

Frameworks de stockage et de traitement distribués

Étude de cas avec Apache Hadoop

Khaled Jouini

j.khaled@gmail.com

Institut Supérieur d'Informatique et des Technologies de Communication

La reproduction de ce document par tout moyen que ce soit, sans l'avis de l'auteur, est interdite conformément aux lois protégeant la propriété intellectuelle

Ce document peut comporter des erreurs, à utiliser donc en connaissance de cause!

Plan du cours

- 1 Apache Hadoop, quésaco?
- 2 Systèmes de Fichiers Distribués HDFS
 - Architecture Master/Slaves
 - Journalisation et failover
 - Haute disponibilité
 - Fédération
 - Formats de fichiers
 - Commandes Hadoop
 - Google File System (GFS)
 - Quiz
 - Exercice
- 3 Traitement distribué avec Map/Reduce et YARN
 - Map/Reduce, quésaco?
 - Session Illustrative
 - Limites de Map/Reduce
 - YARN, quésaco?
 - Architecture de YARN
 - Quiz

Pour aller plus loin...

- *Apache Hadoop YARN: Yet Another Resource Negotiator.*
Kumar V. V. et al.
4th Annual Symposium on Cloud Computing (SOCC). 2013.
- *The Google File System.*
Ghemawat S., Gobioff H. & Leung, S.T.
19th ACM Symposium on Operating Systems Principles (SOSP). 2003.
- *The Hadoop Distributed File System.*
Shvachko K., Kuang H., Radia S., & Chansler R.
26th IEEE Symposium on Mass Storage Systems and Technologies (MSST). 2010.
- *MapReduce: Simplified Data Processing on Large Clusters..*
Dean J. & Ghemawat S.
Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI). 2004.
- *Hadoop The Definitive Guide - Storage and Analysis at Internet Scale.*
Tom White
eBook, O'Reilly, ISBN: 978-1-491-90163-2. July 2015.
Disponible sur ce [Lien](#).
- *Documentation officielle d'Apache Hadoop*

Hadoop HDFS (Hadoop Distributed File System)

Hadoop : **Framework** open source pour le **stockage** et le **traitement** (MapReduce) **distribués**.

Écrit en Java/C¹, inspiré de Google FS et son MapReduce

Principaux supports/contributeurs : Yahoo, FaceBook, IBM, Hortonworks, etc.

Composantes clés

- **HDFS pour le stockage distribué**
- **MapReduce pour le traitement distribué**

Écosystème

- HBASE : BD Distribuées
- ZooKeeper : Orchestrateur, Configuration/synchronisation
- Pig/Hive : Langages de haut niveau
- YARN (Hadoop 2.0) : Traitements autres que MapReduce
- Sqoop (chargement de données relationnelles), Oozie, Storm (flux de données), etc.

¹Mais utilisable avec d'autres langages

Chapitre 2 - Systèmes de Fichiers Distribués HDFS

- Architecture Master/Slaves
- Journalisation et failover
- Haute disponibilité
- Fédération
- Formats de fichiers
- Commandes Hadoop
- Google File System (GFS)
- Quiz
- Exercice

Système de fichiers

Fonctionnalités d'un système de fichiers, entre autres :

- **Manipulation des fichiers** : des opérations sont définies pour permettre la manipulation des fichiers par les programmes d'application, à savoir : créer/détruire des fichiers, insérer, supprimer et modifier un enregistrement dans un fichier.
- **Allocation de la place sur mémoires secondaires** : les fichiers étant de taille différente et cette taille pouvant être dynamique, le SGF alloue à chaque fichier un nombre variable de granules de mémoire secondaire de taille fixe (blocs).
- **Localisation des fichiers** : il est nécessaire de pouvoir identifier et retrouver les données ; pour cela, chaque fichier possède un ensemble d'informations descriptives (nom, adresse. . .) regroupées dans un inode.
- etc.

Principe du Système de Fichiers Distribué Hadoop (HDFS)

Objectif de HDFS :

- **HDFS** est conçu pour stocker et gérer des données massives de manière distribuée.
- Les fichiers sont **divisés en blocs** (également appelés *splits* ou *chunks*). Les blocs d'un même fichier sont typiquement stockés sur des serveurs différents.
- Chaque bloc est répliqué (**3 fois par défaut**) pour assurer la tolérance aux pannes.
- HDFS permet de **créer, supprimer et copier** des fichiers, mais il ne permet pas de les modifier directement.
- Conçu pour des **lectures séquentielles**, et non pour des accès aléatoires.
- Le concept de **localité des données** est important : les traitements sont effectués sur ou près du stockage physique afin de **réduire la transmission des données**.

Namenode et DataNodes

HDFS : repose sur 2 types de noeuds, les NameNodes et les DataNodes

DataNode (serveur de données) : **stocke** et **restitue** les **blocs de données** (les chunks)

NameNode (serveur de métadonnées)

- **Stocke le répertoire**, l'espace des noms, l'arborescence et les métadonnées des fichiers.
- **Centralise la localisation des blocs** dans le cluster HDFS
- Lors de la lecture des blocs d'un fichier, le NameNode est interrogé. Il renvoie pour chaque bloc, l'adresse du DataNode le plus accessible (qui a la plus grande bande passante)
- Les DataNodes envoient périodiquement au NameNode la liste des blocs que chacun héberge.
- Si le NameNode constate qu'un bloc n'est pas suffisamment répliqué, il **initie une réplication sur d'autres DataNodes**

Hadoop HDFS (Hadoop Distributed File System)

HDFS architecture

- Master/Slave architecture
- Master: **NameNode**
 - manages the file system namespace and metadata
 - FsImage
 - Edits Log
 - regulates client access to files
- Slave: **DataNode**
 - many per cluster
 - manages storage attached to the nodes
 - periodically reports status to NameNode

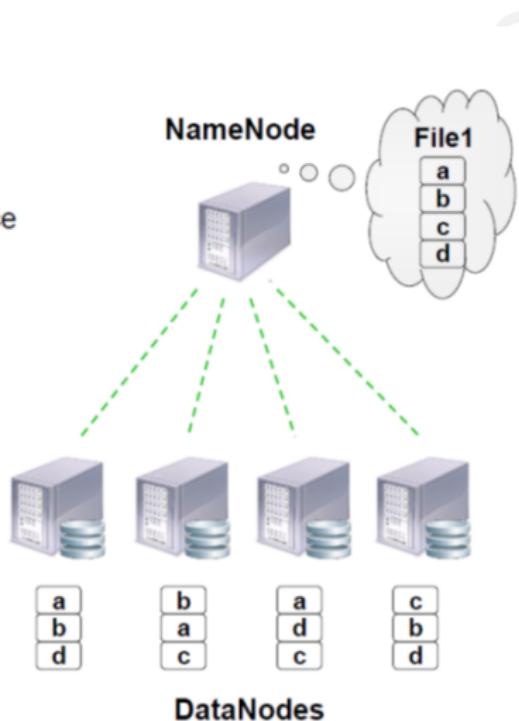


Figure: Figure reprise de "IBM Skills Academy: Big Data Engineer - Student Guide - 2019"

Chapitre 2 - Systèmes de Fichiers Distribués HDFS

- Architecture Master/Slaves
- **Journalisation et failover**
- Haute disponibilité
- Fédération
- Formats de fichiers
- Commandes Hadoop
- Google File System (GFS)
- Quiz
- Exercice

File System Image (fsImage)

- Le NameNode capture périodiquement des **snapshots (fsImage)** des métadonnées et enregistre les modifications intermédiaires dans un fichier edit Log (e.g. renommage d'un fichier, migration d'un chunk, etc.).
- fsImage**
 - Le **fsImage** est un fichier qui contient une capture instantanée (snapshot) de toutes les métadonnées du système de fichiers HDFS.
 - Il représente l'état du système de fichiers à un moment donné : structure des répertoires, noms de fichiers, permissions, blocs et répliques associés.
 - Chargé en mémoire au démarrage pour restaurer l'état du système.

Fichier de journalisation Edits Log

- **editsLog**

- Le **editsLog** est un fichier journal dans lequel toutes les modifications apportées au système de fichiers (ajout, suppression, renommage de fichiers, etc.) sont enregistrées de manière séquentielle.
- Chaque modification est enregistrée dans le editsLog après avoir été appliquée, ce qui permet de reconstruire l'état actuel du système en combinant le fsImage et le editsLog.
- Le editsLog est essentiel pour assurer la persistance des modifications entre deux captures du fsImage.

Failover et Démarrage du NameNode

Processus de démarrage du NameNode

- Lors du démarrage, le **NameNode** charge en mémoire l'image du système de fichiers depuis le fichier **fslimage**.
- Ensuite, le NameNode lit le **editsLog** pour appliquer toutes les modifications survenues après la capture du dernier fslImage.
- Le NameNode combine ces deux fichiers pour restaurer l'état exact et actuel du système de fichiers HDFS.
- Après cela, le NameNode écrit une nouvelle version consolidée du fslimage et vide le editsLog pour préparer le prochain cycle de modifications.
- Le NameNode commence ensuite à accepter des requêtes des clients (création, suppression, modification de fichiers) et à gérer les communications avec les **DataNodes**.

Failover et Démarrage du NameNode

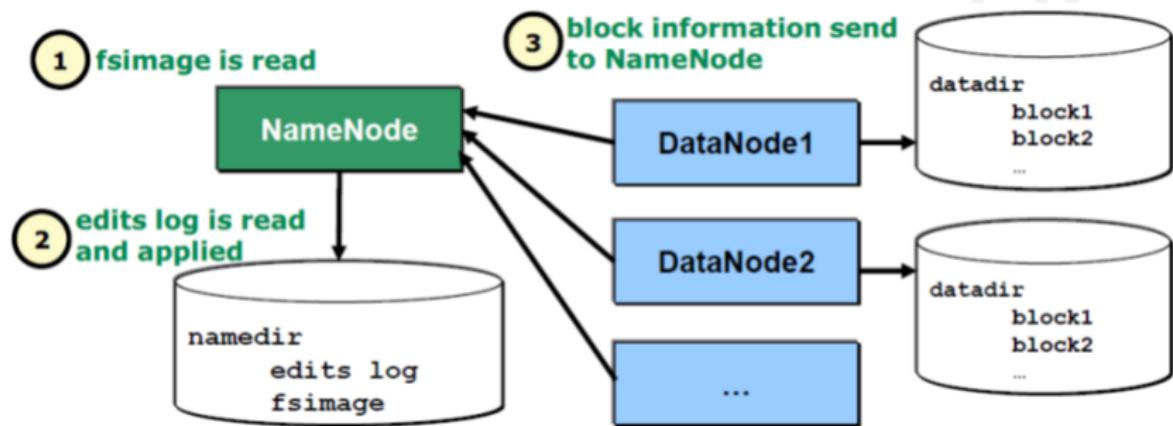


Figure: reprise de "IBM Skills Academy: Big Data Engineer - Student Guide - 2019"

Chapitre 2 - Systèmes de Fichiers Distribués HDFS

- Architecture Master/Slaves
- Journalisation et failover
- **Haute disponibilité**
- Fédération
- Formats de fichiers
- Commandes Hadoop
- Google File System (GFS)
- Quiz
- Exercice

Fonctionnement du Secondary dans Hadoop v1

Le Secondary NameNode dans Hadoop v1

- Dans **Hadoop v1**, le NameNode est un point de défaillance unique (*Single Point of Failure - SPOF*), car il n'existe qu'un seul NameNode actif à la fois.
- Un **Secondaire NameNode** est présent, mais il ne s'agit pas d'un mécanisme de basculement (*failover*) automatique.
- Le rôle du Secondaire NameNode est de périodiquement fusionner le **editsLog** et le **fslimage** pour générer une nouvelle image propre du système de fichiers.
- Le Secondaire NameNode ne prend pas le relais en cas de panne du NameNode. Il sert uniquement à minimiser le temps de récupération en préparant un fslimage à jour.
- En cas de défaillance du NameNode, l'administrateur doit manuellement restaurer un autre NameNode à partir des métadonnées du fslimage et du editsLog sauvegardés.
- Il n'y a **pas de haute disponibilité** dans Hadoop v1, car aucune redondance automatique du NameNode n'est prévue.

