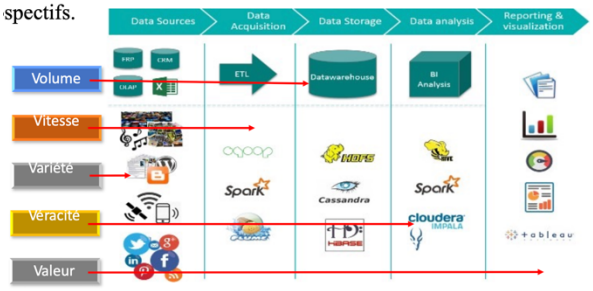
Big Data.

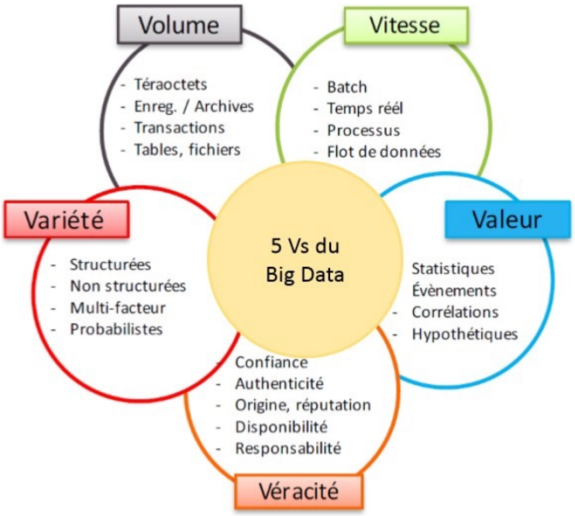
L’informatique décisionnelle:

Business Intelligence

L'informatique décisionnelle, aussi appelée business intelligence(BI), désigne un ensemble de méthodes, de moyens et d'outils informatiques utilisés pour piloter une entreprise et aider à la prise de décision : tableaux de bord, rapports analytiques et prospectifs.

spectifs.





Les piliers de l’architecture BIG DATA

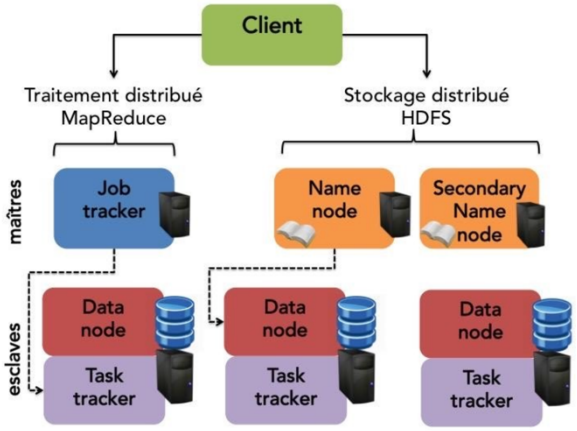
Stockage HDFS/ Amazon/ Azure
Traitement MapReduce/ Spark
Gestion des ressources et planification Yarn/hadoop 2.0
Master/ Name nodes
Slaves/ Data nodes

ECOSYSTÈME HADOOP

Hadoop est un système de gestion de données et de traitements distribués.
Il contient de beaucoup de composants, dont :

- HDFS un système de fichier qui répartit les données sur de nombreuses machines,
- Map-Reduce : Le but est de recueillir une information synthétique à partir d'un jeu de données.
- YARN un mécanisme d'ordonnancement de programmes de type MapReduce

Traitement vs stockage



ECOSYSTÈME HADOOP

Sqoop
-Importe/exporte des données d'une BD automatiquement
• **RDBS ↔ HDFS**
-Exemple: une application web/mySQL

Flume
-Collecter des données de sources et importer dans HDFS
-Logs.

HBase
-Une base de donnée NoSQL (clef/valeur)
-Distribuée
-Sans limite pratique pour la taille des tables
-Intégration avec Hadoop

Oozie
Orchestrer des séquences de tâches MapReduce
Tâches oozie: un graphe orienté acyclique d'actions
Peut être lancée par des évènements ou à un certain temps
• **Àl'ajoutd'unfichierfaire...**
• **À tous les jours à 3h00AM faire...**

Chukwa
-Système de collection de données distribuées
-Opimiser Hadoop pour traiter des log
-Afficher, monitorer et analyser les fichiers log

The master nodes typically utilize higher quality hardware and include a NameNode, Secondary NameNode, and JobTracker,

JobTracker

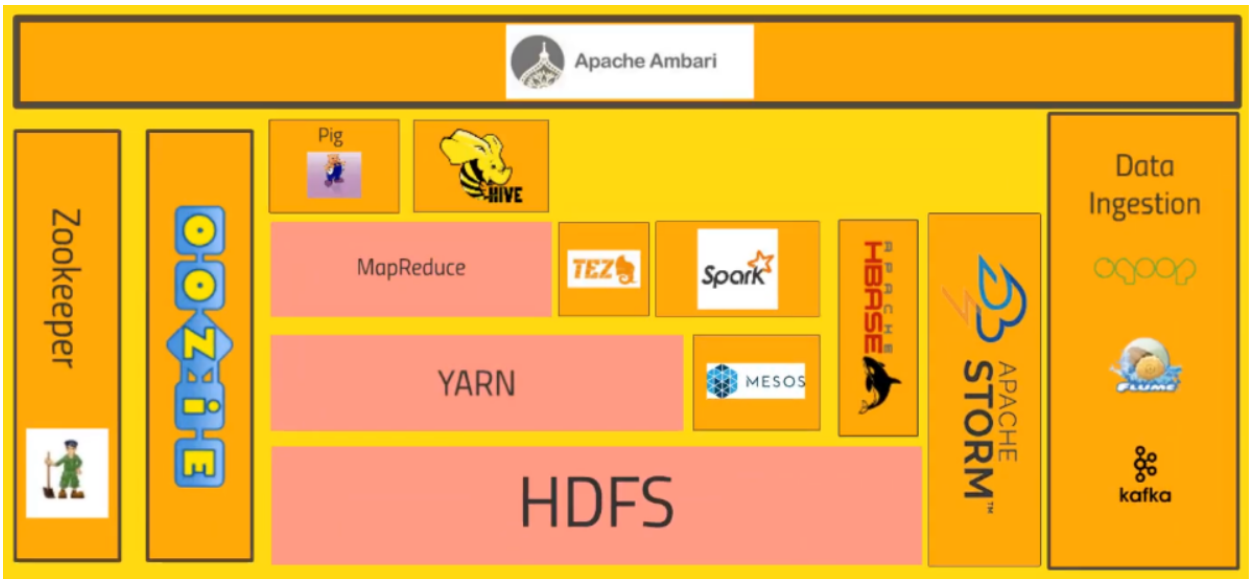
JobTracker est le service au sein de Hadoop qui est responsable de prendre les demandes des clients. Il les attribue aux TaskTrackers sur DataNodes où les données requises sont présentes localement. Si cela n'est pas possible, JobTracker essaie d'affecter les tâches aux TaskTrackers dans le même rack où les données sont localement présentes. Si, pour une raison quelconque, cela échoue également, JobTracker attribue la tâche à un TaskTracker où une réplique des données existe. Dans Hadoop, les blocs de données sont répliqués sur DataNodes pour garantir la redondance, de sorte que si un nœud du cluster échoue, le travail n'échoue pas également.

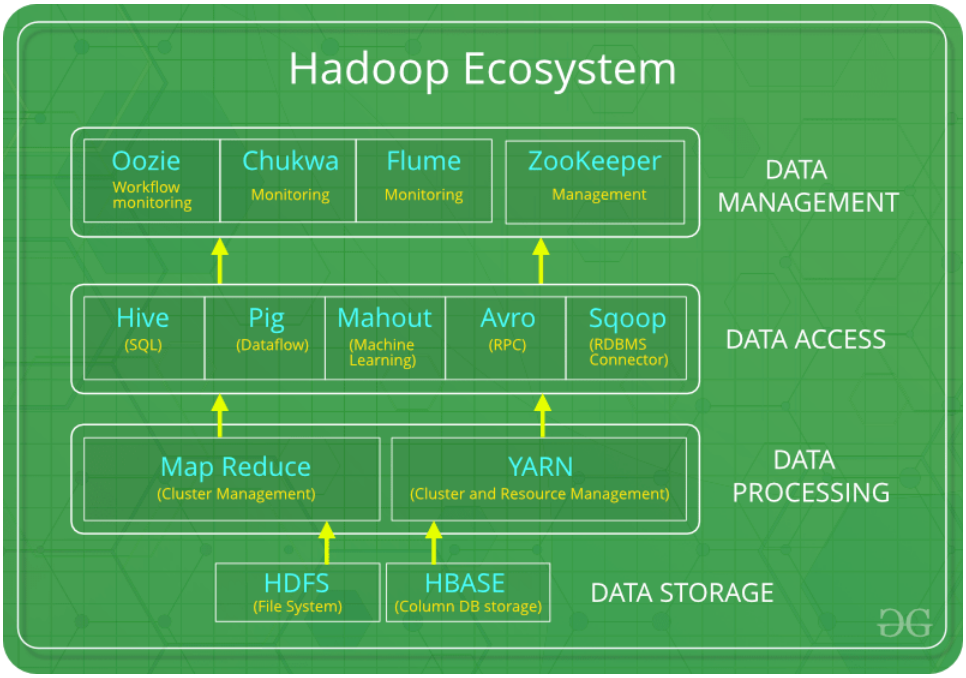
Processus JobTracker:

1. Les demandes de travail des applications client sont reçues par le JobTracker,
2. JobTracker consulte le NameNode afin de déterminer l'emplacement des données requises.
3. JobTracker localise les nœuds TaskTracker qui contiennent les données ou au moins sont proches des données.
4. Le travail est soumis au TaskTracker sélectionné.
5. Le TaskTracker exécute ses tâches tout en étant étroitement surveillé par JobTracker. Si le travail échoue, JobTracker soumet simplement le travail à un autre TaskTracker. Cependant, JobTracker lui-même est un point de défaillance unique, ce qui signifie que s'il échoue, tout le système tombe en panne.
6. JobTracker met à jour son état à la fin du travail.
7. Le demandeur client peut désormais interroger les informations de JobTracker.

TaskTraker

Architecture





HDFS: hadoop distributed file systeme
stock les donnees dans un cluster d'une facon distribute sur les different noeud
duplique les donnes

Yarn: manage the resources into the computer cluster
-when to run task
-what node are available for extra work
-witch node are available and not

MapReduce:
map: transforme data in parallel across all cluster
reduce: for aggregate this data togheter

Pig: if you don't want to use java or python
pig transform script to run on map reduce

Hive: like pig but are used like database

Apache Ambari: have a large view for my cluster

Spark: like map reduce it's reduce for processing data
-sql query
-machine learning
-streaming data in real time

HBASE: no-sql database
-use to expose my data of my cluster transformed by map reduce or …

APACHE STORM: processing streaming data in real time

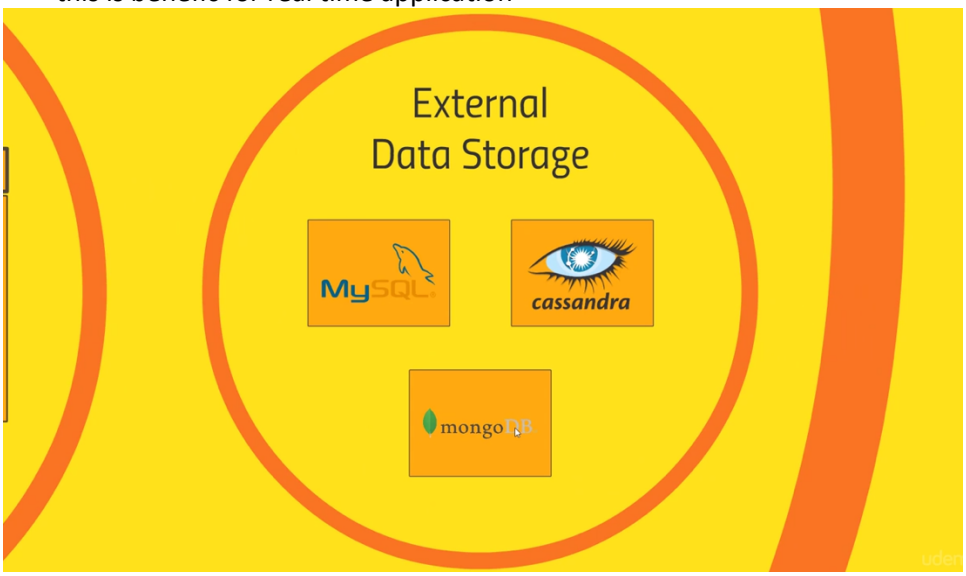
Oozie: manage task
exemple: task in my hadoop cluster that involve (impliquer) many stape
oozie will scageling all this staf into job
Exemple: if I want to loading data into hive and integrate with pig and quering with spark
and transform resource into hbase, oozie can manage all that

Zookeeper: to coordinating everything on your cluster
-track witch node it up, witch node it down
-track how is the current master node

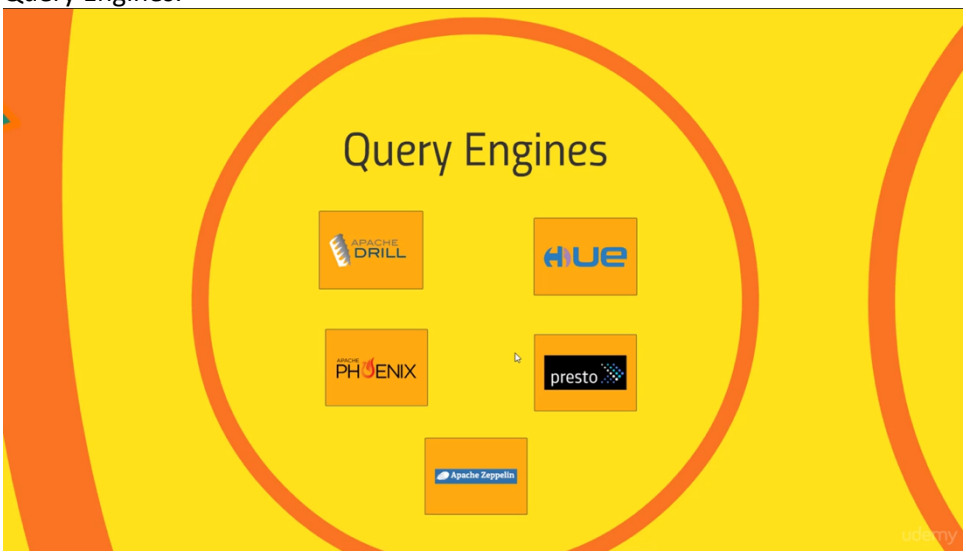
Data Ingestion:
- how we get data into your cluster and onto HDFS from external sources
-sqoop: tying (attacher) your hadoop database into a relational database
is basically a connector between Hadoop (HDFS) and database
-Flume: transfert logs from web app in real time by listening
-kafka: transfert data from any source in real time to my cluster

External Data Storage:

you can not only import data from Sqoop into your cluster ,
but you can also export it to MySql as well
Spark have the ability to write to any JDBC or ODBC database
this is benefic for real time application



Query Engines:



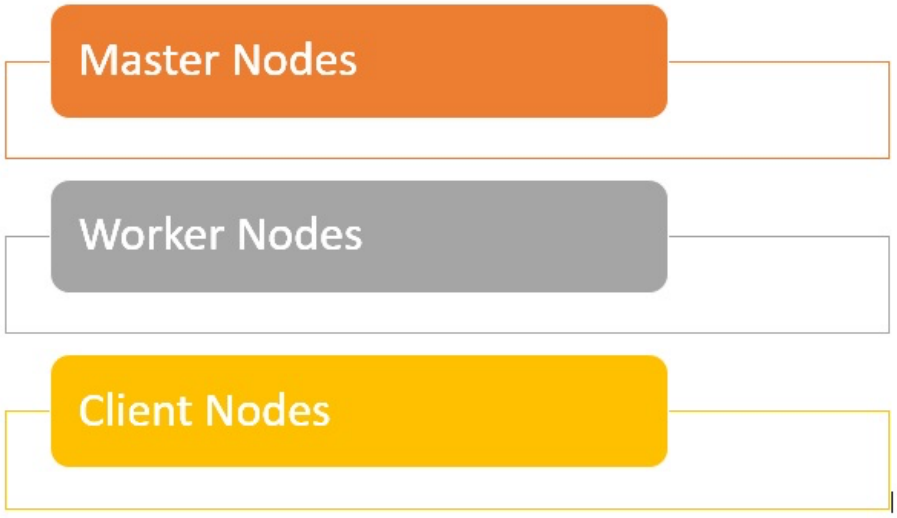
Apache DRILL:
write sql queries that will work across a wide range of NoSQl databases
ecrire une requete sql on meme temp sur plusieur BD

Hadoop Cluster Architecture

Saturday, 15 April 2023
12:34

Hadoop Cluster Architecture

Hadoop clusters are composed of a network of master and worker nodes that orchestrate and execute the various jobs across the Hadoop distributed file system. The master nodes typically utilize higher quality hardware and include a NameNode, Secondary NameNode, and JobTracker, with each running on a separate machine. The workers consist of virtual machines, running both DataNode and TaskTracker services on commodity hardware, and do the actual work of storing and processing the jobs as directed by the master nodes. The final part of the system are the Client Nodes, which are responsible for loading the data and fetching the results.



