BIG DATA : introduction

Modèle de données :

- Structurées (schéma prédéfini) exp : BDR
- Semi-Structurées (schéma défini ultérieurement) exp : XML
- Non-Structurées (sans schéma) exp : TEXTE MEDIA

Le big data c'est un ensemble d'outils théoriques et pratiques qui permet de stocker, traiter, visualiser des données énormes.

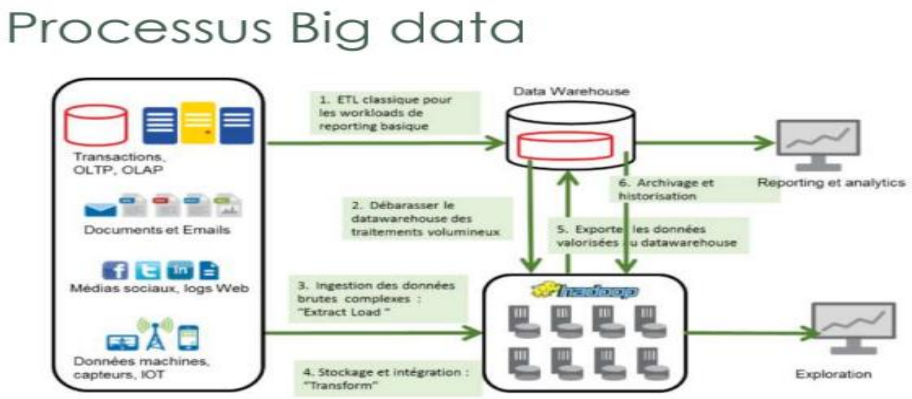
Les besoins du big data = caractéristiques = 4V : Volume, Véracité (fiabilité de l'info, est ce qu'il est correcte ou non), Variété, Vitesse=Vélocité

On trouve le big data : assurance, bourses, élections présidentielles, santé, les opérateurs téléphoniques...

Hadoop : c'est un système d'exploitation distribué (cad on dispose de plusieurs machines, rx, cluster de machine) qui permet le stockage et le traitement de données volumineuses

HDFS : utilisé pour stocker les données

MapReduce : utilisé pour traiter les données



Science de données : extraction intelligentes et efficace des connaissances à partir des big data

La Science de données englobe les activités, outils et méthodes qui permettent d'exploiter les données dans tous les domaines (science, médecine, marketing...)

BIG DATA : Chp2 : HADOOP

HADOOP : Permet le stockage et traitement de données volumineuses

Bien qu'il peut aussi bien fonctionner sur une seule machine, sa vraie puissance n'est visible qu'à partir d'un environnement composé de plusieurs ordinateurs.

Atouts de Hadoop :

- La gestion des défaillances
- La sécurité et persistance des données
- La complexité réduite : capacité d'analyse et de traitement des données à grande échelle.

- Le coût réduit : Hadoop est open source, et malgré leur massivité et complexité, les données sont traitées efficacement et à très faible coût.
- Hadoop n'est pas conçu pour des requêtes temps réel ou de faible latence.
- Hadoop est performant dans le traitement des batch hors ligne d'un grand volume de données.

Le projet Hadoop consiste essentiellement en deux grandes parties:

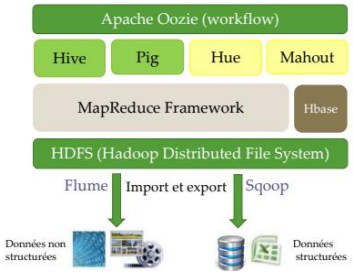
- Stockage des données : HDFS (Hadoop Distributed File System)
- Traitement des données : MapReduce

Principe :

- Diviser les données
- Les sauvegarder sur une collection de machines, appelées cluster
- Traiter les données directement là où elles sont stockées, plutôt que de les copier à partir d'un serveur distribué

Il est possible d'ajouter des machines au cluster, au fur et à mesure que les données augmentent.

Ecosystème Hadoop :



- **HDFS:** Hadoop Distributed FileSystem
- **MapReduce:** framework de traitement des données distribuées
- **HBase:** base de données Hadoop orientée colonne; supporte batch et lecture aléatoire.
- **Oozie:** Planificateur et manager du workflow Hadoop
- **Pig:** Langage de traitement de données
- **Hive:** Data warehouse avec interface SQL.
- **Hue :**
  - Front-end graphique pour le cluster
  - Fournit : Un navigateur pour HDFS et HBase et Des éditeurs pour Hive, Pig, Impala et Sqoop

