Cours1 Hadoop Ecosystème

Master2 Management et Sciences de Données

Mr DIATTARA Ibrahima
Architecte Solution Big Data

Sommaire

OVIEW: Définitions

Big Data : Principe des « 4V » (Volume, Variety, Velocity et Valeur)

Le Big Data est une nouvelle génération de technologies et d'architectures pour extraire de la valeur à partir de très large volumes et variétés (+sources , +types(structurées , semi structuré et non structurées) de données avec une grande vélocité de collecte , de transformation et d'analyse

Data-Lake est une zone logique de données ou se déverse l'ensemble des données de l'entreprise pour répondre aux besoins dits Big-Data (agrégation, corrélation, traitement visualisation ..).

Data warehouses: Les données ne sont pas chargées dans l'entrepôt de données tant que leur utilisation n'a pas été définie.

Cluster: ensemble de machine avec master et une ou plusieurs esclaves

Datacenter :lieu où se trouvent différents équipements électroniques, des ordinateurs, des systèmes **de** stockage (rack, …)

Rack/Baie une armoire technique qui centralise des éléments (Disk, …)

Introduction

Hadoop se caractérise en version 1 par l'implémentation d'un ***filesystem distribué (HDFS)*** couplé à un mécanisme de parallélisassions des algorithmes (***MapReduce)***

Objectifs HDFS

Historique

- Besoins de Google :
  - Stocker et traiter des Peta octets de données sur des milliers de noeuds
  - Tolérante aux défaillances
- 2003 : Développement chez Google :
  - Google FS : Système de fichiers distribué et Tolérant aux pannes
  - MapReduce : Paradigme de programmation
- 2006 : Développement d'une solution libre par Apache
  - Hadoop FS : Système de fichiers distribué proche de Google FS
  - Hadoop = Hadoop FS + MapReduce

2 Objectifs :

- **Tolérance aux pannes (hardware et software)**

  !Utilisation matériel conventionnel
- **Gestion des fichiers de grande taille** (Accès rapide)

!Exemple : Yahoo! Utilise Hadoop sur 4000 nœuds pour stocker des petabytes de données

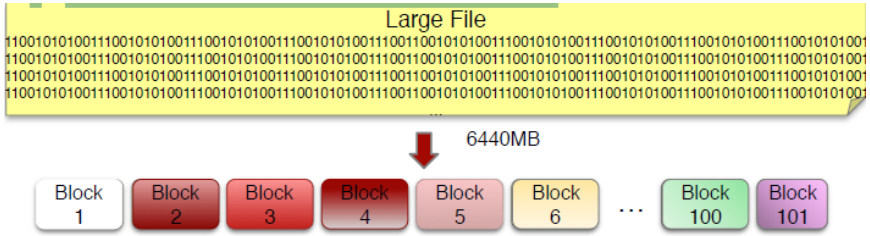
• Fichiers découpés en blocs

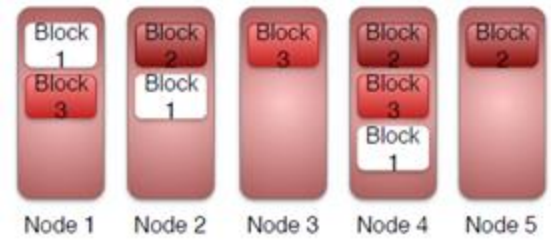
-Permet le stockage distribué de gros fichiers

-Répliquer les blocs pour les tolérances de panne et partager la charge entre les DataNodes

• Taille bloc : 64 Mo (par défaut)

• 128 Mo dans beaucoup d'installations

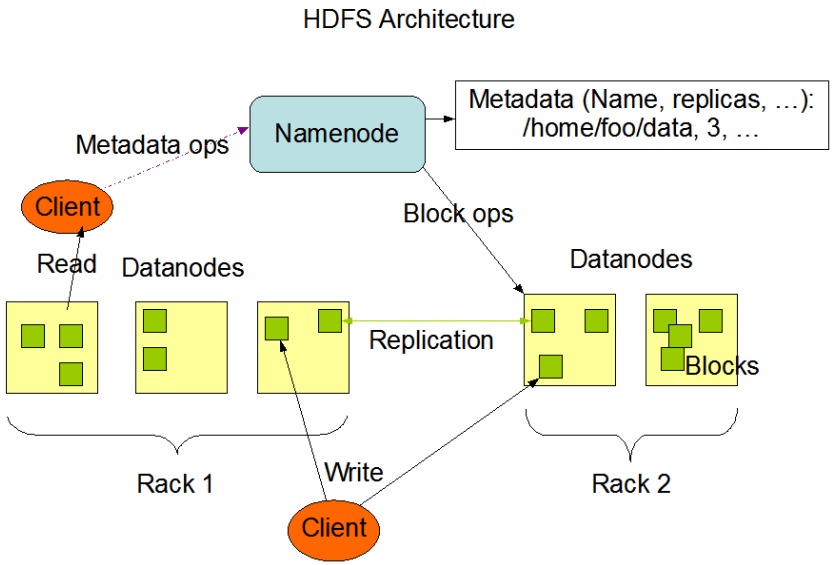




Compromis entre fiabilité et bande passante

- Réplication sur noeuds de mêmes Datacenter/Rack => meilleure bande passante mais aucune tolérance optimale aux pannes
- Réplication sur Datacenter/rack différents =>bonne tolérance aux pannes mais faible bande passante

