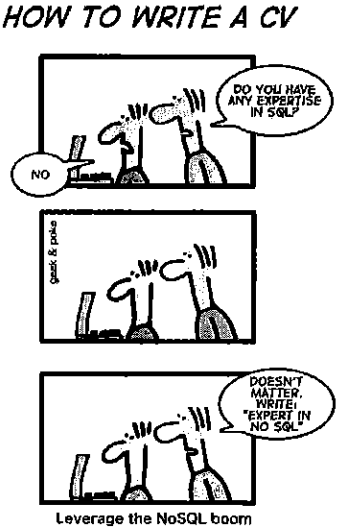
NoSQL, enjeux et solutions

Bienvenue



Leverage the NoSQL boom

Présentation

Rudi Bruchez
http://rudi.developpez.com/
www.babaluga.com – rudi@babaluga.com
consultant et formateur Bases de données

GUSS

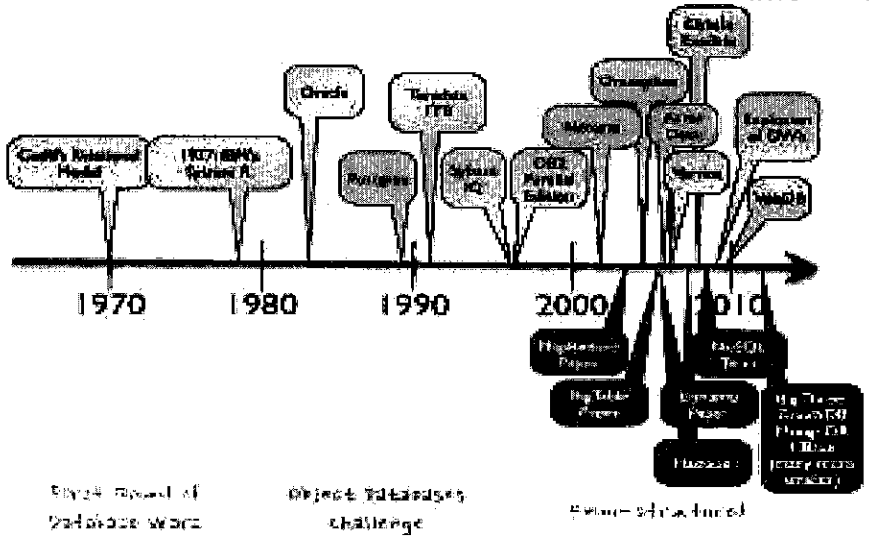
Programme

- Introduction au NoSQL
- Différences entre Relationnel et NoSQL
- Les choix techniques du NoSQL
- Les différentes bases NoSQL libres
- Mettre en place une solution NoSQL
- Maintenir et superviser ses bases NoSQL
- Aller ou non vers le NoSQL
- Comment se présente le futur ?

Introduction au NoSQL

- **Introduction au NoSQL**
  - Historique du mouvement NoSQL
  - Les différentes approches de gestion de bases de données
  - Les bases hiérarchiques, le modèle relationnel, les bases objets, les bases XML, le NoSQL
  - Les différences entre les bases relationnelles sur le Cloud, et NoSQL
  - Pourquoi le NoSQL ?
  - Quelques scénarios d'utilisation

Historique



http://www.benstopford.com/2012/06/30/thoughts-on-big-data-technologies-part-1/

Différentes approches

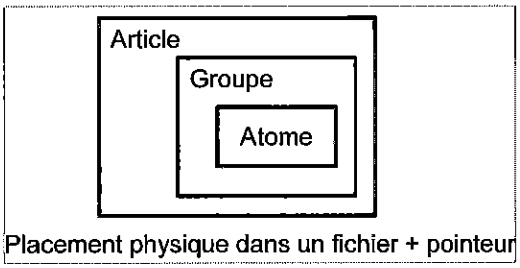
- ISAM – dépendance physique du fichier par rapport au stockage
- Réseau et hiérarchique
- Relationnel
- Objet
- XML
- Graphe
- Document

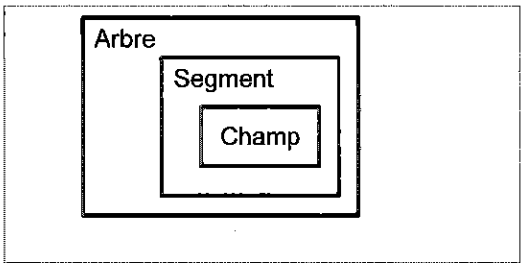
Bases de données réseau et hiérarchiques

- Modèles anciens, basés sur une dépendance aux fichiers

- **Réseau** – CODASYL
- IMS → COBOL = curseurs
- Il y a des relations, basées sur du 1-plusieurs

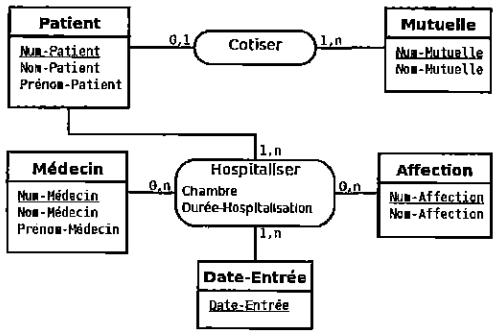
- **Hiérarchique**
- IMS → COBOL = DL1
- Structure en arbre pour représenter des associations père-fils





Pour parcourir un arbre (bases hiérarchiques), il faut faire de la navigation. DL1 est (pardon, était) un langage navigationnel. Les curseurs permettent de mémoriser un positionnement dans l'arbre. Le curseur se balade dans des adresses de segments. On considère des notions de niveau, de chemin, de parents, etc.

Le modèle relationnel



- Schéma proche de la réalité du business
- Normalisation = la donnée est présente une seule fois
- Accès optimisé
- Contraintes dans le modèle
- Nécessité d'une modélisation préliminaire
- Conception solide

Bases de données orientées objet

- Stockent des représentations de classes
- Gèrent l'héritage et les associations
- Parfois incluent les méthodes
- Définition dans la norme SQL
- Hybridation avec les SGBDR = objet-relationnel
  - assemblies .NET dans SQL Server,
  - AS OBJECT dans Oracle
- Db4o – libre
  - Stocke directement des objets
- Tout ceci n'est pas très intéressant en terme de recherches

Bases XML

- Natives :
  - Stockent directement du XML
  - BaseX – stocke dans une représentation tabulaire, REST et différents API d'accès
  - Exist – Xquery, REST, et différents API d'accès
- Hybrides : toujours Oracle et SQL Server
  - Types natifs (selon la norme SQL:2003)
  - Support du Xquery
  - Indexation en B-tree

L'émergence du Cloud

Niveau de cloud	Service
Application	Software as a Service (SaaS) : l'application est découpée en services
Plate-forme	Platform as a Service (PaaS) : la plate-forme est granulaire
Infrastructure	Infrastructure as a Service (IaaS) : l'infrastructure est virtualisée
Données	Data as a Service (DaaS) : les données sont disponibles sur le réseau

Cloud – Database as a service

Microsoft

- Relationnel
  - SQL Azure
- NoSQL
  - MongoDB
  - Neo4J
  - Hadoop

Amazon

- Relationnel
  - Amazon Relational Database Service
- NoSQL
  - SimpleDB
  - Neo4J
  - Hadoop
  - DynamoDB

Rien de nouveau sous le soleil

Naissance du NoSQL

- **Google**
- Papiers sur leurs technologies
  - MapReduce
  - Système de fichiers distribué
  - Base de données orientée colonne

- **Amazon**
- Papier sur leur développement d'un entrepôt de paires clé-valeur = Dynamo
- Utilisé pour gérer le panier d'achat sur les sites d'Amazon → solidité prouvée
- DynamoDB
  http://aws.amazon.com/fr/dynamodb/

Naissance du NoSQL

- **Google →**
- MapReduce → Hadoop
  - Doug Cutting à Yahoo, pour Nutch
- Système de fichiers distribué → HDFS
- BD orientée colonne →
  - Hbase
  - Hypertable
  - Cassandra (Facebook)

- **Amazon →**
- Dynamo a généré un grand nombre de projets
  - Project Voldemort (LinkedIn)
  - Riak (Basho)

Pourquoi le NoSQL

- **Big Data**
- Trop de TB, on doit trouver de nouvelles solutions
- Les contraintes des moteurs relationnels sont trop lourds
- Le contexte matériel a évolué
- Usage web : charge impossible à prévoir

- **Performances et aisance**
- "web scale"
- Modélisation et jointures
- Défaut d'impédance
- Langage déclaratif
- Esprit open source
- Distribution
- Cache mémoire

Différences entre Relationnel et NoSQL

- **Différences entre Relationnel et NoSQL**
  - Les bases de données relationnelles : leurs forces et leurs limites
  - Le transactionnel et l'ACID
  - La structuration forte des données
  - Comment choisir, comment gérer l'interopérabilité

L'esprit

- **SGBDR**
- Structuration forte, modèle contraignant et méticuleusement pensé
- Ère des logiciels commerciaux
- Maximisation sur une seule machine
- Demande de l'expertise

- **NoSQL**
- Abandon des contraintes du relationnel
- Esprit développeur, langages modernes
- Ère du libre et de l'open source
- Prise en compte de l'usage "jetable" des machines
- Facile, sympa

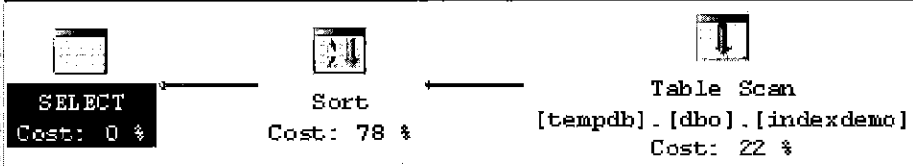
Edgar Frank Codd

- Chercheur britannique, créateur du modèle relationnel dans les années 70
- Théorie des ensembles
- Algèbre relationnelle
- Les 12 règles de Codd
  - Langage relationnel, indépendance logique et physique
- OLTP / OLAP

Le langage SQL

- Traitement déclaratif et ensembliste
- Optimisation d'une requête déclarative
- Manipulation ensembliste transformée derrière le rideau en traitement impératif

```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM dbo.indexdemo ORDER BY id
```



Les contraintes du langage SQL

- Demande un bon niveau de compétences pour faire du bon travail
- Il faut réfléchir optimisation – ce qui n'est pas forcément enseigné
- Favorise le copier-coller
- Déclaratif
  - *SQL, Lisp, and Haskell are the only programming languages that I've seen where one spends more time thinking than typing*. Philip Greenspun, blog, 07-03-2005

Les contraintes du relationnel

- ACID
  - Atomicité
  - Cohérence
  - Isolation
  - Durabilité
- Contraintes fortes dès la conception
  - Indispensables pour gérer la montée en charge
  - Demande un bon niveau de technicité
  - Modèle de développement tendant au waterfall

Le modèle

Relationnel

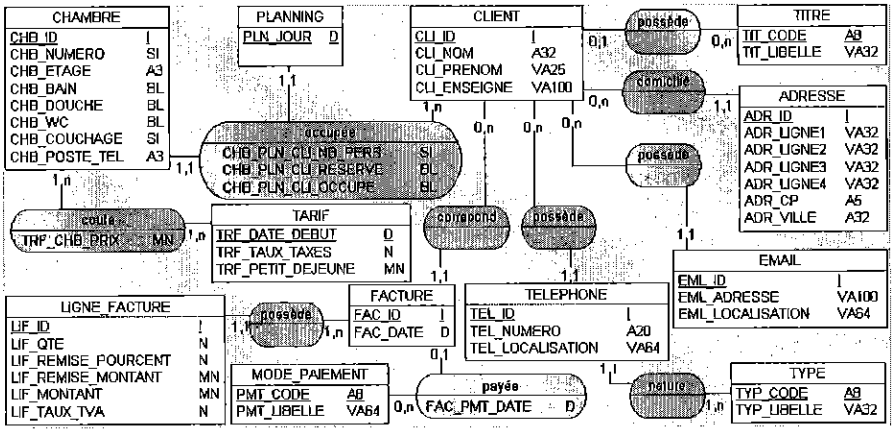
- Schéma contraignant
- Couplage moyen-faible avec le code client
- Développement incrémentiel
- Langage de bas niveau, rapidité
- Optimisation par la conception et le code

NoSQL

- Schema-less ou schema-later
- Couplage variable. Très fort ou très faible
- Développement agile
- Langages de haut niveau
- Optimisation par le matériel – les muscles contre le cerveau
  - Mais ça ne veut pas dire que c'est facile

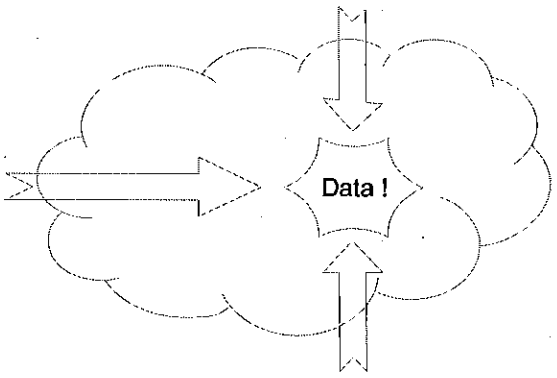
La structuration forte des données

- Le schéma préexiste à tout



Que vaut ma donnée ?

- Est-elle unique ?
- Est-elle partout ?
- Est-elle protégée ?
  - De quel côté ?
- Est-elle prioritaire ?
  - C'est la donnée ou le développeur ?
- Qu'est devenue la normalisation



NULL et schéma sparse

- La nécessité d'établir un schéma dans le modèle relationnel impose des colonnes pré-établies
  - C'est une force et une contrainte
  - Les cellules inconnues existent, elles sont marquées NULL
- Les bases NoSQL n'ont pas de schéma pré-établi
  - Orienté colonnes : familles de colonnes pré-établi
  - Orienté documents : schema-less
    - Mais ... indexation secondaire ?
  - Dans un document JSON, il n'y a pas de NULL.

Pourquoi pas de jointures ??

extrait de l'introduction de *Hbase, the definitive Guide* (Lars George, O'Reilly 2011)

"With more site popularity, you are asked to add more features to your application, which translates into more queries to your database. The **SQL JOIN**s you were happy to run in the past are suddenly slowing down and are simply not performing well enough at scale. You will have to **denormalize** your schemas. If things get even worse, you will also have to cease your use of **stored procedures**, as they are also simply be- coming too slow to complete. Essentially, you reduce the database to just storing your data in a way that is optimized for your access patterns.

Your load continues to increase as more and more users join your site, so another logical step is to **prematerialize** the most costly queries from time to time so that you can serve the data to your customers faster. Finally, you start **dropping secondary indexes** as their maintenance becomes too much of a burden and slows down the database too much. You end up with queries that can only use the primary key and nothing else."

Défaut d'impédance Objet / relationnel

```
dim sql

sql = "Insert into Users (UserName, FirstName, LastName, EMailAddress) "
sql = sql & "Values('" & request.form("UserName") & "', '" & request.form("FirstName")
sql = sql & "', '" & request.form("LastName") & "', '" & request.form("EmailAddress") &
"')"

objConn.execute(sql)
```

- LINQ ?

```
NorthwindDataContext db = new NorthwindDataContext();

Product product = db.Products.Single(p => p.ProductName == "Toy 1");

product.UnitPrice = 99;
product.UnitsInStock = 5;

db.SubmitChanges();
```

Défaut d'impédance Objet / relationnel ORM ?