- **Mahout :** Bibliothèque d'apprentissage automatique

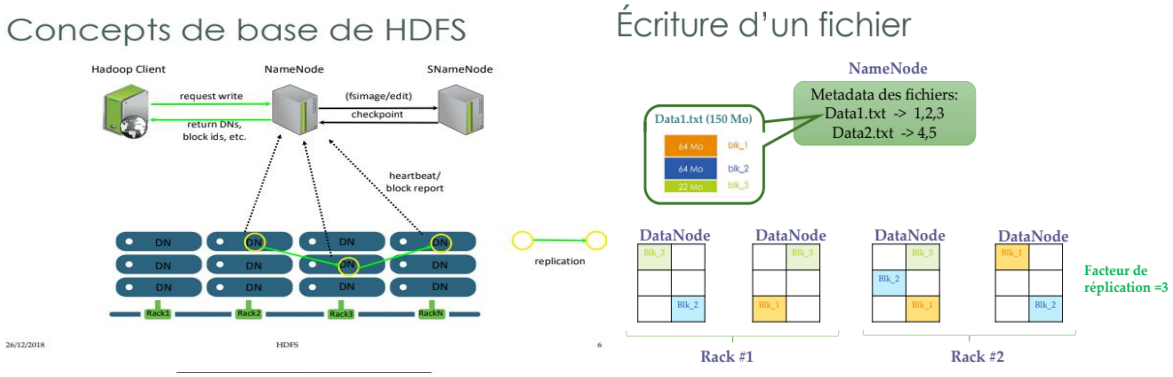
Connexion du HDFS à partir d'outils externes

- Sqoop : Prend les données à partir d'une base de données traditionnelle, et les met dans HDFS, comme étant des fichiers délimités, pour être traitées avec d'autres données dans le cluster
- Flume: Système distribué permettant de collecter, regrouper et déplacer un ensemble de données (des logs) à partir de plusieurs sources vers le HDFS

BIG DATA : Chp3 : HDFS

Objectifs de HDFS

- Garantir l'intégrité des données malgré une défaillance système
- Permettre un traitement rapide des fichiers d'une grande taille
- HDFS est moins adapté à beaucoup de fichiers d'une petite taille
- Rapprocher la lecture de la localisation des fichiers plutôt que de rapprocher les fichiers à la lecture



Les dataNodes sont des supports de stockage des blocs provenant de tous les NN

Les DN envoie un rapport à tous les NN.

NameNode : Gère l'espace de noms, l'arborescence du système de fichiers et les métadonnées des fichiers et des répertoires. Il centralise la localisation des blocs de données répartis dans le cluster.
DataNode : Stocke et restitue les blocs de données. Lors du processus de lecture d'un fichier, le NameNode est interrogé pour localiser l'ensemble des blocs de données. Pour chacun d'entre eux, le NameNode renvoie l'adresse du DataNode le plus accessible. Les DataNodes communiquent de manière périodique au NameNode la liste des blocs de données qu'ils hébergent.

Problèmes DataNode : Si l'un des nœuds a un problème, les données seront perdues

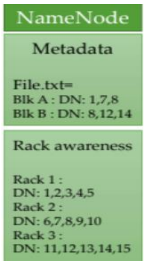
Solution :

- Hadoop réplique chaque bloc 3 fois
- Il choisit 3 nœuds au hasard, et place une copie du bloc dans chacun d'eux
- Si le nœud est en panne, le NN le détecte, et s'occupe de répliquer encore les blocs qui y étaient hébergés pour avoir toujours 3 copies stockées

NameNode : Méta data

Types de Méta-data :

- Liste de fichiers et liste de blocs pour chaque fichiers (taille par défaut 64MB)
- Liste des dataNodes pour chaque bloc
- Attributs de fichier, exemple : temps d'accès, facteur de réplication
- Fichier log
- Enregistrement de création et suppression de fichiers.



Chaque namenode contient les fichiers suivants :

- Version :Contient les informations sur le namenode.
  On y retrouve :l'identifiant pour le système de fichiers, la date de création, le type de stockage, la version de la structure des données HDFS.
- Edits :Regroupe toutes les opérations effectuées dans le système.Chaque fois quand une action d'écriture est effectuée, elle est d'abord placée dans ce fichier de logs. Le fichier edits sert un peu comme le backup
- Fsimage :C'est un fichier binaire utilisé comme un point de contrôle pour les méta-données. Il joue aussi un rôle important dans la récupération des données dans le cas d'une défaillance du système
- Fstime :stocke les informations sur la dernière opération de contrôle sur les données.

Problème NameNode :

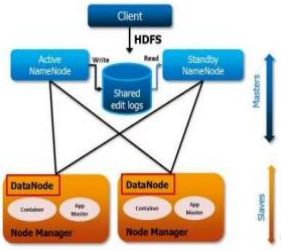
- Limité à 4,000 nœud par cluster
- Dans le cas d'un problème d'accès (réseau), les données seront temporairement inaccessibles
- Si le disque du NN est défaillant, les données seront perdues à jamais
- Dans ce cas, l'administrateur doit restaurer le NameNode, MANUELLEMENT, en utilisant le NameNode secondaire.

