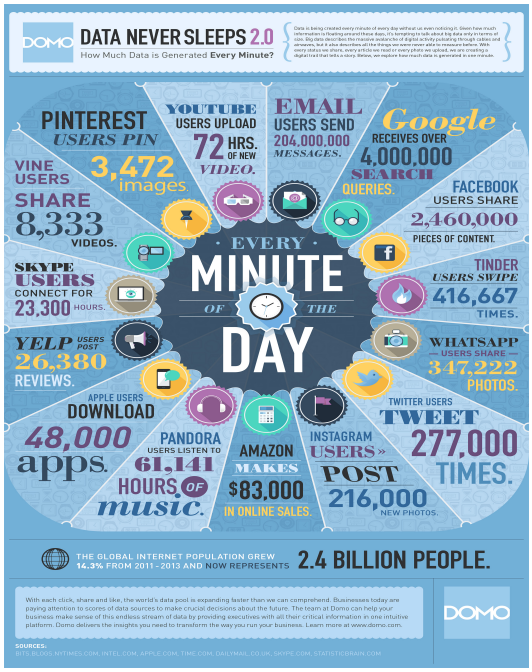
BigData & Bases NoSQL

Plan

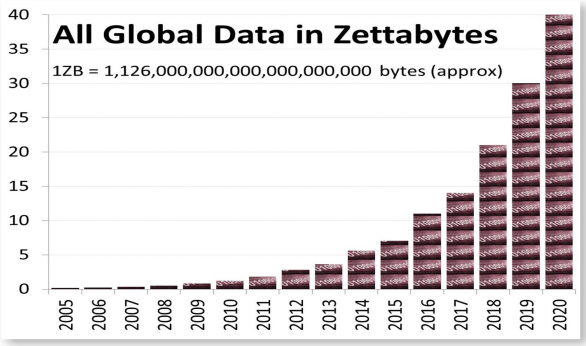
Contexte

- Croissance de la quantité des données exponentielle

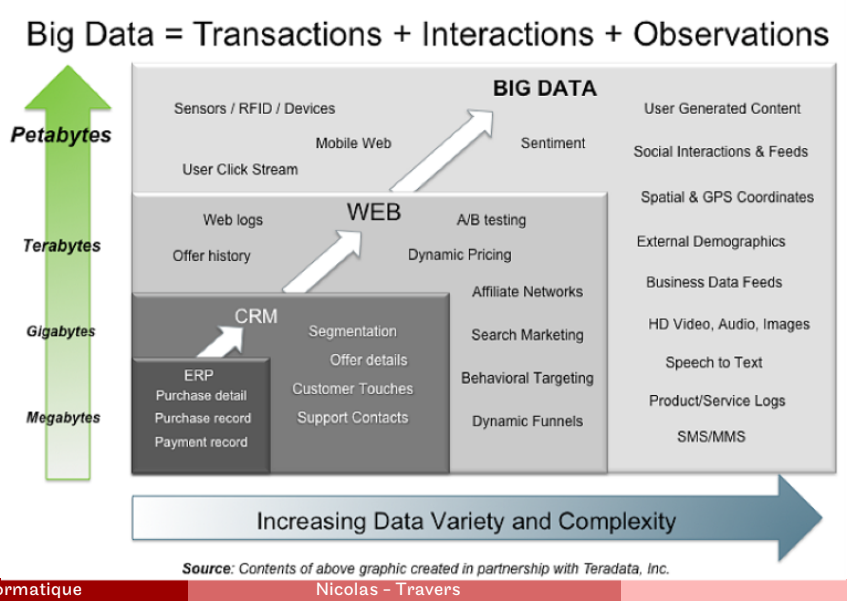


Contexte (2)

- La quantité de données digitales produites double tous les 2 ans.
- En d'autres termes, on a produit autant de données digitales ces 2 dernières années que tout ce qui a été produit auparavant.

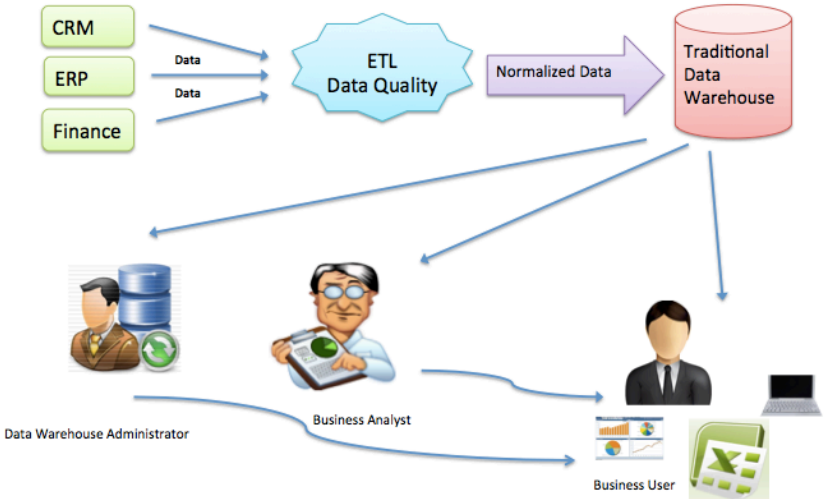


Volume et Variété



Source: Contents of above graphic created in partnership with Teradata, Inc.

