Fiche de Révision Complète : Écosystème Hadoop

Généré pour l'examen de Big Data

30 décembre 2025

Table des matières

1 Chapitre 1 : Introduction et Framework Hadoop

1.1 Concepts Fondamentaux

- **Définition :** Hadoop est un framework Open Source (géré par la fondation Apache) développé en Java pour le stockage et le traitement distribué de volumes massifs de données (Big Data).
- **Philosophie clé :** "Bring the computation to the data" (Apporter le calcul vers les données). Au lieu de déplacer des pétaoctets de données vers un serveur de calcul (ce qui saturerait le réseau), on envoie le petit programme de calcul directement là où les données sont stockées.
- **Les 7V du Big Data :**
  - **Volume :** Quantité massive de données.
  - **Vélocité :** Vitesse de génération et de traitement.
  - **Variété :** Données structurées, semi-structurées et non structurées.
  - **Véracité :** Fiabilité et précision des données.
  - **Valeur :** Transformer les données en informations utiles.
  - **Visualisation :** Représentation graphique pour l'analyse.
  - **Viralité :** Vitesse de diffusion de l'information.

1.2 Architecture Distribuée : Scale-Up vs Scale-Out

- **Scale-Up (Vertical) :** Consiste à ajouter plus de puissance (RAM, CPU) à une seule machine. Cette méthode est coûteuse et a des limites physiques.
- **Scale-Out (Horizontal) :** Consiste à ajouter plus de machines (nœuds) au cluster. C'est l'approche utilisée par Hadoop, permettant d'utiliser du matériel standard ("commodity hardware").

1.3 Architecture Master/Slave

Un cluster Hadoop fonctionne selon une architecture Maître/Esclave :

- **Master Nodes (Maîtres) :** Coordonnent le cluster et gèrent les métadonnées (ex : NameNode, ResourceManager).
- **Slave Nodes (Esclaves) :** Exécutent les tâches réelles de stockage et de calcul (ex : DataNode, NodeManager).

1.4 L'Écosystème Hadoop (Outils Principaux)

- **HDFS :** Système de fichiers distribué.
- **MapReduce :** Moteur de traitement historique.
- **Hive :** Permet de faire des requêtes SQL sur Hadoop (Data Warehousing).
- **Pig :** Langage de script pour les transformations de données (ETL).
- **Sqoop :** Import/Export de données entre Hadoop et les bases de données relationnelles (RDBMS).
- **Flume :** Ingestion de flux de données en temps réel (logs, réseaux sociaux).

— **Zookeeper :** Service de coordination pour les applications distribuées.
— **Oozie :** Planificateur de workflows (Scheduler).

2 Chapitre 2 : HDFS (Hadoop Distributed File System)

2.1 Définition et Architecture

HDFS est inspiré du GFS (Google File System). Il est conçu pour être fiable et tolérant aux pannes sur du matériel standard. Il repose sur deux démons principaux :

1. **NameNode (Master) :**
   - C'est le cerveau du système. Il gère les **métadonnées** (noms des fichiers, arborescence, permissions).
   - Il sait quel fichier est découpé en quels blocs et sur quels DataNodes ces blocs se trouvent.
   - Il ne stocke **PAS** les données réelles.
   - Dans Hadoop v1, c'est un SPOF (*Single Point of Failure*) : s'il tombe, le cluster est inutilisable.
2. **DataNode (Slave) :**
   - Il stocke les données réelles sous forme de blocs sur son disque local.
   - Il gère les lectures et écritures demandées par le client.
   - Il envoie un signal de vie (**Heartbeat**) au NameNode toutes les 3 secondes pour signaler qu'il est fonctionnel.
3. **Secondary NameNode :**
   - Ce n'est PAS un remplaçant du NameNode (contrairement à son nom).
   - Il aide le NameNode à effectuer des "Checkpoints" (fusionner les journaux de modifications avec l'image du système de fichiers) pour éviter que les logs ne deviennent trop volumineux.

2.2 Stockage : Blocs et Réplication

- **Blocs :** Les fichiers volumineux sont découpés en morceaux appelés blocs.
  - Taille par défaut (Hadoop 1.x) : 64 Mo.
  - Taille par défaut (Hadoop 2.x) : **128 Mo**.
- **Réplication (Tolérance aux pannes) :**
  - Chaque bloc est copié plusieurs fois pour éviter la perte de données en cas de panne disque/machine.
  - Facteur de réplication par défaut : **3**.

