

Master 1

FRAMEWORK LOGICIEL POUR LE BIG DATA

COURS 2

R.JAZIRI

PLAN DU COURS

- Big data, les fondamentaux
- Introduction aux fondements technologiques (Hadoop, ...),
- Les Bases de données Nosql
- Stratégies de valorisation des données en entreprise (typologie générale, focus sectoriels, études de cas concrètes...)

Framework Hadoop

- Framework Hadoop
- Le pattern MapReduce
- HDFS

Données massives...

Données brutes (non structurées,)

Quantités importantes

La lenteur des disques durs...

HDD : 0.05 GO par seconde (50 MO/sec)

RAM : 5 GO par seconde

Pour y pallier : deux grandes familles de solutions

1. La Parallélisation

2. Le travail in memory

HADOOP ?



- Projet Open source
- Ecrit en Java
- Optimisé pour gérer:
 - Grand volume de donnée avec traitement en parallèle
 - Variété de donnée (structuré, non structuré, semi structuré)
 - Utilisation de matériels commode pas cher,
- Très bonne performance
- N'est pas adapté aux traitement interactive en ligne (OLTP)
- N'est pas adapté aux traitement analytique en ligne (OLAP)
- N'est pas adapté aux applications décisionnelles
- Ce n'est pas un système pour remplacer le modèle relationnel

HADOOP s'est imposé

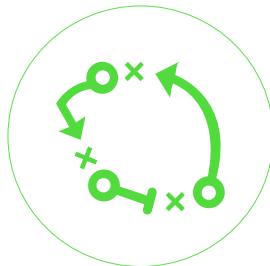
« Hadoop est un **framework open source** de traitement de données évolutif pour **le stockage et le traitement par lot de très grande quantité de données**. C'est un outil permettant la mise en cluster de calcul de manière la plus simple possible. »



Hadoop est extrêmement adapté pour :



Calculer la taille de plusieurs milliers de documents.



Trouver le nombre d'occurrence d'un pattern dans un très grand volume de données.



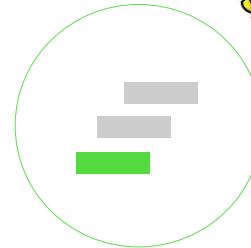
Classifier des très grands volumes de données provenant des paniers d'achats de clients.

Synthèse qu'apporte Hadoop : une rupture en quatre points :



RAPPORT PERFORMANCE / PRIX

- Traiter des requêtes en minutes sur des To, et en utilisant un système de serveur très bon marché
- La data est répartie intelligemment. Il n'est même plus nécessaire de la gérer



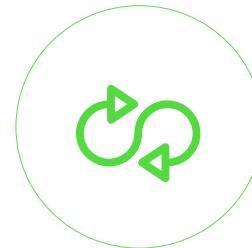
SCALABLE

- Rajouter des ordinateurs sans avoir à refaire toute l'architecture
- Processer des Petabytes de données sur un seul cluster



Tolérance

- Information imprimée 3 fois et en fragments.
- Si une machine avec une version est en panne, une autre a une copie de la paire key/value, ce qui permet de résoudre la même sous-tâche



SOUPLESSE

- Stocker et analyser des données structurées et non structurées



HADOOP

- Le projet Hadoop consiste en deux grandes parties:
 - Stockage des données : HDFS (Hadoop Distributed File System)
 - Traitement des données : MapReduce
- Principe :
 - Diviser les données
 - Les sauvegarder sur une collection de machines, appelées cluster
 - Traiter les données directement là où elles sont stockées, plutôt que de les copier à partir d'un serveur distribué
- Il est possible d'ajouter des machines au cluster, au fur et à mesure que les données augmentent,

Parallélisation via architecture distribuée

Vue globale



Système distribué

Un nœud = un processeur + un disque

Un cluster pilote plusieurs nœuds en parallèle

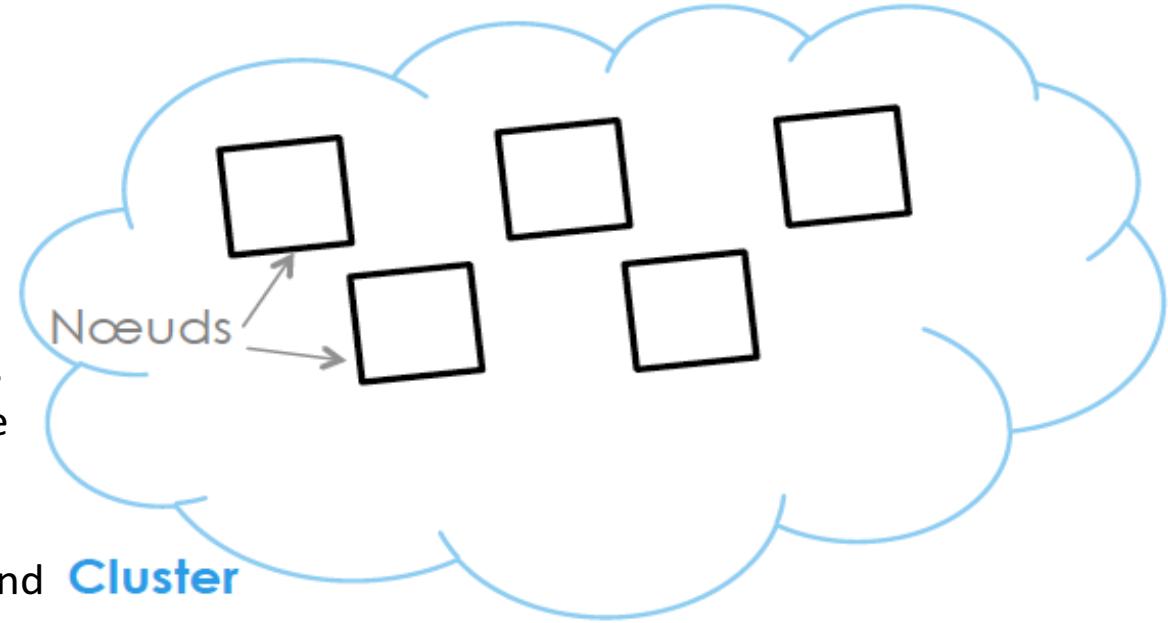
Hadoop distribue les DATA par nœuds selon HDFS

Map distribue le TRAVAIL par nœuds selon HDFS

Reduce recueille et réduit les contributions des nœuds

Parallélisation via architecture distribuée

HDFS : Hadoop Distributed File System

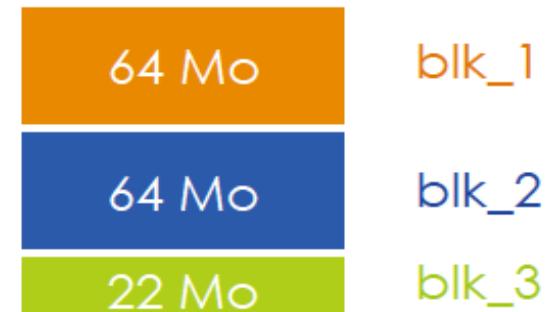


HDFS est un système de fichiers distribué, extensible et portable
Écrit en Java

Permet de stocker de très gros volumes de données sur un grand nombre de machines (noeuds) équipées de disques durs banalisés (Cluster)

Quand un fichier mydata.txt est enregistré dans HDFS, il est décomposé en grands blocs (par défaut 64Mo), chaque bloc ayant un nom unique: blk_1, blk_2...

mydata.txt (150 Mo)

