Un peu d'histoire

Saagie

Hadoop a été créé par Doug Cutting, le créateur d' Apache Lucene.

- 2004 : sortie du papier de Google sur GFS et MapReduce
- 2005 : développement de NDFS pour Nutch (crawler web)
- 2005 : naissance de Hadoop, sous-projet d' Apache Lucene
- 2008 : Top projet de la fondation Apache

HDFS

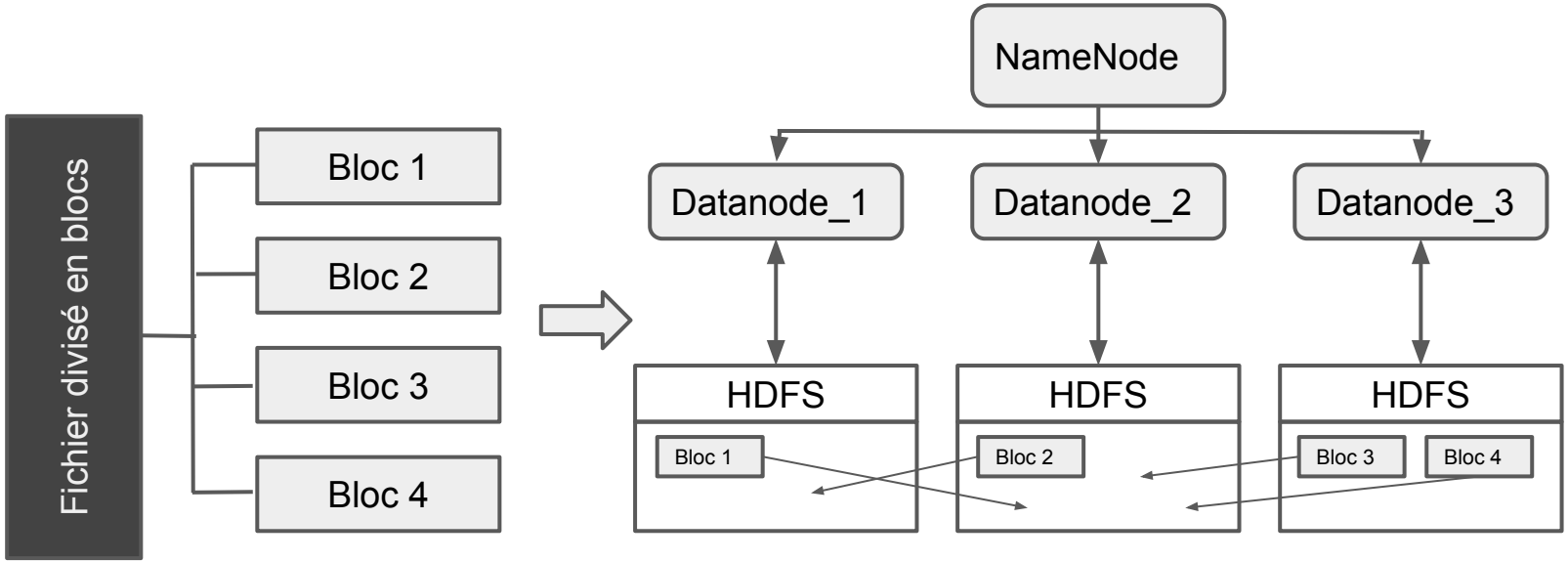
Conception

Saagie

HDFS a été conçue pour répondre aux problématiques suivantes :

- Défaillance matérielle.
- Détection d'erreur et récupération automatique.
- Lecture en continu (streaming).
- Stockage de gros fichier de données - du Gigabytes au Terabytes.
- Utilisation de matériel de base.
- Modélisation de cohérence simple - écrire une fois, lire plusieurs fois.
- Déplacement du calcul / déplacement des données.

Worker Pattern



Stockage et Réplication des blocs dans HDFS

Namenode

Saagie

- Gère le système de fichier, maintient l'arborescence et les métadonnées de tous les fichiers et les répertoires.
- Persistant grâce à deux fichiers : le fsimage et l'editlogs.
- Métadonnées montées en mémoire: liste des fichiers, liste des blocs pour chaque fichier, liste des datanodes pour chaque bloc (seulement en mémoire), attributs de fichier(dernier accès, facteur de réplication…)
- Interagit avec les clients, les datanodes, le namenode secondaire.
- Sans namenode, aucun moyen de reconstruire un fichier à partir de la liste des blocs présents sur les datanodes ⇒ HDFS down.

Datanodes

Saagie

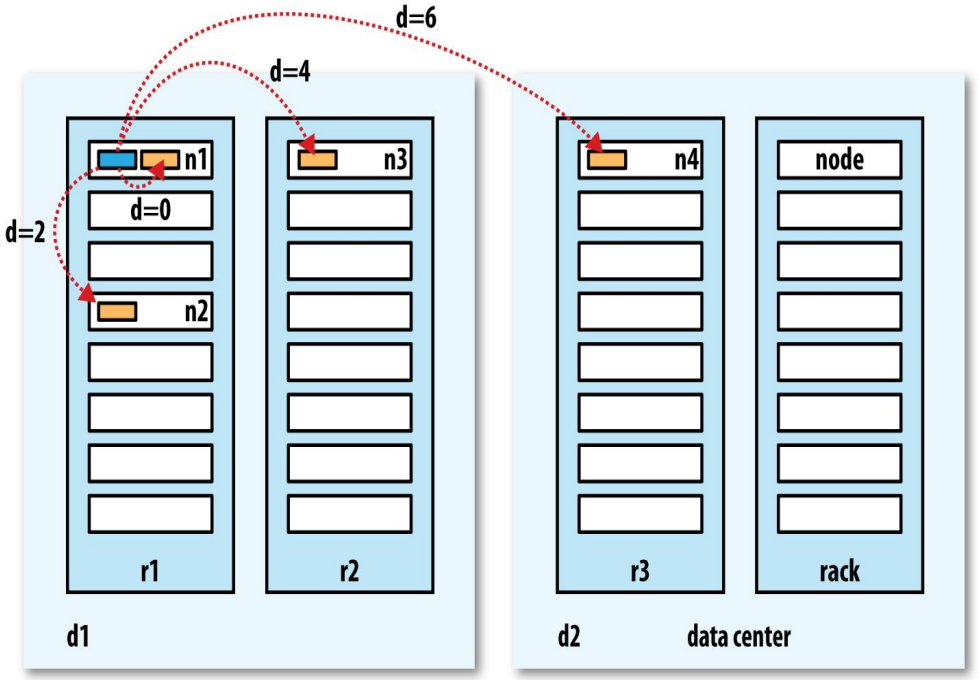
- Stockent les blocs de données et retournent les fichiers quand le namenode ou le client lui demandent.
- Reçoivent les blocs soit d'un client soit d'un autre datanode.
- Reçoivent les demandes de suppression de blocs depuis le namenode.
- Envoient périodiquement la liste des blocs des données qu'ils stockent au namenode.
- Démarrage d'un datanode : chargement des données → enregistrement au namenode → lancement de sa boucle principale (BlockScanner, Heartbeats, traitement des demandes du namenode …)

Blocs

Saagie

- Quantité minimum de données qui peut être lue ou écrite (par défaut 128 MB).
- Minimise les coûts de “seek”.
- Taille d’un fichier peut être plus gros que n’importe quel disque du cluster.
- Simplifie le sous-système de stockage : bloc de taille fixe et les datanodes ne sont pas concernés par les méta-données.
- Permet la tolérance aux pannes et la haute dispo.

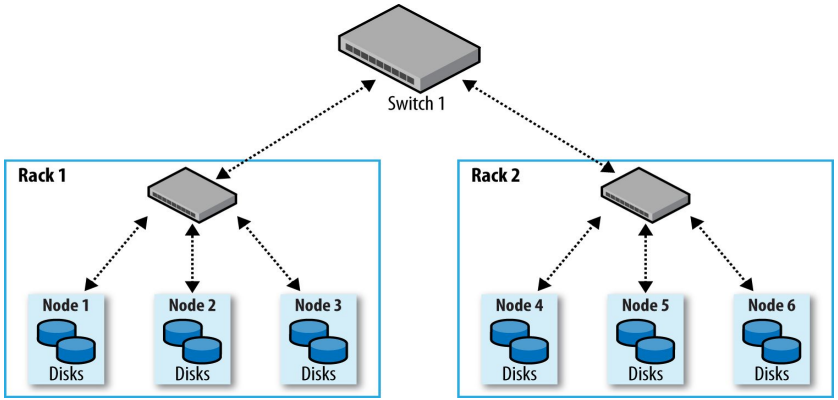
Distance entre les noeuds



Connaissance de la topologie du cluster

Saagie

- Obtenir une performance maximum du cluster.
- Améliorer la fiabilité des données, la disponibilité et l'utilisation de la bande passante.
- Permet de fournir un stockage fiable lorsqu'il y a une défaillance de datanodes.
- Résolution des noms de dns (et donc d'une ip) à un identifiant de rack.



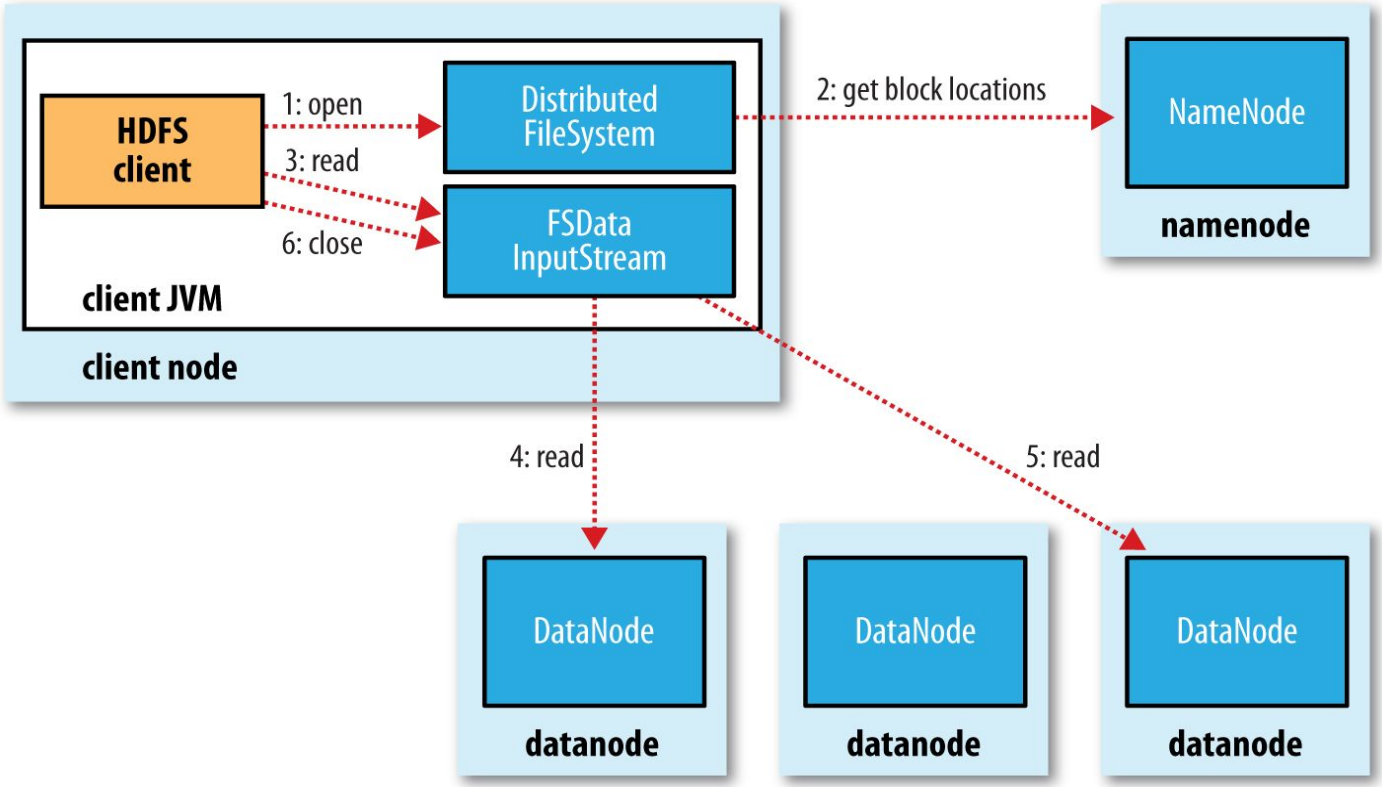
Emplacement des réplicas

Saagie

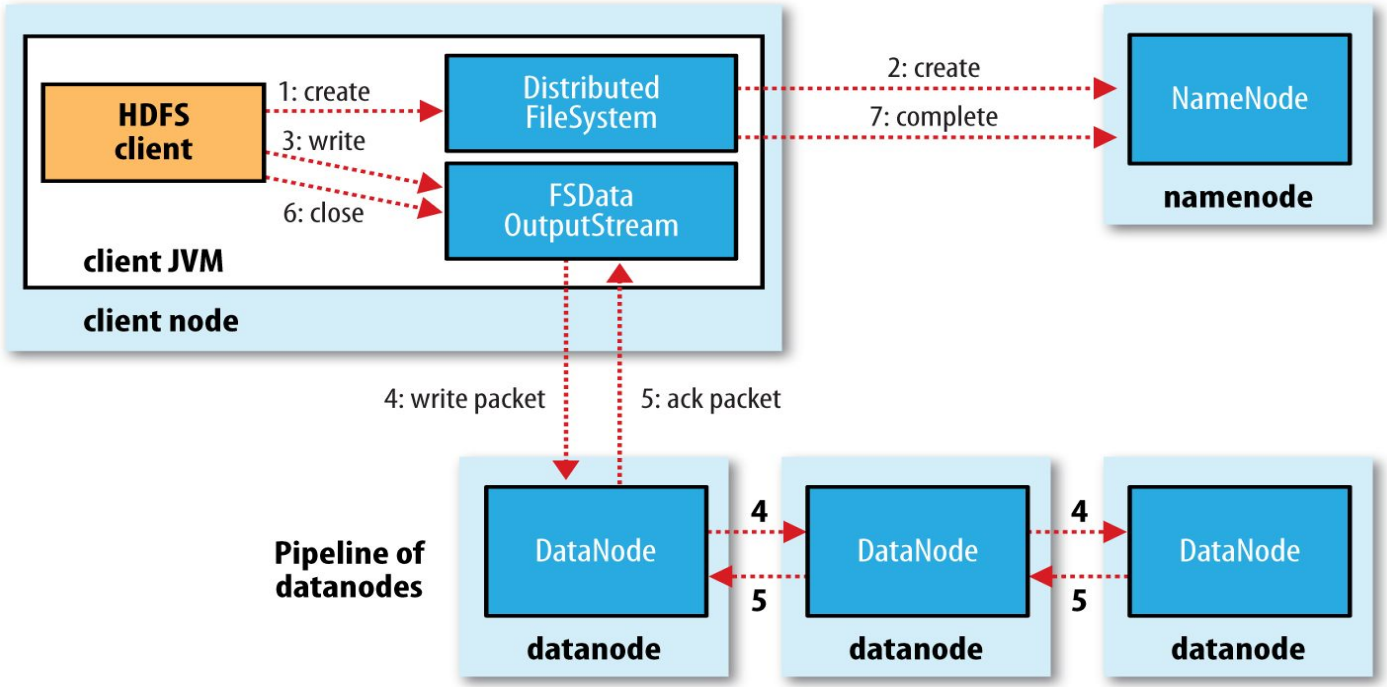
- Premier réplica sur le même noeud que le client ou au hasard.
- Second réplica sur un rack différent choisi au hasard.
- Troisième réplica sur le même rack que le second mais sur un noeud différent.
- Autres réplicas placés sur des noeuds aléatoirement (en évitant d'en mettre trop sur le même rack).

HDFS essaye de satisfaire une requête de lecture à partir du réplica le plus proche du lecteur (data locality).

Lecture d'un fichier en Java



Ecriture d'un fichier en Java



Gestion des erreurs lors de l'écriture

Saagie

En cas d'échec d'un datanode:

- Pipeline (mapping bloc -> datanodes) fermée.
- Paquets de l'ackqueue ajoutés en tête de la dataqueue.
- Renommage du bloc courant.
- Envoie du nom de ce bloc au namenode pour le supprimer sur le datanode qui a échoué (à son retour).
- Suppression du datanode défaillant de la pipeline afin d'en créer une nouvelle à partir des datanodes restants.
- Paquets restants du bloc écrits sur les (RF -1) datanodes.
- Réplication ultérieure des blocs sous répliqués.

Gestion des droits

Saagie

- Modèle de permissions basé sur le model POSIX : fichier et répertoire associé à un propriétaire et à un groupe.
- Permissions différentes pour le propriétaire, pour le groupe et pour les autres utilisateurs.
- Possibilité de positionner un“sticky bit” sur des répertoires : empêche la suppression et le déplacement d’un fichier à l’intérieur du répertoire (excepté pour le super utilisateur, le propriétaire du fichier ou du répertoire).
- Super utilisateur : utilisateur qui lance le namenode.
- Support pour les ACLs.

