



Bases de données NoSQL

Raja CHIKY
raja.chiky@isep.fr
2013-2014

Plan

- ▶ Introduction
- ▶ Concepts de base autour des BDs distribués
- ▶ MapReduce
- ▶ Bases de données clé-valeur
- ▶ Bases de données orientées colonne
- ▶ Bases de données orientées document
- ▶ Bases de données orientées graphe





Introduction

Introduction

- ▶ Des entreprises face à de nouveaux usages
 - ▶ Une augmentation exponentielle des volumes de données à traiter
 - ▶ La prise en compte des données en tant que ressources stratégiques
 - ▶ Des technologies en constante évolution
 - ▶ Accroissement des capacités de calculs et d'analyse
 - ▶ Coût de stockage moins onéreux
-
- 
- L'émergence du Big Data
- ▶ Comment mettre à profit des ressources jusqu'alors inexploitées?
 - ▶ Comment mettre à profit des sources disponibles ailleurs?

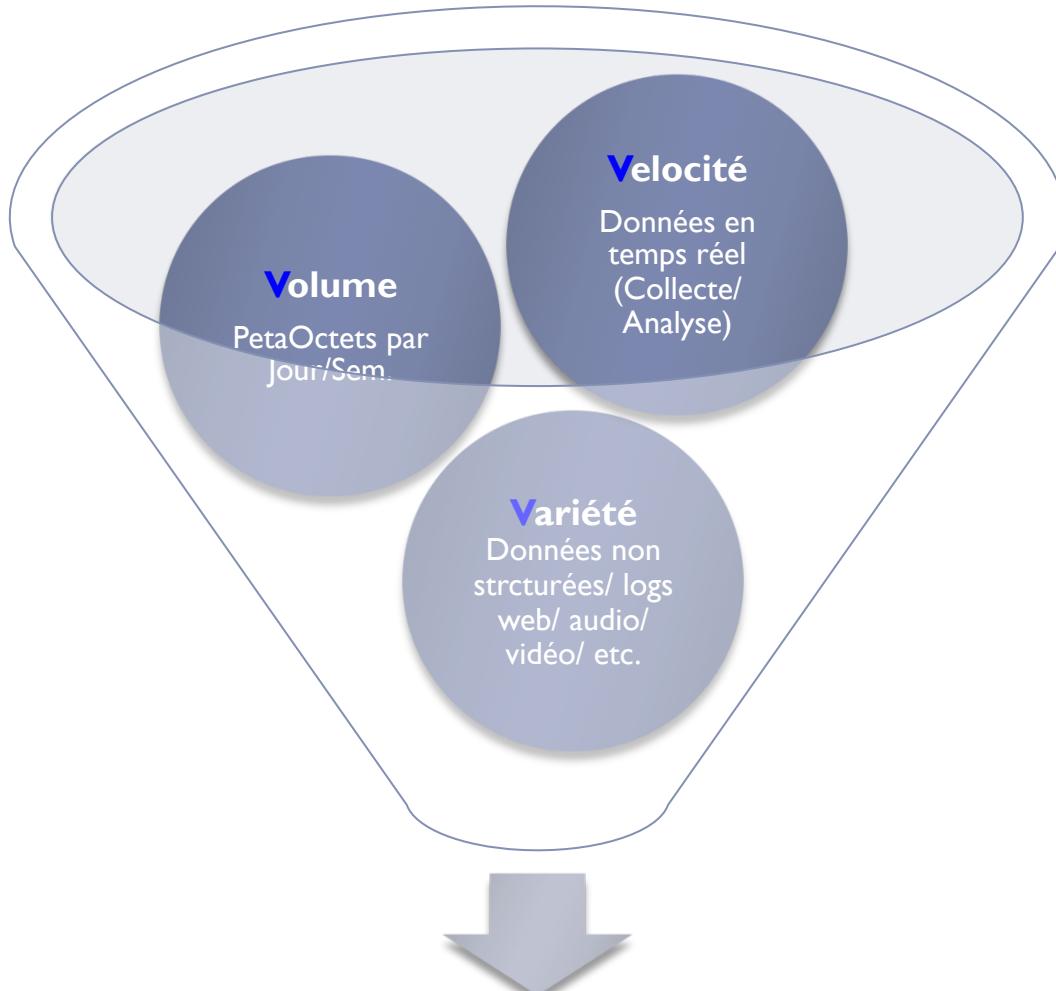


Big Data: exemples

- ▶ **Google**
 - ▶ 50 trillions de pages sur le web (printemps 2013)
 - ▶ 1 trillion = 1000 milliards
 - ▶ > 5 milliards de requêtes chaque jour
- ▶ **Twitter:**
 - ▶ >200 millions tweets par jour
- ▶ **AT&T**
 - ▶ 30 PB de données par jour



Big Data: 3V

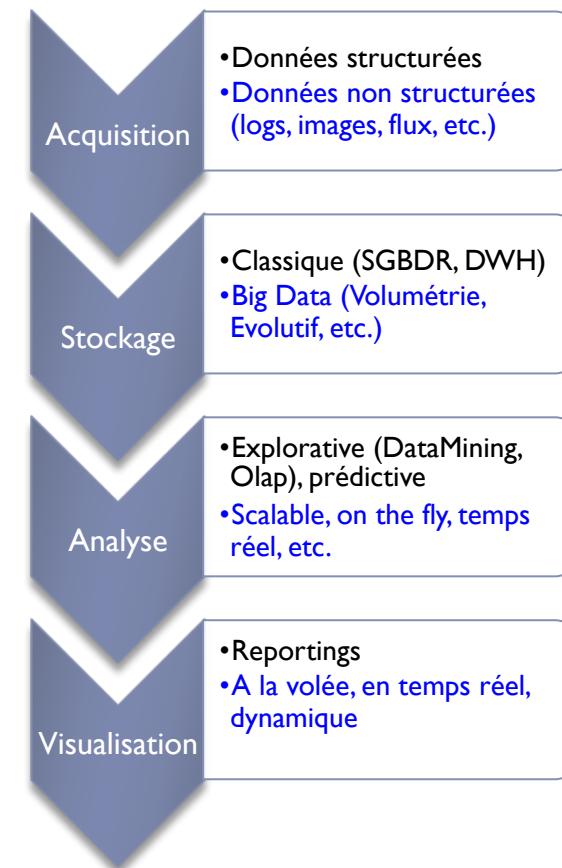


En extraire la **Valeur**



Défis du Big Data

- ▶ Appréhender une croissance exponentielle du volume de données
- ▶ Maîtriser la variété des données et en extraire l'information pertinente (la valeur)
- ▶ Traitement temps réel des informations
- ▶ Rendre l'information accessible au plus tôt
- ▶ Anticiper et faciliter la prise de décision



Exemple simple: Google-Grippe

[d'accueil de Google.org](#) (en
is)

[de la dengue](#)

[de la grippe](#)

ueil

[ectionnez un pays/territoire](#) ▾

[ment ça marche ?](#)

!

