

# **udp** FACULTAD DE INGENIERÍA Y CIENCIAS

**Actividad:**

**Informe Tarea 3 Sistemas Distribuidos**

---

Integrante: Diego Serrano  
Profesor: Nicolás Hidalgo

## Introducción

En el siguiente trabajo se tuvo la introducción a múltiples aplicaciones de análisis de recursos y patrones para cargas masivas de datos encontrados en los centros de datos de Google. Para esta oportunidad, se utilizarán herramientas provenientes de la aplicación Hadoop como HDFS, una herramienta muy similar a un sistema de archivos Unix que está optimizada para trabajar con enormes cantidades de datos, la herramienta Apache Pig, una herramienta compatible con Hadoop que está diseñada para generar programas que realicen transformaciones en los datos de entrada para producir datos de salida deseados, y la herramienta Apache Hive que consiste en un sistema de data warehousing diseñado para facilitar el análisis de conjuntos de datos masivos utilizando HiveQL, que es similar al lenguaje SQL. Se tiene que utilizar las herramientas anteriores con el objetivo de completar varias subtarefas que ayudarán en la realización de un análisis sobre datos provenientes de un archivo CSV, con tal de entender los patrones de carga y el uso de recursos existentes en Google.

## Desarrollo

Para dar inicio a esta actividad, es necesario desarrollar e instalar aplicaciones como Hadoop, Apache Pig y Apache Hive de tal forma que se pueda realizar las tareas propuestas. Para ello, se utilizó una imagen de Docker creada por suhothayan que se llama hadoop-spark-pig-hive, la cual cuenta con la versión 2.9.2 de Hadoop, la versión 0.17.0 de Apache Pig y la versión 2.3.5 de Apache Hive. Mediante esta imagen, es posible desarrollar el entorno de trabajo de la actividad con facilidad, ya que quita la necesidad de crear imágenes Docker para cada programa por separado que deben eventualmente levantarse mediante Docker Compose.

```
# hadoop
RUN curl -s http://www.eu.apache.org/dist/hadoop/common/hadoop-2.9.2/hadoop-2.9.2.tar.gz | tar -xz -C /usr/local/
RUN cd /usr/local && ln -s ./hadoop-2.9.2 hadoop

ENV HADOOP_PREFIX /usr/local/hadoop
RUN sed -i '/^export JAVA_HOME/ s:.*:export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64\nexport HADOOP_PREFIX=/usr/'
RUN sed -i '/^export HADOOP_CONF_DIR/ s:.*:export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop/:' $HADOOP_PREFIX/etc
#RUN . $HADOOP_PREFIX/etc/hadoop/hadoop-env.sh

RUN mkdir $HADOOP_PREFIX/input
RUN cp $HADOOP_PREFIX/etc/hadoop/*.xml $HADOOP_PREFIX/input
```

Figura 1: La presencia de Apache Hadoop en la imagen hadoop-spark-pig-hive

```
# pig
RUN curl -s http://apache.mirror.anlx.net/pig/pig-0.17.0/pig-0.17.0.tar.gz | tar -xz -C /usr/local
ENV PIG_HOME /usr/local/pig-0.17.0/
RUN ln -s $PIG_HOME /usr/local/pig
ENV PATH $PATH:$PIG_HOME/bin

# hive
RUN curl -s http://apache.mirror.anlx.net/hive/hive-2.3.5/apache-hive-2.3.5-bin.tar.gz | tar -xz -C /usr/local
ENV HIVE_HOME /usr/local/apache-hive-2.3.5-bin/
RUN ln -s $HIVE_HOME /usr/local/hive
ENV PATH $PATH:$HIVE_HOME/bin
```

