Big Data et ses technologies

ETS 2017.07 - Philippe Laflamme

Plan

- Introduction
- Big Data
  - Une définition
  - Pourquoi?
  - Applications
- Outils et technologies
  - Systèmes de fichiers distribués
  - Algorithmes distribués
  - Systèmes de base de données distribués
  - Systèmes d’orchestration
- Conclusion

Introduction

- Ingénieur informatique - Université de Sherbrooke
- Plus de 15 ans d’expérience
  - architecture logiciel, réseaux, systèmes distribués
  - traitement automatique du langage naturel
  - génomique / bioinformatique
  - consultation
  - “big data”
- Ingénieur senior chez **Hopper**
  - Utilisons les données pour aider nos utilisateurs à prendre des décisions éclairées en matière de voyage.

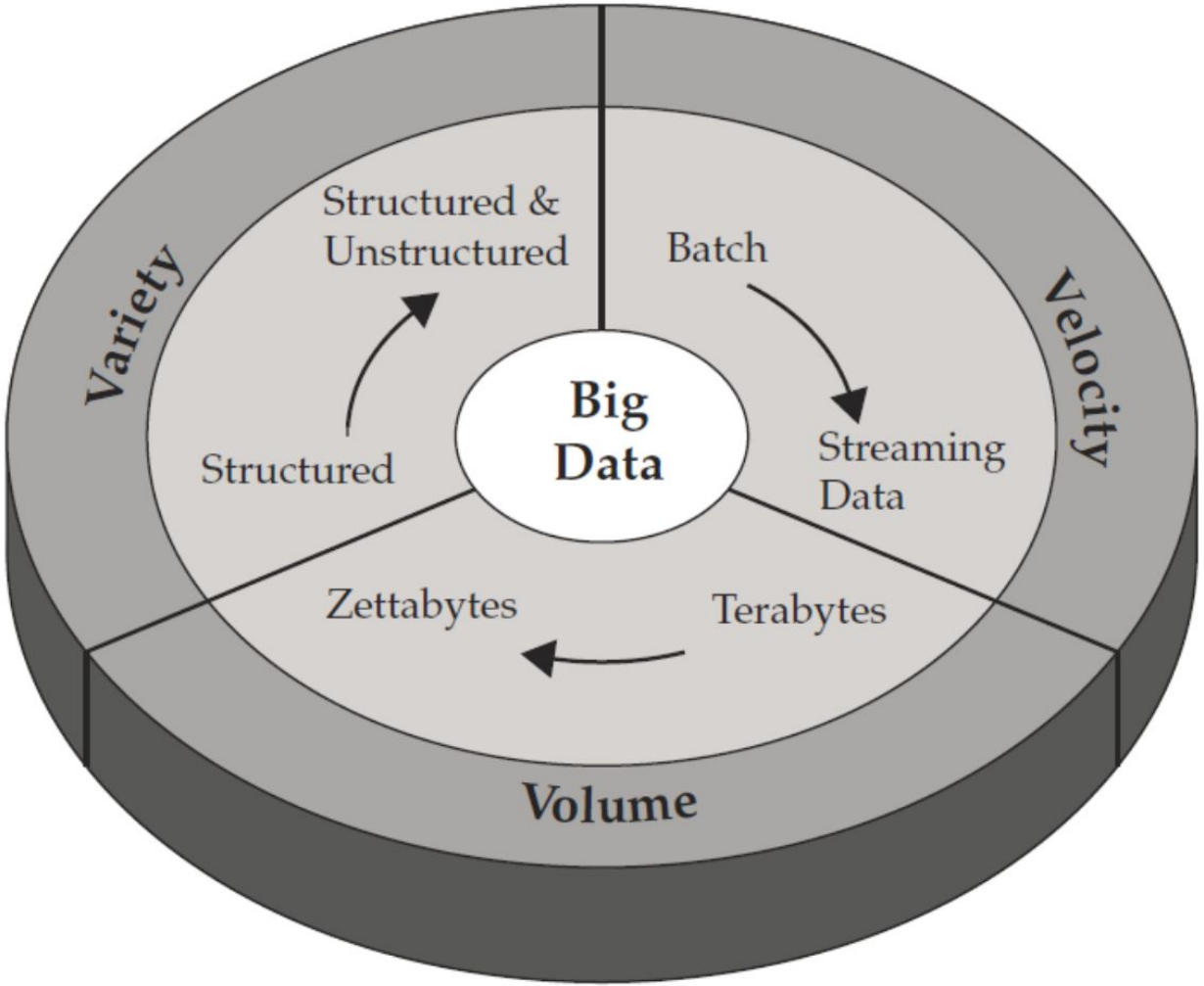
Big Data - Une définition

Big Data - Définition

*“Le Big Data (ou mégadonnées) représente les collections de données caractérisées par un **volume**, une **vélocité** et une **variété** si grands que leur transformation en **valeur** utilisable requiert l’utilisation de technologies et de méthodes analytiques spécifiques."*

Big Data - Définition

- **Volume** - pas d'échantillonnage, on observe et mesure tout
- **Vélocité** - les données et les résultats sont souvent disponibles en temps réel
- **Variété** - puise dans les données textuelles, les photos, audio / vidéo et complète généralement les pièces manquantes en fusionnant plusieurs sources


Variety
Structured &
Unstructured
Batch
Velocity
Big
Data
Structured
Streaming
Data
Zettabytes
Terabytes
Volume

Big Data - Pourquoi?

Big Data - Pourquoi?

- D’où vient ce concept du “big data”?
- Est-ce seulement le “petit” data qui est devenu “big”?
  - Simplement plus de data?
- Quelques pistes:
  - Explosion de la **disponibilité des données**
  - Augmentation de la **capacité de stockage**
  - Augmentation de la **capacité d’analyse**

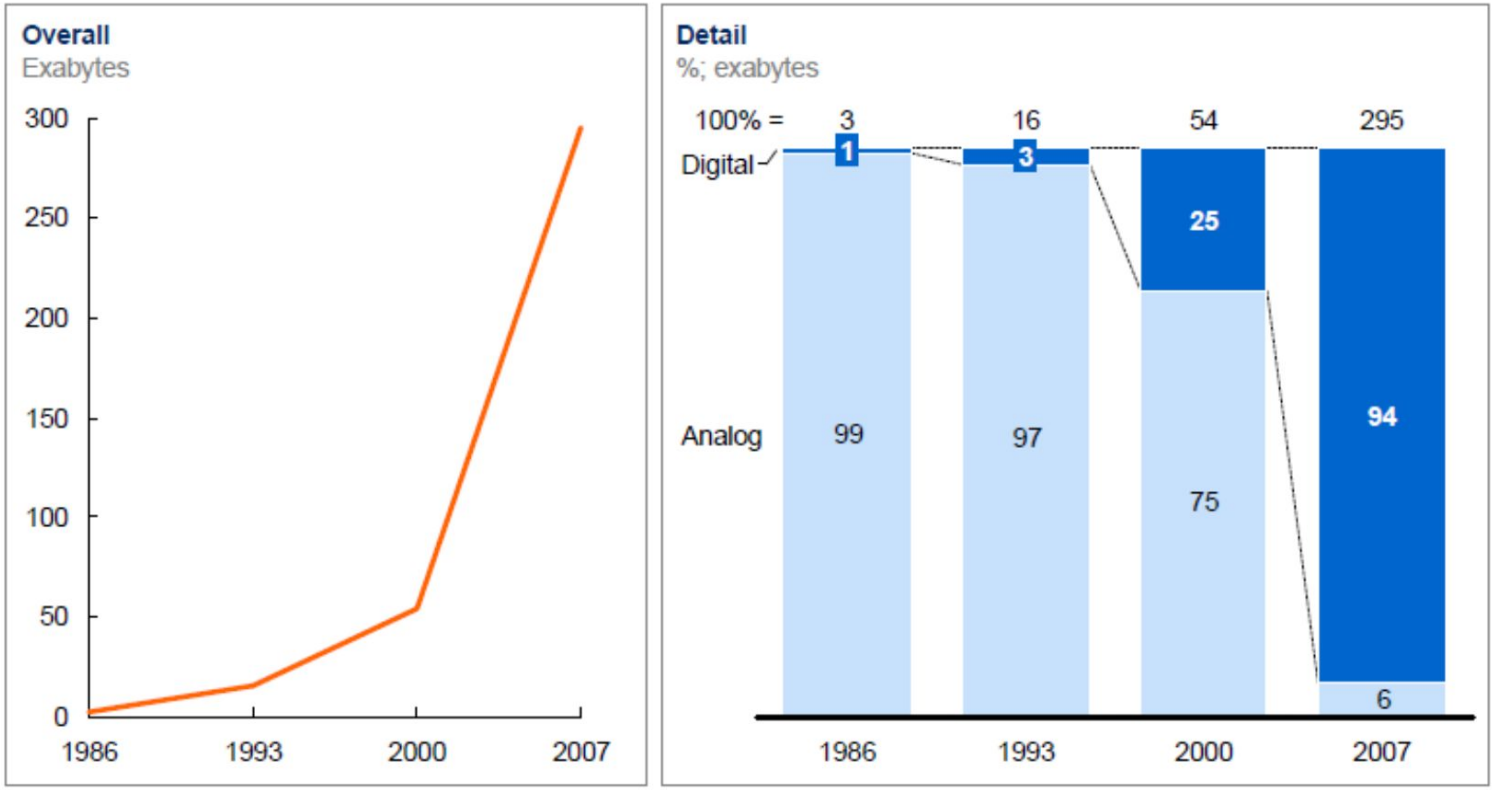
Big Data - Disponibilité des données

“There was 5 exabytes of information created between the dawn of civilization through 2003, but that much information is now created every 2 days, and the pace is increasing.”

