Module MLBDA
Master Informatique
Spécialité DAC

Cours 10 – NoSQL

Systèmes NoSQL (not only SQL)

- **Systèmes qui abandonnent certaines propriétés des SGBDR** (one size does not fit all):
  - Le langage d'interrogation
  - Le contrôle du schéma
  - La concurrence d'accès
- **Pourquoi ?**
  - Passage à l'échelle, flexibilité, besoins induits par les applications (jeux, réseaux sociaux, OLAP…), limitations du relationnel
- **Comment ?**
  - Très forte distribution des données et des traitements : nombreux serveurs, nouveaux traitements (MapReduce, ..)
  - Adaptation élastique à la charge et au volume (clouds)

Constat

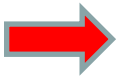
- **Big Data**
  - Forte évolution de la quantité de données produites, de la vitesse de production, de la diversité des données, et de leur usage.
- **Nouveaux besoins**
  - Aide à la décision (OLAP), analyse de données, découverte d'information, …
- **Evolutions du hardware**
  - Infrastructures « passant à l'échelle » peu coûteuses : CPU, mémoires, stockage SSD
  - Cloud : grande infrastructure sans coût initial, économie de maintenance

Changement d'échelle

- **Données**
  - Web2.0 : réseaux sociaux, news, blogs,…
  - Graphes, ontologies
  - Flux de données : capteurs, GPS,…

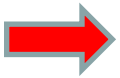
Très gros volumes, données pas ou peu structurées

- **Traitements**
  - Moteurs de recherche
  - Extraction, analyse
  - Recommandation, filtrage collaboratif



Transformation, agrégation, indexation

- **Infrastructures**
  - Clusters, réseaux mobiles, data centers, microprocesseurs multicoeurs



Distribution, redondance, parallélisation

Big Data
(Caractéristiques)

- Volume :
  - quantité et taille, allant de tera à zettabytes
- Variété:
  - données structurées (BD) et non structurées : texte, image, son, multimedia, document…
- Vélocité (vitesse): flux de données, capteurs…
- Véracité : imperfection, qualité des données
- Visualisation, visibilité : représentation des données (graphes..)
- Volatilité : forte évolution des données
- Valeur
- Vie privée
- …

Traitements

- **Besoins des applications**
  - Distribuer les traitements et répartir les données sur de nombreuses machines
    - Éclatement des données sur de nombreux nœuds
    - Traitements analytiques en parallèle sur les nœuds
  - Flexibilité du schéma
  - Flexibilité de la cohérence (transactions ACID trop rigides)
  - Coût
    - Du matériel peu coûteux, facilement remplaçable en cas de panne
- **Les SGBDR ne sont pas adaptés à la gestion des Big Data**

Evolution des capacités

- **Evolutions du hardware**
  - Infrastructures « passant à l'échelle » peu coûteuses : CPU, mémoires, stockage SSD
  - Cloud : grande infrastructure sans coût initial, économie de maintenance
- **Evolution des traitements**
  - Utilisation du parallélisme : Map Reduce,..
  - Machine learning

