



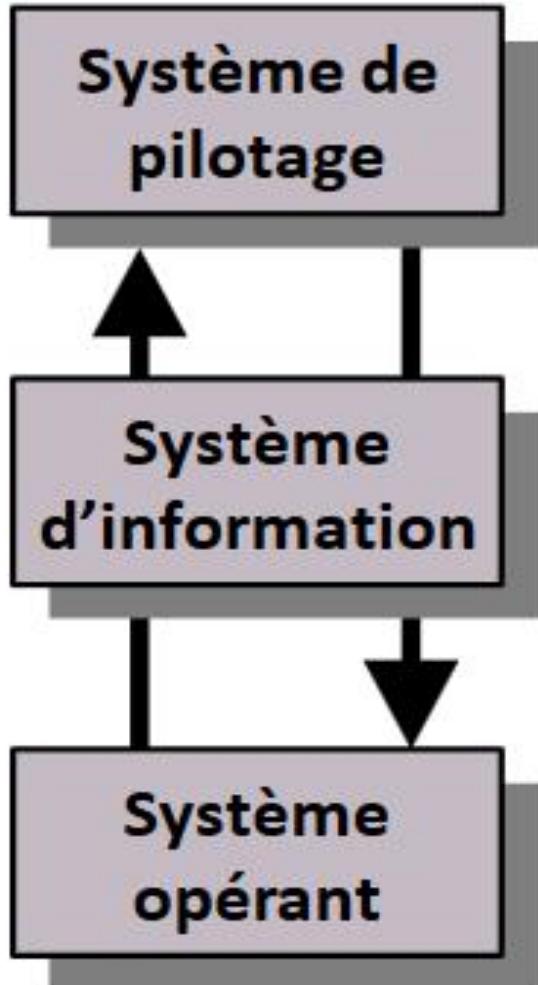
Big Data

AVANT-PROPOS

- Ce document est un support de cours qui se développe au fil des années afin de donner un support résumé et simplifié aux étudiants.
- Ce support de cours est développé selon un plan pédagogique qui a objectif de présenter un tutoriel simple et efficace. Afin de renforcer les connaissances, plusieurs exemples et exercices d'application sont présentés vers la fin de ce documents.



CHAPITRE I- INTRODUCTION BIG DATA



Activité :

- réfléchir : adaptation à l'environnement, conception
- décider : prévisions, allocation, planification
- contrôler : qualité

Activité :

- générer des informations
- mémoriser
- diffuser
- traiter

Activité :

- transformer
- produire



Valable ?

TRACES NUMÉRIQUE



TRACES NUMÉRIQUE

- La notion de Big Data est un concept s'étant popularisé en 2012 pour traduire le fait que les entreprises sont confrontées à des volumes de données à traiter de plus en plus considérables et présentant un fort enjeux commercial et marketing
- Ces Grosses Données en deviennent difficiles à travailler avec des outils classiques de gestion de base de données
- Il s'agit donc d'un ensemble de technologies, d'architecture, d'outils et de procédures permettant à une organisation de très rapidement collecter, traiter et analyser de larges quantités et contenus hétérogènes et changeants et d'en extraire les informations pertinentes à un coût accessible.

TRACES NUMÉRIQUE

- L'utilisation des données numériques : ***collection, stockage, l'analyse...***
- Objectif : L'amélioration, la prise de décision, la prévision, la gestion des risques
- Cette utilisation devient plus en plus compliqué avec la variété et le volume de données circulées

BIG DATA



LES DONNÉES NUMÉRIQUES

- L'explosion **quantitative** des données numériques a obligé les chercheurs à trouver de nouvelles manières de voir et d'analyser le monde des données.
- Il s'agit de découvrir de nouveaux ordres de grandeur pour capture, recherche, partage, stockage, analyse et présentation des données.
- le BD est partout, et utile pour l'entreprise : e-commerce, publicité, comportement de navigation sur Internet... Il s'agit d'un concept permettant de stocker un nombre **indicible** d'informations sur une base numérique.

LES DONNÉES NUMÉRIQUES

- Dans le domaine de l'usage, il répond à une nécessité de travailler la donnée plus profondément.
- Si les quantités de données semblent impressionnantes, encore faut-il savoir comment traiter et utiliser le big data pour les rendre utile à votre entreprise.

DÉFINITION DE BIG DATA

- Le Big Data (BD), littéralement « grosses données », ou méga données, parfois également appelés données massives. Il désigne un ensemble très volumineux de données qu'aucun outil classique de gestion de base de données ou de gestion de l'information ne peut vraiment travailler. Inventé par les géants du web, Le BD se présente comme une solution dessinée pour permettre à tout le monde d'accéder en temps réel à des bases de données géantes.

GÉNÉRALITÉS

Une minute sur internet en 2016 ?



GÉNÉRALITÉS

- Outils numériques plus performants et plus connectés:
ordinateurs et smartphones



- Open Data

- Réseaux sociaux: facebook
- Données d'administrations publiques

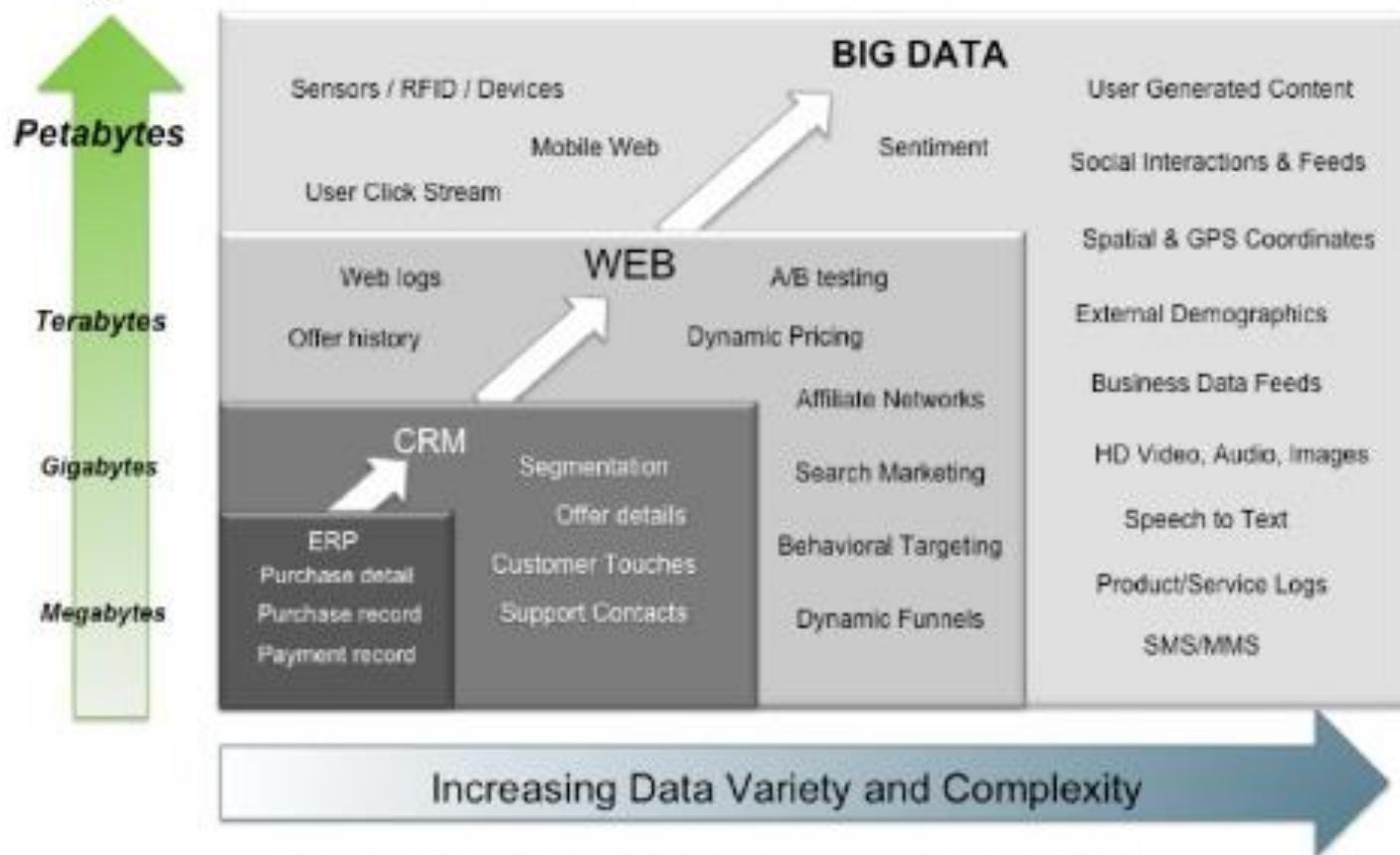


- Internet des Objets

- RFID (cartes de transport, codes bar, ...)
- Ericsson a estimé le nombre d'objets connectés dans le monde à 50 milliards en 2020 (12 milliards en 2013)

GÉNÉRALITÉS

Big Data = Transactions + Interactions + Observations



La croissance des données

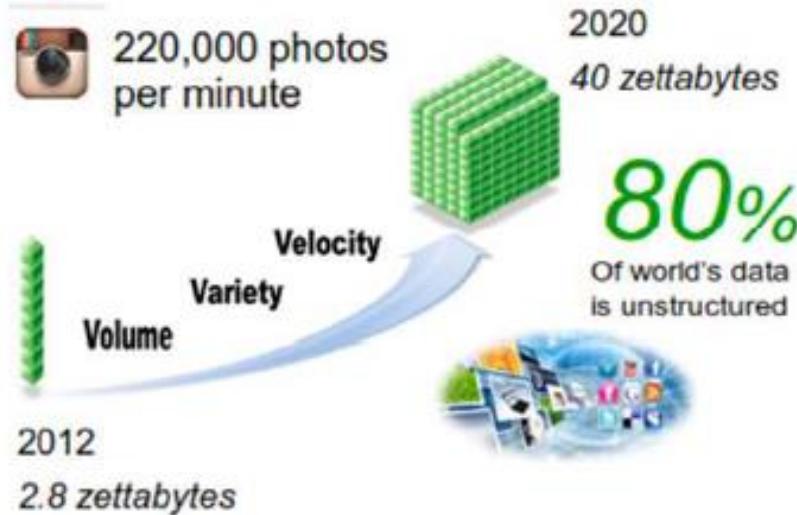
- Chaque jour, 2,5 trillions (2.5×10^{18}) d'octets de données sont générées.
- 90% des données ont été créées au cours des deux dernières années

 2.5 million items per minute

 300,000 tweets per minute

 200 million emails per minute

 220,000 photos per minute



Organizations need deeper insights

1 in 3 Business leaders frequently make decisions based on information they don't trust, or don't have

1 in 2 Business leaders say they don't have access to the information they need to do their jobs

83% of CIOs cited "Business intelligence and analytics" as part of their visionary plans to enhance competitiveness

60% of CEOs need to do a better job capturing and understanding information rapidly in order to make swift business decisions

Information is at the center of a new wave of opportunity

Système d'unités / Système binaire d'unités

International System of Units (SI)			Binary Usage (deprecated)
kilobyte	KB	10^3	2^{10}
megabyte	MB	10^6	2^{20}
gigabyte	GB	10^9	2^{30}
terabyte	TB	10^{12}	2^{40}
petabyte	PB	10^{15}	2^{50}
exabyte	EB	10^{18}	2^{60}
zettabyte	ZB	10^{21}	2^{70}
yottabyte	YB	10^{24}	2^{80}

International Electrotechnical Commission (IEC) - 1999		
kibibyte	KiB	2^{10}
mebibyte	MiB	2^{20}
gibibyte	GiB	2^{30}
tebibyte	TiB	2^{40}
pebibyte	PiB	2^{50}
exbibyte	EiB	2^{60}
zebibyte	ZiB	2^{70}
yobibyte	YiB	2^{80}

Source: Wikipedia, <http://en.wikipedia.org/wiki/Kibibyte>

Définition

- Lisa Arthur (**Forbes contributor**) définit la Big Data:
"a collection of data from traditional and digital sources inside and outside a company that represent a source of ongoing discovery and analysis."

- **Quelques exemples d'utilisation de la Big Data**

- | | |
|--|---|
| <ul style="list-style-type: none">• Science• Astronomy• Atmospheric science• Genomics• Biogeochemical• Biological• Social• Social networks• Social data<ul style="list-style-type: none">* Twitter* Facebook* LinkedIn | <ul style="list-style-type: none">• Commercial• Web / event / database logs• Sensor networks• Internet text and documents• Medical records• Photographic archives• Video / audio archives• Government• Military and homeland security surveillance |
|--|---|

Exemples de volumétrie (2017)

- 3,3 millions de publications sur Facebook sont créées chaque minute.
- 65 972 photos Instagram sont téléchargées chaque minute.
- 448 800 tweets sont créés chaque minute.
- 500 heures de vidéos YouTube sont téléchargées chaque minute.
- On analyse moins de 1% des données créées .
- Il est impossible de suivre le rythme de croissance des données et il n'est pas nécessaire de tout stocker.
- En 2020, on estime à 37% les données utiles susceptibles d'être analysées

Données au repos et Données en mouvement

- Informations rapides en temps réel, souvent transitoires sur les réseaux:
 - Informations provenant de capteurs
 - Informations provenant des journaux en temps réel et des moniteurs d'activité
 - Contenu en streaming comme l'audio et la vidéo
 - Transactions à grande vitesse telles que les tickers (*rapports d'évolutions de prix*), systèmes de trafic, ...et



Information streams

- Informations stockées en dehors des systèmes conventionnels.
Les données peuvent provenir du Web ou de différents systèmes internes.

- Collection de données qui étaient en mouvement
- Informations provenant des médias sociaux, journaux, courriels, ...
- Documents non structurés ou semi-structurées
- Données structurées de divers systèmes



Information oceans

LES CARACTÉRISTIQUES DU BIG DATA

- 2001 : 3Vs (Gartner)



- 2012 : 4Vs (IBM)



LES CARACTÉRISTIQUES DU BIG DATA

- 2015 : 5Vs

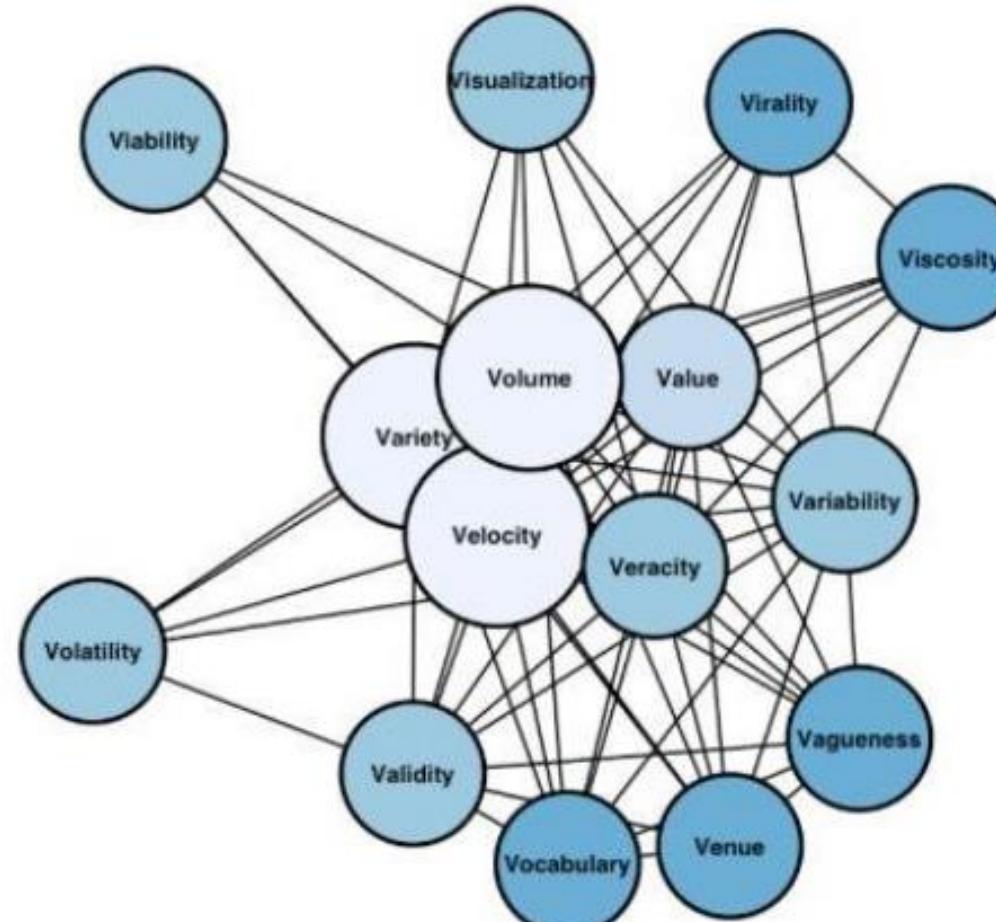


- 2016: : 7Vs



LES CARACTÉRISTIQUES DU BIG DATA

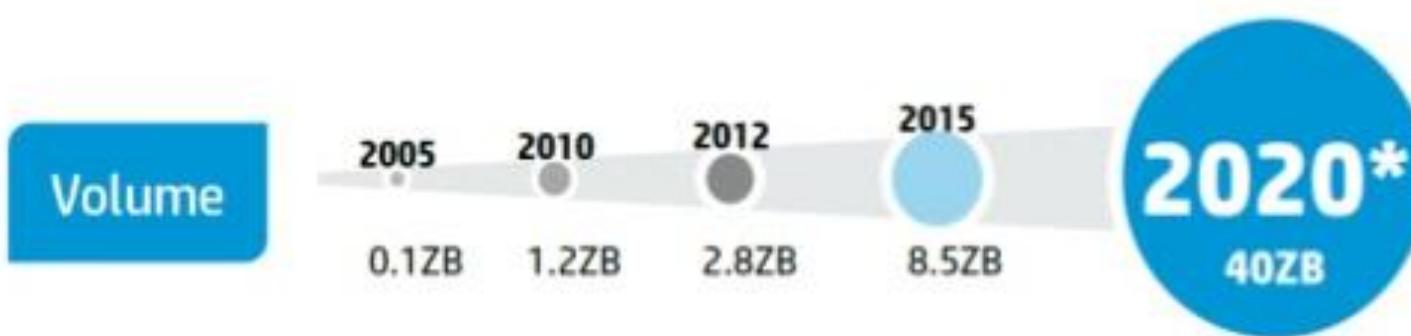
- 2017 : 10Vs ?



LES CARACTÉRISTIQUES DU BIG DATA

– Volume (1/2)

- Croissance sans cesse des données à gérer de tout type, souvent en teraoctets voir en petaoctets.
- Chaque jour, 2.5 trillions d'octets de données sont générées.
- **90% des données créées dans le monde l'ont été au cours des 2 dernières années (2014).**
- Prévision d'une croissance de 800% des quantités de données à traiter d'ici à 5 ans.



LES CARACTÉRISTIQUES DU BIG DATA

– Volume (2/2)

- 4,4 zettaoctets de données
= 4,4 trillion de gigaoctets
- En 2013, il y a autant de données que les étoiles connues dans tout l'univers.

- 44 zettaoctets de données
= 44 milliards gigaoctets
- 62 fois le nombre de tous les sables dans toutes les plages de la terre.

octets

1 Mégoctet = 10^6 octets

1 Gigaoctet = 10^9 octets

1 Téraoctet = 10^{12} octets

1 Pétaoctet = 10^{15} octets

1 Exaoctet = 10^{18} octets

1 Zettaoctet = 10^{21} octets

40

zettaoctets
de données
en 2020

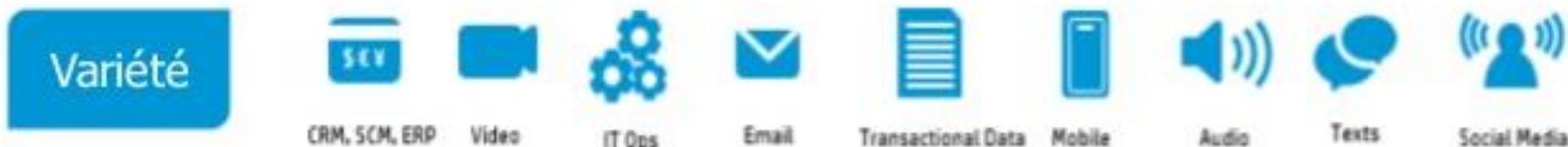
50

milliards
d'objets connectés
à la même date

LES CARACTÉRISTIQUES DU BIG DATA

– Variété

- Traitement des données sous forme structurée (bases de données structurée, feuilles de calcul venant de tableur, ...) et non structurée (textes, sons, images, vidéos, données de capteurs, fichiers journaux, medias sociaux, signaux,...) qui doivent faire l'objet d'une analyse collective.



- Diversité des données



LES CARACTÉRISTIQUES DU BIG DATA

– Vitesse (Velocity)

- Utilisation des données en temps réel (pour la détection de fraudes, analyse des données, ...).
- Fait référence à la vitesse à laquelle de nouvelles données sont générées et la vitesse à laquelle les données sont traitées par le système pour être bien analysées.
- La technologie nous permet maintenant d'analyser les données pendant qu'elles sont générées, sans jamais mettre en bases de données.
- Streaming Data
 - des centaines par seconde
- 100 Capteurs
 - dans chaque voiture moderne pour la surveillance



VELOCITY

LES CARACTÉRISTIQUES DU BIG DATA

– Vérité

- Fait référence à la qualité de la fiabilité et la confiance des données.
- Données bruités, imprécises, prédictives, ...
- La génération des données par Spambots est un exemple digne de confiance.
 - L'élection présidentielle de 2012 au Mexique avec de faux comptes Twitter.
- DES MILLIONS DE DOLLARS \$ PAR AN
 - Ce que la pauvre qualité des données coutent pour l'économie des Etats-Unis.
- 1 à 3 CHEFS D'ENTREPRISE
 - Ne font pas confiance à l'information qu'ils utilisent.

Veracity



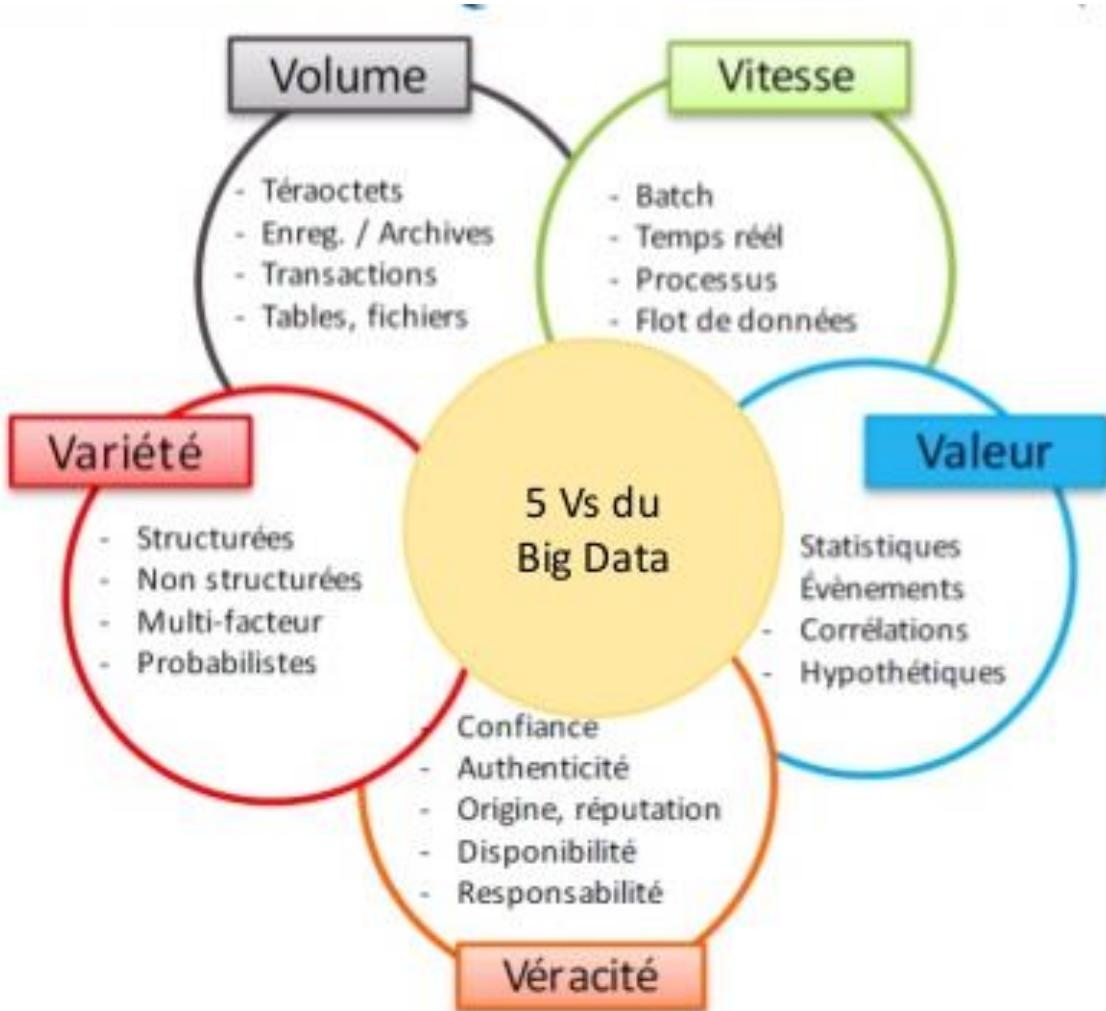
LES CARACTÉRISTIQUES DU BIG DATA

– Valeur

- La démarche Big Data n'a de sens que pour atteindre des objectifs stratégiques de **création de valeur** pour les clients et pour l'entreprise; dans tous les domaines d'activité : commerce, industrie, services ...
- Le succès d'un projet Big Data n'a d'intérêt aux utilisateurs que s'il **apporte de la valeur ajoutée** et de nouvelles connaissances.

