

I. TÉCNICAS DE BIG DATA PARA MACHINE LEARNING

- a. **¿Qué es Big Data?**: capacidad de almacenamiento y procesamiento actual disponible.
- b. Actualmente los datos se caracterizan por ser:
 - i. Altamente masivos.
 - ii. Son heterogéneos, es decir, pueden venir de diferentes fuentes o dominios.
 - iii. Su manejo debe ser distribuido.
 - iv. Son de alta complejidad.
- c. ¿Cómo obtener información a partir de este tipo de datos?
 - i. Aumentar el procesamiento.
 - ii. Aumentar el almacenamiento.
- d. Para poder procesar esta cantidad de información los supercomputadores:
 - i. Procesamiento distribuido, clústeres basados en hardware de menor costo.
 - ii. Usa miles de nodos, CPUs y núcleos.
 - iii. El aumento de nodos si es necesario se conoce como escalamiento horizontal.
 - iv. Permite un procesamiento altamente paralelo.
 - v. Foco en alto throughput, confiable, escalabilidad y robustez.
- e. Cualquier solución de hardware se deberá sustentar por:
 - i. Reemplazo de procesamiento centralizado por procesamiento distribuido.
¿Cómo almacenar y procesar grandes volúmenes de datos a la mayor velocidad posible?
 - ii. Reemplazo de hardware especializado por commodity hardware. ¿Cómo aumentar capacidad del sistema de manera barata y sin sufrir pérdida de información?
- f. Se necesitaría un software especializado para hardware distribuido.
- g. **Ecosistema Hadoop**: Conjunto de herramientas de software que provee una base para la gestión de sistema distribuido, más una batería de herramientas para el análisis de grandes volúmenes de datos.
 - i. **HDFS**: Sistema de archivo distribuido que realiza el almacenamiento.

- ii. **YARN**: Administrador de recursos de procesamiento con una API como punto de entrada.
 - iii. **Sqoop y Hive**: Parte de las opciones disponibles para el análisis de datos.
 - iv. **Pig**: Herramienta alternativa para el análisis de datos distribuido.
 - v. **Flume**: Recopilación y movimiento de grandes volúmenes de datos.
- h. **Spark y Spark ML**: Herramienta de aplicación de algoritmo sobre Big Data.
- i. **Google Colaboratory**: Proyecto que permite difundir el conocimiento y facilitar la investigación en el área de Machine Learning.
- j. **Instalación y configuración de Google Colaboratory**:
 - i. Ingresar a la cuenta de Google.
 - ii. Ingresar al Drive y crear un directorio de trabajo para mantener el orden, por ejemplo, **BigDataML**.
 - iii. Antes de crear el primer notebook es necesario conectarnos a colab, clic derecho dentro del directorio y **más > conectar más aplicaciones**.
 - iv. Buscar **colaboratory**.
 - v. Clic sobre la App e instalar.
 - vi. Crear el archivo notebook haciendo clic derecho y eligiendo Google Colaboratory.
 - vii. Antes de programar hay que iniciar la máquina virtual haciendo clic sobre el botón **Conectar**, aparecerá un indicador de memoria y disco utilizado.
 - viii. Otra forma de ejecutar el código es con **Shift + Enter**.
 - ix. Para ejecutar comandos de consola, en una celda de código se antepone el signo de exclamación (!).
 - x. Los **snippets** son parte de código que permite acelerar el desarrollo, por ejemplo, para cargar documentos se hace,
 - 1. Acceder a los snippets (<>).
 - 2. Buscar “upload” e insertar el primer resultado.
 - 3. Ejecutar y listo.

```
▶ from google.colab import files  
  
uploaded = files.upload()  
  
for fn in uploaded.keys():  
    print('User uploaded file "{name}" with length {length} bytes'.format(  
        name=fn, length=len(uploaded[fn])))
```

Elegir archivos Sin archivos seleccionados

k. Instalación de Hadoop:

- i. Creamos un nuevo archivo llamado **instalación_hadoop.ipynb**.
- ii. Seguimos los pasos de las imágenes siguientes:

INSTALACIÓN DE HADOOP POR MEDIO DE SCRIPT

Montamos el disco:

```
[1] from google.colab import drive  
drive.mount('/content/drive')  
  
Mounted at /content/drive
```

Cargamos el File Upload y subimos el archivo para su instalación

```
[2] from google.colab import files  
  
uploaded = files.upload()  
  
for fn in uploaded.keys():  
    print('User uploaded file "{name}" with length {length} bytes'.format(  
        name=fn, length=len(uploaded[fn])))
```

▶ Elegir archivos hadoop_colab_installer.py
• **hadoop_colab_installer.py**(text/x-python) - 13812 bytes, last modified: 29-05-2022 - 100% done
Saving hadoop_colab_installer.py to hadoop_colab_installer.py
User uploaded file "hadoop_colab_installer.py" with length 13812 bytes

Ejecutamos el instalador con:

1. Con Open cargamos el archivo.
2. Con Read leemos el archivo.
3. Con Exec se ejecuta.

```
[4] exec(open('hadoop_colab_installer.py').read())

Active services:
 3169 NodeManager
 2898 ResourceManager
 3395 JobHistoryServer
 2980 NameNode
 3050 DataNode
 3436 Jps
```

I. El ecosistema Hadoop:

- i. Se requiere un enfoque distribuido basado en HW y SW.
- ii. Es una solución para Big Data.
- iii. Proyecto Apache en lenguaje Java.
- iv. Creado por Doug Cutting y Mike Cafarella.
- v. Posee un sistema de archivos distribuido (HDFS) que administra archivos y directorios dentro del clúster.
- vi. Posee un administrador o manejador de recursos del clúster que es independiente de la aplicación (YARN Map Reduce v2).
- vii. Posee una serie de motores de procesamiento distribuido (Pig, Mahout, R Connectors, Hive).
- viii. Entrega 3 grandes beneficios, estos son **eficiencia** (procesamiento de grandes volúmenes de datos en tiempos razonables), **escalabilidad** (permite aumentar las capacidades computacionales agregando nuevas máquinas al clúster) y **confiabilidad** (los mecanismos de tolerancia a fallos vienen incorporados en el diseño).
- ix. **¿Cómo logra sus objetivos?**
 1. Eficiencia -> Localidad de datos evitando transferencia por la red.
 2. Escalabilidad -> Facilitando integración de nuevos nodos.
 3. Confiability -> Implementando replicación de datos.

x. ¿Cuándo usar HDFS?

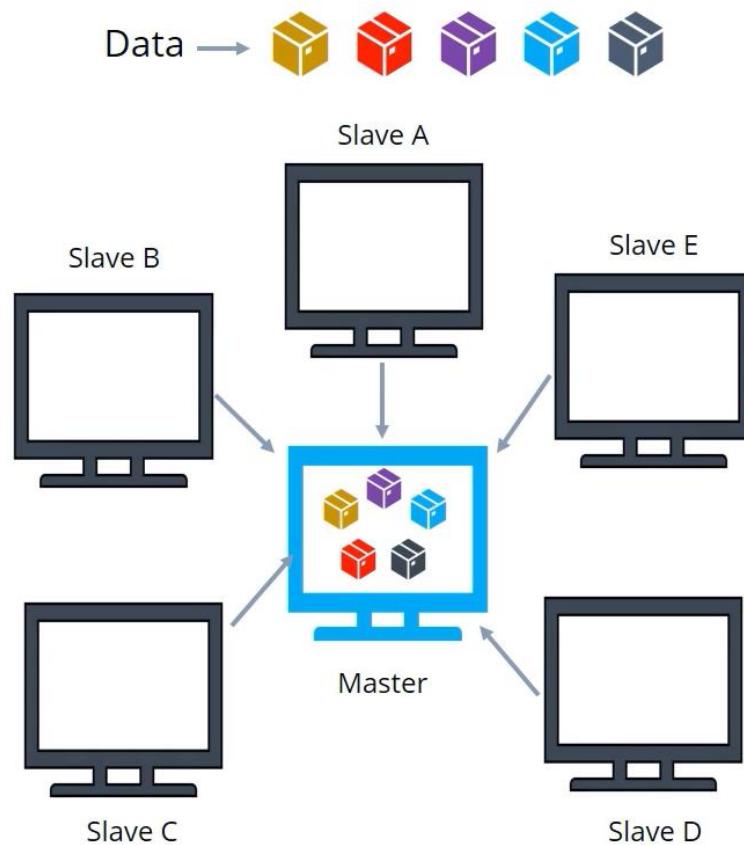
1. Cuando hay grandes cantidades de datos (TERA, PETABYTES).
2. Patrón de acceso tipo streaming.
3. No se requiere modificar los datos: Write once, read many times.

xi. ¿Cuándo no usar HDFS?