- Hibernate / Nhibernate
- Entity Framework
- *Et le plan d'exécution ??*

```
SessionFactory sessionFactory = new

Configuration().configure().buildSessionFactory();
session =sessionFactory.openSession();

Contact contact = new Contact();
contact.setId(3);
contact.setFirstName("Rudi");
contact.setLastName("Bruchez");
contact.setEmail("rudi@babaluga.com");
session.save(contact);
```

Les spécificités du NoSQL en regard du relationnel

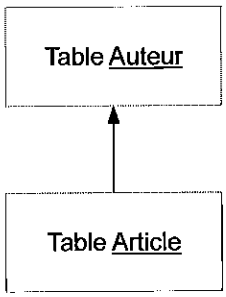
- Oublier les bonnes pratiques ... en quelque sorte
- Relaxation de l'ACIDité
- Bases orientées document = “self-contained data”
- Schéma souple : sans schéma ou schéma déferré
- Souplesse à tous les niveaux
- Data storage = entrepôts vs systèmes de gestion
- Traitement procédural ou isolé vs traitement ensembliste
  - Langage impératif vs langage ensembliste
  - Influence des langages fonctionnels

Oublier les bonnes pratiques

- Les SGBDR sont très conservateurs
  - Qualité des données – mais on n'y arrive jamais
  - Isolation forte des transactions – mais les problèmes sont rares
    - Deadlocks
    - Qui utilise le niveau d'isolation repeatable read ??
  - Le serveur s'occupe de tout
    - En NoSQL, le client fait pas mal de boulot : structuration des données, gestion des conflits...
  - Quand on est en isolation Read Uncommitted, les problèmes sont rares

Normalisation vs "self-contained data"

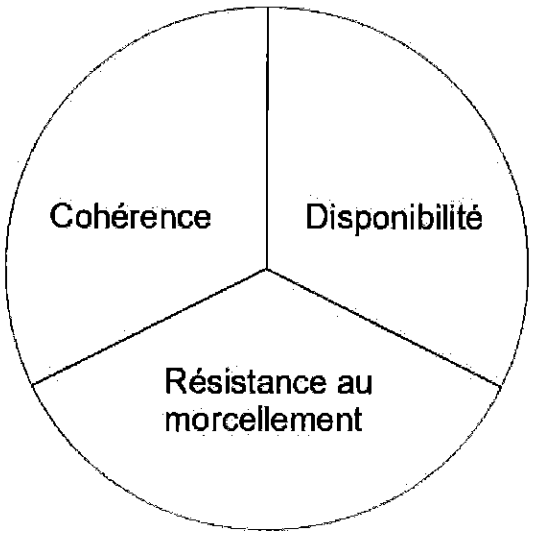
- "Document embedding"



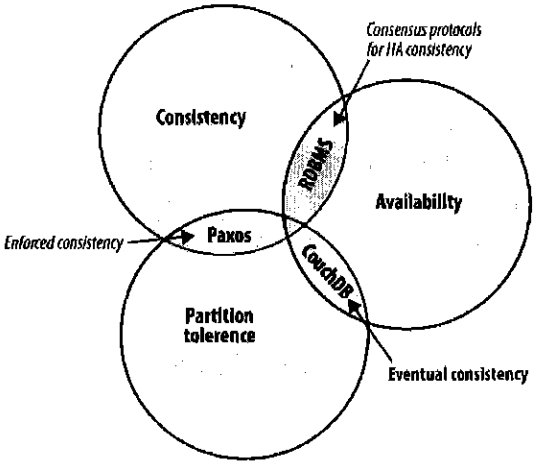

```
{
      "auteur":
      {
            "prénom":"Annie",
            "nom":"Brizard",
            "e-mail":"annie.brizard@cocomail.com"
      },
      "titre":"pourquoi les éléphants ont-ils de grandes
oreilles ?"
}
```

Le théorème CAP

- Principe énoncé par Eric Brewer
- **Consistency** (cohérence) – tous les clients voient les mêmes données;
- **Availability** (disponibilité) – les clients peuvent toujours accéder à toutes les données
- **Partition tolerance** (résistance au morcellement) – une base de données peut être partitionnée sur plusieurs serveurs.



Carte du CAP

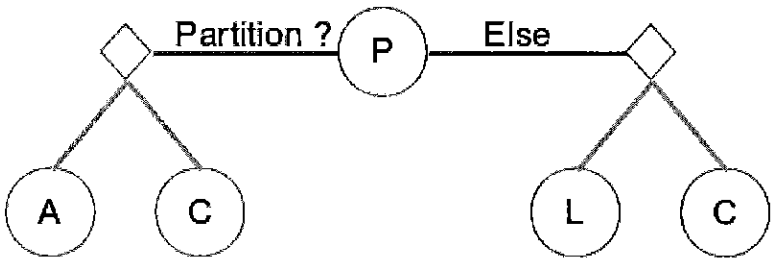


- Nous avons vu que pour monter en charge nous devons partitionner les données
- Cela nous laisse deux choix :
  - Disponibilité
  - Cohérence
- En général on abandonne la cohérence
  - La cohérence devient “finale”

Tolérance au morcellement

- La vraie question : comment synchroniser les données ?
- Si la cohérence doit être élevée, la synchronisation diminue la disponibilité
  - *Chaque nœud d'un système devrait être capable de prendre des décisions basées purement sur un état local. Si vous devez faire quelque chose sous une charge élevée avec des noeuds indisponibles, et que vous devez atteindre un consensus, vous êtes perdus. Si vous voulez monter un charge, un algorithme basé sur le consensus sera votre goulet d'étranglement.*
    - Werner Vogels, Amazon CTO et Vice President

Une variante : PACELC



- S'il y a partitionnement, la lutte s'organise entre la Disponibilité et la cohérence
- Par contre, la balance se fait entre latence et cohérence
- Bref, la **cohérence** est toujours en question.

PACELC est donc un concept de Daniel Abadi. Sa présentation est disponible à cette adresse : http://fr.slideshare.net/abadid/cap-pacelc-and-determinism. Le propos d'Abadi est que le théorème CAP est trop simple, voire simpliste, et qu'il est principalement utilisé comme excuse pour abandonner rapidement la cohérence. IL est facile d'affirmer « Le théorème CAP dit qu'on ne peut pas partitionner, conserver de bonnes performances et maintenir la cohérence des données, donc je laisse tomber la cohérence des données ». Mais, même si on reste dans l'optique du CAP, on peut avoir la disponibilité (A, availability) et la cohérence si on n'a pas de partition. Un système peut très bien être partitionné ou non, et donc gérer différemment la cohérence dans ces cas.

Le raisonnement est le suivant : Si on est en présence d'une partition (P), le système va-t-il choisir entre la disponibilité (A) ou la cohérence (C) ? Sinon (E pour Else), le système va-t-il choisir entre la latence (L) ou la cohérence (C) ?

Un système comme Dynamo est de type PA/EL : en présence d'une partition il privilégie la disponibilité au détriment de la cohérence, sinon il choisit la latence. En d'autres termes, il n'y a jamais de souci de maintenir la cohérence. De l'autre côté du spectre, un moteur relationnel respectant l'ACIDité de la transaction sera PC/EC : la cohérence sera toujours privilégiée. On se dit donc qu'il manque le meilleur des choix : PA/EC, où la cohérence est privilégiée quand son coût est acceptable. Cette option n'est pas encore réellement mise en place dans les moteurs NoSQL. Il l'est dans des moteurs « hybrides » comme GenieDB (http://www.geniedb.com/), un moteur de stockage pour MySQL qui opffre une couche de montée en charge horizontale.

Fallacies of Distributed Computing

- Un système distribué devrait être construit sur le présupposé que tout va aller mal
- Fallacies of Distributed Computing
  - Le réseau est fiable.
  - La latence est de zéro.
  - La bande passante est infinie.
  - Le réseau est sécurisé.
  - La topologie ne change pas.
  - Il y a un seul administrateur.
  - Le coût du transport est de zéro.
  - Le réseau est homogène.

Pourquoi on en arrive là ?

- Big data !
- Si big data, alors ...
  - On ne peut pas tout mettre sur la même machine : la montée en charge verticale ne suffira jamais.
  - Il se peut même que les données ne tiennent pas sur une seule baie.
  - Si on atteint ces volumes, les SAN deviennent une solution trop chère.
- Il faut donc distribuer les données

Big data

- Les moteurs relationnels ne sont pas conçus pour être distribués
- Problèmes = ACIDité et consistance
- Il y a une limite à ce qu'une seule machine peut supporter
- Comment faire du scale-out ?
  - Partitionnement – sharding
  - Maître-esclave

Premières solutions au big data

- **Maître-esclave**
- Écritures sur le maître, qui réplique vers les esclaves
- Lectures sur les esclaves
- Désynchronisation possible
- Difficultés à supporter les gros volumes

- **Sharding**
- Bonne solution pour écritures et lectures
- Pas de redondance
- Implication du client
- Pas de jointure possible
- Perte des contraintes d'intégrité

Big Data = NoSQL ?

- Les SGBDR sont-ils insolubles dans le Big Data ?
- Michael Stonebreaker (http://cacm.acm.org/blogs/blog-cacm/50678-the-nosql-discussion-has-nothing-to-do-with-sql/fulltext)
  - Les SGBDR doivent offrir du sharding automatique
  - Les problèmes viennent des allers-retours avec le serveur
  - Les ralentissements des SGBDR viennent de :
    - Journalisation – écriture des transactions sur disque pour durabilité
    - Verrouillage – pour isolation
    - Latching – verrous de bas niveau pour protéger les structures dans un environnement multi-threads
    - Gestion du Buffer – la localisation des données dans des pages de données

Quelques scénarios d'utilisation Big Data

- Facebook :
  - Cassandra pour l'analytique – pas conservé pour le temps réel
  - Hbase = +135 Mia de messages par mois (2010)
- Linkedin
  - Centaines de millions d'accès par jour < 10ms avec Voldemort (2009)
- *Selon IDC le marché du Big Data devrait croître à une allure de 40% par an entre 2010 et 2015.*

Distribution des données

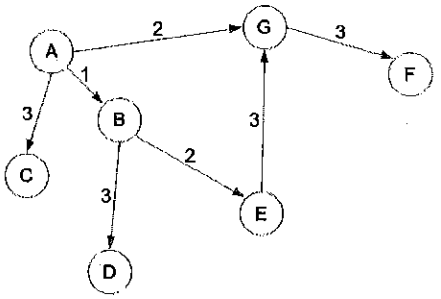
- NoSQL gère des données disponibles par une clé
  - Les systèmes NoSQL sont tous des systèmes de paires clé-valeur
- Normalement on y accède surtout par la clé
- Il faut donc savoir sur quel(s) noeud(s) se trouve une clé
  - Sharding
  - Ou distribution sans maître, avec des protocoles de recherche des noeuds qui contiennent la clé ... et la capacité de trouver un autre noeud si celui-ci est tombé
    - Par exemple, consistant hashing avec DHT.

Distribution des données avec maître omniscient

- La distribution des données est gérée par une machine maître, qui connaît l'emplacement des partitions
- SPOF
- Il vaut donc mieux avoir un cluster de maîtres
  - Exemple du sharding en MongoDB
- Avatange : simplicité d'implémentation
- Par contre, point d'entrée rigide, non-partitionnable

distribution sans maître – gossip protocol

- Le protocole de bavardage (gossip protocol)
  - Architecture en peer-to-peer
  - Théorie des épidémies
  - Chaque T secondes, un noeuf en choisit un autre pour réconcilier leurs vues = progression exponentielle
  - Ne permet qu'une cohérence finale
  - Permet la découverte de noeuds, la détection d'erreurs, ...
- Exemple – Cassandra
  - Niveau de cohérence paramétrable



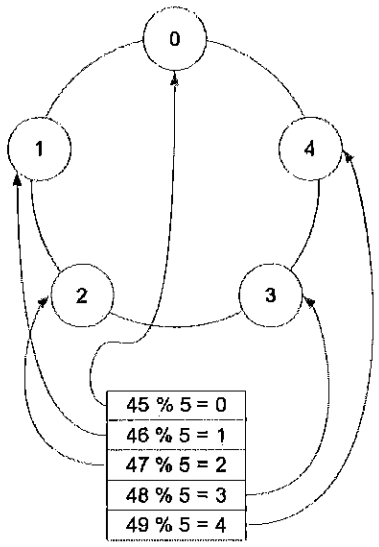
Réplication incrémentale

- En situation de shared nothing ...
  - On maintient des versions
  - On réplique ces versions
  - ... quand on veut, même en asynchrone
  - Une forme de téléphone Arabe
  - *Les entretiens radio des ministres vs le porte-parole du gouvernement.*
- Chaque noeud est toujours capable de travailler indépendamment

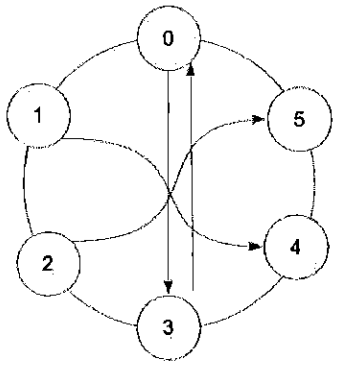
Réplication incrémentale

- Exemple de CouchDB = Réplication de fusion
- Détection et résolution automatique de conflits
- Comme CouchDB maintient des versions de ses données, les conflits sont simplement gérés en conservant des versions du document
- Vous pouvez récupérer ces versions par programmation et faire ce que vous voulez

Distribution des données
Sharding par modulo



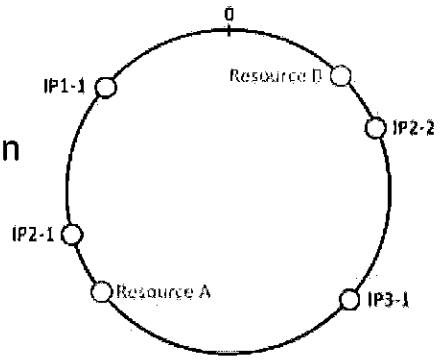
45 % 5 = 0
46 % 5 = 1
47 % 5 = 2
48 % 5 = 3
49 % 5 = 4



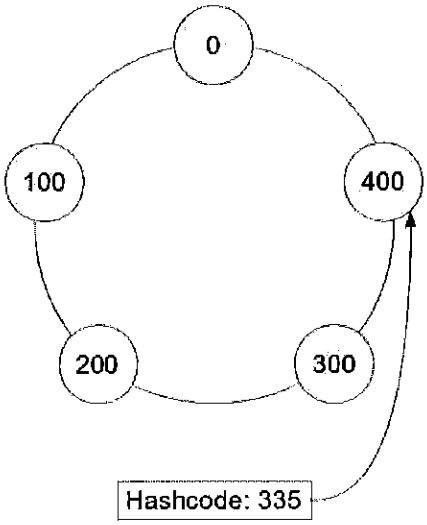
0 → 45 % 6 = 3
1 → 46 % 6 = 4
2 → 47 % 6 = 5
3 → 48 % 6 = 0
4 → 49 % 6 = 1

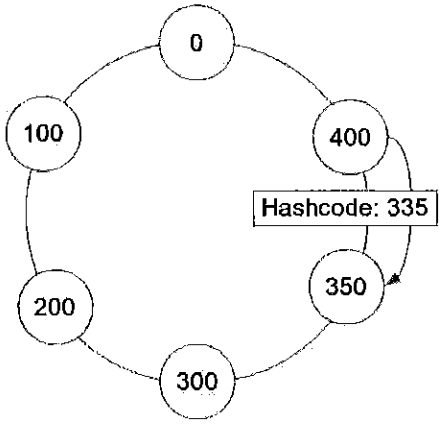
Distribution des données
Hachage consistant

- Le consistent hashing – memcached, Dynamo, Riak, Cassandra
- On représente les noeuds dans un anneau (ring)
- On attribue une valeur de hash à chaque noeud
  - (HashCode = entier)
- Chaque noeud stocke les clés supérieures à sa valeur, et inférieures à la suivante dans le ring.



Distribution des données
Hachage consistant





La transaction – Atomicité

- En général, l'atomicité est garantie sur une ligne
- Documents : atomicité niveau document (CouchDB, MongoDB)
  - il n'y a pas forcément besoin d'être plus atomique que ça.

```
db.runCommand( { findAndModify :
<collection>, <options> } )
```

- Base orientées colonnes : atomicité de la ligne
  - Verrouillage au niveau de la ligne (Hbase, Hypertable)
  - Niveau famille de colonnes en Cassandra (donc la ligne)
- En Redis : "fausses" transactions à l'aide de MULTI

La cohérence (consistency)

- Un mini-chapitre à lui tout seul :
  - Cohérence distribuée
  - Respect des règles métiers
  - Cohérence finale
  - Gestion des versions
  - Les mécanismes de résolution de conflit
  - La néguentropie
  - ...

