

## Apunts CE\_5075 2.1

Iloc: [Institut d'Ensenyaments a Distància de les Illes  
Balears](#)  
Curs: Big data aplicat  
Llibre: Apunts CE\_5075 2.1

Imprès per: Carlos Sanchez Recio  
Data: dilluns, 28 d'octubre 2024, 16:04

# Taula de continguts

## 1. Introducció

## 2. Distribucions Hadoop

- 2.1. Cloudera Data Platform (CDP)
- 2.2. HortonWorks Data Platform (HDP)
- 2.3. Amazon Elastic MapReduce (EMR)
- 2.4. Microsoft Azure HDInsight
- 2.5. Google Dataproc
- 2.6. La nostra elecció per al curs

## 3. MapReduce

- 3.1. Un primer exemple
- 3.2. Un exemple amb codi

## 4. YARN

- 4.1. Interfície web de YARN

## 5. Pig

- 5.1. Maneres d'executar Pig
- 5.2. Pig Latin. Model de dades
- 5.3. Pig Latin. Sentències
- 5.4. Exemple amb un fitxer de log
- 5.5. Exemple de comptar paraules
- 5.6. Exemple amb dades estructurades
- 5.7. Més informació

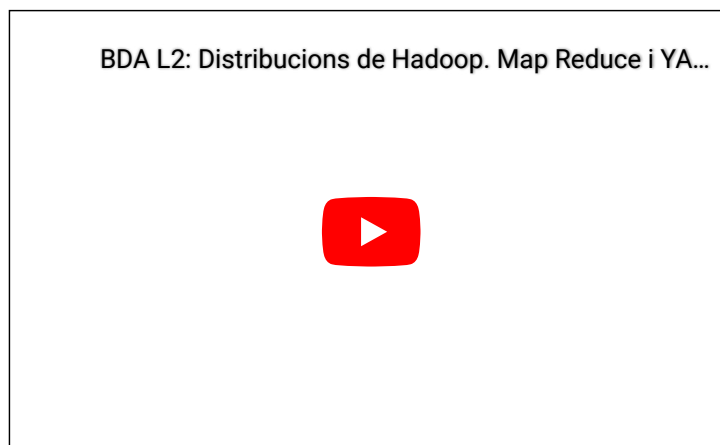
# 1. Introducció

En el lliurament anterior vàrem introduir Apache Hadoop i un dels components del seu nucli, el sistema d'arxius distribuït HDFS.

En aquest lliurament analitzarem els altres dos components del nucli de Hadoop. Són MapReduce, el framework per al processament distribuït i paral·lel de dades a gran escala, i YARN, el framework d'administració de recursos del clúster. També introduïrem Apache Pig, una eina que ens permet escriure aplicacions distribuïdes d'una manera molt més senzilla que directament amb MapReduce. Ho farem utilitzant un llenguatge anomenat Pig Latin. Els nostres scripts en Pig Latin són després traduïts a codi MapReduce de manera automàtica.

Abans de tot això, però, veurem què són les distribucions Hadoop. Atès que l'ecosistema Hadoop està format per una gran quantitat d'eines, instal·lar i gestionar una infraestructura Hadoop es fa realment complex. És per això que tenim disponibles diverses distribucions que incorporen, a més dels components principals del nucli, un conjunt d'eines de l'ecosistema Hadoop. Aquestes distribucions solen incloure una consola d'administració, facilitant enormement la tasca de l'administrador de la infraestructura. En el capítol 2 analitzarem les distribucions Hadoop més utilitzades: són les de Cloudera, de HortonWorks, així com les que han desenvolupat Amazon i Microsoft per a les seves plataformes de *cloud*. I també presentarem les màquines virtuals de dues distribucions Hadoop que utilitzarem al llarg del curs.

En el següent vídeo pots veure un resum del que tractarem en aquest lliurament.



**Vídeo:** Resum del Lliurament 2

## 2. Distribucions Hadoop

Ja vàrem veure al lliurament 1 que l'ecosistema Hadoop està format per un gran conjunt d'eines, la majoria projectes Apache de codi obert, però també d'altres propietàries. Instal·lar i gestionar totes les eines necessàries per a un projecte de big data acaba essent una tasca molt complexa.

És per això que existeixen diverses distribucions que integren un conjunt d'eines de l'ecosistema Hadoop, incloent sempre els components principals del nucli (HDFS, MapReduce i YARN). Aquestes distribucions solen incloure també una consola de gestió unificada, des d'on podem controlar cada una de les eines. D'aquesta manera es facilita enormement la tasca d'instal·lació i administració de la infraestructura de Big Data.

Algunes d'aquestes distribucions Hadoop varen néixer com a codi obert o bé tenien una versió de comunitat oberta. Desafortunadament en els darrers anys hem vist que les dues principals distribucions, Cloudera i HortonWorks, s'han fusionant i han decidit tancar les seves versions obertes. A més d'aquestes, els serveis de nuvol també ofereixen les seves distribucions pròpies. És el cas d'EMR (Elastic MapReduce) d'Amazon, Azure HDInsight de Microsoft i Dataproc de Google.

A continuació comentarem breument les principals característiques de les distribucions Hadoop més populars. Acabarem el capítol introduint les màquines virtuals de les dues distribucions Hadoop que utilitzarem al llarg del curs: CDH de Cloudera i HDP de HortonWorks.

## 2.1. Cloudera Data Platform (CDP)

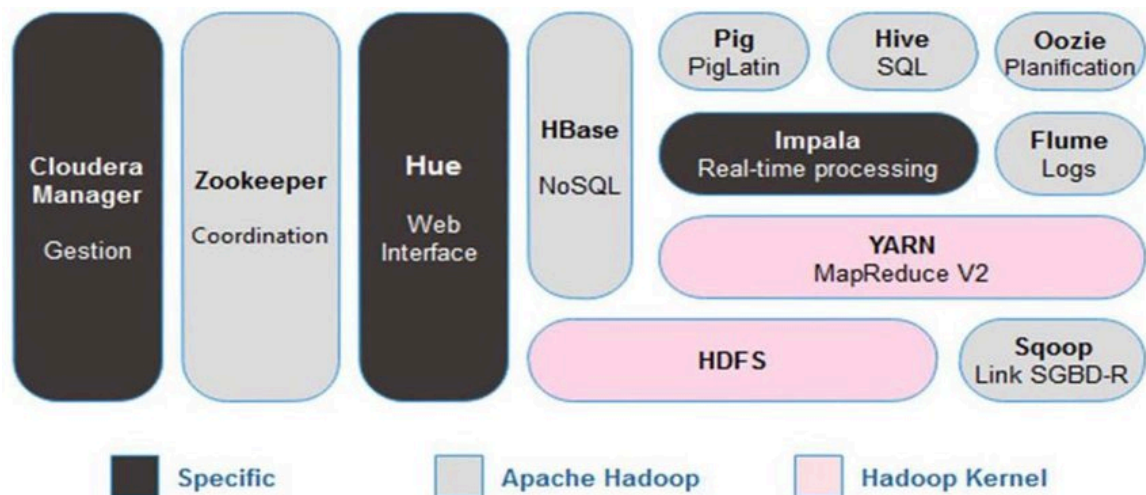
Cloudera Data Platform (CDP) és probablement la distribució de Hadoop més coneguda i utilitzada.

Cloudera afegeix un conjunt d'eines propietàries que van ser dissenyades per a optimitzar la gestió dels clústers i oferir millors experiències de cerques. Algun dels components desenvolupats per Cloudera són:

- Impala: és un motor basat en SQL, per a temps real i paral·lel·litzat, que realitza cerques de dades en el sistema d'arxius HDFS.
- Cloudera Manager: és la consola que permet gestionar i desplegar els components en el clúster Hadoop.
- Hue (Hadoop User Experience): és una consola que permet a l'usuari interactuar amb les dades i executar scripts per als diferents components de Hadoop continguts en el clúster.

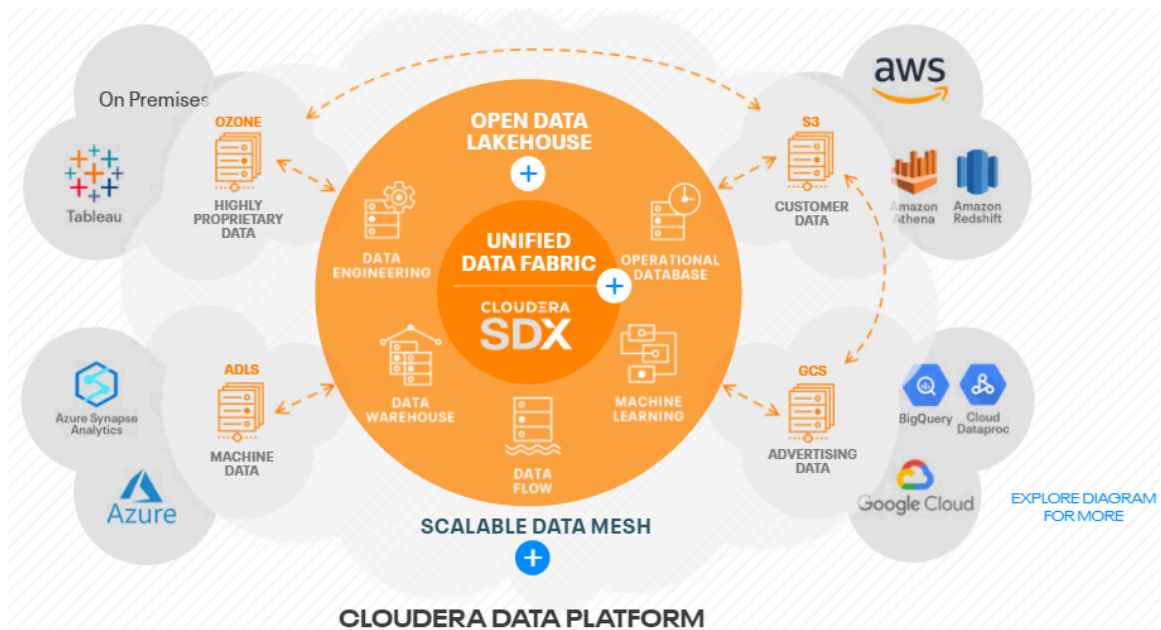
La següent imatge mostra els components que formen la distribució Hadoop de Cloudera:

- Components que són part del nucli de Hadoop: YARN, MapReduce i HDFS.
- Components que són part de l'ecosistema Hadoop d'Apache: Zookeeper, Hbase, Hive, Pig, Flume, Oozie i Sqoop.
- Components desenvolupats per Cloudera: Cloudera Manager, Impala i Hue.



**Imatge:** Components de la distribució Hadoop de Cloudera. Font: Cloudera

Un dels punts que fan que CDP sigui tan utilitzat és que està integrat completament amb les diverses plataformes del nícul. Això facilita molt l'escalabilitat.



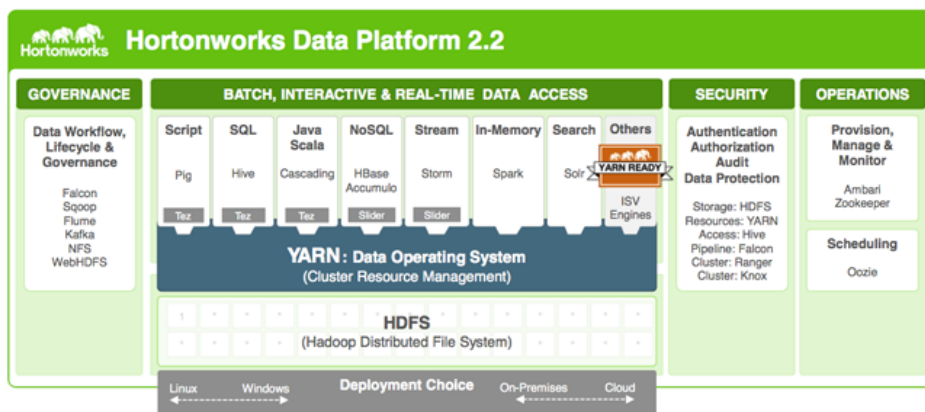
**Imatge:** Integració de CDP i plataformes de nivol. Font: Cloudera

Cloudera oferia fins fa poc una distribució 100% de codi obert, anomenada CDH (Cloudera Distributed Hadoop). Però arran de la fusió amb HortonWorks, Cloudera ha canviat dràsticament la seva política. Des de gener de 2021, tot el nou software de Cloudera (i de HortonWorks) només està disponible per a usuaris amb una subscripció de pagament. Si us interessen els detalls, els trobareu [aquí](#).

## 2.2. HortonWorks Data Platform (HDP)

HortonWorks oferia una distribució Hadoop completament de codi obert: HortonWorks Data Platform (HDP), que durant alguns anys va competir amb CDP, la plataforma de dades de Cloudera. Com a eina web de gestió de la infraestructura, HDP utilitza Apache Ambari, un projecte de codi obert desenvolupat originalment per HortonWorks i que va passar a l'Apache Software Foundation, equivalent a Cloudera Manager.