1. Acceso con baja latencia.
2. Muchos archivos pequeños.
3. Múltiples tareas que escriben un mismo archivo.

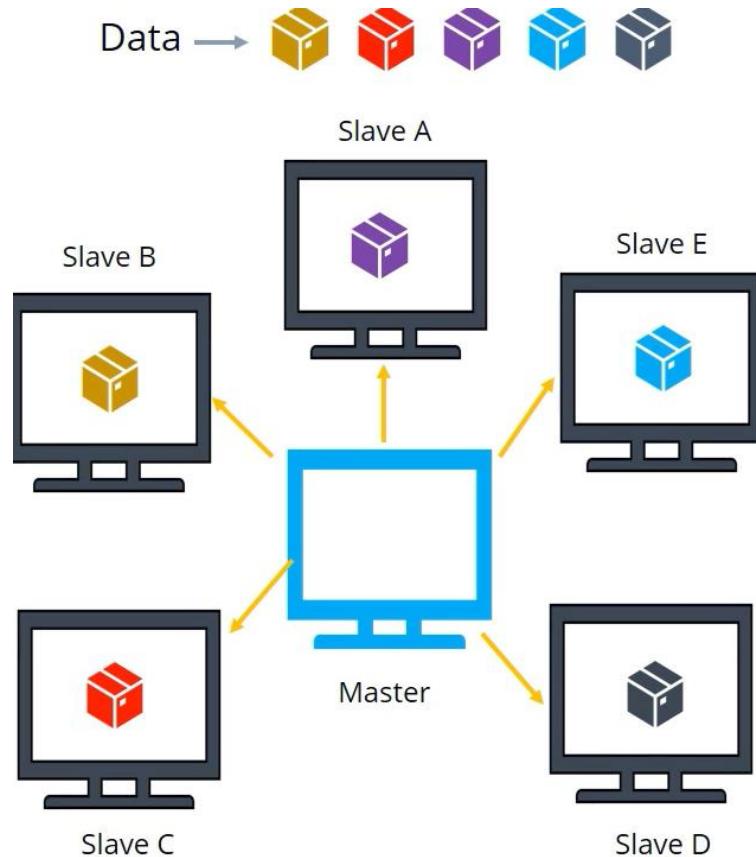
xii. El enfoque tradicional:

1. Mover datos hacia el cómputo (Master).
2. Poca eficiencia para grandes volúmenes de datos.
3. Conocido como escalabilidad vertical, ya que para aumentar el rendimiento se debe aumentar la capacidad del nodo maestro.



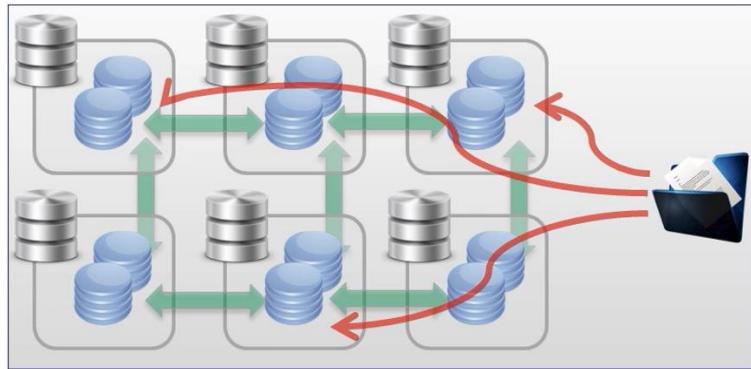
xiii. El enfoque que se usa con hadoop es:

1. Mover el computo hacia los datos, procesa en forma local y los resultados se mueven hacia el master.
2. Se elimina problemas de transporte de datos.
3. Se conoce como escalabilidad horizontal, ya que para aumentar el computo solo vusta con agregar nuevos nodos.



xiv. Hadoop es altamente confiable por la replicación de datos.

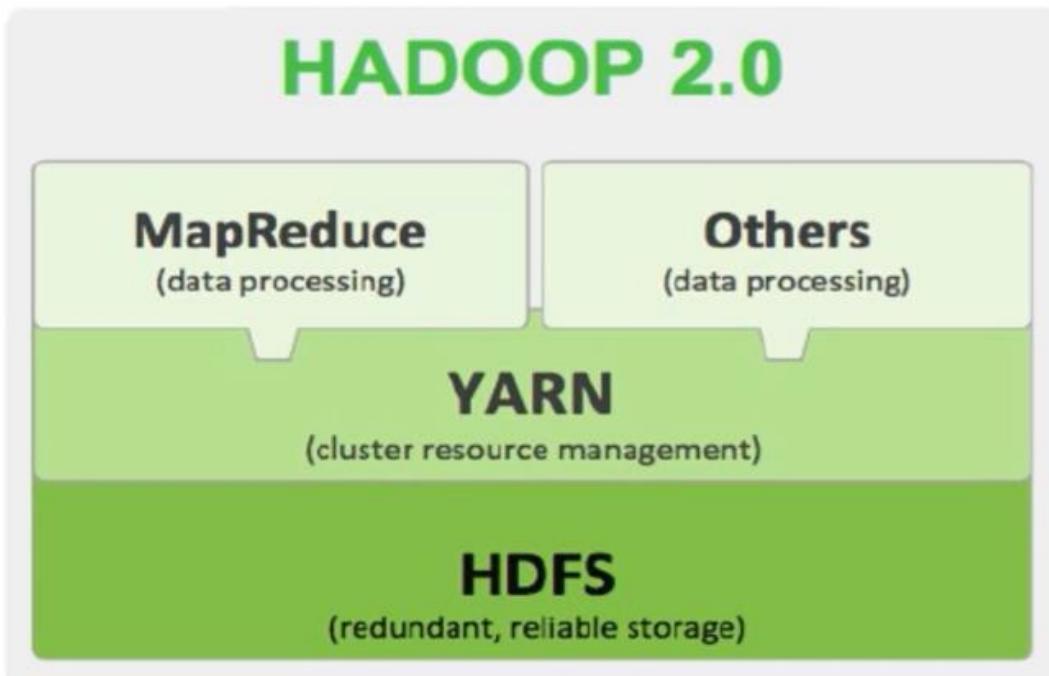
1. Archivos particionados en bloques.
2. Los bloques son distribuidos en el clúster.
3. Los bloques son replicados para mantener redundancia.



xv. Algunas fallas en la replicación de datos:

1. Canal de comunicación perdido (enlaces).
2. Nodos con fallos de HW o SW.

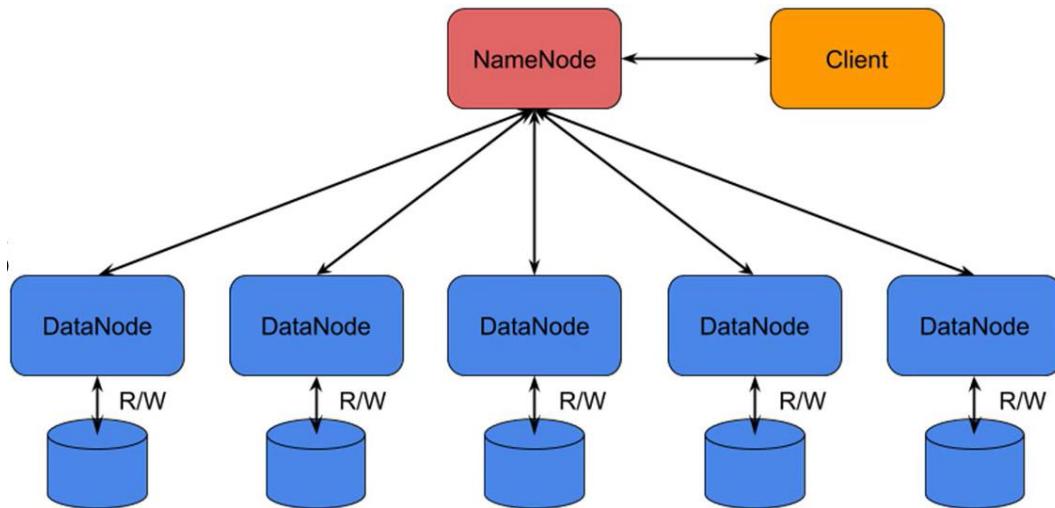
xvi. Hadoop usa arquitectura por módulos:



xvii. HDFS:

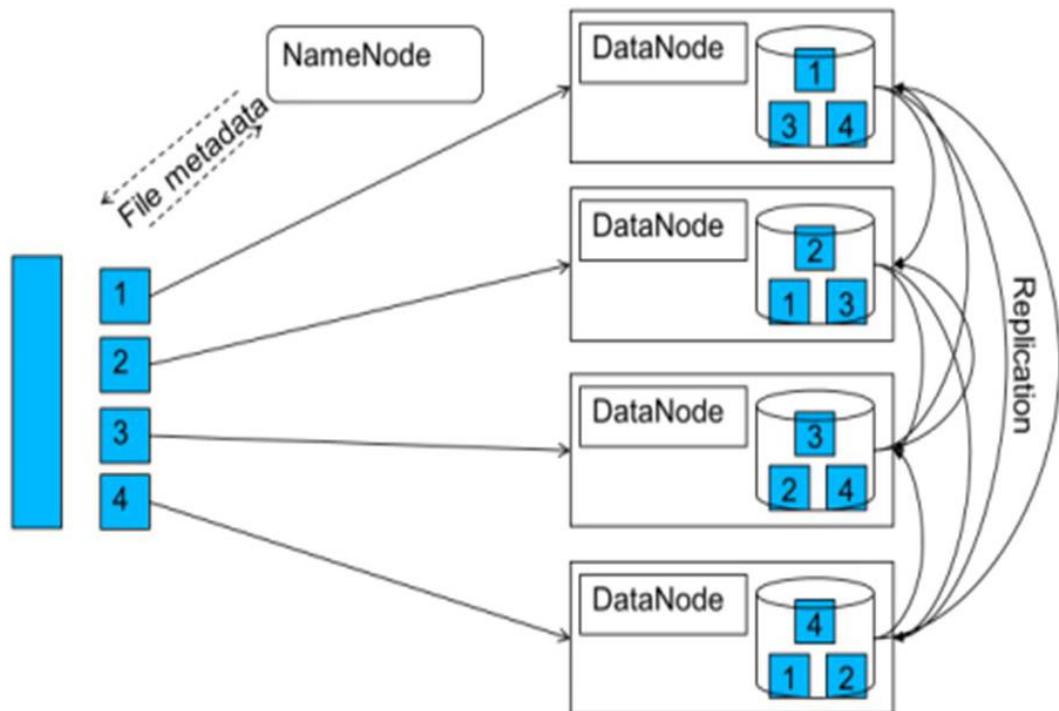
1. Resiliencia.
2. Escalabilidad.
3. Localidad.
4. Portabilidad.
5. Patrón de acceso streaming.

xviii. En HDFS el tamaño por defecto de un bloque es de 128 MB.



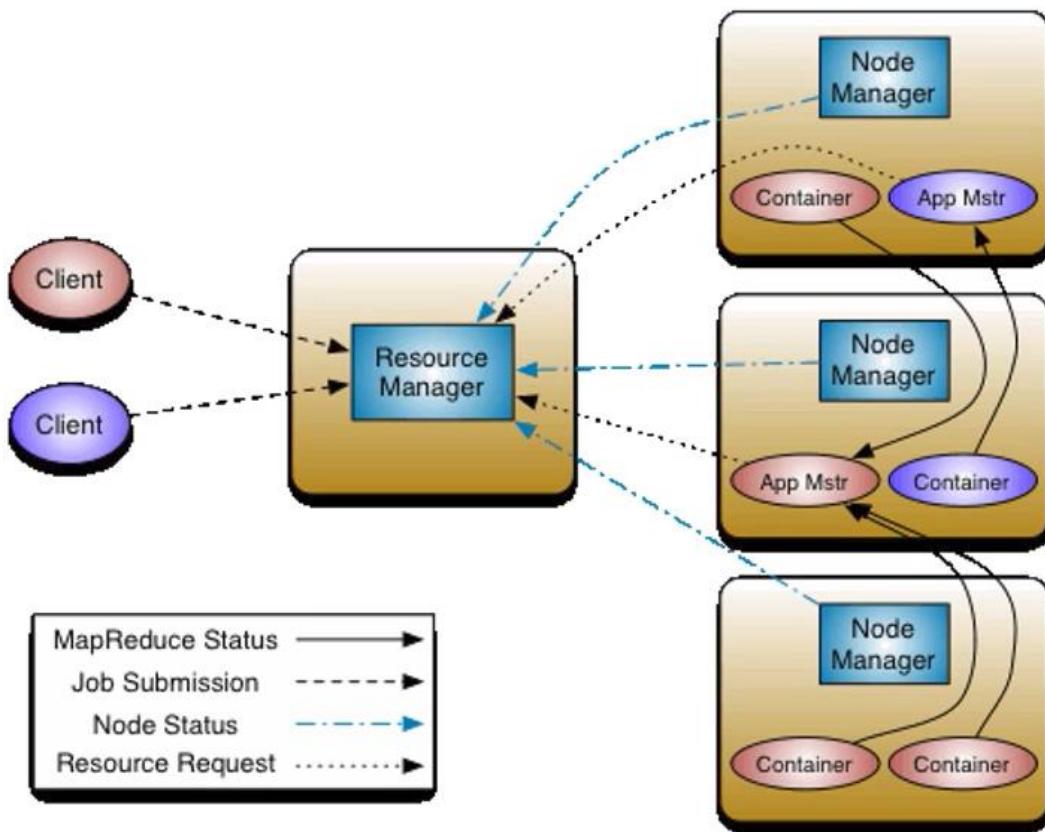
xix. Para almacenar un archivo en HDFS:

1. Particionamiento de archivo en bloques.
2. Escritura de bloques.
3. Replicación.



xx. Yet Another Resource Negotiator (YARN):

1. **Resource Manager**: administrar la asignación de recursos.
2. **Node Manager**: se encarga de containers y monitorea su consumo de recursos (CPU, Memoria, HHDD, NW).
3. **Application Master**: administra la planificación de aplicación y coordinación.
4. Corren en contenedores que son creados por el Resource Manager.

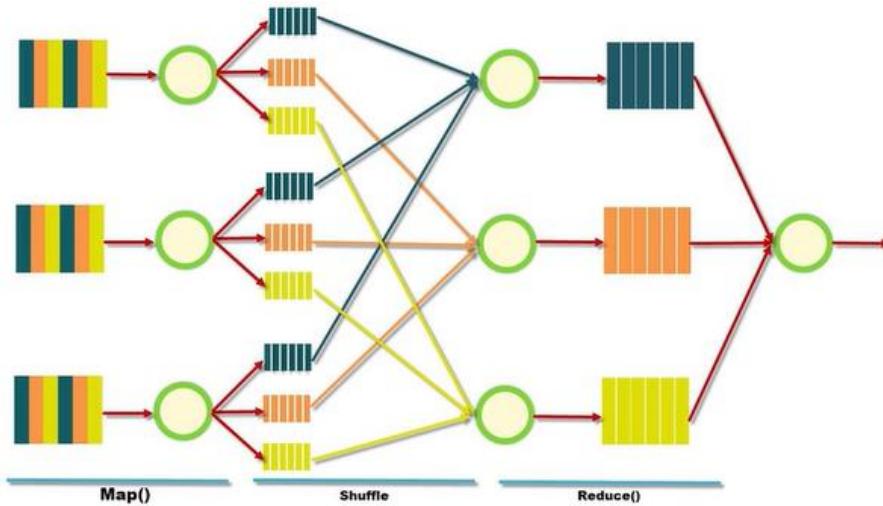


m. Interactuar con HDFS y YARN:

- i. Los comandos HDFS se dividen en 3 categorías; **Usuario** (permite la interacción de los usuarios con el clúster, por ejemplo, copiar archivos), **Administración** (el administrador del clúster permite obtener información de este o realizar modificaciones que cambien su configuración), **Debut** (ayuda a buscar fallas).
- ii. Los comandos YARN igual tiene de **Usuario**, y **Administración**.

n. Hadoop MapReduce:

- i. **MapReduce**, es un modelo de programación de grandes volúmenes de datos, diseñado para llevar de forma distribuida el computo hacia los datos.
- ii. Permite generar cálculos de forma distribuida llevando el computo a los datos.
- iii. Algunas características:
 1. **Localidad de los datos**: los datos no se moverán entre los nodos del clúster, entonces, el procesamiento deberá llevar el computo hacia los datos. Esto da como ventaja el minimizar el tiempo de copiado y minimizar el tráfico en la red.
 2. **Procesamiento paralelo de los datos**: Trabajo en paralelo.
 3. **Interfaz simple**.
 4. **Robustez**: Posee el mecanismo de tolerancia a fallos basado en la redundancia de datos.
- iv. **¿Cómo funciona MapReduce?**
 1. Los datos son divididos en trozos independientes (Split),
 2. Cada Split es procesado por una tarea Map independiente, para generar tuplas <key, value>.
 3. En la etapa shuffle, las tuplas son ordenadas por key, generando una representación intermedia.
 4. Las representaciones intermedias son procesadas por las tareas Reduce, generando el resultado final.