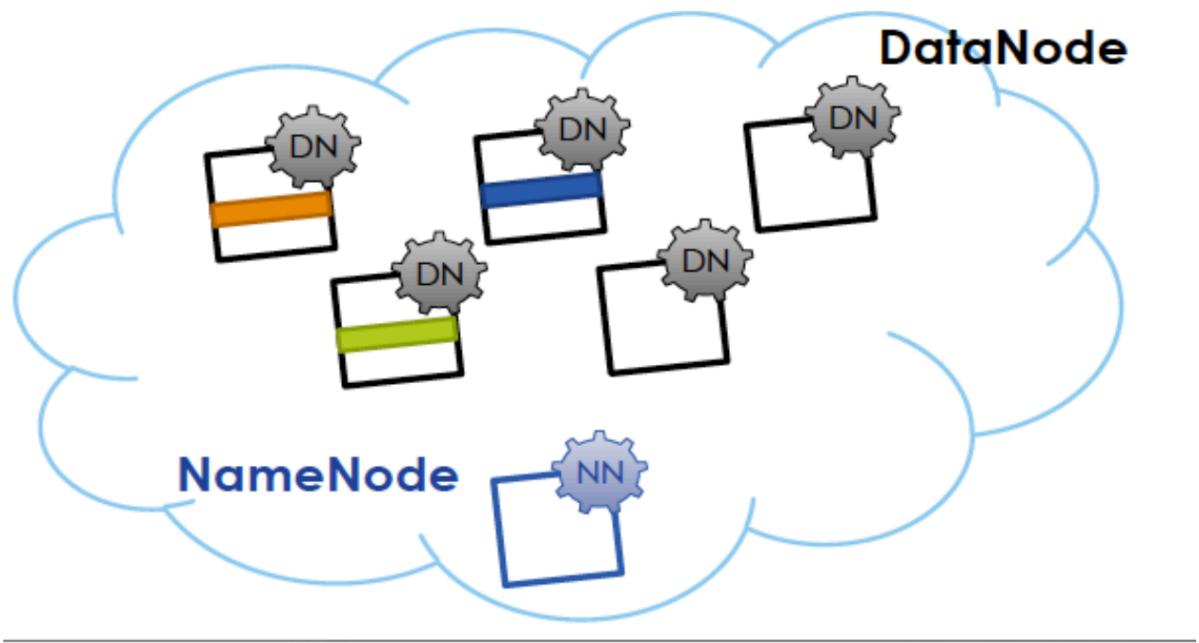


Parallélisation via architecture distribuée

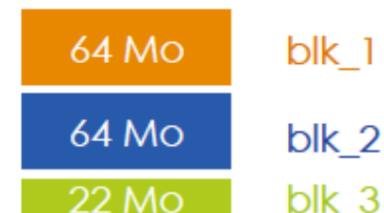
HDFS : Hadoop Distributed File System



- Chaque bloc est enregistré dans un noeud différent du cluster
- **DataNode** : démon sur chaque noeud du Cluster
- **NameNode** :
 - Démon s'exécutant sur une machine séparée
 - Contient des métadonnées
 - Permet de retrouver les noeuds qui exécutent les blocs d'un fichier

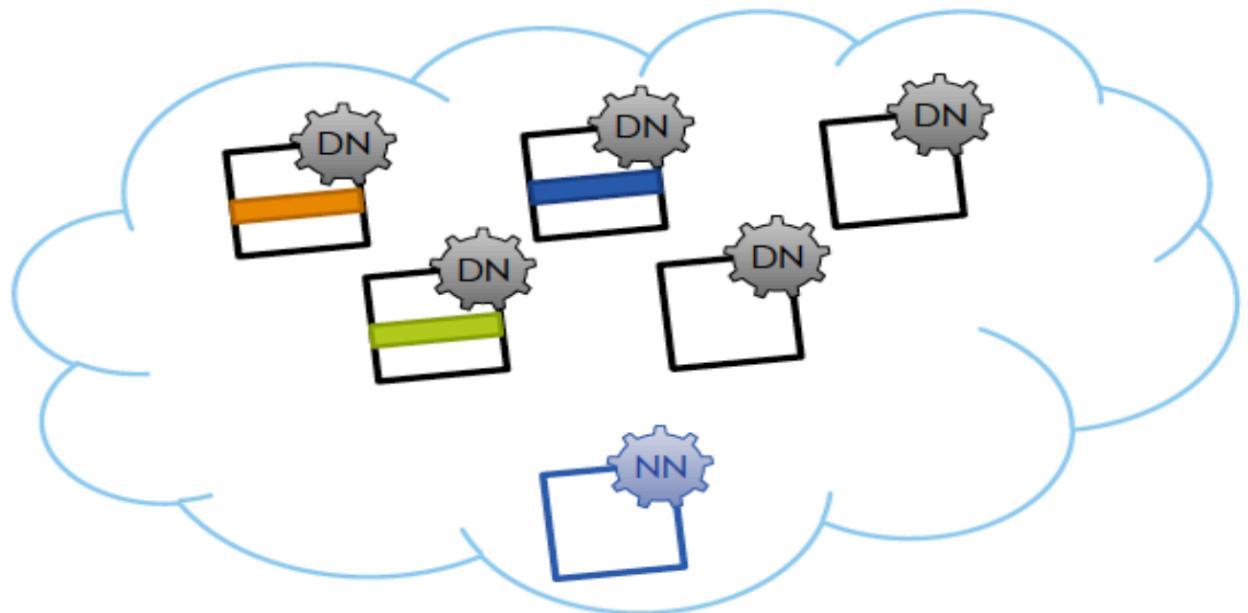


mydata.txt (150 Mo)



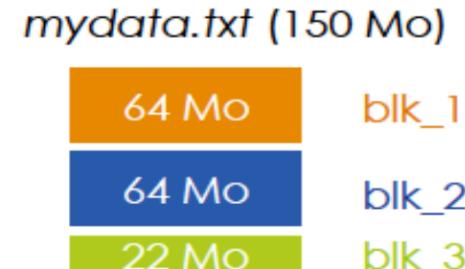
Parallélisation via architecture distribuée

HDFS : Hadoop Distributed File System



Quels sont les problèmes possibles?

- Panne de réseau ?
- Panne de disque sur les DataNodes ?
- Pas tous les DN sont utilisés ?
- Les tailles des blocks sont différentes ?
- Panne de disque sur les NameNodes ?



Parallélisation via architecture distribuée

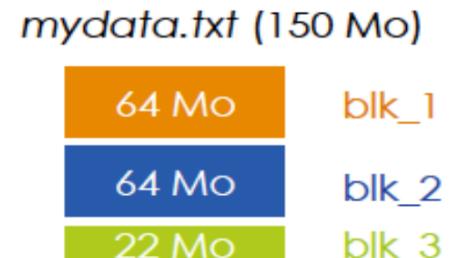
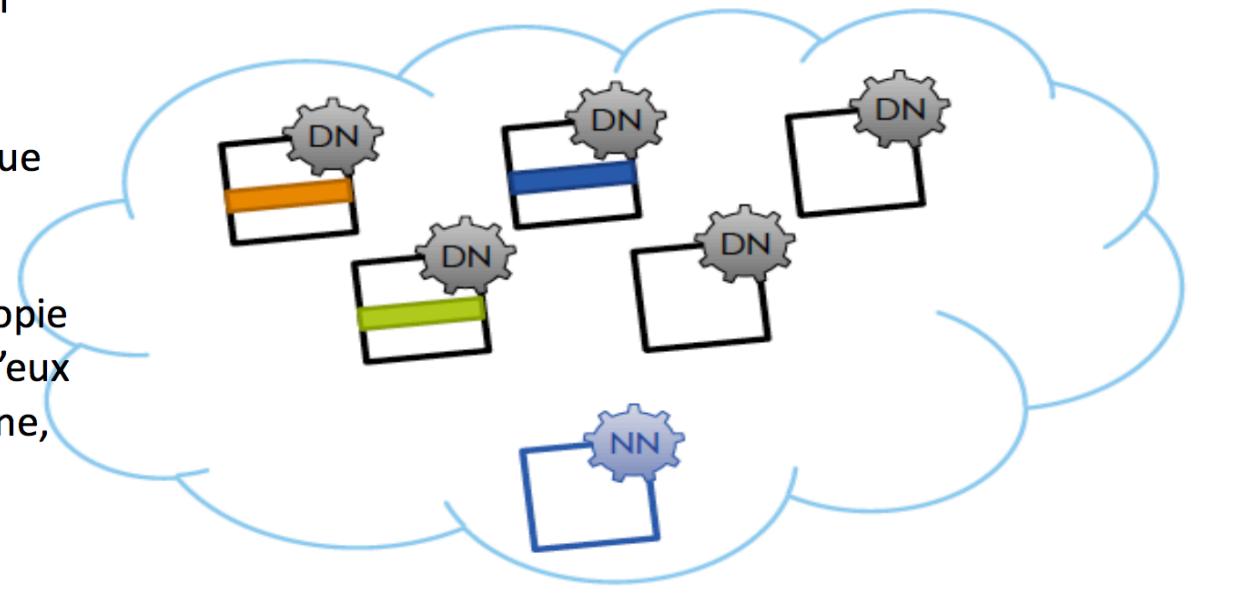
HDFS : Hadoop Distributed File System



- Si l'un des noeuds a un problème, les données seront perdues

- Hadoop réplique chaque bloc 3 fois
- Il choisit 3 noeuds au hasard, et place une copie du bloc dans chacun d'eux
- Si le noeud est en panne, le NN le détecte, et s'occupe de répliquer encore les blocs qui y étaient hébergés pour avoir toujours 3 copies stockées

- Concept de Rack Awareness (rack = baie de stockage)
- Si le NameNode a un problème ?

