Ciclo de vida Analítico del dato

Práctica 1: Pig y Hive

Daniel Pérez Efremova

Índice

Aclaraciones

La práctica se ha realizado con un pc personal, es decir, sin la máquina virtual del curso. No se ha utilizado dockers.

Aunque no se pedía para el ejercicio, se han desarrollado scripts para facilitar y agilizar el proceso de pruebas. Como demostración de autoría y por sencillez en la exposición, se adjuntan en los anexos varias capturas en función del programa:

- **Pig local**: Capturas de pantalla de la consola donde se ha ejecutado la consulta.
- **Pig mapreduce**: captura de pantalla de la interfaz de hadoop donde se ve la ruta donde se almacenan los correspondientes resultados en formato csv.
- **Hive**: captura de pantalla de la interfaz de hadoop donde se ven las bases de datos creadas y su contenido.

Para facilitar la corrección del ejercicio, se ha preparado el script bash *ejecuciones.sh*. Para usarlo, basta ejecutarlo pasando como argumento la ruta local al fichero *athleteEvents.csv* para los scripts de Pig local. Este script se encarga de subir a HDFS los datos, reiniciar el metastore, levantar el servicio Derby y ejecutar todas las consultas secuencialmente.

Creación de directorios en HDFS

En primer lugar se crean los directorios *hadoop/dataset* y *hadoop/out*.

Código 1: Creación de los directorios de trabajo

```
daniel@daniel:~$ hdfs dfs -ls /

drwxr-xr-x - daniel supergroup 0 2021-09-18 17:53 /test
drwx------ - daniel supergroup 0 2021-09-19 11:55 /tmp
drwxr-xr-x - daniel supergroup 0 2021-09-18 18:12 /user

daniel@daniel:~$ hdfs dfs -mkdir /hadoop/dataset
daniel@daniel:~$ hdfs dfs -mkdir /hadoop/out

daniel@daniel:~$ hdfs dfs -ls /hadoop/
Found 2 items
drwxr-xr-x - daniel supergroup 0 2021-09-19 17:52 /hadoop/dataset
drwxr-xr-x - daniel supergroup 0 2021-09-19 18:01 /hadoop/out
```

Después, se copia el archivo *athleteEvents.csv* en HDFS, se comprueba que efectivamente está en el directorio y se listan las primeras líneas. En el código 2 se ve que el archivo está cargado con un tamaño aproximado de 42 Mb.

Código 2: Ingesta de datos

```
daniel@daniel:~$ hdfs dfs -put ~/Downloads/athleteEvents.csv
/hadoop/dataset

daniel@daniel:~$ hdfs dfs -ls /hadoop/dataset
Found 1 items
-rw-r--r-- 1 daniel supergroup 41500688 2021-09-19 18:08
 /hadoop/dataset/athleteEvents.csv

daniel@daniel:~$ hdfs dfs -cat /hadoop/dataset/athleteEvents.csv | head

"ID","Name","Sex","Age","Height","Weight","Team","NOC","Games","Year","
   Season","City","Sport","Event","Medal"
"1","A Dijiang","M",24,180,80,"China","CHN","1992 Summer",1992,"Summer","
   Barcelona","Basketball","Basketball Men's Basketball",NA
"2","A Lamusi","M",23,170,60,"China","CHN","2012 Summer",2012,"Summer","
   London","Judo","Judo Men's Extra-Lightweight",NA
"3","Gunnar Nielsen Aaby","M",24,NA,NA,"Denmark","DEN","1920 Summer",1920,
   "Summer","Antwerpen","Football","Football Men's Football",NA ...
```

Creación de la base de datos para Hive

Para crear la base de datos, se usa el script *BD_practicas.hql*. Este script se encarga de crear dos bases de datos:

- **practicas**. Almacena los datos del ejercicio.
- **resuls_P1_pighive**. Almacena los resultados de las consultas en formato .csv.

Código 3: Creación de bases de datos

```
CREATE DATABASE IF NOT EXISTS practicas;
CREATE DATABASE IF NOT EXISTS resuls_P1_pighive;
```

Por otro lado, se crea la tabla atletas con los datos del ejercicio en *practicas* y se sobrescribe con los datos desde HDFS. Finalmente, se muestran por consola las bases de datos y las primeras líneas de los datos, para controlar que todo se ha ejecutado correctamente.

Código 4: Creación de bases de datos

```
SET input='hdfs:///hadoop/dataset/athleteEvents.csv';
CREATE TABLE IF NOT EXISTS practicas.atletas (
ID int,
Name string,
Sex string,
Age int,
Height int,
Weight int,
Team string,
NOC string,
Games string,
Year int,
Season string,
City string,
Sport string,
Event string,
Medal string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
"separatorChar" = ",",
"quoteChar"     = "\""
)
STORED AS TEXTFILE
TBLPROPERTIES("skip.header.line.count"="1");

LOAD DATA INPATH {hiveconf:input}
OVERWRITE INTO TABLE practicas.atletas;

-- Comprobacion creacion DB
SHOW DATABASES;

-- Comprobacion de la carga bde datos
SELECT * FROM practicas.atletas LIMIT 5;
```

Consultas

Esta sección se divide en cuatro partes donde se presentan y explican las ideas detrás de cada script solución. En cada código se presentan la consultas, tanto para Pig como para Hive.