Décisionnel : ancienne méthode



Décisionnel vs 3V

- L'approche classique incompatible avec les 3V du BigData :
- Le **Volume**: les entrepôts sont conçus pour gérer des Go ou To de données alors que la croissance exponentielle des données nous conduit aux Po ou Eo
- Le **type (Variety)**: le nombre de types, incluant les données textuelles semi ou non structurées, augmente
- La **vitesse (Velocity)**: les données sont créées de plus en plus vite et nécessitent des traitements en temps-réel

Contexte (3) : encore des chiffres

Compagnies	Données traitées (2014)	Données stockées (2014)
Google	100 Po	15 000 Po
Ebay	100 Po	90 Po
Facebook	600 To	300 Po
Twitter	100 To	100 To
Baidu	10-100 Po	2 000 Po
NSA	29 Po	10 000 Po

Contexte : et le business

- On estime que le volume de données professionnelles double tous les 1,2 ans
- Les ¾ des décideurs estiment que les Big Data vont affecter significativement leurs systèmes de stockage
- Le Big Data serait un marché à 50 milliards de $ en 2017
- En Europe, l'utilisation du Big Data pour améliorer l'efficacité des “traitements” permettrait d'économiser 100 milliards de $

Contexte : Conséquences

- Les volumes à gérer sans précédents impliquent :
  - Données **hétérogènes**, **complexes** et souvent liées
    - produites par des applications parfois différentes,
    - par des utilisateurs différents,
    - avec des liens explicites (par exemple citations, ancres url, etc) ou implicites (à extraire ou à apprendre)
  - **Nombreux serveurs/clusters**
    - un serveur unique ne peut stocker cette quantité d'information, garantir des temps d'accès pour grand nombre d'utilisateur, faire des calculs rapides, etc
  - **Besoin de distribuer les calculs et les données**
    - comme plusieurs serveurs/clusters, besoin d'algorithmes permettant le calcul et la distribution des données à large échelle

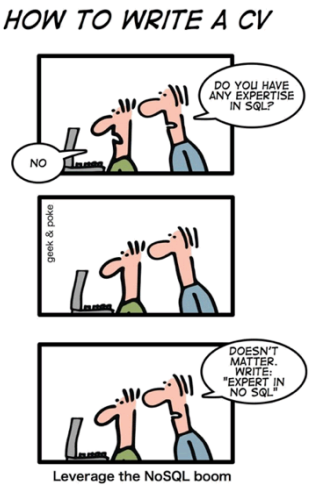
DataCenters

- Data centers de quelques grands acteurs du Big Data
- **Google DataCenter :** 70000 servers/data center et 16 data centers, ~1M de serveurs
- **Facebook :** 5 data centers
- **Amazon :** 7 data centers, 450 000 severs
- **Microsoft :** ~1M serveurs

Big Data : Exemples d'utilisation

- Décodage du génôme humain: le génôme d'une personne (env. 100Go) décodé en 30mns
- Prédiction des résultats des élections US en 2012 à partir de l'analyse de tweets
- Découverte d'un effet secondaire dû à la prise de deux médicaments par analyse des requêtes d'internautes (Yahoo)
- Étude des déplacements de population (migration, tourisme, circulation urbaine, etc)

NoSQL ne remplace pas les SGBDR



SGBDR vs Distribution

- Fonctionnalités
  - Jointures entre les tables
  - Langage d'interrogation riche
  - Contraintes d'intégrité solides

- Limites dans le contexte distribué :
  - **Comment distribuer/partitionner les données**
  - Liens entre entités -> Même serveur
  - Mais plus on a de liens, plus le placement des données est complexe

SGBDR vs Distribution

ACID vs BASE

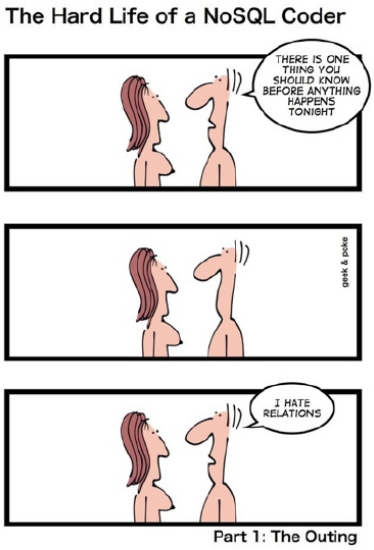
- Propriétés **ACID** pour les transactions
  - ***A**tomicité* : une transaction s'effectue entièrement ou pas du tout
  - ***C**ohérence* : le contenu d'une base doit être cohérent au début et à la fin d'une transaction
  - ***I**solation* : les modifications d'une transaction ne sont visibles/modifiables que quand celle-ci a validé
  - ***D**urabilité* : une fois la transaction validée, l'état de la base est permanent (non affecté par les pannes ou autre)
- Systèmes distribués : modèle **BASE**
  - ***B**asically **A**vailable* : garantie minimale pour taux de disponibilité face grande quantité de requêtes
  - ***S**oft-state* : l'état du système peut changer au cours du temps même sans nouveaux inputs (cela est du au modèle de consistance).
  - ***E**ventually consistent* : tous les réplicas atteignent le même état, et le système devient à un moment consistant, si on stoppe les inputs