[agation du virus](#)

ès élevée

avée

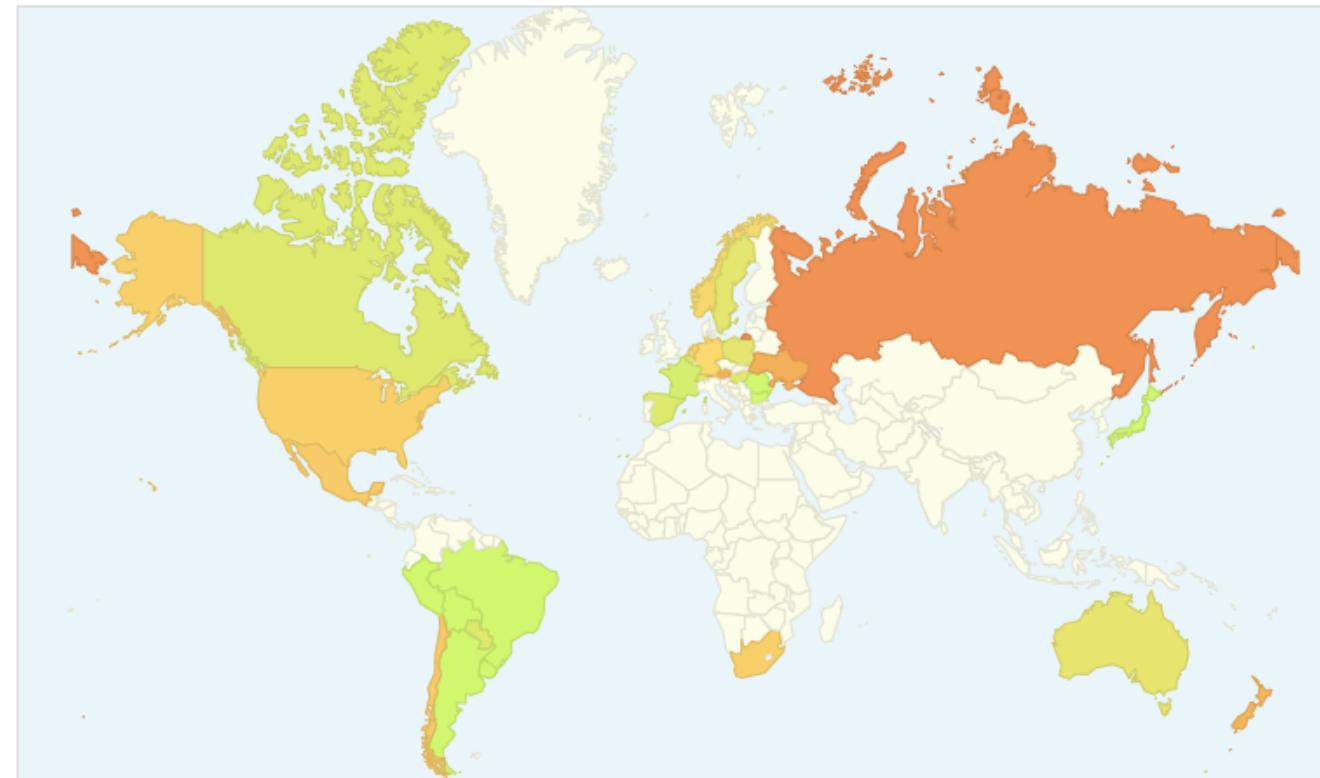
modérée

isse

nimale

Suivez l'évolution de la grippe dans le monde entier

Certains termes de recherche semblent être de bons indicateurs de la propagation de la grippe. Afin de vous fournir une estimation de la propagation du virus, ce site rassemble donc des données relatives aux recherches lancées sur Google. [En savoir plus »](#)



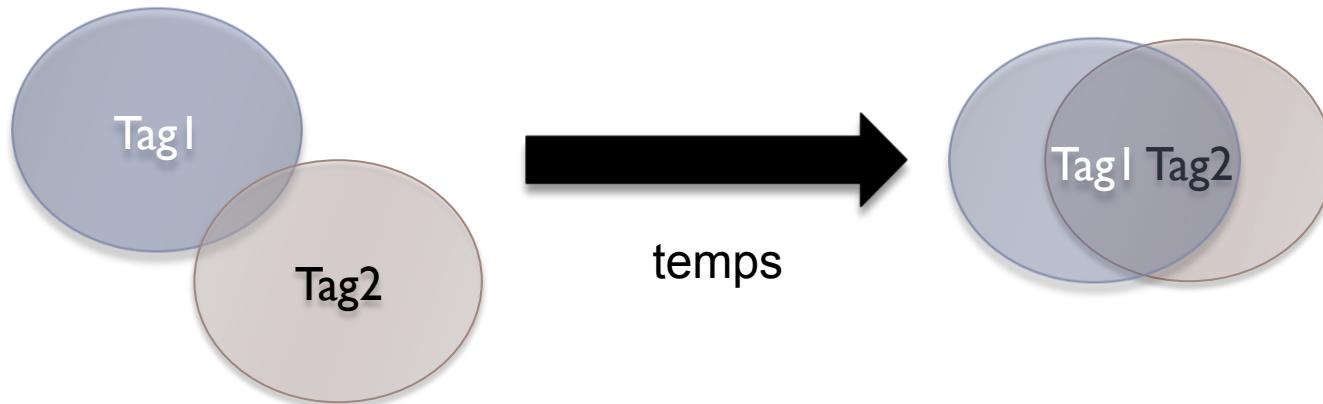
<http://www.google.org/flutrends/>

Exemple 2: extraction de tendances

- ▶ Extraire les tendances dans les flux textuels
- ▶ Un grand volume de données bruitées et non structurées
- ▶ Exemple de tendances

#benoitXVI #demanion

#armstrong #dopage







Concepts de base autour des BD's distribuées

Extensibilité – Sharding - MapReduce

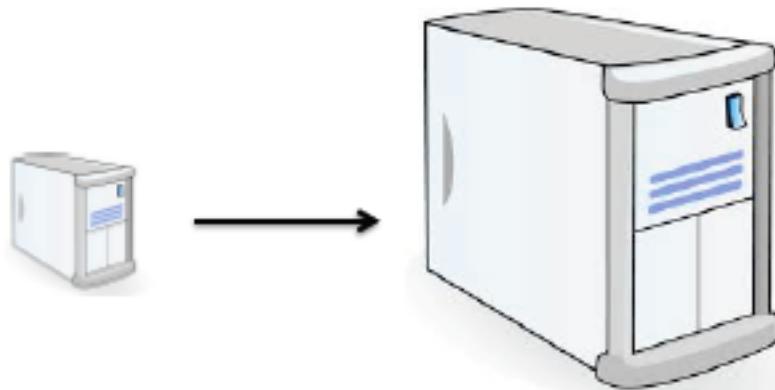
Extensibilité (scalability)

- ▶ L'extensibilité est la propriété d'un système, d'un réseau ou d'un processus qui témoigne de sa capacité à gérer des charges de travail importantes en toute souplesse ou à être agrandi sans difficulté
- ▶ Deux types:
 - ▶ Verticale
 - ▶ Horizontale