- Master nodes are responsible for storing data in HDFS and overseeing key operations, such as running parallel computations on the data using MapReduce.
- The worker nodes comprise most of the virtual machines in a Hadoop cluster, and perform the job of storing the data and running computations. Each worker node runs the DataNode and TaskTracker services, which are used to receive the instructions from the master nodes.

- Client nodes are in charge of loading the data into the cluster. Client nodes first submit MapReduce jobs describing how data needs to be processed and then fetch the results once the processing is finished.

Client in Hadoop refers to the Interface used to communicate with the Hadoop Filesystem. There are different type of Clients available with Hadoop to perform different tasks.
The basic filesystem client **hdfs dfs** is used to connect to a Hadoop Filesystem and perform basic file related tasks. It uses the ClientProtocol to communicate with a NameNode daemon, and connects directly to DataNodes to read/write block data. To perform administrative tasks on HDFS, there is **hdfs dfsadmin**. For HA related tasks, **hdfs haadmin**. There are similar clients available for performing **YARN** related tasks.
These Clients can be invoked using their respective CLI commands from a node where Hadoop is installed and has the necessary configurations and libraries required to connect to a Hadoop Filesystem. Such nodes are often referred as Hadoop Clients.

> For example, if I just write an hdfs command on the Terminal, is it still a "client" ?

Technically, **Yes**. If you are able to access the FS using the hdfs command, then the node has the configurations and libraries required to be a Hadoop Client.
PS: APIs are also available to create these Clients programmatically.

A Hadoop cluster ideally has 3 different kind of nodes: the masters, the edge nodes and the worker nodes.

The **masters** are the nodes that host the core more-unique Hadoop roles that usually orchestrate/coordinate/oversee processes and roles on the other nodes — think HDFS NameNodes (of which max there can only be 2), Hive Metastore Server (only one at the time of writing

this answer), YARN ResourceManager (just the one), HBase Masters, Impala StateStore and Catalog Server (one of each). All master roles need not necessarily have a fixed number of instances (you can have many Zookeeper Servers) but they all have associated roles within the same service that rely on them to function. A typical enterprise production cluster has 2–3 master nodes, scaling up as per size and services installed on the cluster.

Contrary to this, the **workers** are the actual nodes doing the real work of storing data or performing compute or other operations. Roles like HDFS DataNode, YARN NodeManager, HBase RegionServer, Impala Daemons etc — they need the master roles to coordinate the work and total instances of each of these roles usually scale more linearly with the size of the cluster. A typical cluster has about 80-90% nodes dedicated to hosting worker roles.

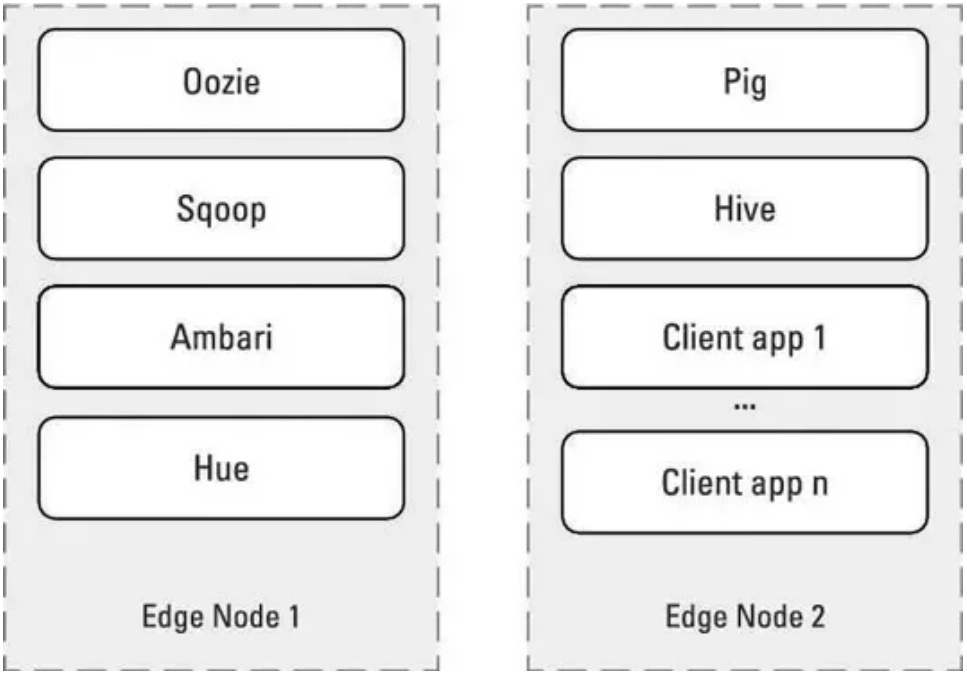
Put simply, **edge nodes** are the nodes that are neither masters, nor workers. They usually act as gateways/connection-portals for end-users to reach the worker nodes better. Roles like HiveServer2 servers, Impala LoadBalancer (Proxy server for Impala Daemons), Flume agents, config files and web interfaces like HttpFS, Oozie servers, Hue servers etc — they all fall under this category. Most of each of these roles can be installed on multiple nodes (assigning more nodes for each role helps prevent everybody from connecting to one instance and overwhelming that node).

The **purpose** of introducing edge nodes as against direct worker node access is: one — they act as network interface for the cluster and outside world (you don't want to leave the entire cluster open to the outside world when you can make do with a few nodes instead. This also helps keep the network architecture costs low); two — uniform data/work distribution (users directly connecting to the same set of few worker nodes won't harness the entire cluster's resources resulting in data skew/performance issues); and three — edge nodes serve as staging space for final data (stuff like data ingest using Sqoop, Oozie workflow setup etc).

That said, there is no formal rule that forces cluster admins to adhere to strict distinction between node types, and most Hadoop service roles can be assigned to any node which further blurs these

boundaries. But following certain role-co-location guidelines can significantly boost cluster performance and availability, and some might be vendor-mandated.

Edge nodes are the interface between the Hadoop cluster and the outside network. For this reason, they're sometimes referred to as *gateway* nodes. Most commonly, edge nodes are used to run client applications and cluster administration tools.
They're also often used as staging areas for data being transferred into the Hadoop cluster. As such, Oozie, Pig, Sqoop, and management tools such as Hue and Ambari run well there. The figure shows the processes you can run on Edge nodes.



Node = any physical host that belongs to your cluster
Services = HDFS, Hive, Impala, Zookeeper etc → they are installed on nodes

Roles = HDFS NameNode, HDFS DataNode, HiveServer2, Hive Metastore Server, Impala Catalog Server, Impala StateStore etc → they come together to make up the services
Now, NameNode is a role for HDFS service that must be installed on a node (or on two nodes if you need active and secondary nodes to ensure availability), just like all the other Hadoop service roles need to be installed on the cluster nodes. The node on which NameNode is installed then goes on to be known as the master node for the cluster. Since a cluster can have multiple master nodes (all of which may not necessarily have the role NameNode running on them), therefore sometimes nodes are addressed by the roles themselves to identify the right host(s) and so the master node on which the NameNode role is installed becomes known as the Name Node. NameNode itself is nothing but a directory listing/tracker of HDFS data that exists on the Data Nodes i.e. nodes on which HDFS DataNode role is installed. From a cluster architecture standpoint, data nodes are usually the worker nodes of the cluster.
I have provided a more detailed explanation of the Master/Edge Node/Worker Node distinction here - What is a simple explanation of edge nodes? (Hadoop)

Name Node: demande des traitement
 -sait ou est stocker

Data Node: fait des traitement

HDFS

Monday, 10 April 2023
17:12

https://stackoverflow.com/questions/45276427/unable-connect-to-docker-container-outside-docker-host

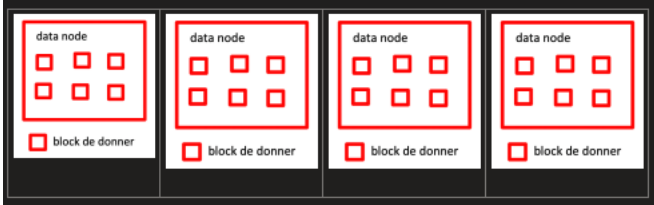
HDFS hadoop distributed file system
Allows data to be stored across entire cluster
Allows applications to access this data quickly

Permet de stocker de gros fichiers on block
Chaque block mesure 128 megabytes

Mon fichier serat separer sur plusieur block on fonction de sa taille

Mon fichier peut etre stocker sur different ordinateur
Se qui veut dire que plusieur ordinateur peuvent acceder au partie de mon fichier
et de les analyser on parallel

Mes block de donnees seront dupliquer sur plusieur noeud
Si un node tombe on panne aucun block de donner ne serat perdus



Name node: connait l'emplacement de chaque block de donner de mon fichier repartie
Sur les data node , Il track tous se qui a etait ajouter, modifier ou supprimer.

Data node: c'est la partie avec l'aquelle mon app vat communiquer
apres avoir demander l'emplacement du fichier du name node

-Reading file

Mon client demande au Hdfs Client de demander au name node l'emplecement de mon fichier
Le Hdfs Client irat recuperer les block de donner du fichier pour le client depuis les data node contenant les block de donner du fichier

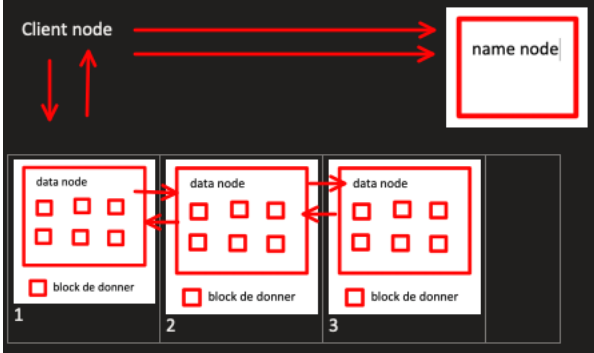
Client Node → HDFS Client Library →

What is hdfs client

Hdfs client is the Hadoop interface that allows users to interact with the Hadoop file system. There are various clients available in haddop. The basic one is **hdfs dfs** which connects the Hadoop distributed file system.
The other hdfs clients are hdfs dfsadmin, which is used to perform the administration work on the Hadoop file system.

Client in Hadoop refers to the Interface used to communicate with the Hadoop Filesystem. There are different type of Clients available with Hadoop to perform different tasks.
The basic filesystem client **hdfs dfs** is used to connect to a Hadoop Filesystem and perform basic file related tasks. It uses the ClientProtocol to communicate with a NameNode daemon, and connects directly to DataNodes to read/write block data. To perform administrative tasks on HDFS, there is **hdfs dfsadmin**. For HA related tasks, **hdfs haadmin**. There are similar clients available for performing **YARN** related tasks.

-Writing a File



Client demande a stocker un fichier
Name node cree une entrer pour lui et designe un data node
Client demande au data node de stocker le fichier
Le data node conserner vat demander au autre data node dupliquer les block de donner

3 valide --> 2 valide --> 1 valide -> client node valide --> informe --> name node (ok)

-->Namenode Resilience

On cas si le name node tombe on panne

- Back Up Metadata:
  Name node writes to local disk and NFS , sow if the name node die you can at least restore the Log from that NFS backup.
  Problem: mais donner ne sont pas syncroniser on meme temp donc certaine donner seront perdu

- Secondary Namenode: have a copy for my primary name node
- HDFS Federation: Each namenode manages a specific namespace volume
  si un name node tombe on panne je ne perdrais pas les autre name node
  juste une partie des donnes serait perdu

--> HDFS High Availability:
- The HDFS NameNode High Availability feature enables you to run redundant NameNodes in the same cluster in an Active/Passive configuration with a hot standby. This eliminates the NameNode as a potential single point of failure (SPOF) in an HDFS cluster.
- zookeeper track active namenode

https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.6.0/bk_hadoop-high-availability/content/ch_HA-NameNode.html

HDFS est le système de fichiers distribué utilisé par le framework Hadoop.

Caractéristiques

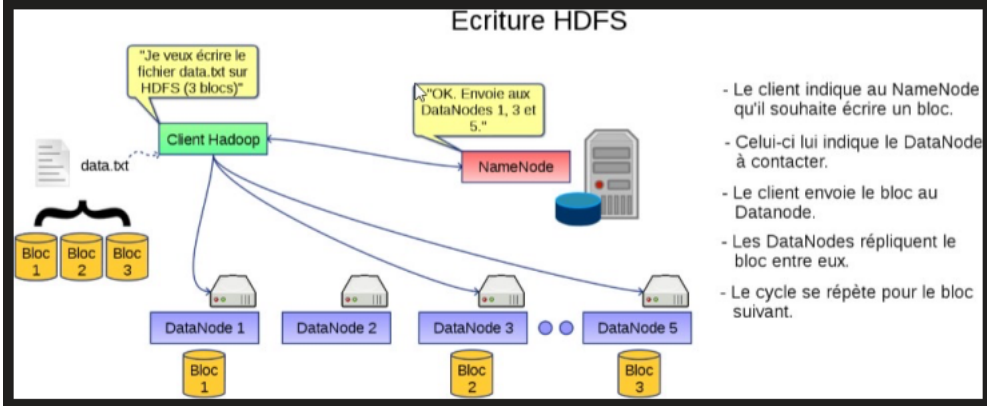
- En Java.
- Stocke des données structurées ou non sur un ensemble de serveurs distribués.
- Redondant, résilient.
- Découpage et distribution en blocks des données :

• **Blocksize :** taille unitaire de stockage (généralement 64 Mo ou 128 Mo). •**Replication factor :** nombre de copies d'une données devant être réparties sur les différents noeuds.
Par défaut la valeur de **replication factor** est 3

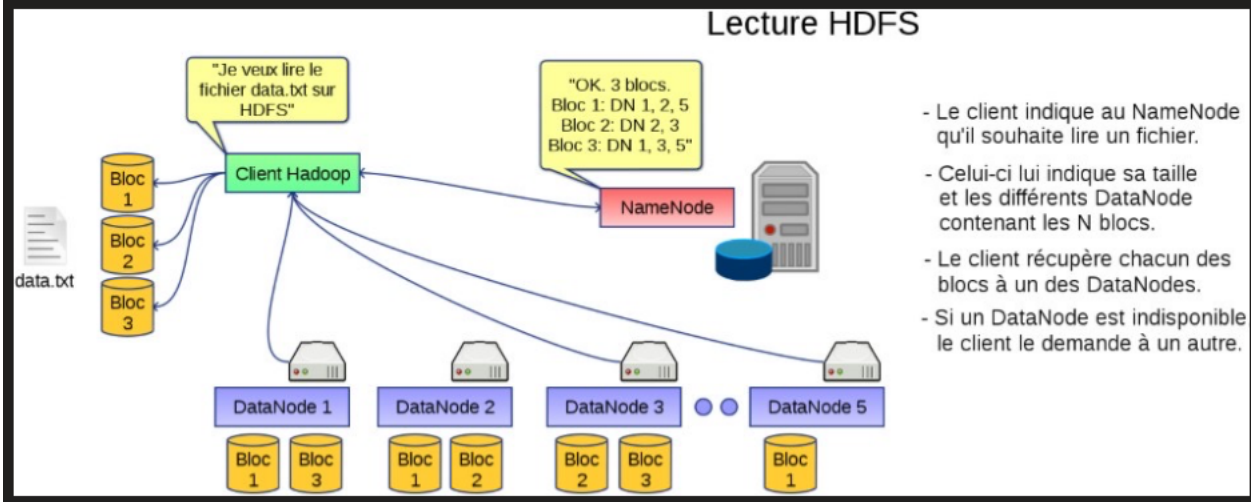
DataNode : démon sur chaque nœud du cluster ➢ **NameNode** : démon s'exécutant sur une machine séparée

- Contient des meìtadonneìes
- Permet de retrouver les n□uds qui exeìcutent les blocs d'un fichier

Ecriture d'un fichier



Lecture d'un fichier



HDFS Command

Monday, 10 April 2023
21:23

-Afficher le contenu du répertoire racine

hadoop fs -ls /

-Creer un repertoire dans la racine

hadoop fs -mkdir /ml-100k

wget http://media.sundog-soft.com/hadoop/ml-100k/u.data

copier un fichier dans un repertoir hdfs

hadoop fs -put u.data /ml-100k

Recuperer un fichier depuis un repertoire hdfs

hadoop fs -get /user/hive/warehouse/movie_names/u.item /home/hdoop

Supprimer un fichier

hadoop fs -rm /ml-100k/u.data

Lire le contenue d'un fichier

hadoop fs -cat /user/hive/warehouse/movie_names/u.item

Supprimer un repertoire

hadoop fs -rmdir /ml-100k

Copier un fichier
#hadoop fs -cp /user/cloudera/d1/* /user/cloudera/d2

Deplacer un fichier
#hadoop fs -mv /user/cloudera/d1/* /user/cloudera/d2

Creer un repertoire
#hadoop fs -mkdir /user/rep1

Creer toute une repertoire
#hadoop fs -mkdir -p /rep1/rep2/rep3

Ajouter du contenu a un fichier
Hadoop fs -appendToFile /local/text.txt /destinationHDFS/data.txt
-Depuis le clavier
Hadoop fs -appendToFile   - /destinationHDFS/data.txt