- v. **Conviene** utilizar MapReduce cuando sea la lectura de datos masivos.
- vi. Cuando **no conviene** utilizar MapReduce, cuando se necesario actualizar o modificar datos.
- vii. **Modelo de programación:** Ejemplo MapReduce, contador de palabra:

```

1 from google.colab import files
2
3 uploaded = files.upload()
4
5 for fn in uploaded.keys():
6   print('User uploaded file "{name}" with length {length} bytes'.format(
7     name=fn, length=len(uploaded[fn])))

```

1. Código para subir el instalador de Hadoop.

```

1 from google.colab import drive
2 drive.mount( '/content/drive' )

```

1. Se carga la librería Google y la clase Drive.
2. Se monta la unidad de Google Drive.

```
1 %%writefile word_count_mapper.py
2 #! /usr/bin/python
3
4 import sys
5 import re
6
7 word_counter = dict()
8 for line in sys.stdin:
9     line = line.strip()
10    words = re.findall( '[\w]+', line, re.UNICODE )
11    for word in words:
12        if word in word_counter:
13            word_counter[word] += 1
14        else:
15            word_counter[word] = 1
16    for word in word_counter:
17        print( '%s\t%s' % (word, str(word_counter[word])) )
18
```

```
1 !chmod u+x word_count_mapper.py
```

1. Permite escribir en un archivo Word_count_mapper.py.
2. Interprete de Python.
- 3.
4. Librería para leer el archivo de entrada.
5. Librería para usar expresiones regulares.
- 6.
7. Se crea un diccionario.
8. Se recorre el archivo de entrada, línea x línea.
9. Se quita los espacios inicial y final de la línea.
10. Se crea una lista con cada palabra de la línea.
11. Se recorre la lista, palabra por palabra.
12. Pregunta si se existe la palabra en el diccionario, si es así, aumenta en 1 el valor.
13. Si no está la palabra en el diccionario crea la tupla con el valor en 1.
14. Se recorre el diccionario y se comienza a escribir el archivo.

```
1 !chmod u+x word_count_mapper.py
```

1. Se le da permiso de ejecución al archivo.

```
1 %%writefile word_count_reducer.py
2 #! /usr/bin/python
3
4 import sys
5
6 word_counter = dict()
7 for pair_word_one in sys.stdin:
8     pair_word_one = pair_word_one.strip()
9     key, value = pair_word_one.split( '\t' )
10    if key in word_counter:
11        word_counter[key] += int( value )
12    else:
13        word_counter[key] = int( value )
14
15 for word in sorted( word_counter, key = word_counter.get, reverse = True )[:50]:
16     print( '%s\t%s' % (word, str(word_counter[word])) )
```

1. Del 1 al 16 similar al archivo anterior.

```
1 !chmod u+x word_count_reducer.py
```

1. Se da permiso de ejecución al archivo.

```
1 from google.colab import files
2
3 uploaded = files.upload()
4
5 for fn in uploaded.keys():
6     print('User uploaded file "{name}" with length {length} bytes'.format(
7         name=fn, length=len(uploaded[fn])))
```

1. Se carga el archivo a analizar.

```
1 !hdfs dfs -mkdir my_hadoop_dir
```

```
1 !hdfs dfs -put harry_potter.txt my_hadoop_dir
```

```
1 !hdfs dfs -ls my_hadoop_dir
```

1. Se coloca el archivo en las carpetas para darle orden.

```
1 !hadoop jar hadoop/share/hadoop/tools/lib/hadoop-streaming-2.10.1.jar \
2         -input my_hadoop_dir/harry_potter.txt \
3         -output my_hadoop_dir/wc_output_dir \
4         -mapper /content/word_count_mapper.py \
5         -reducer /content/word_count_reducer.py
6
7
```

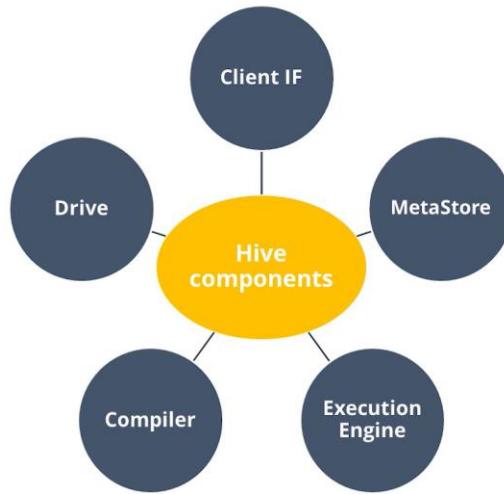
```
1 !hdfs dfs -ls my_hadoop_dir/wc_output_dir
```

```
1 !hdfs dfs -cat my_hadoop_dir/wc_output_dir/part-00000
```

1. Se ejecuta y verifica el MapReduce.

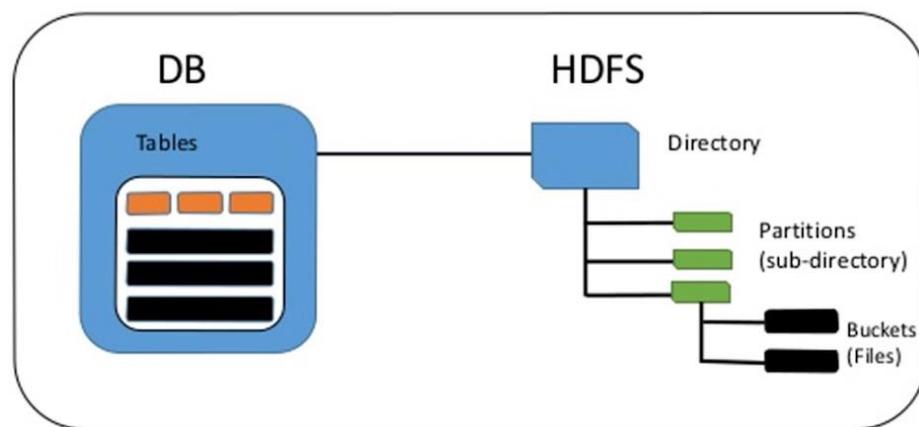
o. Apache Hive:

- i. Permite ejecutar rutinas para altos volúmenes de datos.
- ii. Estas rutinas son realizadas en Sqoop basado en SQL.
- iii. Hive es un lenguaje tipo SQL para procesamiento distribuido.
- iv. Hive permite manejar y consultar datos débilmente estructurados, como si estos fueran estructurados.
- v. Provee una interfaz similar a SQL: HiveQL.
- vi. Utiliza trabajos MapReduce para la ejecución de manera transparente para el usuario.
- vii. Almacena resultados en HDFS.
- viii. Es rápido, escalable y extensible.
- ix. La Arquitectura de Hive es:



1. **CLI o web UI:** Interfaces para clientes permitiendo la interacción.
2. **Driver:** Maneja comando HiveQL recibidos.
3. **Compiler:** Compila los comandos HiveQL.
4. **Metastore:** Almacena el catálogo del sistema y los metadatos.
5. **Execution Engine:** Ejecuta las tareas siguiendo el orden establecido por el grafo acíclico vdirigido.

x. Modelo de datos de Hive:



xi. Otras características:

1. Permite almacenar datos en distintos formatos. Por defecto archivos de texto.
2. No existe in-place insert, update, ni delete sobre tablas.
3. Alta latencia.