En la següent imatge es poden veure els principals components d'HDP:



**Imatge:** Components d'HDP. Font: Cloudera.

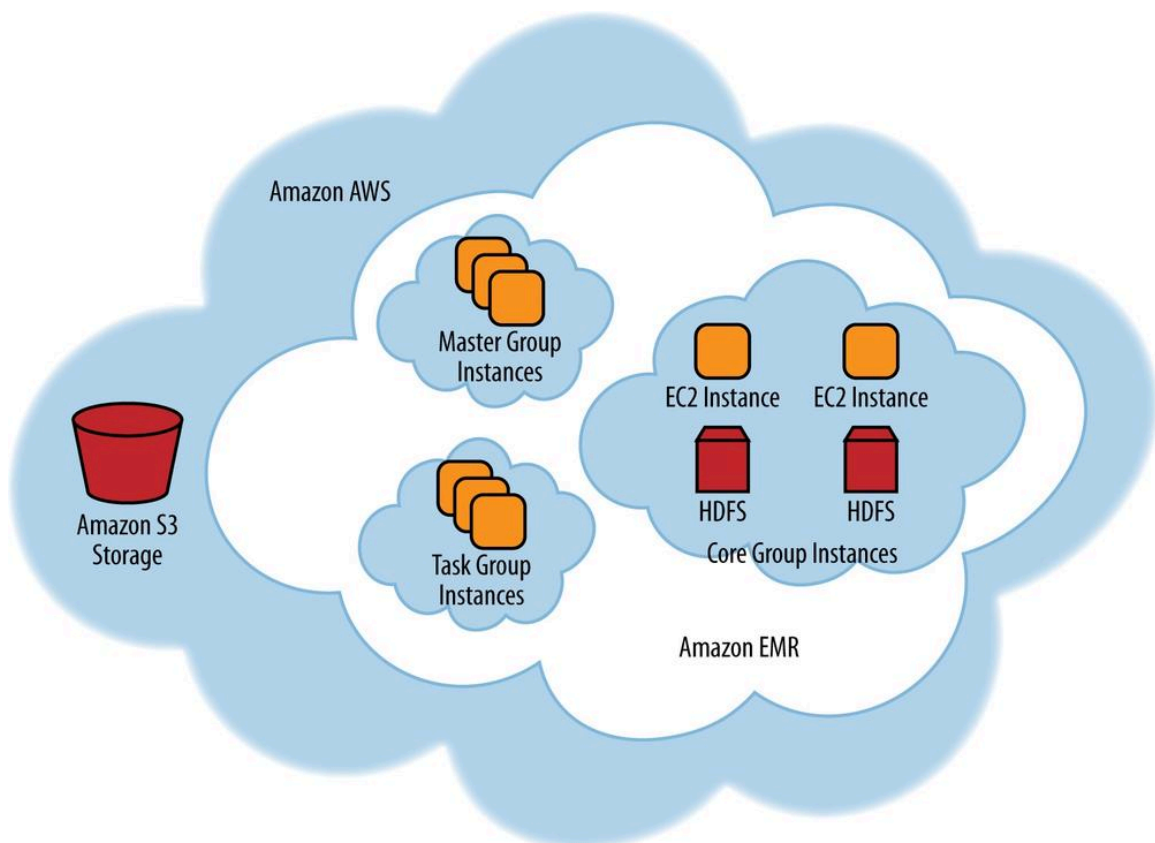
No obstant això, com ja hem comentat, arran de la fusió de HortonWorks amb Cloudera (més aviat, Cloudera ha absorbit HortonWorks), tot i que totes les eines que utilitzen són de codi obert, és necessari una subscripció (de pagament) a Cloudera per poder accedir al nou software de HortonWorks. De fet, HDP s'ha deixat de comercialitzar, recomanant-se la migració a Cloudera Data Platform (CDP).

## 2.3. Amazon Elastic MapReduce (EMR)

**Elastic MapReduce (EMR)** és la plataforma d'Amazon que permet analitzar i processar grans quantitats de dades mitjançant la distribució del treball de còmput a través d'un clúster de servidors virtuals. El clúster és gestionat sota la plataforma Hadoop, on un node és designat com a mestre per a controlar la distribució de tasques.

El servei Amazon EMR ha realitzat millores i personalitzacions en els components de Hadoop per a treballar de forma integrada amb Amazon Web Services (AWS). Els clústers Hadoop que s'executen en Amazon EMR usen les següents instàncies: Amazon Elastic Compute Cloud (Amazon EC2) que consisteix en servidors virtuals Linux per als nodes mestre i esclau, Amazon Simple Storage Service (Amazon S3) per al consum i emmagatzematge massiu de dades i Cloud-Watch per a supervisar el rendiment del clúster i generació d'alertes.

La següent imatge mostra l'arquitectura general d'Amazon EMR:



**Imatge:** Arquitectura d'Amazon EMR. Font: oreilly.com

Amazon EMR ofereix un emmagatzematge molt flexible, que suporta diverses solucions:

- Amazon S3
- HDFS
- Amazon Dynamo DB, una base de dades NoSQL
- Altres magatzems de dades d'AWS: Amazon Redshift, Amazon Glazier i Amazon Relational Database Service

També té suport per a diverses solucions de big data i aprenentatge automàtic. Algunes de les més utilitzades són:

- Apache Spark
- Apache Hive
- Apache HBase
- Apache Flink
- TensorFlow

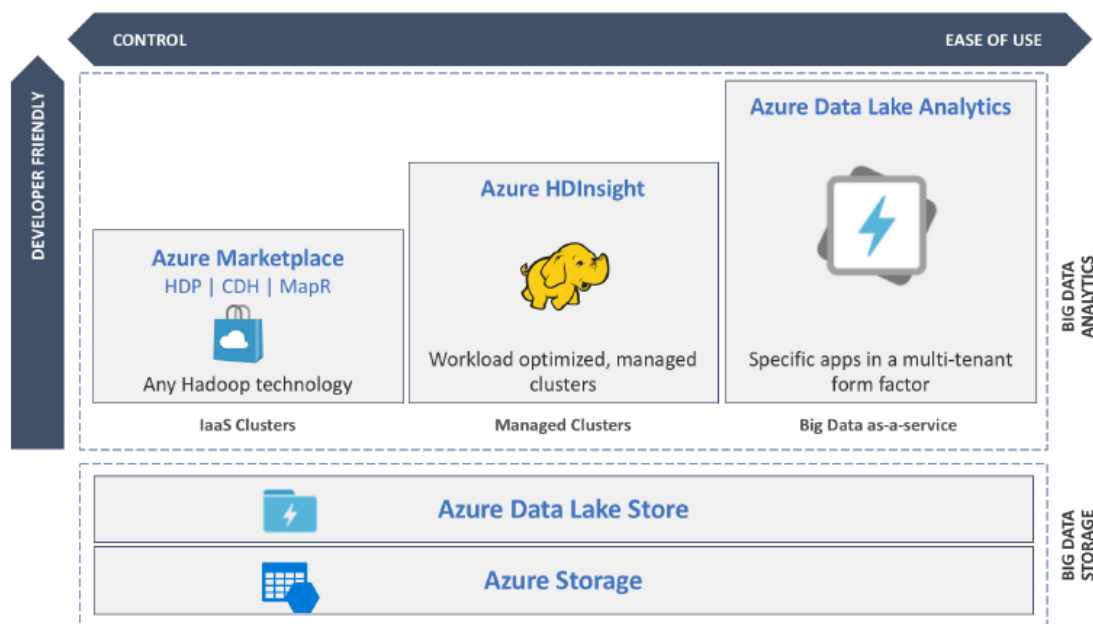




## 2.4. Microsoft Azure HDInsight

La plataforma Azure implementa una solució per a clústers Hadoop denominada **HDInsight**. Originalment HDInsight emprava HortonWorks Data Platform (HDP), però arran de la fusió de HortonWorks i Cloudera, Microsoft va desenvolupar la seva pròpia solució, basada en components lliures de d'Apache. Aquesta solució permet implementar i gestionar clústers Hadoop en el nívol amb la finalitat de proporcionar un sistema òptim per a processar, analitzar i generar informes garantint una alta confiabilitat i disponibilitat. A més, proporciona la possibilitat de l'aprovisionament automàtic de clústers de Hadoop.

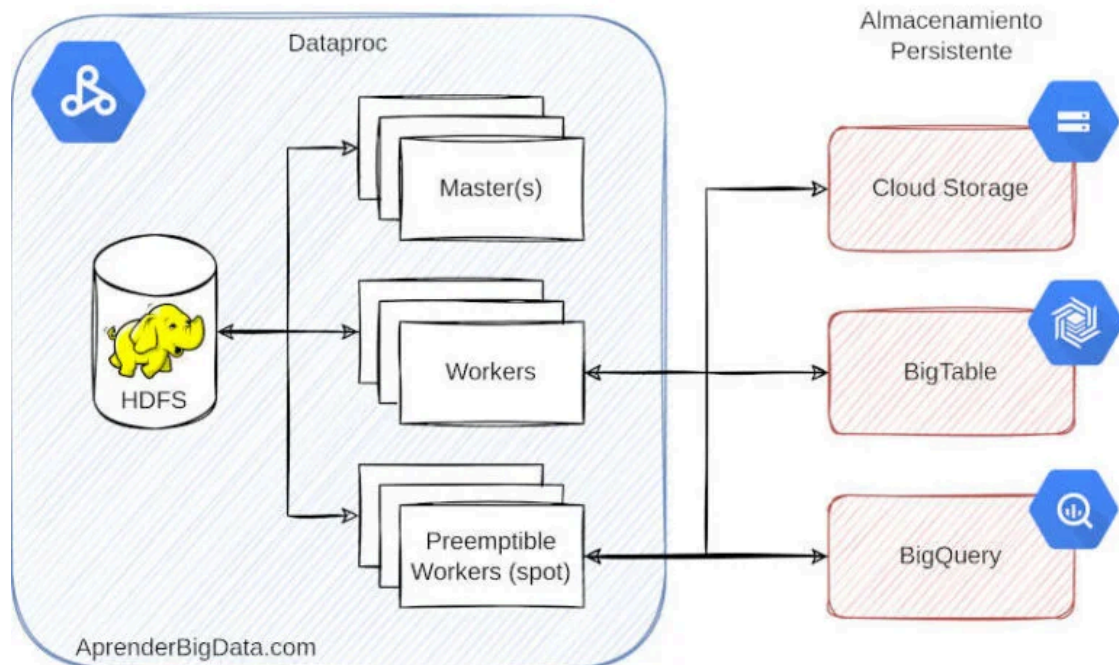
La següent imatge mostra els elements principals de l'arquitectura de big data de Microsoft Azure, incloent Azure HDInsight:



**Imatge:** Arquitectura de big data de Microsoft Azure

## 2.5. Google Dataproc

Igual que Amazon i Microsoft, Google també té el seu propi servei per oferir clústers Hadoop en el seu nígil, Google Cloud. Es tracta de [Dataproc](#), un servei totalment gestionat i molt escalable per a executar Apache Hadoop i més de 30 eines i *frameworks* de programari lliure relacionats. Dataproc està basat en eines de codi obert i és flexible, segur i eficient.



**Imatge:** Components de Dataproc. Font: [aprenderbigdata.com](#)

## 2.6. La nostra elecció per al curs

La fusió de Cloudera i Hortonworks ha tngut un fort impacte en el món del big data, ja que eren les dues principals empreses del sector. Alguns experts consideren que ha estat positiu perquè reforça el sector. En canvi, d'altres plantegen que és un començament de la fi de Hadoop com a infraestructura predominant del software distribuït i dels sistemes de big data. Per exemple, aquest [article](#) ofereix una visió interessant al respecte.

Un dels efectes importants que ja hem comentat és la nova política de Cloudera/HortonWorks respecte al codi obert. Això realment va en contra de tota la filosofia que ha guiat la comunitat de big data en general i de Hadoop en particular. Veurem en el futur pròxim quin impacte té i si apareixen noves distribucions obertes que cobreixin el buit que han deixat Cloudera i HortonWorks.