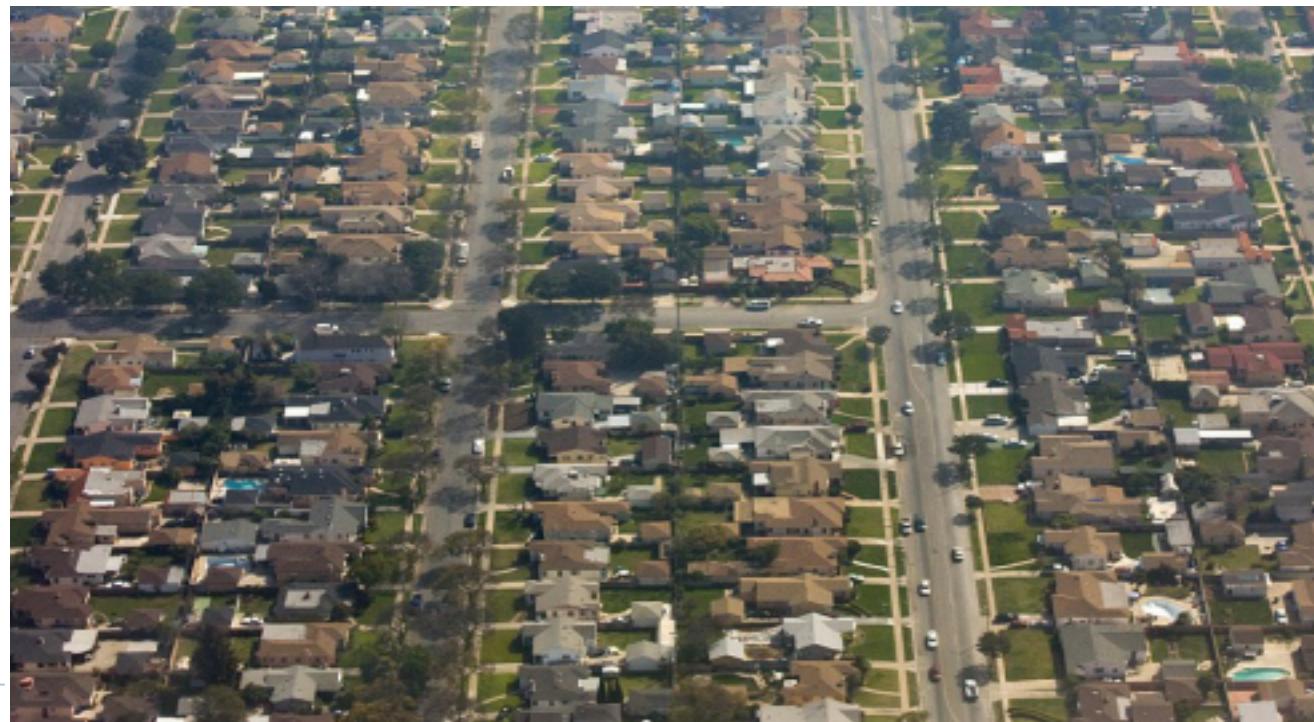
Extensibilité verticale

- ▶ En pratique
 - ▶ Remplacer les serveurs par des machines plus puissantes
- ▶ Coût important
- ▶ Mise en place facile



Extensibilité horizontale

- ▶ En pratique:
 - ▶ Ajout de machines ordinaires
- ▶ Coût moins important
- ▶ Mise en place plus délicate



Monter en charge un SGBDR

- ▶ Problèmes de passage à l'échelle quand le jeu de données est trop volumineux
- ▶ SGBDR n'ont pas été conçues pour être distribués
- ▶ Solutions de bases de données distribuées (multi-nœuds)
=>«passage à l'échelle horizontale»
- ▶ Différentes approches:
 - ▶ Maître-esclave
 - ▶ Partitionnement (sharding)



Monter en charge un SGBDR

▶ Maître/esclave

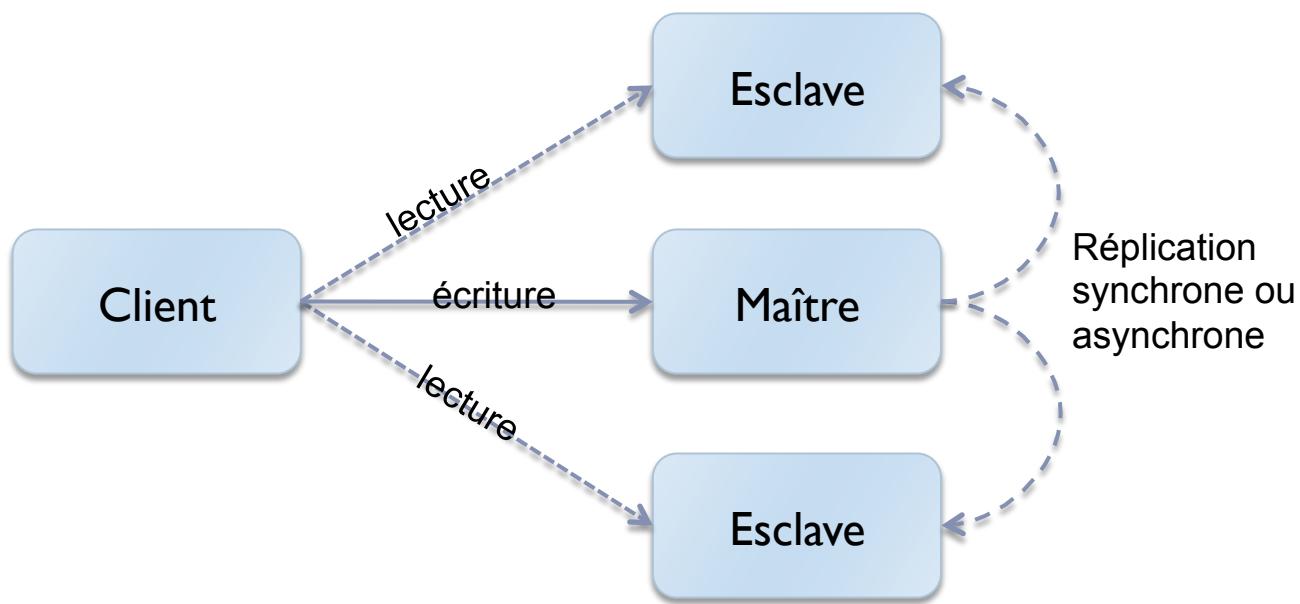
- ▶ Toutes les écritures sont envoyées au maître
- ▶ Les lectures peuvent être incorrectes tant que les écritures n'ont pas été propagées
- ▶ La volumétrie des données peut poser des problèmes si le maître doit dupliquer les données aux esclaves

▶ Partitionnement

- ▶ Efficaces pour lectures et écritures
- ▶ Manque de transparence, les applications doivent s'adapter au partitionnement
- ▶ Perte des contraintes référentielles (relations) et jointure entre les partitions

