Emergence et principes des BdD NoSQL

Motivations et émergence

Motivation et émergence

1969-1970 : arrivée du modèle relationnel

Rappel des principes :

- Repose sur des relations entre les valeurs des données
  (indépendamment de leur emplacement en mémoire)
- Manipulation à travers une algèbre et un langage de haut niveau
- Dissocie représentation-et-interrogation du stockage, et sera quand même efficace!
  (les moteurs de SGBD finiront même par être plus efficace que les solutions *ad hoc*)

Mais des contraintes :

- Toutes les lignes d'une Relation ont les mêmes colones
  (valeur *NULL* si absence de données)
- Modifications par séquences atomiques pour que la BdD soit toujours totalement cohérente
- Conception d'un schéma de base qui s'impose à toute la BdD
- Interrogation par *jointures* de nombreuses (petites) Relations

Motivation et émergence

1969-1970 : arrivée du modèle relationnel

On ne peut pas mettre n'importe quoi dans une BdD Relationnelle !

- Toutes les données doivent respecter le schéma initial…
  …difficile de faire entrer des données imprévues !

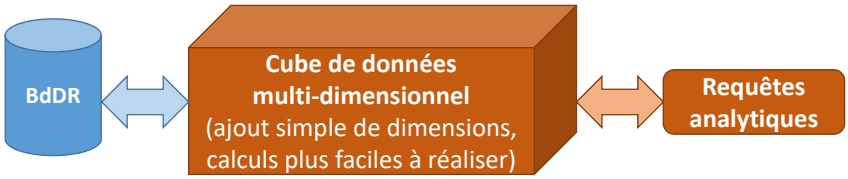
- Le langage Relationnel SQL est adapté pour extraire des informations selon des conditions sur leurs valeurs
  → requêtes OLTP (OnLine Transaction Processing) : **OK**

  …mais n'est pas adapté pour faire des calculs de statistiques complexes sur ces valeurs, ni sur des données volumineuses!
  → requêtes OLAP (OnLine Analytical Processing) : **problème…**

Motivation et émergence

1969-1970 : arrivée du modèle relationnel

De nouvelles solutions apparaissent pour les besoins en *analytics*

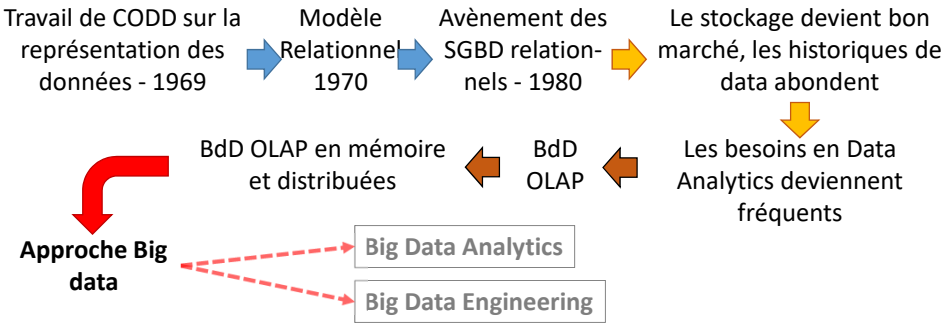


- OLAP, puis In-memory OLAP (plus rapide)
- CEP (*Complex Event Processing*) pour supporter des flux de mises à jour, et réagir automatiquement aux changements des données
- … la BdD SQL ne servait presque plus qu'à du stockage…

Mais besoin d'outils encore plus innovants : prémisses du Big Data analytics (Data Science) et du Big Data Engineering

Motivation et émergence

Evolution des technologies de BdD traditionnelles



Deux types de requêtes / d'utilisation :

Requêtes OLTP : approche transactionnelle classique des BdD Relationnelles

Requêtes OLAP : besoin en analyse de données, peu favorable aux BdD Relationnelles

Motivation et émergence

Théorème / problème « CAP »

