UNE EVOLUTION NATURELLE VERS LE BIG DATA

Corentin Burnay - BSTORM

CONSULTING BSTORM

PARTIE CONCEPTUELLE

CONSULTING BSTORM

PARTIE CONCEPTUELLE

- Plan
  - Des SGBD relationnels traditionnels
  - Utilisation OLTP et SGBD relationnels
  - Limites d'une utilisation OLTP : vers une utilisation OLAP
  - Evolution des structures de données
    - Modèles OLAP en étoile et en flocon
      - Modèle en étoile Vs. modèle en flocon
  - Business intelligence (BI)
    - Utilisation du modèle OLAP en BI
    - Data cube
    - OLTP Vs. OLAP
    - Architecture ETL
    - Data Mining
    - « What-if » Analysis

PARTIE CONCEPTUELLE

- BI Vs. BIG DATA
- Données non structurées
- BIG DATA
  - Les 3V du BIG DATA
    - Volume
      - Volume & solution Apache Hadoop
      - Hadoop & pattern MapReduce
    - Vélocité
    - Variété
  - Exemple contextuels

DES SGBD RELATIONNELS TRADITIONNELS

- Quels sont les premiers défis des SGBD?
  - *Stocker* et *questionner* des données
    - Organiser de grandes quantités de données afin d'en optimiser la conservation et la restitution
- A partir des années 1980, le *modèle relationnel* s'imposa comme le standard de structuration des données

- *Mémento* : *Un système de gestion de base de données employant un schéma relationnel est appelé « SGBDR »*

DES SGBD RELATIONNELS TRADITIONNELS

- En quoi consiste réellement le *modèle relationnel*?
  - Le *modèle relationnel* conçu par Edgar F. Codd repose principalement sur une *algèbre relationnelle*
  - Il permet la modélisation des relations existantes entre plusieurs informations et la structuration entre elles
    - Cette modélisation est ensuite retranscrite physiquement ou implémentée dans une base de données relationnelle

- *<u>Mémento</u> : L '« algèbre relationnelle » est une théorie mathématique proche de la théorie des ensembles qui définit des opérations pouvant être effectuées sur des relations*

UTILISATION OLTP ET SGBD RELATIONNELS

- Le *modèle relationnel* est performant pour une utilisation strictement transactionnelle d'une base de données
  - ***Exemple*** : Un client passe commande auprès d'une entreprise. Avant d'ajouter une nouvelle ligne dans la table *facture*, on requête la table *paiement* pour s'assurer de la solvabilité de ce client. Si c'est le cas, on ajoute une nouvelle facture. Pour chaque produit afférant à la facture ajoutée, on décrémente son stock dans la table *produit*.
  - En résumé, il s'agit d'opérations ou de transactions permettant d'effectuer des modifications d'informations en temps réel
- OLTP signifie « ***O****n-****L****ine* ***T****ransactional* ***P****rocessing* » pour *traitement transactionnel en ligne*

LIMITES D'UNE UTILISATION OLTP : VERS UNE UTILISATION OLAP

- Quid des besoins statistiques?
  - L'utilisation transactionnelle (OLTP) d'une base de données ne permet pas une grande performance analytique et prédictive des données
    - ***Exemple*** : Comment obtenir une analyse complète et détaillée des tendances générales de ventes par type de produit, par régions, par mois,...?
    - On parle ici, de requêtes OLAP signifiant « ***O****n-****L****ine* ***A****nalytical* ***P****rocessing* **»**

LIMITES D'UNE UTILISATION OLTP : VERS UNE UTILISATION OLAP

- Quid des besoins statistiques?
  - Les requêtes OLAP ont pour but d'agréger des résultats de données dans le but de dégager des tendances générales et de construire des modèles statistiques à partir de ces données
    - Il est impératif qu'elles soient formulées sur de grands volumes de données pour une performance parfaite
  - Les requêtes OLTP sont inadaptées pour le traitement analytique d'importantes quantités de données et pour la construction de modèles statistiques prédictifs (utilisés notamment en *informatique décisionnelle ou Business Intelligence (BI)(voir ci-après))*

EVOLUTION DES STRUCTURES DE DONNÉES

- ***Modèles OLAP en étoile et en flocon***
  - Suite à l'évolution des besoins des entreprises et à l'explosion du volume de données à traiter, les structures relationnelles traditionnelles ont été remplacées par des structures plus adaptées à un traitement des données OLAP
    - Dès lors, deux modèles supplémentaires ont fait leur apparition
    - Il s'agit des modèles en « *étoile* » et en « *flocon* »
      - Ces modèles s'inscrivent typiquement dans une logique de structuration « *multidimensionnelle* » permettant de stocker des données atomiques ou agrégées
      - Cette structuration des données est nettement plus adaptée à un grand nombre de parcours sur données volumineuses
- ***Mémento :*** *Les données qui serviront à un traitement OLAP sont généralement stockées dans des « entrepôts de données » (ou Data Warehouse) dont l'architecture repose sur la structure de ces modèles.*

EVOLUTION DES STRUCTURES DE DONNÉES

- ***Modèles OLAP en étoile et en flocon***
  - Pour concevoir un modèle en « *étoile* » ou en « *flocon* », il est impératif de ne plus réfléchir en termes de « *tables* » et « *relations* » (comme pour un modèle relationnel-OLTP) mais plutôt en termes de « *dimensions* » et « *faits* »
    - Les « *faits* » ou « *mesures* » sont les éléments mesurés dans l'analyse comme les *montants*, les *quantités*, les *taux d'intérêt*s, etc...
      - Il s'agit de « *tables* » contenant les « *informations opérationnelles métriques* » et relatives aux fonctionnement et activités d'une entreprise
    - Les « *dimensions* » sont les axes d'analyse comme la *catégorie de produits* (type de produit, gamme, etc...), la *géographie clientèle* (région, département, etc...), etc...
      - Il s'agit de tables contenant les dimensions explorées par l'analyse

BUSINESS INTELLIGENCE

CONSULTING BSTORM

BUSINESS INTELLIGENCE

- Qu'est-ce que l'*informatique décisionnelle* ou *Business Intelligence (BI)*?
  - Il s'agit de l'ensemble des techniques et des outils permettant de collecter des données brutes pour les modéliser et les restituer ensuite sous forme d'informations pertinentes et utiles pour l'analyse business
  - La BI représente une aide décisionnelle extrêmement précieuse pour les gestionnaires
    - Elle a pour but premier d'identifier de nouvelles opportunités d'affaires et d'implémenter une stratégie opérationnelle efficace dans le but d'établir un avantage concurrentiel de marché relativement long terme

BUSINESS INTELLIGENCE

- ***Utilisation du modèle OLAP en BI***
  - Différentes applications de la BI sur base d'une utilisation OLAP
    - Decision support systems
    - Query and reporting
    - Statistical analysis, forecasting and data-mining
    - Benchmarking
    - Etc,...

