Práctica 2: Ecosistema en Hadoop

Big Data

Máster en Ingeniería Informática

Autora: Almudena García Jurado-Centurión

Índice

Enunciado

Tal y como se ha expuesto en la práctica 1, pretendemos obtener una lista de la medida de radiación para cada una los diez municipios, es decir, una lista de la media de radiación media, en la que se muestren dos columnas: municipio y media de la radiación media, de modo que observando esta información, tendremos una aproximación al rendimiento que daría en él nuestro huerto solar. Además se pide la media de precipitaciones de cada municipio en el apartado b.

Estudio experimental

Para aprobar dicha práctica sólo se pide que se implemente en uno de estos ecosistemas: Hive o Pig.

Sobre la configuración de la práctica anterior, se realizarán las siguientes pruebas:

- Realización con Hive :Se realizará la práctica con Hive. o
- Realización con Pig: Se realizará la práctica con Pig.

Resolución

En este caso, se ha optado por utilizar ambas plataformas, replicando el ejercicio de la Práctica 1 tanto con Pig como con Hive.

Se compararán los resultados de ambas plataformas, tanto entre sí, como con los de la implementación original de la Práctica 1, centrándose tanto en la corrección de los datos obtenidos, como la medición temporal de la ejecución de cada tarea.

Especificaciones del sistema

- **Sistema operativo:** Debian GNU/Linux 11 (bullseye)
- **Versión de Hadoop:** 2.10.1

  - Doble instalación: *standalone* y pseudodistribuida
- **Versión de Java:** java-1.8.0-openjdk-amd64
- **Versión de Pig:** 0.17.0
- **Versión de Hive:** 2.3.7

Hardware

- Modelo del equipo: Thinkpad T440p
- Memoria RAM: 16 GB 1666 MHz
- Procesador: i7 4702MQ
- Disco duro: Samsung 860 EVO SSD 500 GB

Configuración de Hadoop

Instalación dual: *standalone* y pseudodistribuida

- **Instalación *standalone*:** `/usr/local/hadoop_std`
- **Instalación pseudodistribuida:** `/usr/local/hadoop`
  - Directorio HDFS: `hdfs://0.0.0.0:9000/user/almu`

La instalación a utilizar se indica definiendo su ruta en la variable de configuración `HADOOP_HOME`, en el fichero `.bashrc` de nuestro usuario (`~/.bashrc`).

Apache Pig

Apache Hive es una plataforma que permite la ejecución de tareas MapReduce desde sentencias de alto nivel. Su funcionamiento se basa en la aplicación de filtros y transformaciones de los datos, expresadas de forma secuencial dentro de un script.

Instalación de Pig

Para instalar Pig, simplemente lo descargamos de la página web y lo extraemos en un directorio. Para un acceso mas cómodo, se configurarán algunas variables de entorno dentro de la consola de Linux.

- **Descarga del software**

  Descargamos el software de la página oficial y lo extraemos

  ```
  wget https://apache.mirror.wearetriple.com/pig/pig-0.17.0/pig-0.17.0-src.tar.gz
  tar -xvf pig-0.17.0-src.tar.gz
  ```

  Esto nos creará un nuevo directorio llamado `pig-0.17.0` en nuestro directorio actual

- **Instalación del software**

Copiamos dicho directorio a una ubicación mas adecuada. En este caso, lo copiaremos en `/usr/local/pig`

```
sudo cp pig-0.17.0 /usr/local/pig
```

- **Configuración de variables de entorno**

  Añadimos a nuestro fichero `.bashrc`, las variables de configuración de Pig, y añadimos la ruta del ejecutable al PATH. Estas se sumarán a las ya configuradas para Hadoop

```
#Apache Hadoop
export JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-amd64"
export HADOOP_HOME="/usr/local/hadoop"
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export PATH=$PATH:$HADOOP_HOME/bin

#Apache Pig
export JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-amd64"
export PATH=$PATH:$JAVA_HOME/bin

export PIG_HOME=/usr/local/pig
export PATH=$PATH:$PIG_HOME/bin
```

- **Probando la instalación**

  Ejecutamos el comando `pig --version` para comprobar la versión de Pig, y ver si funciona. Si todo ha ido bien, veremos algo como esto:

```
almu@debian:~$ pig --version
Apache Pig version 0.17.0 (r1797386)
compiled Jun 02 2017, 15:41:58
almu@debian:~$
```

Implementación de las tareas

Una vez instalado Pig, pasamos a implementar las tareas requeridas en nuestra práctica

Planteamiento

El planteamiento de nuestra práctica se dividirá en varias etapas: carga de los datos, filtrado y conversión de los datos, tarea 1 y tarea 2. La primera tarea se

destinará a calcular la media de radiación de cada estación, y la segunda calculará la estación de máxima radiación y las precipitaciones por año de la misma.

En este caso, debido a la simplicidad del problema, implementaremos ambas tareas sobre el mismo fichero; reciclando las relaciones obtenidas en la preparación de los datos y la primera tarea, para su uso en la segunda tarea.

Preparación de los datos

- **Carga de los datos**

  Para la carga de los datos utilizaremos la instrucción `load`. Debido a que algunos datos numéricos tienen comillas y otros problemas que podrían entorpecer la conversión a otros tipos, dichos datos se cargarán como cadenas de caracteres, los cuales se procesarán posteriormente para resolver dichos problemas y aplicar una conversión al tipo apropiado.

  El código queda tal que así:

```
  raw = load 'RIA_exportacion_datos_diarios_Huelva_20140206.csv'
using PigStorage(';')
  AS (id_region:int, name_region:chararray, id_station: chararray,
name_station: chararray,
      date: chararray, dia: int, tempmax: chararray, hormintempmax:
int, tempmin: chararray,
      hormintempmin: int, temp_mid: chararray, humidity_max: chararray,
      humidity_min: chararray, humidity_mid: chararray, speed_wind:
chararray,
      dir_wind: chararray, radiation: chararray, rain: chararray
  );
```

  Esto leerá los datos del fichero csv, y los cargará en una relación llamada `raw`. El esquema de datos a usar será el correspondiente a las columnas del fichero csv, aunque sin conservar los nombres de las columnas originales.

- **Filtrado y conversión de los datos**

  Una vez cargados los datos del fichero en nuestra relación, pasamos a filtrar las columnas que nos interesan, y aplicar las transformaciones necesarias para corregir las deficiencias de formato.

  Empezamos eliminando la cabecera del fichero. Para eso, aplicamos un filtro que nos elimine las filas cuyo campo `id_station` (el ID de la estación) sea una cadena de caracteres.

```
  --Skip header
  filter_data = FILTER raw BY id_station != 'IDESTACION';
```

  Filtradas las filas que nos interesan, filtramos y transformamos las columnas

```
  --Extract relevant data, fixing problems and applying casting to
their types
  replace_data = FOREACH filter_data GENERATE
      (int) SUBSTRING(id_station,1, (int)SIZE(id_station)-1) as id_station,
--Remove ""
      SUBSTRING(name_station,1, (int)SIZE(name_station)-1) as name_station,
--Remove ""
      (float) REPLACE(radiation, ',', '.') as radiation, --Fix decimal
separator
      (float) REPLACE(rain, ',', '.') as rain, --Fix decimal separator
      (int) SUBSTRING(date, (int)SIZE(date)-4, (int)SIZE(date)) as
year; --Extract year from date
```

- El campo `id_station` tiene el número rodeado por comillas. Eliminamos las comillas, filtrando la subcadena sin los caracteres de los extremos, y convertimos el valor a un número entero.
- El nombre de la estación (`name_station`) también está entrecomillado. Quitamos las comillas mediante otra subcadena
- El valor de radiación usa la coma como separador decimal. Utilizamos un reemplazo, para cambiar la coma por el punto (separador decimal por defecto en Pig), y lo convertimos a coma flotante.
  - ⋆ Aplicamos la misma transformación para el valor de precipitaciones
- El año está contenido dentro del campo fecha, así que filtramos su valor mediante una subcadena, y lo convertimos a entero.

Los datos obtenidos los almacenamos en la relación `replace_data`, la cual utilizaremos para los cálculos de las tareas 1 y 2.

Tarea 1. Cálculo de la radiación media de un determinado conjunto de estaciones

- **Subtarea 1.1: Filtrado de las estaciones que se correspondan con los ID seleccionados**

  Empezamos aplicando un filtro para obtener únicamente las filas de las estaciones que nos interesan. En este caso, aquellas con ID entre 2 y 10.

```
  --Filter the stations which corresponds with ID
  filter_station = FILTER replace_data BY id_station > 1 AND id_station
< 11;
```

  Utilizamos una sentencia `FOREACH` para quedarnos únicamente con las columnas de nombre y radiación

```
  station_radiation = FOREACH filter_station GENERATE name_station,
radiation;
```

Los datos filtrados se almacenan en la relación `station_radiation`.

- **Subtarea 1.2: Cálculo de las medias de radiación de cada estación**

  Una vez obtenidos los nombres de las estaciones y sus valores de radiación, agrupamos los datos para calcular la media de radiación de cada una de ellas.