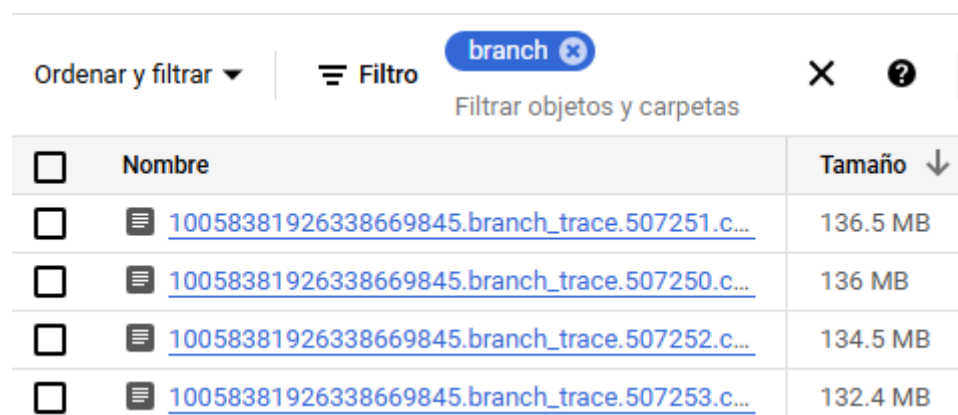
Figura 2: La presencia de Apache Hive y Apache Pig en la imagen hadoop-spark-pig-hive

Luego de encontrar la imagen para el entorno de trabajo, esta se puede levantar mediante el uso de un contenedor Docker, por lo que nos dirigimos a una terminal para correr el comando “docker pull suhothayan/hadoop-spark-pig-hive:2.9.2”, acción que se encarga de descargar la imagen seleccionada anteriormente en el ordenador, de tal forma que se puede trabajar con esta a través de la terminal. Con esto hecho, si se desea ingresar al entorno de trabajo, se puede utilizar el comando “docker run -it -p 50070:50070 -p 8088:8088 -p 8080:8080 suhothayan/hadoop-spark-pig-hive:2.9.2 bash” para inicializar un nuevo contenedor Docker en la imagen seleccionada, además de ingresar en el entorno una vez que se haya creado. Dentro de este entorno, es necesario crear una carpeta extra llamada “myfiles”, en donde se pueda almacenar todos los archivos que se reciban desde fuera del contenedor.

```
PS C:\Users\deser\Downloads\AnalisisDatos> docker run -it -p 50070:50070 -p 8088:8088 -p 8080:8080 suhothayan/hadoop-spark-pig-hive:2.9.2 bash
/etc/bootstrap.sh: line 9: /usr/local/spark/conf/spark-env.sh: Permission denied
/
* Starting OpenBSD Secure Shell server sshd
Waiting for hdfs to exit from safemode
Safe mode is OFF
Started
root@6f156f70a51f:/# exit
```

Figura 3: Acceso al contenedor Docker luego de ejecutar el comando “docker run”

Una vez desarrollado el entorno de trabajo en el contenedor Docker, se procede a acceder a la página Google Workload Traces para obtener un dataset CSV que contiene información sobre el uso de CPU, memoria y recursos en los centros de datos de Google. Es necesario destacar que se descargó el segundo archivo CSV más pesado del directorio delta/trace-1, donde se utilizó un filtro con la palabra “branch” y se ordenaron los datasets según tamaño para encontrar el archivo.



Ordenar y filtrar ▼	Filtro	Filtrar objetos y carpetas
<input type="checkbox"/>	<b>Nombre</b>	<b>Tamaño ↓</b>
<input type="checkbox"/>	<a href="#">10058381926338669845.branch_trace.507251.c...</a>	136.5 MB
<input type="checkbox"/>	<a href="#">10058381926338669845.branch_trace.507250.c...</a>	136 MB
<input type="checkbox"/>	<a href="#">10058381926338669845.branch_trace.507252.c...</a>	134.5 MB
<input type="checkbox"/>	<a href="#">10058381926338669845.branch_trace.507253.c...</a>	132.4 MB

Figura 4: Filtro realizado para encontrar el dataset en Google Cloud (Segundo archivo)

El archivo descargado se procede a descomprimir para extraer el dataset CSV, y se procede a renombrar el archivo a “branch\_traces.csv” con el objetivo de facilitar las referencias a este archivo en las subtarefas siguientes. Una vez realizado el cambio, se utiliza el comando “cp branch\_traces.csv [container\_ID]/myfiles” para crear una copia del dataset que pueda ser almacenada en la carpeta myfiles del contenedor Docker.

