": 1}`
  - répare et compacte la base courante
- `{"serverStatus": 1}`
  - retourne des données statistiques sur l'état du serveur

Réplication

Réplication

Principe

- A l'origine :
  - une réplication maître / esclave
- Maintenant :
  - mise en oeuvre de *Replica Sets*
  - sorte de cluster maître / esclaves assurant un *failover* automatique
  - nécessite au moins 2 noeuds et 12 au maximum
  - un mécanisme d'élection élit le maître (celui reçoit les requêtes modifiantes)
  - les autres noeuds peuvent être utilisés pour les lectures
  - transparence vis-à-vis des Clients

293

Réplication

Types de noeud

- Un noeud peut être configuré pour être :
  - `secondary-only` : possède les données mais ne peut être promu primaire
    - utile s'il est géographiquement éloigné ?
  - `hidden` : invisibles pour les Clients
    - utile pour les backups, le test, le reporting ?
  - `delayed` : applique les modifications du primaire après un délai réglable ("rolling backup")
    - utile pour corriger les erreurs humaines ?
  - `arbitre` : ne possède pas de donnée, ne sert qu'à l'élection
  - `non-voting` : pour les sets plus larges
    - seuls 7 noeuds peuvent voter
    - ou en cas de partitionnement du réseau ?

294

Réplication

Failover et élection

- Si le primaire devient indisponible et qu'une majorité de noeud peuvent communiquer entre eux :
  - ils vont élire un nouveau primaire si une majorité de votes se dégage
  - le nouveau primaire doit être celui qui possède les données les plus récentes
  - le paramètre `members[n].priority` permet de donner une priorité à chaque noeud (1 par défaut)
- Le primaire effectue des *heartbeats* avec les secondaires et si le nombre des noeuds joints est inférieur à la majorité, il redevient un secondaire

Réplication

Exemples de mise en oeuvre

- Avec un cluster de 3 noeuds :
  - chacun doit être lancé avec l'option `--replSet` suivie du nom du *replica set*
  - le *replica set* est initialisé avec `rs.initiate()`
  - les noeuds sont ajoutés par `rs.add()`
  - la configuration est consultable par `rs.conf()`
  - l'état du *replica* est consultable à tout moment par `rs.status()`
- Note :
  - les opérations d'administration doivent généralement être faites sur le primary, c'est à dire que :
    `> db.isMaster()` doit retourner `{"ok": 1}`

Réplication

Conversion d'un noeud en replica set

- Il faut arrêter l'instance `mongod` et la relancer avec l'option `--replSet`
- Utiliser un client `mongo` pour se connecter sur l'instance et créer le *replica* par :

```
> rs.initiate()
```

- Ajouter les autres noeuds comme précédemment avec `rs.add()`

Réplication

Le log des opérations (OpLog)

- Collection limitée (capped) qui contient les modifications effectuées par le primaire
- Chaque secondaire maintient une copie de l'OpLog et rejoue localement son contenu
- La commande `db.printReplicationInfo()` affiche la taille de l'OpLog
  - la taille peut être modifiée par le paramètre `oplogSize`

Réplication

Configuration du *replica set*

- La commande `rs.conf()` affiche la configuration
- Il est possible de modifier l'objet et l'utiliser pour reconfigurer le replica avec `rs.reconfigure()` :

```
> cfg = rs.conf()
> cfg.members[0].priority = 0
> ...
> rs.reconfig(cfg)
```

  - note : la commande `rs.reconfig()` doit être faite sur un noeud susceptible de devenir un *primary*

Réplication

Choisir le type des noeuds

- Secondary-Only :
  - `members[n].priority = 0`
- Hidden :
  - `members[n].priority = 0`
  - `members[n].hidden = true`
- Delayed :
  - `members[n].priority = 0`
  - `members[n].slaveDelay = 3600`
    - `members[n].hidden` devient `true`
- Non-voting :
  - `members[n].votes = 0`

Réplication

Cas du noeud de type Arbitre

- Utile si le nombre de noeuds est pair
- Créer un répertoire pour stocker les informations du replicat mais pas les données :
  - `mkdir /data/arb`
- Lancer une instance spéciale de `mongod` :
  - `mongod --port 3333 --dbpath /data/arb --replSet rs0`
- Se connecter sur le primaire et lancer la commande :

```
> rs.addArb('nom_host:3333')
```


