**How to import any notepad/csv file into hive?**

Open(launch) hdp dashboard

-> Go to grid icon on top right corner

-> select Hive view->upload table

->file type-csv & select the delimeter-> select the file

-> give table name & columns

**What is hadoop?**

It is an open source software platform for **Distributed storage** & **Distributed Processing** of **Large datasets** on **computer clusters** built from commodity hardware

**Why hadoop?**

**Hadoop Ecosystem**

**HDFS-** Hadoop distributed file system

**YARN** -yet another resource negotiator

**MapReduce-** It helps to process data

**Pig-** High level scripting language which sits on MapReduce.

**Hive-** It is same as Pig which uses sql

**Ambari-** It lets you have a view into actual state of clusters &applications running on it

**Mesos- I**t is also a resource negotiators, It manages resources

**Spark**- It sits on top of YARN or Mesos.

**Tez-**

**HBase-** NoSQL Database

**Storm-** Used for processing streaming data.

**Oozie-** It is used to schedule job on clusters.

**Zookeeper**- It is used to coordinate the applications.keeps track of who is up & down etc

**Sqoop- I**t is a connector b/w Hadoop & legacy databases

**Flume-** Transporting weblogs

**Kafka**

**External Data Storage-** MySQL,MongoDB,Casandra etc

**Query Engines-** Hue, Apache drill, Apache phoenix,Presto,Apache zeppelin etc

1. **HDFS**

\*It is optimized for dealing with large files by breaking those files into blocks of default size of 128mb

\*Those blocks are stored across several commodity computers with such that if 1 node fails, the data can be extracted from other node.

\*It also allows processing of Data

**HDFS Architecture**

**\*Name Node-**It keeps track of what is all in the data node.

**\*Data Node-**It actually stores each block of each file.

**Reading Data->**Client node first asks Name node for the Data, then it gives the address of data node where data is, to the client node. Then the the client node reaches to that Data node to Retrieve data

**Writing Data->**

**What happens if Name node fails?**

\*To avoid it, Name node writes the Backup of metadata to tha local disk & NFS

**\*secondary namenode**

**\*HDFS federation**

**How to upload files in HDFS?**

**Using UI**

Dashboard->click HDFS-> click on grids(right top)->View Files->users->maria-> create a folder,open it -> click upload (file on top right)

**Using Terminal**

hadoop fs -ls

hadoop fs -mkdirml-100k

First we have to get files in local file-system before uploading it to Hadoop fs

wget [http://media.sundog-soft.com/hadoop/ml-100k/u.data](http://media.sundog-soft.com/hadoop/ml-100k/u.data" \t "http://localhost:4200/vt100Link)

This isnt working

**MapReduce Fundamental Concepts**

It distributes processing of data on clusters.

**Mappers** - Divides data up into partitions that are mapped (Transformed)

**Reducer** - It aggregates the data.

Mapperconverts raw source data into key-value pairs.

Shuffle& sort- shuffles all the values for each unique key & then sorts values.

Reduces- Then process the incoming list of values.

**MapReduce on a cluster level**

Here each partition is given to different nodes.

**Map Reduce Real Example using python.**

We have to find out how many times each rating occurred.

There are 2 python functions need to be written

1. Mapper function - It takes each row, breaks each column, returns (yields) the **rating** column & **count of rating**(i.e 1 as we take 1 row at a time)

def mapper\_get\_rating(self , \_ , line):

(userid,movieid,rating,timestamp) = line.split(‘\t’)

Yield rating , 1 #(here rating is key, 1 is value)

1. Reducer function- This function aggregates each rating & sums the values(which gives number of times each rating occurred)

Def reducer\_count\_rating(self,key,values): # (here values is list or tuple )

Yield key, sum(values)

**Program** RatingsBreakdown.py

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()

**Commands to do MapReduce in terminal**

**Running Locally (without using hadoop)**

python RatingsBreakdown.py u.data

**Running using hadoop**

**p**ython RatingsBreakdown.py -r hadoop --hadoop-streaming-jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar u.data

Here highlighted part is location of hadoop which must be mentioned in sandbox but in cloud platform it isnt necessary.

