BigData

LABORATORIO BIDDATA

PROGRAMACIÓN CON MAPREDUICE

ALCANCE

- Comprender el modelo de programación de Map Reduce
- Familiarizarse con la API mrjob de Python para trabajos de MapReduce
- Resolver problemas simples con Map/Reduce

INTRODUCCIÓN

MapReduce es un framework de procesamiento de datos en paralelo que sigue una estrategia de "dividir y conquistar". Un trabajo se divide en tareas más pequeñas, las cuales se procesan en paralelo en múltiples nodos. En otras palabras, MapReduce es una interfaz simple y poderosa que permite la paralelización y distribución automática de cálculos a gran escala, combinada con una implementación de esta interfaz que logra un alto rendimiento en grandes clústeres de PC de bajo costo.

Las instrucciones, comandos, script y los archivos entregados están desarrollados para ser trabajados en Linux. Para la edición de los ejercicios de Python se recomienda el uso de VisualStudio, pero ustedes son libres de usar el IDE de su preferencia. No se recomienda el uso de Júpiter Notebook, ya que deberá guardar sus desarrollos en archivos *.py y luego ser ejecutados en la línea de comandos. Deberá usar la terminal de línea de comando (CLI) para comprobar los resultados. Recuerdes, en laboratorios futuros usted interactúa con cluster remotos y con Hadoop.

RECURSO A UTILIZAR

Mrjob, Es una biblioteca de Python para implementar la estructura de programación de MapReduice, y se puede desplegar en diferentes configuraciones para el procesamiento de datos:

a. Realizar el test en equipos locales (este será el alcance de este lab)
b. Desplegar el desarrollo en plataformas locales de datos distribuido como Hadoop.
c. Correr en despliegues en la nueve usando cluster de Amazon Elastic MapReduce (EMR) o Google Cloud Dataproc (Dataproc)

En este contexto, mrjob es el framework para los scripts de procesamiento de datos en Python y nos permite la ejecutan en paralelo en una cantidad escalable de nodos, y lo más importante, sin tener que administrar manualmente la infraestructura del cluster.

Documentación: https://mrjob.readthedocs.io/en/latest/

Regexp, es la librería que nos permite por medio de expresiones regulares (regex) buscar y manipular texto.

Documentación: https://docs.python.org/3/library/re.html

BigData

CONFIGURACIÓN INICIAL

1. Se usará Python 3 con el paquete mrjob para el laboratorio. Deberá cargar estos paquetes y realizar la respectiva configuración, por lo tanto, en una CLI, ejecutar:

   `$ Pip3 install mrjob`

   Para más información, revise la documentación de mrjob, o la instalación por medio de pip https://pypi.org/project/mrjob/

2. Usando la CLI de Linux, cree una carpeta para cada proyecto de BigData en el home del usuario de linux. Recuerde el repaso de linux, por ejemplo, ejecute en CLI:

   `$ mkdir BigData`

3. Al interior de la carpeta, anteriormente creada, debe crear una subcarpeta por cada uno de los laboratorios. Para el caso de hoy es `Lab1`.
4. Ahora vamos a usar pequeños datos de entrada los cuales serán almacenados en una carpeta llamada `input` en cada laboratorio. Cree esta carpeta. Para el proceso iniciar de comprensión del modelo de programación en Mapreduce, se va a trabajar con archivos de libros en texto plano, ejemplo con el libro digital ***Tale of Two Cities, Charles Dickens*** que se encuentra en https://www.gutenberg.org/files/98/98-0.txt. Por lo tanto, ejecute en el CLI las siguientes instrucciones:

```
mkdir input && wget https://www.gutenberg.org/files/98/98-0.txt -o input/ToTCities.txt
```

Nota. Recuerde 1) mirar en que carpeta está ejecutando los comandos y 2, asegúrese que comprende cada término usado en la instrucción

