BIG DATA

TSOPZE N.
norbert.tsopze@facsciences-uy1.cm

Définition

- Quand la capacité de traitement dépasse celle des SGBD conventionnels
- Passage à l'échelle des méthodes existantes
  - Grandes masses de données
  - Changement rapide
  - Aucune structure adaptée à l'architecture de la base de données

règle des 3V (5V)

Explosion des données dans le domaine numérique

- un grand **Volume** de données,
- une importante **Variété** de ces mêmes données et
- une **Vitesse** de traitement s'apparentant parfois à du temps réel.
- **Valeur**, vision économique
- **Véracité**, aspect qualitatif

Sources de données

- dispositifs mobiles interconnectés (téléphones, ordinateurs,...)
- Accès aux réseaux sociaux
- Sites de commerce électronique
- Apparition des nouvelles technologies

Importances

- Analyse et extraction des connaissances : données sur les clients, données géographiques,...
- Proposition des nouvelles services : potentiels clients,...

Outils, techniques et stratégies

Limites de SGBD relationnels

- La propriété d'**A**tomicité assure qu'une transaction se fait au complet ou pas du tout
- La propriété de **C**ohérence assure que chaque transaction amènera le système d'un état valide à un autre état
- La propriété d'**I**solation assure que l'exécution simultanée de transactions produit le même état que celui qui serait obtenu par l'exécution en série des transactions. Chaque transaction doit s'exécuter en isolation totale
- La propriété de **D**urabilité assure que lorsqu'une transaction a été confirmée, elle demeure enregistrée même à la suite d'une panne d'électricité, d'une panne de l'ordinateur ou d'un autre problème.

NoSQL

Dans un contexte centralisé, les contraintes **ACID** sont plutôt faciles à garantir. Dans le cas de systèmes distribués, il est en revanche nécessaire de distribuer les traitements de données entre différents serveurs.
Il devient alors difficile de maintenir les contraintes ACID à l'échelle du système distribué entier tout en maintenant des performances correctes.

Comme on le verra, la plupart des SGBD de la mouvance NoSQL ont donc été construits en ne tenant pas compte des contraintes ACID, quitte à ne pas proposer de fonctionnalités transactionnelles.

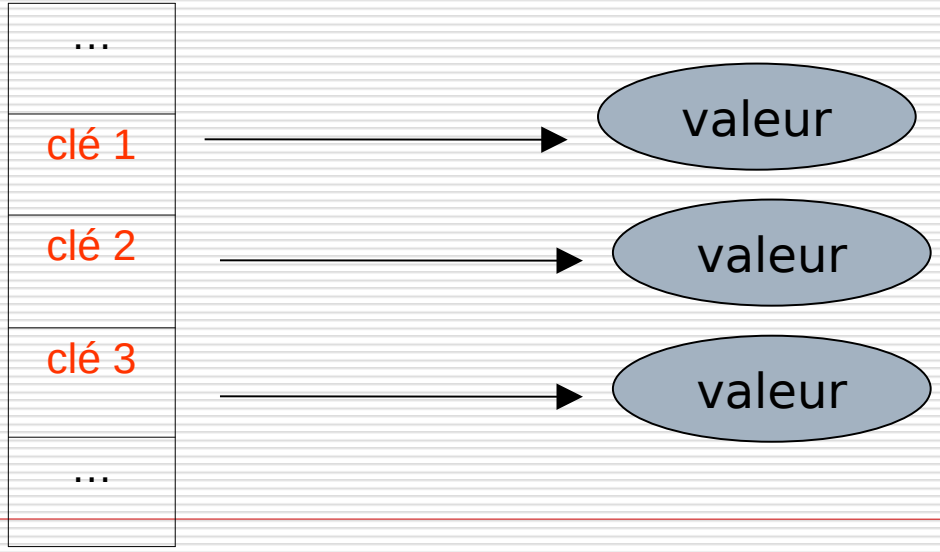
NoSQL

NoSQL = “**Not Only SQL**” et non “No SQL”.
quatre grandes catégories :

- **Paradigme clé / valeur,**
- **Bases orientées colonnes,**
- **Bases documentaires,**
- **Bases orientées graphes**.