Cohérence distribuée ?

- Multi-maîtres – MySQL avec Tungsten
  - Il faut un protocole de résolution de conflits
- Maître-esclave -- MongoDB
- Partitionnement
- Sharding – MongoDB
- Verrouillage ?
  - Cohérence forte = verrouillage pessimiste
  - Cohérence finale = verrouillage optimiste
    - MVCC – Multi-Version Concurrency Control
    - Protocole de bavardage – P2P

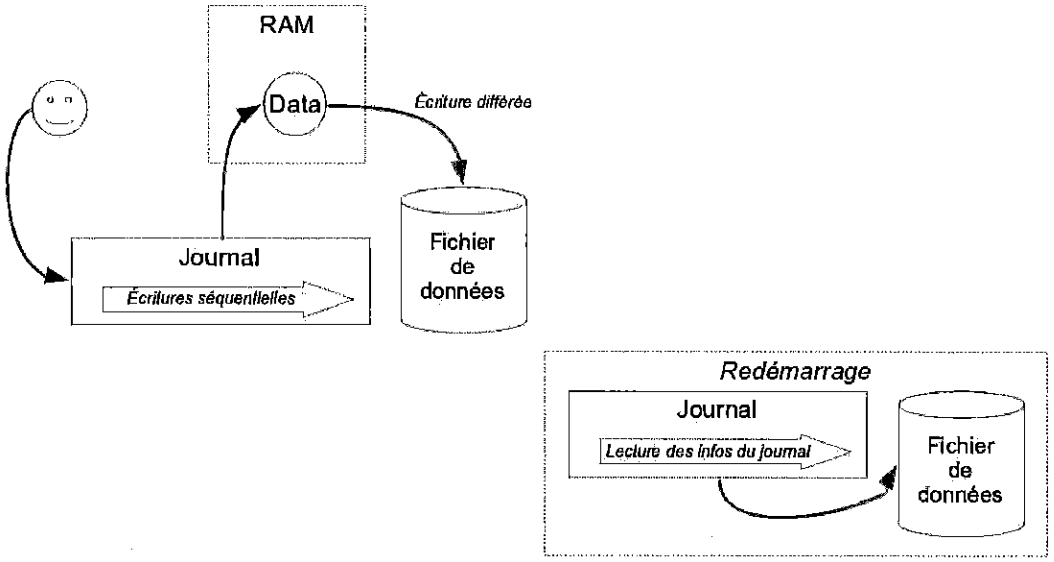
Cohérence – Contraintes ?

- Contraintes : intégrité, check, types, clé
- Validation en NoSQL ?
  - Parfois oui : CouchDB – validation par fonctions Javascript
    - Des déclencheurs, autrement dit
    - *By working with the grain and letting CouchDB do this for us, we save ourselves a tremendous amount of CPU cycles that would otherwise have been spent serializing object graphs from SQL, converting them into domain objects, and using those objects to do application-level validation.*
    - Vous connaissez l'addition des pommes et des bananes ?

La transaction – Durabilité

- Dans les SGBDR, la durabilité est réalisé à l'aide d'un journal de transaction (write-ahead transaction log)
  - La durabilité dépend donc toujours des ressources matérielles – le disque pour un système avec journalisation
  - La durabilité n'est pas toujours garantie en NoSQL
  - MongoDB incorpore un journal depuis sa version 1.8, qui peut être désactivé
  - Redis – persistence paramétrable, mais jamais 100 % durable (il n'y a pas de concept de transaction)
    - On dépend ici la la stabilité de la machine et de la RAM.

La transaction – durabilité
Write-ahead log

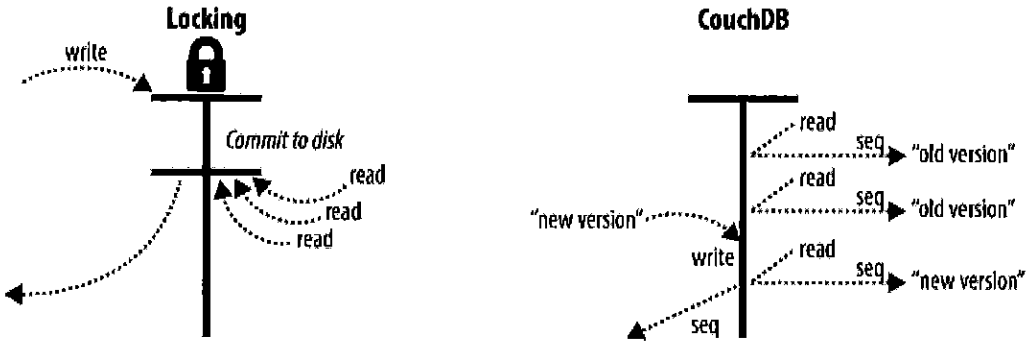


Verrouillage optimiste par maintien de versions

- clones de Bigtable
  - Chaque colonne est versionnée par un timestamp
  - La modification peut mentionner la dernière version
- CouchDB
  - Les documents sont versionnés
  - Erreur si la modification ne mentionne pas la dernière version

Multi-Version Concurrency Control

- Exemple de CouchDB



- Au lieu de verrouiller, on écrit chaque fois une nouvelle version complète du document
- Chaque lecture est faite sur la dernière version

La transaction – Cohérence

- **ACID**
- cohérence forte et immédiate – respect d'un état global
- La cohérence est considérée d'un point de vue transactionnel = respect des contraintes et de l'atomicité
- **BASE**
- *Basically Available Soft-state Eventual consistency*
- Eventual = finale
  - Propagation DNS
  - Protocoles de quorum
- La cohérence est un processus – la cohérence locale basée sur un log n'est pas forcément plus intéressante qu'une cohérence distribuée par réplication de la donnée.

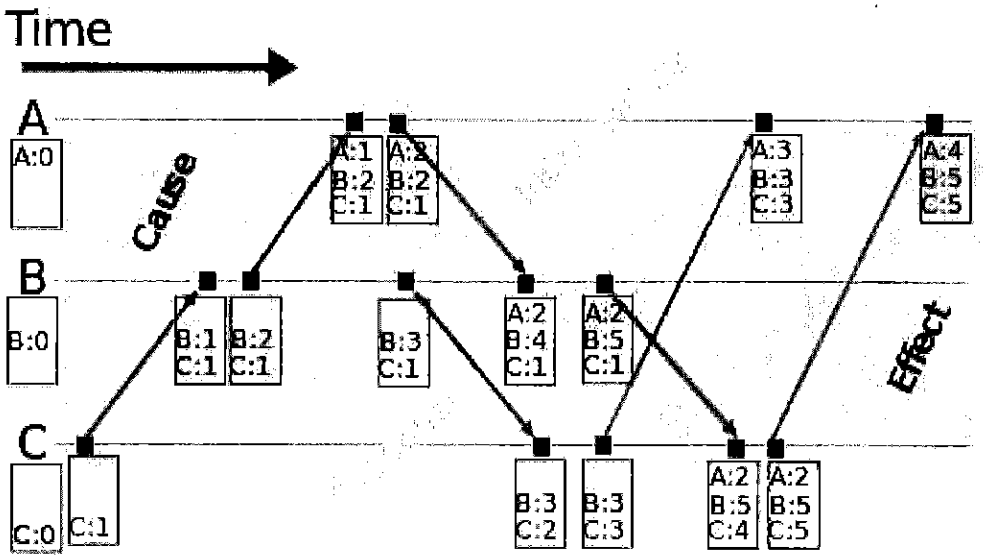
BASE – modèles

- **B**asically **A**vailable, **S**oft state, **E**ventual consistency
- Eventual consistency = consistance finale
  - Cohérence causale
    - Si on écrit B après avoir vu A, tout client qui lit B peut voir A
  - RYOW – Read Your Own Writes
  - Cohérence non-causale
    - Une mise à jour finira par être disponible sur tous les noeuds
  - Propagation des mises à jour au fil de l'eau
    - Exemple imparfait de la propagation DNS

Cohérence finale – comment ?

- **Timestamps**
  - chaque modification est marquée de sa date. On sait quelle est la dernière modification, mais on ne peut pas déterminer la causalité
- **Vector clocks**
  - Tableau d'horloges qui représentent des noeuds. Chaque noeud incrémente son horloge à chaque modification
- **Log Structured Merge Tree**
  - Les écritures sur disque sont regroupées, les modifications sont conservées en mémoire avant écriture. Évite les écritures aléatoires

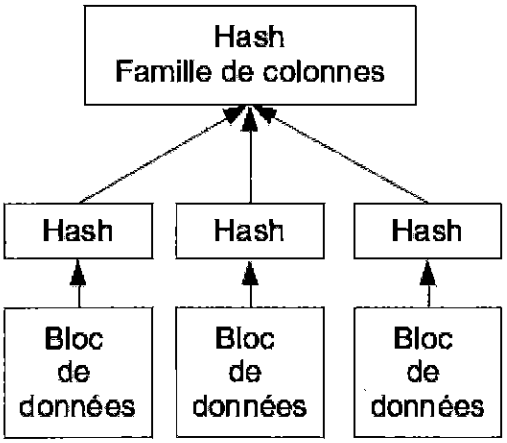
Vector Clocks



Cohérence finale
Mécanismes de néguentropie

- Anti-entropie : capacité du système à maintenir un ordre malgré la tendance naturelle au désordre.
- Réconciliation régulière des différences à l'aide d'arbres de Merkle (un arbre de hachages) (*voir prochain slide*)
- Inspiration Dynamo : Riak et Cassandra
- L'impact du calcul de l'arbre doit être minimisé
  - En Cassandra, l'arbre de Merkel est calculé par un validateur durant des opérations de compactage

Arbre de Merkle (= arbre de hachage)



- Hash de hashs
- Permet de vérifier rapidement des structures volumineuses en calculant des hachages par blocs.
- Les niveaux supérieurs sont échangés à travers le réseau pour synchronisation, et seuls les sous-arbres modifiés sont ensuite échangés

Souplesse à tous les niveaux

- Dans certains moteurs (CouchDB, MongoDB) : plusieurs fonctionnalités
  - Capped collections en MongoDB
- Mais ce n'est pas forcément réservé aux bases NoSQL
  - MS SQL Server par exemple :
    - Relationnel
    - XML
    - Spatial
    - ColumnStore
    - Décisionnel OLAP et tabulaire ...

La recherche de données

- Les bases NoSQL ne sont pas vraiment faites pour ça
- Accès à travers la clé
- Index secondaires ... bof
- Solutions diverses :
  - Index secondaire quand même
  - Recherche en plein texte
    - Elastic search

Les choix techniques du NoSQL

- **Les choix techniques du NoSQL**
  - L'architecture distribuée, le Cloud Computing
  - La structure souple des données : JSON
  - L'ouverture
  - La base orientée document. La base orientée colonne. Les paires clé-valeur
  - Le support variable de la cohérence transactionnelle
  - Le stockage de données temporaires : memcache
  - La distribution : MapReduce

Avant de parler technique ...

- Il y a des choix conceptuels ...
  - Relax (CouchDB ↔ Ruby On Rails)
  - La complexité est gérée automatiquement – par exemple la gestion des incidents
  - L'optique du développeur plus que celle de l'administrateur ou de l'architecte.
  - "web scale"
  - Utilisation des standards du web
    - JSON
    - REST
    - ...

Les outils de développement, et les moteurs de bases de données relationnelles ont été traditionnellement l'affaire de spécialistes, et difficiles à mettre en oeuvre. Le mouvement NoSQL essaie de prendre les choses de façon différente.

Implication du client dans les bases NoSQL

- Jointures déplacées vers le client
  - Aller-retours pour résoudre les recherches de clés
- Verrouillage optimiste
  - Avertissement en cas de conflit. Au client de traiter
- Fonctionnement largement procédural
  - Pas de traitement ensembliste

Traitement du côté du serveur ?

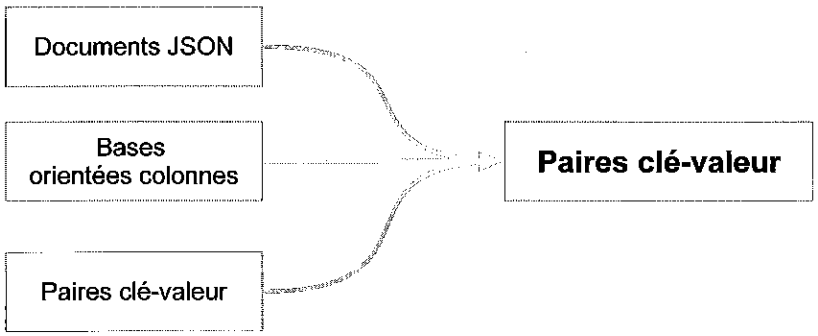
- Peu de traitements au niveau du serveur, avec quelques contre-exemples:
  - db.eval() en MongoDB – évaluation de Javascript
  - Fonctions dans des documents de design en CouchDB
  - Déclencheurs asynchrones dans Cassandra (cassandra-triggers)
  - Coprocesseurs dans Hbase.
  - MapReduce, surtout

Ouverture

- Protocoles déjà connus
  - JSON
  - REST
  - Contre-exemple : protobuf – mais c'est normal
- Par contraste, les protocoles de niveau réseau des SGBDR commerciaux sont propriétaires
  - Limitation d'accès – par exemple driver ODBC pour Linux pour MS SQL Server.

Structures de données

- Les données d'un moteur NoSQL peuvent toujours se résumer à une paire clé-valeur



Structure de données – paire clé-valeurs

- Structure classique en programmation :
  - = tableau associatif
  - = table de hachage
  - = vecteur
- + Simple, très rapide, facile à partitionner
- - Ne couvre que des besoins simples

```
$dico = array( "lundi"=>"dodo",
               "mardi"=>"dodo",
               "mercredi"=>"resto");
echo $dico["lundi"];
foreach($dico as $key=>$value)
{
    echo "Le $key c'est $value.";
}
```

Structure de données – JSON

- Javascript Object Notation
  - Sérialisation aisée des classes

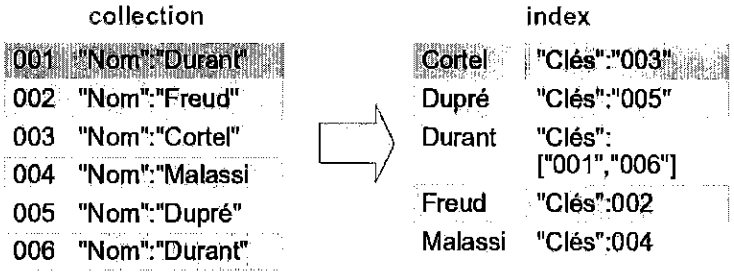
```
{
      "auteur":
      {
            "prénom":"Annie",
            "nom":"Brizard",
            "e-mail":"annie.brizard@cocomail.com"
      },
      "titre":"pourquoi les éléphants ont-ils de grandes
oreilles ?",
      "mots-clés":
      [
            "faune",
            "afrique",
            "questions intriguantes"
      ]
}
```

Structures de données – documents

- La **CLÉ**, rien que la CLÉ, toujours la CLÉ
- Obsession sur la clé. Accès par la clé très rapide. Donc il ne faut y accéder que par la clé
  - Les reste fait l'objet “d'index secondaires” -- vade retro !
    - Exemple de Vodex et Riak
  - *As well as the massive speed improvements, we can partition our data over multiple nodes, without affecting our ability to query each node in isolation. BigTable, Hadoop, SimpleDB, and memcached restrict object lookups by key for exactly these reasons.*

Comment faire des recherches ?