Challenge

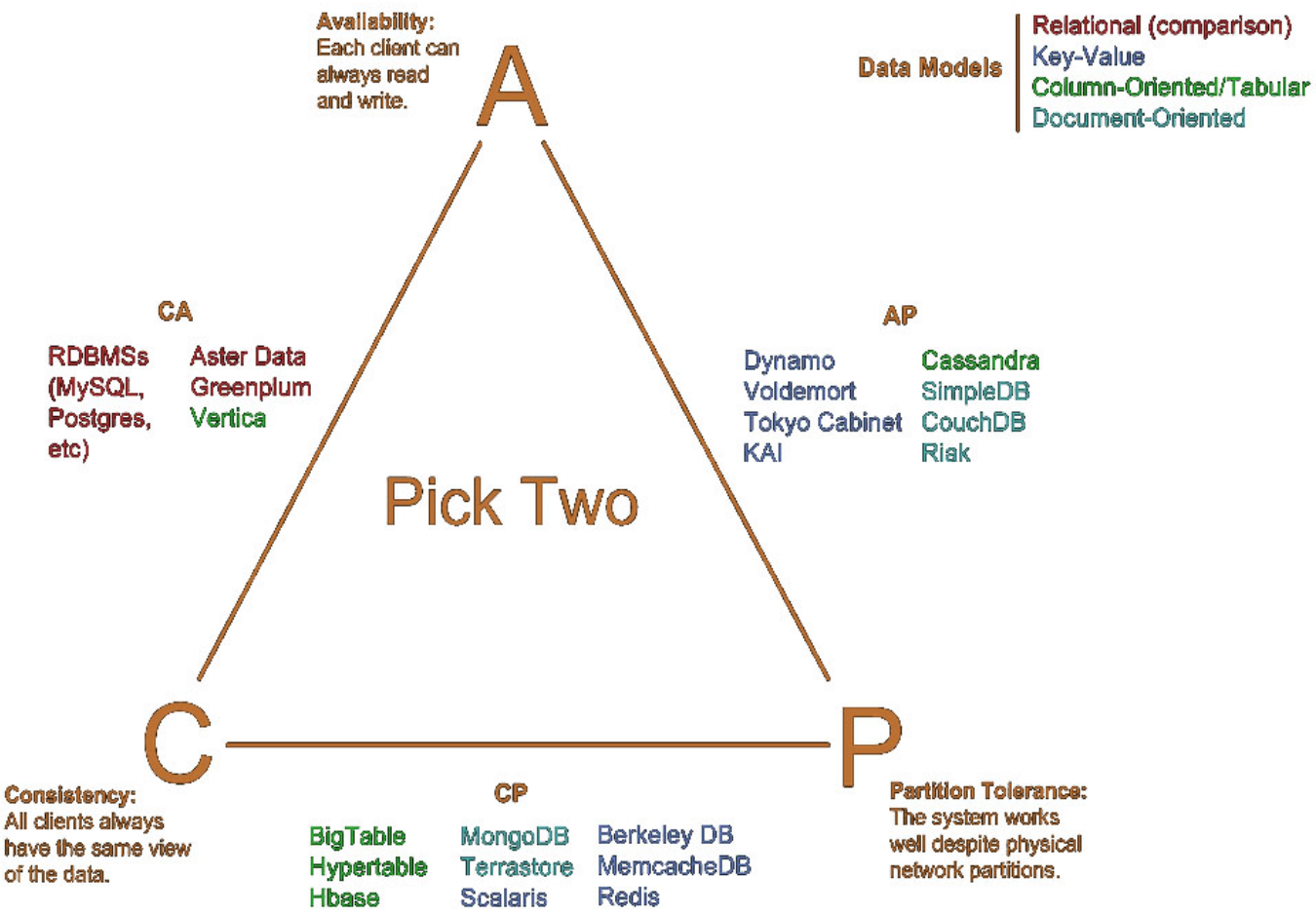
- Pouvoir gérer
  - efficacement
  - de très larges volumes de données
  - complexes et hétérogènes, distribuées, fragmentées, répliquées…
- Permettre des
  - traitements variés
  - avec des exigences diverses (qualité, disponibilité, performance…),

Nécessité de distribution

Théorème CAP (Brewer)

- Aucun système distribué ne peut garantir simultanément plus de deux des trois propriétés suivantes :
  - **Consistency** (cohérence) : un service est exécuté entièrement ou pas du tout (propriétés ACID). Tous les nœuds voient les mêmes données en même temps.
  - **Availability** (disponibilité) : le service est toujours accessible, même en cas de panne d'un nœud.
  - **Partition tolerance** : aucune panne autre que la rupture totale du réseau ne doit empêcher le système de fonctionner.
- Toutes ces propriétés sont souhaitables, mais il n'est pas possible de les avoir simultanément.

Visual Guide to NoSQL Systems

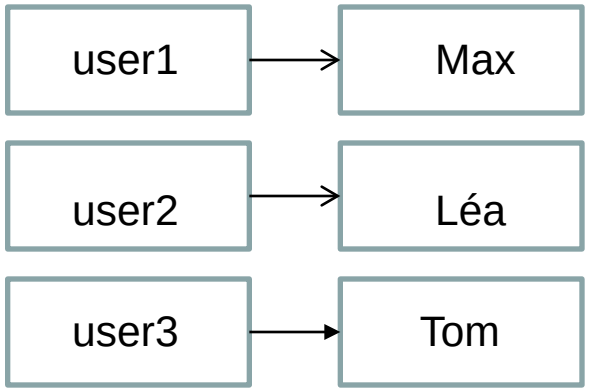


Systèmes NoSQL

- Abandon des contraintes fortes des SGBDR (propriétés ACID)
- Modèle de données plus flexible, plus intuitif (le plus simple étant clé-valeur)
- Exploitation de ce modèle pour distribuer plus facilement et plus efficacement (stockage et indexation, partitionnement, parallélisme)
- Langages spécialisés, API simples
- Matériel peu coûteux
- **4 catégories** :
  - Stockage de couples clé-valeur
  - Documents : JSON, XML
  - Bases orientées colonnes
  - Bases de graphes

Clef-valeur

- La base est une table de hachage distribuée, et chaque objet est identifié par une clef unique, qui est la seule façon de l'interroger.