xii. Sqoop:

1. Herramienta que permite importar y exportar datos estructurados a Hadoop.
2. Flexibilidad en formato de almacenamiento (Hive, Hbase, texto, etc.)
3. El procesamiento es realizado mediante MapReduce.

xiii. Sqoop en la práctica:

1. Se debe contar con una base de datos relacional, que permite consultar SQL.
2. Luego, basta con ejecutar el siguiente comando, esto generará uno o más archivos en DHFS con el contenido de la tabla:

```
sqoop import --connect jdbc:mysql://host/db  
          --username user  
          --password password  
          --fields-terminated-by '\t'  
          --table table
```

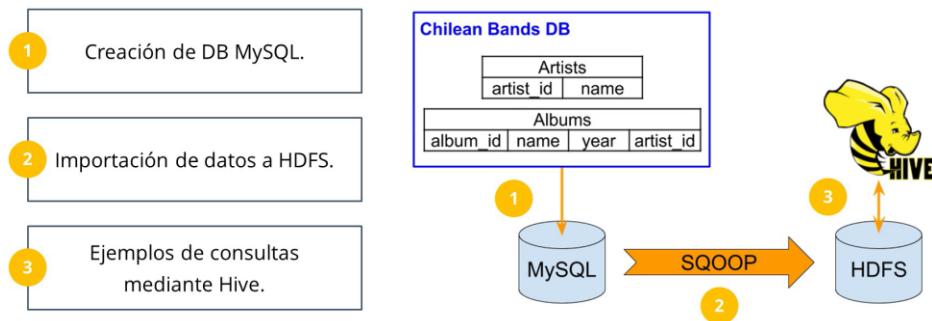
xiv. Cludera Impala:

1. Baja latencia.
2. Permite consultar en leguaje SQL.
3. Utiliza el mismo metastore que Hive.
4. No utiliza MapReduce para consultar.
5. Uso intensivo de RAM.
6. Cada nodo ejecuta un Impala Engine.

xv. Práctica Apache Hive:

1. Apache Hive: plataforma para análisis de datos.
2. HiveSQL brinda gran facilidad al manejo de datos.

3. Sqoop: herramienta para transferencia entre RDBMS y Hadoop, creando el metadata necesario dentro del metastore de hive.



Instalación Hadoop y creación de base de datos

```
[1] 1 from google.colab import drive  
  
[2] 1 drive.mount('/content/drive')  
  
[3] 1 from google.colab import files  
2  
3 uploaded = files.upload()  
4  
5 for fn in uploaded.keys():  
6   print('User uploaded file "{name}" with length {length} bytes'.format(  
7       name=fn, length=len(uploaded[fn])))  
  
[4] 1 exec(open('hadoop_colab_installer.py').read())  
  
[7] 1 # Verificar el contenido del directorio warehouse, inicialmente vacío.  
2 !hdfs dfs -ls /user/hive/warehouse  
  
[8] 1 # Subimos el fichero SQL.  
2 from google.colab import files  
3  
4 uploaded = files.upload()  
5  
6 for fn in uploaded.keys():  
7   print('User uploaded file "{name}" with length {length} bytes'.format(  
8       name=fn, length=len(uploaded[fn])))  
  
[10] 1 # Crear la base de datos testdb e injectar la información del archivo subido.  
2 !mysql -u root --password=password testdb < chilean_bands_database.sql  
  
[11] 1 # Ejecutar la consola MySQL para verificar la creación correcta y  
2 # revisar las tablas con "show tables;", "select * from <nombre_tabla>".  
3 !mysql -u root -p testdb
```

Sqoop

Modo de uso:

```
Sqoop <command> <parameters>
```

```
[12] 1 # Se conecta a MySQL y se listan las bases de datos.  
2 !sqoop list-databases --connect jdbc:mysql://localhost \  
3           --username root \  
4           -P
```

```
[13] 1 # Se conecta a MySQL y se listan las tablas.  
2 !sqoop list-tables --connect jdbc:mysql://localhost/testdb \  
3           --username root \  
4           -P
```

▶

```
1 # Copiar tabla Artists a HDFS.  
2 # Default target-dir: /user/<db_username>  
3 !sqoop import --connect jdbc:mysql://localhost/testdb \  
4           --username root \  
5           -P \  
6           --table Artists \  
7           --target-dir /user/root/artists_table
```

```
[15] 1 # Verificamos el directorio para ver si genero correctamente los ficheros.  
2 !hdfs dfs -ls /user/root/artists_table
```

```
[19] 1 # Se habre uno de los ficheros.  
2 !hdfs dfs -cat /user/root/artists_table/part-m-00000
```

```
[30] 1 # Copiar tabla Albums a HDFS mientras el año de la columna year sea menor a 1995.  
2 !sqoop import --connect jdbc:mysql://localhost/testdb \  
3           --username root \  
4           -P \  
5           --table Albums \  
6           --target-dir /user/root/albums_table \  
7           --where "year < 1995"
```

```
[31] 1 # Verificar el directorio y ficheros.  
2 !hdfs dfs -ls /user/root/albums_table  
  
[33] 1 # Revisar un fichero.  
2 !hdfs dfs -cat /user/root/albums_table/*  
  
[35] 1 # Para almacenar datos y escribir la metadata en el metastore, realizado esto los  
2 # datos serán almacenado dentro del warehouse de hive, se usa --hive-import y  
3 # con --hive-table se le coloca otro nombre a la tabla, si no mantiene el nombre.  
4 !sqoop import --connect jdbc:mysql://localhost/testdb \  
5           --username root \  
6           -P \  
7           --table Albums \  
8           --hive-import \  
9           --hive-table hivealbums  
  
[36] 1 # Verificamos el directorio warehouse.  
2 !hdfs dfs -ls /user/hive/warehouse  
  
[37] 1 # Verificamos el directorio hivealbums.  
2 !hdfs dfs -ls /user/hive/warehouse/hivealbums  
  
[38] 1 # Almacenamos todas las tablas de la base de datos dentro del warehouse de hive.  
2 !sqoop import-all-tables --connect jdbc:mysql://localhost/testdb \  
3           --username root \  
4           -P \  
5           --hive-import  
  
[39] 1 # Verificamos el directorio.  
2 !hdfs dfs -ls /user/hive/warehouse
```

HiveQL

```
[41] 1 # Ingresar a HiveQL.  
2 !hive
```

Escritura de script mediante cell magic "writefile"

```
%%writefile <filename>
```

Ejecución de script HiveQL

```
hive -f <HiveQL query file>
```

```
[42] 1 %%writefile select_01.sql
2 SELECT name, year FROM albums;

[43] 1 !hive -f select_01.sql

[47] 1 %%writefile select_02.sql
2 SELECT artists.name, albums.name, albums.year FROM artists JOIN albums ON artists.artist_id = albums.artist_id;

[48] 1 !hive -f select_02.sql

[51] 1 %%writefile select_03.sql
2 SELECT artists.name, albums.name, albums.year FROM artists JOIN albums ON artists.artist_id = albums.artist_id ORDER BY albums.year;

[52] 1 !hive -f select_03.sql

[53] 1 %%writefile select_04.sql
2 SELECT artists.name, COUNT(*) FROM artists JOIN albums ON artists.artist_id = albums.artist_id GROUP BY artists.name;

[54] 1 !hive -f select_04.sql

[55] 1 %%writefile select_05.sql
2 SELECT * FROM albums WHERE year < 1995;

[56] 1 !hive -f select_05.sql

[57] 1 %%writefile insert_artist.sql
2 INSERT INTO artists VALUES (4, 'Lucybell');
3 SELECT * FROM artists;

[58] 1 !hive -f insert_artist.sql

[59] 1 # Crear una tabla vacia e insertar datos desde un archivo de texto.
2
3 from google.colab import files
4
5 uploaded = files.upload()
6
7 for fn in uploaded.keys():
8   print('User uploaded file "{name}" with length {length} bytes'.format(
9     name=fn, length=len(uploaded[fn])))

[60] 1 !hdfs dfs -mkdir chilean_bands
2 !hdfs dfs -mkdir chilean_albums
3 !hdfs dfs -put chilean_bands.txt chilean_bands
4 !hdfs dfs -put chilean_bands_albums.txt chilean_albums

[62] 1 # El delimitador es la coma ya que asi se realizo el archivo txt.
2 !hdfs dfs -cat chilean_bands/chilean_bands.txt

[63] 1 %%writefile create_tables.sql
2
3 CREATE TABLE chilean_bands (artist_id INT, name STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
4 LOAD DATA INPATH 'chilean_bands' INTO table chilean_bands;
5
6 CREATE TABLE chilean_albums (album_id INT, name STRING, year INT, artist_id INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
7 LOAD DATA INPATH 'chilean_albums' INTO table chilean_albums;
```

```

[66] 1 !hive -f create_tables.sql

[67] 1 !hdfs dfs -ls /user/hive/warehouse

[68] 1 !hive

[69] 1 # Al usar Load data todo su contenido se movera al warehouse de hive, es decir,
2 # los directorios creados deberian estar vacios.
3 !hdfs dfs -ls chilean_albums

[70] 1 # Eliminamos tablas.
2 %%writefile drop_tables.sql
3
4 DROP TABLE chilean_bands;
5 DROP TABLE chilean_albums;

[71] 1 !hive -f drop_tables.sql

[72] 1 # Verificamos la eliminacion
2 !hdfs dfs -ls /user/hive/warehouse

