TAREA 3 SISTEMAS DISTRIBUIDOS

JORGE ALFREDO TORO MACIAS

Introducción

Apache Hadoop es un entorno de trabajo para aplicaciones distribuidas bajo el contexto de grandes volúmenes de datos. Construidas sobre Hadoop existen dos infraestructuras para el trabajo de agrupación, consulta y análisis de datos: Apache Hive y Apache Pig. Ambas plataformas procesan los datos bajo el concepto del modelo de programación MapReduce.

MapReduce es un modelo de programación desarrollado para el procesamiento y generación de grandes volúmenes de datos en un clúster de computadores. La idea principal es dividir el procesamiento de los datos en dos fases: Map (mapeo) y Reduce (reducción). Estas fases se ejecutan de manera distribuida en múltiples nodos de un clúster, lo que permite procesar grandes cantidades de datos de manera eficiente y escalable.

En la fase de mapeo los datos de entrada se dividen en fragmentos más pequeños y se procesan de manera independiente en los diferentes nodos del clúster. Cada nodo realiza una función de mapeo que toma un conjunto de datos y los transforma en pares clave-valor. El resultado de esta fase es una lista de pares clave-valor intermedios.

Antes de pasar a la fase de Reducción, existe una fase de transición donde los pares clave-valor intermedios se agrupan y se ordenan según la clave para prepararlos.

En la fase de Reduce cada nodo toma los datos intermedios que tienen la misma clave y los combina en un conjunto más pequeño de pares clave-valor, produciendo así el resultado final.

A continuación se estudiará y evidenciará el funcionamiento de Apache Hadoop y sus servicios derivados. Se utilizarán datos tomados de Google Workload Traces, los cuales corresponden a datos de millones de procesos de distintas aplicaciones dentro de Google. Estos serán estudiados utilizando las herramientas de Apache Hive y Apache Pig para el procesamiento y análisis de datos.

Explicación de los códigos

En primera instancia se deben levantar los servicios a utilizar: Hive y Pig. Para esto se tomó una imagen que contenía los dos servicios Hadoop.

Build the image

```
docker build  -t suhothayan/hadoop-spark-pig-hive:2.9.2 .
```

Pull the image

```
docker pull suhothayan/hadoop-spark-pig-hive:2.9.2
```

Start a container

In order to use the Docker image you have just build or pulled use:

```
docker run -it -p 50070:50070 -p 8088:8088 -p 8080:8080 suhothayan/hadoop-spark-pig-hive:2.9.2
```

Se siguieron los pasos indicados por el autor para levantar la imagen.

```
                                        NAMES
58d8a9d8e625   suhothayan/hadoop-spark-pig-hive:2.9.2   "/etc/bootstrap.sh -…"   8 minutes ago   Up 8 minutes   8030-8033/tcp, 8040/tcp, 8042/tcp, 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp, 8081/tcp, 9000/tcp, 13562/tcp, 40661/tcp, 50010/tcp, 0.0.0.0:8088->8088/tcp, :::8088->8088/tcp, 50020/tcp, 50075/tcp, 50090/tcp, 0.0.0.0:50070->50070/tcp, :::50070->50070/tcp   peaceful_nobel
[ale@fedora Descargas]$ docker exec -it peaceful_nobel /bin/bash
root@58d8a9d8e625:/#
```

Una vez se descarga la imagen procedemos a ejecutarla y abrir una shell de bash dentro de ella para el uso de comandos. Se evidencia el correcto funcionamiento cuando ahora somos el usuario ‘root’ dentro del contenedor corriendo.

```
Copy files/folders between a container and the local filesystem
[ale@fedora Descargas]$ docker cp branch.csv 58d8a9d8e625:/root/
Successfully copied 4.26GB to 58d8a9d8e625:/root/
[ale@fedora Descargas]$
```

Con el dataset a utilizar ya descargado y dentro del directorio “Descargas” se procede a copiarlo al contenedor para su posterior copia al sistema HDFS. Esto se hace mediante el comando docker copy tal cual se observa en la figura de arriba.

```
root@11dbcc803d20:/# hdfs dfs -put /root/branch.csv /user/hive/warehouse/branch/
root@11dbcc803d20:/#
```