Se accede nuevamente al contenedor Docker utilizando el comando “docker start -i [container\_ID]”, y nos dirigimos a la carpeta myfiles para notar que el dataset ya se encuentra disponible en el contenedor. Se continua con el proceso utilizando el comando “hdfs dfs -put branch\_traces.csv /user/root”, acción que se encarga de nuevamente realizar una copia del dataset para que esta vez pueda ser almacenada en el sistema de archivos de Apache Hadoop, logrando así que las herramientas Apache Pig y Apache Hive sean capaces de trabajar con el dataset.

```
PS C:\Users\deser\Downloads\AnalisisDatos> docker cp branch_traces.csv 4004420b8b5a:/myfiles
Successfully copied 1.74GB to 4004420b8b5a:/myfiles
PS C:\Users\deser\Downloads\AnalisisDatos> docker start -i 4004420b8b5a
/etc/bootstrap.sh: line 9: /usr/local/spark/conf/spark-env.sh: Permission denied
/
* Starting OpenBSD Secure Shell server sshd
Waiting for hdfs to exit from safemode
Safe mode is OFF
Started
root@4004420b8b5a:/#
```

Figura 5: Copia del archivo branch\_traces.csv a la carpeta “myfiles” del contenedor Docker

```
root@8b29d5e97132:/myfiles# ls
branch_traces.csv  derby.log  hiveTable1.sql  metastore_db  pigAnalyze2.pig  pigAnalyze3.pig
root@8b29d5e97132:/myfiles# hdfs dfs -ls /user/root
Found 2 items
-rw-r--r--  1 root hadoop    1740460837 2024-06-23 04:33 /user/root/branch_traces.csv
drwxr-xr-x  - root supergroup          0 2019-07-21 16:09 /user/root/input
root@8b29d5e97132:/myfiles#
```

Figura 6: Archivo branch\_traces.csv en la carpeta “myfiles” y en el sistema de archivos HDFS

Una vez realizado este proceso, se puede comenzar a utilizar Apache Pig para conseguir un análisis exploratorio inicial del dataset. Para ello, se diseñó un script en PigLatin que se encarga de cargar el dataset utilizando PigStorage, dejando todas las columnas en formato “chararray” ya que no se sabe a qué tipos de datos pertenece cada una de las columnas en el dataset, y luego se muestra en pantalla las primeras 25000 filas del archivo para tener una buena idea de cómo funciona la estructura de los datos, así como también identificar los patrones que sigue el dataset en términos básicos.

```
AnalisisDatos > pigAnalyze.pig
1  archivo = load '/user/root/branch_traces.csv' using PigStorage(';')
2  AS (field1:chararray, field2:chararray, field3:chararray, field4:chararray);
3
4  limitedData = LIMIT archivo 25000;
5
6  dump limitedData;
7  explain limitedData;
8  describe limitedData;
```