HDFS: Nodes



NameNode(Maitre) : Est le maitre du cluster on a un seul Namenode par cluster, il a pour rôle:

1. Contrôle l' activité des DataNodes
2. Contient 2 type de fichiers : Namespace((FsImag) et Logs (editlog) :

- namespace: Maintient arborescence et meta-données
- Editlog: Contient les actions réalisées sur le cluster (read, delete, …)

Backup Node(Adjoint Maitre) : Sauvegarde l'état du NameNode

DataNodes(Slave) :Stockage des fichiers et exécution des tâches, Informent périodiquement le NameNode, blocs stockés avancement des tâches)

La lecture de data est réalisée sur le DatNnode guidé par le NameNode qui indique les meilleurs DataNodes à contacter

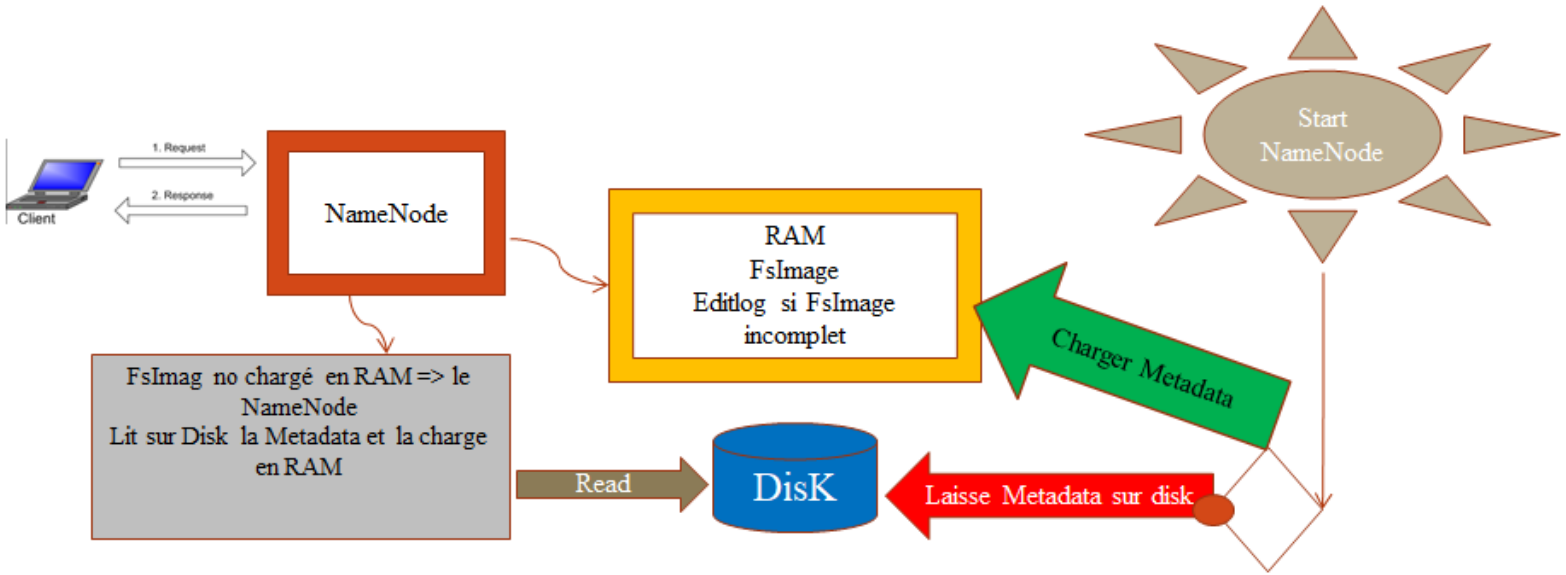
HDFS: FsImage & Edit log

Toutes les modifications (renommer, supprimer, changement d'autorisation) seront stockées dans **EditLog.**