Primera consulta (C1)

La idea en los scripts consiste en crear un subconjunto del dataset que contenga sólo las medallas de Oro. Después, calcular el mínimo de la columna *Age* sobre este subconjunto. Finalmente, se filtra por la edad mínima.

Código 5: C1 para PIG

```
medallas_oro = FILTER atletas BY Medal == 'Gold';

edad_minima = FOREACH (GROUP medallas_oro ALL)
GENERATE MIN(medallas_oro.Age) AS valor:int;

medallistas = FOREACH (
FILTER medallas_oro BY Age == edad_minima.valor)
GENERATE Name, Games, Year, Sport, Event;
DUMP medallistas;

STORE medallistas INTO '$output' USING PigStorage (',');
```

Código 6: C1 para Hive

```
SELECT name, age, year, games, sport, event FROM practicas.atletas WHERE
    Age IN
(
SELECT MIN(Age) FROM practicas.atletas WHERE Medal = 'Gold'
) AND Medal = 'Gold';
```

Segunda consulta (C2)

La estrategia consiste en filtrar los datos de atletas por los que tengan a España por equipo. Después, basta ordenarlos descendentemente por la variable nombre.

Código 7: C2 para Pig

```
espanoles = FILTER atletas BY Team == 'Spain';
espanoles_ord = ORDER espanoles BY Name;
DUMP espanoles_ord;
```

Código 8: C2 para Hive

```
SELECT DISTINCT * FROM practicas.atletas WHERE Team='Spain' ORDER BY Name
    DESC;
```

Tercera consulta (C3)

En esta consulta se distinguen dos partes, la primera para obtener el mejor atleta en la disciplina de vela y la segunda para obtener todas las olimpiadas por continente.

Para la primera parte, se considera el mejor atleta al que tenga más medallas de oro en toda la historia de los juegos olímpicos. Primero, se filtran los datos de atletas por la disciplina sailing y por medallas de oro[1]. Después, para obtener el número de medallas de cada atleta se hace un recuento de las veces que aparece cada deportista en los datos filtrados. Lo siguiente es calcular el máximo de medallas sobre el recuento[2]. Finalmente, se seleccionan los deportistas que tengan un número de medallas de oro igual al máximo calculado.

Código 9: C3 para el mejor atleta Sailing en Pig

```
-- filtro por medallas de oro y sailing
```

[1]Este paso puede ser conflictivo, porque existen valores perdidos en la columna *Medal*.

[2]Si pudiésemos garantizar que el máximo es único, bastaría con ordenar los datos descendentemente y coger el primer valor, simplificando mucho la consulta. Este no es el caso y se recurre al cálculo explícito del máximo.

```
deportistas = FOREACH (
FILTER atletas BY Sport == 'Sailing' and Medal=='Gold')
GENERATE Name;
group_deportistas = GROUP deportistas BY Name;
deportistas_count = FOREACH group_deportistas
GENERATE group AS Name, COUNT(deportistas) AS Medallas;

-- calculo del maximo de medallas de oro
max_medallas = FOREACH (GROUP deportistas_count ALL)
GENERATE MAX(deportistas_count.Medallas) AS valor:int;

mejores = FOREACH (
FILTER deportistas_count BY Medallas == max_medallas.valor)
GENERATE Name, Medallas;
DUMP mejores;
```

Código 10: C3 para el mejor atleta en Hive

```
-- tabla auxiliar para contar medallas de oro en sailing

CREATE TEMPORARY TABLE  recuento_sailing AS
SELECT Name, COUNT(*) AS recuento FROM practicas.atletas
WHERE Medal = 'Gold' AND Sport = 'Sailing'
GROUP BY Medal, Name;

SELECT Name, recuento FROM recuento_sailing
WHERE recuento IN (SELECT MAX(recuento) FROM recuento_sailing);
```

Para la segunda parte, se seleccionan los valores distintos de las variables que contienen el año, ciudad y estación. Para esto, basta agrupar por las tres variables y generar las ternas de valores de cada grupo.

Código 11: C3 para las olimpiadas por continente en Pig

```
olimp_agrup = GROUP atletas BY (Year, Season, City);
olimpiadas = FOREACH olimp_agrup GENERATE
group.Year,
group.City,
group.Season;

DUMP olimpiadas;
```

Código 12: C3 para las olimpiadas por continente en Hive

```
SELECT DISTINCT Year, City, Season FROM practicas.atletas;
```

Cuarta consulta (C4)

La estrategia en esta consulta consiste en contar cuántas medallas tiene cada deportista (de oro, plata y bronce). Para realizar el recuento, se crean 3 columnas dummy, una para cada tipo de medalla. Cada columna tomará el valor 1 si el campo *medalla* toma ese valor y 0 en caso contrario. Una vez se tienen estas nuevas columnas, basta agrupar los datos por el nombre del atleta y sumar las tres columnas. Por último, para obtener el total de medallas se suman las tres columnas.