**Challenge:** Find the most popular movie

\*The movie which is rated most of times(not high) is most popular.

Now change the .py file to find which movie is rated most

\*Here also add 1 more reducer to sort the movies with ratings

**Hadoop uploads**

**Command:Hadoop fs -put or**

**hadoop fs -copyFromLocal u.data ml-100k/u.data**

**\***When we put file in hdfs first Name node is contacted,then client sends metadata to namenode.

\*The Namenode looks at the file at hdfs & make sure nobody can write to this file except client.And namnode sends the list ip address of data node to hadoop client where client can send the data to.

\*The client then devides the big file in blocks before it sends to datanode.

\*The blocks were sent 1 by 1 to 1 datanode,which are then replicated in other datanodes such that if 1 datanode fails we can still have the backup(The default replication factor is 3).

The datanodes sends heartbeat & Blockreport to namenode, like this the namenode knows which blockis present in which node.

If the data node stops sending blockreports & heartbeat then the namenode considers it as dead & creates new data nodes & starts replicating files.(It waits for 10 minutes before replicating & creating new node)

**hdfs fsck u.data** - tells number of blocks created for this.

**hdfs fsck u.data - files -locations -blocks**

**hadoop fs -D dfs.blocksize=1m -put u.data u.data2**

For user defined block size

In order to use ambari as admin

Open terminal-> become root-> write command

**ambari-admin-password-reset->** give password

->then login as admin in ambari

Writing mappers & reducers by hand takes long time hence Pig.

**Pig -** Piglatin is a scripting language that lets you use SQL-like syntax to write map & reduce steps, it sits on **mapreduce** & Tez.

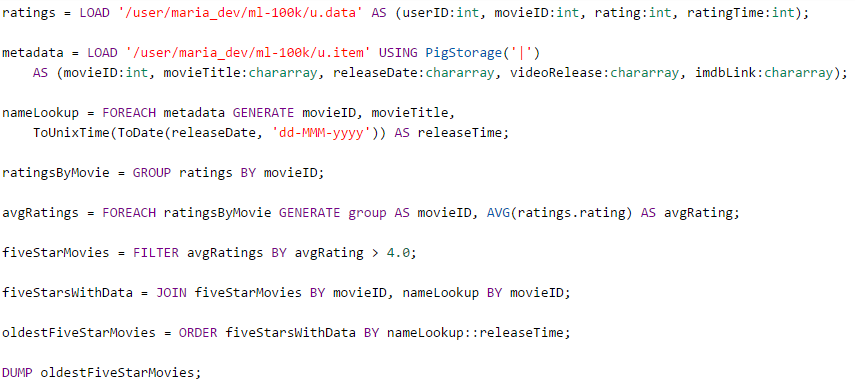
It is highly extensible with user defined functions.

**Running pig**

1. Using terminal
2. Or saving as a script & running it
3. Or running it from ambari

To write new script using ambari

Click on right top grids-> Pig view-> new script->give a name



ratings= LOAD '/user/maria\_dev/ml-100k/u.data' AS (userID:int,movieID:int, rating:int, ratingTime:int);

metadata= LOAD '/user/maria\_dev/ml-100k/u.item' USING PigStorage('|')

AS (movieID:int,movieTitle:chararray,releaseDate:chararray, videoRelease:chararray,imdbLink:chararray);

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

ratingsByMovie = GROUP ratings BY movieID;

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

fiveStarMovies= FILTER avgRatings BY avgRating>4.0;

fiveStarMoviesWithData = JOIN fiveStarMovies BY movieID, nameLookup by movieID;

oldestFiveStarMovies = ORDER fiveStarMoviesWithData BY nameLookup::releaseTime;

DUMP oldestFiveStarMovies;