Mediante el comando hdfs observado se movió el archivo .csv del dataset al correspondiente directorio de HDFS para su utilización con las plataformas de Hive y Pig.

```
root@11dbcc803d20:/# pig -x mapreduce
24/06/21 21:52:08 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
24/06/21 21:52:08 INFO pig.ExecTypeProvider: Trying ExecType : MAPREDUCE
24/06/21 21:52:08 INFO pig.ExecTypeProvider: Picked MAPREDUCE as the ExecType
2024-06-21 21:52:08,806 [main] INFO  org.apache.pig.Main - Apache Pig version 0.17.0 (r1797386) compiled Jun 02 2017, 15:41:58
2024-06-21 21:52:08,806 [main] INFO  org.apache.pig.Main - Logging error messages to: //pig_1719006728804.log
2024-06-21 21:52:08,864 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /root/.pigbootup not found
2024-06-21 21:52:09,463 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.ad
dress
2024-06-21 21:52:09,463 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: hdfs://11dbcc803d20
:9000
2024-06-21 21:52:10,520 [main] INFO  org.apache.pig.PigServer - Pig Script ID for the session: PIG-default-7f3d616e-195a-4e6b-a7dc-6cf02615bfc4
2024-06-21 21:52:10,520 [main] WARN  org.apache.pig.PigServer - ATS is disabled since yarn.timeline-service.enabled set to false
grunt>
```

Con el comando **pig -x mapreduce** dentro de la bash del contenedor se inicializa el servicio de Apache Pig con modo de trabajo MapReduce. Se puede observar que se ha logrado entrar al cliente del servicio cuando en la consola indica **grunt>** y espera a que el usuario haga consultas.

```
grunt> data = LOAD '/user/hive/warehouse/branch/branch.csv'
>>        USING PigStorage(',')
>>        AS (address: chararray, operation: chararray, flag: int, target: chararray);
grunt>
```

La consulta realizada corresponde a la carga de los datos del dataset al almacenamiento de Apache Pig. Se identifica en cada campo el tipo de dato al que corresponde cada columna de branch.csv.

```
grunt> limited_data = LIMIT data 100;
grunt> DUMP limited_data;
```

```
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
(0x14fe7d34,conditional_jump,0,0x14fe7d40)
(0x14fe7d3e,conditional_jump,1,0x14fe7d2e)
grunt>
```

Al mostrar los primeros 100 datos podernos asegurarnos de que se hayan importado con éxito y evidenciamos la utilización de la primera consulta.

```
grunt> root@11dbcc803d20:/# hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/apache-hive-2.3.5-bin/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop-2.9.2/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Logging initialized using configuration in jar:file:/usr/local/apache-hive-2.3.5-bin/lib/hive-common-2.3.5.jar!/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hiv
e 1.X releases.
```

Por otro lado, al hacer uso del comando **hive** dentro de la bash del contenedor se inicializa el servicio de Apache Hive.

```
hive> CREATE TABLE branch (
    >     address STRING,
    >     operation STRING,
    >     flag INT,
    >     target STRING
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 13.097 seconds
```

La consulta realizada corresponde a la creación de la tabla para la posterior importación de los datos desde el brach.csv. Al igual que en Pig, se definen los tipos de cada dato y se definen algunas reglas de formato para una correcta importación.

```
hive>
    >
    >
    > LOAD DATA INPATH '/user/hive/warehouse/branch/branch.csv' INTO TABLE branch;
Loading data to table default.branch
OK
Time taken: 1.725 seconds
```

La consulta **LOAD DATA INPATH** es utilizada para copiar los datos del dataset branch.csv a la tabla creada con anterioridad. Para esto se debe especificar la ruta donde se encuentra ubicado el archivo.

```
hive> SELECT * FROM branch LIMIT 100;
OK
branch_addr     branch_type     NULL    target
0x7f6cae86edbf  conditional_jump        1       0x7f6cae86edc2
0x7f6cae86edd2  return  1       0x149fae9b
0x149faea6      conditional_jump        1       0x149faee6
0x149faef4      return  1       0x14fe7d12
0x14fe7d1d      conditional_jump        0       0x15e134e4
0x14fe7d34      conditional_jump        0       0x14fe7d40
0x14fe7d3e      conditional_jump        1       0x14fe7d2e
0x14fe7d34      conditional_jump        0       0x14fe7d40
0x14fe7d3e      conditional_jump        1       0x14fe7d2e
0x14fe7d34      conditional_jump        0       0x14fe7d40
0x14fe7d3e      conditional_jump        1       0x14fe7d2e
0x14fe7d34      conditional_jump        0       0x14fe7d40
0x14fe7d3e      conditional_jump        1       0x14fe7d2e
0x14fe7d34      conditional_jump        0       0x14fe7d40
0x14fe7d3e      conditional_jump        1       0x14fe7d2e
0x14fe7d34      conditional_jump        0       0x14fe7d40
0x14fe7d3e      conditional_jump        1       0x14fe7d2e
0x14fe7d34      conditional_jump        0       0x14fe7d40
0x14fe7d3e      conditional_jump        1       0x14fe7d2e
0x14fe7d34      conditional_jump        0       0x14fe7d40
0x14fe7d3e      conditional_jump        1       0x14fe7d2e
0x14fe7d34      conditional_jump        0       0x14fe7d40
0x14fe7d3e      conditional_jump        1       0x14fe7d2e
0x14fe7d34      conditional_jump        0       0x14fe7d40
0x14fe7d3e      conditional_jump        1       0x14fe7d2e
0x14fe7d34      conditional_jump        0       0x14fe7d40
0x14fe7d3e      conditional_jump        1       0x14fe7d2e
0x14fe7d34      conditional_jump        0       0x14fe7d40
0x14fe7d3e      conditional_jump        1       0x14fe7d2e
0x14fe7d34      conditional_jump        0       0x14fe7d40
0x14fe7d3e      conditional_jump        1       0x14fe7d2e
0x14fe7d34      conditional_jump        0       0x14fe7d40
```