Figura 7: Script en PigLatin para realizar el primer análisis del dataset

Se procedió a ejecutar el script en PigLatin utilizando la línea de comando “pig pigAnalyze.pig” y se obtuvieron los siguientes resultados en pantalla:

```
(0x1b25c11,conditional_jump,0,0x1b15396,,,)  
(0x1b25c1b,conditional_jump,0,0x1b15390,,,)  
(0x1b25c29,conditional_jump,0,0x1b25be4,,,)  
(0x1b25c41,conditional_jump,0,0x1b15396,,,)  
(0x1b25c4e,conditional_jump,0,0x1b153f6,,,)  
(0x1b25c5e,direct_call,1,0x1537c20,,,)  
(0x1537c36,conditional_jump,0,0x1537ca7,,,)  
(0x1537c4f,conditional_jump,0,0x1537ca7,,,)  
(0x1537c5a,conditional_jump,0,0x1af5bd3,,,)  
(0x1537c76,conditional_jump,0,0x1af5bbd,,,)  
(0x1537c99,conditional_jump,0,0x1af5bbd,,,)  
(0x1537ca6,return,1,0x1b25c63,,,)  
(0x1b25c70,conditional_jump,0,0x1b1540a,,,)  
(0x1b25c7a,conditional_jump,0,0x1b1542d,,,)  
(0x1b25c98,return,1,0x11974c7,,,)  
(0x11974ca,conditional_jump,0,0x18f7359,,,)  
(0x11974e0,conditional_jump,0,0x18f7295,,,)  
(0x11974fd,conditional_jump,0,0x18f7295,,,)  
(0x1197519,conditional_jump,0,0x18f72af,,,)  
(0x119753c,conditional_jump,0,0x18f72af,,,)  
(0x1197545,conditional_jump,0,0x18f72c5,,,)  
(0x119754e,direct_call,1,0x1b251e0,,,)  
(0x1b251f7,direct_call,1,0x14e5580,,,)  
(0x14e558e,conditional_jump,0,0x1a5e265,,,)  
(0x14e55a0,conditional_jump,0,0x1a5e265,,,)  
(0x14e55ac,return,1,0x1b251fc,,,)  
(0x1b25206,conditional_jump,0,0x1b14b6c,,,)  
(0x1b25225,direct_call,1,0x14e5540,,,)  
(0x14e5555,conditional_jump,0,0x1a5e25e,,,)  
(0x14e5567,conditional_jump,0,0x1a5e25e,,,)  
(0x14e556d,return,1,0x1b2522a,,,)  
(0x1b25237,return,1,0x1197553,,,)  
(0x119755d,direct_call,1,0x3ad7720,,,)  
(0x3ad7720,indirect_jump,1,0x7f492702ada0,,,)  
(0x7f492702adbfc,conditional_jump,0,0x7f492702adc2,,,)
```

Figura 8: Fragmento de los datos impresos en pantalla con el script en PigLatin

A través de esta imagen es posible observar varias características con respecto al dataset que se está trabajando para esta actividad. En primer lugar, se puede notar que los datos de la primera y cuarta columna del archivo tienen una estructura similar, ya que ambos corresponden a números hexadecimales que tienen la posibilidad de repetirse en ciertas filas, especialmente los datos de la cuarta columna, lo que sugiere que estas columnas representan direcciones de branch. Por otro lado, la segunda columna presenta varios nombres de acciones que parecen ser realizadas por la dirección de la primera columna, indicando acciones como saltos condicionales e indirectos, llamadas directas, retornos, entre otros nombres, sugiriendo con esta información que la segunda columna corresponde al tipo de branch que pertenece el branch con dirección hexadecimal en la primera columna. Finalmente se tiene la tercera columna, en donde se puede ver que los datos almacenados corresponden exclusivamente a valores de 0 o 1, indicando de cierta manera que la tercera columna cumple la acción de un booleano, donde 0 puede tratarse de un “no”, y el 1 se refiere a un “sí”. Otras características a destacar en la tercera columna es que para todo tipo de branch que sea distinto a “conditional\_jump”, el valor que posee la tercera columna en dichas filas siempre va a ser igual a “1”, sin embargo aquellas filas que poseen un branch de tipo “conditional\_jump” no necesariamente cuentan con un valor binario de tipo 0.

Una vez finalizado el primer análisis exploratorio del dataset a través de la herramienta Apache Pig, es posible proceder en el desarrollo con la herramienta Apache Hive, ahora que ya se comprende la estructura que posee el dataset CSV. Para ello, fue necesario moverse al directorio “usr/local/hive/bin” del contenedor Docker, ya que de otra forma el programa es incapaz de procesar secuencias HiveQL. Una vez se accede al programa usando “hive” en la terminal, se escriben las secuencias HiveQL del archivo “hiveTable1.txt”, la cual se encarga de crear una tabla con 4 columnas utilizando las comas del archivo CSV como delimitadores de cada columna. Una vez creada la tabla, se inserta branch\_traces.csv dentro de la tabla y se presentan en pantalla las primeras 25000 filas del dataset.

```
AnalisisDatos > ≡ hiveTable1.txt
1 CREATE TABLE CSVData (field1 STRING, field2 STRING, field3 INT, field4 STRING)
2
3 ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
4
5 LOAD DATA INPATH '/user/root/branch_traces.csv' INTO TABLE CSVData;
6
7 SELECT * FROM CSVData LIMIT 25000;
```