-- Eric Schmidt, PDG Google, 2010

Data storage has grown significantly, shifting markedly from analog to digital after 2000

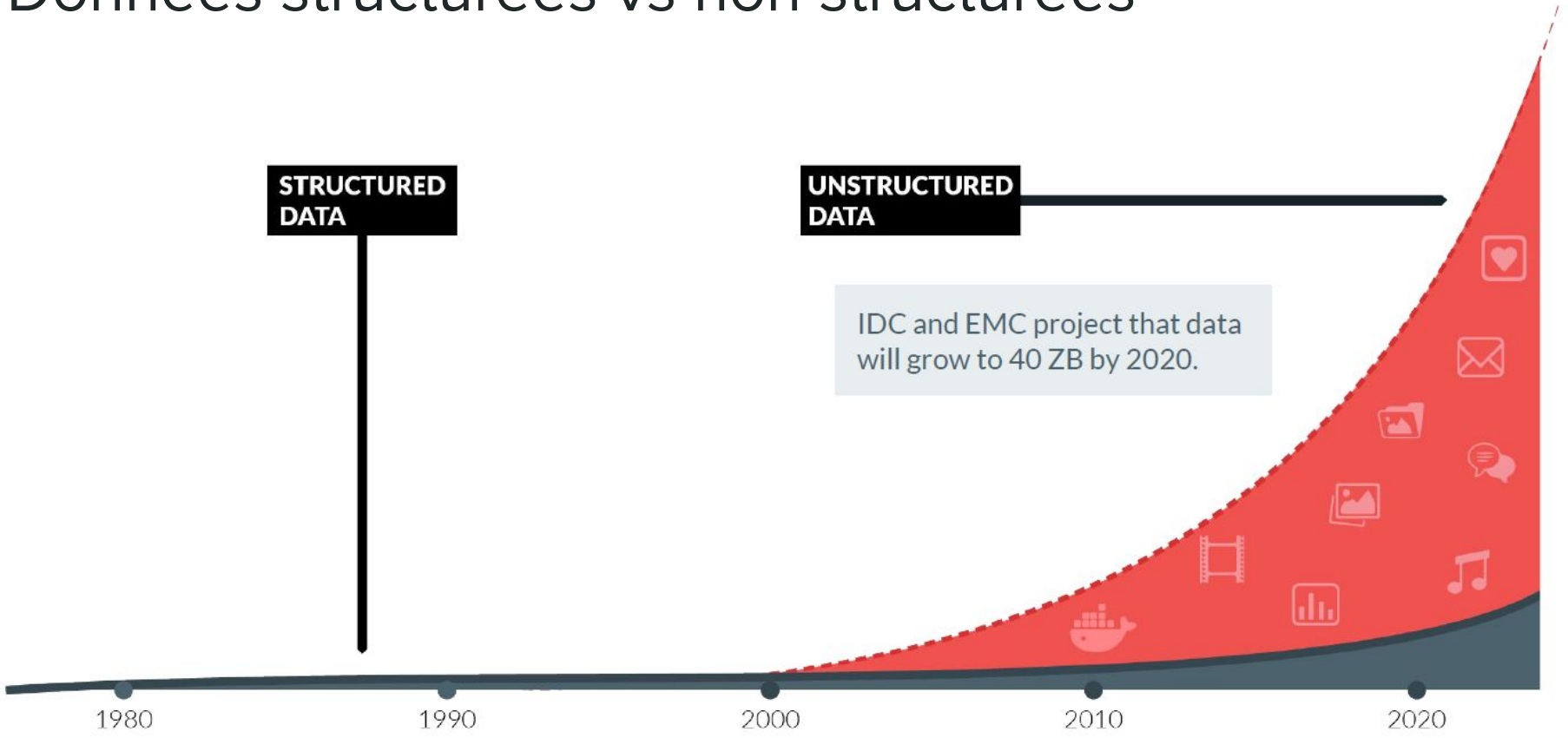
Global installed, optimally compressed, storage



NOTE: Numbers may not sum due to rounding.

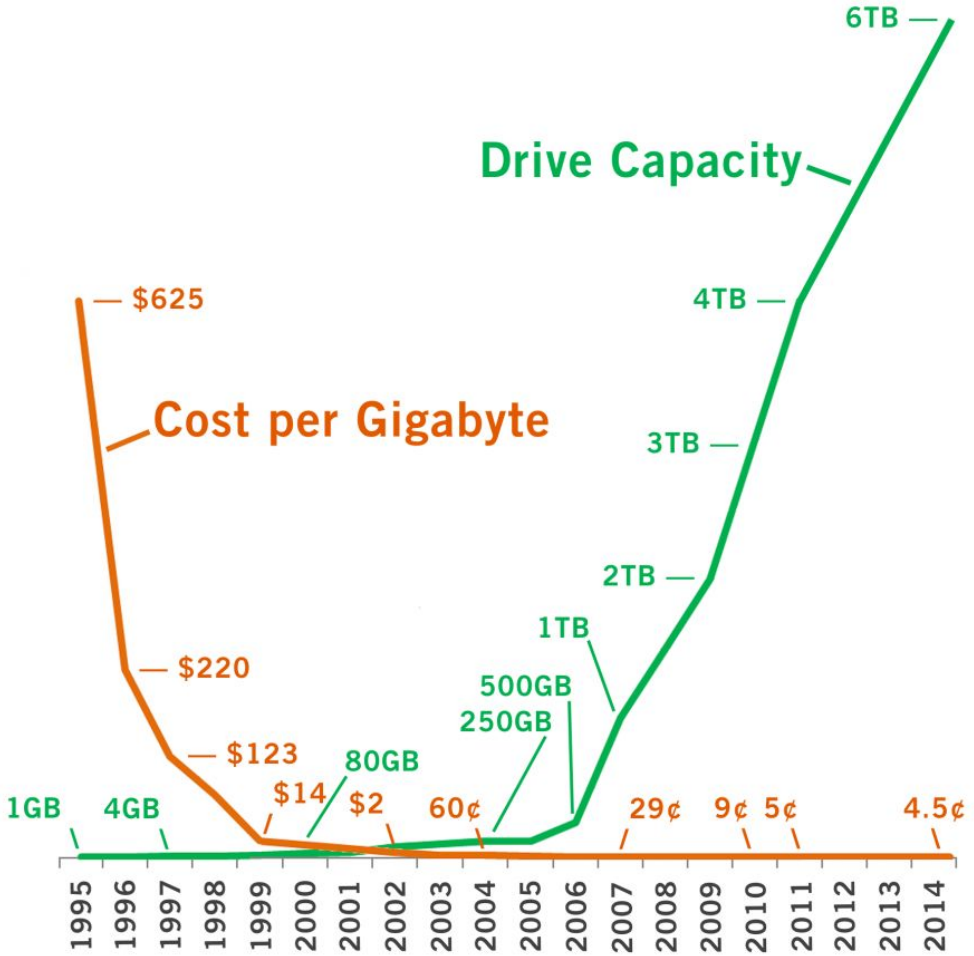
SOURCE: Hilbert and López, "The world's technological capacity to store, communicate, and compute information," *Science*, 2011

Données structurées vs non structurées



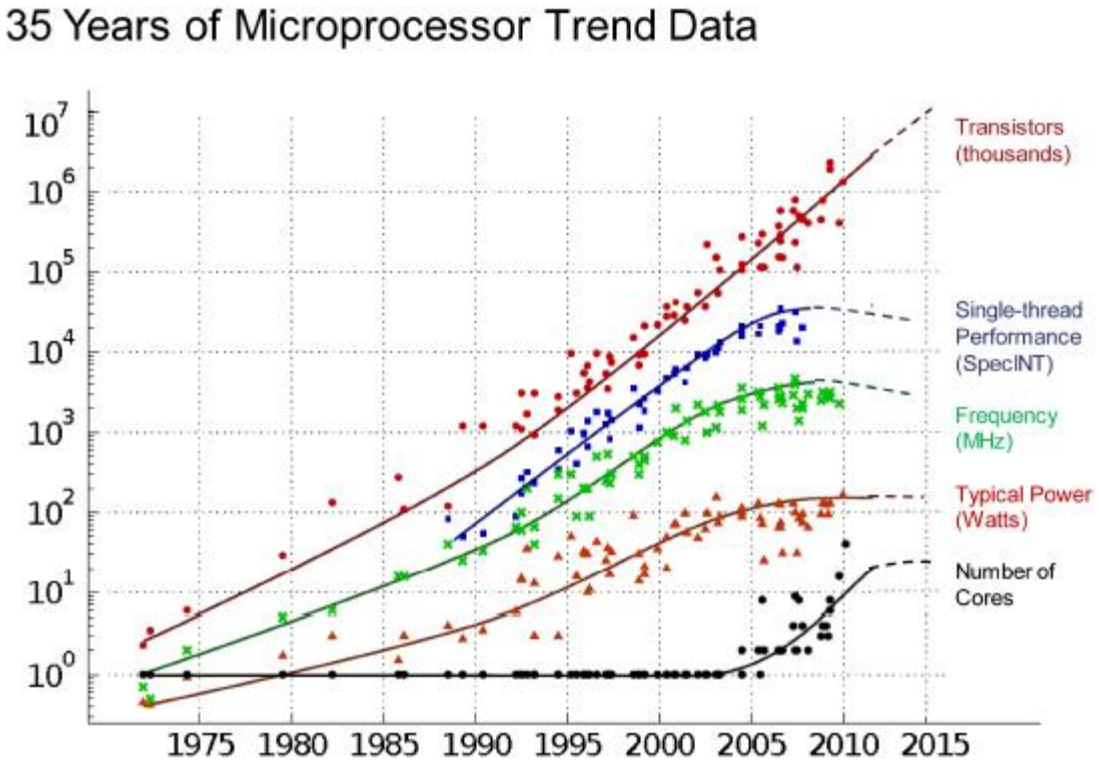
Big Data - Capacité de stockage

- Entre 2000 et 2006, la capacité des disques a augmenté par 10x alors que le prix par Gb a chuté du même ratio.
- Une augmentation de **100x** à prix constant.



Big Data - Capacité d’analyse

- La loi de Moore en action pendant environ 35 ans
- Plus récemment, la capacité d’analyse augmente grâce à l’ajout de coeurs dans les unités centrales



Big Data - Pourquoi?

- Augmentation exponentielle de la quantité de données non structurées
  - Email, chat, blog, **web**, musique, photo, vidéo, etc.
- Augmentation de la capacité de stockage et d'analyse
  - L'utilisation de plusieurs machines en parallèle devient accessible
- Les technologies existantes ne sont pas conçues pour ingérer ces données
  - Base de données relationnelles (tabulaires), mainframes, tableurs (Excel), etc.
- De "nouvelles" technologies et techniques d'analyse sont nécessaires
  - "Google File System" - Google 2003
  - "MapReduce: Simplified Data Processing on Large Clusters" - Google, 2004
  - Hadoop: circa 2006
- D'où le"Big Data": pas strictement plus de data...