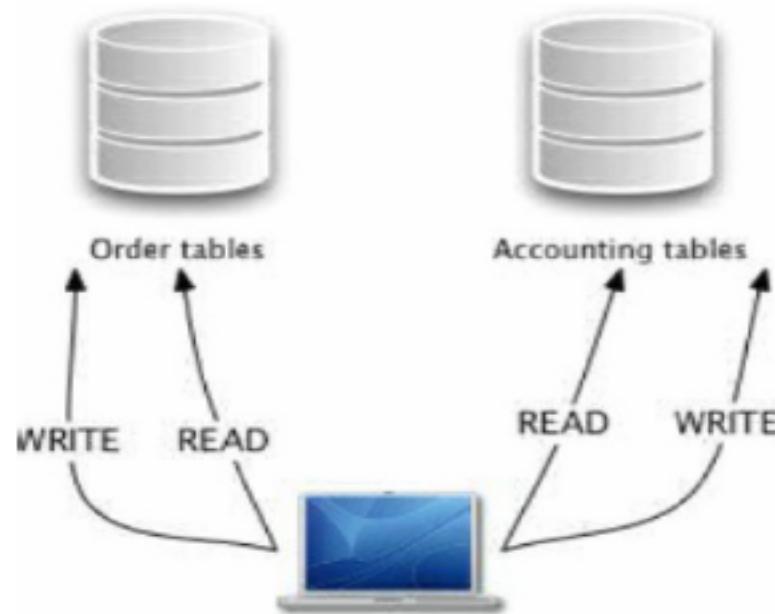


Maître/esclave

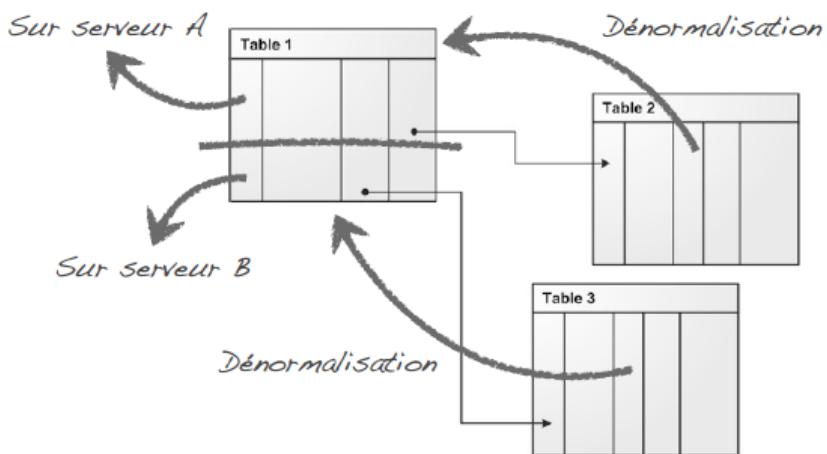
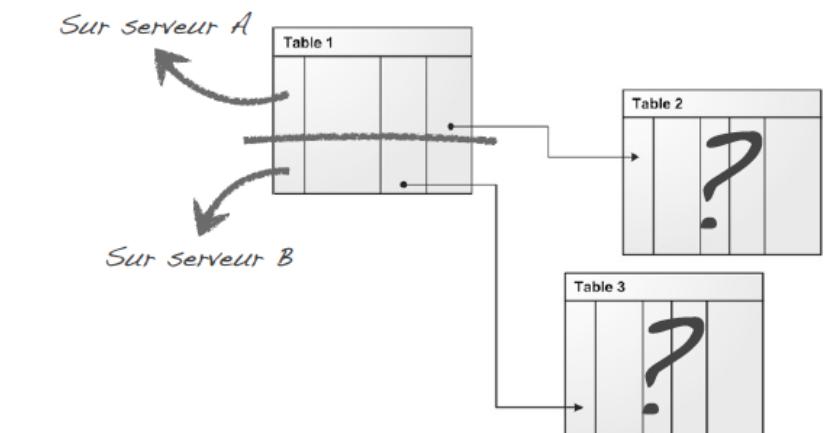
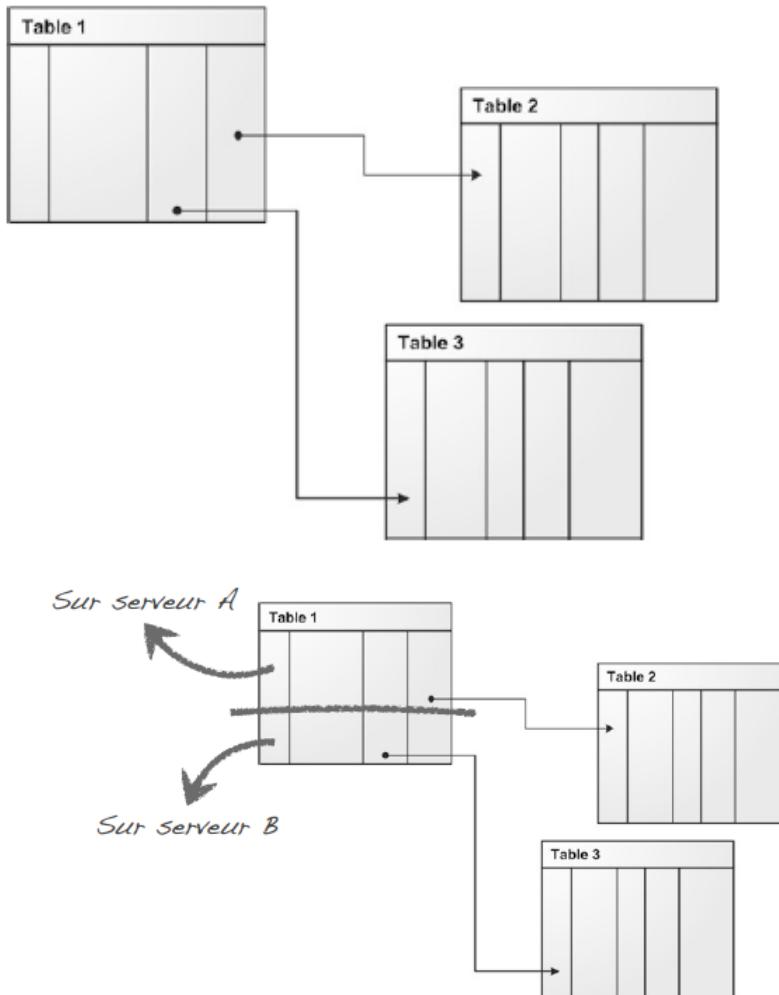


Partitionnement vertical

- ▶ Vertical
 - ▶ Les serveurs stockent différentes tables d'une base de données



Partitionnement horizontal



On perd alors beaucoup de l'intérêt du relationnel !