  Aplicamos la sentencia `GROUP BY` sobre la relación `station_radiation` obtenida en el paso anterior, utilizando el campo `name_station` para agrupar los datos según el nombre de la estación.

  ```
  --Group the rows by station name, to calculate its average
  radiation_gr = GROUP station_radiation BY name_station;
  ```

  Una vez agrupados los datos, usamos una sentencia `FOREACH` para iterar sobre la relación, aplicando la función `AVG()` para hacer la media del campo `radiation`, y mostrando dicho valor junto al nombre de su estación correspondiente. Dentro de la sentencia, utilizaremos la relación `station_radiation`, obtenida en el paso anterior, para iterar sobre ella y realizar las operaciones. Utilizamos la opción `FLATTEN()` para extraer el campo `name_station` de la relación y mostrarlo individualmente.

  Los datos se almacenarán en la relación `avg_radiation`. La nueva columna con el valor medio de radiación se llamará `mid_radiation`.

  ```
  --Extract the radiation average of each station
  avg_radiation = FOREACH radiation_gr GENERATE
      FLATTEN(station_radiation.name_station) as name_station,
      AVG(station_radiation.radiation) as mid_radiation;
  ```

  Esta operación nos mostrará, por cada fila existente en la relación original, el nombre de la estación almacenada en dicha fila, y la media de radiación de la misma. Debido a esto, se mostrarán muchas filas repetidas, mostrando el nombre de la estación tantas veces como hubiera en la relación original. Para resolverlo, aplicaremos una sentencia `DISTINCT`, que mantendrá únicamente aquellas filas que sean diferentes del resto.

  ```
  --Remove repeated rows
  avg_radiation_dist = DISTINCT avg_radiation;
  ```

  El resultado del `DISTINCT` se almacenará en la relación `avg_radiation_dist`.

- **Exportación de los datos al fichero**

  Una vez resuelta la tarea, exportamos los datos a un fichero. Para ello, usamos la sentencia `STORE INTO`.

  ```
  --Store results in a file
  STORE avg_radiation_dist INTO 'output_radiation' using PigStorage('\t');
  ```

Los datos resultantes se almacenarán en el directorio `output_radiation`, usando tabuladores como separadores de las columnas.

Tarea 2: Cálculo de las precipitaciones por año de la estación de mayor radiación media

En este caso, a diferencia de la práctica 1, la tarea 2 recogerá los datos directamente de las relaciones obtenidas de la parte de preparación de los datos y la tarea 1; en lugar de recoger los datos de ficheros separados del programa.

Esta tarea se dividirá en varias subtareas: obtención del nombre de la estación de mayor radiación media, filtrado de los datos de precipitaciones y fecha de dicha estación, y cálculo de la suma de lluvias por año de la misma.

- **Tarea 2.1: Cálculo de la estación de mayor radiación media**

  Para esta tarea nos basaremos en la relación `avg_radiation_dist`, obtenida en la tarea 1. La tarea se realizará en dos partes: en primer lugar ordenaremos la tabla de forma descendente, y posteriormente filtraremos la primera fila.

  El uso de esta metodología se debe a su mayor eficiencia. Aunque Pig ofrece la función `MAX()` para el cálculo del máximo, esta requiere de dos tareas MapReduce para obtener el nombre de la estación asociada a dicho máximo, lo cual resulta muy ineficiente en términos de paralelización. Esta técnica permite realizar la misma labor en un solo MapReduce, lo cual genera una ejecución mas rápida y eficiente.

  Para ordenar la tabla usaremos la sentencia `ORDER BY`, utilizando como campo de ordenación el valor de radiación media (`avg_radiation`). El filtro de la primera fila lo realizaremos con la orden `LIMIT N`, que filtra las primeras N filas, indicando 1 como valor de N.

  ```
  --Find the maximum average radiation value
  max_avg_radiation = ORDER avg_radiation_dist BY mid_radiation DESC;
  max_rad_station = LIMIT max_avg_radiation 1;
  ```

- **Tarea 2.2: Filtrado de valores de la estación de mayor radiación media**

  Una vez obtenido el nombre de dicha estación, pasamos a filtrar sus campos. Las relaciones de la tarea 1 no incluyen los campos de precipitaciones y fecha, así que recurrimos a la relación anterior `replace_data`.

  Filtramos los datos con una sentencia `FILTER BY`, usando como campo de filtrado el nombre de la estación.

  ```
  --Filter the rain values of this station
  filter_rain = FILTER replace_data BY name_station == max_rad_station.name_stati
  ```

De las filas obtenidas, usamos una sentencia `FOREACH` para obtener los campos que nos interesan. En este caso, el nombre de la estación, su cantidad de precipitaciones y el año de medida.

```
station_rain = FOREACH filter_rain GENERATE name_station, rain,
year;
```

Estos datos se almacenarán en la relación `station_rain`.

- **Tarea 2.3: Cálculo de la suma de precipitaciones por año**

  Con los datos ya filtrados, procedemos a realizar el cálculo. Para ello, agrupamos los datos según el año, e iteramos sobre ellos para calcular su suma.

  La agrupación de los datos la realizamos mediante una sentencia `GROUP BY`, indicando como campo de agrupación el campo `YEAR`, correspondiente al año de medida.

  ```
  rain_gr = GROUP station_rain BY year;
  ```

  Con los datos ya agrupados, ejecutamos una sentencia `FOREACH` para calcular la suma, utilizando para ello la función `SUM` con el atributo `rain` correspondiente al valor de precipitaciones. También extraemos los datos del nombre de la estación y del año, utilizando para ello la función `FLATTEN()`. Para extraer dos campos desde la misma función, indicamos sus nombres entre paréntesis separados por comas.

  ```
  rain_year = FOREACH rain_gr GENERATE
                      FLATTEN(station_rain.(name_station, year)),
                      SUM(station_rain.rain);
  ```

  Y volvemos a utilizar la sentencia `DISTINCT` para eliminar las filas repetidas, al igual que hicimos en la Tarea 1.

  ```
  rain_year_dist = DISTINCT rain_year;
  ```

- **Exportación de los resultados a un fichero**

  Una vez terminada la tarea a resolver, exportamos los resultados de la misma a un fichero de texto. Para ello, volvemos a utilizar la sentencia `STORE INTO`. Los datos resultantes se almacenarán en el directorio `output_rain`, en un fichero de texto con las columnas separadas por tabuladores.

  ```
  STORE rain_year_dist INTO './output_rain' using PigStorage('\t');
  ```

Ejecución del script

Una vez programado el script con la implementación de las tareas a realizar, pasamos a ejecutarlas sobre Pig. En la ejecución comprobaremos si los resultados

son correctos y concuerdan con los de la Práctica 1. También mediremos sus tiempos de ejecución en diferentes modos, y los compararemos tanto entre sí, como con los tiempos de la Práctica 1.

Pig soporta dos modos de ejecución: en local, y en modo mapreduce. El modo local ejecuta las tareas en una sola máquina y proceso, sin utilizar HDFS (buscando los ficheros directamente en nuestro disco duro). El modo mapreduce hace uso de la arquitectura cluster de Hadoop, accediendo a los datos en el HDFS y permitiendo la ejecución con múltiples procesos y máquinas.

Además, en nuestro sistema disponemos de dos instalaciones independientes de Hadoop: una en modo `standalone` (todo en un único proceso) y otra en modo pseudodistribuido (en varios procesos, y haciendo uso del HDFS); así que realizaremos las pruebas sobre ambas instalaciones.

En total, realizaremos 4 pruebas, con ambos modos (local y mapreduce) sobre Hadoop *standalone* y pseudodistribuido.

Ejecución del script (general)

Para ejecutar el script de Pig, se usa el siguiente comando

```
pig -x local/mapreduce [ruta_script]
```

donde `ruta_script` es la ruta del script a ejecutar. Las opciones `local` y `mapreduce` son exclusivas, así que solo se puede elegir una de ellas.

Ejecutamos en modo local con:

```
pig -x local [ruta_script]
```

Ejecutamos en modo mapreduce con:

```
pig -x mapreduce [ruta_script]
```

En nuestro caso, el script se llama `practica2.pig`, así que lo ejecutamos con

```
pig -x local/mapreduce practica2.pig
```

NOTA: Pig necesita que los directorios de destino no existan, así que debemos borrarlos antes de iniciar la ejecución.

Para facilitar la ejecución, nos creamos un script de bash para cada modo, con la secuencia de comandos a utilizar en cada caso.

Ejecución en modo local

Para ejecutar el script en modo local, usamos el comando

```
pig -x local/mapreduce practica2.pig
```

Para el modo local no se necesita hacer uso del HDFS, por lo tanto la ejecución se reduce al borrado de los directorios de salida, y la ejecución del script de Pig.

- **Instalación standalone**
  - **Diseño del script**

    Añadimos un bloque de código para asegurarnos que la variable `HADOOP_HOME`, que almacena la ruta de instalación de Hadoop, apunta a la instalación *standalone*. La instalación *standalone* se encuentra en la ruta `/usr/local/hadoop_std`, mientras que la pseudodistribuida se encuentra en `/usr/local/hadoop`. Utilizaremos el comando `sed` con una expresión regular para editar la ruta en el fichero `.bashrc`.