Figura 9: Secuencias HiveQL para crear la tabla con la estructura del dataset

Ejecutar dichas líneas en la terminal de Hive causaron la obtención de los siguientes resultados en pantalla:

```
hive> CREATE TABLE CSVData (field1 STRING, field2 STRING, field3 INT, field4 STRING)
>
> ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
OK
Time taken: 0.689 seconds
hive> LOAD DATA INPATH '/user/root/branch_traces.csv' INTO TABLE CSVData;
Loading data to table default.csvdata
OK
Time taken: 0.81 seconds
hive> █
```

Figura 10: Resultados obtenidos al cargar las secuencias HiveQL en Hive



0x14e5a1b	conditional_jump	0	0x14e5b05
0x14e5a29	conditional_jump	0	0x1a5e6fe
0x14e5a3d	conditional_jump	0	0x1a5e78b
0x14e5a67	direct_call	1	0x18bcf40
0x18bcf47	conditional_jump	0	0x18bcfd2
0x18bcf70	conditional_jump	0	0x18bcfee
0x18bcfba	conditional_jump	0	0x18bcffc
0x18bcfd1	return	1 0x14e5a6c	
0x14e5a76	conditional_jump	0	0x1a5e755
0x14e5ad8	conditional_jump	0	0x14e5ace
0x14e5b04	return	1 0x14e5851	
0x14e5862	conditional_jump	0	0x14e5870
0x14e586f	return	1 0x14d0989	
0x14d09c7	return	1 0x14cc594	
0x14cc5a0	direct_call	1	0x18bcf40
0x18bcf47	conditional_jump	0	0x18bcfd2
0x18bcf70	conditional_jump	0	0x18bcfee
0x18bcfba	conditional_jump	0	0x18bcffc
0x18bcfd1	return	1 0x14cc5a5	
0x14cc5c6	conditional_jump	0	0x14cf744
0x14cc60a	conditional_jump	0	0x1a215b3
0x14cc631	conditional_jump	0	0x14ce615
0x14cc646	conditional_jump	0	0x14cf73a
0x14cc64f	direct_call	1	0x14d09e0
0x14d09ea	conditional_jump	0	0x1a2cbc2
0x14d09f7	conditional_jump	0	0x14d0a18
0x14d0a05	conditional_jump	0	0x1a2cb61
0x14d0a12	conditional_jump	0	0x1a2cb13
0x14d0a25	direct_call	1	0x14e5820
0x14e5833	conditional_jump	0	0x1a5e604
0x14e584c	direct_call	1	0x14e5a00
0x14e5a1b	conditional_jump	0	0x14e5b05
0x14e5a29	conditional_jump	0	0x1a5e6fe
0x14e5a3d	conditional_jump	0	0x1a5e78b
0x14e5a67	direct_call	1	0x18bcf40
0x18bcf47	conditional_jump	0	0x18bcfd2
0x18bcf70	conditional_jump	0	0x18bcfee

Figura 11: Fragmento de los datos impresos con la secuencia SELECT \* FROM CSVData

## Análisis

Una vez implementada la tabla con la estructura del dataset, se procede a realizar las operaciones de filtrado específicas que permitirán identificar las características que tiene el dataset seleccionado al inicio de la actividad, presentando datos como el número total de registros que tiene el dataset, la frecuencia de cada tipo de branch utilizando Pig y Hive, la relación y proporción entre los tipos de branch y el valor de “taken” (0 o 1), entre otros.

Para el primer caso, se implementó un script en PigLatin llamado “pigAnalyze2.pig”, el cual se encarga de cargar el dataset almacenado en HDFS mediante PigStorage para luego contar cada una de las filas del archivo y finaliza con la impresión del número de filas existentes en el dataset.

```
AnalisisDatos > ≡ pigAnalyze2.pig
1  #Script para contar el numero de filas en el dataset
2
3  archivo = load '/user/root/branch_traces.csv' using PigStorage(';')
4  AS (field1:chararray, field2:chararray, field3:chararray, field4:chararray);
5
6  conteo = FOREACH (GROUP archivo ALL) GENERATE COUNT(archivo);
7
8  DUMP conteo;
```