Fonctionnement du Secondary dans Hadoop v1

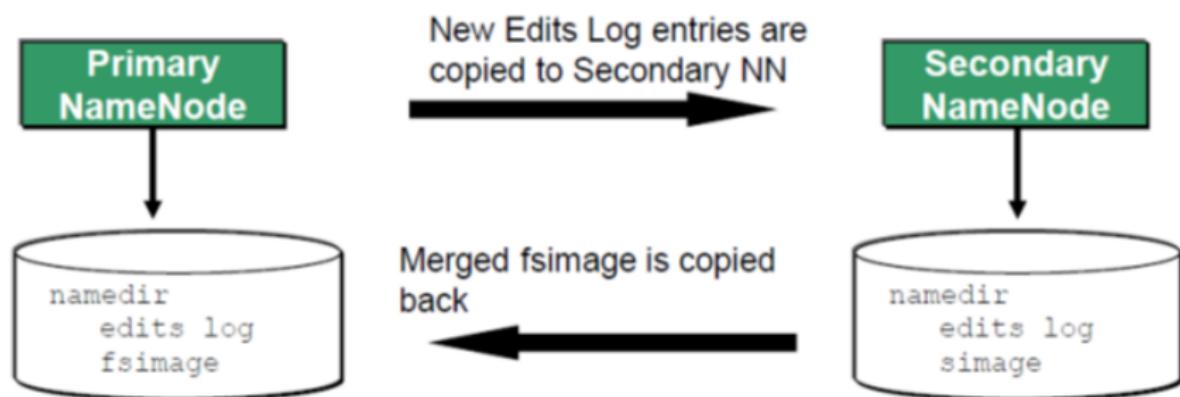


Figure: Figure reprise de "IBM Skills Academy: Big Data Engineer - Student Guide - 2019"

Haute Disponibilité (HA) dans Hadoop v2

- Introduction de la **Haute Disponibilité (HA)** avec des **NameNodes redondants**.
- Utilisation d'un **NameNode actif** et d'un **NameNode en mode Standby**.
- En cas de panne du NameNode actif, le NameNode en mode Standby peut automatiquement prendre le relais.

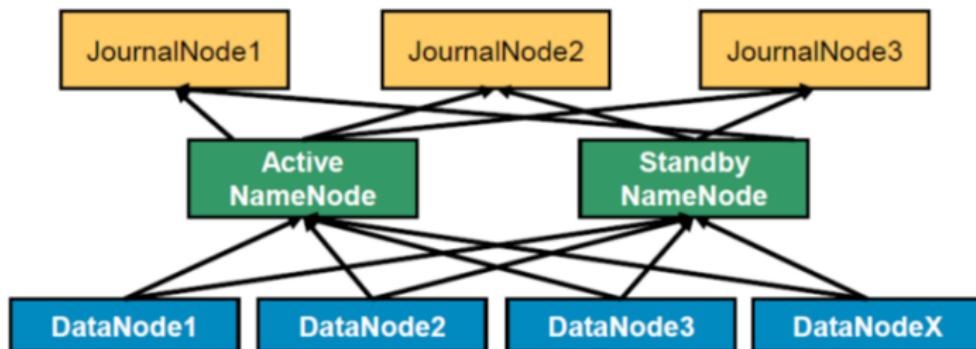


Figure reprise de "IBM Skills Academy: Big Data Engineer - Student Guide - 2019"

Architecture de la Haute Disponibilité dans Hadoop v2

Composants principaux

- **NameNode Actif** : Gère les opérations régulières de HDFS (lecture, écriture, gestion des métadonnées).
- **NameNode en mode Standby** : Reste synchronisé avec le NameNode actif en recevant les modifications des métadonnées via les **JournalNodes**.
- **JournalNodes** : Un ensemble de nœuds qui stockent les modifications du système de fichiers dans des journaux partagés entre le NameNode actif et le Standby.
- **ZooKeeper** : Utilisé pour coordonner le basculement automatique entre le NameNode actif et le Standby en cas de panne.

Fonctionnement du Basculement en HA dans Hadoop v2

Synchronisation des NameNodes

- Les **modifications des métadonnées** sont écrites dans le **JournalNode** par le NameNode actif.
- Le NameNode en mode Standby lit ces journaux pour se maintenir à jour avec les modifications.
- Ce mécanisme garantit que le Standby peut prendre le relais immédiatement si nécessaire.

Basculement automatique :

- Si le NameNode actif devient indisponible, le processus de basculement est déclenché par **ZooKeeper**.
- Le NameNode Standby devient **actif** et commence à gérer les opérations de HDFS.
- Ce basculement se fait sans interruption significative du service, garantissant la **continuité des opérations**.

Chapitre 2 - Systèmes de Fichiers Distribués HDFS

- Architecture Master/Slaves
- Journalisation et failover
- Haute disponibilité
- **Fédération**
- Formats de fichiers
- Commandes Hadoop
- Google File System (GFS)
- Quiz
- Exercice

Fédération dans Hadoop HDFS

Problème dans Hadoop traditionnel :

- Dans les versions antérieures de Hadoop, il y avait un seul **NameNode** pour gérer tout l'espace de noms HDFS.
- Ce NameNode unique devenait un goulet d'étranglement en termes de **scalabilité** et de **performance**.
- La capacité de gestion du cluster était limitée par la quantité de mémoire disponible pour ce NameNode unique.

Solution : Hadoop a introduit la **fédération HDFS** pour permettre une gestion plus évolutive et efficace des espaces de noms.

Architecture de la fédération dans Hadoop HDFS

Principes clés de la fédération

- **Multiples NameNodes** : Chaque NameNode gère une partie de l'espace de noms de manière indépendante.
- Chaque NameNode est responsable de son propre **namespace** et des métadonnées associées (fichiers, répertoires).
- Les **DataNodes** sont partagés entre tous les NameNodes et stockent les blocs de données de manière distribuée.
- Les DataNodes peuvent servir plusieurs espaces de noms simultanément, ce qui permet une meilleure utilisation des ressources.

Avantages de la fédération dans Hadoop HDFS

Scalabilité améliorée

- La fédération permet de **scaler horizontalement** en ajoutant plus de NameNodes au fur et à mesure que la taille du cluster augmente.
- Les applications peuvent s'exécuter sur plusieurs NameNodes simultanément, optimisant ainsi l'utilisation des ressources.

Isolation des espaces de noms

- Chaque espace de noms est isolé, ce qui permet de dédier un espace de noms spécifique à une application ou à une équipe.
- Cela facilite également la gestion des permissions et des accès aux données.

Fiabilité améliorée :

- En cas de défaillance d'un NameNode, les autres espaces de noms restent disponibles, augmentant ainsi la **disponibilité globale**.
- Les DataNodes continuent de servir les autres NameNodes même si l'un d'entre eux devient indisponible.

Chapitre 2 - Systèmes de Fichiers Distribués HDFS

- Architecture Master/Slaves
- Journalisation et failover
- Haute disponibilité
- Fédération
- **Formats de fichiers**
 - Parquet : Format de stockage orienté colonne
 - Organisation interne d'un fichier Parquet
 - Encodage RLE et Dictionary dans parquet
 - Compression
 - Parquet : Compatibilité et Comparaison
- Commandes Hadoop
- Google File System (GFS)
- Quiz
- Exercice

Formats de fichiers supportés par HDFS (1/2)

- **Fichiers texte usuels** (y compris csv, tsv, etc.) :

- Formats classiques pour stocker des données tabulaires sous forme de texte.
- Faciles à lire, mais peu efficaces pour les traitements distribués à grande échelle, surtout pour les requêtes d'agrégation.

- **Sequence Files** :

- Fichiers binaires, optimisés pour les tâches MapReduce.
- Chaque enregistrement comporte une paire **clé-valeur**, facilitant le traitement distribué.

- **Avro** :

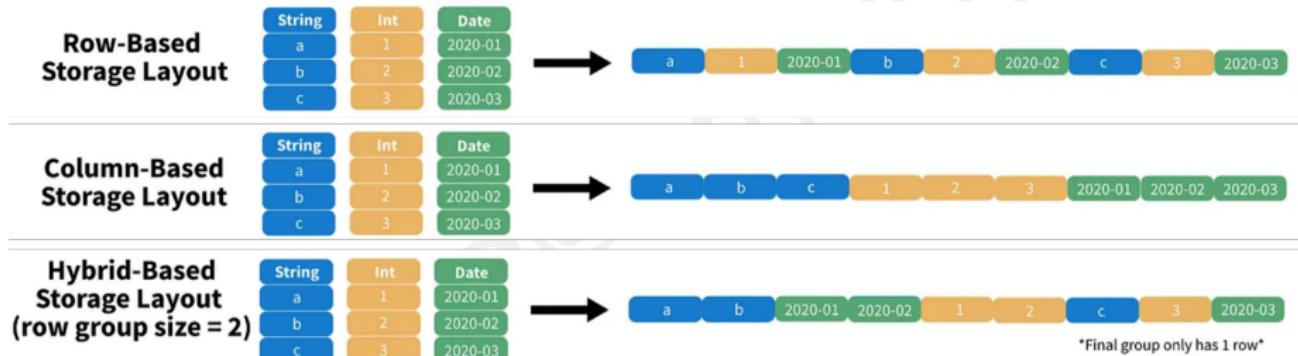
- framework de sérialisation / désérialisation.
- Les données sont décrites via un schéma JSON, permettant une compatibilité évolutive (lecture/écriture de versions différentes de données).
- Prend en charge à la fois un format JSON et un format binaire plus compact pour un stockage et un transfert de données plus efficaces.

Formats de fichiers supportés par HDFS (2/2)

Parquet et RCFile : Formats de stockage orienté colonne

- Formats sont optimisés pour les **charges de travail analytiques** et le traitement de grandes quantités de données.
- **Parquet** en particulier est un format de fichier orienté colonne largement utilisé dans les environnements Big Data pour son efficacité en termes de stockage et de traitement.
- **Stockage orienté colonne** :
 - Contrairement au format orienté ligne (comme CSV), où toutes les données d'un enregistrement sont stockées ensemble, dans un format orienté colonne, les valeurs de chaque colonne sont stockées ensemble.
 - Cette approche est particulièrement efficace pour les **requêtes d'agrégation**, comme les calculs de moyennes, sommes ou autres sur une ou plusieurs colonnes, car seules les colonnes nécessaires sont lues.
 - Parquet compresse également efficacement les données, réduisant l'empreinte de stockage.