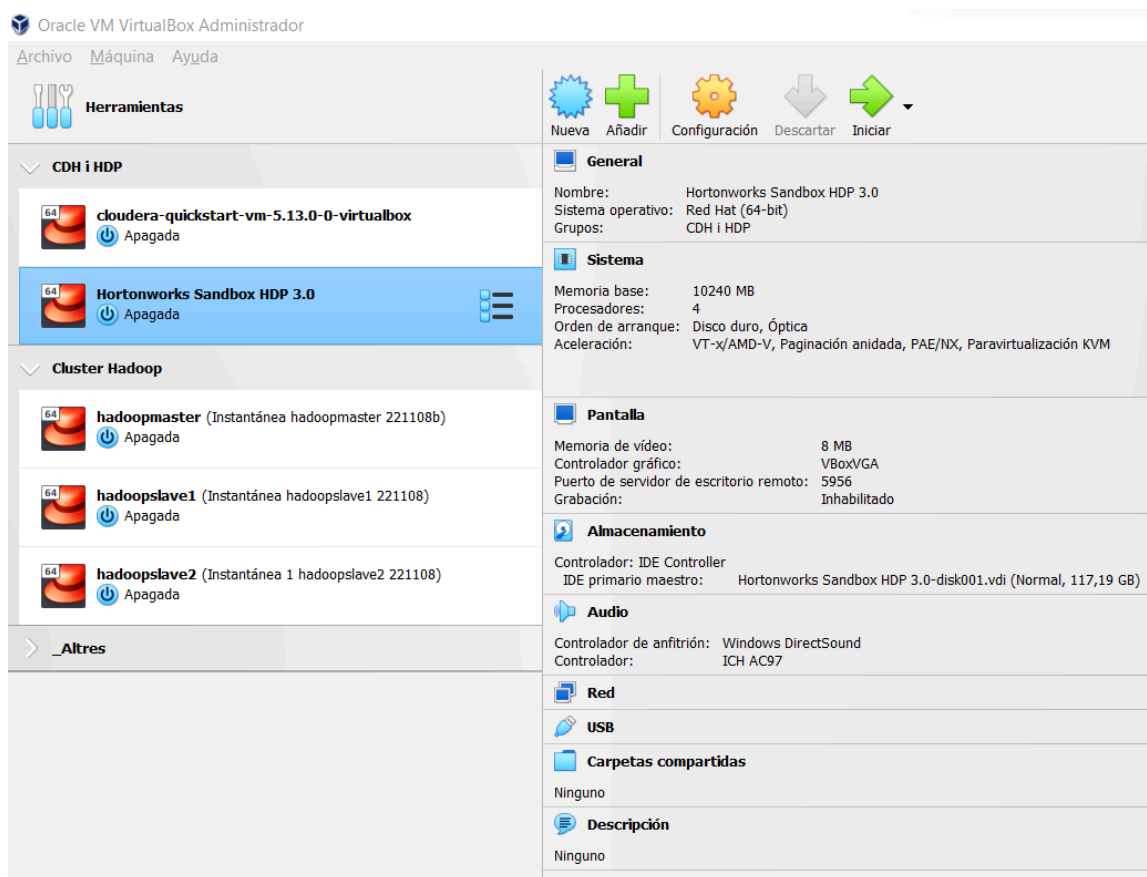
Mentrestant, en aquest curs emprarem les darreres versions obertes de CDH (Cloudera Distributed Hadoop), principalment, i de HDP (HortonWorks Data Platform). Són més antigues que les que s'estan comercialitzant, però tenen tots els components que ens interessin en aquest curs.

Una vegada que ja hem après com configurar un clúster Hadoop en el lliurament anterior, per facilitar la feina, a partir d'ara treballarem normalment amb un clúster de node únic. I ho farem també amb màquines virtuals sobre Oracle VirtualBox.

Podem descarregar una màquina virtual per a Oracle VirtualBox ja configurada amb CDH des de [https://downloads.cloudera.com/demo\\_vm/virtualbox/cloudera-quickstart-vm-5.13.0-0-virtualbox.zip](https://downloads.cloudera.com/demo_vm/virtualbox/cloudera-quickstart-vm-5.13.0-0-virtualbox.zip). La darrera versió disponible és la 5.13.

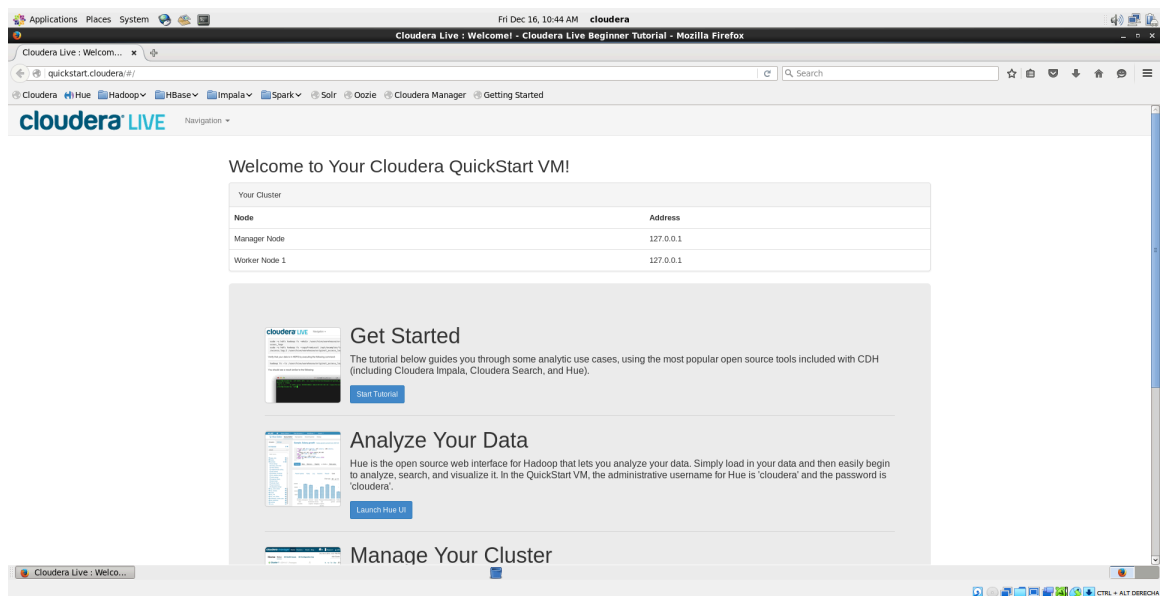
I també podem descarregar el HortonWorks HDP Sandbox, una màquina virtual per a Oracle VirtualBox ja configurada amb HDP, des de [https://archive.cloudera.com/hwx-sandbox/hdp/hdp-3.0.1/HDP\\_3.0.1\\_virtualbox\\_181205.ova](https://archive.cloudera.com/hwx-sandbox/hdp/hdp-3.0.1/HDP_3.0.1_virtualbox_181205.ova). La darrera versió disponible és la 3.0.1.

En ambdós casos, només hem d'importar la màquina virtual en Oracle VirtualBox, amb el botó *Importar*, o simplement fent doble clic sobre l'arxiu .ova. Podem, si volem, modificar alguns dels paràmetres de configuració (per exemple, canviar la mida del disc, de la RAM o del número de cores). Però en principi, no és necessari. En la següent imatge podem veure les dues màquines virtuals importades en VirtualBox, amb els detalls de la màquina virtual d'HDP.



**Imatge: Màquines virtuals de CDH i HDP en Oracle Virtual Box**

En la imatge següent es mostra la màquina virtual ja en marxa amb la distribució CDH:



**Imatge: Màquina virtual amb CDH**

### 3. MapReduce



**MapReduce** és un model de programació, i la seva implementació associada, per a processar i generar grans conjunts de dades sobre un clúster d'ordinadors, amb un algorisme paral·lel i distribuït.

Font: Wikipedia

MapReduce es compon de dues funcions (o mètodes), *map* i *reduce*, d'aquí el seu nom. La funció *map* du a terme un filtrat i ordenació de les dades, mentre que la funció *reduce*, a partir del resultat de *map*, fa una operació d'agregació (per exemple, comptar, sumar o calcular la mitjana).

Les primeres implementacions del model MapReduce les va fer Google, com a tecnologia propietària, per al càlcul del PageRank (el sistema de rànkings emprat pel cercador de Google). Posteriorment, va ser desenvolupat en l'àmbit del projecte Nutch i, dins Yahoo, va passar a ser una part principal del projecte Hadoop. Hi ha diverses implementacions de MapReduce, però la més popular és la de Hadoop.

Un sistema o infraestructura MapReduce, com el que proporciona Hadoop, permet l'execució d'aplicacions de manera distribuïda, executant diverses tasques en paral·lel, gestionant les comunicacions de transferència de dades entre les diverses parts del sistema i proporcionant redundància i tolerància a errors.

En tot cas, no qualsevol problema pot ser resolt de manera eficient mitjançant MapReduce. En general, MapReduce està especialment indicat per abordar problemes amb grans conjunts de dades, arribant a petabytes. Per aquesta raó, MapReduce es fa servir sobre un sistema d'arxius distribuït, com HDFS en el cas de Hadoop.

A continuació veurem dos exemples del funcionament de l'algorisme MapReduce. En el primer, ho farem de manera teòrica, mentre que en el segon veurem el codi Python per implementar les funcions de *map* i de *reduce*.

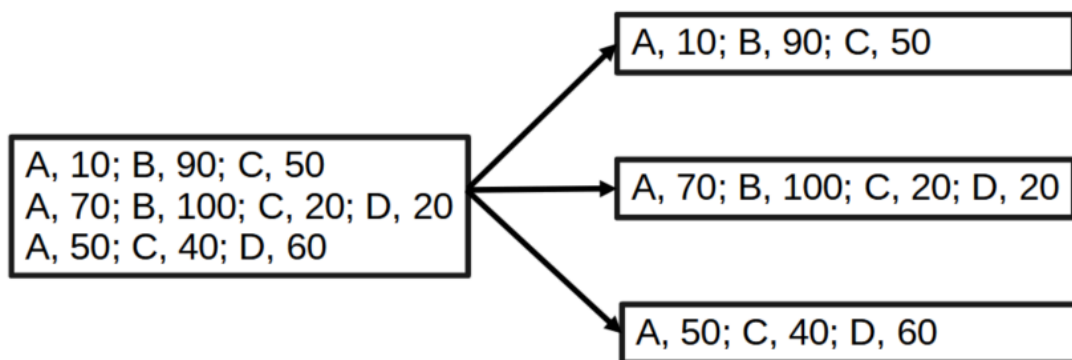
### 3.1. Un primer exemple

Per entendre com funciona MapReduce, veurem un petit exemple.

Anam a suposar que tenim un dataset en HDFS amb les dades de votacions d'unes eleccions. Per simplificar, suposem que només hi ha 4 partits: A, B, C i D. En cada fila del dataset, tenim el recompte d'una taula electoral. Per entendre millor el funcionament de MapReduce i poder veure-ho gràficament, anam a concentrar-nos només en les tres primeres files del dataset:

```
A, 10; B, 90; C, 50
A, 70; B, 100; C, 20; D, 20
A, 50; C, 40; D, 60
```

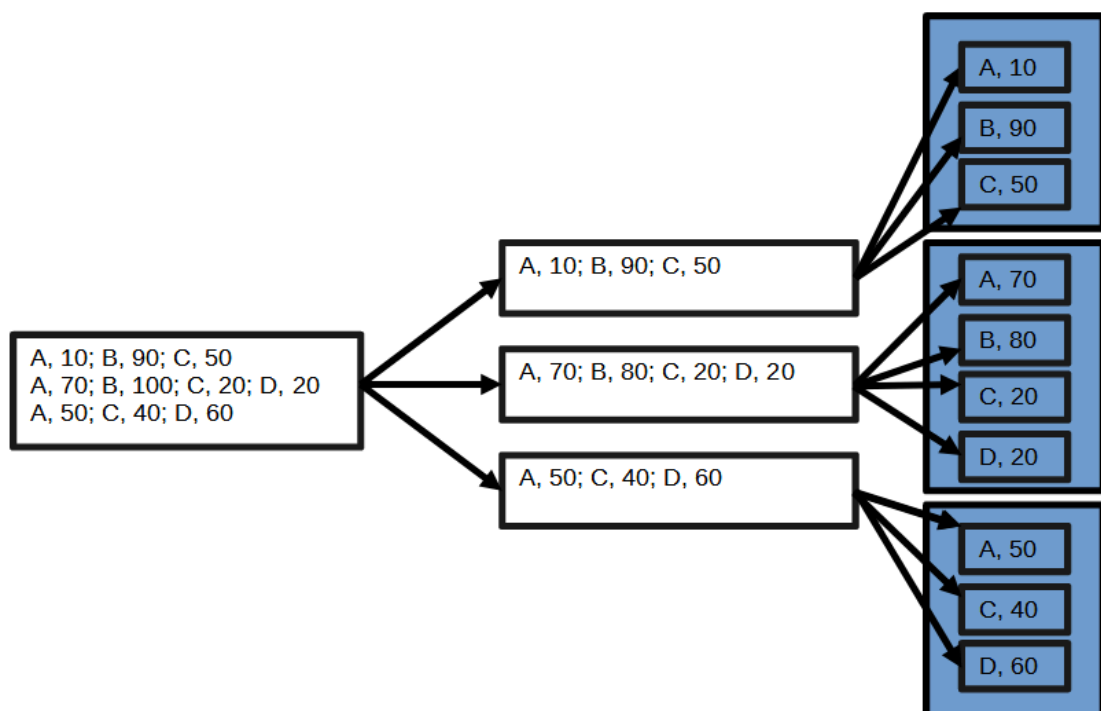
En una fase prèvia, anomenada separació (*splitting*), dividim les dades d'entrada en fragments més petits. La mida d'aquestes divisions està establerta en un paràmetre de Hadoop i pot o no coincidir amb la mida d'un bloc HDFS. En el nostre cas, per simplificar, anam a suposar que cada divisió està formada per una línia:



**Imatge:** Fase de separació (*splitting*)

De cada un d'aquests fragments se n'encarregarà un node (*worker*) mapejador diferent.