Réplication

Ajouter un noeud

- Etat initial :
  - soit le répertoire des données est vide
  - soit on y copie les données d'un autre noeud préalablement arrêté ou verrouillé (`db.fsyncLock()`)
    - attention à la "fenêtre des OpLogs"
- Sur le primary, ajouter le noeud :

```
> rs.add('nom:port')
```

Réplication

Supprimer un noeud

- Stopper `mongod`
- Sur le primary, retirer le noeud :

```
> rs.remove('nom:port')
```

- Si un noeud change de nom, il faut le "remplacer" :
  - `...`
  - `members[n].host = "nouveau_nom:port"`
  - `rs.reconfig(cfg)`

Réplication

Autres opérations d'administration

- Ajuster la priorité d'un noeud
- Changer la taille de l'OpLog
  - implique un arrêt du primary, une modification de l'oplog et un redémarrage comme secondaire
- Resynchroniser un noeud du replica set
  - soit en arrêtant `mongod` et en repartant d'un répertoire vide (comme lors de l'ajout d'un noeud)
  - soit en recopiant un snapshot s'un autre noeud

Securité : le *keyfile*

- Pour renforcer la sécurité de la communication entre les noeuds, il est possible d'utiliser un secret partagé : le *keyfile*
  - avec l'option `keyfile` du fichier de configuration
  - avec l'option `--keyFile`
- Caractéristiques du fichier *keyfile* :
  - fait moins de 1 Ko
  - codé en Base64
  - aucune permission `rwx` pour `go`
  - exemple de génération :
    - `openssl rand -base64 753`

Accès au Replicat par les clients

- Le *replica set* doit être transparent

Réplication

Options d'écriture

- Après une écriture, il est possible de demander des confirmations de bonne propagation sur les secondaires :
  - la commande `gestLastError` permet d'obtenir l'erreur ou les informations après l'écriture
  - l'attribut `w` permet d'indiquer le nombre de noeuds dont on souhaite un acquittement d'écriture
    - la valeur "`majority`" est possible
  - si le nombre demandé ne peut être atteint (des noeuds sont HS), l'attribut `wtimeout` (exprimé en ms) permet de débloquer l'appel

Réplication

Options d'écriture (2)

- Exemple de configuration :
  - `db.runCommand({getLastError: 1, w: 2, wtimeout: 5000})`
- Configuration par défaut :
  - `cfg = rs.conf()`
  - `cfg.settings = {}`
  - `cfg.settings.getLastErrorDefaults = {w: 2, wtimeout: 5000}`
  - `rs.reconfig(cfg)`

Replica Set : - Haute disponibilité
- Répartition de la charge de ceux qui veulent lire.

Options de lecture

- Il est possible d'indiquer des préférences de lecture :
  - par défaut les lectures sont faites sur le primaire
  - le client peut indiquer ses préférences avec la commande `readPref()` :
    - `primary` : valeur par défaut
    - `primaryPreferred` : lecture sur le primaire puis sur un secondaire si indisponible
    - `secondary` : lecture uniquement sur un secondaire
    - `secondaryPreferred` : secondaire mais s'il n'y en a pas, lit depuis le primaire
    - `nearest` : choisit le plus près en terme de temps de réponse
  - les "*tag sets*" permettent de définir des préférences personnalisées

Cluster & Sharding

Partitionnement = Sharding

Cluster

Introduction

- MongoDB permet de réaliser des Clusters en répartissant automatiquement les données sur un ensemble de noeuds
- La technique de partitionnement s'appelle le "*sharding*" (*shared-nothing*)
- On choisit les collections qui seront partitionnées
- Le partitionnement est basé sur une clé de "*sharding*" : c'est un champ qui doit exister dans chaque document
- Les documents sont regroupés par *chunks* dans chaque noeud du cluster

311

Cluster

Clés de partitionnement et *Chunks*

- La clé de *sharding* permet d'associer un document à un chunk
- Les *chunks* peuvent migrer d'un noeud à un autre afin d'optimiser la répartition des données
- Exemple :
  - 4 *shards* contiennent 500 Go de données
  - on ajoute un nouveau *shard*
  - il faut idéalement y transférer 100 Go de données (sous forme de *chunks*) en provenance des *shards* existants
- Les *chunks* ont une taille maximale et sont découpés en 2 parties quand ils la dépasse

