Hadoop/HBase

Joseph ELIA

Table des matières

1.HADOOP:

Avant de commencer par l'introduction de Hadoop ,nous allons parler de l'état actuel des données. L'état des données peut être résumer en un mot...**Beaucoup**.90% des données dans le monde d'aujourd'hui ont été crées au cours des deux dernières années. Ces données proviennent de partout: des messages sur des sites de social media , de photos et de vidéos, des signaux GPS de téléphone, etc.

Il ya trois types de données, structuré, non structuré et semi-structuré. La majorité des données créées sont des données **non-structuré** (emails,pdf,documents...) et **semi-structuré** (xml,csv...).Cette augmentation de données est appelé "**data explosion**" qui a forme le Big Data.

Alors que font les grandes entreprises avec ces données? Google par exemple semble être le pionnier de tout, Ils ont dit que nous devons indexer des milliards de pages, donc ils ont construit la technologie **MapReduce** avec **GFS** (Google File System) et c'est ce que **HADOOP** est basé sur.

a)Qu'est-ce que Hadoop

Hadoop est une solution logicielle distribuée,Il s'agit d'un système réparti évolutif et tolérant aux pannes pour le stockage et le traitement des données.

b)L'architecture de Hadoop

Il y a deux composants principaux dans Hadoop : HDFS ,qui est le stockage et MapReduce,qui est la récupération et le traitement des données.

HDFS:est un self-Healing , High bandwidth Cluster Storage. Si nous mettons un grand fichier (petabyte) a l'intérieur de notre Hadoop Cluster, HDFS le divisera

en blocs (64MB par defaut) le distribuera ensuite à travers les nœuds du cluster. Dans la configuration HDFS il ya un facteur de réplication(RF=3 par defaut).Ce que cela signifie, c'est quand nous mettons un fichier dans Hadoop, il va s'assurer qu'il ya 3 copies de chaque bloc répartis sur tous les nœuds du cluster.L'avantage de la RF est que si nous perdons un nœud, il va auto-guérir et ré-reproduire les blocs qui sont sur ce nœud dans le reste des nœuds dans le cluster(C'est là ou la notion de tolérance aux pannes vient).Cette procédure est effectuée à l'aide du **NameNode.** Généralement **Hadoop** a un **namenode** et tout le reste sont des **DataNodes**. En princide le NameNode est un serveur Metadata Il tient en mémoire l'emplacement de chaque bloc et de chaque noeud et même s'il ya plusieurs racks configurés.

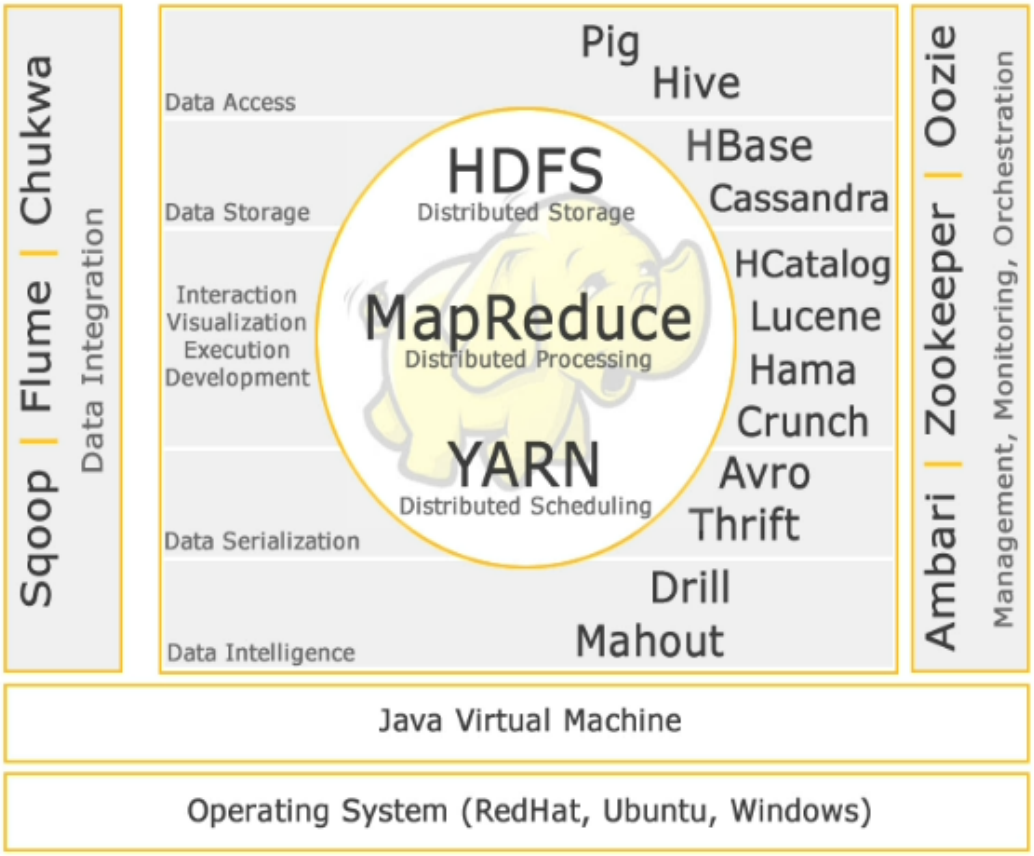
MapReduce: Hadoop est un système de base de traitement **Batch**,Alors nous travaillons sur toutes les données du cluster et nous ne faisons aucune recherche parce que cela ralentit la récupération des données. MapReduce consiste à travailler sur toutes les données à l'intérieur du cluster, Il ya un processus en deux étapes **Mapper** et **Reducer.** Les programmeurs écrivent le mappeur et disent à la Cluster quels points de données a récupérer, ainsi le **Reducer** prendra toutes ces données et les regroupe. La flexibilité de Hadoop est venu quand un grand nombre de services ont émergé qui font MapReduce de façon moins compliqué.