    El script se llamará `run_pig_std_local.sh` (incluido en el directorio) y su código es el siguiente:

```
#!/bin/bash

#Change to Hadoop standalone installation
if test -d /usr/local/hadoop_std
then
    HADOOP_STD=$(grep hadoop_std ~/.bashrc | wc -l)
    if test $HADOOP_STD == 0
    then
        sed -i '/HADOOP_HOME=/s|hadoop|hadoop_std|' ~/.bashrc
        export HADOOP_HOME
        export JAVA_HOME
    fi
fi

rm -rf output_ra*

pig -x local practica2.pig
```

    Pig ya devuelve los tiempos de ejecución al terminar las tareas, por tanto no será necesario ningún código adicional para medirlo.

  - **Ejecución**

    Ejecutamos el script con

```
bash run_pig_std_local.sh
```

    Este comenzará la ejecución de las tareas de Pig, lo cual tardará varios segundos.
    Una vez terminada la ejecución, si todo ha ido bien, veremos algo similar a esto:

```
  2020-12-25 15:39:09,619 [main] INFO  org.apache.pig.backend.hadoop.execution
- Success!
  2020-12-25 15:39:09,642 [main] INFO  org.apache.pig.Main - Pig
script completed in 6 seconds and 410 milliseconds (6410 ms)
```

En este caso, la ejecución ha durado 6,4 segundos.

– **Resultados**

Comprobamos si los resultados son correctos. Comprobamos si los directorios de salida se han creado correctamente

```
almu@debian:~/Practicas_BigData/Practica2/Pig$ ls
output_radiation                                  run_pig_pseudodist_local.
output_rain                                       run_pig_pseudodist_mr.sh
practica2.pig                                     run_pig_std_local.sh
RIA_exportacion_datos_diarios_Huelva_20140206.csv  run_pig_std_mr.sh

almu@debian:~/Practicas_BigData/Practica2/Pig$ ls -r output_ra*
output_rain:
_SUCCESS  part-r-00000

output_radiation:
_SUCCESS  part-r-00000
```

Vemos que los directorios se han creado correctamente, y que contienen los ficheros de salida.

Si entramos en `output_radiation/part-r-00000` , veremos el resultado de la primera salida

```
Lepe    18.879762046745903
Aroche  17.919251145115762
Moguer  18.341574146820463
Niebla  18.07869350540072
Almonte 18.123927680463083
Gibraleón   18.70950440160565
El Campillo 18.261276060114252
La Palma del Condado    18.071459669487474
La Puebla de Guzmán 17.962850858576193
```

Lo comparamos con el mismo fichero de la Práctica 1.

```
"Moguer"    18.3416
"Lepe"  18.8797
"La Puebla de Guzmán"   17.9628
"La Palma del Condado"  18.0714
"Gibraleón" 18.7095
```

```
"Niebla"    18.0787
"Aroche"    17.9192
"El Campillo"   18.2613
"Almonte"   18.1239
```

Vemos que los resultados son muy similares, con diferencias en el orden de los elementos, el entrecomillado, y el número de decimales que se muestran. Al margen de la precisión decimal, los resultados son los mismos entre ambas versiones.

Si entramos en `output_rain/part-r-00000`, veremos los resultados de la segunda salida

```
Lepe    1999    8.59999993443489
Lepe    2000    639.8000022023916
Lepe    2001    339.9999998062849
Lepe    2002    570.8000042140484
Lepe    2003    877.8000039607286
Lepe    2004    445.59999710321426
Lepe    2005    332.3999994844198
Lepe    2006    613.3999973833561
Lepe    2007    438.9999995082617
Lepe    2008    576.1999998092651
Lepe    2009    561.1999970972538
Lepe    2010    882.9999948441982
Lepe    2011    479.4000007510185
Lepe    2012    430.4000012129545
Lepe    2013    342.39999932050705
Lepe    2014    62.199999421834946
```

Los comparamos con los de la Práctica 1.

```
1999    8.6
2000    639.8
2001    340
2002    570.8
2003    877.8
2004    445.6
2005    332.4
2006    613.4
2007    439
2008    576.2
2009    561.2
2010    883
2011    479.4
2012    430.4
2013    342.4
```

```
2014    62.2
```

De nuevo, vemos bastante similitud entre ambos resultados. En el caso de esta nueva versión, hemos añadido una nueva columna para indicar el nombre de la estación de mayor radiación, y hay un mayor número de decimales. Pero, comprobando la suma, los resultados concuerdan.

- **Instalación pseudodistribuida**

  Repetimos el experimento en modo mapreduce. Para ello, nos creamos otro script

  – **Diseño del script**

  El script es idéntico al anterior, simplemente cambiando el bloque que controla la variable `HADOOP_HOME`, para que realice el cambio de la ruta en sentido inverso.

  El script se llamará `run_pig_pseudodist_local.sh` (incluido en dlrectorio), y su código es el siguiente:

```
#!/bin/bash

#Change to Hadoop standalone installation
if test -d /usr/local/hadoop_std
then
    HADOOP_STD=$(grep hadoop_std ~/.bashrc | wc -l)
    if test $HADOOP_STD == 1
    then
        sed -i '/HADOOP_HOME=/s|hadoop_std|hadoop|' ~/.bashrc
        export HADOOP_HOME
        export JAVA_HOME
    fi
fi

rm -rf output_ra*

pig -x local practica2.pig
```

  – **Ejecución**

  Ejecutamos el script

```
bash run_pig_pseudodist_local.sh
```

  La ejecución tardará unos segundos. Si todo ha ido bien, veremos algo como esto:

```
2020-12-25 15:44:23,050 [main] INFO  org.apache.pig.backend.hadoop.execution
- Success!
```

```
2020-12-25 15:44:23,065 [main] INFO  org.apache.pig.Main - Pig
script completed in 6 seconds and 339 milliseconds (6339 ms)
```

En este caso, la ejecución ha durado 6 segundos y 339 milisegundos, un resultado similar al del modo *standalone*.

– **Resultados**

Volvemos a comprobar los resultados. Entramos en `output_radiation/part-r-00000` para ver los resultados de la Tarea 1.

```
Lepe    18.879762046745903
Aroche  17.919251145115762
Moguer  18.341574146820463
Niebla  18.07869350540072
Almonte 18.123927680463083
Gibraleón   18.70950440160565
El Campillo 18.261276060114252
La Palma del Condado    18.071459669487474
La Puebla de Guzmán 17.962850858576193
```

Y en `output_rain/part-r-00000` para ver los resultados de la Tarea 2.

```
Lepe    1999    8.59999993443489
Lepe    2000    639.8000022023916
Lepe    2001    339.9999998062849
Lepe    2002    570.8000042140484
Lepe    2003    877.8000039607286
Lepe    2004    445.59999710321426
Lepe    2005    332.3999994844198
Lepe    2006    613.3999973833561
Lepe    2007    438.9999995082617
Lepe    2008    576.1999998092651
Lepe    2009    561.1999970972538
Lepe    2010    882.9999948441982
Lepe    2011    479.4000007510185
Lepe    2012    430.4000012129545
Lepe    2013    342.39999932050705
Lepe    2014    62.199999421834946
```

Vemos que los resultados concuerdan exactamente con los de la ejecución anterior.

Ejecución en modo mapreduce

Tras la ejecución en modo local, pasaremos a repetir el experimento en modo mapreduce. Este modo aprovecha completamente la arquitectura cluster de Hadoop,

permitiendo hacer uso del HDFS para acceder a los datos y repartir la ejecución en diferentes procesos (dependiendo de la instalación de Hadoop).

Volveremos a repetir el experimento con las dos instalaciones de Hadoop: *standalone* y pseudodistribuida.

- **Instalación *standalone***
  - **Diseño del script**

    La instalación *standalone* no hace uso del HDFS, por lo que todos los accesos a los ficheros se realizarán directamente en local

    El script es similar a los anteriores, cambiando la opción `local` por `mapreduce` en la llamada a Pig.

    Este se llamará `run_pig_std_mr.sh` (incluido en el directorio), y su código es el siguiente:

```
#!/bin/bash

#Change to Hadoop standalone installation
if test -d /usr/local/hadoop_std
then
    HADOOP_STD=$(grep hadoop_std ~/.bashrc | wc -l)
    if test $HADOOP_STD == 0
    then
        sed -i '/HADOOP_HOME=/s|hadoop|hadoop_std|' ~/.bashrc
        export HADOOP_HOME=$HADOOP_HOME
        export JAVA_HOME
    fi
fi

rm -rf output_ra*

pig -x mapreduce practica2.pig
```

  - **Ejecución**

    Ejecutamos el script con

```
bash run_pig_std_mr.sh
```

    Si todo ha ido bien, veremos algo similar a esto:

```
  2020-12-25 17:19:09,318 [main] INFO  org.apache.pig.backend.hadoop.execution
- Success!
  2020-12-25 17:19:09,337 [main] INFO  org.apache.pig.Main - Pig
script completed in 41 seconds and 377 milliseconds (41377 ms)
```

La ejecución ha tardadi 41,37 segundos, un tiempo considerablemente mayor a los obtenidos en el modo local

- **Resultados**

  Revisamos los directorios `output_radiation` y `output_rain` para ver los resultados de las tareas 1 y 2.