Nuevamente se mostraron los 100 primeros datos para demostrar la correcta importación de los datos.

Consultas con Hive

Se realizaron consultas para identificar patrones, tendencias y relaciones significativas en los datos. Estas permitirían entender la distribución de los datos, la frecuencia de ocurrencia de ciertos eventos y posibles correlaciones entre variables mediante el análisis de los resultados obtenidos.

1. Obtener estadísticas básicas como el número total de registros en el dataset.

```
hive> SELECT COUNT(*) AS total_records
    > FROM branch;
```

```
OK
105985494
Time taken: 59.018 seconds, Fetched: 1 row(s)
hive> _
```

La primera consulta fue hecha en Hive y corresponde a un operador **COUNT** que selecciona el total del conjunto de datos. La respuesta retornada es 105985494, lo que significa que hay 105985494 registros en el dataset.

2. Contar la frecuencia de cada tipo de branch utilizando Pig y Hive.

```
hive>
    > SELECT operation, COUNT(*) AS occurrence
    > FROM branch
    > GROUP BY operation
    > ORDER BY occurrence DESC;
```

```
OK
conditional_jump        92093015
return  4462301
direct_call     3873518
direct_jump     3790370
indirect_jump   1174838
indirect_call   588157
interrupt       1727
context_switch  1567
branch_type     1
Time taken: 128.577 seconds, Fetched: 9 row(s)
hive>
```

La siguiente consulta corresponde al conteo de cada tipo de branch o “operation” (como fue definido en la tabla). Los resultados obtenidos fueron los siguientes:

- 92093015 conditional jump
- 4462301 return
- 3873518 direct call
- 1174838 indirect jump
- 588157 indirect call
- 1727 interrupt
- 1567 context switch

El tipo de branch, u operaciones, corresponde al registro de las siguientes acciones que hizo un proceso. Por ejemplo hubieron 92093015 procesos que hicieron un salto condicional de una dirección de memoria a otra, o 1727 procesos que fueron interrumpidos.

3. Analizar la relación entre los tipos de branch y el valor de “taken” (1 o 0).

```
hive>
    > SELECT operation,
    >        COUNT(*) AS total_count,
    >        SUM(CASE WHEN flag = 1 THEN 1 ELSE 0 END) AS taken_count,
    >        SUM(CASE WHEN flag = 0 THEN 1 ELSE 0 END) AS not_taken_count
    > FROM branch
    > GROUP BY operation
    > ORDER BY total_count DESC;
```

```
OK
conditional_jump        92093015        19205738        72887277
return  4462301 4462301 0
direct_call     3873518 3873518 0
direct_jump     3790370 3790370 0
indirect_jump   1174838 1174838 0
indirect_call   588157  588157  0
interrupt       1727    1727    0
context_switch  1567    1567    0
branch_type     1       0       0
Time taken: 150.869 seconds, Fetched: 9 row(s)
hive>
```

El valor **taken** corresponde a un número entero que toma dos valores: 1 o 0. Este valor indica si se cumple la condición requerida por el tipo de branch (operación). Por ejemplo, para un salto condicional (conditional_jump) el taken puede tomar el valor de 1 si se cumple esa condición, y 0 si no.

En los resultados obtenidos se muestra en la primera columna el tipo de branch, luego el total de registros, la cantidad de veces que taken tomó el valor de 1 y finalmente la cantidad de veces que taken era igual a 0. Observamos que para las operaciones no condicionales el taken tomó el valor de 1 para el total de los registros, mientras que para los saltos condicionales (conditional_jump) la veces que tomó el valor de 1 son muy menor a las veces que tomó el valor de 0, por lo que solo 19205738 saltos condicionales de 92093015 fueron realizados por los procesos.