HDFS Java

Tuesday, 16 May 2023
11:24

```
Private static final String NAME_NODE="hdfs://localhost:9000";
FileSystem hdfs = FileSystem.get( URI.create("hdfs://0.0.0.0:9000"),new Configuration() );

Path workingDir = hdfs.getWorkingDirectory();

Path FolderPath = new Path (hdfs.getWorkingDirectory()+"/"+"TestDirectory");


CreateFolder(hdfs,FolderPath);
DeleteFolder(hdfs,FolderPath);
CopieLocalFileToHDFS(hdfs,FolderPath);
CreatFile(hdfs,FolderPath);
WriteIntoFile(hdfs,FolderPath);
ReadFile(hdfs,FolderPath);


++++++++++++++++++++++++++++++++++++++++++++++

Static void DeleteFolder(FileSystemhdfs,PathFolderPath) throwsIOException{
 if(hdfs.exists(FolderPath)){
    hdfs.delete(FolderPath,true);
  }
}

++++++++++++++++++++++++++++++++++++++++++++++

Static void CreateFolder(FileSystemhdfs,PathFolderPath) throwsIOException{

   hdfs.mkdirs(FolderPath);

}

++++++++++++++++++++++++++++++++++++++++++++++

Static void CopieLocalFileToHDFS(FileSystemhdfs,PathFolderPath) throwsIOException{

  Path
localFilePath=newPath("/Users/aymanehinane/Desktop/Home/BigData/PrepaExam/hdfs/src/main/resour
ces/data.txt");
  hdfs.copyFromLocalFile(localFilePath,FolderPath);

}

++++++++++++++++++++++++++++++++++++++++++++++

Static void CreatFile(FileSystemhdfs,PathFolderPath) throwsIOException{

  Path FilePath=newPath(FolderPath+"/"+"data2.txt");
  hdfs.createNewFile(FilePath);

}

++++++++++++++++++++++++++++++++++++++++++++++

Static void WriteIntoFile(FileSystemhdfs,PathFolderPath) throwsIOException{
```

```
  StringBuildersb=new StringBuilder();
  sb.append("helloworld");
//for(inti=1;i<=5;i++)
//{
//sb.append("Data"+i+"\n");
//}
  byte[]contenuOctet=sb.toString().getBytes();
  PathFilePath=newPath(FolderPath+"/"+"data2.txt");
  FSDataOutputStreamflux=hdfs.create(FilePath);
  flux.write(contenuOctet);
  flux.close();
}

++++++++++++++++++++++++++++++++++++++++++++++

Static void ReadFile(FileSystemhdfs,PathFolderPath)throwsIOException{

  PathFilePath=newPath(FolderPath+"/"+"data3.txt");
  BufferedReaderbfr=newBufferedReader(newInputStreamReader(hdfs.open(FilePath)));
  Stringstr=null;

  while((str=bfr.readLine())!=null){
   System.out.println(str);
  }
}
```

Hive

Saturday, 8 April 2023
16:50

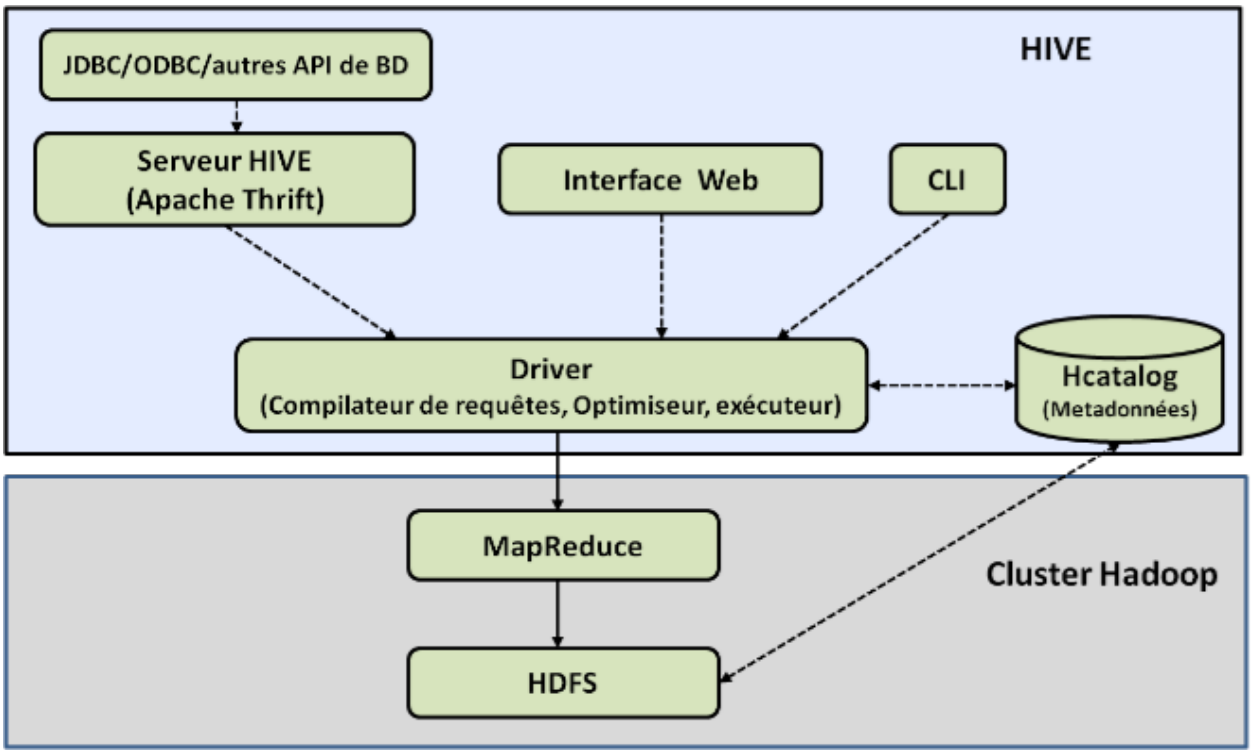
HIVE

C'est quoi Hive ?

• Une infrastructure pour Entrepôt de Données
• On peut tout simplement le considérer comme un Data Warehouse ! **Comment ça marche ?**
● Hive a pour fondation Hadoop
● Hive stocke ses données dans HDFS
●Hive compile des requêtes SQL en jobs MapReduce et les exécute sur le cluster Hadoop

Dans Hive on utilise des requetes HiveQL

Le **Metastore** est un catalogue qui contient les metadonnées des tables stockées dans Hive



--to start hive
Hive

quit;

hive> set hive.server2.thrift.port;
hive.server2.thrift.port=10000

nohup $HIVE_HOME/bin/hive --service hiveserver2 &

./hive --service hiveserver2 --hiveconf hive.server2.thrift.port=10000 --hiveconf hive.root.logger=INFO,console --hiveconf hive.server2.enable.doAs=false

netstat -anp | grep 10000

/home/hdoop/apache-hive-3.1.2-bin/bin/beeline -u jdbc:hive2://0.0.0.0:10000

U.data ---> userID:int , movieID:int , rating:int , ratingTime:int
U.item ---> movieID:int , movieTitle:chararray , releaseDate:chararray , videoRelease:chararray , imdbLink:chararray

hadoop fs -ls /

hadoop fs -mkdir -p /user/hive/warehouse
hadoop fs -chmod g+w /user/hive/warehouse

ratings

user_id
movie_id
rating
rating_time

/opt/shared-folder/ml-100k
u.data

hdfs://localhost:9000
/user/hive/warehouse

hadoop fs -put /opt/shared-folder/ml-100k/u.data /user/hive/warehouse

hadoop fs -ls /user/hive/warehouse

```
CREATE EXTERNAL TABLE IF NOT EXISTS movie_names
(movie_id INT,name STRING,column3 STRING,column4 STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS TEXTFILE
LOCATION '/user/hive/warehouse/movie_names';

Exit;
```

\\s+

```
CREATE EXTERNAL TABLE IF NOT EXISTS ratings
(user_id INT,movie_id INT,rating INT,rating_time INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
LOCATION '/user/hive/warehouse/ratings';
```

```
SHOW TABLES;
drop table rating;
```

```
root@c7556f98e9b1:/opt/shared-folder/ml-100k# cd $HIVE_HOME/bin
root@c7556f98e9b1:/home/hdoop/apache-hive-3.1.2-bin/bin# schematool -initSchema -dbType derby
```

```
select movie_id,rating,count(movie_id) as ratingCount
from ratings group by movie_id,rating
 order by ratingCount desc limit 1;
```

Hive Java

Wednesday, 26 April 2023
10:02

```
public static void main(String[] args) throws SQLException {
 / / Register driver and create driver instance

Class.forName("org.apache.hadoop.hive.jdbc.HiveDriver");
/ / get connection
        Connection con =
DriverManager.getConnection("jdbc:hive://localhost:10000/d
efault", "", "");
        Statement stmt = con.createStatement();
         stmt.executeQuery("CREATE DATABASE userdb");
```

```
      Resultset res = stmt.executeQuery("SELECT *
FROMemployee WHEREsalary>30000;");
      System.out.println("Result:");
      System.out.println(" ID \ t Name\ t Salary \ t Designation \ t
Dept " ) ;
     while (res.next()) {
         System.out.println(res.getInt(1) + " " + res.getString(2) + "
" + res.getDouble(3) + " " + res.getString(4) +" "
+res.getString(5));
    }

   con.close();


}
```

Map Reducer

Monday, 10 April 2023
23:15

-Distributed the processing of data on your cluster
-Divides your data up into partitions that are mapped (transformed) and Reduced (aggregated) by mapper and reduce functions you define

Example:
How many movies did each user rate in the movieLens data set ?

The Mapper converts raw source data into key/value pairs:

Input Data ---> Mapper ---> K1:V K2:V K3:V K1:V K1:V

The key is the value that I will aggregate in
The key : user Id (the key represent the entity that are concerned)
The value: movie Id (the value represent the entity that i want process)

Mapper Extract and Organize what We Care about

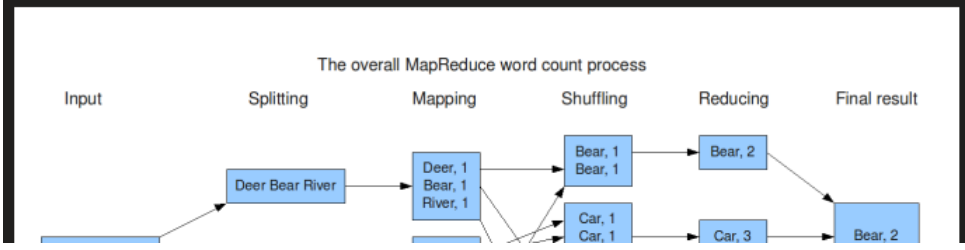
User Id	Movie Id	Ranting
196	242	3
186	302	3
196	377	1
244	51	2
166	346	1
186	474	4
186	265	2

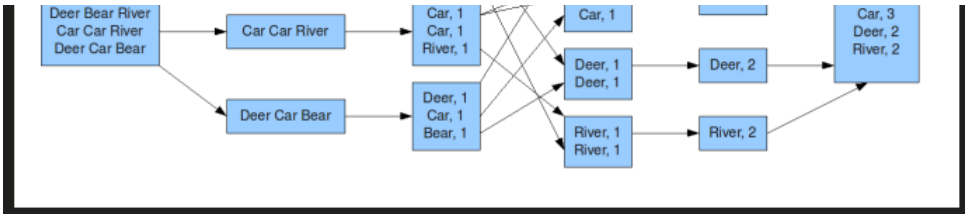
Data ----> Mapper ----> 196:242 186:302 196:377 244:51 166:346 186:474 186:265 ---> List<key,value>

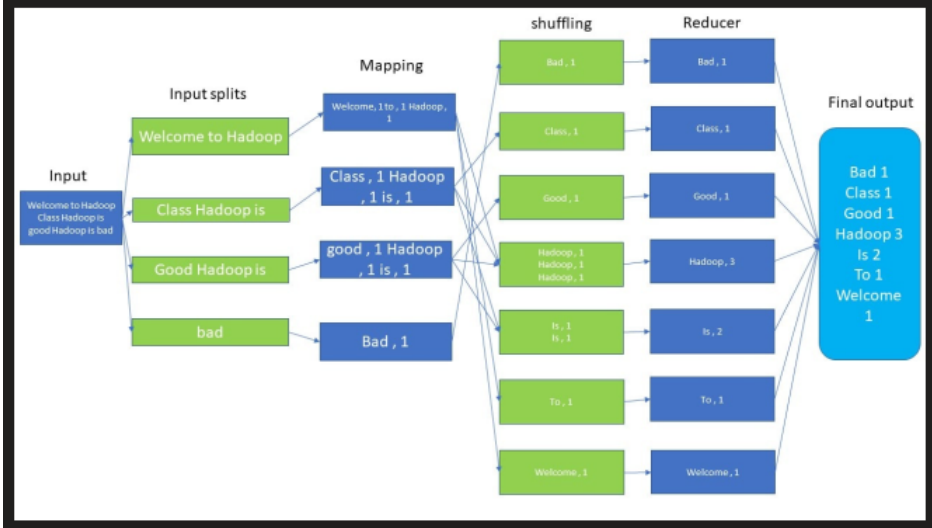
aggregate all the value for each unique key
List<key,value> ---> MapReduce Sort and Group the Mapped Data (" shuffle and Sort ") --> 166:346 186:302,274,265 196:242,377 244:51

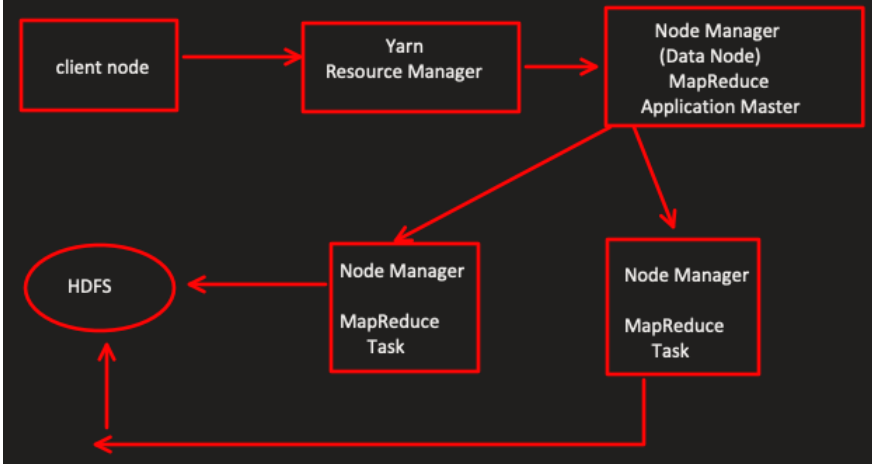
The Reducer Processes Each Key's Values

---> shuffle and sort ---> List< key, List<value> > ----> Reduce (length(movies) --> length(List<value>)) ---> 166:1 186:3 196:2 244:1









-MapReduce Process distribution

We can actually reduce different keys in parallel across different computers as well.
So in a nutshell that's basically how map reduce can distribute a job across an entire cluster.

The mapping stage can be split up across different computers where each computer receives a different chunk of the input data.
And after the shuffle and sort operation you can have different computers responsible for different

client node

Yarn Resource Manager :
it's what manages what gets run on which machines.
OK so it's keeping track of what machines in my cluster are available which ones have capacity and so on and so forth.

Node Manager :
Node manager that just keeps track of what this node is doing you know is it up is actually operating stuff like that.
So the application master is now responsible for keeping an eye on all the different individual map and reduce tasks.
And it works with the resource manager to actually distribute that work across your cluster.

Client Node ---> communicate with (I want to kick off a map reduce job) ---> Yarn Resource Manager
Client Node ---> copy the data that I want to process into the hdfs
Resource Manager ---> send the job to Node Manager --> Node Manager

client node: there's a resource manager that's keeping track of all the different computers and what has availability.

application master: responsible for keeping an eye on all of your tasks

node managers: are keeping an eye on the individual PCs as a whole.
And they all talk to our data on our HDFS cluster.

Map : extraction/calcul d'une information sur chaque n-uplet.
Reduce : regroupement de ces informations

Job Tracker –

- JobTracker process runs on a separate node and not usually on a DataNode.
- JobTracker is an essential Daemon for MapReduce execution in MRv1. It is replaced by ResourceManager/ApplicationMaster in MRv2.
- JobTracker receives the requests for MapReduce execution from the client.
- JobTracker talks to the NameNode to determine the location of the data.
- JobTracker finds the best TaskTracker nodes to execute tasks based on the data locality (proximity of the data) and the available slots to execute a task on a given node.

- JobTracker monitors the individual TaskTrackers and the submits back the overall status of the job back to the client.
- JobTracker process is critical to the Hadoop cluster in terms of MapReduce execution.
- When the JobTracker is down, HDFS will still be functional but the MapReduce execution can not be started and the existing MapReduce jobs will be halted.

Task Tracker-

- TaskTracker runs on DataNode. Mostly on all DataNodes.
- TaskTracker is replaced by Node Manager in MRv2.
- TaskTracker will be in constant communication with the JobTracker signalling the progress of the task in execution.
- Mapper and Reducer tasks are executed on DataNodes administered by TaskTrackers.
- TaskTrackers will be assigned Mapper and Reducer tasks to execute by JobTracker.
- TaskTracker failure is not considered fatal. When a TaskTracker becomes unresponsive, JobTracker will assign the task executed by the TaskTracker to another node.