Big Data - Les applications

Applications

- Recherche - PageRank (1996)
- Santé
- Éducation
  - MOOC
- Commerce de détail
  - Amazon, WallMart
- Génomique
  - High-throughput sequencing
- Science / recherche fondamentale
  - LHC
- Machine Learning / Deep Learning
- Recommendation
  - Netflix, Hopper!
- Urbanisme
- Gouvernements
- Média
  - journalisme de données
- Fraude (détection / prévention)
- IoT

Les techniques et les technologies

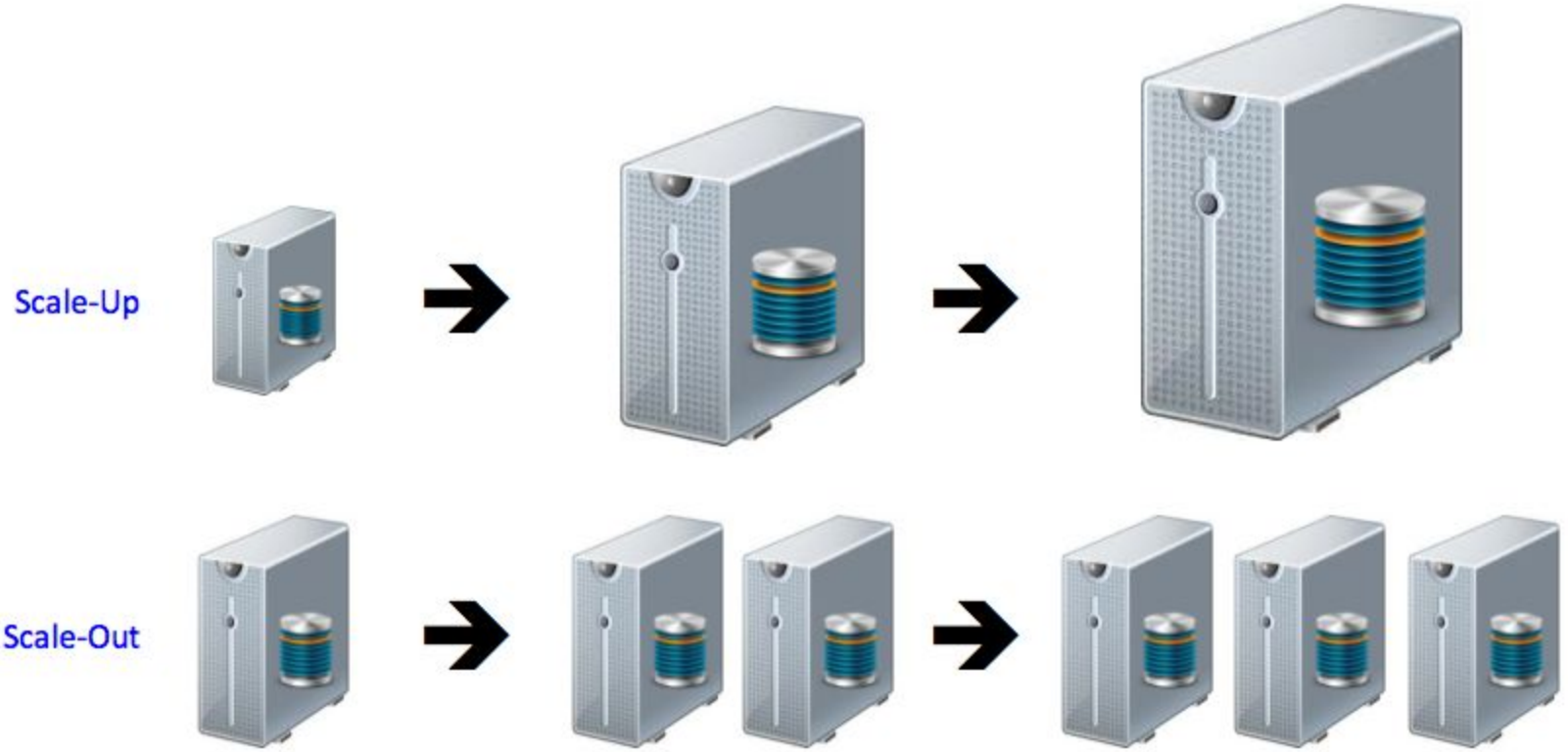
Technologies - systèmes de fichiers distribués

- La base du “Big Data”: le stockage
- **Volume**: une énorme quantité de stockage (à un coût raisonnable)
- **Vélocité**: agrandir la capacité de manière progressive
- **Variété**: un système de fichier “général”, qui permet de stocker n’importe quel genre de donnée

Technologies - systèmes de fichiers distribués

- Systèmes traditionnels: SAN
- Coût initial (très) élevé
- Peut atteindre une très grande capacité, mais éventuellement limitée
  - Installer ou migrer un SAN est très coûteux (temps + $)
- Généralement sur du matériel et/ou OS propriétaire qui ne permet pas de lancer ses propres tâches
- Système “**scale-up**” plutôt que “**scale-out**”

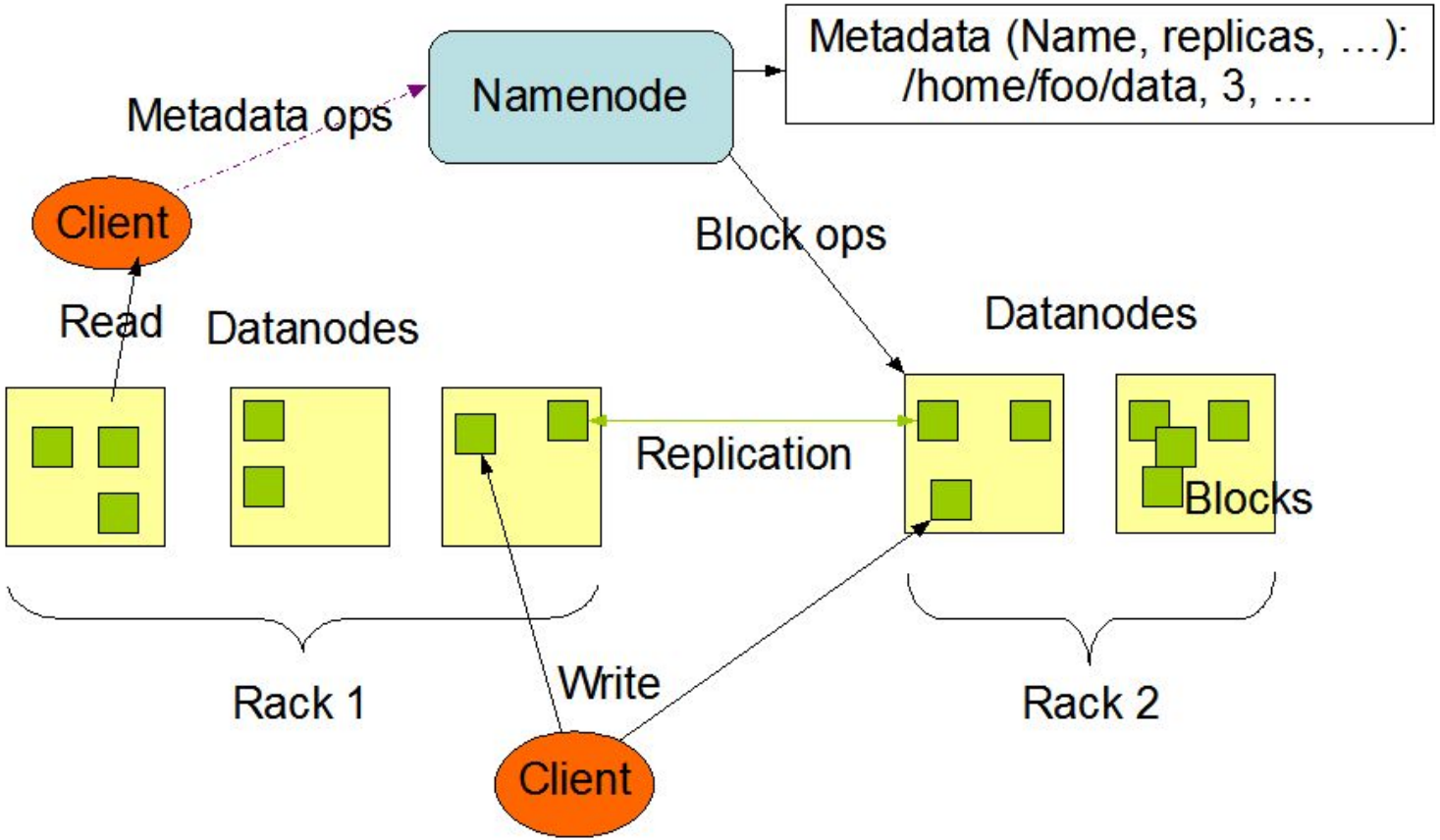
Scale up vs. Scale Out



Technologies - systèmes de fichiers distribués - HDFS

- Du besoin est né “Google File System” qui a inspiré la création de Hadoop / HDFS
- Hadoop Distributed File System est composé de 2 services:
  - **`Namenode`**: service de méta-données
  - **`Datanode`**: service de bloc de données
- Un fichier inscrit dans HDFS sera divisé en blocs
  - généralement de 64MB chacun
- Les blocs sont répliqués (généralement 3x) et distribués sur plusieurs `Datanode`
- Les clients du système de fichiers interrogent le `Namenode` pour:
  - connaître la structure de l’arbre de fichiers
  - découvrir où se trouvent les blocs d’un fichier
- Les clients accèdent aux données directement auprès des `Datanode`

HDFS Architecture



Technologies - systèmes de fichiers distribués - HDFS

- Les fichiers sont divisés en blocs: la perte d'un seul de ces blocs causerait une corruption du fichier
- Les blocs sont répliqués afin de survivre à la perte d'un ou plusieurs blocs
- Le `Namenode` tente de placer les blocs afin d'éviter la perte de toutes les copies d'un même bloc d'un seul coup (défaillance d'un "rack" ou d'une "switch")
  - "Rack-awareness"
- Les blocs peuvent être placés n'importe où, le client doit interroger le `Namenode` pour trouver un bloc
- Un `Namenode` "stand-by" peut être déployé
  - Nécessite d'autres services: `Journalnode`
  - Basculement automatique: `Zookeeper` et "`zkfc`"

Technologies - systèmes de fichiers distribués - HDFS

- Avantages:
  - En production dans des milliers de compagnies (“battle-hardened”)
  - Documentation
  - Compatibilité - pratiquement tout l’écosystème “Big Data” parle HDFS
  - Disponibilité du support (les “vendeurs” Hadoop)
- Désavantages:
  - La configuration “HA” est complexe et fragile
  - Ne supporte que la réplication pour éviter la perte de données
  - Aucune capacité de fédération (grappes de HDFS)
  - La nécessité d’interroger le `Namenode` pour chaque lecture
  - Nombre de blocs limité par l’espace mémoire du `Namenode` (*Scale-up* du `Namenode`)
  - Relativement lent

Technologies - systèmes de fichiers distribués - S3

- Produit d'Amazon sur la plateforme AWS
- Paradigme de "**`object-store`**":
  - **`bucket/key` → données**
  - aucune hiérarchie: on peut obtenir la liste des clés d'un "*bucket*" (relativement coûteux / lent)
- Option intéressante lorsque les données sont produites depuis AWS (EC2 ou EMR)
- Permet de rendre des données publiques très facilement
- Plusieurs options d'authentification et d'autorisation
- S3 est aussi un protocole: plusieurs autres implémentations existent