312

Cluster

Choix de la *shard key*

- Champs simples ou multiples qui existent dans tous les documents
- Idéalement :
  - possède une forte cardinalité :
    - si la clé possède peu de valeurs, il y aura peu de *chunks* !
  - possède une bonne distribution lors des écritures :
    - si la clé est la date courante : toutes les écritures iront sur le dernier *chunk*
  - rend les requêtes efficaces et faisant en sorte qu'une seule instance `mongod` soit consultée
- La *shard key* doit aussi être une *primary key*

313

Cluster

Clé hachée (v2.4)

- Pour faciliter l'équilibrage de la répartition des données dans les shards, il est possible d'utiliser une *hash key* :
  - on pose un index de type "hash" sur un champ
    - attention : il y a des restrictions
    - attention avec les flottants => les valeurs utilisées pour le calcul du hash sont les parties entières
  - l'index haché servira aux tests d'égalité
    - mais ne peut être utilisé pour les tests d'inégalité
  - déclaration d'un index haché :

```
> db.coll.ensureIndex({a : "hashed"})
```

  - déclaration du sharding avec clé hachée :

```
> db.shardCollection("base.coll", {a: "hashed"})
```

314

Cluster

Composants du cluster

- Des serveurs de configuration (`mongod`) :
  - maintiennent, dans la base `config`, les méta-données du cluster et déterminent quel *shard* est responsable de quel *chunk*
    - en production, il en faut 3 pour la fiabilité
- Des shards (`mongod`) :
  - au moins deux instances qui stockent les données
    - le *Primary Shard* contient les collections non-shardées
  - en production, chaque `shard` est un *replica set*
- Des noeuds de requêtage (`mongos`) :
  - utilisés par les clients pour interroger les *shards*
  - stockent les méta-données en cache mémoire

315

Cluster

Déploiement d'un cluster

- Déployer les serveurs de configuration
  - instances `mongod` avec l'option `--configsvr`
- Démarrer des instances de routage (`mongos`)
  - avec l'option `--configdb` qui indique la localisation des serveurs de configuration
- Activer les *shards*
  - ces sont des instances `mongod` "normales"
  - se connecter sur une instance `mongos` et ajouter les *shards* :

```
> sh.addShard("nom_shard:port")
```

  - si le *shard* est un *replica set* :

```
> sh.addShard("nom_replica/nom_noeud:port")
```

316

Cluster

Déploiement d'un cluster (2)

- Activer le *sharding* pour des bases
  - ne redistribue pas les données, seulement les méta-données :

```
> sh.enableSharding("nom_base")
```

- Activer le *sharding* pour des collections

```
> sh.shardCollection("nom_coll", { key: 1})
```

- L'état du cluster peut être consulté ainsi :

```
> sh.status()
```

317

Cluster

Administration du cluster

- Afficher la liste des *shards* :
  - `mongos> use admin`
  - `mongos> db.runCommand({listShards: 1})`
- Afficher les bases clusterisées :
  - `mongos> use config`
  - `mongos> db.databases.find({ partitioned: true })`
- Voir le détail du cluster :
  - `mongos> sh.status()`

318

Cluster

Administration du cluster (2)

- Ajout d'un *shard*

```
mongos> sh.addShard("nouveau_shard")
```

- Suppression d'un shard

```
mongos> db.runCommand({removeShard:
  "nom_shard")}
```

  - démarre la migration des *chunks*
  - rappeler la commande pour savoir quand la migration est achevée

- si le *shard* est le *primary shard* d'une base, il faut ensuite migrer les méta-données de la base :

```
mongos> db.runCommand({movePrimary: "mabase", to:
  "autre_shard"})
mongos> db.runCommand({removeShard: "nom_shard"})
```


Cluster

Gestion des Chunks

- Effectuée automatiquement par MongoDB
- Mais possibilité de réaliser des opérations par anticipation :
  - `sh.splitAt()` et `sh.splitFind()` permettent de scinder un *chunk* en fonction d'une position ou d'une valeur
    - pour anticiper l'insertion de nouvelles valeurs ?
  - `moveChunk` : déplace un *chunk* sur un *shard*
- Modification de la taille des chunks :

```
> use config
> db.settings.save({_id: "chunksize",
  value: <taille> })
```