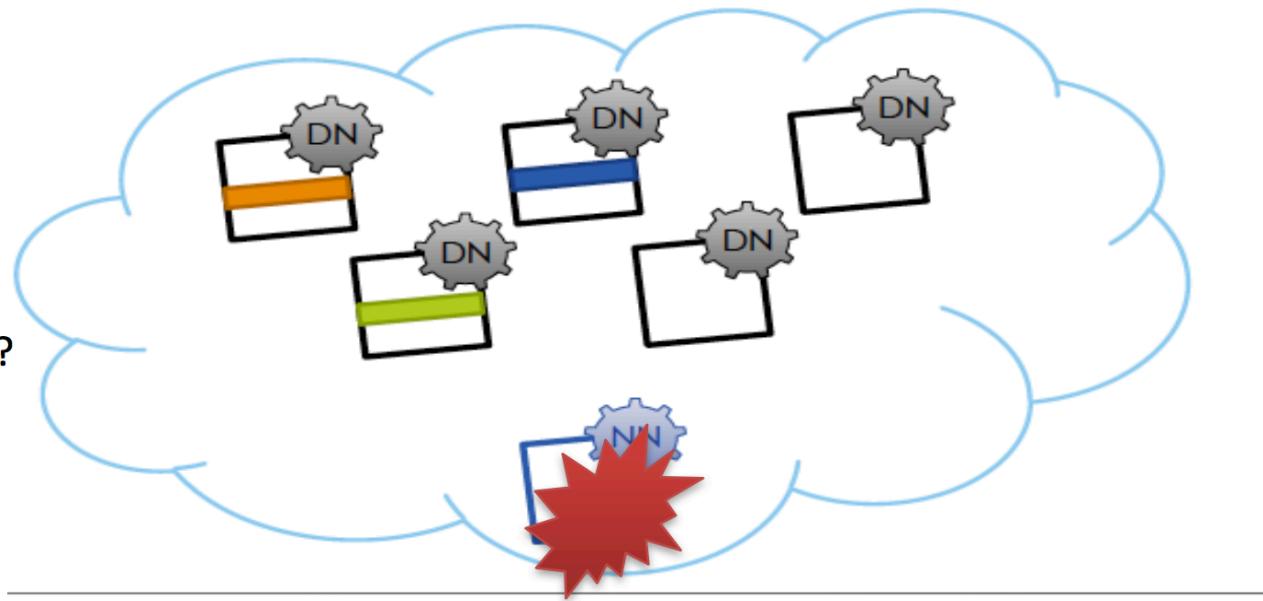


Parallélisation via architecture distribuée

HDFS : Hadoop Distributed File System



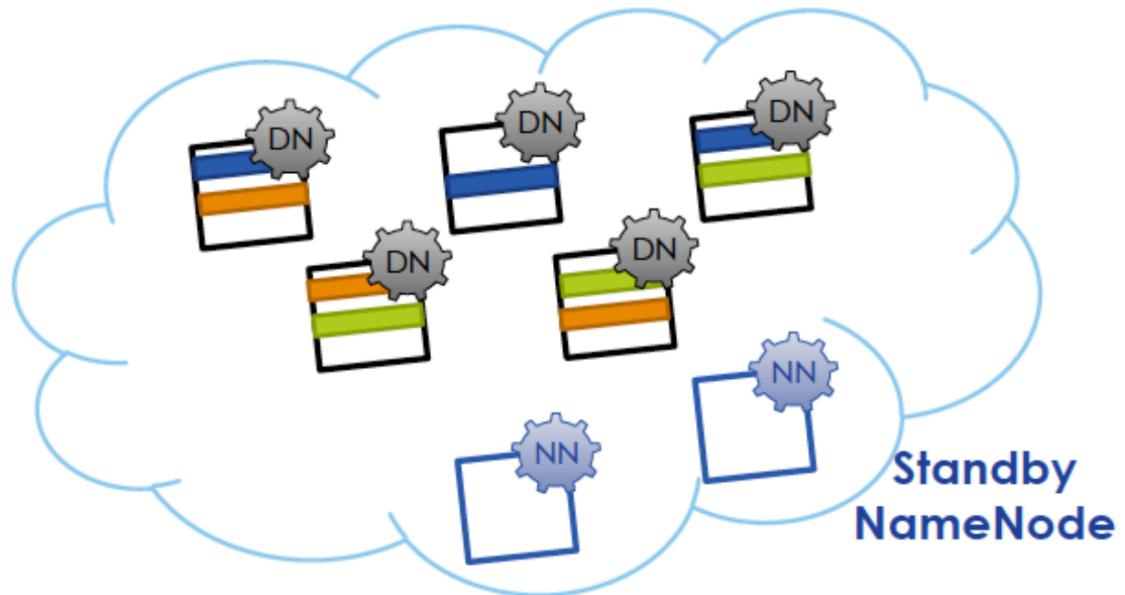
- Si le NameNode a un problème :
 - Données inaccessibles?
 - Données perdues à jamais**
 - Pas de problème**
- Si c'est un problème d'accès (réseau), les données sont temporairement inaccessibles
- Si le disque du NN est défaillant, les données seront perdues à jamais



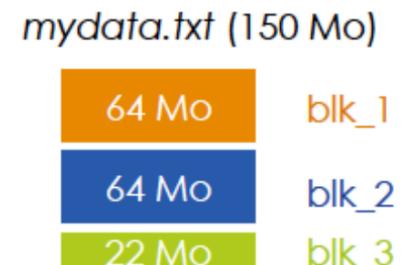
<i>mydata.txt (150 Mo)</i>		
64 Mo	<i>blk_1</i>	
64 Mo	<i>blk_2</i>	
22 Mo	<i>blk_3</i>	

Parallélisation via architecture distribuée

HDFS : Hadoop Distributed File System



Pour éviter cela, le NameNode sera dupliqué, non seulement sur son propre disque, mais également quelque part sur le système de fichiers du réseau
Définition d'un autre NN (standby namenode) pour reprendre le travail si le NameNode actif est défaillant

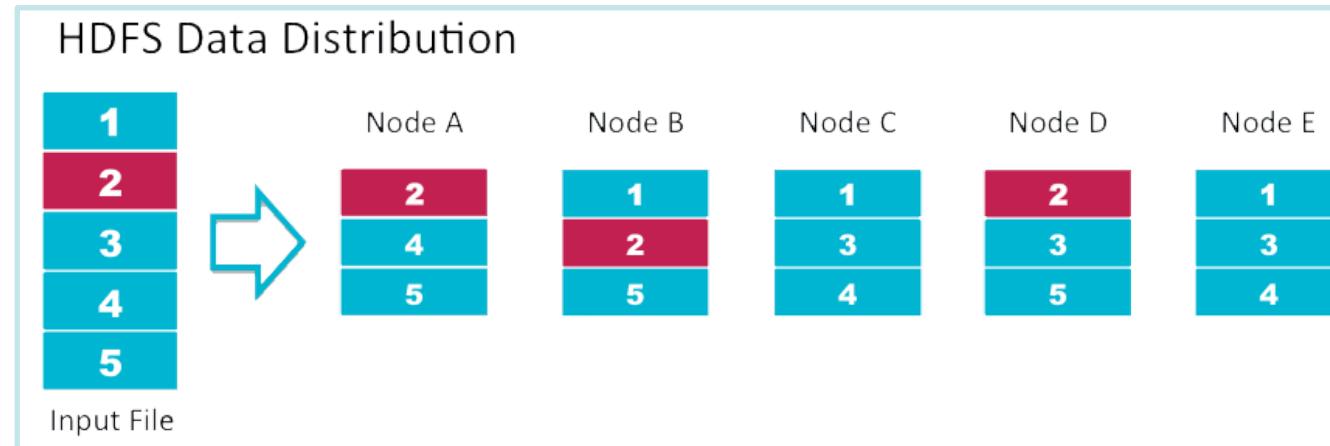


Parallélisation via architecture distribuée

HDFS : Hadoop Distributed File System



- **Architecture « Scale-Out »** - Ajouter des serveurs et accroître la capacité de la solution
- **Haute disponibilité** - Servir les flux de travail et des applications critiques
- **Tolérance aux erreurs** - Récupération des échecs, automatique et transparente
- **Accès flexible** - Utilisable par de multiples « **Framework** » ouverts.
- **Équilibrage de charge** - Disposition intelligente des données pour un efficacité et une utilisation maximale.
- **RéPLICATION modulable** - Plusieurs copies de chaque fichier assurent la protection des données et les performances de calcul.
- **Sécurité** - les permissions sont organisées par utilisateurs et groupes avec l'intégration de LDAP en option.