Base toujours perçue cohérente, même pendant des mises-à-jour

Disponibilité garantie en l'absence de pannes

Résistance aux pannes en distribué

En mode distribué à large échelle :

- on n'a jamais toutes les data à jour en même temps
- on ne peut pas différer des requêtes chaque fois qu'on fait une maj (on ne traiterait jamais de requêtes!)
- on ne peut pas arrêter de fonctionner dès que des parties de la BdD sont en pannes

Motivation et émergence

Théorème / problème « CAP »

Base toujours perçue cohérente, même pendant des mises-à-jour

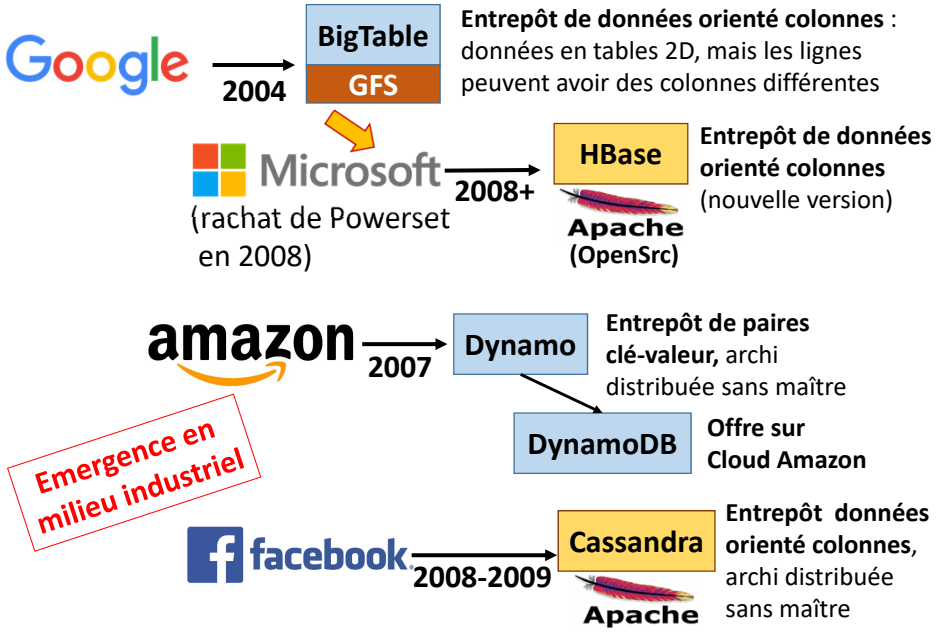
Disponibilité garantie en l'absence de pannes

Résistance aux pannes en distribué

En mode distribué à large échelle :

- **on ne peut vérifier que 2 propriétés sur 3**
  (théorème CAP, E. Brewer 2000-2002)
- **le Big Data et le NoSQL renonce surtout à la garantie de *consistency***

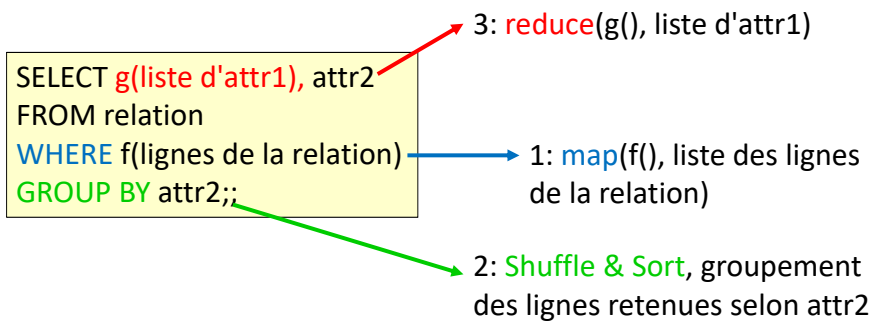
Motivation et émergence



Principes du NoSQL

Principes du NoSQL

***Map-Reduce* pour reproduire la requête SQL type :**



La solution « *Map-Reduce* » permet d'implanter facilement des requêtes « *Select-From-Where-GroupBy* » :

- sur un système distribué à grande échelle
- sur des données structurées complexes et/ou hétérogènes

Passe à l'échelle
Plus générique
Plus compliqué!