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

NoSQL : une solution

- NoSQL : **Not Only SQL**
  - Nouvelle approche de stockage et de gestion de données
  - Permet le passage à l'échelle via un contexte hautement distribué
  - Gestion de données complexes et hétérogènes
    - Pas de schéma pour les objets
- Ne remplace pas les SGBDR !!
  - Quantité de données énorme (PétaBytes)
  - Besoin de temps de réponse
  - Cohérence de données faible

Les bases de données NoSQL



BD NoSQL : Caractéristiques

- Pas de relations
  - Pas de schéma physiques ou dynamiques
  - Notion de “collections”
- Données éventuellement complexes
  - Imbrication, tableaux
- Distribution de données (milliers de serveurs)
  - Parallélisation des traitements (Map/Reduce)
- Replication des données
  - Disponibilité vs Cohérence (pas de transactions)
  - Peu d’écritures, beaucoup de lectures

Sharding : Passage à l'échelle

- Distribution des blocs de données sur un ensemble de serveurs
- Partitionnement horizontal
- Trois types de techniques :
  1. Basée sur l'allocation de ressources : *HDFS*
  2. Basée sur une structure arborescente : *Index non-dense*
  3. Basée sur le hachage : *Hachage Cohérent*

Sharding : HDFS

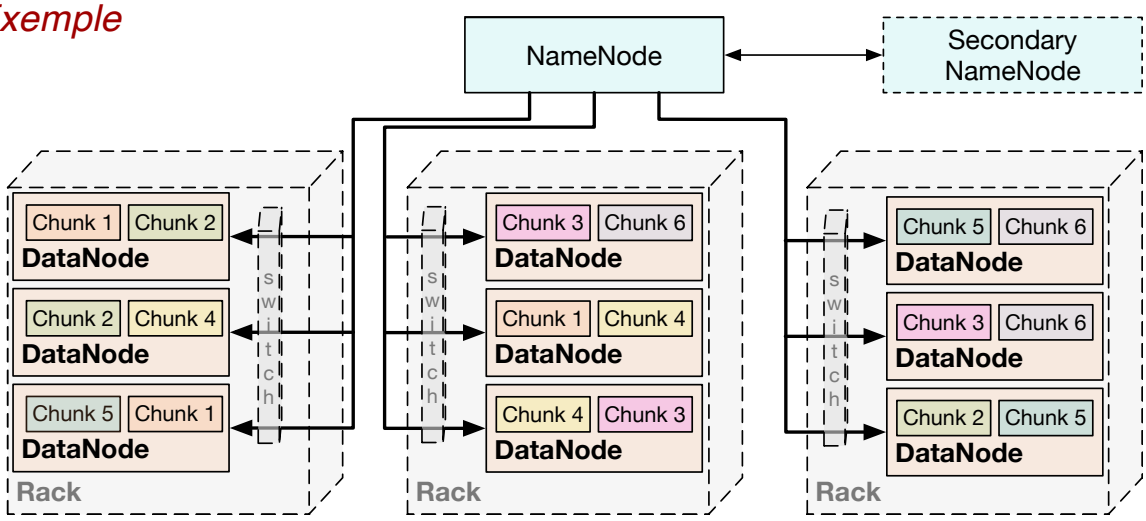
Allocation de ressources

- **HDFS**[1]
  - Système de fichier distribué
    - Dépend de la charge des serveurs
    - Distribution, tolérance aux pannes
    - Allocation dynamique et optimisée

(1) Voir cours sur *Hadoop Distributed File System* http://chewbii.com/transparents-hdfs-hadoop/

Sharding : HDFS

Exemple



Sharding : HDFS

Solutions

APACHE HBASE

Spark SQL

HIVE

Distribution de calcul

Tolérance aux pannes

Sharding : Clustered index

Technique arborescente

- **Index non-dense[2] distribué**
  - Données triées physiquement
    - Découpées en blocs sur les nœuds (souvent 256Mo)
  - Distribution, tolérance

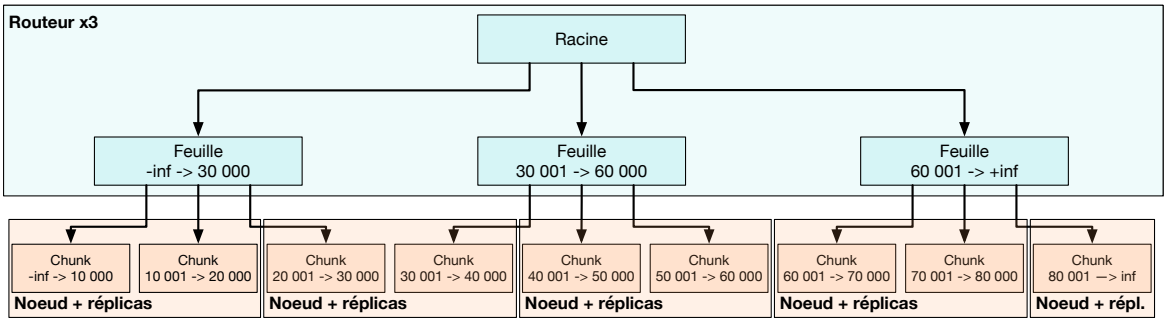
➡Requêtes par intervalles / regroupement

⚠Bien choisir la clé pour le tri

(2) Voir cours sur index dense vs non-dense http://chewbii.com/videos-optimisation-bases-de-donnees/ (Vidéo 4)