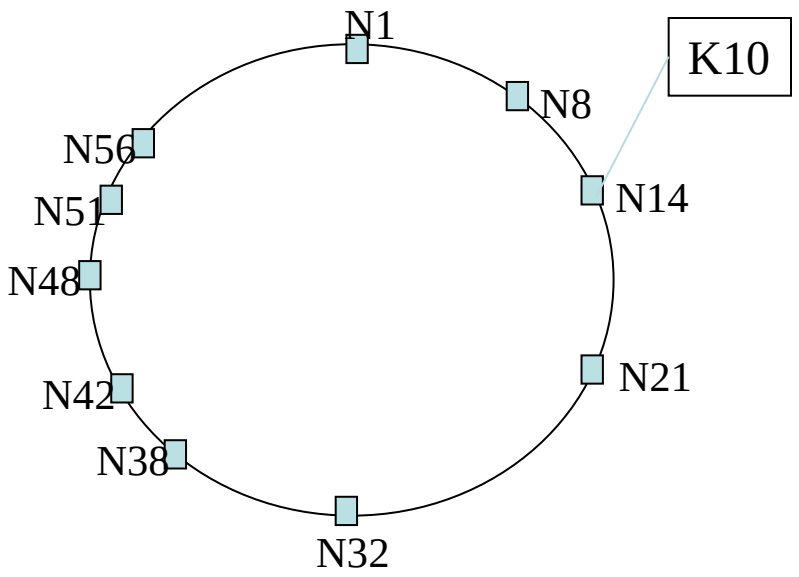


- Exemples : S3, Amazon EC2, Voldemort, DynamoDB, Riak, Redis …

Clef-valeur : stockage

- Les données sont fortement distribuées et la base est une table de hachage distribuée (consistent hashing)
  - Les nœuds sont répartis sur un anneau
  - Les ressources sont réparties uniformément sur les différents nœuds de l'anneau

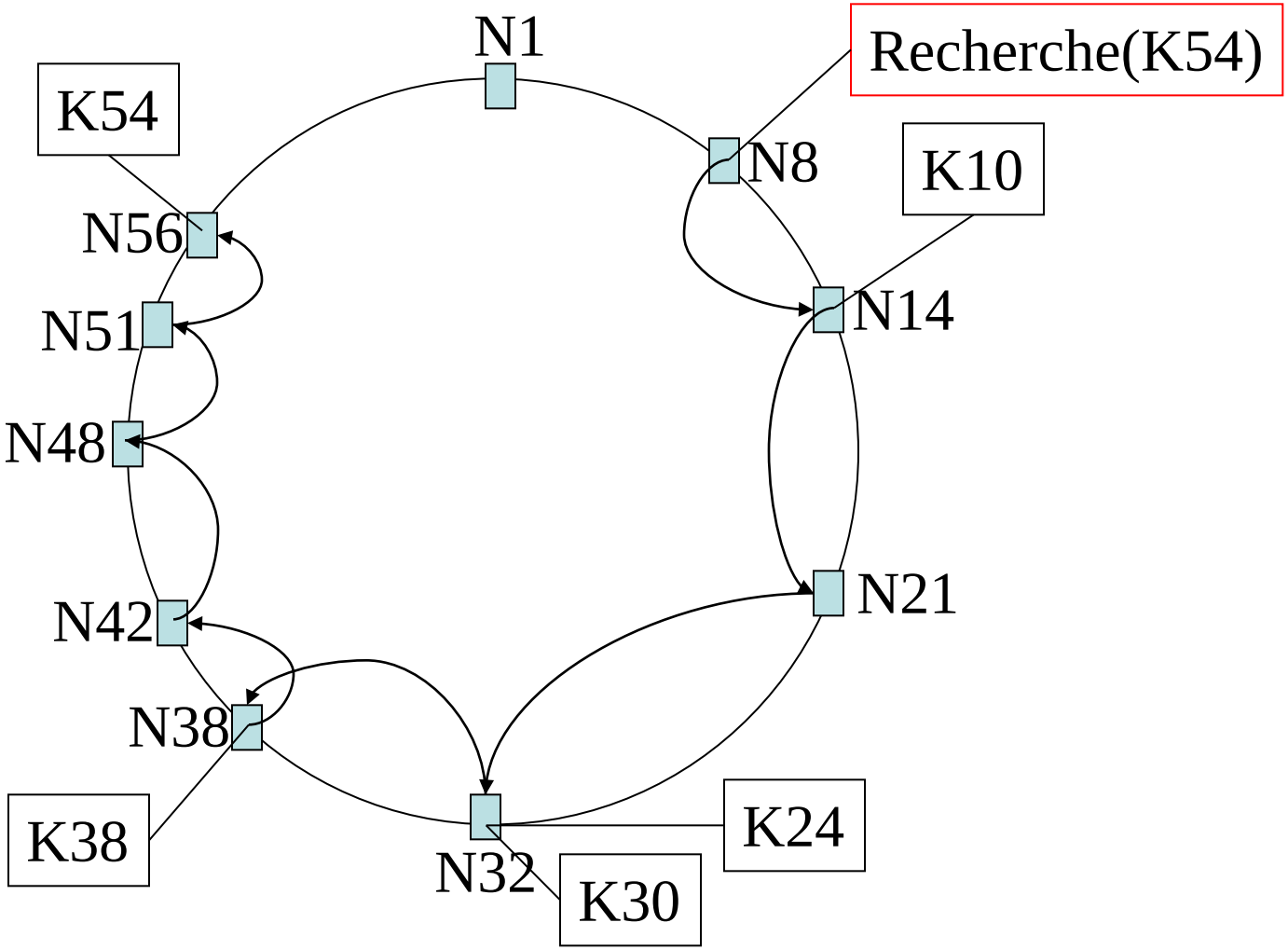
Chaque nœud est alloué sur
l'anneau en fonction de hash(IP)
Hash(ressource)=k
k placé sur le nœud
immédiatement supérieur à k.
Recherche par propagation



