Introduction à NoSQL

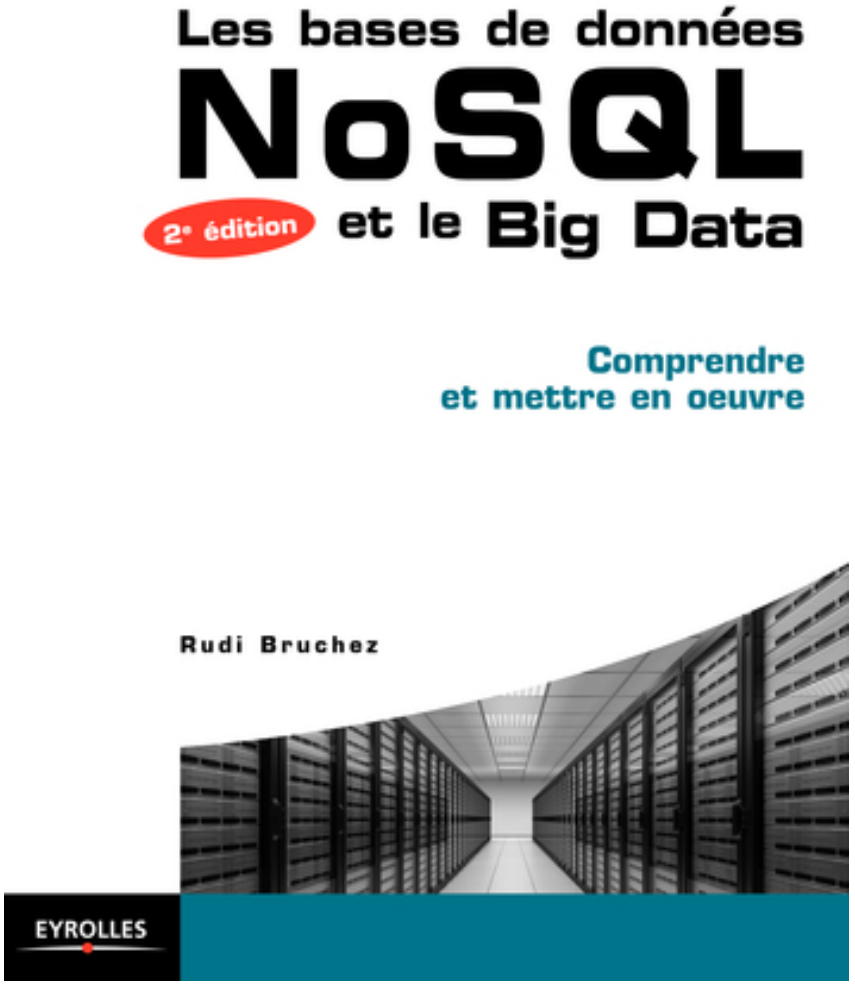
Data science
Master 2 ISIDIS

Sébastien Verel
verel@univ-littoral.fr
http://www-lisic.univ-littoral.fr/~verel

Université du Littoral Côte d'Opale
Laboratoire LISIC
Equipe OSMOSE

Bibliographie

Ce cours s'appuie très fortement sur ce livre :



Avant-propos

NoSQL = Not only SQL

- Jeune en France, moins aux USA
- Impulsé par les nouveaux besoins des entreprises du web
- Vieille opposition "sociale" en informatique ?
  Développeur approche ludique et libre
  vs. Administrateur approche protection
- Non ce n'est pas seulement l'opposition à SQL :
  expérimentations, autres modèles de données très simples,
  Nouveaux besoins, nouveaux outils !

Quelques remarques

Modèle relationnel

- Dominant depuis les années 1980
  - Quels sont les fondements de ce modèle ?
  - Qu'est-ce qui constitue son efficacité ?

Quelques remarques

Modèle relationnel

- Dominant depuis les années 1980
  - Quels sont les fondements de ce modèle ?
  - Qu'est-ce qui constitue son efficacité ?