```

Creación de tablas con datos externos

Si se desea que al momento de crear las tablas los archivos no sean movidos al warehouse de hive y no se pierdan cuando se eliminan los datos se hace:

```

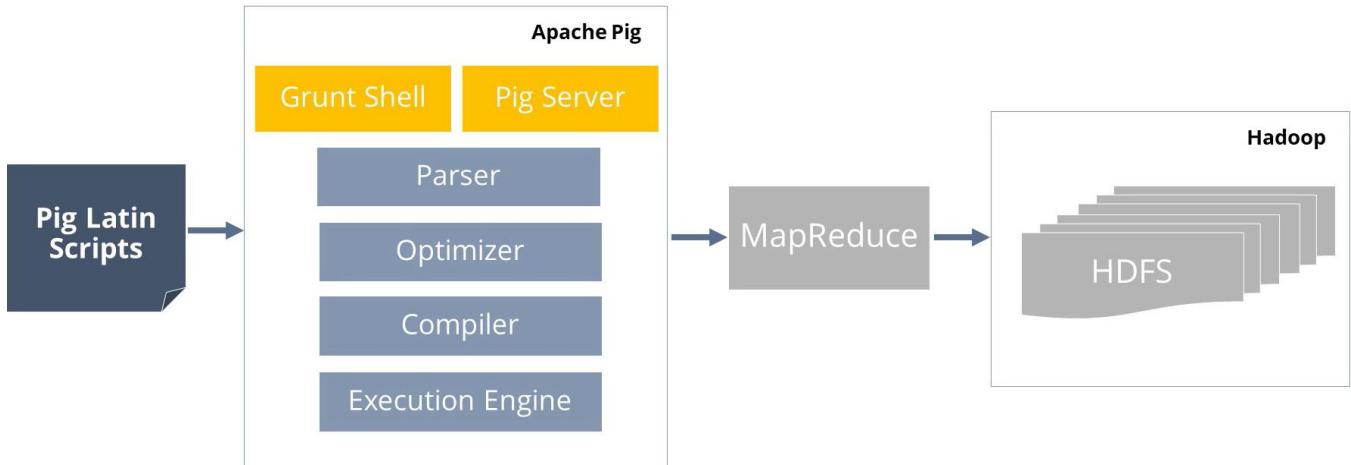
CREATE EXTERNAL TABLE <nombre_tabla> <fields>
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
LOCATION <data_path>;

```

p. Apache PIG:

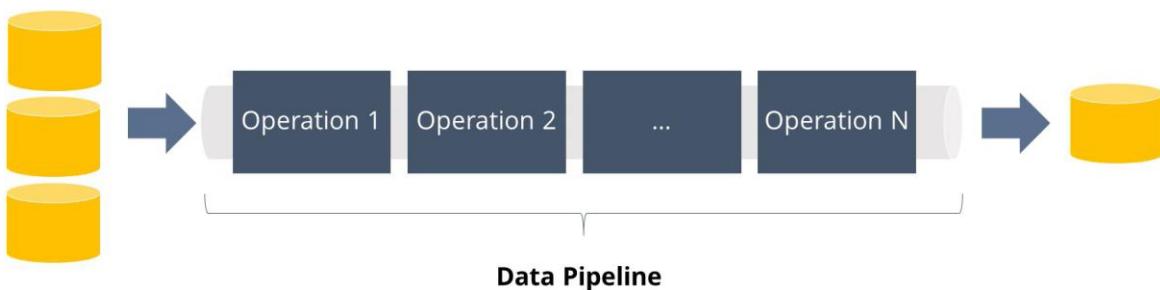
- i. **Análisis de datos con Hive:** SQL puede ser algo restrictivo en su operatoria,
 - 1. Lenguaje declarativo.
 - 2. No hay mucho espacio para optimizar por parte del usuario.
 - 3. Dificultar para almacenar resultados intermedios (checkpoints).
- ii. MapReduce (java) entrega más libertad, pero resulta complejo en desarrollo.
- iii. Una opción atractiva sería un punto intermedio entre java y SQL. Conocido como Apache Pig.
- iv. **Apache Pig**, programa para análisis de big data formada por un lenguaje propietario para la programación de flujos de datos y un ambiente de ejecución para correr dichos programas.
- v. Ocupa un lenguaje de alto nivel llamado **Pig Latin**.
- vi. Posee un compilador que produce secuencias MapReduce.

- vii. Rendimiento similar a MapReduce en java o Python.
- viii. Se enfoca en tareas del tipo DAG (Grafo Acíclico Dirigido).
- ix. Arquitectura Apache Pig:



1. Hay 2 módulos de operación, modulo operativo (Grunt Shell), se despliega con el comando pig. El segundo método es el modo bash con un script (Pig Server – !pig -f <nombre_script>).
2. Al recibir una instrucción se lleva al módulo Parser, que se encarga de comprobar la sintaxis y los tipos de datos generando un DAG que representaran las operaciones y las sentencias ejecutadas.
3. Optimizer, aplicara procedimiento de optimización. A la salida generar una nueva versión de DAG con las mejoras.
4. Compiler, traduce en un conjunto de MapReduce.
5. Execution Engine, ejecutara en orden.
6. Finalizado lo anterior desplegará el resultado o almacenado en hdfs.

- x. Paradigma de pipeline de datos:



1. Datos de entrada a la izquierda.
2. Un conjunto de operaciones a la izquierda.
3. Resultado a la derecha.

xi. Pig Latin:

1. Pig Latin trabaja con relaciones.
2. Fields son los datos.

Fields → datos
María, Iván, ..., 23, 39, ..., 12355987, ...

3. Tupla es una lista ordenada de datos.

Tuples → lista ordenada de fields (datos)
(María, 23, 12355987)

4. Bags es una colección desordenada de tuplas.

Bags → colección desordenada de tuplas
{ (Pedro, 50, 16783987), (Juan, 35, 92648024), (Diego, 82, 46284552) }

5. Relations son Bag externos.

Relations → Bag externo
{ (M, { (Ana, 32, 74856194), (Rosa, 76, 77213509), (Marta, 19, 64728593) }),
(H, { (Pedro, 50, 16783498), (Juan, 35, 92648024), (Diego, 82, 46284552) }) }

6. Tipos de datos:

int	entero de 32-bits con signo
long	entero de 64-bits con signo
float	punto flotante de 32-bits
double	punto flotante de 64-bits
chararray	cadena de caracteres en UTF-8
bytrarray	arreglo de bytes
boolean	booleano
datetime	fecha y hora
biginteger	Java BigInteger
bigdecimal	Java BigDecimal

xii. Sentencias Pig Latin:

Se organizan como:

```

LOAD
:
Transformations to process data
:
DUMP or STORE

```

1. Constructor básico para procesar datos.
2. Operan sobre relaciones para producir relaciones.
3. Puede ser escrita en múltiples líneas, pero finalizadas por “;”.
4. Lectura de datos con LOAD.
5. Generación de resultados con DUMP o STORE.
6. Si no se ejecuta DUMP o STORE, solo se hará validación.

xiii. Ejemplo Pig:

LOAD

```
A = LOAD 'data.txt' USING PigStorage( ',' ) AS ( name: chararray, age: int, gpa: float );
```