Código 13: C4, recuento de medallas en Pig

```
dummies = FOREACH atletas GENERATE
```

```
Name,
(CASE WHEN Medal == 'Gold' THEN 1 ELSE 0 END) AS Gold:int,
(CASE WHEN Medal == 'Silver' THEN 1 ELSE 0 END) AS Silver:int,
(CASE WHEN Medal == 'Bronze' THEN 1 ELSE 0 END) AS Bronze:int;
--(CASE WHEN Medal == 'NA' THEN 1 ELSE 0 END) AS NA:int;

dummies_group = GROUP dummies BY Name;
dummies_count = FOREACH dummies_group GENERATE
group as Name,
SUM(dummies.Gold) AS Medallas_oro:int,
SUM(dummies.Silver) AS Medallas_plata:int,
SUM(dummies.Bronze) AS Medallas_bronce:int;

total_medallas = FOREACH dummies_count GENERATE
Name,
Medallas_oro,
Medallas_plata,
Medallas_bronce,
--Medallas_NA,
(Medallas_oro+Medallas_plata+Medallas_bronce) AS Total;
```

Código 14: C4, recuento de medallas en Hive

```
-- tabla auxiliar para contar las medallas de cada atleta
CREATE TEMPORARY TABLE recuento_medallas AS
SELECT Name,
SUM(CASE WHEN Medal = 'Gold' THEN 1 ELSE 0 END)  AS Gold,
SUM(CASE WHEN Medal = 'Silver' THEN 1 ELSE 0 END) AS Silver,
SUM(CASE WHEN Medal = 'Bronze' THEN 1 ELSE 0 END) AS Bronze
FROM practicas.atletas
GROUP BY Name;

-- tabla auxiliar para contar el total de medallas por atleta
CREATE TEMPORARY TABLE recuento_medallas_total AS
SELECT *, Gold+Silver+Bronze AS Total FROM recuento_medallas;
```

Finalmente, para obtener el atleta con más medallas, se realiza un proceso similar al de la consulta C3, se calcula el máximo de la columna que almacena el total de medallas y se filtra por ese valor[3]. Para calcular el atleta con más medallas de oro, se realiza el mismo proceso anterior pero sobre la columna recuento de medallas de oro.

Código 15: C4 filtro por el deportista con más medallas en total y de oro en Pig

```
max_medallas = FOREACH (GROUP total_medallas ALL)
GENERATE MAX(total_medallas.Total) AS valor:int;

max_medallas_oro = FOREACH (GROUP total_medallas ALL)
GENERATE MAX(total_medallas.Medallas_oro) AS valor:int;

mejores = FOREACH (
FILTER total_medallas BY Total == max_medallas.valor)
GENERATE Name, Medallas_oro, Medallas_plata, Medallas_bronce;

mejores_oro = FOREACH (
```

[3]Nuevamente, no podemos garantizar que el máximo sea único y se recurre al cálculo explícito del máximo

```
FILTER total_medallas BY Medallas_oro == max_medallas_oro.valor)
GENERATE Name, Medallas_oro, Medallas_plata, Medallas_bronce;

DUMP mejores;
DUMP mejores_oro;
```

Código 16: C4 filtro por el deportista con más medallas en total y de oro en Hive

```
SELECT Name, Gold FROM recuento_medallas WHERE Gold IN (SELECT MAX(Gold)
   FROM recuento_medallas);

SELECT Name, Total FROM recuento_medallas_total WHERE Total IN (SELECT MAX
   (Total) FROM recuento_medallas_total);
```

Resultados

En esta sección se presenta una comparativa de tiempos de ejecución para cada consulta en cada una de las tecnologías y, en las consultas que aplique, la respuesta.

Los tiempos de ejecución de cada script se midieron usando el comando Linux *time*. En la Figura 1 se observan los resultados. Los tiempos más bajos se obtienen para Pig local. Sin embargo, los tiempos de ejecución para Pig mapreduce[4] y Hive no son significativamente distintos.

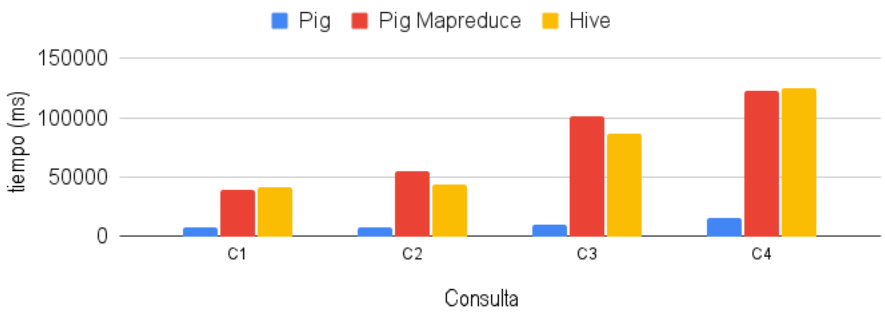

Figura 1: Gráfico de tiempos de ejecución

En la tabla 1 se muestra la cantidad de trabajos. En el primer valor los *map* y en el segundo los *reduce*. Se ve que Hive realiza la menor cantidad de tareas para cada consulta, seguido de pig Mapreduce y, finalmente, Pig local es el que más tareas realiza.