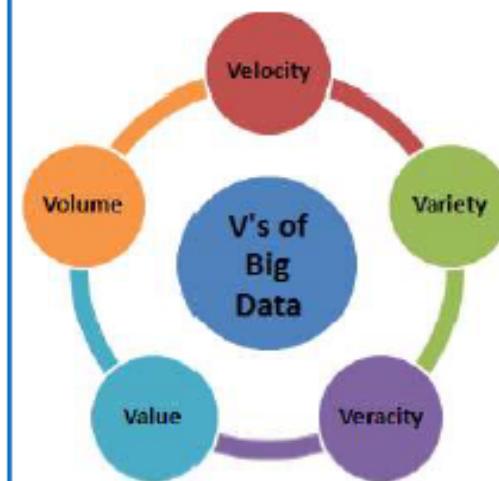


LES CARACTÉRISTIQUES DU BIG DATA



Caractéristiques du Big Data

- Il n'y a pas de définition unique du Big Data, mais certains éléments sont communs à toutes les définitions: la vitesse, le volume, la variété.
- Extraire, de manière rapide et rentable, des connaissances à partir de données volumineuses , variées et d'une vitesse élevée.



Volume: varie de terabytes à zettabytes. On doit traiter efficacement ce volume croissant.

Variété: Gérer divers types et structures de données

Vélocité: Analyser les flux de données en continu et les grands volumes de données persistantes.

Véracité: authenticité et fiabilité des données nécessitant une *grande rigueur* dans la collecte, le recouplement, le croisement et l'enrichissement des données pour lever l'incertitude pour garantir l'intégrité des données.

Valeur: le point le plus important, une solution Big Data doit apporter une valeur ajoutée pour l'entreprise en répondant à des objectifs commerciaux ou Marketing qui orientent l'utilisation des Big Data.

LES AVANTAGES DU BIG DATA

- Aide intelligente pour la prise de décision,
- Réductions des coûts
- Réductions de temps
- Optimisation des processus de l'entreprise et développement de nouveaux produits ou services
-

Big Data dans un système décisionnel

Traditional Approach

Structured & Repeatable Analysis

Business Users

Determine what question to ask



IT

Structures the data to answer that question

Monthly sales reports
Profitability analysis
Customer surveys

Big Data Approach

Iterative & Exploratory Analysis

IT

Delivers a platform to enable creative discovery



Business

Explores what questions could be asked

Brand sentiment
Product strategy
Maximum asset utilization

Secteurs d'utilisation des Big Data



Analyse du sentiment et de l'expérience client multicanal, ...



Détection à temps des causes potentiellement mortelles dans les hôpitaux pour intervenir, ...



Prédire les conditions météorologiques pour planifier l'utilisation optimale des éoliennes et optimiser les dépenses , ...



Prendre des décisions de risque basées sur des données transactionnelles en temps réel, ...



Identifier les criminels et les menaces provenant de sources vidéo, audio et de données disparates

Projets Big Data

- **Déforestation** : Projet Planetary SKIN (Par la NASA, CISCO, ...)
- **Suivi astronomique en direct** : Projet LSST (Large Synoptic Survey Telescope)
(20 Tb chaque nuit)
- **Traitement du Cancer**: projet ICGC (International Cancer Genome Consortium),
analyse de plusieurs BD sur des tumeurs de 50 types de cancers
- **Détection d'épidémies en temps réel** : Projet Healthmap puis
ResistanceOpen pour la collecte, le stockage et l'analyse de divers
types de données pour le suivi et la surveillance d'épidémie au
niveau mondial.

Approche traditionnelle Vs Approche Big Data

Approche traditionnelles

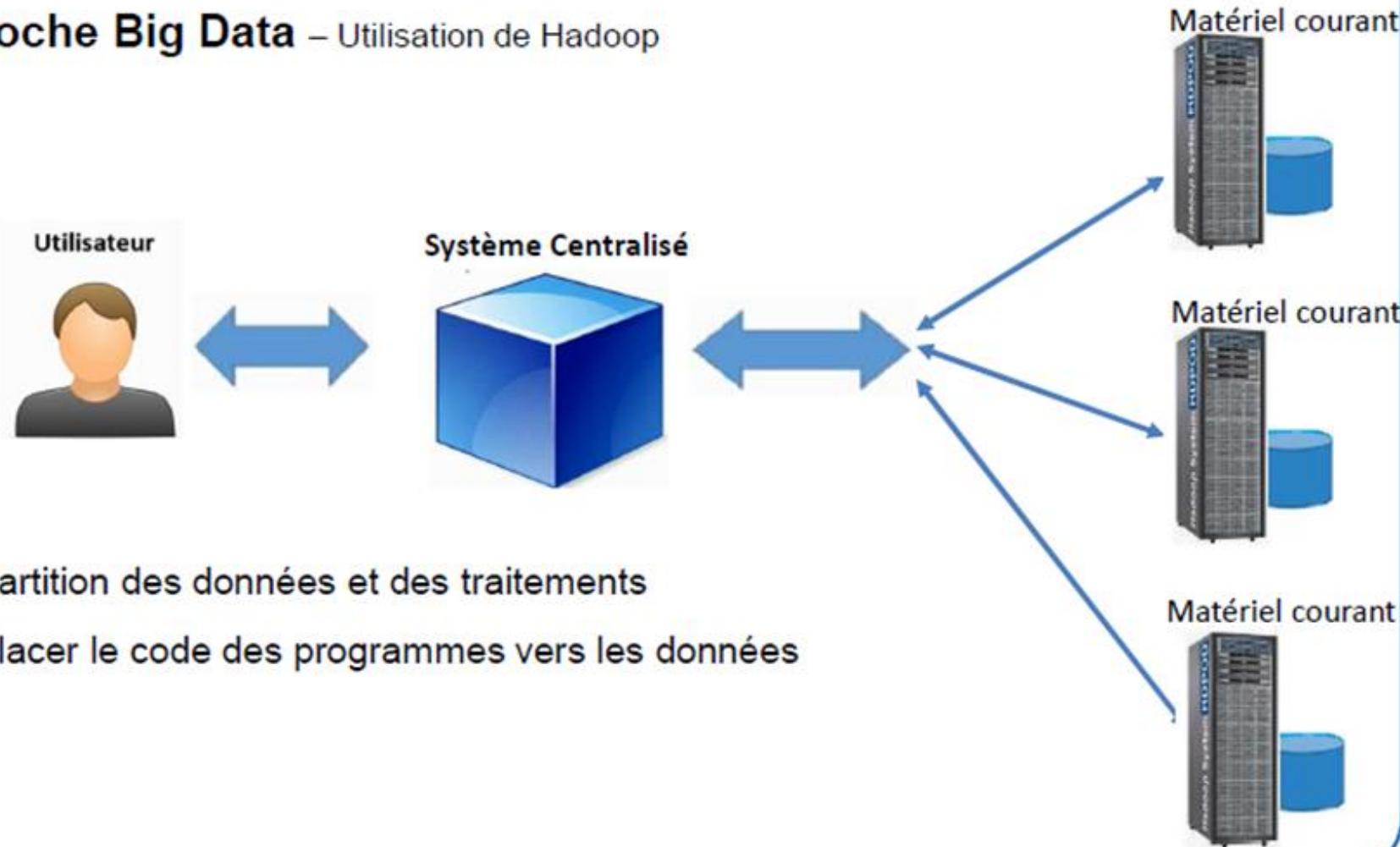


Insuffisance lorsqu'il y a:

- Grande croissance de données
- Des données sans schémas

Approche traditionnelle Vs Approche Big Data

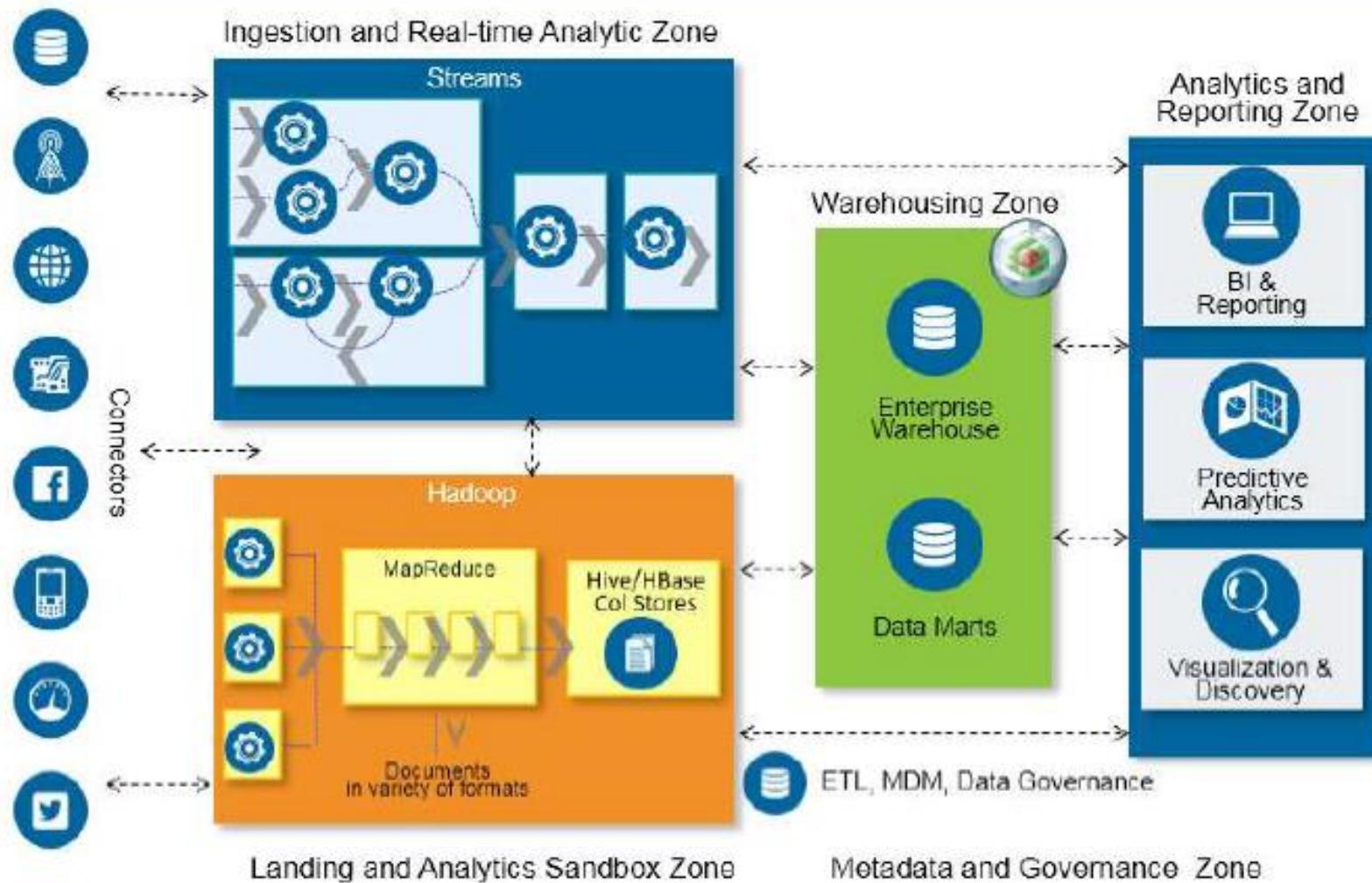
Approche Big Data – Utilisation de Hadoop





CHAPITRE 2- L'ÉCOSYSTÈME BG

Intégration Hadoop / S.I et Décisionnel



Introduction

❑ La vitesse de lecture et d'écriture (*Latence du disque*) ne s'est pas beaucoup amélioré au cours des dernières années (*autour de 70 - 80MB / sec – disque récents* jusqu'à 320 MB/s (selon l'interface ATA/IDE ou SCSI – SSD peut atteindre 1000M/s)

❑ **Problème:** Combien de temps faut-il pour lire 1 To de données?

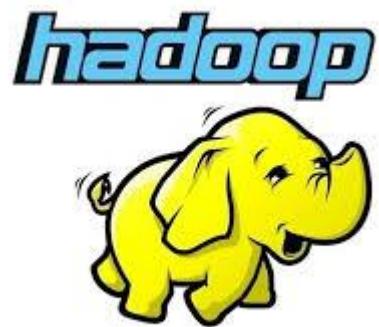
1 To (à 80 Mo / sec):	1 disque - 3,4 heures
	10 disques - 20 min
	100 disques - 2 min
	1000 disques - 12 sec

❑ **Solution:** Le traitement parallèle des données

- **Calcul distribué:** plusieurs ordinateurs apparaissent comme un super ordinateur, communiquent les uns avec les autres par le passage de messages, opèrent ensemble pour atteindre un but commun.

❑ **Défis:** Hétérogénéité (matériel-os) - Ouverture - Évolutivité - Sécurité - Concurrence - Tolérance aux pannes - Transparence

HADOOP



- Hadoop est un framework logiciel open source permettant de stocker des données, et de lancer des applications sur des grappes de machines standards. Cette solution offre un espace de stockage massif pour tous les types de données, une immense puissance de traitement et la possibilité de prendre en charge une quantité de tâches virtuellement illimitée. Basé sur Java, ce framework fait partie du projet Apache, sponsorisé par Apache Software Foundation.

POURQUOI

Pourquoi Hadoop est important ?

- Les avantages apportés aux entreprises par Hadoop sont nombreux. Grâce à ce framework, il est possible de stocker et de traiter de vastes quantités de données rapidement. Face à l'augmentation en hausse du volume de données et à leur diversification, principalement liée aux réseaux sociaux et à l'internet des objets, il s'agit d'un avantage non négligeable.
- le modèle de calcul distribué d'Hadoop permet de traiter rapidement le Big Data. **Plus le nombre de nœuds de calcul utilisés est important, plus la puissance de traitement est élevée.** Les données et les applications traitées sont protégées contre les échecs hardware. Si un nœud tombe en panne, les tâches sont directement redirigées vers d'autres nœuds pour s'assurer que le calcul distribué n'échoue pas. De multiples copies de toutes les données sont stockées automatiquement.

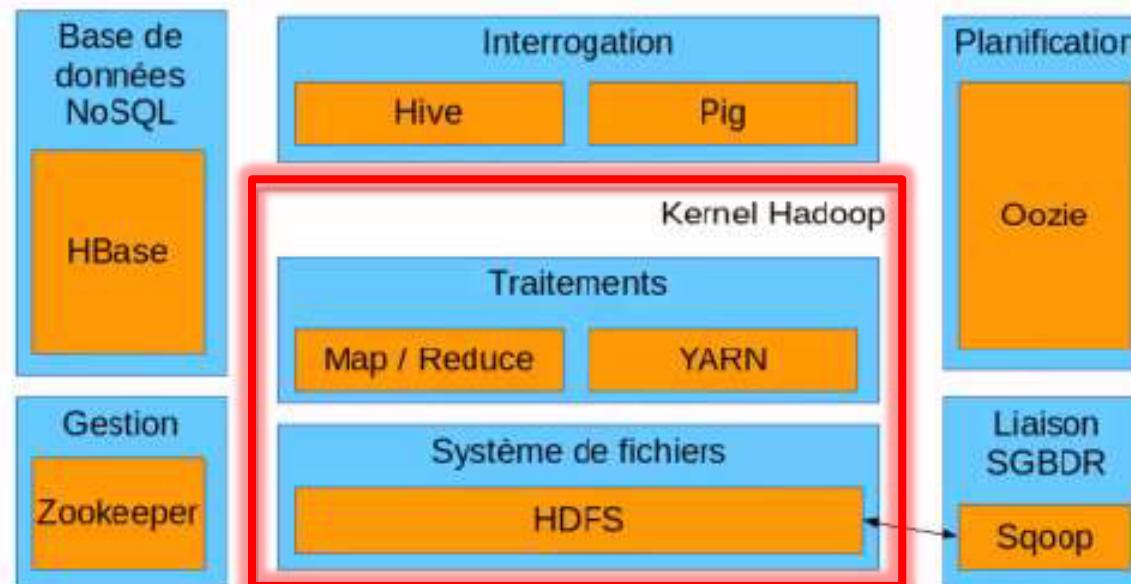
POURQUOI

Pourquoi Hadoop est important ?

- Le framework open source est donc **gratuit et repose sur des machines standards pour stocker de larges quantités de données**. Enfin, il est possible d'adapter le système pour prendre en charge plus de données en ajoutant simplement des nœuds. L'administration requise est minimale.

L'écosystème Hadoop

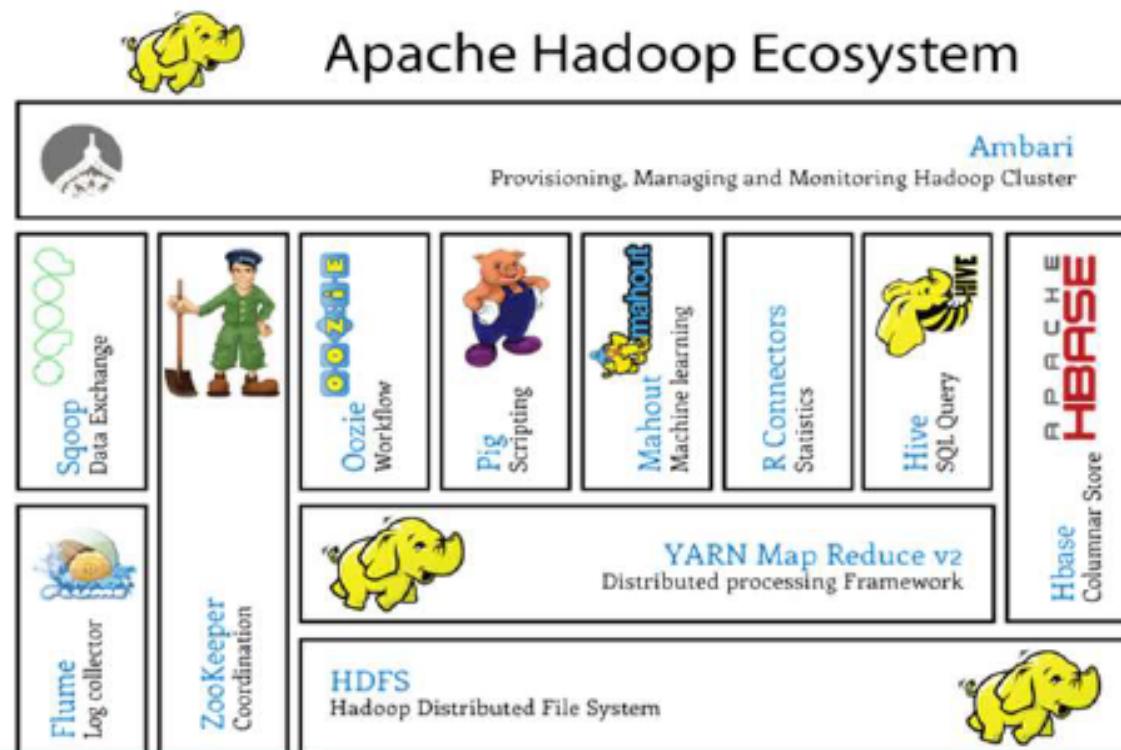
- Hadoop est basé sur:
 - **MapReduce/YARN**: framework de traitement distribué
 - **HDFS** : pour un stockage distribué
- Hadoop est écrit en **Java** et sous License Apache
- Plusieurs projets sont liés à Hadoop ⇒ Ecosystème Hadoop



Simon Gilliot : Introduction à la plate-forme Hadoop et à son écosystème

L'écosystème Hadoop

- Composants de base: **HDFS – MapReduce /Yarn**
- D'autres outils existant pour:
 - La simplification des opérations de traitement sur les données
 - La gestion et coordination de la plateforme
 - Le monitoring du cluster



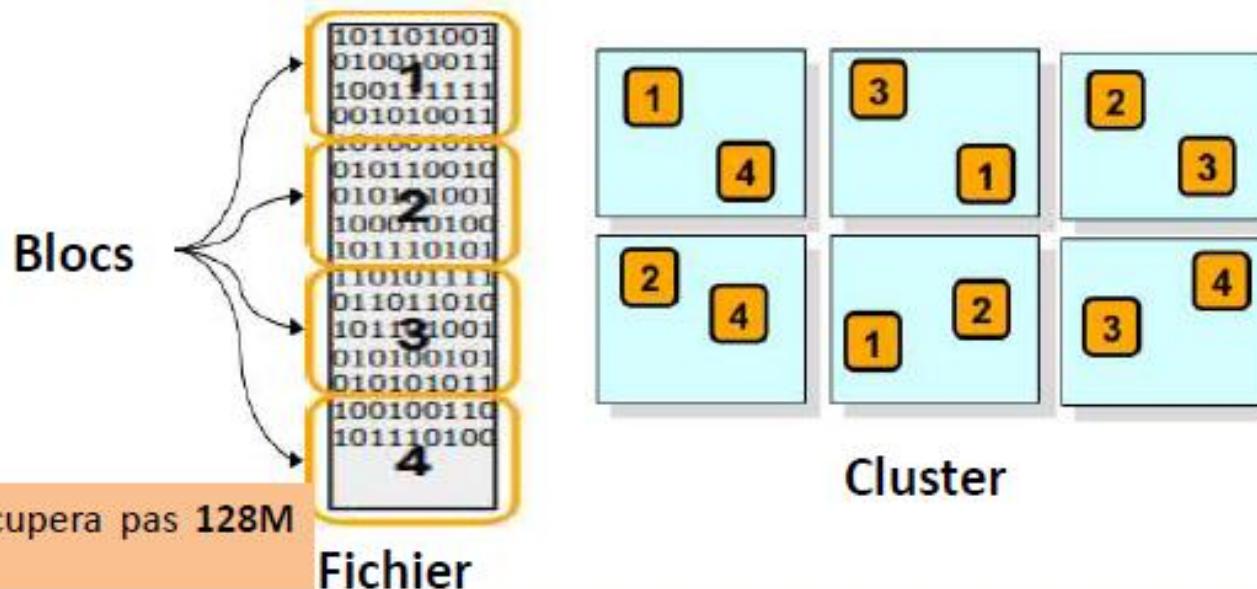
Hadoop: HDFS

HDFS : Système de fichiers conçu pour stocker des fichiers très volumineux dans un cluster d'ordinateurs pas chers.

- HDFS stocke les données sur plusieurs nœuds (**cluster**)
- Il existe des clusters **Hadoop** qui stockent des Po de données.
- Ecrire une fois/lire plusieurs fois (**WORM**: *write once read many times*): Les données sont généralement générées ou copiées à partir de la source. Des analyses (^{batch}) sont ensuite effectuées sur cet ensemble de données au fil du temps. Chaque analyse impliquera une grande proportion, sinon la totalité, de l'ensemble de données.
- HDFS réplique les données sur plusieurs nœuds afin d'éviter leur perte en cas de panne de certains nœuds ⇒ Fiabilité, Disponibilité
- Chaque fichier est décomposé en plusieurs blocs qui seront répliqués sur les nœuds.

Hadoop: HDFS

- **Bloc:** taille par défaut 128Mo sur Hadoop 2.x (64Mo sur Hadoop 1.x) (bloc OS = 4k Octets par exemple)
- La taille est grande pour minimiser le temps de recherche (localisation) des blocs.
- Chaque bloc est répliqué sur un petit nombre (*facteur de réPLICATION*) de machines physiquement séparées. (*par défaut trois*)



Hadoop: Architecture de HDFS

- Architecture maître/esclave

- **NameNode**

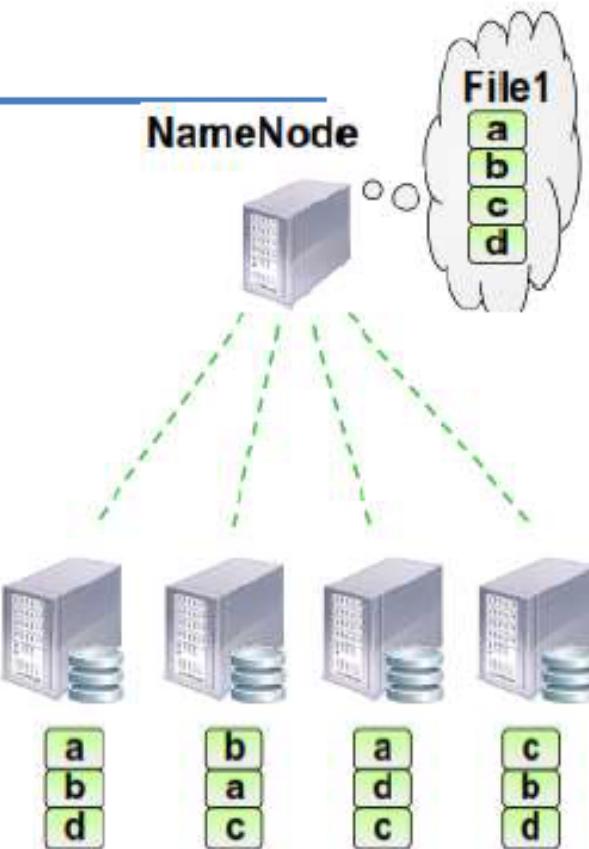
- Gère l'espace de noms du système de fichiers: Il gère l'arborescence du système de fichiers et les métadonnées de tous les fichiers (*noms, facteurs de réPLICATION, propriétés*) et répertoires de l'arborescence.