Partitionnement horizontal

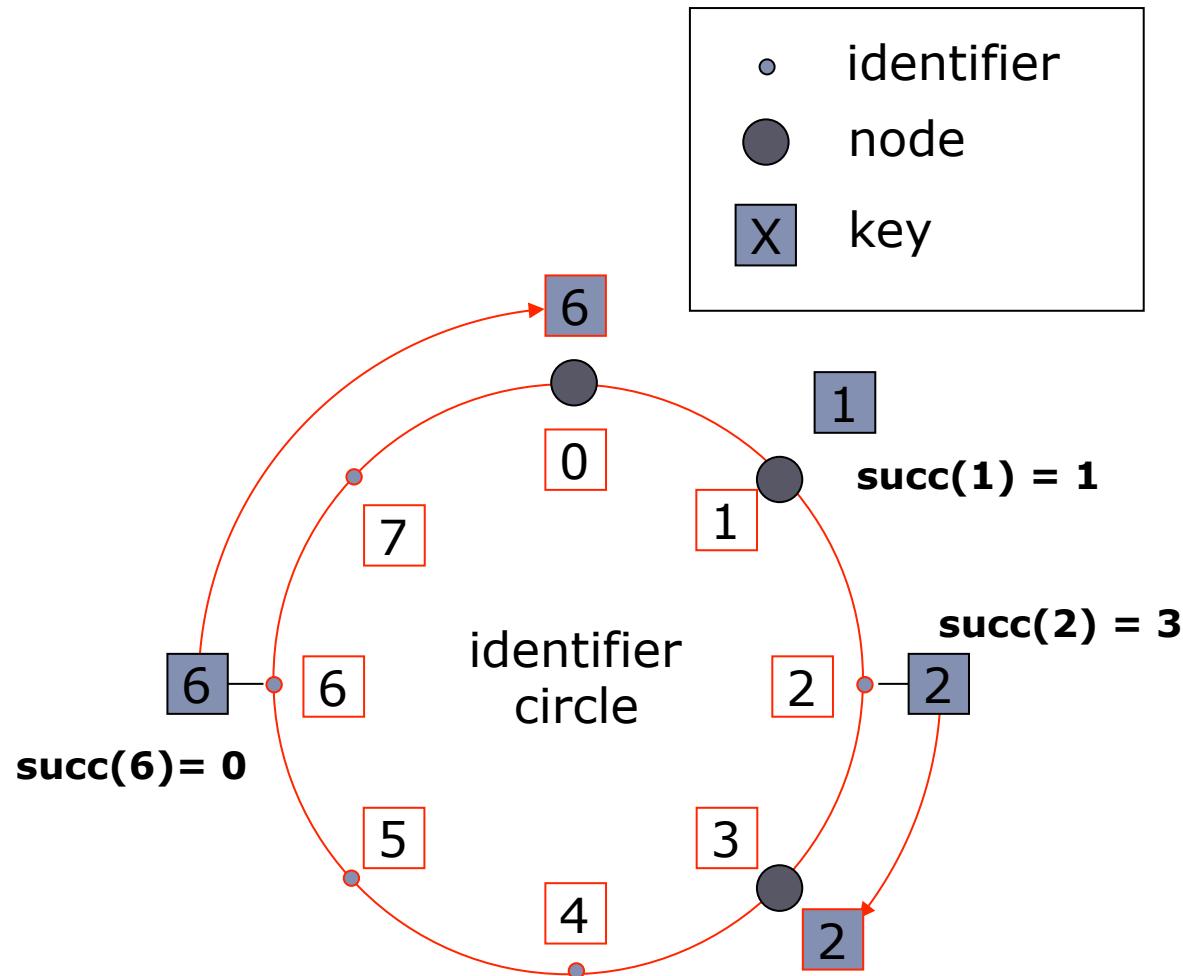
- ▶ Chaque serveur est responsable d'un sous ensemble de données (identifié par un intervalle de clés)
- ▶ Ajout d'une nouvelle machine
 - ▶ comment répartir la charge (load balancing) ?
- ▶ Localisation des données
 - ▶ Distributed Hash Tables (**DHT**)



DHT

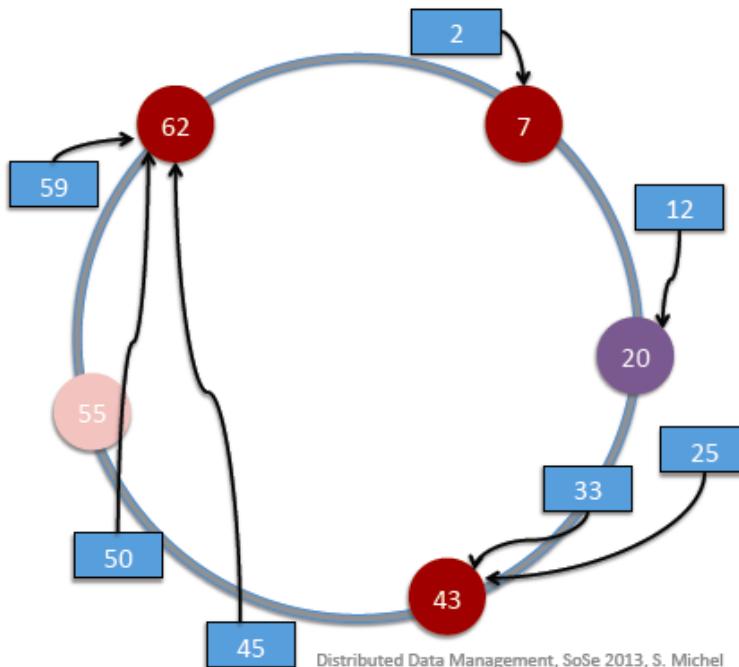
- ▶ Mécanisme distribué et décentralisé permettant d' associer des valeurs de clés à un contenu par **hachage** de la clé
 - ▶ Chaque participant gère une partie de la table de hachage.
 - ▶ Performance en recherche de l' ordre de **$\log(N)$**
 - ▶ Facilement reconfigurable lorsqu' un site se connecte ou se déconnecte
- ▶ Mécanisme général pour la localisation de ressources distribuées identifiées par des clés
 - ▶ Les clés et les noeuds (adresse IP) sont hachés sur le même anneau (cercle)
 - ▶ Clés et pairs sont affectés à un identifiant ds [0..m]
 - ▶ Succ(kid) : le plus petit identifiant de noeud qui soit supérieur ou égal à $k \bmod m$
 - ▶ Pred(kid) : le plus grand identifiant de noeud inférieur à $k \bmod m$
- ▶ Chaque clé d' identifiant k est gérée par le noeud suivant ou égal, i.e., d' identifiant supérieur ou égal
 - ▶ $\text{Pred}(\text{Succ}(k)) < \text{kid} \leq \text{Succ}(k)$
- ▶ Fonction, de hachage consistante comme MD5, SHA1 pour éviter les collisions et bien équilibrer la charge

Exemple (1)



Exemple (2)

Removed Server (id 55)





MapReduce

Motivation: traitement de données à grande échelle

- ▶ Traiter de grands volumes de données ($> 1 \text{ To}$)
- ▶ Paralléliser les traitements sur des centaines / milliers de processeurs

... Et tout cela de façon très simplifiée



MapReduce

- ▶ Introduit par Google en 2004
- ▶ But: faire des traitements parallèles (indexation, datamining, ...)
- ▶ Déplacer les traitements vers l'infrastructure de stockage
- ▶ MapReduce Offre:
 - **Parallélisation** et distribution automatique des traitements
 - Tolérance aux pannes
 - Equilibrage de charge
 - **Abstraction** propre pour les programmeurs