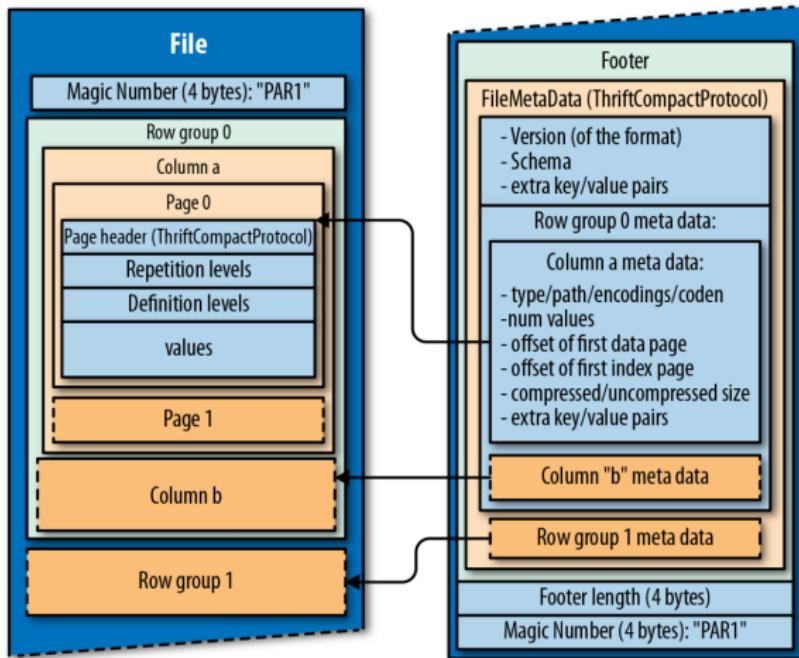
Parquet : Format de stockage orienté colonne



Parquet : Format de stockage orienté colonne

- **Structure** : Organisé en blocs de données contenant des pages, avec un schéma intégré qui décrit la structure des données.
- **Compression efficace** : Parquet prend en charge plusieurs algorithmes de compression (**Snappy**, **GZIP**, **LZO**) qui permettent de réduire la taille des fichiers et d'améliorer les performances de stockage.
- **Optimisation des requêtes** : Grâce à son organisation en colonnes, seules les colonnes nécessaires sont lues, ce qui est particulièrement avantageux pour les requêtes d'agrégation.
- **Pratiques d'optimisation** :
 - **Partitionnement des données** pour améliorer la vitesse de lecture.
 - **Pruning de colonnes** : Lire uniquement les colonnes nécessaires.

Organisation interne d'un fichier Parquet



Organisation interne d'un fichier Parquet - Partie 1

Structure d'un fichier Parquet : Un fichier Parquet est constitué de plusieurs composants organisés de manière hiérarchique pour optimiser les performances de stockage et de lecture.

- **Header** : Contient des informations de base sur le fichier, comme les métadonnées globales et le schéma des données.
- **Row Groups** : Unité de base pour la gestion des données dans Parquet. Un fichier Parquet peut contenir plusieurs row groups.
 - Chaque row group est divisé en colonnes.
 - Contient des informations statistiques sur les colonnes, ce qui facilite le filtrage lors des requêtes.

Organisation interne d'un fichier Parquet - Partie 2

- **Pages** : Les row groups sont subdivisés en pages, qui sont les unités de compression.
 - Les pages contiennent les valeurs d'une colonne pour un sous-ensemble de lignes dans un row group.
 - Les pages peuvent être de différents types : pages de données, pages de dictionnaire, pages de index, etc.
- **Footer** : Contient les métadonnées détaillées, comme les informations sur les row groups, les statistiques et le schéma. C'est la dernière partie du fichier.

Encodage RLE dans Parquet

Run-Length Encoding (RLE) : RLE encode les valeurs répétées consécutives sous forme d'une paire "valeur + longueur".

- **Principe** : RLE remplace les séquences consécutives de valeurs identiques par une seule valeur et la longueur de la séquence.

- **Exemple** :

- Données d'origine :

[Vrai, Vrai, Vrai, Faux, Faux, Vrai, Vrai, Vrai, Vrai, Faux]

- Encodage RLE :

[(Vrai, 3), (Faux, 2), (Vrai, 4), (Faux, 1)]

- **Efficacité** : Idéal pour les colonnes avec des données répétitives (ex : colonnes booléennes ou catégorielles).
- **Avantages** : Réduit la taille des fichiers en encodant les séquences répétitives de manière compacte.

Encodage Dictionary dans Parquet

Dictionary Encoding : Parquet crée un dictionnaire des valeurs uniques dans une colonne, puis encode les valeurs avec des indices dans ce dictionnaire.

- **Principe** : Chaque valeur unique est associée à un indice dans un dictionnaire, et les données sont ensuite encodées avec ces indices.

- **Exemple** :

- Données d'origine :

```
["Pomme", "Pomme", "Banane", "Pomme", "Banane", "Cerise", "Pomme"]
```

- Dictionnaire :

```
{"Pomme" : 0, "Banane" : 1, "Cerise" : 2}
```

- Encodage Dictionary :

```
[0, 0, 1, 0, 1, 2, 0]
```

- **Efficacité** : Idéal pour les colonnes avec peu de valeurs distinctes (faible cardinalité).
- **Avantages** : Réduit la taille des données en remplaçant les valeurs textuelles par des indices compacts.

Méthodes de compression pour Parquet

Parquet prend en charge plusieurs algorithmes de compression qui optimisent le stockage et réduisent la taille des fichiers. Quelques exemples courants incluent

- **Snappy**

- Compression rapide avec un bon compromis entre vitesse et taille réduite.
- Idéale pour les scénarios où la vitesse d'écriture et de lecture est primordiale (utilisé par défaut dans Apache Spark).

- **GZIP**

- Compression plus élevée que Snappy, mais avec des performances de lecture/écriture plus lentes.
- Utilisée dans les scénarios où la taille des fichiers compressés est critique.

- **LZO**

- Compression rapide comme Snappy, avec de bonnes performances sur les grandes quantités de données.
- Moins efficace en termes de réduction de taille par rapport à GZIP.

Parquet : Compatibilité et Comparaison

Compatibilité avec Big Data : largement supporté par des outils Big Data comme **Apache Spark, Hive, Impala et Drill**.

- **Comparaison avec CSV et JSON :**

- **CSV** : Moins performant pour les requêtes analytiques, car orienté ligne.
- **JSON** : Flexible, mais moins efficace en termes de stockage et de performance.
- **Parquet** : Plus efficace pour les calculs analytiques grâce à la compression et la lecture sélective des colonnes.

Chapitre 2 - Systèmes de Fichiers Distribués HDFS

- Architecture Master/Slaves
- Journalisation et failover
- Haute disponibilité
- Fédération
- Formats de fichiers
- **Commandes Hadoop**
- Google File System (GFS)
- Quiz
- Exercice

Commandes Hadoop - Commandes POSIX et préfixe HDFS

Commandes POSIX dans Hadoop : Hadoop utilise des commandes qui fonctionnent de manière similaire aux commandes POSIX utilisées sur un système de fichiers local.

- **Exemples de commandes POSIX :** `mkdir`, `ls`, `rm`, `cp`, etc.

• Préfixe `hdfs dfs`

- Le préfixe `hdfs dfs` est nécessaire pour indiquer que la commande s'exécute sur HDFS plutôt que sur le système de fichiers local.
- Exemple : `hdfs dfs -ls /user/khaled`

Commandes Hadoop - Commandes avancées et opérations multiples

Les commandes Hadoop permettent d'effectuer des opérations avancées sur HDFS, telles que la gestion de répertoires, la collecte de statistiques sur les fichiers, et la manipulation de plusieurs fichiers à la fois.

• **Commandes statistiques**

- `hdfs dfs -du` : Affiche l'espace disque utilisé par un fichier ou répertoire.
- `hdfs dfs -ls -h` : Liste les fichiers avec la taille en format lisible (Go, Mo).

• **Opérations sur plusieurs fichiers**

- `hdfs dfs -getmerge` : Fusionner plusieurs fichiers locaux en un seul fichier.
- `hdfs dfs -put` : Copier plusieurs fichiers locaux dans HDFS.

• **Exemple pratique**

- Créez un répertoire : `hdfs dfs -mkdir /user/khaled/myfolder`
- Copiez un fichier volumineux : `hdfs dfs -copyFromLocal /path/to/largefile /user/khaled/data`

Commandes Hadoop - Gestion des permissions et utilisateurs HDFS

HDFS gère des permissions de fichiers similaires à celles d'un système de fichiers UNIX, mais avec des spécificités adaptées au contexte distribué de Hadoop.

• Types de permissions

- r : Permission de lecture (read).
- w : Permission d'écriture (write).
- x : Permission d'exécution (execute).

• Gestion des utilisateurs et des groupes

- HDFS permet de gérer les utilisateurs et les groupes de manière centralisée.
- Chaque fichier ou répertoire HDFS a un propriétaire et un groupe associé.

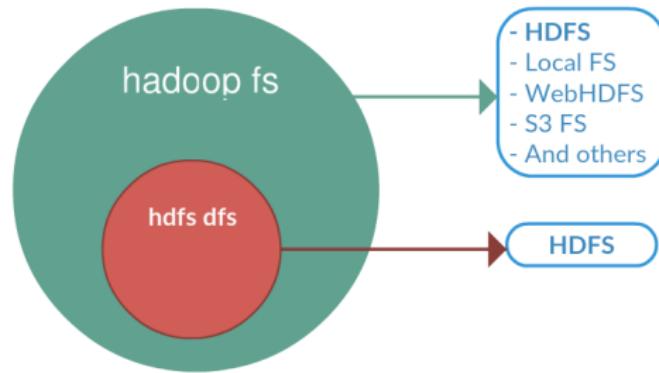
• Exemple de gestion des permissions

- Donner les permissions de lecture/écriture à un groupe : `hdfs dfs -chmod 770 /user/khaled/data`

Outils d'administration Hadoop

Apache Ambari : Un outil d'administration web pour gérer et superviser un cluster Hadoop.

- Fournit une interface graphique pour surveiller l'état du cluster, gérer les services, et vérifier les performances.
- Permet de configurer les services Hadoop comme HDFS, YARN, MapReduce, etc.
- Ambari inclut des tableaux de bord pour visualiser les métriques des services Hadoop, avec des alertes pour les problèmes de performance.