Principes du NoSQL

Résumé « NoSQL *vs* SQL » :

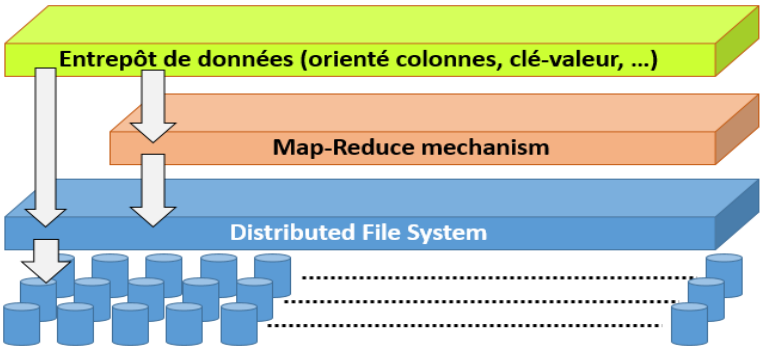
- Grande souplesse dans la nature et le format des données stockées (en résumé : pas de schéma!)
- Exploitation de très grosses volumétries en temps raisonnable grâce au relâchement des contrainte d'intégrités et de cohérence
- Augmentation des performances par la distribution massive du stockage et des traitements : s'appuie sur un mécanisme *Map-Reduce* et un système de fichiers distribué

Mais beaucoup de critiques sur :

- La faiblesse de performance d'Hadoop et des premières architectures BigData…
- La difficulté de rassembler et d'exploiter des données hétérogènes (pas de schéma… complexifie la couche applicative)

Principes du NoSQL

Architecture de principe d'une BdD NoSQL :



Certaines BdD NoSQL sont bâties :

- Au dessus d'Hadoop : HDFS et Map-Reduce d'Hadoop
- Au dessus d'HDFS mais ajoutent leur couche Map-Reduce
- Au dessus d'une architecture complètement propre (MongoDB)

Principes du NoSQL

Classification des BdD NoSQL :

- Stockage / Entrepôt de paires clé-valeur
  (Redis, Riak)
- BdD orientés documents
  (MongoDB → voir TD)
- BdD orientés colonnes
  (BigTable, HBase, Cassandra)
- BdD spécialisés pour des index inversés
  (Elasticsearch)
- BdD orientés graphes
  (Neo4J)

Les différentes solutions NoSQL

Solutions NoSQL

Entrepôts de paires clé-valeur

La solution la plus extensible (« *scalable* »), mais simple/pauvre :

- Statistiquement, la plupart des applications demandent à lire des données à partir de leurs identifiants
  → engendre le besoin de BdD stockant des paires clé-valeur
- Le composant clé de ces bases est leur fonction de hachage
  → distribution et recherche des données dans le système distribué
- Mécanisme final très efficace mais avec peu de fonctionnalités
  → développements dans la couche applicative, en *Map-Reduce*

Rmq : Initialement des *valeurs* binaires et opaques, puis des valeurs structurées hiérarchiques analysables (ex : format JSON)

Solutions NoSQL

BdD NoSQL orientées documents

On associe des clés à des documents à structure hiérarchique

- Les valeurs ne sont plus opaques
- On peut manipuler les champs des données
- Manipulation de doc web au format HTML/XML, ou doc JSON

On stocke des données prêtes à être interrogées sans jointure :

- Exploitation rapide
- Mais la jointure doit être faite lors de l'écriture
- Si besoin de croiser des informations : **plus complexe et lent**

Rmq : L'utilisation de documents structurées non opaque permet de les analyser et de produire des index inversés

Rmq : sont devenues de très grande taille, proches des entrepôts de paires clé-valeur