Et NoSQL

- Totems NoSQL : Cassandra, MongoDB, Redis, HBase, Riak...
- Points communs : abondons du modèle relationnel
- Développement de nouvelles stratégies, nouveaux algorithmes, nouvelles pratiques dans la gestion de données

Système de gestion de (bases de) données

But

Organiser les données : optimiser la conservation et la restitution
Stoker et retrouver (sécurité, intégrité, cohérence, etc.)

Système de gestion de (bases de) données

Historique

- Année 1950, Modèle hiérarchique :
  - Relation parents-enfants : record, record type
  - Structure d'arbre : avantages et inconvénients connus
  - moteur IMS d'IBM (mission spatiale apollo)
- 1959, Codalsyl et le Cobol :
  - Définition d'un standard ISO
    - entreprises, universitaires, gouvernements,
  - Langage navigationnel :
    - pointeurs posés sur "entité courante"
  - Approche procédurale :
    - boucles, tests, etc.
  - Modèle de données réseau :
    - liens entre des articles par des "sets"
- 1970, Modèle relationnel...

Modèle relationnel

- Edgard Franck Codd (britannique),
  mathématiques, chimie,
  doctorat en informatique (univ. Michigan).
  Labo IBM San Jose, Californie
- "A relationnel Model of Data for Large Shared Data Banks", Communications of ACM, 1970.
- Système de relations basé uniquement sur les valeurs des données
- Manipulation à l'aide d'un langage de haut niveau implémentant une algèbre relationnelle

Modèle relationnel (suite)

- Manipulation à l'aide d'un langage de haut niveau implémentant une algèbre relationnelle
- Pas de préoccupation du stockage physique
  pas de pointeur par exemple, etc.
- Langage algébrique, déclaratif de haut niveau...
  On décrit le "comment", on décrit seulement le résultat : requête déclarative, puis moteur efficace côté serveur
- Deux parties : séparation gestion physique et logique

Historique des implémentations :

1974, Système R, sequel, (Lab. San Jose), index en B-tree
fin 1970, PostgreeSQL, (Berkeley)
1979, Oracle, (Lary Ellison)
1979, SQL/DS, DB2 (IBM)

Modèle relationnel

1985 : Les 12 régles de Cood (de 0 à 12...)

- Très performant : utilisation transactionnelle,
- Requête OLTP (OnLine transactional Processing) :
  Exemple typique, ERP, facture d'un client
- Requête OLAP (OnLine Analysis Processing)
  Parcourir grande quantité données pour calculer agrégats,
  Ici, optimisation de requêtes et index pas très efficaces
  Schéma en étoile ?
- Besoins statistiques :
  Reporting, tableau de bord, analyse d'historique voire prédictive
  Typiquement analyse de tendances

Big data : les fameux 3V (Gartner)

Big data : les fameux 3V (Gartner)