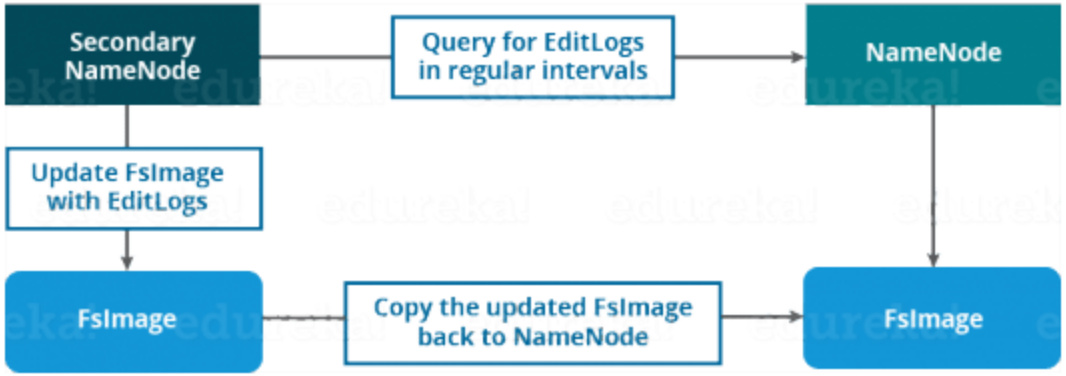
Les modifications d' Editlog sont ensuite fusionnées, afin de créer un nouveau fichier **FsImage** (emplacement blocs , ……)

NameNode stocke les métadonnées en **mémoire RAM,** afin de servir les demandes client multiples le plus rapidement possible.

Si cela n'est pas fait, alors pour chaque opération, le NameNode doit lire les métadonnées sur **Disque** et les écrire sur la mémoire



Backup Name Node



Le Second NameNode lit les métadonnées de la RAM du NameNode et les écrit sur le disque

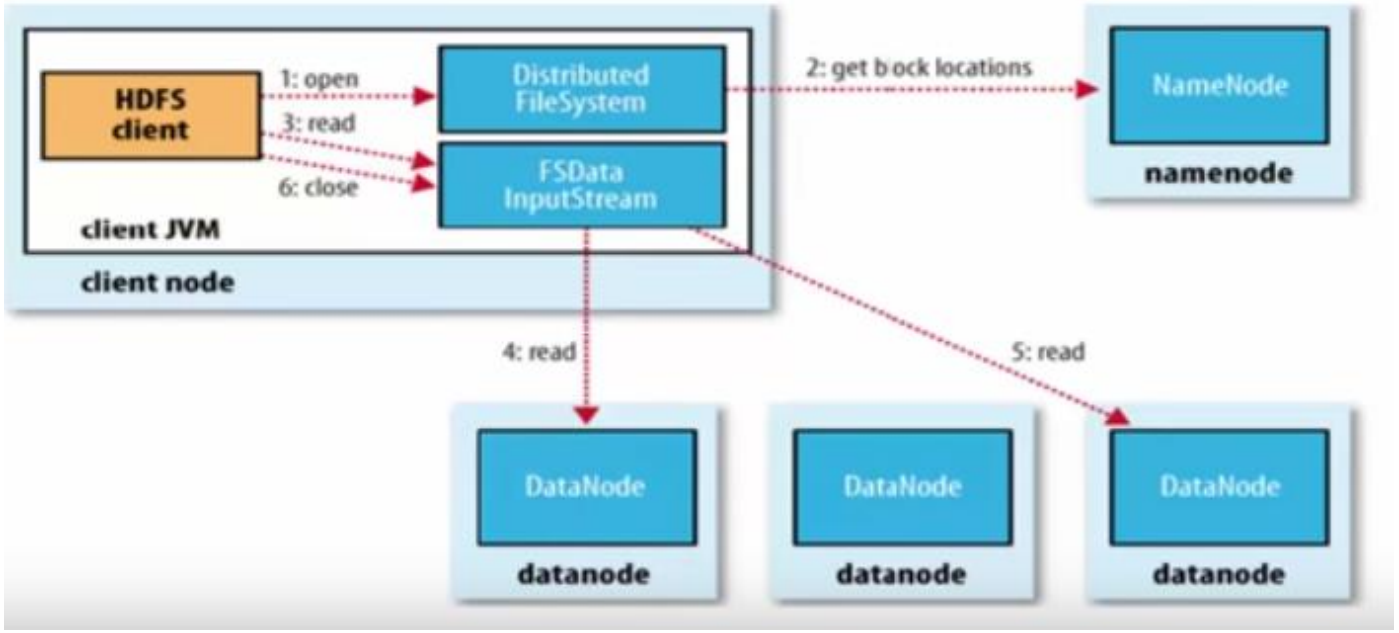
Il télécharge les EditLogs à partir du NameNode à intervalles réguliers et s'applique FsImage

La nouvelle FsImage est recopiée dans le NameNode, qui est utilisé chaque fois que le NameNode est démarré la prochaine fois