Ces informations sont stockées sur le disque local dans deux fichiers:

- **FsImage** (métadonnées) chargé en mémoire au démarrage du **NameNode**.

- **EditLog** : pour enregistrer chaque modification apportée aux métadonnées du système de fichiers. Par exemple, création d'un nouveau fichier dans HDFS, le NameNode insère un enregistrement dans le **EditLog** indiquant cela. De même, la modification du facteur de réPLICATION d'un fichier entraîne l'insertion d'un nouvel enregistrement dans **EditLog**

- Régule l'accès aux fichiers par les clients (équilibrer la charge)



DataNodes

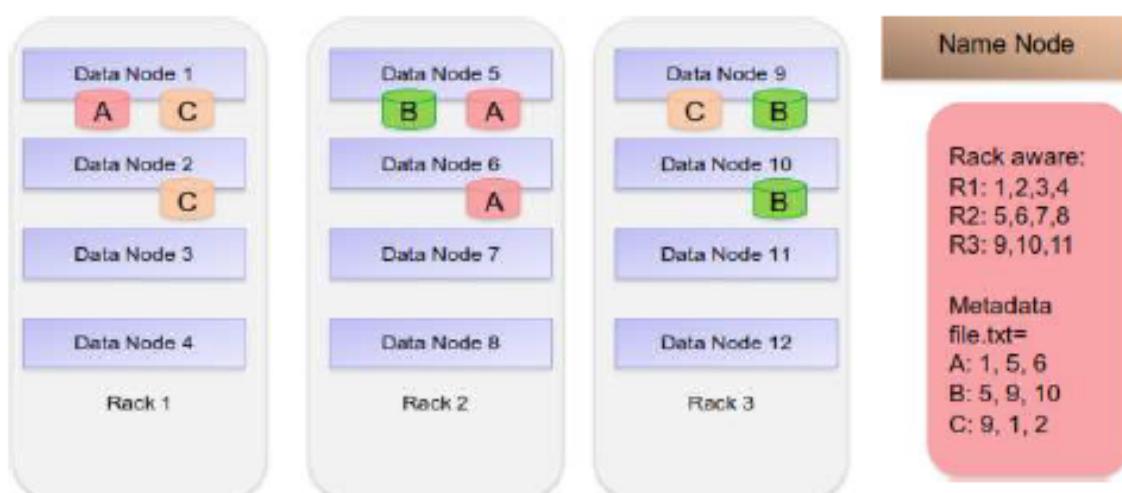
Réf: IBM BigInsights Foundation v4.0

(Course code DW613 ERC 1.0)

Hadoop: Architecture de HDFS

- Le **NameNode** conserve les **métadonnées en mémoire** pour assurer un accès rapide ce qui nécessite une grande taille mémoire pour son fonctionnement.
- Les blocs de fichiers sont répliqués sur plusieurs nœuds (**DataNodes**).
- Le nombre de copies de chaque bloc est contrôlé par le paramètre de réPLICATION (Par défaut 3)
- RéPLICATION sur plusieurs racks:

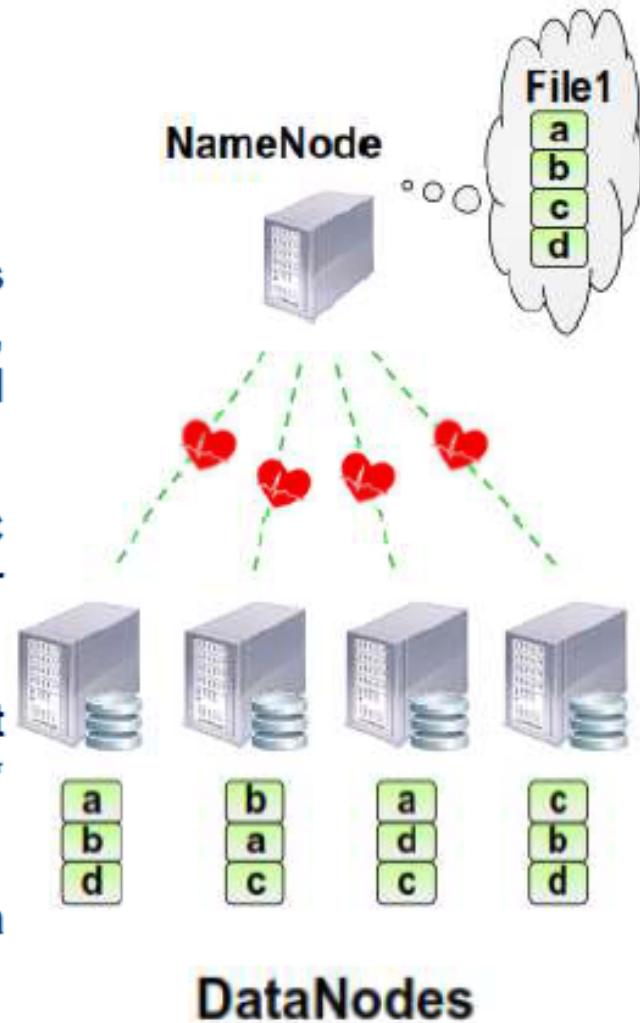
Une copie sur un premier rack - La 2ème et la 3ème copies peuvent être ensemble sur un autre Rack.



Hadoop: Architecture de HDFS

- **DataNode**

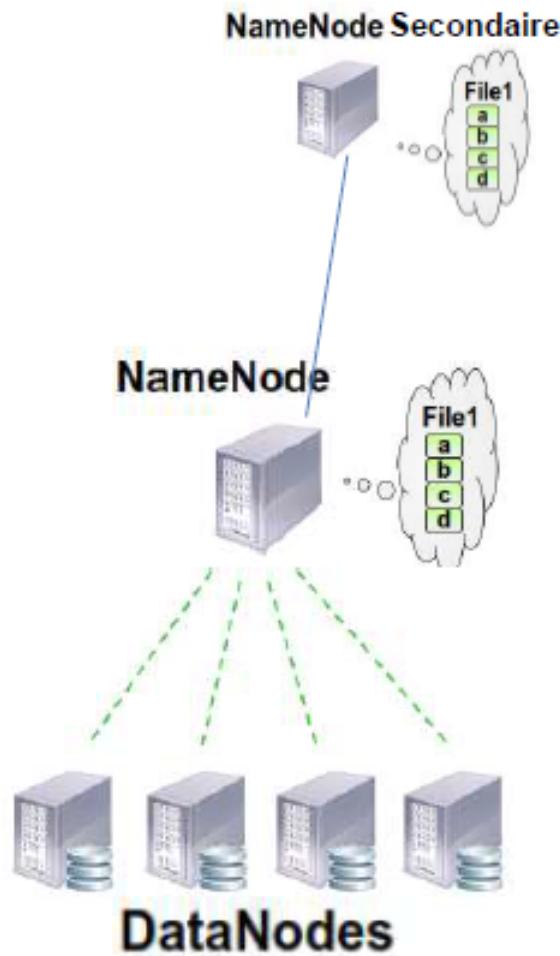
- Nombreux DataNodes par cluster (*un par nœud*)
- Gère le stockage (read/write) attaché aux nœuds
- Envoie périodiquement un signal au NameNode sous forme de **heartbeat** (id, espace stockage, espace utilisé, ...) avec un intervalle indiqué dans `hdfs-site.xml` (`dfs.heartbeat.interval`, par défaut 3s).
- Renvoie périodiquement (`dfs.blockreport.intervalMsec` par défaut 21600000 Ms) au NameNode un rapport sur les blocs qu'il stocke (id, taille, date de création, ...)
- Si un **DataNode** tombe en panne, ses blocs seront répliqués sur d'autres **DataNodes** (`timeout = 2 * heartbeat.recheck.interval + 10 * heartbeat.interval`)
- Effectue également la création, la suppression et la réPLICATION de blocs sur ordre du NameNode.
- Les données sont stockées sur plusieurs nœuds



Hadoop: Architecture de HDFS

• Secondary NameNode

- Le NameNode est un point de défaillance unique (*Single Point of Failure - SPOF*): Si la machine exécutant le NameNode est perdue, toutes les informations (meta-data) sur les fichiers de données seraient perdues. Par conséquent, on ne pourra pas reconstruire les fichiers à partir des blocs sur les DataNodes.
- Hadoop sauvegarde les métadonnées du système de fichiers (**FsImage**): NameNode écrit son état persistant sur plusieurs systèmes de fichiers (disque local, disque distant).
- Utiliser un NameNode secondaire: n'a pas le même rôle que le NameNode. Il fusionne périodiquement l'image de l'espaces des noms (**FsImage**) et le journal d'édition (**EditLog**) pour éviter que ce dernier devienne trop volumineux. Il s'exécute sur une machine physique distincte (*nécessite beaucoup de CPU et de RAM pour effectuer la fusion*)



Hadoop: Architecture de HDFS

- **Secondary NameNode (suite)**
 - La **réPLICATION** des métadonnées du **NameNode** et l'utilisation du **NameNode secondaire** permet d'éviter la perte de métadonnées et de conserver des copies récentes.
 - Mais elle n'offre pas une **haute disponibilité** du système de fichiers:
 - En cas d'échec du **NameNode**, tous les clients, y compris les jobs MapReduce, ne pourraient pas lire, écrire ou répertorier les fichiers, car le **NameNode** est le seul référentiel des métadonnées et du mapping fichier-bloc.
 - Dans un tel cas, l'ensemble du système Hadoop serait **hors service** jusqu'à ce qu'un nouveau **NameNode** puisse être mis en ligne.
 - Pour récupérer un **NameNode défaillant**, l'administrateur démarre un **nouveau NameNode** avec l'une des copies de métadonnées du système de fichiers et configure les **DataNodes** et les applications clientes (du Data Center) pour utiliser ce nouveau **NameNode**.

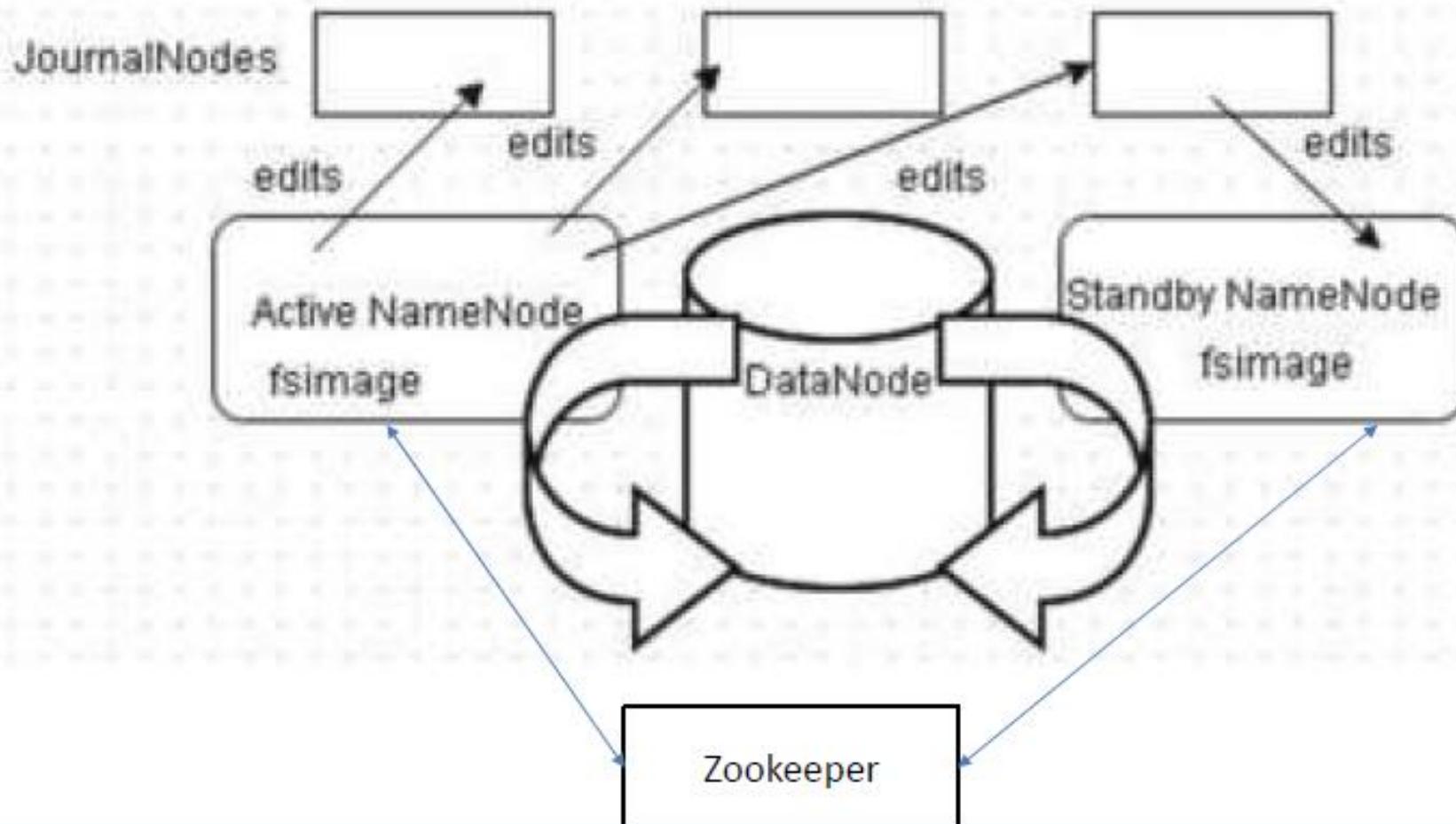
Hadoop: Architecture de HDFS

- **Secondary NameNode (suite)**

- Le nouveau **NameNode** ne peut répondre aux demandes des clients tant qu'il n'a pas (1) chargé son **FsImage dans la mémoire**, (2) relu son journal d'édition (EditLog) , et (3) reçu suffisamment de rapports de blocs des **DataNodes** pour quitter le mode sans échec (Safe Mode).
- Sur les grands clusters avec un grand nombre de fichiers et de blocs, le temps nécessaire pour qu'un **NameNode** démarre à froid peut être de 30 minutes ou plus.
- Le long temps de récupération est également un problème pour l'entretien de routine.
- Hadoop 2 a remédié à cette situation en ajoutant la prise en charge de la haute disponibilité (**HA**) HDFS: il y a un pair de NameNodes Passif/Actif. En cas de panne du Namenode actif, le passif devient actif pour continuer à répondre aux demandes des clients sans interruption significative.

Hadoop: Architecture de HDFS

Haute Disponibilité dans HDFS (high-availability – HA)



Hadoop: Architecture de HDFS

Haute Disponibilité dans HDFS (high-availability – HA) (Hadoop 2)

1. Le NameNode actif et le NameNode passif communiquent avec un groupe (quorum) de JournalNodes (généralement 3)
2. Le NameNode actif envoie les "edits" (logs) aux JournalNodes pour les partager avec le NameNode passif.
3. Le NameNode passif récupère les "edits" depuis l'un des JournalNodes et les applique à la copie Fsimage qu'il a afin de la synchroniser avec celle du NameNode actif.
4. Les DataNodes envoient périodiquement les rapports sur les blocs aux NameNodes actif et passif.
5. ZooKeeper est un démon de coordination utilisé pour coordonner le basculement rapide du NameNode actif vers le NameNode passif. ZooKeeper a deux fonctions dans un cluster HDFS haute disponibilité:
 - Déetecter une défaillance du NameNode actif. Les deux NameNodes conservent une session active dans ZooKeeper. Lorsqu'il y a défaillance du NameNode actif, ZooKeeper détecte cette défaillance suite à l'interruption de la session du NameNode actif.
 - ZooKeeper initie un basculement vers NameNode passif et fournit un mécanisme pour la l'élection du NameNode actif lorsqu'il y a plusieurs candidats (NameNodes passifs)

Hadoop: Modes de fonctionnement

Hadoop possède trois modes d'exécution :

- **Mode local (standalone)** : Hadoop fonctionne sur une seule machine et tout s'exécute dans la même JVM (*Java Virtual Machine*). En mode local le système de gestion de fichiers utilisé est celui du système hôte (Exemple Linux: ext3, ext4 ou xfs) et non HDFS.
- **Mode pseudo-distribué**: Hadoop fonctionne sur une seule machine, mais chacun des deamons principaux s'exécute dans sa propre JVM. Le système de fichier utilisé est HDFS dans ce mode.
- **Mode totalement distribué**: c'est le mode d'exécution réel d'Hadoop. Il permet de faire fonctionner le système de fichiers distribué et les deamons sur un ensemble de machines différentes.

Hadoop: Commandes HDFS

- Nous utiliserons Hadoop avec un seul nœud pour tester et utiliser quelques commandes HDFS.
- Les clients HDFS utilisent la propriété **fs.default.name** ou **fs.defaultFS** de Hadoop pour déterminer l'URL de l'hôte et le port du **NameNode** HDFS.
- Nous l'utiliserons sur localhost (**fs.defaultFS = hdfs://localhost: 8020**) avec le port HDFS par défaut, **8020**.
- Les propriétés et paramètres Hadoop sont dans des fichiers de configuration:

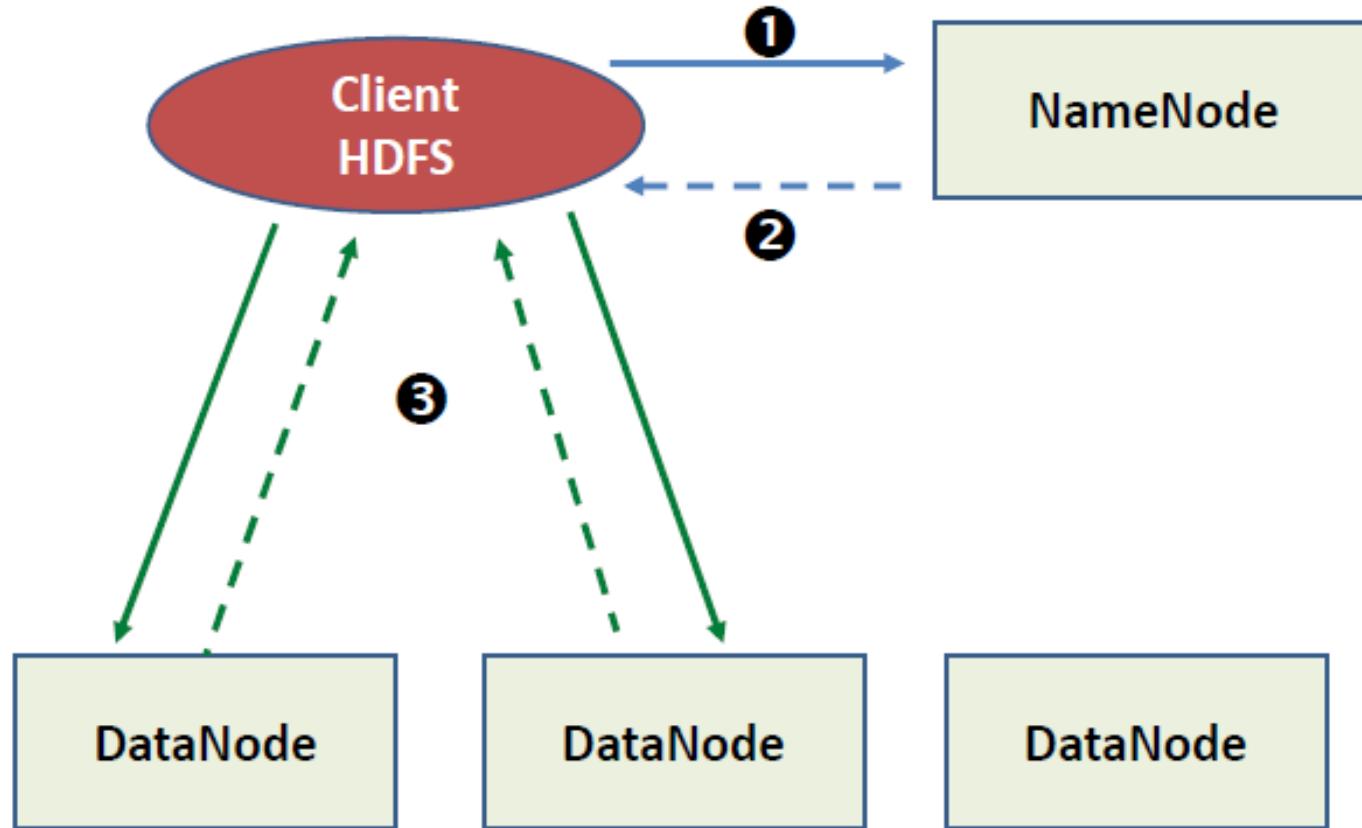
`/etc/hadoop/conf/core-site.xml`

`/etc/hadoop/conf/hdfs-site.xml`: facteur de réPLICATION (**dfs.replication**),
NameNode secondaire
(**dfs.namenode.secondary.http-address**),

...

.....

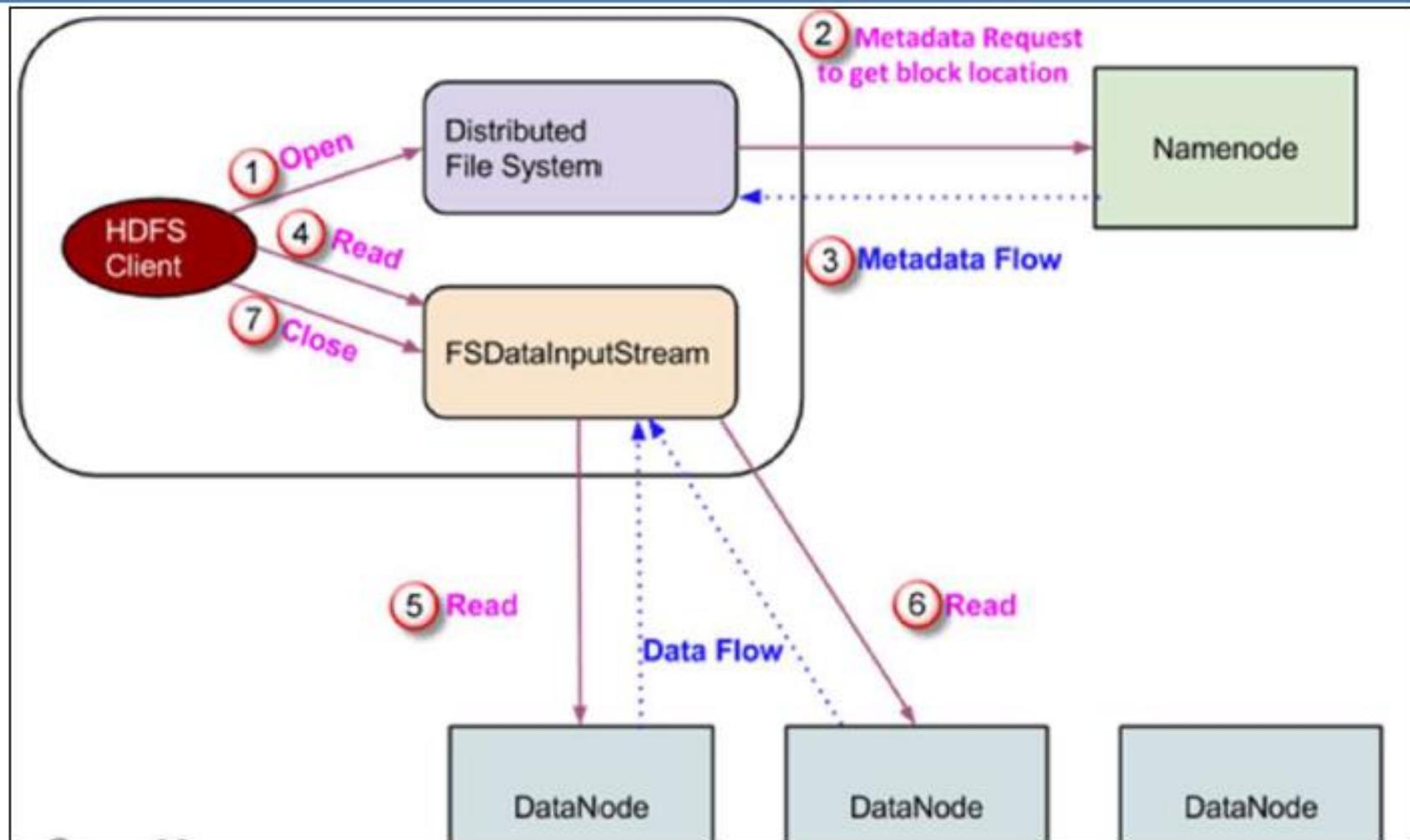
Hadoop: Lecture d'un fichier HDFS (1/4)



Hadoop: Lecture d'un fichier HDFS (2/4)

1. Le client (application HDFS) communique avec le NameNode pour demander la lecture d'un fichier (ou plusieurs, un répertoire par exemple)
2. Le NameNode envoie des métadonnées au client avec le détail du nombre de blocs, des identifiants et tailles des blocs, des emplacements des blocs et du nombre de réPLICATION.
3. Le client communique avec les DataNodes où les blocs sont présents.
4. Le client commence à lire les blocs en parallèle à partir des DataNodes en se basant sur les métadonnées reçues du NameNode.
5. Pour améliorer les performances de lecture, l'emplacement de chaque bloc est choisi en fonction de sa distance par rapport au client: d'abord lecture des blocs dans le même DataNode, puis un autre DataNode dans le même rack, et enfin un autre DataNode dans un autre rack.
6. Une fois que le client reçoit tous les blocs, il les combine pour former un fichier.