LE HDFS : LE STOCKAGE

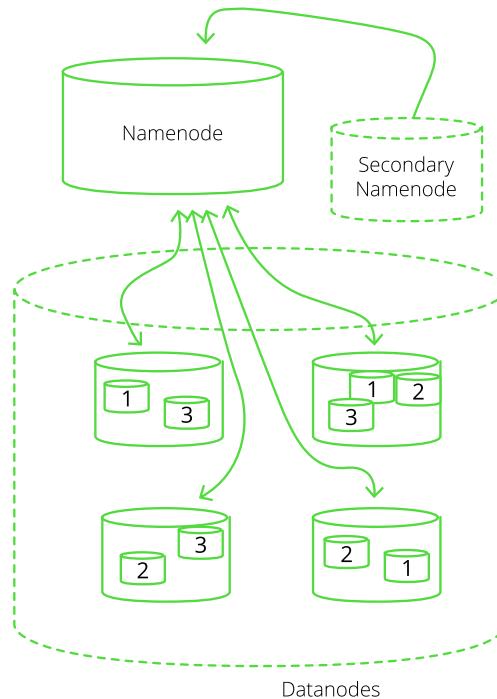
HDFS permet de gérer intelligemment des données en exploitant au maximum les machines serveurs en réseaux



Ex : Réaliser un comptage du nombre d'utilisateurs inscrits sur Facebook.

- Découpage de la liste en plusieurs parties, chaque partie étant stockée sur une grappe de serveurs différente.
- **Le traitement est réparti sur l'ensemble des nœuds de calcul**

Hadoop stocke effectivement toute l'information **trois fois**, et **en fragments** distribués sur les différentes machines du cluster.



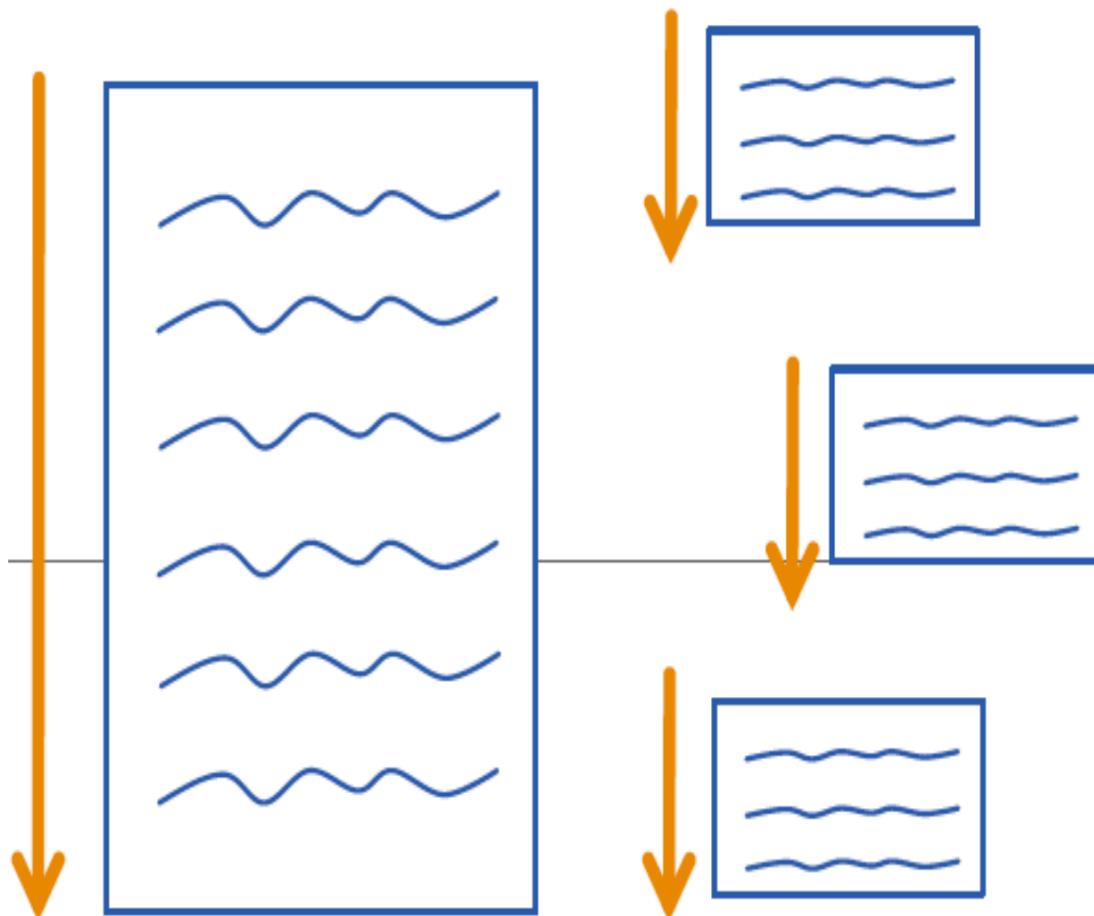
Au lieu d'adosser le traitement à une grappe unique, comme c'est le cas pour une architecture plus traditionnelle, cette distribution de l'information permet ainsi de **répartir ce traitement sur l'ensemble des nœuds de calcul sur lesquels la liste est répartie**.

Parallélisation via architecture distribuée

Map + Reduce : Stratégie d'accès et de reconstitution



- ❑ Patron d'architecture de développement permettant de traiter des données volumineuses de manière parallèle et distribuée
- ❑ A la base, le langage Java est utilisé, mais grâce à une caractéristique de Hadoop appelée Hadoop Streaming, il est possible d'utiliser d'autres langages comme Python ou Ruby
- ❑ Au lieu de parcourir le fichier séquentiellement (bcp de temps), il est divisé en morceaux qui sont parcourus en parallèle.