  `output_radiation`:

```
Lepe    18.879762046745903
Aroche  17.919251145115762
Moguer  18.341574146820463
Niebla  18.07869350540072
Almonte 18.123927680463083
Gibraleón   18.70950440160565
El Campillo 18.261276060114252
La Palma del Condado    18.071459669487474
La Puebla de Guzmán 17.962850858576193
```

  `output_rain`:

```
Lepe    1999    8.59999993443489
Lepe    2000    639.8000022023916
Lepe    2001    339.9999998062849
Lepe    2002    570.8000042140484
Lepe    2003    877.8000039607286
Lepe    2004    445.59999710321426
Lepe    2005    332.3999994844198
Lepe    2006    613.3999973833561
Lepe    2007    438.9999995082617
Lepe    2008    576.1999998092651
Lepe    2009    561.1999970972538
Lepe    2010    882.9999948441982
Lepe    2011    479.4000007510185
Lepe    2012    430.4000012129545
Lepe    2013    342.39999932050705
Lepe    2014    62.199999421834946
```

  Vemos que se mantienen los mismos resultados de las ejecuciones anteriores.

- **Instalación pseudodistribuida**

  Repetimos el experimento con la instalación pseudodistribuida de Hadoop. En este caso, haremos uso del HDFS, copiando el fichero csv en el directorio `/user/almu`, en donde también se exportarán los directorios `output_radiation` y `output_rain` tras la ejecución del script de Pig.

  - **Diseño del script**

Nos basamos en el script anterior, añadiendo las instrucciones para copiar los datos hacia y desde el HDFS a nuestro directorio. Dado que los directorios de salida no pueden existir antes de la ejecución, los borramos también del HDFS antes de llamar a Pig.

El script se llamará `run_pig_pseudodist_mr.sh`, y su código es el siguiente:

```
#!/bin/bash

USER="almu"
HADOOP_DIR="hdfs://0.0.0.0:9000/user/$USER"

#Change to Hadoop PseudoDistributed installation
if test -d /usr/local/hadoop_std
then
    HADOOP_STD=$(grep hadoop_std ~/.bashrc | wc -l)
    if test $HADOOP_STD == 1
    then
        sed -i '/HADOOP_HOME=/s|hadoop_std|hadoop|' ~/.bashrc
        export HADOOP_HOME=$HADOOP_HOME
        export JAVA_HOME
    fi
fi

echo $HADOOP_HOME

hadoop fs -copyFromLocal RIA_exportacion_datos_diarios_Huelva_20140206.csv
hadoop fs -rm -r -f $HADOOP_DIR/output_radiation $HADOOP_DIR/output_rain

rm -rf output_ra*

pig -x mapreduce practica2.pig

hadoop fs -copyToLocal $HADOOP_DIR/output_radiation ./output_radiation
hadoop fs -copyToLocal $HADOOP_DIR/output_rain ./output_rain
```

– **Ejecución**

Ejecutamos el script con:

```
bash run_pig_pseudodist_mr.sh
```

Si todo ha ido bien, veremos algo similar a esto:

```
2020-12-25 17:30:29,854 [main] INFO  org.apache.pig.backend.hadoop.execution
- Success!
2020-12-25 17:30:29,869 [main] INFO  org.apache.pig.Main - Pig
script completed in 43 seconds and 75 milliseconds (43075 ms)
```

En este caso, la ejecución ha tardado 43,075 segundos, un tiempo ligeramente mas alto que con la instalación *standalone*.

- **Resultados**

  Revisamos los directorios `output_radiation` y `output_rain` para ver los resultados de las tareas 1 y 2 respectivamente.

  `output_radiation`:

```
Lepe    18.879762046745903
Aroche  17.919251145115762
Moguer  18.341574146820463
Niebla  18.07869350540072
Almonte 18.123927680463083
Gibraleón   18.70950440160565
El Campillo 18.261276060114252
La Palma del Condado    18.071459669487474
La Puebla de Guzmán 17.962850858576193
```

  `output_rain`:

```
Lepe    1999    8.59999993443489
Lepe    2000    639.8000022023916
Lepe    2001    339.9999998062849
Lepe    2002    570.8000042140484
Lepe    2003    877.8000039607286
Lepe    2004    445.59999710321426
Lepe    2005    332.3999994844198
Lepe    2006    613.3999973833561
Lepe    2007    438.9999995082617
Lepe    2008    576.1999998092651
Lepe    2009    561.1999970972538
Lepe    2010    882.9999948441982
Lepe    2011    479.4000007510185
Lepe    2012    430.4000012129545
Lepe    2013    342.39999932050705
Lepe    2014    62.199999421834946
```

  Comprobamos que, en ambos casos, los resultados son exactamente los mismos que las ejecuciones anteriores.

Conclusión

La ejecución sobre Apache Pig genera los mismos resultados que los de la Práctica 1, a cambio de tener mas flexibilidad en la división de las tareas MapReduce, y mas facilidad en su implementación.

A cambio, los tiempos son mas elevados que los de la práctica anterior, especialmente en el modo mapreduce, donde los tiempos pasan de unos 7 segundos a 45. Las diferencias en la instalación de Hadoop no producen unas diferencias significativas a nivel temporal; aunque sí se ven grandes diferencias en la forma de la ejecución y el acceso a los datos, especialmente al aplicar el modo mapreduce sobre la instalación pseudodistribuida, en donde se aprovecha el HDFS al máximo.

Es posible que el acceso a la información en el HDFS enlentezca la ejecución, provocando estos grandes retrasos en la misma. También podría deberse al tiempo requerido para la división de los datos entre los *mapper* y *reducer*, que ocupe un tiempo de ejecución significativo. Al ser un fichero tan pequeño, este tiempo no es compensable durante el procesamiento.

Es posible que, con un conjunto de datos mayor, los retardos producidos por estos dos factores se puedan compensar con la reducción de tiempos de la ejecución distribuida.

Apache Hive

Apache Hive es otra plataforma que permite ejecutar tareas MapReduce sobre Hadoop, programándolas desde alto nivel. A diferencia de Pig, Hive funciona con un modelo similar a los de las bases de datos relacionales, realizando las consultas en un lenguaje similar a SQL.

Instalación de Hive

La instalación de Hive tiene una complejidad bastante mayor a la de Pig. Hive hace uso de bases de datos para almacenar la información, por lo que es necesario instalar un Sistema Gestor de Bases de Datos para su funcionamiento. En este caso, utilizaremos Apache Derby, que viene ya integrado dentro de Hive. Aunque es posible instalar Derby de forma externa, por dificultades técnicas utilizaremos la instalación ya integrada en Hive.

También es necesario configurar un sistema HDFS para su funcionamiento, el cual se podrá utilizar para almacenar la información del conjunto de datos y sus resultados, de forma semejante a Pig.

Para la instalación, nos basaremos en estos tutoriales, tomando el primero como base para la instalación, y el último para la configuración

- Hive - Installation - Tutorials Point
- How to Install Apache Hive on Ubuntu - PhoenixNAP
- Apache Hive Installation on Ubuntu - edureka!

En este caso, debido a requerimientos de Hive, solo utilizaremos la instalación pseudodistribuida de Hadoop.

Descarga e instalación de Hive

Empezamos descargando el software. Al igual que con los anteriores, Hive no está disponible en los repositorios de nuestra distribución, así que hay que realizar la instalación de forma manual.

Descargaremos la versión 2.3.7, la última disponible para Hadoop 2 (las versiones 3.X se corresponden con Hadoop 3). Descargamos el fichero terminado en "bin", que contendrá los binarios ya compilados del programa, y lo extraemos en nuestro directorio.

```
wget https://apache.brunneis.com/hive/hive-2.3.7/apache-hive-2.3.7-bin.tar.gz
tar -xvf apache-hive-2.3.7-bin.tar.gz
```

Esto nos creará un nuevo directorio, llamado `apache-hive-2.3.7-bin`, en nuestro directorio actual.

Lo copiamos a su ruta definitiva, en `/usr/local/hive`

```
sudo cp apache-hive-2.3.7-bin /usr/local/hive
```

Configuración de las variables de entorno

- **Añadiendo nuevas variables al fichero `.bashrc`**

  Una vez copiado el software en su ubicación definitiva, configuramos sus variables de entorno, y añadimos la ruta de instalación al PATH. Para ello, añadimos las siguientes líneas a nuestro fichero `~/.bashrc`

  ```
  #Apache Hive
  export HIVE_HOME=/usr/local/hive
  export PATH=$PATH:$HIVE_HOME/bin
  export CLASSPATH=$CLASSPATH:/$HADOOP_HOME/lib/*:.
  export CLASSPATH=$CLASSPATH:$HIVE_HOME/lib/*:.
  ```

  Esto permitirá invocar a Hive desde la terminal, y permitirá a Hive encontrar las rutas necesarias para su ejecución.