Hadoop: Lecture d'un fichier HDFS (3/4)



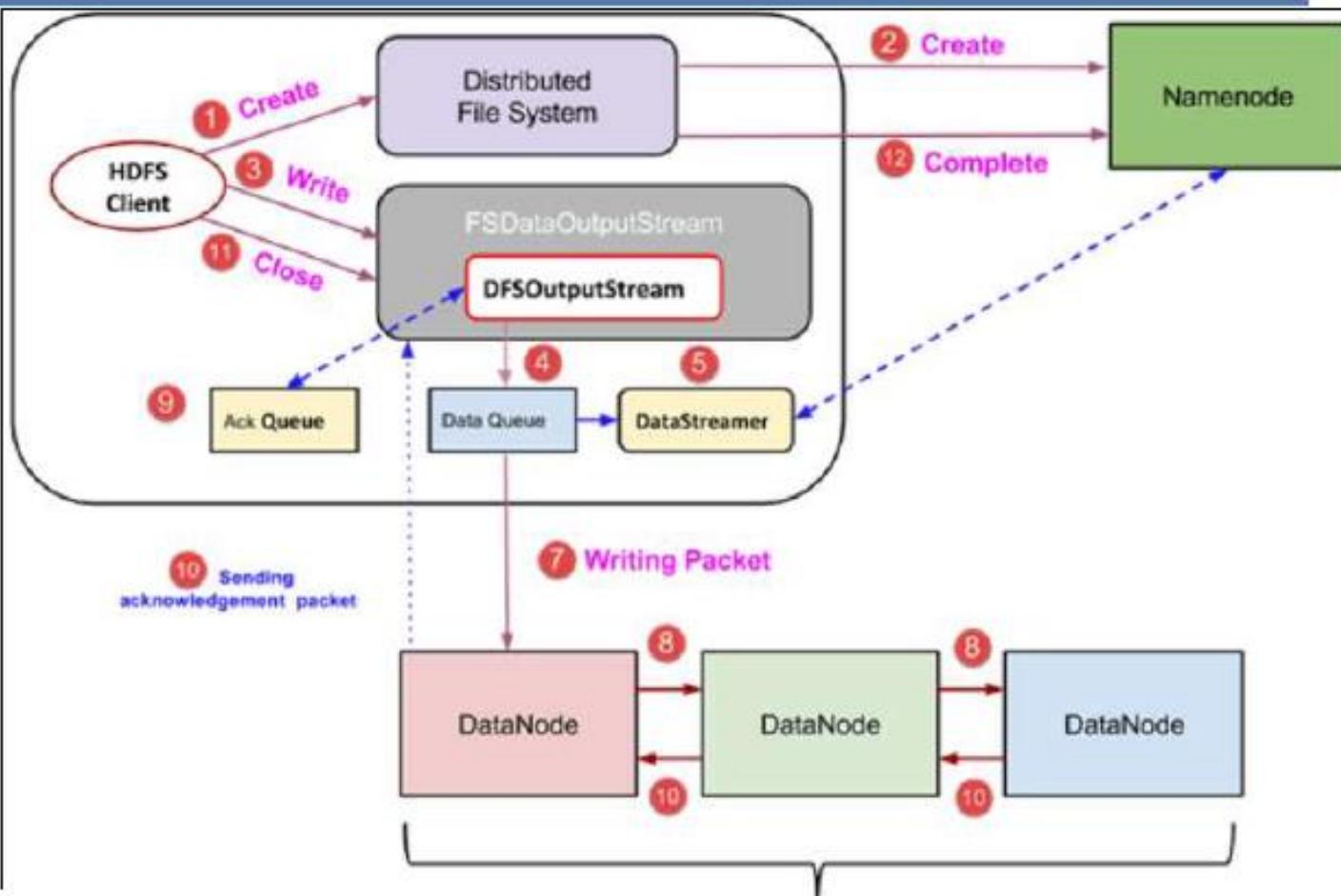
Référence: <https://www.guru99.com/learn-hdfs-a-beginners-guide.html>

Hadoop: Lecture d'un fichier HDFS (4/4)

- ❶ Le client appelle la méthode **open()** de l'objet **FileSystem** pour lire un fichier HDFS.
- ❷ Cet objet se connecte au **NameNode** à l'aide d'un appel **RPC** pour avoir les métadonnées du fichier à lire telles que les emplacements des blocs du fichier.
- ❸ Le NameNode envoie les métadonnées au client avec le détail du nombre de blocs, des identifiants et les tailles de blocs, des emplacements des blocs et du nombre de réPLICATION.
- ❹ Le client appelle la méthode **read()**, ce qui permet à un objet **DFSInputStream** d'établir une connexion avec un **DataNode** contenant un bloc du fichier
- ❺❻ Le client lit ce bloc sous forme de flux via des appels répétitifs de **read()**. A la fin d'un bloc, **DFSInputStream** ferme la connexion avec le **DataNode**.
- ❼ La méthode **close()** est appelée à la fin de la lecture de tous les blocs.

Une fois que le client reçoit tous les blocs, il les combine pour former un fichier.

Hadoop: Ecriture d'un fichier dans HDFS (1/4)



Référence:

<https://www.guru99.com/learn-hdfs-a-beginners-guide.html>

Hadoop: Ecriture d'un fichier dans HDFS (2/4)

1. Le client appelle la méthode **create()** de l'objet **DistributedFileSystem** qui permet de créer un nouveau fichier pour écriture.
2. Cet objet se connecte au **NameNode** à l'aide de **RPC** et lui demande la création d'un nouveau fichier. A ce stade on n'associe aucun bloc à ce fichier. Le **NameNode** vérifie s'il y a déjà un autre fichier de même nom et si le client a les droits nécessaires pour créer le fichier. Dans ce cas, une entrée (dans le **FsImage**) est créée pour le nouveau fichier. Dans le cas contraire, une exception **IOException** est soulevée.
3. Un objet de la classe **FSDataOutputStream** est retourné au client afin de l'utiliser pour écrire le contenu du fichier dans **HDFS** en invoquant la méthode **write()** (en boucle)
4. L'objet **FSDataOutputStream** utilise un autre objet **DFSOutputStream** qui gère la communication avec les **DataNodes** et le **NameNode**. **DFSOutputStream** crée à fur et à mesure des paquets (64k) avec les données écrites par le client. Ces paquets sont mis dans une file d'attente appelée **DataQueue**. **DFSOutputStream** décompose ainsi les fichiers à écrire en paquets.

Hadoop: Ecriture d'un fichier dans HDFS (3/4)

5. Le composant **DataStreamer** consomme les paquets de **DataQueue** et demande au **NameNode** l'allocation de nouveaux blocs. Le **NameNode** envoie les métadonnées au client avec le détail du nombre de blocs, des identifiants de blocs, des emplacements des blocs et du nombre de réPLICATION. Ainsi les **DataNodes** à utiliser pour la réPLICATION de chaque bloc seront identifiés.
6. Le processus de réPLICATION d'un bloc commence par la création d'un pipeline à l'aide de **DataNodes** identifiés dans l'étape 5. Dans le schéma, le facteur de réPLICATION est 3, il y a donc 3 **DataNodes** dans le pipeline.
7. Le **DataStreamer** envoie des paquets vers le premier **DataNode** du pipeline
8. Chaque **DataNode** d'un pipeline stocke le paquet reçu et le transmet au **DataNode** suivant du pipeline.
9. **DFSOOutputStream** gère une autre file d'attente, "Ack Queue", pour stocker les paquets qui attendent un accusé de réCEPTION de **DataNodes**.

Hadoop: Ecriture d'un fichier dans HDFS (4/4)

10. Lorsque l'accusé de réception d'un paquet dans la file d'attente est reçu de tous les nœuds de données du pipeline, il est supprimé de la file «Ack Queue». En cas d'échec de DataNode, les paquets de cette file d'attente sont utilisés pour relancer l'opération.
11. Le client termine l'écriture des données par l'appel de la méthode `close()`, ce qui entraîne l'envoie des paquets de données restants vers le pipeline, puis l'attente de l'accusé de réception.
12. Une fois l'accusé de réception final reçu, le `NameNode` est contacté pour lui indiquer que l'opération d'écriture de fichier est terminée.

Travaux Pratiques



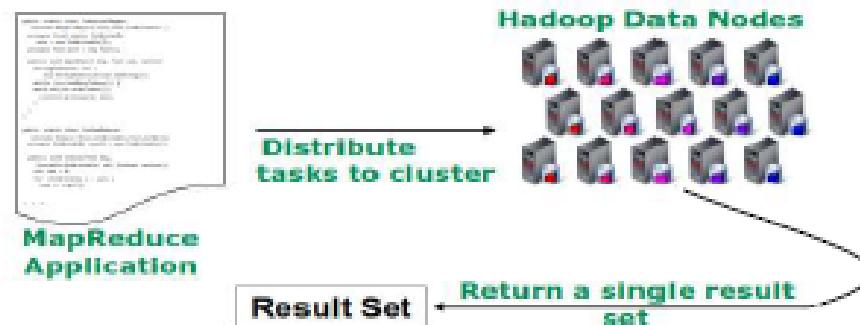
MapReduce/YARN



Hadoop: MapReduce

• Principe:

- Les données sont distribuées sur les nœuds du cluster sous forme de blocs.
- Ces données ne seront pas déplacées, via le réseau, vers un programme devant les traiter.
- Pour de meilleures performances, chaque bloc de données est traité **localement** (*principe de data locality*), minimisant les besoins d'échanges réseaux entre les machines.
- Un programme doit donc être exécuté sur les nœuds contenant les données à traiter, où il va exploiter leurs ressources de calcul et de mémoire (CPU + RAM)
- Le programme doit être écrit selon un modèle bien déterminé: **MapReduce**.
- Le système de fichiers distribués (DFS) est au cœur de MapReduce. Il est responsable de la distributions des données à travers le cluster, en faisant en sorte que l'ensemble du cluster ressemble à un système de fichiers géant.



Hadoop: MapReduce

- **Intérêt:**

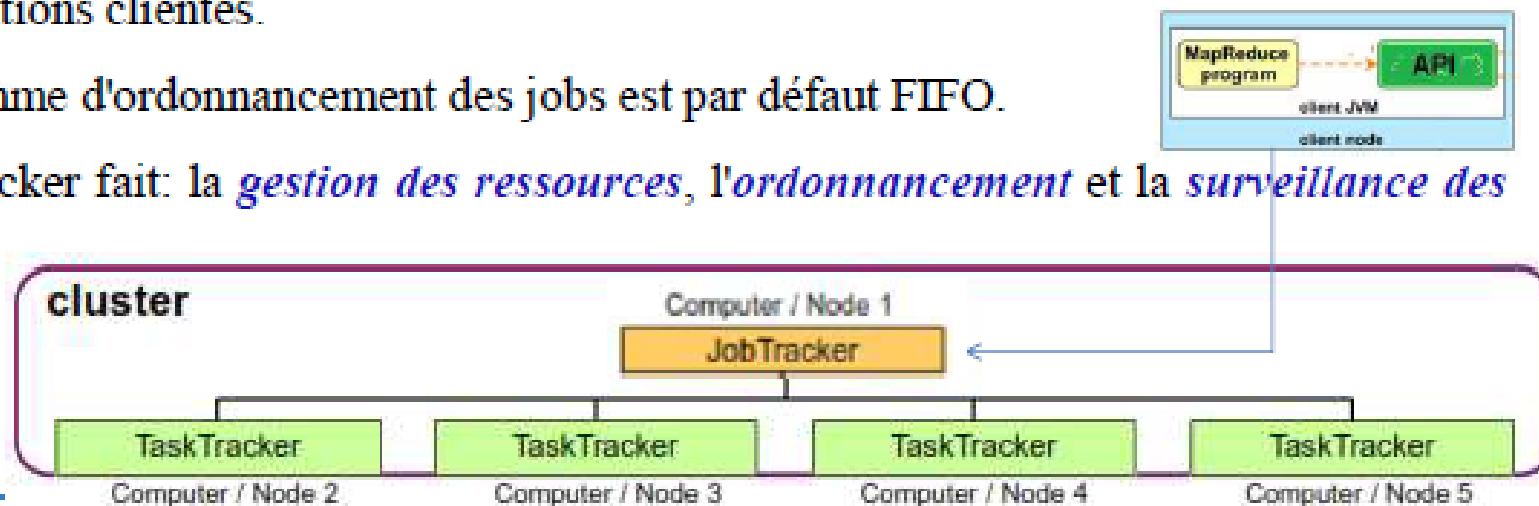
- Simplifier le développement de programme devant traiter des données distribuées.
- Le développeur n'a pas à se soucier du travail de **parallélisation et de distribution** du traitement. MapReduce permet au développeur de ne s'intéresser qu'à la partie algorithmique.
- Un programme MapReduce contient deux fonctions principales **Map ()** et **Reduce ()** contenant les traitements à appliquer aux données.

- **Architecture MR1 (dans Hadoop 1)**

- **Maître/Eclave:** L'unique maître (**JobTracker**) contrôle l'exécution des deux fonctions sur plusieurs esclaves (**TaskTrackers**).

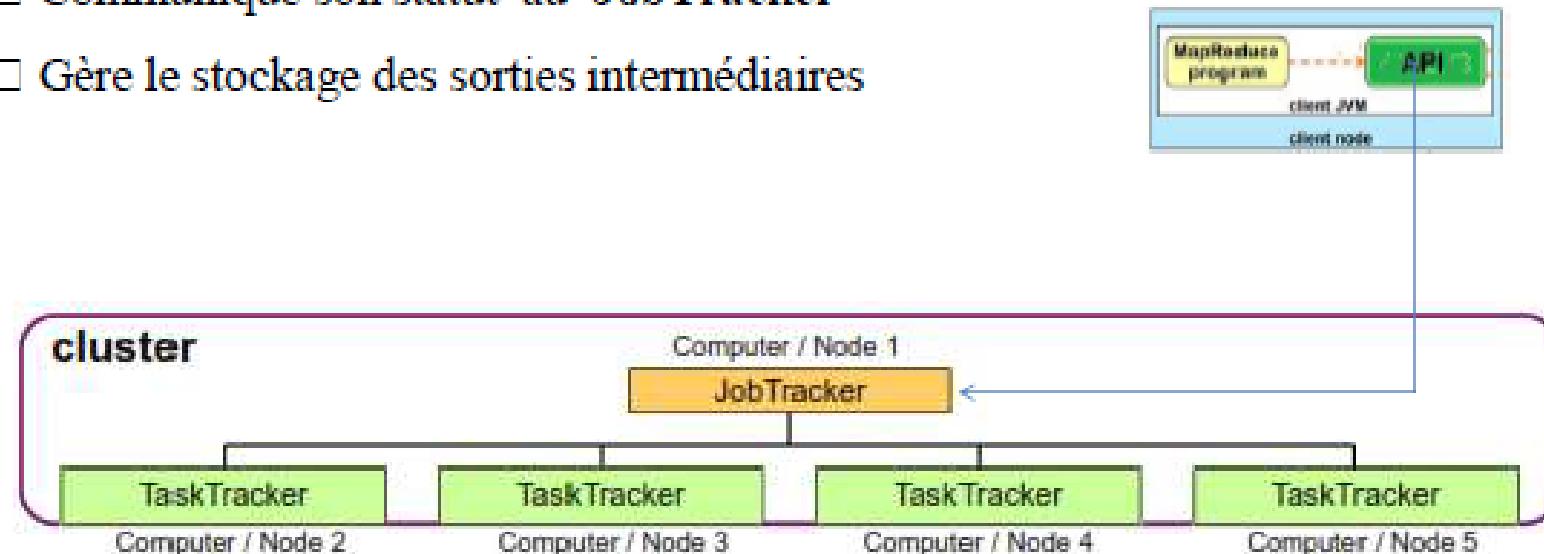
MapReduce: Job Tracker

- Reçoit les **jobs MapReduce** envoyés par les clients (**Applications**)
 - Communique avec le **NameNode** pour avoir les emplacements des données à traiter
 - Passe les tâches **Map** et **Reduce** aux nœuds **TaskTrackers**
 - Surveille les tâches et le statut des **TaskTrackers**
 - Relance une tâche si elle échoue.
 - Surveille l'état d'avancement des jobs et fournit des informations à ce sujet aux applications clientes.
 - Algorithme d'ordonnancement des jobs est par défaut FIFO.
- ➔ JobTracker fait: la *gestion des ressources*, l'*ordonnancement* et la *surveillance des tâches*



MapReduce: TaskTracker

- Exécute les tâches Map et Reduce
- Chaque TaskTracker possède un nombre de "slots" pour exécuter les tâches Map et Reduce ([mapreduce.tasktracker.map.tasks.maximum](#) et [mapreduce.tasktracker.reduce.tasks.maximum](#) dans [mapred-site.xml](#)).
- Communique son statut au JobTracker
- Gère le stockage des sorties intermédiaires



MapReduce: Modèle de programmation

- Etape "Map"

- Les fichiers d'entrée sont divisés en pièces **logiques** appelées "splits"
- Un **split** est une division **logique** des données d'entrée d'une tâche **Map**, tandis qu'un **bloc HDFS** est une division **physique** des données.
- La taille d'un split peut être définie par l'utilisateur dans son programme.
- Les nœuds esclaves (**TaskTracker**) traitent les splits individuellement en parallèle (sous le contrôle global du **JobTracker**)
- Chaque nœud esclave stocke son résultat dans son **système de fichiers local** où un Reducer (thread exécutant la fonction **Reduce**) est capable d'y accéder

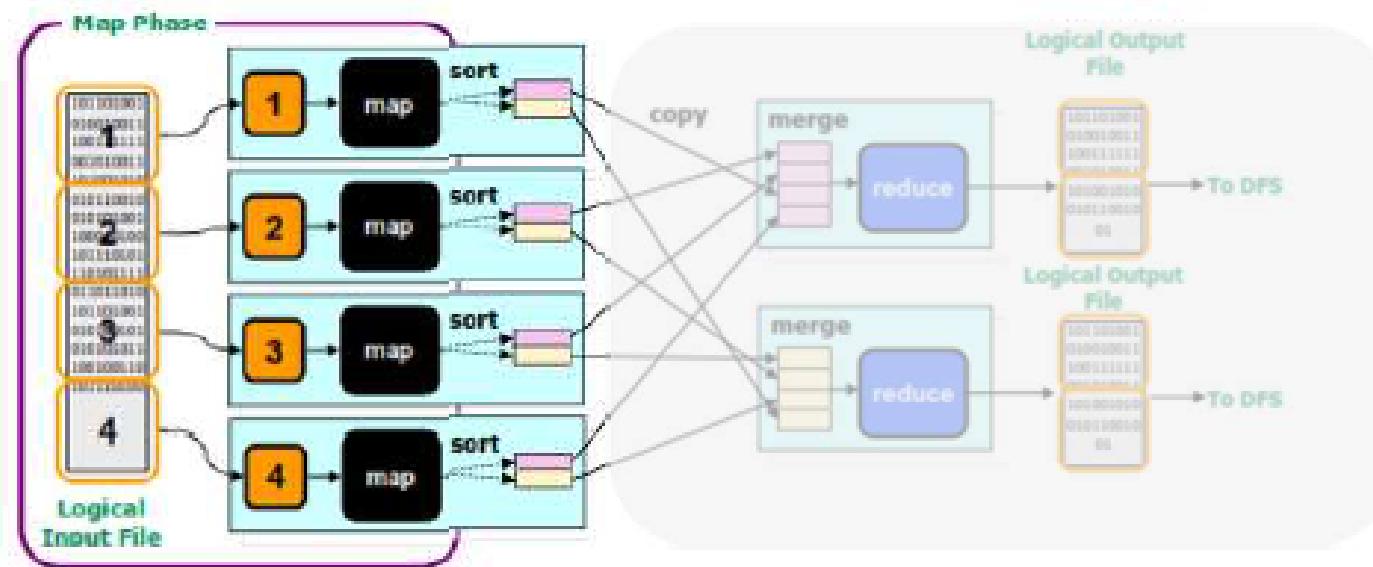
- L'étape "Reduce"

- Les données (*résultats des opérations Map*) sont agrégées par des nœuds esclaves (sous le contrôle du **JobTracker**)
- Plusieurs tâches **Reduce** parallélisent l'agrégation.
- Les sorties sont stockées dans **HDFS** (et donc répliquées)

MapReduce: La phase de Map

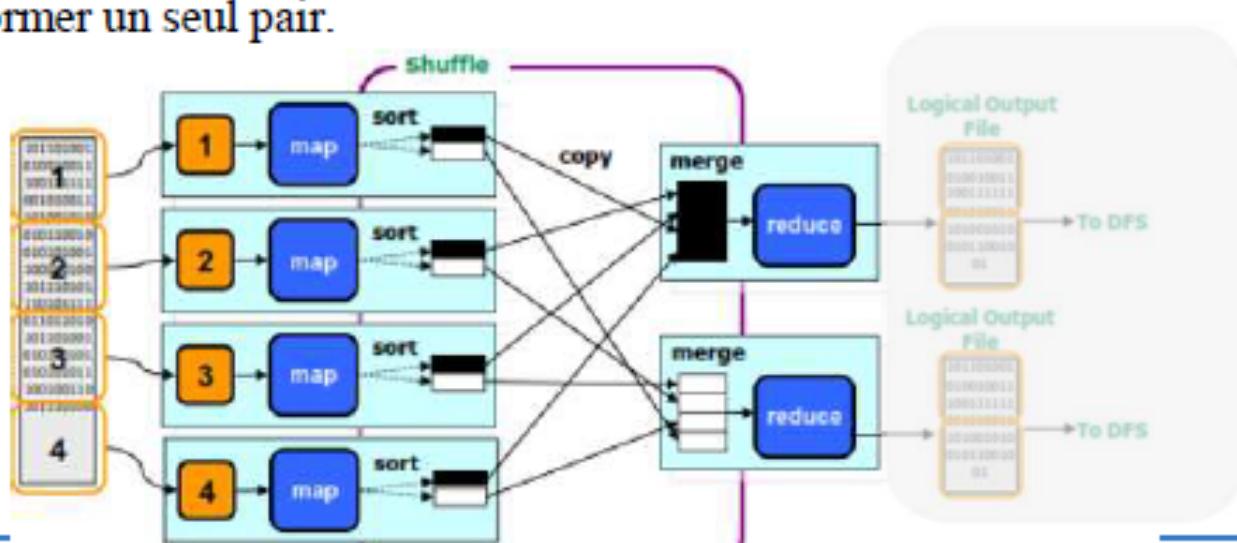
Mapper

- Petit programme (*généralement*), distribué dans le cluster et local aux données
- Traite une partie des données en entrée (*appelée split*)
- Chaque Map analyse, filtre ou transforme un split qui est un ensemble de pairs **<key, value>**.
- Produit des pairs **<clé, valeur>** groupés



MapReduce: La phase de Shuffle (réorganisation)

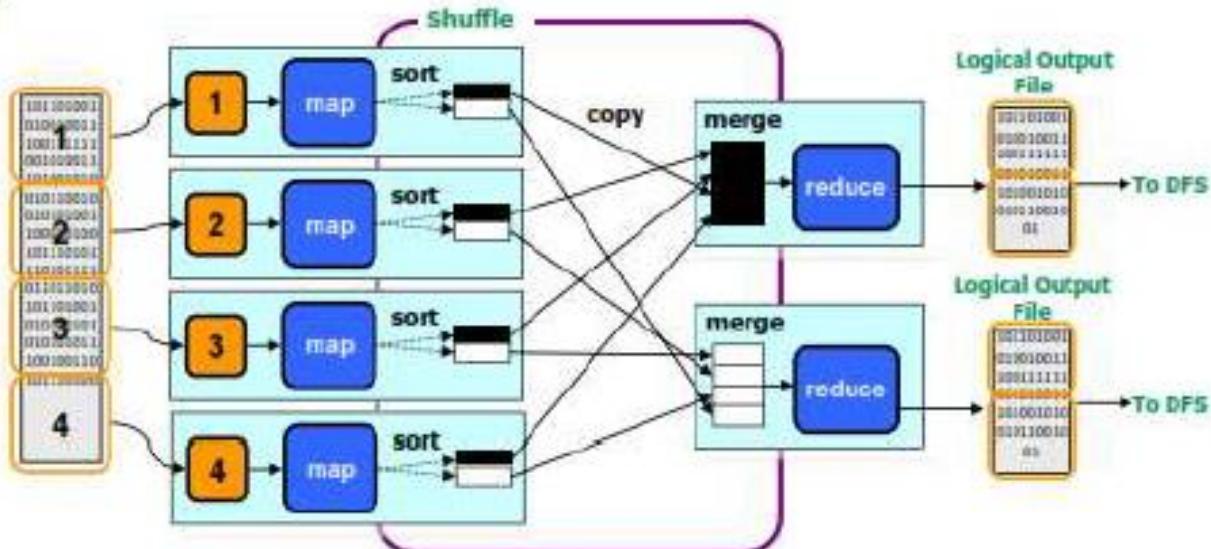
- Le traitement de cette phase est **préprogrammé** dans MapReduce
- Le résultat (pairs *<clé, valeur>*) produit par chaque Map est localement regroupée (tri) par clé.
- **Un nœud** est choisi pour traiter les données (pairs) **pour chaque clé unique**
- Les pairs de même clé sont envoyés (copiés) au même nœud qui exécutera la phase Reduce sur ces données.
- Avant l'exécution de la tâche Reduce, les pairs de même clé sont **fusionnés** sur ce nœud pour former un seul pair.