Parallélisation via architecture distribuée

Map + Reduce : Stratégie d'accès et de reconstitution

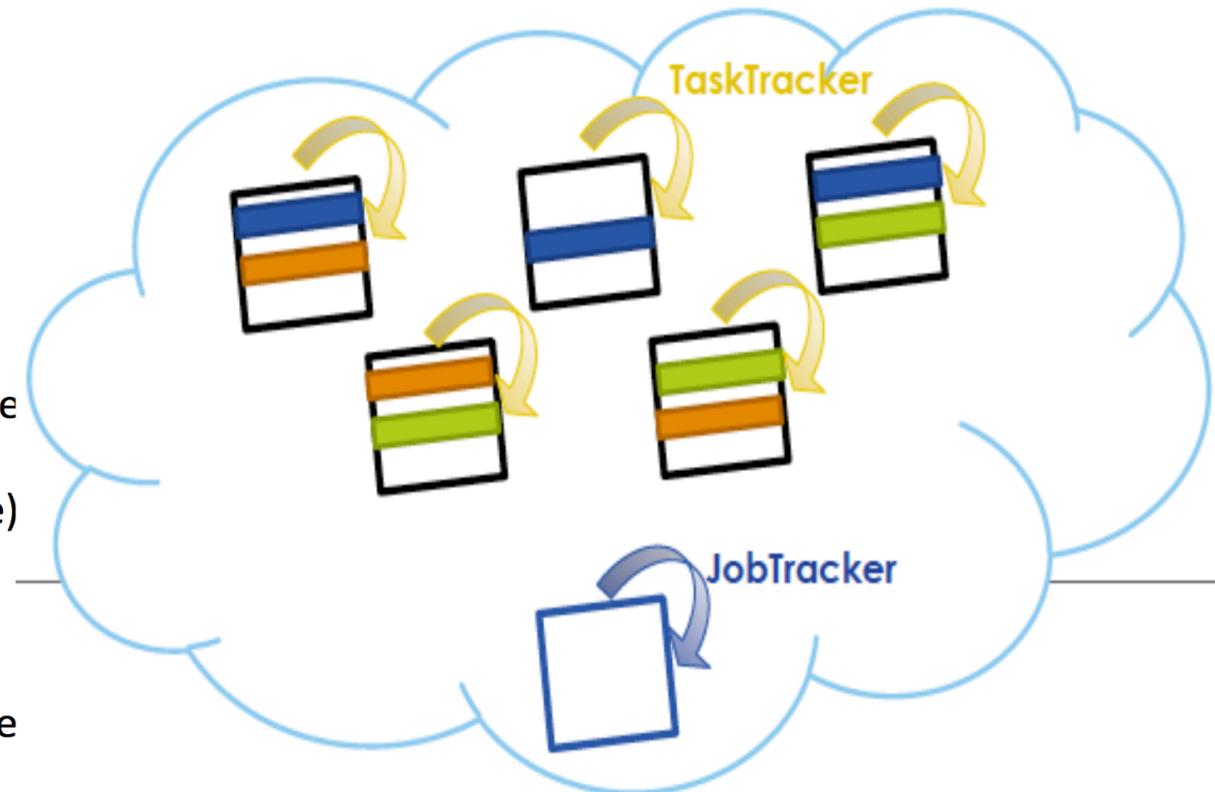


- **JobTracker**

Divise le travail sur les
Mappers et Reducers,
s'exécutant sur les
différents Nœuds

- **TaskTracker**

- S'exécute sur chacun des noeuds pour exécuter les vraies tâches de Map -Reduce
- Choisit en général de traiter (Map ou Reduce) un bloc sur la même machine que lui
- S'il est déjà occupé, la tâche revient à un autre tracker, qui utilisera le réseau (rare)



MAPREDUCE : Le traitement des données

LA DISTRIBUTION ET LA GESTION DES CALCULS EST RÉALISÉ PAR MAPREDUCE. CETTE TECHNIQUE SE DÉCOMPOSE EN 3 ÉTAPES PRINCIPALES :



Unité de calcul simple, avec une entrée et une sortie.

Ex. Compter le nombre de fois qu'il y a le mot « health » dans un document. Chaque ordinateur compte le nombre de « health », et lui associe le chiffre 1 dans la liste.

En JSON :

“health” { “helath” => 1
 “helath” => 1 }

Réarrange les éléments de la liste afin de préparer le Reducing.

Ex. Il est probable que le mot « health » le soit plus présent dans un serveur que dans un autre.

Génère le résultat final.

Ex. Récupérer l'ensemble des « health » : et Hadoop va faire la somme

En JSON :

{ “ health” = 15 }
 { “ helath ”= 49 } { “ health” = 81
 { “ health ”= 17 }

Parallélisation via architecture distribuée

Map + Reduce : Stratégie d'accès et de reconstitution



Sécurité et authentification: Fonctionne avec les droits sur les fichiers HDFS pour s'assurer que seuls les utilisateurs autorisés peuvent utiliser les données dans le système.

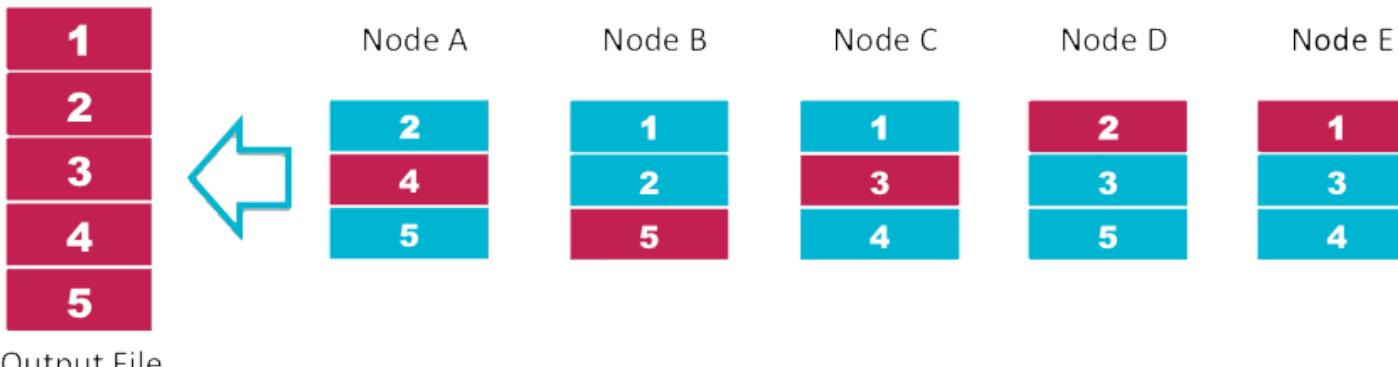
Resource Manager : Utilise les données les plus proches pour déterminer les opérations de calcul optimales .

Planification optimisée: Possibilité de gérer des priorités en fonction des “jobs”

Souplesse : Les procédures peuvent être écrites dans de nombreux langages de programmation ([java](#) ,[python](#), [R](#), ...)

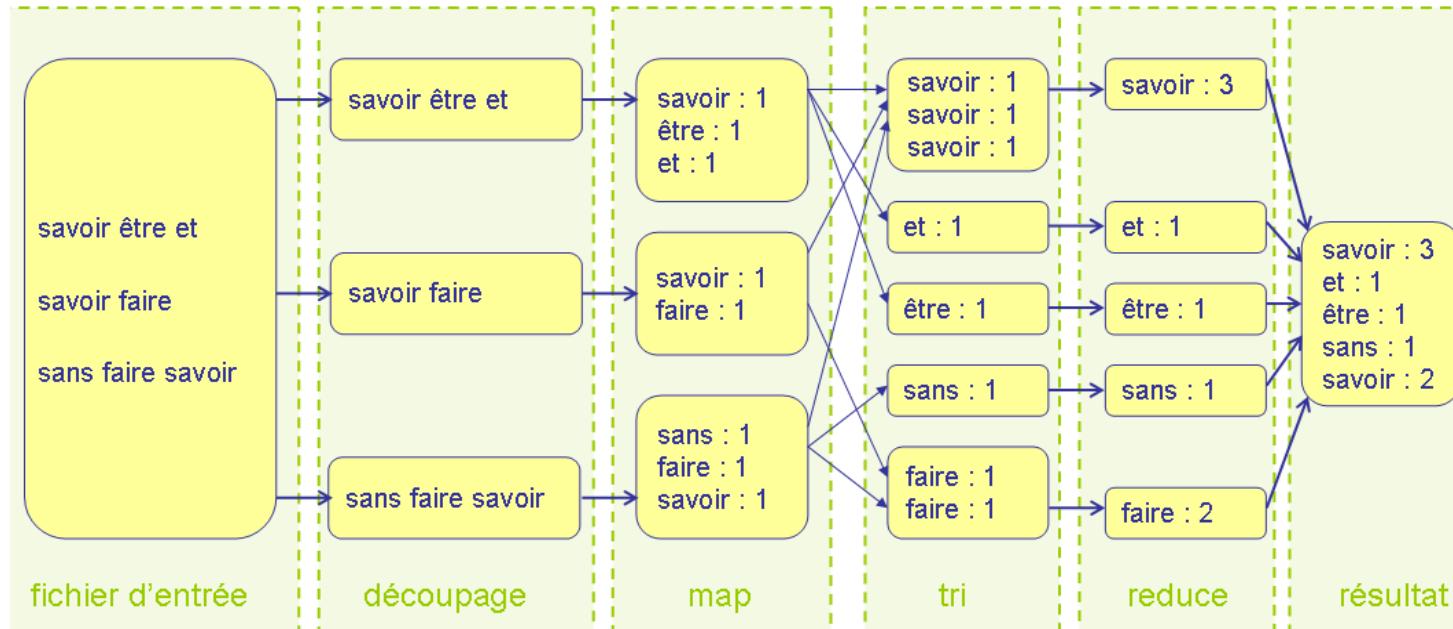
Haute disponibilité et backup: de nombreux services assurent le redémarrage et la continuité des traitements indépendamment des erreurs potentielles.

MapReduce Compute Distribution



Parallélisation via architecture distribuée

Map + Reduce : Stratégie d'accès et de reconstitution



Destiné à faciliter et simplifier le traitement sur de grandes quantités de données, en parallèle et sur de grandes grappes de machines avec une tolérance aux pannes.