4. Calcular la proporción de registros con “taken” igual a 1 para cada tipo de branch.

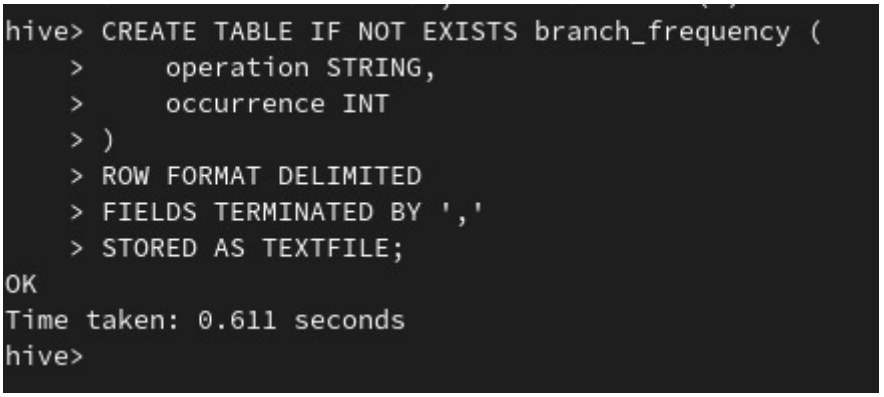
```
hive> WITH total_counts AS (
    >     SELECT operation, COUNT(*) AS total_count
    >     FROM branch
    >     GROUP BY operation
    > ),
    > taken_counts AS (
    >     SELECT operation, COUNT(*) AS taken_count
    >     FROM branch
    >     WHERE flag = 1
    >     GROUP BY operation
    > )
    > SELECT t.operation,
    >        t.taken_count,
    >        tc.total_count,
    >        (t.taken_count * 100.0 / tc.total_count) AS taken_percentage
    > FROM taken_counts t
    > JOIN total_counts tc ON t.operation = tc.operation
    > ORDER BY taken_percentage DESC;
```

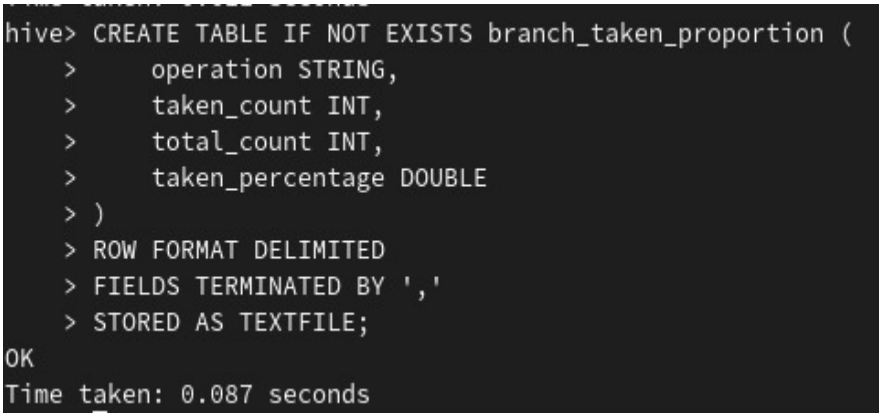
```
OK
direct_call     3873518 3873518 100.000000000000000
indirect_call   588157  588157  100.000000000000000
context_switch  1567    1567    100.000000000000000
direct_jump     3790370 3790370 100.000000000000000
indirect_jump   1174838 1174838 100.000000000000000
return  4462301 4462301 100.000000000000000
interrupt       1727    1727    100.000000000000000
conditional_jump        19205738        92093015        20.854717374602189
Time taken: 281.701 seconds, Fetched: 8 row(s)
hive>
```

Los resultados obtenidos nos demuestran que la relación entre el valor taken y los tipos de branch es de un 100% para todas las operaciones que no corresponden a un salto condicional, pues estas al no depender de una condición el valor es 1 siempre.

Para el salto condicional el resultado es de aproximadamente 21%, lo que significa que solo este porcentaje de saltos se concretaron (taken = 1).

5. Crear tablas en Hive para almacenar los resultados de los análisis realizados.

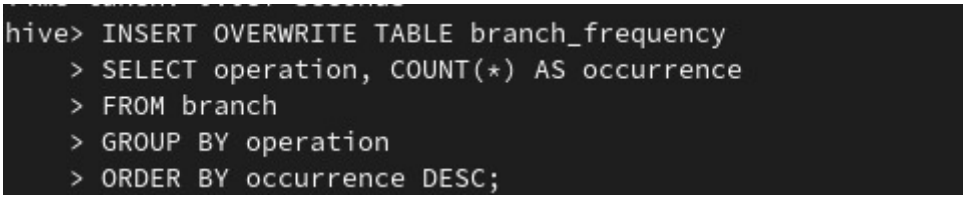


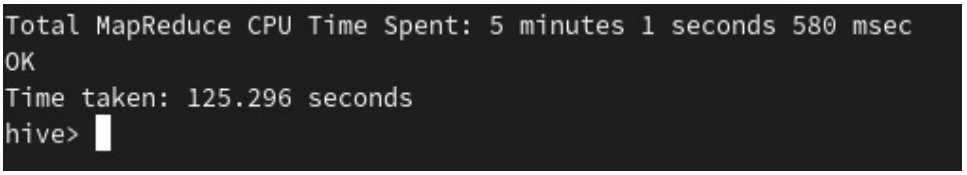


Con el operador **CREATE TABLE** se crea la tabla para los resultados de los análisis realizados. El flag **IF NOT EXISTS** es para indicar que la tabla se debe crear si no existe ya una con ese nombre.