MapReduce: La phase de Reduce

Reducer

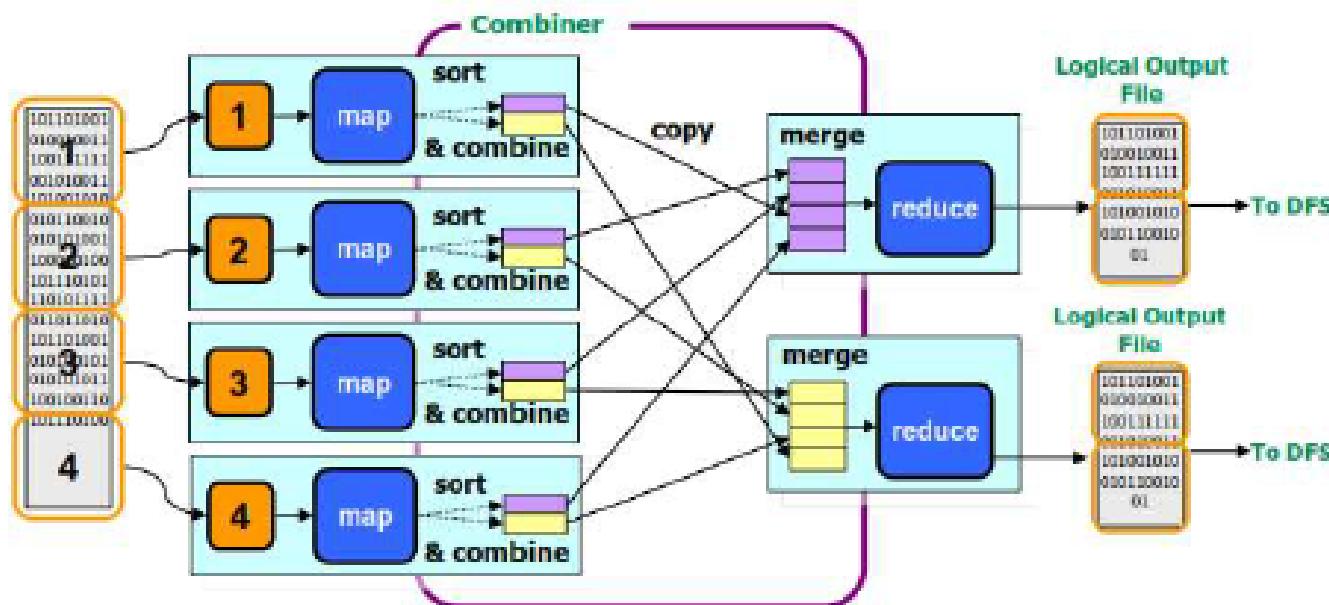
- Petit programme (généralement) qui traite toutes les valeurs de la clé dont il est responsable. Ces valeurs sont passées au **Reducer** sous forme d'un tableau.
- Chaque réducteur écrit son résultat **dans son propre fichier de sortie sur HDFS**



MapReduce: La phase de Combine (optionnel)

Combiner

- Les résultats de **Map** sont triés et fusionnés en local avant d'être envoyés aux nœuds de **Reduce**. Ainsi une partie du travail de ces derniers peut être effectuée par les nœuds ayant réalisé l'opération de **Map**.
- Ceci permettra aussi de minimiser le trafic réseau entre les nœuds de **Map** et ceux de **Reduce**.



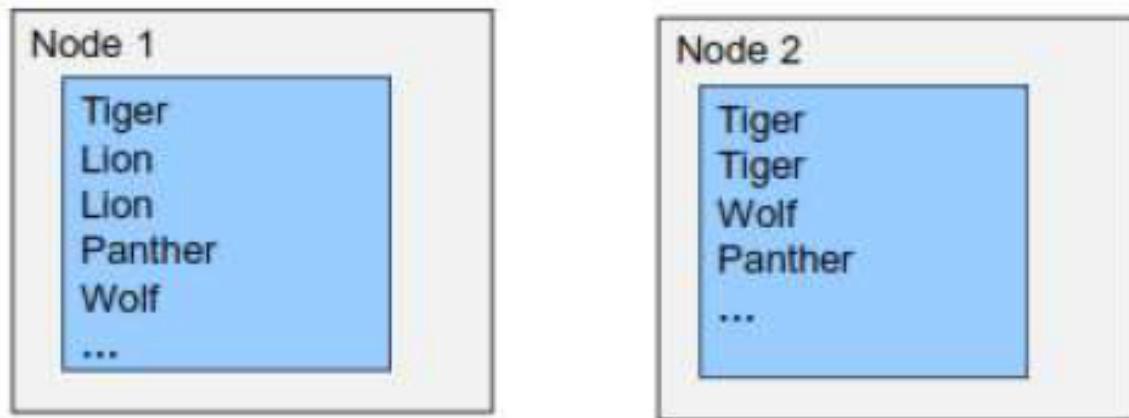
Exemple MapReduce

Objectif: Calculer le nombre de chaque félin (tigre, lion, panthère,) dans un fichier texte sur HDFS contenant les noms d'animaux (un/ligne) qui se répètent.

- En SQL, on aurait pu avoir le résultat par la requête:

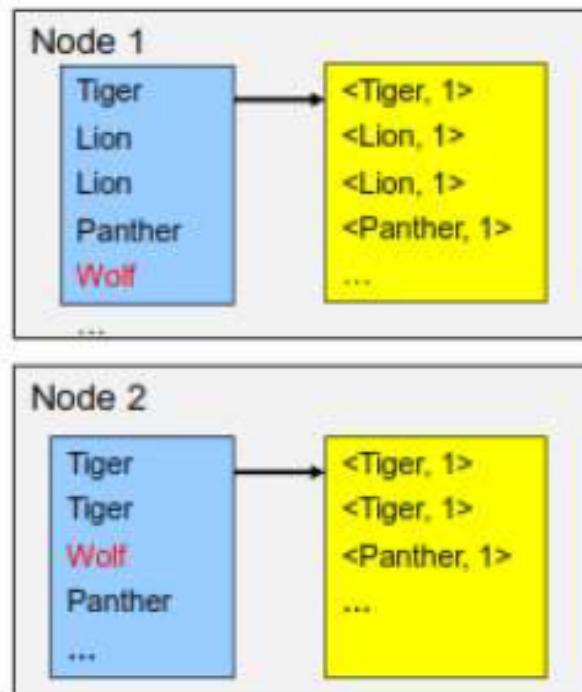
```
SELECT COUNT(NAME) FROM ANIMALS  
WHERE NAME IN (Tiger, Lion ...)  
GROUP BY NAME;
```

- Le fichier a été divisé, par exemple, en deux blocs sur deux nœuds.



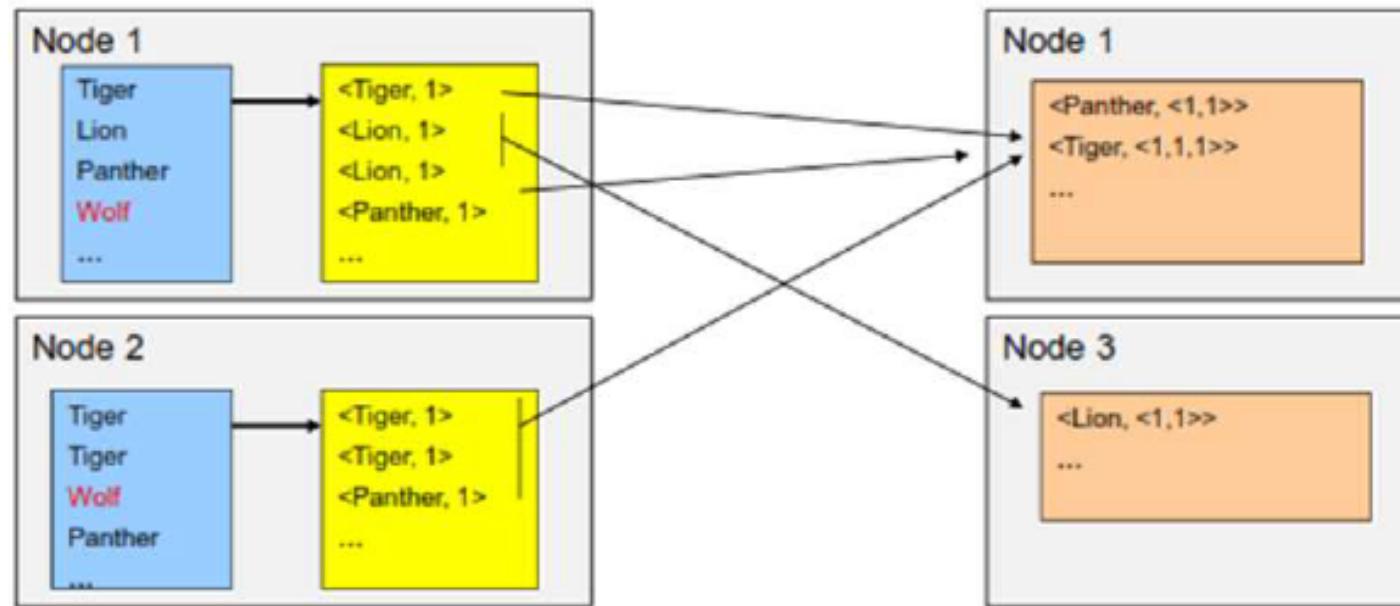
Exemple MapReduce: La tâche Map

- MapReduce peut diviser automatiquement le fichier aux sauts de ligne
- Filtrer les noms de félin et produire des pairs **<clé, valeur>** comme suit:
<nom, 1>
- La tâche de **Map** est exécutée sur chaque **split** localement (*sur les nœuds qui hébergent les blocs de données*)



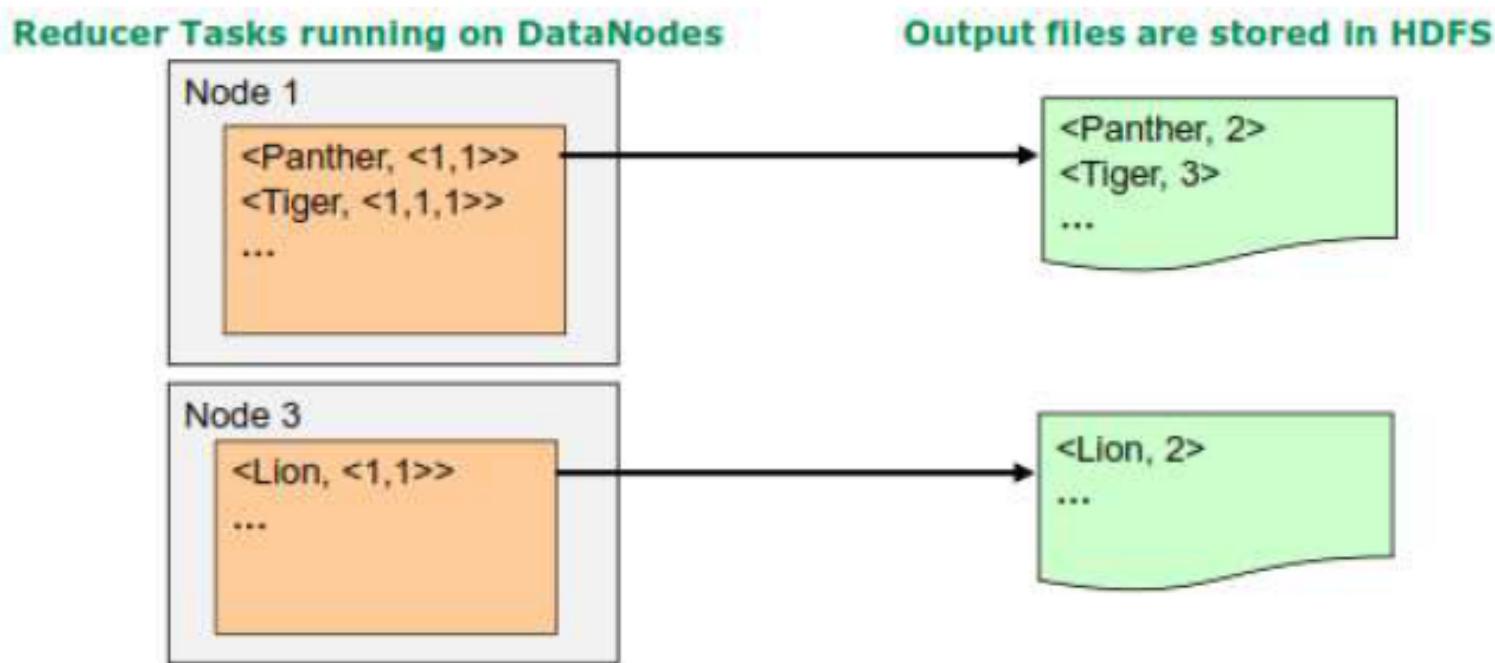
Exemple MapReduce: La tâche Shuffle

- Déplacer toutes les **valeurs** d'une clé vers le même nœud cible
- Les tâches **Reduce** peuvent s'exécuter sur n'importe quel nœud (*nœuds 1 et 3 de l'exemple*)
- Le nombre de tâches **Map** et **Reduce** n'est pas forcément le même.
- Généralement le nombre de tâches **Reduce** est plus petit que celui des **Map**.



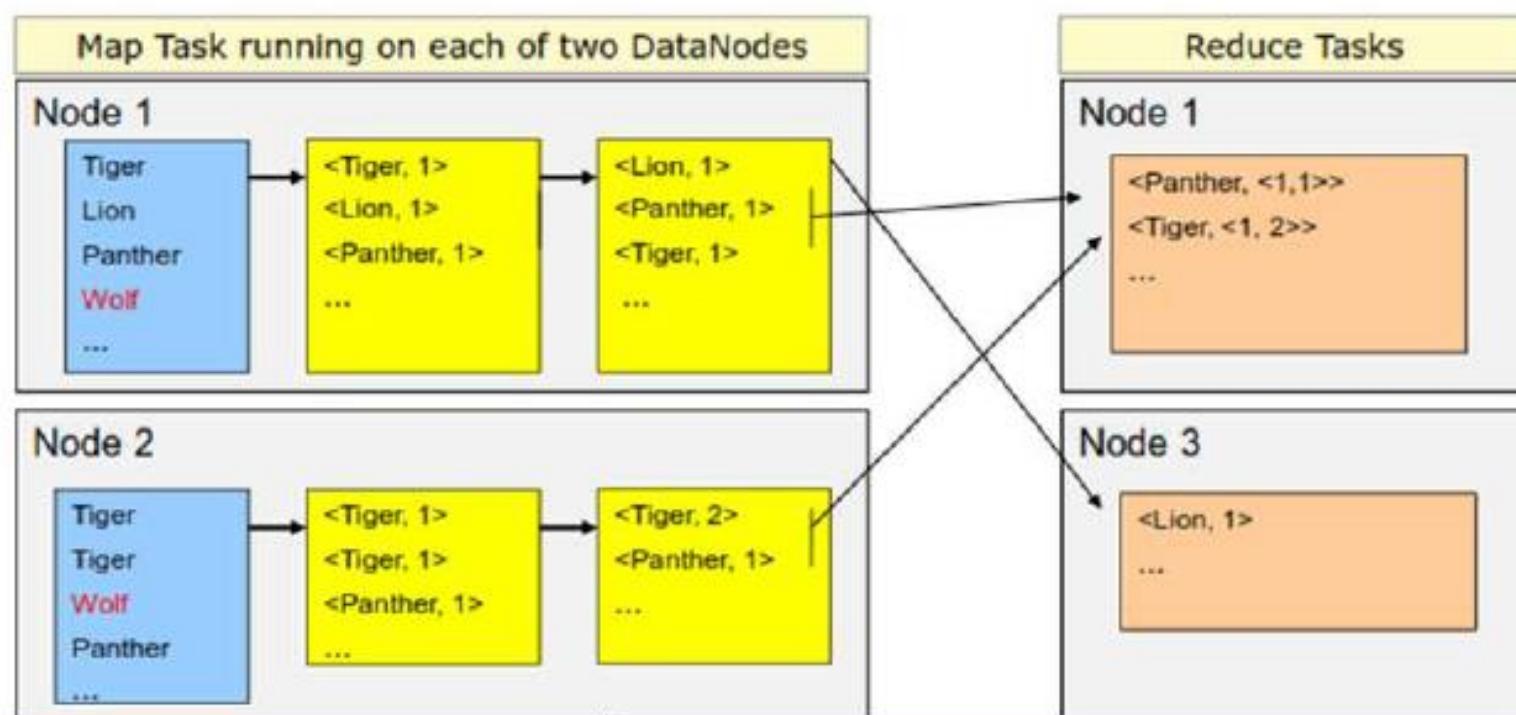
Exemple MapReduce: La tâche Reduce

- La tâche **Reduce** calcule des valeurs agrégées pour chaque clé, dans notre cas, le nombre d'occurrence de la clé (*nom d'animal*)
- La sortie (résultat) d'une tâche **Reduce** est écrite dans un fichier HDFS



Exemple MapReduce: La tâche Combine

- Pour améliorer les performances, les résultats de Map sont **agrégées** sur les nœuds l'ayant produit (avant la phase **Shuffle**)
- On réduit la quantité de données envoyées sur le réseau

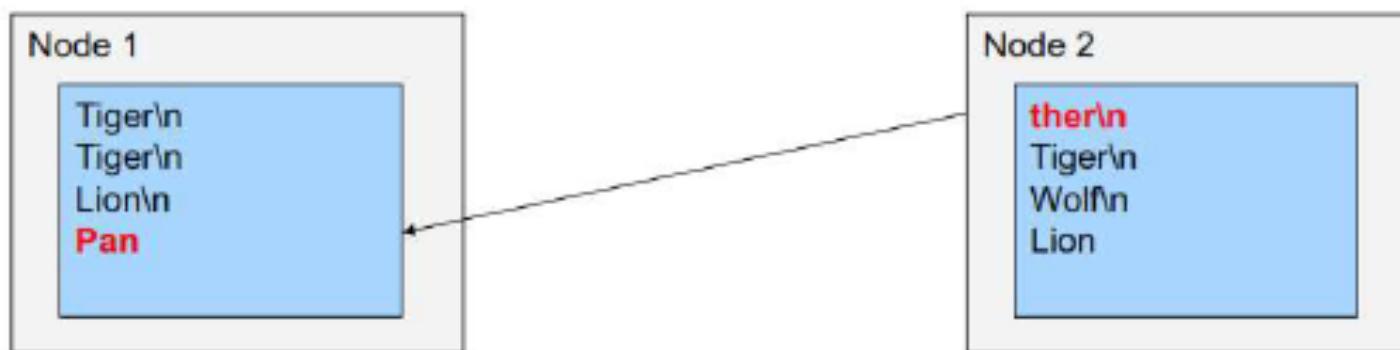


MapReduce: SPLITS

- Les fichiers dans Hadoop sont stockés dans des blocs (128 Mo).
- MapReduce divise les données en fragments ou **Splits**.
- Une tâche **Map** est exécutée sur chaque Split.
- La plupart des fichiers sont sous forme d'enregistrements séparés par des caractères bien définis.
- Le caractère le plus courant est le caractère de fin de ligne.
- La classe **InputFormat** est chargée de prendre un fichier HDFS et le transformer en **Splits** (**InputSplit**).

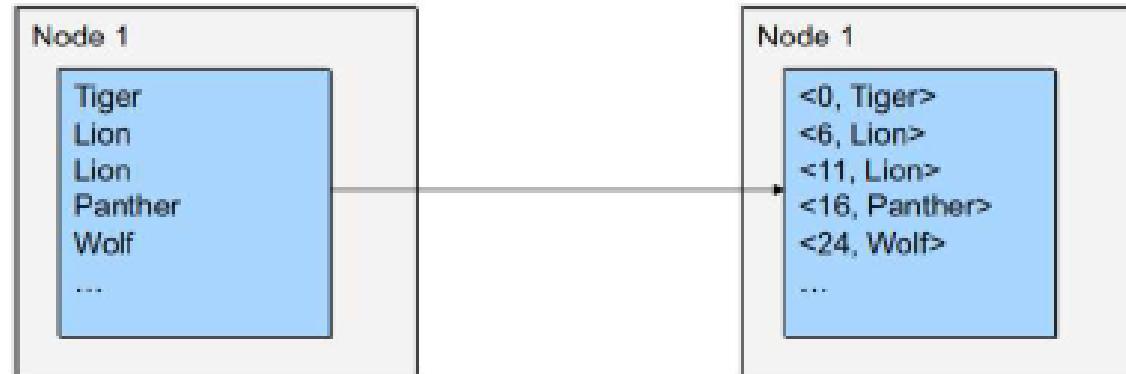
MapReduce: RecordReader

- La plupart du temps, la fin d'un split ne se trouve pas à la fin d'un bloc
- Les fichiers sont lus dans des **Records** par la classe **RecordReader**.
- Dans le cas où le dernier enregistrement d'un bloc se termine dans autre, HDFS enverra la partie manquante du dernier enregistrement via le réseau



MapReduce: InputFormat

- Les tâches MapReduce lisent les fichiers en définissant une classe **InputFormat**.
 - Les tâches Map attendent des pairs **<clé, valeur>**
- Pour lire des fichiers texte de lignes délimitées, Hadoop fournit la classe **TextInputFormat**.
- Elle retourne un pair **<clé, valeur>** par ligne
- La **valeur** est le contenu de la ligne
- La **clé** est le décalage par rapport au début de la première ligne.

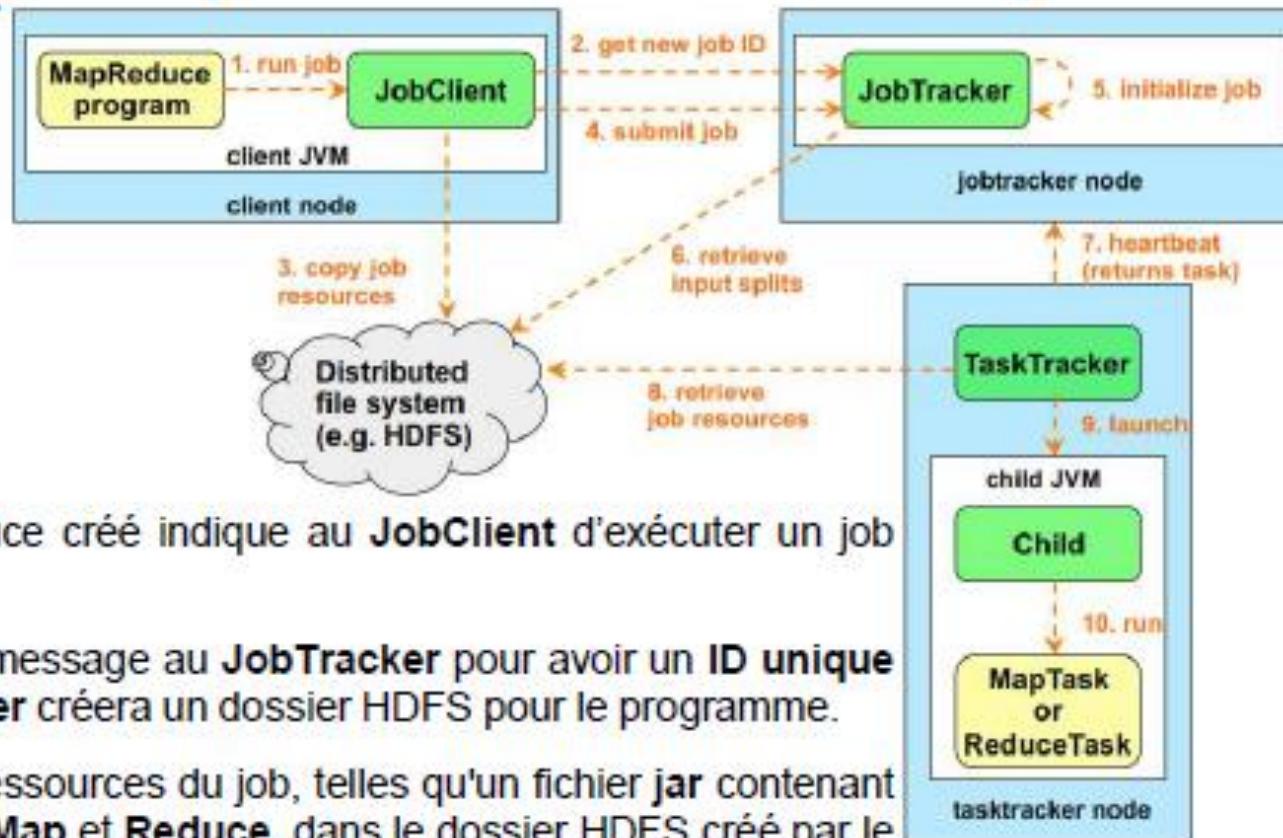


MapReduce: Code de base de la tâche Map

```
public class MyMapper extends Mapper<LongWritable, Text, Text, IntWritable>
{
    ..... // attributs

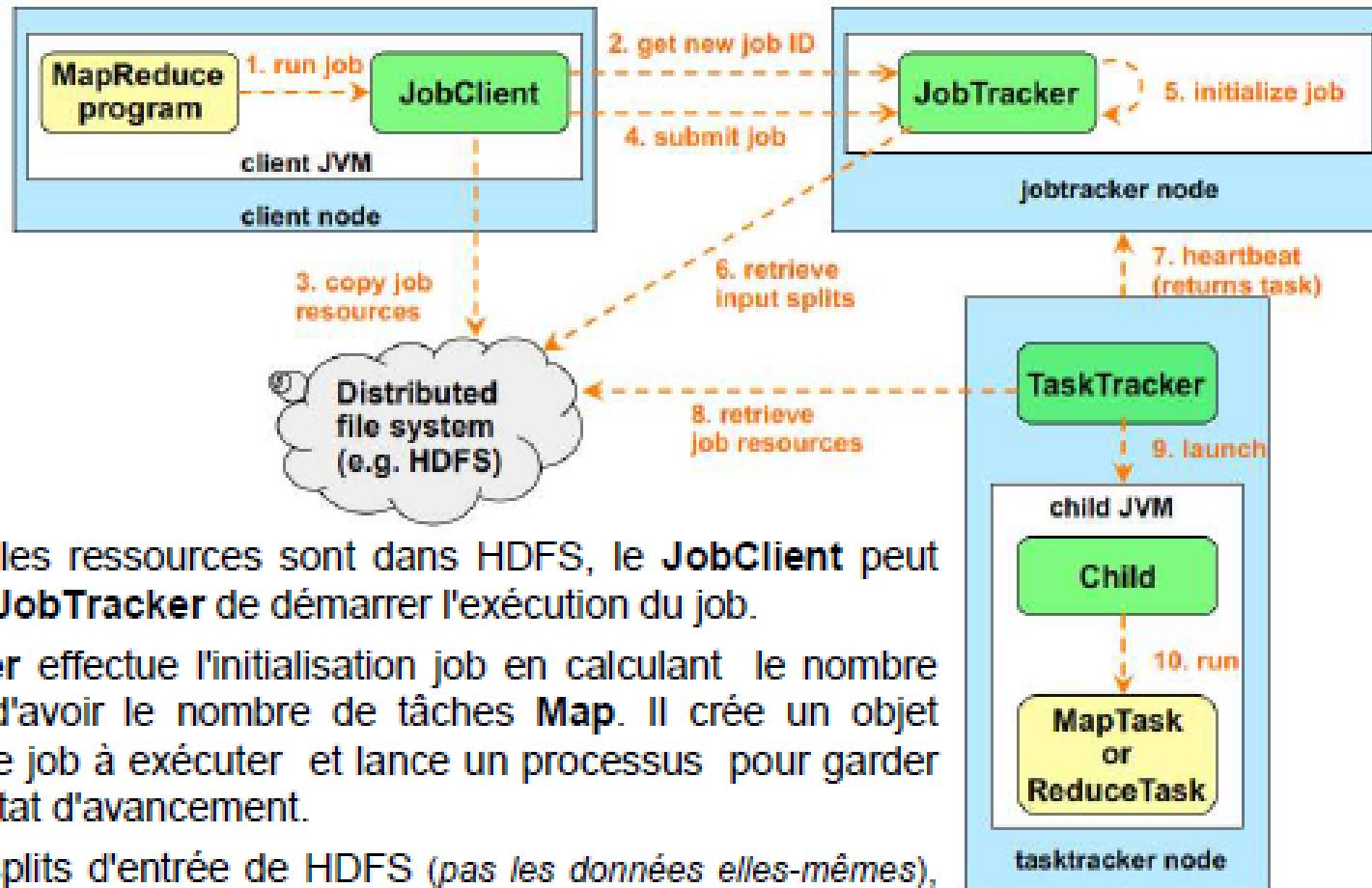
    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException
    {
        ..... // traitement de value
        context.write( new Text(...), new IntWritable(...) ); // génération d'un pair
    }
}
```