YARN

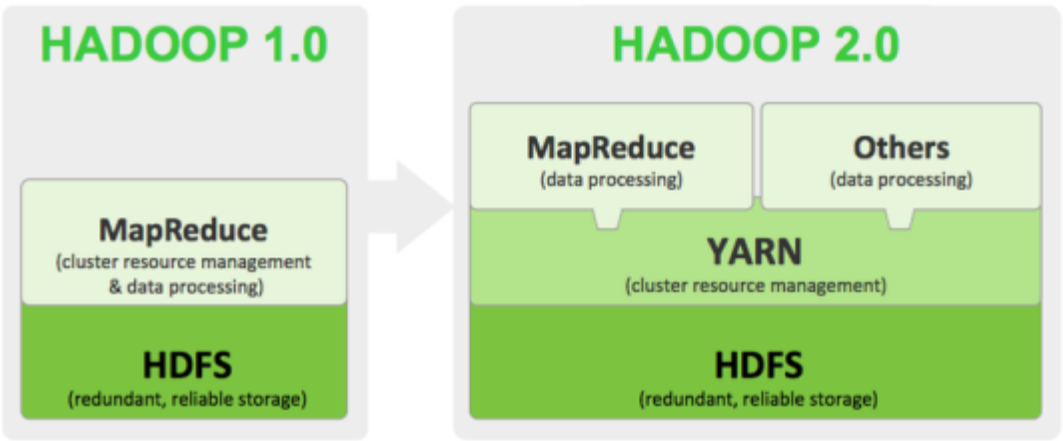
YARN (Yet Another Resource Negociator) est un mécanisme permettant de gérer des travaux (*jobs*) sur un cluster de machines.
YARN permet aux utilisateurs de lancer des *jobs* Map-Reduce sur des données présentes dans HDFS, et de suivre (*monitor*) leur avancement, récupérer les messages (*logs*) affichés par les programmes.
Éventuellement YARN peut déplacer un processus d'une machine à l'autre en cas de défaillance ou d'avancement jugé trop lent.

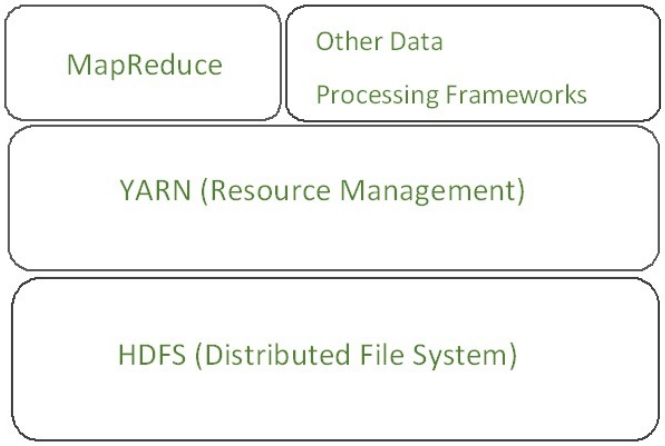
YARN

Tuesday, 16 May 2023
12:06

YARN est constitué de plusieurs composants principaux. Le **gestionnaire de ressources global (ResourceManager)** a pour rôle d'accepter les tâches soumises par les utilisateurs, de programmer les tâches et de leur allouer des ressources.
Sur chaque noeud, on retrouve un **NodeManager dont le rôle de surveiller et de rapporter** au ResourceManager. On retrouve par ailleurs un ApplicationMaster, créé pour chaque application, chargé de négocier les ressources et de travailler conjointement avec le NodeManager pour exécuter et surveiller les tâches.
Enfin, les **containers de ressources** sont contrôlés par les NodeManagers et assigne les ressources allouées aux applications individuelles. Généralement, les containers YARN sont organisés en noeuds et programmés pour exécuter des tâches uniquement si des ressources sont disponibles pour ce faire.

Structure de Hadoop 2





Hadoop 2.0

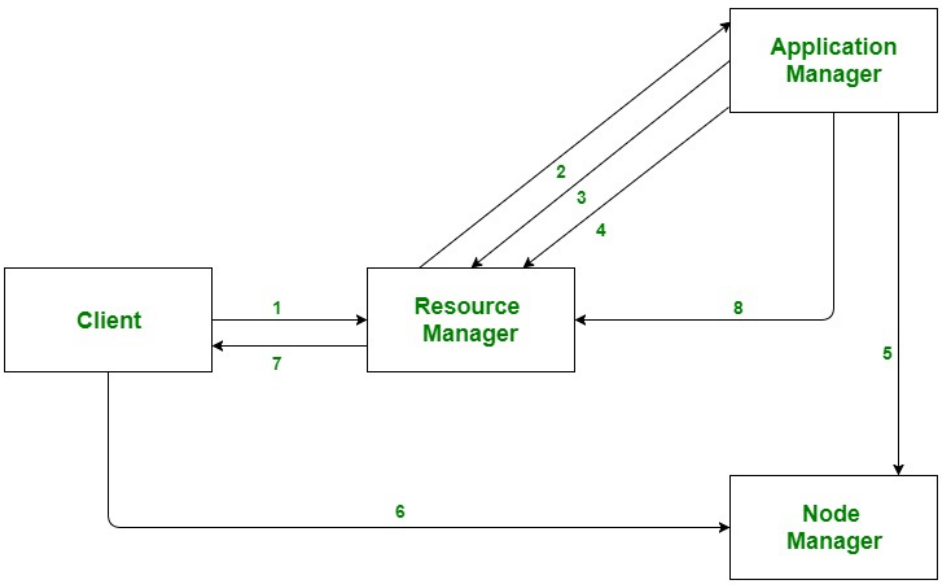
The main components of YARN architecture include:

- **Client:** It submits map-reduce jobs.
- **Resource Manager (jobTracker):** It is the master daemon of YARN and is responsible for resource assignment and management among all the applications. Whenever it receives a processing request, it forwards it to the corresponding node manager and allocates resources for the completion of the request accordingly. It has two major components:
- **Scheduler:** It performs scheduling based on the allocated application and available resources. It is a pure scheduler, means it does not perform other tasks such as monitoring or tracking and does not guarantee a restart if a task fails. The YARN scheduler supports plugins such as Capacity Scheduler and Fair Scheduler to partition the cluster resources.
- **Application manager:** It is responsible for accepting the application and negotiating the first container from the resource manager. It also restarts the Application Master container if a task fails.
- **Node Manager (task tracker):** It take care of individual node on Hadoop cluster and manages application and workflow and that particular node. Its primary job is to keep-up with the Resource

Manager. It registers with the Resource Manager and sends heartbeats with the health status of the node. It monitors resource usage, performs log management and also kills a container based on directions from the resource manager. It is also responsible for creating the container process and start it on the request of Application master.

- **Application Master:** An application is a single job submitted to a framework. The application master is responsible for negotiating resources with the resource manager, tracking the status and monitoring progress of a single application. The application master requests the container from the node manager by sending a Container Launch Context(CLC) which includes everything an application needs to run. Once the application is started, it sends the health report to the resource manager from time-to-time.
- **Container:** It is a collection of physical resources such as RAM, CPU cores and disk on a single node. The containers are invoked by Container Launch Context(CLC) which is a record that contains information such as environment variables, security tokens, dependencies etc.

Application workflow in Hadoop YARN:



- Client submits an application
- The Resource Manager allocates a container to start the Application Manager
- The Application Manager registers itself with the Resource Manager
- The Application Manager negotiates containers from the Resource Manager
- The Application Manager notifies the Node Manager to launch containers
- Application code is executed in the container
- Client contacts Resource Manager/Application Manager to monitor application's status
- Once the processing is complete, the Application Manager un-registers with the Resource Manager

Advantages :

- **Flexibility:** YARN offers flexibility to run various types of distributed processing systems such as Apache Spark, Apache Flink, Apache Storm, and others. It allows multiple processing engines to run simultaneously on a single Hadoop cluster.
- **Resource Management:** YARN provides an efficient way of managing resources in the Hadoop cluster. It allows administrators to allocate and monitor the resources required by each application in a cluster, such as CPU, memory, and disk space.
- **Scalability:** YARN is designed to be highly scalable and can handle thousands of nodes in a cluster. It can scale up or down based on the requirements of the applications running on the cluster.
- **Improved Performance:** YARN offers better performance by providing a centralized resource management system. It ensures that the resources are optimally utilized, and applications are efficiently scheduled on the available resources.
- **Security:** YARN provides robust security features such as Kerberos authentication, Secure Shell (SSH) access, and secure data transmission. It ensures that the data stored and processed on the Hadoop cluster is secure.

Disadvantages :

- **Complexity:** YARN adds complexity to the Hadoop ecosystem. It requires additional configurations and settings, which can be difficult for users who are not familiar with YARN.
- **Overhead:** YARN introduces additional overhead, which can slow down the performance of the Hadoop cluster. This overhead is required for managing resources and scheduling applications.
- **Latency:** YARN introduces additional latency in the Hadoop ecosystem. This latency can be caused by resource allocation, application scheduling, and communication between components.
- **Single Point of Failure:** YARN can be a single point of failure in the Hadoop cluster. If YARN fails, it can cause the entire cluster to go down. To avoid this, administrators need to set up a backup YARN instance for high availability.
- **Limited Support:** YARN has limited support for non-Java programming languages. Although it supports multiple processing engines, some engines have limited language support, which can limit the usability of YARN in certain environments.

Exercice 1 Map Reduce

Sunday, 16 April 2023
12:05
https://grouplens.org/datasets/movielens/

wget http://media.sundog-soft.com/hadoop/RatingsBreakdown.py
wget http://media.sundog-soft.com/hadoop/ml-100k/u.data

For linux

pip install pathlib
pip install mrjob

Mrjob is to simulate map reduce

pip install PyYAML

For mac
Python - m pip install pathlib
```
python –m pip install mrjob
```
Python -m pip install PyYAML

```
python RatingsBreakdown.py u.data
```

```
––search for pa
```

find / -name 'hadoop-streaming*.jar'

hdfs dfsadmin -safemode enter
Safe mode is ON

hdfs dfsadmin -safemode get
Safe mode is ON

hdfs dfsadmin -safemode leave
Safe mode is OFF

hadoop fs -mkdir -p /user/root/tmp/mrjob/RatingsBreakdown.root.20230416.141050.827685/files/wd

python RatingsBreakdown.py -r hadoop --hadoop-streaming-jar /home/hdoop/hadoop-3.3.5/share/hadoop/tools/lib/hadoop-streaming-3.3.5.jar u.data

And again we need to tell it exactly where the Hadoop streaming Jarre is.
This is just the bit of code that integrates map reduce with Python or anything really.

hadoop fs -mkdir /python
hadoop fs -put u.data /python

python RatingsBreakdown.py -r hadoop --hadoop-streaming-jar /home/hdoop/hadoop-3.3.5/share/hadoop/tools/lib/hadoop-streaming-3.3.5.jar hdfs://localhost:9000/python/u.data

How many of each movie ranting exist

Movie ID	Ranting
1005	4
1006	2
1007	3
1008	4
1009	3

Map --> (4,1) (2,1) (3,1) (4,1) (3,1) ---> shuffle & sort --> (4,(1,1)) (2,1) (3,(1,1)) ---> Reduce --> (4,2) (2,1) (3,2)

```
  GNU nano 6.2
from mrjob.job import MRJob
from mrjob.step import MRStep

class RatingsBreakdown(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_ratings,
                   reducer=self.reducer_count_ratings)
        ]

    def mapper_get_ratings(self, _, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield rating, 1

    def reducer_count_ratings(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    RatingsBreakdown.run()
```

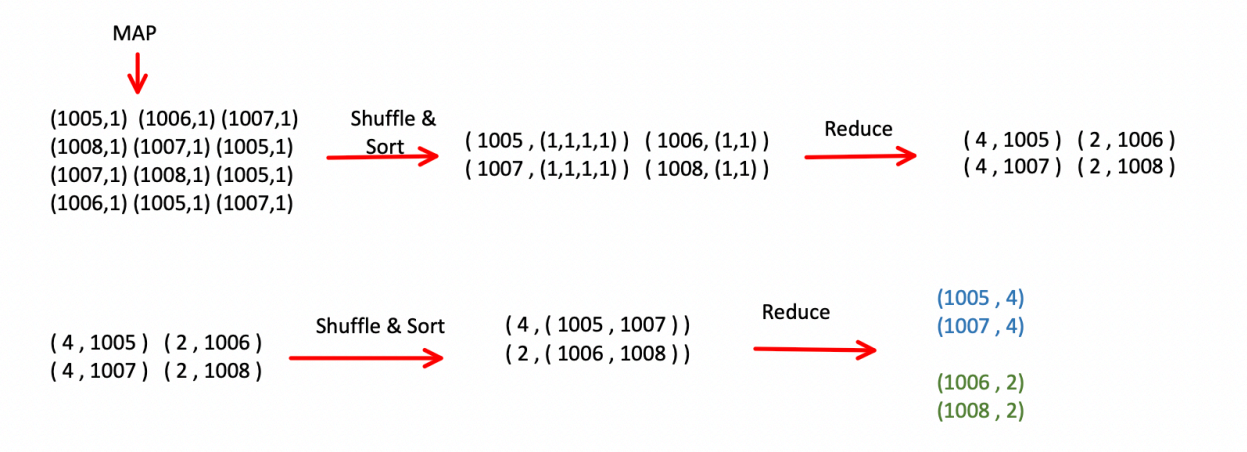
```
MovieCount.py  MovieCountSort.py  RatingsBreakdown.py  ml-100k  u.data
root@c7556f98e9b1:/home/hdoop/Data# nano RatingsBreakdown.py
root@c7556f98e9b1:/home/hdoop/Data# python RatingsBreakdown.py u.data
No configs found; falling back on auto-configuration
No configs specified for inline runner
Creating temp directory /tmp/RatingsBreakdown.root.20230416.204718.372805
Running step 1 of 1...
job output is in /tmp/RatingsBreakdown.root.20230416.204718.372805/output
Streaming final output from /tmp/RatingsBreakdown.root.20230416.204718.372805/output...
"3"     27145
"1"     6111
"2"     11370
"4"     34174
"5"     21203
Removing temp directory /tmp/RatingsBreakdown.root.20230416.204718.372805...
root@c7556f98e9b1:/home/hdoop/Data#
```

++

```
from mrjob.job import MRJob
from mrjob.step import MRStep

//MRJob : This is basically a way to very quickly write mapreduce jobs in Python.
//              It abstracts away a lot of the complexity of dealing with the streaming interface to
mapreduce for

class RatingsBreakdown(MRJob):
   def steps(self):
      return [
         MRStep(mapper=self.mapper_get_ratings,
               reducer=self.reducer_count_ratings)
      ]

//Map
   def mapper_get_ratings(self, _, line):
      (userID, movieID, rating, timestamp) = line.split('\t')
      yield rating, 1

//Reduce
   def reducer_count_ratings(self, key, values):
      yield key, sum(values)

if __name__ == '__main__':
   RatingsBreakdown.run()
```

Movie ID	Ranting

1005	4
1006	2
1007	3
1008	4
1007	3
1005	2
1007	3
1008	4
1005	4
1006	2
1005	5
1007	2



Exercice 2 Map Reduce

Wednesday, 19 April 2023
11:32

Exemple d'explication:

MapReduce est écrit on java , mais il peut etre executer dans different langauge C++,Python…

Dans cette exemple nous allons utiliser Python avec MP job package

Step 1 :

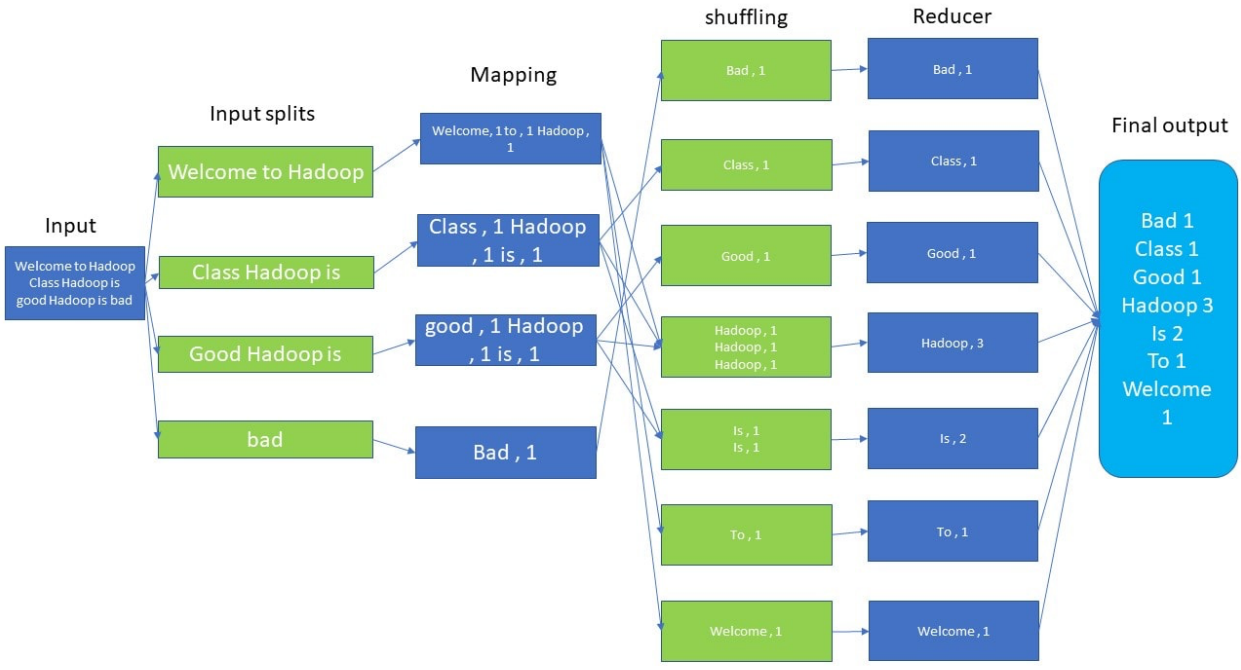
Mapper vat recevoir des lignes text d'un fichier et vat les convertir on (cle,valeur)