Sharding : Clustered index

Exemple



Sharding : Clustered index

Solutions

Regroupement Dynamicité

Sharding : Consistent Hashing

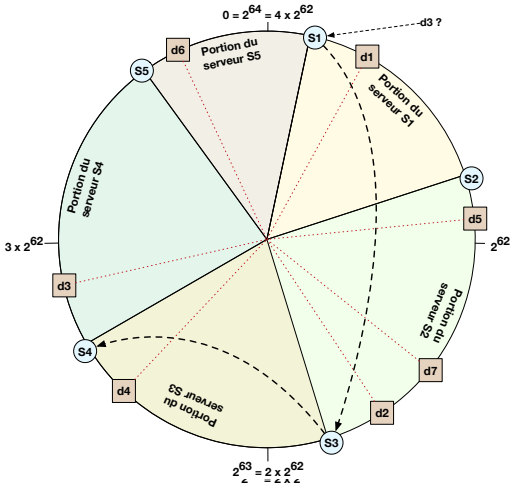
Une table de hachage distribuée

- **Hachage cohérent (DHT[3])**
  - Technique de hachage unique et dynamique pour les données et les serveurs
  - Distribution en anneau (virtuel)
  - Pas de serveur centralisé (tout est client/serveur)
  - Autonomie de gestion

(3) Voir cours sur DHT http://chewbii.com/hachagedynamique/ (Vidéo 4 & 5)

Sharding : Consistent Hashing

Exemple



Sharding : Consistent Hashing

solutions

cassandra

redis

DynamoDB

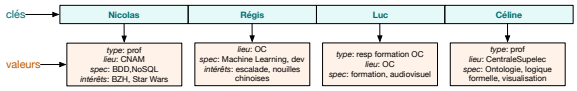
Elasticité

Auto-gestion

NoSQL : une grande famille

4 familles

Clés-Valeurs



Orienté colonnes

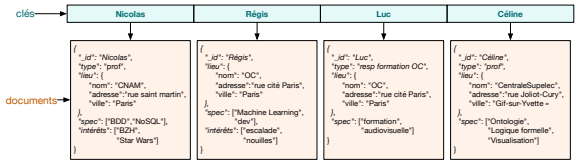
id	type
Nicolas	prof
Céline	prof
Luc	resp formation OC

id	lieu
Céline	Centrale Supelec
Nicolas	CNAM
Régis	OC
Luc	OC

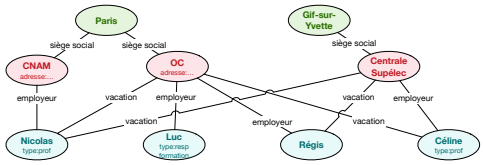
id	spec
Nicolas	BDD
Nicolas	NoSQL
Régis	Machine Learning
Régis	Dev
Luc	formation
Luc	audiovisuel
Céline	Ontologie
Céline	logique formelle
Céline	visualisation

id	intérêts
Nicolas	BZH
Nicolas	Star Wars
Régis	escalade
Régis	nouilles chinoises

Orienté documents



Orienté graphes



I - NoSQL & Clé-Valeurs

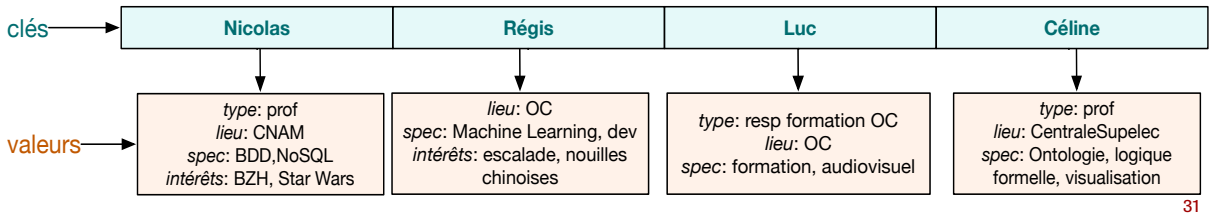
- "*HashMap*" distribué
- Couple Clé+Valeur
  - Pas de schéma pour la valeur (chaine, objet, entier, binaires…) qui peut donc être différente pour chaque
- Conséquences
  - Pas de structure ni de types
  - Pas d'expressivité d'interrogation (pré/post traitement pour manipuler concrètement les données)

I - NoSQL & Clé-Valeurs

Exemple

Relationnel

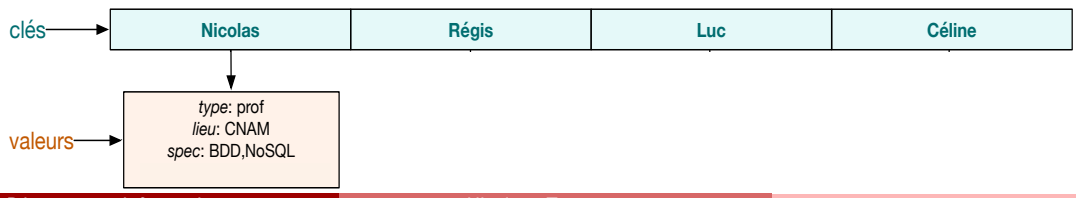
id	type	lieu	spec	intérêts
Nicolas	prof	CNAM	BDD, NoSQL	BZH, Star Wars
Régis		OC	Machine Learning, Dev	escalade, nouilles chinoises
Luc	resp formation OC	OC	formation, audiovisuel	
Céline	prof	CentraleSupelec	Ontologie, logique formelle, visualisation	