Namenode secondaire :

- Son arborescence de répertoire est quasi identique a celui du NN primaire.
- Il lit les méta-data à partir du RAM du NN et les écrit sur le disque dur.
- Il vérifie périodiquement l'état du NN primaire.A chaque vérification, il copie le fichier edits à partir du NN primaire à des intervalles régulier et applique les modifications au fichier fsimage.
- Le nouveau fsimage est copié sur le NN primaire, et est utilisé au démarrage suivant du NN primaire

Le namenode secondaire :

  - Ne garantit pas la haute disponibilité
  - N'améliore pas les performances du namenode



Haute disponibilité du NameNode

- Tous les fichiers edits seront stockés dans un système de fichier partagé
- A un instant données, un seul NN écrit dans les fichiers edits
- LE NN passive lit à partir de ce fichier et garde des informations des meta-data à jour
- En cas de panne de l'active NameNode, le NN passive devient le NN active et commence a écrire dans l'espace partagé.

Commandes Shell :

Toutes les commandes HDFS sont précédées par $hadoop fs - <commande> (ou $hdfs dfs - <commande>)

- cat – : afficher le contenu d'un document
  - Tout le document: $ hadoop fs -cat /dir/file.txt

- Afficher les 25 premières lignes du fichier file.txt :
    - $ hadoop fs -cat /dir/file.txt | head -n 25
- cp – : copier un fichier de HDFS vers HDFS
    - $ hadoop fs -cp /dir/file1 /otherDir/file2
- ls – : afficher le contenu d'un répertoire
    - $ hadoop fs -ls /dir/
- mkdir – : créer un chemin (EXP : $ hadoop fs -mkdir /brandNewDir)

Déplacement de données :

- mv – : déplacer un fichier
    - $ hadoop fs -mv /dir/file1 /dir2/file2
- put – : copier du système de fichier courant vers hdfs
    - $ hadoop fs -put localfile /dir/file1
    - On peut utiliser aussi copyFromLocal
- get – : copier des fichiers a partir de hdfs
    - $ hadoop fs -get /dir/file localfile
    - On peut utiliser aussi copyToLocal

Suppression de données :

- rm – supprimer des fichiers ($ hadoop fs -rm /dir/fileToDelete)
- rmr – supprimer un repetoire ($ hadoop fs –rm -r /dirWithStuff)

États des fichiers :

- du – afficher les tailles des fichiers d'un répertoire (en bytes)
    - hadoop fs -du /someDir/
- dus – afficher la taille totale d'un répertoire
    - hadoop fs –du -s /someDir/

Commande fsck :

- Contrôle les incohérences
- Reporte les problèmes(Blocs manquants/Blocs non répliqués) mais ne corrige pas les problèmes
- C'est le Namenode qui corrige les erreurs reportées par fsck.
- $ hadoop fsck <path> (Exemple : $ hadoop fsck /)

Commande DFSAdmin :

- Operations administratives du HDFS
    - $ hadoop dfsadmin <command> (Exemple: $ hadoop dfsadmin –report)
- report : affiche les statistiques du HDFS
- safemode : entrer en mode safemode (Maintenance, sauvegarde, mise a jours, etc..)

QLQ RQ :

- La configuration de Hadoop se fait à travers des fichiers XML.
- Le fichier relatif à la configuration de HDFS est : hdfs-site.xml . Ce fichier contient les paramètres spécifiques au système de fichiers HDFS dont le facteur de réplication par défaut(= 3)et la taille d'un bloc de données par défaut (= 128 Mo).

- (Notre fichier sous HDFS de taille 201 Mo sera divisé en 2 blocs de données puisque la taille d'un bloc de données par défaut est de 128 Mo.)
- Pour vérifiez l'ensemble du système de fichiers pour détecter les incohérences ou les problèmes : hadoop dfsadmin –report (donne une vue d'ensemble sur le système de fichiers (nombre de blocs corrompus, nombre de blocs manquants, etc.))
- Pour copier ou bien deplacer sous HDFS le chemin est (sous le répertoire /user/cloudera)

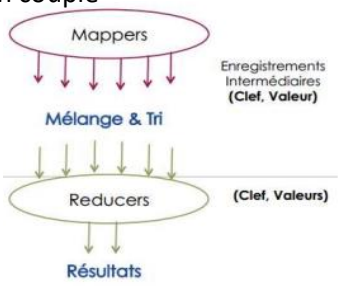
BIG DATA : Chp4 : MapReduce

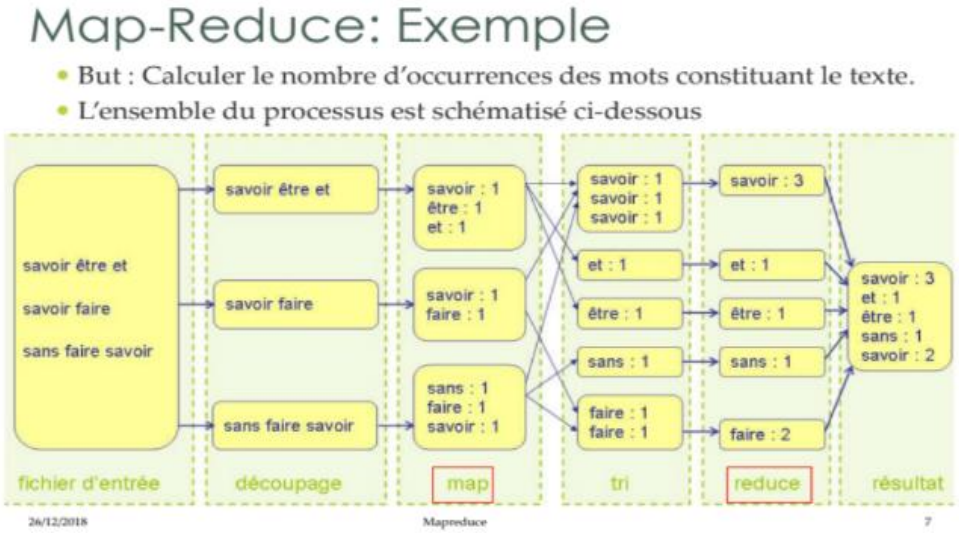
Map-Reduce :