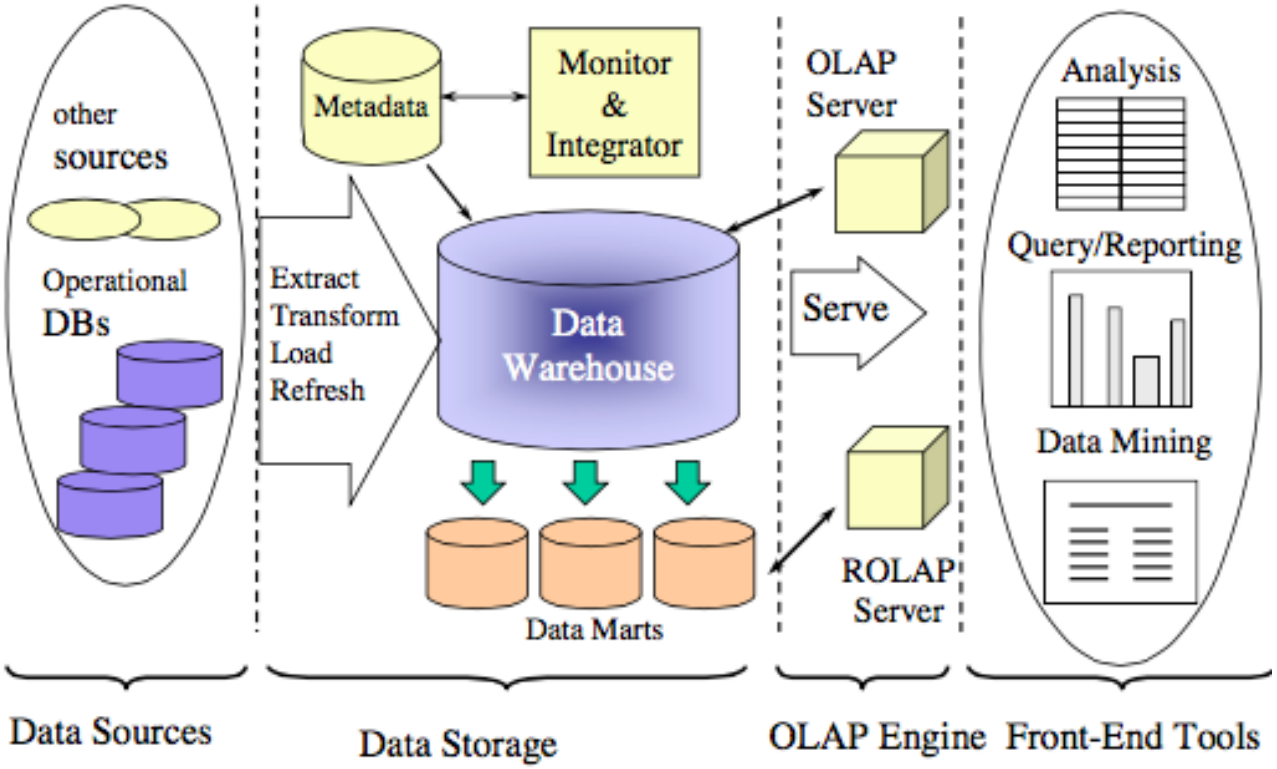
BUSINESS INTELLIGENCE

- ***OLTP Vs. OLAP***

	OLTP	OLAP
Objectif	Besoins opérationnels	Analyse business
donnée	Petite taille, opérationnelle	Grande taille, historique
modèle	normalisé	Dénormalisé/ multidimensionnel
Langage de requête	SQL	Pas de langage unique
requêtes	courtes	longues
Mises à jour	Fréquentes et courtes	Non fréquentes et discontinues
Récupération transactionnelle	nécessaire	Non nécessaire
Efficace pour	Les opérations de mise à jour	Les opérations de requêtes

BUSINESS INTELLIGENCE

- Figure 3 : framework complet de la BI



BUSINESS INTELLIGENCE

- ***Data Mining***
  - Le « *Data Mining* » (*exploration de données*) reprend l'ensemble des connaissances et des outils permettant à ses utilisateurs d'analyser des grandes quantités de données (via des méthodes semi-automatiques voire automatiques) et d'extraire de la valeur depuis des tendances et modèles (*patterns*)
    - Des prédictions peuvent être réalisées sur base des données
    - Construction de modèles afin d'y dégager des structures intéressantes ou motifs pour le business sur base de critères préfixés
  - Cette exploration de données présuppose l'utilisation d'un ensemble d'algorithmes issus de différents domaines d'application
    - Statistiques
    - Intelligence artificielle
    - Informatique décisionnelle
  - Le *data Mining* est donc un préalable à l'*informatique décisionnelle*

  - *<u>Memento:</u> Un pattern est une représentation/modèle synthétique d'un ensemble de données*

BUSINESS INTELLIGENCE

- ***Data Mining***
  - 4 étapes du Data Mining:
    - Sélection et échantillonnage: Sélectionner les informations utiles dans la base de données
    - Transformation et nettoyage: Changement de format si nécessaire pour permettre l'utilisation des algorithmes
    - Data Mining: Lancement des algorithmes afin de générer les patterns
    - Evaluation: Examiner et valider des patterns, retour à l'étape précédente pour ajuster les algorithmes

BUSINESS INTELLIGENCE

- ***BI Vs. BIG DATA***
- Problème de la BI?
  - Les outils offerts par la BI ne permettent pas le traitement et l'analyse de données non structurées
    - Contrairement aux bases de données relationnelles (qui représentent des fichiers structurés), les fichiers non structurés ne possèdent pas de schéma de structuration de données
    - On les qualifie de « schema-less » ou de « schema-free »
  - Le BIG DATA apparaît comme la solution à ce problème en proposant une série d'outils informatique (voir ci-après dans la partie utilisation : No-SQL)
- *Mémento : Typiquement la structure apparentée aux bases de données relationnelles est l'armature des relations entre les entités (tables)*

DONNÉES NON STRUCTURÉES

CONSULTING BSTORM

DONNÉES NON STRUCTURÉES

- Les données non structurées sont des données représentées ou stockées sans format prédéfini
  - [1] Données non structurées textuelles
    - Données générées par courriels, présentations PowerPoint, documents Word, messageries instantanées,...
  - [2] Données non structurées non textuelles
    - Données générées via des supports tels que les images JPEG, les fichiers audio MP3, ou encore les fichiers vidéo Flash
- *Mémento : Une fois extraites, ces données non structurées sont stockées dans des fichiers spécifiques au format « JSON ». Ce format permet de représenter de l'information de manière structurée comme le permet le « XML ».*

DONNÉES NON STRUCTURÉES

- Exemple : format JSON

```
{
    "menu": {
        "id": "file",
        "value": "File",
        "popup": {
            "menuitem": [
                { "value": "New", "onclick": "CreateNewDoc()" },
                { "value": "Open", "onclick": "OpenDoc()" },
                { "value": "Close", "onclick": "CloseDoc()" }
            ]
        }
    }
}
```

DONNÉES NON STRUCTURÉES

- Exemple : format XML

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

BIG DATA

CONSULTING BSTORM

BIG DATA

- Qu'est-ce que le BIG DATA?
  - Il s'agit des données non structurées qui échappent à la capacité de traitement des SGBDR traditionnels
    - Ces données sont trop volumineuses, évoluent trop rapidement et ne sont plus en accord avec l'architecture des bases de données relationnelles traditionnelles
  - Le BIG DATA permet donc de réaligner ces volumes de données avec des architectures et des modèles de structuration de données plus adaptés
- *Mémento : Le phénomène BIG DATA a émergé avec l'apparition de Google, Facebook, Amazon,...gérant des volumes de données colossaux et présentant une évolution rapide.*

BIG DATA

- L'objectif du BIG DATA est de récolter de données qui serviront d'input aux systèmes de gestion de données volumineuses (*big data systems*)
  - Ces données peuvent provenir de...
    - Chats issus de réseaux sociaux
    - Web server log
      - Il s'agit d'un fichier (*log file*) automatiquement créé par un serveur contenant l'historique de navigation d'une session utilisatrice web
    - Senseurs à différents effets
    - Transactions bancaires
    - Scan de documents, de cartes GPS,...
    - Etc,...
- *Mémento : Un « serveur internet » (en anglais web server) est un outil informatique permettant l'exécution de requêtes HTTP. La fonction primaire d'un serveur internet est de stocker, traiter et fournir des pages web à l'utilisateur.*

BIG DATA

- ***Les 3V du BIG DATA***
  - Le challenge du BIG DATA est de traiter 3 aspects
    - [1] Volume
    - [2] Vélocité
    - [3] Variété

BIG DATA

- ***Les 3V du BIG DATA***
  - [1] Volume
    - Il s'agit de la capacité à traiter des volumes de données relativement conséquents
    - C'est le principal challenge aux structures traditionnelles
      - Application d'une *architecture de requêtes distribuées* (*distributed querying*)
      - Prévision d'un *espace de stockage suffisant et modulable* (*scalable storage*)

BIG DATA

- ***Volume & solution Apache Hadoop***
  - La solution Apache Hadoop est une architecture de traitement massif distribuée
    - Elle permet de répartir un *ensemble de données (data set)* sur différents *serveurs ou nœuds de calcul* (*servers or computing nodes*) pour une ***exécution en parallèle et déléguée*** du *problème computationnel* (*request*)
    - Hadoop ne pose pas de contraintes sur la structure de données à traiter
    - Typiquement elle permet de traiter des données semi-structurées voire non structurées
- *Mémento* : « *Apache Hadoop » est développée et proposée en Open Source par Yahoo*