Donc Hadoop est **tolérant aux pannes** grâce a HDFS, **Flexible** dans la façon dont nous pouvons récupérer les données et dans le type de données que nous pouvons mettre dans Hadoop(**nonstructuré**,**semi-structuré et structuré**). En plus Hadoop est également **évolutif**. L'évolutivité est par défaut dans Hadoop parce que nous sommes dans un "distributed computing environement", par example MapReduce Job commencera a ralentir parce que nous ajoutons beaucoup de données au Cluster, pour le résoudre on ajoute simplement plus des DataNodes ce qui augmente la puissance de traitement globale de note Cluster. En plus de cela, Hadoop est Rack-aware (Cette fonctionnalité doit être configurée) .Dans

un environnement multi-rack Hadoop sait quel nœud appartient à ce rack donc il permettra à HDFS de faire plus de localité de données alors chaque fois qu'il reçoit un job de MapReduce, il va trouver le chemin le plus court vers les données.

c)L'écosystème de Hadoop:

L'écosystème Hadoop est divisé en plusieurs services de données qui sont offerts par de nombreux projets conçus pour hadoop :



- **Hadoop Core**:Hadoop core se compose de HDFS et Mapreduce, en plus il ya YARN, c'est une nouvelle version de Mapreduce mais l'architecture est différente (Au lieu d'avoir des Job Mapreduce nous avons des applications Yarn, ce qui est un ensemble de MapReduce Jobs).

- **Data access**:La raison pour laquelle nous avons besoin de plus de moyens d'accéder aux données à l'intérieur de Hadoop est parce que pas tout le monde est un programmeur Java, c, etc de bas niveau qui peut écrire Mapreduce Jobs, et même si vous êtes, certains MapReduce sont difficiles à faire (rejoindre, Groupe, filtrage). Il ya donc quelques projets d'accès aux données(PIG,HIVE...) qui compile le code et le convertit en un MapReduce Job à soumettre dans le cluster.

- **Data Storage** :Hadoop est un système de traitement **Batch**, si nous devons obtenir des données spécifiques, faire des écritures et le traitement des transactions en temps réel sur les données Hadoop de facon rapide, nous devons avoir des Column-oriented databases(Hbase,Cassandra).

- **Data Serialization** :La sérialisation est un moyen que nous pouvons prendre des données d'une application, les changer dans un format que nous pouvons soit stocker sur le disque ou l'échanger, donc une autre application peut désérialiser puis dans un format qu'ils comprennent.