- **Configurando la ruta instalación de Hadoop en Hive**

  Hive necesita saber la ruta donde se encuentra instalado Hadoop, para poder invocarlo en sus tareas. Para ello, definiremos su ruta en su configuración de entorno, en la variable `HADOOP_HOME`.

  La configuración de entorno de Hive se define en el fichero `hive-env.sh` que se almacena en su directorio `conf` dentro de la ruta de instalación. Entramos en la ruta indicada usando el comando `cd`

  ```
  cd $HIVE_HOME/conf
  ```

Este directorio no incluye el fichero `hive-env.sh`, sino que ofrece una plantilla para poder generarlo. Copiamos la plantilla del fichero para generar el fichero de configuración.

```
sudo cp hive-env.sh.template hive-env.sh
```

Y entramos en el fichero para configurar la variable.

```
sudo nano hive-env.sh
```

Buscamos la línea donde se asigna la variable `HADOOP_HOME`, y la modificamos de esta manera, indicando la ruta de nuestra instalación de Hadoop.

```
export HADOOP_HOME=/usr/local/hadoop
```

Con esto, ya Hive será capaz de acceder a Hadoop para ejecutar sus tareas

Creación del directorio de Hive en el HDFS

Utilizando el HDFS configurado en la instalación pseudodistribuida de Hadoop, creamos un nuevo directorio para los datos de Hive. Este se utilizará para almacenar las tablas y metadatos del mismo.

El directorio principal se ubicará en la ruta `/user/hive`, de forma análoga a la ruta utilizada en las implementaciones anteriores con Hadoop y Pig. Los datos se almacenarán en un directorio llamado `warehouse` dentro de la ruta anterior. También crearemos otro directorio llamado `tmp`, que hará de directorio temporal.

Utilizaremos el comando `hdfs dfs` para crear los directorios y asignar los permisos

```
hdfs dfs -mkdir -p /user/hive/warehouse
hdfs dfs -mkdir /tmp
```

Damos permisos de escritura a los miembros del grupo, para facilitar el acceso desde Hive

```
hdfs dfs -chmod g+w /user/hive/warehouse
hdfs dfs -chmod g+w /tmp
```

Configuración del almacenamiento

Terminamos la configuración, indicando en Hive el espacio de almacenamiento a utilizar. Utilizaremos jdbc como conector para acceder a Derby.

Para ello, editamos el fichero `hive-site.xml`, en la configuración de Hive. Este fichero no viene creado por defecto, así que lo creamos:

```
sudo nano $HIVE_DIR/conf/hive-site.xml
```

Dentro del fichero, copiamos las siguientes líneas:

```
<configuration>
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:derby:/usr/local/hive/metastore_db;databaseName=metastore_db;create=true
<description>
JDBC connect string for a JDBC metastore.
To use SSL to encrypt/authenticate the connection, provide database-specific
SSL flag in the connection URL.
For example, jdbc:postgresql://myhost/db?ssl=true for postgres database.
</description>
</property>
<property>
<name>hive.metastore.warehouse.dir</name>
<value>/user/hive/warehouse</value>
<description>location of default database for the warehouse</description>
</property>
<property>
<name>hive.metastore.uris</name>
<value/>
<description>Thrift URI for the remote metastore. Used by metastore
client to connect to remote metastore.</description>
</property>
<property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>org.apache.derby.jdbc.EmbeddedDriver</value>
<description>Driver class name for a JDBC metastore</description>
</property>
<property>
<name>javax.jdo.PersistenceManagerFactoryClass</name>
<value>org.datanucleus.api.jdo.JDOPersistenceManagerFactory</value>
<description>class implementing the jdo persistence</description>
</property>
</configuration>
```

Aplicamos la opción `create=true` para que el metaalmacén se cree de manera automática al arrancar Hive.

Arrancando la base de datos

Una vez con la base de datos y el acceso a la misma ya configurados, arrancamos la misma dentro de Hive.

```
bin/schematool -initSchema -dbType derby
```

Esto inicializará la base de datos, y la añadirá a Hive. Si todo ha ido bien, veremos algo como esto:

```
almu@debian:/usr/local/hive$ bin/schematool -initSchema -dbType derby
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hive/lib/log4j-slf4j-impl-2.6.2.jar!/or
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-lo
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an
explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Metastore connection URL:    jdbc:derby:/usr/local/hive/metastore_db;databaseName=me
Metastore Connection Driver :    org.apache.derby.jdbc.EmbeddedDriver
Metastore connection User:   APP
Starting metastore schema initialization to 2.3.0
Initialization script hive-schema-2.3.0.derby.sql
Initialization script completed
schemaTool completed
```

Probando Hive

Terminada la instalación y las configuraciones, probamos Hive abriendo su intérprete desde la consola. Para ello, ejecutamos el comando `hive`. Si todo ha ido bien, veremos algo como esto

```
almu@debian:/usr/local/hive$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hive/lib/log4j-slf4j-impl-2.6.2.jar!/or
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-lo
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/hive/lib/hive-common-
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 Hive 1.X releases.
hive>
```

Con esto comprobamos que Hive se carga correctamente, pero necesitamos comprobar si el acceso a la base de datos es correcto. Para ello, ejecutamos el comando `show databases` en la consola de Hive.

Si todo ha ido bien, el comando nos mostrará la base de datos por defecto.

```
hive> show databases;
OK
default
Time taken: 4.213 seconds, Fetched: 1 row(s)
```

```
hive>
```

Implementación de las tareas

Una vez con Hive instalado y configurado, pasamos a implementar las tareas. De nuevo, volveremos a implementarlo todo en un único script.

Planteamiento

A diferencia de Pig, Hive almacena los datos en tablas dentro de la base de datos. De esta manera, antes de empezar el procesamiento, deberemos cargar los datos del fichero csv dentro de una tabla ya creada con las mismas columnas que el fichero.

Durante el procesamiento, se irán generando diversas tablas temporales, que almacenarán los datos obtenidos de las sucesivas consultas para su uso posterior. En algunos casos, será necesario almacenar los datos tanto en la tabla temporal como en un fichero de salida. Para ello, se volcarán los datos primero en la tabla, la cual se consultará posteriormente para exportar sus datos al fichero.

Para facilitar el trabajo, los datos se leerán y escribirán directamente en local, dejando el HDFS para las tablas y el procesamiento interno de los datos dentro de Hive.

Carga de los datos del fichero CSV

Antes de empezar el procesamiento, debemos cargar los datos del fichero csv en una tabla. Para ello, debemos crear la tabla con las columnas del fichero, y cargar los datos sobre ella.

Al igual que ya hicimos en Pig, la mayoría de las columnas de la tabla se establecerán como cadenas de caracteres, para resolver los problemas de formato antes de convertirlos a sus tipos correspondientes.

- **Creación del esquema de datos**

  Para mejorar la organización, creamos un esquema de datos llamado `practica2`, dentro del cual se crearán todas las tablas de la práctica. Utilizamos para ello la instrucción `CREATE SCHEMA`, añadiendo la cláusula `IF NOT EXIST` para que mantenga el esquema en caso de existir previamente.

  CREATE SCHEMA IF NOT EXISTS practica2;

- **Creación de la tabla:**

  Dentro del esquema recién creado, creamos una nueva tabla llamada `station_data`, preparada para almacenar los datos del fichero csv. Para ello, añadimos en la nueva tabla los mismos campos del fichero csv (aunque con diferentes nombres).

  Utilizamos la sentencia `CREATE TABLE` para crear la tabla. Añadimos al comando las opciones `ROW FORMAT DELIMITED FIELDS TERMINATED BY ';' STORED`

`AS TEXTFILE`, para indicar que los datos de esa tabla se cargarán desde un fichero de texto cuyos campos están delimitados por el caracter ';'.

Antes de crear la tabla, hacemos un `DROP TABLE` para borrarla en caso de que exista. Esto nos evitará insertar datos repetidos en sucesivas ejecuciones del script.

```
DROP TABLE practica2.station_data;

CREATE TABLE practica2.station_data (IDREGION string, SREGION string,
IDSTATION string,
                                     SSTATION string, SDATE string,
DAY int, TEMPMAX string,
                                     HORMINTEMPMAX string, TEMPMIN
string, HORMINTEMPMIN string,
                                     TEMPMID string, HUMIDITYMAX
string, HUMIDITYMIN string, HUMIDITYMID string,
                                     SPWIND string, DIRWIND string,
RADIATION string, RAIN string
                                     )
                                  ROW FORMAT DELIMITED
                                  FIELDS TERMINATED BY ';'
                                  STORED AS TEXTFILE;
```

- **Carga de los datos desde el fichero csv**

  Una vez creada la tabla, pasamos a cargar los datos desde el fichero csv. Para ello, utilizamos la instrucción `LOAD DATA INTO`, añadiendo la opción `LOCAL INPATH`, seguido de la ruta del fichero, para indicar que los datos se cargarán desde un fichero, el cual está almacenado de forma local.

  ```
  LOAD DATA LOCAL INPATH './RIA_exportacion_datos_diarios_Huelva_20140206.csv'
  INTO TABLE practica2.station_data;
  ```

  Esto cargará todos los datos del fichero en la tabla, aplicando las conversiones a los tipos de datos de cada columna de la misma. En este caso, la mayoría de ellos se convertirán a cadenas de caracteres.