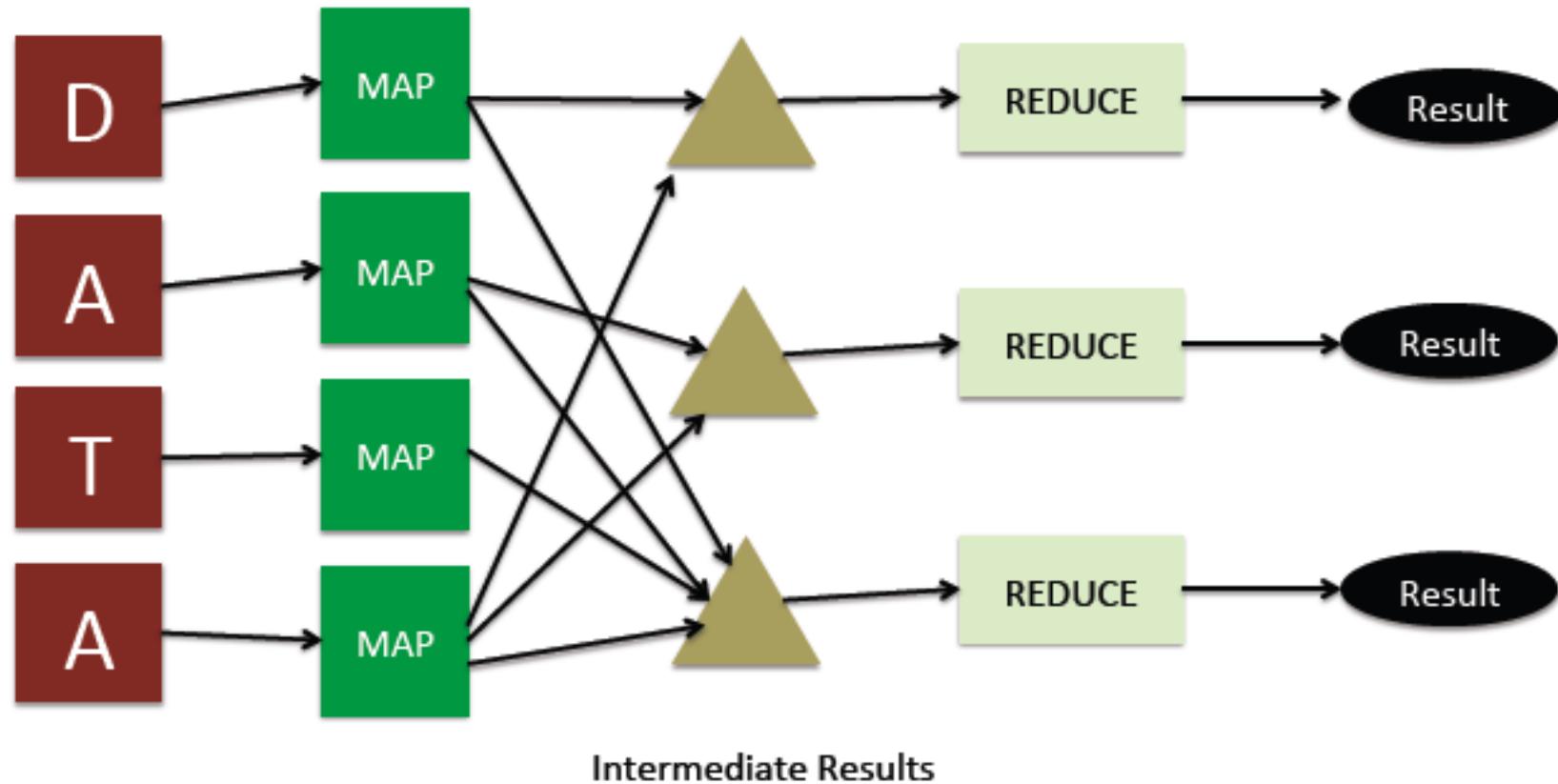


Modèle de programmation

- ▶ Emprunté à la programmation fonctionnelle
- ▶ Les programmeurs implémentent deux méthodes:
 - ▶ **Map**: est une étape de transformation des données sous la forme de paires clé/valeur
 - ▶ **Reduce**: est une étape de fusion des enregistrements par clé pour former le résultat final



MapReduce (architecture globale)



Map

- ▶ map (in_key, in_value) -> (out_key, intermediate_value) list

- ▶ Exemples:

- Majuscules

```
let map(k, v) = emit(k.toUpperCase(), v.toUpperCase())
("foo", "bar") -> ("FOO", "BAR")
("Foo", "other") -> ("FOO", "OTHER")
("key2", "data") -> ("KEY2", "DATA")
```

- Changement de clé

```
let map(k, v) = emit(v.length(), v)
("hi", "test") -> (4, "test")
("x", "quux") -> (4, "quux")
("y", "abracadabra") -> (10, "abracadabra")
```



Reduce

- ▶ `reduce (out_key, intermediate_value list) -> out_value list`
- ▶ Après la fin de `map()`, toutes les valeurs intermédiaires pour une clé de sortie sont regroupées dans une liste
- ▶ `reduce()` combine les valeurs intermédiaires dans une ou plusieurs valeurs finales pour chaque clé de sortie
- ▶ Exemple:

```
let reduce(k, vals) =  
    sum = 0  
    foreach int v in vals:  
        sum += v  
    emit(k, sum)
```

(“A”, [42, 100, 312]) -> (“A”, 454)

(“B”, [12, 6, -2]) -> (“B”, 16)