- Permet de traiter des données volumineuses de manière parallèle et distribuée
- Les langages utilisés : Java, Python ou Ruby
- Au lieu de parcourir le fichier séquentiellement (beaucoup de temps), il est divisé en morceaux qui sont parcourus en parallèle.
- Il se base sur 2 étapes :
    - Mapping (map tasks) : le développeur définit une fonction de mappage dont le but sera d'analyser les données brutes contenues dans les fichiers stockés sur HDFS pour en sortir les données correctement formatées.
    - Réduction (reduce tasks) : consiste a la récupèration des données construites dans l'étape du mappage et les analyser dans le but d'en extraire les informations les plus importantes.

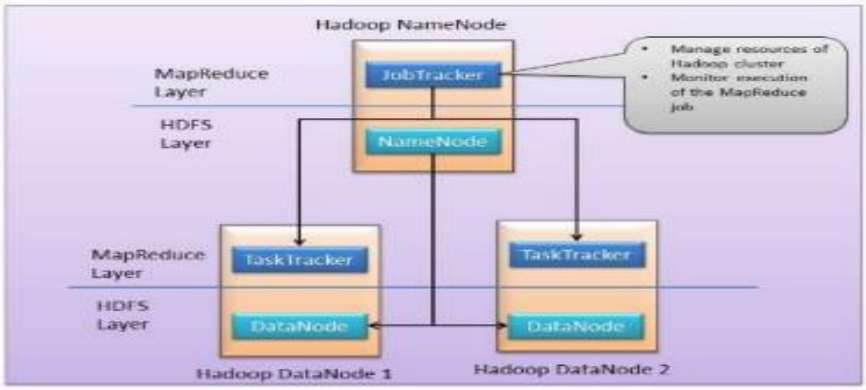
Fonctionnement :

- Les Mappers sont de petits programmes qui commencent par traiter chacun une petite partie des données.Ils fonctionnent en parallèle
- Leurs sorties représentent les enregistrements intermédiaires : sous forme d'un couple (clef, valeur)
- Une étape de Mélange et Tri s'ensuit
    - Mélange : Sélection des piles de fiches à partir des Mappers
    - Tri: Rangement des piles par ordre au niveau de chaque Reducer
- Chaque Reducer traite un ensemble d'enregistrements à la fois, pour générer les résultats finaux





Deux processus JobTracker et TaskTracker :



- **JobTracker :**
  - Planifie les taches
  - Affecte les tâches aux TaskTrackers.
  - Gère les jobs MapReduce et surveille les progrès réalisés
  - Récupère les erreurs, et redémarre les tâches lentes et les taches qui ont échoué
  - JobTracker est remplacé pas ResourceManager en MPR2
  - JobTracker parle avec le NameNode pour determiner l'emplacement des données
- **TaskTracker :**
  - Notifie périodiquement le JobTracker du niveau de progression d'une tâche ou bien le notifie en cas d'erreur afin que celui-ci puisse reprogrammer et assigner une nouvelle tâche.
  - S'exécute sur chacun des nœuds pour exécuter les vraies tâches de MapeReduce
  - TaskTracker s'exécute sur DataNode
  - TaskTracker est remplacé par Node Manager dans MRv2
  - Les tâches Mapper et Reducer sont exécutées sur les DataNodes administrés par les TaskTrackers.

JobTracker est un maître qui crée et exécute le travail. JobTracker qui peut s'exécuter sur le NameNode alloue le travail aux tasktrackers. Il suit la disponibilité des ressources et la gestion du cycle de vie des tâches, suit sa progression, sa tolérance aux pannes

TaskTracker exécute les tâches et signale l'état de la tâche à JobTracker. TaskTracker s'exécute sur DataNodes. Il a pour fonction de suivre les ordres du suivi des travaux et de mettre à jour le suivi des travaux avec son état d'avancement périodiquement.

Gestion des ressources :

MapReduce a un modèle de gestion de mémoire inflexible basé sur les slot.

