

 CentraleSupélec

Big Data : Informatique pour les données et calculs massifs

8 – Emergence et principes des BdD NoSQL


Stéphane Vialle

 université PARIS-SACLAY


ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

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

 CentraleSupélec

Motivations et émergence

 CentraleSupélec

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 colonnes (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

 CentraleSupélec

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...**

CentraleSupélec

Motivation et émergence

1969-1970 : arrivée du modèle relationnel
De nouvelles solutions apparaissent pour les besoins en *analytics*

BdDR ↔ **Cube de données multi-dimensionnel** (ajout simple de dimensions, calculs plus faciles à réaliser) ↔ **Requêtes analytiques**

- 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 : **prémises du Big Data analytics (Data Science) et du Big Data Engineering**

CentraleSupélec

Motivation et émergence

Evolution des technologies de BdD traditionnelles

Travail de CODD sur la représentation des données - 1969 → **Modèle Relationnel** 1970 → Avènement des SGBD relationnels - 1980 → Le stockage devient bon marché, les historiques de data abondent

↓

Les besoins en Data Analytics deviennent fréquents

↙

Bd OLAP en mémoire et distribuées ↔ **Bd OLAP** ↔ **Approche Big data**

↘

Big Data Analytics

Big Data Engineering

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

CentraleSupélec

Motivation et émergence

Théorème / problème « CAP »

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

Disponibilité garantie en l'absence de pannes

Consistency (cohérence) ↔ **Availability** (disponibilité)

Partition tolerance (résistance au morcellement)

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

CentraleSupélec

Motivation et émergence

Théorème / problème « CAP »

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

Disponibilité garantie en l'absence de pannes

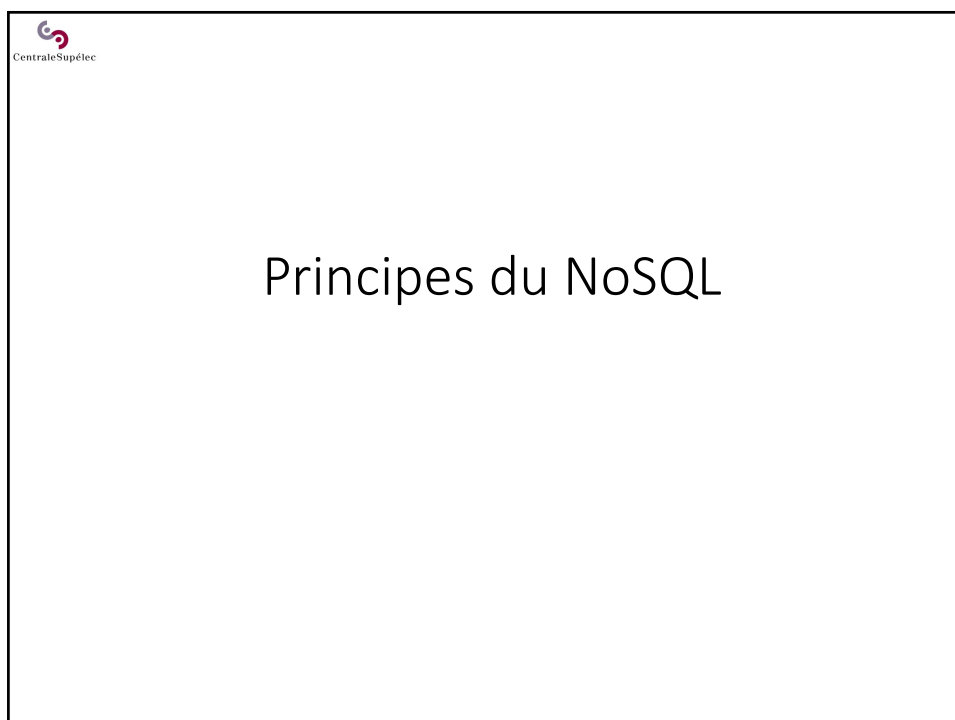
Consistency (cohérence) ↔ **Availability** (disponibilité)

Partition tolerance (résistance au morcellement)

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**



CentraleSupélec

Principes du NoSQL

Map-Reduce pour reproduire la requête SQL type :

```
SELECT g(liste d'attr1), attr2
FROM relation
WHERE f(lignes de la relation)
GROUP BY attr2;
```

3: **reduce**(g(), liste d'attr1)

1: **map**(f(), liste des lignes de la relation)

2: **Shuffle & Sort**, groupement des lignes retenues selon attr2

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

Passé à l'échelle
Plus générique
Plus compliqué!

CentraleSupélec

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 contraintes 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)

CentraleSupélec

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**)

CentraleSupélec

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**)



CentraleSupélec

Les différentes solutions NoSQL



CentraleSupélec

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)

CentraleSupélec

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 structurés 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

CentraleSupélec

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 !

CentraleSupélec

Solutions NoSQL

BdD NoSQL orientées colonnes :

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

↓ Mise à jour : renommage et ajout de colonnes

100	vente-2010	100000	vente-2012	150000	vente-2014	180000
200	vente-2012	1000				
211	vente-2010	500000	achat-2014	3000	achat-2016	10000

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

CentraleSupélec

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

CentraleSupélec

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...