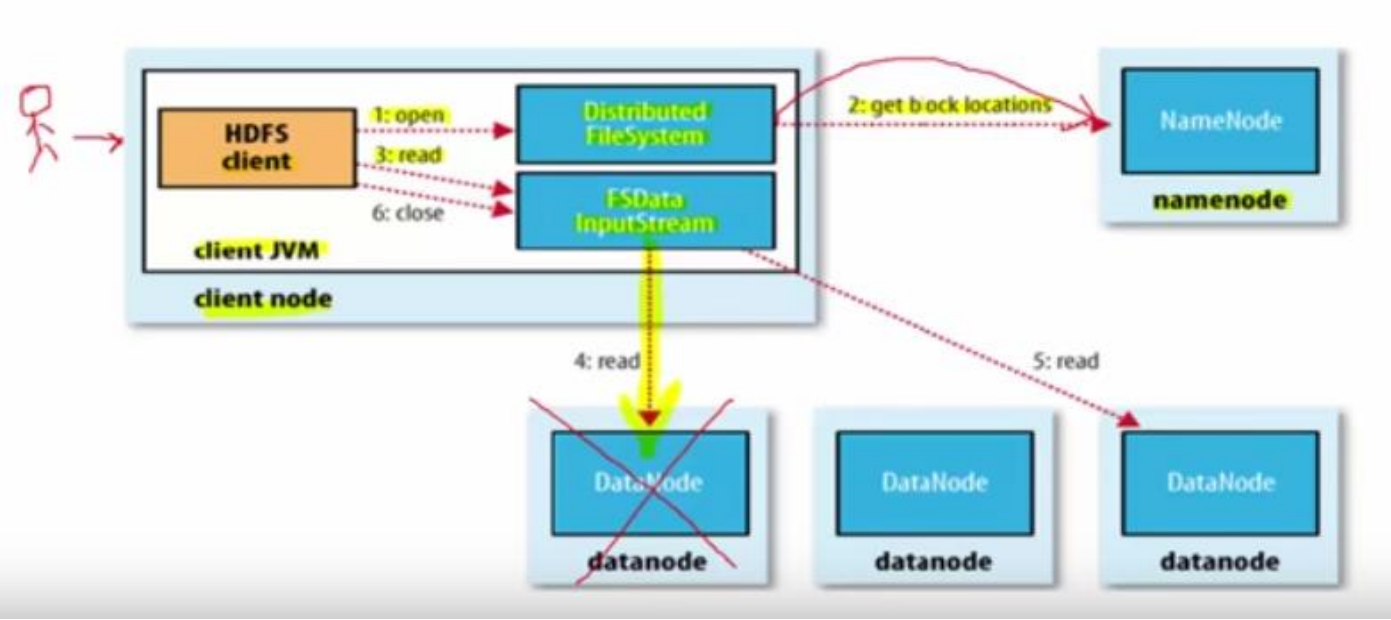
Read

Check ACL client

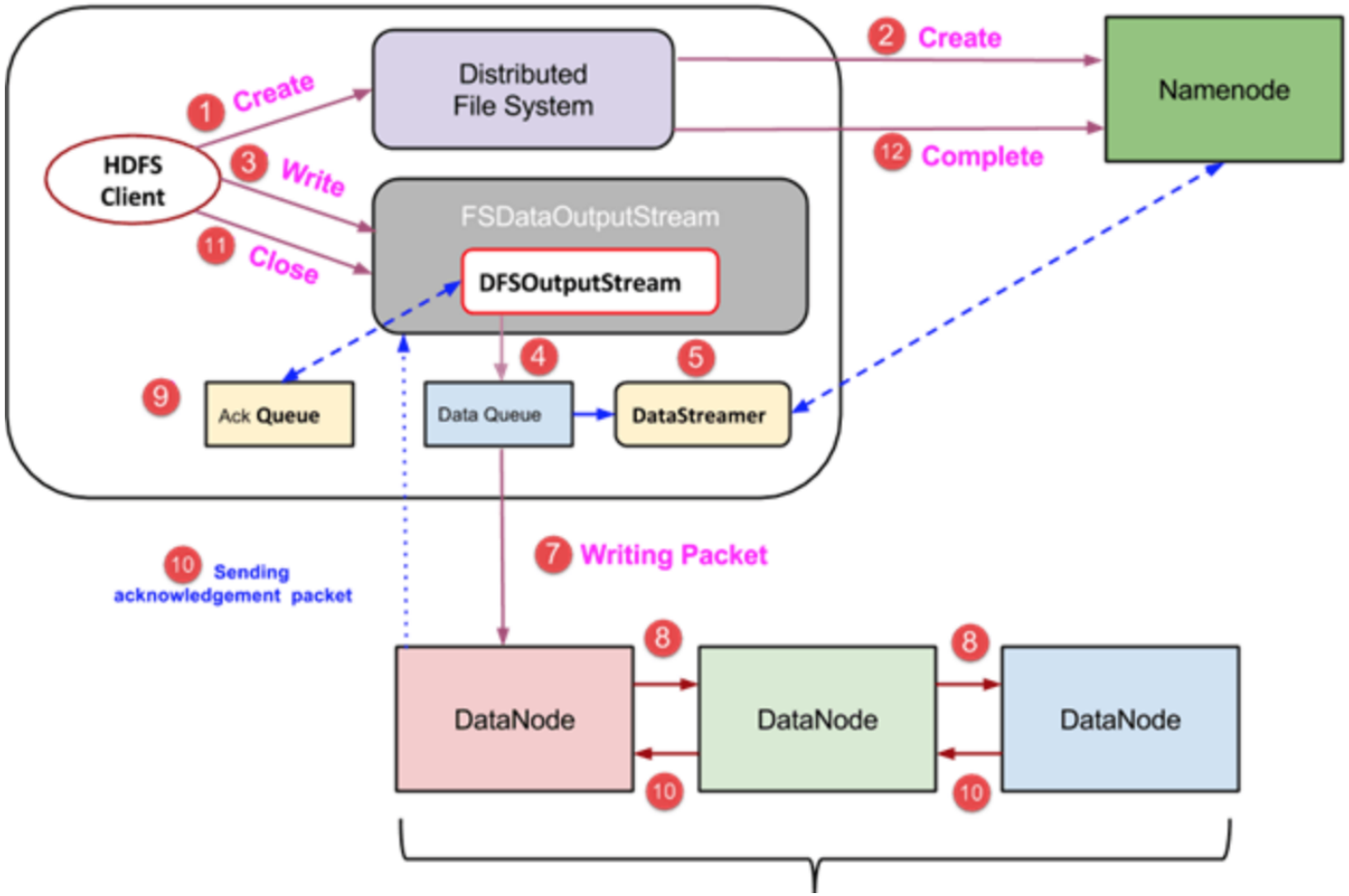
Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …



When a DataNode goes down while reading



Write


Distributed File System
Namenode
HDFS Client
1 Create
2 Create
3 Write
11 Close
12 Complete
FSDataOutputStream
DFSOutputStream
4
5
9
Ack Queue
Data Queue
DataStreamer
7 Writing Packet
10 Sending acknowledgement packet
8
8
DataNode
DataNode
DataNode
10
10

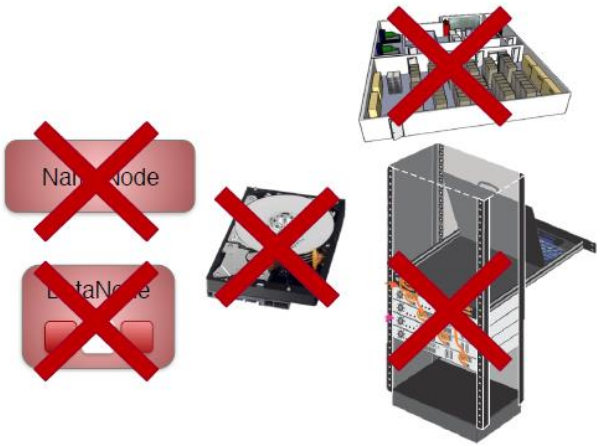
HDFS: Zookeeper

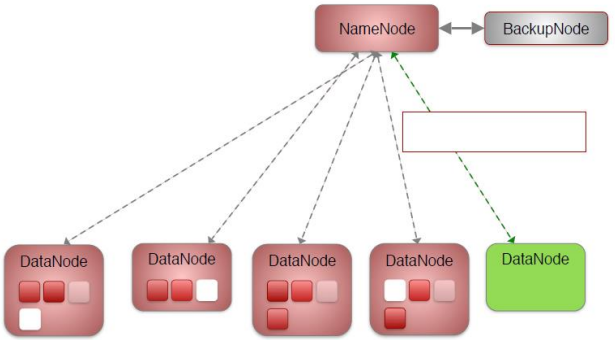
Le mécanisme de bienveillance du NameNode se repose sur ZooKeeper :

Le Zoo a deux rôles dans ce contexte :

- **Détection d'échec** : chacune des machines NameNode(primary, secondery1, ….. seconderyN) du cluster maintient une session persistantée dans ZooKeeper. Si la NameNode Primary tombe en panne, la session ZooKeeper expirera, informant aux NameNode Secondery qu'un basculement doit être déclencher.
- **Election** : ZooKeeper fournit un mécanisme simple pour élire exclusivement un nouveau NameNode Primary .

Exemple Thiès vote Dakar, Dakar vote Thiès et Kébémer vote Thiès => Thiès devient le NameNode