Solutions NoSQL

BdD NoSQL orientées colonnes

Stockent des tables 2D de *clé - ensemble de valeurs*

Ressemblent à des tables relationnelles, mais bcp plus souples

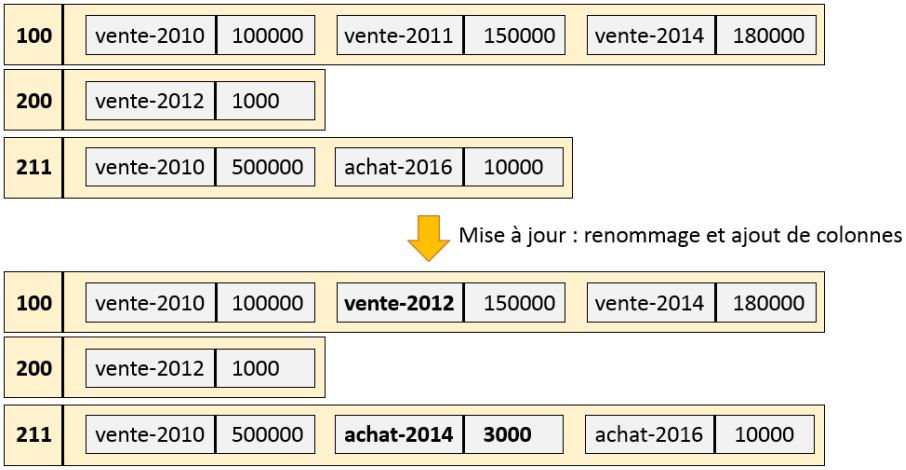
- Les lignes peuvent avoir des colonnes différentes et en nombres différents
- Les colonnes d'une ligne peuvent évoluer dynamiquement en nombre et en nom
- Pas de champ « NULL » contrairement à une table relationnelle (pas de colonne inutile dans une ligne)

Requêtes simples (minimalistes) *vs* BdD SQL, ou traitements complexes en *Map-Reduce*…

BdD NoSQL conçues pour stocker des associations one-to-many comme on en trouve très fréquemment sur le web !

Solutions NoSQL

BdD NoSQL orientées colonnes :



Chaque ligne peut avoir des colonnes différentes
Les colonnes d'une lignes peuvent évoluer dynamiquement

Solutions NoSQL

BdD NoSQL réalisant des index inversés :

Un index inversé est indispensable pour traiter rapidement les requêtes de recherche de documents par mots clés

Mais l'index inversé peut devenir TRES volumineux (plus que les documents analysés) :

- il faut le compresser
- mais pas trop pour que la décompression à la volée soit rapide
- et/ou trouver un format de compression permettant de travailler dans le format compressé

Les BdD NoSQL spécialisées en index inversés apportent:

- des algorithmes optimisés pour construire ces index
- des algorithmes de compression/décompression adaptés

Solutions NoSQL

BdD NoSQL orientées graphes :

Spécialement adaptées pour fouiller le web et les réseaux sociaux

- Apportent des stockage de graphes efficaces (par références)
- Apportent des algorithmes d'analyse de graphes optimisés et adaptés au stockage réalisé

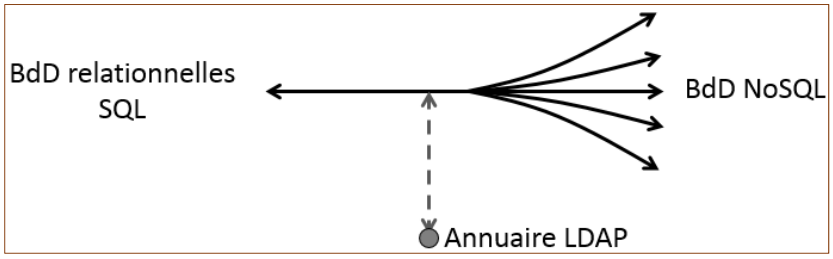
Les autres bases NoSQL pourraient stocker des graphes mais seraient moins efficaces pour les analyser