MapReduce: Etapes d'exécution d'un job



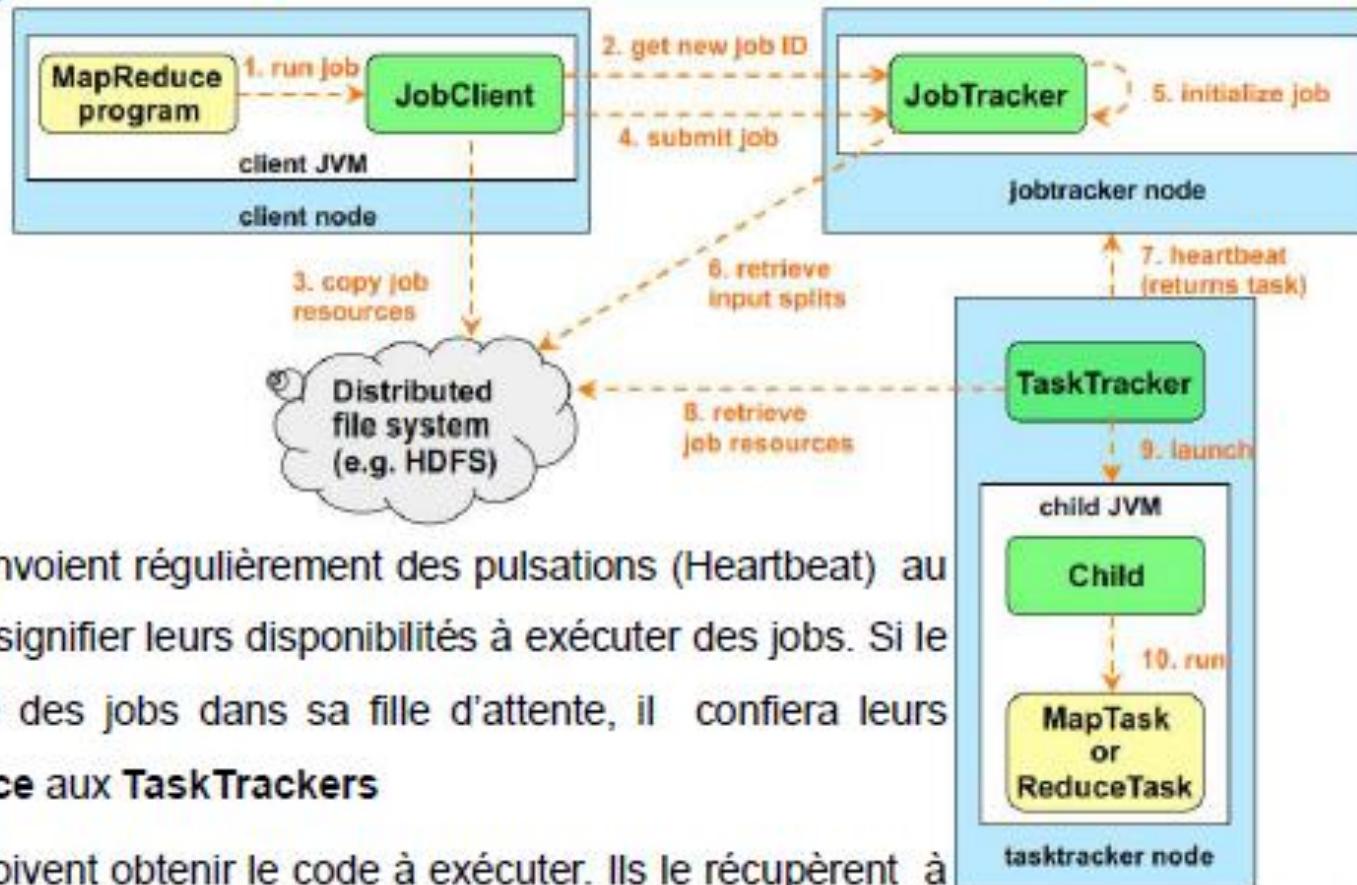
1. Le programme MapReduce créé indique au **JobClient** d'exécuter un job MapReduce.
2. Le **JobClient** envoie un message au **JobTracker** pour avoir un **ID unique** pour le job. Le **JobTracker** créera un dossier HDFS pour le programme.
3. Le **JobClient** copie les ressources du job, telles qu'un fichier **JAR** contenant le code Java des tâches **Map** et **Reduce**, dans le dossier HDFS créé par le **JobTracker** et nommé d'après l'ID du job. Le JAR du job est copié avec une réplication haute (*paramètre mapreduce.submit.replication* dans **mapred-site.xml**, par défaut=10), de sorte qu'il y est beaucoup de copies à travers le cluster pour les **TaskTrackers**.

MapReduce: Etapes d'exécution d'un job



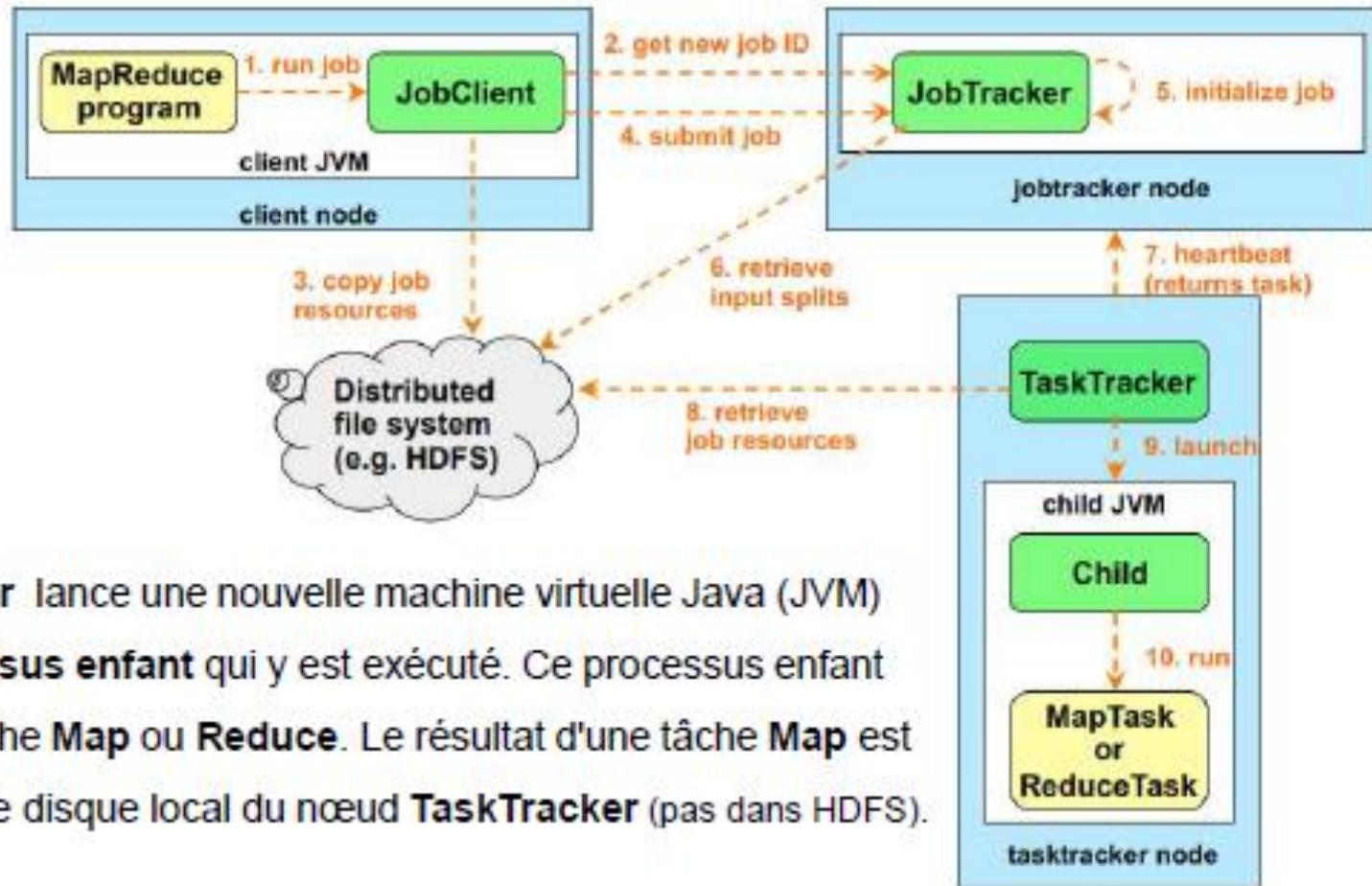
4. Une fois que les ressources sont dans HDFS, le **JobClient** peut demander au **JobTracker** de démarrer l'exécution du job.
5. Le **JobTracker** effectue l'initialisation job en calculant le nombre de split afin d'avoir le nombre de tâches **Map**. Il crée un objet représentant le job à exécuter et lance un processus pour garder trace de son état d'avancement.
6. Il extrait ces splits d'entrée de HDFS (*pas les données elles-mêmes*), récupère le nombre maximal des tâches **Reduce** à créer (paramètre `Mapred.Reduce.tasks` dans `mapred-site.xml`, par défaut=1 100% de la taille du cluster)

MapReduce: Etapes d'exécution d'un job



7. Les **TaskTrackers** envoient régulièrement des pulsations (Heartbeat) au **JobTracker** pour lui signifier leurs disponibilités à exécuter des jobs. Si le **Jobtracker** possède des jobs dans sa file d'attente, il confiera leurs tâches **Map** et **Reduce** aux **TaskTrackers**
8. Les **TaskTrackers** doivent obtenir le code à exécuter. Ils le récupèrent à partir du dossier HDFS partagé contenant les ressources du job copiées dans l'étape 3.

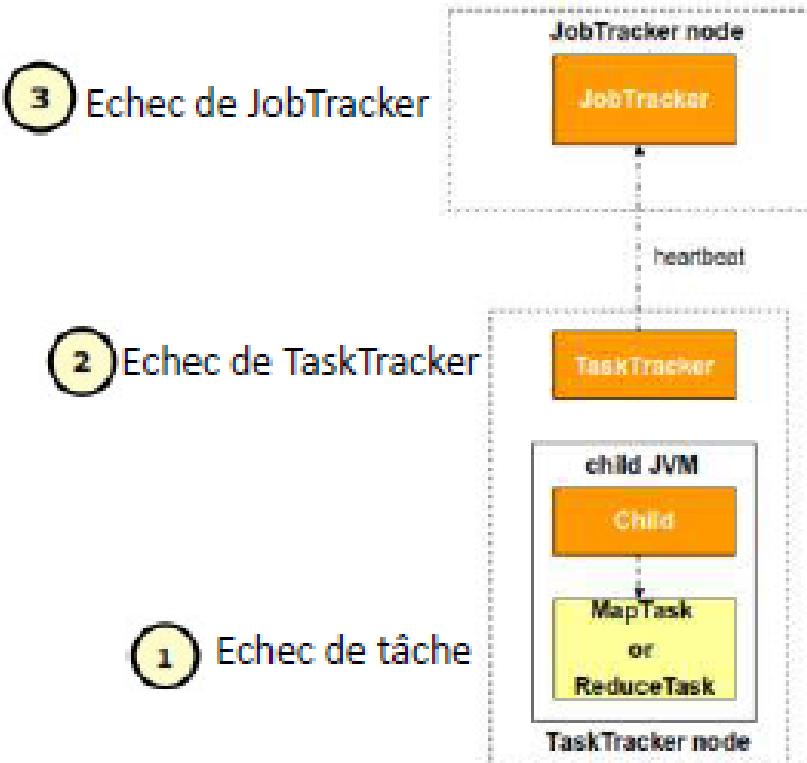
MapReduce: Etapes d'exécution d'un job



9. Le **TaskTracker** lance une nouvelle machine virtuelle Java (JVM) avec un **processus enfant** qui y est exécuté. Ce processus enfant exécute une tâche **Map** ou **Reduce**. Le résultat d'une tâche **Map** est enregistré sur le disque local du nœud **TaskTracker** (pas dans HDFS).

MapReduce: Tolérance aux pannes

- Le moyen principal utilisé par Hadoop pour assurer la tolérance aux pannes consiste à redémarrer des tâches en cas d'échec.
- Si un TaskTracker n'envoie pas une pulsation au JobTracker pendant une période donnée (par défaut, 1 minute), le JobTracker le considère bloqué.
- Le JobTracker sait les tâches Map et Reduce attribuées à chaque TaskTracker.
- En cas d'échec d'un TaskTracker pendant l'exécution de ses tâches Map, elles seront toutes attribuées aux autres TaskTrackers.
- En cas d'échec d'un TaskTracker pendant l'exécution de ses tâches Reduce, les autres TaskTrackers ré-exécuteront seulement les tâches Reduce en cours sur le TaskTracker en échec.



MapReduce 2 – YARN: Limites de MR v1

- Dans l'architecture **MapReduce**, l'exécution de jobs est contrôlée par deux types de processus:
 - Le **JobTracker**, l'unique maître, qui coordonne et attribue toutes les tâches **Map** et **Reduce** exécutées sur le cluster,
 - Un certain nombre de **TaskTrackers**, processus subordonnés, qui exécutent les tâches assignées et rendent compte périodiquement de leur avancement au **JobTracker**.
- Les limitations les plus sérieuses de **MapReduce V1** sont:
 - L'évolutivité
 - L'utilisation des ressources
 - Prise en charge de job non **MapReduce**.

MapReduce 2 – YARN: L'évolutivité de MRv1

- Dans MapReduce, le JobTracker est chargé de **deux responsabilités** distinctes:
 - **Gestion des ressources** de calcul dans le cluster, ce qui implique de gérer la liste des **nœuds actifs**, la liste des **Map** et **Reduce** et des emplacements (**Slots**) disponibles et occupés, ainsi que d'affecter les emplacements disponibles aux jobs et tâches appropriés, en fonction de la politique d'ordonnancement sélectionnée (FIFO,).
 - **Coordination de toutes les tâches** en cours d'exécution sur un cluster, ce qui implique de donner aux **TaskTrackers** l'ordre de démarrer des tâches **Map** et **Reduce**, de surveiller l'exécution des tâches, de relancer les tâches ayant échoué, d'exécuter **spéculativement** des tâches lentes, ... etc.
- Les **responsabilités** du **JobTracker** posaient d'importants problèmes d'évolutivité, en particulier sur un très grand **cluster** où le **JobTracker** devait suivre en permanence des milliers de **TaskTrackers**, des centaines de jobs et des dizaines de milliers de tâches **Map** et **Reduce**

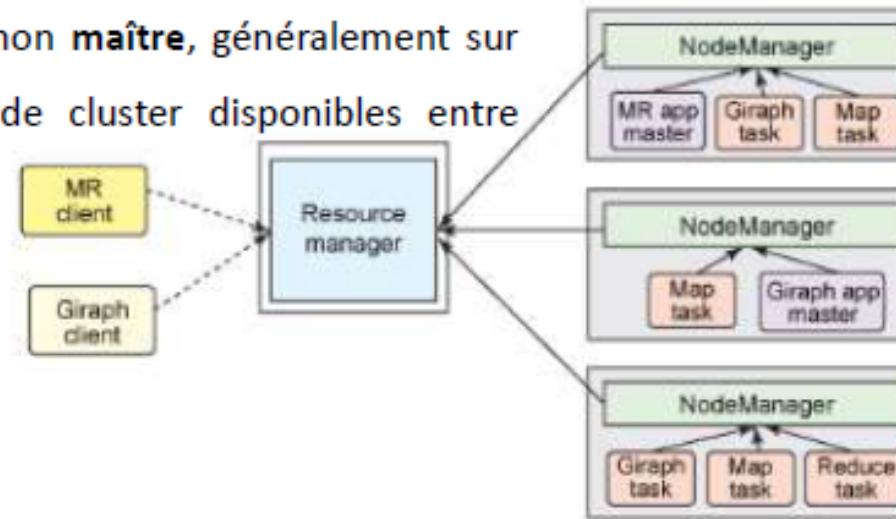
YARN (Yet Another Resource Negotiator)

- MapReduce V1 a subi une refonte complète avec YARN, scindant les deux fonctionnalités principales de JobTracker (*gestion des ressources et planification / surveillance des jobs*) en démons distincts.

- ResourceManager (RM)

- L'unique ResourceManager s'exécute tant que démon maître, généralement sur une machine dédiée. Il partage les ressources de cluster disponibles entre diverses applications concurrentes.
- Lorsqu'un utilisateur soumet une application (job), une instance d'un processus léger appelé ApplicationMaster est démarrée pour coordonner l'exécution de toutes les tâches de l'application (surveillance, redémarrage de tâches ayant échoué, exécution speculative de tâches lentes,) Ces responsabilités étaient précédemment attribuées au JobTracker pour tous les jobs.

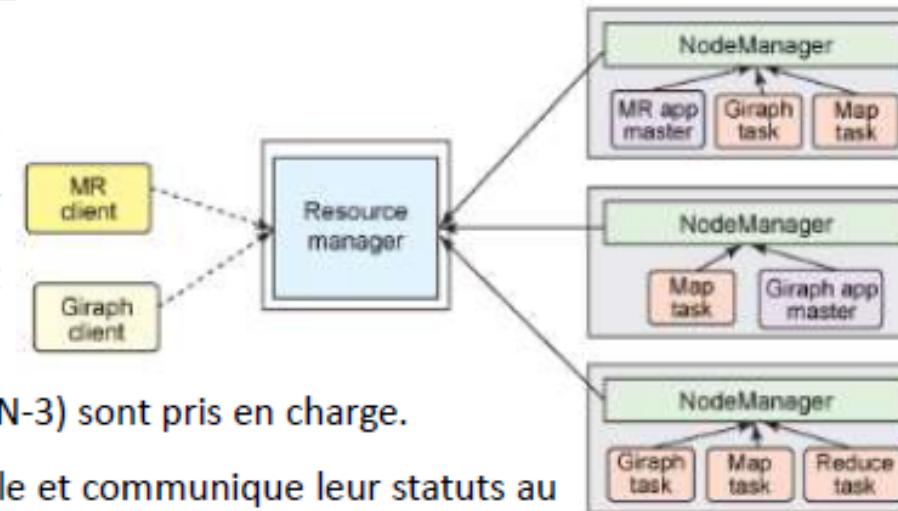
- L'ApplicationMaster et les tâches de son application s'exécutent dans des conteneurs de ressources contrôlés par les NodeManagers



YARN (Yet Another Resource Negotiator)

• NodeManager (NM)

- Il dispose d'un nombre de conteneurs de ressources créés dynamiquement. La taille d'un conteneur dépend de la quantité de ressources qu'il contient (mémoire, CPU, disque, E/S réseau)
- Actuellement, seuls la mémoire et le processeur (YARN-3) sont pris en charge.
- Le **NodeManager** démarre les conteneurs, les surveille et communique leur statuts au **ResourceManager**
- Le nombre de conteneurs sur un nœud dépend de paramètres de la configuration et le nombre total de ressources du nœud (nombre total de processeurs et taille mémoire)
- **ApplicationMaster** peut exécuter n'importe quel type de tâche dans un conteneur. Par exemple, **ApplicationMaster MapReduce** demande à un conteneur de lancer une tâche **Map** ou **Reduce**, tandis qu'un **ApplicationMaster Giraph** demande à un conteneur d'exécuter une tâche **Giraph**.
- Dans YARN, **MapReduce (MRv2)** est considéré simplement comme une application distribuée qui tourne sur YARN.

