# NOSQL et Map Reduce

08/01/2013

# Intro: Système NoSQL

- Tous les problèmes de gestion de données ne sont pas exclusivement résolus à l'aide d'un SGBD relationnel traditionnel.
- NoSQL = Not Only SQL
  - Dans ce cas SQL signifie SGBD traditionnel

# Propriétés d'un SGBD

- Stockage et Accès à des masses de données persistantes
  - Masses: TO, PO, ...
  - Persistence : assurée par un système de fichier

- Propriétés du stockage et de l'accès
  - Efficient : Performance, temps de réponse, débit
    - temps de réponse borné prévisible.
  - Fiable : assurée par un mécanisme journalisation (redo)
  - « Convenable »
    - dans le sens facile d'accès, simplifie l'accès aux données
    - modèle de données simple, langage déclaratif, transaction
  - Multi utilisateur : transaction
  - Sûr : les données sont toujours cohérentes vis-à-vis des contraintes exprimées dans le schéma.

# NoSQL : alternative au SGBD relationnel traditionnels

- Avantages
  - Schéma flexible
  - Rapide à installer, faible coût (licence)
  - Scalabilité
  - Cohérence relâchée → meilleure performances et disponibilité
- Inconvénients :
  - Pas de langage de requêtes déclaratif
    - donc davantage de programmation
  - Cohérence relâchée → moins de garanties

# Exemple n°1 : Log de serveur web

- Données : un fichier de log. Une ligne contient :
  (User, URL, timestamp, complement-info)
- Chargement des données dans un SGBD
  - Nettoyer les données
  - Extraction
  - Vérification
  - Spécifier le schéma
- Chargement dans un système NoSQL
  - Directement sans faire les opérations préliminaires ci-dessus.
  - Contrepartie : les opérations de nettoyage, extraction, vérification, spécification de schéma devront être faites **pendant l'exécution**, mais seulement sur la **portion** des données qu'on manipule, pas sur toutes les données.

# Exemple de requêtes

- Requête 1
  - Trouver les enregistrements selon un critère : User ou URL ou timestamp
  - Traitement facilement parallélisable
  - Bien adapté aux solutions NoSQL
- Requête 2
  - Trouver les paires d'utilisateurs qui accèdent la même URL
    - Auto jointure
    - Moins évident à paralléliser
- Requête 3
  - Age Moyen des utilisateurs accédant la même url
  - Besoin de cohérence assez faible.

# Exemple 2 : Graphe social

- Données:  schéma composé de 2 tables
  - Profile(u, nom, age, genre)
  - Lien(u, v)  u et v sont amis
- Requêtes
  - Trouver les amis des amis de l'utilisateur u1
  - Plus généralement : traversé d'un graphe
    - pas exprimable en SQL
      - Rmq: SQL  se limite aux traversées d'arbre
        - » Cf la clause *connect by* d'oracle

# Catégories des systèmes NoSQL

- MapReduce : OLAP
- KV Stores : OLTP
- Document stores
- Graph database systems
- Remarque sur les column stores
  - Le type de système appelé *column store* n'est pas une catégorie de système NoSQL mais une solution pour organiser les données relationnelles
  - cf. VectorWise

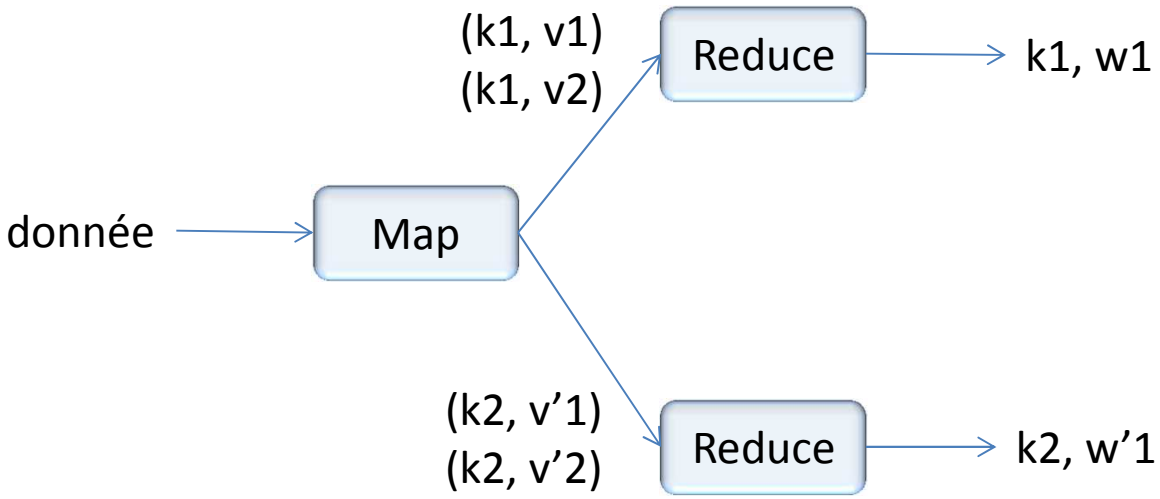
# Map Reduce (MR)