Tabla 1: Número de trabajos realizados.

Consulta	Pig local	Pig mapreduce	Hive
C1	(3, 1)	(2, 1)	(2, 1)
C2	(4, 2)	(3, 2)	(2, 2)
C3	(6, 3)	(5, 4)	(4, 3)
C4	(8, 4)	(6, 4)	(6, 3)

Las respuestas a las consultas son:

[4]El tiempo de ejecución en este caso se midió sin tener en cuenta el tiempo de almacenamiento de los resultados en HDFS

- **C3**: Charles Benedict Ainslie y Paul Bert Elvstrm. Ambos con cuatro medallas de oro.
- **C4**: Michael Phelps con 28 medallas, 23 de ellas de Oro. Es el mejor por ambos criterios, total de medallas y total de medallas de oro.

Conclusiones

A grandes rasgos, en esta práctica hay dos aspectos importantes a comparar entre Pig y Hive:

- La sintaxis para el desarrollo de consultas o creación de conjuntos de datos.
- Eficiencia en el cómputo de las consultas.

Respecto a la sintaxis, queda patente que Hive (HQL), por su gran similitud con SQL y legibilidad, es más accesible y cómodo para el público general. Sin embargo, Pig se demuestra más versátil que Hive, y podría integrarse en procesos *batch* para generar nuevos conjuntos de datos desde datos en bruto, al contrario que Hive, que está más orientado a *reporting* y datos ya estructurados. En este aspecto, se concluye que Pig es una herramienta para el entorno técnico mientras que Hive se centra en el entorno funcional de negocio.

Por otro lado, respecto a la eficiencia de cómputo, el tamaño de los datos no permite llegar a conclusiones claras. Pig local ha sido el más rápido, ya que sobre ficheros de pequeño tamaño, el acceso a disco es mucho más rápido a local que a HDFS. Respecto a Pig mapreduce y Hive, con este experimento no se puede afirmar que uno sea mas eficiente que otro. Sin embargo, no se espera que los tiempos de ejecución aumenten significativamente al variar el tamaño del archivo, algo que en Pig local sí cabe esperar.

Pese a que se ha realizado en modo *standalone*, no cabe duda que ambas tecnologías constituyen una herramienta muy potente para el tratamiento de datos masivos y, en particular, el punto fuerte es que consiguen dar acceso a un gran público al paradigma mapreduce.