Tarea 1: Cálculo de las medias de radiación de cierto grupo de estaciones

Una vez cargados los datos en la tabla, comenzamos el procesamiento de la Tarea 1. Esta se realizará en varias partes: filtrado de los datos de las estaciones de interés, y cálculo de las medias de radiación

Al final de cada parte se almacenarán los datos en una tabla temporal. La tabla creada en el primer paso se utilizará también para la tarea 2.

- **Tarea 1.1: Filtración de los datos de las estaciones de interés**

En primer lugar, realizaremos un filtrado de las filas cuyos ID correspondan a las estaciones que queremos analizar, y los guardamos en una tabla temporal llamada `station_filtered`. Antes de guardar los datos en la tabla, utilizamos la selección de la consulta para aplicar las correcciones de formato necesarias y transformar cada dato a su tipo correspondiente. Para poder aprovechar este trabajo en la Tarea 2, seleccionamos tanto las columnas del nombre de la estación y su radiación, como su fecha de medida y su valor de precipitaciones.

- **Creación de la tabla temporal**

  Para poder almacenar los datos en una tabla temporal, es necesario crearla previamente. Para ello, utilizamos la sentencia `CREATE TEMPORARY TABLE`, añadiendo la cláusula `IF NOT EXISTS` para que no la cree en caso de existir previamente.

```
  CREATE TEMPORARY TABLE IF NOT EXISTS practica2.station_filtered(SSTATION
STRING, RADIATION float,

   YEAR int, RAIN float);
```

- **Consulta y carga de los datos en la tabla**

  Creada la tabla, lanzamos la consulta mediante una sentencia `SELECT FROM WHERE`, utilizando el operador `BETWEEN` dentro del `WHERE` para filtrar las estaciones cuyo ID esté entre 2 y 10. Para almacenar los resultados en la tabla, utilizamos la sentencia `INSERT TABLE`, ejecutando la consulta dentro de ella, para que los datos obtenidos se almacenen automáticamente dentro de la misma.

```
  INSERT OVERWRITE TABLE practica2.station_filtered
          SELECT REGEXP_REPLACE(SSTATION, '\"', ''),
                  CAST(REGEXP_REPLACE(RADIATION, ',', '.') as float),
                  CAST(SUBSTR(SDATE, 7, 10) as int) as YEAR,
                  CAST(REGEXP_REPLACE(RAIN, ',', '.') as float)
          FROM practica2.station_data
          WHERE CAST(REGEXP_REPLACE(IDSTATION, '\"', '') as int)
BETWEEN 2 AND 10;
```

  Dentro de la consulta, aplicamos las siguientes transformaciones:

  - ⋆ **Eliminación de las comillas:** Para ello, usamos la instrucción `REGEX_REPLACE()` sobre el campo filtrado, indicando que reemplace las comillas por una cadena vacía.
  - ⋆ **Corrección del separador decimal:** Dado que Hive utiliza el punto como separador decimal, utilizamos otra instrucción `REGEX_REPLACE()` para reemplazar la coma por el punto en los campos decimales

- ⋆ **Filtrado de subcadenas:** Para extraer el año de la fecha, utilizamos la instrucción `SUBSTR()` para extraer los caracteres del 7º al 10º de dicha cadena.
- ⋆ **Conversión de tipos:** Utilizamos la instrucción `CAST(x as type)` para aplicar conversiones de los datos transformados a sus tipos correspondientes.

- **Tarea 1.2: Cálculo de las medias de radiación**

  En segundo lugar, a partir de los datos de la consulta anterior, calcularemos la media de radiación de cada estación y la guardaremos en un fichero. De nuevo, crearemos otra tabla temporal, llamada `avg_rad_table`, para almacenar los resultados, que serán reutilizados en la Tarea 2.

  – **Cálculo de las medias**

    Para calcular la media de radiación, realizaremos una consulta `SELECT FROM GROUP`, agrupando los datos por el nombre de la estación (campo `SSTATION`), y aplicando la función `AVG()` sobre el campo radiación dentro del `SELECT`. El nuevo campo con la media de radiación se llamará `AVG_RADIATION`.

    Añadimos a la instrucción `INSERT` la cláusula `OVERWRITE` para que sobreescriba la tabla en caso de existir, con los nuevos resultados.

```
  CREATE TEMPORARY TABLE IF NOT EXISTS practica2.avg_rad_table(SSTATION
STRING, AVG_RADIATION FLOAT);
  INSERT OVERWRITE TABLE practica2.avg_rad_table
          SELECT SF.SSTATION, AVG(SF.RADIATION) as AVG_RADIATION
          FROM practica2.station_filtered as SF
          GROUP BY SF.SSTATION;
```

  – **Volcado de los datos en el fichero**

    Una vez almacenados los resultados en la tabla temporal `avg_rad_table`, utilizamos una nueva consulta `SELECT FROM, para seleccionar todos los datos de la tabla. Esta sentencia se ejecutará dentro de una instrucción`ÌNSERT OVERWRITE LOCAL DIRECTORY`, que volcará los datos automáticamente a un fichero de texto almacenado de forma local en el directorio indicado, sobreescribiendo los datos en caso de existir.

    Para indicar el formato de almacenamiento de los datos, aplicamos las claúsulas `ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'`, que almacenará los datos en forma de columnas delimitadas por tabuladores.

```
  --Export data to a output file
  INSERT OVERWRITE LOCAL DIRECTORY './output_radiation' ROW FORMAT
DELIMITED FIELDS TERMINATED BY '\t'
          SELECT *
          FROM practica2.avg_rad_table;
```

Esto almacenará los datos de la tabla en el directorio `output_radiation`, dentro de nuestra ruta actual. Los datos se almacenarán en formato (nombre_estación, radiación media).

Tarea 2: Cálculo de la suma de precipitaciones por año de la estación de mayor radiación media

Con el cálculo de la radiación media de cada estación, pasamos a calcular la suma de precipitaciones de la estación de mayor radiación. Esto lo realizaremos en dos partes: en primer lugar averiguamos el nombre de dicha estación, y posteriormente buscamos los valores de precipitaciones y el año de dichas medidas para dicha estación, y los sumamos agrupados por año.

Para ello, utilizaremos dos de las tablas generadas en la Tarea 1: `station_filtered`, que contiene los datos de precipitaciones y el año de dichas medidas de cada estación, ya filtrados y convertidos a tipos numéricos; y `avg_rad_table`, que contiene las medias de radiación de cada estación.

En este caso, dado que no tiene sentido utilizar una tabla para guardar un solo dato (y Hive no permite guardar el resultado de la consulta en una variable), se realizarán todas las consultas de forma anidada, en lugar de utilizar una tabla temporal.

- **Tarea 2.1: Búsqueda del nombre de la estación con mayor radiación media**

  Empezamos cálculando el valor máximo de radiación media, y el valor al que pertenece. De nuevo, mantendremos la estrategia utilizada en Pig, ordenando la tabla en sentido descendente según el valor de radiación media, y obteniendo la primera fila de la tabla ordenada. Para ello, utilizaremos una subconsulta anidada en el campo `FROM` de la primera consulta. La consulta anidada seleccionará el nombre de la estación, junto a un ranking (`rank_rad`) en función de los valores de radiación ordenadas de forma descendente. Posteriormente, en la consulta principal compararemos el valor de `rank_rad` en el campo `WHERE` para filtrar la fila de la primera posición del ranking, y seleccionaremos de ella el nombre de la estación (`SSTATION`) en el campo `SELECT`.

  Por comodidad, almacenaremos la consulta en una variable. Hive no permite guardar los resultados de una consulta dentro de una variable, por lo que la variable almacenará la consulta en sí, que luego se ejecutará dentro de la consulta principal.

  Definimos la variable con la instrucción `SET`, aplicando el prefijo `hivevar:` para que almacene la variable dentro de Hive.

```
  --Pre-task 2: Get the name of the station with maximum average
radiation
  SET hivevar:MAX_AVG_STA=(SELECT SSTATION
                           FROM (SELECT SSTATION,
```

```
                                    RANK() OVER (ORDER BY AVG_RADIATION
DESC) as rank_rad
                                    FROM practica2.avg_rad_table) sort_tab
                                WHERE sort_tab.rank_rad=1);
```

De esta manera, la variable `MAX_AVG_STA` almacenará la consulta `SELECT FROM WHERE` que se utilizará para obtener el nombre de la estación con mayor radiación media,

- **Tarea 2.2: Cálculo de la suma de precipitaciones agrupadas por año, de la estación de mayor radiación media**

  Finalmente, con la consulta del nombre de la estación ya preparada, lanzamos la consulta principal para obtener la suma de precipitaciones agrupadas por año, y almacenamos sus resultados en un fichero.