Technologies - systèmes de fichiers distribués - S3

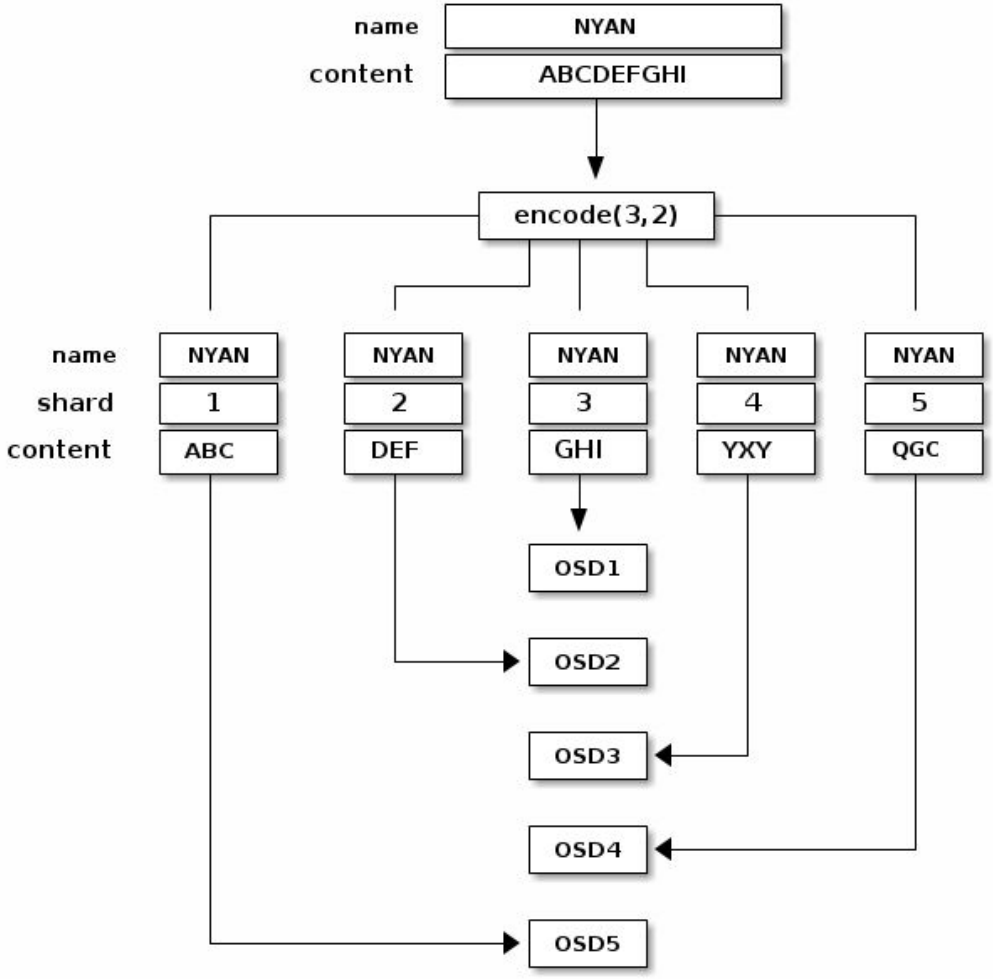
- Avantages
  - Disponible dès le jour 0
  - 99,9% de disponibilité: non disponible durant pas plus de 43 minutes par année
  - Simple d'utilisation (HTTP)
- Désavantages
  - Système propriétaire et opaque
  - Potentiellement coûteux à long terme
  - Moins intéressant lorsqu'on produit les données localement
  - Relativement lent
  - Certaines opérations sont contre-intuitivement très coûteuses (renommer un fichier est une copie) ➜ nécessite une attention particulière

Technologies - systèmes de fichiers distribués - CEPH

- Système de fichiers distribué conçu et développé par RedHat
- Conceptuellement semblable à HDFS
  - différences architecturales fondamentales
- **`mds`**: service de méta-données
- **`ods`**: service de données
- **`mon`**: service de surveillance des autres services
  - détecte les défaillances afin de déplacer les données

Technologies - systèmes de fichiers distribués - CEPH

- Utilise un algorithme de hashing pour déterminer l'emplacement d'un bloc
- Permet à un client ceph de parler directement au service **`ods`** afin d'accéder aux données
- Aucun point de défaillance unique (*SPOF*)
- Supporte les 3 paradigmes d'accès:
  - **object-store** (S3)
  - **block-store** (`/dev/sda1`)
  - **POSIX** (système de fichier traditionnel)
- Possible d'utiliser avec Hadoop
- Supporte le "erasure-coding" ainsi que la réplication pour éviter la perte de données
  - Permet la même résilience que la réplication 3x, mais avec seulement 40% d'amplification des données


name
NYAN
content
ABCDEFGHI
encode(3,2)
name
NYAN
NYAN
NYAN
NYAN
NYAN
shard
1
2
3
4
5
content
ABC
DEF
GHI
YXY
QGC
OSD1
OSD2
OSD3
OSD4
OSD5

Technologies - systèmes de fichiers distribués - CEPH

- Avantages
  - Performant
  - Architecture moderne: aucun point de défaillance unique
  - Énormes déploiements de production (p. ex.: CERN)
  - Polyvalent: permet de consolider plusieurs cas d’usage
- Désavantages
  - Configuration extrêmement complexe
  - Très peu de cas d’usage (public) Hadoop

Techniques - algorithmes distribués

Techniques - algorithmes distribués

- Historiquement dominés par des systèmes haute-performance
  - problèmes “`cpu-bound`”, peu de données, calculs complexes
  - p. ex: MPI (message passing interface), les données sont envoyées aux agents, le calcul effectué et les résultats retournés au “coordonnateur”
- Afin de traiter de très grandes quantités de données, inverser la responsabilité: déplacer l’algorithme vers les données
  - “data-locality”
- Afin de maximiser le parallélisme: réduire la dépendance entre les “agents”
- Formalisation: map / reduce

Techniques - algorithmes distribués - MapReduce

- Algorithme composé de 2 étapes conceptuelles
  - map - transformation des données en paires de clé-valeur
  - reduce - opération d'agrégation (somme, moyenne, etc.) par clé
- L’implémentation nécessite plus de 2 étapes, mais elles sont généralement transparentes pour le programmeur
- Sa résilience et son parallélisme sont ce qui le rendent particulièrement intéressant pour le Big-Data
  - “Embarassingly parallel”
  - Capacité de redémarrer n’importe quelle sous-étape (tant que les données source existent)

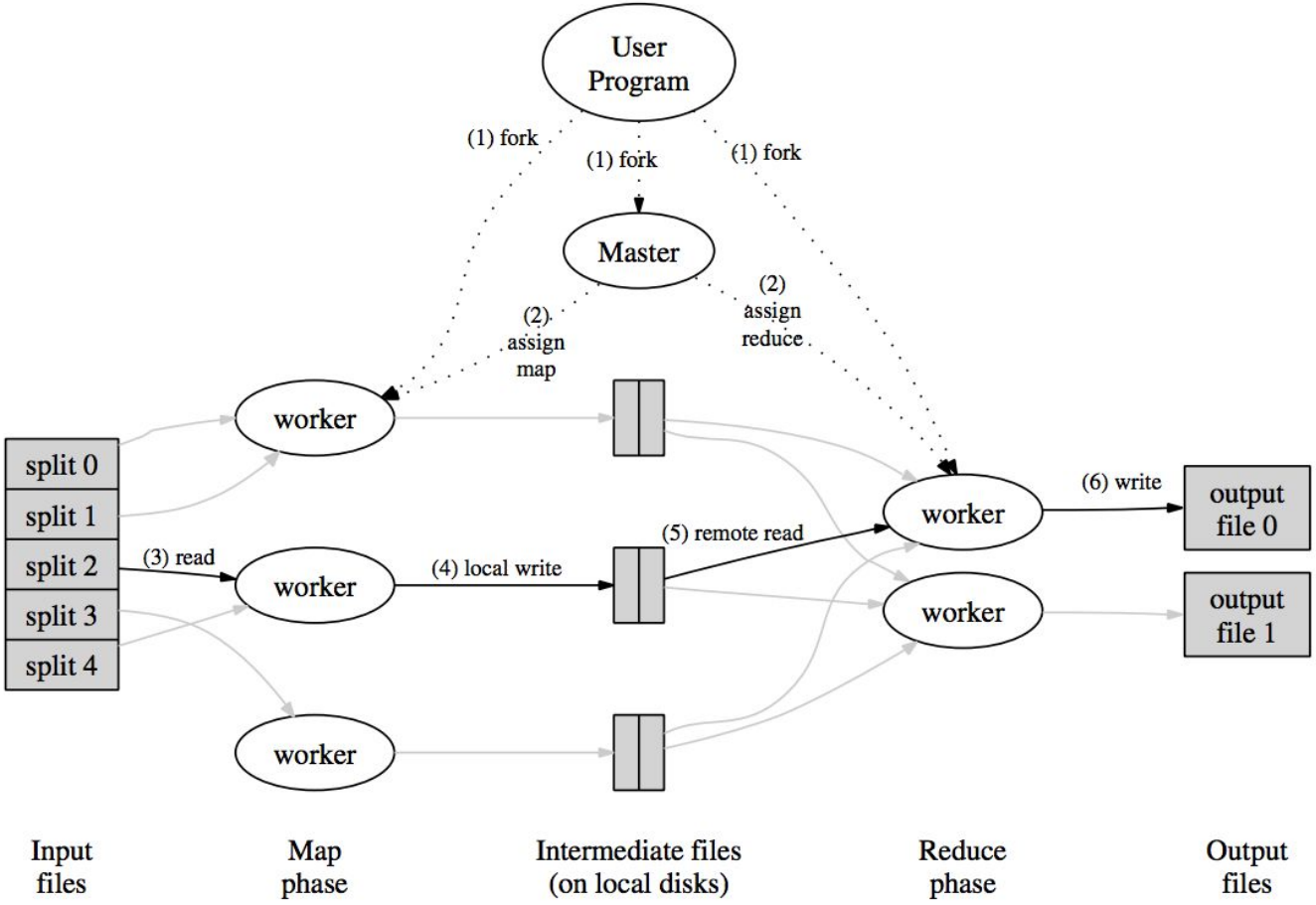
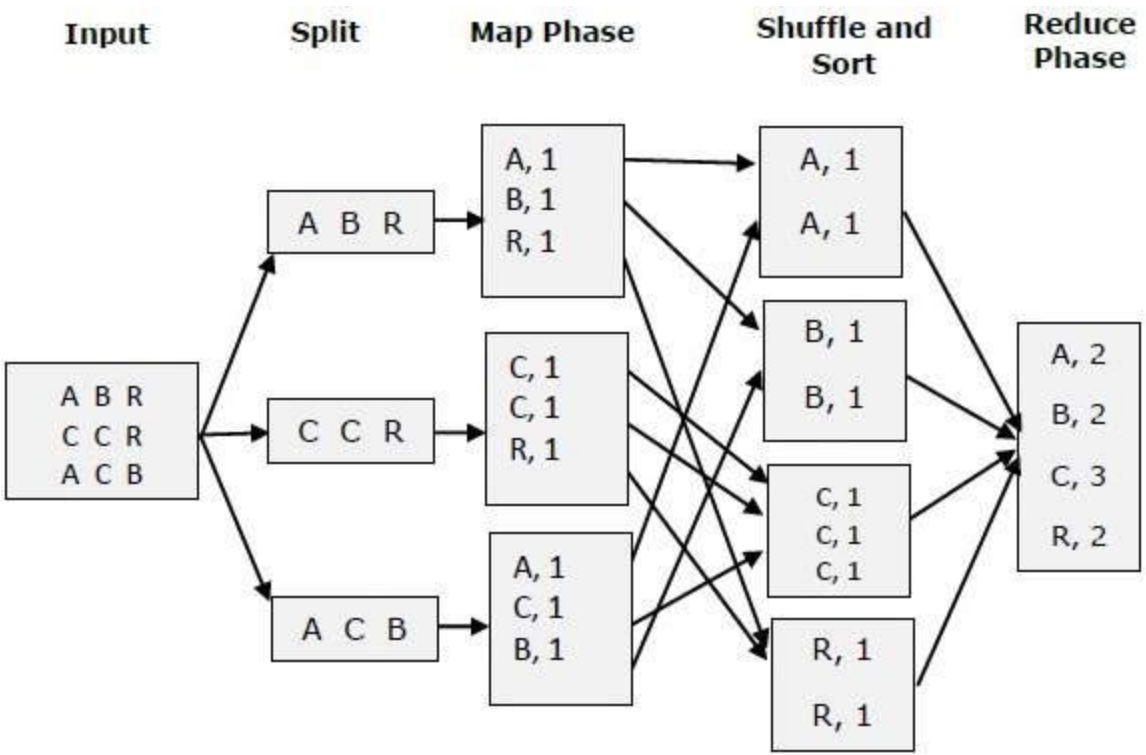