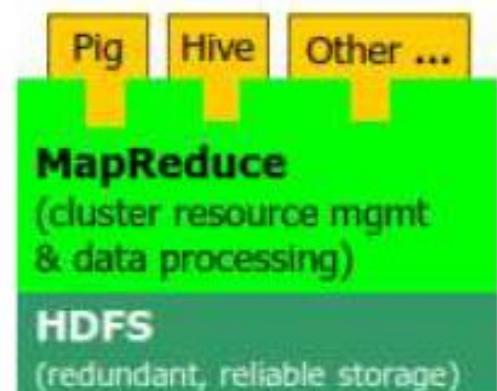


YARN (Yet Another Resource Negotiator)

Hadoop v1 vers Hadoop v2

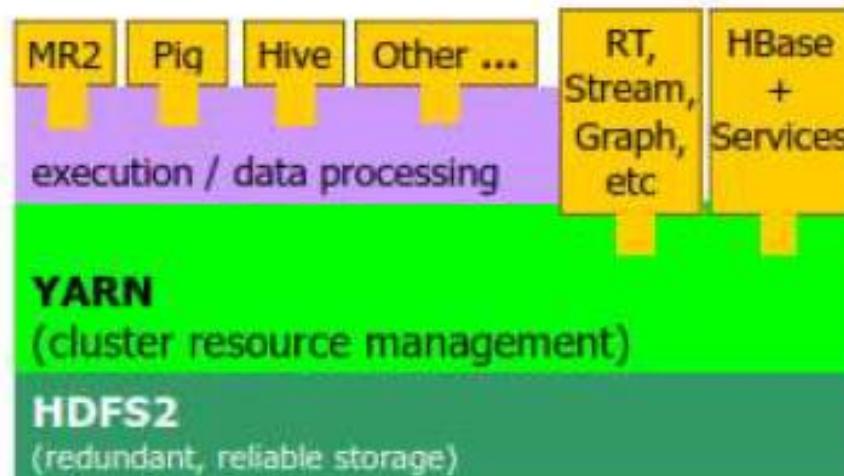
Système à usage unique
Traitement par lots

Hadoop 1.0

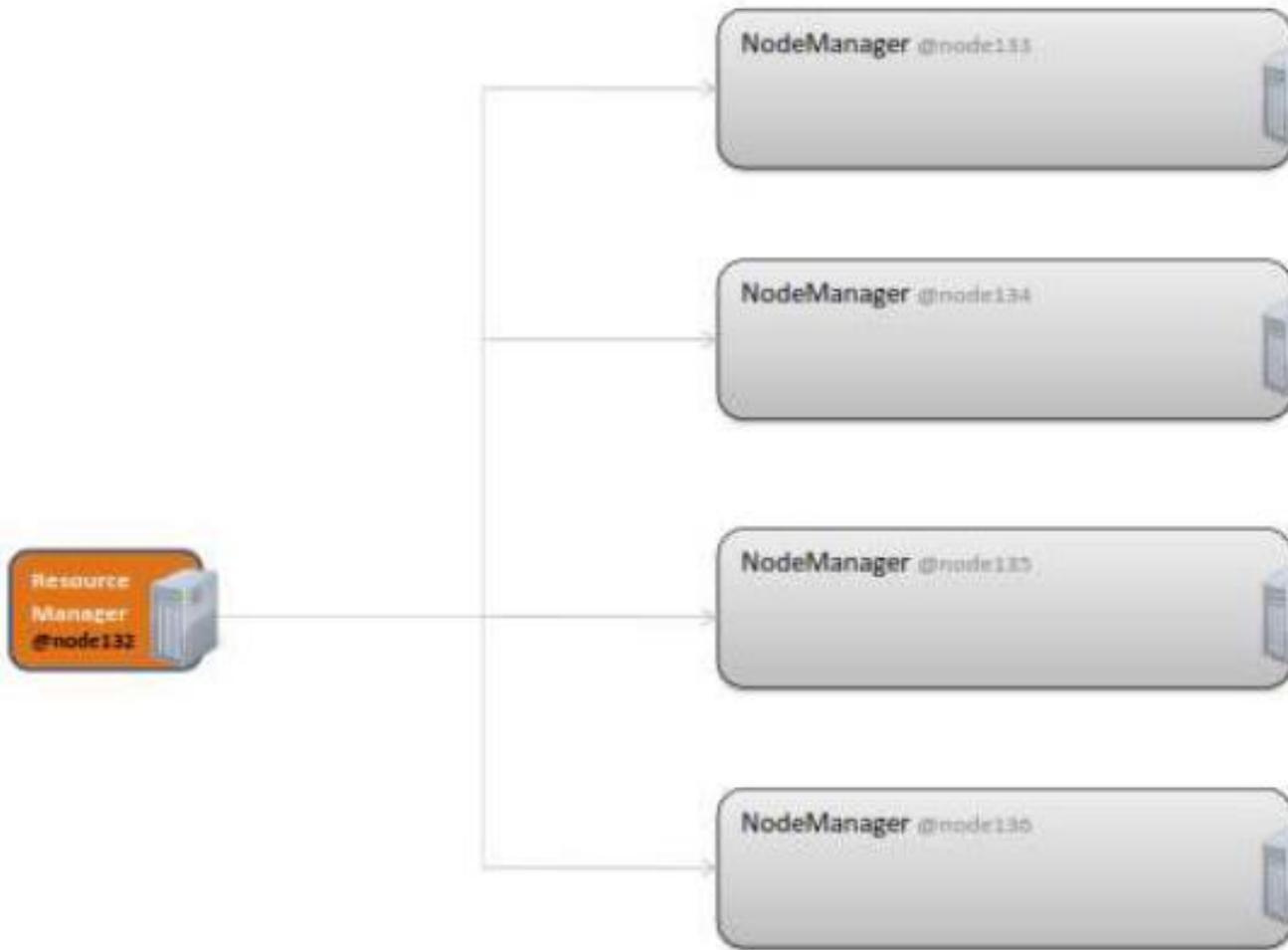


Plateforme à usages multiples
Traitement par lot, interactif, en ligne,
streaming

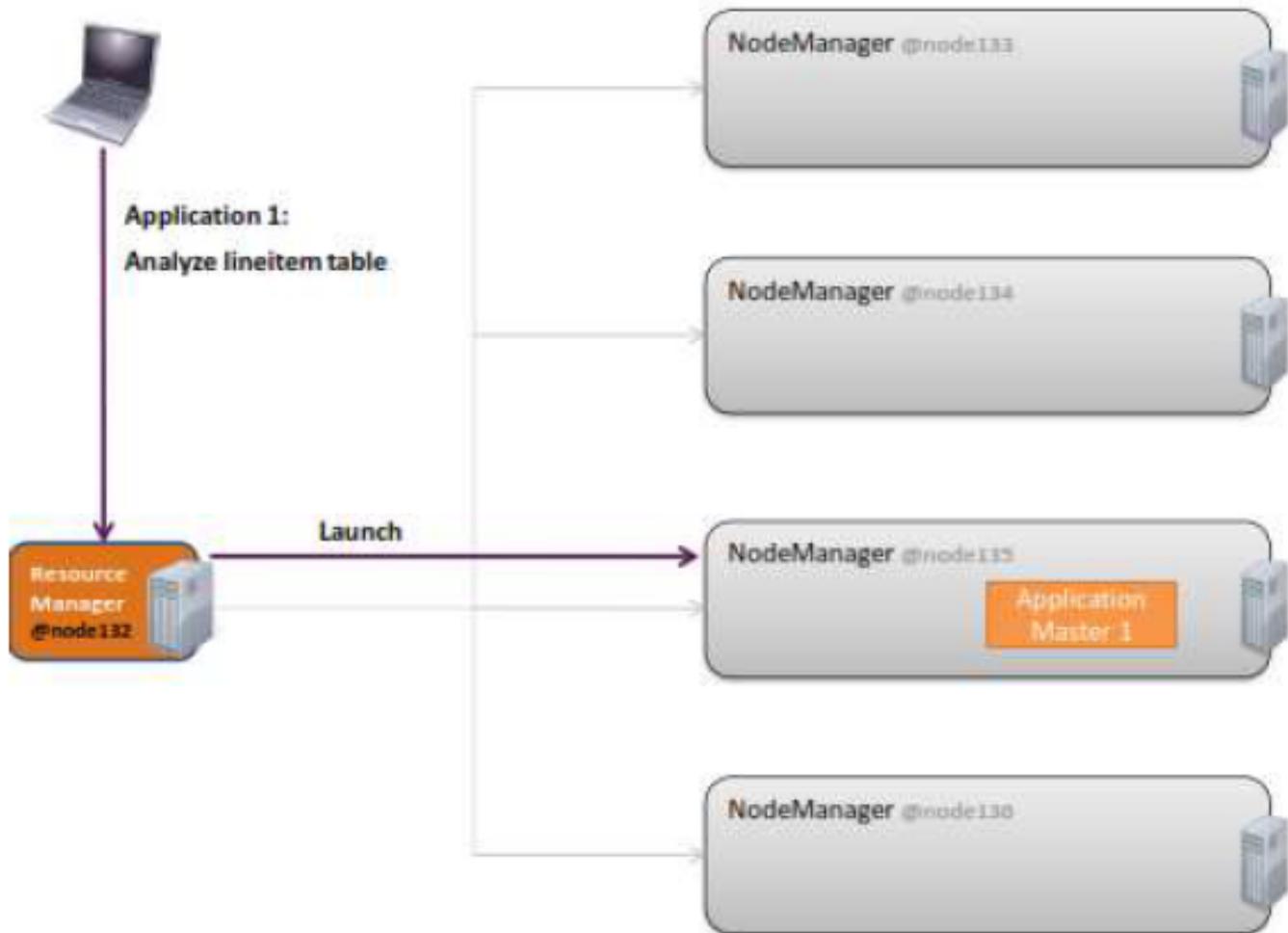
Hadoop 2.0



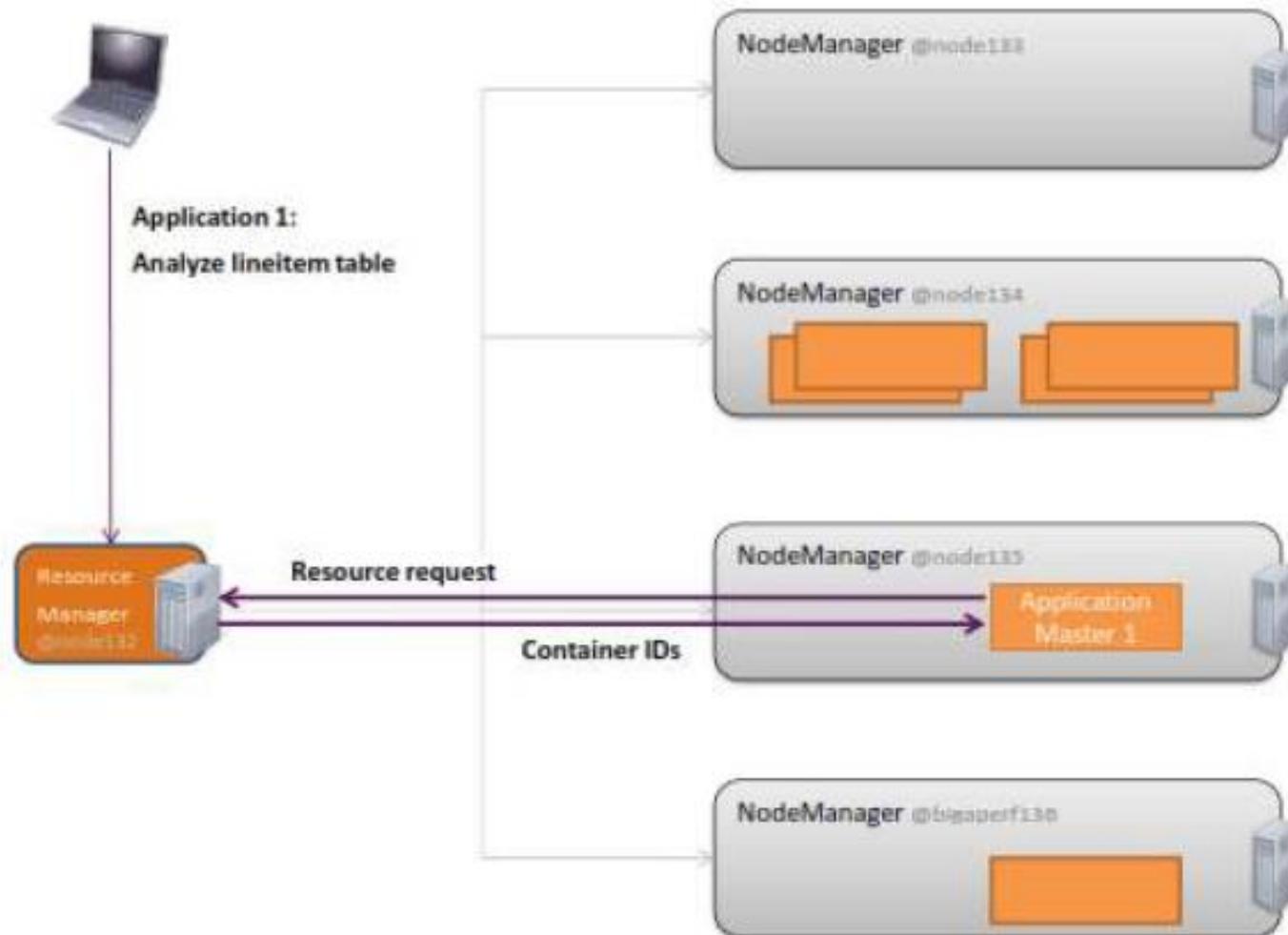
YARN: Exécuter une application dans Yarn (1 / 7)



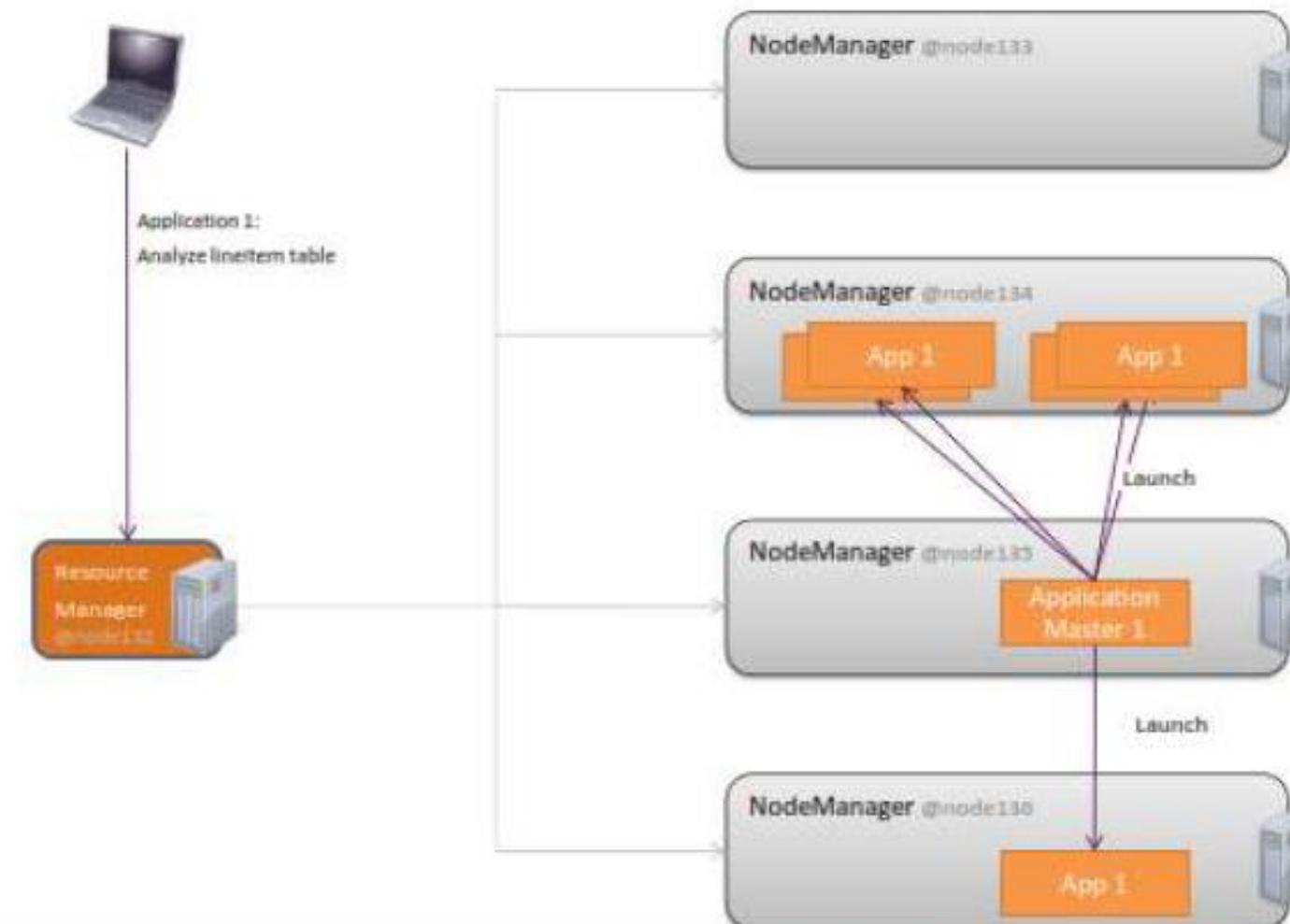
YARN: Exécuter une application dans Yarn (2 / 7)



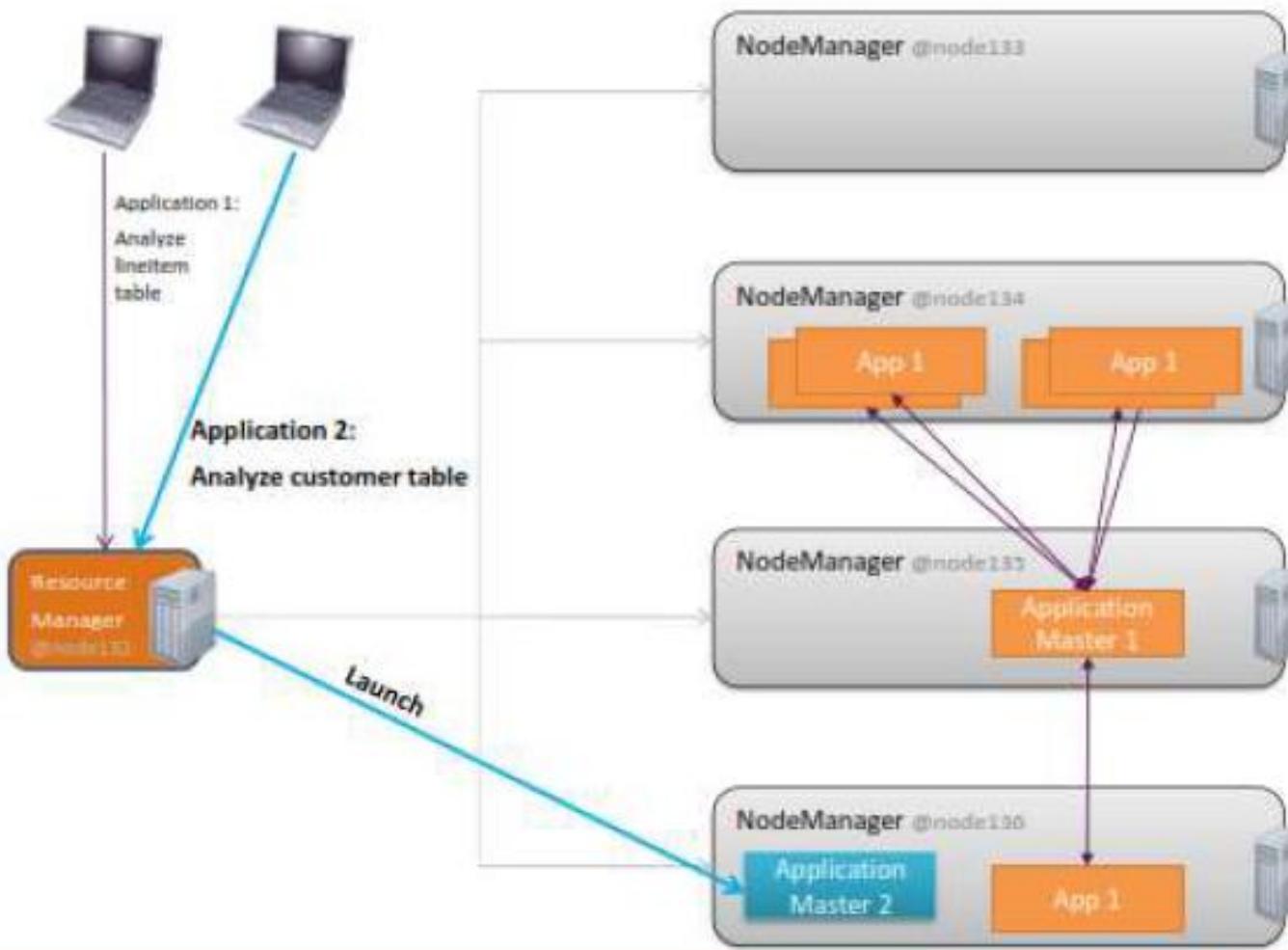
YARN: Exécuter une application dans Yarn (3 / 7)



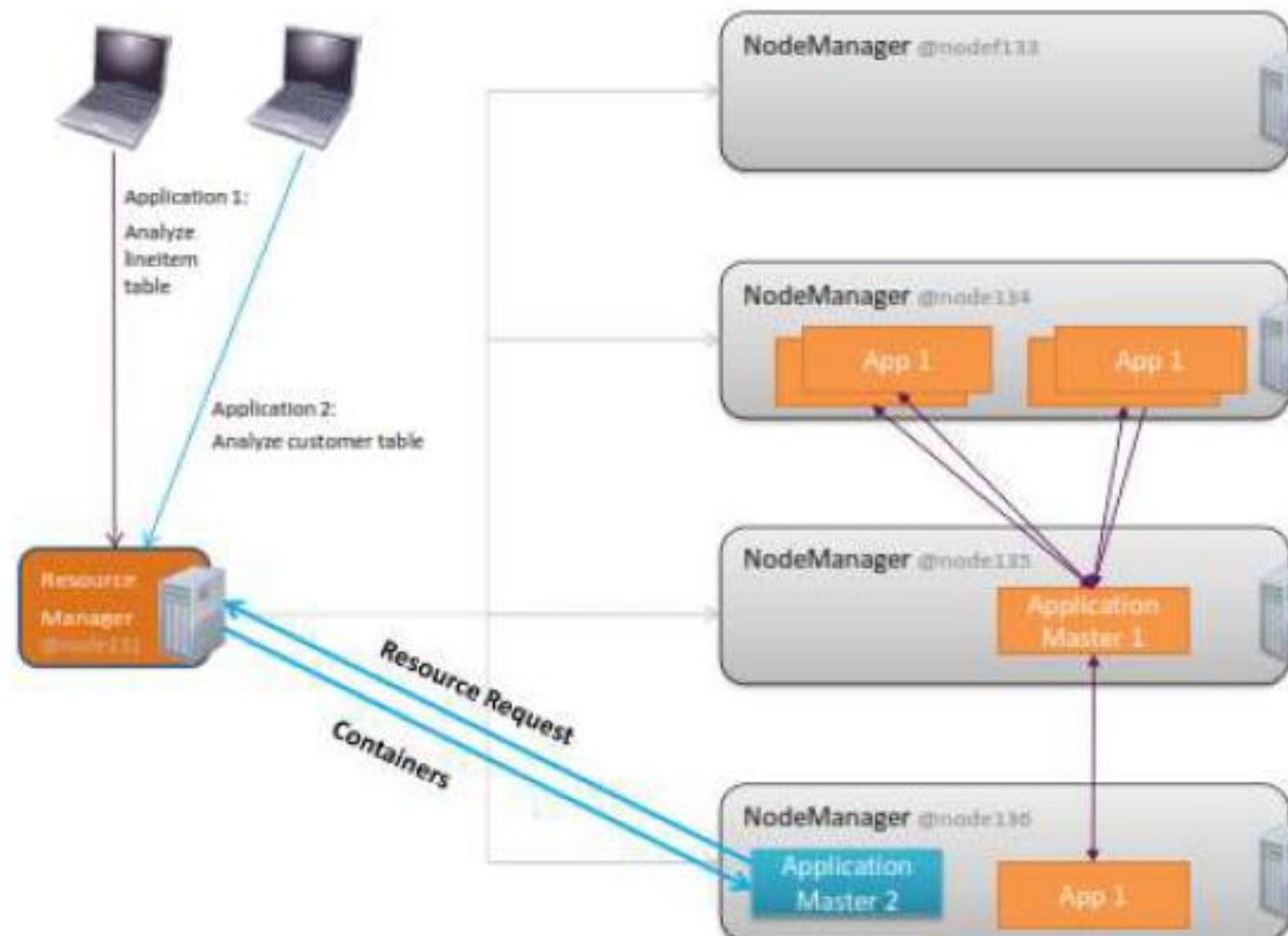
YARN: Exécuter une application dans Yarn (4 / 7)



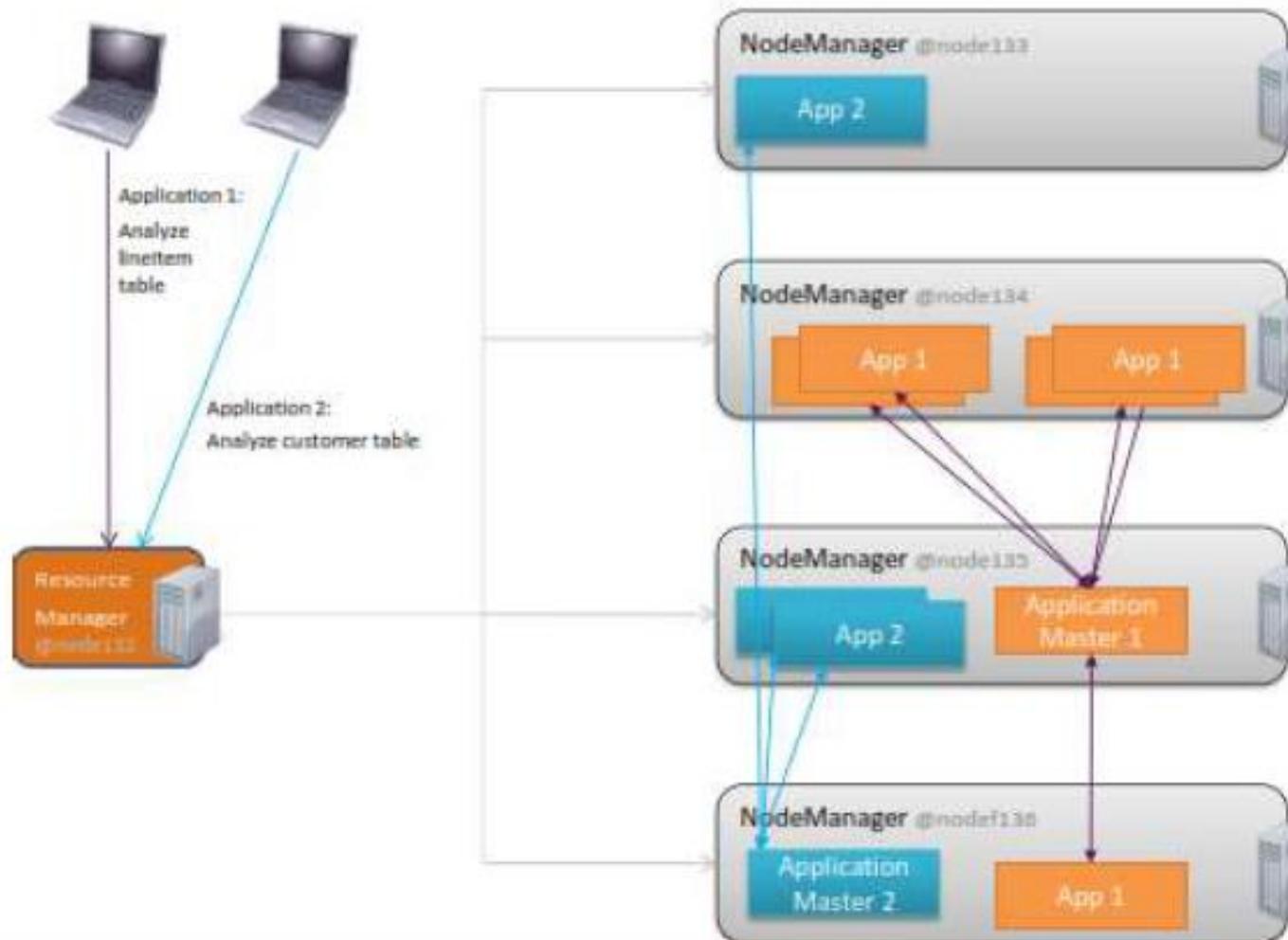
YARN: Exécuter une application dans Yarn (5 / 7)