CentraleSupélec

Solutions NoSQL

Annuaire LDAP vs NoSQL :

The diagram shows a central point with arrows pointing to 'BdD relationnelles SQL' on the left and 'BdD NoSQL' on the right. A dashed arrow points from 'Annuaire LDAP' below to the central point.

- 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 ».

CentraleSupélec

Format de données « JSON »

CentraleSupélec

Format JSON

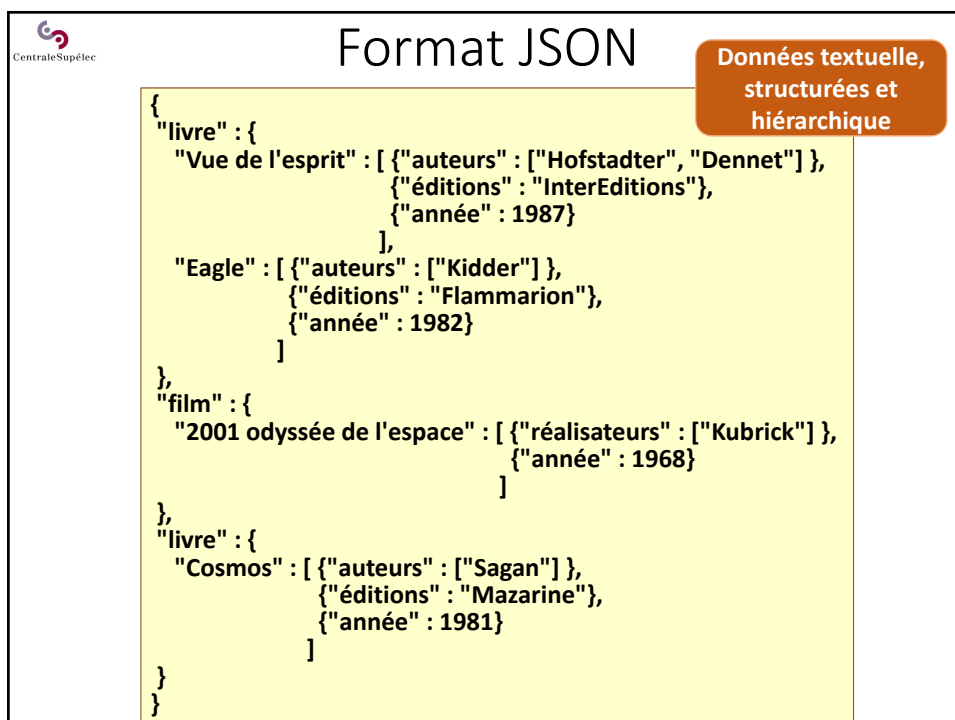
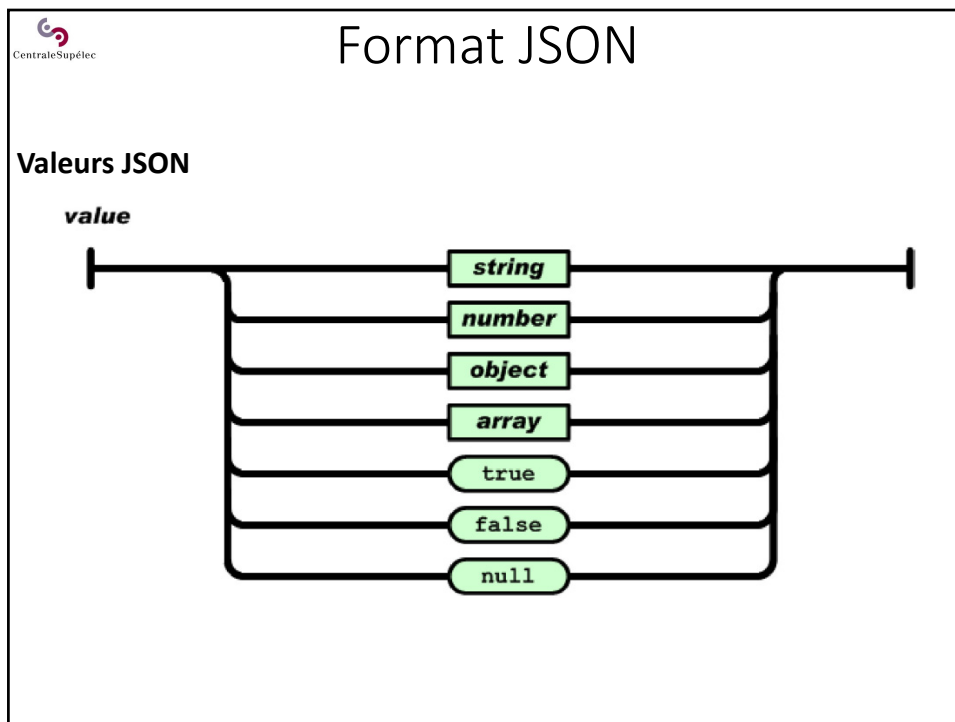
Définition
JSON : JavaScript **O**bject **N**otation
 Un format de données **texte/ASCII**, **structuré** et **hiérarchique**

Collection JSON (ou objet JSON)

object

Tableau JSON

array



Emergence et principes des Bdd NoSQL