Step2 :

Shuffling and Sort est une etape ou MapReduce vat regrouper toute les value qui ont la meme cle (cle , list(value))

Step 3 :

Reduce permet de calculer la list des value pour chaque cle



Exemple d'application 1:

Nous avons un fichier u.data qui contient une list de review pour chaque film

Le premier champ : MovieID
Le dexieme champ : ranting

1005 4
1006 2
1007 3
1008 4
1009 3

Nous voulons compter combien de fois un ranting se repete

Algorithme

Map --> (4,1) (2,1) (3,1) (4,1) (3,1) ---> shuffle & sort

--> (4,(1,1)) (2,1) (3,(1,1)) ---> Reduce --> (4,2) (2,1) (3,2)

Code python

```
 GNU nano 6.2
from mrjob.job import MRJob
from mrjob.step import MRStep

class RatingsBreakdown(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_ratings,
                   reducer=self.reducer_count_ratings)
        ]

    def mapper_get_ratings(self, _, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield rating, 1

    def reducer_count_ratings(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    RatingsBreakdown.run()
```

MRStep permet de declarer le mapper et le reducer

MRJob permet d'ecrir un map reduce Job on python est de l'executer sur different environnemt

```
MovieCount.py  MovieCountSort.py  RatingsBreakdown.py  ml-100k  u.data
root@c7556f98e9b1:/home/hdoop/Data# nano RatingsBreakdown.py
root@c7556f98e9b1:/home/hdoop/Data# python RatingsBreakdown.py u.data
No configs found; falling back on auto-configuration
No configs specified for inline runner
Creating temp directory /tmp/RatingsBreakdown.root.20230416.204718.372805
Running step 1 of 1...
job output is in /tmp/RatingsBreakdown.root.20230416.204718.372805/output
Streaming final output from /tmp/RatingsBreakdown.root.20230416.204718.372805/output...
"3"     27145
"1"     6111
"2"     11370
"4"     34174
"5"     21203
Removing temp directory /tmp/RatingsBreakdown.root.20230416.204718.372805...
root@c7556f98e9b1:/home/hdoop/Data#
```

Exemple d’application 2 :

On veut compter le nombre de fois qu’un film a était noter

```
 GNU nano 6.2
from mrjob.job import MRJob
from mrjob.step import MRStep

class RatingsBreakdown(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_ratings,
                   reducer=self.reducer_count_ratings)
        ]

    def mapper_get_ratings(self, _, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield movieID,1

    def reducer_count_ratings(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    RatingsBreakdown.run()
```

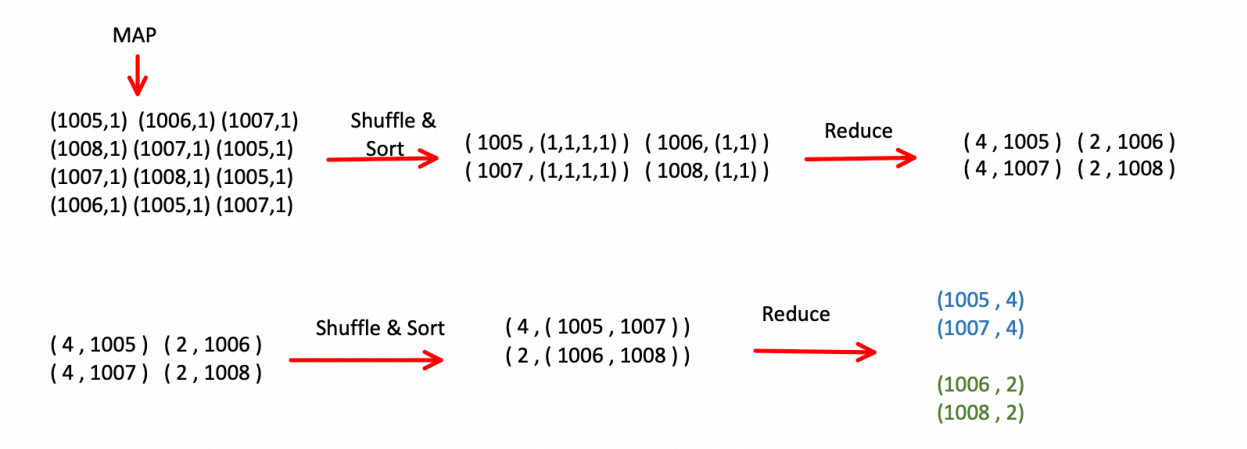
python MovieCount.py u.data

```
"98"    390
"980"   22
"981"   8
"982"   20
"983"   15
"984"   44
"985"   22
"986"   23
"987"   4
"988"   86
"989"   32
"99"    172
"990"   33
"991"   25
"992"   4
"993"   66
"994"   7
"995"   31
"996"   14
"997"   16
"998"   16
"999"   10
```

Exemple d'application 2 :

Pour cette exemple nous voulons regrouper les filme qui ont le meme nombre de review

Algorithme :



```
 GNU nano 6.2
rom mrjob.job import MRJob
from mrjob.step import MRStep

class RatingsBreakdown(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_ratings,
                   reducer=self.reducer_count_ratings),
            MRStep(reducer=self.reducer_sorted_output)
        ]

    def mapper_get_ratings(self, _, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield movieID,1

    def reducer_count_ratings(self, key, values):
        yield str(sum(values)),key

    def reducer_sorted_output(self,count,movies):
        for movie in movies:
           yield movie , count


if __name__ == '__main__':
    RatingsBreakdown.run()
```

Exécution :

hadoop fs -mkdir /python
hadoop fs -put u.data /python

python MovieCountSort.py -r hadoop --hadoop-streaming-jar /home/hdoop/hadoop-3.3.5/share/hadoop/tools/lib/hadoop-streaming-3.3.5.jar hdfs://localhost:9000/python/u.data

```
root@c7556f98e9b1:/home/hdoop/Data# ls
root@c7556f98e9b1:/home/hdoop/Data# python MovieCountSort.py -r hadoop --hadoop-streaming-jar /home/hdoop/hadoop-3.3.5/share/hadoop/tools/lib/hadoop-streaming-
3.3.5.jar hdfs://localhost:9000/python/u.data
No configs found; falling back on auto-configuration
No configs specified for hadoop runner
Looking for hadoop binary in /home/hdoop/hadoop-3.3.5/bin...
Found hadoop binary: /home/hdoop/hadoop-3.3.5/bin/hadoop
Using Hadoop version 3.3.5
Creating temp directory /tmp/MovieCountSort.root.20230416.205906.391625
uploading working dir files to hdfs:///user/root/tmp/mrjob/MovieCountSort.root.20230416.205906.391625/files/wd...
Copying other local files to hdfs:///user/root/tmp/mrjob/MovieCountSort.root.20230416.205906.391625/files/
Running step 1 of 2...
  packageJobJar: [/tmp/hadoop-unjar1894693274270907867/] [] /tmp/streamjob7531455038902271377.jar tmpDir=null
  Connecting to ResourceManager at /127.0.0.1:8032
  Connecting to ResourceManager at /127.0.0.1:8032
  Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1681654231138_0005
  Total input files to process : 1
  number of splits:2
  Submitting tokens for job: job_1681654231138_0005
  Executing with tokens: []
  resource-types.xml not found
  Unable to find 'resource-types.xml'.
  Submitted application application_1681654231138_0005
  The url to track the job: http://c7556f98e9b1:8088/proxy/application_1681654231138_0005/
  Running job: job_1681654231138_0005
  Job job_1681654231138_0005 running in uber mode : false
   map 0% reduce 0%
```

```
"744"   "92"
"578"   "92"
"549"   "92"
"163"   "92"
"17"    "92"
"212"   "92"
"1011"  "93"
"576"   "93"
"820"   "93"
"430"   "93"
"529"   "93"
"412"   "93"
"90"    "95"
"477"   "95"
"131"   "95"
"919"   "96"
"755"   "96"
"290"   "96"
"306"   "96"
"429"   "97"
"356"   "97"
"33"    "97"
"126"   "97"
"1014"  "98"
"155"   "98"
"507"   "98"
"436"   "99"
```

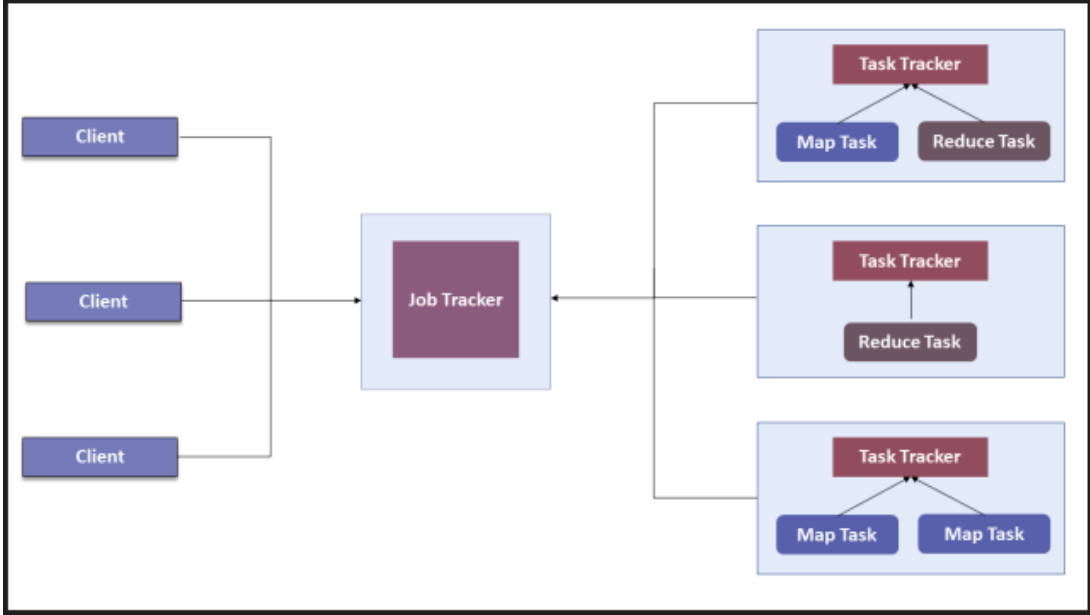
Hadoop YARN 1

Saturday, 15 April 2023
13:29

Hadoop YARN knits the storage unit of Hadoop i.e. HDFS (Hadoop Distributed File System) with the various processing tools. For those of you who are completely new to this topic, YARN stands for "**Y**et **A**nother **R**esource **N**egotiator". I would also suggest that you go through our ***Hadoop Tutorial*** and ***MapReduce Tutorial*** before you go ahead with learning Apache Hadoop YARN. I will be explaining the following topics here to make sure that at the end of this blog your understanding of Hadoop YARN is clear.

Why YARN?

In Hadoop version 1.0 which is also referred to as MRV1(MapReduce Version 1), MapReduce performed both processing and resource management functions. It consisted of a Job Tracker which was the single master. The Job Tracker allocated the resources, performed scheduling and monitored the processing jobs. It assigned map and reduce tasks on a number of subordinate processes called the Task Trackers. The Task Trackers periodically reported their progress to the Job Tracker.



This design resulted in scalability bottleneck due to a single Job Tracker. IBM mentioned in its article that according to Yahoo!, the practical limits of such a design are reached with a cluster of 5000 nodes and 40,000 tasks running concurrently. Apart from this limitation, the utilization of computational resources is inefficient in MRV1. Also, the Hadoop framework became limited only to MapReduce processing paradigm.

To overcome all these issues, YARN was introduced in Hadoop version 2.0 in the year 2012 by Yahoo and Hortonworks. The basic idea behind YARN is to relieve MapReduce by taking over the responsibility of Resource Management and Job Scheduling. YARN started to give Hadoop the ability to run non-MapReduce jobs within the Hadoop framework.

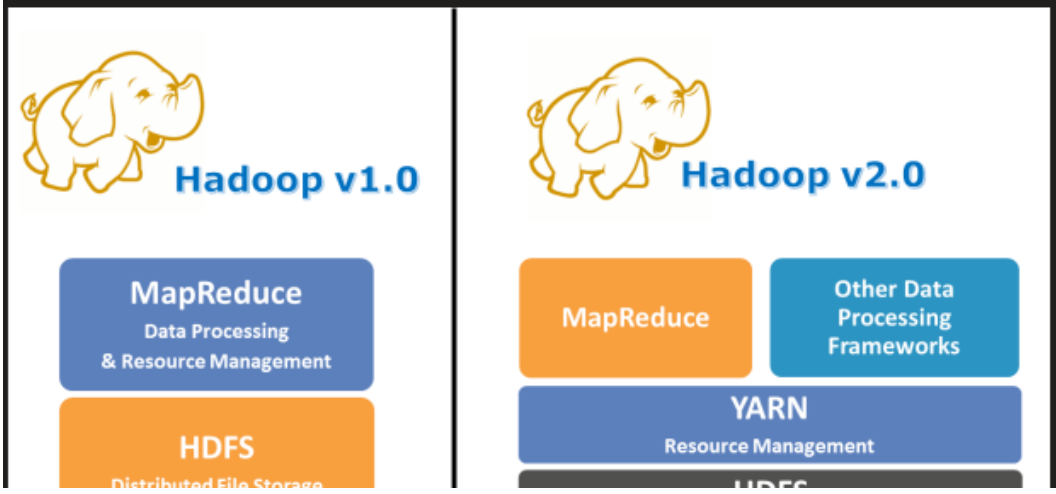
You can also watch the below video where our ***Hadoop Certification Training*** expert is discussing YARN concepts & it's architecture in detail.

Hadoop Yarn Tutorial | Hadoop Yarn Architecture | Edureka

With the introduction of YARN, the ***Hadoop ecosystem*** was completely revolutionalized. It became much more flexible, efficient and scalable. When Yahoo went live with YARN in the first quarter of 2013, it aided the company to shrink the size of its Hadoop cluster from 40,000 nodes to 32,000 nodes. But the number of jobs doubled to 26 million per month.

Introduction to Hadoop YARN

Now that I have enlightened you with the need for YARN, let me introduce you to the core component of Hadoop v2.0, *YARN*. YARN allows different data processing methods like graph processing, interactive processing, stream processing as well as batch processing to run and process data stored in HDFS. Therefore YARN opens up Hadoop to other types of distributed applications beyond MapReduce.



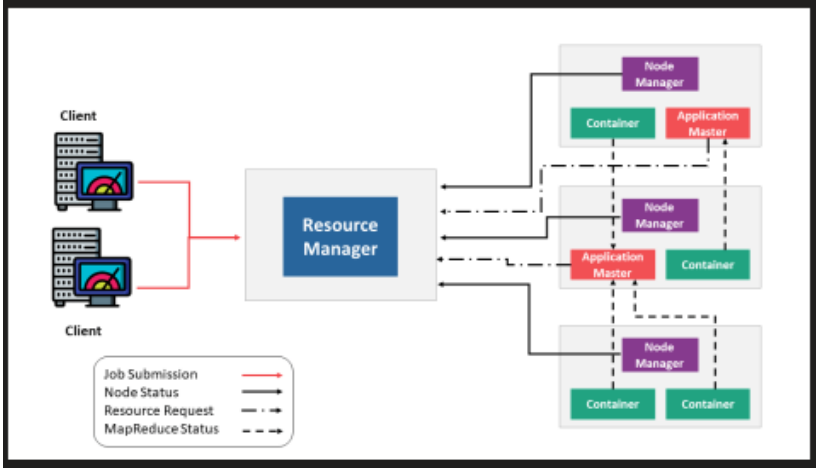
YARN enabled the users to perform operations as per requirement by using a variety of tools like ***Spark*** for real-time processing, ***Hive*** for SQL, ***HBase*** for NoSQL and others.

Apart from Resource Management, YARN also performs Job Scheduling. YARN performs all your processing activities by allocating resources and scheduling tasks. Apache Hadoop YARN Architecture consists of the following main components :

1. **Resource Manager:** Runs on a master daemon and manages the resource allocation in the cluster.
2. **Node Manager:** They run on the slave daemons and are responsible for the execution of a task on every single Data Node.
3. **Application Master:** Manages the user job lifecycle and resource needs of individual applications. It works along with the Node Manager and monitors the execution of tasks.
4. **Container:** Package of resources including RAM, CPU, Network, HDD etc on a single node.