- Origine de Map Reduce (MR)
  - Google puis open source Hadoop
- Pas de modèle de données
  - les données sont dans des fichiers
  - Systèmes de fichiers : GFS ou HDFS
- L'utilisateur défini  plusieurs fonctions
  - map()
  - reduce()
  - reader() pour lire un fichier et construire des enregistrements
  - writer() pour écrire les enregistrements dans un fichier
  - combine()
- Un système Map Reduce apporte
  - un moteur pour traiter les données
  - la tolérance aux pannes  transparente pour l'utilisateur
    - Reprise partielle des sous taches défectueuses

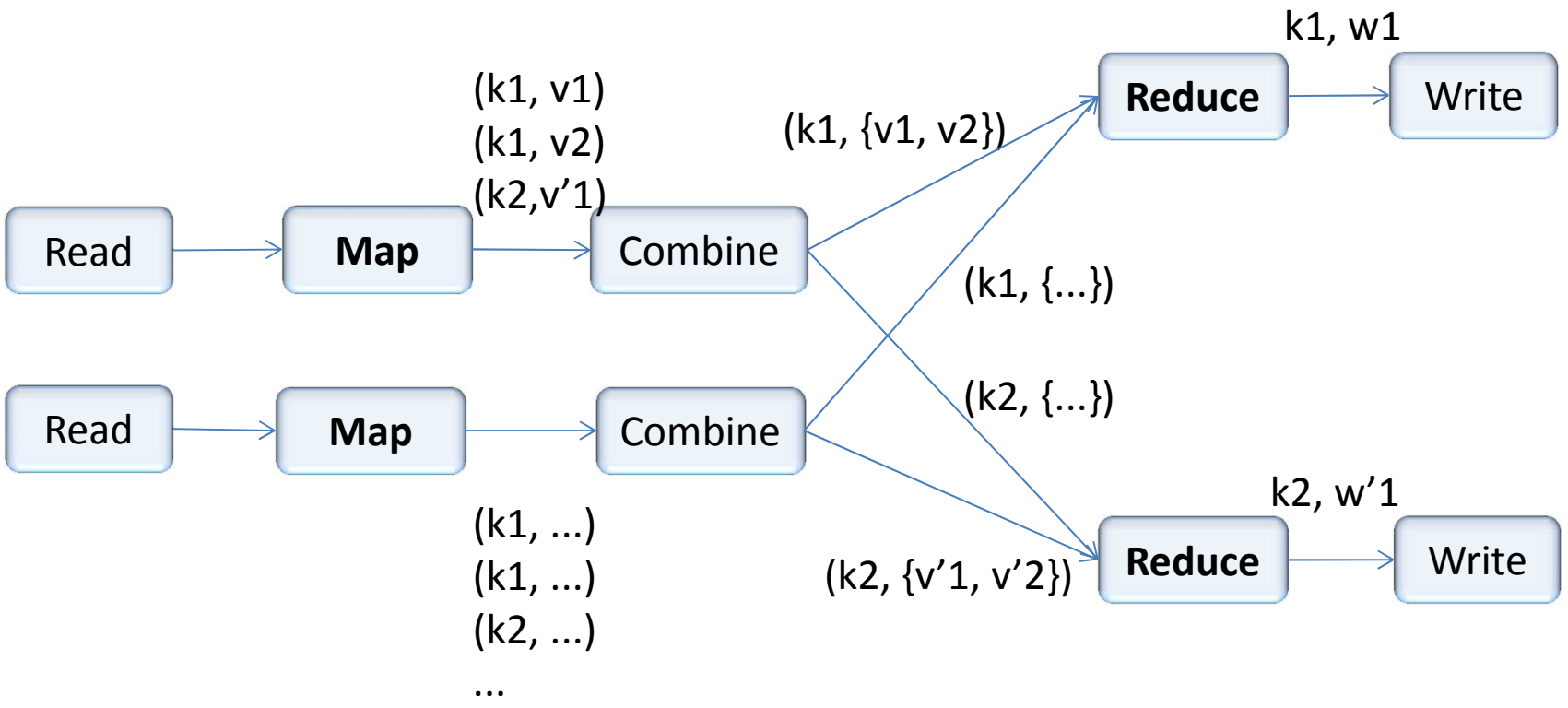
# Principe de Map Reduce

- Traite des données quelconques
  - Fichier: liste d'éléments (un élément par ligne)
- Map : diviser le problème en sous problèmes
  - Map(élément) → 0 ou n couples (clé, valeur)
- Reduce : est évalué pour chaque sous problème
  - Reduce (clé, liste de  valeurs) → 0 ou n élément