DUMP

```
DUMP A;
```

STORE

```
STORE A INTO 'output.txt' USING PigStorage( '|');
```

FILTER:

```
C = FILTER A BY (age >= 18) AND (gpa >= 3.0);
```

FOREACH:

```
D = FOREACH A GENERATE name, $2;
```

GROUP:

```
E = GROUP A BY name;
```

JOIN inner:

```
F = JOIN A BY a1, B BY b1;
```

JOIN outer:

```
G = JOIN A BY a1 [LEFT | RIGHT | FULL ], B BY b1;
```

ORDER:

```
J = ORDER A BY gpa DESC;
```

LIMIT:

```
K = LIMIT J 10;
```

name	year	grade	graduated
pedro	1999	5.5	true
juan	1999	6.8	true
diego	1999	3.1	false
andres	2000	6.2	true
mario	2000	5.9	true
julio	2000	2.3	false

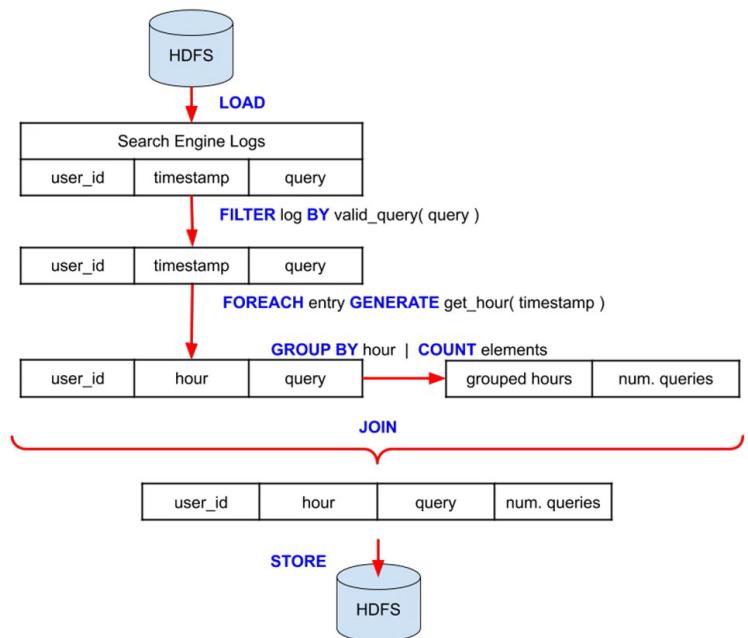
```
A = LOAD 'students.txt' AS (name:chararray, year:int, grade:float, graduated:boolean);
B = FILTER A BY graduated==TRUE;
C = GROUP B BY year;
D = FOREACH C GENERATE group AS year, MAX(B.grade) AS max_grade;
DUMP D;
```

q. Practica Apache Pig:

Datos de entrada

- 1** Eliminaremos queries vacías y URL.
- 2** Extraeremos hora desde el timestamp.
- 3** Calcularemos cantidad de queries por cada hora.

User Id	Timestamp YYMMDDHHMMSS	Search query
5E6E0424B73BB6E3	970916062826	ny times book review
5E6E0424B73BB6E3	970916062853	ny times book review reviews
5E6E0424B73BB6E3	970916062920	ny times book review
5E6E0424B73BB6E3	970916063023	
5E6E0424B73BB6E3	970916063055	
5E6E0424B73BB6E3	970916063121	rosie: anne lamott
5E6E0424B73BB6E3	970916063329	rosie: anne lamott novel
5E6E0424B73BB6E3	970916063453	http://www.typo.com/lamott
5E6E0424B73BB6E3	970916063528	
6B48787CED55274F	970916130238	goldeneye+n64
B2B4FD80D447F15B	970916141152	conflict resolution styles
B2B4FD80D447F15B	970916141159	conflict resolution styles
B2B4FD80D447F15B	970916141253	conflict resolution styles
9BC25E584304AEFA	970916091236	"ribbon stools"
9BC25E584304AEFA	970916091737	change bowel habits



Actividades prácticas con Pig

- Lectura de datos desde HDFS.
- Procesamiento de datos con Pig Latin.
- Escritura de datos a HDFS.

Actividades prácticas con Pig

- Lectura de datos desde HDFS.

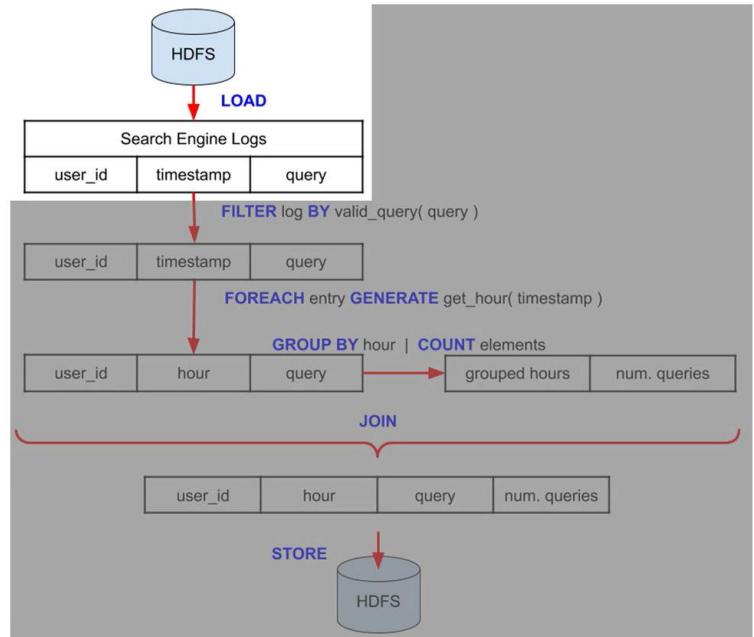


Imagen propia del profesor.

Actividades prácticas con Pig

- Procesamiento de datos con Pig Latin:
 - Filtro de datos por query.

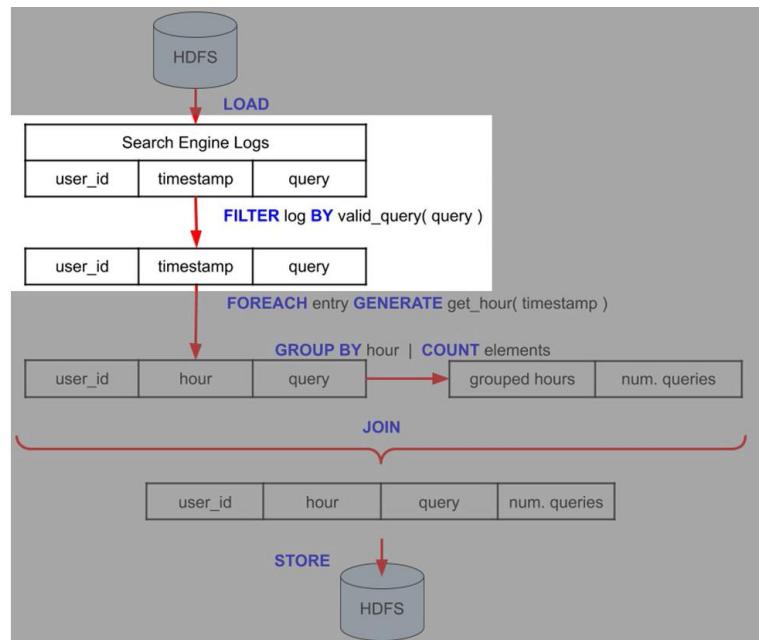
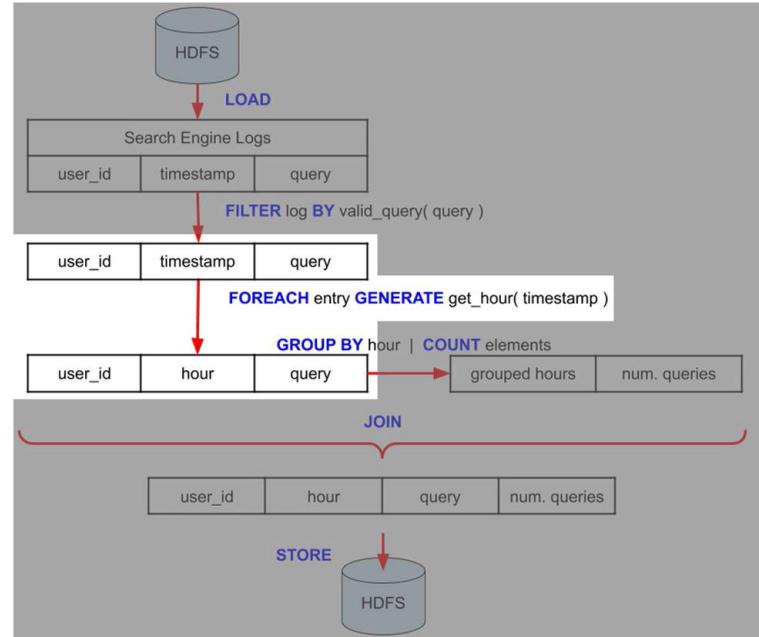


Imagen propia del profesor.