Components of YARN

You can consider YARN as the brain of your Hadoop Ecosystem. The image below represents the YARN Architecture.



The **first component** of YARN Architecture is,

Resource Manager

- It is the ultimate authority in resource allocation

- It is the ultimate authority in resource allocation.
- On receiving the processing requests, it passes parts of requests to corresponding node managers accordingly, where the actual processing takes place.

- It is the arbitrator of the cluster resources and decides the allocation of the available resources for competing applications.
- Optimizes the cluster utilization like keeping all resources in use all the time against various constraints such as capacity guarantees, fairness, and SLAs.

- It has two major components: a) Scheduler b) Application Manager

a) Scheduler

- The scheduler is responsible for allocating resources to the various running applications subject to constraints of capacities, queues etc.
- It is called a pure scheduler in ResourceManager, which means that it does not perform any monitoring or tracking of status for the applications.
- If there is an application failure or hardware failure, the Scheduler does not guarantee to restart the failed tasks.
- Performs scheduling based on the resource requirements of the applications.
- It has a pluggable policy plug-in, which is responsible for partitioning the cluster resources among the various applications. There are two such plug-ins: **Capacity Scheduler** and **Fair Scheduler**, which are currently used as Schedulers in ResourceManager.

b) Application Manager

- It is responsible for accepting job submissions.
- Negotiates the first container from the Resource Manager for executing the application specific Application Master.
- Manages running the Application Masters in a cluster and provides service for restarting the Application Master container on failure.

Coming to the **second component** which is :
Node Manager

- It takes care of individual nodes in a Hadoop cluster and manages user jobs and workflow on the given node.
- It registers with the Resource Manager and sends heartbeats with the health status of the node.
- Its primary goal is to manage application containers assigned to it by the resource manager.

- It keeps up-to-date with the Resource Manager.
- Application Master requests the assigned container from the Node Manager by sending it a Container Launch Context(CLC) which includes everything the application needs in order to run. The Node Manager creates the requested container process and starts it.
- Monitors resource usage (memory, CPU) of individual containers.
- Performs Log management.
- It also kills the container as directed by the Resource Manager.

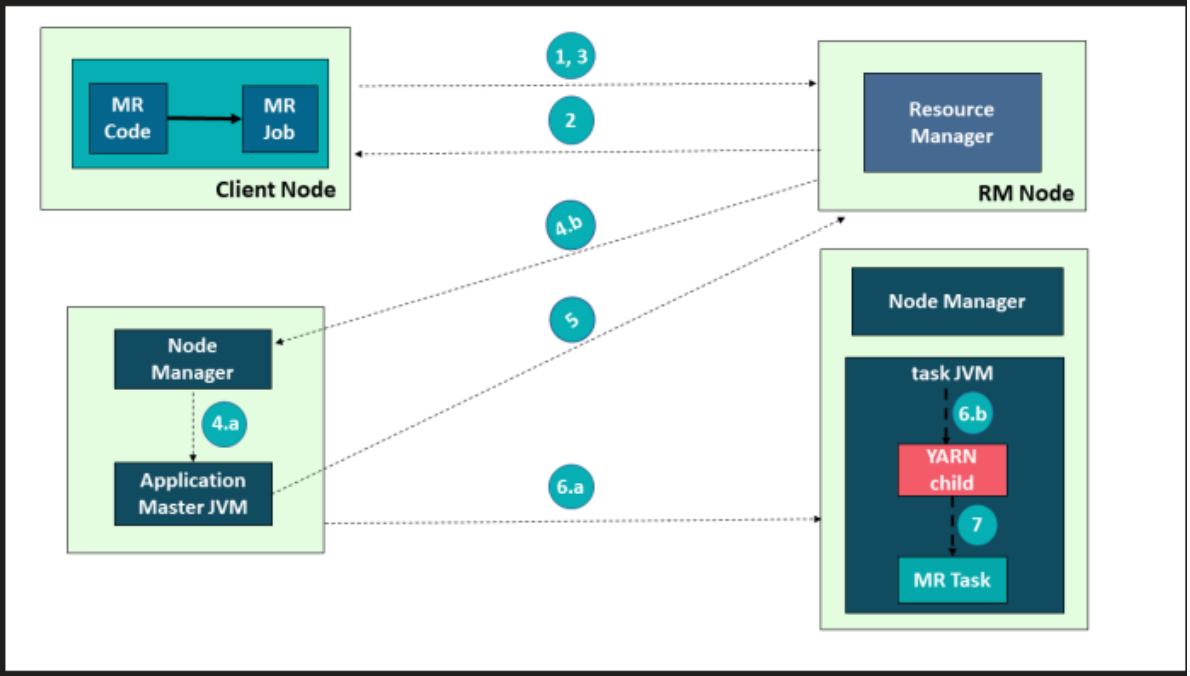
The **third component** of Apache Hadoop YARN is,
Application Master

- An application is a single job submitted to the framework. Each such application has a unique Application Master associated with it which is a framework specific entity.
- It is the process that coordinates an application's execution in the cluster and also manages faults.
- Its task is to negotiate resources from the Resource Manager and work with the Node Manager to execute and monitor the component tasks.
- It is responsible for negotiating appropriate resource containers from the ResourceManager, tracking their status and monitoring progress.
- Once started, it periodically sends heartbeats to the Resource Manager to affirm its health and to update the record of its resource demands.

The **fourth component** is:
Container

- It is a collection of physical resources such as RAM, CPU cores, and disks on a single node.
- YARN containers are managed by a container launch context which is container life-cycle(CLC). This record contains a map of environment variables, dependencies stored in a remotely accessible storage, security tokens, payload for Node Manager services and the command necessary to create the process.
- It grants rights to an application to use a specific amount of resources (memory, CPU etc.) on a specific host.Learn more about Big Data and its applications from the Data Engineering certification.

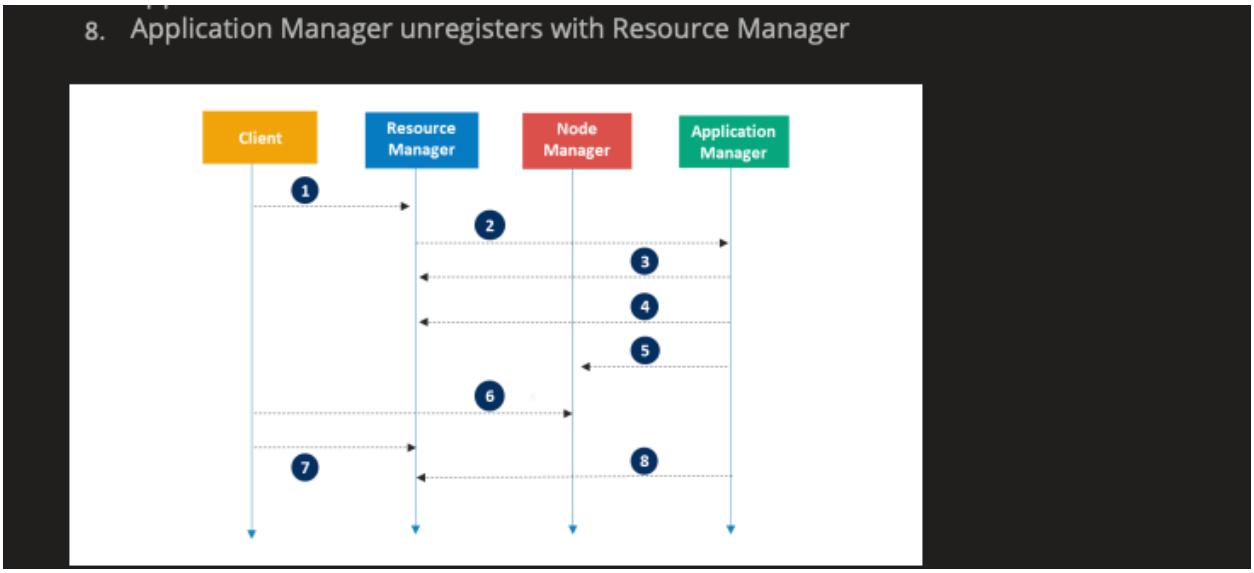
1) Submit the job
2) Get Application ID
3) Application Submission Context
4 a) Start Container Launch
  b) Launch Application Master
5) Allocate Resources
6 a) Container
  b) Launch
7) Execute



Application Workflow in Hadoop YARN

Refer to the given image and see the following steps involved in Application workflow of Apache Hadoop YARN: You can get a better understanding with the Data Engineering Training in India.

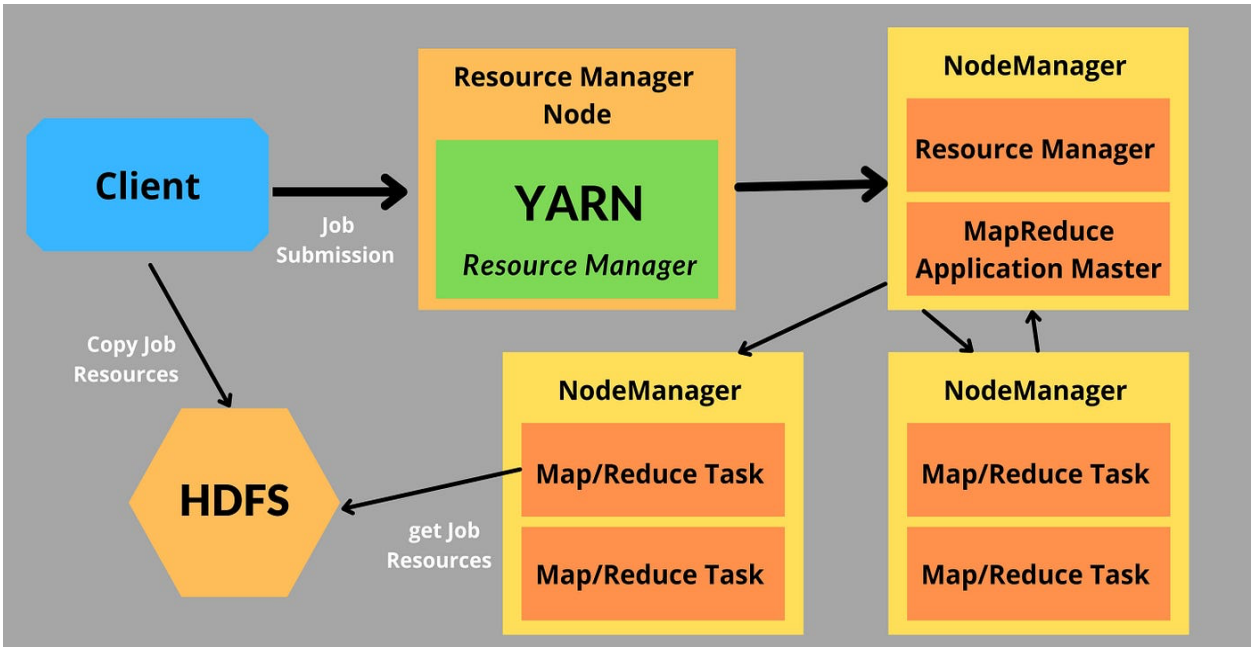
1. Client submits an application
2. Resource Manager allocates a container to start Application Manager
3. Application Manager registers with Resource Manager
4. Application Manager asks containers from Resource Manager
5. Application Manager notifies Node Manager to launch containers
6. Application code is executed in the container
7. Client contacts Resource Manager/Application Manager to monitor application's status



Hadoop YARN 2

Sunday, 16 April 2023
12:58

What's happening under the hood



- Client node will submit MapReduce Jobs to The resource manager Hadoop YARN.
- Hadoop YARN resource managing and monitoring the clusters such as keeping track of available capacity of clusters, available clusters, etc. Hadoop Yarn will copy the needful data Hadoop Distribution File System(HTFS) in parallel.
- Next Node Manager will manage all the MapReduce jobs. MapReduce application master located in Node Manager will keep track of each of the Map and Reduce tasks and distribute it across the cluster with the help of YARN.
- Map and Reduce tasks connect with HDFS cluster to get needful data to process and output data.

Resume Architecture

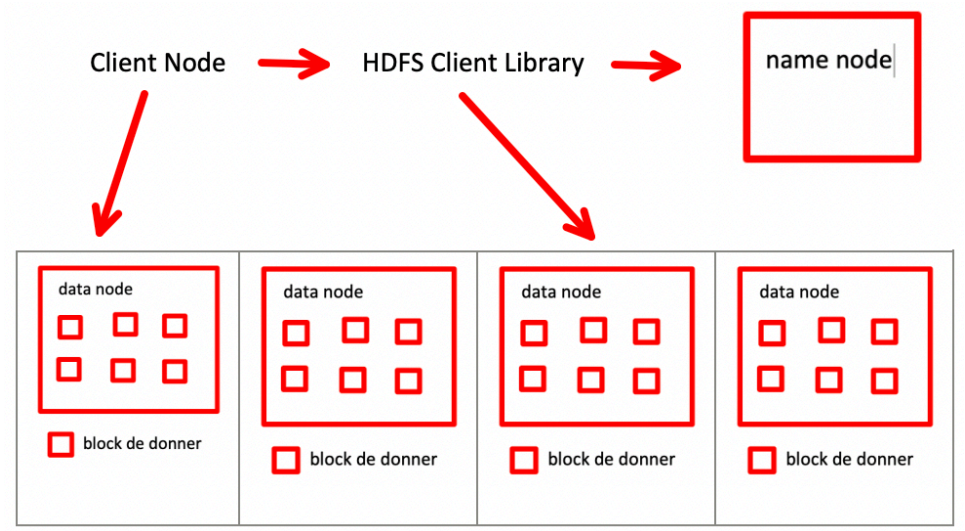
Wednesday, 19 April 2023
11:21

HDFS il gere le stockage

Name Node Resource Manager
Data Node Node Manager

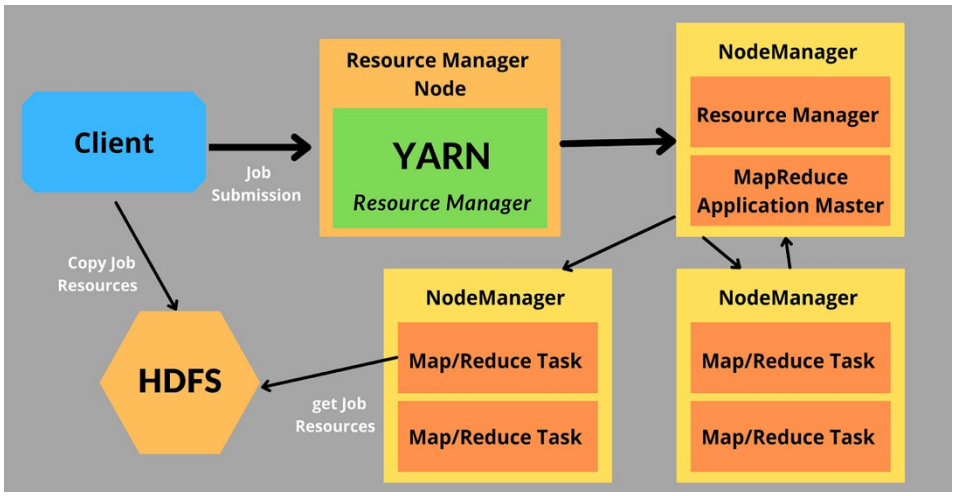
Client ----> hdfs client <client **hdfs dfs>** (ou est la donnees) ----> name Node (hdfs)

Hdfs client ---> data Node



YARN il gere les ressource

Resource manager gere la disponibiliter des resource au niveau des NodeManager de mon cluster
MapReduce application master track l'execution des tache mapReduce



- Client node will submit MapReduce Jobs to The resource manager Hadoop YARN.

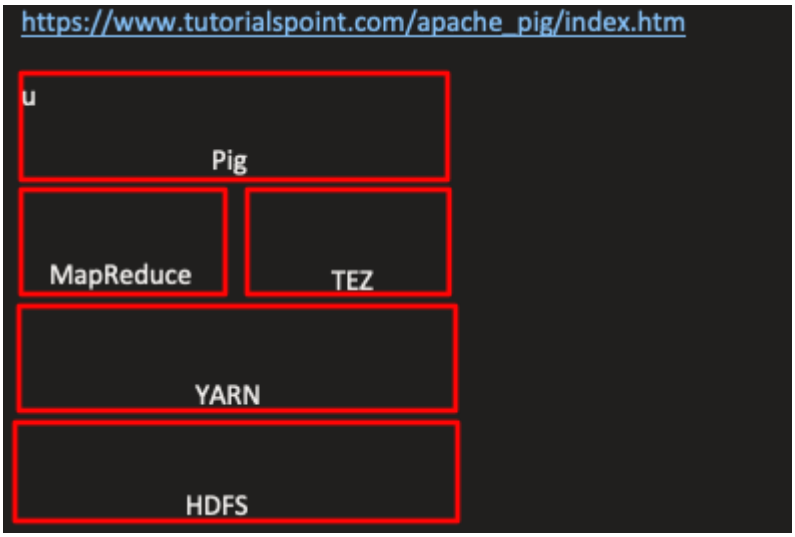
- Hadoop YARN resource managing and monitoring the clusters such as keeping track of available capacity of clusters, available clusters, etc. Hadoop Yarn will copy the needful data Hadoop Distribution File System(HTFS) in parallel.
- Next Node Manager will manage all the MapReduce jobs. MapReduce application master located in Node Manager will keep track of each of the Map and Reduce tasks and distribute it across the cluster with the help of YARN.
- Map and Reduce tasks connect with HDFS cluster to get needful data to process and output data.