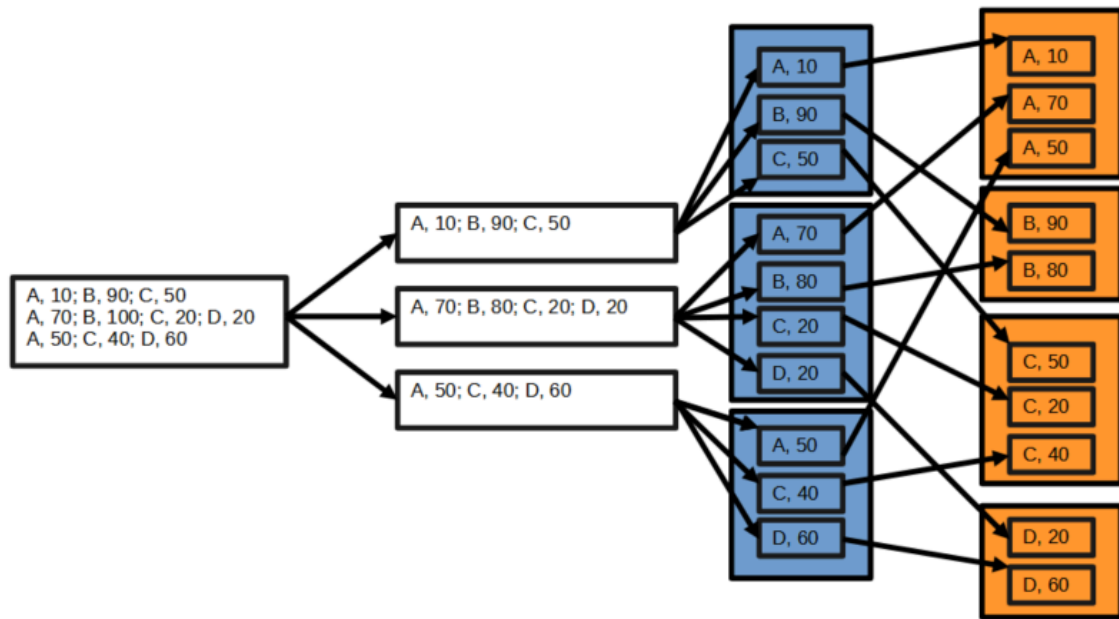
Arribam així a la fase de mapeig (*map*), en la qual els nodes mapejadors, en paral·lel, extreuen una llista de les parelles clau-valor dels seus respectius fragments:



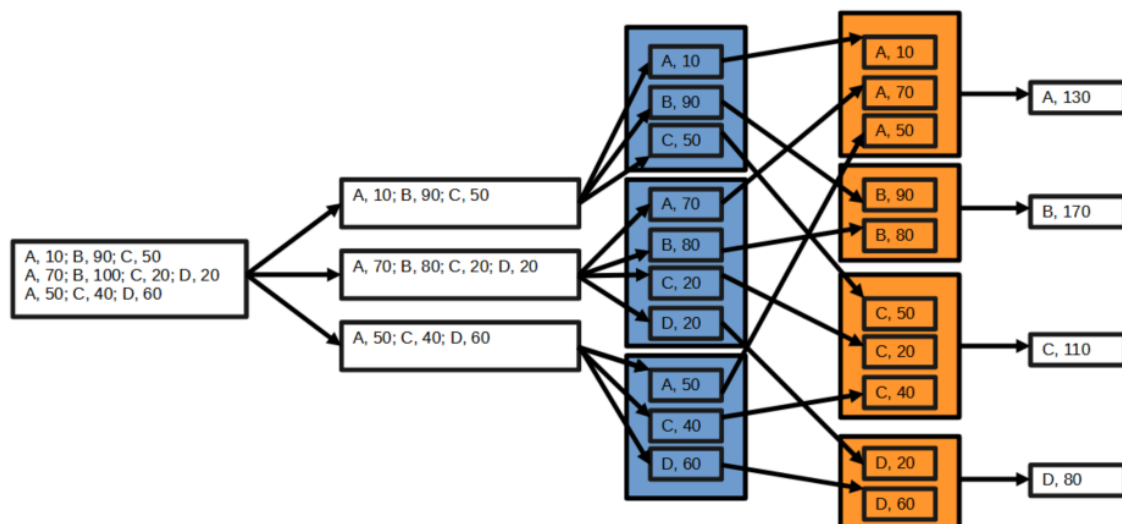
**Imatge:** Fase de mapeig (map)

Cada node mapejador aplica la funció *map* sobre les seves dades locals i escriu la sortida en un emmagatzematge temporal. Notau que aquí la funció *map* és molt senzilla perquè les dades originals eren també molt simples i només tenien un atribut amb el número de vots. Però normalment les dades són més complexes i aquí és necessari fer una operació també més complexa.

A continuació passam a una fase intermèdia entre el mapeig i la reducció, anomenada de mescla (*shuffle* en anglès, *barajado* en castellà). En aquesta fase, els nodes redistribueixen les dades basant-se en les seves claus, de manera que totes les dades corresponents a una clau acabin localitzades en el mateix node. D'aquesta manera, en la fase següent, cada node reductor s'encarregarà només de processar les dades d'una única clau.

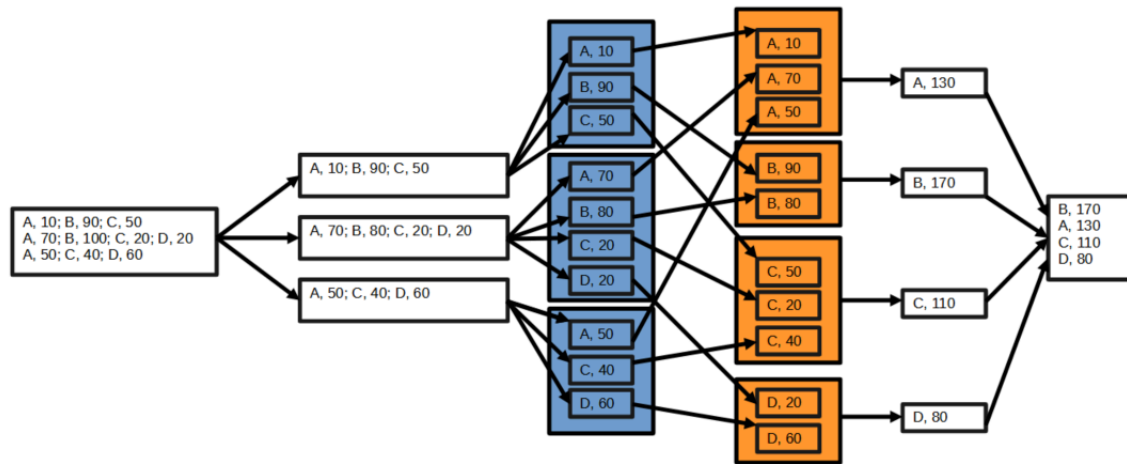
**Imatge:** Fase de mescla (shuffle)

Arribam a la fase de reducció (*reduce*), on els nodes (*workers*) reductors processen en paral·lel el seu grup de dades, cadascun amb una clau diferent. En aquest cas, la funció *reduce* que aplica cada node reductor consisteix en sumar els valors de totes les seves parelles clau-valor (totes tenen la mateixa clau). Normalment sol utilitzar-se aquí alguna (o algunes) operacions d'agregació (suma, mínim, màxim, mitjana, recompte, ...).

**Imatge:** Fase de reducció (reduce)

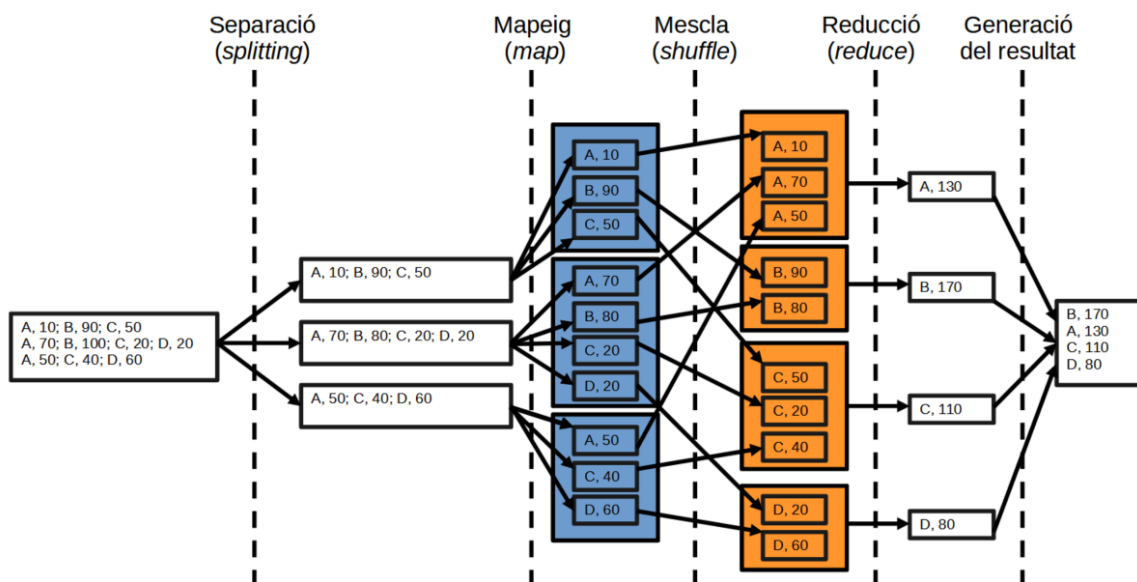


Per acabar, els resultats finals de cada node reductor s'agrupen, normalment s'ordenen, i poden emmagatzemar-se en memòria o, el més habitual, en un arxiu HDFS nou.



Imatge: Generació del resultat

La següent imatge ens mostra tot el procés MapReduce complet, amb el nom de cada fase:



Imatge: Procés MapReduce complet

Hem pogut veure que MapReduce permet el processament distribuït de les operacions de mapeig i reducció. Els nodes mapejadors actuen en paral·lel, aplicant la funció *map*. Cada operació de mapeig és independent de les demés. De la mateixa manera, els nodes reductors apliquen la funció *reduce* també en paral·lel.

A primera vista, pot semblar molt complicat i fins i tot més ineficient que un procés seqüencial. Però la realitat és que MapReduce pot aplicar-se a conjunts de dades molt més grans. A més, tota aquesta arquitectura és tolerant a fallades: si un node mapejador o un node reductor falla, el treball pot reprogramar-se, de manera que se n'encarreguin altres nodes.

Per altra banda, tot i que ho hem descrit com un procés seqüencial, una fase darrera d'una altra, realment molt sovint aquestes fases es poden intercalar, sempre que això no afecti al resultat final. Així doncs, mentre els nodes mapejadors estan fent un treball, els nodes reductors no estan aturats esperant que aquells acabin, sinó que poden estar processant un altre treball. Per exemple, aquí hem vist només 3 línies de l'arxiu d'entrada, que s'han dividit per

ser processades per 3 nodes mapejadors. Però, mentre els nodes reductors estan processant les sortides corresponents a aquestes 3 primeres línies, els mapejadors ja podrien estar processant les següents 3. I així successivament.

De totes aquestes fases, el programador s'ha d'encarregar principalment de programar les funcions de *map* i de *reduce*, ja que d'altres, com la mescla, ja estan integrades en el codi principal de MapReduce de Hadoop.

## 3.2. Un exemple amb codi

En aquest apartat anam a veure un exemple, ara amb codi, de com comptar les ocurrencies de les paraules que apareixen al Quijote, utilitzant MapReduce. I ho farem sobre la màquina virtual de Cloudera Quickstart, de la qual n'hem xerrat al capítol 2.

Al directori arrel de l'usuari cloudera (/home/cloudera), crearem un directori quijote, on descarregarem la versió del Quijote en format text pla:

```
wget t.ly/7pwK
```

Li canviarem el nom al fitxer:

```
mv 7pwK quijote.txt
```

També el podem descarregar des de <https://github.com/tnavarrete-iedib/bigdata/blob/main/quijote.txt>

Abans de començar, anam a mirar quantes paraules té:

```
wc -w quijote.txt
```

El resultat és que conté 187.018 paraules. El que volem en l'exemple és comptar quantes vegades apareix cada paraula en el llibre.

Ara anam a crear un directori HDFS i copiar-hi el fitxer quijote.txt, tal i com vàrem fer en el primer lliurament:

```
hdfs dfs -mkdir quijote  
hdfs dfs -put quijote.txt quijote
```

Podem comprovar que el fitxer està bé:

```
hdfs dfs -ls quijote
```

Ara hem d'implementar les funcions *map* i *reduce*. De les fases de *splitting* i *shuffle*, no ens hem d'encarregar, perquè ja ho fa la pròpia plataforma. Així que ara hauríem d'escriure una classe Java (recordem que el *framework* Hadoop està escrit en Java) per al *mapper* i una altra per al *reducer*. La classe *mapper* ha d'implementar la interfície *org.apache.hadoop.mapred.Mapper*, mentre que la classe *reducer*, ha d'implementar la interfície *org.apache.hadoop.mapred.Reducer*.

Però com que en aquest curs assumim que no teniu coneixements de Java, anam a emprar **HadoopStreaming**, una utilitat disponible en qualsevol distribució Hadoop i que permet executar treballs MapReduce, on el *mapper* i el *reducer* poden ser, enlloc de classes Java, qualsevol executable o script de Shell. En el nostre cas, el que farem és escriure un script de Python per al *mapper* i un altre script de Python per al *reducer*.

El més habitual és emprar Java per escriure aquestes funcions *map* i *reduce*, perquè així tenim accés a totes les capacitats que ofereix l'API Java del *framework*, però aquesta opció ens permetrà emprar Python per poder entendre millor el codi.