Recherche (naïve) d’une ressource

- Sur le nœud i on reçoit la requête : recherche(k)
- Si i possède k, il retourne k
- Sinon, il propage la requête à son successeur (chaque nœud doit stocker l’identification de son successeur)
- Le résultat suit le chemin dans l’ordre inverse
- Recherche linéaire en nombre de nœuds

Exemple de recherche naïve



Amélioration de la recherche

- Avoir une table de routage plus complète
- Pour chaque nœud i :
  - Succ[k]=premier nœud sur l'anneau qui vérifie
    $(i + 2^{k-1}) \bmod 2^m$, $1 <= k <= m$
  - Successeur=succ[1]
  - m entrées dans la table
  - Prédecesseur (utilisé pour la maintenance dynamique du réseau)
  - Donne adresse IP et No de port du nœud

Exemple de recherche

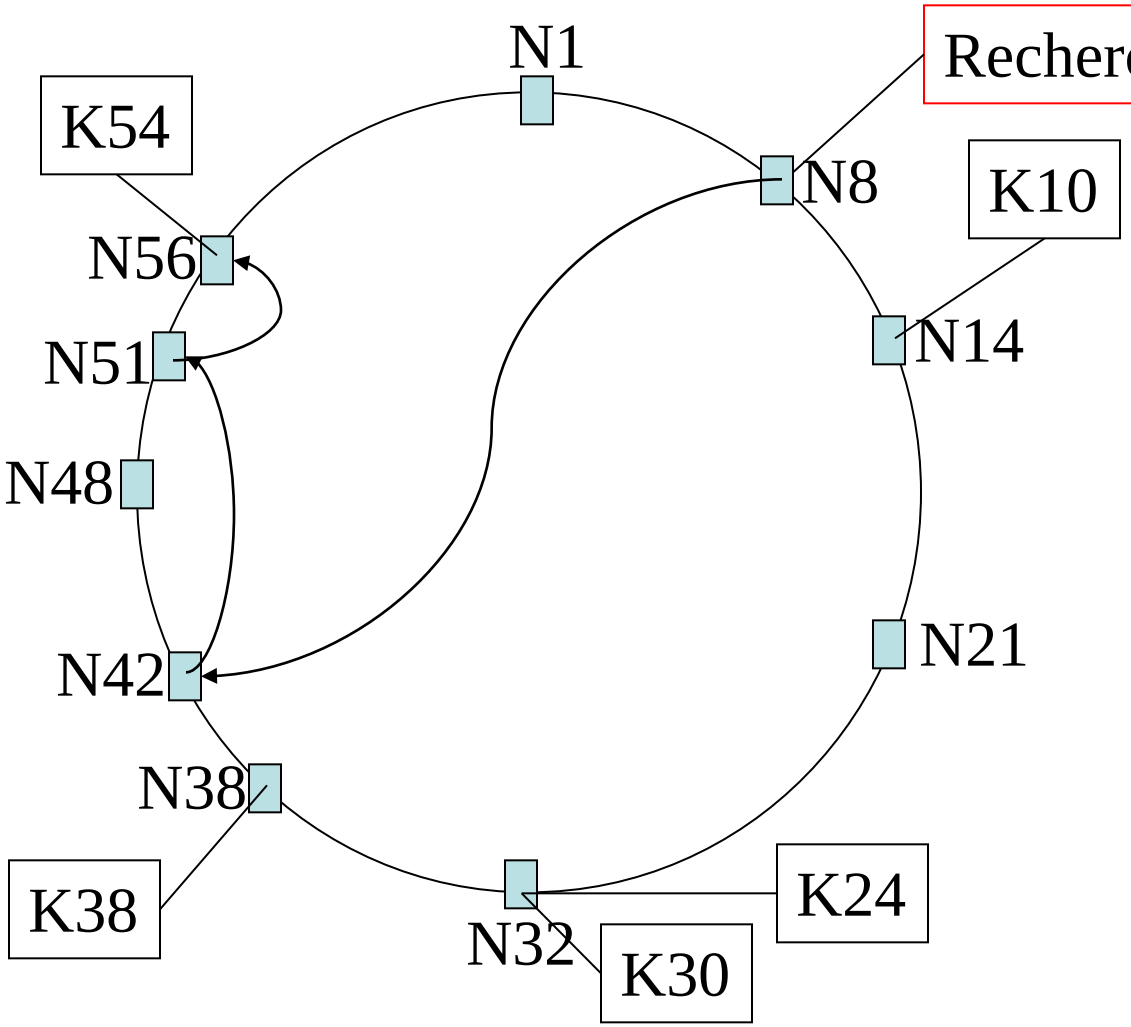


Table routage N8

N8+1	N14
N8+2	N14
N8+4	N14
N8+8	N21
N8+16	N32
N8+32	N42

Clef-valeur : opérations

- **Create** : crée un nouveau couple (clef,valeur). La valeur est n'importe quel objet
- **Read** : lit un objet à partir de sa clef
- **Update** : met à jour la valeur d'un objet à partir de sa clef