Se crearon dos tablas para cada análisis de tipo de branch: una para almacenar las frecuencias, y otra para las proporciones de taken.

6. Almacenar los conteos de frecuencia y las relaciones analizadas en tablas separadas en Hive.





Para la tabla de frecuencia se hizo nuevamente la simple operación de **COUNT** para cada dato de tipo de branch ordenándolos por grupo con **GROUP BY**, y se almacenó en la tabla branch_frequency con la consulta **INSERT** con la indicación de **OVERWRITE TABLE**.

```
hive> WITH total_counts AS (
    >     SELECT operation, COUNT(*) AS total_count
    >     FROM branch
    >     GROUP BY operation
    > ),
    > taken_counts AS (
    >     SELECT operation, COUNT(*) AS taken_count
    >     FROM branch
    >     WHERE flag = 1
    >     GROUP BY operation
    > )
    > INSERT OVERWRITE TABLE branch_taken_proportion
    > SELECT t.operation,
    >        t.taken_count,
    >        tc.total_count,
    >        (t.taken_count * 100.0 / tc.total_count) AS taken_percentage
    > FROM taken_counts t
    > JOIN total_counts tc ON t.operation = tc.operation
    > ORDER BY taken_percentage DESC;
```

```
Total MapReduce CPU Time Spent: 11 minutes 12 seconds 320 msec
OK
Time taken: 276.13 seconds
hive>
```

Para la tabla de proporciones de taken por tipo de branch se volvió a hacer la consulta correspondiente y se insertó en la tabla de branch_taken_proportion con **INSERT OVERWRITE TABLE** después de hacer los cálculos necesarios.

Consultas con Pig

Para la utilización de Pig existe un problema, ya que al realizar cualquier consulta el procesamiento con MapReduce llegaba hasta 60%, 80%, incluso 90%, y se caía el servidor, entrando en un bucle casi infinito de “Retrying to connect to server”. Esto, nuevamente, para cualquier consulta.

La primera consulta hecha en pig, para demostrar que se habían cargado los datos, fue limitando a simplemente los primeros 100. De otra manera, si se hubiera consultado por todos los datos, también habría ocurrido el mismo error.

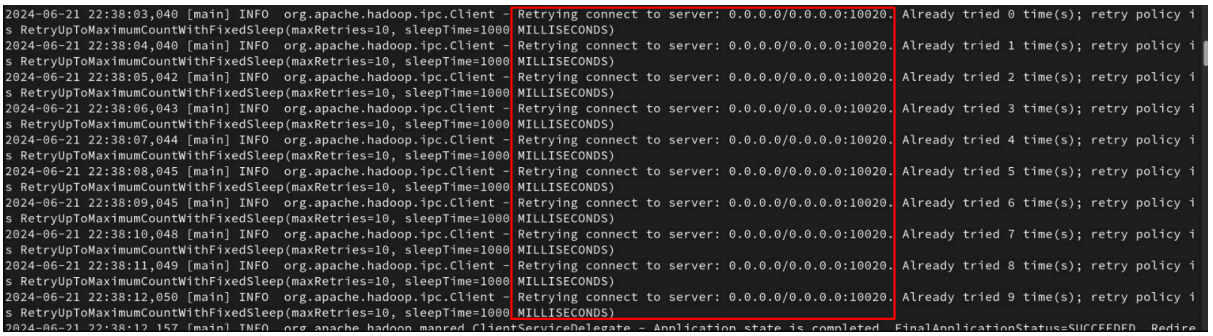
A continuación se observa evidencia del problema contemplado.

```
grunt> data = LOAD '/user/hive/warehouse/branch/branch.csv'
>>        USING PigStorage(',')
>>        AS (address: chararray, operation: chararray, flag: int, target: chararray);
grunt> filtered_data = FILTER data BY flag == 0;
grunt> grouped_data = GROUP filtered_data BY operation;
grunt> operation_stats = FOREACH grouped_data GENERATE group AS operation, COUNT(filtered_data.flag) AS total_flags, AVG(filtered_data.flag) AS average_flag;
grunt> DUMP operation_stats;
```

Se intentó hacer una consulta de cálculo de frecuencia, donde para empezar se filtrarían los datos con el valor de taken = 0, utilizando la consulta **FILTER**. Estos se agruparían por tipo de branch, con la consulta **GROUP** y luego se harían los cálculos correspondientes dentro de un **FOREACH,** para dar como resultado la frecuencia del taken 0 para cada tipo de branch.