5. Crear la estructura de script de MapReduce, para ello puede usar cualquier IDE, ejemplo VisualStudio. Crear el archivo `wordcount.py`, para ellos copie el siguiente esqueleto:

```
# Lab 1. Basic wordcount

from mrjob.job import MRJob
import re

# this is a regular expression that finds all the words inside a String
WORD_REGEX = re.compile(r"\b\w+\b")
# this line declares the class Lab1, that extends the MRJob format.
class Lab1(MRJob):
# this class will define two additional methods: the mapper method goes after this line

# and the reducer method goes after this line

# this part of the python script tells to actually run the defined MapReduce job.
# Note that Lab1 is the name of the class
if __name__ == "__main__":
    Lab1.run()
```

la clase de Python Lab1, imagen superior, necesita implementar los métodos 'mapper' y 'reducer' que definen el cálculo de MapReduce. Para nuestro primer programa, queremos implementar el mismo proceso de conteo de palabras que vimos en la clase anterior. Revise las diapositivas de la conferencia y estudie el pseudocódigo que usamos allí. Recuerda el programa tendrá el siguiente flujo de datos:

- Mapper input: One line of text (a String)
- Mapper output:
  - key: una palabra.
  - value: entero con el valor de 1.
- Reducer output:
  - key: una palabra.
  - value: un estero con el número de ocurrencias de la palabra en todos los mappers

6. Complete los métodos del script. Para el metodo mapper, puede usar:

```
def mapper(self, _, line):
    words = WORD_REGEX.findall(line)
    for word in words:
        yield (word.lower(), 1)
```

Y para el método reducer, puede usar la operación sum().

7. Guarde el script y pruebe ahora el programa MapReduce invocando python en su Terminal.

```
python wordcount.py input/ToTCities.txt > out.txt
```

8. El archivo **out.txt** contiene los resultados de MapReduice. Abrir el archivo y responder ¡Cuál es la palabra que se repite más veces?
9. Ahora, cambie el script para que solo las palabras con 10 o más ocurrencias se almacenan en el archivo de salida. Antes de cambiar el código, piensa si la función mapper o el reduce, es la que debe modificar. El archivo out.txt debería tener menos palabras, limitadas a aquellas con 10 o más ocurrencias. ¿Cuántas palabras tienen 10 o más repeticiones?.
10. Ahora, genere un resultado para todas las palabras del libro, pero donde la salida sea el tamaño de palabras. Es decir, cuantas palabras de 1 letra tiene el documento, cuantas de 2 letras y así … Recuerde que acá debemos cambiar los key del mapper y del reduce. ¿Cuál es la longitud de la(s) palabra(s) más larga(s) en el conjunto de datos Y cuál es la frecuencia de esta palabra?

BigData

EJERCICIOS ADICIONALES

1. Tome un libro digital de https://www.gutenberg.org/ y explore como podría realizar índex invertido. Lo ideal es que cada estudiante tome un libro diferente**. ¿**Cómo se debe modifica el mapper y el reducer?.

2. Descargue un nuevo DataSet del siguiente link tweets2016_olympic_rio.test. Este DataSet contiene una gran colección de mensajes de Twitter recopilados durante los Juegos Olímpicos de Río 2016 de la API de transmisión de Twitter utilizando palabras clave como #Rio2016 o #rioolympics. Los datos se almacenan en formato CSV, y cada línea contiene los siguientes campos:

    a. **epoch_time**: Marca de tiempo UNIX del tweet en milisegundos desde el 01-01-1970.
    b. **tweetId**: ID único del tweet
    c. **tweet**: contenido del tweet que incluye el #hashtags)
    d. **device**: información adicional meta-datos, donde se incluye el dispositivo usado

    ¿Cuántos tweets se publicaron por día? Para ello recuerde que se debe tomar los campos por línea y podría usar como ayuda la siguiente función **fields = line.split(";").** Para saber el día a partir de las marcas de tiempo proporcionada por el DataSet, puede usar la librería time Python y como ejemplo usar el siguiente código:

```
time_epoch = int(fields[0])/1000
day = time.strftime("%d",time.gmtime(time_epoch))
```

    Construya el método **mapper** y **reducer**, a partir de un nuevo proyecto. Ejecute el script de manera local, como lo hizo en el punto 7 del paso anterior. ¿Qué resultados obtuvo?, ¿debe limpiar el dataset? Este proceso se puede mejorar con el método **combiner?.**

3. Construya usted mismo un problema de Bigdata, donde deba utilizar el framework de MapReduce. Para ello siéntase libre de buscar o construir un DataSet público. Y de generar la pregunta a resolver.