Figura 10: Script en PigLatin para contar el número de filas del dataset

Sin embargo, al momento de ejecutar el script en la terminal del contenedor Docker, no se obtuvieron los resultados esperados en pantalla, ya que el proceso nunca terminó de completarse a causa de que la ejecución del script se detuvo luego de mantenerse en ejecución por una cierta cantidad de tiempo. Esto puede deberse a dificultades técnicas causadas por el tamaño del dataset elegido para la actividad, el cual resulta ser demasiado grande para la capacidad que tiene el computador utilizado, limitando así la posibilidad de conocer el tamaño de archivo de la manera deseada. Es por este motivo que se optó por abrir el archivo manualmente para comprobar el número de filas que contiene el dataset CSV, de tal manera que se pudo observar lo siguiente:

```
45675043 0x7f4926fc6dbd,conditional_jump,0,0x7f4926fc70d0
45675044 0x7f4926fc6dd2,conditional_jump,0,0x7f4926fc6e10
45675045 0x7f4926fc6de2,conditional_jump,1,0x7f4926fc7090
45675046 0x7f4926fc7096,conditional_jump,0,0x7f4926fc70d0
45675047 0x7f4926fc709b,return,1,0x119731a
45675048 0x119734b,direct_call,1,0x1197440
45675049 0x1197467,direct_call,1,0x3ad7720
45675050 0x3ad7720,indirect_jump,1,0x7f492702ada0
45675051
```

Figura 11: Revisión manual del archivo branch\_traces.csv

Es mediante este método que se puede comprobar que el archivo cuenta con un número de 45675048 registros con información, justificando el por qué el computador tiene tantas dificultades cargando información dentro del dataset.



Se intentó realizar el mismo caso implementando una consulta HiveQL en Apache Hive del tipo “SELECT COUNT(\*) FROM CSVData” para comprobar si este programa presentaba un resultado distinto al contar las filas del dataset cuando la información está almacenada en una tabla. Los resultados obtenidos se pueden apreciar a continuación:

```
hive> SELECT COUNT(*) FROM CSVData;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different e
z) or using Hive 1.X releases.
Query ID = root_20240623163149_fac7f54b-2f10-41d8-a8df-840a7e3c0cad
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1719160079476_0001, Tracking URL = http://8b29d5e97132:8088/proxy/application_1719160079476_0001/
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_1719160079476_0001
Hadoop job information for Stage-1: number of mappers: 7; number of reducers: 1
2024-06-23 16:32:00,598 Stage-1 map = 0%, reduce = 0%
2024-06-23 16:33:03,442 Stage-1 map = 0%, reduce = 0%
2024-06-23 16:33:59,565 Stage-1 map = 5%, reduce = 0%, Cumulative CPU 50.72 sec
2024-06-23 16:34:03,170 Stage-1 map = 86%, reduce = 0%, Cumulative CPU 83.02 sec
2024-06-23 16:35:03,529 Stage-1 map = 86%, reduce = 0%, Cumulative CPU 83.02 sec
2024-06-23 16:36:03,759 Stage-1 map = 86%, reduce = 0%, Cumulative CPU 83.02 sec
2024-06-23 16:37:03,974 Stage-1 map = 86%, reduce = 0%, Cumulative CPU 83.02 sec
2024-06-23 16:38:04,147 Stage-1 map = 86%, reduce = 0%, Cumulative CPU 83.02 sec
2024-06-23 16:39:04,326 Stage-1 map = 86%, reduce = 0%, Cumulative CPU 83.02 sec
2024-06-23 16:40:04,445 Stage-1 map = 86%, reduce = 0%, Cumulative CPU 83.02 sec
2024-06-23 16:41:04,593 Stage-1 map = 86%, reduce = 0%, Cumulative CPU 83.02 sec
2024-06-23 16:42:04,723 Stage-1 map = 86%, reduce = 0%, Cumulative CPU 83.02 sec
2024-06-23 16:43:04,848 Stage-1 map = 86%, reduce = 0%, Cumulative CPU 83.02 sec
2024-06-23 16:44:04,981 Stage-1 map = 86%, reduce = 0%, Cumulative CPU 83.02 sec
2024-06-23 16:44:45,890 Stage-1 map = 0%, reduce = 0%
2024-06-23 16:45:46,844 Stage-1 map = 0%, reduce = 0%
2024-06-23 16:46:47,788 Stage-1 map = 0%, reduce = 0%
2024-06-23 16:47:47,898 Stage-1 map = 0%, reduce = 0%
```