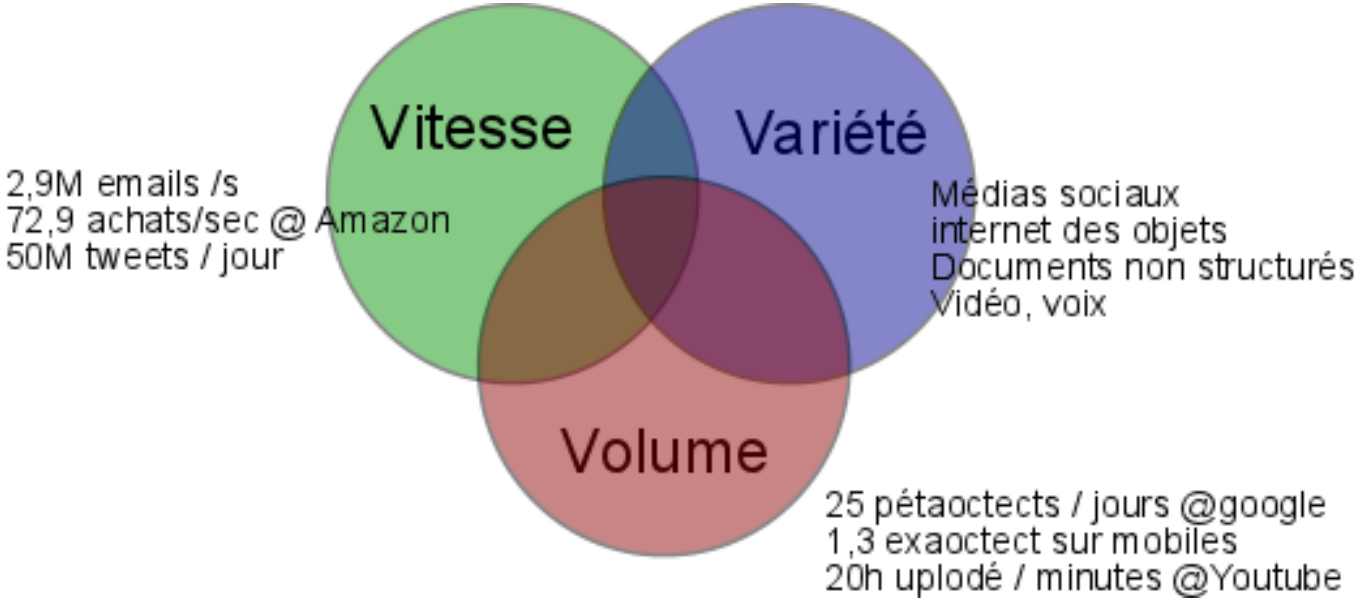


Schéma d'après "Big data et Machine Learning", Dunod, 2015.

Big data, volume des données

- Google : manipulation de grands volumes de données
  Moteur de recherche, maps, youtube, gmail, etc.
  2003, système de fichier distribué GFS, 2004, map-reduce.
- Doug Cutting, et Yahoo
  moteur de recherche Lucene/Nutch
  Implémentation de merge/reduce en java : HDFS, Hadoop
- 2004 présenté en 2006 :
  Système de gestion de données basé sur GFS : BigTable
- 2006, version libre (dév. soc. Powerset, langage naturel) : HBase
  Projet phare du monde NoSQL
  SGBD orienté colonnes, architecture distribuée
  Utilisé par : Ebay, Yahoo, twitter, abode, et surement encore plus demain

NoSQL from big data

- 2007, Amazon Dynamo :

  "Dynamo : Amazon's Highly Available Key-value Store", SOSP 2007.

  Entrepôt clé-valeur

  Architecture distribué sans maître

  Comme bigTable, Dynamo est un code propriétaire non disponible
- 2008/2009, Apache Cassandra,
  (A. Lakshman, P. Malik de facebook)

  Entrepôt orienté colonnes totalement décentralisé

NoSQL from big data

NoSQL, Une avancée ?

- SGBD Relationnel :
  Optimiser le stockage, excellente performance, intégrité
  Capacité de construire modèle de données, structure physique d'index
- MAIS : Informatique distribuée change la donne :
  Nouveaux besoins :
  web scale, recherche plein texte rapide, données semi-structurées, etc.

Tendance humaine : Résister au changement ou s'enthousiasmer...

Pour plus de détails, encore une fois, cf. bibliographie.

Une classification des moteurs : (I) par usage

- Amélioration des performances :
  Utilisation RAM, simplification du modèle de données (clé-valeur), distribution du traitement
- Assouplissement de la structure :
  Simplification structure de données (json, pas d'intégrité référentielle entre des tables, pas de schéma au niveau du serveur)
- Structures spécifiques :
  pour utilisation spécifique par exemple, les moteurs orienté graphe
- Volumétrie : un des buts recherchés !
  distribution du stockage et traitement
  distribution du stockage : plat par fichier, ou DB distribué
  distribution centralisée ou décentralisée

Une classification des moteurs : (II) par schéma de données

- Moteurs orienté clé-valeur :
  accès par clé de hachage
  fonctionnalités simplifiées, moins riche, excellente performances
  ex : Redis, Riak
- Moteurs orientés documents :
  basé sur format json : document (semi-)structuré
  ex : MongoDB, CouchDB, Couchbase Server, Riak (v2)
- Moteurs orientés colonnes :
  inspiré par BigTable, table de hachage distribué
  données ligne (id par clé) / colonne, gros volume
  ex : Cassandra, HBase
- ...