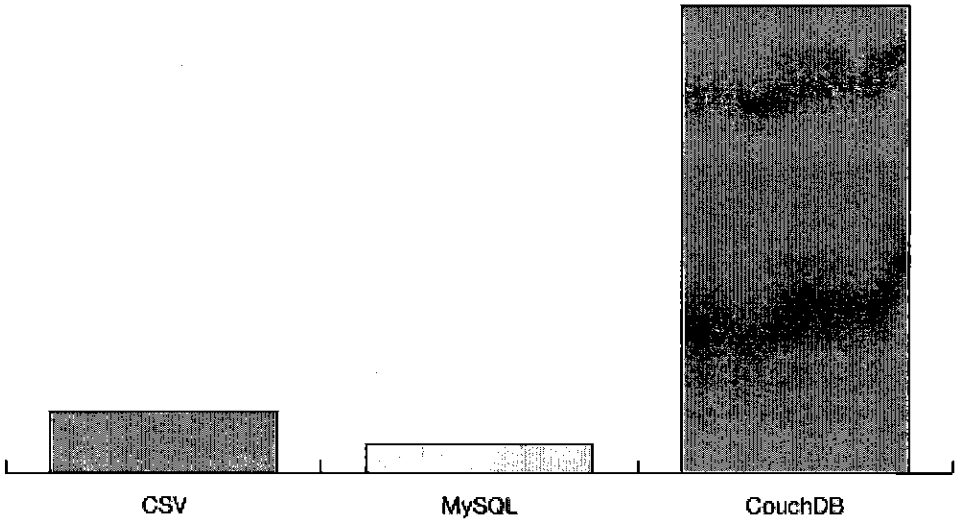
- Index secondaires dans les moteurs qui le supportent
  - MongoDB, Cassandra, Hbase
  - Riak plus récemment (i2)
  - Vues en CouchDB
- Indexation manuelle en paires clé-valeur



Structures de données – documents

- L'idée du document, c'est qui peut et doit être stocké et restitué en une seule fois.
  - Parallèle avec un document bureautique
  - Utilisation simple
  - Dénormalisation
- Cela ne veut pas dire binaire
  - JSON – document structuré
  - En général, possibilité de joindre des fichiers binaires

Structures de données – le stockage ?



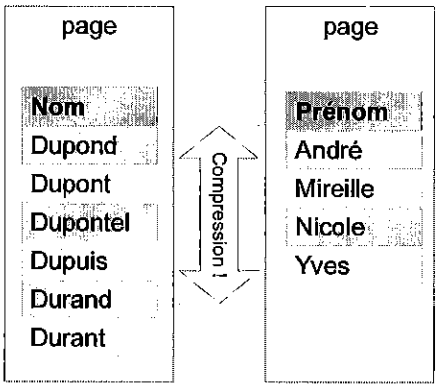
Bases orientées colonne

- Modèle traditionnel
  - *Row store*

page

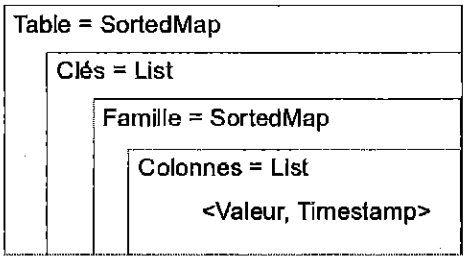
Nom	Prénom	Naissance
Dupond	Yves	04-07-1964
Dupont	Mireille	23-01-2001
Dupontel	*NULL*	*NULL*
Dupuis	Nicole	12-07-1974
Durand	André	*NULL*
Durant	*NULL*	*NULL*

- ColumnStore



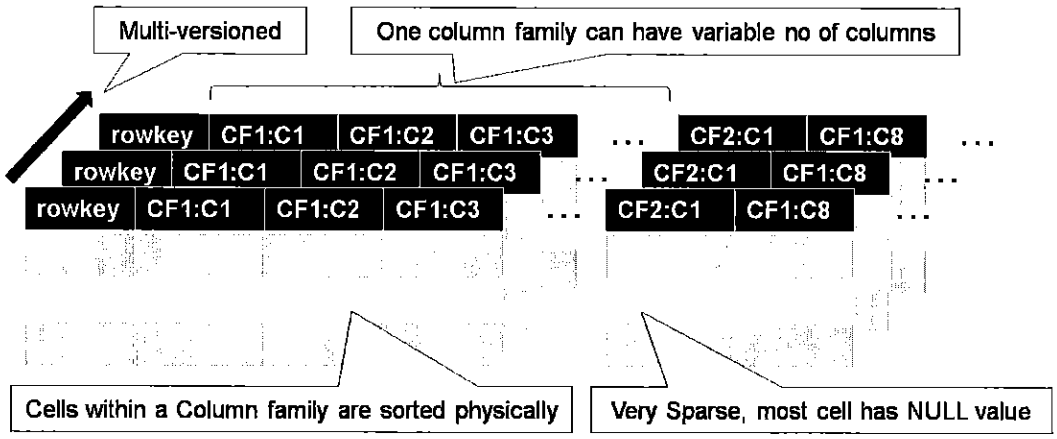
Bases orientées colonne

- Une ligne est identifiée par une clé (RowKey)
- La ligne est séparée en familles de colonnes (*sauf Cassandra*)
  - Imaginons des sous-tables, ou des façons d'agréger des tables filles dans un modèle non-relationnel.



- Le stockage est organisé par famille de colonnes
  - Les familles de colonnes sont prédéfinies
  - Les colonnes sont organisées dans les familles
  - Stockage sparse

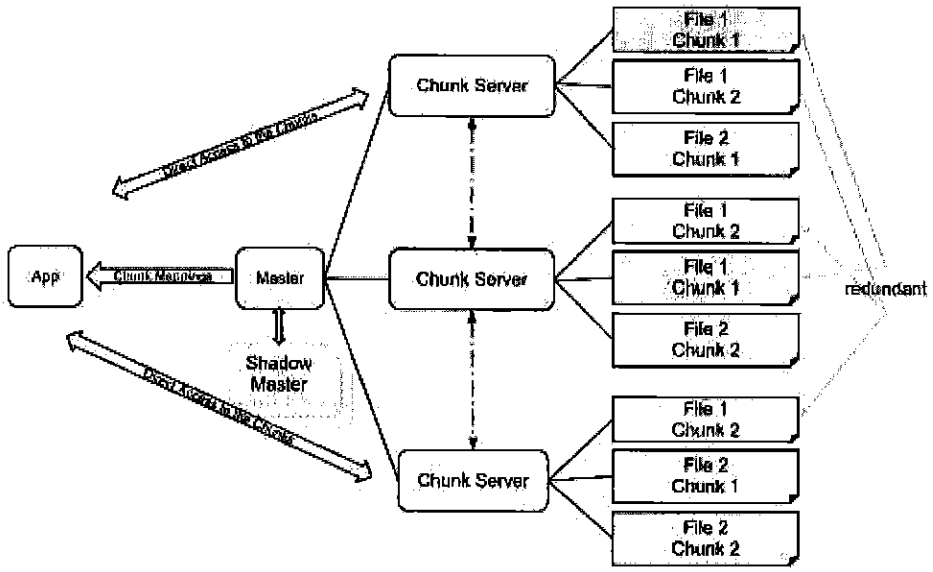
Structures de données – BigTable



Structures de données – BigTable

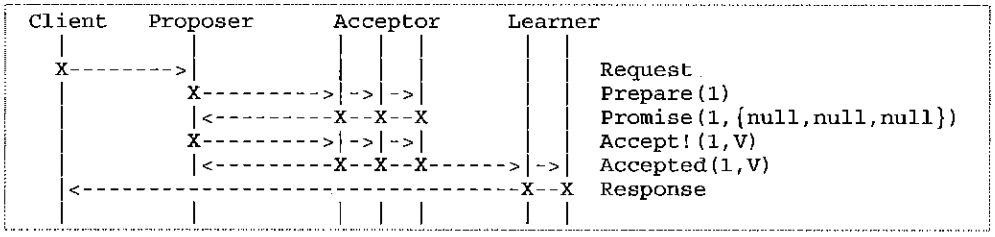
- Entrepôt orienté colonne, implémenté en Hbase, Hypertable, Cassandra
- Tableau trié, distribué, sparse et compressé, persistant et multi-dimensionnel
- Le tableau est indexé par une clé de colonne, une clé de ligne, un timestamp – les données sont donc immutables
- Les lignes sont maintenues dans un ordre lexicographique
  - Des plages de lignes sont partitionnées dynamiquement dans des tablettes (tablets)
- Les colonnes sont regroupées en familles

Google BigTable – GFS

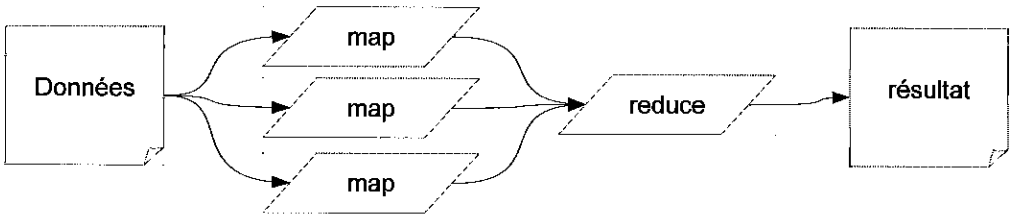


Google BigTable – Chubby

- Système de verrouillage distribué et de réplication
- Paxos
  - Alogithme de consensus pour la distribution

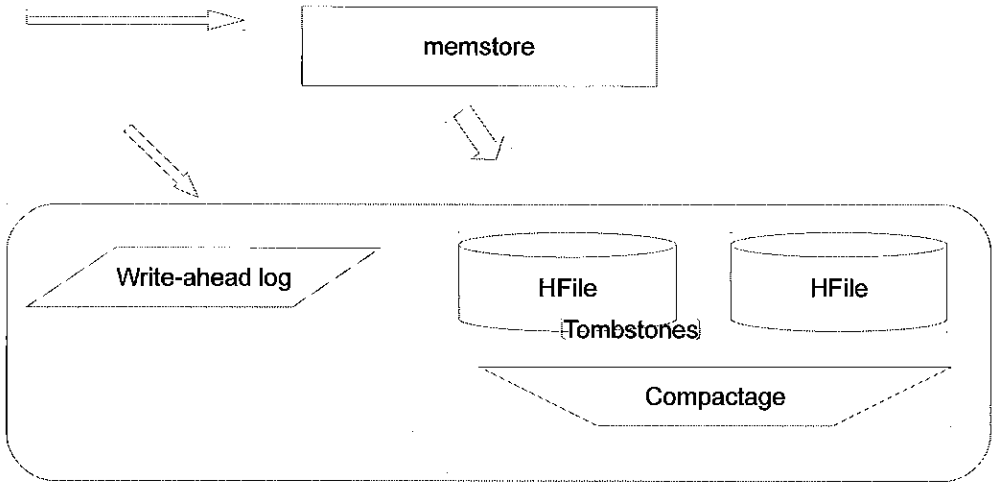


Google BigTable – MapReduce



- Map et Reduce = langages fonctionnels comme Lisp
  - Map = traitement de chaque élément d'une liste
  - Reduce = réduction des éléments par la clé, agrégat
  - Application de fonctions, pas de changement d'état

Bigtable – écritures



Filtres de bloom

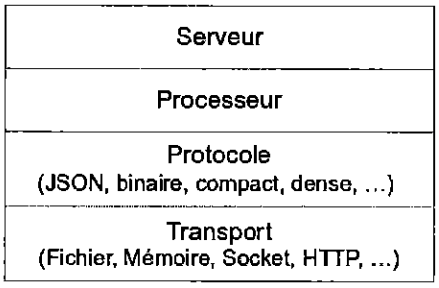
- Le filtre de bloom est utilisé par Hbase, Hypertable, Cassandra pour optimiser les accès
  - Structure de donnée extérieure à un stockage physique (SSTable, Hfile, ...)
  - Algorithme probabiliste
  - Possibilités de faux positifs, mais pas de faux négatifs
  - Remplissage d'un tableau de bits
  - Cassandra : un filtre par SSTable

Les interfaces – REST

- Representational State Transfer
- Le web est REST – états et fonctionnalités basées sur des resosurces accessibles à l'aide d'une URI
- Interface à CouchDB par exemple
  - Sans état = pas de connexion

Les interfaces – Thrift

- Framework de la fondation Apache pour le développement de services interopérables – RPC
- Création et compilation d'interfaces entre services
- Plusieurs moteurs NoSQL publient des interfaces Thrift
  - Cassandra surtout
  - Mais aussi HBase



Les interfaces – protobuf

- Format créé par Google et libéré
- Structure hiérarchique
- Éléments *required* ou *optional*.
- Énumérations
- Le compilateur crée des classes pour un langage cible

```
package passerelles;

message Article {
  required int32 id = 1;
  required string titre = 2;

  message Auteur {
    required string prenom = 1;
    required string nom = 2;
    required string email = 3;
  }

  repeated Auteur auteur = 3;
}

message Articles {
  repeated Article article = 1;
}
```

Les interfaces – avro

- Fondation Apache, créé par Doug Cutting
- Schéma défini en JSON
- Ne nécessite pas de compilation, contrairement à Thrift et Protobuf.

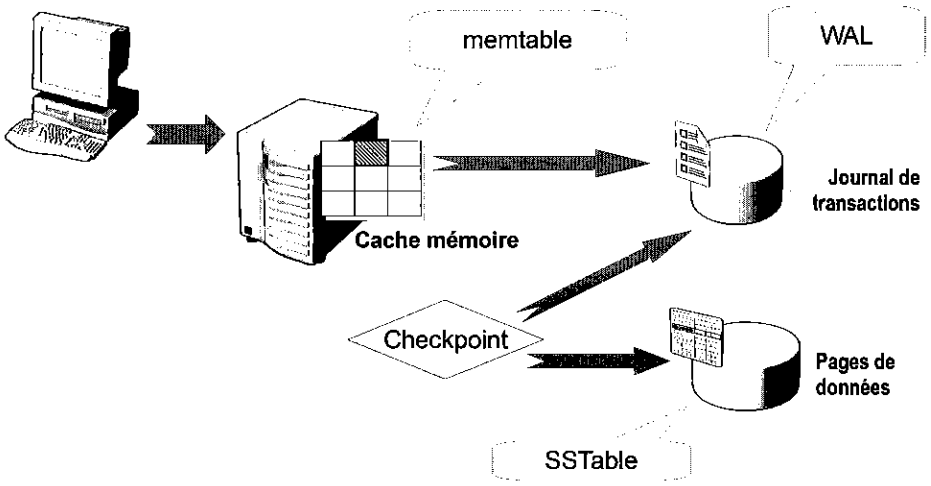
Le cache – introduction

- Un des discours du mouvement NoSQL : les bases de données en mémoire
- Si on résume :
  - Big data = MapReduce = Hadoop
  - Rejet du relationnel = Documents = MongoDB, CouchDB
  - Besoin de grande rapidité = bases “en mémoire” = Redis
- Big data ≠ latence faible
- Bases de données Documents ≠ latence faible
  - C'est juste la distribution qui fait illusion dans les deux cas

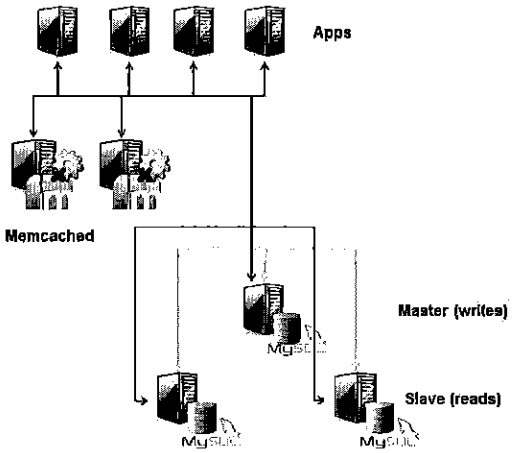
Le cache – hésitations

- Les développements NoSQL proviennent souvent de développeurs open source
  - ► utilisation de MySQL ► souffrent de lenteurs
    - ► recherche de solutions
      - Répartition – on a vu que c'est délicat dans un contexte ACID
      - Cache du côté du client – pas très satisfaisant conceptuellement
  - Les moteurs relationnels travaillent en général en mémoire
    - Buffer = cache de données
    - MySQL avec MyISAM n'a pas de buffer, InnoDB oui, mais il y a des choses à tuner : taille des pages, taille du buffer
    - On en revient aux besoins d'optimisation pour les moteurs relationnels