Figura 12: Resultado obtenido al contar el número de filas de la tabla CSVData

Se puede observar en la imagen que, similar al caso de Apache Pig, intentar contar el número de filas existentes en la tabla CSVData mediante Apache Hive resulta en la consulta cayéndose luego de permanecer en un 86% de progreso en el proceso de mapper de la herramienta por más de 10 minutos, lo que lleva a que el progreso en la consulta regrese a 0%, sin presentar cambios significativos desde ese registro en pantalla. Lo anterior permite reafirmar que existen dificultades técnicas que prohíben al computador procesar secuencias que requieren demasiada memoria RAM.

## **Preguntas de la Entrega**

### **1. ¿Qué ventajas ofrece Apache Pig en comparación con escribir código MapReduce directamente para el procesamiento de grandes volúmenes de datos?**

Apache Pig se presenta como un mejor programa de procesamiento de grandes volúmenes de datos debido a que provee una larga lista de operadores que ayudan en el procesamiento de datasets que contienen demasiada información como filtros, joining, aggregation, entre otros. Además de que resulta ser más fácil de aprender y utilizar, ya que se pueden realizar operaciones de consultas de datos a través de un único archivo de tipo PigLatin, el cual puede cargar la información de un archivo en un número de líneas de código considerablemente menores, sin necesidad de contar con otros archivos o con la necesidad de implementar un archivo jar para ejecutar estos procesos.

### **2. ¿Cómo facilita Apache Hive la consulta y análisis de grandes conjuntos de datos en Hadoop, y en qué situaciones es preferible usar Hive en lugar de Pig?**

Si bien la carga de enormes cantidades de datos en Apache Hive es considerablemente más lento que realizar el mismo proceso en Apache Pig, esta herramienta cuenta con la ventaja de almacenar la información procesada anteriormente en tablas, de tal forma que si se desea consultar por más información sobre un dataset, no es necesario tener que cargar la misma información desde cero, ya que esta se encontrará disponible bajo el nombre que se le haya asignado a la tabla que almacena la información del dataset, lo que significa que Apache Hive puede resultar conveniente para situaciones donde se necesita consultar por múltiples tipos de datos en períodos cortos.

### **3. Explica cómo utilizaste HiveQL para realizar análisis avanzados sobre los Google Workload Traces ¿Qué consultas resultaron ser las más útiles y por qué?**

### **4. Investiga y describe un caso real de uso de Apache Pig y Hive en la industria ¿Cómo estas herramientas han ayudado a resolver problemas específicos de análisis de datos?**

Si se consideran casos reales de uso de las herramientas anteriores, se puede mencionar que la herramienta Apache Hive fue desarrollada y utilizada por Facebook con el objetivo de consultar sobre sus data lakes compuestos por varios petabytes de información almacenados en HDFS y Amazon S3. Dichos datos se podían consultar mediante el uso de un lenguaje similar a SQL, llamado HiveQL, el cual facilitaba el análisis de esta enorme cantidad de información.

Por otro lado, Apache Pig fue una herramienta desarrollada por Yahoo! y eventualmente fue utilizada por otras aplicaciones como Twitter, la cual utiliza esta tecnología para analizar grandes conjuntos de datos para obtener información y tomar decisiones con respecto al comportamiento de los usuarios al momento de realizar tweets en la plataforma.