Une classification des moteurs : (II) par schéma de données

- Index inversés : moteur de recherche
  correspondance entre terme (contenu) et position dans un ans. de données
  ex : ElasticSearch, Solr (basé sur Lucene)
  Permet de rechercher sur structure json
- Structures particulières :
  Aucune autre catégorie
  moteur orienté graphes, représentation particulières, stockage natif, etc.

Points communs entre NoSQL

- Schéma implicite :
  schéma de données non prédéfini côté serveur,
  Application cliente structures les données
  Sauf exception : Cassandra v2
- Absence de relation :
  Pas de relation entre données, pas de référence entre éléments de 2 collections
  Sauf exception : orienté graphe (structurel), Hive avec jointure (fonctionnel)
- Le langage SQL !
  Souvent un langage déclaratif est utilisé
- Logiciel libre
  Pour ces qualités
  Malgré tout, vente additionnelle, ou de services associés

Principe du DB relationnel

BD relationnel

- Présupposé modèle mathématique,
  base de la représentation des données du SI
- Ensemble de données représentées sous forme de table
- Séparation totale entre organisation logique
  et implémentation physique (stockage)
- Données très structurées : attribut fortement typés
  Modélise la réalité dans ce modèle contraignant
- Avantage : Opérations algébriques
- Inconvénient : On "force" les données dans la structure

NoSQL

Approche pragmatique
  pour satisfaire un besoin concret, design par la problématique
  ex. Dynamo d'Amazon (panier d'achat)

Structure de données

BD relationnel

- Structure importante, métadonnées sont fixées préalablement
- Normalisation, pas de redondance
- métaphore : tableau avec entête,
  chaque donnée atomique stockée dans une cellule

Agrégat en NoSQL

Conception pilotée par le domaine (Eric Evans)

Agrégat = Collection d'objets liés par une entité racine

unité d'information complexe que l'on peut traiter

ex. : magasine avec pages, illustrations, etc.

BD relationnelle :

structure à plat, les agrégats sont crées dans la requête SQL

Centralité de la donnée

BD relationnelle

- Plusieurs besoins applicatifs
- Création des agrégats à la demande pour chaque applications
- Centralité de la donnée par rapport aux applications
- Martin Fowler : outils d'intégration, faire converger les applications vers les données structurées (le centre)

NoSQL

Agrégat déjà formé, toutes les types d'applications ne sont pas potentiellement satisfait

Structure de données des moteurs NoSQL

A part les exceptions (graphes Neo4J par exemple), et pour l'instant (évolution, évolution...)

- toujours des agrégats
- clé-valeur, orienté doc, colonne (sans type, ni limite)

Ce ne sont que des modèles

Attention aux deux tendances :

Conservatisme et attrait irréfléchi du nouveau

Modélisation

Modèle relationnel

- Modéliser sous forme de table **reliées** entre elles par des relations sur des clés
- Mener une réflexion préliminaire pour faire la BD relationnelle : première étape du développement (waterfall developpement $\neq$ agile)
- Meilleur la BD relationnelle, meilleurs seront les développements
- Difficulté de réingénieurie

Modélisation

NoSQL

Modèle schema-less :

Pas schéma de modèle défini a priori et donc modifiable

Evidement ne rien exagérer, ajouter une colonne en SGBDR n'est pas infaisable et pour les moteurs NoSQL les modifications ne sont gratuites, peuvent être voire couteuse (dépôt).

Pas de concept de manipulation de côté serveur
Pas de mécanisme d'abstraction des structures de données

Partitionnement des données

Mettre les données qui ont rapport entre elles ensemble

- Difficile lorsque les données sont stockées dans des tables différentes
- "Sharder" en NoSQL : partitionnement effectuer automatiquement par le système

SQL

- Toujours un langage déclaratif en NoSQL
- NoSQL = No Relationnal !
- Passage difficile entre SQL et le monde objet :
  Défaut d'impédance
- En NoSQL, l'impédance est réduite,
  cf. MongoDB en php,
  CouchDB et interface REST

Null

NULL

- SGBDR : valeur pour indiquer rien, ne peut pas être comparer
- NoSQL : pas nécessaire, pas de schéma