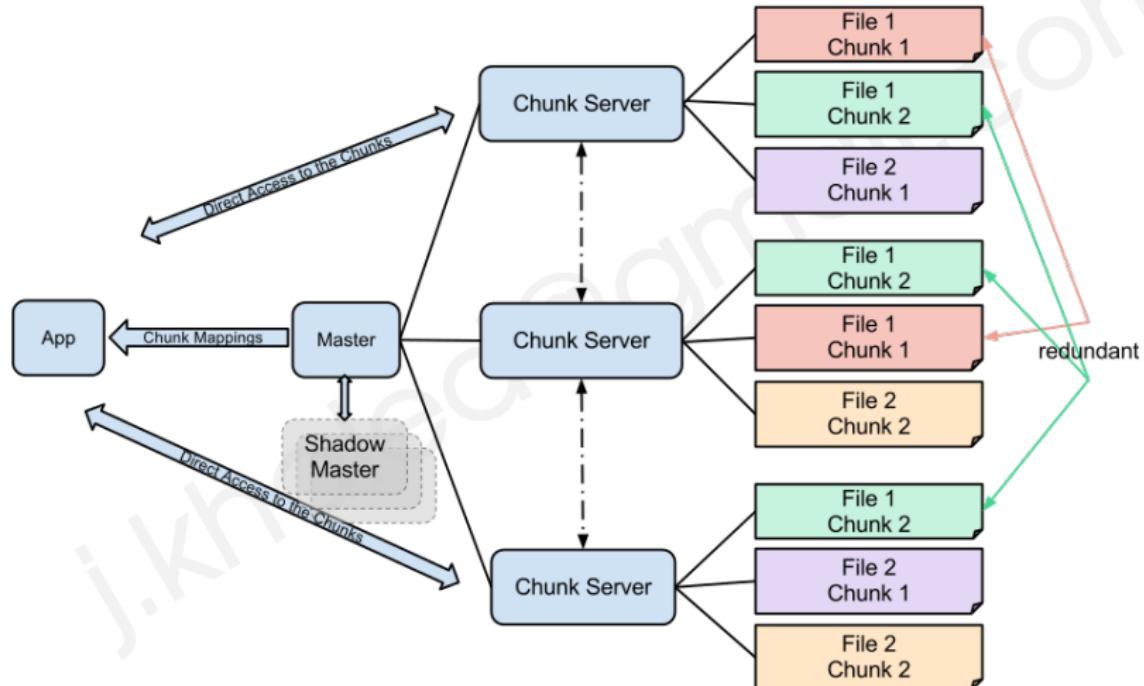


Chapitre 2 - Systèmes de Fichiers Distribués HDFS

- Architecture Master/Slaves
- Journalisation et failover
- Haute disponibilité
- Fédération
- Formats de fichiers
- Commandes Hadoop
- **Google File System (GFS)**
- Quiz
- Exercice

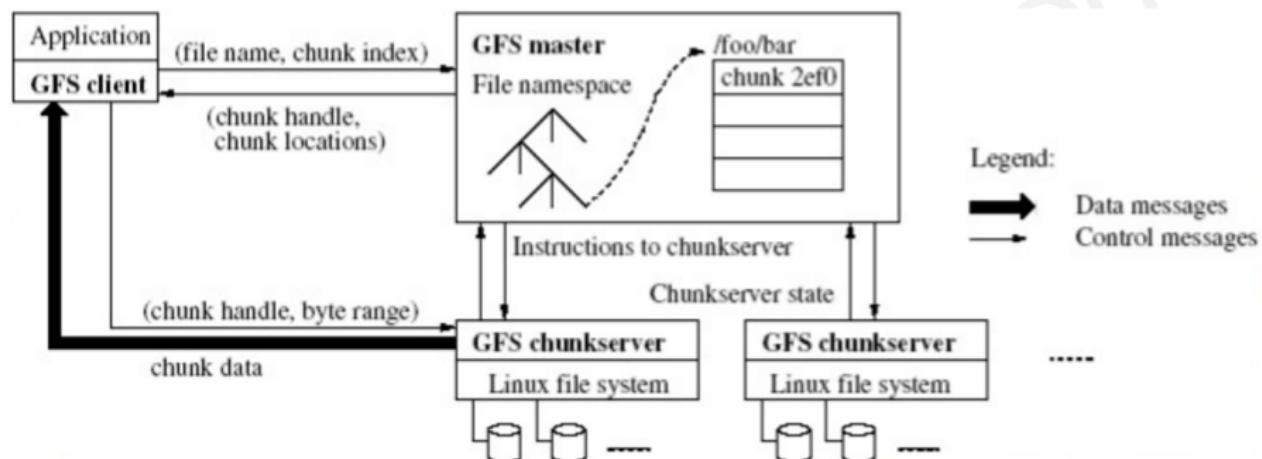
Le sharding de Google File System (GFS)

Architecture un peu différente, mais les grands principes sont là!



Google File System (GFS)

Architecture un peu différente, mais les grands principes sont là!



Chapitre 2 - Systèmes de Fichiers Distribués HDFS

- Architecture Master/Slaves
- Journalisation et failover
- Haute disponibilité
- Fédération
- Formats de fichiers
- Commandes Hadoop
- Google File System (GFS)
- Quiz
- Exercice

Quiz

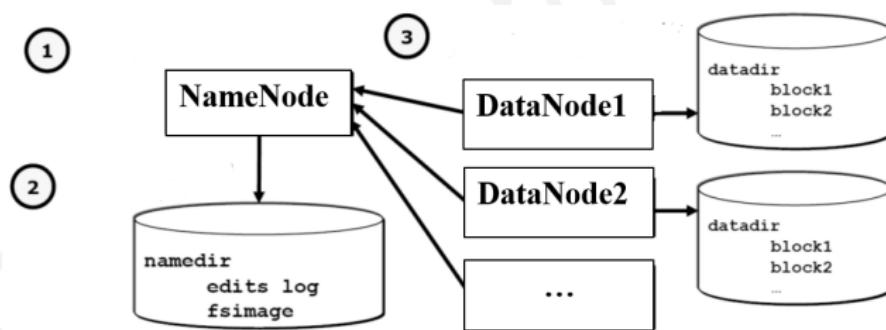
- 1 Expliquez en quoi Hadoop HDFS diffère d'un système de fichiers classique ?
- 2 Quel est le rôle de la localité des données dans les calculs distribués ?
- 3 Décrivez les étapes de démarrage du NameNode
- 4 Quelle est la différence entre le `fslimage` et le `editsLog` dans HDFS ? Quel est leur rôle respectif dans la gestion des métadonnées ?
- 5 Quelles sont les limitations du système de fichiers distribué de Hadoop v1 en termes de haute disponibilité et de basculement automatique ?
- 6 En quoi consiste la fédération HDFS introduite dans Hadoop v2 et comment améliore-t-elle la scalabilité et la gestion des clusters ?
- 7 Comparez les formats de stockage orientés colonne (comme Parquet) avec les formats orientés ligne (comme CSV) en termes d'efficacité pour le traitement Big Data. Pourquoi les formats orientés colonne sont-ils plus adaptés pour les analyses de type OLAP ?

Chapitre 2 - Systèmes de Fichiers Distribués HDFS

- Architecture Master/Slaves
- Journalisation et failover
- Haute disponibilité
- Fédération
- Formats de fichiers
- Commandes Hadoop
- Google File System (GFS)
- Quiz
- Exercice

Exercice

La figure ci-après illustre les **3 principales étapes de démarrage d'un cluster Hadoop HDFS**. Expliquez brièvement chacune de ces étapes.

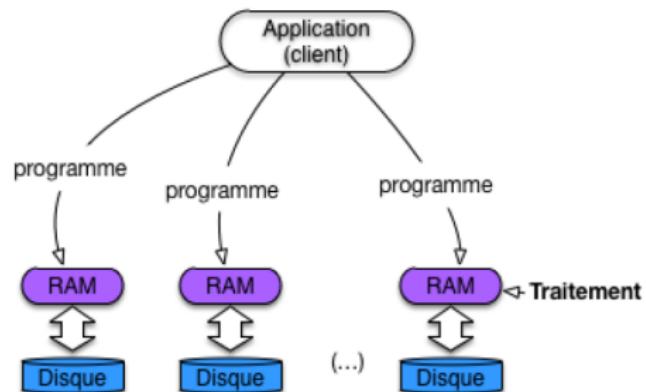
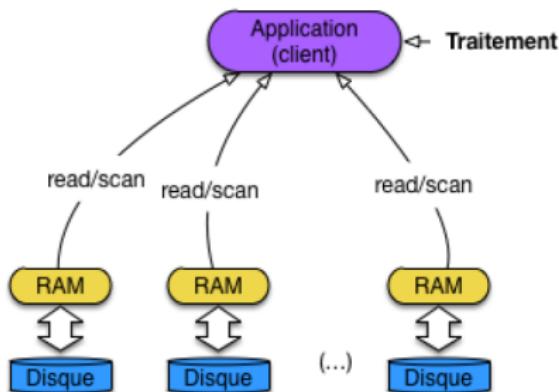


Chapitre 3 - Traitement distribué avec Map/Reduce et YARN

- Map/Reduce, quésaco?
- Session Illustrative
- Limites de Map/Reduce
- YARN, quésaco?
- Architecture de YARN
- Quiz

Localité des données

Client serveur pour traiter des TOs de données ? **Ne marche pas.**



Map/Reduce, quésaco?

- **Map/Reduce** : modèle de calcul **distribué** destiné à traiter efficacement de grands volumes de données sur plusieurs machines.
- Il repose sur deux opérations fondamentales :
 - **Map** : applique une transformation sur chaque donnée pour générer des paires clé-valeur.
 - **Reduce** : les paires clé-valeur intermédiaires sont regroupées par clé, puis agrégées pour produire le résultat final.
- **Exemple intuitif** : Pour compter les occurrences de mots dans un ensemble de documents, chaque mot est transformé en une clé lors de la phase *Map*. Ensuite, la phase *Reduce* additionne les occurrences des mots identiques pour fournir un décompte global.
- Modèle particulièrement adapté aux systèmes massivement parallèles, offrant scalabilité, tolérance aux pannes et une exécution efficace des tâches sur des clusters distribués.

Principe de Fonctionnement

Phase Map

- Appliquée à chaque **élément** : prend un document en entrée dans notre cas.
- Chaque document peut produire **0, 1 ou plusieurs paires (clé, valeur)** en sortie.