- Chaque TaskTracker est configuré au démarrage pour avoir un nombre bien déterminé de slots (map slot, reduce slot) pour l'exécution des tâches
- Une tache est exécutée dans un seul slot
- Les slots sont configurés au démarrage pour avoir un maximum d'espace mémoire

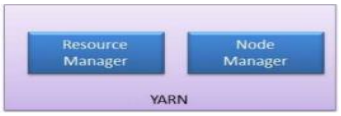
Limites de MPR1 :

- Scalabilité limitée : Le JobTracker s'exécute sur une seule machine.
    - Ses rôles sont : Gestion des ressources ,Ordonnancement et suivi des Job
- Problème de disponibilité : Le JobTracker est un SPOF. S'il est endommagé, tous les jobs doivent être redémarrés
- Problème d'utilisation des ressources : Il y a un nombre prédéfini de map slots et reduce slots pour chaque TaskTrackers.
- Utilisation des applications non-MapReduce : Le JobTracker est intégré à MapReduce et ne supporte que les applications utilisant le framework de programmation MapReduce

Nouveautés Hadoop 2 :

YARN : Possibilité de traitement de Terabytes et Petabytes de données existants dans HDFS en utilisant des application Non-MapReduce

Resource Manager :Séparation des deux fonctionnalités essentielles du jobtracker (gestion des ressources et ordonnancement et suivi des jobs) en deux composants sép



Gestionnaire de Resource et Gestionnaire d'application

Jobtracker et Tascktracker n'existent plus.

Ordonnanceur de capacité :

MPR1 : L'ordonnanceur assure un nombre bien déterminé de slots pour chaque groupe d'utilisateurs. Si on n'a pas de job mapReduce en cours, perte de ressources datanode.

MPR2 : L'ordonnanceur de capacité (Capacity Scheduler) garantit un partage de slot par tous les groupes d'utilisateurs donc maximisation de l'utilisation des datanodes

QLQ RQ :

- Le rôle du job driver est de définir le fichier jar qui va contenir le driver, le mapper et le reducer.

-Le code MapReduce peut être exécuté dans un cluster ou bien dans un IDE comme éclipse. La différence entre les deux modes d'exécution est que, dans le cas d'éclipse, le résultat est généré localement tandis que, pour l'exécution dans un cluster, ils seront générés dans HDFS.

-Lancer un job entier sur Hadoop implique qu'on fera appel au mapper puis au reducer sur une entrée volumineuse, et obtenir à la fin un résultat, directement dans HDFS.

BIG DATA : Chp5 : Langages de requête hadoop : Hive

Pig, Hive –Similarités :

- Réduire la taille des programmes java
- Les applications sont traduites en programmes MapReduce en arrière plan.
- Fonctionnalité extensible
- Interopérabilité avec les autres langages
- Non conçu pour les lecture/ écriture aléatoire ou les requêtes de faible latence.

Pig, Hive –Différences :

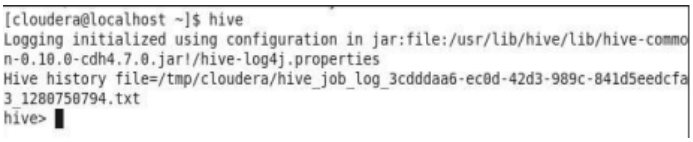
Caractéristiques	Pig	Hive
Développé par	Yahoo!	Facebook
Langage	Pig latin	HiveQL
Type de langage	Data flow	SQL
Structure de données supportée	Complex	Structuré
Schema	optionnel	Obligatoire

Hive : Solution Data Warehouse intégrée dans Hadoop qui fournit un langage de requête similaire au SQL nommé HiveQL

Traduit les requêtes HiveQL en un ensemble de jobs MapReduce qui seront exécutés dans un cluster Hadoop.

Création de base de données :

Démarrage Hive : on tapant juste hive



Création base de données :

hive> create database test;

hive> use test;

Création de table :

Création de table pour le stockage des données qui existent dans le fichier /ch5_data/user-posts.txt

- hive> CREATE TABLE posts (user STRING, post STRING, time BIGINT)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  STORED AS TEXTFILE;
- hive> show tables; **(Afficher la liste des tables)**
- hive> describe posts; **(Description de la table posts)**

Insertion données dans une table :

- hive> LOAD DATA LOCAL INPATH
  '/home/cloudera/ch5_data/hive/user-posts.txt'
  OVERWRITE INTO TABLE posts;
- hadoop fs -ls /user/hive/warehouse

Affichage des données :

- hive> select count (1) from posts;
- hive> select * from posts where user="user2";
- hive> select * from posts where time<=1343182133839 limit 2;

Suppression d'une table :

- hive> DROP TABLE posts;
- hive> exit;
- hadoop fs -ls /user/hive/warehouse

External table : External pour dire que le fichier persiste mm si on supprime la table

- hive> CREATE EXTERNAL TABLE posts
  (user STRING, post STRING, time BIGINT)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  STORED AS TEXTFILE

LOCATION '/user/cloudera/test/';

- Hive charge les fichiers dans le répertoire /user/cloudera/test/ et non pas dans le datawarhouse hive.

Partitions : Pour augmenter sa performance, Hive a la possibilité de diviser les données (en partition).