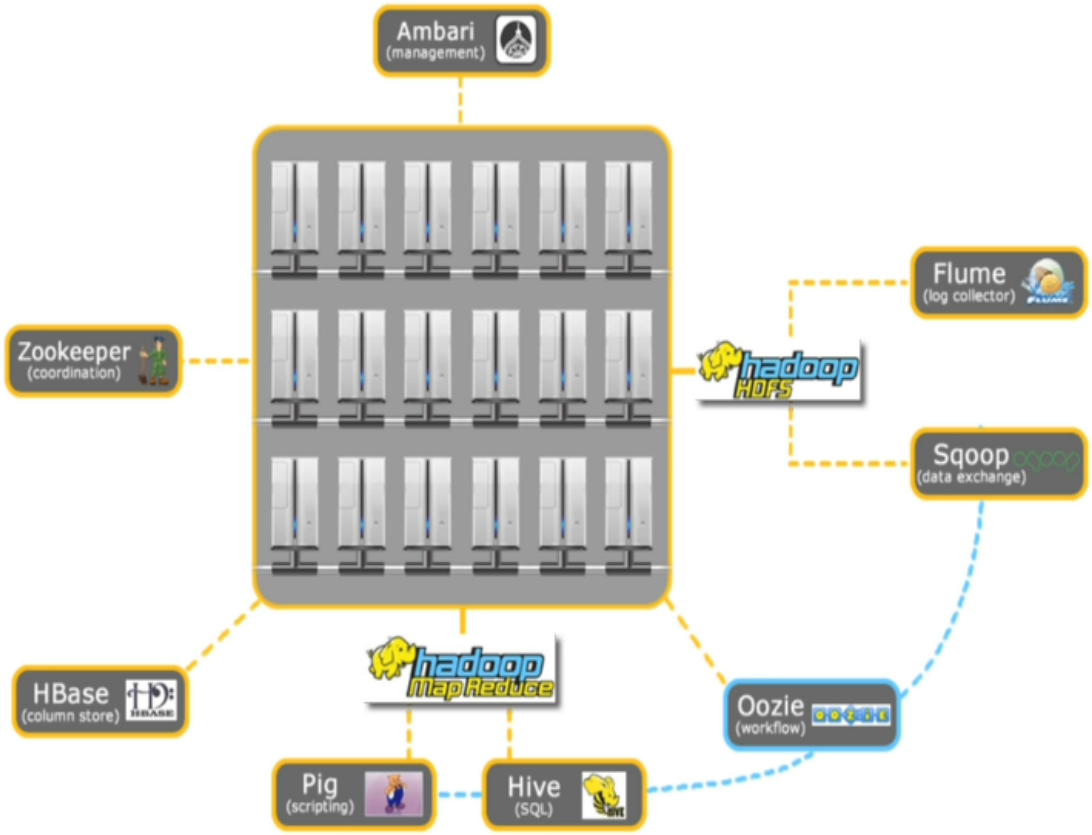
- **Data Intelligence** :L'intelligence de données conquiert trois fonctions principales: le filtrage collaboratif (recommandation engine), le clustering (groupe des textes et des documents connexes) et la classification (une manière de catégoriser des textes et des documents connexes).

- **Interaction, Visualisation,Execution, developpement** :C'est une façon qui nous permet de créer des schémas partagés et d'avoir une vue cohérente de nos données.

- **Data Integration** :L'intégration des données consiste à pousser les données hors du HDFS vers le monde relationnel (pour que les professionnels des données puissent faire leur propre analyses) et à pousser les données du monde relationnel vers hadoop (archivage).

- **Management,Monitoring,Orchestration**:Cette service est un moyen de maintenir tous les services exécutés sur le cluster synchronisés. Il nous donne également un point de gestion centralisé pour ces services.