Figure 1: Execution overview


Input
Split
Map Phase
Shuffle and Sort
Reduce Phase
A B R
C C R
A C B
A B R
C C R
A C B
A, 1
B, 1
R, 1
C, 1
C, 1
R, 1
A, 1
C, 1
B, 1
A, 1
A, 1
B, 1
B, 1
C, 1
C, 1
C, 1
R, 1
R, 1
A, 2
B, 2
C, 3
R, 2

Techniques - algorithmes distribués - MapReduce

- À moins d’avoir un problème très simple, une seule phase M/R n’est pas suffisante
- Généralement, on doit écrire plusieurs phases et les exécuter en chaîne
- Manuellement:
  - laborieux
  - sujet à l’erreur
  - possibilités d’optimisations potentiellement perdues
- Plusieurs abstractions ont été créées
  - Certaines ont des optimisations intéressantes (map-side aggregation)
  - Écrire le Map/Reduce directement n’est pas tellement conseillé aujourd’hui

Techniques - algorithmes distribués - MapReduce

- Hive
  - “compile” le SQL en chaîne de phases M/R
  - permet la gestion de données structurées (tabulaires)
  - s’adresse principalement aux analystes d’affaires (BI), création de rapports, etc.
- Pig
  - langage haut-niveau (quelques emprunts à SQL) compilé en phases M/R
  - s’adresse principalement aux programmeurs / chercheurs
- Cascading
  - abstraction de “pipes-and-filters” sur Hadoop
  - contient un optimiseur d’exécution (plan logique vs. plan physique)
  - interface java, scala, clojure, ruby, python et SQL
  - s’adresse aux programmeurs
- et beaucoup d’autres...

Techniques - algorithmes distribués - Spark

- Ré-implémentation d'une plateforme de calcul distribué
  - contient les mêmes étapes conceptuelles que Map/Reduce
  - n'utilise pas Map/Reduce de Hadoop
  - plan d'exécution plus sophistiqué
  - p. ex.: ne nécessite pas l'écriture sur disque à chaque étape
- Unité opérationnelle est le **RDD**: **resilient distributed dataset**

Resilient Distributed Datasets (RDD)

RDD of Strings

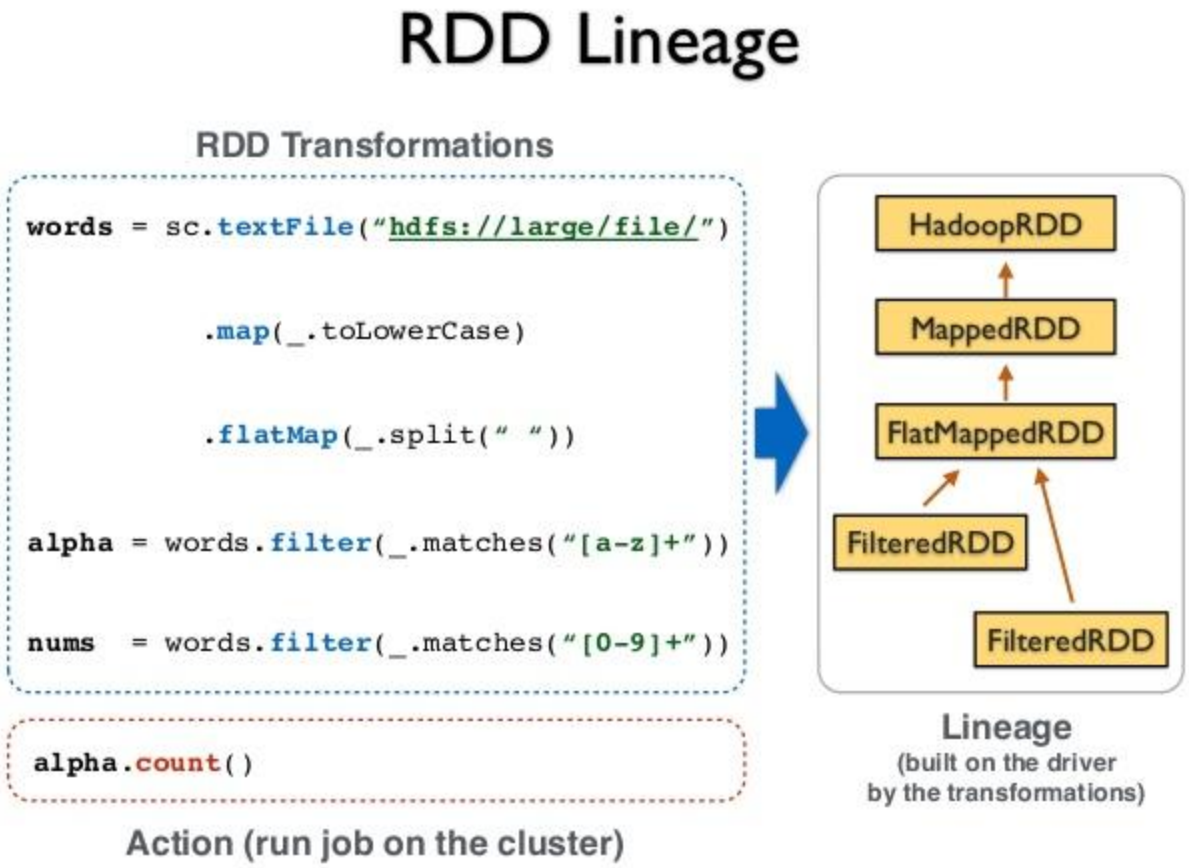
Immutable **Collection** of Objects

Partitioned and **Distributed**

Stored in **Memory**

Partitions **Recomputed on Failure**

- La lignée d'un **RDD** est connue et permet de le recalculer à n'importe quel moment



Techniques - algorithmes distribués - Spark

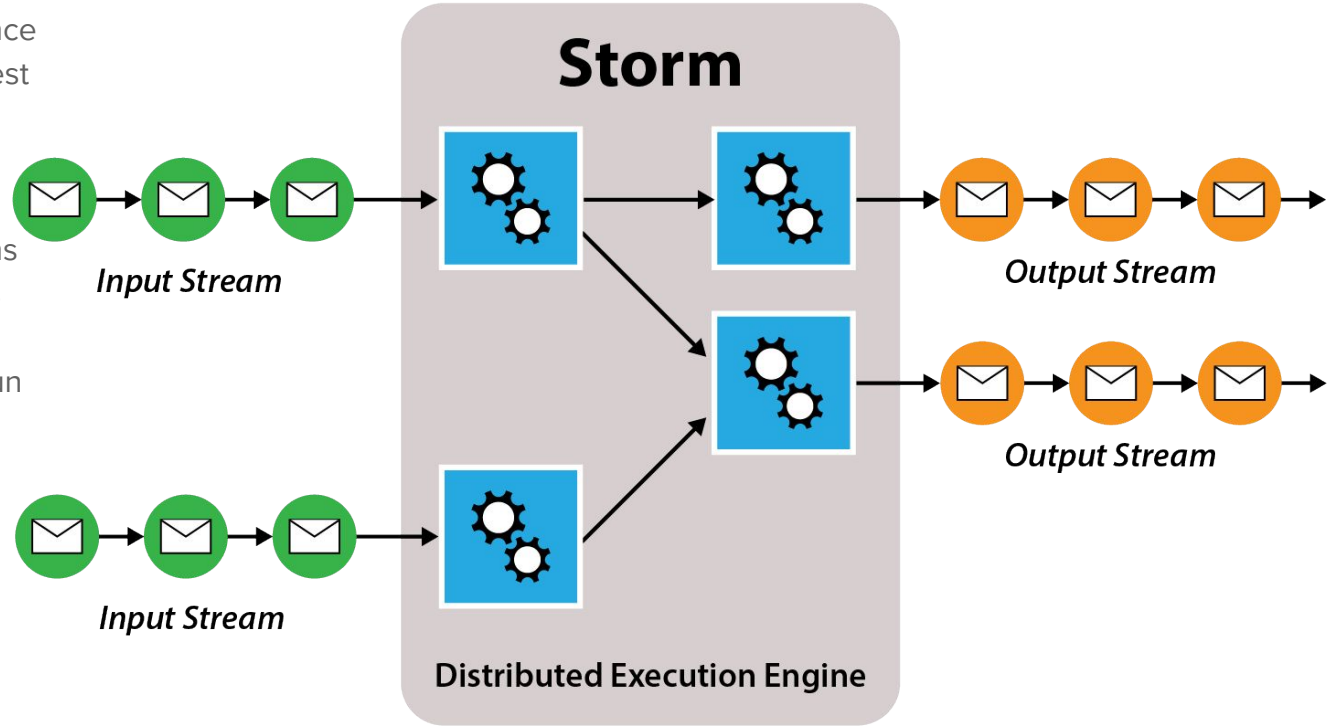
- Spark est une **plateforme**: une même abstraction pour différents cas d'usage
  - spark: tâches “batch”
  - spark-streaming: tâches en flux continu de données
  - spark-mllib: apprentissage machine
  - spark-graphx: manipulation de graph
  - spark-sql: abstraction SQL
  - spark-R: exécution de R sur spark
  - pyspark: exécution de python sur spark
  - des dizaines d'extensions...