Phase de Shuffle et Tri

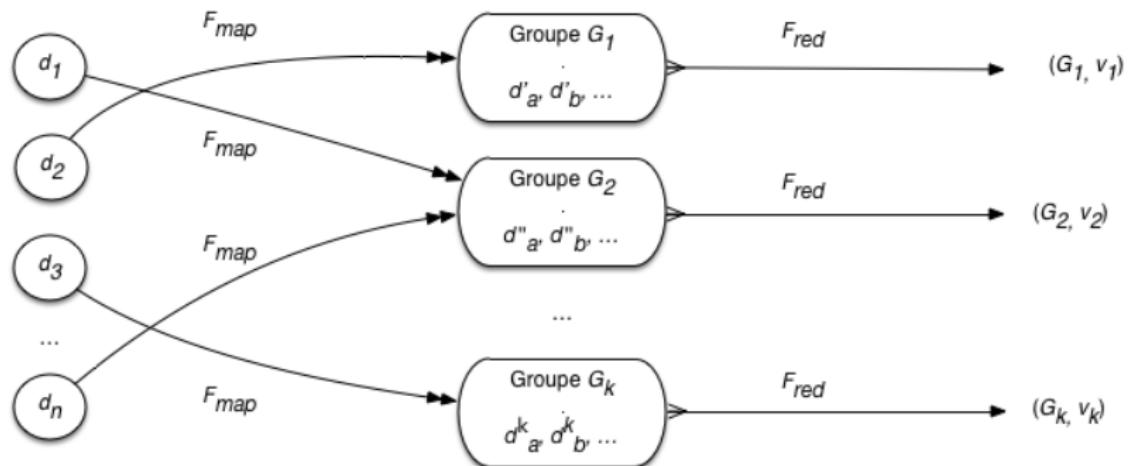
- Les **valeurs** générées par les mappers sont **regroupées par clé** ⇒ **un groupe est créé pour chaque clé**.
- Chaque groupe contient une clé et une liste de valeurs : **(k, list(v))**.
- Ces groupes **(k, liste(v))** sont envoyés aux reducers.

Phase Reduce

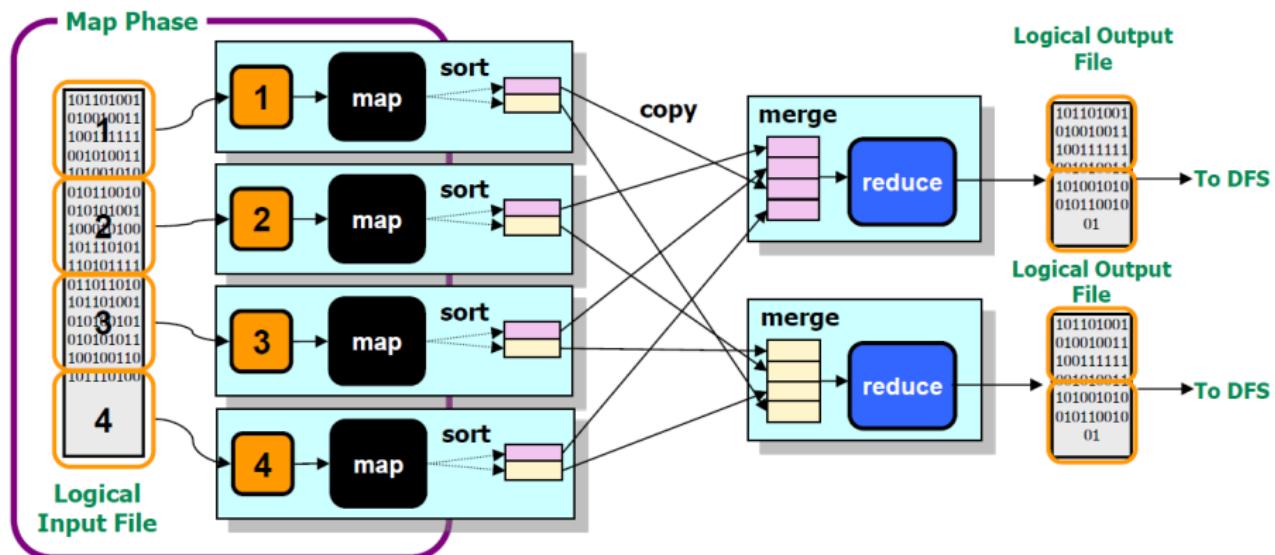
- Appliquée à chaque groupe **(k, liste(v))**, la fonction retourne une nouvelle paire **(k, v)**.
- En théorie, cette phase ne commence qu'une fois que tous les mappers ont terminé.
- En pratique, les reducers peuvent démarrer pendant que les mappers finissent leur travail pour accélérer le traitement.

Principe de Fonctionnement

La collection La fonction de *map* Les groupes La fonction de *reduce* Le résultat



Principe de Fonctionnement

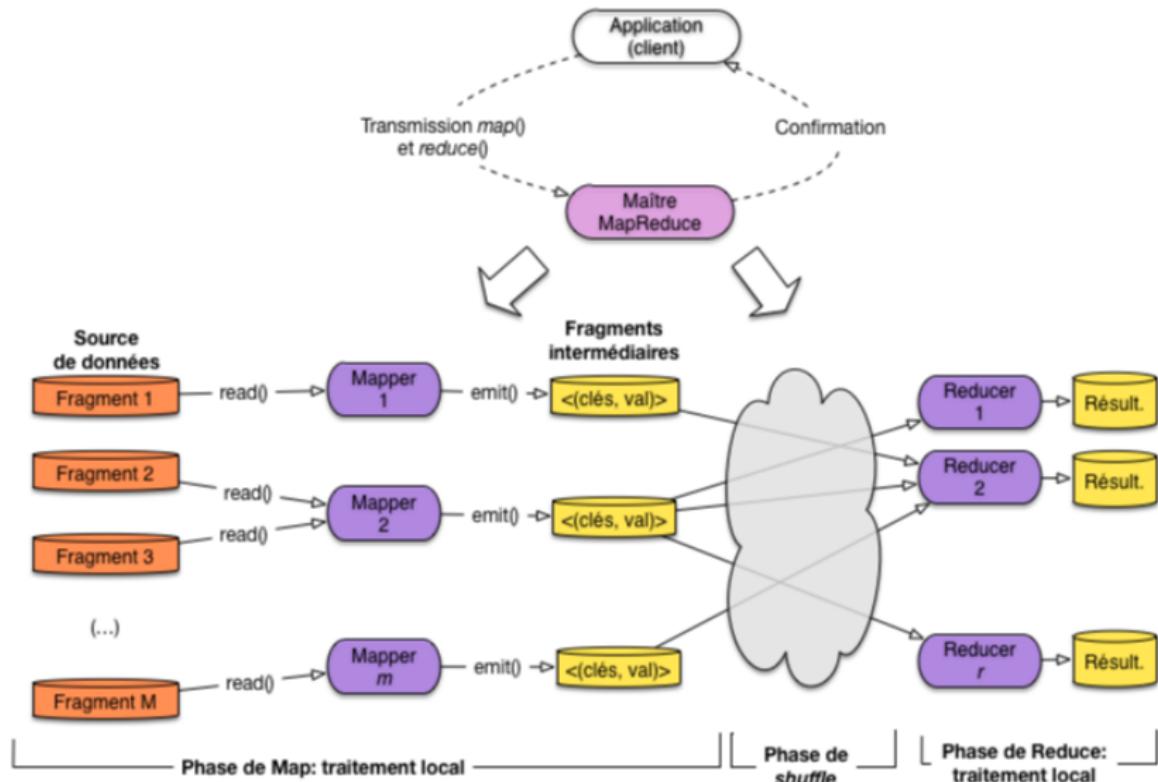


Pseudo-code pour le comptage des mots avec Map/Reduce

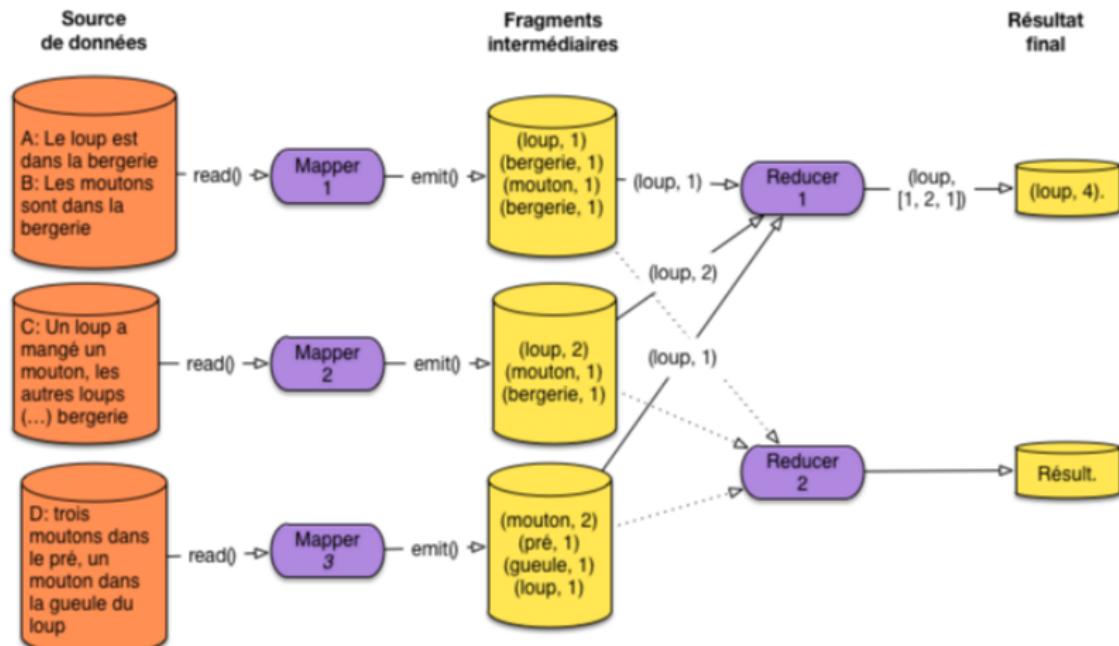
```
1 Map(String key, String document):  
2     // Diviser le document en mots  
3     for each word in document:  
4         Emit(word, 1)
```

```
1 Reduce(String word, Iterator partialCounts):  
2     // Additionner les occurrences des mots  
3     int total = 0  
4     for each count in partialCounts:  
5         total += count  
6     Emit(word, total)
```

Vue d'ensemble



Vue d'ensemble



Chapitre 3 - Traitement distribué avec Map/Reduce et YARN

- Map/Reduce, quésaco?
- **Session Illustrative**
- Limites de Map/Reduce
- YARN, quésaco?
- Architecture de YARN
- Quiz

Session Illustrative : Application MapReduce (WordCount)

Exemple Map/Reduce simple de WordCount

- ➊ Création d'un projet Maven (JDK 1.8).
- ➋ Configuration des dépendances Hadoop dans pom.xml.