AMPLIACIÓ

Si voleu veure els detalls sobre com implementar directament la solució a comptar les paraules d'un text utilitzant classes Java, ho podeu trobar a la documentació de Hadoop:

<https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html>

En primer lloc, en la fase de *map*, s'ha de separar el text en paraules i assignar-li a cada una d'elles una ocurrència. És a dir, ha de generar una llista de parelles *<paraula, ocurrències>*, on encara tendrem un 1 a totes les ocurrències (perquè encara no hem mirat quines estan repetides). A continuació podeu veure el fitxer *mapper.py*, que té en compte que les paraules no estan només delimitades per espais en blanc, sinó que també per altres signes de puntuació. A més, passa totes les paraules a minúscules, perquè no consideri que "La" i "la" són paraules diferents.

```
#!/usr/bin/env python

# import sys because we need to read and write data to STDIN and STDOUT
import sys
import re
sys.path.append('.')

for line in sys.stdin:
    line = line.strip()
    for word in re.compile('\w+').findall(line):
        print '%s\t%s' % (word.lower(), 1)
```

Podem comprovar què fa aquest codi (encara sense executar-lo en Hadoop) així:

```
cat quijote.txt | python mapper.py
```

```
don      1
quijote  1
de       1
la       1
mancha   1
miguel   1
de       1
cervantes 1
saavedra 1
primera  1
parte    1
capi     1
tulo     1
1        1
que      1
trata    1
de       1
la       1
condicio 1
n        1
y        1
ejercicio 1
del      1
```

**Imatge:** Execució del codi del fitxer *mapper.py* (fragment)

Per altra banda, en la fase de *reduce*, per a cada línia que ha generat el *mapper*, ha de comptar quantes vegades apareix la paraula. Obtindrem així una altra llista de parelles *<paraula, ocurrències>*, on ara ja sí que tendrem les ocurrències reals de la paraula. A continuació podeu veure el fitxer *reducer.py*:

```
#!/usr/bin/env python

import sys
sys.path.append('.')

last_key = None
total = 0

for input_line in sys.stdin:
    input_line = input_line.strip()
    this_key, value = input_line.split("\t", 1)
    value = int(value)

    if last_key == this_key:
        total += value
    else:
        if last_key:
            print("%s\t%d" % (last_key, total))
        total = value
        last_key = this_key

if last_key == this_key:
    print( "%s\t%d" % (last_key, total) )
```

Podem veure què fa això, sense executar-ho encara a Hadoop:

```
cat quijote.txt | python mapper.py | sort -k1,1 | python reducer.py
```

```
a      7968
aa      2
aba      7
abad      1
abadejo 2
abades  1
abadesa 1
abajarse 2
abajen  1
abajo   22
abala   1
abalanza 1
aban     8
abandonarme 1
abatanar 1
abece    3
abejas   1
abencerraje 1
abierta  4
abiertas 1
abierto  7
abiertos 5
abindarra 3
```

**Imatge:** Execució del codi del fitxer reducer.py (fragment)

Ara anam a donar permisos d'execució sobre els dos fitxers:

```
chmod 755 *.py
```

I ara ja podem fer l'execució distribuïda en Hadoop, fent servir l'eina HadoopStreaming (tot en una línia):

```
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar
-input quijote/quijote.txt
-output quijote2
-mapper /home/cloudera/quijote/mapper.py
-reducer /home/cloudera/quijote/reducer.py
```

```
[cloudera@quickstart quijote]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar -input quijote/quijote.txt -output quijote2 -mapper /home/cloudera/quijote/mapper.py -reducer /home/cloudera/quijote/reducer.py
packageJobJar: [] [/usr/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.13.0.jar] /tmp/streamjob4789013489144168326.jar tmpDir=null
22/11/29 03:23:42 INFO client.RMPProxy: Connecting to ResourceManager at /0.0.0.0:8032
22/11/29 03:23:42 INFO client.RMPProxy: Connecting to ResourceManager at /0.0.0.0:8032
22/11/29 03:23:44 INFO mapred.FileInputFormat: Total input paths to process : 1
22/11/29 03:23:44 INFO mapreduce.JobSubmitter: number of splits:2
22/11/29 03:23:44 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1669717738346_0001
22/11/29 03:23:45 INFO impl.YarnClientImpl: Submitted application application_1669717738346_0001
22/11/29 03:23:45 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application/1669717738346_0001/
22/11/29 03:23:45 INFO mapreduce.Job: Running job: job_1669717738346_0001
22/11/29 03:23:52 INFO mapreduce.Job: Job job_1669717738346_0001 running in uber mode : false
22/11/29 03:23:52 INFO mapreduce.Job: map 0% reduce 0%
22/11/29 03:23:59 INFO mapreduce.Job: map 50% reduce 0%
22/11/29 03:24:01 INFO mapreduce.Job: map 100% reduce 0%
22/11/29 03:24:06 INFO mapreduce.Job: map 100% reduce 100%
22/11/29 03:24:07 INFO mapreduce.Job: Job job_1669717738346_0001 completed successfully
22/11/29 03:24:07 INFO mapreduce.Job: Counters: 50
    File System Counters
        FILE: Number of bytes read=1808136
        FILE: Number of bytes written=4053500
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=1064589
        HDFS: Number of bytes written=139184
        HDFS: Number of read operations=9
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
    Job Counters
```

**Imatge:** Execució distribuïda del nostre programa MapReduce (fragment)

Això crea un directori *quijote2* en HDFS i allà hi copia el resultat del procés.

```
[cloudera@quickstart quijote]$ hdfs dfs -ls quijote2
Found 2 items
-rw-r--r-- 1 cloudera cloudera 0 2022-11-29 03:24 quijote2/_SUCCESS
-rw-r--r-- 1 cloudera cloudera 139184 2022-11-29 03:24 quijote2/part-000000
```

**Imatge:** Resultat de l'execució

En aquest cas que el fitxer és petit, ha generat un únic fitxer amb dades (*quijote2/part-000000*). Si el fitxer fos més gros, s'hauria fragmentat en més fitxers (recordau que, per defecte, la mida d'un fragment en HDFS és de 128 Mb). Per veure el contingut del fitxer:

```
hdfs dfs -cat quijote2/part-000000
```

Així, podem comprovar que "sancho" apareix 691 vegades, mentre que "dulcinea" ho fa en 90 ocasions.

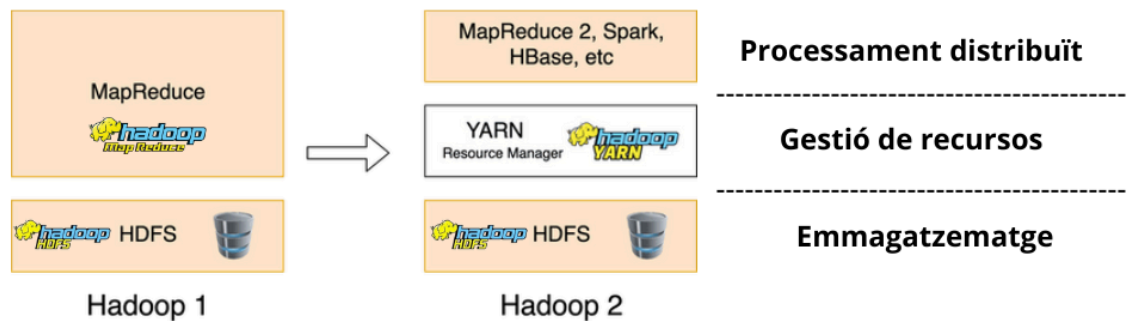
Com en anteriors exemples, hem de tenir present que aquí el fitxer d'entrada és molt petit (aprox. 1 Mb). Però pensau que això es faria igual amb un fitxer (o més d'un) d'un petabyte, que estaria fragmentat i replicat en un nombre considerable de nodes del clúster Hadoop. Aquest és precisament, com hem comentat anteriorment, el gran

avantatge de MapReduce: el fet de poder tractar grans conjunts de dades de forma distribuïda i paral·lela, on una execució seqüencial no seria possible, o bé tardaria un temps inassumible.

## 4. YARN

Ja vàrem veure en el primer lliurament que YARN és un dels components del nucli de Hadoop des de la versió 2. En concret, és el component que permet a Hadoop suportar diversos motors d'execució incloent MapReduce, i proporcionar també un planificador dels treballs que es troben en execució en el clúster.

YARN va aparèixer perquè, tot i l'eficiència de MapReduce, en la seva configuració inicial estava basat en lots (*batch*) i, com a resultat, no era adequat per al processament de dades en temps real ni, fins i tot, en temps quasi-real. Així, des de Hadoop v2, YARN es fa càrrec de les parts de gestió de recursos i planificació que en Hadoop v1 s'executaven en MapReduce.



Imatge: Aparició de YARN en Hadoop v2

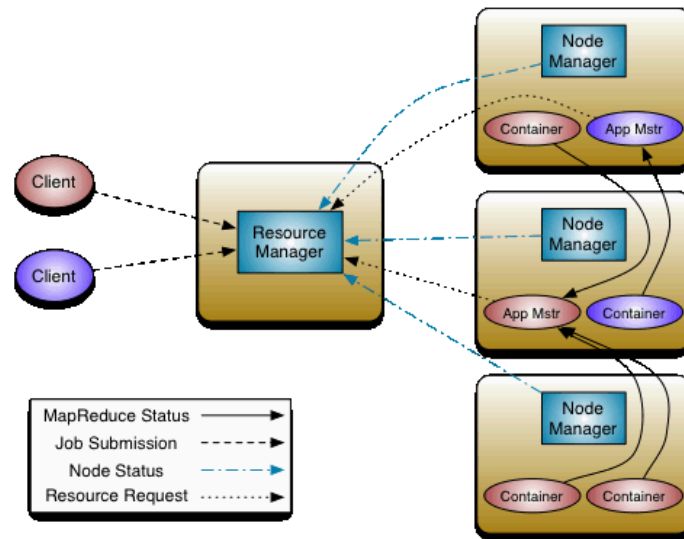
Principalment, s'encarrega de respondre a sol·licituds de clients per a crear un contenidor, supervisar els contenidors que s'estan executant i finalitzar-los si és necessari per a alliberar recursos. Els contenidors són els espais virtuals dins els nodes on s'executen les aplicacions i que, per fer-ho, tenen uns recursos de processador i memòria assignats.

YARN separa les seves dues funcionalitats principals: la gestió de recursos i la planificació i monitoratge de treballs. Amb aquesta idea, és possible tenir un gestor global (*Resource Manager*), mentre que cada aplicació té el seu propi *Application Master*.

YARN es divideix en tres components principals que són els següents:

- **Resource Manager:** és el component que orquestra i arbitra sobre l'ús dels recursos entre totes les aplicacions del sistema. El Resource Manager manté un llistat dels Node Managers actius i dels seus recursos disponibles. Per a complir aquesta tasca utilitza dos sub-components:
  - **Scheduler** o planificador: és l'encarregat de gestionar la distribució dels recursos del clúster. A més, les aplicacions usen els recursos que el Resource Manager els ha proporcionat en funció dels seus criteris de planificació. Aquest planificador no monitora l'estat de cap aplicació ni els ofereix garanties d'execució.
  - **Application Manager:** és el component responsable d'acceptar les peticions de treballs, localitzar el primer contenidor en el qual executar l'aplicació i proporcionar reinicis dels treballs en cas que fos necessari a causa d'errors.
- **Node Manager:** gestiona els treballs en un node concret segons les instruccions del Resource Manager i proporciona els recursos computacionals necessaris per a les aplicacions en forma de contenidors. Els contenidors YARN tenen una assignació de recursos (CPU, memòria, disc i xarxa) fixa d'un node del clúster i el Node Manager és l'encarregat de monitorar aquesta assignació. A més, s'encarrega de mapejar les variables d'entorn necessàries, les dependències i els serveis necessaris per a crear els processos d'execució. Hi ha un Node Manager per node del clúster.
- **Application Master:** és el responsable de negociar els recursos apropiats amb el Resource Manager i monitorar el seu estat i el seu progrés. També coordina l'execució de totes les tasques en les quals pot dividir-se la seva aplicació.





Imatge: Arquitectura YARN. Font: [hadoop.apache.org](http://hadoop.apache.org)

Per a executar una aplicació en YARN, el client contacta amb el Resource Manager i li demana que executi un procés Application Master. El Resource Manager troba un Node Manager que pot iniciar l'Application Master en un contenidor. Precisament el que fa l'Application Master una vegada que s'executa depèn de l'aplicació: pot, o bé executar una operació en el contenidor en el qual s'està executant i retornar el resultat corresponent, o bé sol·licitar més contenidors als Resource Manager i usar-los per a executar una operació distribuïda.

YARN admet la reserva de recursos a través del **Reservation System**, un component que permet als usuaris especificar un perfil de recursos en el temps i amb restriccions temporals (per exemple, terminis), i reservar recursos per a garantir l'execució predictable de treballs importants.

Per a escalar, més enllà d'un nombre petit de milers de nodes, YARN suporta el concepte de federació a través de la funció **YARN Federation**. La federació permet connectar de manera transparent múltiples (sub-)clústers i fer-los aparèixer com un únic clúster massiu. Això es pot utilitzar per a aconseguir una major escalabilitat, i/o per a permetre que diversos clústers independents s'utilitzin junts per a treballs molt grans.