2.3 Stratégie de Placement des Répliques (Rack Awareness)

Pour optimiser la sécurité et la bande passante, HDFS place les 3 copies ainsi :

1. **Réplique 1 :** Sur le nœud local (ou un nœud aléatoire si l'écriture vient de l'extérieur).
2. **Réplique 2 :** Sur un nœud situé dans un **autre rack** (armoire physique).
3. **Réplique 3 :** Sur un autre nœud du **même rack que la réplique 2**.

Avantage : Si un rack entier tombe en panne (ex : panne de switch), la Réplique 1 (sur l'autre rack) est toujours disponible.

2.4 Haute Disponibilité (High Availability - HA) dans Hadoop 2.x

Pour résoudre le problème du SPOF du NameNode :

— On utilise deux NameNodes : un **Actif** et un **Standby** (en veille).

— **JournalNodes :** Un groupe de machines qui partagent les journaux d'édition (Edit-Logs) pour garder le Standby synchronisé en temps réel avec l'Actif.

— **Zookeeper :** Surveille la santé du NameNode Actif et déclenche le basculement automatique (*Failover*) vers le Standby en cas de panne.

3 Chapitre 3 : MapReduce et YARN

3.1 Le Paradigme MapReduce

MapReduce est un modèle de programmation pour le traitement parallèle de données massives. Le traitement se fait en deux étapes principales, séparées par une phase de transfert.

1. **Phase MAP (Diviser et Traiter) :**
   - Les données d'entrée sont découpées (*Input Splits*).
   - Chaque Mappeur traite un bloc de données localement et produit des résultats intermédiaires sous forme de paires <**Clé, Valeur**>.
   - Exemple : <*Mois*, *Montant Vente*>.
2. **Phase SHUFFLE & SORT (Mélanger et Trier) :**
   - C'est l'étape "magique" gérée par le framework.
   - Hadoop déplace les données à travers le réseau pour regrouper toutes les valeurs associées à la **même clé** vers le même Réducteur.
   - Résultat : <*Mois*, [*Montant1*, *Montant2*, ...]>.
3. **Phase REDUCE (Agréger) :**
   - Le Réducteur reçoit la liste des valeurs pour une clé donnée.
   - Il effectue une opération d'agrégation (Somme, Moyenne, Max, etc.) pour produire le résultat final.

3.2 Évolution de l'Architecture : Hadoop v1 vs Hadoop v2 (YARN)

3.2.1 Hadoop v1 : Architecture Monolithique (Obsolète)

- **JobTracker (Maître) :** Gérait à la fois l'allocation des ressources (RAM/CPU) ET la planification des tâches (Scheduling). C'était un goulot d'étranglement majeur.
- **TaskTracker (Esclave) :** Exécutait les tâches dans des **Slots** fixes (ex : 2 slots Map, 2 slots Reduce).
- **Problème :** Manque de flexibilité (slots vides gaspillés) et impossibilité de lancer autre chose que du MapReduce.

3.2.2 Hadoop v2 avec YARN (Yet Another Resource Negotiator)

YARN sépare la gestion des ressources de la gestion des jobs, transformant Hadoop en un véritable système d'exploitation de données "Multi-tenant".

Les nouveaux composants :

- **ResourceManager (RM) - Maître Global :** Gère les ressources pour l'ensemble du cluster. Il est unique.
- **NodeManager (NM) - Esclave Local :** Tourne sur chaque machine. Il gère les **Conteneurs** et surveille l'utilisation locale des ressources.
- **ApplicationMaster (AM) - Par Job :** Un composant spécifique créé pour chaque application lancée. Il négocie les ressources auprès du ResourceManager, puis demande aux NodeManagers d'exécuter les tâches.
- **Le Conteneur (Container) :** Remplace les "slots" fixes. C'est une unité de ressource dynamique (CPU + RAM) qui peut exécuter n'importe quel type de tâche.

3.2.3 Tableau Comparatif Rapide

Critère	**Hadoop v1**	**Hadoop v2 (YARN)**
Gestion Ressources	JobTracker (Surchargé)	ResourceManager (Dédié)
Unité de Calcul	Slots (Statiques)	Conteneurs (Dynamiques)
Applications	MapReduce Uniquement	Multi-tenant (MapReduce, Spark, Storm, etc.)