!Warning :HDFS

HDFS n'est pas fait pour les utilisations ci-dessous

– Accès aux données avec faible latence

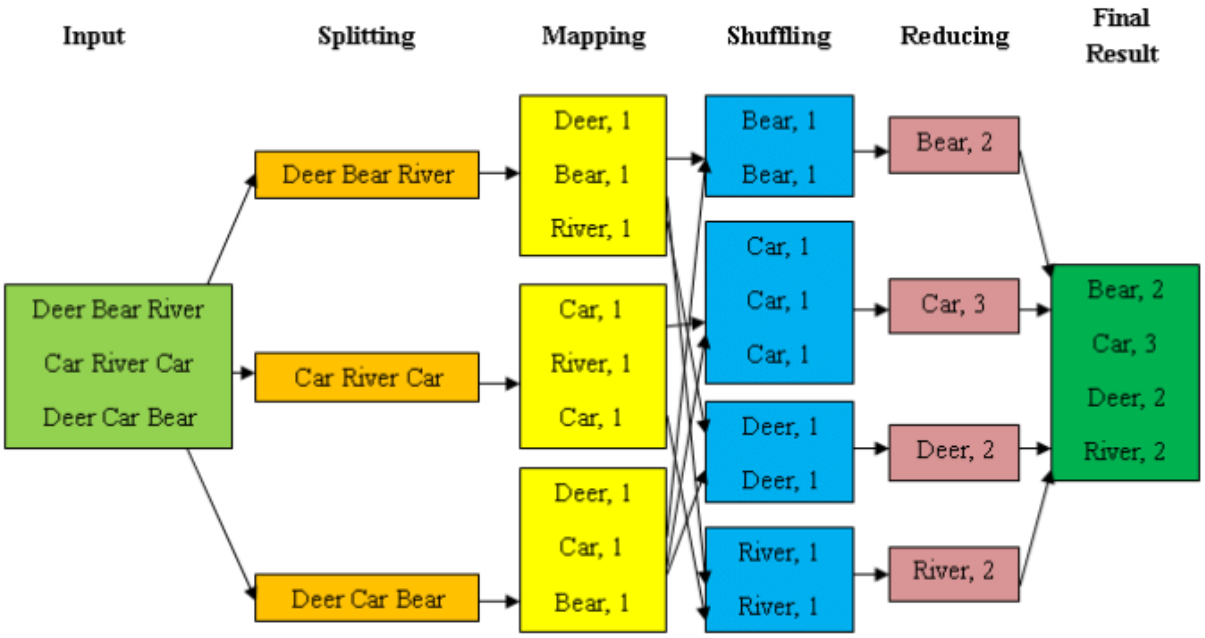
– Grand nombre de petits fichiers

Métadonnées des fichiers chargés en mémoire, Bloc not full use ! (jusqu'à plusieurs dizaine de millions de fichiers par noeud)

– Ecritures arbitraires et multiples(Lock)

HDFS conçu pour écrire uniquement fin de fichiers(append) et même le happend est couteux

Principe MapR
Le Shuffling se base sur des clés



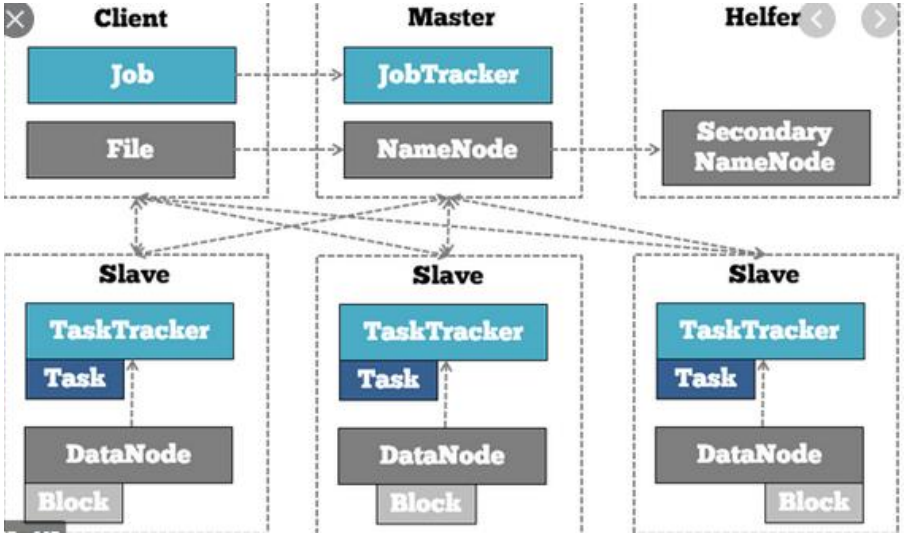

```
import org.apache.spark._
val text = sc.textFile("C:/COURS_BIG_DATA/MSD/test")
val counts = text.flatMap(line => line.split(",")).map(word => (word,1)).reduceByKey(_+_)
```

- **JobTracker (un seule sur le NameNode):** coordonne et suit tous les *jobs*

  -Ordonnance de l'exécution des *jobs* sur des *Tasktrackers*

  -Si un job échoue(panne, ..) , il le relance sur une autre *Tasktracker*

- ***TaskTrackers*** : exécuter les tâches
  - Nombre : dépend des ressources (CPU, ..)
- **L'assignement** des tâches au TaskTracker dépend:
  - Slot disponible(core CPU)
  - Plus proche des données



Ops HDFS

- Core-site.xml

<value>hdfs://hostname:port</value>

……

- hdfs-site.xml

```
<configuration>
        <property>
            <name>dfs.replication</name>
            <value>3</value>
            <name>dfs.name.dir</name>
            <value>file:///opt/hadoop/hdfs/namenode</value>
            <name>dfs.data.dir</name>
            <value>file:///opt/hadoop/hdfs/datanode</value>
        </property>
</configuration>
```

- ……

Les commandes les plus utilisées :

- **Créer un dossier dans HDFS :**

  $ hadoop fs -mkdir Exemple :

  $ hadoop fs -mkdir /user/monDossier

  $ hadoop fs -mkdir /user/monDossier1 /user/monDossier2 /user/monDossier3

  L'option -p est nécessaire si le dossier parent n'existe pas lors de la création d'un sous répertoire.