- **Delete** : supprime un objet à partir de sa clef

- Simple interface de requêtage HTTP accessible depuis n'importe quel LP.
- On ne peut pas effectuer de requête sur le contenu des objets stockés.

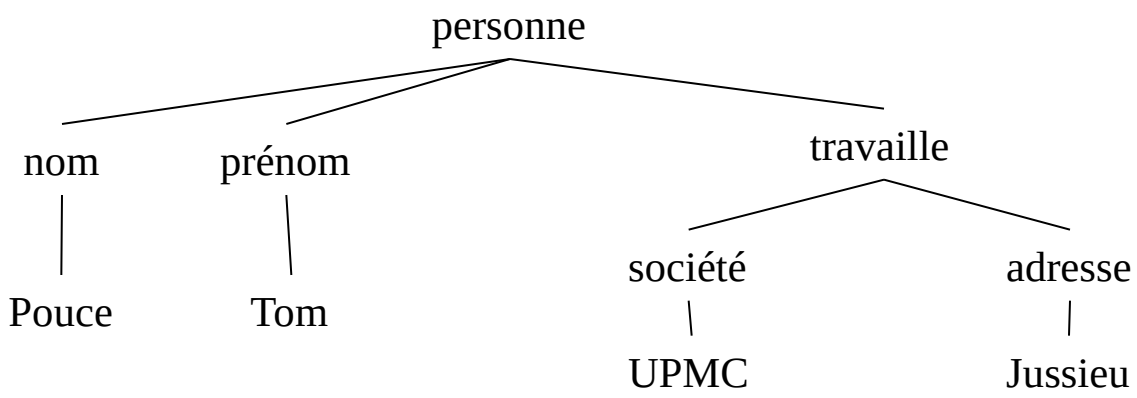
Clef-valeur

+ Modèle de données simple
+ Passage à l'échelle : évolutivité, disponibilité

- Interrogation à partir de la clef seulement
- Pas de traitement de données complexes

Documents

- On stocke des documents (XML, JSON) enregistrés dans des collections.
- Un document a une structure arborescente : liste de champs ayant une valeur pouvant être une liste de champs…
- Pas de schéma : grande flexibilité



- Ex : MongoDB, CouchDB…

Document : modèle

- Un document est composé de champs et de valeurs associées
- Le contenu se décrit avec des listes, des enregistrements imbriqués, des ensembles (structure riche)
- Les données peuvent être typées et/ou structurées, ou non.
- Formats de documents : XML ou JSON, modèles semi-structurés qui permettent de stocker n'importe quel objet, grâce à la sérialisation.