Rmq : **Neo4j** est un cas extrême de technologie très complète et très efficace pour stocker/fouiller/analyser des graphes

- Rapide (codage par ''pointeurs'')
- Stockage compact
- Répliquée sur cluster, mais pas distribuée…

Solutions NoSQL

Annuaires LDAP *vs* NoSQL :



- Stockage hiérarchique des données (très utile pour des décrire des SI)
- Plus aboutis en sécurité/contrôle des accès que les BdD NoSQL
- Implantations anciennes, moins performantes que les BdD NoSQL
- Impose un schéma hiérarchique en arbre (avec possibilité de pontage)
  → contraignant comparé aux BdD NoSQL

→ Les annuaires LDAP sont « à mi-chemin » et « à part ».

Format de données « JSON »

Format JSON

Définition

JSON : JavaScript Object Notation

Un format de données texte/ASCII, structuré et hiérarchique

Collection JSON (ou objet JSON)

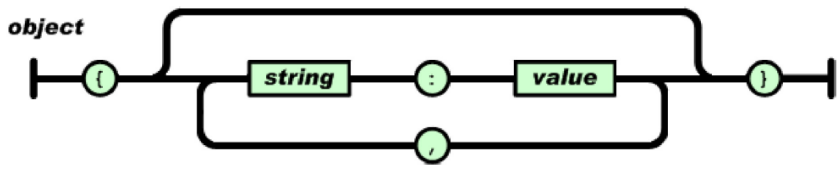
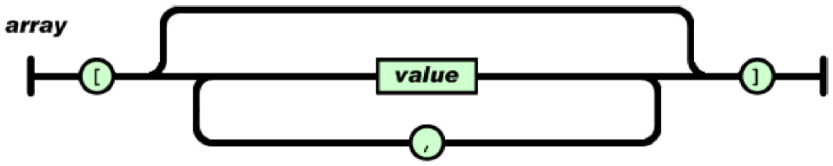
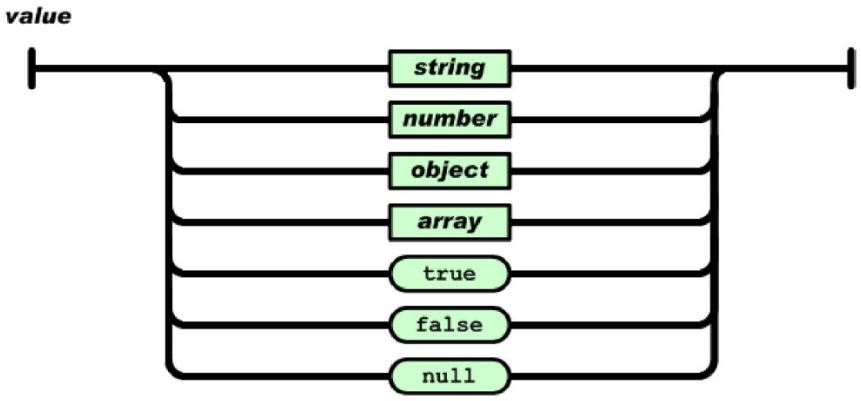


Tableau JSON



Format JSON

Valeurs JSON



Format JSON

Données textuelle, structurées et hiérarchique

```
{
 "livre" : {
   "Vue de l'esprit" : [ {"auteurs" : ["Hofstadter", "Dennet"] },
                         {"éditions" : "InterEditions"},
                         {"année" : 1987}
                       ],
   "Eagle" : [ {"auteurs" : ["Kidder"] },
               {"éditions" : "Flammarion"},
               {"année" : 1982}
             ]
 },
 "film" : {
   "2001 odyssée de l'espace" : [ {"réalisateurs" : ["Kubrick"] },
                                  {"année" : 1968}
                                ]
 },
 "livre" : {
   "Cosmos" : [ {"auteurs" : ["Sagan"] },
                {"éditions" : "Mazarine"},
                {"année" : 1981}
              ]
 }
}
```

Emergence et principes des BdD NoSQL