Techniques - algorithmes distribués - Flux vs. batch

- Initialement, le big-data opérait en “batch”
  - création d’index, rapports, etc. à chaque jour (24 h)
  - pression d’obtenir des résultats de plus en plus rapidement
- Problème fondamental: durée d’une tâche batch doit être < interval entre les résultats
- Nécessité de pouvoir calculer “incrémentalement” en flux continu
  - p. ex.: opérer sur chaque “tweet”
- Différentes plateformes sont créées:
  - Storm (Twitter), Samza (LinkedIn), Spark Streaming

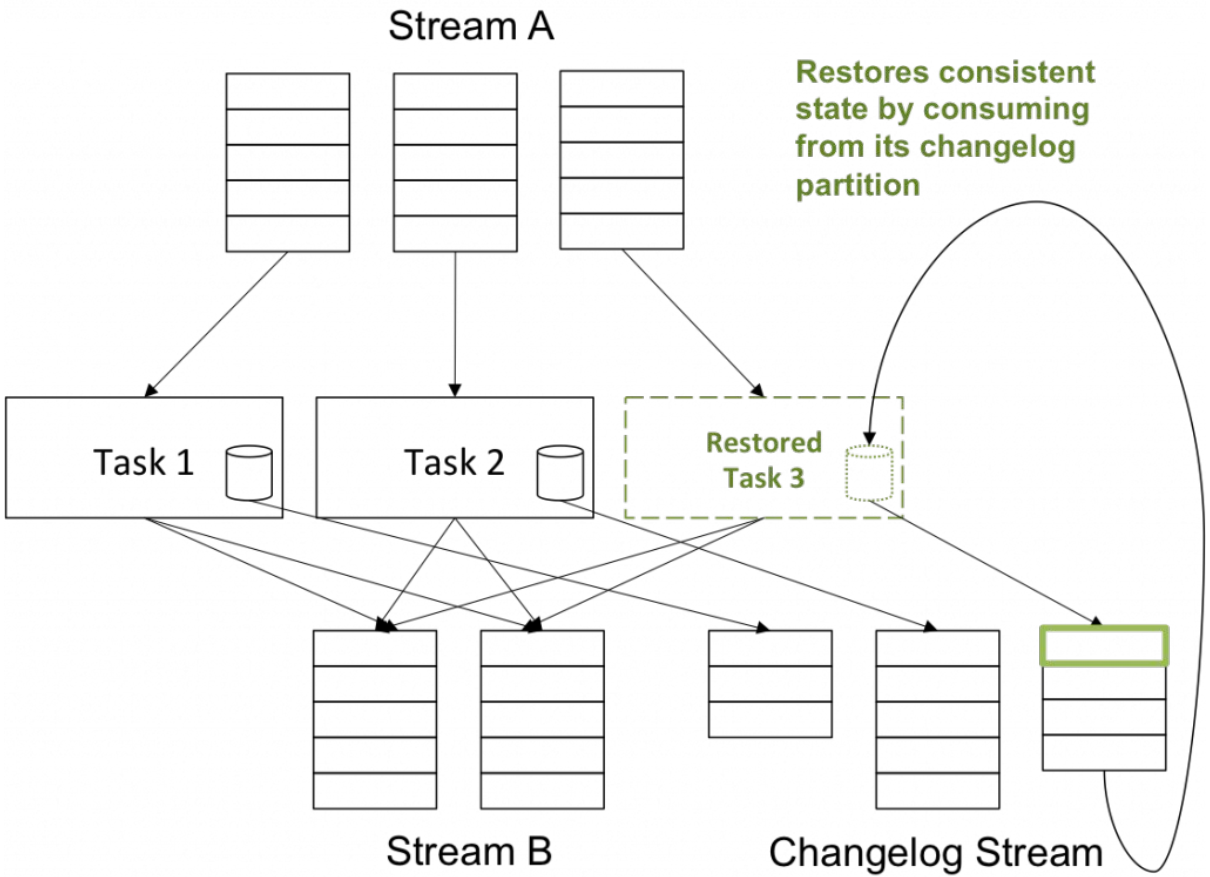
Apache Storm

- Plateforme haute-performance
- Chaque donnée en entrée est traitée individuellement
- La topologie de traitement peut être très complexe
- Développé chez Twitter dans le but de générer les tweets en temps réel
- A depuis été remplacé par un système encore plus performant



Apache Samza

- Développé chez LinkedIn
- Développé autour des garanties et propriétés de Kafka
- Une tâche Samza consomme d'un "topic" Kafka et génère un nouveau flux de données
- Ce flux est généralement envoyé dans un autre "topic" Kafka
- Abstraction de pipes-and-filters sur Kafka



Spark Streaming

e.g: Kafka, Kinesis, Flume, Sockets, Akka etc

Data **Collected**, **Buffered** and **Replicated** by a **Receiver** (one per DStream) then **Pushed** to a **stream** as small **RDDs**

Configurable **Batch Intervals.**
e.g: 1 second, 5 seconds, 5 minutes

DStream Transformations


```
// Example
val entries = stream.transform { rdd => rdd.map(Log.parse) }

// Alternative
val entries = stream.map(Log.parse)
```

Techniques - algorithmes distribués - Flux vs. batch

- Initialement, les tâches batches sont distinctes des tâches en flux
  - duplication de “logique d’affaire”
- Les tâches en flux calculent incrémentalement (possiblement de manière inexacte)
- Les tâches batches sont utilisées pour “réparer”, s’assurer de l’exactitude des données
  - ou simplement complémenter
- C’est le “**lambda architecture**”
  - Architecture qui utilise des processus batch et en flux
- Spark permet aujourd’hui d’unifier les 2 mondes
  - Le mode d’opération d’une tâche est un détail de déploiement

Technologies - systèmes de BD

Technologies - systèmes de base de données

- Les paradigmes de batch et de flux sont insuffisants
  - Il est aussi nécessaire de lire et écrire aléatoirement (random read/write)
- Les bases de données traditionnelles (du type scale-up) ne sont pas appropriées
  - Volume: un seul serveur ne peut plus contenir toutes les données
  - Vélocité: la bande passante d'un seul serveur ne peut pas soutenir le taux de requêtes
  - Variété: les données ne sont pas toutes tabulaires (relationnelles)
- C'est la naissance du "**NoSQL**"
  - un pauvre choix de nom
  - ne décrit pas ce que le système est, mais plutôt ce qu'il n'est pas
  - plusieurs BD "**NoSQL**" ont une interface SQL (ou simili-SQL)

Technologies - systèmes de BD - NoSQL

- Généralement (et non strictement), une BD **NoSQL**:
  - offre un paradigme d'accès ou stockage **non relationnel**
  - est distribuée
  - offre une certaine forme de capacité "scale-out"
  - utilise un design simple (qui offre parfois peu de fonctionnalités)
  - utilise une architecture sans point de défaillance unique
- Étant donné le design simple, une BD NoSQL peut:
  - soutenir un taux de requête très grand
  - survivre à des défaillances réseau ou de noeud
  - offrir une capacité très grande de stockage

Technologies - systèmes de BD - NoSQL

- Le NoSQL n’est pas magique: on échange les garanties des BD traditionnelles contre ces avantages:
  - pas de transaction (begin, commit / rollback)
  - perte des garanties de durabilité (perte d’une écriture confirmée)
  - face aux défaillances réseau, doit choisir entre cohérence et disponibilité
- De nombreux autres problèmes:
  - projets immatures: nombreux bugs, perte de données en production (même sans défaillance)
  - défaillance architecturales fondamentales: algorithmes de consensus brisé
  - installation et/ou opérations parfois très complexe
  - très peu, voire aucun support pour les entreprises (nécessite expertise interne)
- Ce sont, malgré tout, des outils indispensables
  - généralement, le choix se limite à une ou deux technologies
  - le choix est basé sur les garanties fournies ainsi que la “famille”

Technologies - systèmes de BD - NoSQL

- Les familles (selon le paradigme d’accès):
  - Document
  - Clé-valeur / famille de colonnes
  - Graphe
  - Structuré / semi-structuré
- Non exhaustif, c’est une catégorisation parmi d’autres

Technologies - systèmes de BD - Document

- Accès par **Document**
  - unité de stockage est un “document” (p. ex. : un objet JSON)
  - le paradigme est généralement une collection de document JSON
  - les documents peuvent être arbitrairement larges
  - les documents d’une même collection ne sont pas nécessairement homogènes (**schemaless**)
  - on accède au document sous une clé primaire
  - parfois, certains champs peuvent être indexés (index secondaires)
  - on peut interroger par la clé primaire ou une clé secondaire (le cas échéant)
  - supporte généralement la manipulation de sous-documents
  - offre généralement un langage pour chercher parmi les documents
  - se combine généralement facilement avec des applications web (JSON + HTTP/REST)
    - populaire auprès des plateformes de langage dynamique

Technologies - systèmes de BD - Document

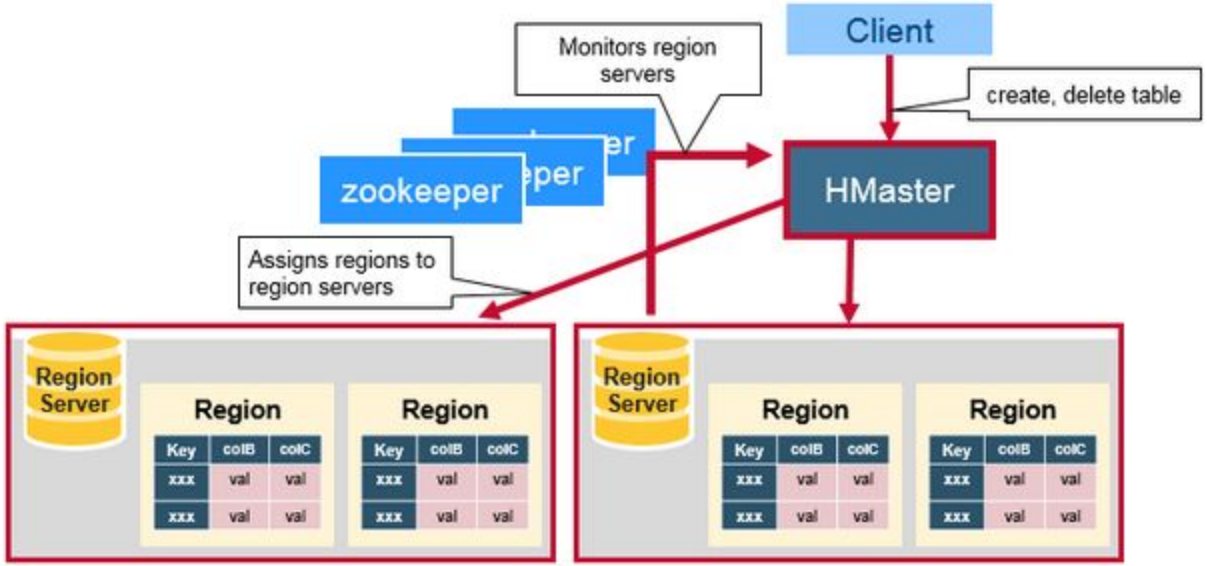
- MongoDB
  - réplication par master-slave (un noeud primaire, plusieurs noeuds secondaires)
  - réplication asynchrone:
    - possible de perdre une écriture confirmée
    - possible de lire des données désuètes
  - possible que plus d'un noeud se considère primaire (??)
    - toutes les anomalies sont possibles!
- Elasticsearch
  - réplication par master-slave
  - réplication synchrone
  - utilise un algorithme de consensus non prouvé
    - dans certaines condition de défaillance (même simple), toutes les anomalies sont possibles
  - on ne recommande pas de l'utiliser comme stockage primaire, mais plutôt comme index

Technologies - systèmes de BD - Clé-valeur

- Paradigme clé-valeur
  - une géante table d'association **clé → valeur** (`Array[Byte] → Array[Byte]`)
  - deux grandes familles:
    - clés ordonnées lexicographiquement
    - hash-ring
  - La base de donnée n'interprète pas la clé ni la valeur (au-delà de l'ordonnancement)
  - paradigme extrêmement simple:
    - taux de requêtes très élevé
    - capacité "scale-out" évidente (attention pour la famille hash-ring)
  - offre généralement très peu de fonctionnalités
  - certaines offrent des colonnes (**`clé → c1:v c2:v'`**)
  - certaines permettent de regrouper des colonnes afin de mieux contrôler l'accès disque
  - elles sont souvent la base sur laquelle des paradigmes plus évolués sont construits