Cohérence des données, transactionnel

Transaction

Unité d'action respectant 4 critères

- Atomique : tout ou rien
- Cohérente : les données respecte les règles (consistance)
- Isolée : pendant la transaction
- Durable : l'état des données durablement inscrit

NoSQL

Cohérence prend un autre sens en NoSQL

Principe de l'agrégat en NoSQL :

plus besoin de transaction explicite

cf. modifications de plusieurs tables en SGBDR

Distribution synchrone ou asynchrone

- Synchrone : modification de tous les noeuds avant de passer à une autre action
- Asynchrone : modification de certains possibles avant la fin du autre action (possiblement gain de temps si peu d'interaction)

- En synchrone, garantie la cohérence des données (consensus !)
- Dans l'idéal tous les traitements devraient pouvoir participer au consensus, mais risque de défaillances

Assurer le consensus en tolérant les pannes n'est possible que sur un réseau synchrone (preuve existe)

Transaction distribuée

- 2 phases commits (2PC) :
  `Prepare_To_Commit`, `Ready_To_Commit` ;
  `Commit`, `Commit_Success`.
- 2PC utilisé en SGBDR, mais absence de tolérance aux pannes
- Rédibitoire pour le NoSQL
- Même si on peut considérer la majorité au lieu de l'unanimité (réduit les perf.)

Théorème CAP

Eric Brewer, Berkeley, 2000

Impossible pour un système distribué de garantir :

- **Cohérence** :
  Tous les noeuds sont à jour sur les données au même moment,
- **Disponibilité (Availability)** :
  La perte d'un noeud n'empèche pas le système de fonctionner
- **Tolérance au partitionnement (Partition tolerance)** :
  Chaque noeud doit pouvoir fonctionner de manière autonome

SGDBR assure C&P mais pas A,
ce qui peut être recherché par les NoSQL.

Durabilité de la transaction

- Parait naturelle, sauf en cas d'utilisation de la RAM
- Utilisation d'un journal : Write-Ahead Logging (WAL)

NoSQL et le big data et le décisionnel

Modes

- OLTP : dynamique et opérations fréquentes sur les données
- OLAP : traitement analytique en ligne

OLTP	OLAP
Mise à jour au fil de l'eau	Insertions massives
Recherche sélective	Agrégation de valeurs sur de grands ensembles de données
Type de données divers	Principalement donnée chiffrées et dates
Utilisation partagées entre mises à jour et lectures	Utilisation en lecture seule la plupart du temps
Besoins temps réels	Besoins de calculs set des données volumineuses, souvent sans exigence

Limitations du OLAP

Ancêtre du big data : contexte matériel différent, volume réduit

- Modélisation des données préliminaire :
  modèle en étoile,
  table des faits et table des liens
- Structuration :
  suppose outils importation et transformation (ETL)

OLAP et big data

Données n'est pas structurée :

On ne suppose pas de traitement à venir

Stocker l'intégralité des données "brutes",
sans transformation, sans remodélisation.

Hadoop MR :

Fonctions qui vont s'adapter à la structure de données

Fonctions au plus proche des données

Conception de Nathan Marz

Développe Apache Storm (traitement en temps réel)

Architecture Lambda :

- Données immuables, ajout uniquement (append only)
- Ressemble à un journal de log, trace l'historique de la donnée (jamais de perte, yes !)
- Traitement comme Hadoop fourni des vues au besoin

Les choix techniques du NoSQL

- 1/ Interface avec code client
- 2/ Architecture distribuée
- 3/ Big data analytique

1/ Interface avec code client

- difficulté SQL (déclaratif, impédance, ok ORM mais pas toujours performant)
- Meilleur intégration code serveur / code client (proche sérialisation)
- Très facile d'appeler les fonctions nosql pour mettre en objet :
  ex clé-valeur hachmap de redis/python