Exemple avec JSON

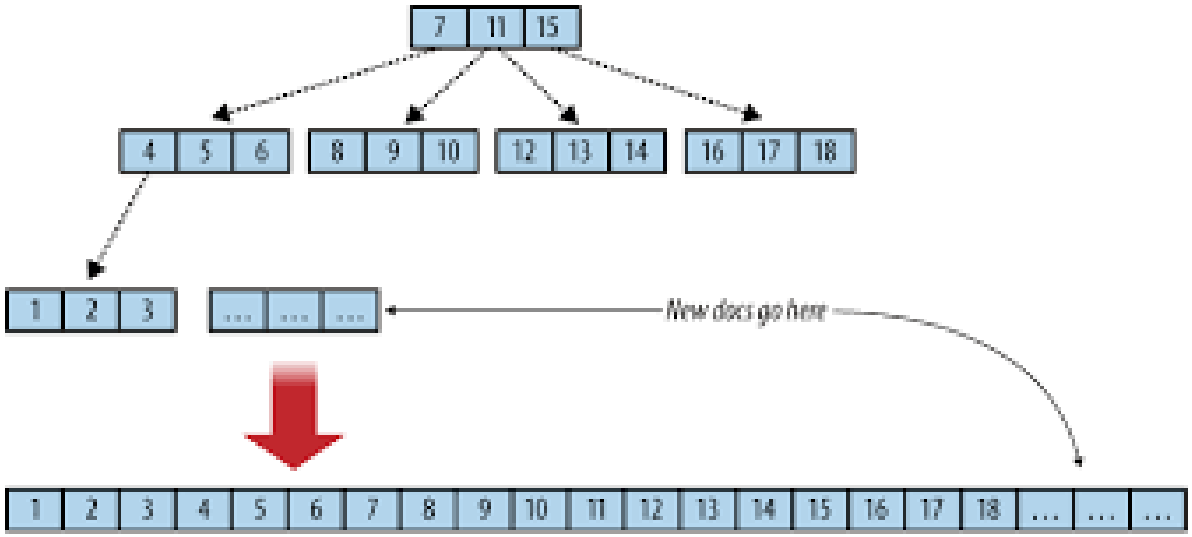
```
{nom: "Alan", tél: 2157786,  email: "agb@abc.com"}
```

Extension naturelle : les valeurs sont elles-mêmes d'autres structures.

```
{nom: {prénom: "Alan",  famille: "Black"},
tél: 2157786,
email: "agb@abc.com"}
```

Document : stockage

- Index : B-tree, structure de hachage



Document

+ Modèle de données simple et puissant
+ Passage à l'échelle
+ Maintenance faible
+ Forte expressivité du langage de requêtes : requêtes complexes sur structures imbriquées.

- Efficace pour les interrogations par clef mais peut être limité pour les interrogations par le contenu des documents, limité aux données hiérarchiques

Colonnes

- Les données sont stockées par colonne, dans des tables (relation universelle).
- Chaque colonne est définie par un couple (clef, valeur).
- Les colonnes peuvent être groupées en supercolonnes et en famille de colonnes
- Chaque ligne est identifiée par un identifiant unique
- Les colonnes sont regroupées par ligne