I - NoSQL & Clé-Valeurs

interrogation

- CRUD
  - CREATE (**clé**, **valeur**)
    - CREATE (**"Nicolas"**, **"type:'prof',lieu:'CNAM',spec:'BDD,NoSQL',interets:'BZH,Star Wars' "**) → **OK**
  - READ(**clé**)
    - READ(**"Nicolas"**) → **"type:'prof',lieu:'CNAM',spec:'BDD,NoSQL',interets:'BZH,Star Wars' "**
  - UPDATE(**clé**, **valeur**)
    - UPDATE(**"Nicolas"**, **"type:'prof',lieu:'CNAM,CS',spec:'BDD,NoSQL' "**) → **OK**
  - DELETE(**clé**)
    - DELETE(**"Nicolas"**) → **OK**



I - NoSQL & Clé-Valeurs

solutions

Amazon SimpleDB

Efficacité Facilité de mise en œuvre

I - NoSQL & Clé-Valeurs

applications

- Exemples d'utilisation:
- Logs de sites Web ou d'application
- Profils utilisateurs de site Web/réseaux sociaux
- Données de capteurs
- Cache Web ou BD
- Paniers sur sites de e-commerce
- …

II - NoSQL & Colonnes

- Stockage des données par colonnes
  - SGBD : tuples (lignes)

- Facile d'ajouter une colonne (pas une ligne!)
  - Schéma peut être dynamique (d'un tuple à l'autre)

II - NoSQL & Colonnes

exemple

id	type	lieu	spec	intérêts
Nicolas	prof	CNAM	BDD, NoSQL	BZH, Star Wars
Régis		OC	Machine Learning, Dev	escalade, nouilles chinoises
Luc	resp formation OC	OC	formation, audiovisuel	
Céline	prof	CentraleSupelec	Ontologie, logique formelle, visualisation	

id	type
Nicolas	prof
Céline	prof
Luc	resp formation OC

id	lieu
Céline	Centrale Supelec
Nicolas	CNAM
Régis	OC
Luc	OC

id	spec
Nicolas	BDD
Nicolas	NoSQL
Régis	Machine Learning
Régis	Dev
Luc	formation
Luc	audiovisuel
Céline	Ontologie
Céline	logique formelle
Céline	visualisation

id	intérêts
Nicolas	BZH
Nicolas	Star Wars
Régis	escalade
Régis	nouilles chinoises

II - NoSQL & Colonnes

interrogation

- Requêtes sur les colonnes
  - Combien de **professeurs (type)** à **CentraleSupelec (lieu)**

id	type
Nicolas	prof
Céline	prof

id	lieu
Céline	Centrale Supelec

id	spec
Nicolas	BDD
Nicolas	NoSQL
Régis	Machine Learning
Régis	Dev
Luc	formation
Luc	audiovisuel
Céline	Ontologie
Céline	logique formelle
Céline	visualisation

id	intérêts
Nicolas	BZH
Nicolas	Star Wars
Régis	escalade
Régis	nouilles chinoises

II - NoSQL & Colonnes

solutions

Google Cloud BigTable

APACHE HBASE

elasticsearch

Spark SQL

Agrégations

Corrélations

II - NoSQL & Colonnes

applications

- Exemples d'utilisation:
- Comptage (vote en ligne, compteur, etc)
- Journalisation
- Recherche de produits dans une catégorie (Ebay)
- Reporting large échelle (agrégats calculés sur une colonne)

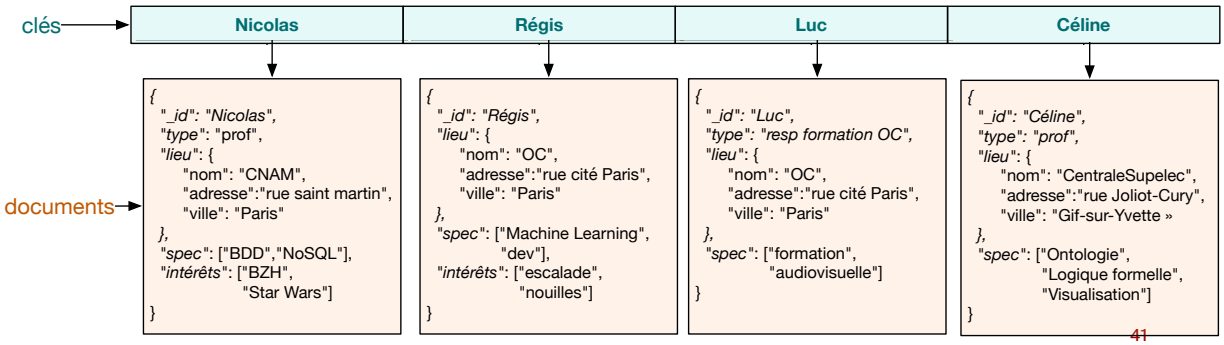
III - NoSQL & Documents

- Basé sur le modèle clé-valeur
  - Ajout de données semi-structurées (JSon ou XML)
  - Idem que "clé-valeur" mais "valeur = document"
    - Document composé de clés/valeurs
    - Types simples (Int, String, Date)
    - Schéma non nécessaire (peut varier d'un document à l'autre)
    - Imbrication de données (schéma arborescent)
    - Listes de valeurs
- Requêtes : Interface HTTP
  - Plus complexe que CRUD
  - Chaque clé du document peut être interrogée