BIG DATA

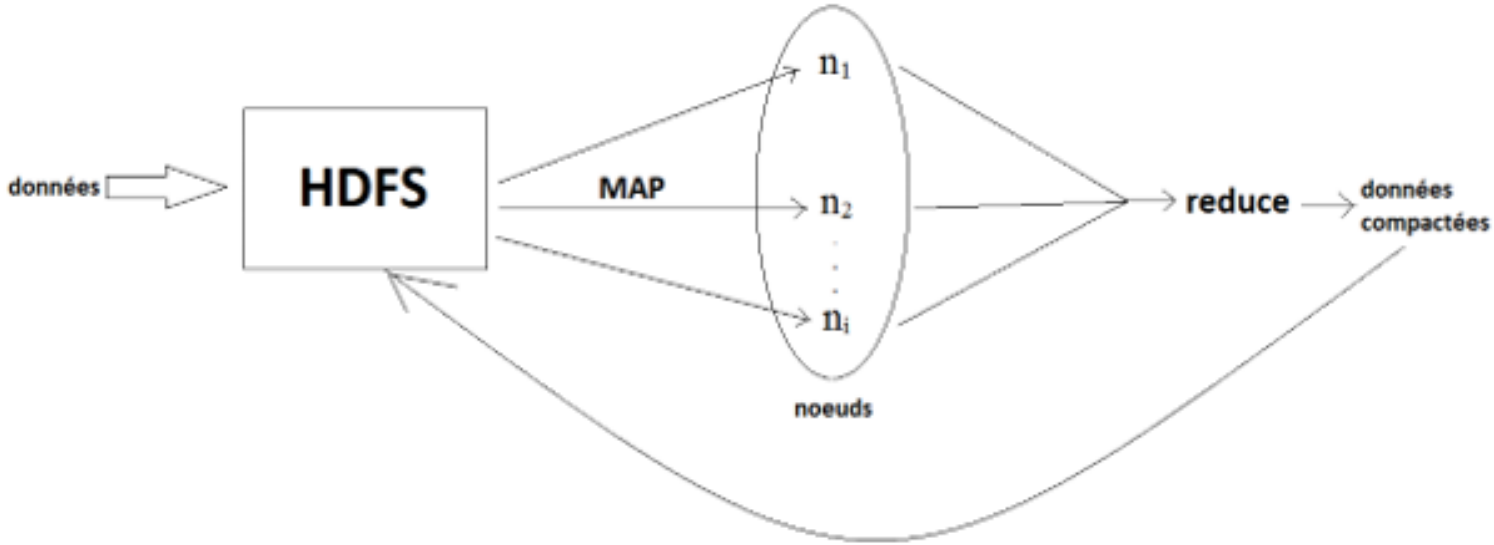
- ***Hadoop & pattern MapReduce***
  - Hadoop implémente le pattern MapReduce
    - Ce pattern permet de distribuer une requête en la découpant sur différents serveurs pour l'exécuter en parallèle
      - Cette distribution ou délégation du problème computationnel permet de résoudre la problématique des données non structurées trop volumineuses pour un seul serveur tout en améliorant les performances de calculs
      - Il s'agit de la « *montée en charge horizontale* » impliquant une augmentation du nombre de « *nœuds de calcul* »
        - Passage à une « *architecture distribuée* » revêtant une structuration de données horizontale parmi plusieurs serveurs
- *<u>Mémento</u> : Un pattern informatique est un modèle ou une structure générale permettant de résoudre une problématique associée récurrente*

BIG DATA

- ***Hadoop & pattern MapReduce : 3 phases***
  - Chargement des données dans un « *système de fichier distribué* » (HDFS)
    - Cette phase a pour but de rendre les données disponibles à un ensemble de « *nœuds de calcul* » (computing nodes)
  - Opérations de *distribution* (Map) et de *réduction* des données (Reduce/recombined)
    - L'opération de distribution a pour but de répartir l'exécution de la requête sur différents serveurs
      - Découpage de la requête en multiples sous-requêtes
    - L'opération de réduction a pour but d'agréger les résultats
      - Récupération des résultats de chaque sous-requête pour former un résultat agrégé
  - Retour des résultats au « *système de fichier distribué* »

BIG DATA

- ***Hadoop & pattern MapReduce***
  - Modélisation du pattern



BIG DATA

- ***Les 3V du BIG DATA***
  - [2] Vélocité
    - Il s'agit de la vitesse avec laquelle les données sont générées, stockées, partagées et mises à jour
    - La vélocité ne caractérise pas uniquement les données générées (*input data*) mais prend également en compte le temps de réponse d'une requête faite sur ces données (*speed of the feedback loop = system's outputs*)
    - Des volumes de données croissants doivent être analysés en temps *quasi-réel* pour répondre aux *besoins organisationnels*
- *<u>Mémento</u> : Dans la littérature, ces volumes données croissants sont appelés « streaming data » ou « complex event processing »*

BIG DATA

- ***Les 3V du BIG DATA***
  - [3] Variété
    - Il s'agit de l'hétérogénéité des sources de données à traiter
      - Exemples : données issues de fenêtres de « chats » de réseaux sociaux, données récoltées par différents senseurs, etc...
    - Etant donné cette hétérogénéité des sources et cette inadéquation avec les structures relationnelles traditionnelles, ces sources ne peuvent être traitées de manière optimale
    - Les bases de données non-relationnelles et la technologie No-Sql (voir partie technique) permettent d'offrir une réponse adaptée à cette problématique d'hétérogénéité

EXEMPLES CONTEXTUELS

- *Patrick, un utilisateur de Facebook souhaite consulter un profil. Pour cela, il tape « Marie » dans la barre de recherche des profils. Quelques secondes plus tard, Facebook lui retourne le profil associé de Marie.*
  - Il s'agit typiquement d'un cas d'utilisation du BIG DATA
- Patrick est assimilé à une « *session utilisatrice* »
- Lorsque Patrick tape le nom « Marie » dans la barre de recherches, celui-ci adresse une requête à l'un des serveurs web de Facebook
- Une fois cette requête distribuée et découpée sur plusieurs serveurs, elle peut être analysée et traitée
- Après traitement de la requête, Facebook retourne une page web (*information product*) avec le profil de Marie et les profils associés de ses amis Facebook
  - Ceci fait référence à la ***théorie des graphes et des réseaux*** permettant de remonter des informations corrélées (profils Facebook associés au profil de Marie) et distribuées sur différents serveurs à la fois
    - Cela est rendu possible grâce à *l'architecture distribuée*
- *Mémento : Le résultat du traitement de la requête HTML, (prenant, pour l'exemple, la forme d'une page web), est appelé « code client » à ne pas confondre avec « session utilisatrice » ou « session client »*

PARTIE TECHNIQUE

PARTIE TECHNIQUE

- Plan
  - Introduction
    - Qu'est-ce qu'un moteur No-Sql
    - Rôles d'un moteur No-Sql
    - Différents types de moteurs No-Sql
      - Schéma et contenu
      - Théorème CAP
      - Modèle PACELC
  - Les choix techniques du No-Sql
    - Interface avec le code client
      - Fonctionnalités serveur
        - L'exemple de CouchDB
        - Création d'une vue en Python
    - Protocoles d'accès aux données
      - Interfaces natives
      - Interfaces REST

PARTIE TECHNIQUE

- Architecture distribuée
  - Distribution avec maître
    - Réplication maître-esclave
    - Sharding
  - Distribution sans maître
    - Protocole de « bavardage »
    - Hachage consistant