Here is the life-cycle of MapReduce **Application Master**(AM):

- Each application running on the Hadoop cluster has its own, dedicated Application Master instance, which runs in a container on a slave node. One Application Master per application.
- Throughout its life (while the application is running), the Application Master sends heartbeat messages to the Resource Manager with its status and the state of the application's resource needs.
- The Application Master oversees/supervise the full life-cycle of an application, all the way from requesting the needed containers from the Resource Manager to submitting container lease requests to the Node Manager.
- Each application framework that's written for Hadoop must have its own Application Master implementation. Example: MapReduce application has a specific Application Master that's designed to execute map tasks and reduce tasks in sequence.

Pig

Thursday, 20 April 2023
11:23



TEZ permet d'enchainer plusieur MapReducer

Pig fonctionne 10 fois plus rapide si il fonctionne on haut de TEZ

U.data ---> userID:int , movieID:int , rating:int , ratingTime:int
U.item ---> movieID:int , movieTitle:chararray , releaseDate:chararray , videoRelease:chararray , imdbLink:chararray

data1.txt

1|aymane|hinane|22|casa
2|yohane|hinane|23|rabat
3|alie|ahmed|22|casa
4|noor|midawi|21|casa

tuple.txt

1|(aymane,hinane)|22|(casa,mdina)
2|(yohane,hinane)|23|(rabat,m6)
3|(noor,alaoui)|21|(casa,mdina)
4|(ahmed,masour)|22|(rabat,h5)

tuple: une série de données.
Par exemple : (12, John, 5.3)

bag: un ensemble de tuples
Par exemple : { (12, John, 5.3), (8), (3.5, TRUE, Bob, 42) }

map: une série de couples (clé#valeur)
Par exemple : [qui#22, le#3] (ici avec deux couples clé#valeur) Au sein d'un type map, chaque clé doit être unique

Un tuple peut tout à fait contenir d'autres tuples, des bags, ou encore autres types simples et complexes.
Par exemple:
({ (1, 2), (3, John) }, 3, [qui#23])

bag contenant lui-même trois tuples
map contenant un seul couple (clé#valeur)

mots.txt
hello world
hello world
hello aymane

show the most popular five star movies

pig -x local script1.pig
3 seconds and 935 milliseconds

pig -x mapreduce script1.pig

11 minutes, 16 seconds and 861 milliseconds

```
ratings = LOAD '/home/hdoop/Data/u.data' AS (userID:int,movieID:int,rating:int,ratingTime:int);

DUMP ratings;
```

(721,262,3,877137285)
(913,209,2,881367150)
(378,78,3,880056976)
(880,476,3,880175444)
(716,204,5,879795543)
(276,1090,1,874795795)
(13,225,2,882399156)
(12,203,3,879959583)

DESCRIBE ratings;

ratings: {userID: int,movieID: int,rating: int,ratingTime: int}

---

ratingsByMovie = GROUP ratings BY movieID;

DUMP ratingsByMovie;

(1656,{(713,1656,2,888882085),(883,1656,5,891692168)})
(1658,{(733,1658,3,879535780),(894,1658,4,882404137),(782,1658,2,891500230)})
(1659,{(747,1659,1,888733313)})
(1662,{(762,1662,1,878719324),(782,1662,4,891500110)})
(1664,{(870,1664,4,890057322),(880,1664,4,892958799),(839,1664,1,875752902),(782,1664,4,8914996
99)})

DESCRIBE ratingsByMovie;

ratingsByMovie: {group: int,ratings: {(userID: int,movieID: int,rating: int,ratingTime: int)}}

---

avgRatings = FOREACH ratingsByMovie GENERATE group AS movieID,AVG(ratings.rating) AS avgRating;

DUMP avgRatings;

(1673,3.0)
(1674,4.0)
(1675,3.0)
(1676,2.0)
(1677,3.0)
(1678,1.0)

DESCRIBE avgRatings;

avgRatings: {movieID: int,avgRating: double}

---

fiveStarMovies = FILTER avgRatings BY avgRating > 4.0;

DUMP fiveStarMovies;

(1039,4.011111111111111)
(1064,4.25)
(1122,5.0)

(1125,4.25)
(1142,4.045454545454546)
(1169,4.1)
(1189,5.0)
(1191,4.333333333333333)
(1194,4.064516129032258)
(1201,5.0)
(1203,4.0476190476190474)

```
DESCRIBE fiveStarMovies;
```

```
fiveStarMovies: {movieID: int,avgRating: double}
```

```
metadata = LOAD '/home/hdoop/Data/u.item' USING PigStorage('|')
        AS
(movieID:int,movieTitle:chararray,releaseDate:chararray,videoRelease:chararray,imdbLink:chararray);
```

```
DUMP ratings;
```

(880,476,3,880175444)
(716,204,5,879795543)
(276,1090,1,874795795)
(13,225,2,882399156)
(12,203,3,879959583)

```
DESCRIBE metadata;
```

```
metadata: {movieID: int,movieTitle: chararray,releaseDate: chararray,videoRelease: chararray,imdbLink:
chararray}
```

```
nameLookup = FOREACH metadata GENERATE movieID,movieTitle,
        ToUnixTime(ToDate(releaseDate,'dd-MMM-yyyy')) AS releaseTime;
```

```
DUMP nameLookup;
```

(1679,B. Monkey (1998),886723200)
(1680,Sliding Doors (1998),883612800)
(1681,You So Crazy (1994),757382400)
(1682,Scream of Stone (Schrei aus Stein) (1991),826243200)

```
DESCRIBE nameLookup;
```

```
nameLookup: {movieID: int,movieTitle: chararray,releaseTime: long}
```

---

fiveStarsWithData = JOIN fiveStarMovies BY movieID,nameLookup BY movieID;

DUMP fiveStarsWithData;

(1594,4.5,1594,Everest (1998),889488000)
(1599,5.0,1599,Someone Else's America (1995),831686400)
(1639,4.333333333333333,1639,Bitter Sugar (Azucar Amargo) (1996),848620800)
(1642,4.5,1642,Some Mother's Son (1996),851644800)
(1653,5.0,1653,Entertaining Angels: The Dorothy Day Story (1996),843782400)

DESCRIBE fiveStarsWithData;

fiveStarsWithData: {fiveStarMovies::movieID: int,fiveStarMovies::avgRating:
double,nameLookup::movieID: int,nameLookup::movieTitle: chararray,nameLookup::releaseTime: long}

---

oldestFiveStartMovies = ORDER fiveStarsWithData BY nameLookup::releaseTime;
DUMP oldestFiveStartMovies;

(100,4.155511811023622,100,Fargo (1996),855878400)
(181,4.007889546351085,181,Return of the Jedi (1983),858297600)
(515,4.203980099502488,515,Boot, Das (1981),860112000)
(1251,4.125,1251,A Chef in Love (1996),861926400)
(251,4.260869565217392,251,Shall We Dance? (1996),868579200)
(316,4.196428571428571,316,As Good As It Gets (1997),882835200)
(1293,5.0,1293,Star Kid (1997),884908800)
(1191,4.333333333333333,1191,Letter From Death Row, A (1998),886291200)
(1594,4.5,1594,Everest (1998),889488000)
(315,4.1,315,Apt Pupil (1998),909100800)

##

Le livre le plus critique:
qui a une moyen3 de note < 2 et qui a etait le plus note

```
ratings  = LOAD '/home/hdoop/Data/u.data' AS (userID:int,movieID:int,rating:int);

--DUMP ratings;

ratingByMovie = GROUP ratings BY movieID;

--DUMP ratingByMovie;

--avgRatings = FOREACH ratingByMovie GENERATE group as
movieID,ratings.userID,COUNT(ratings.userID) as userCount,COUNT(ratings.rating) as
ratingCount,AVG(rati>
--DUMP avgRatings;

avgRatings = FOREACH ratingByMovie GENERATE group as movieID,AVG(ratings.rating) as
ratingAVG,COUNT(ratings.rating) as ratingCount;

movieLessThan = Filter avgRatings BY ratingAVG < 2.0;
--DUMP movieLessThan;


orderMovieAvgRanting = Order movieLessThan By ratingCount  DESC;

limit_data = LIMIT orderMovieAvgRanting 1;

--DUMP limit_data;

metaData = LOAD '/home/hdoop/Data/u.item' USING PigStorage('|')
        AS
(movieID:int,movieTitle:chararray,releaseDate:chararray,videoRelease:chararray,imdbLink:chararray);


movieData = JOIN limit_data BY movieID , metaData BY movieID;

--DUMP movieData;


######################################################################
```

Pig integrated with Tez take less time and also do it in a very quick and efficient manner.

Tez uses

what's called a directed acyclic graph to

actually analyze all the interrelationships

between the different steps that you're doing

and try to figure out the

most optimal path for executing things.

Pig Note

Friday, 21 April 2023
02:14

Startup
=======
run pig locally:
$ magicpig -x local script.pig # doesn't work

expand pig macros:
$ magicpig -dryrun script.pig

Commands
========

Loading
----
grunt> A = LOAD '/path/to/file' USING PigStorage(':') AS (field1:float, field2:int, ...);

Load data from /path/to/file and name the fields field1, field2, ... sequentially. Fields are split by ':'. field1 is loaded as a float, field2 as an integer. The 'AS' part is optional. Other basic types are int, long, float, double, bytearray, boolean, chararray.

Pig also supports complex types are: tuple, bag, map. For example,

grunt> A = LOAD '/path/to/file' USING PigStorage(':') AS (field1:tuple(t1a:int, t1b:int,t1c:int), field2:chararray);

This will load field1 as a tuple with 3 values that can be referenced as field1.t1a, field1.t1b, etc. Don't worry about bags and maps.

Saving
------
grunt> STORE A INTO '/path/out' USING AvroStorage();

Save all of A's fields into /path/out in the format defined by AvroStorage

Generating
---
grunt> B = FOREACH A GENERATE $0 + 1, org.apache.pig.tutorial.ExtractHour(field2) as field2hour;

Use this to select the first field and add one to it, and select the hour part of field2 from A and rename it as field2hour. You have your basic arithmetic operations and %. You can also use "*" to select all fields.

Here's an example of a nested foreach. Only distinct, filter, limit, and order are supported inside a nested foreach.

```
daily   = load 'NYSE_daily' as (exchange, symbol); -- not interested in other fields
grpd    = group daily by exchange;
uniqcnt  = foreach grpd {
        sym     = daily.symbol;
        uniq_sym = distinct sym;
        generate group, COUNT(uniq_sym); };
```

Here's an example for selecting _ranges_ of fields,

```
rices = load 'NYSE_daily' as (exchange, symbol, date, open, high, low, close, volume, adj_close);
beginning = foreach prices generate ..open; -- produces exchange, symbol, date, open
middle   = foreach prices generate open..close; -- produces open, high, low, close
end      = foreach prices generate volume..; -- produces volume, adj_close
```

Here's how to use the "<condition> ? <if true> : <if false>" construct

```
B = FOREACH A GENERATE field1 > 500 ? 1 : -1
```

Filtering
---
grunt> B = FILTER A by $1 > 2;

Remove entries from A whose second field is <= 2. You can use basic numerical comparisons, along with "is null" and "matches" (for glob matching). e.g,

grunt> B = FILTER A by (url matches '*.apache.com');

Grouping
---
grunt> B = GROUP A BY (field1, field2), B by (fieldA, fieldB);

Match entries in A to entries in B if (field1, field2) == (fieldA, fieldB), and return values of both grouped by their respective keys. If A and B share the same field names, use A.field1 or B.field1. The key will have alias "group".

grunt> B = GROUP A ALL;

Group all of A into a single group instead of by field

Joining
---
grunt> C = JOIN A BY field1 LEFT, B BY field2 USING 'replicated' PARALLEL 5;

Joint two variables by field names. Must be done after a GROUP operation on both A and B. Uses 'replicated' method to join, alternatives being 'skewed', 'merge', and normal (hash). Uses 5 reduce tasks. Does a left join (all rows in A will be kept, but rhs might be null).

There exist multiple types of joins with their own performance characteristics. They are listed below in order of preference.

TypeAlign smallest toNumber of joinableOther restrictionsHow it worksWhen to use
replicated
right2+inner/left outer onlyLoads rightmost alias into distributed cacheOne alias is really small
mergeright2aliases must be sorted; inner onlySamples an index of right hand alias, then splits data into mappers who use index to lookup where to start reading from on rightaliases are already sortedskewedleft2Samples right side's keys to determine which keys have too many entries. All but the heaviest are handled by hash join, and all others are treated similar to a replicated join with the left loaded into memory.
One or more aliases have a heavily skewed distribution over the keys (e.g., one has 100k entries, the other 5)hashleft2+Groups data by key, then sends the leftmost alias to each reducer and loads it into memory, then second etc. Last alias is streamed through.When you have no other choice.

Flattening
---
grunt> B = FOREACH A GENERATE flatten(field1) as field1flat, field2;

If the argument to flatten is a tuple, then flatten it like a list: e.g., ((a,b), c) -> (a, b, c).
If the argument is a bag, then make the cross product: e.g. a:{(b,c), (d,e)} -> {(a,b,c), (a,d,e)}

Unique
---
grunt> B = DISTINCT A;

Use this to turn A into a set. All fields in A must match to be grouped together

Sorting

---
grunt> B = ORDER A by (field1, field2);

Cross Product
---
grunt> C = CROSS A, B;

Take the cross product of A and B's elements. Your machine will shit itself if you do this. You cannot cross an alias with itself due to namespace conflicts.

Splitting
---
grunt> SPLIT B INTO A1 IF field1 < 3, A2 IF field2 > 7;

Cut B into two parts based on conditions. Entries can end up in both A1 and A2.

Subsampling
---
grunt > B = SAMPLE A 0.01;

Sample 1% of A's entries (in expectation)

Viewing interactively
---
grunt> B = limit A 500;
grunt> DUMP B;
grunt> DESCRIBE B;

Take the first 500 entries in A, and print them to screen, then print out the name of all of B's fields.

Built in Functions
==========

* Eval Functions
  * AVG
  * CONCAT
  * COUNT
  * COUNT_STAR
  * DIFF
  * IsEmpty
  * MAX
  * MIN
  * SIZE

    * SUM
    * TOKENIZE
  * Math Functions
    * ABS
    * ACOS
    * ASIN
    * ATAN
    * CBRT
    * CEIL
    * COS
    * COSH
    * EXP
    * FLOOR
    * LOG
    * LOG10
    * RANDOM
    * ROUND
    * SIN
    * SINH
    * SQRT
    * TAN
    * TANH
  * String Functions
    * INDEXOF
    * LAST_INDEX_OF
    * LCFIRST
    * LOWER
    * REGEX_EXTRACT
    * REGEX_EXTRACT_ALL
    * REPLACE
    * STRSPLIT
    * SUBSTRING
    * TRIM
    * UCFIRST
    * UPPER
  * Tuple, Bag, Map Functions
    * TOTUPLE
    * TOBAG
    * TOMAP
    * TOP

Macros in Pig

==========

Basic usage example. Prefix arguments with "$". You can use aliases and literals, as they're literally subbing text in.

```
DEFINE group_and_count (A, group_key, reducers) RETURNS B {
  D = GROUP $A BY '$group_key' PARALLEL $reducers;
  $B = FOREACH D GENERATE group, COUNT($A);
}; X = LOAD 'users' AS (user, age, zip);
Y = group_and_count (X, 'user', 20);
Z = group_and_count (X, 'age', 30);Here's an example of one macro calling another.
DEFINE foreach_count(A, C) RETURNS B {
  $B = FOREACH $A GENERATE group, COUNT($C);
}; DEFINE group_with_parallel (A, group_key, reducers) RETURNS B {
  C = GROUP $A BY $group_key PARALLEL $reducers;
  $B = foreach_count(C, $A);
};Here's an example where a string is transformed into interpreted pig.
/* Get a count of records, return the name of the relation and . */
DEFINE total_count(relation) RETURNS total {
 $total = FOREACH (group $relation all) generate '$relation' as label, COUNT_STAR($relation) as total;
};
/* Get totals on 2 relations, union and return them with labels */
DEFINE compare_totals(r1, r2) RETURNS totals {
 total1 = total_count($r1);
 total2 = total_count($r2);
 $totals = union total1, total2;
};
/* See how many records from a relation are removed by a filter, given a condition */
DEFINE test_filter(original, condition) RETURNS result {
 filtered = filter $original by $condition;
 $result = compare_totals($original, filtered);

};



emails = load '/me/tmp/inbox' using AvroStorage();
out = test_filter(emails, 'date is not null'); Pitfalls
--------
- You can't use parameter substitution in a macro (pass them in explicitly as arguments)
- You can't use any of the debug commands (DESCRIBE, ILLUSTRATE, EXPLAIN, DUMP) in a macro
- If you do a filter on a numeric condition and the input is null, the result is false.  e.g., null > 0 == false

Parameters
==========
```

Here's how to do direct substitutions with variables in Pig. Note that "%".

```
%default parallel_factor 10;
wlogs = load 'clicks' as (url, pageid, timestamp);
grp   = group wlogs by pageid parallel $parallel_factor;
cntd  = foreach grp generate group, COUNT(wlogs);
```

Functions outside of Pig
==========

There exist a boatload of ways to write [U]ser [D]efined [F]unctions. Below is a simple EvalFunc<ReturnType> for doing a map operation, but there are also FilterFunc. EvalFunc<ReturnType> is also used for aggregation operations, but can be more efficient using the Algebraic and Accumulator interfaces.