Intégrité des données

Saagie

- Checksum (CRC-32C) calculé pour chaque nouveau fichier
  Vérification des données reçues par les dn avant stockage (avec checksum).
- Journal persistant de vérifications de checksum (permet de détecter les mauvais disques) mis à jour après lecture (client/DataBlockScanner).
- Correction des erreurs grâce aux réplicas :
  - le client signale au namenode le bloc et le datanode à partir duquel il essayait de le lire
  - le namenode le marque corrompu et programme une copie du bloc pour maintenir le niveau de réplication attendu

Persistance du namenode

Saagie

La persistance des données du nanenode est assurée par 2 fichiers :

- editlogs: maintient un suivi de chaque changement qui se fait sur HDFS (ajout/suppression de fichiers, déplacement entre répertoires...)
- fsimage: stocke les détails de l'inode (date de modification, temps d'accès, autorisations d'accès, réplication)

Namenode et redémarrage

Saagie

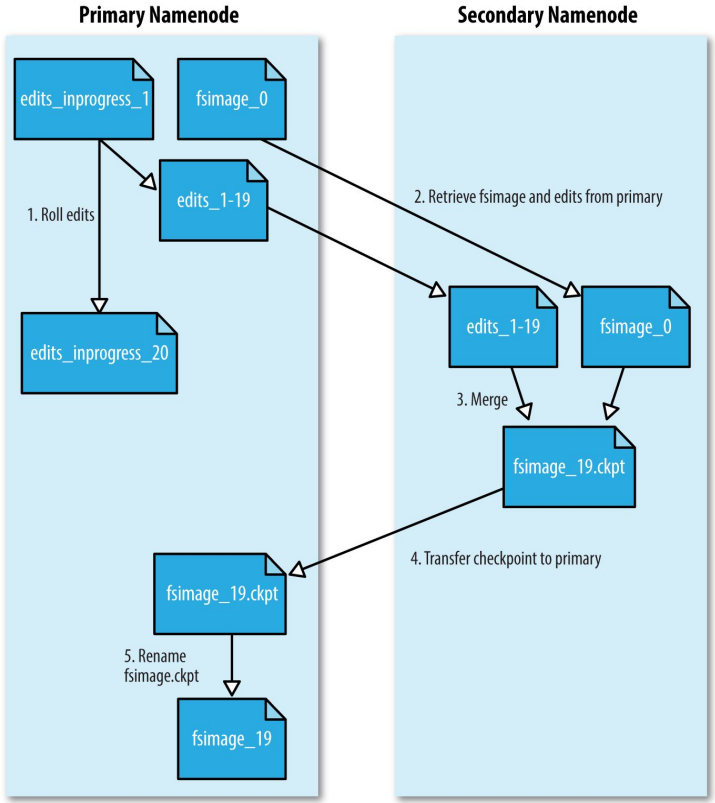
Lors d'un redémarrage, le namenode du cluster :

- charge en mémoire le fsimage
- répercute les changements de l'editlogs
- réceptionne les rapports de blocs des datanodes pour reconstruire le mapping blocs/datanodes

Le temps de répercussions de l'editlogs peut être long en fonction de la taille du fichier.

Processus de checkpointing

Saagie



Mode sécurisé

Saagie

Au démarrage, le namenode entre en mode sécurisé (only read).

Il charge en mémoire le fsimage et répercute les changements de l'editlogs.

Il effectue le processus de checkpointing sans recours au namenode secondaire.

Il laisse le temps nécessaire aux datanodes de s'enregistrer et d'envoyer leurs listes de blocs.

Lorsque la condition de réplication minimale est atteinte, plus un temps additionnel de 30 secondes le mode sécurisé est quitté.

Haute disponibilité

Saagie

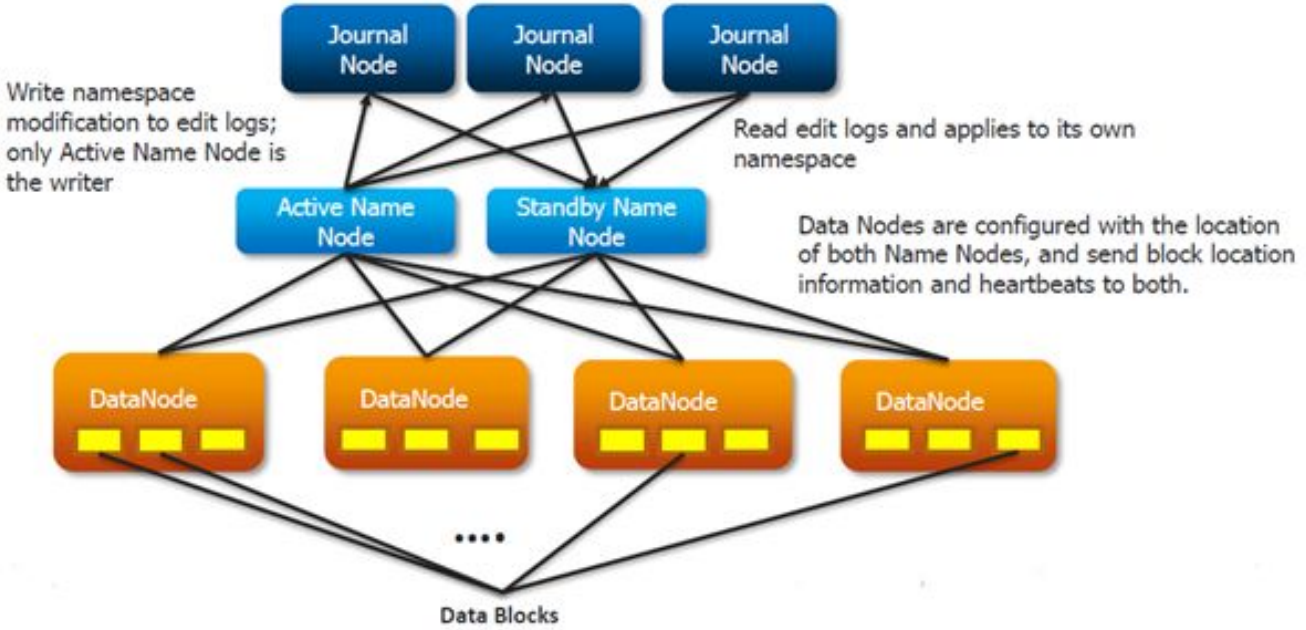
- Le namenode secondaire assure une protection contre la perte de données mais le namenode principale reste un SPOF.
- Besoin d'un nouveau namenode principale avec copie des métadonnées s'il échoue ⇒ nécessite une reconfiguration des clients.
- HDFS HA : paire de namenodes avec failover:
  - utilisation d'un système de stockage partagé hautement disponible pour le fichier d'editlogs (NFS ou QJM)
  - envoie des rapports de blocs des datanodes aux deux namenodes (car le mapping est en mémoire)
  - clients configurés pour traiter le failover
  - utilisation de Zookeeper pour élire le namenode actif

QJM

Saagie

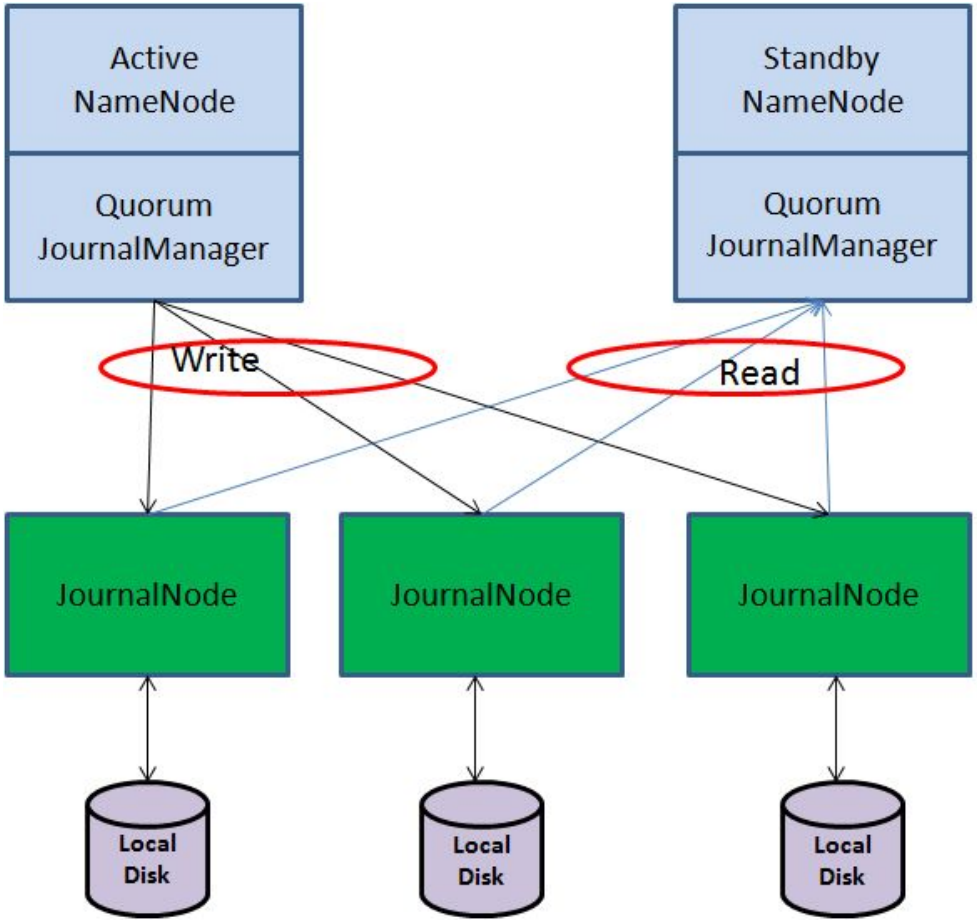
- Implémentation dédiée à HDFS dont le seul but est de fournir un edit log hautement disponible.
- Se lance comme un groupe de journal nodes où chaque nouvelle entrée doit être écrite sur une majorité de noeuds (typiquement 3).
- Si le namenode actif échoue, le namenode inactif prend le relais très rapidement (le temps de détecter que le NN actif a échoué).
- QJM répond à la problématique de “failover and fencing”:
    - permet qu’à un seul namenode d’écrire dans le journal node à la fois
    - le précédent namenode actif peut toujours servir les requêtes de lecture aux clients

QJM


Journal Node
Journal Node
Journal Node
Write namespace modification to edit logs; only Active Name Node is the writer
Read edit logs and applies to its own namespace
Active Name Node
Standby Name Node
Data Nodes are configured with the location of both Name Nodes, and send block location information and heartbeats to both.
DataNode
DataNode
DataNode
DataNode
Data Blocks

QJM Fencing

Saagie



Shell

Saagie

- hadoop fs {args} : système de fichier générique donc peut être utilisé avec FS, HFTP FS, S3 FS, HDFS
- hadoop dfs {args} : spécifique à HDFS mais a été déprécié au profit de hdfs dfs

hdfs dfs {args} pour toutes les opérations concernant HDFS
hdfs dfs -put <localsrc> ... <dst>
hdfs dfs -mkdir [-p] <paths>
hdfs dfs -appendToFile <localsrc> ... <dst>
hdfs dfs -copyFromLocal <localsrc> URI
hdfs dfs -ls [-d] [-h] [-R] [-t] [-S] [-r] [-u] <args>
hdfs dfs -du [-s] [-h] URI [URI ...]
hdfs dfs -df [-h] URI [URI ...]

Saagie

Autres groupes de commande en relation avec hdfs :

- **namenode|secondarynamenode|datanode|dfsadmin|fsck|balancer|fetchdt|oiv|dfsgroups**

Par exemple, % ***hdfs fsck / -files -blocks*** % liste les blocs qui constituent chaque fichier du système de fichier.

Exercice

Saagie

- A l'aide d'un client Hadoop: (username: hadoop)
  - Lister des fichiers sur le HDFS.
  - Créer un répertoire personnel /user/hadoop/<nom>
  - Ecrire le fichier /user/hadoop/<nom>/sample.txt sur le HDFS
  - Lire le fichier /user/hadoop/<nom>/sample.txt depuis le HDFS
- Namenode: 51.15.133.130:9000
- Namenode webhdfs: 51.15.133.130:50070
- Datanode webhdfs: 51.15.133.130:50075

```
System.setProperty("HADOOP_USER_NAME", "hadoop")
val configuration = new Configuration()
configuration.set("fs.default.name", "hdfs://51.15.133.130:9000")
configuration.set("dfs.datanode.use.datanode.hostname", "true")
configuration.set("dfs.client.use.datanode.hostname", "true")
```

MapReduce

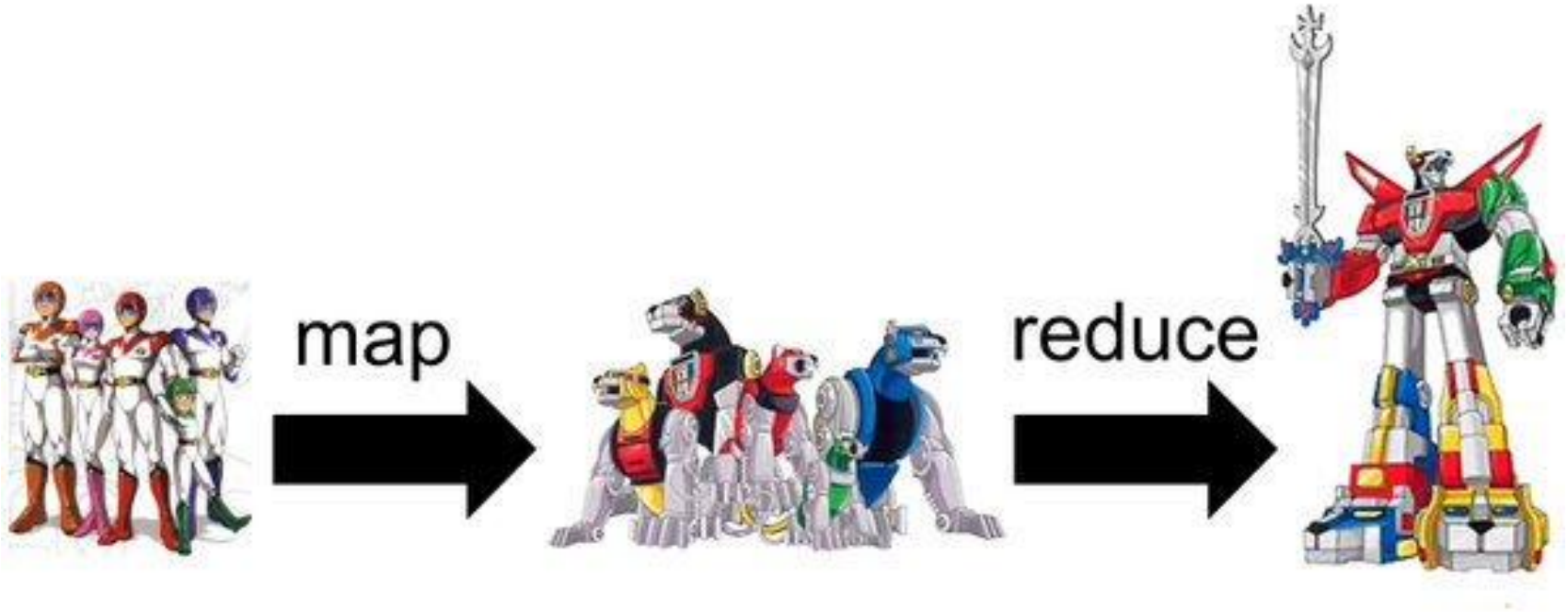
Introduction

Saagie

- Paradigme de programmation et un environnement d'exécution proposé par Google (2003).
- Parallélisation de calcul manipulant de gros volumes de données dans des clusters de machines.
- Modèle de programmation fonctionnelle, ordre des opérations sans importance.
- Environnement d'exécution permettant d'automatiser : la parallélisation, la gestion des éventuelles pannes et la gestion des communications entre les machines.

Exemple

Saagie



Exemple

Saagie

Exemple

Saagie

map, filter, and reduce explained with emoji 😂

```
map([🐮, 🍢, 🐔, 🌽], cook)
=> [🍔, 🍟, 🍗, 🍿]

filter([🍔, 🍟, 🍗, 🍿], isVegetarian)
=> [🍟, 🍿]

reduce([🍔, 🍟, 🍗, 🍿], eat)
=> 💩
```

Vue d'ensemble

Saagie

- Un programme MapReduce = données en entrée + fonctions Map et Reduce + infos de configuration.
- Flux de données de paires <clé, valeur>.
- Données d'entrée divisées en blocs de taille fixe appelés split qui seront traités par différentes tâches.
- 1 tâche de map pour chaque split
- Exécution de la tâche de map sur un noeud où les données d'entrée sont présentes. ⇒ taille du split tend vers la taille d'un bloc HDFS.
- Tâches planifiées par Yarn et lancées sur les noeuds du cluster.

Exemple

Saagie

- Données brutes au format ASCII orienté ligne, où chaque ligne est un enregistrement de température du NCDC.
- Calcul de la température max par année.

```
0067011990999991950051507004...9999999N9+00001
+99999999999…
0043011990999991950051512004...9999999N9+00221
+99999999999…
0043011990999991950051518004...9999999N9-00111+
99999999999…
0043012650999991949032412004...0500001N9+01111
+99999999999…
0043012650999991949032418004...0500001N9+00781
+99999999999…
```

- En entrée du job, un text input qui nous donne pour chaque ligne du fichier une valeur texte.
- La clé est le décalage du début de la ligne par rapport au début du fichier :

```
(0,0067011990999991950051507004...9999999N9+0000
1+99999999999...)
(106,0043011990999991950051512004...9999999N9+00
221+99999999999...)
(212,0043011990999991950051518004...9999999N9-00
111+99999999999...)
(318,0043012650999991949032412004...0500001N9+01
111+99999999999...)
(424,0043012650999991949032418004...0500001N9+00
781+99999999999...)
```

Exemple

La fonction de Map va extraire la température et l'année et les retourner sous forme de pair (année, température) :

(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
(1949, 78)

La sortie de la fonction Map est traitée par le framework MapReduce avant d'être envoyé à la fonction de Reduce. Ce traitement trie et groupe les pairs clé-valeur par clé :

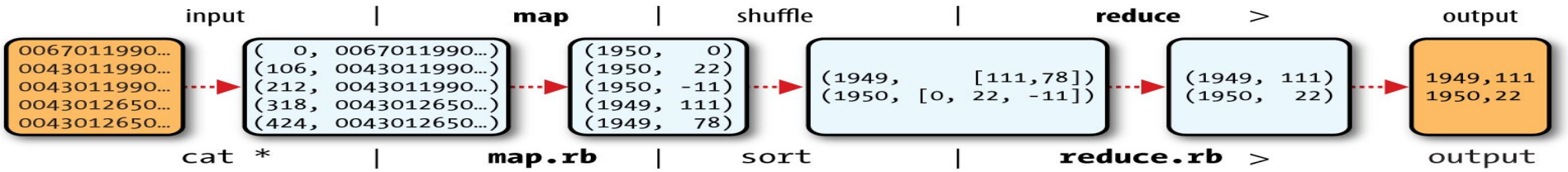
(1949, [111, 78])
(1950, [0, 22, −11])

Exemple

Chaque année apparaît avec une liste de toute ses températures.