# Architecture fonctionnelle de Map Reduce

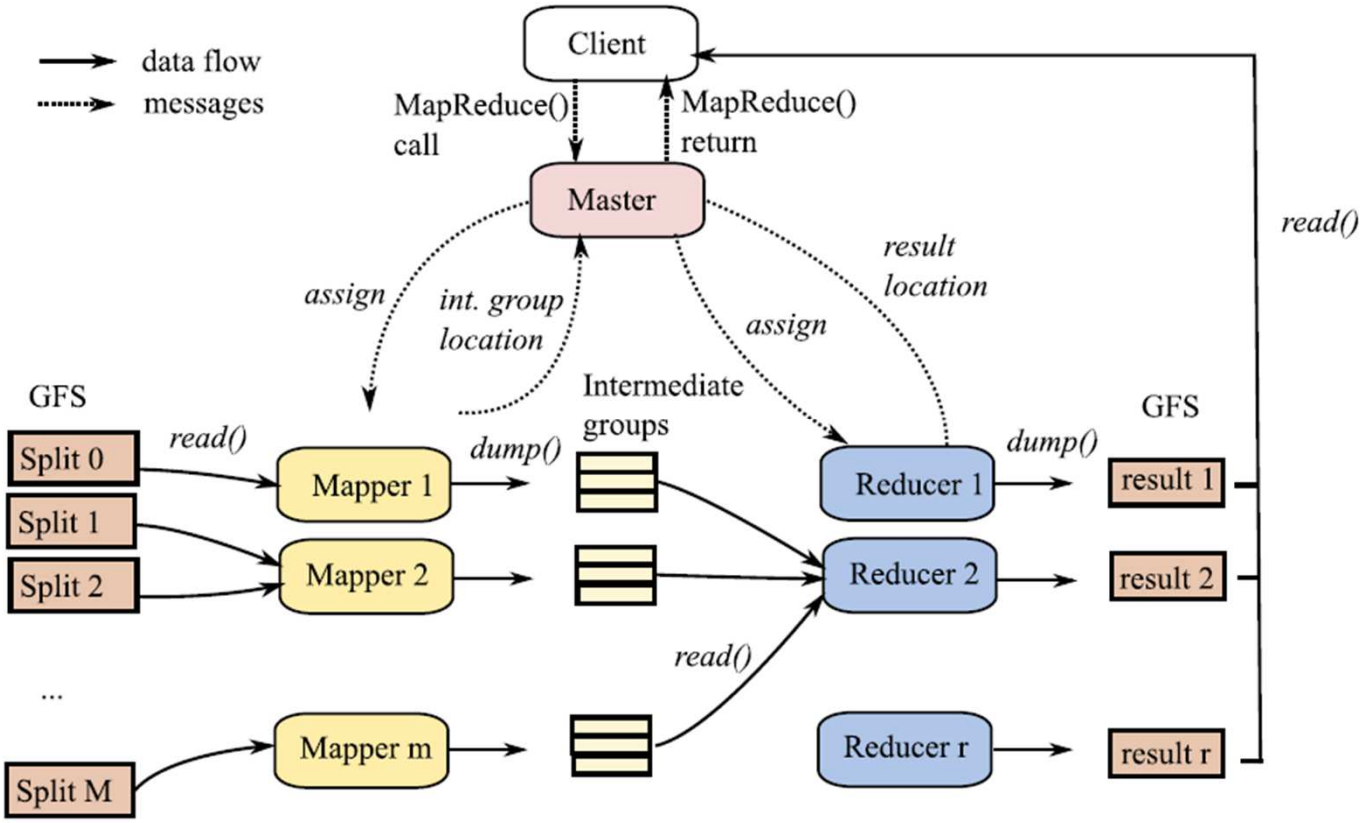


# Architecture Générale




Combine: prétraitement (regroupement, tri, ...) avant d'invoquer la fonction reduce