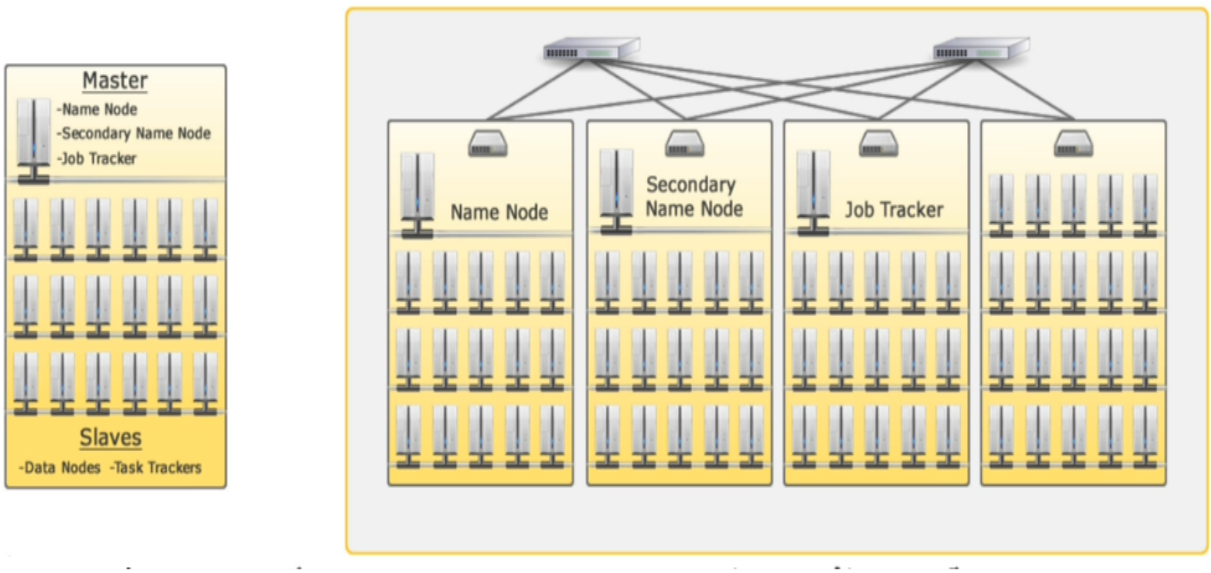
d)Hadoop Stack Implementation



Voici un exemple de base de la Stack Hadoop, essentiellement il ya Core Hadoop (MapReduce, HDFS). Et puis il ya des technologies pour travailler avec les données (les moyens facile **PIG**, **HIVE**). Et des technologies d'intégration de données, **SCOOP** si nous voulons intégrer avec base de données relationnelle, **FLUME** si nous voulons apporter nos logs à hadoop. **AMBARI** pour la gestion et le provisionnement du cluster entier. **ZOOKEEPER** pour la coordination, **Hbase** pour le stockage de données et nous devrions probablement ajouter **HCATALOG** pour avoir une vue cohérente à travers ces technologies.

e) L'Architecture HDFS

La première chose à savoir sur l'architecture HDFS est que c'est une architecture Master/Slave. L'architecture Master/Slave est dans laquelle un dispositif (Master) contrôle un ou plusieurs autres dispositifs (Slave). Le NameNode est un Daemon HDFS et est le contrôleur de tous les DataNode. Dans le monde MapReduce, nous avons "JOB Tracker" le contrôleur pour tous les "Task Trackers". Voici les rôles principaux du NameNode , Secondary-NameNode et DataNode dans HDFS.



Le **NameNode** gère toutes les opérations du système de fichiers, de sorte que toute requête qui vient dans le système de fichiers (créer un répertoire, un fichier, lire / écrire un fichier) va le parcourir. Il contient en mémoire un SnapShot de ce que ressemble le système de fichiers(-fsImage) . Il gère également le "Block Mappings",donc le NameNode sait où tous les blocs sont dans le cluster.

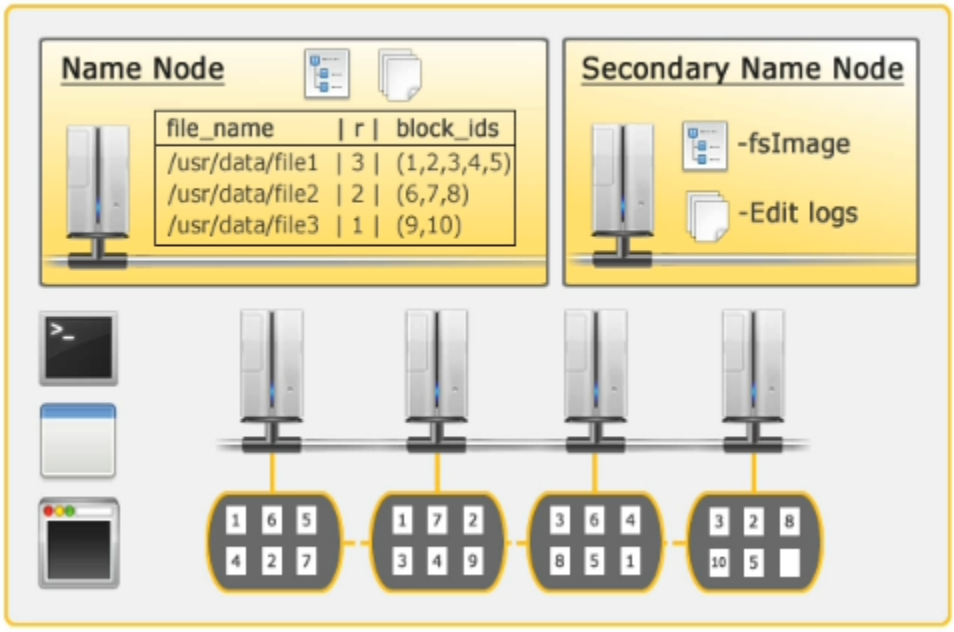
Les **Datanode** sont ceux qui effectuent réellement toute l'opération de block. Bien sûr, ils reçoivent des instructions du NameNode d'où mettre les blocs et

comment mettre les blocs. Les Datanodes sont également responsables de la réplication, encore une fois le namenode va être le contrôleur il va envoyer les instructions de l'endroit où répliquer les blocs et le datanode seront ceux qui font la réplication physique.

Le **Secondary NameNode**: Comme le nom n'implique pas, il est juste là pour prendre un instantané de la namenode de temps en temps (system restore pour le NameNode).