La fonction Reduce n'a plus qu'à itérer sur chaque liste pour trouver le max :

(1949, 111)
(1950, 22)



Exemple - Code du Mapper

Saagie

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MaxTemperatureMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
   private static final int MISSING = 9999;
   @Override
   public void map(LongWritable key, Text value, Context context)
           throws IOException, InterruptedException {
       String line = value.toString();
       String year = line.substring(15, 19);
       int airTemperature;
       if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs
           airTemperature = Integer.parseInt(line.substring(88, 92));
       } else {
           airTemperature = Integer.parseInt(line.substring(87, 92));
       }
       String quality = line.substring(92, 93);

       if (airTemperature != MISSING && quality.matches("[01459]")) {
           context.write(new Text(year), new IntWritable(airTemperature));
       }
   }
}
```

Exemple - Code du Reducer

Saagie

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MaxTemperatureReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
   @Override
   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
       int maxValue = Integer.MIN_VALUE;
       for (IntWritable value : values) {
           maxValue = Math.max(maxValue, value.get());
       }
       context.write(key, new IntWritable(maxValue));
   }
}
```

Exemple - Code du Job

Saagie

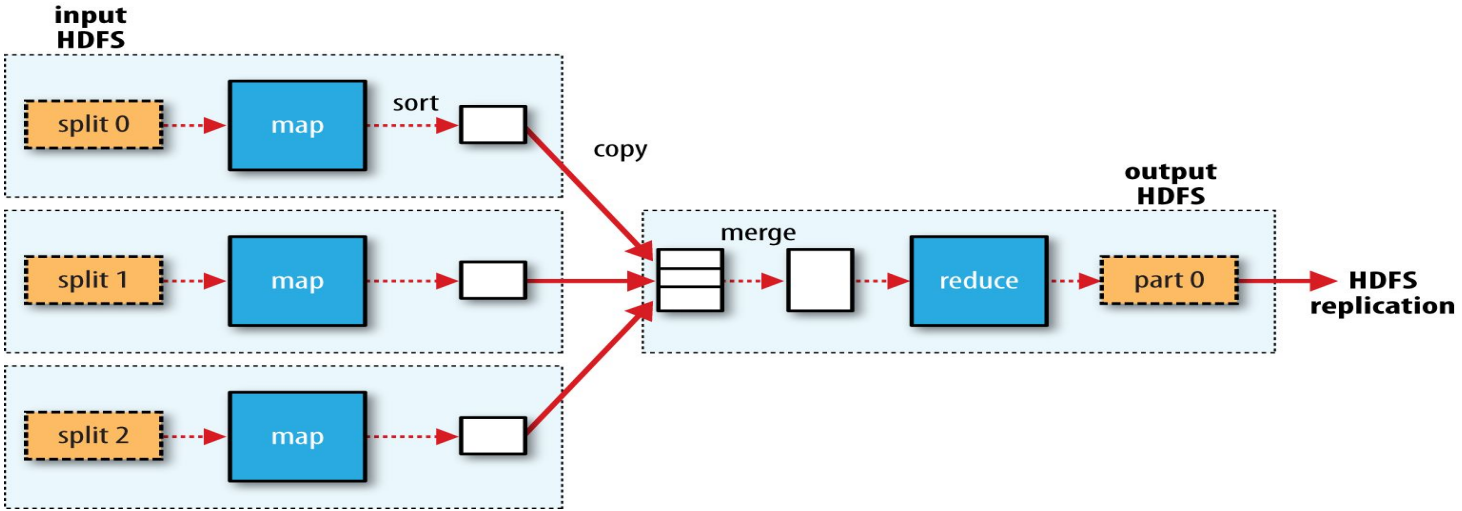
```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxTemperature {
   public static void main(String[] args) throws Exception {
       if (args.length != 2) {
           System.err.println("Usage: MaxTemperature <input path> <output path>");
           System.exit(-1);
       }
       Job job = new Job();
       job.setJarByClass(MaxTemperature.class);
       job.setJobName("Max temperature");
       FileInputFormat.addInputPath(job, new Path(args[0]));
       FileOutputFormat.setOutputPath(job, new Path(args[1]));
       job.setMapperClass(MaxTemperatureMapper.class);
       job.setReducerClass(MaxTemperatureReducer.class);
       job.setOutputKeyClass(Text.class);
       job.setOutputValueClass(IntWritable.class);
       System.exit(job.waitForCompletion(true) ? 0 : 1);
   }
}
```

Reduce

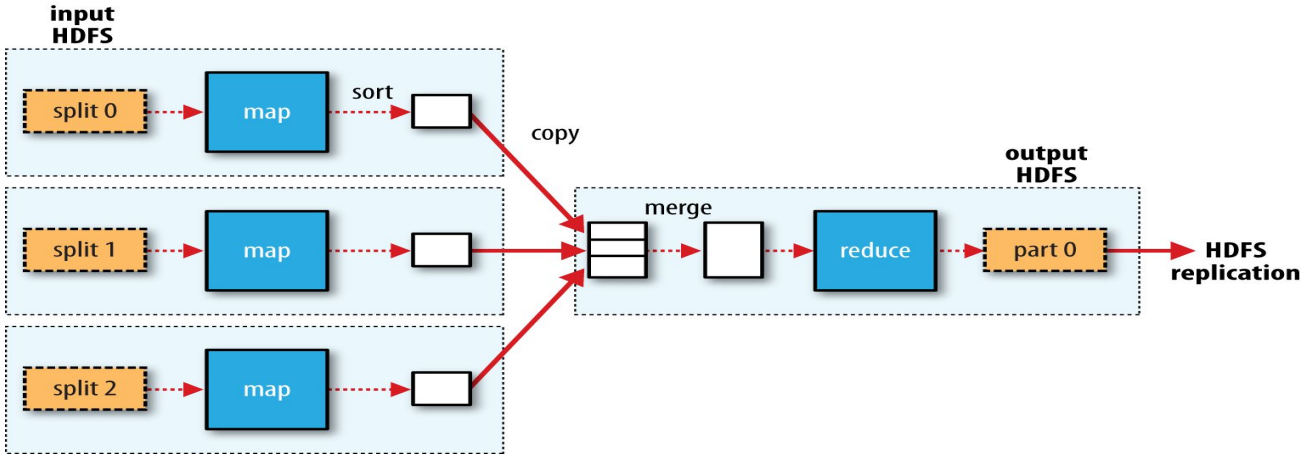
Saagie

- Tâches de Reduce ne bénéficient pas de la localité des données.
- Entrée d'une tâche de Reduce = à la sortie de toutes les tâches de Map
- Sortie de la tâche de Reduce stocké dans HDFS pour question de fiabilité.



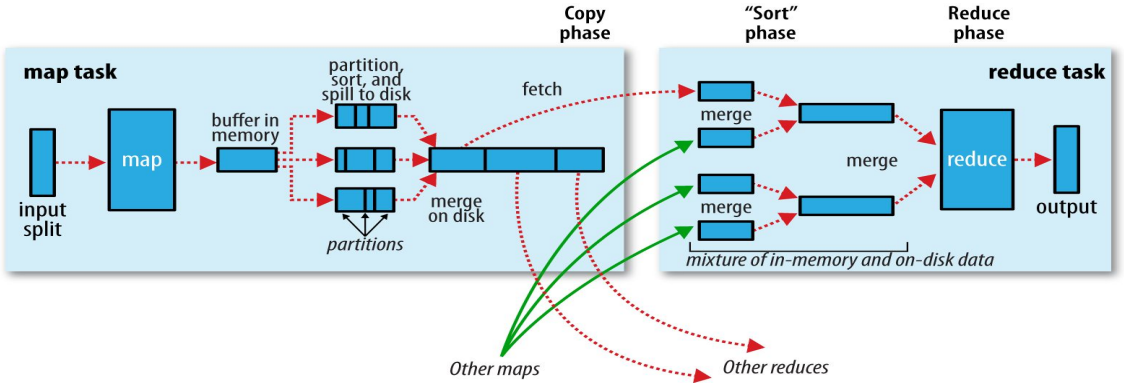
Reduce

- Possibilité de configurer plusieurs réducteurs ⇒ les tâches de Map partitionnement leurs sorties (1 partition pour chaque réducteur).
- Chaque partition créée peut contenir plusieurs clés mais les enregistrements pour une clé donnée sont dans une seule partition.



Shuffle

- Transmission des données de la phase Map vers la phase Reduce
- Responsabilité des Maps : stockage local partitionné des couples de sortie (clé, valeur)
- Responsabilité des reduces :
  - copy : téléchargement sur chaque Map de la partition qui lui est associée
  - merge : agrégation de l'ensemble des partitions téléchargées par clé
  - sort : tri des différentes clés définissant l'ordre de lecture par le Reduce

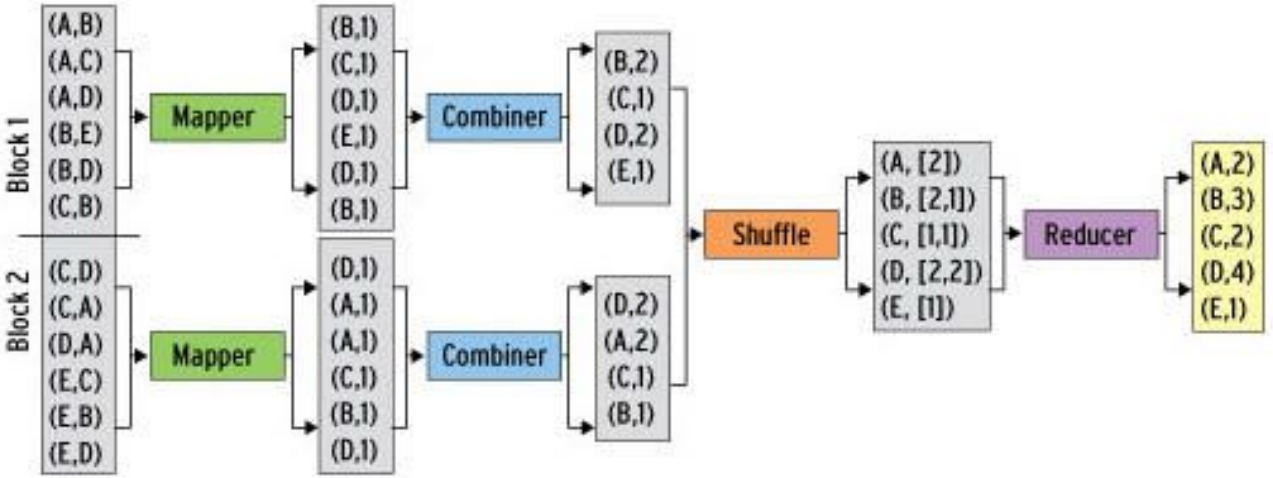


Combiner

Saagie

- Minimiser les données transférées entre les tâches de Map et de Reduce.
- Hadoop ne garantit pas combien de fois la fonction sera appelé. La fonction doit donc fournir le même résultat qu'elle soit appelée 0, 1 ou plusieurs fois.
- Avec l'exemple précédent, imaginons deux tâches de Map qui produisent :
  (1950, 0) (1950, 25)
  (1950, 20) (1950, 15)
  (1950, 10)
- Avec une fonction combiner qui trouve le max pour chaque sortie de Map, la fonction de Reduce sera appelé avec : (1950, [20, 25])
- Ça fonctionne car max(0, 20, 10, 25, 15) = max(max(0, 20, 10), max(25, 15)) = max(20, 25) = 25
- Avec la fonction mean, ça ne fonctionne pas : mean(0, 20, 10, 25, 15) = 14 mais mean(mean(0, 20, 10), mean(25, 15)) = mean(10, 20) = 15 (!= 14)

Combiner


Block 1
(A,B)
(A,C)
(A,D)
(B,E)
(B,D)
(C,B)
Block 2
(C,D)
(C,A)
(D,A)
(E,C)
(E,B)
(E,D)
Mapper
Mapper
(B,1)
(C,1)
(D,1)
(E,1)
(D,1)
(B,1)
(D,1)
(A,1)
(A,1)
(C,1)
(B,1)
(D,1)
Combiner
Combiner
(B,2)
(C,1)
(D,2)
(E,1)
(D,2)
(A,2)
(C,1)
(B,1)
Shuffle
(A, [2])
(B, [2,1])
(C, [1,1])
(D, [2,2])
(E, [1])
Reducer
(A,2)
(B,3)
(C,2)
(D,4)
(E,1)

MapReduce 1.x - Jobtracker

Saagie

- Gère l'ensemble des ressources du système.
- Reçoit les jobs des clients.
- Ordonnance les différentes tâches des jobs soumis.
- Assigne les tâches aux Tasktrackers.
- Ré affecte les tâches défaillantes.
- Maintient des informations sur l'état d'avancement des jobs.

MapReduce 1.x - Tasktracker

Saagie

- Exécute les tâches données par le Jobtracker.
- Exécution des tâches dans une autre JVM (Child).

Pourquoi une v2?

Saagie

Architecture trop centralisée de Hadoop 1.x :

- vulnérable aux défaillances du JobTracker, goulot d'étranglement (passage à l'échelle difficile)
- mélange de la gestion des ressources avec la gestions des jobs

Apport de Hadoop 2.x (YARN) :

- séparation de la gestion des ressources du cluster et de la gestion des jobs MapReduce
- généralisation de la gestion des ressources du cluster applicable à toute application

Exercice

Saagie

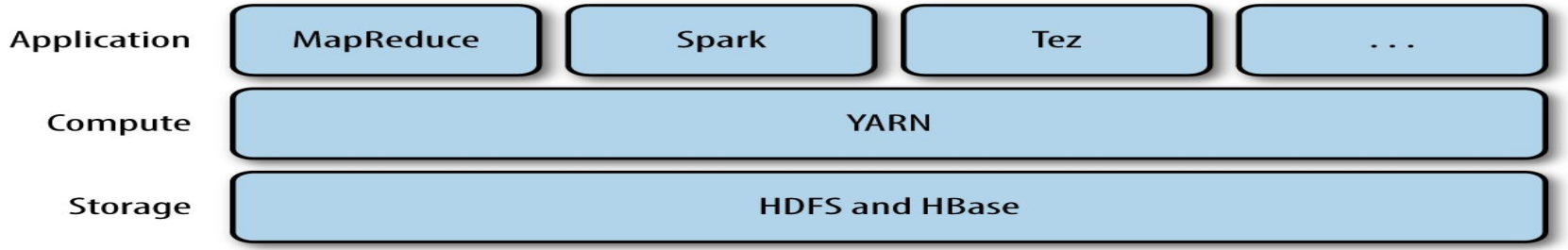
- Effectuer en local un word count sur le fichier sample.txt
- Trois éléments différents:
  - Le mapper qui associe un nombre à un mot.
  - Le reducer qui fera la somme pour chacun de ces mots.
  - La configuration du job qui lancera son exécution.

YARN

Yet Another Resource Manager

Saagie

- Système de gestion de ressource de cluster introduit par Hadoop 2 pour améliorer Map Reduce.
- APIs pour demander et travailler avec les ressources du cluster qui ne sont pas utilisés directement par le code utilisateur mais par des frameworks construit sur YARN directement (Spark, MapReduce, Tez…) ou indirectement (Pig, Hive, Crunch…) .
- 2 types de démons longue durée : gestionnaire de ressources (resource manager) et gestionnaire de noeud (node manager).



Répartition des fonctionnalités de la v1

Saagie

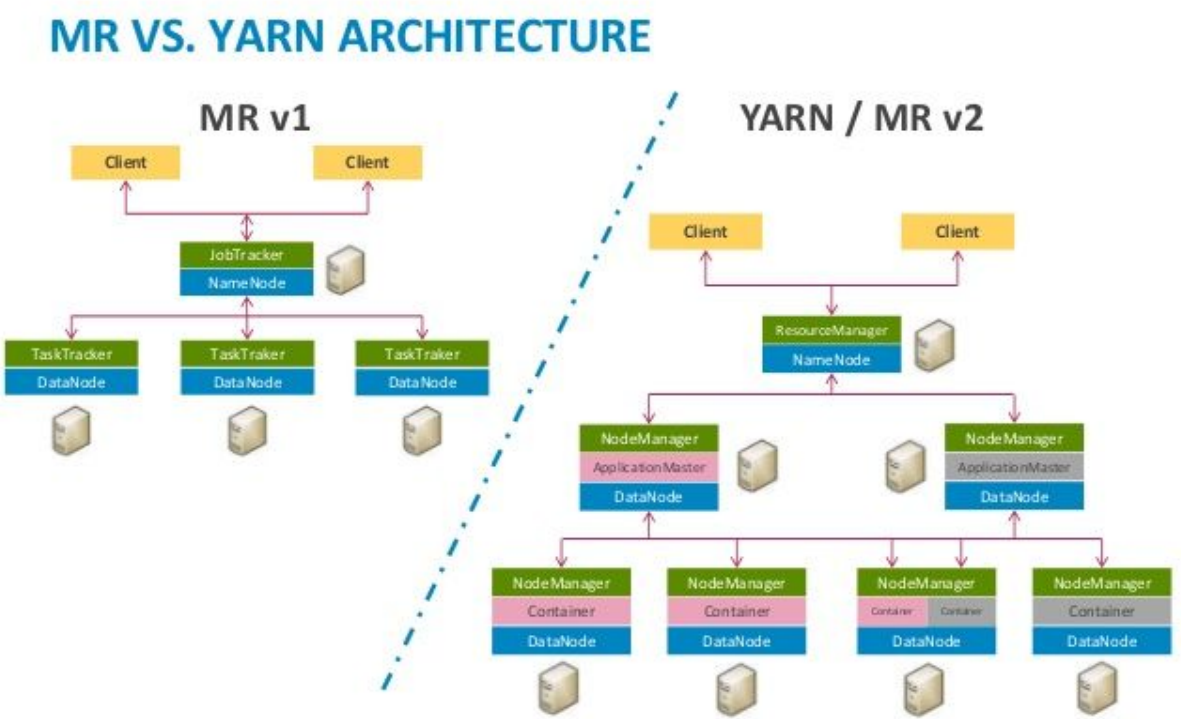
Répartition des fonctionnalités du Jobtracker :

- le ResourceManager (RM) : JVM s'exécutant sur le nœud maître qui contrôle toutes les ressources du cluster et l'état des machines esclaves et ordonnance les requêtes clientes
- une ApplicationMaster (AM) par application soumise (= 1 ou plusieurs Job) : JVM s'exécutant sur une machine esclave qui négocie avec le RM les ressources nécessaires à l'application

Répartition des fonctionnalités du Tasktracker sur une machine esclave :

- un NodeManager (NM) : JVM sur une machine esclave qui gère les ressources du nœud
- des Containers : abstraction des ressources sur un nœud, dédiée soit à une JVM ApplicationMaster, soit à une JVM YarnChild qui exécute une tâche Map ou Reduce

Répartition des fonctionnalités de la v1



- **YARN** : **Y**et **A**nother **R**esource **N**egotiator
- **MR** : MapReduce

ResourceManager

Saagie

- Gère le cluster en maximisant l'utilisation des ressources.
- Réceptionne les requêtes clientes :
  - soumission et suppression des applications
  - informations sur l'état courant des applications
- Gère l'ensemble des applications :
  - ordonnance et stocke l' état des applications
  - alloue un container pour l'ApplicationMaster sur un NodeManager
  - redémarre l'application en cas de défaillance
- Réceptionne et ordonnance les requêtes de containers des ApplicationMaster
- Affecte des containers sur les NodeManagers en fonction des ressources disponibles.
- Gère la sécurité des communications.