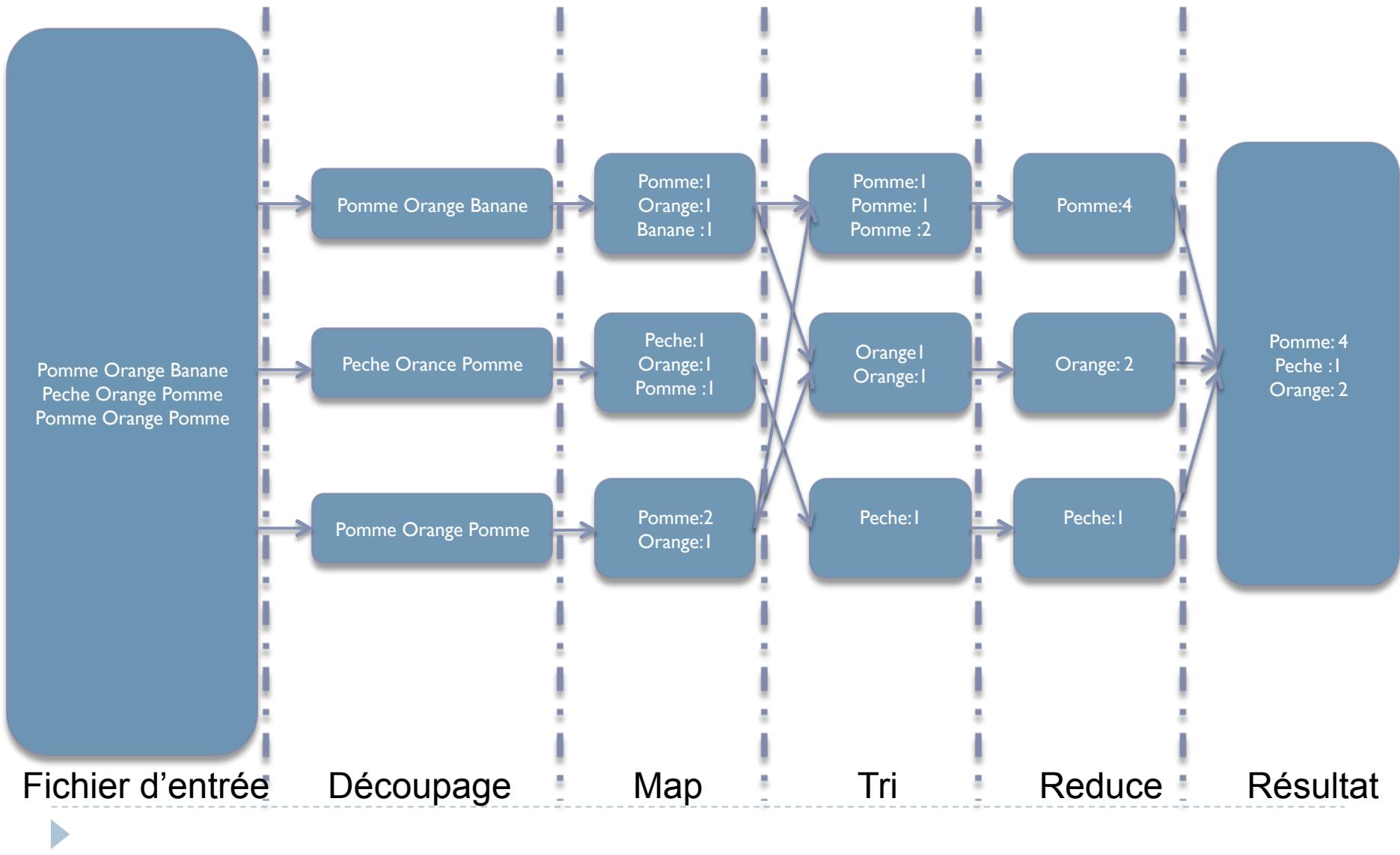


Map and Reduce

- ▶ Map ($k1, v1$) → list($k2, v2$)
 - ▶ Reduce($k2, \text{list}(v2)$) → list($k3, v3$)
Les clés permettent de regrouper les données et les machines/tâches
-
- ▶ Par exemple:
 - ▶ $k1$ = identifiant d'un document
 - ▶ $v1$ = contenu du document
 - ▶ $k2$ = mot
 - ▶ $v2$ = occurrence
 - ▶ $k3$ =mot
 - ▶ $v3$ = occurrence finale



Cas d'utilisation – Word Count



Traitement vers les données

- ▶ Données stockées dans des **systèmes de fichier distribués**
 - ▶ Ex. Google File System, Hadoop Distributed File System
- ▶ Plusieurs blocks (chunks généralement de 64Mo)
- ▶ Noeud maître (**master node**) connaît les localités des données
 - ▶ Reçoit les jobs
 - ▶ Calcule le nombre de map et reduce nécessaires
 - ▶ Sélectionne et active les nœuds esclaves
- ▶ Noeud maître envoie les traitements vers les nœuds esclaves (**worker nodes**)



MapReduce

- ▶ **Applications**
 - ▶ Indexation de document dans un moteur de recherches, grep distribué, tri distribué, statistiques d'accès au web, construction d'index inversé, clustering de documents, etc.
- ▶ **Plusieurs implémentations: C#, C++, Erlang, Java, Python, etc.**
- ▶ **La plus connue:**
 - ▶ Apache Hadoop Map Reduce



Hadoop

- ▶ Projet Open Source (Apache Software Foundation)
- ▶ Environnement d'exécution distribué
- ▶ Java
- ▶ Sous-projets: MapReduce, HDFS, Hbase, Pig, Hive, Mahout, etc.
- ▶ Grande volumétrie de données
 - ▶ Exemple de Yahoo: 500GB de données triées en 59s avec un cluster de 1400 nœuds.
- ▶ Applications:
 - ▶ LastFm: statistiques hebdomadaires (Top artistes, Top Titres)
 - ▶ Facebook: reporting et statistiques internes (croissance du nombre d'utilisateurs, consultation des pages, etc.)
- ▶ <http://wiki.apache.org/hadoop/PoweredBy>
 - ▶ Amazon, Apple, Ebay, IBM, Google, Microsoft, SAP, Twitter, etc



Ecosystème Hadoop

- ▶ Avro: Serialisation données
- ▶ Chukwa: supervision de clusters
- ▶ Flume: collecte de données log en temps réel
- ▶ Hbase: BD Colonnes
- ▶ Hive: Requête sur les données (proche de SQL)
- ▶ Pig: Langage de requête (scripting)
- ▶ Sqoop: transfert de données entre SGDBDR et hadoop
- ▶ Oozie: Workflow et orchestration de jobs
- ▶ Hue: Interface graphique pour Hadoop
- ▶ ...



Use cases

- ▶ Facebook has multiple Hadoop clusters deployed now - with the biggest having about 2500 cpu cores and 1 PetaByte of disk space. We are loading over 250 gigabytes of compressed data (over 2 terabytes uncompressed) into the Hadoop file system every day and have hundreds of jobs running each day against these data sets.
(source: https://www.facebook.com/note.php?note_id=16121578919)
- ▶ Google : the size of one phase of the computation [of the index] dropped from approximately 3800 line of C++ code to approximately 700 lines when expressed using MapReduce. (source: The art of rails, Edward Benson, p38)
- ▶ The Yahoo ! Search Webmap is a Hadoop application that runs on a more than 10,000 core Linux cluster and produces data that is now used in every Yahoo ! Web search query. (source: en.wikipedia.org)
- ▶ Other users: Amazon, ebay, Netflix, Spotify, IBM, LinkedIn, etc.



Implémentation des opérations d'algèbre relationnel en MapReduce

- ▶ **Sélection:**
 - ▶ Map retourne l'enregistrement s'il correspond aux critères de sélection
 - ▶ Pas de Reduce
- ▶ **Jointure:**
 - ▶ Map renvoie une paire intermédiaire composée du champ de jointure en clé et l'enregistrement en valeur
 - ▶ Reduce reçoit pour chaque clef de jointure la liste des enregistrements correspondants et se charge de faire le produit cartésien de ces enregistrements



MapReduce: conclusions

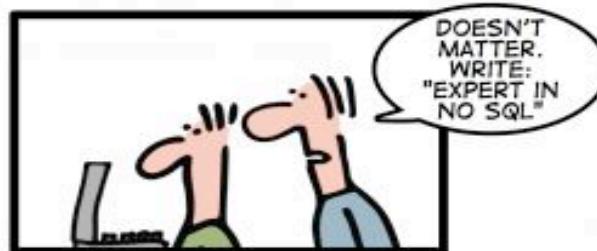
- ▶ MapReduce s'est avéré être une abstraction très utile
- ▶ Simplifie grandement les calculs à grande échelle chez les grands acteurs du web (Google, facebook, etc.)
- ▶ Paradigme de programmation fonctionnelle peut être appliqué à des applications à grande échelle
- ▶ Fun & Simple: on ne se concentre que sur le problème et on laisse la librairie gérer le « sale boulot » (réPLICATION, tolérance aux fautes, répartition de charge, etc.)