  Para almacenar los resultados en un fichero, volvemos a utilizar la instrucción `INSERT OVERWRITE LOCAL DIRECTORY` con la ruta del directorio de salida y el formato del fichero. Dentro de ella, escribimos la consulta a ejecutar, de la cual se almacenarán los resultados dentro del fichero.

  La consulta principal se realizará sobre la tabla `station_filtered`, obtenida al comienzo de la tarea 1, que contiene los datos de precipitaciones y del año de dichas medidas. Para obtener las sumas de precipitaciones agrupadas por año, utilizamos una sentencia `GROUP BY` sobre los campos del nombre de la estación y el año, y aplicaremos una función `SUM()` dentro del `SELECT` para sumar las precipitaciones de dicha agrupación. También seleccionamos los campos del nombre de la estación y el año. En el campo `WHERE` de esta consulta, aplicamos la consulta almacenada en la variable anterior, y comparamos su valor con el del campo `SSTATION`, correspondiente al nombre de la estación.

  ```
    --Calculate the sum of rain of each year, from this station, Export
  results to file
    INSERT OVERWRITE LOCAL DIRECTORY './output_rain' ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t'
            SELECT SSTATION, YEAR, SUM(RAIN) as SUM_RAIN
            FROM practica2.station_filtered
            WHERE SSTATION IN ${hivevar:MAX_AVG_STA}
            GROUP BY SSTATION,YEAR;
  ```

  Los resultados de la consulta se almacenarán en el directorio `output_rain`, en formato (nombre_estación, año, suma_precipitaciones), con las columnas separadas por tabuladores.

Ejecución

Una vez implementadas las tareas del script, pasamos a ejecutarlo desde Hive. El script se llama `practica2.hql`, y lo ejecutaremos con la orden

```
hive -f practica2.hql
```

Hive no necesita que los directorios de salida no existan previamente, sino que, en caso de existir, simplemente sobreescribe su contenido. De igual forma, la lectura y la escritura de los ficheros se realizarán directamente en local. Por esta razón, para esta versión del proyecto no utilizaremos ningún script adicional para ejecutar las tareas.

En este caso, realizaremos un único experimento. Hive utiliza la instalación pseudodistribuida de Hadoop, y no dispone de diferentes modos de ejecución (Al conectarse a Hadoop, aplica el MapReduce internamente, sin requerirlo al usuario), así que no existen mas variantes a comprobar en la ejecución del proyecto.

A la hora de medir los tiempos, y dado que Hive podría haber preservado las tablas de ejecuciones anteriores, se borrarán todos los datos de dichas ejecuciones, tanto las tablas como el esquema de datos, y se desactivará la instrucción `DROP TABLE` de la tabla principal. Esto nos permitirá medir los tiempos de forma justa y precisa, partiendo de una situación inicial sin ningún dato precargado en la base de datos.

- **Borrado de los datos anteriores**

  Para borrar el esquema y las tablas, entramos a la consola de Hive y ejecutamos los comandos `DROP TABLE` y `DROP SCHEMA`.

  ```
  almu@debian:~/Practicas_BigData/Practica2/Hive$ hive
  hive> drop table practica2.station_data;
  OK
  Time taken: 4.318 seconds
  hive> drop table practica2.station_filtered;
  OK
  Time taken: 0.068 seconds
  hive> drop table practica2.avg_rad_table;
  OK
  Time taken: 0.054 seconds
  hive> drop schema practica2;
  OK
  Time taken: 0.216 seconds
  hive>
  ```

  Comprobamos que se han borrado correctamente

  ```
  hive> show databases;
  OK
  default
  Time taken: 0.162 seconds, Fetched: 1 row(s)
  hive> show schemas;
  OK
  default
  ```

```
  Time taken: 0.056 seconds, Fetched: 1 row(s)
  hive> select * from practica2.station_data;
  FAILED: SemanticException [Error 10001]: Line 1:14 Table not found
'station_data'
  hive>
```

Vemos que el esquema no aparece en el listado, y que la consulta a la tabla principal indica que esta no existe.

- **Ejecución del script**

  Pasamos a ejecutar el script para analizar sus tiempos y comprobar el resultado

  ```
  hive -f practica2.hql
  ```

  Si todo ha ido bien, veremos al final algo como esto

  ```
  MapReduce Jobs Launched:
  Stage-Stage-3:  HDFS Read: 14487452 HDFS Write: 14487078 SUCCESS
  Stage-Stage-4:  HDFS Read: 14487452 HDFS Write: 14487078 SUCCESS
  Stage-Stage-2:  HDFS Read: 14487452 HDFS Write: 14487078 SUCCESS
  Total MapReduce CPU Time Spent: 0 msec
  OK
  Time taken: 10.714 seconds
  ```

Desglosado de la ejecución

Hive no devuelve los resultados de la ejecución completa, sino que devuelve los tiempos y el resultado de cada instrucción individual. Por ello, hemos de recorrer la salida del comando para analizar los tiempos y la ejecución de cada tarea.

Cada línea de `OK` se corresponde a la ejecución de una de las instrucciones del comando, y está acompañada por una línea llamada `Time taken`, con el tiempo requerido para la misma.

- **Conexión**

  El comando comienza estableciendo la conexión con Hadoop y la base de datos.

  ```
  almu@debian:~/Practicas_BigData/Practica2/Hive$ hive -f practica2.hql
  SLF4J: Class path contains multiple SLF4J bindings.
  SLF4J: Found binding in [jar:file:/usr/local/hive/lib/log4j-slf4j-impl-2.6.2.ja
  SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf
  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/hive/lib/hive-co
Async: true
```

- **Creación del esquema**

  Una vez conectado, ejecuta la primera instrucción para crear el esquema de bases de datos: `CREATE SCHEMA IF NOT EXISTS practica2;`

```
  OK
  Time taken: 4.588 seconds
```

  Vemos que ha durado 4,6 segundos.

- **Creación de la tabla principal y carga de los datos**

```
  Creación de la tabla principal `station_data`: `CREATE TABLE practica2.station_
( ...)`

      OK
      Time taken: 0.468 seconds

  Carga de los datos del fichero: `LOAD DATA LOCAL INPATH ...`

      Loading data to table practica2.station_data
      OK
      Time taken: 0.802 seconds
```

- **Creación de la tabla temporal `station_filtered`:** `CREATE TEMPORARY TABLE IF NOT EXISTS practica2.station_filtered(...)`

```
  OK
  Time taken: 0.121 seconds
```

- **Ejecución de la primera consulta, con el filtrado de las columnas e inserción en `station_filtered`**: `INSERT OVERWRITE TABLE practica2.station_filtered SELECT REGEXP_REPLACE(SSTATION, '\"', ''), ...`

  Vemos que la consulta se ha dividido en 1 tarea *map*, con 3 etapas *resolver*. No hay *reducer* en este caso.

```
  Query ID = almu_20201226160318_92aa595a-740d-4cbd-8108-cd27221b94b9
  Total jobs = 3
  Launching Job 1 out of 3
  Number of reduce tasks is set to 0 since there's no reduce operator
  Job running in-process (local Hadoop)
  2020-12-26 16:03:21,659 Stage-1 map = 100%,  reduce = 0%
  Ended Job = job_local307619113_0001
  Stage-4 is selected by condition resolver.
  Stage-3 is filtered out by condition resolver.
```

```
Stage-5 is filtered out by condition resolver.
```

Vemos también la escritura de los datos de la consulta en la tabla almacenada en el HDFS. Observamos que los datos se almacenan en el directorio `/tmp` antes de cargarlos en tabla. Esta tarea requiere de una única etapa.

```
Moving data to directory hdfs://0.0.0.0:9000/tmp/hive/almu/95e8132d-948b-49f2-9
Loading data to table practica2.station_filtered
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 6115588 HDFS Write: 7243272 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 2.947 seconds
```

- **Creación de la segunda tabla temporal, `avg_rag_table`:** `CREATE TEMPORARY TABLE IF NOT EXISTS practica2.avg_rad_table(...)`

```
OK
Time taken: 0.093 seconds
```

- **Segunda consulta, con carga en la tabla `avg_rad_table`:** `INSERT OVERWRITE TABLE practica2.avg_rad_table SELECT SF.SSTATION, AVG(SF.RADIATION) as AVG_RADIATION ...`

  Vemos que la consulta se divide en una tarea con una sola etapa con un *map* y un *reduce*.

```
Query ID = almu_20201226160321_bed15775-659d-41d6-9091-4ebfc1e52e93
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data
size: 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>
Job running in-process (local Hadoop)
2020-12-26 16:03:23,257 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local1741341745_0002
```

Vemos también la carga de los datos dentro de la tabla. Esto requiere de una sola etapa.

```
Loading data to table practica2.avg_rad_table
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 14486544 HDFS Write: 14486811 SUCCESS
```

```
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 1.448 seconds
```