ApplicationMaster

Saagie

- Négocie l'allocation des ressources et communique avec les NodeManager pour les utiliser.
- Gère le cycle de vie de l'application, les défaillances des containers, l'ajustement de la consommation des ressources.
- Hearbeat avec le ResourceManager :
  - message aller : état courant de l'application, containers demandés (nombre requis, préférence de localité), container libérés
  - message retour : liste des containers nouvellement alloués et leurs clés de sécurité , ressources disponibles, possibilité de directive d'arrêt ou de resynchronisation
- Communique avec les NodeManager pour démarrer/arrêter les container alloués qui exécuteront les tâches de map et reduce (JVM YarnChild).

NodeManager

Saagie

- Héberge les containers.
- Libère les containers à la demande du RM ou de l'AM correspondant.
- Maintient des clés de sécurité provenant du RM afin d'authentifier les utilisations des containers.
- Monitore l'état de santé du nœud et des containers et hearbeat avec le RM:
  - message aller : états des containers hébergés, états du noeud
  - message retour : des clés de sécurité pour la communication avec les AM, les containers à libérer, possibilité de directive d'arrêt ou de resynchronisation

Demande de ressources

Saagie

Demande flexible de ressources :

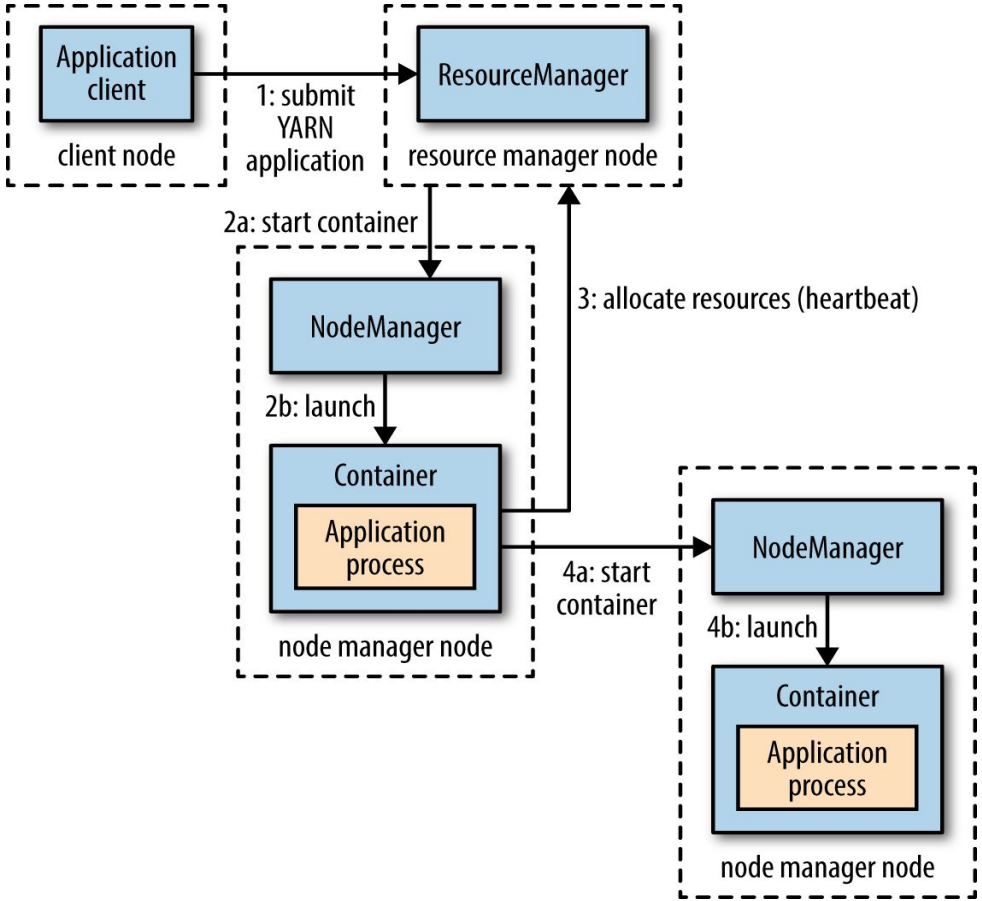
- ressources informatiques (CPU, RAM) pour chaque container
- contraintes de localités (noeud rack/spécifique)

Les contraintes de localité peuvent ne pas être respectées.

Pour le traitement d'un bloc HDFS, l'application demandera un container sur un des nœuds hébergeant 1 des 3 réplicas du bloc, ou sur un noeud du même rack d'un des 3 réplicas, ou, à défaut, sur un noeud du cluster.

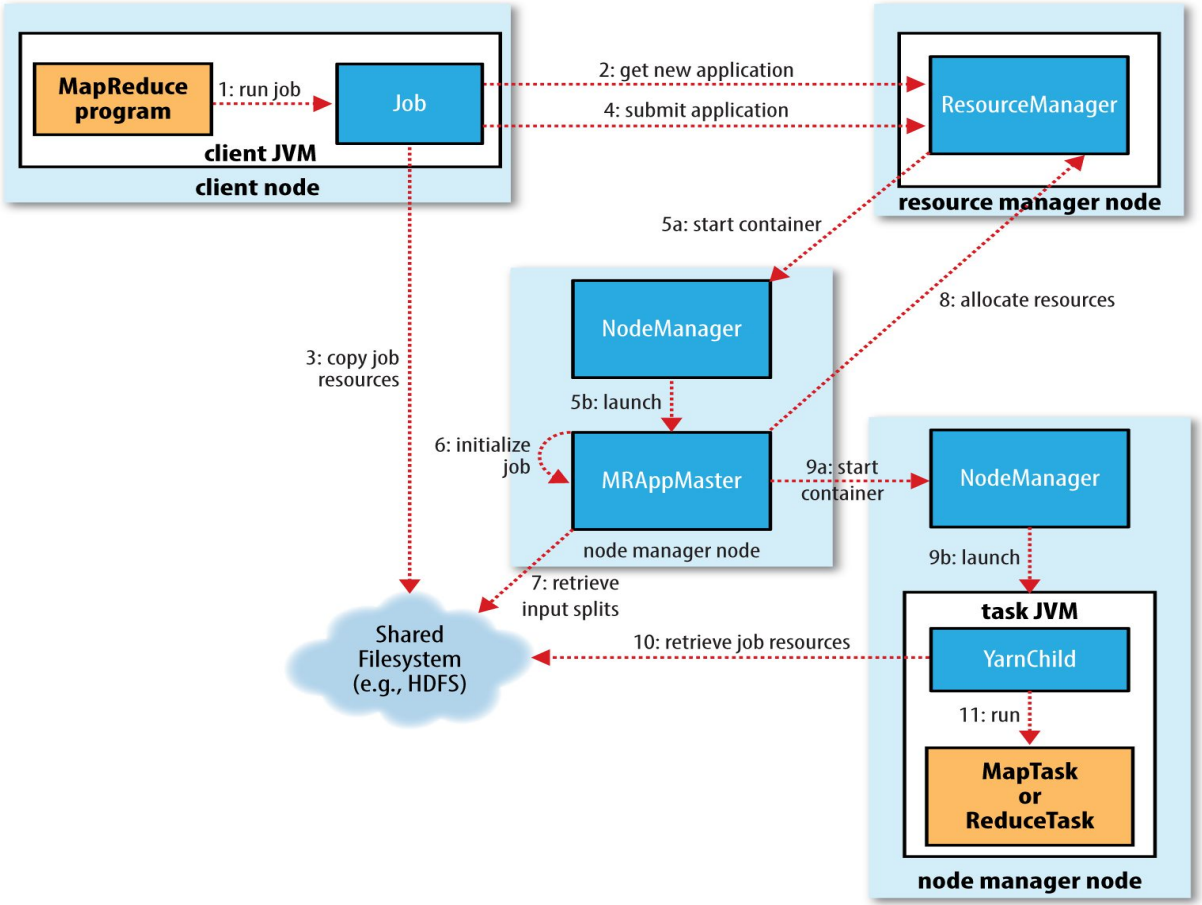
Demande en avance ou dynamique des ressources.

Soumission d'une application



Soumission d'un job MapReduce

Saagie



Cycle de vie d'une application

Saagie

Application de courte durée à application de longue durée.

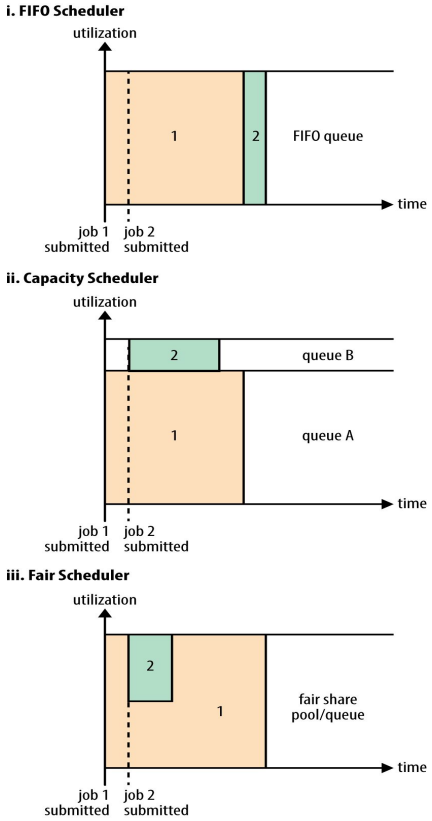
3 modèles de cycle de vie :

- une application par job utilisateur (MapReduce)
- une application par workflow/ jobs multiples : réutilisation des containers et possibilité de mettre en cache les données intermédiaires (Spark)
- une application permanente partagés par différents utilisateurs (Impala)

Scheduling

Saagie

- FIFO Scheduler : simple à comprendre, pas de configuration mais pas adapté pour les clusters partagés.
- Capacité Scheduler : file d'attente distincte dédié, permet au “petit” job de commencer dès qu'il est soumis , même si cela est au détriment de l’utilisation globale du cluster puisque la capacité de file d'attente est réservé pour des jobs dans cette file d'attente.
- Fair Scheduler : pas de réservation, ressources dynamiquement équilibrées entre toutes les tâches en cours.
- Possibilité de configuration par queue.



Compression et format de fichiers

Formats de compression

Saagie

Compression format	Tool	Algorithm	Filename Extension	Splittable?
DEFLATE	N/A	DEFLATE	.deflate	No
gzip	gzip	DEFLATE	.gz	No
bzip2	bzip2	bzip2	.bz2	Yes
LZO	lzop	LZO	.lzo	No
LZ4	N/A	LZ4	.lz4	No
Snappy	N/A	Snappy	.snappy	No

Compression

Saagie

- Réduit l'espace nécessaire pour le stockage des fichiers et accélère le temps de transfert à travers le réseau.
- bzip2 compresse plus efficacement que gzip mais est plus lent.
- LZO, LZ4 et Snappy optimisés pour la vitesse de compression.
- LZ4 et Snappy significativement plus rapide que LZO pour la décompression.

Solutions de stockage dans le BIG DATA

Saagie

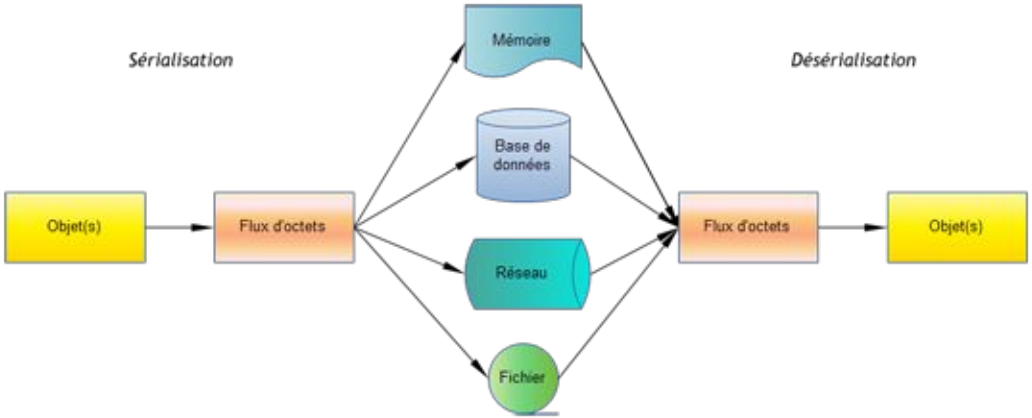
1. Utiliser un fichier de format conteneur (avro, orc, parquet…) qui supportent à la fois la compression et le fractionnement avec compresseur rapide.
2. Utiliser un format de compression qui prend en charge le fractionnement comme bzip2 ou un format qui peut être indexé pour soutenir le fractionnement comme LZO.
3. Diviser le fichier en morceaux dans l'application, et compresser chaque morceau séparément en utilisant n'importe quel format de compression pris en charge. Choisir la taille d'un morceau de sorte que les morceaux compressés soit environ égale à la taille d'un bloc HDFS.
4. Stocker le fichier décompressé.

Rappel - Sérialisation/Désérialisation

Saagie

Sérialisation: processus de transformation des objets structurés dans un flux d'octets pour transmission sur réseau ou pour écriture persistante.

Désérialisation: processus de transformation d'un flux d'octets en une série d'objets structurés.



Choix du format dans le Big Data

Saagie

Dans un contexte Big Data, le format de stockage doit être :

- compacte : pour rendre efficace l'utilisation de l'espace de stockage
- rapide : pour minimiser la surcharge en lecture ou l'écriture de gros volumes de données
- extensible : pour lire des données écrites dans un format plus ancien
- inter-opérable : lire ou écrire des données persistantes en utilisant différents langages

Gérer les types dans le code complique les deux derniers points.

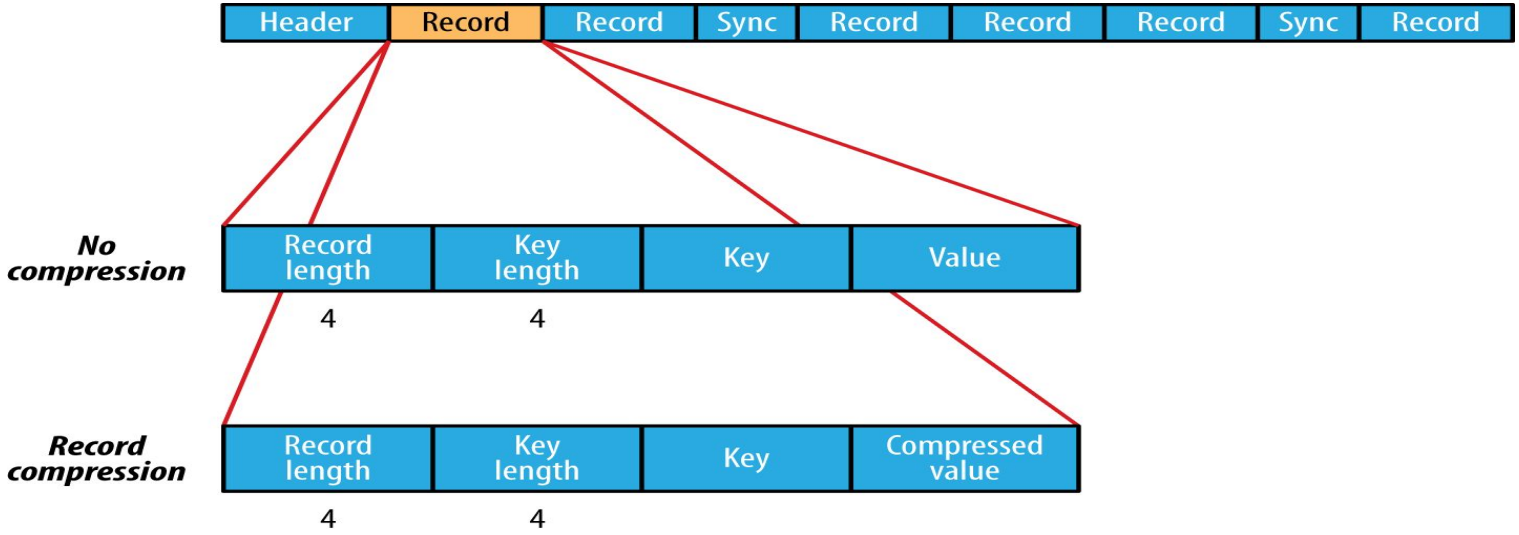
Solution : utiliser un framework de sérialisation utilisant un mode déclaratif (IDL).

Sequence File

Saagie

- Paires de <clé, valeur>.
- 3 premiers bytes sont les bytes “SEQ” et sont suivis par un byte de versionning.
- 1 en-tête : noms des clé et classes des valeurs, détails de compression, marqueur sync, métadonnées…
- 1 ou plusieurs enregistrements + marqueurs sync (environ 1% de surcharge).
- Enregistrement avec ou sans compression, clés et valeurs sérialisés.