Anexo

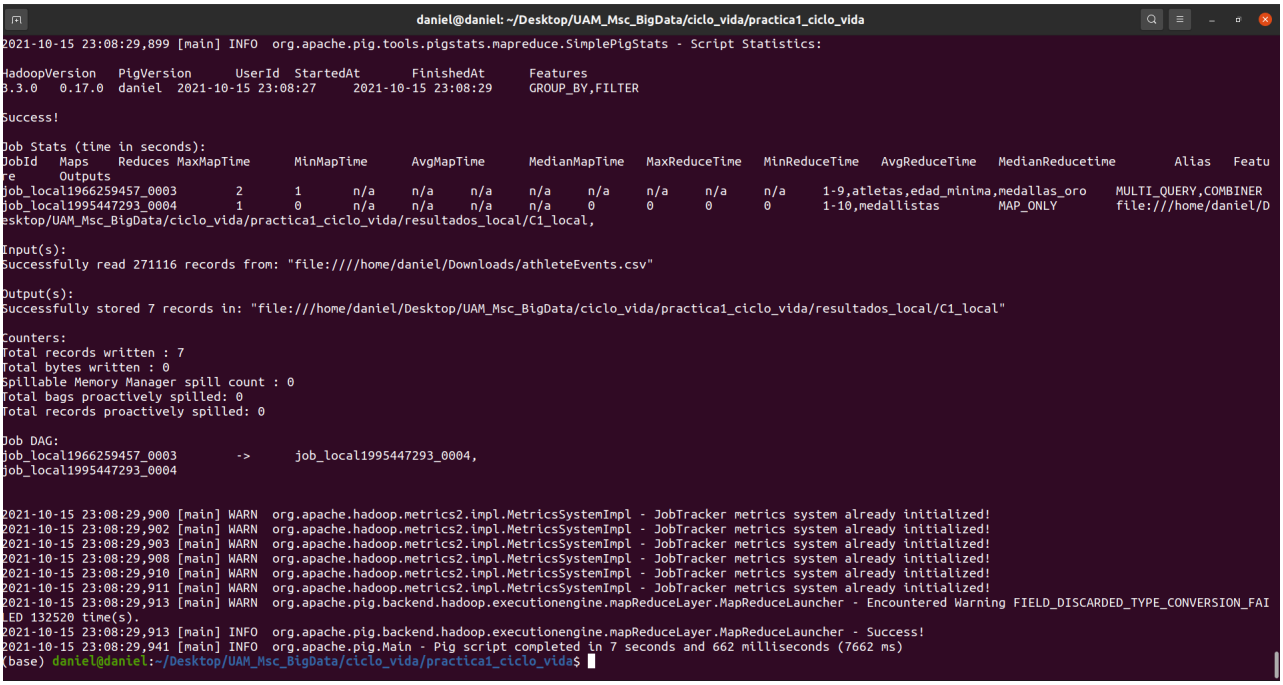

Figura 2: Salida por consola C1 pig local

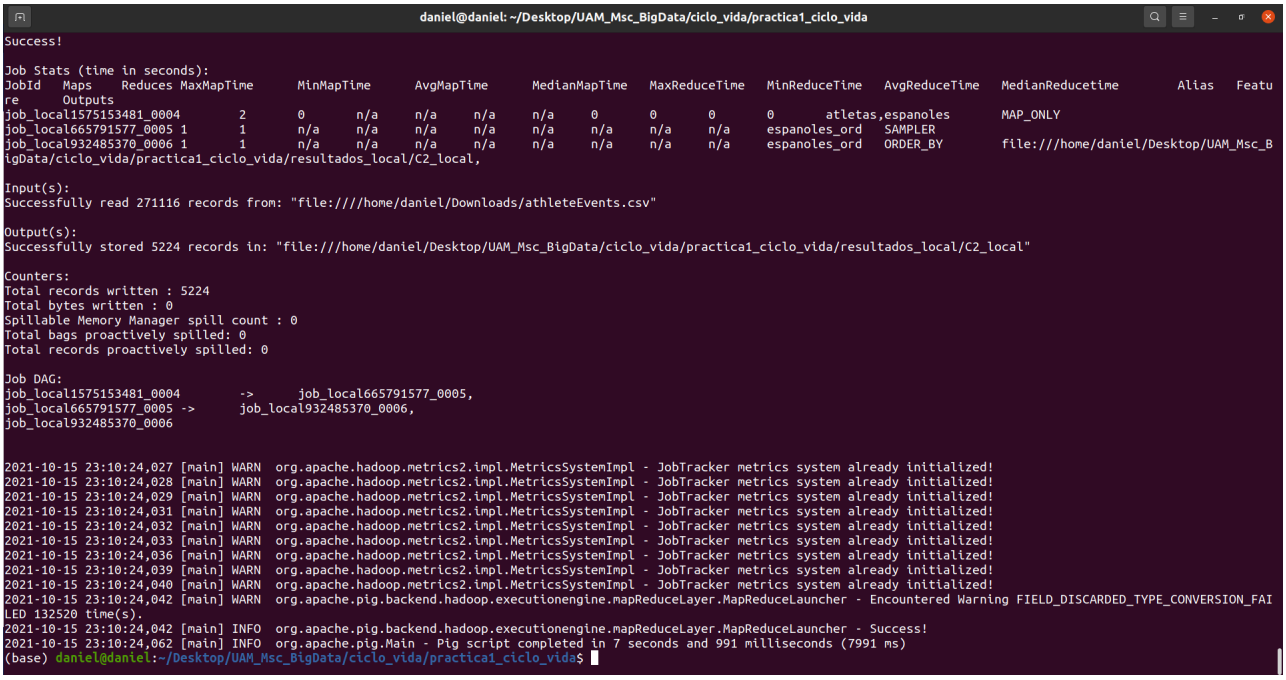

Figura 3: Salida por consola C2 pig local

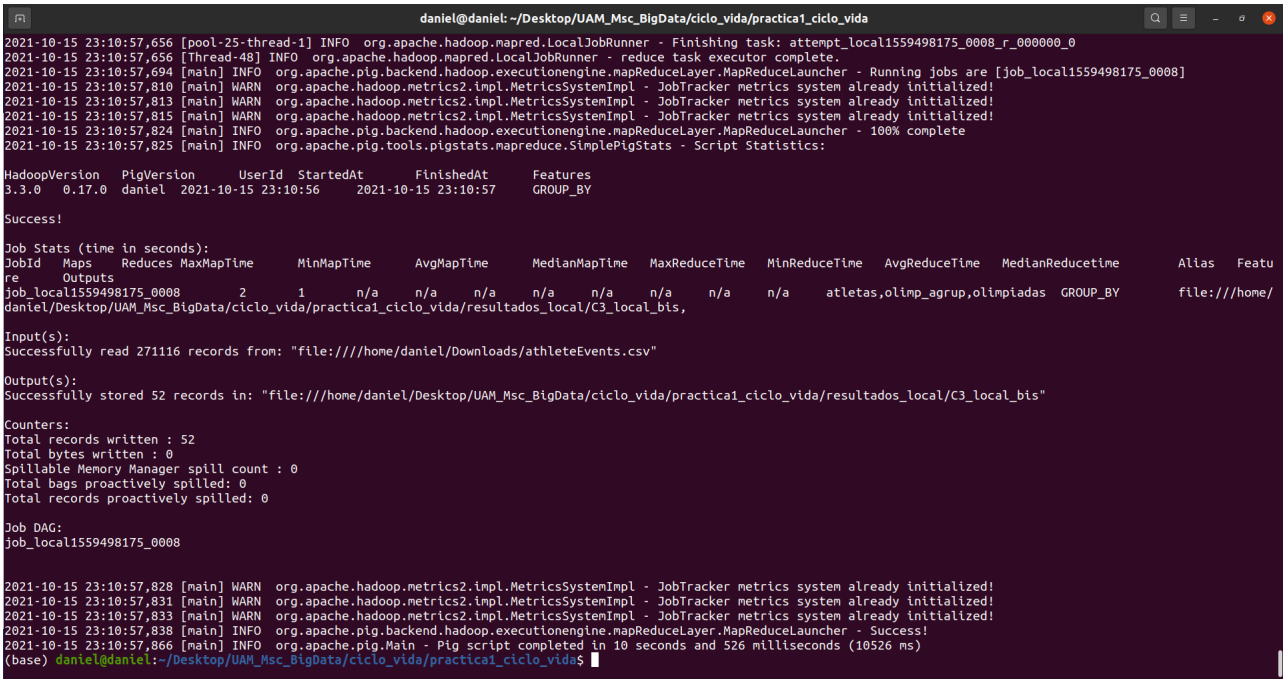