III - NoSQL & Documents

exemple

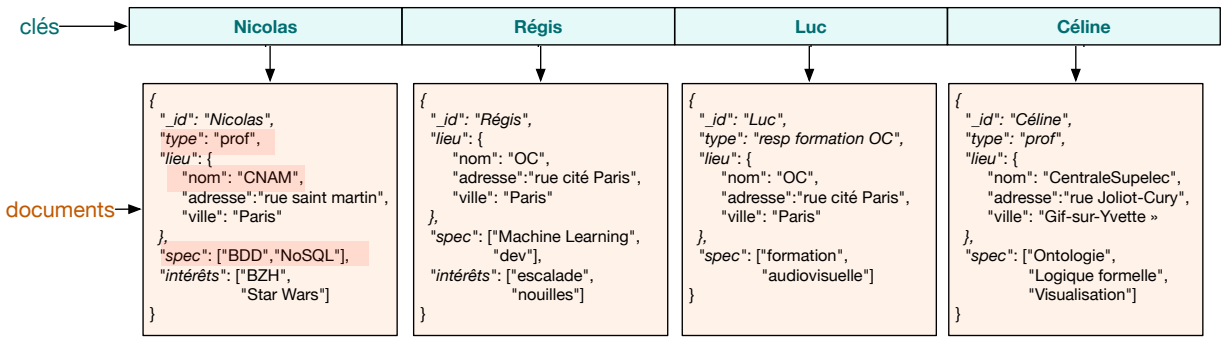
id	type	lieu	spec	intérêts
Nicolas	prof	CNAM	BDD, NoSQL	BZH, Star Wars
Régis		OC	Machine Learning, Dev	escalade, nouilles chinoises
Luc	resp formation OC	OC	formation, audiovisuel	
Céline	prof	CentraleSupelec	Ontologie, logique formelle, visualisation	



III - NoSQL & Documents

interrogation

- Requêtes sur le contenu des documents
  - **Etablissement (lieu.nom)** des **professeurs (type)** spécialisé en **BDD (*in* spec)**



III - NoSQL & Documents

solutions

Requêtes riches Gestion d'objets

III - NoSQL & Documents

applications

- Exemples d'utilisation:
- Gestion de contenu: bibliothèques numériques, collections de produits, dépôts de logiciels « xxxStores », collections multimédia, etc
- Collection d'événements complexes
- Gestion de boîtes email
- Gestion des historiques d'utilisateurs sur réseaux sociaux

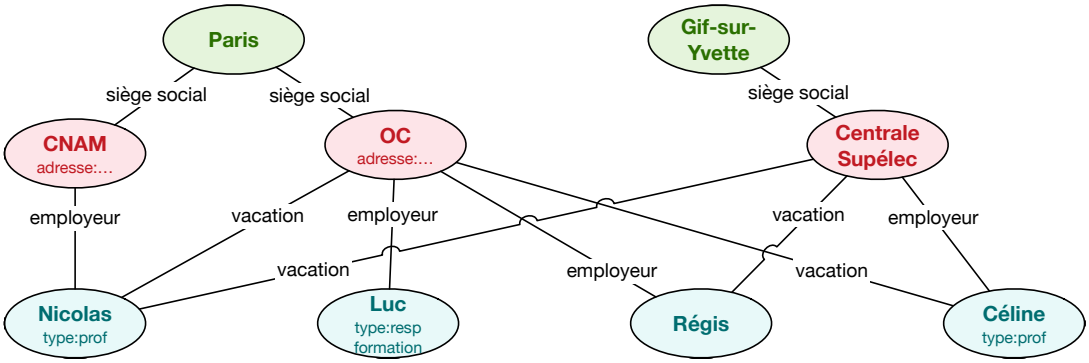
IV - NoSQL & Graph

- Stockage des noeuds, relations et propriétés
  - Théorie des graphes
  - Interrogation par traversées de graphe
  - Appel des données sur demande (parcours performants)
  - Modélisation non triviale

IV - NoSQL & Graph

exemple

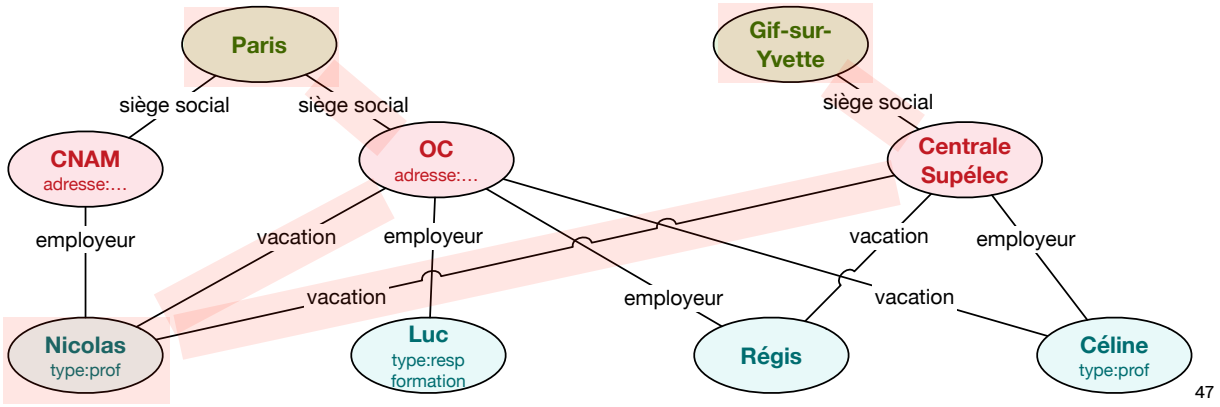
id	type	lieu	spec	intérêts
Nicolas	prof	CNAM	BDD, NoSQL	BZH, Star Wars
Régis		OC	Machine Learning, Dev	escalade, nouilles chinoises
Luc	resp formation OC	OC	formation, audiovisuel	
Céline	prof	CentraleSupelec	Ontologie, logique formelle, visualisation	