Dans une implémentation multi-rack, nous avons quelques défis supplémentaires qui se présentent, La première est la prévention des **pertes de données**. Comme nous l'avons dit avant HDFS est fiable en assurant qu'il ya plusieurs copies de blocs à travers le cluster. Donc ce n'est pas un gros problèmes si nous perdons un datanode. La perte de données se produit lorsque nous perdons un rack entier, grâce à la fonctionnalité de "Rack-awareness", le NameNode va s'assurer que plusieurs copies de données se trouvent sur plusieurs racks. Le deuxième défi est la performance du réseau, en général, la communication en rack interne a une bande passante beaucoup plus élevée et une latence faible que la communication externe. Alors il va s'assurer d'obtenir toutes les données du même rack.

f) HDFS Internals:



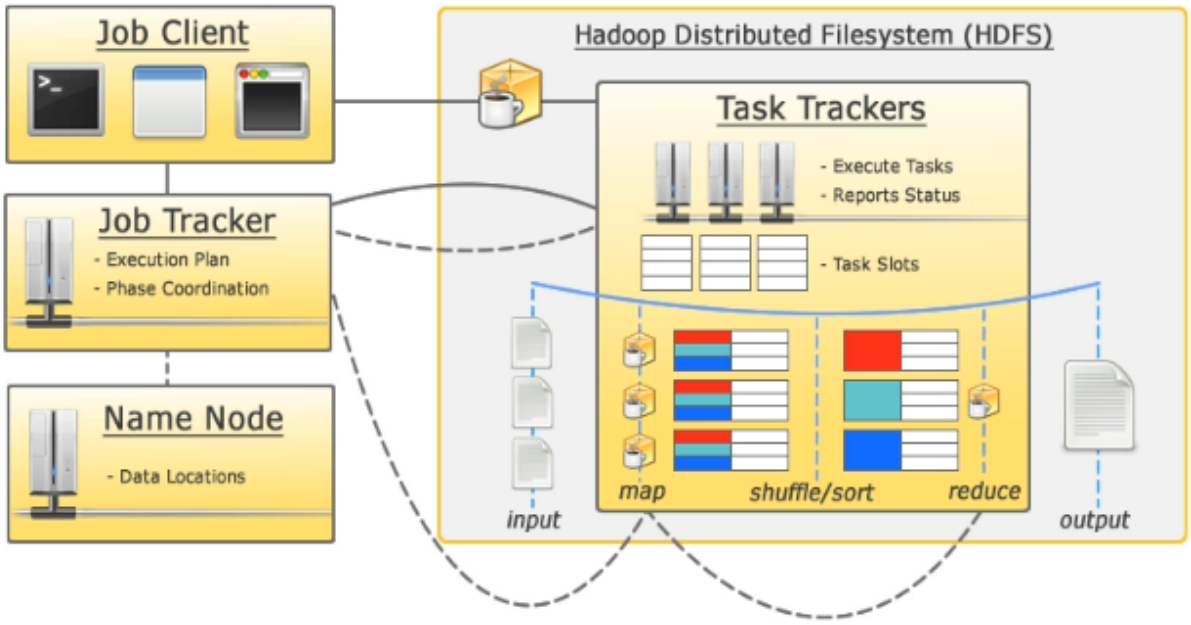
Le namenode est le noeud le plus important du cluster, car il s'agit d'un "Single point of failure". Donc si le namenode tombe en panne , le cluster est en panne. Toutes les modifications sont apportées au File System sont stockées dans le fichier "edit logs". Donc lors d'un Reboot, le namenode va prendre le fichier FSImage et le restaurer sur le disque, puis le fusionner avec le fichier "edit log".

Dans un environnement de production, nous ne redémarrage pas le namenode très souvent, ainsi le fichier "Edit Logs" continuera à augmenter. Si nous perdons le namenode dans un événement catastrophique, nous perdrons tous les changements qui se sont produits. C'est pourquoi il est nécessaire d'avoir un Secondary-NameNode.

Le **Secondary-NameNode** prend toutes les responsabilités de fusionner le «log» avec le fsImage, il va périodiquement se diriger vers le namenode et récupérer le fichier «edit logs» et le fsimage. Il va alors fusionner les deux ensemble et telecharge le fsimage de nouveau au namenode.

Le **datanode** parle au namenode. Il envoie des "Heartbeat" chaque 3 secondes au namenode. C'est ainsi que le namenode sait que le datanode est en ligne, car si le datanode n'envoie pas un "HeartBeat" après 10 minutes, le namenode va le considérer comme un nœud mort.

g) L'Architecture MapReduce:



MapReduce est un framework programmable pour extraire les données en parallèle hors du cluster. Il est une programmation de bas niveau, c'est pourquoi «HIVE» et «PIG» ont été développés parce que contrairement à MapReduce, ils sont simples et conviviale.