## 4.1. Interfície web de YARN

En el primer lliurament també varem veure que YARN exposa una interfície web en el port 8088. En la màquina virtual de Cloudera també la tenim al port 8088:

The screenshot shows the Cloudera YARN web interface. The top navigation bar includes links for Cloudera, Hue, Hadoop, HBase, Impala, Spark, Solr, Oozie, Cloudera Manager, and Getting Started. The main header displays the Hadoop logo and the title 'All Applications'. On the left, there is a sidebar with navigation options: Cluster, About Nodes, Applications, NEW, NEW SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED, Scheduler, and Tools. The main content area is divided into several sections: Cluster Metrics, Cluster Nodes Metrics, User Metrics for dr.who, and a table of applications. The Cluster Metrics section shows 0 apps submitted, 0 pending, 0 running, and 0 completed. The Cluster Nodes Metrics section shows 1 active node and 0 decommissioning nodes. The User Metrics for dr.who section shows 0 apps submitted, 0 pending, 0 running, and 0 completed. The applications table is empty, showing 'No data available in table'.

**Imatge:** Interfície web del Resource Manager

Aquí podem veure que tenim un únic node actiu, que fa tant de mestre (NameNode) com d'esclau (DataNode). I també veim que no tenim cap aplicació que s'hagi executat. En canvi, si no heu aturat els serveis de Hadoop des que heu executat l'aplicació MapReduce per comptar les paraules del Quijote, podreu veure aquí que sí que us apareix l'aplicació.

Cluster Metrics															
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved					
1	0	0	1	0	0 B	8 GB	0 B	0	8	0					
Cluster Nodes Metrics															
Active Nodes		Decommissioning Nodes		Decommissioned Nodes		Lost Nodes		Unhealthy Nodes		Rebooted Nodes					
1	0			0		0		0		0					
User Metrics for dr.who															
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserved	VCores Used	VCores Pending	VCores Reserved			
0	0	0	0	0	0	0	0 B	0 B	0 B	0	0	0			
Show 20 ▾ entries															
Search: <input type="text"/>															
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCoers	Allocated Memory MB	Reserved CPU VCoers	Reserved Memory MB	Progress	Tracking UI
application_1669798153666_0001	cloudera	streamjob7572944766326299982.jar	MAPREDUCE	root.cloudera	Wed Nov 30 01:09:31 -0800 2022	Wed Nov 30 01:09:50 -0800 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	<div></div>	<a href="#">History</a>
Showing 1 to 1 of 1 entries															
First Previous 1 Next Last															

**Imatge:** Interfície web del Resource Manager amb una aplicació

Si feim clic sobre l'aplicació, en veurem els detalls:

Application Overview	
User:	cloudera
Name:	streamjob7572944766326299982.jar
Application Type:	MAPREDUCE
Application Tags:	
State:	FINISHED
FinalStatus:	SUCCEEDED
Started:	Wed Nov 30 01:09:31 -0800 2022
Elapsed:	18sec
Tracking URL:	<a href="#">History</a>
Diagnostics:	

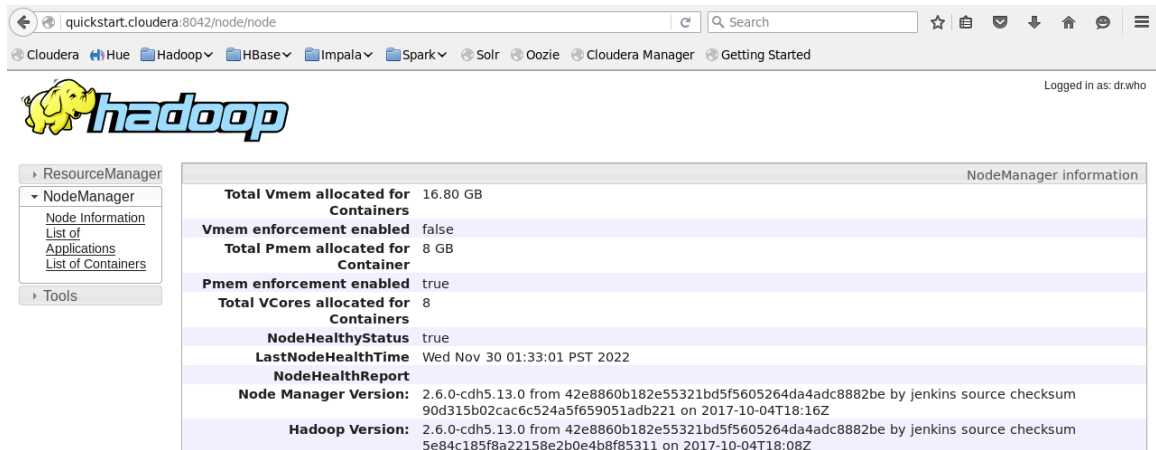
Application Metrics	
Total Resource Preempted:	<memory:0, vCores:0>
Total Number of Non-AM Containers Preempted:	0
Total Number of AM Containers Preempted:	0
Resource Preempted from Current Attempt:	<memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt:	0
Aggregate Resource Allocation:	68723 MB-seconds, 39 vcore-seconds

ApplicationMaster			
Attempt Number	Start Time	Node	Logs
1	Wed Nov 30 01:09:31 -0800 2022	quickstart.cloudera:8042	<a href="#">logs</a>

**Imatge:** Interfície web del Resource Manager, detalls d'una aplicació MapReduce

Podem veure que l'execució ha tardat 18 segons, i que ha utilitzat 68.723 MB de memòria i 39 segons dels cores virtuals (vcores). Després veurem que en aquesta màquina tenim 8 vcores disponibles per als contenidors. En aquest cas, on només tenim un node en el clúster i la tasca és molt petita, la informació no té massa interès. Però quan tenim diversos nodes, amb tasques molt més grans, sí que és interessant veure quins recursos s'estan utilitzant, mentre l'aplicació està en execució.

Per acabar, també tenim accés a la interfície del Node Manager al port 8042:

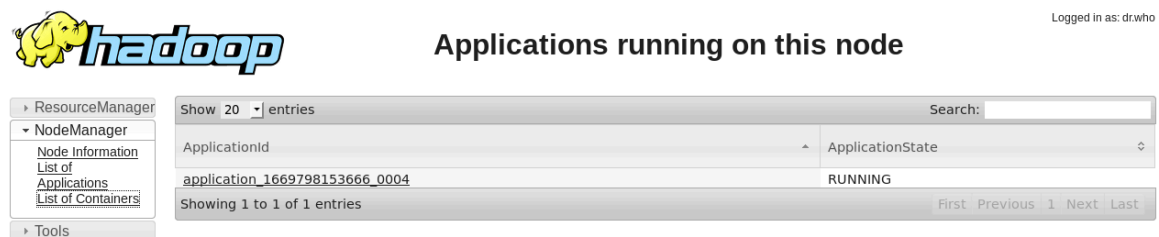


The screenshot shows the Hadoop Node Manager web interface. The left sidebar contains a navigation menu with options: ResourceManager, NodeManager (selected), Node Information, List of Applications, List of Containers, and Tools. The main content area displays 'NodeManager information' with the following details:

Total Vmem allocated for Containers	16.80 GB
Vmem enforcement enabled	false
Total Pmem allocated for Container	8 GB
Pmem enforcement enabled	true
Total Vcores allocated for Containers	8
NodeHealthyStatus	true
LastNodeHealthTime	Wed Nov 30 01:33:01 PST 2022
NodeHealthReport	
Node Manager Version:	2.6.0-cdh5.13.0 from 42e8860b182e55321bd5f5605264da4adc8882be by jenkins source checksum 90d315b02cac6c524a5f659051adb221 on 2017-10-04T18:16Z
Hadoop Version:	2.6.0-cdh5.13.0 from 42e8860b182e55321bd5f5605264da4adc8882be by jenkins source checksum 5e84c185f8a22158e2b0e4b8f85311 on 2017-10-04T18:08Z

**Imatge:** Interfície web del Node Manager

Aquí és on podem veure que tenim 8 cores virtuals (vcores) i 16,80 GB de memòria (vmem) disponibles per als contenidors. També podem veure les aplicacions que s'estan executant sobre el node en aquest moment (enllaç *List of Applications*) i quants contenidors n'utilitza (enllaç *List of Containers*). Per fer-ho, podem esborrar el directori d'HDFS *quijote2* i tornar a llençar l'aplicació. En la següent imatge podem observar que apareix la nostra aplicació en execució:



The screenshot shows the Hadoop Node Manager web interface with the 'List of Applications' view selected. The left sidebar is the same as the previous image. The main content area displays 'Applications running on this node' with a search bar and a table of applications.

ApplicationId	ApplicationState
application_1669798153666_0004	RUNNING

Showing 1 to 1 of 1 entries. Navigation links: First Previous 1 Next Last.

**Imatge:** Interfície web del Node Manager, List of Applications

I que en aquests moments estan utilitzant dos contenidors:



The screenshot shows the Hadoop Node Manager web interface with the 'List of Containers' view selected. The left sidebar is the same as the previous images. The main content area displays 'All containers running on this node' with a search bar and a table of containers.

ContainerId	ContainerState	logs
container_1669798153666_0004_01_000001	RUNNING	<a href="#">logs</a>
container_1669798153666_0004_01_000002	RUNNING	<a href="#">logs</a>

Showing 1 to 2 of 2 entries. Navigation links: First Previous 1 Next Last.

**Imatge:** Interfície web del Node Manager, List of Containers

Si miram els detalls del contenidor, podem veure que necessita 2048 MB de memòria i 1 vcore per a la seva execució:

Container information	
<b>ContainerID</b>	container_1669798153666_0005_01_000001
<b>ContainerState</b>	RUNNING
<b>ExitStatus</b>	N/A
<b>Diagnostics</b>	
<b>User</b>	cloudera
<b>TotalMemoryNeeded</b>	2048
<b>TotalVCoresNeeded</b>	1
<b>logs</b>	<a href="#">Link to logs</a>

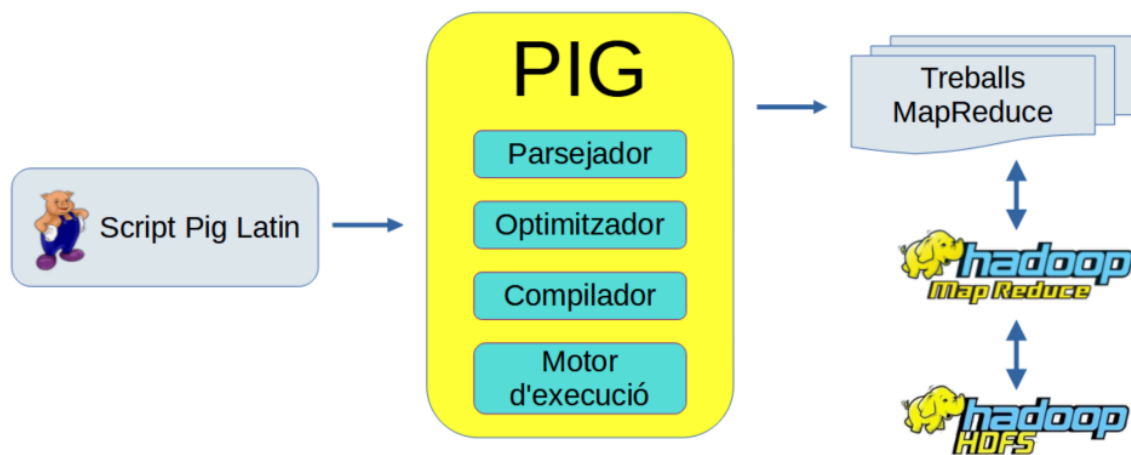
**Imatge:** *Interfície web del Node Manager, informació del contenidor*

## 5. Pig

Apache Pig és una eina de l'ecosistema Hadoop que ens permet escriure aplicacions MapReduce d'una manera més senzilla, amb un major nivell d'abstracció. A diferència de MapReduce i de YARN, que hem vist en els capítols anteriors, Pig no és un dels components del nucli de Hadoop.

Pig va ser desenvolupat originalment per Yahoo en 2006 i va ser adoptat posteriorment per l'Apache Software Foundation com a un subprojecte de Hadoop.

Una part fonamental de Pig és **Pig Latin**, un llenguatge de programació molt orientat als fluxos de dades típics de les aplicacions de Big Data. Pig s'encarrega de traduir les operacions escrites en Pig Latin a MapReduce, abans de ser executades en la plataforma Hadoop. D'aquesta manera, no ens hem d'ocupar dels detalls de MapReduce, sinó només en escriure les operacions amb les dades, d'una manera més senzilla, amb un major nivell d'abstracció.



Imatge: Arquitectura de Pig

Com es mostra a la imatge, Apache Pig té diversos components. Els scripts de Pig Latin (o comandes Pig en el Grunt Shell, després en xerrarem) passen pel **parsejador** (*parser*), que comprova la sintaxi i genera un graf dirigit acíclic (DAG). En aquest graf, les operacions lògiques es representen com a nodes i els fluxos de dades com a arestes. Seguidament, l'**optimitzador** (*optimizer*) aplica diverses tècniques d'optimització sobre aquest graf per a dividir, fusionar, transformar i reordenar els operadors. A continuació el **compilador** (*compiler*) genera una sèrie de treballs MapReduce. Finalment, el motor d'execució (*execution engine*) envia aquests treballs, en l'ordre adequat, a Hadoop MapReduce perquè els executi. Aquests treballs MapReduce accediran a HDFS per llegir les dades d'entrada i, normalment per escriure-hi les dades de sortida.