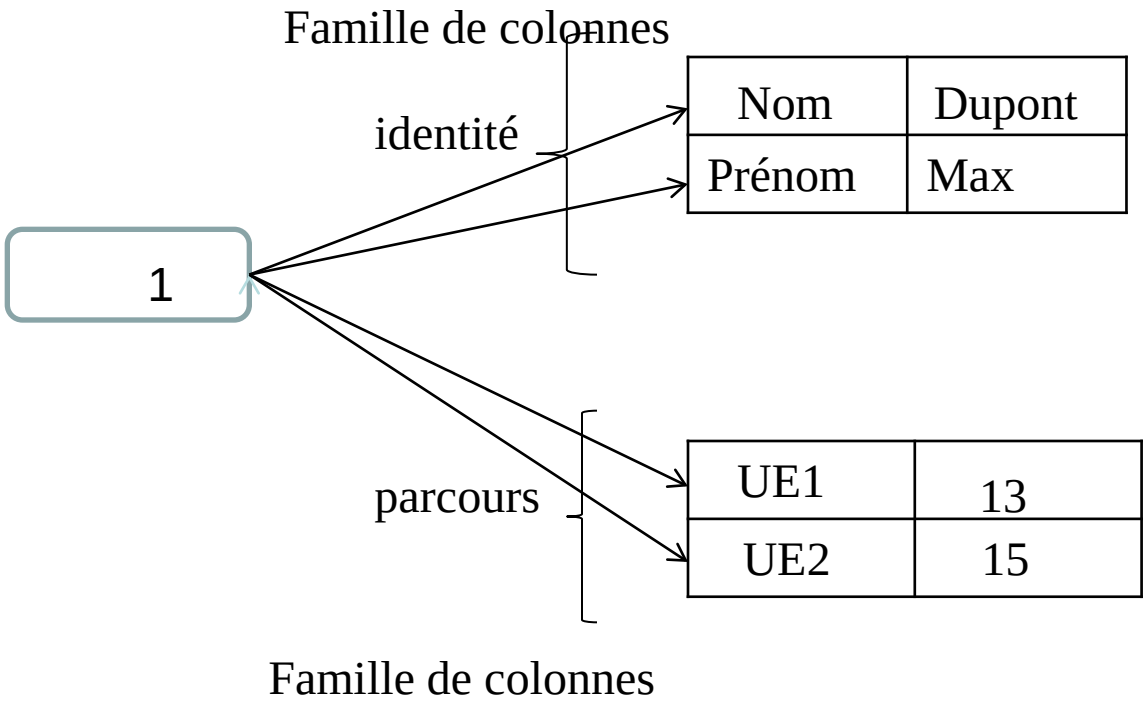
Clé	**Nom**
1	Dupont
2	Durand
3	Martin

Clé	**Prénom**
1	Max
2	Léa
3	Tom

Clé	**parcours**
1	UE1, UE2
2	UE2
3	UE1

- Ex : MonetDB, BigTable, Cassandra, Hadoop/Hbase…

Exemple de base colonne



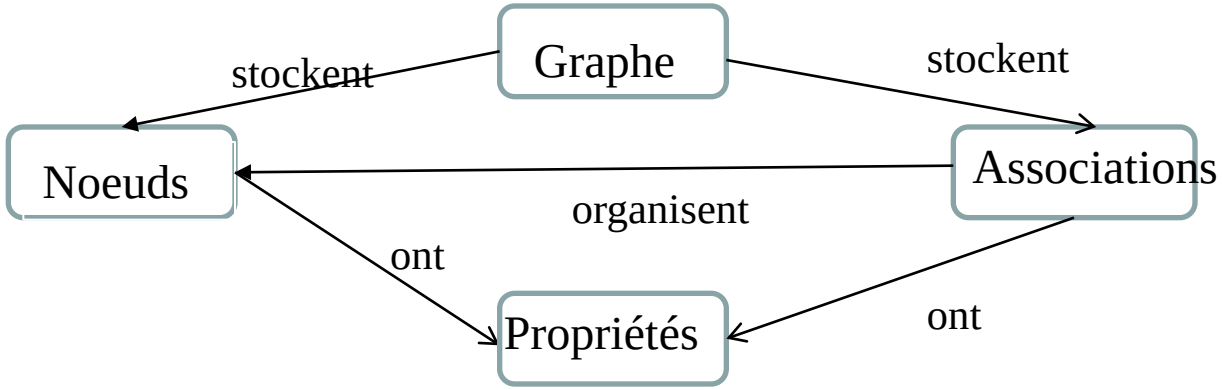
Chaque famille de colonnes peut être traitée comme une table séparée (partitionnement)

Bases colonne

- Permet d’avoir un très grand nombre de valeurs sur une même ligne et de stocker des relations 1:N.
- Ajout facile de colonnes, facilité de partitionnement, de compression, d’indexation.
- Modèle efficace avec indexation sur les colonnes
- Bien adapté à l’OLAP
- Ne supporte pas les données structurées complexes
- Les modifications de structures en colonne nécessitent de la maintenance.

Graphes

- Stockage des nœuds, des arcs et de propriétés (triplets sujet, prédicat, objet)
- Bien adapté à la manipulation d'objets complexes organisés en réseaux : réseaux sociaux, cartographie, web sémantique…



- Ex : Néo4j, Titan, DB2 RDF…

Graphes

- On utilise le principe (clef, valeur) pour accéder rapidement d'un objet aux couples (prédicat, sujet) qui le concernent, d'un prédicat aux couples (objet, sujet), d'un sujet aux couples (objet, prédicat).

- On utilise le langage SPARQL pour interroger

Graphes

- Modèle riche et évolutif (ajout de sujets, d'objets et de prédicats) bien adapté aux situations où il faut modéliser beaucoup de relations,
- Un langage d'interrogation généraliste (SPARQL) et des langages spécialisés
- La répartition des données peut être problématique pour des masses de triplets

Bases NoSQL et partitionnement

- Assurer les propriétés A (availability) et P (partition tolerance)
- Forte distribution des données avec des techniques de partitionnement éprouvées (consistent hashing, sharding..)
- Paradigme de calcul parallèle adapté aux masses de données: MapReduce
  - Conçu par Google, mais basé sur des modèles connus de programmation parallèle