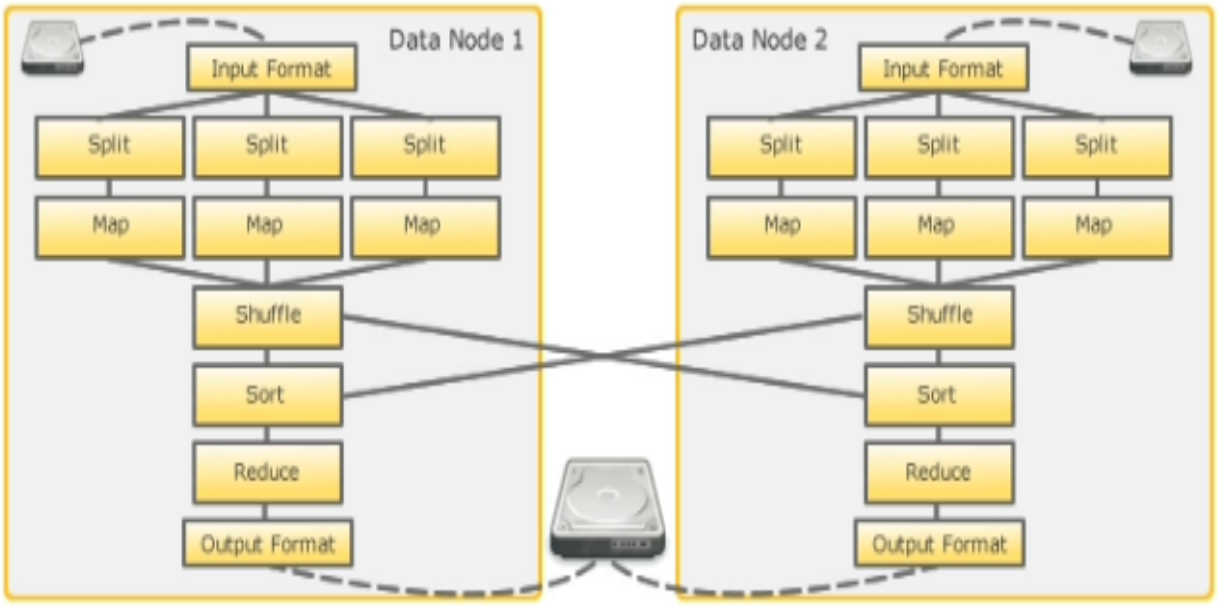
Les principaux composants de l'architecture Mapreduce sont «job Client», «JOB Tracker» et «Task Trackers».

Le «Job Client» soumet les travaux, un job contient un fichier binaire compilé (fonctions Mapper / Reducer).

La responsabilité de Job Trackers est de proposer un plan d'exécution et de coordonner et planifier les tâches à travers les Task Trackers.

Les Task Trackers exécutent les MapReduce Jobs, ils décomposent les Jobs en tâches et les placent dans des Task Slots pour les exécuter. Il signale également le progrès au Job Tracker ainsi si quelque chose échoue, le Job Tracker sera mis a jour et relance la tâche sur un autre Task Tracker.

h) MapReduce Internals:



La fonction MapReduce est constituée de 6 phases:

1. Input Format : détermine comment les fichiers sont analysés dans le pipeline. le default Input Format est le Text Input Format.
2. Split: utilise le format d'entrée pour retirer les données du disque hors de HDFS et les diviser afin qu'il puisse être envoyé au Mapper. Il divise le texte Input Format ligne par ligne. Donc si il ya un grand fichier avec de nombreuses lignes, vous pourriez avoir des milliers de Mappers en cours d'exécution simultanément

3. Map: Transforme les divisions d'entrée en paires clé / valeur sur la basé sur le code défini par l'utilisateur.
4. Shuffle/Sort: déplace les Output Map vers les réducteurs et les trie par la clé. Il va prendre en compte tous les noeuds de données qui font partie de la tâche MapReduce, qui est le partitionnement et le regroupement des données, puis le tri et l'envoyer au Reducer.
5. Reducer: regroupe les paires clé / valeur basées sur le code défini par l'utilisateur
6. Output Format : Détermine comment les résultats sont écrits dans le répertoire de sortie et le remettre dans HDFS.

i) Conclusion:

En conclusion, nous pouvons dire que Hadoop fournit une solution OpenSource pour une gestion de données puissante tout en permettant une scalabilité via l'ajout de librairies et une haute disponibilité grâce aux réplications de données.
En résumé, les avantages majeurs d'Hadoop sont :

- Prix : Hadoop est OpenSource, il permet le traitement des données de la taille de pétaoctets.
- Rapidité : Il permet un traitement parallèle des données
- Scalabilité : Etant Opensource, il permet une personnalisation illimitée avec l'ajout de librairies selon nos besoins.