- hive> CREATE TABLE posts (user STRING, post STRING, time BIGINT)
  PARTITIONED BY(country STRING)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  STORED AS TEXTFILE;
- hive> describe posts;

Chargement de données dans des tables partitionnées :

- hive> LOAD DATA LOCAL INPATH
  '/home/cloudera/ch5_data/hive/user-posts-US.txt'
  OVERWRITE INTO TABLE posts
  PARTITION(country='US');
- hive> LOAD DATA LOCAL INPATH
  '/home/cloudera/ch5_data/hive/user-posts-AUSTRALIA.txt'
  OVERWRITE INTO TABLE posts
  PARTITION(country='AUSTRALIA');

Table partitionnée : Les partitions sont physiquement stockés dans des répertoires distincts

- hive> show partitions posts;
- $ hadoop fs –ls -R /user/hive/warehouse/posts

/user/hive/warehouse/posts/country=AUSTRALIA

/user/hive/warehouse/posts/country=AUSTRALIA/user-posts-AUSTRALIA.txt

/user/hive/warehouse/posts/country=US

/user/hive/warehouse/posts/country=US/user-posts-US.txt

Jointure :

Soit les 2 tables suivantes : posts et likes

- hive> select * from posts limit 10;
- hive> select * from likes limit 10;
- hive> CREATE TABLE posts_likes (user STRING, post STRING, likes_count INT);
- hive> INSERT OVERWRITE TABLE posts_likes
  SELECT p.user, p.post, l.nblike
  FROM posts p JOIN likes l ON (p.user = l.user);
- hive> select * from posts_likes limit 10;

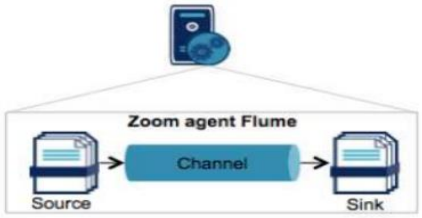
BIG DATA : Chp6 : Flume Et Sqoop

Ils permettent l'import et l'export des données

- **Sqoop** pour les bddRelationnelles
- **Flume** traite les données non structurées

- Flume est un service distribué pour assurer la collecte de données en temps réel, leur stockage temporaire et leur diffusion vers une cible.

Pour utiliser Flume, on doit lancer un agent Flume qui est un processus JVM qui héberge les composants via lesquels les événements circulent d'une source externe vers la destination.



Le « canal » ou « Channel » Flume est une zone tampon qui permet de stocker les messages avant qu'ils soient consommés.

La « cible » ou « Sink » Flume consomme par lot les messages en provenance du « canal » pour les écrire sur une destination comme HDFS par exemple.

Apache Sqoop permet de charger automatiquement nos données relationnelles de MySQL en HDFS, tout en préservant la structure.

L'importation d'une base de données de MYSQL a l'aide du sqoop dans HIVE peut etre soit sans métadonnées soit avec métadonnées (AVRO files).

Dans le scénario sans métadonnées, on se propose d'importer la totalité des la base de données dans HIVE.

Dans le scénario avec métadonnées, Sqoop a migré les donnes relationnelles vers un format binaire (Avro) tout en gardant leurs structure dans un fichier de schéma d'extension « .avsc » qui contient le schéma relationnel de la table

Il est à noter que le schéma et les données sont stockés dans des fichiers séparés (.avro et .avsc).

Le schéma est appliqué uniquement lorsque les données sont interrogées

Les schémas Avro ont été générés dans le système local contrairement aux données qui sont générées sous HDFS.

BIG DATA : Chp7 : Elasticsearch

Elasticsearch est un outil de recherche distribué en temps réel et un outil d'analyse. Il est utilisé pour l'analyse, la recherche full text et la recherche structurée. Elasticsearch permet un stockage de document temps réel distribué

Index : Il s'agit d'un espace logique de stockage de documents de même type, découpé sur un à plusieurs PrimaryShards et peut être répliqué sur zéro ou plusieurs SecondaryShards.

PrimaryShard : Par défaut, l'index est découpé en 5 ShardsPrimary. Il n'est pas possible de changer le nombre de partitions après sa création.

SecondaryShard : Il s'agit des partitions répliquées. Il peut en avoir zéro à plusieurs par PrimaryShard.

ElasticSearch se base sur un stockage de données orienté document.

Un document correspond à un simple enregistrement dans un shard ElasticSearch. Le contenu de chaque document est indexé. Les types sont contenus dans un index.

BDR	ElasticSearch
Base de données	Index

2

annoufi & Ines Slimene Esprit 201

a Analytics

Tables	Types
Lignes	Documents
Colonnes	Champs

Ajout d'un nouveau document :

- POST /biblio/livres/2
  {
  "title":"HDFS",
  "description":" Data Storage in Hadoop ",
  "category":"Hadoop",
  "tags":["HDFS","Hadoop","Cloudera"]
  }

Pour récupérer le document : GET /biblio/livres/2

Lister tous les documents d'un index : GET /biblio/livres/_search?pretty

Modification du document dont l'id est 2:

- POST /biblio/livres/2/_update
  { "doc" : {
  "tags":["big data","HDFS","Hadoop","Cloudera"]
  }}

Recherche d'un document qui contient le tag « big data »:

- GET /biblio/livres/_search
  { "query":{
  "match":{"tags":"big data"} }

Pour afficher la liste des index crées : GET /_cat/indices?v

Pour supprimer un index : delete /biblio/

BIG DATA : Chp8 : Spark

Spark c'est un ecosystéme unifié pour le traitement d'un grand volume de données structurées ou non structurées.