INTRODUCTION

- ***Qu'est-ce qu'un moteur No-Sql?***
  - Un « *moteur No-Sql* » est un « *logiciel* » ou « *module informatique* » permettant de gérer un ensemble de base des données non relationnelles
    - On le qualifie parfois de « *SGBD No-Sql* »
    - Les « *SGBD No-Sql* » ne sont plus fondés sur une logique de « tables » contrairement aux SBDGR
  - La technologie No-Sql à été crée dans le but de répondre aux besoins du BIG DATA
  - La différence majeure de philosophie du « *No-Sql* » par rapport à celle du « *Sql* » est de faire reposer les contraintes et les difficultés de traitement sur le programmeur et son « *code client* » plutôt que de les imputer directement au SGBD
    - Il est toutefois possible de définir des contraintes au niveau du SGBD (voir choix techniques du No-Sql)
- ***Mémento :*** *Un « serveur » est un dispositif informatique offrant des services à des clients. Parmi les services les plus courants : la possibilité de stockage en base de données et la consultation de celle-ci.*

INTRODUCTION

- ***Qu'est-ce qu'un moteur No-Sql?***
  - La plupart des « *moteurs No-Sql* » offrent des « *API* » sous forme d'un ensemble de fonctions et de bibliothèques informatiques qui permettent de manipuler les objets d'une base de données non relationnelle
  - Le « code client » fait directement appel aux différentes fonctions proposées par l'« API »
    - Exemple de code client : le code PHP d'un site web
  - A la différence des « *scripts Sql* » de structure traditionnelle « *select-from-where » qui sont interprétés par les SBDG Sql,* les SGBD No-Sql ne passent pas par cet « intermédiaire de traduction » et opèrent directement le traitement communiqué par l'appel de fonctions du « code client »
- ***Mémento :*** *Une « API » pour « Application Programming Interface » signifie « interface de programmation client »*

INTRODUCTION

- ***Rôles d'un moteur (SGBD) No-Sql***
  - ***Organiser les données***
    - Le données sont organisées en différentes entités stockées sur le disque permettant un accès rapide à celles-ci
  - ***Gérer les données***
    - Garantir la cohérence finale des données à travers tous les nœuds (*théorème CAP*)
      - Il s'agit de la capacité à renvoyer la même valeur à toutes les sessions-client, à tous les observateurs
  - ***Assurer l'accès aux données***
    - Il faut pouvoir accéder aux données stockées par plusieurs sessions-client ou systèmes de transaction distribués (ex : *HDFS*)

INTRODUCTION

- ***Rôles d'un moteur (SGBD) No-Sql***
  - ***Protéger les données contre les incidents***
    - Les moteurs No-Sql garantissent la protection des données par « *réplication* » ou copie de celles-ci sur différents serveurs
  - ***Accroitre les performances***
    - Le « *partitionnement* » des données aussi automatique que possible permet une augmentation de la vitesse de traitement et des performances

THÉORÈME CAP

CONSULTING BSTORM

THÉORÈME CAP

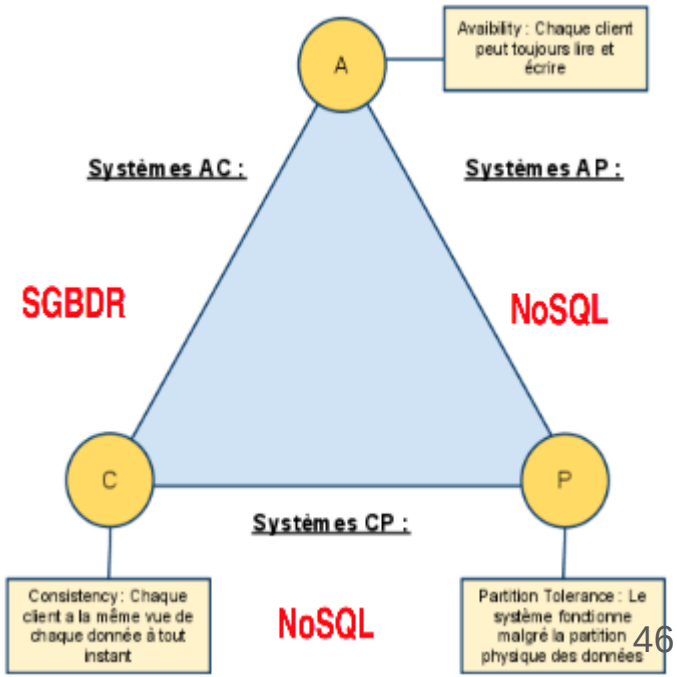
- ***Différents types de moteurs No-Sql : théorème CAP***
  - L'informatique distribuée se focalise principalement sur le « *calcul* » et non sur les « *données* »
    - Pour des besoins de calcul on crée des « *clusters* » ou des « *grids* » de travail dans le but d'accroitre la puissance de calcul en la multipliant par le nombre de « *nœuds* »
  - Un algorithme « *MapReduce* » (implémenté par la plupart des moteurs No-Sql) qui distribue le travail et agrège les résultats, n'est pas difficile à concevoir et à mettre en pratique
    - Le ***calcul distribué*** est relativement simple
  - Par contre, la ***distribution des données*** est une tâche nettement plus complexe
- ***Mémento :*** *Il faut faire la différence entre un « **cluster** » et un « **grid** » de machines : un cluster est un groupe de machines situé dans le même « data center » alors qu'un grid est un groupe de machines distribuées dans plusieurs « data centers ».*

THÉORÈME CAP

- ***Différents types de moteurs No-Sql : théorème CAP***
  - Afin de formaliser et d'encadrer cette tâche, le « *théorème CAP* » énonce ***trois propriétés d'un système distribué (SD)***:
    - ***Cohérence*** (*consistency*) : tous les nœuds sont mis à jour sur les données au même instant
    - ***Disponibilité*** (*availability*) : la perte d'un nœud n'empêche pas le système de fonctionner et de servir l'intégralité des données
    - ***Résistance au morcellement*** (*partition tolerance*) : chaque nœud doit pouvoir fonctionner de manière autonome

THÉORÈME CAP

- Les SGBDR sont du type « AC » car ils assurent la **disponibilité** et la **consistance** des informations
- Un système NoSQL est « CP » s'il assure la **consistance** mais pas la disponibilité
  - ➔ Pour chaque donnée un seul nœud est responsable (ex: *HBase*)
- Un système NoSQL est « AP » s'il assure la **disponibilité** mais pas la consistance
  - ➔ Les données sont copiées sur chaque nœud (ex: *Cassandra*)



THÉORÈME CAP

HBase:

- Key characteristics:
  - Distributed and scalable big data store
  - Strong consistency
  - Built on top of Hadoop HDFS
  - CP on CAP
- Good for:
  - Optimized for read
  - Well suited for range based scan
  - Strict consistency
  - Fast read and write with scalability
- Not good for:
  - Classic transactional applications or even relational analytics
  - Applications need full table scan
  - Data to be aggregated, rolled up, analyzed cross rows
- Usage Case: Facebook message

THÉORÈME CAP

- **Cassandra:**
- Key characteristics:
  - High availability
  - Eventually consistent
  - Trade-offs between consistency and latency
  - Minimal administration
  - No SPF (Single point of failure) – all nodes are the same in Cassandra
  - AP on CAP
- Good for:
  - Simple setup, maintenance code
  - Fast random read/write
  - Flexible parsing/wide column requirement
- Not good for:
  - Relational data
  - Transactional operations (Rollback, Commit)
  - Primary & Financial record
  - Stringent and authorization needed on data
  - Dynamic queries/searching on column data
- Usage Case: Twitter, Travel portal

THÉORÈME CAP

- **MongoDB:**
- Key characteristics:
  - Schemas to change as applications evolve (Schema-free)
  - Full index support for high performance
  - Replication and failover for high availability
  - Auto Sharding for easy Scalability
  - Rich document based queries for easy readability
  - Master-slave model
  - CP on CAP
- Good for:
  - RDBMS replacement for web applications
  - Semi-structured content management
  - Real-time analytics and high-speed logging, caching and high scalability
  - Web 2.0, Media, SAAS, Gaming
- Not good for:
  - Highly transactional system
  - Applications with traditional database requirements such as foreign key constraints
- Usage Case: Craigslist, Foursquare

LES ORIENTATIONS NO-SQL

CONSULTING BSTORM

LES ORIENTATIONS NO-SQL

Les No-SQL orientés Clé / valeur :

- Ce modèle peut être assimilé à une hashmap distribuée - les données sont, donc, simplement représentées par un couple clé/valeur:
  - La valeur peut être une simple chaîne de caractères, un chiffre, un objet sérialisé…
- Cette absence de structure ou de typage ont un impact important sur le requêtage. En effet, toute l'intelligence portée auparavant par les requêtes SQL devra être portée par l'applicatif qui interroge la BD.
- Néanmoins, la communication avec la BD se résumera aux opérations PUT, GET et DELETE.
- Les solutions les plus connues sont Redis, Riak et Voldemort créé par LinkedIn.