- **Lister le contenu d'un dossier:**

  $ hadoop fs -ls Exemple :

  $ hadoop fs -ls /user hadoop fs -ls /user/monDossier

- **Charger un ou plusieurs fichiers du local à HDFS:**

  $ hadoop fs -put Exemple :

  $ hadoop fs -put /home/monFichier.txt /user/monDossier

- **Exporter un ou plusieurs fichiers de HDFS au local:**

  $ hadoop fs –get Exemple :

  $ hadoop fs -get /user/monDossier/monFichier.txt

https://www.formation-bigdata.com/les-commandes-hdfs/

HDFS: Problématique: Accès Ponctuel

Dans l'écosystème Hadoop, le stockage des données se fait sur le système de fichier HDFS. L'accès à une donnée ponctuelle stockée dans un Datafile (fichier), revient à faire un « full scan » sur une partie du cluster voir même sur l'ensemble des nodes ce qui est une opération coûteuse en temps et en calcul.

Pour résoudre ce problème différentes solutions sont proposées sur la nouvelle version de Hadoop

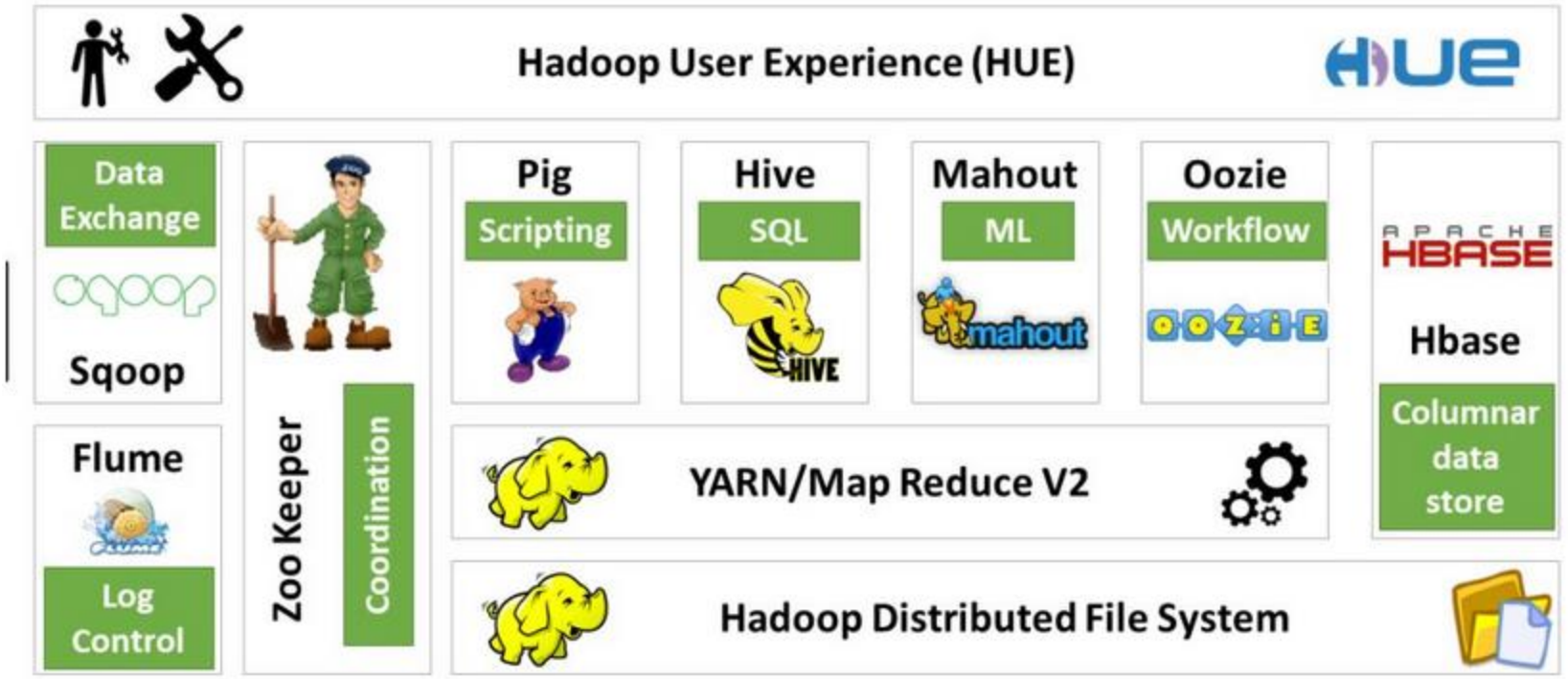
Hbase: (Base Nosql orientée colonne

Pig

Hive **Apache Hive** est une infrastructure **DataWarehouse**

Par ailleurs d'autres composants ont été rajoutés dans l'écosystéme Hadoop, afin de répondre aux demandes métiers