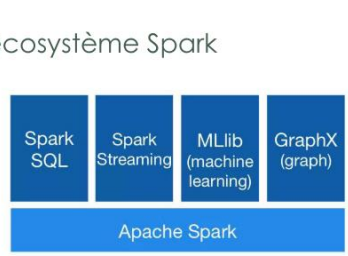
Hadoop et Spark :

- Spark complementaire a l'ecosystéme Hadoop
- Remplace Mapreduce et il est plus rapide qu'Hadoop
- Il utilise la mémoire pour traiter les données par contre hadoop fait le traitement sur disque
- Il est capable de travailler sur disque aussi
- Spark est écrit en Scala et s'exécute sur la machine virtuelle Java (JVM)
- Il est ecrit en scala mais il supporte le dvp sur d'autre langage

L'écosystème Spark :

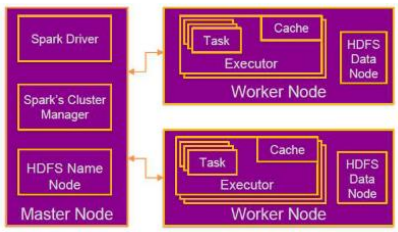
(RDD c'est la structure fondamentale sur laquelle repose le traitement des données via spark qui existe dans le cœur spark qui est apache spark)

- Spark sql : traite les données structurées
- Spark streaming : traitement temps-réel des données en flux.
- SparkMLlib : MLlib est une librairie de machine learning qui contient tous les algorithmes de machine learning.
- Spark GraphX : Traiter les données graphes



Spark est distribué car il traite les données volumineuses

L'architecture :



Un cluster spark à un seul Master et un nombre indéterminé de slaves/Workers.

Spark context : c'est le point d'entré de n'importe quelle application spark, il doit etre instancié a chaque application

SparkContext permet d'établir une connexion avec le cluster manager.

Peut être utilisé pour créer les RDD et les accumulateurs, et diffuse les variables dans le cluster.

Il est recommandé d'avoir un seul SparkContext active par JVM, donc on doit appeler la méthode stop() du SparkContext active avant de créer un autre.

Spark driver :

- lancer le spark context
- planifie l'exécution des job
- enregistre les métadatas des RDD
- contient plusieurs composants qui sont responsables de la traduction du code spark en job (ensemble de taches appelés tasks) qui sera exécute dans le cluster

Spark's Cluster Manager (eq a jobTracker en mapReduce) : responsable de l'allocation des ressources aux job lancés par le drivers.

L'executor c'est l'eq a taskTracker : responsable de l'exécution des taches qui lui est assigné par le cluster manager (dans le worker node on peut avoir plusieurs executor)

L'executors travaille indépendamment dans chaque tache et peuvent interagirent ensemble.

Worker node : est un nœud qui permet d'exécuter un programme dans le cluster. Si un process est lance pour une application, l'application requiert des executors dans les nœuds workers

Exécution d'un job spark

- Quand un client valide un code spark, le **driver traduit le code** qui contient des actions et des transformations en un graphe acyclique direct logique (DAG).
- A cette étape, le **driver optimise l'ordre de déroulement des transformations** et convertit le DAG logique en plan d'exécution physique
- Le driver **crée des petites unités physique d'exécution** (les tasks) pour chaque étape.
- Le driver **négocie les ressources** avec le cluster manager
- Le cluster manager **lance les executors** dans les nœuds workers au nom du driver.
- A ce moments, le driver **envoi les taches au cluster manager** en ce basant sur l'emplacement des données.
- Avant l'exécution des taches, ils s'enregistrent avec le driver pour qu'il puissant les surveiller.
- Le Driver planifie aussi les future taches en se basant sur leurs emplacement en mémoire.
- Quand la méthode main du programme driver se termine ou si on lance la méthode stop () du Spark Context, le driver terminera tout les executors et libère les ressources du cluster manager.

RDD : Resilient Distributed Dataset :

- Un dispositif pour traiter une collection de données par des algorithmes parallèles robustes.
- Un RDD ne contient pas vraiment de données, mais seulement un traitement qui n'est effectué que lorsque cela apparaît nécessaire. On appelle cela l'évaluation paresseuse.
- Variables partagées entre des traitements et distribuées sur le cluster de machines.
- Spark fait en sorte que le traitement soit distribué sur le cluster, donc calculé rapidement, et n'échoue pas même si des machines tombent en panne.
- RDD utilise des opérations mapreduce qui permettent de traiter et de générer un large volume de données avec un algorithme parallèle et distribuée.
- On peut charger les données a partir de n'importe quelle source et la convertir en RDD et les stocker en mémoire pour calculer les résultats.
- RDD est composé d'un ensemble de partitions. Une partition est une division logique de données qui est créée suite a des transformations d'autre partition existante.