LES ORIENTATIONS NO-SQL

Les No-SQL orientés Colonne :

- Ce modèle ressemble à première vue à une table dans un SGBDR à la différence qu'avec une BD NoSQL orientée colonne, le nombre de colonnes est dynamique.
  - Dans une table relationnelle, le nombre de colonnes est fixé dés la création du schéma de la table et ce nombre reste le même pour tous les enregistrements dans cette table.
  - Dans ce modèle, le nombre de colonnes peut varier d'un enregistrement à un autre ce qui évite de retrouver des colonnes ayant des valeurs NULL.
- Comme solutions, on retrouve principalement HBase (implémentation Open Source du modèle BigTable publié par Google) ainsi que Cassandra (projet Apache qui respecte l'architecture distribuée de Dynamo d'Amazon et le modèle BigTable de Google).

LES ORIENTATIONS NO-SQL

Les No-SQL orientés document :

- Ce modèle se base sur le paradigme clé valeur.
  - La valeur, dans ce cas, est un document de type JSON ou XML.
- L'avantage est de pouvoir récupérer, via une seule clé, un ensemble d'informations structurées de manière hiérarchique.
- La même opération dans le monde relationnel impliquerait plusieurs jointures.
- Pour ce modèle, les implémentations les plus populaires sont CouchDB d'Apache, RavenDB (destiné aux plateformes .NET/Windows avec la possibilité d'interrogation via LINQ) et MongoDB.

LES ORIENTATIONS NO-SQL

Les No-SQL orientés graphe :

- Ce modèle de représentation des données se base sur la théorie des graphes.
- Il s'appuie sur la notion de noeuds, de relations et de propriétés qui leur sont rattachées.
- Ce modèle facilite la représentation du monde réel, ce qui le rend adapté au traitement des données des réseaux sociaux.
- La principale solution est Neo4J.

LES CHOIX TECHNIQUES DU NO-SQL

CONSULTING BSTORM

INTERACTIONS AVEC UN SERVEUR NO-SQL

- Il est question à présent de détailler les caractéristiques techniques du No-Sql
- On peut classifier ces caractéristiques en ***deux grandes tendances*** correspondant à des besoins et à des utilisations spécifiques
  - Certains moteurs No-Sql sont principalement utilisés pour résoudre des problèmes d'interrogation de données volumineuses (= utilisation « BIG DATA »)
  - D'autres, sont utilisés pour opérer le traitement le plus rapide possible des données moins volumineuses mais demandant un flux optimisé de ces données (= utilisation « performance »)
- ***Mémento :*** *Sachez que les outils No-Sql diffèrent suivant que l'on se trouve dans un cas d'utilisation plutôt que l'autre.*

INTERACTIONS AVEC UN SERVEUR NO-SQL

- Le langage SQL repose sur une programmation déclarative, ensembliste et parfois fonctionnelle ne favorisant pas la compréhension naturelle des développeurs habitués aux langages impératifs
- La force du mouvement No-SQL réside principalement dans la *facilité de développement* et dans une *meilleure intégration du code serveur dans le code client*
  - Le langage du serveur No-Sql est littéralement le code client
    - Le code client est la programmation directe d'un serveur (ou d'une application quelconque) permettant d'interroger une/des bases de données
  - Une des qualités des bases No-Sql est qu'elles s'adaptent au programmeur et à ses contraintes ce qui permet une plus grande flexibilité de conception
- Les moteurs No-Sql sont implémentés comme bibliothèques issues des langages de développement les plus fréquemment utilisés
  - Pour appeler les fonctions d'un moteur No-Sql, il suffit d'importer sa bibliothèque

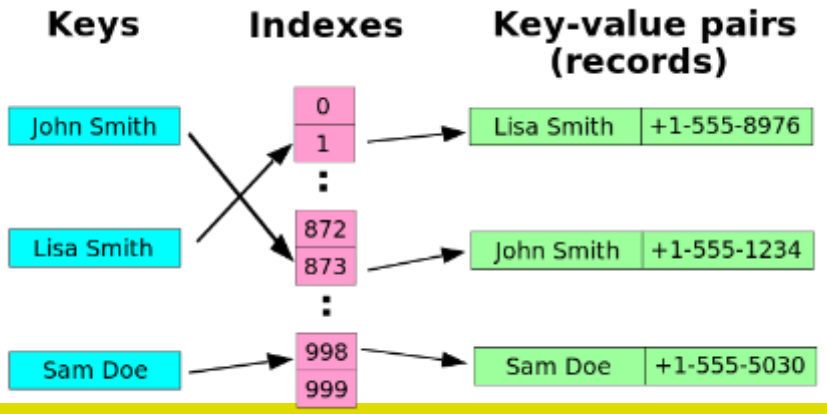
INTERACTIONS AVEC UN SERVEUR NO-SQL

- Exemple de code client (Python)

```
#!/usr/bin/python

import redis

r = redis.StrictRedis(host='localhost', port=6379, db=2)

r.hmset('contact:ajar', dict(
                prenom='Emile',
                nom='Ajar',
                Profession='Ecrivain'
        )
)

dic = r.hgetall('contact:ajar')
print dic
```

INTERACTIONS AVEC UN SERVEUR NO-SQL

- Explication
  - Redis est un moteur No-Sql servant d'entrepôt de paires « clé-valeur » en mémoire permettant de manipuler des structures de données (similaires aux structures issues des programmes clients) comme des tableaux, des strings, des ensembles, des tables de hachages, etc...



INTERACTIONS AVEC UN SERVEUR NO-SQL

- La fonction « *hmset* » permet de stocker directement les données (via une table de hachage) sous la clé « *'contact:ajar'* »
  - Cette table de hachage sera traduite par le moteur Redis pour pouvoir ensuite être exploitée en Python

INTERACTIONS AVEC UN SERVEUR NO-SQL

- Explication
  - On récupère ensuite la table de hachage grâce à la méthode « *hgetall* » en lui passant la clé en paramètre
  - La commande « print » nous renvoie le résultat suivant :

```
{'nom': 'Ajar', 'Profession': 'Ecrivain', 'prenom': 'Emile'}
```

FONCTIONNALITÉS D'UN SERVEUR NO-SQL

- Les bases No-Sql (contrairement aux bases Sql), n'imposent aucune complexité déléguée supplémentaire vers le serveur
  - Le serveur No-Sql renvoie un document ou une collection de documents
  - Ensuite, le programmeur prévoit dans son code client des lignes de codes permettant d'ouvrir ces documents et de les parcourir, de les parser
- Les moteurs No-Sql sont utilisés comme de simple entrepôts de données

LES CHOIX TECHNIQUES DU NO-SQL

- Le modèle relationnel est conceptuellement plus riche au niveau du serveur de par sa possibilité de création de contraintes, de déclencheurs (*triggers*), de procédures, etc...
  - Toute fonctionnalité permettant de transformer le serveur en une véritable couche applicative
    - On parle de « *middle-tiers* » pour qualifier un tel serveur
- La plupart des moteurs No-Sql n'offre pas cette possibilité à l'exception de « *CouchDB* »

FONCTIONNALITÉS D'UN SERVEUR

NO-SQL

CONSULTING BSTORM

FONCTIONNALITÉS D'UN SERVEUR NO-SQL

- « *CouchDB* » utilise le mécanisme des « documents design » permettant de rendre exécutable une partie de code du coté du serveur
- Ces « documents design » sont traduits du coté serveur par un serveur de requête (query server)
  - Le code contenu dans un « document de design » est envoyé au serveur de requête
  - Le langage de développement au niveau serveur de « CouchDB » est JavaScript