YARN: Exécuter une application dans Yarn (6 / 7)

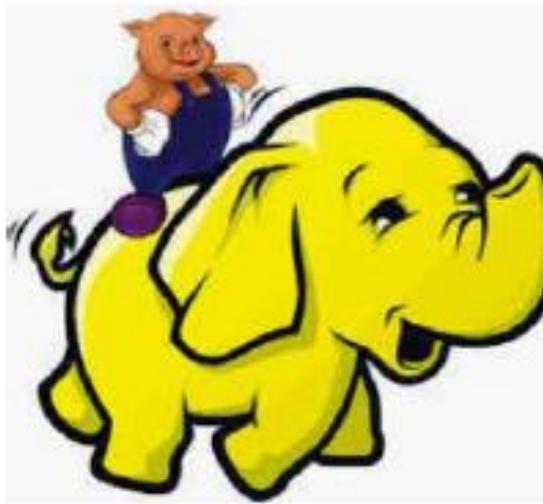


YARN: Exécuter une application dans Yarn (7 / 7)



Travaux Pratiques

Pig

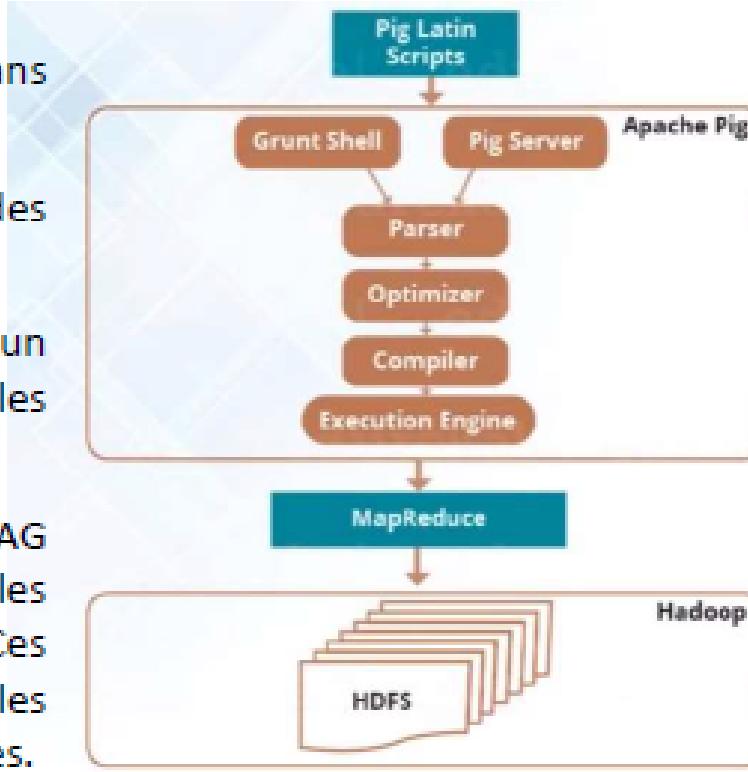


Pig : Introduction

- Crée par Yahoo
- Pig est une **couche d'abstraction** au dessus de Hadoop
- Pig est constitué de:
 - Langage **Pig Latin** utilisé pour exprimer des flux de données à obtenir à partir de données en entrée.
 - **Environnement d'exécution** des programmes Pig Latin: deux environnements disponibles
 - Exécution **locale** dans une seule JVM
 - Exécution **distribuée** sur un cluster Hadoop.
- **Pig Latin** est un *langage de script* pour explorer des données de masse.
- Un programme Pig Latin est constitué d'une série d'opérations, ou **transformations**, appliquées aux données d'entrée pour produire des données de sortie.
- Les opérations décrivent un flux de données
- L'environnement d'exécution Pig traduit ces opérations en une série de jobs **MapReduce** sur le cluster Hadoop.

Pig: Architecture

- **Pig Latin Script:** Contient des commandes Pig dans un fichier (.pig)
- **Grunt Shell:** Shell Pig pour exécuter des commandes pour une exploration interactive des données.
- **Pig Server:** regrouper les commandes Pig dans un fichier et l'envoyer vers ce serveur afin de les exécuter.
- **Parser:** vérifie la syntaxe du script et produit un DAG (graphe acyclique dirigé) représentant les commandes Pig Latin et les opérateurs logiques. Ces derniers sont représentés comme des nœuds et les flux de données sont représentés comme des arêtes.
- **Optimizer:** Effectue des optimisations logiques telles que la projection afin de réduire le flux de données
- **Compiler:** Compile le plan logique optimisé en une série de jobs MapReduce.
- **Execution engine:** soumet les jobs MapReduce au clusetr Hadoop dans un ordre spécifique pour produire les résultats souhaités. Ces derniers sont affichés ou stockés dans un fichier HDFS.



Apache Pig Architecture

Pig: Modèle de données

Atom: donnée élémentaire qui peut être de plusieurs types.

Elle représente un champ (field) dans un enregistrement par exemple.

Tuple : similaire à une ligne dans une table relationnel et représentée par '()'

Exemple: (Rabat, 2008, 3, 60)

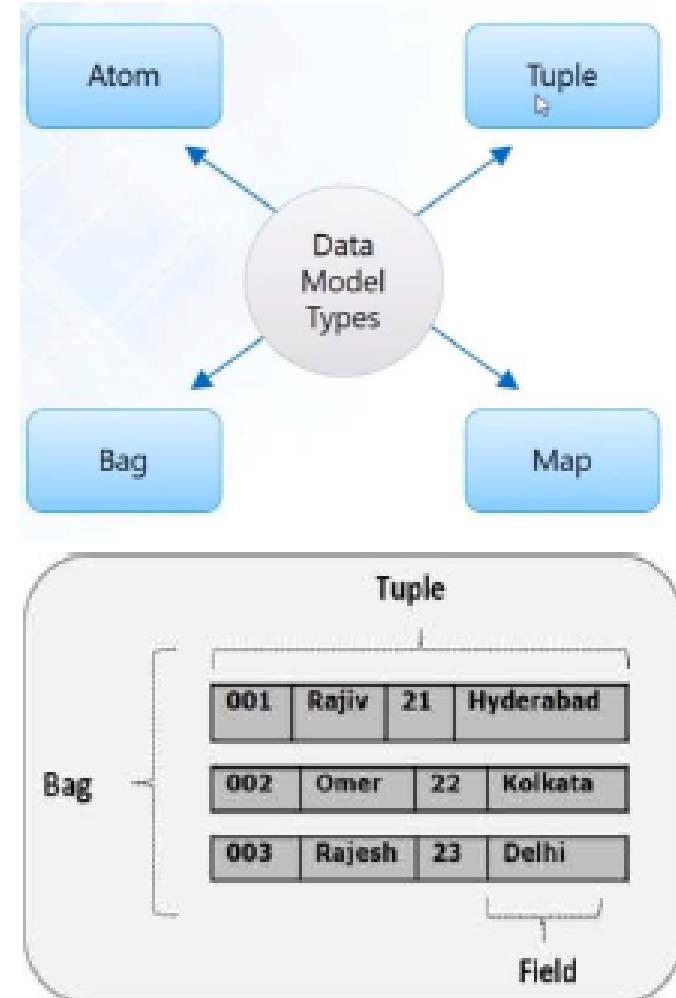
Bag: Ensemble non ordonné de tuples qui peuvent avoir n'importe quel nombre de champs (*schéma flexible*). Un bag est mis entre '{ }'. Il est similaire à une table dans le modèle relationnel.

Exemple: {(Rabat, 2008, 3, 60), (Rabat, 2008, 2, 80)}.

Map: une série de paires clé-valeur. La clé doit être unique et de type chararray. La valeur peut être de n'importe quel type. Un Map est représenté par '[]' où la clé et la valeur sont séparées par '#'

Exemple: [nom #Ali, âge #10]

Exemple de tuple complexe: (3 , [nom#Faridi, age#23] , { (math, 12.5), (info, 15) })



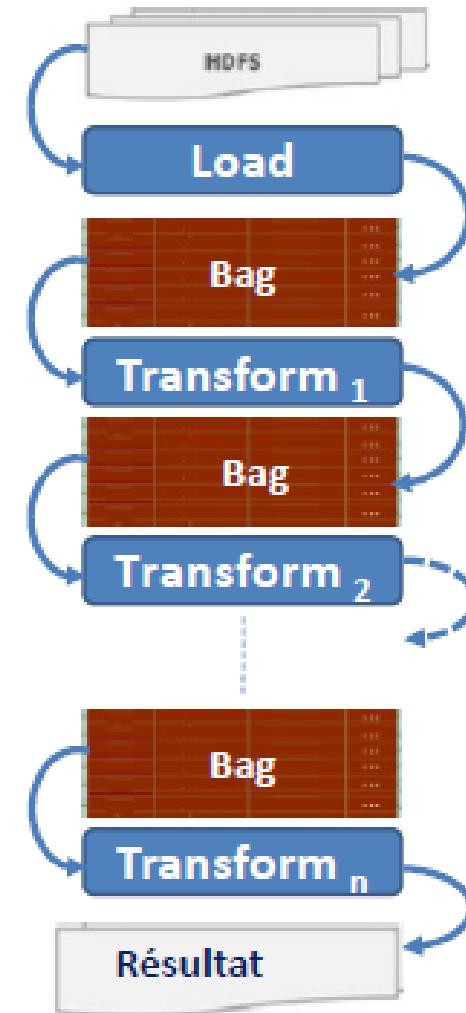
Pig: Structure d'un script Pig Latin

- ❑ Un programme Pig est constitué de 3 étapes:
 - LOAD: pour charger des données à partir de HDFS en leur donnant un schéma.
 - TRANSFORM/PROCESS
 - Opérateurs relationnels: **FILTER, ORDER, DISTINCT, JOIN, GROUP, FOREACH, UNION**, etc.
 - Ces transformations sont traduites en plusieurs tâches Map /Reduce
 - DUMP ou STORE
 - Afficher le résultat sur l'écran ou le stocker dans un fichier HDFS
- ❑ Commentaires: */* commentaire sur plusieurs lignes */* ou *-- commentaire en une ligne*
- ❑ Types de données de Pig:
 - Types simples: **int, long, float, double, chararray, bytearray, boolean**
 - Types complexes:
 - Tuple: ensemble de champs ordonné (Alifi, 38, Prof)
 - Collection de tuples (Bag) {(Faridi, 18, Etudiant), (Nasser, 29, Médecin)}
 - Map de paires clé#valeur

Pig: Structure d'un script Pig Latin

Scénario de traitement dans un script Pig

- Chaque transformation est une application d'un opérateur relationnel (*Sélection-Filtrer, Jointure, Ordre, Groupement, ...*)
- Chaque transformation est traduite en un job MapReduce
- Une transformation peut agir sur plusieurs Bag (*Join, ...*)



Pig: Chargement

nom_bag = LOAD 'source' [USING fonction] [AS schéma]

- *Source*: nom de fichier ou répertoire (pour charger tous ses fichiers)
- *Fonction*: nom d'une fonction de chargement des données. Elle désigne une classe Java chargée d'interpréter les données depuis le système de fichiers (HDFS ou Local) et d'en sortir des types Pig (tuples / bag / map). Il existe dans Pig plusieurs fonctions dont celle par défaut est: **PigStorage("Séparateur")**
- *Schéma*: utilisé pour donner des noms et des types explicites aux différents membres du bag des données d'entrée chargées. On utilise la syntaxe suivante:

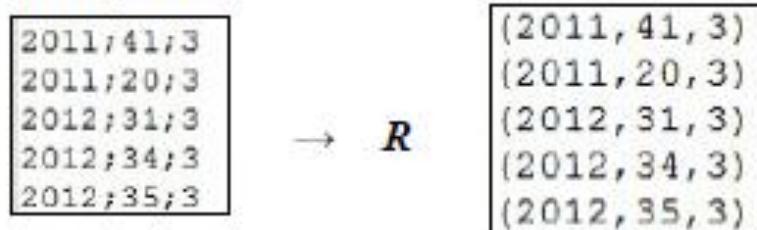
(NOM1:TYPE1, NOM2:TYPE2, ...)

où le type est optionnel

Il est recommandé de préciser explicitement le schéma de chargement avec tous les types et noms des membres définis

Pig: Chargement

Exemple 1: chargement du fichier temp.csv

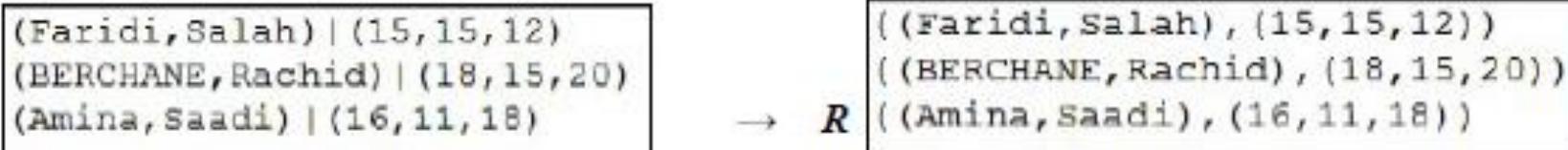


```
R = LOAD '/user/cloudera/pig_lab/input/temp.csv' USING PigStorage(',')
```

AS (year:chararray, temperature:int, categorie:int);

- Accès aux membres du bag: $R.\$0$ et $R.\$2$ désignent respectivement $R.year$ et $R.categorie$

Exemple 2: chargement du fichier Notes.csv



```
R = LOAD '/user/cloudera/pig_lab/input/Notes.csv' USING PigStorage('|')
```

AS (infos:tuple(nom, prenom), notes:tuple(n1:float,n2:float,n3:float));

- Accès aux membres du bag: $R.infos.prenom$, $R.notes.n1$, ...

Pig: Affichage/ Sauvegarde

- Dans le shell interactif, on peut avoir une description d'un bag avec la commande **DESCRIBE**

Exemple : **R = LOAD '/user/cloudera/pig_lab/input/Notes.csv' USING PigStorage(',')
AS (infos:tuple(nom, prenom), notes:tuple(n1:float,n2:float,n3:float));
DESCRIBE R;**

On obtient sur écran:

R: {infos: (nom: bytearray,prenom: bytearray), notes: (n1: float,n2: float,n3: float)}

- On affiche les résultats (*données d'un bag ou container*) avec **DUMP**, par exemple: **DUMP R;**
- La commande **STORE** sauvegarde les données sur le système de fichier (HDFS ou local selon le mode d'utilisation de Pig): **STORE alias INTO 'repertoire' [USING fonction];**

Exemple:

**R = LOAD '/user/cloudera/pig_lab/input/temp.csv' USING PigStorage(',')
AS (year:chararray, temperature:int, categorie:int);**

STORE R INTO '/user/cloudera/pig_lab/input2' USING PigStorage(',');

STORE R INTO '/user/cloudera/pig_lab/input2' USING JsonStorage();

2011, 41, 3
2011, 20, 3
2012, 31, 3
2012, 34, 3

```
{"year":"2011","temperature":41,"categorie":3}  
{"year":"2011","temperature":20,"categorie":3}  
{"year":"2012","temperature":31,"categorie":3}  
{"year":"2012","temperature":34,"categorie":3}
```

Pig: Traitement de données [Filter , Order By]

- L'opérateur **FILTER** permet de filtrer les éléments d'un bag selon une condition:

DEST = FILTER source BY expression;

Exemple: F = **FILTER R BY** temperature > 15;

F = **FILTER R BY** temperature > 15 AND categorie IN (1, 3, 5);

F = **FILTER R BY** year MATCHES '2015';

F = **FILTER R BY** year MATCHES '*5';

R = **FILTER M BY** nom MATCHES '*Amr*.'

- L'opérateur **ORDER** trie les éléments selon une colonne ou plusieurs:

DEST = ORDER SOURCE BY champs [ASC|DESC];

Exemple: O = **ORDER R BY** temperature ASC;

O = **ORDER R BY** year DEC, temperature ASC;

Pig: Traitement de données [Group By]

- L'opérateur **GROUP** permet de grouper les tuples selon un champ:

dest = GROUP source BY champ;

- Il génère des tuples constitués de **deux champs**:

- Le premier (**group**), est un champ simple dont la valeur est celle du champ utilisé dans l'appel de **GROUP**

- Le second portant le **nom** de l'alias sur lequel on a appliqué **GROUP** est un **bag** qui contient tous les tuples pour lesquels la valeur de groupement a été rencontrée

Exemple: R = **LOAD** '/user/cloudera/pig_lab/input/temp.csv' **USING** PigStorage(',')
AS (year:chararray , temperature:int , categorie:int);

G = **GROUP** R **BY** categorie; -- *groupement par un champ*

DESCRIBE G;

On obtient: G: {group:int , R:{(year:chararray,temperature:int,categorie:int)}}}

G1 = **GROUP** R **BY** (year , categorie); -- *groupement par deux champs*

DESCRIBE G1;

On obtient: G1: {group: (chararray, int), R:{(year:chararray,temperature:int,categorie:int)}}}

(2011, 41, 3)
(2011, 20, 3)
(2012, 31, 3)
(2012, 34, 3)
(2012, 35, 3)

Pig: Traitement de données [Foreach...Generate]

- L'opérateur **FOREACH ... GENERATE** permet de parcourir les tuples d'un bag et de générer d'autres tuples: *dest = FOREACH source GENERATE expression;*
- Dans *expression* on peut générer des tuples, des map et/ou des bag et appeler des fonctions à la volée.
- Exemple:

```
R = LOAD '/user/cloudera/pig_lab/input/temp.csv' USING PigStorage(',')  
      AS (year:chararray, temperature:int, categorie:int);
```

```
FR = FILTER R BY temperature != 999 AND categorie IN (1, 4, 9);
```

```
GR = GROUP FR BY year;
```

```
MaxT= FOREACH GR GENERATE group, MAX(FR.temperature);
```

```
DUMP MaxT;
```

\$0

Pig: Traitement de données [Foreach...Generate]

L'opérateur **FOREACH** peut être utilisé pour plusieurs besoins:

1. Extraire certains champs de chaque tuple d'un bag ou les réorganiser.

Exemple: Extraction des champs year et categorie

```
grunt> R = LOAD '/user/cloudera/pig_lab/input/temp.csv' USING PigStorage(';')  
          AS (year:chararray, temperature:int, categorie:int);
```

```
grunt> R1 = FOREACH R GENERATE year , categorie; -- sélection de year et categorie
```

```
grunt> R2 = DISTINCT R1; -- R2 contiendra les tuples de R1 sans doublons
```

```
grunt> R3 = ORDER R2 BY year , categorie ;
```

```
grunt> DUMP R3;
```

Exemple: Réorganisation des champs

```
grunt> R = LOAD '/user/cloudera/pig_lab/input/temp.csv' USING PigStorage(';')  
          AS (year:chararray , temperature:int , categorie:int);
```

```
grunt> R1 = FOREACH R GENERATE year , (temperature, categorie) AS details;
```

```
grunt> DESCRIBE R1;
```

R1: {year: chararray , details: (temperature: int,categorie: int)}

Pig: Traitement de données [Fonctions de calcul]

2. Faire des **calculs** sur les champs de chaque tuple d'un bag via des opérateurs arithmétiques et des fonctions prédéfinies (**MAX**, **MIN**, **AVG**, **COUNT**, ...)

Exemple: Calcul arithmétique

```
grunt> R = LOAD '/user/cloudera/pig_lab/input/temp.csv' USING PigStorage(',')  
          AS (year:chararray, temperature:int, categorie:int);
```

```
grunt> R1 = FOREACH R GENERATE year , categorie * temperature AS produit;
```

```
grunt> DESCRIBE R1;  
R1: {year: chararray, produit : int}
```

Exemple: La moyenne de température par année et catégorie.

```
grunt> R = LOAD '/user/cloudera/pig_lab/input/temp.csv' USING PigStorage(',')  
          AS (year:chararray, temperature:int, categorie:int);
```

```
grunt> FR = FILTER R BY temperature != 999;
```

```
grunt> GR = GROUP FR BY (year, categorie);
```

```
GR: {group: (year: chararray, categorie: int), FR: {(year: chararray, temperature: int, categorie: int)}}
```

```
grunt> MoyT = FOREACH GR GENERATE group AS year_cat, AVG (FR.temperature) AS moyenne;
```

```
MoyT: {year_cat: (year: chararray, categorie: int), moyenne: double}
```

Pig: Traitement de données [Groupement d'opérations]

- L'opérateur **FOREACH** permet d'effectuer plusieurs traitement sur le tuple courant avant de générer un tuple en sortie: $dest = \text{FOREACH} source \{ opération_1;$

...

$opération_n;$

GENERATE *expression*;

}

- Exemple:**

```
R = LOAD '/user/cloudera/tp2/input/temp.csv'  
      USING PigStorage(',')  
      AS (year : int, temperature: int, categorie: int)
```

```
G = GROUP R BY year;
```

```
DESCRIBE G ;
```

G: {group: int, R: {(year : int, temperature: int, categorie: int)}}}

```
F = FOREACH G {  
    LC1 = FOREACH R GENERATE categorie;  
    LC2 = DISTINCT LC1;  
    GENERATE group, LC2;  
};
```

```
DUMP F;
```

G:
(2011, {(2011, 23, 3),(2011,17,9),(2011,20,3)})
(2013, {(2013, 23, 9),(2013,17,9)})

F:
(2011, {(3),(9)})
(2013, {(9)})

Pig: Traitement de données [Join]

- L'opérateur **JOIN** permet de faire des jointure entre deux bags:

bag3 = JOIN bag1 BY champ1 , bag2 BY champ2;

bag3 contiendra des tuples dont chacun est composé d'un tuple de *bag1* et d'un autre de *bag2* si les deux ont la même valeur de *champ1* et *champ2*.

- Jointure sur plusieurs champs : *bag3 = JOIN bag1 BY (ch11, ch12) , bag2 BY (ch21, ch22) ;*
- Exemple: temp.csv (year, temperature, **categorie**)

region.csv (**categorie**, longitude, latitude, rayon)

```
grunt> R1 = LOAD '/user/cloudera/pig_lab/input/temp.csv' USING PigStorage(';')
          AS (year:chararray, temperature:int, categorie:int);
```

```
grunt> R2 = LOAD '/user/cloudera/pig_lab/input/region.csv' USING PigStorage(';')
          AS (categorie:int, longitude:int, latitude:int, rayon:int);
```

```
grunt> R3 = JOIN R1 BY categorie, R2 BY categorie;
```

```
R3: {R1::year: chararray, R1::temperature: int, R1::categorie: int, R2::categorie, R2::longitude: int,
      R2::latitude: int, R2::rayon:int}
```

```
grunt> R4 = FOREACH R3 GENERATE R1:: year, R1::temperature, R2::longitude, R2::latitude;
```

Pig: Traitement de données [Cross]

- L'opérateur **CROSS** permet de faire un produit cartésien entre deux bags:

bag3 = CROSS bag1 , bag2;

bag3 contiendra des tuples dont chacun est composé d'un tuple de *bag1* et d'un autre de *bag2*

- Exemple: `users.csv (id, nom, nbquestion, nbréponse)`

```
grunt> R1 = LOAD '/user/cloudera/pig_lab/input/users.csv' USING PigStorage(',')  
      AS (id:long, nom:chararray, nbq:int , nbr:int);
```

```
grunt> R2 = LOAD '/user/cloudera/pig_lab/input/users.csv' USING PigStorage(',')  
      AS (id:long, nom:chararray, nbq:int , nbr:int);
```

```
grunt> R3= CROSS R1, R2 ;
```

```
R3: { R1::id: int,R1::nom: chararray, R1::nbq: int, R1::nbr: int, R2::id: int,R2::nom: chararray, R2::nbq: int, R2::nbr: int }
```

```
grunt> R4 = FILTER R3 BY R1::nbq == R2::nbq; Paires d'utilisateurs ayant posé le même nombre  
      de questions
```

```
grunt> R5 = FOREACH R4 GENERATE R1::nom As nom1, R2::nom As nom2, R1::nbq As nbq;
```

Pig: Utilisation de Pig en ligne de commandes

1. Lancer la commande pig: ...]\$ **pig**
2. On obtient le shell pig: **grunt>**
3. Charger un fichier:

```
grunt> R = LOAD '/user/cloudera/pig_lab/input/temp.csv' USING
          PigStorage(';) AS (year:chararray, temperature:int, categorie:int);
```

R est un bag (son contenu est temporaire) sur lequel on peut appliquer des commandes (describe, dump, ...)

4. Appliquer des commandes à l'alias (bag) obtenu par **Load**:

```
grunt> DESCRIBE R
```

R: {year: chararray,temperature: int,categorie: int}

```
grunt> DUMP R;
```

(2011,17,4)

(2011,14,4)

(2011,29,4)

(2011,17,4)

(2012,25,3)

.....

Pig: Utilisation de Pig en ligne de commandes

5. Filtrer des tuples à partir du résultat de Load:

```
grunt> FR = FILTER R BY temperature != 999 AND categorie IN (1, 4, 9);
```

```
grunt> DUMP FR;
```

....

(2014,7,9)

....

6. Faire un groupement de tuples:

```
grunt> GR = GROUP FR BY year;
```

```
grunt> describe GR;
```

GR: {group: chararray, **FR** : {year: chararray, temperature: int, quality: int}}

```
grunt> DUMP GR;
```

(2011, {(2011,32,1),(2011,251),...})

.....

Travaux Pratiques