Java

.. code::

```
  /**
   * A simple UDF that takes a value and raises it to the power of a second
   * value.  It can be used in a Pig Latin script as Pow(x, y), where x and y
   * are both expected to be ints.
   */
  public class Pow extends EvalFunc<Long> {

      public Long exec(Tuple input) throws IOException {
          try {
              /* Rather than give you explicit arguments UDFs are always handed
               * a tuple.  The UDF must know the arguments it expects and pull
               * them out of the tuple.  These next two lines get the first and
               * second fields out of the input tuple that was handed in.  Since
               * Tuple.get returns Objects, we must cast them to Integers.  If
               * the case fails an exception will be thrown.
               */
              int base = (Integer)input.get(0);
              int exponent = (Integer)input.get(1);
              long result = 1;

              /* Probably not the most efficient method...*/
              for (int i = 0; i < exponent; i++) {
                  long preresult = result;
                  result *= base;
```

```
            if (preresult > result) {
                // We overflowed.  Give a warning, but do not throw an
                // exception.
                warn("Overflow!", PigWarning.TOO_LARGE_FOR_INT);
                // Returning null will indicate to Pig that we failed but
                // we want to continue execution
                return null;
            }
        }
        return result;
    } catch (Exception e) {
        // Throwing an exception will cause the task to fail.
        throw new IOException("Something bad happened!", e);
    }
}

public Schema outputSchema(Schema input) {    // Check that we were passed two fields
    if (input.size() != 2) {
        throw new RuntimeException(
            "Expected (int, int), input does not have 2 fields");
    }


    try {
        // Get the types for both columns and check them.  If they are
        // wrong figure out what types were passed and give a good error
        // message.
        if (input.getField(0).type != DataType.INTEGER ||
                input.getField(1).type != DataType.INTEGER) {
            String msg = "Expected input (int, int), received schema (";
            msg += DataType.findTypeName(input.getField(0).type);
            msg += ", ";
            msg += DataType.findTypeName(input.getField(1).type);
            msg += ")";
            throw new RuntimeException(msg);
        }
    } catch (Exception e) {
        throw new RuntimeException(e);
    }


    // Construct our output schema which is one field, that is a long
    return new Schema(new FieldSchema(null, DataType.LONG));
```

```
    }
  }
```

Python

- in Pig,

.. code::

```
  grunt> Register 'test.py' using jython as myfuncs;
  grunt> b = foreach a generate myfuncs.helloworld(), myfuncs.square(3);
```

- in Python

.. code::

```
  @outputSchemaFunction("squareSchema")
  def pow(n1, n2):
   return n1**n2

  @schemaFunction("squareSchema")
  def squareSchema(input):     # this says int -> int, long -> long, float -> float, etc
   return input
```

.. code::

```
  @outputSchema("production:float")
  def production(slugging_pct, onbase_pct):
     return slugging_pct + onbase_pct
```

Embedding Pig in Python
=======================

Pig doesn't have control structures (if, for, etc), so jobs that inherently have time-varying file names or are repeated until convergence can't be controlled from within Pig. You can fix this by embedding Pig in Python via Jython.

.. code::

```
from org.apache.pig.scripting import *

# Compile a pig job named "scriptname"
P1 = Pig.compile("scriptname",
 """
  A = LOAD 'input';
  B = FILTER A BY field > $lower_bound;
  STORE B INTO '$outpath';

  n_entries = GROUP B BY ALL;
  n_entries = FOREACH n_entries GENERATE COUNT(*);
  STORE n_entries INTO '$count_path';
 """
)

# pass parameters
P1_bound = P1.bind(
 {
   'lower_bound': 500,
   'outpath': '/path/to/save',
   'count_path': '/tmp/count'
 }
)

# Do some non-pig operations
Pig.fs( "RMR %s /path/to/save" )

# run script
stats = P1_bound.runSingle()

# check if successful?
if stats.isSuccessful():
  print 'Yay!  it succeeded!'

  # extract alias 'n_entries' from the script and get its first element
  count = float(str(stats.result("n_entries").iterator().next().get(0)))
  print 'Output %d rows' % (count,)
```

Pig Exercice

Tuesday, 16 May 2023

09:32
Un tuple peut contenir d'autres tuple , des bag ou des types primitive

({ (aymane,hinane) , (casa,ain) } , (22,16.5), maroc)

++

```
ratings =LOAD 'hdfs://localhost:9000/python/u.data'
            AS (userID:int,movieID:int,rating:int,ratingTime:int);
DUMP ratings;
DESCRIBE ratings;
```

(13,225,2,882399156)
(12,203,3,879959583)
ratings: {userID: int,movieID: int,rating: int,ratingTime: int}

++

```
metadata = LOAD 'hdfs://localhost:9000/python/u.item' USING PigStorage('|')
          AS
(movieID:int,movieTitle:chararray,releaseDate:chararray,videoRelease:chararray,imdbLink:chararray);

DUMP metadata;
DESCRIBE metadata;
```

(1680,Sliding Doors (1998),01-Jan-1998,,http://us.imdb.com/Title?Sliding+Doors+(1998))
(1681,You So Crazy (1994),01-Jan-1994,,http://us.imdb.com/M/title-exact?You%20So%20Crazy%20(1994))
(1682,Scream of Stone (Schrei aus Stein) (1991),08-Mar-1996,,http://us.imdb.com/M/title-exact?Schrei%20aus%20Stein%20(1991))

metadata: {movieID: int,movieTitle: chararray,releaseDate: chararray,videoRelease: chararray,imdbLink: chararray}

++

```
user2 = LOAD './data1.txt' USING PigStorage('|')
          AS (id:int,lastName:chararray,firstName:chararray,age:int,city:chararray);
DUMP user2;
DESCRIBE user2;
```

(1,aymane,hinane,22,casa)
(2,yohane,hinane,23,rabat)
(3,alie,ahmed,22,casa)
(4,noor,midawi,21,casa)
user2: {id: int,lastName: chararray,firstName: chararray,age: int,city: chararray}

++

```
filter1 = FILTER user2 BY age > 21 AND id>1;
DUMP filter1;
```

```
(2,yohane,hinane,23,rabat)
(3,alie,ahmed,22,casa)
```

++

```
filter2 = FILTER user2 By lastName MATCHES 'yohane';
DUMP filter2;
```

```
(2,yohane,hinane,23,rabat)
```

++

```
filter3 = ORDER user2 BY age ASC;
DUMP filter3;
```

```
(4,noor,midawi,21,casa)
(3,alie,ahmed,22,casa)
(1,aymane,hinane,22,casa)
(2,yohane,hinane,23,rabat)
```

++

```
filter4 = GROUP user2 BY age;
DUMP filter4;
DESCRIBE filter4;
```

```
(21, {(4,noor,midawi,21,casa)} )
(22, { (3,alie,ahmed,22,casa) , (1,aymane,hinane,22,casa) } )
(23,{ (2,yohane,hinane,23,rabat) } )

filter4: {group: int,user2: {(id: int,lastName: chararray,firstName: chararray,age: int,city: chararray)}}
```

++

```
z = FOREACH filter4 GENERATE FLATTEN(user2);
DUMP z;
DESCRIBE z;
```

```
(4,noor,midawi,21,casa)
(3,alie,ahmed,22,casa)
(1,aymane,hinane,22,casa)
(2,yohane,hinane,23,rabat)
```

z: {user2::id: int,user2::lastName: chararray,user2::firstName: chararray,user2::age: int,user2::city: chararray}

++

```
A = FOREACH user2 GENERATE age;
DUMP A;
DESCRIBE A;
```

(22)
(23)
(22)
(21)
A: {age: int}

++

```
B = FOREACH user2 GENERATE (firstName,lastName) as fullName,age;
DUMP B;
DESCRIBE B;
```

((hinane,aymane),22)
((hinane,yohane),23)
((ahmed,alie),22)
((midawi,noor),21)
B: {fullName: (firstName: chararray,lastName: chararray),age: int}

++

```
C= LOAD './tuple.txt' USING PigStorage('|')
   AS
(id:int,name:tuple(lastName:chararray,firstName:chararray),age:int,address:tuple(ville:chararray,rue:chararray));
```

(1,(aymane,hinane),22,(casa,mdina))
(2,(yohane,hinane),23,(rabat,m6))
(3,(noor,alaoui),21,(casa,mdina))
(4,(ahmed,masour),22,(rabat,h5))

C: {id: int,name: (lastName: chararray,firstName: chararray),age: int,address: (ville: chararray,rue: chararray)}

++

```
user3 = LOAD './data1.txt' USING PigStorage('|')
        AS (id:int,prenom:chararray,nom:chararray,age:int,city:chararray);
```

(1,aymane,hinane,22,casa)

(2,yohane,hinane,23,rabat)
(3,alie,ahmed,22,casa)
(4,noor,midawi,21,casa)
user3: {id: int,prenom: chararray,nom: chararray,age: int,city: chararray}

++

groupByAge = GROUP user3 BY age;
DUMP groupByAge;
DESCRIBE groupByAge;

(21, { (4,noor,midawi,21,casa) })
(22, { (3,alie,ahmed,22,casa),(1,aymane,hinane,22,casa) })
(23, { (2,yohane,hinane,23,rabat) })

groupByAge: {group: int,user3: {(id: int,prenom: chararray,nom: chararray,age: int,city: chararray)}}

++

D = FOREACH user3 GENERATE (nom,prenom) as name,age;
DUMP D;
DESCRIBE D;

((hinane,aymane),22)
((hinane,yohane),23)
((ahmed,alie),22)
((midawi,noor),21)
D: {name: (nom: chararray,prenom: chararray),age: int}

++
groupD = GROUP D BY age;
DUMP groupD;
DESCRIBE groupD;

(21,{ ((midawi,noor),21) })
(22,{ ((ahmed,alie),22) , ((hinane,aymane),22) })
(23,{ ((hinane,yohane),23) })
groupD: {group: int,D: {(name: (nom: chararray,prenom: chararray),age: int)}}

++

flatternD = FOREACH groupD GENERATE FLATTEN(D);
DUMP flatternD;
DESCRIBE flatternD;

((midawi,noor),21)
((ahmed,alie),22)
((hinane,aymane),22)

((hinane,yohane),23)
flatternD: {D::name: (nom: chararray,prenom: chararray),D::age: int}

++

```
mots = LOAD './mots.txt' USING TextLoader as ligne:chararray;
DUMP mots;
DESCRIBE mots;
```

(hello world)
(hello world)
(hello aymane)
mots: {ligne: chararray}

++++++++++++++++++++++++++

```
E = FOREACH mots GENERATE TOKENIZE(ligne) AS mots;
DUMP E;
DESCRIBE E;
```

({(hello),(world)})
({(hello),(world)})
({(hello),(aymane)})
E: {mots: {tuple_of_tokens: (token: chararray)}}

++++++++++++++++++++++++++

```
F = FOREACH E GENERATE FLATTEN(mots) as mot;
DUMP F;
DESCRIBE F;
```

(hello)
(world)
(hello)
(world)
(hello)
(aymane)
F: {mot: chararray}

++++++++++++++++++++++++++

```
G = GROUP F BY mot;
DUMP G;
DESCRIBE G;
```

(hello,{(hello),(hello),(hello)})
(world,{(world),(world)})
(aymane,{(aymane)})

G: {group: chararray,F: {(mot: chararray)}}

++++++++++++++++++++++++++

J = FOREACH G GENERATE group as mot,COUNT(F) as occurence;
DUMP J;
DESCRIBE J;

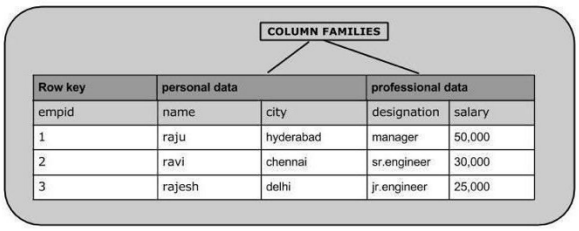
(hello,3)
(world,2)
(aymane,1)
J: {mot: chararray,occurence: long}

Hbase

Saturday, 13 May 2023
19:13
start-hbase.sh

HBase

• Hbase est un SGBD NoSQL orienté colonne.



HBase utilise le système de fichiers Hadoop pour stocker ses données. Il aura un serveur maître et des serveurs de région. Le stockage de données sera sous la forme de régions (tables). Ces régions seront divisées et stockées dans des serveurs régionaux.

create 'emp', 'personal data', 'professional data'

id	personal data	professional data

Affiche la liste des tables
>list

ALTER TABLE my_table ADD COLUMNS (new_column1 int COMMENT 'New integer column', new_column2 string COMMENT 'New string column');

put ‘<table name>’,’row1’,’<colfamily:colname>’,’<value>’

put 'emp','1','personal data:name','raju'

get 'emp', 'row1', {COLUMN ⇒ 'personal:name'}

delete 'emp', '1', 'personal data:city'

ALTER TABLE existing_parquet ADD COLUMNS (address.house_no INTEGER);

Hbase Java

Tuesday, 16 May 2023
12:29

```
public static void main(String[] args) throws IOException {

  Configuration con = HBaseConfiguration.create();
  HBaseAdmin admin = new HBaseAdmin(con);
```

```
   HTableDescriptor tableDescriptor = new
HTableDescriptor(TableName.valueOf("emp"));
   tableDescriptor.addFamily(new HColumnDescriptor("personal"));
   tableDescriptor.addFamily(new HColumnDescriptor("professional"));
   admin.createTable(tableDescriptor);

  // Getting all the list of tables using HBaseAdmin object
  HTableDescriptor[] tableDescriptor = admin.listTables();
  // printing all the table names.
  for (int i=0; i<tableDescriptor.length;i++ ){
      System.out.println(tableDescriptor[i].getNameAsString());
  }


/ / Instantiating columnDescriptor class
HColumnDescriptor columnDescriptor = new
HColumnDescriptor("contactDetails");
/ / Adding column family
admin.addColumn("employee", columnDescriptor);


HTable hTable = new HTable(config, "emp");
// Instantiating HTable class
Put p = new Put(Bytes.toBytes("row1"));
// adding values using add() method
// accepts column family name, qualifier/row name ,value
p.add(Bytes.toBytes("personal"), Bytes.toBytes("name"),Bytes.toBytes("raju"));
p.add(Bytes.toBytes("personal"), Bytes.toBytes("city"),Bytes.toBytes("hyderabad"));
p.add(Bytes.toBytes("professional"),Bytes.toBytes("designation"),
Bytes.toBytes("manager")); p.add(Bytes.toBytes("professional"),Bytes.toBytes("salary"),
Bytes.toBytes("50000"));
hTable.put(p); // Saving the put Instance to the HTable.

Getg=newGet(Bytes.toBytes("row1"));
// Instantiating Getclass
 Result result = table.get(g);
 / / Reading the data
/ / Reading values from Result class object
byte [ ] value = result.getValue(Bytes.toBytes("personal"),Bytes.toBytes("name"));
byte [ ] value1 = result.getValue(Bytes.toBytes("personal"),Bytes.toBytes("city"));
/ / Printing the values
String name = Bytes.toString(value);
String city = Bytes.toString(value1);
```

```
System.out.println("name: " +name +" city: " + city);


HTable table = new HTable(conf, "employee");
// Instantiating Delete class
Delete delete = new Delete(Bytes.toBytes("row1"));
delete.deleteColumn(Bytes.toBytes("personal"), Bytes.toBytes("name"));
delete.deleteFamily(Bytes.toBytes("professional"));
 table.delete(delete); // deleting the data
table.close();


 }
```

Revision Controle

Tuesday, 16 May 2023
21:30

Les 5V du pig DATA

Volume,variete,veracite,valeur,vitesse

Hadoop fs -put source destination
Hadoop fs -get destination source

Hadoop fs -rm source
Hadoop fs -cp source destination
Hadoop fs -mv

Hadoop fs -mkdir
Hadoop fs -mkdir -p

#ajouter du contenue a un fichier
Hadoop fs -appendToFile source destination
Hadoop fs -appendToFile - destination

++++++++++++++++++++++++++++++++++++++

FileSystem hdfs = FileSystem.get(Uri.create(),new Configuration())

Path folder = new Path()

Hdfs.mkdirs(folder);

```
Hdfs.delete( folder , true);

Copy du local vers hdfs

Hdfs.copyFromLocalFile(local,desti);

Hdfs.createNewFile( path );


#Ecrire dans hdsf

StringBuilder st = new StringBuilder();

For()
  st.append()

Byte[] bit = st.toString( ).getByte();

fsDataOutputStream flux= hdfs.create(path);
Flux.write()
Flux.end();



#lire

BufferReader bf = new BufferReader( new InputStreamReader(hdsf.open(path)) )

String st = null;

While( (str = bfr.readLine() ) )
```