Sequence File

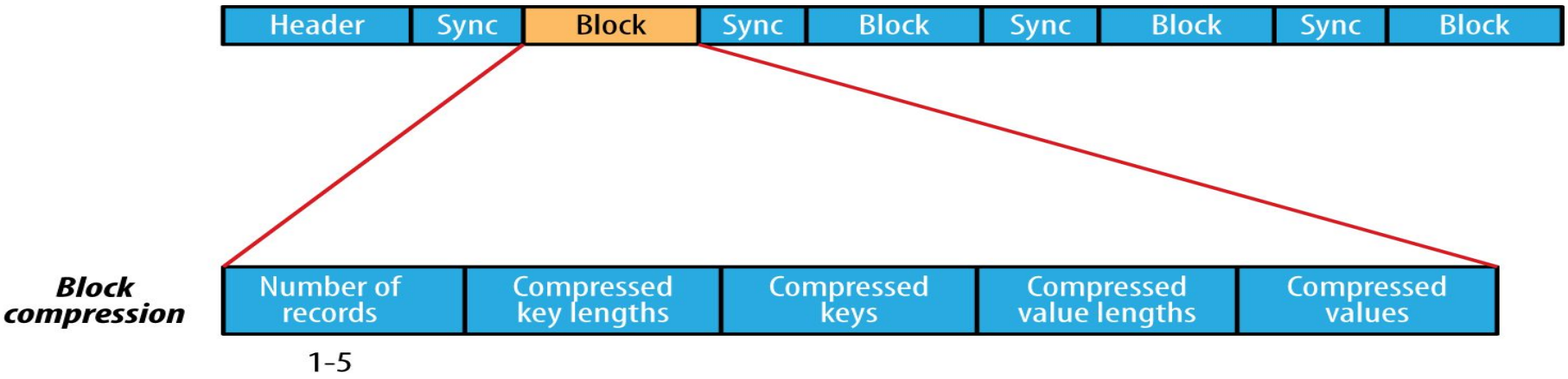


Sequence File - Compression par blocs

Saagie

- Compression de plusieurs enregistrements à la fois : plus compact et peut tirer parti des similitudes entre enregistrements.
- Enregistrements ajoutées au bloc jusqu'à atteindre une taille min de bytes.
- Bloc séparé par un marqueur sync.
- Format d'un bloc = 1 champ nombre d'enregistrement + 4 champs compressés.

Sequence File - Compression par blocs



Avro

Saagie

- Créé par Doug Cutting (encore lui) pour répondre à l'inconvénient des Writables : l'absence de portabilités.
- Données écrites avec un schéma indépendant.
- Génération de code optionnel ⇒ Possibilité de lire et écrire des données qui sont conformes à un schéma donné sans que le code en ait connaissance (le schéma doit toujours être présent).
- Schéma JSON.
- Système de sérialisation de données indépendant du langage.
- Évolution du schéma : schéma de lecture pas forcément identique au schéma d'écriture.
- Format container pour des séquences d'objets.
- Compressible et splittable.

Avro - Types primitifs

Saagie

null : no value

boolean : binary value

int : 32 bit integer

long : 64-bit signed integer

float : single precision (32-bit)

double : double precision (64- bit)

bytes : sequence of 8-bit unsigned bytes

string : unicode character sequence

Avro - Types complexes

Saagie

array	Collection ordonnées d'objets. Tous les objets d'un tableau particulier doivent avoir le même schéma	{
		"type": "array",
		"items": "long"
		}

map	Collection non ordonnée de paire clé-valeur. Clé forcément de type String, et valeurs n'importe quel type de même schéma	{
		"type": "map",
		"values": "string"
		}

record	Collection de champs nommés de n'importe quel type	{
		“type": "record",
		"name": "WeatherRecord",
		"doc": "A weather reading.",
		"fields": [
		{"name": "year", "type": "int"},
		{"name": "temperature", "type": "int"},
		{"name": "stationId", "type": "string"}
		]
		}

Avro - Types complexes

Saagie

enum	Ensemble de valeurs nommées	`{`
		`"type": "enum",`
		`"name": "Cutlery",`
		`"doc": "An eating utensil.",`
		`"symbols": ["KNIFE", "FORK", "SPOON"]`
		`}`

fixed	Nombre fixe de 8-bit non signé	`{`
		`"type": "fixed",`
		`"name": "Md5Hash",`
		`"size": 16`
		`}`

union	Tableau JSON , où chaque élément est un schéma. Données représentés par une union doivent correspondre à l'un des schémas	`[`
		`"null",`
		`"string",`
		`{"type": "map", "values": "string"}`
		`]`

Avro - Api Java

Saagie

- Mapping générique, utilisé quand le schéma utilisé n'est pas connu avant le runtime.
- Mapping spécifique, génération de code pour représenter les données pour un schéma Avro donné. Optimisation pratique quand le schéma est connu en avance.
- Mapping par réflexion, types Avro vers types Java.

Avro - Exemple de schéma

Saagie

```
{
"type": "record",
"name": "StringPair",
"doc": "A pair of strings.",
"fields":
  [
    {"name": "left", "type": "string"},
    {"name": "right", "type": "string"}
  ]
}
```

Avro - Writer Java générique

Saagie

```
Schema.Parser parser = new Schema.Parser();
Schema schema = parser.parse(
getClass().getResourceAsStream("StringPair.avsc"));
GenericRecord datum = new GenericData.Record(schema);

datum.put("left", "L");
datum.put("right", "R");

ByteArrayOutputStream out = new ByteArrayOutputStream();
DatumWriter<GenericRecord> writer =
new GenericDatumWriter<GenericRecord>(schema);
Encoder encoder = EncoderFactory.get().binaryEncoder(out, null);
writer.write(datum, encoder);
encoder.flush();
out.close();
```

Avro - Reader Java générique

Saagie

```
DatumReader<GenericRecord> reader = new GenericDatumReader<GenericRecord>(schema);
Decoder decoder = DecoderFactory.get().binaryDecoder(out.toByteArray(),null);

GenericRecord result = reader.read(null, decoder);

assertThat(result.get("left").toString(), is("L"));
assertThat(result.get("right").toString(), is("R"));
```

Avro - Writer Java spécifique

Saagie

```
StringPair datum = new StringPair();
datum.setLeft("L");
datum.setRight("R");

ByteArrayOutputStream out = new ByteArrayOutputStream();
DatumWriter<StringPair> writer =
new SpecificDatumWriter<StringPair>(StringPair.class);
Encoder encoder = EncoderFactory.get().binaryEncoder(out, null);
writer.write(datum, encoder);
encoder.flush();
out.close();
```

Avro - Reader Java spécifique

Saagie

```
DatumReader<StringPair> reader = new SpecificDatumReader<StringPair>(StringPair.class);
Decoder decoder = DecoderFactory.get().binaryDecoder(out.toByteArray(),null);

StringPair result = reader.read(null, decoder);

assertThat(result.getLeft(), is("L"));
assertThat(result.getRight(), is("R"));
```

Avro - Conteneur d'objets

Saagie

- Stocker des séquences d'objets Avro.
- Fichier de données portables : écrire en Python et le lire en C.
- Header avec métadonnées.
- Séries de blocs contenant les objets avros sérialisés.
- Blocs séparés par un marqueur sync ⇒ splittable.

```
File file = new File("data.avro");

DatumWriter<GenericRecord> writer = new
GenericDatumWriter<GenericRecord>(schema);

DataFileWriter<GenericRecord>
dataFileWriter = new
DataFileWriter<GenericRecord>(writer);

dataFileWriter.create(schema, file);
dataFileWriter.append(datum);
dataFileWriter.close();
```

Avro - Résolution de schéma

Saagie

Nouveau schéma	Writer	Reader	Action
Ajout de champ	Ancien	Nouveau	Le reader utilise la valeur par défaut du nouveau champ (car pas écrite par l'ancien).
	Nouveau	Ancien	Le reader ne connaît pas le nouveau champ, donc il l'ignore (projection).
Suppression de champ	Ancien	Nouveau	Le reader ignore le champ supprimé (projection).
	Nouveau	Ancien	Si l'ancien schema avait une valeur par défaut pour le champ, il l'utilise sinon erreur.

Parquet - Introduction

Saagie

- Format de stockage en colonne qui gère de manière efficace les données imbriqués.
- Très efficace en terme de taille de fichier et performance de requête (valeurs d'une colonne stockées les unes à côtés des autres).
- Champs imbriqués peuvent être lus indépendamment des autres champs.
- Spécification qui définit le format d'une manière neutre, plusieurs implémentations de cette spécification.

Parquet - Timeline

Saagie

- 2010 : papier sur Dremel.
- Automne 2012 : Twitter & Cloudera unissent leurs efforts pour développer un format en colonne.
- Mars 2013 : Intégration dans Hive.
- Juin 2013 : release 1.0, 18 contributeurs dans plus de 5 organisations.
- Mai 2014 : Apache Incubator. Plus de 40 contributeurs, 26 releases.
- 2015 : Parquet 2.0 arrivé en tant que release Apache
- Décembre 2015: Impala 2.3 gère les types complexes :)

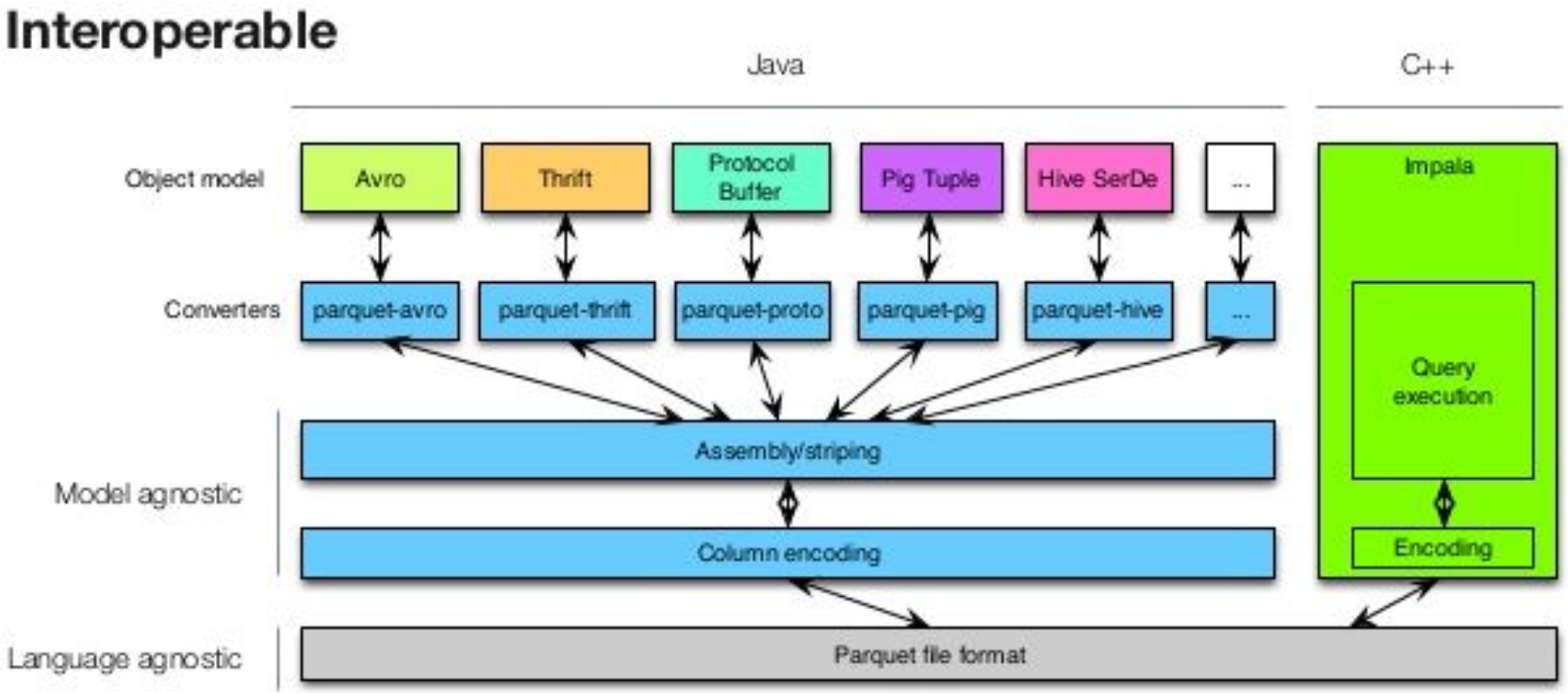
Parquet - Ecosystème

Saagie

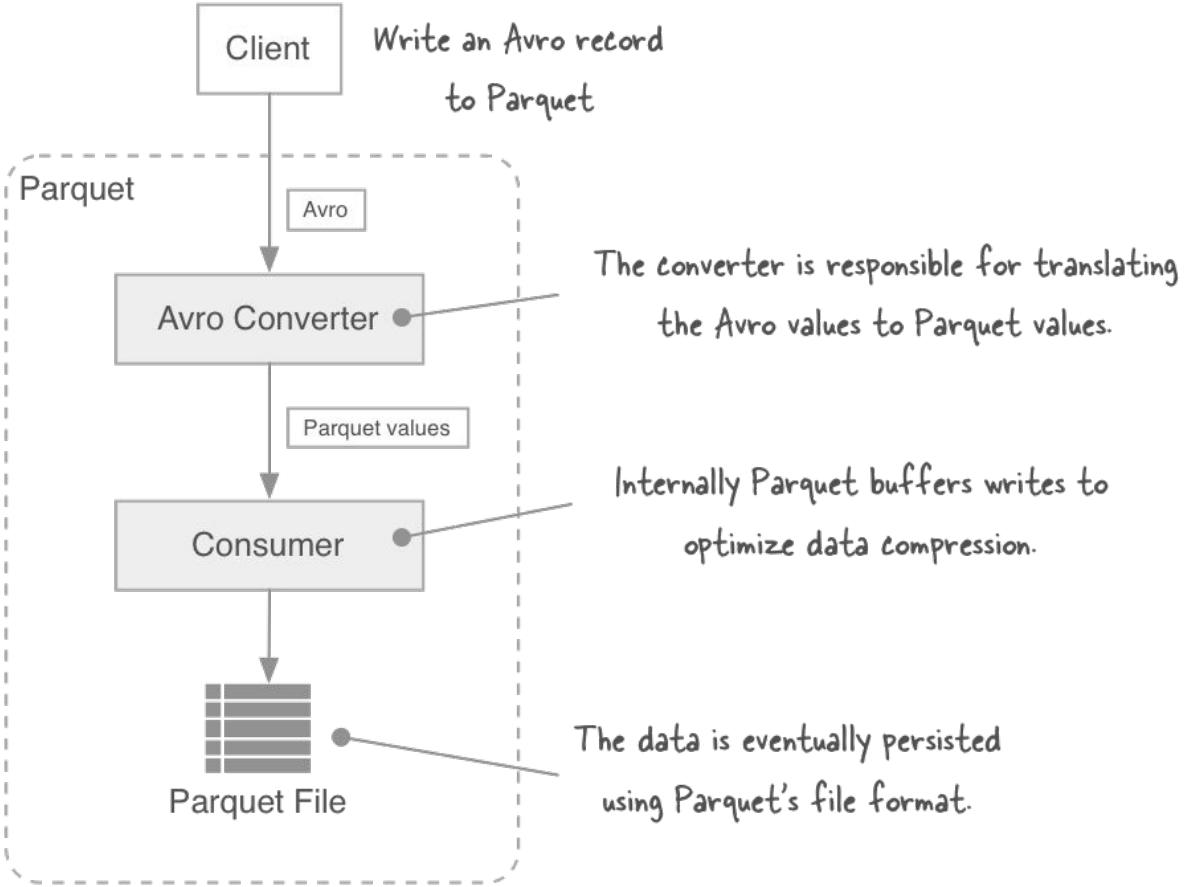
- Moteur de requetage : Hive, Impala, HAWQ, IBM Big SQL, Drill, Tajo, Pig, Presto, SparkSQL
- Frameworks : Spark, MapReduce, Cascading, Crunch, Scalding, Kite
- Modèle de données : Avro, Thrift, ProtocolBuffers, POJOs

Parquet - Intégration

Saagie



Parquet - Intégration



Parquet - Types primitifs

Saagie

boolean : Binary value

int32 : 32-bit signed integer

int64 : 64-bit signed integer

int96 : 96-bit signed integer

float : Single-precision (32-bit) IEEE 754 floating-point number

double : Double-precision (64-bit) IEEE 754 floating-point number

binary : Sequence of 8-bit unsigned bytes

fixed_len_byte_array : Fixed number of 8-bit unsigned bytes

Parquet - Types logiques

Saagie

Annotation type logique	Description	Exemple de schéma

UTF8	Chaîne de caractères UTF-8. Annote binary.	`message m {`
		`required binary a (UTF8);`
		`}`

ENUM	Ensemble de valeurs de données. Annote binary.	`message m {`
		`required binary a (ENUM);`
		`}`

DECIMAL(precision,scale)	Précision arbitraire d'un nombre décimal signé. Annote int32, int64, binary, ou fixed_len_byte_array.	`message m {`
		`required int32 a`
		`(DECIMAL(5,2));`
		`}`

DATE	Une date sans heure. Annote int32. Représenté par le nombre de jours depuis l'époque Unix (1 Janvier 1970).	`message m {`
		`required int32 a (DATE);`
		`}`

Parquet - Types logiques

Saagie

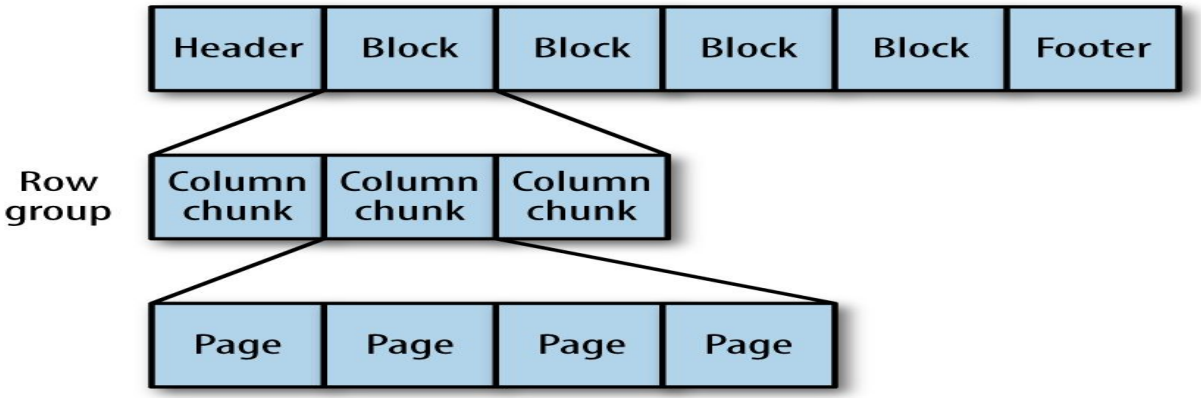
LIST	Une collection ordonnée de valeurs. Annote group	`message m {`
		`  required group a (LIST){`
		`  repeated group list {`
		`  required int32 element;`
		`  }`
		`  }`
		`}`

MAP	Une collection non ordonnée de paires clé-valeur. Annote group	`message m {`
		`required group a (MAP) {`
		`  repeated group key_value {`
		`  required binary key (UTF8);`
		`  optional int32 value;`
		`  }`
		`  }`
		`}`

Parquet - Structure interne

Saagie

- 1 header : nombre magique PAR1.
- Blocs de données.
- 1 footer :
  - version, schéma et métadonnées pour chaque bloc du fichier
  - fin de fichier : longueur du footer (4 bytes) + PAR1 ⇒ seek fin de fichier à la lecture
- Fichier splittable.