Le cache – SGBDR vs NoSQL



Le cache des données = memcached



- Memcached est utilisé traditionnellement dans une ferme applicative
- Maintient une table de hachache distribuée de paires clé-valeur

http://www.clusterdb.com/mysql-cluster/scalabale-persistent-ha-nosql-memcache-storage-using-mysql-cluster/

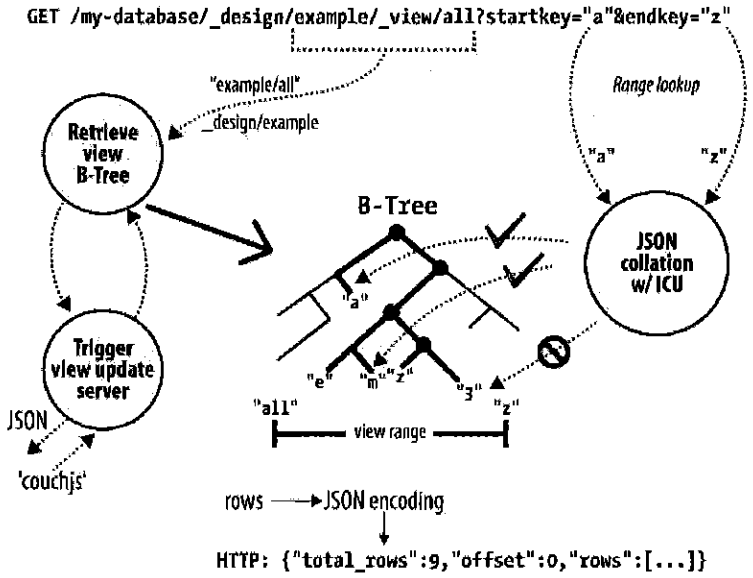
Les différentes bases NoSQL libres

- **Les différentes bases NoSQL libres**
  - Les bases orientées "documents" : CouchDB, MongoDB
  - Les bases orientées "clé-valeur" : Riak, Redis
  - Les bases orientées "colonnes" : Hadoop, Hbase, Hypertable, Cassandra

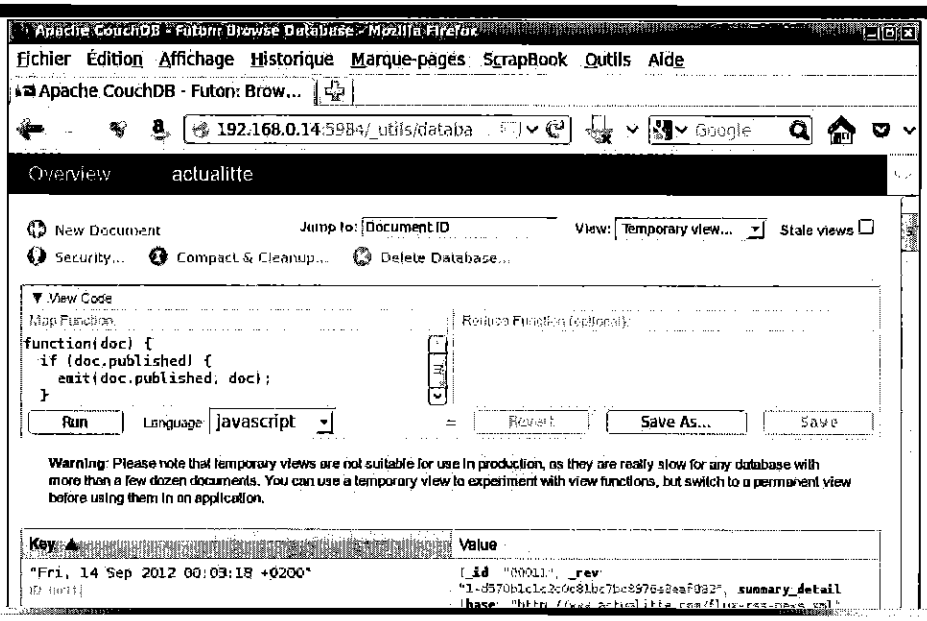
CouchDB

- Licence Apache
- Erlang
- Moteur de stockage B-Tree
- Orienté document = JSON
- Interface REST
  - Donc interrogeable en Javascript
- Peut contenir des applications entières
- Vues en MapReduce

CouchDB – B-Tree



MapReduce en CouchDB



CouchDB – les vues

- Utilise MapReduce en interne
  - Traitement isolé de chaque document – donc distribuable
  - MapReduce makes use of two functions, “map” and “reduce,” which are applied to each document in isolation.
  - Being able to isolate these operations means that view computation lends itself to parallel and incremental computation.
  - Being able to access results by key alone is a very important restriction because it allows us to make huge performance gains.

CouchDB – jointures ?

- Pseudo jointures par collation de documents

```
function(doc) {
  if (doc.type == "post") {
    map([doc._id, 0], doc);
  } else if (doc.type == "comment") {
    map([doc.post, 1], doc);
  }
}
```

CouchDB – Design Documents

- Documents qui contiennent du code
  - Documents avec un id qui commence par _design/
  - Vous pouvez mettre des pages statiques en pièce jointe
  - Peuvent contenir des instructions de reformatage
    - Équivalent CouchDB du XSLT
- Serveur de requêtes par défaut en Javascript
  - Mais d'autres serveurs de requêtes existent

CouchDB – suivi de modifications

- CouchDB permet de s'abonner aux modifications d'une collection de documents.
- http://localhost:5984/actualitte/**_changes**
- Fonctionnalités de :
  - Long polling
  - Suivi en continu
  - Filtrage
- Le suivi de modification permet la réplication

CouchDB en REST

- Futon, interface web
  - http://127.0.0.1:5984/_utils/
- Utiliser REST avec l'outil curl
  - curl -X PUT http://127.0.0.1:5984/ephemere
  - curl -X GET http://127.0.0.1:5984/_all_dbs
  - curl -X DELETE http://127.0.0.1:5984/ephemere
- Servir directement depuis CouchDB = CouchApps

Répartition en CouchDB ?

- Rien n'est pévu en natif pour faire du sharding.
- Solutions :
  - Réplications manuelles utilisant des réplications filtrées
  - Utilisation de BigCouch
    - Application des principes de Dynamo à CouchDB
    - https://github.com/cloudant/bigcouch
  - CouchBase
    - Damien Katz, le créateur de CouchDB, arrête son développement pour se concentrer sur CouchBase.
    - Couchbase = CouchDB + memcached

Choisir CouchDB ?

- Tout dépend de votre charge
- Si elle est raisonnablement importante, CouchDB est un choix agréable
- Sinon, vous allez avoir des ralentissements
  - Pas prévu pour une latence faible sur des gros besoins
- Éviter REST sur des besoins de faible latence
- Beaucoup d'écritures ? -- MVCC devient un problème
  - Espace de stockage
  - Phase de compactage coûteuse
- Fonctionnement gourmand – disque et RAM

MongoDB

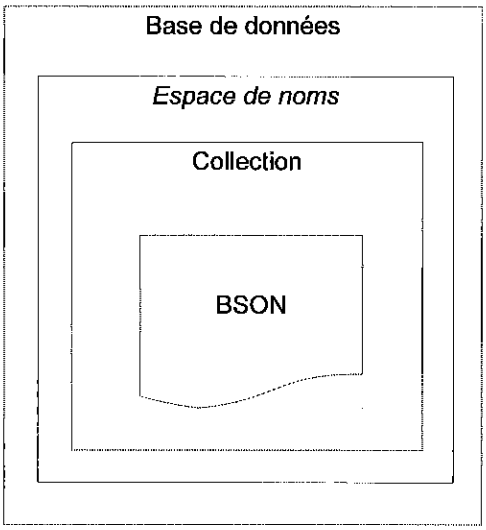
- Société 10Gen – C++
- AGPL
- Documents BSON
- Indexation
- Réplication maître-esclave
- Sharding
- Gestion de documents avec GridFS

MongoDB – caractéristiques

- Gère des collections de documents
- Invite interactive Javascript
- Pilotes pour différents langages développés par 10Gen
- durabilité des transactions, assurée par une journalisation read-ahead redo log
  - Désactivable
- Sharding théoriquement jusqu'à mille noeuds

MongoDB – documents

- Les documents BSON (JSON "binaire") sont conservés dans des collections
- La collection est optionnellement préfixée par un espace de noms
- Les collections / namespaces sont maintenus dans une base de données



MongoDB – détails techniques

- Durabilité assurée par un WAL / RARL
  - Le journal est supprimé si MongoDB stoppe proprement. Si on trouve un journal au démarrage, on le rejoue.
  - Pour désactiver : démarrage avec --nojournal
- Index secondaires en B-Tree
  - Index sparses : seuls sont indexés les documents qui contiennent l'élément.
- Interface REST possible, à indiquer dans le fichier de configuration
  - rest = true dans `/etc/mongodb.conf`

MongoDB – sharding et réplication

- **Sharding**
- MongoDB effectue un sharding automatique
- Impose une clé immutable
- Requêtes redirigées vers le shard par le démon mongos
- Des démons mongod séparés pour la config

- **Réplication**
- Le modèle de réplication est maître-esclave
- En général 3 (impair pour former quorum)
- MongoDB choisit et promeut le maître automatiquement

- Un process MongoDB est appelé un noeud
- En général un noeud par machine

Réplication :

Les nœuds peuvent être liés dans une topologie de réplication maître-esclave.
Elle inclut un basculement automatique du maître. À un moment donné, un nœud est choisi comme maître par MongoDB et reçoit donc toute les écritures, qu'il distribue aux esclaves. En cas de défaillance du nœud maître, MongoDB bascule automatiquement et promeut un autre nœud maître. Les applications clientes sont averties et redirigent toujours les écritures vers le bon maître dans un jeu de réplicas.

Le jeu de réplicas est constitué d'un nombre impair de processus, Pour toujours d'avoir deux machines qui vont former quorum pour élire un nouveau maître.

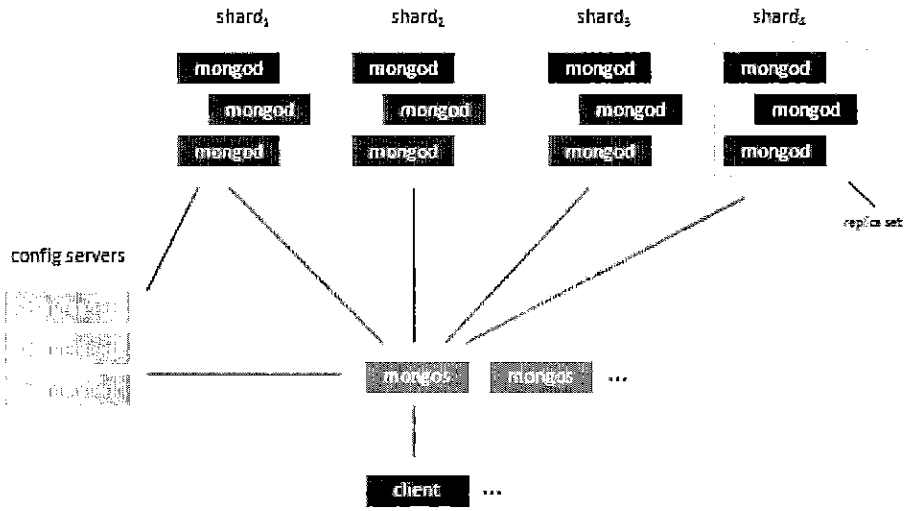
Si vous disposez d'un nombre pair de nœuds, vous pouvez ajouter un arbitre, qui est un démon mongod léger dont le but est uniquement de former quorum.

Inutile de dédier un serveur pour l'arbitre, il peut très bien tourner sur une machine déjà occupée à autre chose.

la réplication est facile à mettre en place.
démarrez vos démons avec l'option --replSet <NomDeRéplica>
Puis créer un document de configuration :
Nous créons un document de configuration, et nous initialisons la réplication :

```
config = {_id: 'ReplicaSet1', members: [
            {_id: 0, host: 'localhost:15000'},
            {_id: 1, host: 'localhost:15001'},
            {_id: 2, host: 'localhost:15002'}]
        };
rs.initiate (config);
```

MongoDB – sharding et réplication



MongoDB – sharding et réplication

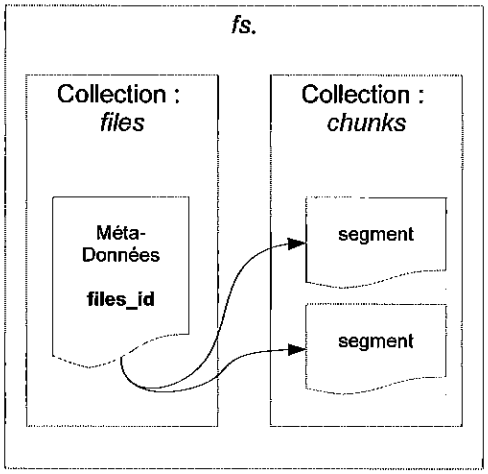
- Support au niveau client:
- Connexion :
  - `<?php $m = new Mongo("mongodb://localhost:27017,localhost:27018"); ?>`
- Écriture en mode synchrone :
  - `db.articles.insert(article, safe=True)`
- Nombre d'écritures sur les réplicas :
  - db.articles.insert(article, w=2)
  - db.articles.insert(article, w=majority)

MongoDB – administration

- Sauvegarde et restauration :
  - Mongodump
    - Sauvegarde souple, par database ou collection, avec un filtre ...
  - Mongorestore
    - Restauration tout aussi souple, une database, une collection, ...
- Import / export
  - Mongoimport
    - Sources : CSV, TSV, JSON
  - Mongoexport
    - Exporte une collection dans les mêmes formats

MongoDB – GridFS

- La taille d'un document BSON est limité à l'heure actuelle à 16 mo
- Les documents binaires peuvent être stockés dans GridFS
  - Séparés en segments, par défaut 256 ko
  - API inclus dans les drivers
  - Utilitaire mongofiles



MongoDB – mini SIG

- Gère pour l'instant seulement des points, pas de polygones
- Création d'index géographiques
- Opérateurs de recherche géographiques, par exemple:
  - $near – géométrie plane
  - $nearSphere – géographie sphérique

MongoDB – invite interactive

- Invite : mongo → JavaScript (SpiderMonkey)

```
show dbs;
use passerelles;
show collections;
db.articles.find().forEach(printjson);

db.articles.ensureIndex({date_création=-1});

var doc = {
     "auteur":
     [...]
} ;
db.articles.save(doc);
```

MongoDB
utilisation d'un index secondaire

- Exemple de recherche en utilisant le driver Python

```
query = {
      "mots-clés":"afrique",
      "catégories":"éducation"
}

articles = db.articles.find(query).sort(("date_création",
pymongo.DESCENDING))
for article in articles:
      print article.get("auteur")
```

```
articles = db.articles.find(
{"catégories":"éducation" }).limit(10)

article = db.articles.find_one({"mots-clés":"afrique",
      "catégories":"éducation",
      "date_création": "12/05/2012 23:30"})
```

Choisir MongoDB ?