FONCTIONNALITÉS D'UN SERVEUR NO-SQL

- Pour créer une vue, il suffit d'utiliser l'API de « CouchDB » appelée « Futon » (disponible en ligne sur le web)
- Cette API offre la possibilité de créer une vue temporaire sur nos données (Temporary view)
- Via cette API, on entre le code Python de notre vue dans la section « Map Fonction »
  - Il faut indiquer au préalable à « CouchDB » que la vue est en Python grâce au tapis déroulant « Language »

FONCTIONNALITÉS D'UN SERVEUR NO-SQL

- Exemple de création d'une vue (Python)

```
def fun(doc):
    if doc['auteur']['nom'] == 'Brizard':
        yield None, doc['auteur']
```

FONCTIONNALITÉS D'UN SERVEUR NO-SQL

- Explication
  - Cette fonction teste si le document ou si la collection de documents stockée sur le serveur contient un auteur dont le nom est « *Brizard* »
  - On fait l'hypothèse que c'est le cas et voyons ensuite le résultat stocké dans « CouchDB »
  - Lorsque le document est trouvé, la vue renvoie uniquement la partie « auteur » de celui-ci grâce à la fonction « *yield* »
    - La résultat de l'exécution de la fonction apparait dans l'API
    - Ce résultat est appelé « document de design »

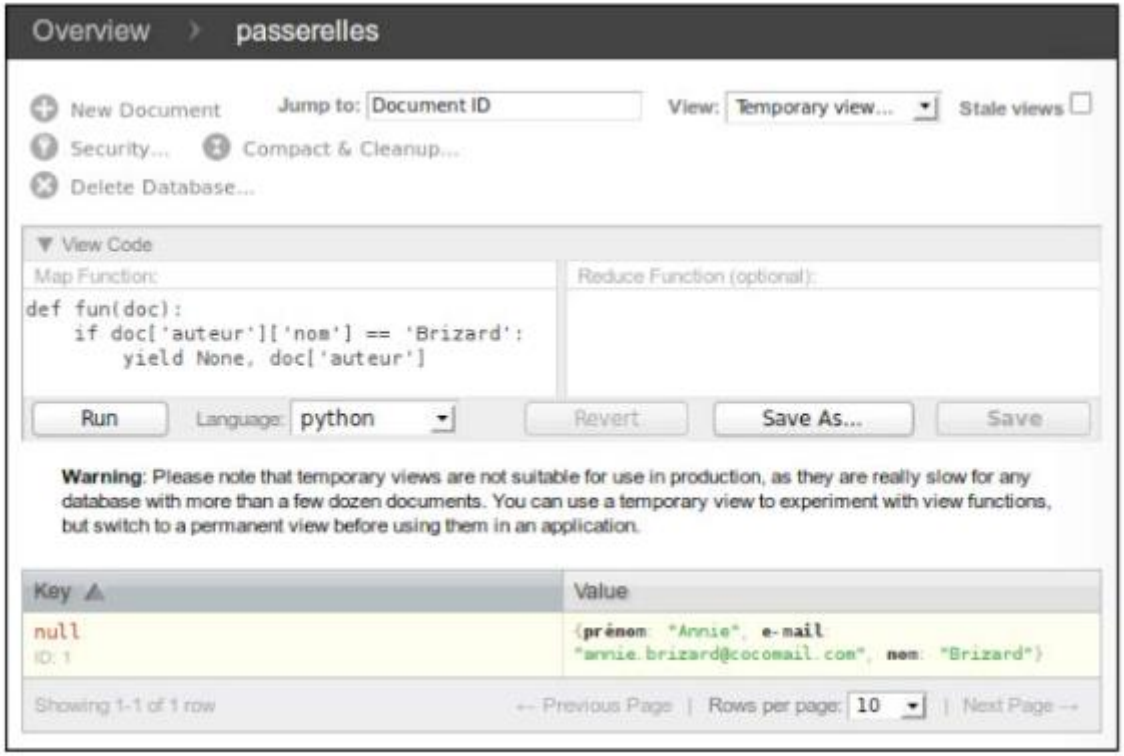
FONCTIONNALITÉS D'UN SERVEUR NO-SQL

- Explication : document trouvé


```
{
    "_id": "1",
    "_rev": "2-1a97e3eb387fd93cf6e1abb2a1aed56b",
    "auteur": {
        "prénom": "Annie",
        "nom": "Brizard",
        "e-mail": "annie.brizard@cocomail.com"
    },
    "titre": "je change mon titre, il était trop bizarre"
}
```

FONCTIONNALITÉS D'UN SERVEUR NO-SQL

- Explication : résultat de la vue (API)



PROTOCOLES D'ACCÈS AUX DONNÉES

CONSULTING BSTORM

PROTOCOLES D'ACCÈS AUX DONNÉES

- Les moteurs No-Sql implémentent différentes méthodes d'accès aux données
  - Cela se fait par l'intermédiaire d'une « interface » entre le client et le serveur
  - La philosophie des moteurs No-Sql est de privilégier le traitement du coté client plutôt que de l'attribuer au serveur
- Il est question à présent de présenter les « protocoles d'interfaçage » les plus communs
  - Notez que pour chacun de ceux-ci, il existe des bibliothèques pour tous les grands langages clients afin de faciliter leur utilisation

PROTOCOLES D'ACCÈS AUX DONNÉES

- ***Interfaces natives***
  - Certains moteurs No-Sql comme « MongoDB » (voir ci-après) fonctionnent sur base d'un protocole « natif » pour entrer en communication avec le serveur
    - Il s'agit d'une communication en mode « question-réponse »
    - Ce protocole permet d'envoyer des messages composés d'un « en-tête » et d'un « corps de message »
    - La structure des messages envoyés est relativement simple
    - Les documents envoyés au serveur sont en format « BSON », un genre JSON « *binarisé* » (données compactées)

PROTOCOLES D'ACCÈS AUX DONNÉES

- ***Interfaces natives***
  - Exemple : structure d'un message d'insertion (serveur)

```
struct {
   MsgHeader header; // en-tête de message contenant l'opcode et d'autres informations
    int32 ZERO; // 0 - réservé pour une utilisation future

    cstring fullCollectionName; // le nom de la collection impliquée
    BSON[] documents; // un ou plusieurs documents à insérer dans la collection
}
```

PROTOCOLES D'ACCÈS AUX DONNÉES

- ***Interfaces natives***
  - Explication
    - On précise dans la structure du message le nom de la collection cible
      - « fullCollectionName »
    - On insère un ou plusieurs documents dans cette collection cible
      - « BSON[ ] documents » est un tableau de documents JSON « binarisés »
    - Développer en MongoDB revient à appeler les méthodes d'un objet « *collection* » (issu d'une bibliothèque) pour insérer, supprimer, requêter des documents, etc...
      - Les moteurs No-Sql utilise le paradigme Orienté-Objet (O_O)

PROTOCOLES D'ACCÈS AUX DONNÉES

- ***Interfaces REST***
  - Il existe un autre mode de communication entre machines dans le contexte No-SQL:
    - « *REST* » (*Representational State Transfer)* est une architecture propre à l'appel des ressources disponibles dans un environnement « hétérogène » et « distribué » comme le web
  - Comment fonctionne le protocole « *REST* »
    - Il repose sur le protocole de base du web : le HTTP

PROTOCOLES D'ACCÈS AUX DONNÉES

- ***Interfaces REST***
  - HTTP est basé sur l'échange de documents
    - Il comprend des verbes ou des méthodes qui permettent d'interagir avec un serveur web
      - Les verbes HTTP s'enrichissent au fil du temps
  - Pourquoi ne pas implémenter ces méthodes dans une interface « *REST* » pour communiquer avec des ressources distantes?
    - Plusieurs moteurs No-Sql implémentent une interface « *REST* »
      - Comme « CouchDB », « Riak », « HBase », etc…
    - C'est simple et élégant d'un point de vue conceptuel
- ***Mémento :*** *Un fournisseur de ressources qui respecte les règles du modèle REST est dit « RESTful ».*

PROTOCOLES D'ACCÈS AUX DONNÉES

- ***Interfaces REST***
  - Quelques méthodes utilisées dans les demandes REST
    - « *GET* »
      - Permet de récupérer une représentation d'une ressource, c'est à dire tout type de donnée
    - « *HEAD* »
      - Comme *GET*, elle permet d'envoyer une demande sans retourner directement tout le résultat
        - Ne retourne que l'en-tête
        - Utile pour récupérer des « métadonnées »
- ***Mémento :*** *Une métadonnée sert à qualifier ou à décrire une autre donnée. Ces données sont à la base du web « sémantique ».*