Ecosystème Hadoop

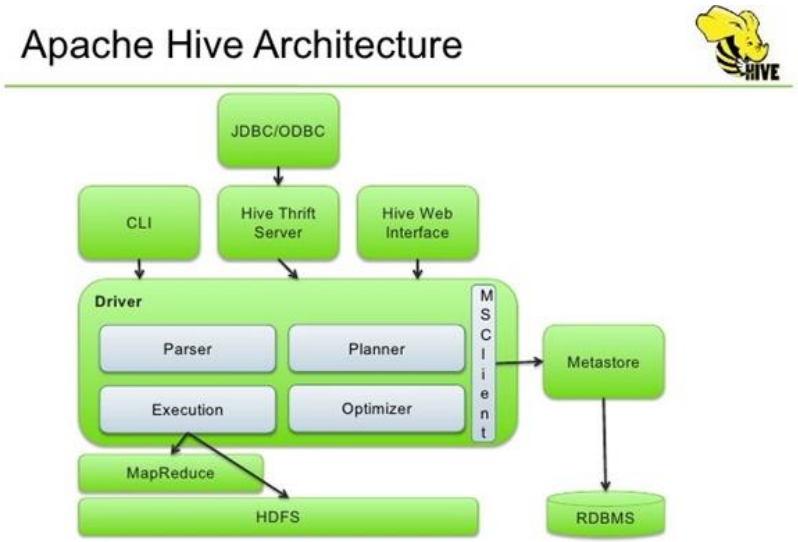


Apache Hive est un logiciel de Data Warehouse initialement créé par Facebook. Il permet d'effectuer facilement et rapidement des requêtes SQL sur HDFS

Metastore: C'est le référentiel de métadonnées (tables, son emplacement et son schéma, partition)

Driver : reçoit les instructions HiveQL. Il gere le cycle de vie des requete

Parser: convertis une requête HiveQL en une entrée MapReduce



Application:**Create Database**

```
hive> create database retail;
OK
Time taken: 5.275 seconds
hive>
```

To list out the databases in Hive warehouse, enter the command '**show databases**'.

```
hive> show databases;
OK
default
retail
Time taken: 0.228 seconds
hive>
```

Hive:Application **Create Table**

Sur Hive on a 3 types de tables :

Table Simple & table temporaire :Possibilité de (read, delete, update, ..)

Table External(Read onely) => SQL sur HDFS

Table Temporaire

```
hive> create temporary table t3(col1 int, col2 string);
OK
Time taken: 0.047 seconds
hive> create temporary table t2 like t1;
OK
Time taken: 0.262 seconds
hive>
```

DWgeek.com

Table External

```
create external table test_ext (name string, message
string)
row format delimited
fields terminated by '\t'
lines terminated by '\n'
location '/testtable'
tblproperties ("skip.header.line.count"="1");
```

Table Simple

```
hive> CREATE TABLE IF NOT EXISTS employee ( eid int, name String,
salary String, destination String)
COMMENT 'Employee details'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

Select *from test_ext;

Application: **Hive load JSON**

http://yakushev-bigdata.blogspot.com/2018/03/json-in-hive-232-with-hive-json-serde.html

```
[hadoop@hadoop-master lib]$ whoami
hadoop
[hadoop@hadoop-master lib]$ pwd
/opt/hive/lib
[hadoop@hadoop-master lib]$
wget http://www.congiu.net/hive-json-serde/1.3.8/hdp23/json-serde-1.3.8-jar-with-dependencies.jar

restart hiveserver2 (or try hive> ADD JAR json-serde-1.3.8-jar-with-dependencies.jar)
Next step is creating hive table and load json data.
```

```
DROP TABLE json1;

CREATE EXTERNAL TABLE default.json1(
  ts     INTEGER,
  device INTEGER,
  metric string,
  value  DOUBLE
  )
  ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe' STORED AS TEXTFILE;

select * from json1

json1.ts |json1.device |json1.metric |json1.value |
---------|-------------|-------------|------------|

LOAD DATA INPATH 'hdfs://10.242.5.88:9000/user/data/js_db/js1.json' OVERWRITE INTO TABLE json1;
```

Attention fichier deplacé

Application: **Hive load JSON**

```
CREATE EXTERNAL TABLE js_db.json2(
  ts     INTEGER,
  device INTEGER,
  metric string,
  value  DOUBLE
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 'hdfs://10.242.5.88:9000/user/data/js_db';
```

Hive :Conclusion

- Pour l'exécution des requêtes, Hive repose sur le modèle de programmation MapReduce ce qui entraîne une certaine lenteur, comparé à Spark
- Hive ne gère que les données structurées . Du coup, si vos données sont non structurées ou sémi structurées , Hive n est pas une option envisageable (faire appel à des libs externes comme serd dans le cas d'un JSON)

Conclusion

- Vu les limites des composants de Hadoop on peut dire que ce dernier ne répond pas exhaustivement aux 4Vs
- Nous allons scruter le FrameWork Spark dans le cours2, afin de surmonter les limites de Hadoop