## 5.1. Maneres d'executar Pig

Podem executar els scripts Pig de tres maneres diferents:

- **Mode interactiu (Grunt Shell):** podem obrir el Grunt Shell, que és un shell interactiu que ens permet escriure i executar sentències de Pig Latin, una a una.
- **Mode batch:** podem escriure totes les sentències de Pig Latin en un únic fitxer amb l'extensió .pig i executar-les totes de cop.
- **Mode incrustat (UDF):** Pig proporciona la possibilitat de definir les nostres funcions (User Defined Functions) en altres llenguatges de programació com ara Java (i en menor mesura, Python), i usar-les dins del nostre script.

En aquest curs ens centrarem en les dues primeres.

Per altra banda, podem executar Apache Pig en dos modes: mode local i mode MapReduce.

En el **mode local**, es fa feina amb una única màquina virtual Java (JVM) en el host local, i totes les dades, tant d'entrada com de sortida, estan en el sistema de fitxers local. No treballa amb HDFS ni amb Hadoop. Aquest mode només es fa servir per a proves, no per a sistemes en producció.

En el **mode MapReduce** (o també anomenat mode HDFS), s'utilitzen dades que estan en HDFS. Així, cada vegada que executam sentències de Pig Latin per a processar dades, es fa una traducció i s'invoca un treball MapReduce que executarà una operació sobre les dades emmagatzemades en HDFS, de manera distribuïda. És el mode per defecte.

Així, per obrir el Grunt Shell per executar sentències Pig Latin de manera interactiva, si ho volem fer amb el mode local, escriurem l'ordre:

```
pig -x local
```

Mentre que per a treballar amb el mode mapreduce escriurem:

```
pig -x mapreduce
```

o, com que és el mode per defecte, simplement:

```
pig
```

```
grunt> [cloudera@quickstart ~]$
[cloudera@quickstart ~]$ pig -x mapreduce
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
2022-12-01 03:00:37,157 [main] INFO org.apache.pig.Main - Apache Pig version 0.12.0-cdh5.13.0 (reexported) compiled Oct 04 2017, 11:09:03
2022-12-01 03:00:37,157 [main] INFO org.apache.pig.Main - Logging error messages to: /home/cloudera/pig/1669892437149.log
2022-12-01 03:00:37,167 [main] INFO org.apache.pig.impl.util.Util - Default bootstrap file /home/cloudera/pig/bootstrap not found
2022-12-01 03:00:37,606 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-12-01 03:00:37,606 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-12-01 03:00:37,606 [main] INFO org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: hdfs://quickstart.cloudera:8020
2022-12-01 03:00:38,457 [main] INFO org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to map-reduce job tracker at: localhost:8021
2022-12-01 03:00:38,457 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-12-01 03:00:38,482 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-12-01 03:00:38,483 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-12-01 03:00:38,532 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-12-01 03:00:38,532 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-12-01 03:00:38,566 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-12-01 03:00:38,567 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-12-01 03:00:38,589 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-12-01 03:00:38,590 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-12-01 03:00:38,610 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-12-01 03:00:38,706 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-12-01 03:00:38,730 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-12-01 03:00:38,730 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-12-01 03:00:38,747 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-12-01 03:00:38,747 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-12-01 03:00:38,766 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-12-01 03:00:38,766 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
grunt> █
```

**Imatge:** Inici de Grunt Shell (en mode mapreduce) en la Cloudera Quickstart VM

Això obrirà el Grunt Shell, on podem escriure sentències Pig Latin. Per sortir del shell, hem d'escriure l'ordre *quit* o pitjar <ctrl>+D.



## 5.2. Pig Latin. Model de dades

El model de dades Pig Latin treballa amb relacions, bosses, tuples i camps.

Per entendre aquests conceptes, vegem un exemple molt senzill. Tenim aquestes dades:

id	nom	edat
1	Joan	50
2	Aina	30
3	Pep	40

Un **camp** és un fragment de dades, que pren un valor atòmic. En l'exemple, *nom* és un camp. En Pig, els valors atòmics poden ser de tipus *integer*, *double*, *float*, *byte array* i *char array* (una cadena de caràcters).

Una **tupla** és un conjunt ordenat de camps. És l'equivalent a una fila en una taula d'una base de dades relacional. Les tuples es representen entre parèntesis, amb els valors de cada camp, separats per comes. Per exemple: *(1, Joan, 50)*

Una **bossa** (*bag* en anglès, també anomenada sac) és una col·lecció de tuples. Pot contenir totes les tuples o només un subconjunt d'elles (per exemple, quan retornam el resultat d'una consulta). Se sol representar entre claus, amb les tuples separades per comes. Per exemple: *{(1, Joan, 50), (3, Pep, 40)}*

Una **relació** representa una taula completa. En realitat una relació és també una bossa, ja que és una col·lecció de tuples: es denomina una bossa exterior (*outer bag*).

Si ho comparam amb una base de dades relacional, una relació és una taula i una tupla és una fila. Però hi ha una diferència important: les tuples de pig no tenen perquè tenir totes la mateixa estructura. Així que en una mateixa relació podem tenir tuples amb diferent número de camps o que, per exemple, el 3er camp de dues tuples siguin de tipus de dades diferents.



## 5.3. Pig Latin. Sentències

Les sentències Pig Latin són les construccions bàsiques que s'utilitzen per a processar dades amb Pig. Les sentències Pig Latin poden abastar diverses línies i han d'acabar amb un punt i coma (;).

Una sentència Pig Latin és un operador que pren una relació (taula) com a entrada i produeix una altra relació com a sortida. Només hi ha dues excepcions, les sentències LOAD i STORE, que serveixen per llegir i escriure dades en el sistema d'arxius.

Els scripts Pig Latin s'estructuren normalment en tres fases: extracció, transformació i càrrega o emmagatzematge. És el que s'anomena comunament com a processos **ETL** (*Extraction, Transformation, Load*). Vegem cada una de les fases i les sentències més habituals en cada una d'elles.

### Fase d'extracció

L'operació d'extracció es fa mitjançant la sentència **LOAD**, que permet llegir dades des del sistema d'arxius (ja sigui el local o HDFS, segons el mode d'execució).

Vegem un exemple:

```
alumnes = LOAD 'alumnes.txt';
```

Això llegeix el fitxer alumnes.txt (del sistema d'arxius local o d'HDFS, segons el mode d'execució) i crea una bossa amb una tupla per a cada línia. Si el contingut del fitxer alumnes.txt és aquest:

```
1, Joan, 50, home
2, Aina, 30, dona
3, Pep, 40, home
```

Això crea una relació (anomenada *alumnes*) amb tres tuples:

```
(1, Joan, 50, home)
(2, Aina, 30, dona)
(3, Pep, 40, home)
```

Però aquí cada tupla té només un camp, que conté tot el text d'una línia. Normalment ens interessa especificar els detalls del format del fitxer, cosa que farem emprant les clàusules **USING** i **AS**, així com la funció **PipStorage**. La funció *PipStorage* serveix per carregar i emmagatzemar dades com a fitxers de text estructurat. Té un paràmetre que identifica el caràcter emprat per separar els camps, que per defecte és el tabulador ('\t'). Seguint el nostre exemple, ho escriuríem així:

```
alumnes = LOAD 'alumnes.txt'
  USING PigStorage(',')
  AS ( id:int, nom:chararray, edat:int, sexe:chararray );
```

Amb *USING PigStorage(',')* indicam que els camps estan separats per comes, i amb la clàusula *AS* especificam l'esquema de cada fila del fitxer i, per tant, de cada tupla de la relació que es genera.

Això genera una relació amb tres tuples, cada una amb quatre camps, *id*, *nom*, *edat* i *sexe*.

Ho podem comprovar que l'estructura és la correcta, escrivint:

```
DESCRIBE alumnes;
```

La qual cosa ens mostra l'esquema de la relació alumnes:

```
alumnes: {id: int,nom: chararray,edat: int,sexe: chararray}
```

I si volem veure les dades, podem emprar l'operador DUMP:

```
DUMP alumnes;
```

la qual cosa ens escriu a la consola:

```
(1, Joan, 50, home)
(2, Aina, 30, dona)
(3, Pep, 40, home)
```

Fixau-vos que és quan s'executa la sentència DUMP quan es generen i executen els treballs MapReduce.

### Fase de transformació

Pig disposa d'un ampli conjunt d'operadors per al processament i la transformació de les dades. Els més usats són els següents:

- **FILTER:** Selecciona tuples (files) d'una relació (taula) basant-se en alguna condició.
- **FOREACH ... GENERATE:** Genera transformacions de dades basades en columnes de dades.
- **GROUP/COGROUP:** Agrupa les dades en una o diverses relacions. Són operacions idèntiques i poden treballar amb una o més relacions. Per a facilitar la lectura, GROUP s'utilitza en les sentències que impliquen una relació i COGROUP s'utilitza en les sentències que impliquen dues o més relacions.
- **JOIN (inner/outer):** Realitza una unió externa de dues relacions basada en valors de camp comuns.
- **UNION:** Fusiona el contingut de dues o més relacions.
- **SPLIT:** Divideix el contingut d'una relació en diverses relacions.

Seguint l'exemple dels alumnes, ens interessa saber quins són els alumnes amb més de 45 anys. Ho escriuríem així:

```
alumnemes45 = FILTER alumnes BY edat>=45;
```

Això ens retorna una relació amb una única tupla:

```
(1, Joan, 50, home)
```

Mentre que els alumnes de menys de 45 serien aquests:

```
alumnemenys45 = FILTER alumnes BY edat<45;
```

Això ens retorna una relació amb dues tuples:

```
(2, Aina, 30, dona)
(3, Pep, 40, home)
```

Quan necessitam una operació que processa totes les files d'una relació, empram l'operador FOREACH ... GENERATE. Un cas típic és quan volem fer una projecció, per exemple, per retornar una relació només amb el nom i l'edat de cada tupla:

```
alumnemes2 = FOREACH alumnes GENERATE nom, edat;
```

Vegem ara un exemple d'una operació típica d'anàlisi de dades. Anam a calcular l'edat mitjana de cada sexe. Per fer-ho, primer haurem d'agrupar les dades pel camp sexe:

```
alumneseixe = GROUP alumnes BY sexe;
```

És important entendre l'estructura de la relació *alumneseixe* que s'ha generat. Fent un *DESCRIBE alumneseixe* obtenim això:

```
alumneseixe: {group: chararray,
  alumnes: {(id: int,nom: chararray,edat: int,sexe: chararray)}}
```

Podem observar que *alumneseixe* té dos camps, un amb el valor pel qual hem agrupat (el sexe en el nostre cas) i l'altre amb cada una de les tuples que pertanyen al grup. Fem un DUMP per veure millor com són les dues tuples que ha generat, una per al grup "dona" i l'altra per al grup "home":

```
( dona, {(2, Aina, 30, dona)})
( home, {(3, Pep, 40, home), (1, Joan, 50, home)})
```

Ara podem emprar FOREACH ... GENERATE per a obtenir l'edat mitjana de cada grup, fent servir també la funció AVG:

```
edatsmitjanes = FOREACH alumnessexe GENERATE group, AVG(alumnes.edat);
```

Si feim un DUMP de la relació *edatsmitjanes*, obtenim:

```
( dona, 30.0)
( home, 45.0)
```

### Fase de càrrega o emmagatzematge

Per emmagatzemar les dades, tenim disponible l'operador **STORE**, que permet escriure una relació en el sistema d'arxius (ja sigui local o HDFS, segons el mode d'execució). Per exemple:

```
STORE edatsmitjanes INTO 'edatsmitjanes';
```

Això crea un directori *edatsmitjanes* (en local o en HDFS) i hi posa l'arxiu *\_SUCCESS* i els arxius necessaris amb els blocs (parts) de les dades (recordau que per defecte, en Hadoop2 els blocs són de 128 Mb). És a dir, si estam en el mode d'execució local, crea la mateixa estructura que empraria per a HDFS. La única diferència és que la guarda en el sistema d'arxius local, enlloc de fer-ho en HDFS.

Per defecte, el caràcter separador és el tabulador. Si volem canviar-ho fem servir la clàusula USING amb la funció PigStorage. Per exemple, per emprar comes:

```
STORE edatsmitjanes INTO 'edatsmitjanes' USING PigStorage (',');
```

Dins aquest apartat d'operadors per a l'emmagatzematge, tenim també **DUMP**, que ja hem vist que escriu una relació a la consola. En tot cas, hem de tenir clar que DUMP només s'ha d'utilitzar en entorns de proves, mentre que en producció sempre emprarem STORE.



IMPORTANT

És important tenir clar que quan s'executa una sentència d'emmagatzematge és quan es generen i executen els treballs MapReduce associats a totes les sentències de càrrega i transformació que s'han escrit anteriorment. Fins aquest moment només s'han utilitzat estructures en memòria per representar les operacions que s'han de dur a terme, però encara no s'ha fet realment feina amb les dades, que no oblidem que solen estar distribuïdes en el clúster Hadoop. És en aquest moment quan intervenen l'optimitzador, el compilador i el motor d'execució de Pig.