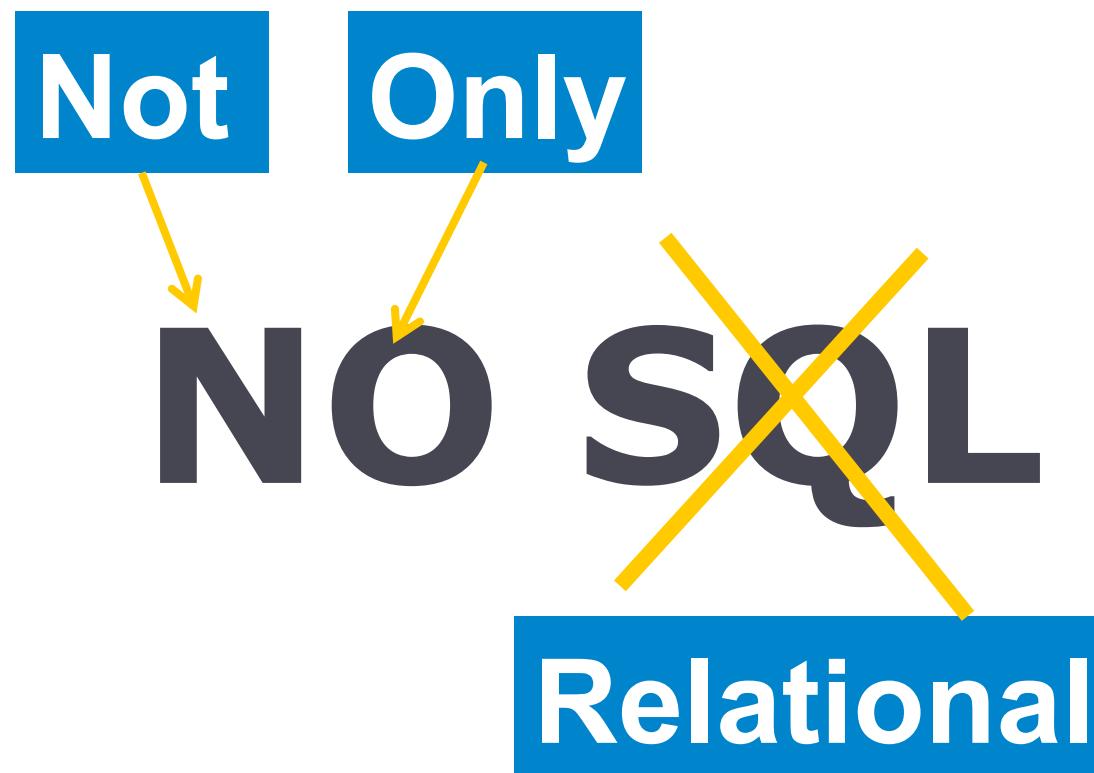
NoSQL

HOW TO WRITE A CV



Leverage the NoSQL boom

Fausses et vraies idées autour du NoSQL



NoSQL?

- ▶ No SQL => Not Only SQL
- ▶ SQL ne doit pas mourir mais des solutions de stockage doivent être envisagées pour des applications particulières (applications web en particulier)
- ▶ Nom exact: Bases de données non relationnelles
- ▶ Le modèle **ACID** ne permet pas un passage à l'échelle dans un environnement distribué, limitant par exemple les **débits en écriture** (les plus coûteux)
 - ▶ Atomicity
 - ▶ Consistency
 - ▶ Isolation
 - ▶ Durability



RDBMS

- ▶ MySQL, PostgreSQL, SQLite, Oracle, etc.
- ▶ Besoins:
 - ▶ Schémas
 - ▶ Cohérence forte
 - ▶ Transactions
 - ▶ Matures et opérationnelles
 - ▶ Expertise disponible



Quand utiliser NoSQL

- ▶ **Grand volume de données**
 - ▶ Besoin d'utiliser une architecture fortement distribuée
 - ▶ Google, Amazon, Yahoo, Facebook - 10-100K serveurs
- ▶ **Requêtes massives**
 - ▶ Eviter les jointures car très gourmandes en temps de traitement
- ▶ **Schéma évolutif**
 - ▶ Flexibilité et évolutivité du schéma à grande échelle



Nouveaux besoins

- ▶ Applications web: explosion des données et des requêtes
- ▶ Latence: temps de réponse faible et prévisible
- ▶ Passage à l'échelle et élasticité
- ▶ Haute disponibilité
- ▶ Schémas flexibles / données semi-structurées
- ▶ Distribution géographique (multitude de datacenters)
- ▶ Pas besoin de : transaction/ forte cohérence/ intégrité/
requêtes complexes



Exemples d'applications

- ▶ Recherche des utilisateurs: stockage des mots clés
- ▶ Alertes (ex: actualités, changement de prix, etc.)
- ▶ Activités dans les réseaux sociaux (connexion/
déconnexion, messages postés, chargement de photos,
tags, etc.)
- ▶ Traces des utilisateurs: clics (articles, publicités, etc.)





YAHOO!

source
forge

The logo for Facebook, featuring the word "facebook" in white on a blue background.

Google™

The logo for LinkedIn, featuring the word "Linked" in black and "in" in white on a blue background.

The logo for Twitter, featuring the word "twitter" in a light blue, bubbly font.

The logo for Ubuntu One, featuring the word "ubuntu" in black and "one" in orange and red.

The logo for Amazon.com, featuring the word "amazon.com" in black with a yellow arrow underneath, and the tagline "and you're done." in black.

NoSQL

▶ Avantages

- ▶ Passage à l'échelle
- ▶ Haute disponibilité
- ▶ Elasticité
- ▶ Flexibilité du schéma
- ▶ Gestion des données éparses (NULL) et semi-structurées
- ▶ Evolutivité horizontale

▶ Inconvénients

- ▶ Pas de standards
- ▶ Contrôle d'accès insuffisant
- ▶ Requête limité
- ▶ Applications client compliquées à cause de « Eventual consistency »



Théorème du CAP (E.Brewer, N. Lynch 2000)

- ▶ Trois propriétés d'un système: la cohérence (**Consistency**), la disponibilité (**Availability**) et la partition (**Partition tolerance**)
 - ▶ Consistency : l'ensemble des clients voient la même donnée, même après des mises à jour
 - ▶ Availability : tous les clients peuvent trouver de la donnée répliquée, même lors d'un crash
 - ▶ Partition tolerance : la base de données est tolérante au partitionnement
- ▶ Au maximum 2 propriétés respectées par n'importe quel système de données distribué
- ▶ Pour passer à l'échelle, on utilise le partitionnement. Ce qui implique un choix entre la cohérence ou la disponibilité





Merci pour votre attention

Raja.chiky@isep.fr