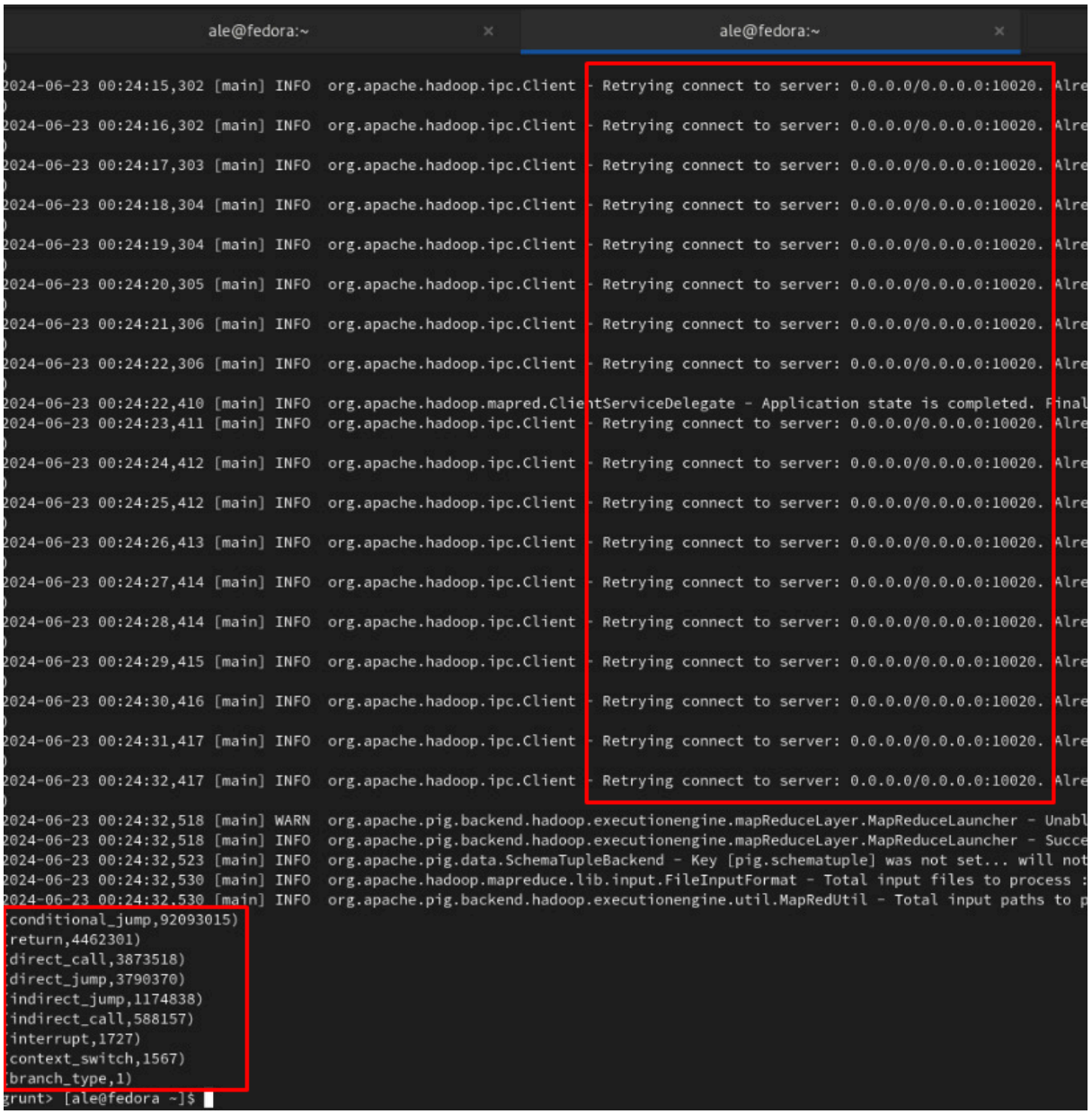
```
grunt> data = LOAD '/user/hive/warehouse/branch/branch.csv'
>>        USING PigStorage(',')
>>        AS (address: chararray, operation: chararray, flag: int, target: chararray);
grunt> filtered_data = FILTER data BY flag == 0;
grunt> grouped_data = GROUP filtered_data BY operation;
grunt> operation_stats = FOREACH grouped_data GENERATE group AS operation, COUNT(filtered_data.flag) AS total_flags, AVG(filtered_data.flag) AS average_flag;
grunt> DUMP operation_stats;
2024-06-21 22:26:56,848 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: GROUP_BY,FILTER
2024-06-21 22:26:56,891 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2024-06-21 22:26:56,946 [main] INFO  org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, ColumnMapKeyPrune, ConstantCalculator, GroupByConstParallelSetter, LimitOptimizer, LoadTypeCastInserter, MergeFilter, MergeForEach, NestedLimitOptimizer, PartitionFilterOptimizer, PredicatePushdownOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFilter, StreamTypeCastInserter]}
2024-06-21 22:26:57,062 [main] INFO  org.apache.pig.impl.util.SpillableMemoryManager - Selected heap (PS Old Gen) of size 699400192 to monitor. collectionUsageThreshold = 489580128, usageThreshold = 489580128
2024-06-21 22:26:57,162 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompiler - File concatenation threshold: 100 optimistic? false
2024-06-21 22:26:57,191 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.CombinerOptimizerUtil - Choosing to move algebraic foreach to combiner
2024-06-21 22:26:57,266 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size before optimization: 1
2024-06-21 22:26:57,266 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size after optimization: 1
2024-06-21 22:26:57,412 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2024-06-21 22:26:57,769 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
2024-06-21 22:26:57,788 [main] INFO  org.apache.pig.tools.pigstats.mapreduce.MRScriptState - Pig script settings are added to the job
2024-06-21 22:26:57,796 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.reduce.markreset.buffer.percent is deprecated. Instead, use mapreduce.reduce.markreset.buffer.percent
2024-06-21 22:26:57,797 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - mapred.job.reduce.markreset.buffer.percent is not set, set to default 0.3
2024-06-21 22:26:57,800 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.output.compress is deprecated. Instead, use mapreduce.output.fileoutputformat.compress
2024-06-21 22:26:57,804 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - Reduce phase detected, estimating # of required reducers.
2024-06-21 22:26:57,805 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - Using reducer estimator: org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.InputSizeReducerEstimator
2024-06-21 22:26:57,823 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.InputSizeReducerEstimator - BytesPerReducer=1000000000 maxReducers=999 totalInputFileSize=4264974508
2024-06-21 22:26:57,824 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - Setting Parallelism to 5
2024-06-21 22:26:57,824 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.reduce.tasks is deprecated. Instead, use mapreduce.job.reduces
2024-06-21 22:26:57,824 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - This job cannot be converted run in-pro
```

```
]
2024-06-21 22:36:57,183 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 54% complete
2024-06-21 22:36:57,183 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Running jobs are [job_1719005460953_0002]
2024-06-21 22:37:48,938 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 58% complete
2024-06-21 22:37:48,943 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Running jobs are [job_1719005460953_0002]
2024-06-21 22:37:55,982 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 73% complete
2024-06-21 22:37:55,982 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Running jobs are [job_1719005460953_0002]
2024-06-21 22:37:56,987 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 86% complete
2024-06-21 22:37:56,987 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Running jobs are [job_1719005460953_0002]
2024-06-21 22:37:58,990 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Running jobs are [job_1719005460953_0002]
2024-06-21 22:38:02,008 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
```