- Choix intéressant pour les besoins moyens
- Raisonnablement rapide avec index
  - Meilleur en tout cas que CouchDB
- Approche orientée développeur
- Possibilités de tuning
- Il y a encore mieux pour les besoins de faible latence
  - Un SGBDR peut faire mieux sur un environnement non shardé

Riak

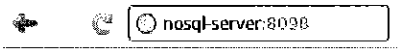
- Société Basho
- Développé en Erlang
- Architecture sans maître
- Recherche full text intégrée
- Conception fortement orientée vers la distribution des données
- Licence libre ou commerciale si besoin de fonctionnalités d'entrerprise

Riak – détails techniques

- Interfaces REST ou Protobuf
  - `cli = riak.RiakClient(host="192.168.0.22", port=8087, transport_class=riak.RiakPbcTransport)`

Interface REST

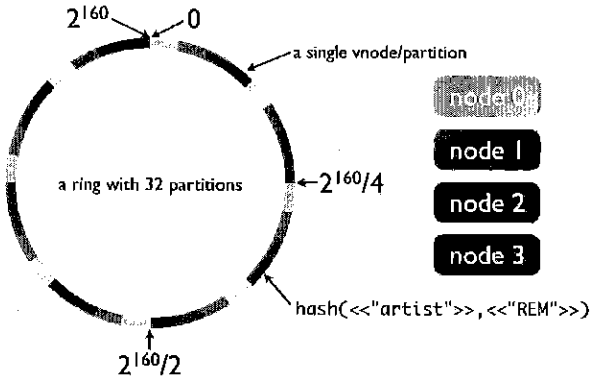
```
curl -X PUT -H "Content-Type: application/json" -d '
{
       "auteur":
       {
              "prénom":"Annie",
              "nom":"Brizard",
              "e-mail":"annie.brizard@cocomail.com"
       },
       "titre":"pourquoi les éléphants ont-ils de grandes
oreilles ?"
}' http://localhost:8098/buckets/passerelles/keys/1?
returnbody=true

curl -X GET
http://localhost:8098/buckets/passerelles/keys?keys=true

curl -X GET
http://localhost:8098/buckets/passerelles/keys?keys=stream

curl -X GET
http://localhost:8098/buckets/passerelles/keys/1

curl -X DELETE
http://localhost:8098/buckets/passerelles/keys/1
```



- riak_kv_wm_buckets
- riak_kv_wm_buckets
- riak_kv_wm_index
- riak_kv_wm_keylist
- riak_kv_wm_link_walker
- riak_kv_wm_link_walker
- riak_kv_wm_mapred
- riak_kv_wm_object
- riak_kv_wm_object
- riak_kv_wm_ping
- riak_kv_wm_props
- riak_kv_wm_stats

Riak

- Maintient une table de hachage distribuée, qui réplique les valeurs de façon à se garantir contre la perte d'un noeud.
- Technique du ring map
- Redistribution automatique en cas d'ajout de noeuds



Riak – configuration des noeuds

- Configuration aisée
- `./riak-admin status | grep ring_members`
- La première machine est automatiquement membre du cluster, il faut ajouter les autres :
- `sudo ./riak-admin cluster join riak@192.168.0.22`
- `sudo ./riak-admin cluster plan`
- `sudo ./riak-admin cluster commit`

Choisir Riak ?

- Implémentation solide
- En amélioration
- Optimisé pour l'accès aléatoire, pas pour les traitements volumineux
  - = pas d'analytique, à réserver au temps réel
- Les index secondaires sont maintenant disponibles
  - Recherches ok, mais certaines opérations sur i2 sont lentes.
- Ne pas utiliser le MapReduce en Javascript, il est trop lent
  - Utiliser le MR en Erlang

Redis

- Base de données “en mémoire”
- Possible de persister les données
- Entrepôt de paires clé-valeurs typées
  - Chaînes
  - Listes, hashs, sets, sorted sets
- On est ici dans le domaine de la faible latence – rapidité !
- Est c'est du C, pas du Java

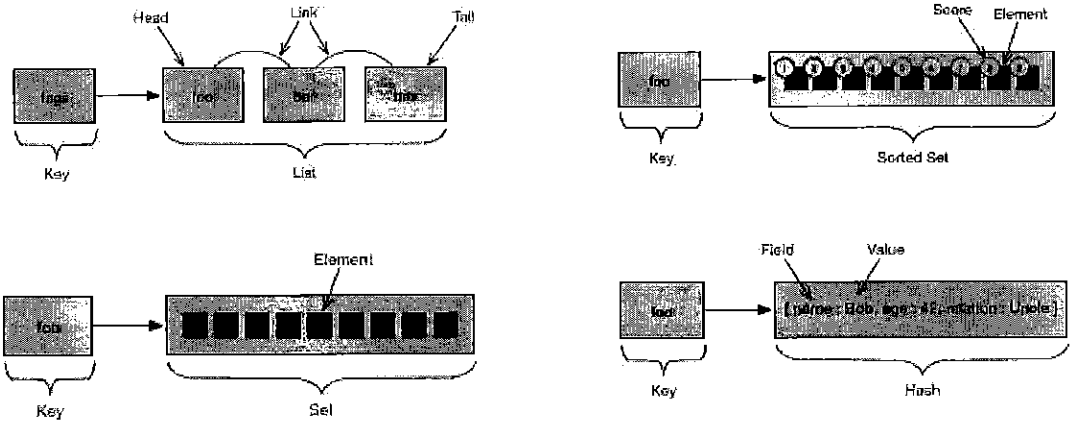
Redis est un moteur de base de données en mémoire (on pourrait l'appeler un serveur de structure de données) développé à la base par un programmeur italien nommé Salvatore Sanfilippo. Son développement à partir de 2009 a été très rapide et sa popularité fut foudroyante. En 2010, VMWare recruta Salvatore, puis Pieter Noordhuis, un contributeur important de Redis, pour travailler à plein temps sur le développement du produit, tout en le laissant en licence libre (BSD).

Redis

- **Re**mote **Di**ctionary **S**erver
  - Version 1.0 : 2009
  - Version 2.0 : 2010
  - Version stable : 2.4.1
- Évolution de memcached
- Développé à partir de 2009 par Salvatore Sanfilippo
  - Développement et corrections de bugs rapides
- En 2010, VMWare engage Salvatore pour travailler à plein temps sur Redis
  - + Pieter Noordhuis, contributeur Redis

Redis – richesse des types
et des opérateurs

- Oui, c'est du clé-valeur, mais enrichi par rapport à memcached



Redis – types de données

- **Chaîne** – binary-safe : INCR "1"
- **Liste** – liste de chaînes, ordre décidé à l'insertion
  - LPUSH (gauche) , RPUSH (droite)
- **Ensemble** (set) – collection non triée, excellentes performances
  - opérateurs ensemblistes : union, intersection et différence
- **Hachage** (hash) – dictionnaire, stockage très optimisé en mémoire (stockage de mio de hashs ok)
  - Chaque hash peut stocker 2^{32}-1 paires
- **Ensemble trié** (sorted set) – chaque élément est associé à un score.

Redis – clés-valeurs

- La clé peut contenir n'importe quels caractères
  - On peut générer des namespaces :
    - articles:1:tags
  - Sans aller trop grand, une clé n'a pas besoin d'être très petite pour les performances
- Une chaîne Redis est “binary-safe”
  - On peut lui appliquer des opérateurs de calcul
  - On peut y stocker du binaire, un objet sérialisé, etc.

Redis – configuration

- Comme le reste, très simple
- Les bases de données sont simplement des numéros, des buckets.
- Ne tourne pas en démon par défaut

redis.conf

```
daemonize yes
port 6379
bind 127.0.0.1
timeout  120
loglevel notice
logfile /var/log/redis.log
syslog-enabled no
databases 16
```

Redis – richesse des commandes

- Des commandes sont disponibles pour chaque type de données. Exemple pour les hashs :

HDEL	supprime des clés d'un hash
HGETALL	retourne la table de hachage dans une clé
HINCRBY	Incrémente la valeur d'une clé
HKEYS	retourne toutes les clés d'une table
HLEN	retourne le nombre de clés d'une table
HMGET	retourne les valeurs des clés en paramètre
HMSET	attribue des valeurs à des clés
HSETNX	attribue une valeur seulement si elle n'existe pas
HVALS	retourne toutes les valeurs d'une table

Redis
Fonctionnalités et Programmation

- Invite interactive : redis-cli
  - Permet de saisir les commandes de base de Redis
  - et (plus récemment) d'exécuter des scripts lua.
  - Lance des commandes dans un shell script. Ex.
    - Redis-cli monitor
- Belle quantité de clients
  - 5 pour PHP par exemple
- Support du pipelining
- Pseudo-transactions
  - MULTI
  - WATCH
- TTL par clé
- Pub/Sub

Redis – exemples de fonctionnalités

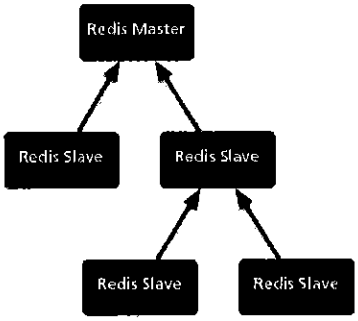
- Les listes peuvent être utilisées comme des capped collections.
  - Commandes LPUSH, LTRIM
- Les listes peuvent être utilisées comme des piles
  - Commandes BRPOP, BLPOP
- Tops et Rankings avec des sorted sets
  - ZRANGE ... WITHSCORES
  - ZRANGEBYSCORE

Redis – Pub/Sub

- Publish / Subscribe = design pattern pour un échange orienté messages
  - Des messages sont envoyés dans un canal
  - Des récepteurs s'abonnent au canal
  - Faiblement couplé = pas besoin de connaître l'émetteur ou le récepteur
- Commandes
  - PUBLISH
  - SUBSCRIBE / PSUBSCRIBE – UNSUBSCRIBE / PUNSUBSCRIBE

Redis – persistance et réplication

- **Réplication** maître-esclave simple et très facile à mettre en oeuvre



- **Persistance** configurable
  - Rien – on est en mémoire
  - AOF – journalisation des écritures en apprend-only
  - RDB – snapshot persisté sur disque à intervalles réguliers

Redis – administration et surveillance

- Commande MONITOR pour suivre les requêtes
- Redmon : interface web tierce
- Redis-Sentinel
  - En développement, mais déjà utilisable en production
  - Monitoring
  - Notification
  - Basculement automatique

Choisir Redis

- OUI
  - Rapide comme l'éclair
  - Opérations en mémoire
    - Listes de préférences,
    - Personnes connectées
    - Analytiques
    - => beaucoup d'écritures et de changements
  - Structures “relativement” simples
- NON
  - Volumes de données importants
  - Besoins transactionnels
  - Sécurité des écritures
  - Données d'archives
  - Besoins complexes de traitement des données
  - single-threaded

Bases orientées colonnes

OLTP

- Bases orientées documents
- Paires clé-valeur
- **MapReduce**

OLAP

- Bases orientées colonne
- **MapReduce**

MapReduce est partout

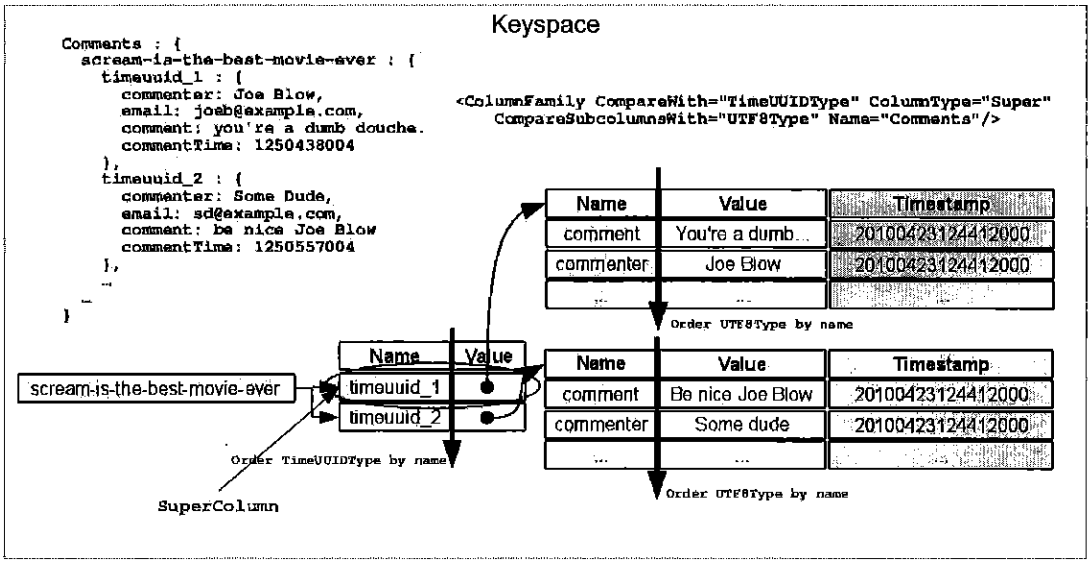
Apache Cassandra

- Base orientée colonne, très utilisée, développée originellement par Facebook
- Totalement distribué et décentralisé
- Hybride lignes / colonnes : stockage regroupé par famille de colonnes
- Montée en charge verticale des lectures ET écritures
- Réplication des données multi-noeuds automatique
- Langage CQL
- Supporte MapReduce, Pig et Hive

Cassandra – configuration

- /etc/cassandra/cassandra.yaml
  - Configuration du cluster
  - Token du noeud
  - Authenticator
  - Authority
  - Partitioner
  - Caches
  - Seed provider
  - Snitch
- La configuration des bases (keyspaces) est faite dans le keyspace system.

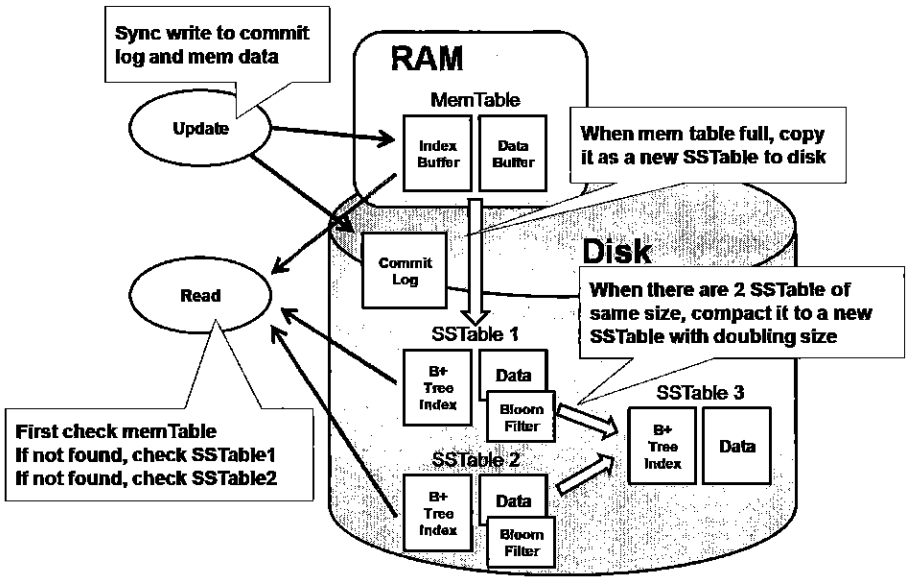
Cassandra – modèle de données



Cassandra – Modèle de données

- Keyspace = supertable, voire base de données
- Column family = groupe de colonnes
  - Statique : métadonnées définies (nom des colonnes)
  - Dynamique : le nom de la colonne est choisi à l'insertion
    - Il y a des validateurs et des comparateurs
- Colonnes spéciales :
  - Expirantes – TTL défini, effacées et supprimées au compactage
  - Compteur – maintient un compteur
  - Super-colonnes

Cassandra – Architecture



Cassandra – create keyspace

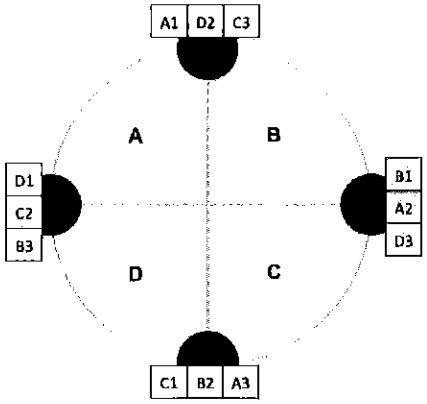
```
create keyspace CarDataStore
        with replication_factor = 1
        and placement_strategy =
'org.apache.cassandra.locator.SimpleStrat
egy';
```

```
CREATE KEYSPACE <ks_name>
   WITH strategy_class = <value>
   [ AND strategy_options:<option> = <value>
[...] ];
```

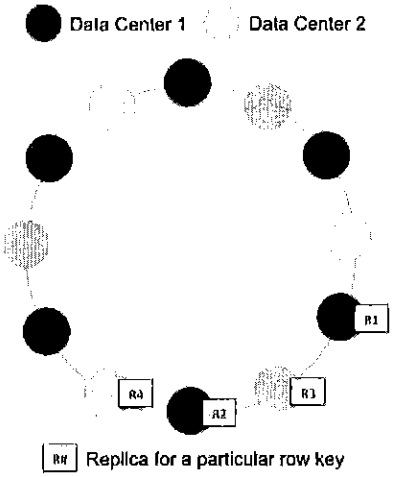
- Replication Strategy
  - SimpleStrategy
    - Sans considération de rack ou de data center
  - NetworkTopologyStrategy
    - Plusieurs data centers
    - Tente de répliquer les données entre data centers
    - endpoint_snitch doit être configuré

Cassandra – Replication Strategy

- SimpleStrategy



- NetworkTopologyStrategy



Cassandra – snitches

- Stratégie de mappage des IP avec des data centers
- SimpleSnitch – une seule zone
  - strategy_options: replication_factor=<#>
- DseSimpleSnitch – DataStax Enterprise – mixage Hadoop/données
  - strategy_options: Analytics ou Cassandra
- RackInferringSnitch – déduction du data center basé sur l'IP
  - xxx.data-center.rack.node
- PropertyFileSnitch
  - /etc/cassandra/cassandra-topology.properties
- EC2 ...