Cluster

Cluster Balancer

- Process responsable de la migration des *chunks*
  - lancé par une instance `mongos` qui pose alors un verrou dans la collection `lock`
- Migre les *chunks* quand la répartition n'est pas égale
  - pour éviter un overload CPU ou I/O ou de bande passante, utilise une valeur seuil : *migration threshold*
    - différence entre le *shard* qui possède le plus de *chunks* et celui qui en possède le moins
    - valeurs par défaut : 2,4,8 selon le nombre de *chunks* (<20, 21-80 ou >80)

Cluster

Gestion du Balancer

- Pour vérifier si le Balancer est actif :

```
> sh.getBalancerState()
```

  - `state = 2` montre que le Balancer est actif
- Arrêter ou redémarrer le Balancer :

```
> sh.stopBalancer()
> sh.startBalancer()
```

- Possibilité de définir une fenêtre de temps pour réaliser la migration des *chunks* :

```
> use config
> db.settings.update({_id: "balancer"}, {$set :
  {activeWindow: {start: "23:00", stop: "06;30"}}})
```

- Pour supprimer la fenêtre :

```
> db.settings.update({_id: "balancer"}, {$unset:
  {activeWindow: true}})
```

Cluster

Sécurisation du cluster

- Le *keyfile* des *replica sets* peut aussi être utilisé pour sécuriser les communications entre les processus `mongod` et mongos

Cluster

Backup et restauration

- En utilisant `mongodump` / `mongorestore`
- Avec les clichés du système de fichiers
  - arrêter le Balancer d'abord !
- Ne pas oublier les données de configuration
  - la base `config` !
- La documentation officielle donne une procédure :
  - arrêter le Balancer
  - arrêter un membre de chaque *replica set*
  - arrêter un serveur de configuration
  - sauver la base `config` et le journal
  - sauver les *shards*
  - relancer les processus et le Balancer

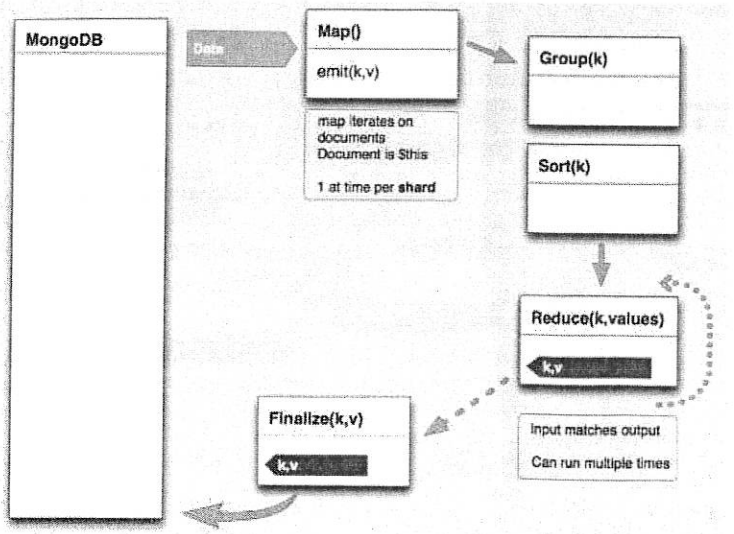
Hadoop

Intégration avec Hadoop

- MongoDB stocke les données
- Hadoop est utilisé par le Framework MapReduce
  - plus rapide et plus puissant que JavaScript !
  - ne charge pas les noeuds de données MongoDB
  - meilleur parallélisme
- Nécessite un connecteur spécifique ainsi que l'API Java pour MongoDB