Paradigme clé / valeur

- chaque objet est identifié par **une clé unique** qui constitue la seule manière de le requêter.
- La structure de l'objet est libre et le plus souvent laissé à la charge du développeur de l'application (*XML*, *JSON*, ...), la base ne gérant, généralement que des chaînes d'octets.



JSON

- La vitesse de traitement.
- La simplicité de mise en oeuvre.
- On n'a *pas besoin de parser un fichier XML* pour extraire des informations à travers le net, car JSON est reconnu nativement par JavaScript.
- Les contenus binaires peuvent être intégré et échangés sur le net avec une représentation textuelle spéciale avec une commande comme: new Buffer(file).toString('base64').

JSON (Exemple)

```
{
 "menu": "Fichier",
 "commandes": [
    {
       "title": "Nouveau",
       "action":"CreateDoc"
    },
    {
       "title": "Ouvrir",
       "action": "OpenDoc"
    },
    {
       "title": "Fermer",
       "action": "CloseDoc"
    }
  ]
}
```

XML

- XML est extensible quand au langage, on peut créer des formats comme RSS (*format* d'information et d'échange sur Internet) , SVG (scalable vector graphic).
- Il est largement utilisé et reconnu par tous les langages de programmation.
- Il est plus facile à lire et convient mieux pour les fichiers destinés aux non programmeurs.
- XML aussi bien que JSON ne conviennent pas pour stocker directement des données binaires de taille importante.

XML

```
<?xml version="1.0" ?>
<root>
 <menu>Fichier</menu>
 <commands>
   <item>
      <title>Nouveau</value>
      <action>CreateDoc</action>
   </item>
   <item>
      <title>Ouvrir</value>
      <action>OpenDoc</action>
   </item>
   <item>
      <title>Fermer</value>
      <action>CloseDoc</action>
   </item>
 </commands>
</root>
```

Paradigme clé / valeur

quatre opérations **C**reate **R**ead **U**pdate et **D**elete (**CRUD**):

- **Create** : créer un nouvel objet avec sa clé → create(key, value)
- **Read** : lit un objet à partir de sa clé → read(key)
- **Update** : met à jour la valeur d'un objet à partir de sa clé → update(key, value)
- **Delete**: supprime un objet à partir de sa clé → delete(key)

Les bases de ce type disposent pour la plupart d'une interface **HTTP REST (REpresentational State Transfer:** un mode d'architecture pour les systèmes hypermédia distribués) permettant de procéder très simplement à des requêtes, et ceci depuis n'importe quel langage de développement.

Bases documentaires

- Collections de documents.
  - champs
  - valeurs associées, pouvant être requêtées.
- Type de valeurs
  - simple (entier, chaine de caractère, date, ...),
  - composées de plusieurs couples clé/valeur.
- Ces bases sont dites “*schemaless*”. A ce titre il n’est pas nécessaire de définir au préalable les champs utilisés dans un document. Les documents peuvent être très hétérogènes au sein de la base.
- Possibilité d’effectuer des requêtes sur le contenu des objets.

Bases documentaires

- On conserve généralement une interface d'accès HTTP REST permettant d'effectuer simplement des requêtes sur la base mais celle-ci est plus complexe que l'interface CRUD des bases clés/valeurs. Les formats de données JSON et XML sont le plus souvent utilisés afin d'assurer le transfert des données sérialisées entre l'application et la base.

- Du fait de leur conception plus simple que les bases de données relationnelles, ces bases conservent des performances élevées. Elles disposent par ailleurs de fonctionnalités très intéressantes en terme de flexibilité du modèle documentaire, ce que les bases relationnelles ne peuvent offrir.