**5. ¿Qué otras herramientas o tecnologías, además de Pig y Hive, podrías haber utilizado para analizar los Google Workload Traces, y que ventajas o desventajas tendrían?**

Si se habla de aplicaciones que tienen un nivel de popularidad similar a Pig y Hive, se puede considerar a la herramienta Apache Spark como analizador de los Google Workload Traces. Esta herramienta está diseñada para este tipo de trabajos ya que esta apuesta por la velocidad de respuesta, siendo capaz de ejecutar considerablemente más rápido que el MapReduce de Hadoop, además de que la utilización de esta herramienta es relativamente fácil, permitiendo al usuario escribir servicios en lenguajes como Python, Scala, entre otros. Sin embargo, la desventaja más notoria de esta herramienta es el hecho de que no cuenta con un motor de memoria integrado, lo que significa que esta depende de muchos elementos compartidos, sin mencionar que se necesita un alto uso de memoria RAM para el procesamiento de consultas en Apache Spark.

**6. Compara y contrasta el uso de Apache Pig y Hive en términos de facilidad de uso, rendimiento y casos de uso específicos. ¿Cuál prefieres y por qué?**

En términos de facilidad de uso, se puede destacar a Apache Pig que el desarrollo de los scripts en PigLatin es relativamente sencillo, requiriendo menos de 10 líneas de ejecución para la mayoría de las consultas que se realizan en conjuntos de datos enormes, además de que en el desarrollo de la actividad, Apache Pig otorgó respuestas mucho más rápidas y efectivas en comparación a lo que se realizó en Apache Hive.

Por otro lado, se tiene que resaltar en Apache Hive que resulta muy conveniente la implementación de tablas distribuidas como método de almacenamiento de memoria, luego de haber sido procesada en una consulta HiveQL inicial, facilitando así el desarrollo de otras consultas que se necesiten realizar en el dataset en un período corto de tiempo y que no requieran de la solicitud de muchos datos en pantalla. Es por estas razones que en el caso personal, se ha elegido a Apache Pig como la herramienta de procesamiento preferida, ya que fue mucho más consistente al momento de entregar resultados en pantalla en comparación a los resultados obtenidos en Apache Hive.

**7. Investiga y describe cómo podrías optimizar el rendimiento de las consultas en Hive. ¿Qué prácticas recomendadas existen para mejorar la eficiencia de las consultas?**

Las prácticas más comunes y eficientes al momento de tratar con datasets como los trabajados en la actividad anterior son el particionamiento de tablas, que consiste en la división de tablas en partes más pequeñas basadas en una o más columnas, de tal forma que se acelera el acceso a datos y provee la posibilidad de performar operaciones en datasets más pequeños. Otra práctica muy común para mejorar la eficiencia de Hive es el bucketing, una técnica que permite dividir los datos de una tabla en archivos más manejables, lo cual permite mejorar el rendimiento de las consultas HiveQL en comparación a consultas que no realizan este truco.

**8. ¿Qué son las UDF (User Defined Functions) en el contexto de Apache Pig y Hive, y cómo podrían haber sido útiles en esta tarea?**

User Defined Functions corresponden a funciones personalizadas que permiten a los usuarios definir sus propias funciones para realizar tareas específicas que no se pueden desarrollar fácilmente en una secuencia SQL. Teniendo en cuenta esto, las UDF pudieron resultar mucho más útiles de implementar para esta tarea, ya que tanto en Apache Pig como en Apache Hive se necesita realizar grandes y complejas secuencias de comandos HiveQL o scripts PigLatin con el fin de conseguir información que es relativamente pequeña, y que en UDF podría solucionarse mediante la implementación de programas sencillos y cortos, sin tener que generar comandos complejos.

**9. Discute la importancia de la visualización de datos en el análisis de grandes conjuntos de datos. ¿Qué herramientas podrías utilizar para visualizar los resultados obtenidos de Pig y Hive?**

Teniendo en cuenta los requerimientos de la tarea, una herramienta que resulta esencialmente conveniente para la visualización de resultados obtenidos en pantalla que es compatible tanto con Apache Pig como Apache Hive, es la herramienta Apache Zeppelin. Esta herramienta corresponde a un servidor que es capaz de publicar las visualizaciones sobre fuentes de datos Big Data, el cual permite escribir y ejecutar scripts/consultas, para luego visualizar los datos obtenidos en formatos como tablas, gráficos y diagramas, de tal forma que facilita el desarrollo de un análisis de información en conjuntos de datos masivos.