Parquet - Structure interne

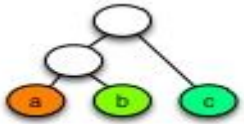
Saagie

- Un block contient un row group.
- Row group: partitionnement logique des données en ligne constitué de column chunk.
- Column chunk: données d'une colonne particulière contiguës dans le fichier.
- Page: partie des données d'une même colonne ⇒ bonne candidate pour la compression.
- 2 niveaux de compressions.

Parquet - Structure interne

Saagie

Columnar storage



Nested schema

Logical table representation

a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Row layout

a1	b1	c1	a2	b2	c2	a3	b3	c3	a4	b4	c4	a5	b5	c5

Column layout

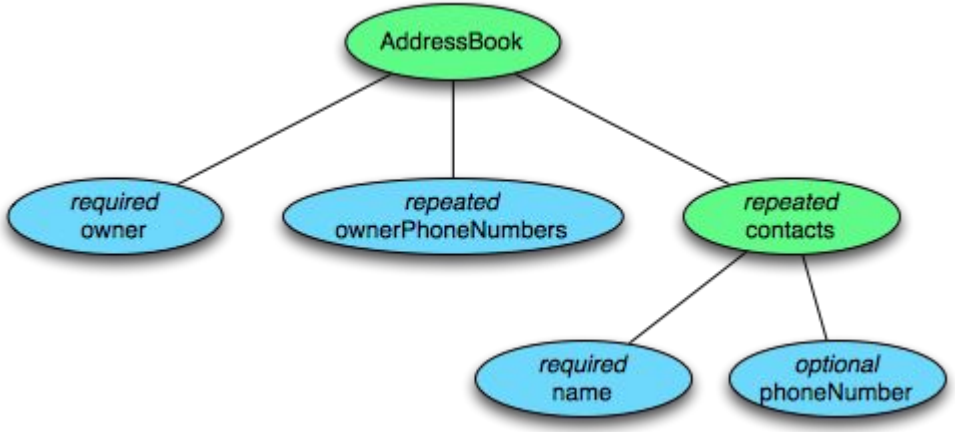
a1	a2	a3	a4	a5	b1	b2	b3	b4	b5	c1	c2	c3	c4	c5

encoding

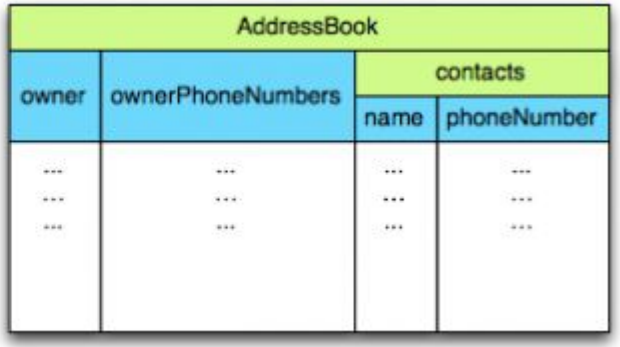
encoded chunk	encoded chunk	encoded chunk

Parquet - Structure imbriquées

Saagie



Column	Type
owner	string
ownerPhoneNumbers	string
contacts.name	string
contacts.phoneNumber	string

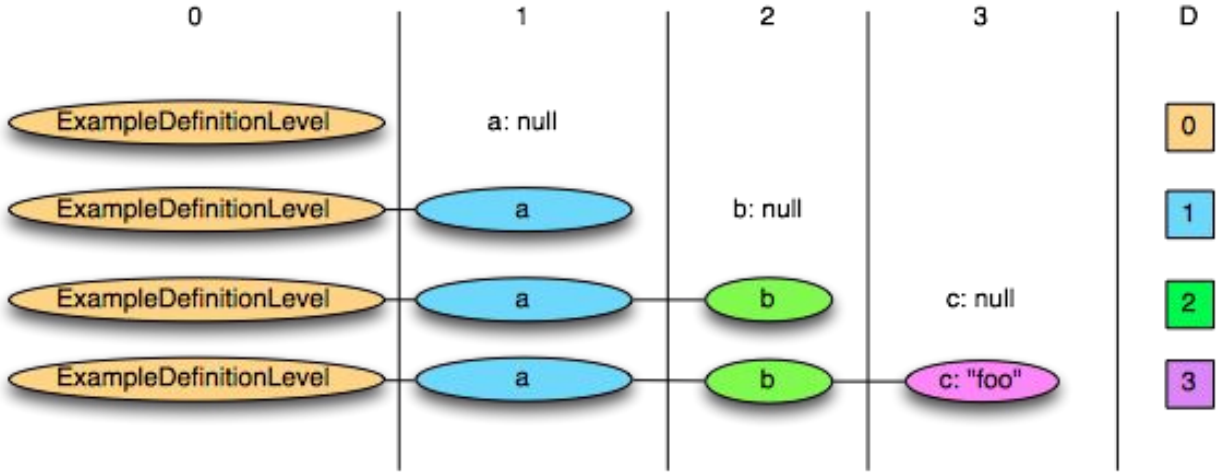


Parquet - Niveau de définition

Saagie

- Permet de supporter les champs imbriqués.
- Niveau pour lequel le champ n'est pas null.

```
message ExampleDefinitionLevel {
 optional group a {
   optional group b {
     optional string c;
   }
 }
}
```



Parquet - Niveau de répétition

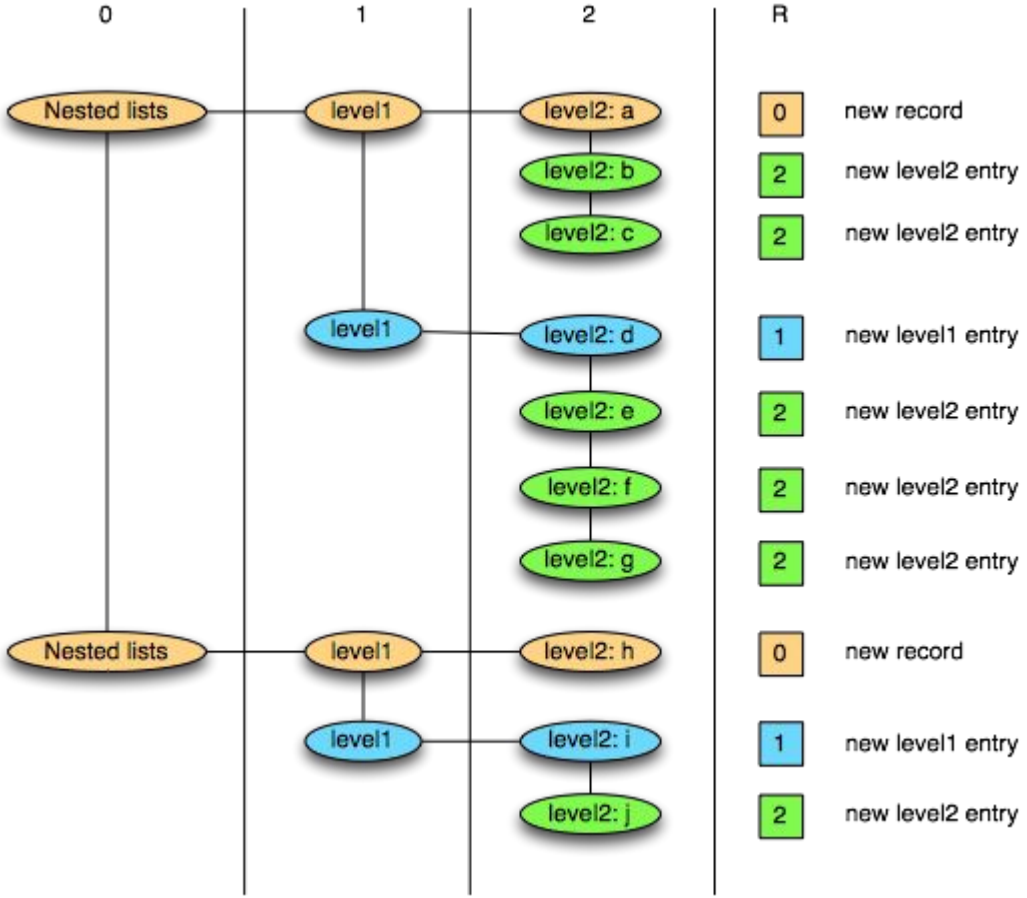
- Pour supporter les champs répétés, besoin de stocker lorsque de nouvelles listes commencent dans une colonne de valeur.

Schema:	**Data:** `[[a,b,c],[d,e,f,g]],[[h],[i,j]]`

`message nestedLists {`
`  repeated group level1 {`
`  repeated string level2;`
`  }`
`}`
	`{`
	`  level1: {`
	`  level2: a`
	`  level2: b`
	`  level2: c`
	`  },`
	`  level1: {`
	`  level2: d`
	`  level2: e`
	`  level2: f`
	`  level2: g`
	`  }`
	`}`
	`{`
	`  level1: {`
	`  level2: h`
	`  },`
	`  level1: {`
	`  level2: i`
	`  level2: j`
	`  }`
	`}`

Repetition level	Value
0	a
2	b
2	c
1	d
2	e
2	f
2	g
0	h
1	i
2	j

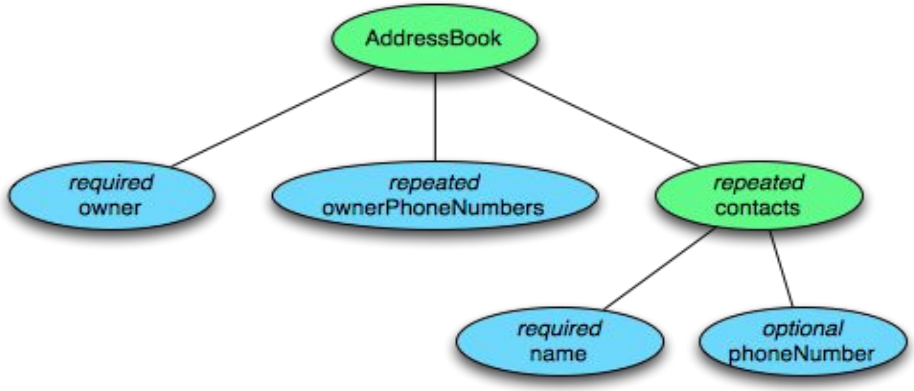
Parquet - Niveau de répétition



Parquet - Exemple de champs imbriqués

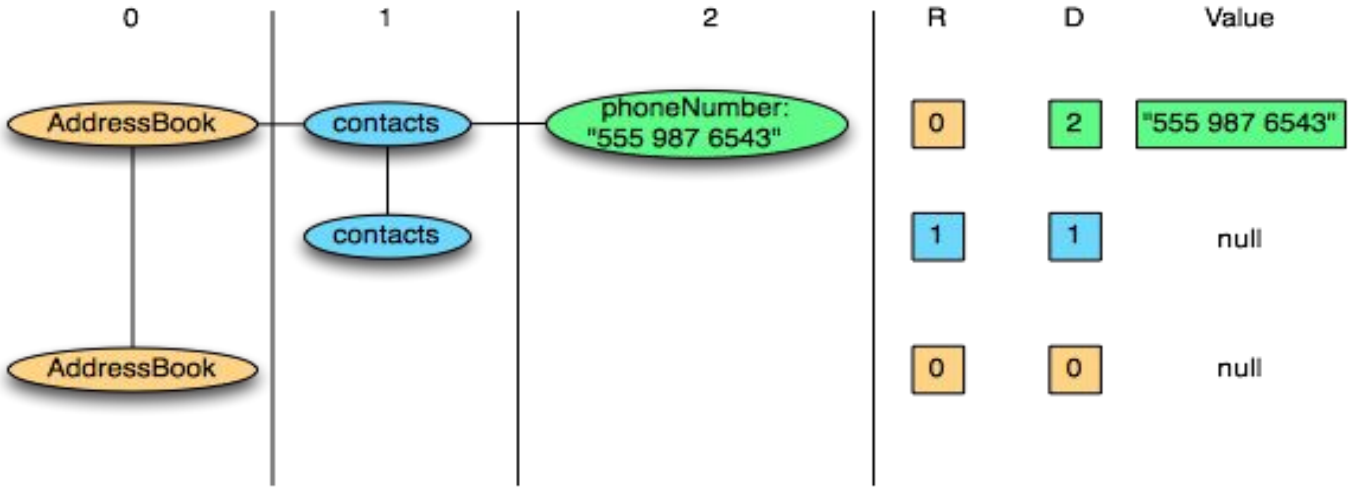
Saagie

```
AddressBook {
 owner: "Julien Le Dem",
 ownerPhoneNumbers: "555 123 4567",
 ownerPhoneNumbers: "555 666 1337",
 contacts: {
   name: "Dmitriy Ryaboy",
   phoneNumber: "555 987 6543",
 },
 contacts: {
   name: "Chris Aniszczyk"
 }
}
AddressBook {
 owner: "A. Nonymous"
}
```



Column	Max Definition level	Max Repetition level
owner	0 (owner is *required*)	0 (no repetition)
ownerPhoneNumbers	1	1 (*repeated*)
contacts.name	1 (name is *required*)	1 (contacts is *repeated*)
contacts.phoneNumber	2 (phoneNumber is *optional*)	1 (contacts is *repeated*)

Parquet - Exemple de champs imbriqués



Parquet - Exemple de champs imbriqués

Saagie

Pour construire les enregistrements à partir de la colonne, nous la parcourons :

- R=0, D=2, Value = “555 987 6543”:
  - R = 0 signifie un nouvel enregistrement, création des enregistrements imbriqués à partir du root jusqu’au niveau de définition (ici 2)
  - D = 2 qui correspond au max signifie que la valeur est définie et insérée
- R=1, D=1:
  - R = 1 signifie une nouvelle entrée dans la liste des contacts
  - D = 1 signifie que le contact est défini mais pas le phoneNumber, contact vide
- R=0, D=0:
  - R = 0 signifie un nouvel enregistrement, création des enregistrements imbriqués à partir du root jusqu’au niveau de définition
  - D = 0 car contacts est actuellement null, donc nous avons une AddressBook vide

Parquet - Encoding

Saagie

- Delta encoding : pour les ensembles de données triées où la variation est moins importante que la valeur absolue (timestamp, id auto généré, métriques…).
- Run length encoding : valeurs répétitives encodées en une seule + nb de répétitions.
- Dictionary encoding : dictionnaire de valeurs lui même encodé, les valeurs sont codés comme des nombres entiers représentant les indices dans le dictionnaire (server ip, enum…).
- Bit packing.
- Encoding choisit automatiquement basé sur le type de colonne (exemple: boolean = run length encoding + bit packing).