Technologies - systèmes de BD - HBase

- HBase
  - Inspiré de BigTable (Google)
  - la distribution des clés est divisée en régions
  - chaque région est prise en charge par un noeud
  - un même noeud peut prendre en charge plusieurs régions
  - si un noeud n’est pas disponible, les clés dans cette région sont également non disponibles, mais le reste du système peut continuer d’opérer normalement
  - offre des “colonnes” qui peuvent être regroupées en “familles”
  - chaque valeur est un tuple: (`clé, famille, colonne, valeur, timestamp`)
  - on peut garder plusieurs “versions” d’une même valeur
  - offre des opérations atomiques au niveau d’une clé
    - `“compare and set”`

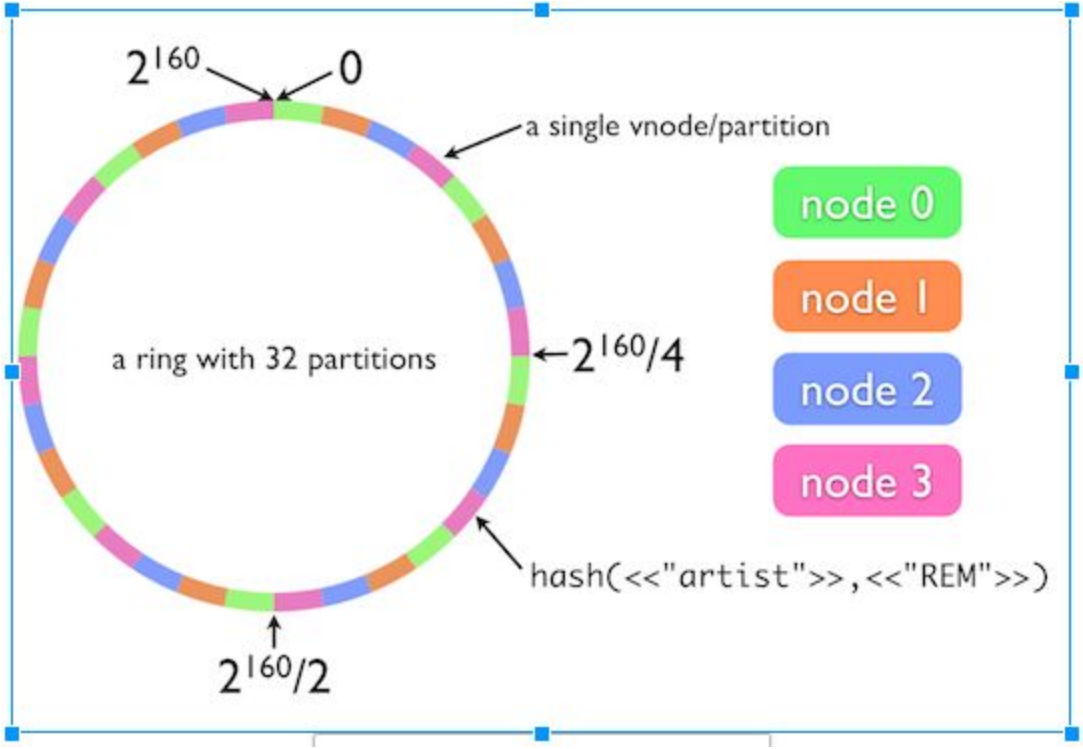

Client
Monitors region servers
create, delete table
zookeeper
HMaster
Assigns regions to region servers
Region Server
Region
Key
colB
colC
xxx
val
Region Server
Region
Key
colB
colC
xxx
val
Region
Key
colB
colC
xxx
val

Technologies - systèmes de BD - HBase

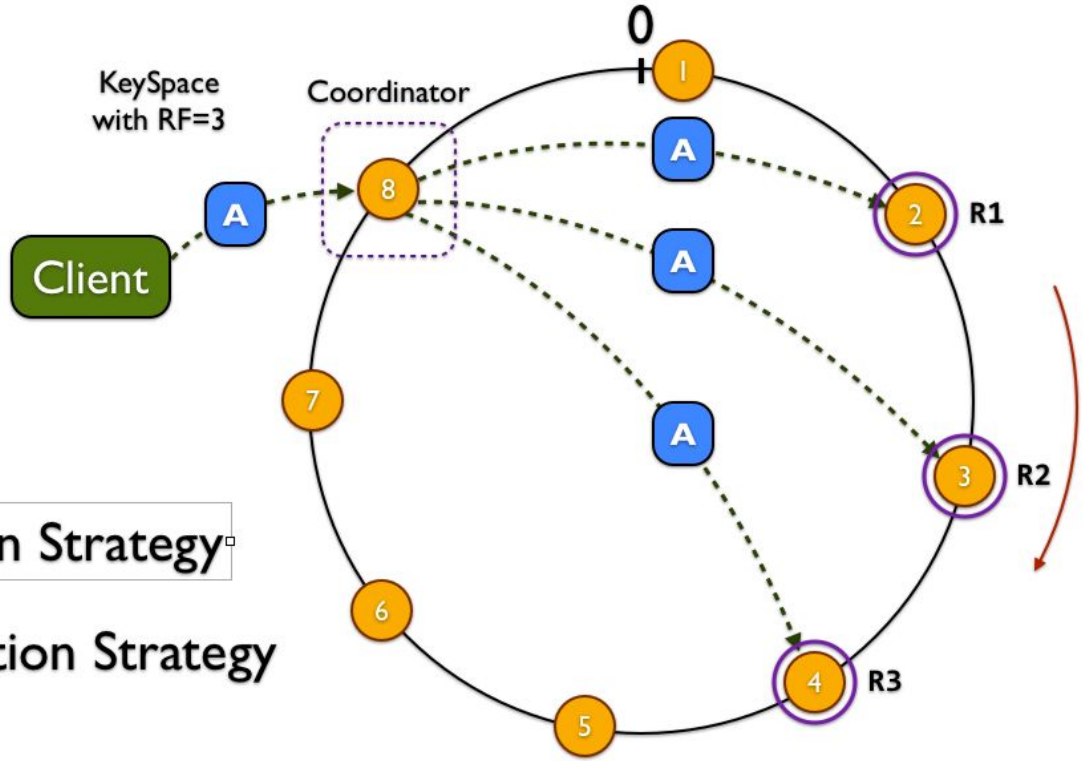
- HBase (cont.)
  - Nécessite une infrastructure HDFS
  - Utilise Zookeeper comme service de coordination
  - Installation et opérations plutôt lourdes
  - Implémenté en Java, donc peut nécessiter plusieurs manipulations de JVM pour réduire l’impact des GC
  - supporte la réplication asynchrone
  - certains cas d’usage sont gigantesques: Facebook Messenger (aux dernières nouvelles)

Technologies - systèmes de BD - Cassandra

- Cassandra
  - créé chez Facebook, inspiré de DynamoDB (Amazon)
  - chaque clé est hashée et ensuite assignée à plusieurs noeuds (réplication)
  - pour relire la valeur, on hash la clé et on interroge un (ou plusieurs) noeud qui devrait la posséder
  - si un noeud n'est pas disponible lors de l'écriture, on passe au suivant
    - mécanisme de "réparation" des clés doit alors être effectué
  - il est possible que 2 noeuds aient une valeur distincte pour une même clé
    - mécanisme de décision: quelle valeur est la bonne?
    - Cassandra utilise le "timestamp" ➜ dépend de la précision des horloges sur les noeuds


2^160
0
a single vnode/partition
node 0
node 1
node 2
node 3
a ring with 32 partitions
2^160/4
hash(<<"artist">>,<<"REM">>)
2^160/2