Bases orientées colonnes

- Les *bases orientés colonnes* sont résolument les plus complexes à appréhender parmi la mouvance NoSQL. Bien que l'on obtienne au final un schéma relativement proche des bases documentaires, l'organisation sous-jacente des données permet à ces bases d'exceller dans les traitements d'analyse de données et dans les traitements massifs.
- Les concepts essentiels sont les suivants :
- **Colonne** : il s'agit de l'entité de base représentant un champ de donnée. Chaque colonne est définie par un **couple clé / valeur**.
- Une colonne contenant d'autres colonnes est nommée **super-colonne**.

Paradigme graphe

- Ce paradigme est le moins connu de ceux de la mouvance NoSQL. Ce modèle s'appuie principalement sur deux concepts : d'une part l'utilisation d'un moteur de stockage pour les objets (qui se présentent sous la forme d'une **base documentaire**, chaque entité de cette base étant nommée **noeud**).
- D'autre part, à ce modèle, vient s'ajouter un mécanisme permettant de décrire les arcs (relations entre les objets), ceux-ci étant orientés et disposant de propriétés (nom, date, ...).
- Par essence ces bases de données sont nettement plus efficaces que leur pendant relationnel pour traiter les problématiques liées aux réseaux (cartographie, relations entre personnes,...). En effet, lorsqu'on utilise le modèle relationnel, cela nécessite un grand nombre d'opérations complexes pour obtenir des résultats.

Modélisation

scalabilité

- Explosion de la taille des données à gérer ➔ temps de réponse satisfaisant.
- Deux solutions sont envisageables :
  - scalabilité verticale: Augmenter les ressources de la machine; augmenter à l'infini.
  - scalabilité horizontale: ajouter d'autres machines au cluster.

Dénormalisation

- dupliquer certaines données de la table B dans la table A afin d'optimiser les requêtes qui pourront se contenter d'interroger la table A sans avoir à faire la jointure entre A et B
- Intégrer dans les systèmes de gestion NoSQL

schéma de données orienté document

- Collection= ensemble de documents
- Stockage possible des documents de nature différents au sein d'une collection
- intérêt des collections :
  - simplifier la compréhension de la base
  - accélérer les recherches;
  - ajouter des indexes sur les documents et sur les champs. Regrouper les documents par collections rend l'indexation plus efficace.

document

- objets ayant des propriétés (sans méthodes)
- Possibilité d'imbriquer
- Exemple
  - Clé/Valeur : {"texte" : "Hello, world!"}

«texte» est la Clé et «Hello, world!» est la donnée.

  - plusieurs Clé/Valeur :

{"texte" : "Hello, world!", "numero" : 2}

Modélisation - documents

- Trois approches:
  - Imbrication de documents
  - Duplication d’un sous ensemble de champs
  - Duplication de la clé primaire

Imbrication de documents

- avoir le document B imbriqué dans le document A.
- Il existe deux cas d'utilisation pour cette stratégie :
  - toutes les requêtes concernant le document parent concerneront aussi les données du document imbriqué.
    Exemple : imbrication de l'adresse dans le document de personne.
  - On peut modifier les documents parents et enfants d'une manière atomique. Cependant, l'imbrication de document n'est pas possible si on veut créer le document enfant avant le document parent.

Duplication d’un sous ensemble de champs

- dupliquer le minimum de données du document B dans A et de préférence, celles qui ne changent pas ou pas souvent, car lors de leur mise à jour, nous devrions faire un update sur l’ensemble des documents qui les contiennent, plutôt qu’à un seul endroit dans une base de données normalisée.
- Augmentation du temps d’écriture.

Duplication de la clé primaire

- duplique que la clé primaire du document B dans le document A. Pour récupérer les données, l'application doit faire deux requêtes : une première requête pour récupérer la clé primaire, puis une seconde pour récupérer les données de l'autre coté de la relation.
- Dans une relation N-N entre A et B, on mets les clefs des éléments de B dans un champ de A et vis-versa.