Parquet - Delta encoding

Saagie

Delta encoding

values: 100 101 101 102 101 101 102 101 99 100 105 107 114 116 119 120 121

reference: 100

block 1: 101 101 102 101 101 102 101 99

block 2: 100 105 107 114 116 119 120 121

8 * 64bits values = 64 bytes

deltas

8 * 64bits values = 64 bytes

100 | 1 0 1 -1 0 1 -1 -2 | 1 5 2 7 2 3 1 1

Delta encoding

100 | 1 0 1 -1 0 1 -1 -2 | 1 5 2 7 2 3 1 1

min delta

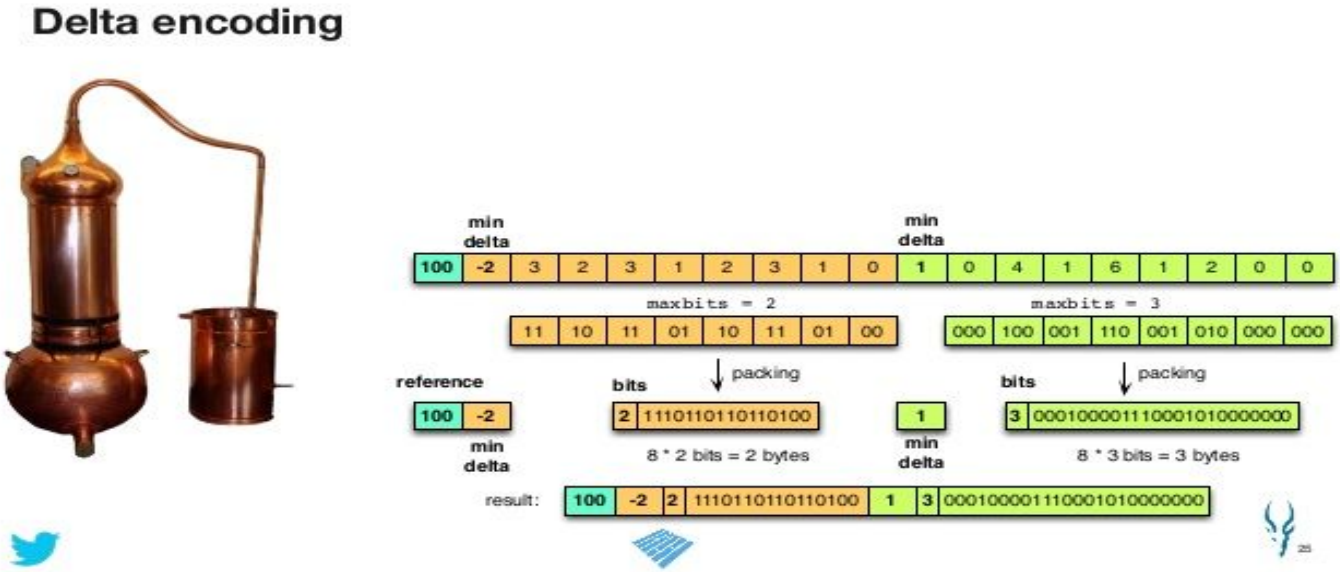
make deltas > 0 by subtracting min

min delta

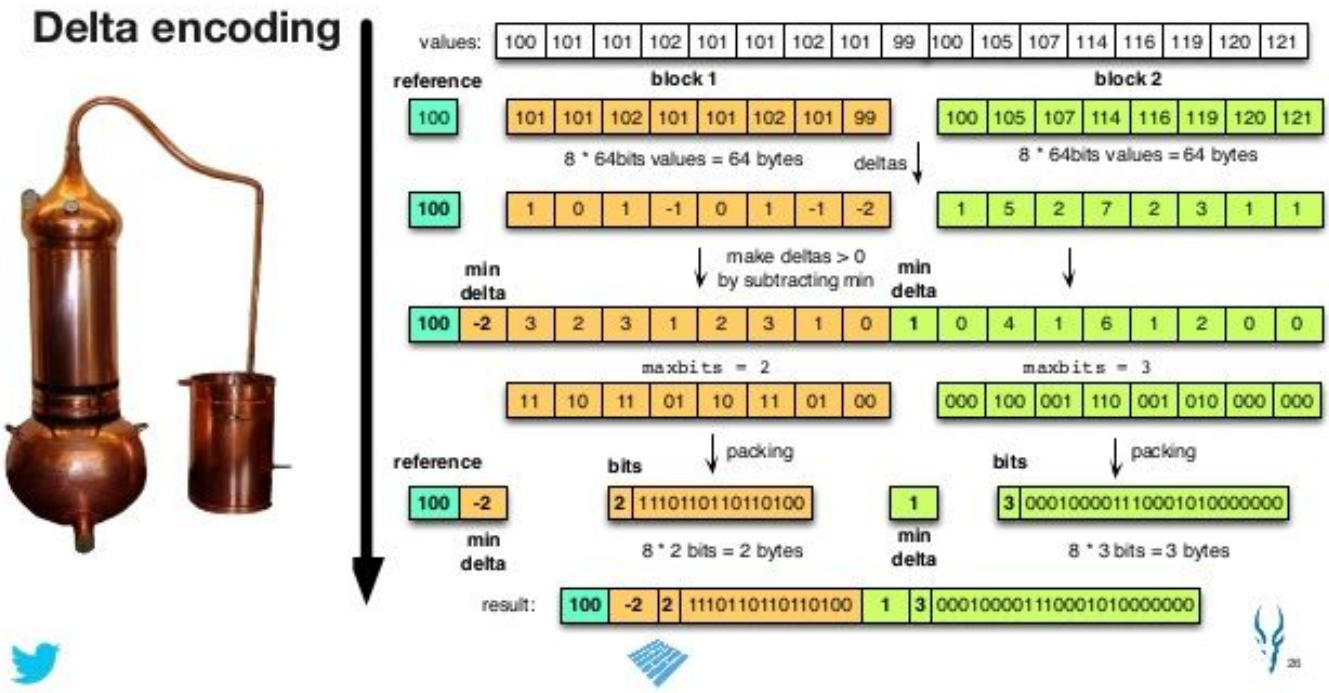
100 -2 3 2 3 1 2 3 1 0 1 0 4 1 6 1 2 0 0

Parquet - Delta encoding

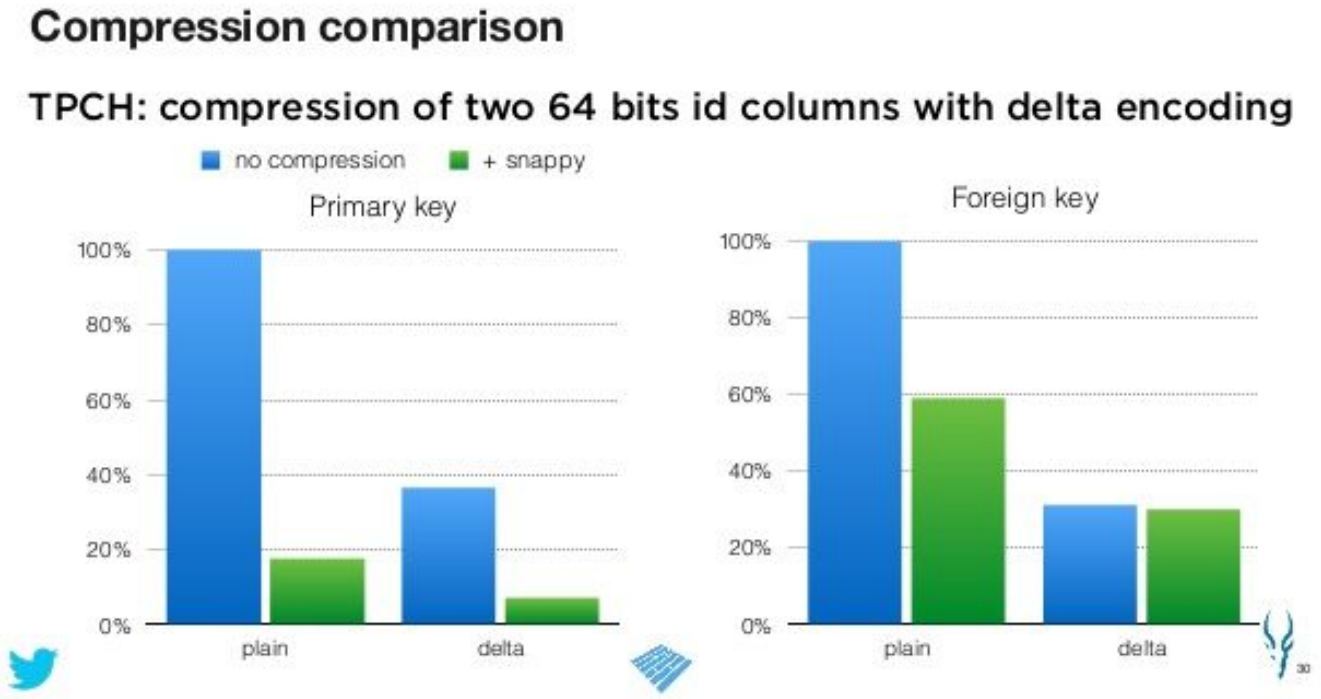
Saagie



Parquet - Delta encoding



Parquet - Benchmark



Parquet - Ecriture

Saagie

```
MessageType schema = MessageTypeParser.parseMessageType(
"message Pair {\n" +
" required binary left (UTF8);\n" +
" required binary right (UTF8);\n" +
"}");

GroupFactory groupFactory = new SimpleGroupFactory(schema);
Group group = groupFactory.newGroup()
.append("left", "L")
.append("right", "R");
```

Parquet - Ecriture

Saagie

```
Configuration conf = new Configuration();
Path path = new Path("data.parquet");
GroupWriteSupport writeSupport = new GroupWriteSupport();
writeSupport.setSchema(schema, conf);

ParquetWriter<Group> writer = new ParquetWriter<Group>(path, writeSupport,
ParquetWriter.DEFAULT_COMPRESSION_CODEC_NAME, ParquetWriter.DEFAULT_BLOCK_SIZE,
ParquetWriter.DEFAULT_PAGE_SIZE, ParquetWriter.DEFAULT_IS_DICTIONARY_ENABLED,
ParquetWriter.DEFAULT_IS_VALIDATING_ENABLED,
ParquetProperties.WriterVersion.PARQUET_1_0, conf);

writer.write(group);
writer.close();
```

Parquet - Lecture

Saagie

```
GroupReadSupport readSupport = new GroupReadSupport();
ParquetReader<Group> reader = new ParquetReader<Group>(path, readSupport);
Group result = reader.read();

assertNotNull(result);
assertThat(result.getString("left", 0), is("L"));
assertThat(result.getString("right", 0), is("R"));
assertNull(reader.read());
```

De Avro vers Parquet

Saagie

```
Schema.Parser parser = new Schema.Parser();
Schema schema = parser.parse(getClass().getResourceAsStream("StringPair.avsc"));
GenericRecord datum = new GenericData.Record(schema);
datum.put("left", "L");
datum.put("right", "R");

// Avro → Parquet
Path path = new Path("data.parquet");
AvroParquetWriter<GenericRecord> writer = new AvroParquetWriter<GenericRecord>(path, schema);
writer.write(datum);
writer.close();

// Parquet → Avro
AvroParquetReader<GenericRecord> reader = new AvroParquetReader<GenericRecord>(path);
GenericRecord result = reader.read();
assertThat(result.get("left").toString(), is("L"));
assertThat(result.get("right").toString(), is("R"));
```

Parquet - Projection

Saagie

```
Schema projectionSchema = parser.parse(
getClass().getResourceAsStream("ProjectedStringPair.avsc")
);

Configuration conf = new Configuration();
AvroReadSupport.setRequestedProjection(conf,projectionSchema);

AvroParquetReader<GenericRecord> reader =
new AvroParquetReader<GenericRecord>(conf, path);

GenericRecord result = reader.read();
assertNull(result.get("left"));
assertThat(result.get("right").toString(), is("R"));
```

ORC File : Optimized Row-Columnar File

Saagie

- Créé par Hortonworks en Janvier 2013 pour améliorer les performances de rapidité d'Apache Hive et l'efficacité du stockage des données dans Hadoop.
- Format de fichier colonnaire auto-descriptif.
- Optimisé pour des grandes lectures en streaming, mais avec un support intégré pour trouver rapidement les lignes requises.
- Permet au lecteur de lire, décompresser et traiter seulement les valeurs qui sont nécessaires pour la requête en cours.
- Choisit le codage le plus approprié pour le type et crée un index interne lors de l'écriture.

ORC File - Type

Saagie

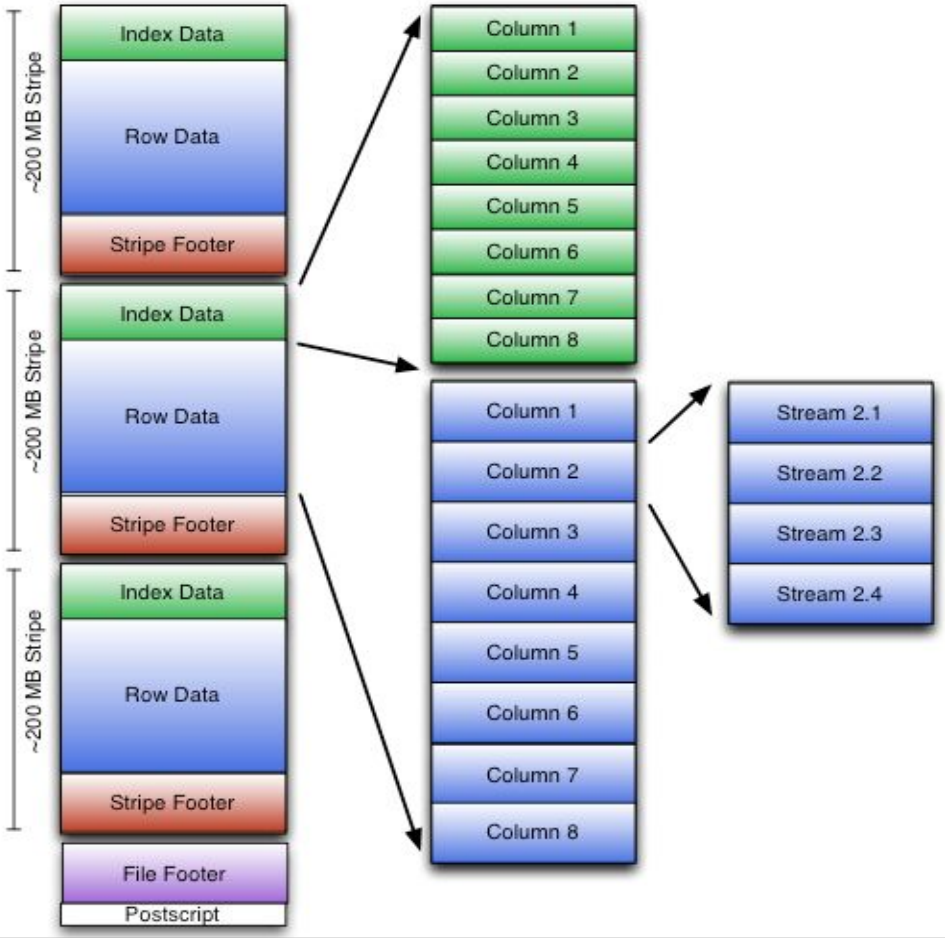
- Integer
  - boolean (1 bit)
  - tinyint (8 bit)
  - smallint (16 bit)
  - int (32 bit)
  - bigint (64 bit)
- Floating point
  - float
  - Double
- String types
  - string
  - char
  - varchar
- Binary blobs
  - Binary
- Date/time
  - timestamp
  - Date
- Compound types
  - struct
  - list
  - map
  - union

ORC - Structure

Saagie

- Fichier divisé en stripe d'environ 64 Mo par défaut.
- Les stripes dans un fichier sont indépendants les uns des autres.
- Dans chaque stripe, les colonnes sont séparées les unes des autres de sorte que le lecteur puisse lire seulement les colonnes qui sont nécessaires.

ORC - Structure



ORC - Indexes

Saagie

- Niveau fichier - des statistiques sur les valeurs dans chaque colonne dans le fichier entier.
- Niveau stripe - des statistiques sur les valeurs dans chaque colonne pour chaque stripe.
- Niveau de la ligne - des statistiques sur les valeurs dans chaque colonne pour chaque ensemble de 10.000 lignes dans une stripe.

ORC - Indexes

Saagie

- Les statistiques niveau fichier et niveau stripe sont dans le footer de sorte qu'ils soients faciles d'accès pour déterminer si le reste du fichier doit être lu.
- Les index de niveau de ligne comprennent à la fois les statistiques de colonne pour chaque groupe de lignes et la position pour chercher le début du groupe de lignes.
- Les statistiques de colonne contiennent toujours le nombre de valeurs et s'il y a des valeurs nulles présentes. La plupart des types primitifs contiennent les valeurs minimales et maximales.
- Les indexs de tous les niveaux sont utilisés par le reader en utilisant des arguments de recherche.

ORC : Pushdown

Saagie

- Les requêtes SQL ont généralement un certain nombre de conditions WHERE qui peuvent être utilisés pour éliminer facilement des lignes.
- Grâce aux indexs, ORC accepte les prédicats pushdown.
    - Les indexes permettent de déterminer quels stripes dans un fichier doivent être lus et les indexs de ligne peuvent restreindre la recherche à un ensemble particulier de 10.000 lignes.

ORC - Index Bloom filter

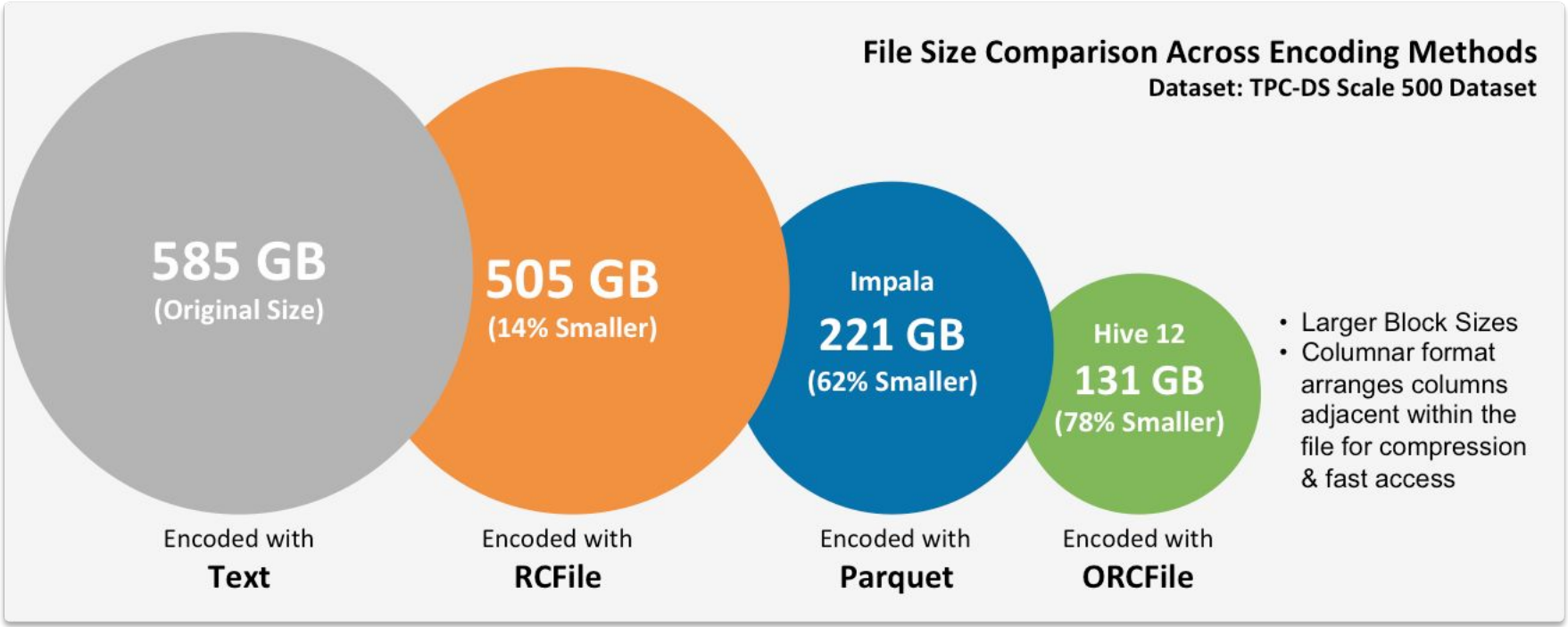
Saagie

- Ajouté aux indexes ORC à partir de Hive 1.2.0.
- Un prédicat pushdown peut en faire usage pour mieux réduire les groupes de lignes qui ne satisfont pas la condition de filtre.
- Les indexes bloom filter se composent d'un flux bloom filter pour chaque colonne spécifiée dans les propriétés de la table.
- Un flux enregistre une entrée bloom filter pour chaque groupe de lignes (par défaut 10.000 lignes) dans une colonne.
- Seuls les groupes de lignes qui satisfont l'évaluation min / max de l'index de ligne seront évaluées par l'index bloom filter.