2.HBase:

HDFS et MapReduce ont un désavantage en commun et qui est des résultats avec une latence élevée. Le traitement par Batch est conçu pour fonctionner avec les données entiere. Si nous voulons obtenir des données spécifiques HDFS et MapReduce lira toutes les données avant de l'obtenir. Donc HBase est conçu pour les requêtes à faible latence et l'accès en temps réel à une ligne ou à une série de lignes à l'intérieur du cluster Hadoop.

a)Qu'est-ce que HBase?

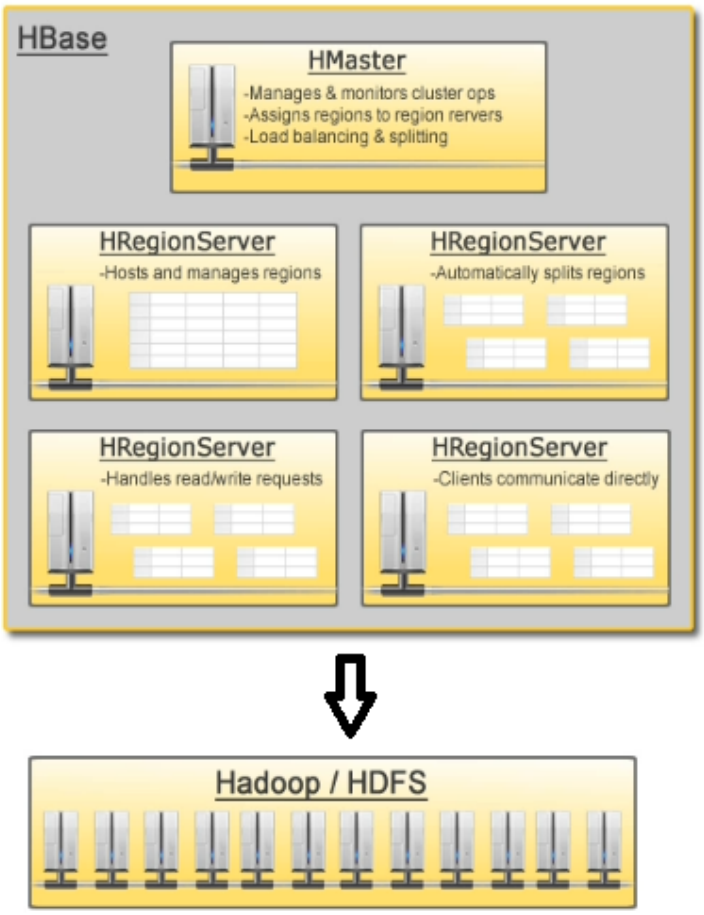
Avant de commencer avec Hbase, nous devons comprendre la différence entre les bases de données orientées par lignes et par colonnes.

Bases de données orientées lignes : Toutes les colonnes sont stockées ensemble dans une rangée sur le disque. Donc les DataStores orientées par lignes sont bonnes pour les bases de données OLTP **(On-line Transaction Processing)**, parce que chaque fois que vous envoyez une requête à une base de données orientée par lignes, il va lire la ligne entière.Et c'est pourquoi les dataStores orienté par lignes fonctionne bien avec Lecture/écriture d'une ligne. Ils sont également construits pour un petit nombre de lignes et de colonnes

Bases de données orientées Colonnes : Les banques de données orientées par colonne sont conçues pour les bases de données OLAP **(On-line Analytical Processing)**, au lieu d'avoir une ligne de tous les champs, nous avons des colonnes avec toutes les valeurs stockées dans cette colonne. La raison pour laquelle cela est bon parce que les colonnes sont stockées séparément sur le disque. Donc si vous voulez récupérer une donnée spécifique, la requête n'accédera qu'à la colonne où la valeur est située au lieu de lire la ligne entière. Vous pouvez également atteindre des taux de compression très élevés parce que la colonne va avoir un petit nombre de valeurs uniques (distinctes).

HBASE est une base de données orientée colonne qui fonctionne sur le dessus de Hadoop / HDFS.La Base de données HBase est composer de grande tables, qui vont se diviser et se répartir sur tous les nœuds du Cluster. Il sont sans schéma (structure) et n'ont aucun type de données à définir. Et parce que nous travaillons avec des DataStore orienté colonne, il n'y a pas de doublons ni de valeurs vides grace à la notion de compression.

b)L'architecture de HBase :



Comme HDFS, l'architecture HBase est une architecture Master/Slave.
Le serveur maître est comme le namenode, il va gérer et surveiller les opérations de cluster Hbase, assigner des régions à RegionServers et il est