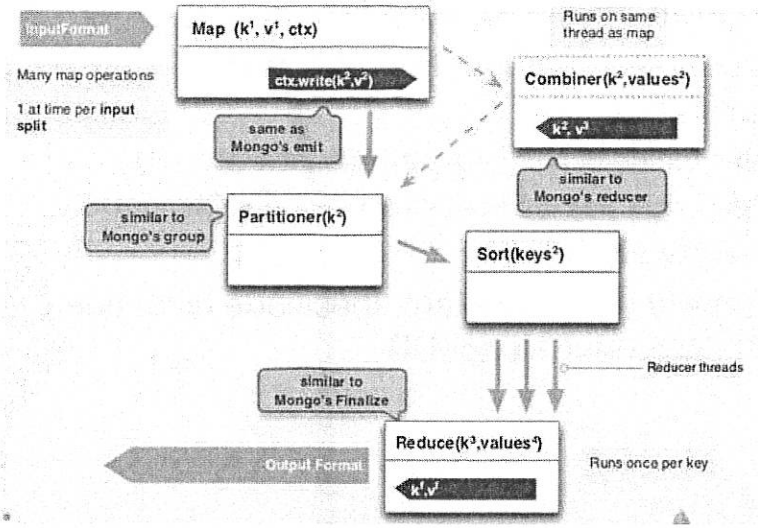
Hadoop

Rappel MapReduce MongoDB



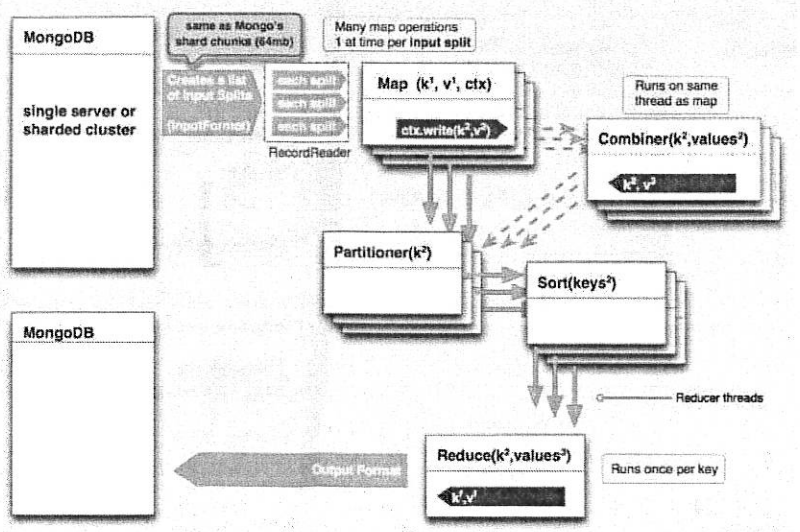
Hadoop

Rappel MapReduce Hadoop



Hadoop

MapReduce via Hadoop



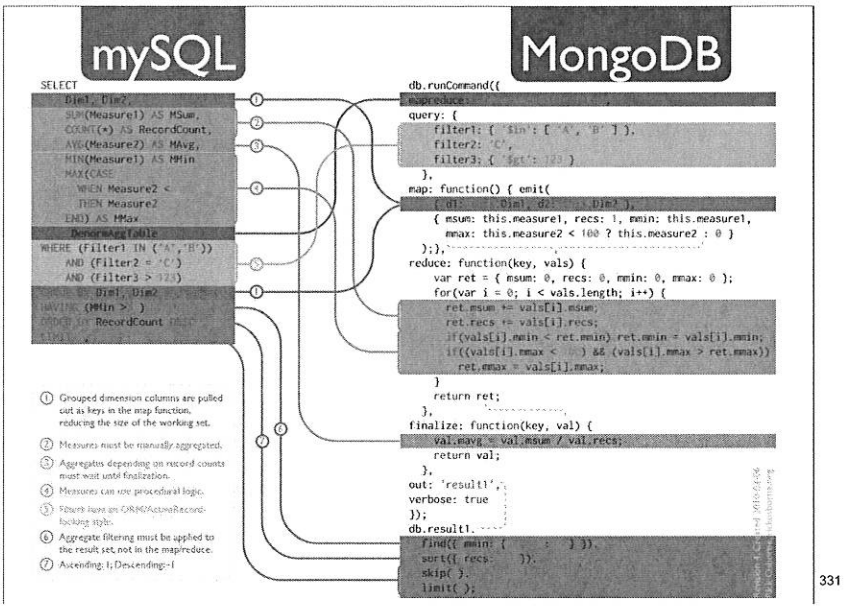
Conclusion

Conclusion

- LE SGBD le plus en vogue, capable de gérer le Big Data

MongoDB

Comparaison SQL et MongoDB



MongoDB

Roadmap

- Prise en compte du déploiement multi-site
  - implémenté dans la v2.2 !
- Intégration de Kerberos V, LDAP/AD pour l'authentification et le chiffrement
- Du hachage sur les clés de sharding
- Nouvel algorithme de Map/Reduce (v8)
- Calcul d'intersection de polygones et utilisation des geo-keys pour le sharding
- Recherche full-text
- Possibilité d'audit
- Sécurité au niveau des attributs...

Comparaison SQL et MongoDB