http://billmill.org/bloomfilter-tutorial/

ORC - Compression

Saagie



HIVE

Introduction

Saagie

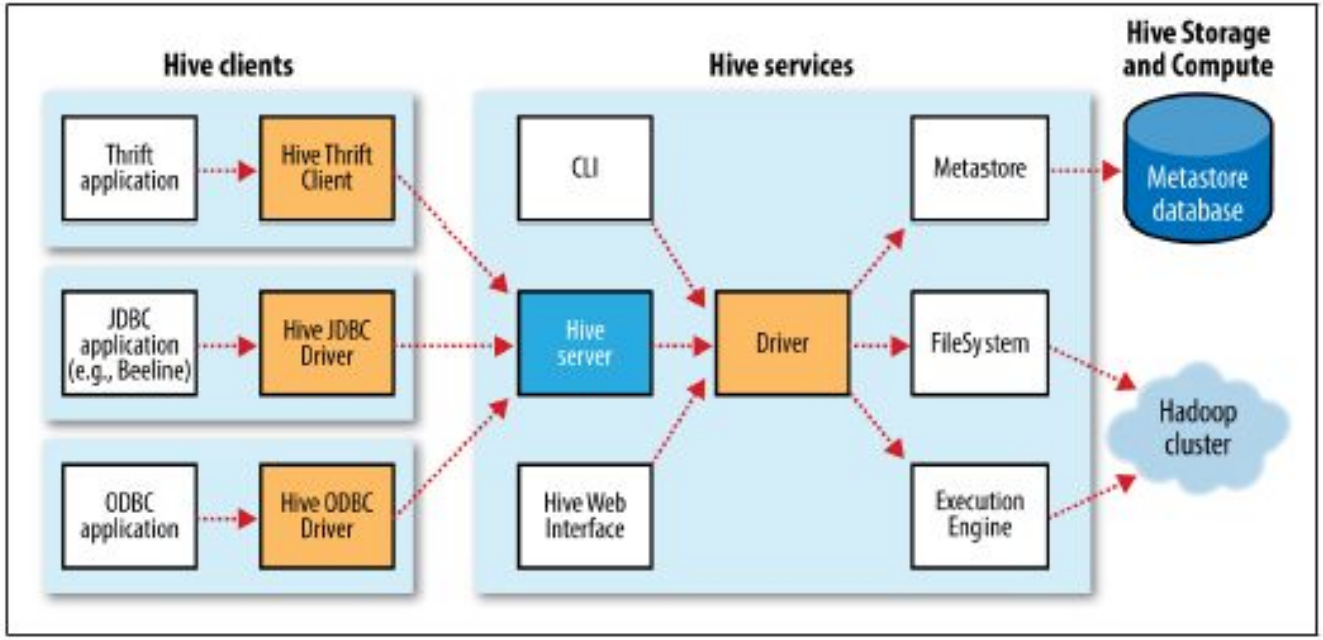
- Requêteur SQL sur les données stockées dans le HDFS.
- Initialement créé pour gérer les grandes quantités d'informations générées sur Facebook.
- Ecrit en Java.
- Le fichier hive-site.xml contient tous les paramètres de Hive. Il est possible de le manipuler directement ou via la commande SET.
- Chaque requête génère un ou n scripts Map Reduce.
- Hive se base sur des jobs MapReduce qui sont exécutés sur YARN. Ce dernier se charge de fournir les ressources pour exécuter les requêtes.

Services

Saagie

- CLI (Command Line Interface)
  - Permet d'exécuter des commandes Hive en ligne de commande.
- Hiveserver2
  - Service permettant aux différents clients d'interagir avec Hive core. Hiveserver gère l'authentification et l'accès concurrentiel. Par défaut le port de communication de HiveServer2 est le 10000.
- Metastore
  - Base de données stockant les métadonnées des tables. La base de données peut être hébergé sur Derby, Postgresql (defaut) ou Mysql.

Clients


Hive clients
Hive services
Hive Storage and Compute
Thrift application
Hive Thrift Client
JDBC application (e.g., Beeline)
Hive JDBC Driver
ODBC application
Hive ODBC Driver
CLI
Hive server
Hive Web Interface
Driver
Metastore
FileSystem
Execution Engine
Metastore database
Hadoop cluster

Stockage

Saagie

- Table qui est composée d'un ou n fichiers stockés sur le HDFS (par défaut dans le répertoire /user/hive/warehouse).
- Les tables sont elles même stockées dans des schémas (par défaut dans le schéma Default).
- Hive a la possibilité de prendre des données en dehors de ce répertoire, ce sont les tables externes.
    - requêtage transparent pour les utilisateurs, la différence réside dans la suppression de la table ; les données de la table interne sont supprimées, les données de la table externe ne le sont pas
- Le format de stockage par défaut est le ROW FORMAT DELIMITED. Chaque champ est séparé par défaut par le caractère ASCII '001' et les lignes sont séparées par le saut de ligne '\n'.
- D'autres formats de stockages sont disponibles (avro, parquet, orc ...).

Metastore

Saagie

- Le metastore stocke les informations sur les tables Hive. Quelques exemples de métadonnées stockées :
  - tableName
  - owner
  - inputFormat/outputFormat
  - totalSize
- Peut être stocké sur un SGBD indépendant (Mysql, Postgesql, …).
- Le metastore peut fonctionner de 3 manières différentes :
  - embarqué
  - local
  - remote

SQL vs HQL

Saagie

Feature	SQL-92	HiveQL
Updates	UPDATE, INSERT, DELETE	UPDATE*, INSERT, DELETE* * Format ORC uniquement
Indexes	Supported	Supported
Data types	Integral, floating-point, fixedpoint, text and binary strings, temporal	Boolean, integral, floatingpoint,bfixed-point, text andbinary strings, temporal, array,map, struct
Functions	Hundreds of built-in functions	Hundreds of built-in functions
Multitable inserts	Not supported	Supported
CREATE TABLE...AS SELECT	Not valid SQL-92, but found in some databases	Supported
SELECT	SQL-92	Supported
Joins	SQL-92, or variants (join tables in the FROM clause, join condition in the WHERE clause)	Inner joins, outer joins, semi joins, map joins, cross joins
Subqueries	Supported	Supported
Views	Updatable (materialized or nonmaterialized)	Read-only (materialized views not supported)

Insertion de données

Saagie

- Insertion des données de manière unitaire :
  - Ce type de chargement génère 1 fichier par commande INSERT. Il est recommandé d'utiliser le chargement unitaire sur des tables de petites tailles. L'insertion n'est pas parallélisée.
  - `INSERT INTO TABLE students  VALUES ('fred', 35, 1.28), ('barney', 32, 2.32)`
- Insertion via une requête :
  - Ce type de chargement est performant mais impose que la donnée soit déjà présente dans le metastore.
  - `INSERT INTO TABLE students SELECT name, id FROM people WHERE  job = 'student'`
- Insertion via un fichier
  - Ce type de chargement permet de charger des tables à partir d'un fichier dans le HDFS ou en local. Hive déplacera le fichier dans le répertoire par défaut. Lors du chargement Hive ne vérifie pas que le fichier est compatible avec les données du metastore (séparateur, nombre de champs, …).
  - `LOAD DATA LOCAL INPATH 'input/ncdc/micro-tab/sample.txt'`
  - `OVERWRITE INTO TABLE records`

Impala

Présentation

Saagie

- Requêteur SQL basé sur les fichiers dans le HDFS.
- N'utilise pas de MapReduce mais du MPP.
- N'utilise pas non plus de Master mais uniquement des "démons". Chaque démon peut faire office de master et de worker.
- Impala est basé sur une infrastructure proche de la machine, en LLVM (Low Level Virtual Machine) et codé en C++.
- Impala est utilisé pour du requêtage et de l'analyse de données. Hive pour de la manipulation de données en batch.
- Impala peut partager le même metastore que Hive.

Clients

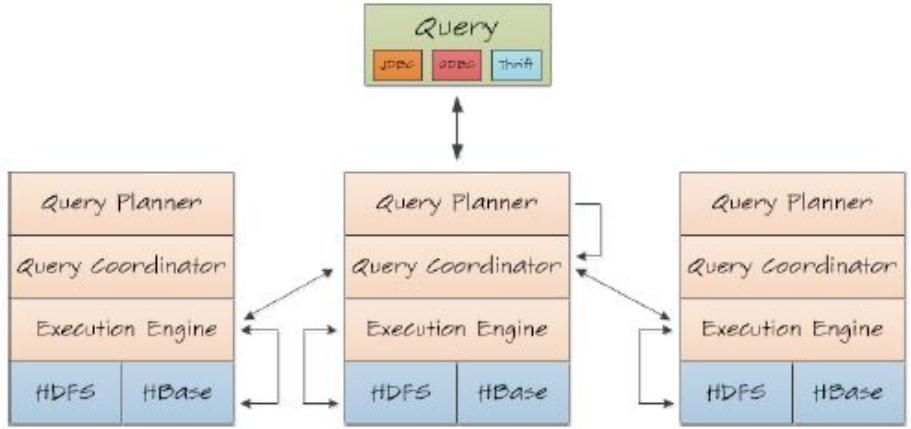
Saagie

- Impala-shell
- Hue
- JDBC driver : Ce client est utilisé par les applications Java.
- ODBC Driver : Ce client est utilisé pour le protocole ODBC.
- Impyla : Client pour Python
- Impala-ruby : Client pour Ruby
- RODBC : ODBC pour langage R

Architecture

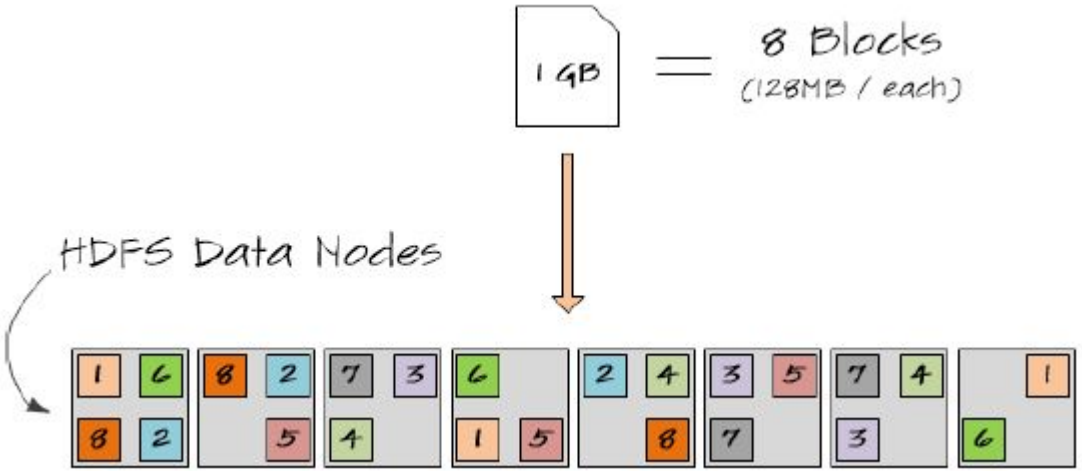
Saagie

- Démon s'exécutant sur chaque machine. Chaque démon a le rôle d'exécuteur de requête. Un seul démon a le rôle de coordinateur.
- Chaque démon contient les éléments suivants :
  - Query Planner : permet de définir un schéma d'exécution à la requête
  - Query Coordinator : fait office de "Reducer" mais a aussi le rôle d'Execution Engine pendant l'exécution d'une requête
  - Execution Engine : exécuteur de requête



Localité des données

- Chaque démon est capable de chercher les données stockées sur son disque ce qui évite les transferts réseaux.
- Par défaut les blocs HDFS sont dupliqués 3 fois.



Services

Saagie

- L’intégrité des services Impala est assuré par le Statestore :
  - processus unique qui tourne sur une seule machine
  - son rôle est de s’assurer de la “bonne santé” des démons
  - en cas de défaillance d’un démon le statestore informe tous les autres démons
- Impala et Hive partagent le même Metastore.
  - le service Catalog se charge de lire et de de mettre à jour les données du metastore
- Le Statestore et le Catalog ne connaissent pas à l’avance leurs démons, ce sont les démons qui s’enregistrent à eux. Cette solution permet d'ajouter ou supprimer des démons dynamiquement.

Exécution

Saagie

- QUERY ANALYSIS : Analyse de la syntaxe SQL.
- PLANNING : Permet de de définir un plan d'éxécution de la requête. Il est possible de ne pas exécuter la requête et de voir le plan d'exécution en ajoutant la commande EXPLAIN au début de celle-ci.
- COORDINATION : Rôle assuré par le démon interrogé. La coordination envoie les travaux à tous les autres démons.
- EXECUTORS : Lisent les données sur le disque et effectuent les travaux.
- FINAL RESULT : Les résultats de chacun des démons est retourné au coordinateur.

Création de table

Saagie

```
CREATE TABLE tab1
    (
       id INT,
       col_1 BOOLEAN,
       col_2 DOUBLE,
       col_3 TIMESTAMP
    )

CREATE TABLE tab2 LIKE PARQUET ‘/user/hdfs/fil.dat’

CREATE TABLE tab4 LIKE tab3


CREATE TABLE tab3 AS SELECT var1, var3, COUNT(*) FROM tab6 GROUP BY 1,2
```

Création de table - Options

Saagie

- IF NOT EXISTS : Ne crée pas la table si elle existe déjà.
- EXTERNAL : Crée une table en dehors du répertoire /user/hive/warehouse. Le fichier ne sera pas supprimé si on supprime la table.
- PARTITIONED BY (col_name) : Un champ de la table sert de partition. 1 répertoire par valeur de partition est créé dans le répertoire /user/hive/warehouse/<table>. Permet d'augmenter les performances pour les clauses WHERE.
- ROW FORMAT : Permet de sélectionner le délimiteur dans une table au format texte.
- STORED AS : Permet de définir le format de stockage (ex : Parquet).

Insertion de données

Saagie

- INSERT INTO/OVERWRITE
  - Ce type de chargement est performant mais impose que la donnée soit déjà présente dans le metastore. Le mot clé INTO permet d'ajouter des données, en le remplaçant par OVERWRITE les données sont remplacées.
- LOAD DATA INPATH INTO/OVERWRITE TABLE
  - Permet de charger des fichiers plats dans le HDFS dans une table. Pour les formats Avro, RCFile et SequenceFile le fichier à charger doit déjà être au bon format. Le schéma des fichiers n'est pas vérifié au chargement. Le fichier est déplacé vers le répertoire Hive.
- INSERT VALUES
  - Ce type de chargement génère 1 fichier par commande INSERT. Il est recommandé d'utiliser le chargement unitaire sur des tables de petites tailles. L'insertion n'est pas parallélisée.

Fonctions

Saagie

- Exemples de fonctions mathématiques :
  - ABS() : Valeur absolue
  - EXP() : Exponentiel
  - FLOOR() : Tronque la valeur
  - ROUND() : Arrondi la valeur
  - FMOD() : Modulo
  - GREATEST(val a[, val b ...]) : Renvoie la valeur la plus élevée
  - LEAST(val a[, val b ...]) : Renvoie la valeur la plus basse
  - RAND() : Renvoie une valeur aléatoire
- Exemples de fonctions d'agrégat :
  - AVG() : Moyenne
  - COUNT([DISTINCT | ALL]) : Compte
  - MAX(), MIN() : Maximum, minimum
  - SUM() : Somme

Fonctions analytiques

Saagie

```
SELECT ownerid, `timestamp`,
ROW_NUMBER() OVER(
 PARTITION BY ownerid ORDER BY `timestamp`
)
FROM cart
GROUP BY ownerid, `timestamp`;
```

id	timestamp	rownumber
3	1425044112395	1
3	1425044322164	2
3	1428932739115	3
4	1425045549400	1
4	1425046145272	2
4	1425046607686	3
5	1425045560553	1
6	1425045813028	1
6	1425052538694	2
6	1425055359006	3
6	1425305133232	4
6	1425306162346	5
6	1425311494464	6

Fonctions analytiques

```
SELECT x, DENSE_RANK() OVER(PARTITION BY
property ORDER BY x) as rang, property

FROM table1;
```

x	rank	property
2	1	even
4	2	even
6	3	even
8	4	even
10	5	even
7	1	lucky
7	1	lucky
7	1	lucky
1	1	odd
3	2	odd
5	3	odd
7	4	odd
9	5	odd

Opérationnel

Saagie

- INVALIDATE METADATA : Lorsqu'une table est créée dans Hive elle n'est pas visible dans Impala. La commande permet à Impala de relire le metastore.
- REFRESH TABLE : Permet de mettre à jour les données lorsqu'il y a de nouveaux fichiers dans le répertoire assigné à la table.

Optimisations

Saagie

- COMPUTE STATS : Permet à Impala d'établir des statistiques sur les tables (taille des tables, nombres de valeurs distincts par champs, …) et ainsi d'améliorer les plans d'exécution.
- SHOW TABLE STATS : Montre les stats d'une table une fois la commande COMPUTE STATS effectuée.
- LIMIT : A rajouter dans la clause SELECT pour limiter le nombre de lignes retournée. Intéressant pour avoir un échantillon.
- Préférer des tables uniques avec redondance plutôt qu'un schéma type OLTP.
- Le format et la compression des données peuvent influencer sur les performances.

Cas d'usages : Hive vs Impala

Saagie

Privilégier Hive pour :

- écrire de gros volumes de données.
- écrire dans des formats non gérés par Impala
- requêter sur d'énormes volumes de données (lorsque la mémoire RAM est saturée)

Privilégier Impala pour :

- interrogation SQL rapides
- sur de gros volumes de données