Figura 4: Salida por consola C3 pig local

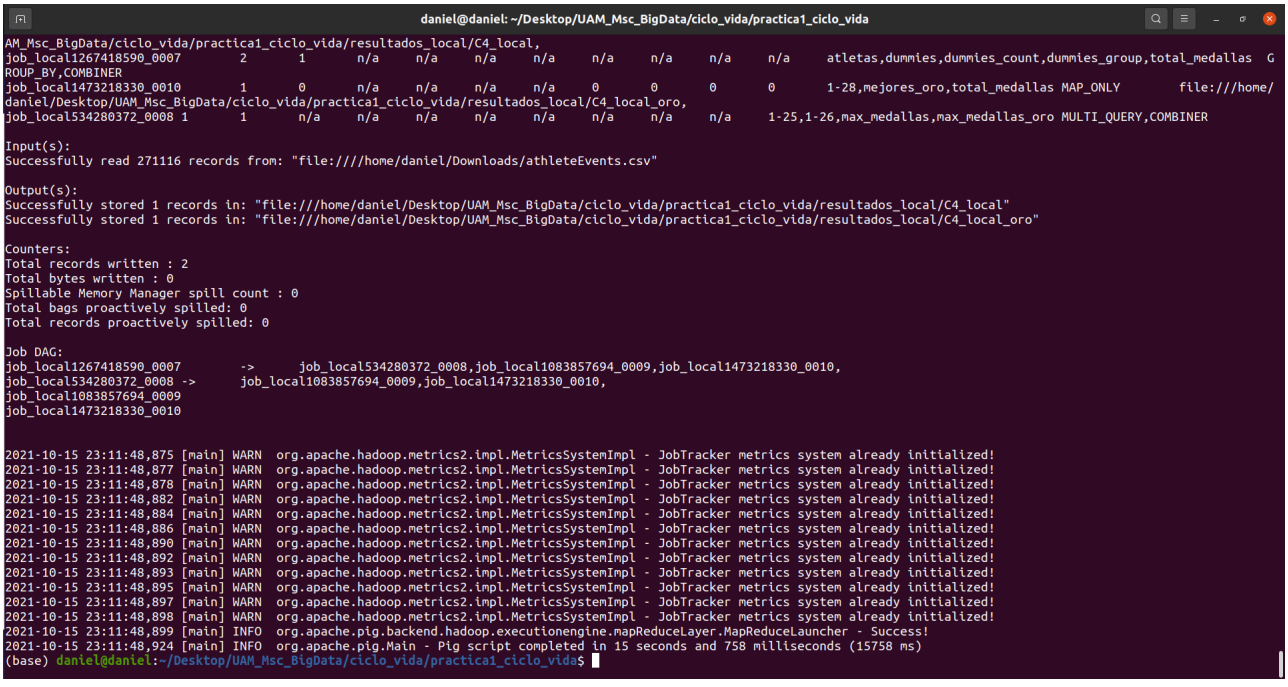

Figura 5: Salida por consola C4 pig local

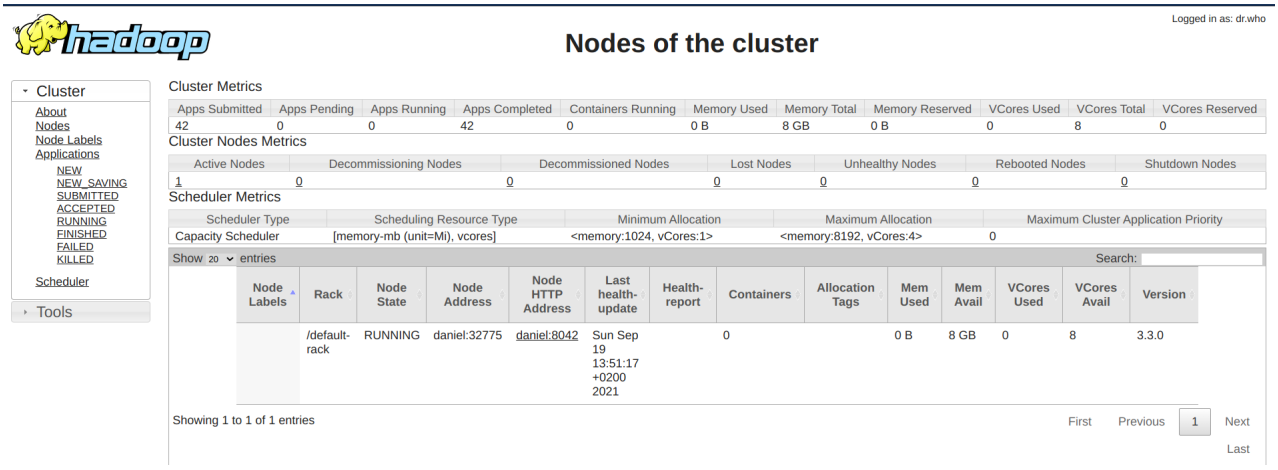