Parallélisation via architecture distribuée

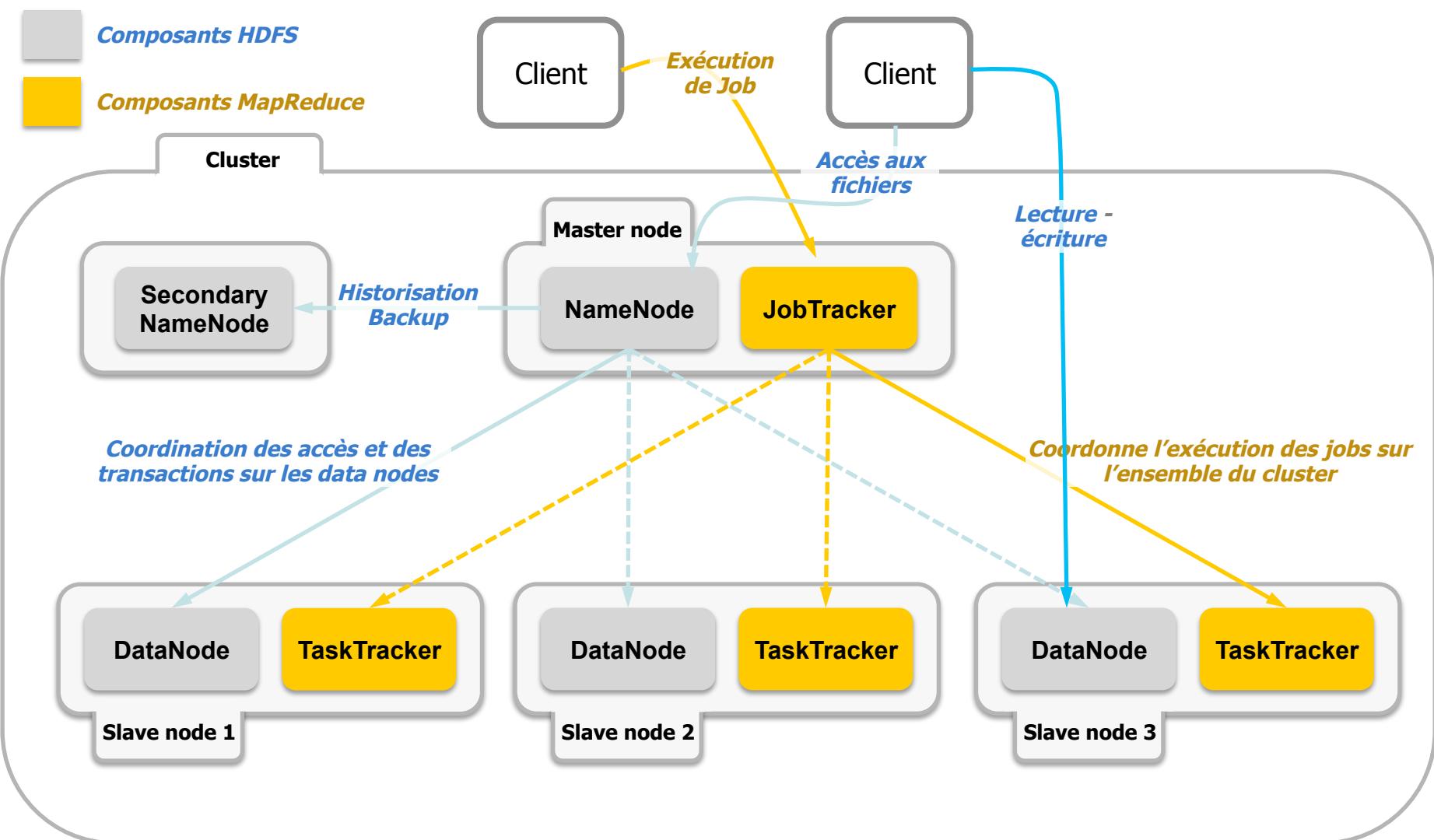
Map + Reduce : Exemple

```
function map(String name, String document):
    // name: document name
    // document: document contents
    for each word w in document:
        emit (w, 1)

function reduce(String word, Iterator partialCounts):
    // word: a word
    // partialCounts: a list of aggregated partial counts
    sum = 0
    for each pc in partialCounts:
        sum += ParseInt(pc)
    emit (word, sum)
```

Parallélisation via architecture distribuée

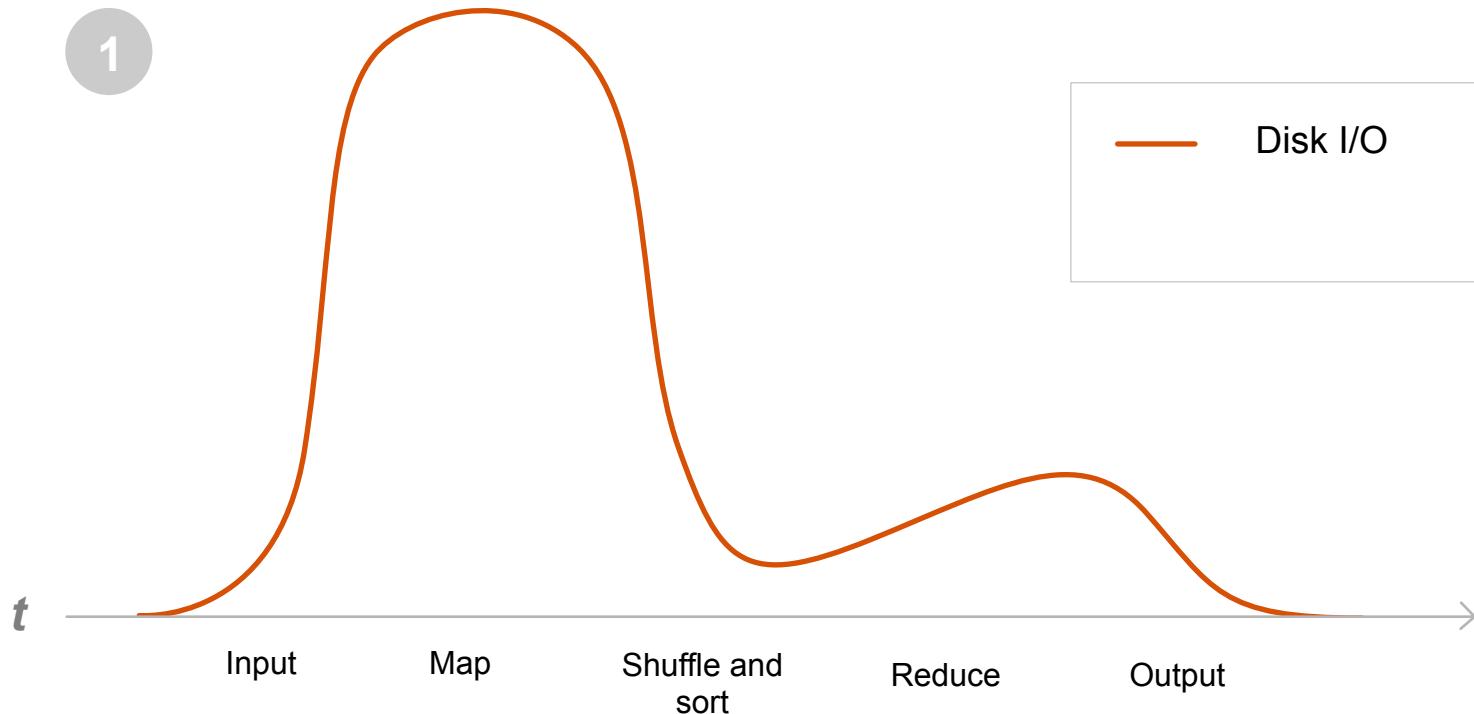
Organisation de la plateforme



Consommation des ressources pendant le « MAP »

Capacité de calcul, bande passante, stockage des données, tout à explosé...**sauf le temps d'écriture (Disk I/O).**

Le **Map Reduce** divise justement l'écriture linéairement par le nombre du nœud : **10 ordinateurs qui font 10 % du travail, vont 10 fois plus rapidement qu'un ordinateur tout seul.**