## 5.4. Exemple amb un fitxer de log

En aquest apartat veurem un senzill exemple que treballa amb un fitxer de log. En concret, volem extreure els missatges de warning del fitxer de log del NameNode d'HDFS: `/var/log/hadoop-hdfs/hadoop-hdfs-namenode-quickstart.cloudera.log`

Entrarem en pig en mode local:

```
pig -x local
```

Carregarem el fitxer mitjançant l'operador *LOAD*:

```
missatges = LOAD '/var/log/hadoop-hdfs/hadoop-hdfs-namenode-quickstart.cloudera.log';
```

Ara hem d'aplicar un filtre (*FILTER ... BY ...*), per obtenir només els missatges de warning. Aquests missatges, després de la data i hora, comencen per la cadena `WARN`. Per tant, el filtre serà `$0 MATCHES '.*WARN+.*'`:

```
warnings = FILTER missatges BY $0 MATCHES '.*WARN+.*';
```

I ara ja podem guardar el resultat mitjançant un operador *STORE*:

```
STORE warnings INTO 'warnings';
```

En el moment en què executem aquesta sentència es durà a terme la traducció i execució dels treballs MapReduce associats a totes les sentències Pig prèvies (recordau que la traducció i execució del codi MapReduce es fa quan s'executa una sentència *STORE* o *DUMP*). Això ens crearà un directori `warnings` al directori `/home/cloudera`. En el fitxer `part-m-00000` podrem trobar el resultat, és a dir, tots els missatges de warning.



Amb aquest exemple hem vist com carregar dades en Pig (amb *LOAD*), en aquest cas des d'un fitxer de log; com aplicar un filtre senzill (amb *FILTER ... BY ...*) i com exportar el resultat a un fitxer (amb *STORE*).

## 5.5. Exemple de comptar paraules

En aquest exemple anam a veure una implementació del càlcul de les ocurrències de les paraules d'un text, que ja varem veure al capítol de MapReduce. Executarem Pig en mode *mapreduce*.

En primer lloc, hem de carregar el fitxer de text quijote/quijote.txt, que ja teníem en HDFS i l'hem de llegir per línies:

```
linies = LOAD 'quijote/quijote.txt' AS (linia:chararray);
```

Per entendre-ho millor, anam a suposar que en el fitxer de text només tenim el famós principi de El Quijote:

*En un lugar de la Mancha, de cuyo nombre no quiero acordarme,*

Com que només tenim una línia, el resultat seria aquest:

```
(En un lugar de la Mancha, de cuyo nombre no quiero acordarme,)
```

Ara començam la transformació. Processant línia a línia (FOREACH linies GENERATE ...), volem obtenir una tupla amb totes les paraules (on cada paraula és un *chararray*) del text. Per això emprarem aquestes dues funcions:

- TOKENIZE, que transforma una línia en una bossa (*bag*) de paraules. La funció té en compte els espais i signes de puntuació per separar les paraules.
- FLATTEN, que converteix una bossa en diverses tuples

Ho escrivim així:

```
paraules = FOREACH linies GENERATE FLATTEN(TOKENIZE(linia)) as paraula;
```

El resultat seria aquest:

```
(En)
(un)
(lugar)
(de)
(la)
(Mancha)
(de)
(cuyo)
(nombre)
(no)
(quiero)
(acordarme)
```

Ara hem d'agrupar les paraules, emprant GROUP ... BY ...:

```
paraulesagrupades = GROUP paraules BY paraula;
```

El resultat és:

```
(En, {(En)})
(de, {(de), (de)})
(la, {(la)})
(no, {(no)})
(un, {(un)})
(cuyo, {(cuyo)})
(lugar, {(lugar)})
(Mancha, {(Mancha)})
(nombre, {(nombre)})
(quiero, {(quiero)})
(acordarme, {(acordarme)})
```

Podem veure que la paraula "de" és l'única apareix dues vegades.

Ara anam a fer el recompte de quantes vegades apareix cada una de les paraules:

```
paraulescomptades = FOREACH paraulesagrupades GENERATE group, COUNT(paraules);
```

El resultat seria aquest:

```
(En,1)
(de,2)
(la,1)
(no,1)
(un,1)
(cuyo,1)
(lugar,1)
(Mancha,1)
(nombre,1)
(quiero,1)
(acordarme,1)
```

Recordau que fins que no executam una operació d'emmagatzematge (STORE o DUMP), no es generen i executen els treballs MapReduce. Per veure-ho en pantalla seria:

```
DUMP paraulescomptades ;
```

Resumint, el codi per fer el recompte de paraules en Pig seria aquest:

```
linies = LOAD 'quijote/quijote.txt' AS (linia:chararray);

paraules = FOREACH linies GENERATE FLATTEN(TOKENIZE(linia)) as paraula;

paraulesagrupades = GROUP paraules BY paraula;

paraulescomptades = FOREACH paraulesagrupades GENERATE group, COUNT(paraules);

DUMP paraulescomptades ;
```



IMPORTANT

Només cinc línies, molt més senzill que el que havíem vist en l'apartat de MapReduce!

Aquest és el gran avantatge de Pig, que ens permet escriure codi MapReduce d'una manera molt més senzilla.

Podríem guardar aquestes 5 línies en un fitxer (per exemple, `quijote.pig`) i executar-lo en mode batch, des del shell del sistema operatiu:

```
pig quijote.pig
```

O si fos en mode local:

```
pig -x local quijote.pig
```

També podem executar el script des del shell de Grunt amb l'ordre `exec`:

```
exec myscript.pig
```

Si volem introduir comentaris dins dels fitxers de codi Pig Latin, ho farem amb `--` per a comentaris d'una línia i entre `/*` i `*/` per a comentaris de més d'una línia:

```
-- això és un comentari d'una línia
/* Això és un comentari
de més d'una línia */
```

Quan treballem amb cadenes de caràcters com en aquest cas, és molt habitual fer feina amb les funcions que Pig Latin incorpora per a strings. Per exemple, si només ens interessassin les paraules que comencin per a, podríem definir un filtre amb la funció `STARTSWITH`, abans de fer l'agrupació:

```
paraulesa = FILTER paraules BY (STARTSWITH(paraula, 'a'))
```

IMPORTANT

Podeu trobar totes les funcions per treballar amb cadenes de caràcters que incorpora Pig Latin a <https://pig.apache.org/docs/r0.17.0/func.html#string-functions>

RESUM

Amb aquest exemple hem vist com extreure les paraules d'un text (amb les funcions *TOKENIZE* i *FLATTEN*), com agrupar les files (amb *GROUP ... BY ...*) i com fer un recompte de les ocurrències de cada un dels grups, mitjançant *FOREACH ... GENERATE ...* i la funció *COUNT*.

També hem vist com executar un fitxer amb un script Pig Latin en mode batch i com introduir comentaris dins del codi.

## 5.6. Exemple amb dades estructurades

En aquest apartat veurem un exemple d'un cas bastant habitual, on el que volem és treballar amb grans fitxers de dades amb una estructura determinada. En concret, carregarem les dades des d'un fitxer CSV.

Treballarem amb el dataset de [vendes de videojocs](https://www.kaggle.com/datasets/tnavarrete/bigdata-24-25) que podeu descarregar de Kaggle, una comunitat en línia sobre ciència de dades i aprenentatge automàtic creada per Google i que té multitud de datasets. Pots també descarregar-ho des de <https://github.com/tnavarrete-iedib/bigdata-24-25/blob/main/vgsales.csv> o des de la consola amb:

```
wget https://raw.githubusercontent.com/tnavarrete-iedib/bigdata-24-25/refs/heads/main/vgsales.csv
```

El fitxer té les següents columnes i tipus de dades:

- ranking: int
- nom: chararray
- plataforma: chararray
- any: int
- genere: chararray
- publicador: chararray
- vendes\_nordamerica: float (en milions de dòlars)
- vendes\_europa: float (en milions de dòlars)
- vendes\_japo: float (en milions de dòlars)
- vendes\_altres: float (en milions de dòlars)
- vendes\_globals: float (en milions de dòlars)

En aquest exercici, podem treballar amb Pig en mode local o bé carregar el fitxer CSV en HDFS i treballar en mode *mapreduce*.

Per carregar el fitxer en Pig, ho faríem així:

```
videojocs = LOAD 'vgsales.csv' USING PigStorage(',') AS (  
  ranking:int,  
  nom:chararray,  
  plataforma:chararray,  
  any:int,  
  genere:chararray,  
  publicador:chararray,  
  vendes_nordamerica:float,  
  vendes_europa:float,  
  vendes_japo:float,  
  vendes_altres:float,  
  vendes_globals:float );
```

Apareixeran alguns missatges d'avís de que s'han fet conversions de tipus automàtiques, que podem obviar.

El problema de fer-ho així, però, és que ens agafa també la primera línia del fitxer CSV, on tenim la capçalera, com si fossin dades. Per no tenir-la en compte, podem aplicar un filtre: considerem només les files que tenen el rànquing major que 0:

```
videojocs2 = FILTER videojocs BY ranking > 0.0;
```

Una altra manera més senzilla, sense haver d'aplicar cap filtre, és emprar un altra classe per importar les dades. En lloc d'utilitzar *PigStorage*, podem fer servir *CSVExcelStorage*. Primer hem de registrar la llibreria *piggybank.jar* (que està a */usr/lib/pig*), on està la classe *CSVExcelStorage*, i després fer la càrrega, especificant les opcions 'NO\_MULTILINE', 'UNIX', 'SKIP\_INPUT\_HEADER':



```
REGISTER /usr/lib/pig/piggybank.jar;
```

```
videojocs2 = LOAD 'vgsales.csv'  
  USING org.apache.pig.piggybank.storage.CSVExcelStorage(  
    ',', 'NO_MULTILINE', 'UNIX', 'SKIP_INPUT_HEADER' )  
  AS ( ranquing:int,  
    nom:chararray,  
    plataforma:chararray,  
    any:int,  
    genere:chararray,  
    publicador:chararray,  
    vendes_nordamerica:float,  
    vendes_europa:float,  
    vendes_japo:float,  
    vendes_altres:float,  
    vendes_globals:float );
```

En qualsevol dels dos cassos tenim en *videojocs2* les dades del fitxer CSV. Anam primer a comptar el número de files. Per aplicar la funció COUNT, s'ha de fer primer una agrupació. En el nostre cas, com que en l'agrupació volem totes les files, farem un GROUP ... **ALL**:

```
grup = GROUP videojocs2 ALL;  
numero = FOREACH grup GENERATE COUNT(videojocs2);  
DUMP numero;
```

Això ens diu que tenim 16.598 files.

De la mateixa manera podem recuperar el màxim dels valors d'un camp, fent servir la funció MAX (o MIN per al mínim). En el següent exemple trobam el valor més alt de les vendes globals:

```
maxim = FOREACH grup GENERATE MAX(videojocs2.vendes_globals);  
DUMP maxim;
```

El resultat és 82.74 milions de dòlars.

A continuació anam a fer un cas típic d'anàlisi de les dades. El que volem saber ara és, dels jocs de plataforma Wii, quines vendes totals hem tengut per a cada gènere.

Primer, haurem de fer un filtre, perquè només consideri els de la plataforma Wii:

```
wii = FILTER videojocs2 BY plataforma == 'Wii';
```

A continuació, hem d'agrupar les dades pel gènere:

```
generes = GROUP wii BY genere;
```

Ens queda ara fer la suma per a cada una de les agrupacions (gèneres):

```
vendes = FOREACH generes GENERATE group, SUM(wii.vendes_globals) as vendes_globals;
```



Amb això ja hem vist quasi totes les funcions d'agregació disponibles en Pig Latin, que són:

AVG (per obtenir la mitjana), MIN, MAX, SUM i COUNT.

Abans de veure els resultats, anam a ordenar-los de més a menys vendes globals (ordre descendent):

```
vendes_ordenades = ORDER vendes BY vendes_globals DESC;
```

Podem exportar *vendes\_ordenades* a un fitxer o fer un DUMP per veure els resultats finals, que són aquests:

```
(Sports,292.05999721959233)
(Misc,221.06000000797212)
(Action,118.57999973371625)
(Platform,90.74000111036003)
(Racing,61.279999643564224)
(Simulation,36.96999970078468)
(Shooter,28.770000020042062)
(Fighting,23.859999924898148)
(Adventure,18.429999897256494)
(Puzzle,15.6700001899153)
(Role-Playing,14.05999998562038)
(Strategy,5.230000013485551)
```

Amb l'operador LIMIT podem limitar el nombre de tuples retornades. Així doncs, podem limitar-ho a 1 per recuperar les dades del gènere amb més vendes (seria l'equivalent a calcular un màxim):

```
vendes_max = LIMIT vendes_ordenades 1;
```

Podem observar així que el gènere amb més vendes per a la plataforma Wii és el d'esports amb més de 292 milions de dòlars.



RESUM

Amb aquest exemple hem vist com carregar dades des d'un fitxer CSV (emprant la classe *CSVExcelStorage*), com fer un recompte de les files, com filtrar per una condició, com fer agrupacions i ordenacions, i com limitar el número de resultats. Aquestes són les operacions més habituals per fer una anàlisi de dades amb un dataset.

## 5.7. Més informació

Podeu trobar molta més informació sobre Apache Pig a la documentació oficial: <https://pig.apache.org/docs/latest/index.html>

En concret, és molt recomanable veure les funcions que incorpora Pig Latin: <https://pig.apache.org/docs/latest/func.html>

També us pot resultar molt útil aquest tutorial introductori: [https://www.tutorialspoint.com/apache\\_pig/index.htm](https://www.tutorialspoint.com/apache_pig/index.htm)

En concret, podeu veure la part dedicada a les funcions més utilitzades: [https://www.tutorialspoint.com/apache\\_pig/apache\\_pig\\_eval\\_functions.htm](https://www.tutorialspoint.com/apache_pig/apache_pig_eval_functions.htm) .