Partitioning and Replication

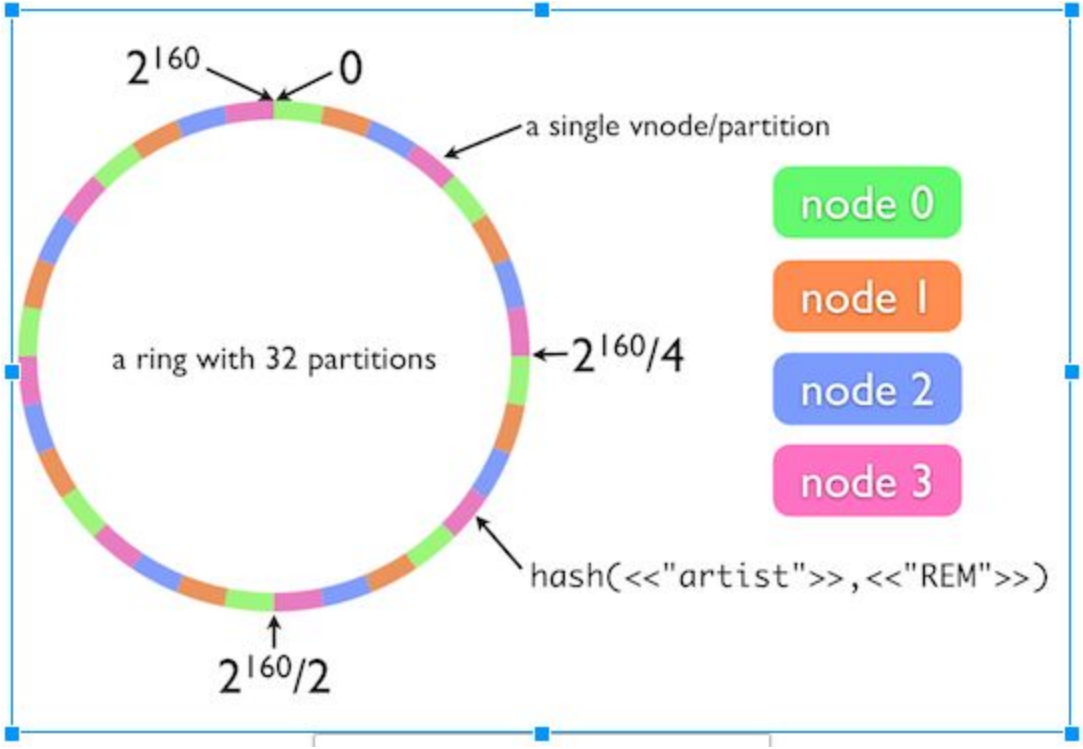


Partition Strategy

Replication Strategy

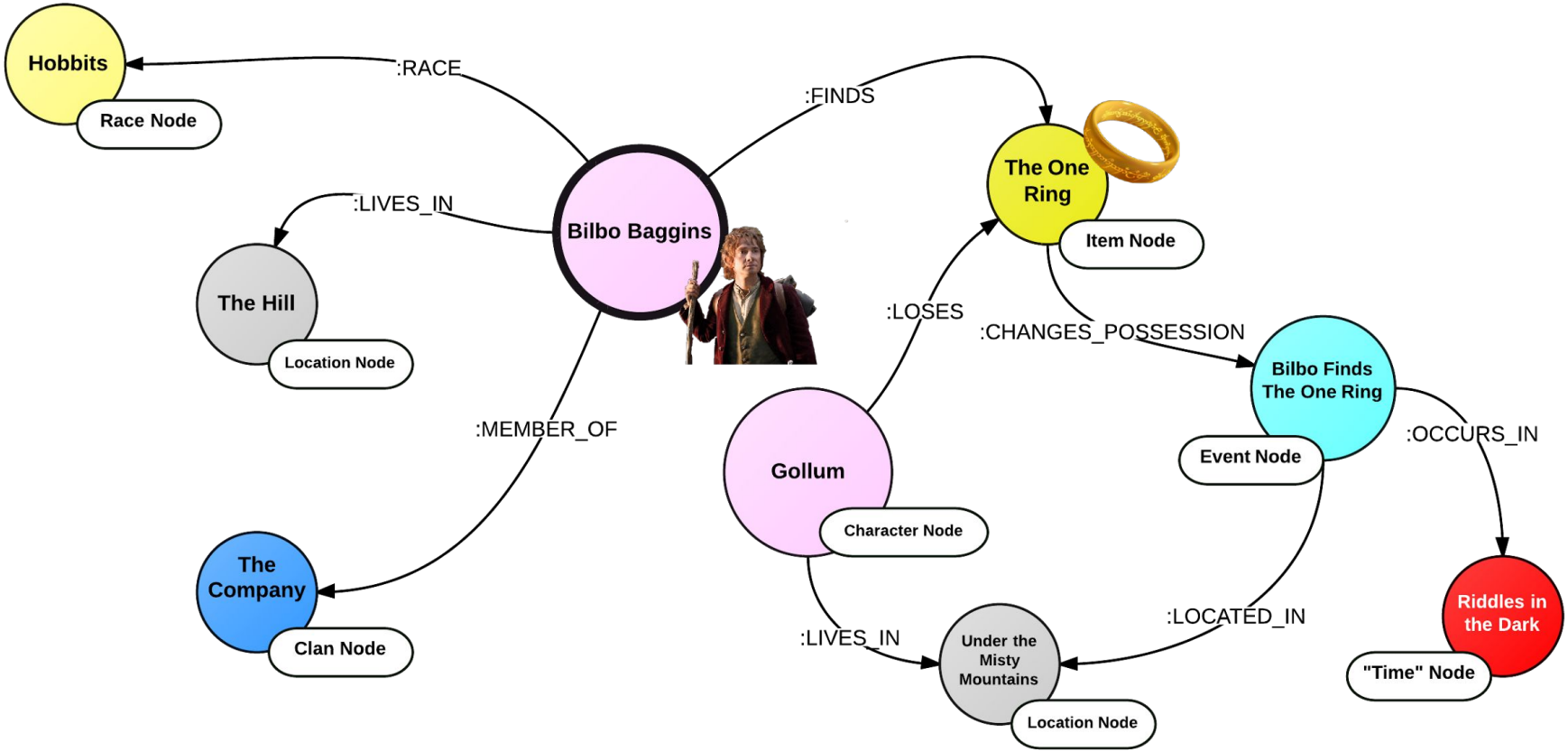
Technologies - systèmes de BD - Cassandra

- Cassandra (cont.)
  - offre des familles de colonnes
  - offre un langage d’interrogation simili-SQL: le CQL
  - capacité d’étendre un même cluster Cassandra dans plusieurs centres
  - installation et opérations très simples, aucune dépendance au-delà d’une JVM
  - un très gros utilisateur: Netflix
  - capacité de scale-out -- attention!


2^160
0
a single vnode/partition
node 0
node 1
node 2
node 3
a ring with 32 partitions
2^160/4
hash(<<"artist">>,<<"REM">>)
2^160/2

Technologies - systèmes de BD - Graphe

- Paradigme d’accès est celui de noeuds et liens (nodes and edges)
  - les noeuds peuvent représenter des concepts hétérogènes
  - les liens représentent les relations entre les noeuds
- Généralement pour des problèmes spécialisés
  - différent types de réseaux (sociaux, télécommunication, etc.), transport, routes
- Généralement implémenté sur un paradigme plus simple (clé-valeur ou autre)
- Permet des interrogations très complexes
- Distribuer un graphe n’est pas une mince affaire!
  - toujours le sujet de beaucoup de recherches


Hobbits
Race Node
:RACE
:FINDS
The One Ring
Item Node
:LIVES_IN
Bilbo Baggins
The Hill
Location Node
:LOSES
:CHANGES_POSSESSION
Bilbo Finds The One Ring
Event Node
:OCCURS_IN
:MEMBER_OF
Gollum
Character Node
The Company
Clan Node
:LIVES_IN
Under the Misty Mountains
Location Node
:LOCATED_IN
Riddles in the Dark
"Time" Node

Technologies - systèmes de BD - Graphe

- Peu de cas d’utilisation publique
  - Google, Facebook et Twitter utilisent des BD graph, mais ne partagent pas beaucoup de détails à leur sujet
- Facile d’imaginer l’utilité pour le domaine du marketing:
  - “Trouver les gens de 30 à 35 ans n’ayant pas d’enfant, qui ont eux-mêmes ainsi que leur conjoint manifesté un intérêt pour le surf dans les 2 dernières années, qui utilisent un téléphone haut de gamme, qui ont visité la page d’un voyageur dans les 30 derniers jours ayant du contenu relié au surf et dont au moins 2 amis ont voyagé à l’étranger dans les 3 dernières années.”

Technologies - systèmes de BD - Structurées

- Paradigme structuré ou semi-structuré
  - le paradigme tabulaire ou même relationnel (celui-là même que nous avions délaissé!)
  - offre généralement le concept de table et de colonnes typées (string, int, etc.)
  - certaines offrent même les garanties des “bonne vieilles” BD relationnelles
    - atomicité, isolation, etc.
  - certaines offrent le SQL, d’autres un langage simili-SQL

Technologies - systèmes de BD - Kudu

- Apache Kudu
  - architecture semblable à celle de HBase
  - différence principales:
    - ne dépend pas de HDFS pour répliquer les données
    - utilise un algorithme de consensus pour chaque région
    - offre un paradigme structuré (tables + colonnes typées)
    - meilleure performance des lectures séquentielles (scan)
    - implémenté en C++ plutôt que Java
  - les clés sont ordonnées ou distribuées uniformément (hash)
  - compatible avec l’écosystème Hadoop

Technologies - systèmes de BD - CockroachDB

- CockroachDB
  - pourrait aussi être dans la famille des “NewSQL”
  - système multi-paradigme (clé-valeur, semi-structuré, structuré), mais seul le dernier niveau est accessible depuis les clients
  - système distribué qui offre toutes les fonctionnalitées des bases de données traditionnelles
    - utilise même le protocole d’une d’elles: PostgreSQL
  - inspiré du système Spanner créé chez Google
    - un système qui nécessite des horloges atomiques et des GPS!
    - chaque écriture doit attendre au moins autant de temps que la différence permise entre les horloges (environ 10 ms avec l’équipement spécialisé)
  - CockroachDB fait un choix différent de celui de Spanner qui permet d’opérer avec des horloges normales
    - La lecture d’une clé **récemment modifiée** doit attendre au moins aussi longtemps que la différence permise entre les horloges (environ 100 ms généralement)

Technologies - systèmes de BD - CockroachDB

- CockroachDB (cont)
  - se veut une base de données géo-répliquée
    - opérée dans plusieurs centres de données distants
  - projet encore très jeune (v1.0 mai 2017) qui doit encore faire ses preuves
  - probablement moins performant, mais a beaucoup plus de potentiel d'évolution qu'une BD traditionnelle qui sera toujours limitée à la capacité d'un seul noeud

Technologies - systèmes de BD - Autres

- D’autres paradigmes existent
- Certaines sont dans leur propre catégorie
  - Redis: base de données de structure de données (list, set, queue, stack, etc.)
  - Zookeeper: service de coordination de systèmes distribués
    - Obtention de “lock” distribué
    - Barrières distribuées
    - Élection de participant primaire (leader election)
    - Généralement utilisé pour coordonner l’état d’un système < 1MB entre plusieurs noeuds
  - etcd:
    - Conceptuellement similaire à Zookeeper
    - Implémenté en Go plutôt que Java
    - Beaucoup plus performant


LOST
CONFUSED
UNSURE
PUZZLED
#129985691

Technologies - systèmes de base de données

- Comment choisir?!?
- Parmi toutes les dimensions pour catégoriser les BD, 2 sont particulièrements fondamentales
  - **le paradigme d'accès / stockage**
  - **les garanties de cohérence des données**
- Le choix du paradigme d'accès peut être guidé par:
  - les besoins d'affaire
    - doit-on supporter des requêtes ad-hoc?
    - a-t-on besoin d'un schéma?
    - doit-on supporter des requêtes qui croisent des dizaines de domaines?
  - la familiarité des systèmes / opérations
    - si 100% des gens sont familiers avec SQL, peut-être utiliser SQL?
    - Hadoop est déjà en place et en opération: HBase / Kudu?
    - utilisation d'un langage dynamique (JavaScript) ➜ Document JSON

Technologies - systèmes de base de données

- Le choix du paradigme d’accès peut être guidé par:
  - Performances:
    - quel taux de requêtes doit-on supporter?
    - peut-on accepter des lectures plus lentes au profit d’une application plus simple?
  - Risques:
    - qui pourra nous aider si le système “brise”?
    - a-t-on des garanties de disponbilité (p. ex.: 99,9% disponible)?

Technologies - systèmes de base de données

- Probablement le paramètre le plus important: la garantie de **cohérence des données**
- Les systèmes distribués doivent faire un choix entre la cohérence ou la disponibilité des données lorsqu'une défaillance survient:
  - **strong-consistency** vs. **eventual-consistency**
- Strong consistency:
  - sémantique "read-your-own-write": après avoir écrit on est garanti de relire la même donnée ou une donnée plus récente
- Eventual consistency:
  - après avoir écrit, on peut relire cette même donnée, une donnée plus récente ou **l'ancienne donnée**

Technologies - systèmes de base de données

- Chaque système de base de donnée est dans un camp ou dans un autre
- Certaines offrent une garantie dite “ajustable”, mais le système en soi est fondamentalement l’un **ou** l’autre

- La garantie de cohérence des données ne peut pas changer dans le cours de vie d’une application **sans une ré-architecture fondamentale**…

Technologies - systèmes d’orchestration

Technologies - systèmes d’orchestration

- Systèmes distribués sont nécessairement plus complexes à opérer
  - Installation et mise à jour
  - surveillance (monitoring)
  - détection et basculement lors de défaillance
  - backup / restore
  - utilisation des ressources disponibles
- Tout doit s’effectuer sur plusieurs noeuds et souvent de manière coordonnée
- Opérer ces systèmes manuellement n’est ni recommandé, ni souhaitable
- De nouvelles plateformes émergent pour répondre à ces besoins
  - YARN, Mesos, Kubernetes, Docker Swarm

Conclusion

- Les 3 Vs du Big Data ont poussé les limites des systèmes traditionnels
- Nécessaire d’opérer sur plusieurs noeuds en parallèle
- De nouvelles techniques et technologies ont vu le jour
  - systèmes de fichiers distribués
  - algorithmes distribués
  - systèmes de base de données distribués
- Les systèmes distribués sont complexes
  - doivent faire des compromis
  - ces compromis dictent les caractéristiques fondamentales de ces systèmes
  - important de bien les connaître
- De nouvelles plateformes émergent pour simplifier la gestion de ces systèmes