Figura 6: Resumen del servicio Hadoop funcionando correctamente

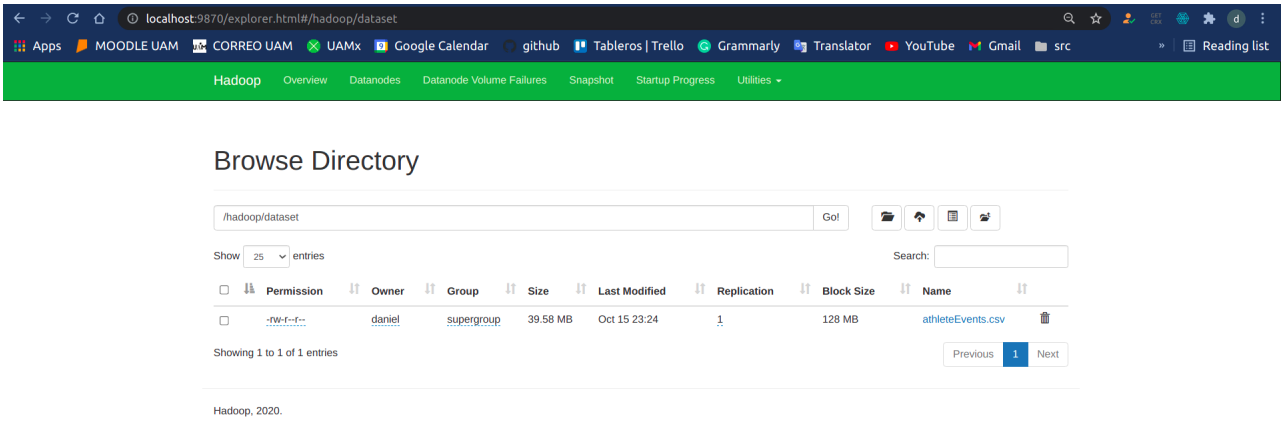

Figura 7: ruta a los datos en HDFS

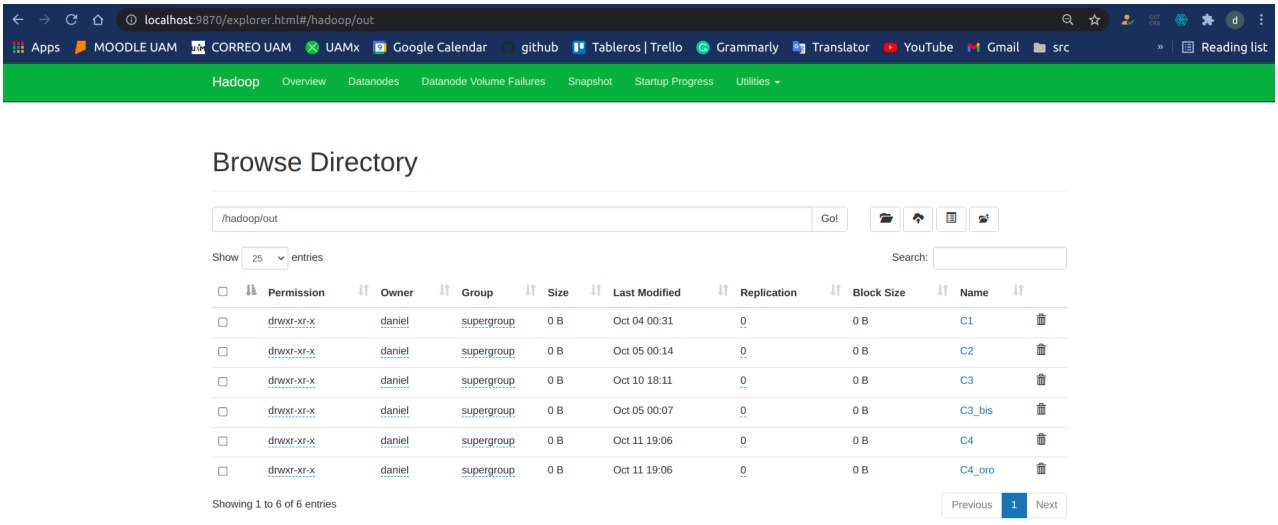

Figura 8: ruta a los resultados de PIG mapreduce en HDFS