Fonctionnalité côté serveur

- En NoSQL (pour l'instant), pas de déclencheurs, de fonctions, procédures stockées
- Cela se passe coté client,
- ou avec map-réduce :

  map : une vue,

  reduce : calcul d'un agrégat

Protocoles d'accès aux données

Avec maintien d'une connexion

- Interfaces natives : communication questions/réponses par socket, utilisation port TCP
  ex : MongoDB
- Protocol Buffer (protobuf) :
  Définition de la structure du message dans un fichier
  Code fourni pour manipuler le message ensuite (compilation). Ex. Riak, HBase
- Thrift : Bibliothèque de développement de services
  Définition de la structure du message
  Définition des opérations supportées, puis on compile.
  Incorpore tous les niveaux de la création de service (Transport, Protocole, Processeur, Serveur)
- Apache Avro : même idée, mais plus souple
  Pas de compilation, utilisation de JSON pour le format

Protocoles d'accès aux données

- Trouver un exemple message envoyé par mongoDB
- Trouver un exemple de fichier `.proto`, un exemple d'utilisation
- Trouver un exemple de fichier de définition Thrift

Protocoles d'accès aux données

Interface REST (sans maintien de connexion)

Representational State Transfert (Roy Fielding)

- Architecture de transfert adapté au web
- basé sur HTTP : échange de documents et actions (verbes) voulues au serveur HTTP
- Utilisation de ce protocole de communication très courante
- Sans état : demande et réponse contenu dans l'appel

ex : CouchDB

Quels sont les 6 méthodes principales REST ?

2/ Architecture distribuée

Distribution des traitements de données

- Avec maître :
  Conserve la configuration du système,
  reçoit les requêtes des clients et
  redirige vers les bonnes machines
- Sans maître :
  Comment rediriger les requêtes ?
  Comment maintenir l'état du système ?
  Comment répartir les données ?

Distribution avec maître

- Réplication maître-esclave :
  Ecriture sur maître qui réplique données sur esclaves
  Autorisation ou non à lire sur les replica (attention cohérence)
- Sharding : partitionner les données sur plusieurs machines
  Gérer automatiquement par le système,
  par nature distribué,
  répartition de charge possible.
  $\Rightarrow$ Plus facile à faire lorsque maître-esclave

Problèmes avec la distribution maître/esclaves :

- Défaillance du maître,
- Goulot d'étranglement

Distribution sans maître

Techniques

- Gossip protocol : propagation virale
  - Engagement périodique d'une communication entre noeuds,
  - Sert à maintenir les infos sur l'état du système
  - Sert à échanger information de partitions
  - Sert à distribuer les demandes de lecture/écriture d'un client
- Table de hachage distribuée :
  - Répartition de l'espace de clé,
  - Redirection par les pairs
- Hachage consistant : voir tableau
- Hachage avec des VNodes : (bientôt Cassandra, Couchbase server)
  - BD découpée en beaucoup de blocs ($>$ nb noeuds)
  - Chaque noeud physique héberge beaucoup de blocks
  - A l'arrivée d'un noeud, des blocks sont transferés

Cohérence finale (1)

Définition

CAP : Cohérence dans système distribué impossible avec A&P
Cohérence finale dans sys distri asynchrone :
système ouvert (lecture/écriture) mais mais en place de techniques de réconciliation en cas d'incohérence (à la fin disons)