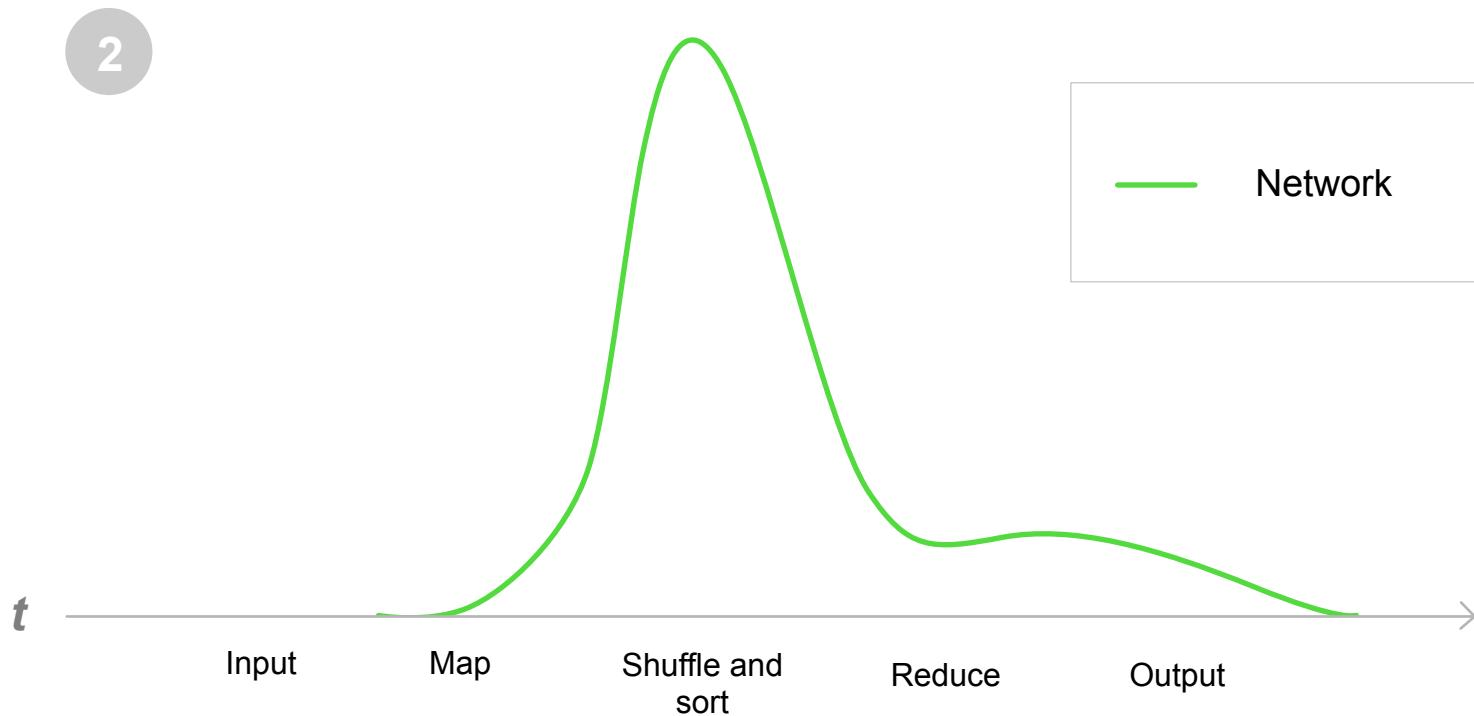


Durant la phase map, la ressource nécessaire la plus utilisée est principalement disk I/O.

L'utilisation des ressources pendant le « SHUFFLE »

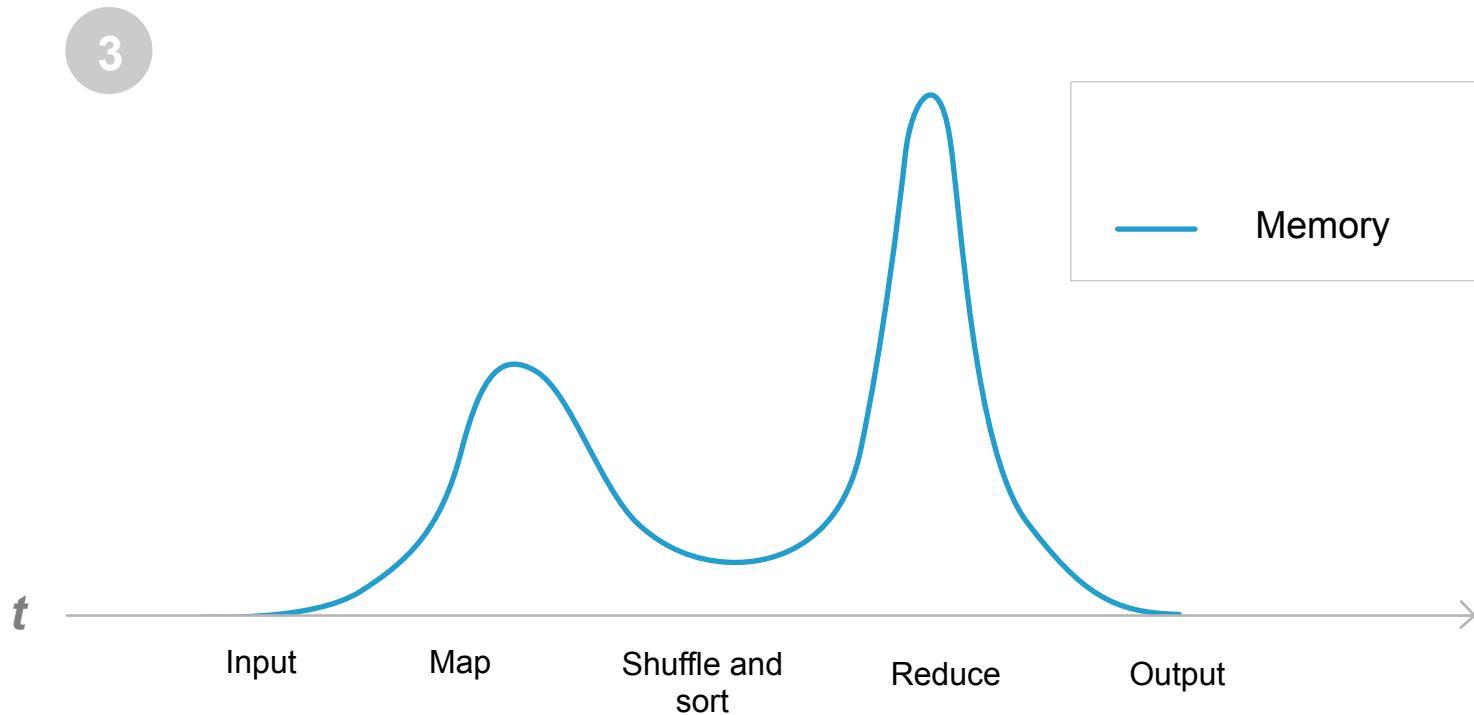
Pendant la partie Shuffle, **chaque machine travaille de manière autonome.**

La charge de travail est répartie en utilisant le réseau.



'utilisation des ressources pendant le « REDUCE »

Pendant la **phase de Reduce**, la principale ressource utilisée est la mémoire. **Reduce permet de distribuer l'opération en Mémoire**



Inconvénient d'HADOOP



- **Difficulté d'intégration avec d'autres systèmes informatiques:** le transfert de données d'une structure Hadoop vers des bases de données traditionnelles est loin d'être Trivial
- **Administration complexe :** Hadoop utilise son propre langage. L'entreprise doit donc développer une expertise spécifique Hadoop ou faire appel à des prestataires Extérieurs
- **Traitements de données différés et temps de latence important:** Hadoop n'est pas fait pour l'analyse temps réel des données.
- **Produit en développement continu.**

La lenteur des disques durs...

