**Compte-rendu HDFS et MapReduce**

**Hypothèses de départ :**

* Panne matérielle très embêtant
  + Système tolérant aux pannes
* Accès continu des données
* Gestion de données de plus en plus massive et traitement léger
  + Système distribué (stockage et traitement)
* Mise à jour génère beaucoup d’incohérence (si mal gérée)
  + Une seule écriture et plusieurs lectures 🡪 immutable

**Définition HDFS (Hadoop Distributed File System)**

* Système de maintient de fichier à travers des nœuds d’un cluster

{ Dans un disque dur, l’arborescence des fichiers est abstraite. En réalité les fichiers sont splittés et stockés dans des blocs séparés du disque dur et ceci de manière discontinue en fonction des blocs disponibles. Lorsqu’on récupère un fichier, un algorithme système rassemble et reconstitue le fichier}

* HDFS gère les blocs d’un fichier sur les nœuds d’un cluster.
* Pour mettre en place et utiliser HDFS, il faut installer :
  + Un NameNode (et SNode)
  + Plusieurs DataNode

**NameNode (nœud maître)**

* Un service (ou composant) service qui gère les métadonnées, répertoire, découpage et reconstitution des fichiers.
* Il est responsable d’élire des machines pour accueillir les réplicas.
* Unique dans le cluster mais dispose d’un backup (sauvegarde) en cas de panne

**DataNode (nœud esclave)**

* Un service qui gère le stockage, réplication, opération et restitution des blocs de fichier
* Il est installé sur tous les nœuds à part le nœud maître
* Par défaut, la capacité d’un bloc est de 128 Mo
* Par défaut, le facteur de réplication est de 3 (c’est paramétrable au niveau du fichier)

**Notions importantes :**

* Un client :
  + Toutes les applications qui peuvent interagir avec le HDFS
  + Soumet une demande lecture ou d’écriture
* Rack est un ensemble de serveurs connecté à un rooter pour se connecter à l'extérieur. L’idéal est de répliquer les blocs sur deux racks différents
* La réplication des données se fait automatiquement dans le but de faciliter la récupération en cas de perte de nœud (plusieurs fichiers dans différents nœuds) 🡺 tolérance aux pannes

**Gestion des pannes :**

* Chaque dataNode envoie un signe de vie (timestamp sous forme de **ping**) au nameNode dans un intervalle de temps régulière 🡪 Heartbeat
* Dès lors que le namenode ne reçoit pas de signal (timestamp non modifié), c’est que le datanode a lâché.
* Lorsqu’un datanode est en panne, le Zookeeper efface de son arbre de nœuds temporaire le nœud correspondant

{ Le Zookeeper a un rôle principale de coordinateur et aussi il sert de serveur qui fonctionne en mode corom (minimum 3 et toujours impaire) }

**Ecriture**

1. **Client demande au nameNode s’il peut écrire dans HDFS**
2. **NameNode répond en spécifiant les dataNodes (A1 et A2 par exemple)**
3. **Le client écrit et dispatch les blocs dans A1 et A2**
4. **Le dataNode A1 et A2 envoient un accusé de réception au NameNode**
5. **Réplication vers d’autres dataNodes (A2, A3 et A4 par exemple)**
6. **Ces dataNode accuse la réception des duplicas**

**Lecture**

1. **Client demande au NameNode de lire un fichier (avec ces métadonnées)**
2. **Le nameNode lui répond : les données sont dans ces dataNodes avec leurs adresse IP**
3. **Le client lit séquentiellement les blocs à partir des datanodes**

Pourquoi utiliser Files View dans Hortonwork ?

Exemple : un job spark traite et charge des données d’un serveur externe sous forme de fichier dans un répertoire HDFS. Pour vérifier sur le chargement s’est bien effectué, on doit aller dans Files View pour visualiser.

Pourquoi utiliser hdfs dfs dans la console shell de Hortonwork raj\_ops ?

Pour créer des scripts de traitement pour récupérer ou envoyer des fichiers entre HDFS et Linux (et vice versa)