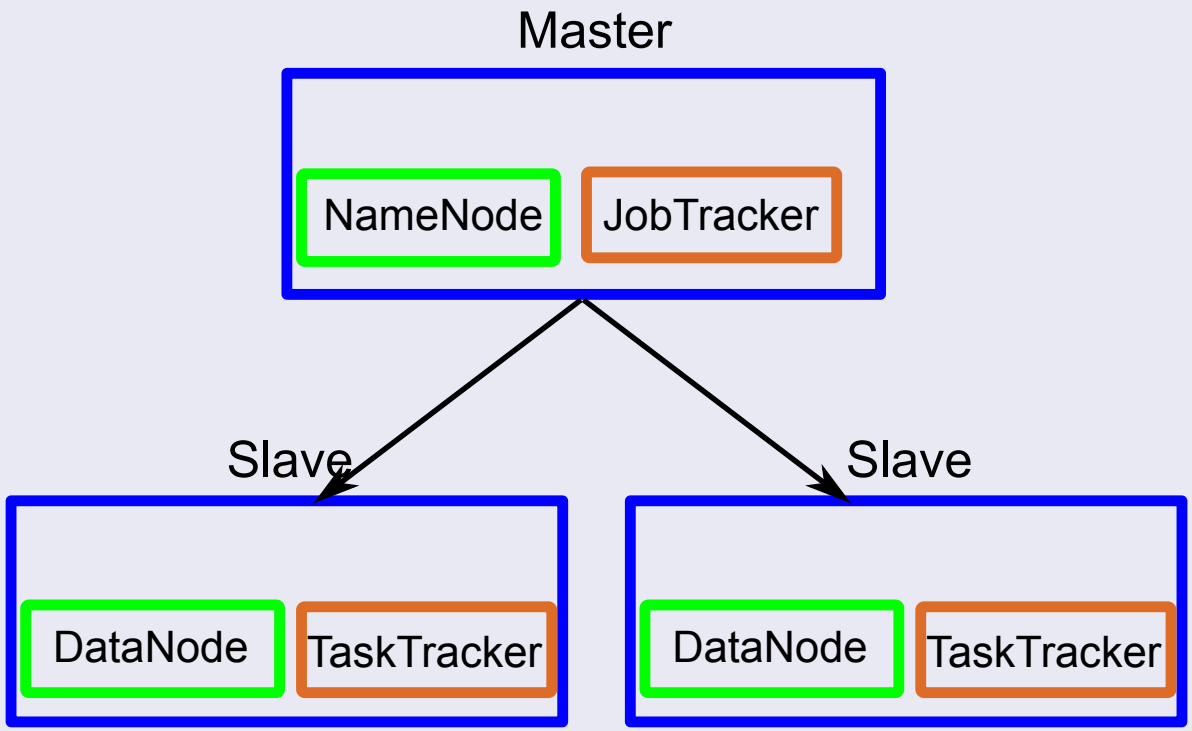
Incohérence ? Deux étapes nécessaires :

- Détecter les différences
- Gestion du conflit : Multi-Version concurrency control

Cohérence finale (2)

- Détecter les différences :
  - Notion d'Anti-entropie :
    - Maintenir de l'ordre lorsque la tendance est inverse
  - Technique : arbre de Merkel (cf tableau) :
    - Mécanisme de comparaison d'un volume de donnée
- Gestion du conflit : Multi-Version concurrency control
  - Risque dans le système :
    - "MaJ de la même donnée sur 2 machines en même temps"
  - Techniques :
    - Numéro de version (couchDB) et refuse si num n'est le dernier
    - Time stamps (BigTable) :
      Donnée identifiées par row, column, timestamps
      Attention aux horloges dont il faut prendre grand soin (NTP) !
    - Vector clocks (Riak) :
      Création d'un vecteur (node, version) attaché aux données qui permet ensuite de voir et gérer les conflits