# MR : Modèle d'exécution



# Execution MR dans Hadoop

- Etape 1 : Reader, Map, Combine
  - Lire les fichiers d'entrée, pour chaque élément lu
    - Produire une liste de couples
  - Combine (avec tri local)
    - Regrouper les couples par clé
- Etape 2 : Shuffle : Transmission des couples
  - répartir les couples sur plusieurs machines
    - Par hachage ou trié
- Etape 3 : Merge, Combine, Reduce, Write
  - Fusionner les données pour avoir une seule liste de valeurs par clé
    - Fusion en plusieurs passes si nécessaire (selon ram dispo)
    - Invoquer Combine à chaque passe
  - Invoquer reduce pour chaque valeur de clé puis écrire le résultat dans un fichier

# Exemple 1

- Données

  D (clé, champ)
- Requête (grep)
  - Select * from D where champ like ‘%xyz%’
- MR
  - Map( ligne)
    - (clé, champ) ← split(ligne)
    - If ( champ contient ‘xyz’ )
    - then ajouter (clé, champ) en sortie
  - Pas besoin de reduce()

# Exemple : word count

- Afficher le nombre d'occurrences de chaque mot d'un fichier
  - Fichier très grand
- Données
  - un fichier texte
- Map
  - Argument d'entrée : une ligne du fichier
  - Retourne:  une liste de couples (mot, null)
  - Corps:
    - Extraire les mots de la ligne
    - Construire ($mot_1$, null), ($mot_2$, null), ….
- Reduce
  - Argument d'entrée : (mot, liste)
    - La liste est {null, null, …}
  - Retourne le couple (mot, longueur de la liste)

# Exemple 2

- Données
  - Document(URL, contents)
  - Ranking(pageURL, pageRank, avgDuration)
  - UserVisit(sourceIP, destURL, visitDate, adRevenue, userAgent, countryCode, languageCode, search Word, duration )

# Requetes d'analyse : sélection

- Sélection
  - Select pageURL, pageRank
  - From Rankings
  - where pageRank > x
- MR
  - Map(ligne de Ranking)
    - (pageURL, pageRank, avgDuration) ←Split(ligne)
    - If pageRang > X
    - Then ajouter (pageURL, pageRank) en sortie
  - Pas de reduce

# Requête d'analyse : agrégation

- Agrégation
  - Select sourceIP, sum(adRevenue)
  - From UserVisit group by sourceIP
- MR
  - Map
    - Ajouter (sourceIP, adRevenue)
  - Combine
    - Regrouper par sourceIP et somme des adRevenue
  - Reduce :
    - Somme des adRevenue

# Requête d’analyse : jointure

- Jointure
  - Select sourceIP, avg(pageRank), sum(adRevenue)
  - From Ranking r, UserVisit v
  - Where r.pageURL = v.destURL
  - And visitDate between jan-2011 and jan-2012
  - Group by sourceIP

# Jointure avec MR

- Etape 1/2 :
  - Sélectionner les visites dans l'intervalle de date
  - Jointure avec Ranking

  - Entrée: le contenu des 2 fichiers Ranking et UserVisit
  - Map
    - si la ligne est une visite alors filtrer selon la date
    - Ajouter en sortie des couples (K,v)
      - Clé composé de (destURL,'V') pour les visites
        - » Et de (pageURL,'R') pour les rankings

# Jointure avec MR

- Shuffle
  - Répartir par hachage du **préfixe** de la clé
    - Seulement l'URL sans le tag R ou V
- Reduce
  - Tri par URL croissant
  - Séparer les R et les V
  - Jointure
  - Sortie:  contruire (k, v)
    - k = sourceIP
    - v = (pageRank, adRevenue)

# Jointure avec MR

- Etape 2/2
  - Map() = id()
    - car la clé est déjà celle souhaitée
  - Reduce
    - Moyenne des pageRank et somme des adRevenue

# Extensions de MR

- MR online
  - Éviter de matérialiser les résultats intermédiaires
  - Enchainer plusieurs phases MR
- MR pour la fouille de données
  - Mapping entre le langage R et Hadoop
  - Voir le système Ricardo

# Langage de requête pour MR

- Langage déclaratif pour MR
  - Hive QL : syntaxe proche de SQL
  - Pig Latin : syntaxe algébrique
  - Optimisation de requêtes
    - HadoopDB : remplace HDFS par PostgreSQL
      - Hadapt : fragmentation des données par hachage
      - Hachage référentiel
        - » en fonction d'une clé étrangère
        - » Possibilité d'utiliser une clé étrangère indirecte
    - Hadoop++
      - Index et jointure parallèle
      - Extensibilité : Implémenté en tant que fonction UDF

# Biblio

- J. Widom : introduction to database systems
  - Open classroom, Stanford Univ
- Webdam : projet INRIA+ livre
- Hadoop
- Hadoop DB
- Hive
- Pig
- Ricardo
- Shark (AmpLab Berkeley)