- **Tercera consulta, y volcado de los datos al directorio `output_radiation`:** `INSERT OVERWRITE LOCAL DIRECTORY './output_radiation' ... SELECT *`

  Vemos que la consulta requiere de una tarea *map*, con una única etapa.

```
Query ID = almu_20201226160323_a1fe7b31-05f2-4ccc-83ea-6ea84b399a4b
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Job running in-process (local Hadoop)
2020-12-26 16:03:24,614 Stage-1 map = 100%,  reduce = 0%
Ended Job = job_local749066614_0003
```

  También vemos la carga de los datos en el directorio local `output_radiation` desde el HDFS. Esto requiere de una sola etapa.

```
Moving data to local directory output_radiation
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 7243539 HDFS Write: 7243539 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 1.316 seconds
Warning: Value had a \n character in it.
```

- **Cuarta consulta, y carga de los datos en `output_radiation`:** `INSERT OVERWRITE LOCAL DIRECTORY './output_rain' ... SELECT SSTATION, YEAR, SUM(RAIN) as SUM_RAIN ...`

  Esta es muchísimo mas compleja que las anteriores, y requiere de 3 tareas, con 4 etapas. Vemos que las etapas 3 y 4 incluyen tanto *map* como *reduce*.

```
Query ID = almu_20201226160324_6bb45ea0-9c57-4ab6-86cd-57f65a536a4b
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks not specified. Estimated from input data
size: 1
Job running in-process (local Hadoop)
2020-12-26 16:03:26,121 Stage-3 map = 100%,  reduce = 100%
Ended Job = job_local1298522935_0004
Launching Job 2 out of 3
Number of reduce tasks not specified. Estimated from input data
size: 1
Job running in-process (local Hadoop)
```

```
  2020-12-26 16:03:27,303 Stage-4 map = 100%,  reduce = 100%
  Ended Job = job_local1554512246_0005
```

También vemos la carga de los datos al directorio temporal del HDFS. Para ello, se crea una tarea local dentro de Hadoop.

```
  2020-12-26 16:03:32 Starting to launch local task to process map
join;  maximum memory = 477626368
  2020-12-26 16:03:33 Dump the side-table for tag: 0 with group count:
9 into file: file:/tmp/almu/95e8132d-948b-49f2-9620-7ef7af786bb8/hive_2020-12-26_
  2020-12-26 16:03:33 Uploaded 1 File to: file:/tmp/almu/95e8132d-948b-49f2-9620-
(526143 bytes)
  2020-12-26 16:03:33 End of local task; Time Taken: 1.29 sec.
  Execution completed successfully
  MapredLocal task succeeded
```

Y la tercera tarea MapReduce, con 3 etapas con escrituras al HDFS.

```
  Launching Job 3 out of 3
  Number of reduce tasks not specified. Estimated from input data
size: 1
  2020-12-26 16:03:35,341 Stage-2 map = 100%,  reduce = 100%
  Ended Job = job_local1013871726_0006
  Moving data to local directory output_rain
  MapReduce Jobs Launched:
  Stage-Stage-3:  HDFS Read: 14487452 HDFS Write: 14487078 SUCCESS
  Stage-Stage-4:  HDFS Read: 14487452 HDFS Write: 14487078 SUCCESS
  Stage-Stage-2:  HDFS Read: 14487452 HDFS Write: 14487078 SUCCESS
  Total MapReduce CPU Time Spent: 0 msec
  OK
  Time taken: 10.714 seconds
```

Análisis de tiempos

Haciendo recopilación vemos los siguientes tiempos:

- **Creación del esquema:** 4,6 segundos
- **Creación de la tabla principal:** 0,468 segundos
- **Carga de los datos del fichero csv en la tabla principal:** 0,802 segundos
- **Creación de la tabla temporal `station_filtered`:** 0,121 segundos
- **Primera consulta y carga de los datos en `station_filtered`:** 2,94 segundos
- **Creación de la tabla temporal `avg_rad_table` :** 0,1 segundos
- **Segunda consulta y carga de los datos en `avg_rad_table`:** 1,448 segundos

- **Tercera consulta y volcado de los datos al directorio `output_radiation`**: 1,316 segundos
- **Cuarta consulta y volcado de los datos al directorio `output_rain`**: 10,714 segundos

---

Total: 22,509 segundos

Comprobamos que los tiempos son mas elevados que los de la Práctica 1 y el modo local de Apache Pig. Pero son considerablemente mas reducidos que los obtenidos con el modo *mapreduce* de Pig.

Observamos que las operaciones que consumen mas tiempo son las consultas, en especial aquellas que requieren subconsultas anidadas. La última consulta, con 3 niveles de anidación, ha ido la que ha consumido mas tiempo, suponiendo algo mas del triple de tiempo que la segunda operación mas larga.

Resultados

Con la ejecución ya completada, pasamos a revisar los resultados obtenidos, para compararlos con los de las implementaciones anteriores.

Entramos en el directorio `output_radiation` para ver el resultado de la primera tarea

```
Almonte 18.123928
Aroche  17.91925
El Campillo 18.261276
Gibraleón   18.709505
La Palma del Condado    18.071459
La Puebla de Guzmán 17.96285
Lepe    18.879763
Moguer  18.341574
Niebla  18.078693
```

Observamos que los resultados son muy parecidos a los obtenidos con Pig, con diferencias en el orden de los elementos y la precisión decimal.

Entramos en el directorio `output_rain` para ver el resultado de la segunda tarea

```
Lepe    1999    8.59999993443489
Lepe    2000    639.8000022023916
Lepe    2001    339.9999998062849
Lepe    2002    570.8000042140484
Lepe    2003    877.8000039607286
Lepe    2004    445.59999710321426
Lepe    2005    332.3999994844198
Lepe    2006    613.3999973833561
```

```
Lepe    2007    438.9999995082617
Lepe    2008    576.1999998092651
Lepe    2009    561.1999970972538
Lepe    2010    882.9999948441982
Lepe    2011    479.4000007510185
Lepe    2012    430.4000012129545
Lepe    2013    342.39999932050705
Lepe    2014    62.199999421834946
```

En este caso, los resultados son prácticamente idénticos a los obtenidos con Pig.

Comparativa general de tiempos

Recopilando los tiempos de la Práctica 1 y las diferentes pruebas con Apache Pig y Hive, obtenemos los siguientes

- **Ejecución directa de los procesos mapreduce sobre consola, conectados con tuberías:** 0,351 segundos
- **Hadoop Streaming (*standalone* y pseudodistribuida):** ~6 segundos
- **Apache Pig:**
  - *standalone*-local: 6,4 segundos
  - pseudodistribuida-local: 6,34 segundos
  - *standalone*-mapreduce: 41,37 segundos
  - pseudodistribuida-mapreduce: 43,075 segundos.
- **Apache Hive (pseudodistribuida con mapreduce):** 22,509 segundos

Comprobamos como la implementación MapReduce de la Práctica 1 sigue siendo la mas rápida. La ejecución en *streaming* afecta mucho a los tiempos de la Práctica 1, pero aún así, siguen siendo mas reducidos que en el resto de implementaciones.

Notamos como la implementación de Pig incrementa mucho sus tiempos al ejecutarse en modo mapreduce. Mientras los tiempos del modo local son apenas superiores a los de la Práctica 1 sobre Hadoop Streaming, en el modo mapreduce aumentan significativamente hasta mas de 40 segundos.

En cambio, la implementación de Hive logra que el incremento de tiempo de la ejecución en MapReduce sobre HDFS se reduzca, quedándose en unos 23 segundos, notablemente inferiores a los mas de 40 segundos de la ejecución mapreduce de Pig.

Conclusiones

Apache Pig y Apache Hive son dos herramientas bastante diferentes para un mismo propósito: facilitar la implementación de tareas MapReduce sobre Hadoop.

Pig requiere un proceso de instalación mucho mas sencillo que el de Hive, además de tener una sintaxis algo mas fácil. Por contra, Hive permite una ejecución

mas óptima, haciendo uso de una base de datos incrustada dentro del HDFS. Mientras Pig admite dos modos de ejecución, permitiendo ejecutar el script sin utilizar MapReduce ni HDFS, en Hive ambos son de uso obligado y se utilizan automáticamente sin que el usuario lo solicite.

Hive también es mas versátil, permitiendo el uso de bases de datos externas, conexión distribuida, varios intérpretes de comandos; e incluso pudiéndose ejecutar sobre otros entornos como Spark (en lugar de Hadoop). Esto lo hace mas potente que Apache Pig, aunque el proceso de instalación lo hace mas complicado de desplegar y configurar.

En nuestro caso, no hemos podido aprovechar todos los recursos de Hive, al presentar dificultades en la configuración de la conexión con host y puertos, y el levantamiento de las bases de datos externas. Pero, pese a esto, los resultados han sido mas que aceptables, mejorando bastante los resultados obtenidos con Pig.

Tras las pruebas, concluimos que ambas herramientas, pese a permitir una programación mas sencilla, tienen el coste de unos tiempos de ejecución mas elevados que la implementación directa MapReduce sobre Hadoop.