3/ Big data analytique : Schéma Hadoop 1



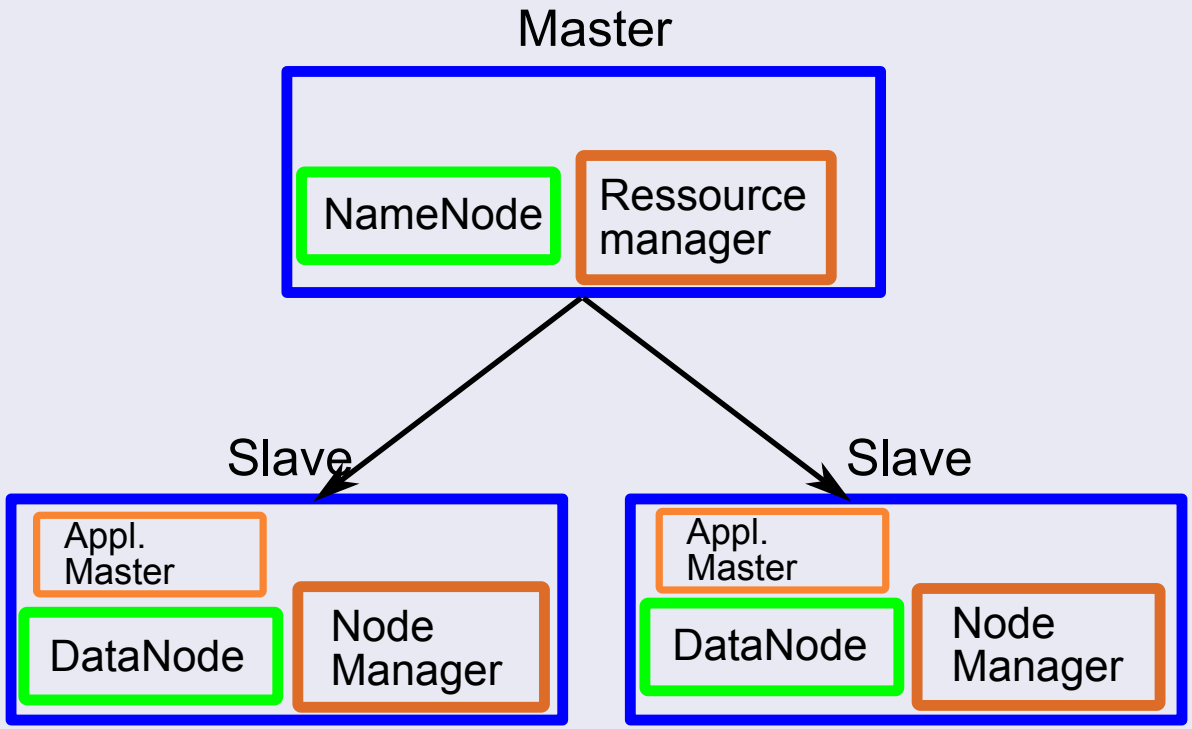
Pour HDFS : NameNode, DataNode
Pour MapReduce :

JobTracker : reçoit demande client et communique avec le NameNode, gère l'ensemble de déroulement de l'exécution en communicant avec les TaskTracker

TaskTracker : exécute un map/réduce.

3/ Big data analytique : Schéma Hadoop 2 - Yarn

Une architecture générique (map/reduce, ou autres à définir)



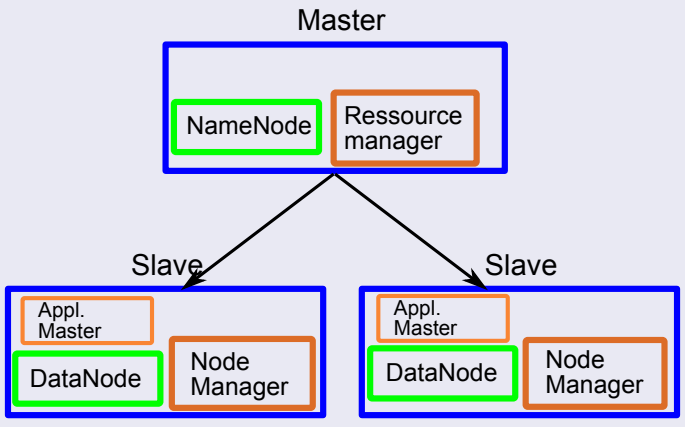
Ressource Manager : Scheduler (planifie les taches) et Application Manager (ordonnancer des taches, s'occupe de l'exécution, négocie avec les Application Masters)

Node Manager : exécution local du travail,

"responsible for launching the applications, containers, monitoring their

3/ Big data analytique : Schéma Hadoop 2 - Yarn

Une architecture générique (map/reduce, ou autres à définir)



Ressource Manager : Scheduler (planifie les taches) et Application Manager (ordonnancer des taches, s'occupe de l'exécution, négocie avec les Application Masters)
Node Manager : exécution local du travail,
"responsible for launching the applications, containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager." from cloudera.

Travaux

Pendant 30 minutes, faites une recherche sur l'un des sujets suivants :

- HBase
- CouchDB et Couchbase Server
- MongoDB
- Riak
- Redis
- Cassandra
- Neo4j

Puis, Exposer au reste du groupe.

Travaux

Quelques suggestions de question :

- Architecture globale
- Type de donnée manipulée
- Algorithmes/Technologies utilisées
- Mode de distribution
- Cohérence
- Langage de développement
- Protocole
- Licence, date
- Points forts
- Contextes d'utilisation

Fournir des exemples de codes raisonnés d'utilisation.