Dépendances Maven

```
1 <dependencies>
2   <dependency>
3     <groupId>org.apache.hadoop</groupId>
4     <artifactId>hadoop-common</artifactId>
5     <version>2.7.2</version>
6   </dependency>
7   <dependency>
8     <groupId>org.apache.hadoop</groupId>
9     <artifactId>hadoop-mapreduce-client-core</artifactId>
10    <version>2.7.2</version>
11  </dependency>
12  <dependency>
13    <groupId>org.apache.hadoop</groupId>
14    <artifactId>hadoop-hdfs</artifactId>
15    <version>2.7.2</version>
16  </dependency>
17  <dependency>
18    <groupId>org.apache.hadoop</groupId>
19    <artifactId>hadoop-mapreduce-client-common</artifactId>
20    <version>2.7.2</version>
21  </dependency>
22 </dependencies>
```

Classe TokenizerMapper

```
1 package bigdata.tpl;
2 import org.apache.hadoop.io.IntWritable;
3 import org.apache.hadoop.io.Text;
4 import org.apache.hadoop.mapreduce.Mapper;
5 import java.io.IOException;
6 import java.util.StringTokenizer;
7
8 public class TokenizerMapper extends Mapper<Object, Text, Text,
9     IntWritable> {
10    private final static IntWritable one = new IntWritable(1);
11    private Text word = new Text();
12
13    public void map(Object key, Text value, Context context) throws
14        IOException, InterruptedException {
15        StringTokenizer itr = new StringTokenizer(value.toString());
16        while (itr.hasMoreTokens()) {
17            word.set(itr.nextToken());
18            context.write(word, one);
19        }
20    }
21 }
```

Listing 2: TokenizerMapper.java



Classe IntSumReducer

```
1 package bigdata.tpl;
2 import org.apache.hadoop.io.IntWritable;
3 import org.apache.hadoop.io.Text;
4 import org.apache.hadoop.mapreduce.Reducer;
5 import java.io.IOException;
6
7 public class IntSumReducer extends Reducer<Text, IntWritable, Text,
8     IntWritable> {
9     private IntWritable result = new IntWritable();
10
11     public void reduce(Text key, Iterable<IntWritable> values, Context
12         context) throws IOException, InterruptedException {
13         int sum = 0;
14         for (IntWritable val : values) {
15             System.out.println("value: " + val.get());
16             sum += val.get();
17         }
18         System.out.println("--> Sum = " + sum);
19         result.set(sum);
20         context.write(key, result);
21     }
22 }
```

Classe WordCount

```
1 package bigdata.tpl;
2 import org.apache.hadoop.conf.Configuration;
3 import org.apache.hadoop.fs.Path;
4 import org.apache.hadoop.io.IntWritable;
5 import org.apache.hadoop.io.Text;
6 import org.apache.hadoop.mapreduce.Job;
7 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
8 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
9
10 public class WordCount {
11     public static void main(String[] args) throws Exception {
12         Configuration conf = new Configuration();
13         Job job = Job.getInstance(conf, "word count");
14         job.setJarByClass(WordCount.class);
15         job.setMapperClass(TokenizerMapper.class);
16         job.setCombinerClass(IntSumReducer.class);
17         job.setReducerClass(IntSumReducer.class);
18         job.setOutputKeyClass(Text.class);
19         job.setOutputValueClass(IntWritable.class);
20         FileInputFormat.addInputPath(job, new Path(args[0]));
21         FileOutputFormat.setOutputPath(job, new Path(args[1]));
22         System.exit(job.waitForCompletion(true) ? 0 : 1);
23     }
24 }
```

Listing 4: WordCount.java

Chapitre 3 - Traitement distribué avec Map/Reduce et YARN

- Map/Reduce, quésaco?
- Session Illustrative
- **Limites de Map/Reduce**
- YARN, quésaco?
- Architecture de YARN
- Quiz

Hadoop v1 : JobTracker et TaskTracker

Hadoop Map/Reduce utilise 2 types de processus

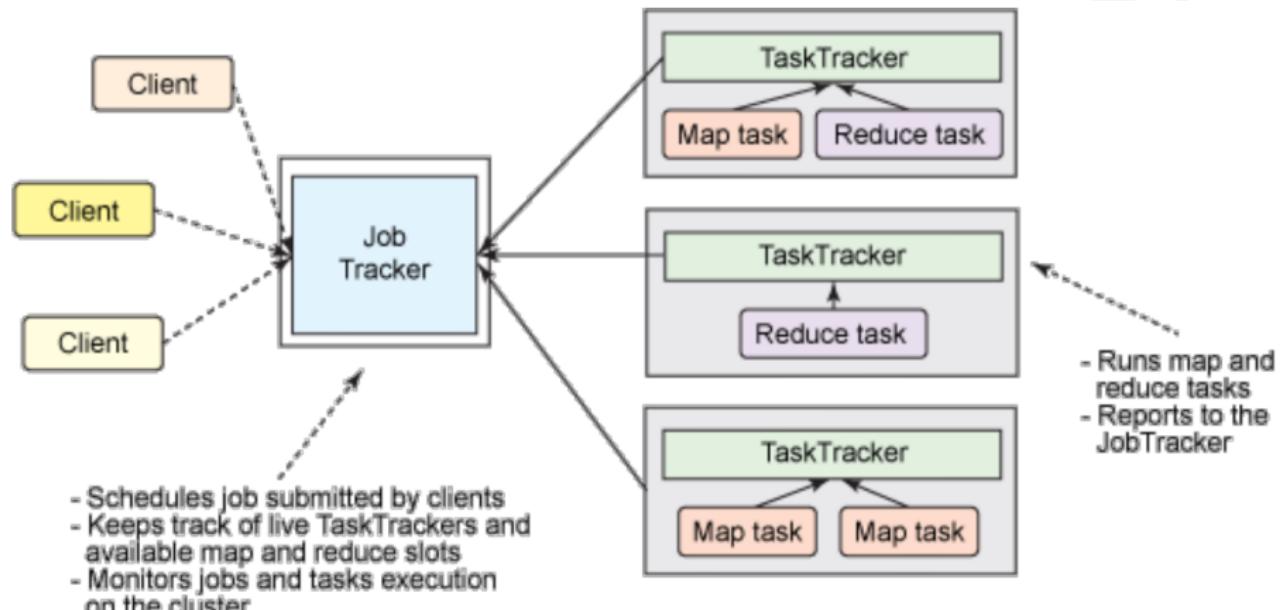
- **JobTracker**

- Responsable de l'**orchestration** des jobs (gestion des soumissions, surveillance des tâches et reprise en cas de panne).
- Le JobTracker décide comment distribuer les tâches Map et Reduce entre les nœuds du cluster.
- Il communique avec le NameNode pour localiser les données à traiter et planifie les tâches sur les nœuds qui hébergent ou sont proches des données.

- **TaskTracker**

- Gère l'**exécution des tâches** (Map ou Reduce) sur chaque nœud du cluster.
- Chaque nœud a un TaskTracker qui exécute les tâches assignées par le JobTracker.
- Il surveille l'état des tâches sur son nœud et envoie des rapports (heartbeats) réguliers au JobTracker pour indiquer la progression ou signaler des erreurs.

Hadoop v1 JobTracker et TaskTracker



Hadoop v1 : Déroulement typique (1/2)

- Déroulement typique :

- ➊ Les applications clientes soumettent les jobs Map/Reduce au JobTracker, qui les planifie.
- ➋ Le **JobTracker consulte le NameNode** pour déterminer quels nœuds contiennent les données nécessaires et planifie les tâches Map sur ces nœuds ou des nœuds proches (**principe de la localité des données**).
- ➌ Si un nœud est surchargé, les tâches Map peuvent être planifiées sur un autre nœud dans la même rack (zone physique de stockage).
- ➍ Chaque **TaskTracker exécute localement la fonction Map()** qui lui a été assignée. Il traite les blocs de données stockés localement ou proches.

Hadoop v1 : Déroulement typique (2/2)

- Déroulement typique (suite) :
 - ⑤ Le TaskTracker **envoie périodiquement des heartbeats** au JobTracker pour informer de la progression ou signaler des échecs.
 - ⑥ Si un TaskTracker échoue (s'il cesse d'envoyer des heartbeats), le JobTracker réaffecte les tâches incomplètes à d'autres TaskTrackers disponibles.
 - ⑦ Après la phase Map, le JobTracker planifie et **injecte les tâches Reduce** sur les nœuds disponibles. Chaque nœud a un nombre fixe de slots pour gérer les tâches en attente.
- **Le JobTracker est un point de défaillance unique (SPOF)** dans Hadoop v1. Si le JobTracker tombe en panne, tout le système est affecté. Cette limitation a été résolue dans Hadoop v2 avec YARN (Yet Another Resource Negotiator), qui introduit une gestion plus distribuée et résiliente.

Le problème du "Busy JobTracker" dans Hadoop v1

• Le JobTracker dans Hadoop v1

- Le JobTracker est responsable de la gestion centrale de toutes les tâches MapReduce dans le cluster.
- Il doit surveiller les tâches, gérer les échecs, répartir les nouvelles tâches, et suivre les heartbeats des TaskTrackers.

• Surcharge du JobTracker :

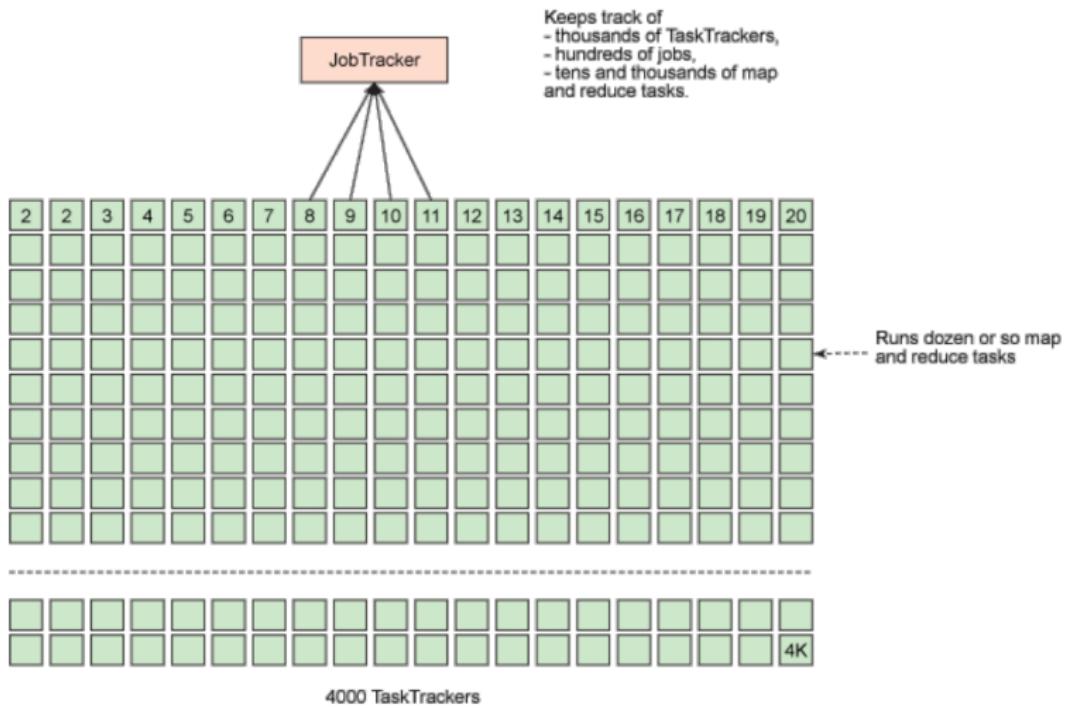
- Avec un grand nombre de tâches à gérer, le JobTracker peut devenir surchargé, ce qui ralentit l'allocation des tâches et la gestion des échecs.
- Un nombre élevé de **heartbeats** des TaskTrackers, combiné à la gestion des tâches et des échecs, peut submerger le JobTracker.

• Résolution dans Hadoop v2 (YARN) :

- Hadoop v2 sépare la gestion des ressources du traitement des tâches en introduisant **YARN** (Yet Another Resource Negotiator), réduisant ainsi la charge sur un gestionnaire central.