IV - NoSQL & Graph

interrogation

- Requêtes sur les graphes
  - **Personnes** faisant des **vacations** à **Paris** *et* à **Gif-sur-Yvette**



IV - NoSQL & Graph

solutions

neo4j

Azure Cosmos DB:

OrientDB®

FlockDB

Réseaux

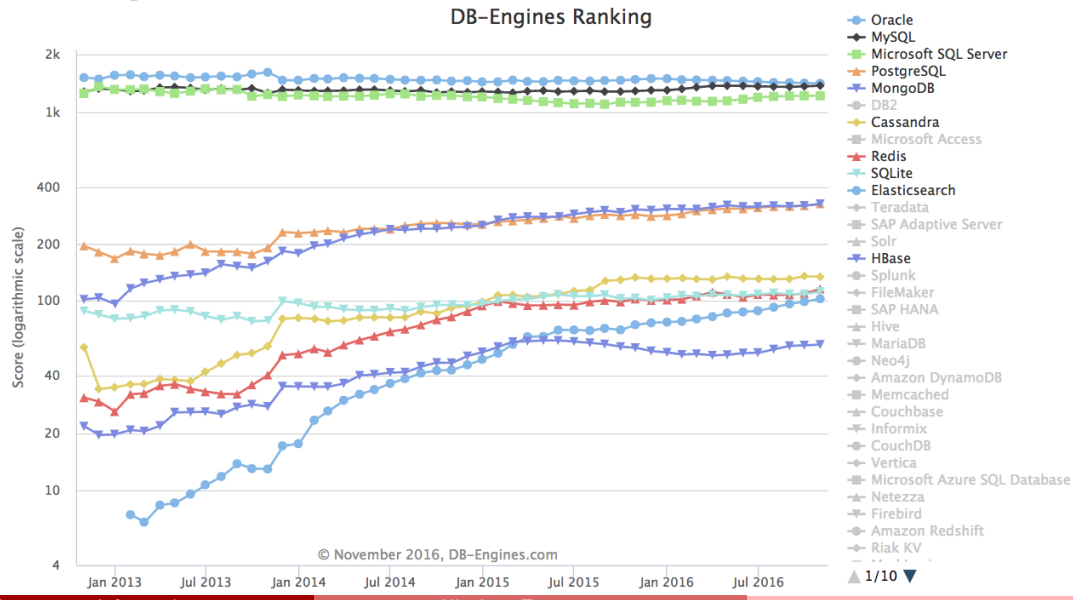
Recommandation

IV - NoSQL & Graph

applications

- Exemples d'applications:
- Calcul sur les graphes sociaux (recommandations, plus courts chemins, atteignabilité,…)
- Calculs sur les réseaux des SIG: réseaux routiers, canalisations, électricité, …
- Web social (linked data)

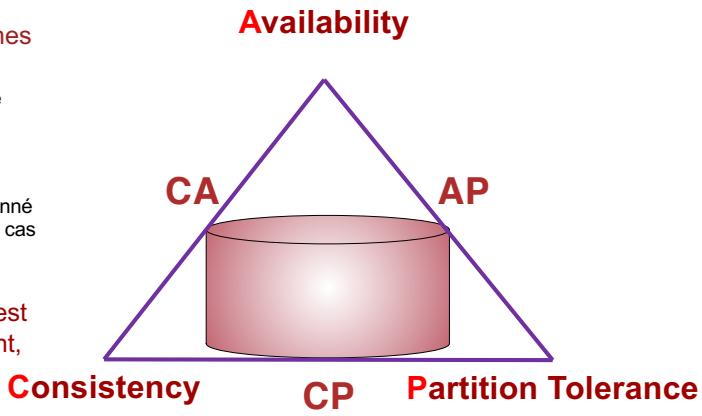
DB-Engines



"Théorème de CAP"

- Théorème de Brewer (2000)
- 3 propriétés fondamentales pour les systèmes distribués
  1. ***Consistency***: Tous les serveurs voient la même donnée (valeur) en même temps (ou ***Cohérence***)
  2. ***Availability***: Si un serveur tombe en panne, les données restent disponibles
  3. ***Partition Tolerance***: Le système même partitionné doit répondre correctement à toute requête (sauf en cas de panne réseau)

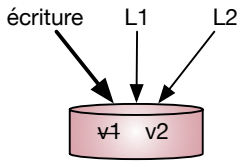
Théorème : "Dans un système distribué, il est impossible que ces **3 propriétés** co-existent, vous devez choisir **2** d'entre elles".