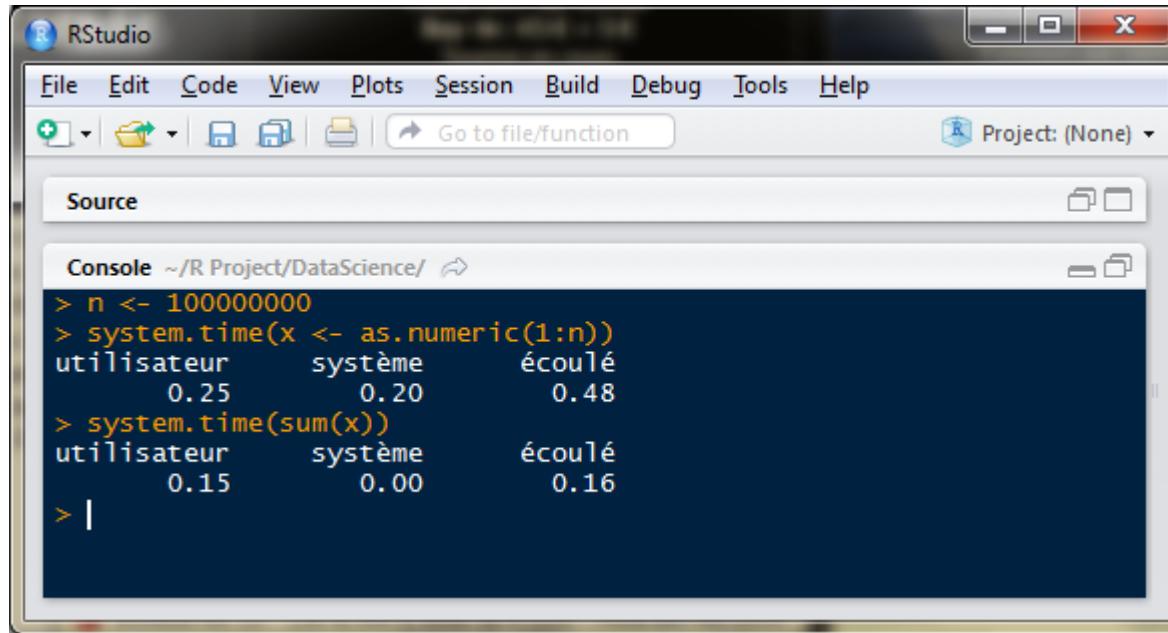
HDD : 0.05 GO par seconde (50 MO/sec)

RAM : 5 GO par seconde

Pour y pallier : deux grandes familles de solutions

1. La Parallélisation
2. Le travail in memory

Travailler **in memory** ... Un débit de **plusieurs GO par seconde**

A screenshot of the RStudio IDE. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Tools, and Help. Below the menu is a toolbar with icons for New Project, Open Project, Save, Print, and Go to file/function. A "Project: (None)" dropdown is shown. The main window has two tabs: "Source" and "Console". The "Console" tab shows the following R code and its execution times:

```
> n <- 100000000
> system.time(x <- as.numeric(1:n))
utilisateur      système     écoulé
       0.25        0.20       0.48
> system.time(sum(x))
utilisateur      système     écoulé
       0.15        0.00       0.16
> |
```

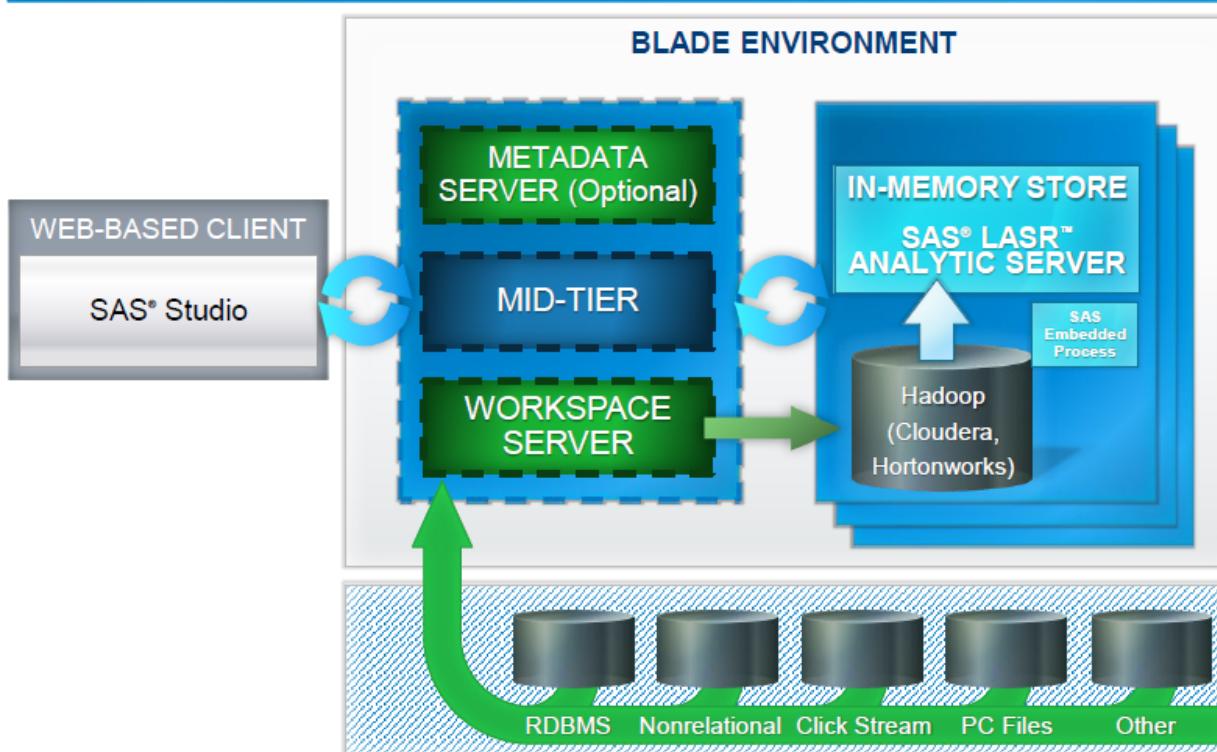


Exploitation de la rapidité du travail en mémoire : multiples variantes de in memory

HIGH LEVEL ARCHITECTURE

DISTRIBUTED DEPLOYMENT ON COMMODITY HARDWARE (ON SHARED HADOOP CLUSTER)

SAS® IN-MEMORY STATISTICS FOR HADOOP



SPARK : In Memory distribué

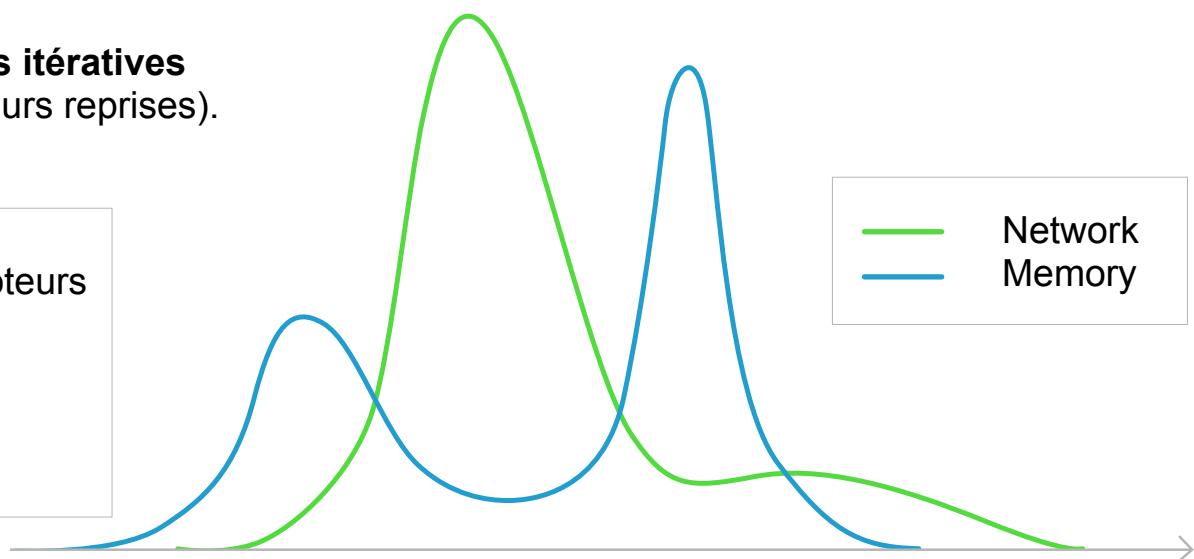
Spark s'interface avec Hadoop HDFS mais les données sont en mémoire

Il réduit le temps de latence de l'accès à la donnée : la 2^{ème} lecture instantanée (une fois que la donnée a été chargée).



Spark est très adapté aux **requêtes itératives** (même données sollicitées à plusieurs reprises).

utilisé par exemple pour les moteurs de recommandation des e-commerçants.



Organisation de l'écosystème et de ses évolutions : Les « Distributions »