Cassandra – outil nodetool

- Donne des informations en temps réel sur le noeud.
  - Info
  - Cfstats
  - Cfhistograms
  - Tpstats
  - Compactionstats
- Permet d'effectuer des opérations
  - Gestion de l'anneau : rejoindre, quitter
  - Activer / désactiver Thrift
  - Compactage
  - Réparation (arbres de Merkle)

Cassandra – données et cache

- /var/lib/cassandra par défaut
  - Emplacement des données, du commit log et des caches sauvegardés
- Cassandra peut maintenir des caches en mémoire.
  - Configurable via cassandra.yaml
  - Cache des clés – accès aux clés plus rapide, taille raisonnable du cache
  - Cache des lignes – cache intégral des données. Plus volumineux, à tuner.

Cassandra – index secondaires

- À indiquer dans la définition de la famille de colonnes
  - index_type : KEYS = table de hachage

```
update column family articles with
    key_validation_class = int
    AND
    column_metadata =
    [
        {column_name: 'auteur_prenom', validation_class: UTF8Type},
   {column_name: 'auteur_nom', validation_class: UTF8Type,
        index_type :KEYS},
        {column_name: 'auteur_email', validation_class: UTF8Type},
        {column_name: 'titre', validation_class: UTF8Type},
    ];
```

Cassandra – cohérence paramétrable

- "tunable consistency" à l'écriture et à la lecture

ANY	écriture au moins sur un nœud, même si c'est un *hinted handoff*
ONE	L'écriture ou la lecture doit être faite au moins sur un nœud. L'écriture doit être inscrite dans le commitlog et la memtable.
QUORUM	L'écriture ou la lecture doit être faite sur au moins le quorum des réplicas de cette donnée, (N/2 + 1)
LOCAL_QUORUM	au moins sur le quorum des réplicas de cette donnée dans le data center local.
EACH_QUORUM	L'écriture ou la lecture doit être faite sur au moins un quorum des réplicas de cette donnée présent dans chaque data center.
ALL	L'écriture ou la lecture doit être faite sur tous les réplicas.

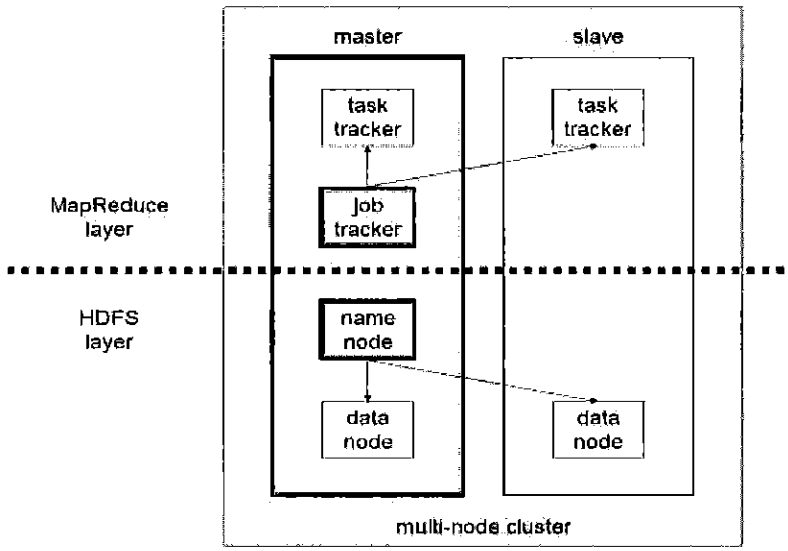
Cassandra 1.0 – Compression

- Fonctionnalité demandée depuis longtemps
- Diminution des I/O, très favorable en lecture
  - Intéressante en écriture aussi, pas de seek avant un write, les SSTables sont immutables
- Compression dans la SSTable et dans la memtable
- Algorithmes :
  - SnappyCompressor (bibliothèque Snappy)
    - Plus rapide
  - DeflateCompressor (Java zip)
    - Plus compact

Hadoop

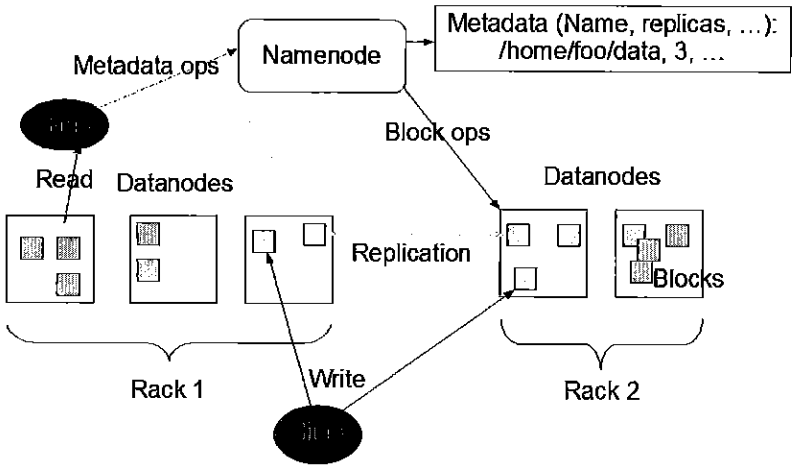
- Projet Apache
- Supporté maintenant partout
  - Tous les clouds, MapReduce implémenté dans beaucoup de moteurs NoSQL
- HDFS
- Hive et Pig

Hadoop

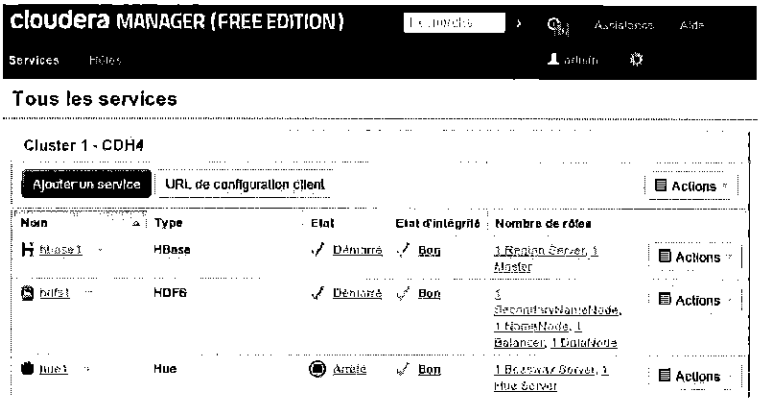

master
slave
task tracker
task tracker
Job tracker
MapReduce layer
HDFS layer
name node
data node
data node
multi-node cluster

HDFS

HDFS Architecture



Hadoop packagé : Cloudera



- Installation packagée des modules du stack Hadoop pour les principales distributions

Hive

- Système de DataWarehouse utilisant Hadoop et HDFS pour l'analyse des données
- Requêtage à l'aide de HiveQL
- Pas loin du ressenti d'un moteur relationnel

```
CREATE TABLE u_data (
  userid INT,
  movieid INT,
  rating INT,
  unixtime STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH 'ml-data/u.data'
OVERWRITE INTO TABLE u_data;
```

Pig

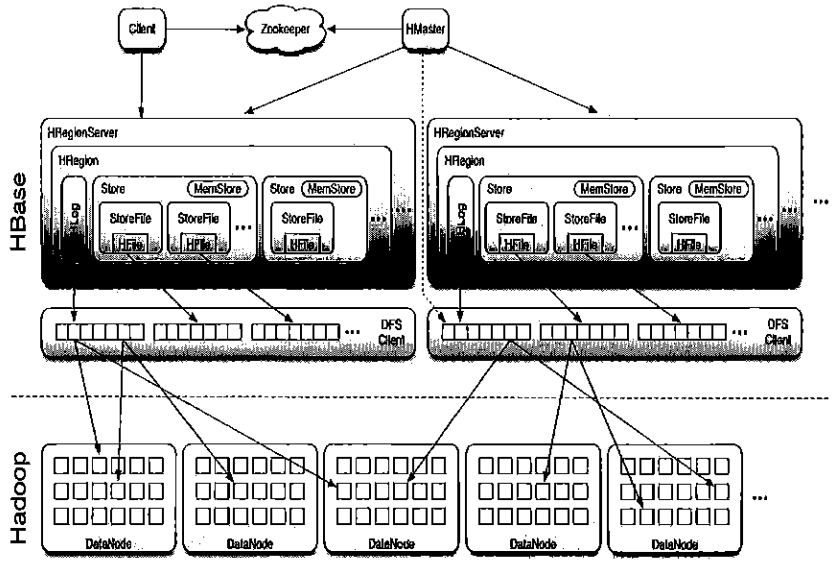
- Moteur d'extraction de données utilisant Hadoop pour paralléliser les requêtes
- Utilise un langage “latin”
  - Langage d'extraction et de manipulation de données semi-impératif
  - Architecturé autour de MapReduce
  - Traitement distribué de données volumineuses

```
A = load 'passwd' using PigStorage(':');
B = foreach A generate $0 as id;
store B into 'id.out';
```

HBase

- Base de données orientée colonne, libre et open source
- Implémentation du concept BigTable sur HDFS
- Très performant pour le Big Data
  - Gère des petabytes sur des milliers de machines
- Plus proche du concept de tables que les entrepôts document
- Scriptable en Jruby
- Développement actif, soutenu par des grands (Facebook)

Hbase – architecture générale

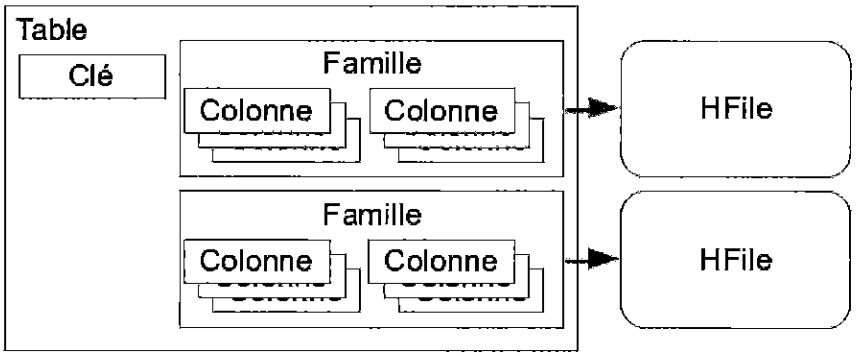


Hbase – démons

Démon	description
RegionServer	Démon de gestion des régions, qui permettent de distribuer les tables.
Zookeeper	Service centralisé de maintenance de configuration et de synchronisation pour les applications distribuées. *http://zookeeper.apache.org/*. C'est le pendant du Chubby de Google. Il sert à maintenir l'état des noeuds, à promouvoir un nouveau maître si le courant est tombé, etc.
HMaster	Le démon Hbase, le moteur de bases de données lui-même. Un Hmaster gère plusieurs RegionServer et centralise la gestion d'un cluster Hbase.
HDFS	Gère la distribution du stockage au niveau du système de fichiers

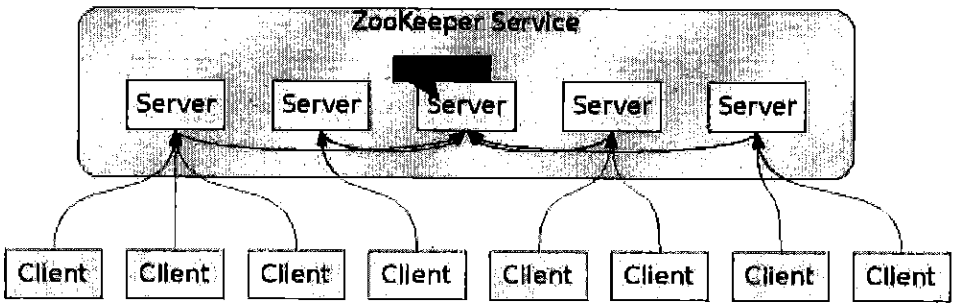
Hbase – modèle de données

- Une **table** est composée de **clés**, et de **familles de colonnes**, qui contiennent des **colonnes** versionnées par un **timestamp**.
- Le stockage est organisé par famille de colonnes



Zookeeper

- Coordinateur centralisé de la fondation Apache
  - Consensus
  - Gestion de groupes
  - Détection de présence



Hbase – programmation client

- Java : org.apache.hadoop. hbase.client
- Autres langages : Thrift ou Avro
  - Utilisation de Thrift "natif" sans surcouche
  - Ou quelques pilotes en développement par la communauté, par exemple happybase pour Python
- Interface REST disponible

Hbase – administration

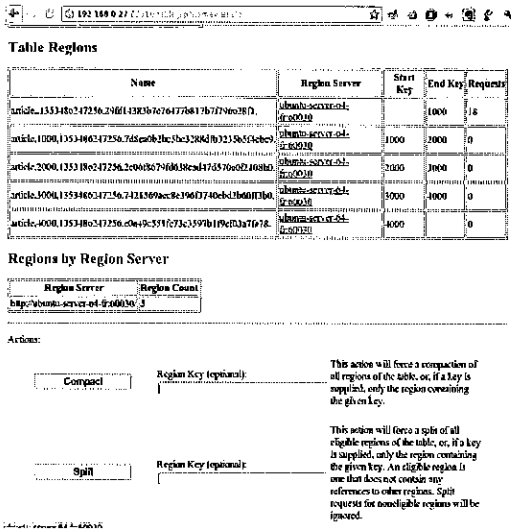


Table Regions

Name	Region Server	Start Key	End Key	Requests
article,,1353480247250.29fdf143f3b7e76477b817b7f79be3f3.	ubuntu-server-04:60020		1000	18
article,1000,1353480247250.7d5ea0b2bc5bc32804fb3235b53ebe9	ubuntu-server-04:60020	1000	2000	0
article,2000,1353480247250.2c00d679fdd38ead47d570e0f2468bD.	ubuntu-server-04:60020	2000	3000	0
article,3000,1353480247250.7428369aec8e196f3740eb42b60ff3b0.	ubuntu-server-04:60020	3000	4000	0
article,4000,1353480247250.c0e49c591fc73e3597b1f9ef03a7f678.	ubuntu-server-04:60020	4000		0

Regions by Region Server

Region Server	Region Count
http://ubuntu-server-04:60030/	5

Actions:

Compact — Region Key (optional): — This action will force a compaction of all regions of the table, or, if a key is supplied, only the region containing the given key.

Split — Region Key (optional): — This action will force a split of all eligible regions of the table, or, if a key is supplied, only the region containing the given key. An eligible region is one that does not contain any references to other regions. Split requests for noneligible regions will be ignored.