PROTOCOLES D'ACCÈS AUX DONNÉES

- ***Interfaces REST***
  - Quelques méthodes utilisées dans les demandes REST
    - « *PUT* »
      - Permet de créer une ressource ou de stocker une représentation d'une ressource
        - Permet donc l'insertion et la mise à jour des ressources
    - « *POST* »
      - Méthode offrant des fonctionnalités similaires à *PUT* mais de façon plus souple
        - *POST* envoie une requête au serveur alors que *PUT* envoie une ressource à stocker (via l'URL)

PROTOCOLES D'ACCÈS AUX DONNÉES

- ***Interfaces REST***
  - Quelques méthodes utilisées dans les demandes REST
    - « *DELETE* »
      - Permet de supprimer une ressource existante sur le serveur
    - « *Options* »
      - Retourne les ressources déjà stockées et disponibles sur le serveur

PROTOCOLES D'ACCÈS AUX DONNÉES

- ***Interfaces REST***
  - Exemple : demande REST

```
GET / HTTP/1.1
HOST: www.editions-eyrolles.com
```

  - Explication
    - Pour accéder au site web « *www.editions-eyrolles.com* », le navigateur web envoie une requête HTTP
    - Le verbe « *GET* » demande un document, une ressource

PROTOCOLES D'ACCÈS AUX DONNÉES

- ***Interfaces REST***
  - Conclusion
    - Suivant les besoins, l'interfaçage en REST avec un entrepôts de données No-Sql peut s'avérer être une excellente solution technique de par son aspect simpliste et universel
    - En revanche, cette solution laissera vite transparaitre ses limites si l'on a des besoins de performances

LE SYSTÈMES DISTRIBUÉS

CONSULTING BSTORM

ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- Comme vu précédemment lorsque nous évoquions le « volume » (pour la partie conceptuelle), l'un des avantages du No-Sql est de pouvoir distribuer les données sur plusieurs machines
  - Ceci permet d'assurer la gestion des données de plus en plus volumineuses ainsi qu'une rapide montée en charge
- L'« *architecture distribuée* » est l'un des fondements du No-Sql
- Il y a deux grandes façons de concevoir la « *distribution des traitements et des données* »
  - Avec ou sans maître
    - Les moteurs No-Sql ont le choix d'implémenter une architecture avec ou sans maître

ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Distribution avec maître***
  - Il s'agit de la « *distribution* » ou du « *partitionnement* » de données grâce à une « *machine maître* » qui tient le rôle de configuration du système
    - Elle reçoit les « *requêtes-clients* » et les redirige vers les machines qui contiennent les données désirées

ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Distribution avec maître : réplication maître-esclave***
  - Réplication ≠ distribution
    - Il s'agit d'une copie des données dans le but de prévenir la perte de données
      - C'est en quelque sorte une « *redondance contrôlée* »
  - On « *écrit* » sur le « *maître* » qui va ensuite répliquer les données sur les « *esclaves* »
    - Ces « *esclaves* » ou « *machines secondaires* » peuvent être utilisés en « *lecture* »
- ***Mémento :*** *« MongoDB » utilise cette réplication maître-esclave pour partitionner les données et assurer la redondance.*

ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Distribution avec maître : le sharding***
  - La notion de « *réplication maître-esclave* » est généralement combinée à celle de « *sharding* »
  - Le « *sharding* » consiste en un « *éclatement* » des données en différentes partitions distribuées sur plusieurs machines
    - Le terme spécifique de « *sharding* » est apparu pour désigner une technique de « *clustering* » présentant des caractéristiques un peu particulières
      - L'éclatement est géré de façon aussi automatique que possible par le système (moteur No-Sql)
      - L'éclatement est par nature « distribué »
        - Cela n'a pas de sens de créer plusieurs « *shards* » sur une même machine
      - La répartition de charge est possible

ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Architecture distribuée : distribution sans maître***
  - Le désavantage majeur d'un système distribué avec maître est la présence d'un « *SPOF* » (*Single Point Of Failure*)
    - Il s'agit d'un élément non redondant représentant un risque important de défaillance pour tout le système
  - Une « *architecture sans maître* » implique que chaque machine a la même importance et les mêmes capacités au sein du « *cluster* »
    - Le but est de pouvoir fournir un « *service complet et autonome* » en se connectant à n'importe qu'elle machine, même dans le cas où plusieurs éléments du cluster sont défaillants
- ***Mémento :*** *Dans la littérature, on parle de « système distribué et décentralisé ».*

ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Architecture distribuée : distribution sans maître***
  - Quid d'un système distribué et décentralisé fonctionnel?
    - Quelques questions d'implémentations importantes à se poser...
      - Comment diriger les requêtes-clients sur les machines qui contiennent l'information souhaitée ?
      - Comment maintenir l'état du système?
        - Comment savoir quelles sont les machines qui composent le cluster?
      - Comment et où distribuer les données de façon à assurer la meilleure répartition possible?
    - Nous allons tenter de répondre à ces questions en abordant quelques techniques et concepts

ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

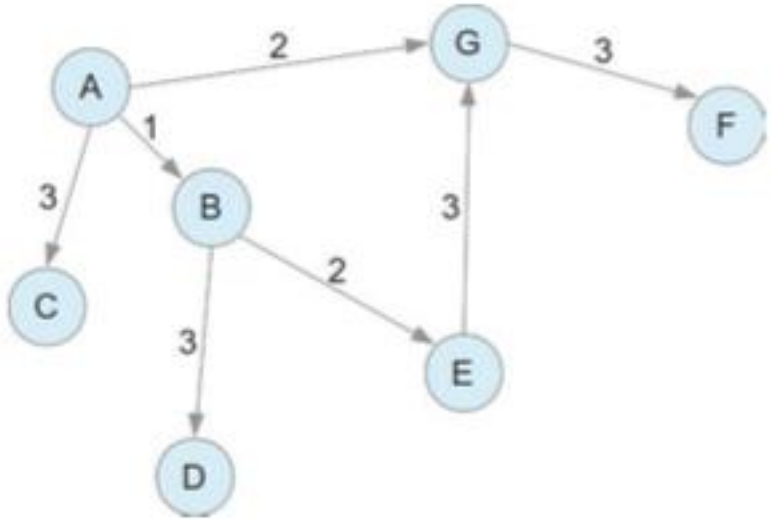
- ***Distribution sans maître : le protocole de bavardage***
  - Le « *protocole de bavardage* » (*Gossip protocol*) est une technique de distribution des informations basée sur la « *théorie des épidémies* »
    - Si une personne est contagieuse et qu'elle entre en contact avec d'autres personnes, ces dernières tomberont malades et contamineront elles-mêmes d'autres personnes à leur tour
      - Cela a rapidement un effet exponentiel
    - Les systèmes distribués fonctionnent de la même manière en utilisant le « protocole de bavardage » afin de transmettre de l'information
      - L'information transmise par bavardage peut être une information relative à la disponibilité d'un nœud dans le système

ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Distribution sans maître : le protocole de bavardage***
  - Dans un tel protocole, les nœuds entame périodiquement la conversation avec un « *voisin proche* », un « pair » choisi en général au hasard pour échanger de l'information
  - Peu de temps après, un nœud qui possédait au départ une information inconnue des autres nœuds communiquera cette information à tous les nœuds de ce système

ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Distribution sans maître : le protocole de bavardage***
  - Exemple : fonctionnement d'un protocole de bavardage



ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

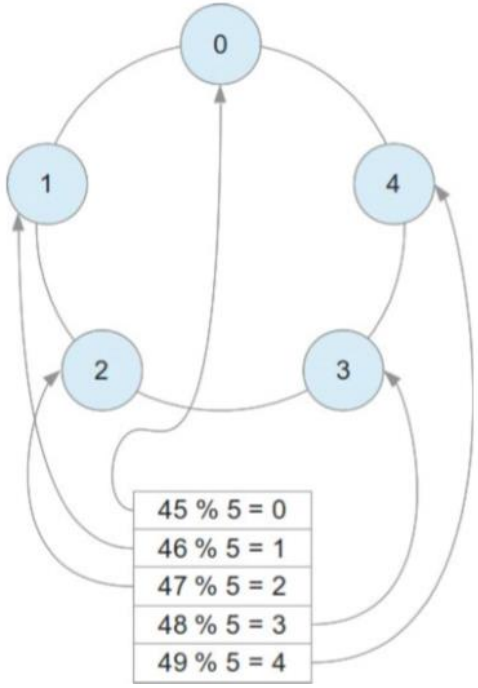
- ***Distribution sans maître : le protocole de bavardage***
  - Explication
    - Le nœud « A » a une information à communiquer au système
    - Pour cela, il va entamer la conversation avec un nœud choisi au hasard parmi l'ensemble de ses voisins proches, de ses pairs
      - Prenons la communication numéro « 1 » avec le nœud « B »
      - Cette communication s'établit dans les deux sens
        - Si le nœud « B » a aussi une information à communiquer, il l'enverra à « A » lors de leur conversation
    - Ensuite, à intervalles réguliers, chaque nœud contactera un autre
    - A la fin de ce processus, tout le système aura pris connaissance de l'information transmise au départ par le nœud « A »

ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Distribution sans maître : le hachage consistant***
  - Bâtir un système distribué implique également de partitionner les données pour les distribuer uniformément entre les nœuds
    - Il faut pour cela se baser sur la « *clé* » (*primaire*) de la donnée (*par exemple l'identifiant du document*) et trouver un moyen, à partir de cette clé, de distribuer les données
  - Dans un système sans maître, les clients doivent pouvoir être redirigés sur les machines qui contiennent les données demandées
  - Une solution relativement simple à cette problématique est la distribution des clés sur les différents nœuds avec un « *modulo* »
    - En général, il s'agit d'un « *hachage* » de la clé à l'aide d'un algorithme qui assure une bonne distribution du résultat et retourne une valeur entière
- ***Mémento :*** *L'opérateur « modulo » exprime le reste de la division entière : 25 modulo 5 = 0 (car le reste de la division de 25/5 donne 0)*

ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Distribution sans maître : le hachage consistant***
  - Exemple : distribution par hashcode



ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Distribution sans maître : le hachage consistant***
  - Explication
    - Nous voulons distribuer nos données de la façon la plus homogène possible sur ***5 nœuds***
    - L'algorithme d'attribution des données est basé sur un « *modulo* »
      - Si nous souhaitons créer un nouvel enregistrement, nous devons calculer un ***modulo 5*** par rapport à la valeur de l'identifiant qui retourne le numéro du nœud auquel nous allons attribuer l'enregistrement

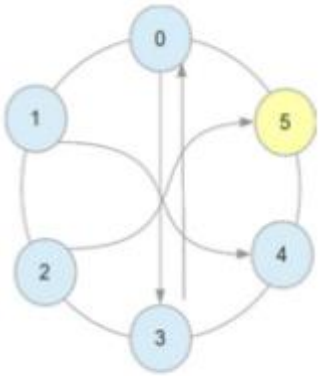
ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Distribution sans maître : le hachage consistant***
  - Question

    - ***Que se passera-t-il si nous ajoutons un nœud dans le système?***

ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Distribution sans maître : le hachage consistant***
  - Exemple : ajout d'un nœud dans le système



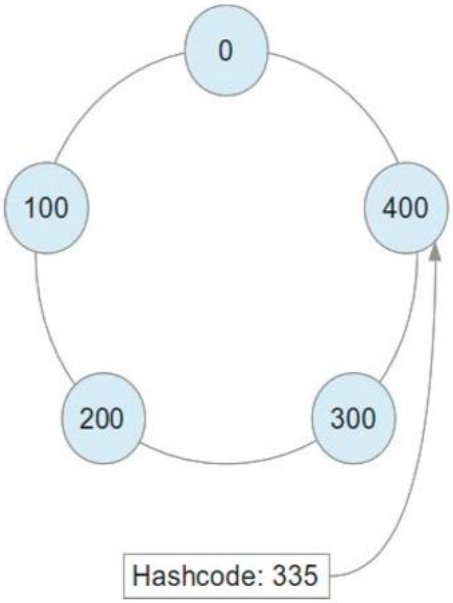
0 → 45 % 6 = 3
1 → 46 % 6 = 4
2 → 47 % 6 = 5
3 → 48 % 6 = 0
4 → 49 % 6 = 1

ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Distribution sans maître : le hachage consistant***
  - Explication
    - Il faut modifier notre « *modulo* » pour le faire correspondre avec le nouveau nombre de nœuds
    - Cela implique que l'attribution d'une majorité des enregistrements doit changer
      - Une réorganisation des données doit donc se faire sur tous les nœuds
    - C'est pour éviter cette problématique que la technique du « *hachage consistant* » (*consistent hashing*) a été développée

ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Distribution sans maître : le hachage consistant***
  - Exemple : distribution par hachage consistant



ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

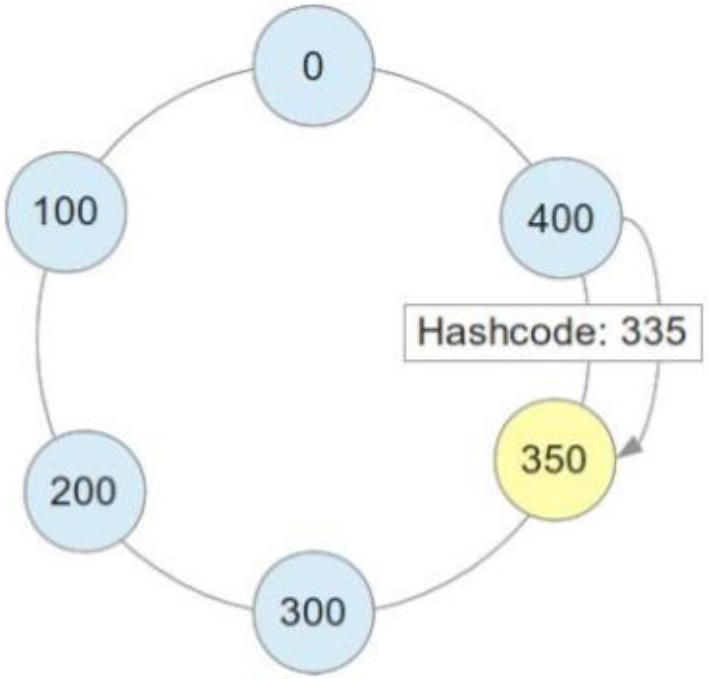
- ***Distribution sans maître : le hachage consistant***
  - Explication
    - Le « *hachage consistant* » a pour but d'attribuer des ***rangées de valeurs*** à chaque nœud du système
      - Conceptuellement, le « *système distribué* » est vu comme un « anneau » (*hash ring*) sur lequel chaque nœud occupe une position
    - Chaque nœud se voit attribuer une « *valeur de hachage* » caractérisant directement la ***rangée*** qui est ***hébergée***
    - L'algorithme de hachage génère un « *hash code* » à partir de la « *clé* », ensuite trouve le nœud dont la « valeur de hachage » est supérieure à ce « *hash code* » et y stocke la donnée
      - Un nœud contient donc toutes les clés inférieures à sa valeur de hachage et supérieures à la valeur du nœud précédent

ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Distribution sans maître : le hachage consistant***
  - Question

    - ***Que se passera-t-il si nous ajoutons un nœud dans le système?***

- ***Distribution sans maître : le hachage consistant***
  - Exemple : ajout d'un nœud avec hachage consistant



ARCHITECTURE ET SYSTÈMES DISTRIBUÉS

- ***Distribution sans maître : le hachage consistant***
  - Explication
    - Lorsqu'un nœud est ajouté à l'« *anneau* » il prend une « *valeur de rangée* » et scinde une rangée existante
    - L'attribution des « *clés* » ne sera donc perturbée que pour un seul nœud
      - Ainsi un nombre peu important de données sera déplacé du nœud faisant l'objet de la réorganisation de données
      - C'est l'avantage principal d'une distribution par « *hachage consistant* »