Es ahí cuando empieza la ejecución del MapReduce de Pig para el procesamiento de los datos. En el primer recuadro rojo podemos observar el avance del procedimiento. En la segunda casilla roja notamos el error mencionado, el cual pasado de un porcentaje de procesamiento se cae el servidor de Pig y no responde más.

Con mucho tiempo corriendo el bucle de reconexión se logró finalizar una sola consulta con Pig, la cual era de las más básicas, para hacer un conteo de registro de cada tipo de branch. Se observa a continuación.



Se logra observar que el error seguía persistente, pero que después de mucho tiempo corriendo e intentando reconectar logró dar con los resultados, los cuales a su vez coinciden con los de la consulta hecha por Hive (frecuencia de cada tipo de branch). De esta manera se concluye que el uso de Pig era el correcto, a pesar de los problemas de conexión con el servidor. Sin embargo, se cree que se logró llegar a un resultado porque era una consulta de caracter básico, ya que a pesar de ser básica tomó mucho tiempo (1 hora) en que retornara, debido al error. De no haber sido así, quizás habría tomado muchos pares de horas para otros tipos de consultas, lo que imposibilitó un mayor análisis con Apache Pig.

Preguntas

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

La primera ventaja radica en la simplicidad de la estructuración, pues el lenguaje utilizado por Apache Pig, PigLatin, es más simple de escribir que un código MapReduce para otro lenguaje como Java, por ejemplo. Esto se debe a que utiliza palabras simples y claves para cada consulta, como GROUP para agrupar, FILTER para filtrar, etc. Por lo que el consultante debe simplemente escribir el tipo de consulta y las variables a consultar, ahorrándose así el escribir muchas líneas de código para todo el funcionamiento.