- Interface web
- Invite interactive
  - hbase shell
- Live backup via dumps

Choisir HBase

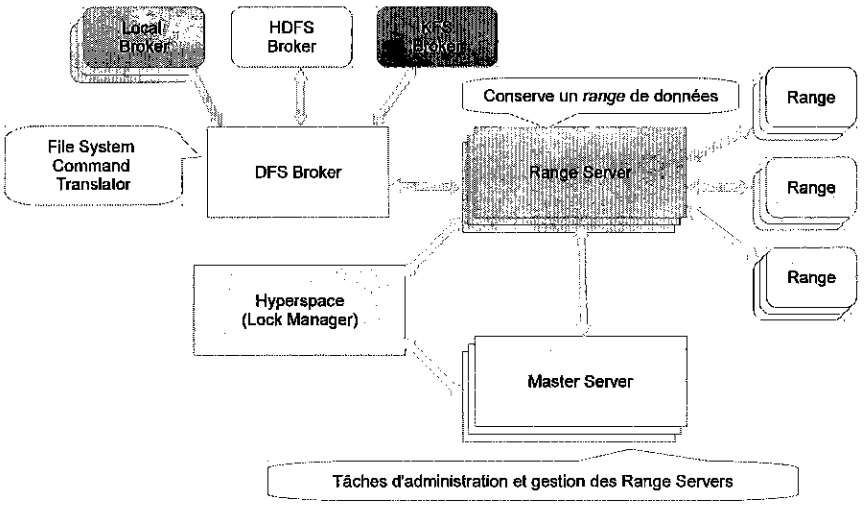
- **Oui**
  - À réserver aux volumes très importants
  - A plus de sens dans un environnement Java
  - Très solide et capable de monter en charge presque à l'infini
- **Non**
  - Sans intérêt sur un système non distribué
  - Complexité amenée par les différentes couches : Hadoop, HDFS, les différents démons, etc.
    - Il faut en avoir vraiment besoin
    - Il faut savoir ce qu'on fait
  - Besoins temps réel

Hypertable

HYPERTABLE INC

- Basé sur Google BigTable
- Développé en C++, recherche les performances
- Distribué par le biais d'un système de fichiers distribué tiers
  - HDFS
  - Kosmix KFS

Hypertable


Local Broker
HDFS Broker
KFS Broker
File System Command Translator
DFS Broker
Conserve un *range* de données
Range Server
Range
Range
Range
Hyperspace (Lock Manager)
Master Server
Tâches d'administration et gestion des Range Servers

Hbase / Hypertable

- **Hbase**
- Plus utilisé, plus grande communauté
  - Hadoop, projet Apache
- Robustesse
  - Par exemple, analytique et très larges volumes
- Haute disponibilité avec Zookeeper

- **Hypertable**
- En production sur
- Performances
  - Par exemple site web avec fort trafic
- haute-disponibilité en travail
  - Hyperspace (implémentation de Chubby (donc Paxos))

Évolutions de la hiérarchie

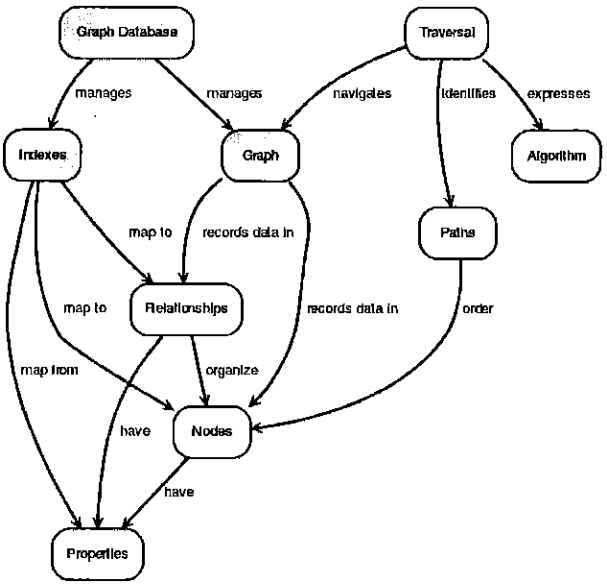
- **SGBDR**
- ORACLE
  - CONNECT BY
- SQL Server
  - CTE récursive
  - HierarchyId
- PostgreSQL
  - Ltree
- MySQL
  - Hmmmm ??

- **NoSQL**
- On passe aux graphes
- Neo4J
- Gephi
- FlockDB

http://docs.postgresqlfr.org/8.3/ltree.html

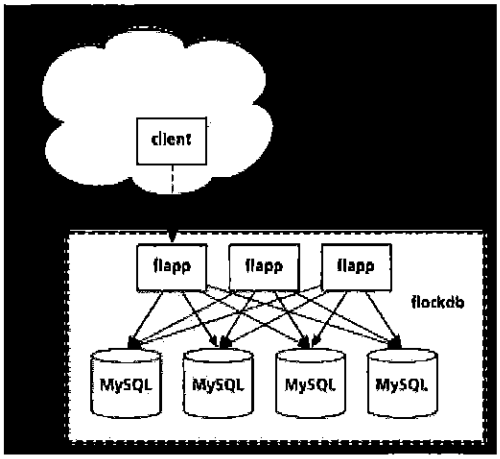
Neo4J

- Ecrit en Java
- Simple à mettre en oeuvre
- Interface REST
- Langage déclaratif Cypher
- Langage Gremlin
  - Script non déclaratif
- Distribué



FlockDB

- C'est Twitter, c'est du réseau social
- Normalement, le travail du graphe, c'est le voyage
  - FlockDB est plutôt optimisé pour la liste des adjacents
- FlockDB est en développement



Mettre en place une solution NoSQL

- **Mettre en place une solution NoSQL**
  - Les choix matériels. Les critères de sélection
  - Migrer ses données vers le NoSQL
  - Qu'est-ce qu'un modèle de données NoSQL ?
  - Les étapes importantes de la migration
  - Les impacts sur le développement client

Choix matériels

- On s'éloigne de la pure informatique d'entreprise
  - Commodity hardware
  - Disques locaux, remplaçables
  - Stockage dans le cloud
  - Montée en charge horizontale, ajout de nouvelles machines
  - Selon les systèmes, ne pas lésiner sur la RAM
    - Memcached, redis, riak

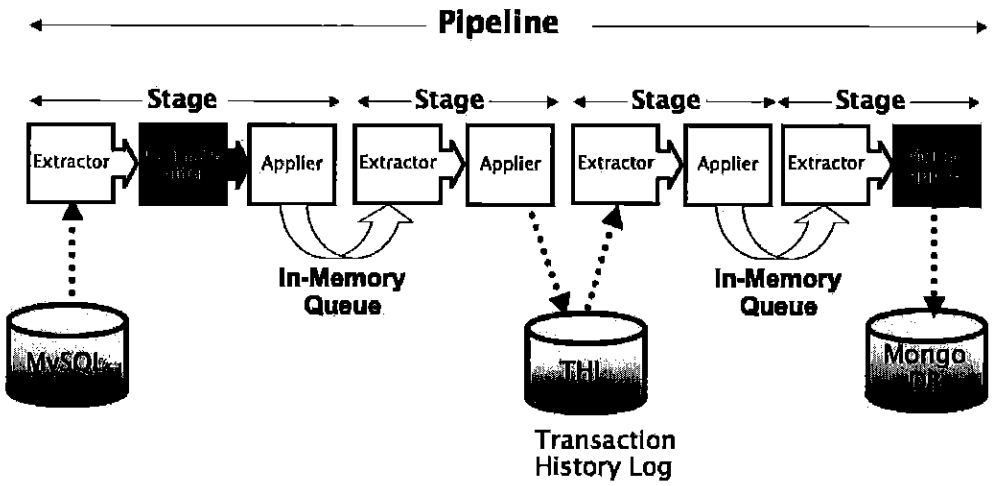
Tester sa solution

- Mode pseudo-distribué
  - HDFS par exemple
  - Hbase en mode pseudo-distribué est pénible à mettre en place, et n'apporte rien de plus qu'un test en mode local
- Hadoop sans HDFS
- Simulation de sharding et réplication sur la même machine
  - MongoDB, CouchDB
- Utilisation de machines virtuelles
  - Virtualbox fonctionne très bien

Migration ou interconnexion ?

- Un des avantages des bases SQL est leur capacité à communiquer – les données sont importables et exportables, non isolées
  - Pilotes ODBC
  - ETL
  - Imports / exports CSV
- Chaque système NoSQL implémente son stockage, la communication est-elle plus difficile ? Pas forcément
  - Structures de données simples et standard – clé-valeur, JSON
  - Interfaces standard – REST, Thrift
  - Sinon : scripts

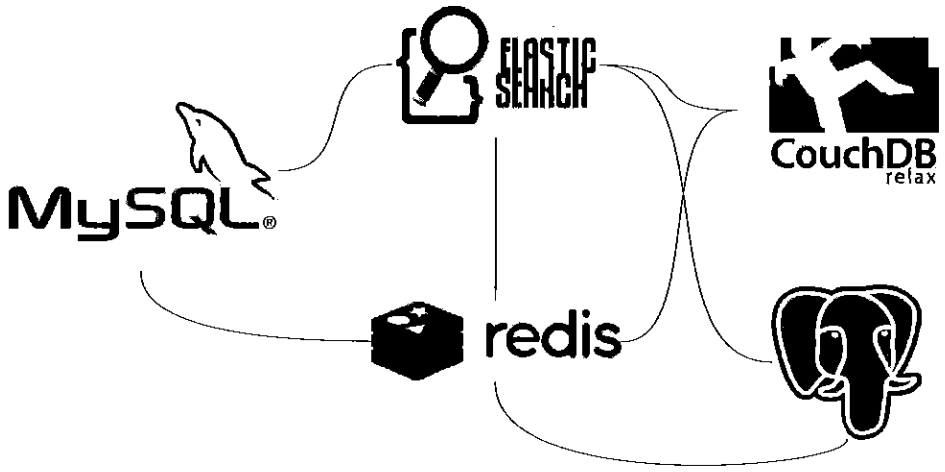
Migration ou interconnexion ?



Migration ou interconnexion ?

- Esprit Unix
- "By and large, Facebook's engineering culture has tended towards choosing the best tools and implementations available over standardizing on any one programming language and begrudgingly accepting its inherent limitations".
- Exemple de Skyrock : redis, couchdb, mongodb, fluxy, DynamoDB, mysql

La fertilisation croisée



ElasticSearch

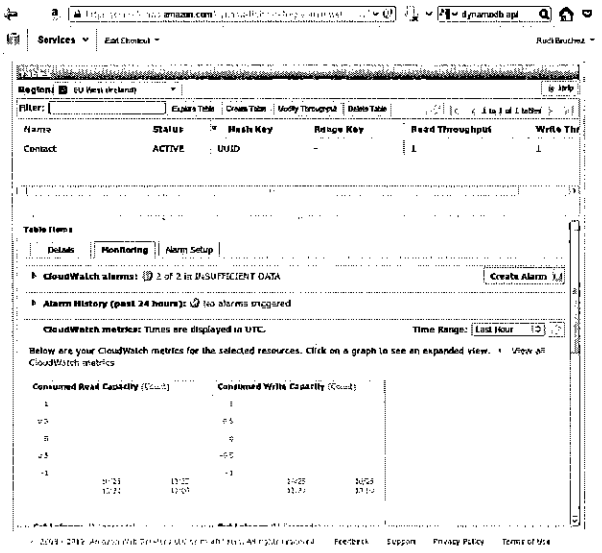
- Basé sur Apache Lucene
- Propose une API REST / JSON par-dessus Lucene
- Recherche distribuée
- Fonctionnalités très riches, élégantes
  - Simplicité de mise en place et d'utilisation
  - Distribution automatique (Lucene n'est pas parallèle, ElasticSearch le parallélise)
  - Richesse des recherches
- Se marie à merveille avec les bases NoSQL

ElasticSearch – intégration

- Collecte par rivières (Rivers)
  - CouchDB
  - MongoDB
- Ou directement :
  - Wikipedia
  - Twitter
  - RSS
- Facettes
  - Analyse multidimensionnelle
- Sharding
- Réplication

DynamoDB

- l'offre d'Amazon qui a eu la croissance le plus rapide de tous leurs services
  - en juin l'offre avait déjà dépassé les prévisions d'Amazon pour 2012.
- permet de monter en charge sans s'occuper des détails techniques
- API pour Java, .NET ou PHP



Maintenir et superviser ses bases NoSQL

- **Maintenir et superviser ses bases NoSQL**
  - Comment assurer la sécurité ?
  - Quels sont les outils de surveillance ?
  - Comment assurer les performances ?
  - A quoi correspond l'optimisation d'un système NoSQL ?

Sécurité des accès

- La sécurité n'est pas une priorité des moteurs NoSQL

Moteur NoSQL	Type de sécurité
MongoDB	Tout ou rien
CouchDB	Admin Party, autentification HTTP
HBase	Authentification SASL (relativement nouveau)
Cassandra	Possibilité d'utiliser des authenticateurs
Redis	Pas vraiment d'aunthentification, mais pas vraiment une base de données
Riak	Uniquement version commerciale
Hypertable	Rien encore

Surveillance et supervision

- Outils intégrés dans chaque moteurs
- Solution commerciale As a Service : http://www.serverdensity.com/ avec plugins
- Solutions libres :
  - Nagios avec plugins
  - Hyperic (VMWare)
  - Munin
  - Zenoss
- Solutions libres par système
  - Exemple : redisLive

Performances

- **Scale out**
  - Déploiement automatique par exemple avec puppet
  - Configuration des noeuds si besoin
- **Optimisation**
  - Index secondaires
  - Correction du code client
  - Fertilisation croisée
    - ElasticSearch

Aller ou non vers le NoSQL

- **Aller ou non vers le NoSQL**
  - A quels usages correspondent les bases de données NoSQL ?
  - Pour quel type de données et de volumétrie ?
  - Comment se présente le futur ?

Choisir son utilisation

NoSQL	relationnel
Réseau sociaux	ERP
Messagerie	e-commerce
Analytique	Panier d'achat (?)
Catalogue	Finance
Documents	

Quelques scénarios d'utilisation documents et in-memory

- MongoDB chez SFR
- Skyrock
- Blogs, chats, ...
- Gestion des données du web
- Amazon Dynamo – panier d'achat
  - Donc bonne cohérence de données distribuée

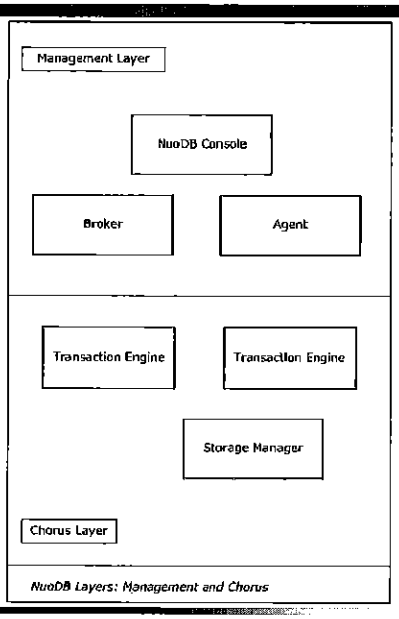
Deux mots sur "NewSQL" -- 1/ VoltDB

VoltDB The NewSQL database for high velocity applications

- Michael Stonebreaker
- Éditions commerciale et communautaire
- ACID compliant partitionné en shared-nothing
- Procédures stockées en Java et SQL
- Impression vieille école

Deux mots sur "NewSQL" -- 2/ NuoDB

- Autrefois NimbusDB.
  En beta 8
- ACID
- Elastique
- En Java



Google Spanner

- Spanner est le successeur annoncé de BigTable
- Réplication cross datacenter
- Par de système de fichiers distribué (GFS)
  - Sharding à travers des groupes Paxos
- Transactions complètement ACID
  - 2PC via des gestionnaires de transactions
- "TrueTime API"
  - Synchronisation de la distribution à l'aide de GPS et d'horloges atomiques

Une émergence ?

- Nosql-discussion
  - "Bonjour, je suis nouveau sur ce groupe, je travaille activement avec NoSQL depuis un an, et j'ai décidé de créer mon propre moteur NoSQL. Au début je voulais juste voir ce que j'étais capable de faire, et maintenant c'est un peu comme un rêve qui devient réalité"
- Les bases de données fascinent
  - L'économie du savoir
  - L'algorithmique est un truc de vieux geeks

La spécialisation des besoins

- Le marché n'est plus unique
- Les bases de données objet et XML ont échoué car
  - Elles ne répondaient pas à un besoin généralisé
  - Les habitudes étaient prises
  - Aucun engouement
- NoSQL fait du bruit