Le problème du "Too Busy JobTracker" dans Hadoop v1

Scalability in MRv1: Busy JobTracker



Chapitre 3 - Traitement distribué avec Map/Reduce et YARN

- Map/Reduce, quésaco?
- Session Illustrative
- Limites de Map/Reduce
- **YARN, quésaco?**
- Architecture de YARN
- Quiz

Introduction à YARN

Dans Hadoop v2, **JobTracker** et **TaskTrackers** sont remplacés par

- **ResourceManager** : Responsable de la gestion des ressources à l'échelle du cluster.
- **ApplicationMaster** : Un par application, responsable de l'exécution d'une tâche spécifique.
- **NodeManagers** : Remplace les TaskTrackers, un par nœud, responsable de la gestion des conteneurs.

Chapitre 3 - Traitement distribué avec Map/Reduce et YARN

- Map/Reduce, quésaco?
- Session Illustrative
- Limites de Map/Reduce
- YARN, quésaco?
- **Architecture de YARN**
- Quiz

ResourceManager

- **ResourceManager** : Le nœud maître de YARN, un par cluster.
- Responsable de la collecte des informations sur les **ressources** disponibles (RAM, CPU, etc.).
- Gère plusieurs services, dont le plus important est le **Scheduler**.
- **Scheduler** : Responsable de l'**allocation des ressources** aux applications tournant sur Hadoop.
 - Plusieurs stratégies d'ordonnancement sont disponibles, comme **FIFO**, **Fair Scheduler**, et **Capacity Scheduler**.

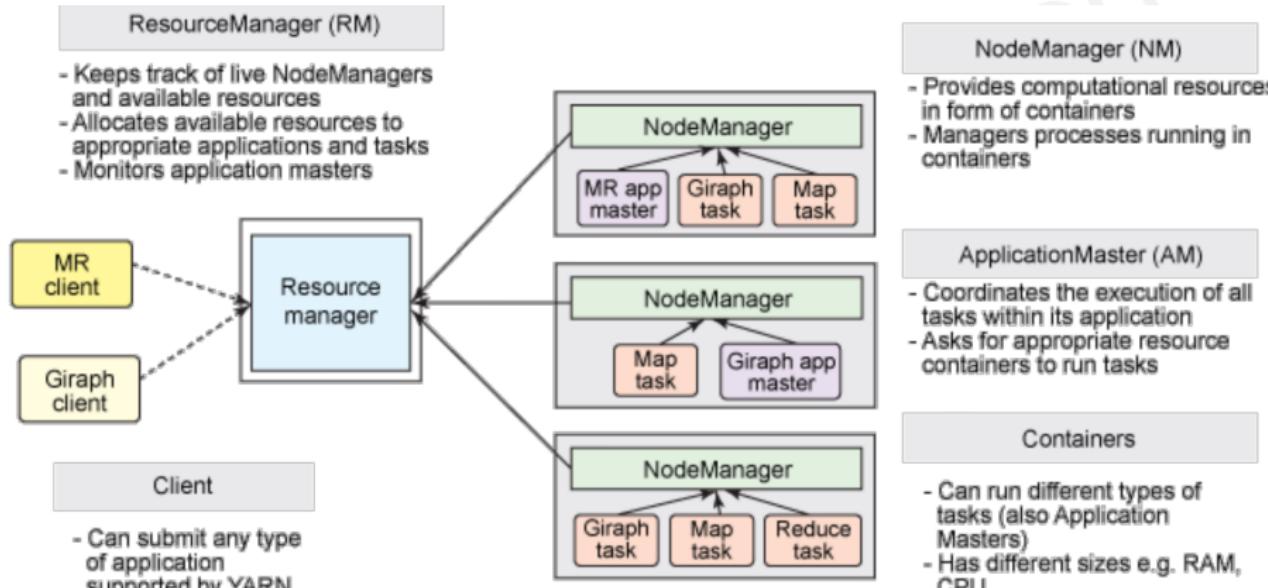
ApplicationMaster

- **ApplicationMaster (AM)** : Un par application, tourne dans un conteneur indépendant.
- L'AM gère l'exécution de l'application en interagissant avec le **ResourceManager**.
- Envoie périodiquement des **heartbeats** au ResourceManager et réclame des ressources supplémentaires au besoin.
- L'AM contrôle l'ensemble du cycle de vie de l'application, de l'allocation des conteneurs à la libération des ressources.

NodeManagers

- **NodeManager** : Un par noeud, remplace les TaskTrackers.
- Gère les **ressources** (mémoire, CPU) au niveau du noeud.
- Crée des **conteneurs dynamiques** sur chaque noeud et informe périodiquement le ResourceManager de l'utilisation des ressources du noeud.
- Contrairement aux slots fixes des TaskTrackers, les conteneurs peuvent être utilisés de manière flexible pour les tâches **Map**, **Reduce**, ou même d'autres frameworks que MapReduce.

Architecture YARN

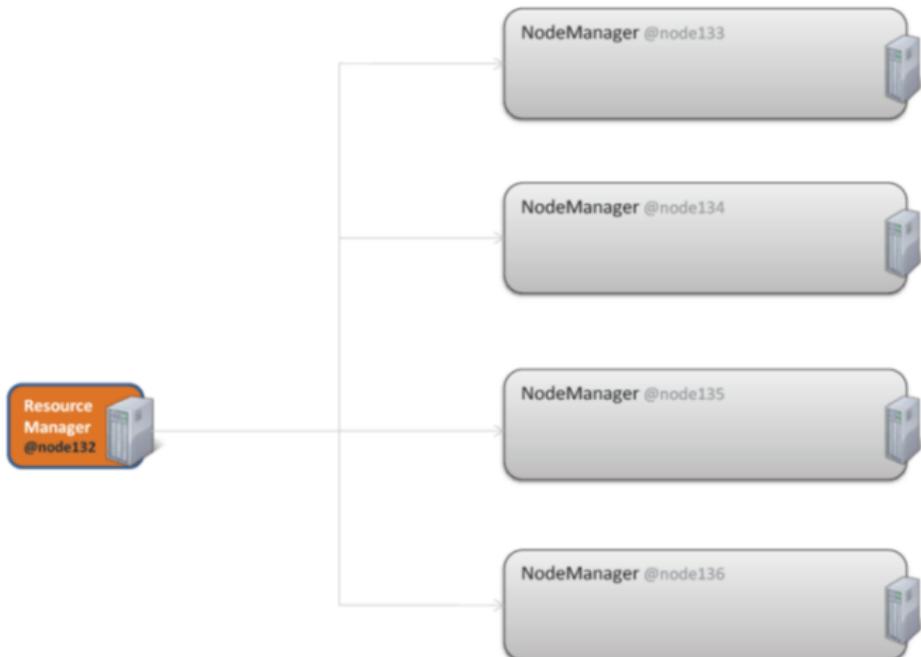


Hadoop v2 YARN : déroulement typique de l'exécution d'une application

- ➊ Le client soumet une application MapReduce au ResourceManager
- ➋ Le ResourceManager négocie un container pour l'ApplicationMaster et lance l'ApplicationMaster.
- ➌ L'ApplicationMaster démarre et s'enregistre auprès du ResourceManager.
- ➍ L'ApplicationMaster négocie des ressources (*resource containers*) pour l'application cliente.
- ➎ Le NodeManager lance des containers pour l'application.
- ➏ Lors de l'exécution, le client notifie l'ApplicationMaster de la progression de l'exécution. Une fois l'exécution terminée, l'ApplicationMaster s'arrête et libère les containers associés à l'application.

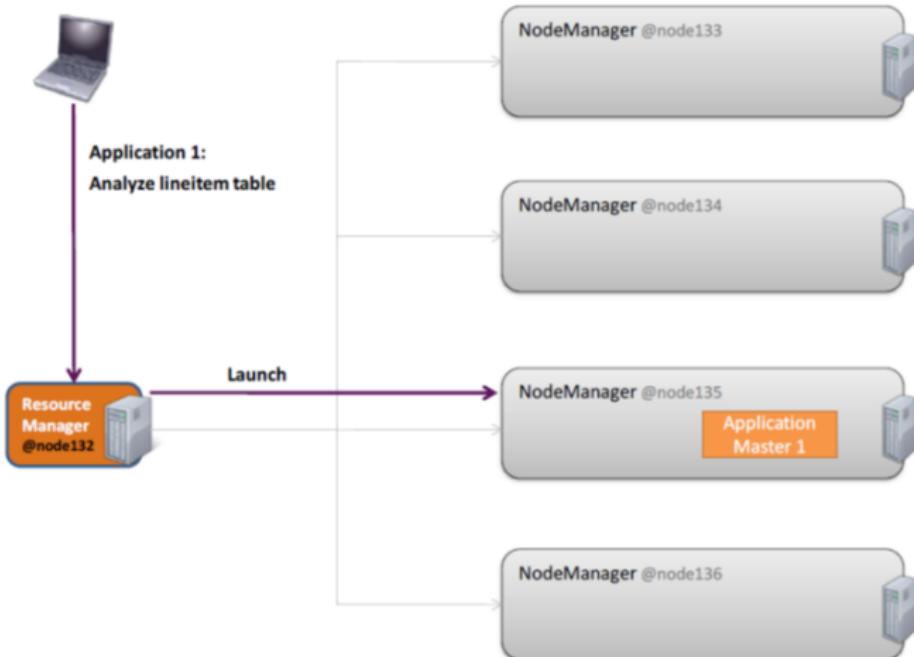
Hadoop v2 YARN : déroulement typique de l'exécution d'une application

Running an application in YARN (1 of 7)



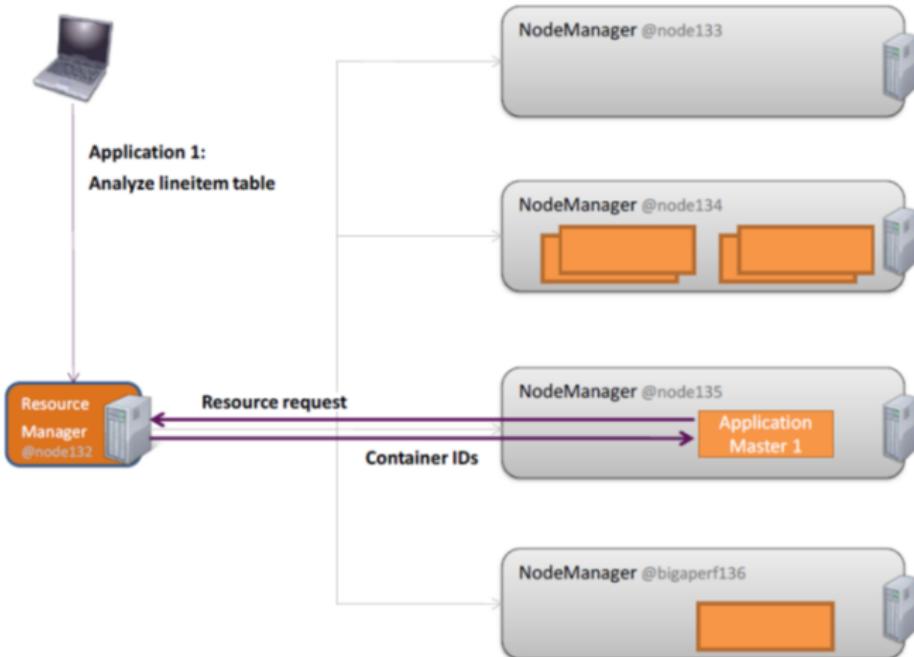
Hadoop v2 YARN : déroulement typique de l'exécution d'une application