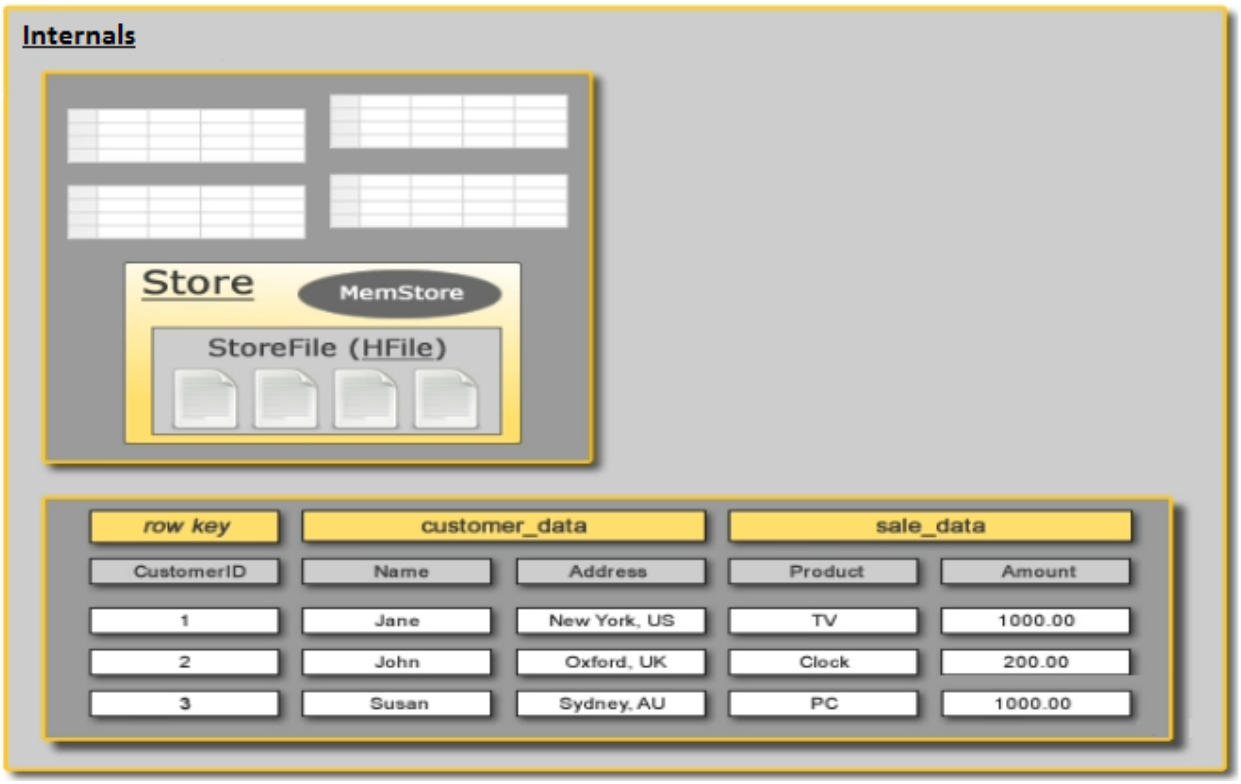
responsable de Load Balancing , de trouver où les données sont situées et de diviser les données.

Les serveurs de région sont comme des noeuds de données, ils hébergent des régions, les régions ne sont que des tables divisés et réparties entre les serveurs de régions. ils gèrent également toutes les requêtes de lecture / écriture reçues des clients.

Quand nous créons une table dans HBase et il est devenu trop grand pour un serveur de région, il serait automatiquement divisé en fonction de la clé dans le milieu et répartis de façon égale sur l'un des autres serveurs de la région dans le cluster.

Zookeeper est une grande partie de Hbase en raison de la coordination impliquée. Chaque fois que le Master Server communique avec ses serveurs de région (équilibrage de charge, fractionnement), il passe par Zookeeper. Les clients (Hive, PIG, Java API...) passent également par Zookeeper et une fois zookeeper leur donne l'accord, ils communiquent avec les Region Servers directement.

c) HBase internals :



à l'intérieur de Region Server on trouve des table et des Regions. Sous la région il ya un Store. Le Store contient 2 chose, un Memstore et zéro ou plus Hfiles (StoreFile). Les données sont situées dans des blocs à l'intérieur des Hfiles. Le Memstore stocke de nouvelles données qui n'ont pas encore été écrites sur disk. Quand le MemStore accumule suffisamment de données, il sera versé dans des HFiles.

Parce que chaque colonne est stockée séparément sur le disque, l'agrégation d'une seule colonne sera si facile. Le problème vient quand nous voulons former des enregistrements (agrégation de plusieurs colonne) , c'est pourquoi les familles de colonnes ont été créées. Les familles de colonnes nous

permettent de grouper des données similaires que nous pourrions interroger ensemble(localité des données).