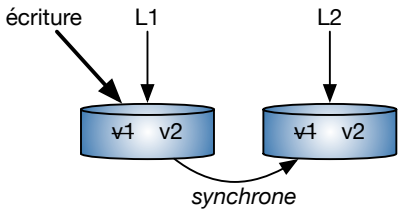
"Théorème de CAP"

illustration

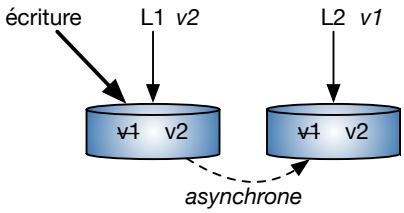
CA
Cohérence + Disponibilité



CP
Cohérence + Distribution

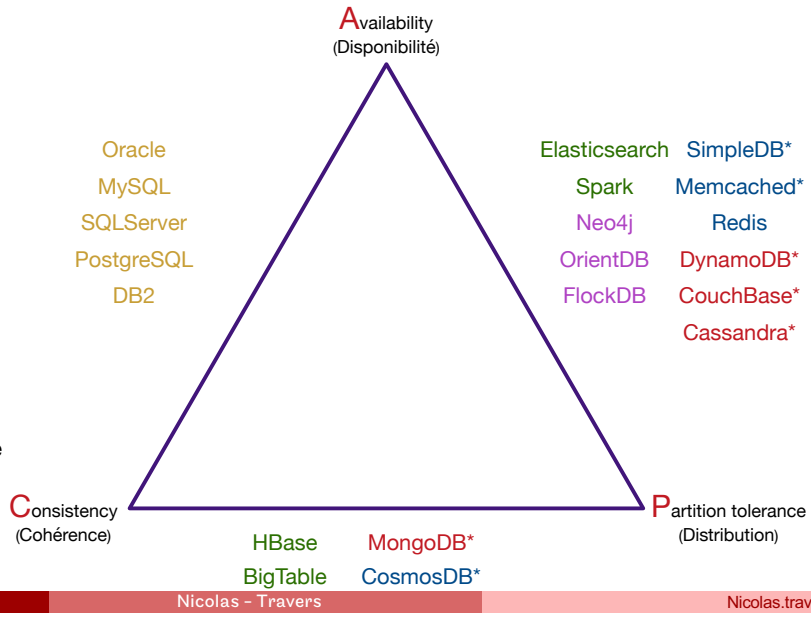


AP
Disponibilité + Distribution



"Théorème de CAP"

Triangle de CAP



Modèle

Relationnel
Orienté Clé-Valeur
Orienté Colonne
Orienté Document
Orienté Graphe

Oracle
MySQL
SQLServer
PostgreSQL
DB2

Elasticsearch
Spark
Neo4j
OrientDB
FlockDB

SimpleDB*
Memcached*
Redis
DynamoDB*
CouchBase*
Cassandra*

HBase
BigTable

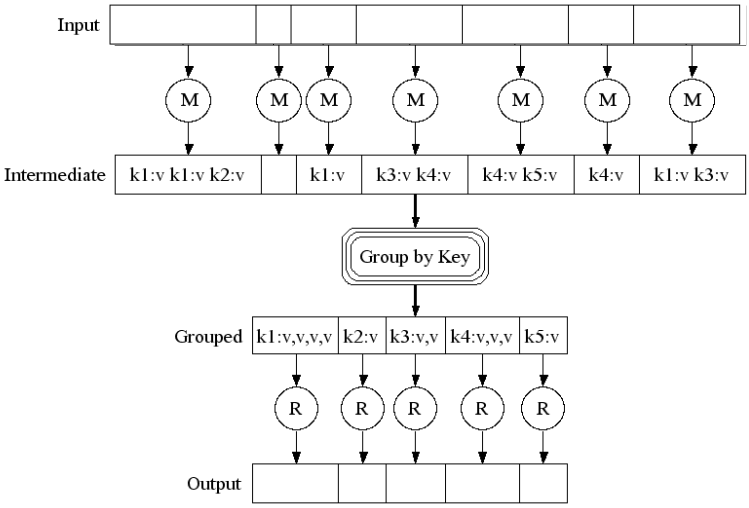
MongoDB*
CosmosDB*

* Possibilité de changer
le mode de cohérence
Cohérence <-> Disponibilité

Map/Reduce (résumé)

- Framework de calcul distribué
- Programme décomposé en 2 fonctions
  - *Map* : transformation de données
    - Entrée : une donnée
    - Sortie : un ensemble de pairs - **clé + valeur**
  - *Reduce* : agrège par clé un ensemble de valeurs
    - *Entrée : liste de valeurs d'une clé* - ***clé + liste(valeurs)***
    - *Sortie : une valeur* - **clé + valeur**
- Passage à l'échelle et Tolérance aux pannes
- *Envoyé à tous les serveurs, appliquée à chaque donnée*
- *Reprise de traitements en cas de panne*

Map/Reduce : Principe



Conclusion

- NoSQL
  - Dédié à un contexte extrêmement distribué
  - Calcul fortement distribué
  - 4 types de calculs complexes

  (clé-valeur, document, colonnes, graphes)
  - Théorème de CAP
- Ne doit pas remplacer automatiquement un SGBD
  - Propriétés ACID
  - Requêtes complexes
  - Performance de jointure