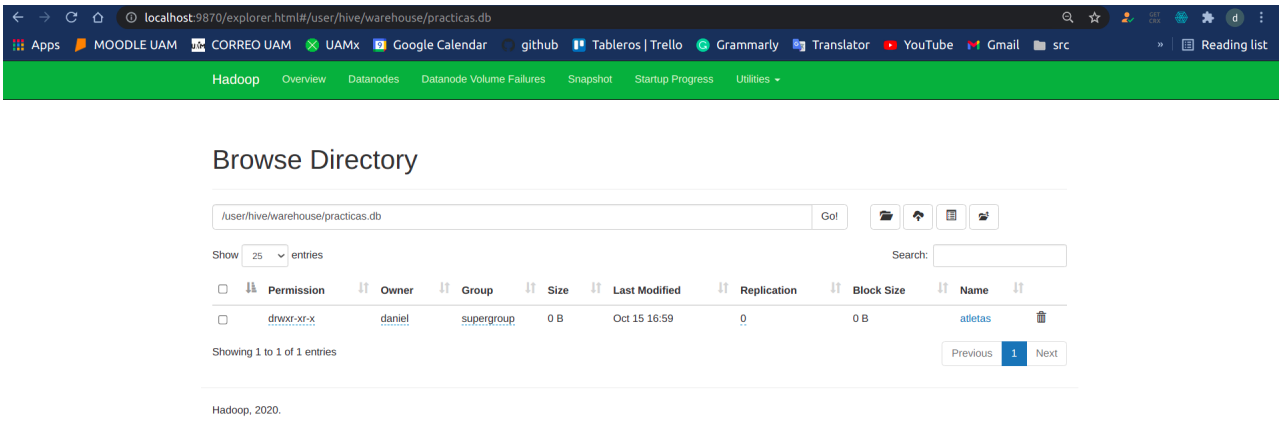

Figura 9: Contenido de la base de datos *practicas*

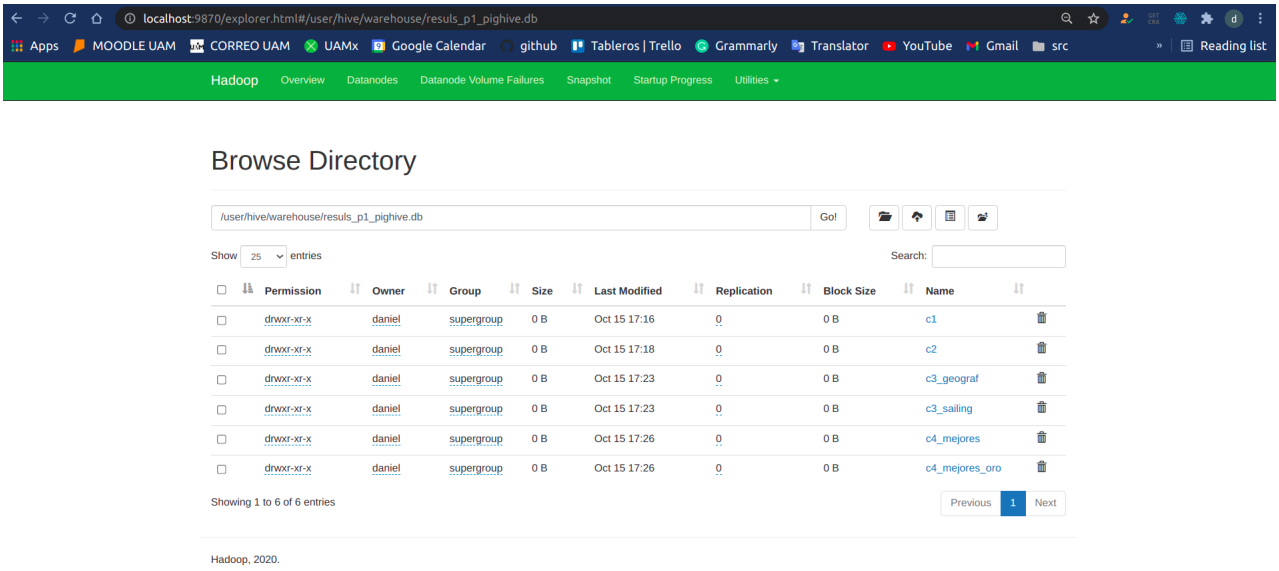

Figura 10: Contenido de la base de datos *resuls_p1_pighive*