- En cas de perte de partition RDD, on peut reprendre les transformations sur le RDD d'origine au lieu de répliquer les données sur plusieurs nœuds.

RDD : Calcul :

- Transformations : Comme avec MapReduce, chaque ligne du fichier constitue un enregistrement. Les transformations appliquées sur le RDD traiteront chaque ligne séparément. Les lignes du fichier sont distribuées sur différentes machines pour un traitement parallèle. Elles créent un nouveau RDD à partir d'un existant
- Actions : Ce sont des méthodes qui s'appliquent à un RDD pour retourner une valeur ou une collection..
  EXP :
  - RDD.collect() retourne le contenu du RDD
  - RDD.count() retourne le nombre d'éléments
  - RDD.first() retourne le premier élément
  - RDD.take(n) retourne les n premiers éléments.
  - RDD.persist() ou RDD.cache() Sauvegarder le RDD en mémoire avant l'execution (action)

Un RDD peut être sauvegardé :

- sous forme de fichier texte avec **saveAsTextFile(path)**
- sous forme de SequenceFile Hadoop avec **saveAsSequenceFile(path),**
- dans un format simple en utilisant la sérialisation Java avec **saveAsObjectFile(path).**

Transformations :

Chacune de ces méthodes retourne un nouveau RDD à partir de celui qui est concerné.

- **RDD.map(fonction)** chaque appel à la fonction doit retourner une valeur qui est mise dans le RDD sortant.
  - val longueursLignes = texteLicence.map(l => l.length)
- **RDD.flatMap(fonction)** chaque item du RDD source peut être transformé en 0 ou plusieurs items ; retourner une séquence plutôt qu'un seul item.
- **parallelize()** partitionner le RDD automatiquement à partir des caractéristiques du cluster sur lequel les calculs doivent être réalisés.
  - val RDD = sc.parallelize(Array(1,2,3,4))
- **RDD.filter(fonction)** la fonction retourne un booléen.
  - linesfilter = texteLicence.filter(line => line.contains("Komal"))

Transformations ensemblistes :

Ces transformations regroupent deux RDD

- RDD.distinct() : retourne un seul exemplaire de chaque élément.
  RDD = sc.parallelize(Array(1, 2, 3, 4, 6, 5, 4, 3))
  RDD.distinct().collect()
- RDD1.union(RDD2) : contrairement à son nom, ça retourne la concaténation et non pas l'union des deux RDD.
- Rajouter distinct() pour faire une vraie union.
  RDD1 = sc.parallelize(Array(1,2,3,4))
  RDD2 = sc.parallelize(Array(6,5,4,3))

RDD1.union(RDD2).collect()

- RDD1.intersection(RDD2) : retourne l'intersection des deux RDD. RDD1.intersection(RDD2).collect()

Transformations de type jointure :

Spark permet de calculer des jointures entre RDD1={(K1,V1). . . } et RDD2={(K2,V2). . . } et partageant des clés K identiques.

RDD1.join(RDD2) : retourne toutes les paires (K, (V1, V2)) lorsque V1 et V2 ont la même clé.

- RDD1 = sc.parallelize(Array((1,"tintin"),(2,"asterix"),(3,"spirou")))
- RDD2 = sc.parallelize(Array((1,1930),(2,1961),(1,1931),(4,1974)))
- print RDD1.join(RDD2).collect()

SparkSQL :

DataFrames : des tables SparkSQL : des données sous forme de colonnes nommées.

RDDSchema : la définition de la structure d'un DataFrame. C'est la liste des colonnes et de leurs types.

Dans l'API SparkSQL les classes : DataFrame représente une table de données relationnelles ;Column représente une colonne d'un DataFrame et Row représente l'un des n-uplets d'un DataFrame

sqlContext :

sqlContext représente le contexte SparkSQL.

Spark SQL fournit SQLContext afin d'encapsuler les fonctions du monde relationnel dans Spark.

Classe DataFrame : définit des méthodes à appliquer aux tables.

Un DataFrame est une collection de données distribuées, organisées en colonnes nommées

Il peuvent être converties en RDD et peuvent être créées à partir de différentes sources de données.

Méthodes de DataFrame :

- df = sqlContext.read.json("path") lecture fichier json
- Show() : afficher le contenu du fichier
- df.printSchema() description schéma du dataframe
- df.select(champ) afficher un champ bien déterminé
- df.filter(condition) retourne un nouveau DataFrame qui ne contient que les n-uplets qui satisfont la condition.
- resultat = achats.filter($"montant" > 30.0)
- distinct() retourne un nouveau DataFrame ne contenant que les n- uplets distincts
- join(autre, condition, type) fait une jointure entre self et autre sur la condition
- Pour classer dans l'ordre décroissant, il faut employer la fonction sort(desc(colonnes)) et sort(colonnes) pour l'ordre croissant