Otra ventaja es la interoperabilidad que ofrece, pues Apache Pig se integra bien con otras herramientas de Hadoop, como Hive y otras. Esto facilita la construcción de sistemas de procesamiento y análisis de datos mucho más complejos que utilizan múltiples tecnologías.

Además Apache Pig es una herramienta ya establecida. Si se quisiera escribir código MapReduce para el procesamiento de datos habría que tener en consideración los grandes volúmenes de datos, lo que quizás complicaría el desarrollo del código al tener que tomar en cuenta cosas como el orden del tiempo de ejecución, etc.

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?

Hive y Pig son ambas dos herramientas poderosas de Hadoop para el procesamiento de datos. Las diferencias son mínimas y radican en preferencias para el usuario.

Cuando la situación requiere un trabajo de datos almacenados en tablas sería preferible ocupar Hive por la estructura y lenguaje que ofrece, que es parecido al trabajo con lenguajes SQL. Por otra parte, si se necesita algo más complejo que no se adapta a una estructura de tablas se puede utilizar Pig.

En resumen, Hive simplifica las consultas al tener un lenguaje similar al SQL, y así mismo simplifica la estructuración de los datos al ser capaz de ordenarlos en tablas.

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

En primer lugar se creó una tabla con la instrucción **CREATE TABLE,** donde se definían las variables a almacenar, con su tipo de dato. Luego, se reescribieron los datos del dataset de Google Workload Traces en la tabla creada.

Se hicieron consultas como la frecuencia de cada tipo de branch, lo cual es útil ya que nos permite identificar qué tipos de branches son más comunes y que podrían necesitar optimización, y también se consultó la proporción de taken por tipo de branch, que permite evaluar la efectividad de las predicciones de cada branch y analizar patrones de ejecución, al ser un valor que indica las acciones de un proceso,.

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?

Un caso bastante conocido es el de Twitter, una de las redes sociales más grandes del mundo, y que día a día genera enormes cantidades de datos. El problema es que Twitter necesitaba una herramienta escalable para almacenar y procesar grandes volúmenes de datos tanto no estructurados como semi-estructurados.

Twitter recurrió entonces a la implementación de los servicios de Apache Pig y Apache Hive.

Con la utilización de Pig solucionó problemas de procesamiento de logs y análisis de los tweets de los usuarios. Esto es muy importante ya que Twitter tenía la necesidad de analizar logs para identificar patrones de uso, detectar errores, y así optimizar el rendimiento de la aplicación.

Con la simplicidad que ofrece HiveQL Twitter logró identificar de manera más óptima tendencias en tiempo real, pudiendo analizar así la popularidad de hashtags y temas específicos, al ejecutar consultas SQL para agrupar y contar los hashtags, además de identificar peaks de actividad y relaciones entre datos.

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

Al investigar se descubrieron otras tecnologías de Apache tales como Apache Spark y Apache Flink.

Flink está diseñado para el procesamiento de flujos de datos en tiempo real con baja latencia, lo cual garantiza una alta consistencia y precisión sobretodo para aplicaciones que requieren una alta precisión en el procesamiento de datos.

Spark ofrece un modelo de programación mucho más rápido que MapReduce debido a su uso de procesamiento en memoria. Esto es útil para iteraciones repetidas como las que se encuentran en el aprendizaje automático y el análisis de gráficos. Además Spark soporta múltiples lenguajes de programación, incluyendo Java, Python, entre otros.

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?

La facilidades que ofrece Pig son: su lenguaje, ya que PigLatin representa un lenguaje de alto nivel más procedural que un lenguaje SQL; y flexibilidad, ya que permite realizar transformaciones complejas y personalizadas en los datos.

Mientras tanto, el lenguaje de Hive es HiveQL que es muy similar a SQL y eso lo hace accesible y familiar para aun más usuarios, que tengan experiencia con bases de datos relacionales. Y ofrece también facilidad y simplicidad al poder ordenar los datos en tablas.

Conclusión

Hadoop nos entrega grandes herramientas de procesamiento de datos como lo son Apache Pig y Apache Hive, los cuales dominan en la industria cuando se refiere a procesamiento de grandes volúmenes de datos. Estos ofrecen características como el modelo de programación MapReduce, el cual optimiza el procesamiento de datos al almacenarlos en distintos clústeres.

Ambas herramientas trabajan de manera similar pero ofrecen facilidades distintas a los usuarios, como agrupación en tablas y un lenguaje familiar en el caso de Hive. Por lo que hemos descubierto y demostrado la utilización de dos herramientas muy poderosas y útiles para el desarrollo de grandes aplicaciones que necesiten trabajar con grandes volúmenes de datos.

Vídeo

https://youtu.be/6PKHsmEQdiQ?si=ewBSkn1mG5J_r8wi