Running an application in YARN (2 of 7)



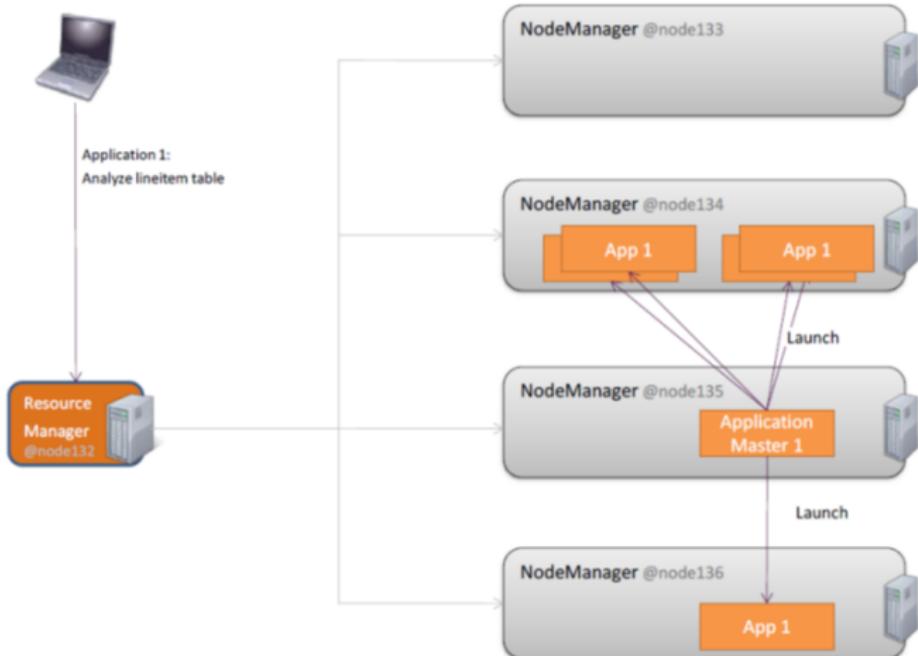
Hadoop v2 YARN : déroulement typique de l'exécution d'une application

Running an application in YARN (3 of 7)



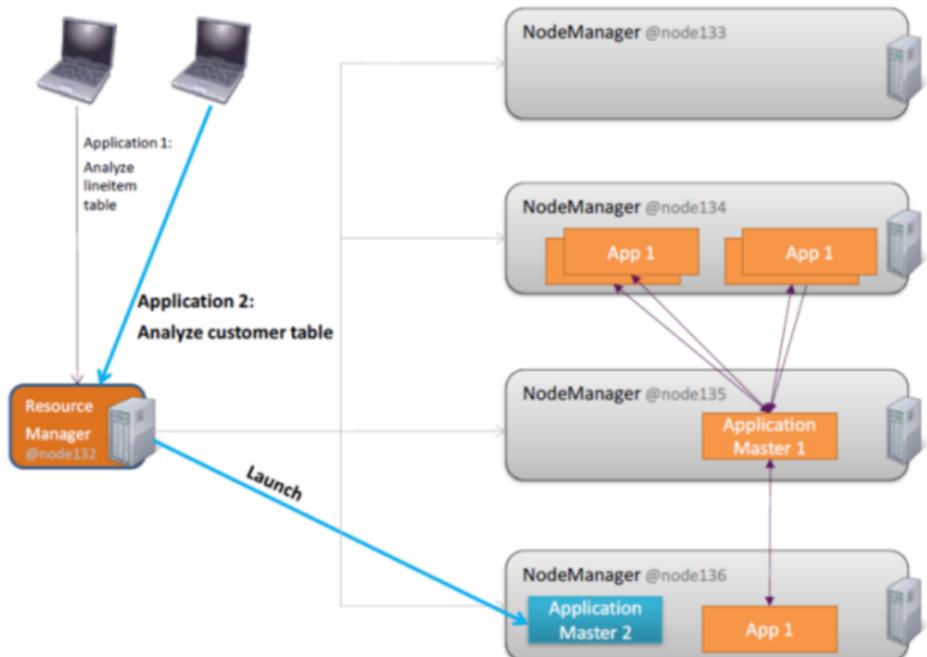
Hadoop v2 YARN : déroulement typique de l'exécution d'une application

Running an application in YARN (4 of 7)



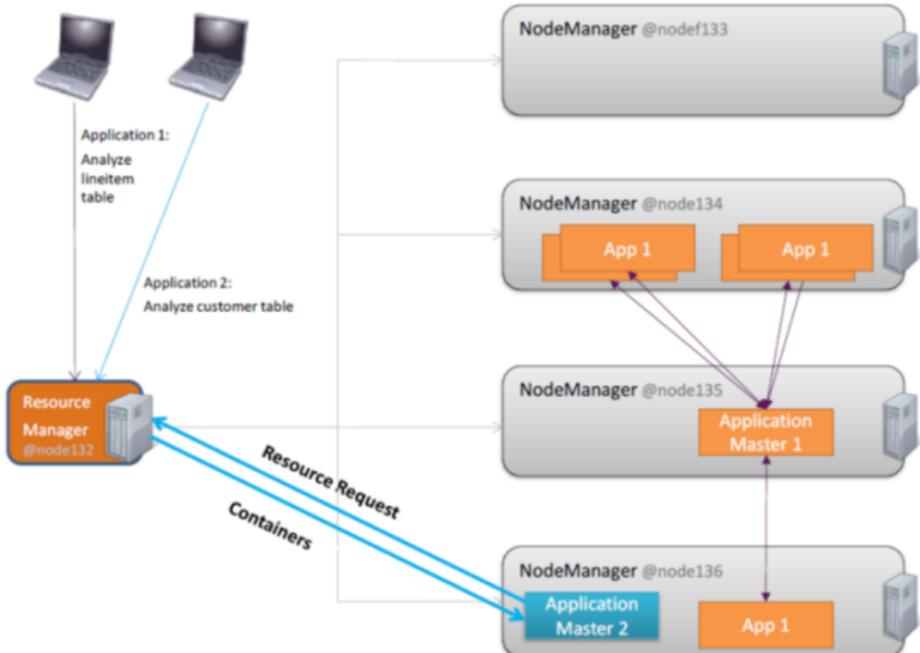
Hadoop v2 YARN : déroulement typique de l'exécution d'une application

Running an application in YARN (5 of 7)



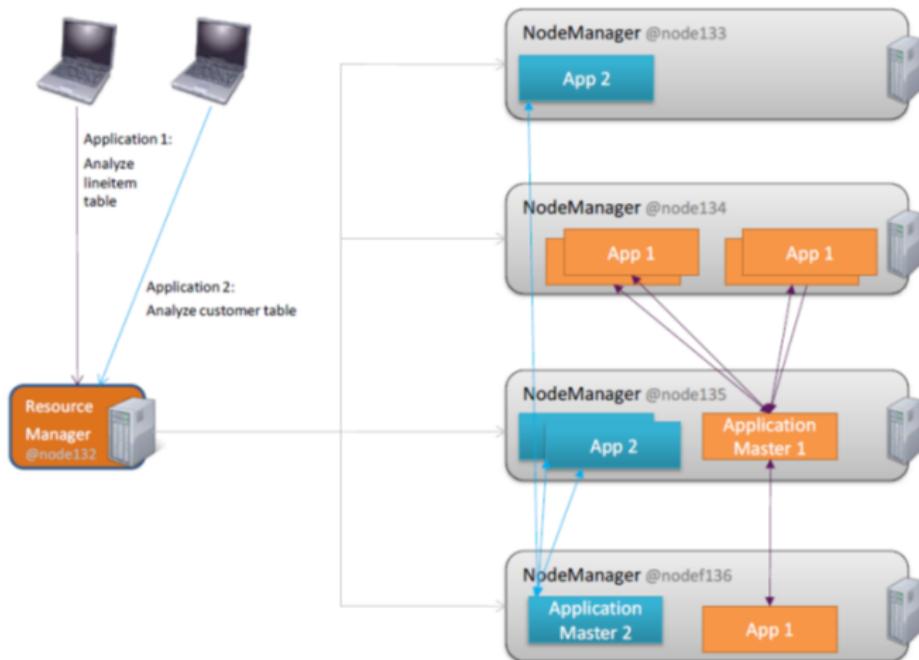
Hadoop v2 YARN : déroulement typique de l'exécution d'une application

Running an application in YARN (6 of 7)



Hadoop v2 YARN : déroulement typique de l'exécution d'une application

Running an application in YARN (7 of 7)



Comparaison entre Hadoop Map/Reduce v1 et Hadoop YARN v2

Caractéristiques	Hadoop Map/Reduce v1	Hadoop YARN v2
Composants principaux	JobTracker, TaskTrackers	ResourceManager, NodeManagers
Gestion des tâches	Centralisée par JobTracker	Décentralisée par ApplicationMaster
SPOF (Single Point of Failure)	Oui, JobTracker unique	Non, gestion distribuée
Scalabilité	Limité par le JobTracker	Haute scalabilité grâce à YARN
Ressources gérées	Slots fixes pour Map/Reduce	Conteneurs dynamiques
Flexibilité	Spécifique à Map/Reduce	Supporte plusieurs frameworks (Spark, etc.)
Ordonnancement des tâches	FIFO principalement	FIFO, Fair Scheduler, Capacity

Chapitre 3 - Traitement distribué avec Map/Reduce et YARN

- Map/Reduce, quésaco?
- Session Illustrative
- Limites de Map/Reduce
- YARN, quésaco?
- Architecture de YARN
- Quiz

Quiz

- ➊ Quel composant central de Hadoop v1 est remplacé par le ResourceManager dans Hadoop v2 YARN ?
- ➋ Quelle est la différence entre un TaskTracker dans Hadoop v1 et un NodeManager dans Hadoop v2 ?
- ➌ Pourquoi le JobTracker dans Hadoop v1 est considéré comme un point de défaillance unique (SPOF) ?
- ➍ Quelles sont les principales stratégies d'ordonnancement prises en charge par le Scheduler dans YARN ?
- ➎ Quelle est la différence entre les slots dans Hadoop v1 et les conteneurs dans Hadoop v2 ?

De Map/Reduce à Spark

• Résumé des Points Clés

- HDFS : Système de fichiers distribué pour le stockage de données massives.
- YARN : Gestionnaire de ressources offrant une meilleure scalabilité et flexibilité.
- **Limites** : Hadoop MapReduce est performant mais peut être lent et complexe à programmer.
- **Apache Spark** : peut tourner sur YARN avec des performances 10 à 100 fois plus rapides dans certains cas que Map/Reduce et une API plus conviviale.
⇒ **Rendez-vous aux prochains cours!**