MapReduce

- Les données sont réparties sur les serveurs.
- MapReduce permet de répartir la charge sur les serveurs (plus le nombre de serveurs est important, plus les traitements sont rapides).
- Gère le cluster, la répartition de la charge, la distribution des données, la tolérance aux pannes
- Est disponible dans de nombreux environnements de programmation

MapReduce

- Le travail du programmeur pour un calcul (un programme complexe peut nécessiter plusieurs calculs) se limite à l'écriture de deux fonctions
  - Map : fait des calculs élémentaires sur des paires (clef,valeur) et retourne une liste de résultats intermédiaires
    - En entrée : ensemble de paires (clef, valeur)
    - En sortie : liste intermédiaire de (clef1, valeur1)
  - Reduce : combine les listes de résultats intermédiaires en une liste de résultats finaux.
    - En entrée : liste intermédiaire de (clef1, valeur1)
    - En sortie : liste de (clef1, valeur2)

Exemple

- Calcul du nombre d'occurrences des mots dans les documents du Web
  - Fonction Map :
    - entrée : identifiant d'un document, contenu d'un document
    - sortie : liste des mots avec leur nombre d'occurrence
  - Sur un nœud Map, on produit une liste

    `((modèle : 2), …(données : 3), .. (base : 5))`

    pour chacun des documents du nœud.
  - Les listes reçues des différents nœuds sont regroupées et triées par mot pour produire des listes

    `((base, 1, 5, 2, 2,…), .., (données, 3, 4, 2,..), …, (modèle, 1, 5, 2, ..),..)`
  - Fonction Reduce :
    - Entrée : mot, liste des nb d'occurrence du mot dans les documents
    - Sortie : liste des mots avec leur nombre d'occurrence total

MapReduce

- Fonctionnement :
  - la distribution des programmes Map et des données est gérée par l'environnement,
  - les noeuds exécutent les Map,
  - l'environnement gère le groupement, le tri, la distribution des listes résultats et des programmes Reduce,
  - les nœuds exécutent les programmes Reduce
  - l'environnement gère la collecte des résultats des Reduce.
  - tout est géré avec des systèmes (clef,valeur) répartis
- Applications
  - les calculs de score en recherche d'information,
  - les calculs de règles d'association (panier de la ménagère)
  - la détection de spam, les réseaux sociaux, la fouille du web, ...

SQL vs NoSQL

- Cohérence forte
  - Logique : schémas, contraintes
  - Physique : transactions ACID
- Distribution des **données**
  - Transactions distribuées
- Ressources **limitées**
  - Optimisation de requêtes
- Langage **standard** : SQL

- Cohérence faible
  - ~~schémas, contraintes~~
  - Cohérence « à terme »
- Distribution des **traitements**
  - Traitements batch
  - mapReduce
- Ressources « **illimitées** »
  - Passage à l'échelle horizontal
- Langages **spécialisés**, API

Conclusion

- 4 caractéristiques de NoSQL :
  - schéma flexible, API simple, relâchement des propriétés ACID et passage à l'échelle
- 4 catégories :
  - Clef-valeur, document, colonnes, graphes
- Les aspects à retenir :
  - modèle, stockage, schéma de partitionnement, sémantique des transactions
- SGBD de la nouvelle génération : non-relationnel, répartis, open-source, passant à l'échelle.
  - Au départ (2009) : développer des SGBD modernes passant à l'échelle du Web.
  - Développement important depuis 2009 : pas de schéma, support pour la réplication, API simples, éventuellement la cohérence (mais pas ACID)
- Approche complémentaire de SQL et du relationnel (approfondissement en M2)

Bibliographie

- Un lien intéressant : http://nosql-database.org/ qui donne une liste classée de toutes les BD NoSQL existantes.
- A. Foucret : Livre blanc sur NoSQL (http://www.smile.fr/Livres-blancs/Culture-du-web/NoSQL)
- NoSQL Data Stores in Internet-scale Computing (tutoriel de Wei Tan, IBM, à la conférence ICWS) : http://researcher.watson.ibm.com/researcher/files/us-wtan/NoSQL-Tutorial-ICWS-2014-public-WeiTan.pdf

La suite

- M1 S2 : BDR (Bases de données réparties)
  - Méthodes d'accès : index, arbres B+, hachage
  - Optimisation de requêtes
  - Algorithmes de jointures
  - Conception et interrogation de BD réparties
  - Transactions réparties
  - SGBD parallèles
- M2 S1 : BDLE (Bases de données à large échelle)
  - Bases de données multidimensionnelles
  - Map Reduce
  - Réplication et transactions à large échelle
  - Grand graphes de données
- M2SI : ASWS (Apprentissage Symbolique et Web sémantique)
  - Web sémantique : RDF, Sparql, OWL