Actividades prácticas con Pig

- Procesamiento de datos con Pig Latin:
 - Extracción de hora.

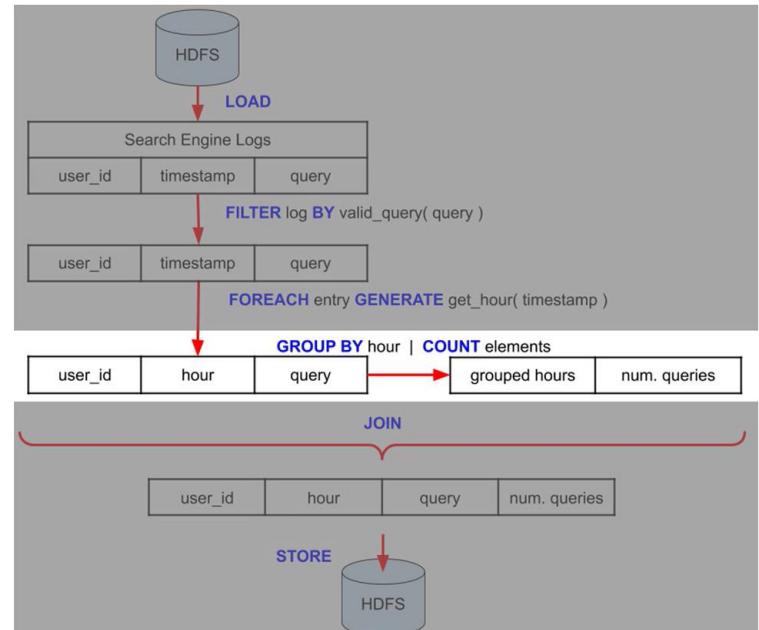
Imagen propia del profesor.



Actividades prácticas con Pig

- Procesamiento de datos con Pig Latin:
 - Agrupación según hora del día.
 - Conteo de elementos por hora.

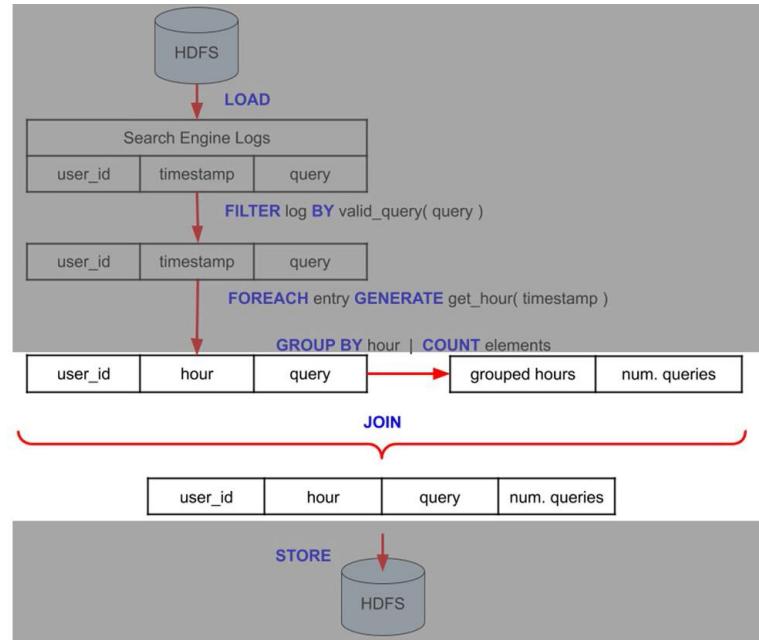
Imagen propia del profesor.



Actividades prácticas con Pig

- Procesamiento de datos con Pig Latin:
 - Mezcla de relaciones.

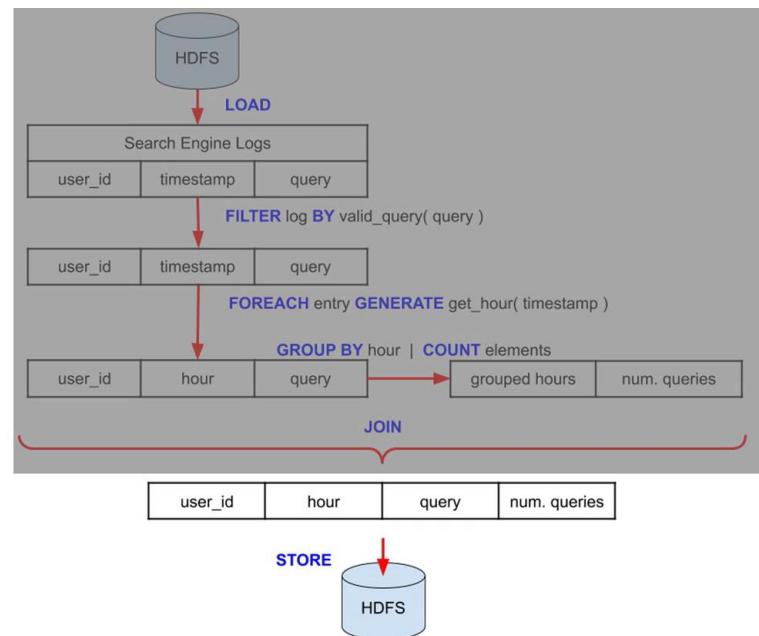
Imagen propia del profesor.



Actividades prácticas con Pig

- Escritura de datos en HDFS.

Imagen propia del profesor.



Apache Pig

Instalación y copia de datos

```
✓ [1] 1 from google.colab import drive  
✓ [2] 1 drive.mount('/content/drive')  
✓ [3] 1 from google.colab import files  
2  
3 uploaded = files.upload()  
4  
5 for fn in uploaded.keys():  
6   print('User uploaded file "{name}" with length {length} bytes'.format(  
7       name=fn, length=len(uploaded[fn])))  
✓ [4] 1 exec(open('hadoop_colab_installer.py').read())  
✓ [5] 1 !tail pig/tutorial/data/excite-small.log  
  
[6] 1 # Copiamos el archivo a hdfs  
2 !hdfs dfs -put pig/tutorial/data/excite-small.log /user/root/
```

Funciones definidas por el usuario (UDFs)

```
[8] 1 %%writefile my_pig_udfs.py  
2  
3 from pig_util import outputSchema  
4 import re  
5 import array  
6  
7 @outputSchema('detected:boolean')  
8 def empty_or_url_detector(query):  
9   if query is None:  
10     return True  
11   query = query.tostring() # 'array' to 'str'  
12   if len(query) == 0 or re.search('(http:)|(https:)|(www.)', query):  
13     return True  
14   return False  
15  
16 @outputSchema('hour:chararray')  
17 def extract_hour(time):  
18   time = time.tostring() # 'array' to 'str'  
19   return time[6:8]  
20
```

Pig Latin

```
[10] 1 !pig
[11] 1 !hdfs dfs -ls
[12] 1 !hdfs dfs -ls num_queries
[13] 1 !hdfs dfs -cat num_queries/part-r-00000
[14] 1 %%writefile practico_pig.pig
2
3 # Registrar el modulo HDFS
4 REGISTER 'my_pig_udfs.py' USING jython AS my_pig_udfs;
5 # Leeremos el archivo
6 raw = LOAD 'excite-small.log' USING PigStorage('\t') AS (user_id, time, query);
7 # Limpian de vacios y URL
8 clean = FILTER raw BY my_pig_udfs.empty_or_url_detector(query) == FALSE;
9 # Creamos una nueva relación
10 houred = FOREACH clean GENERATE user_id, my_pig_udfs.extract_hour(time) as (hour:chararray), query;
11 # Clasificar las query segun las horas
12 grouped_hour = GROUP houred BY hour;
13 # Saber cuantas queries por hora
14 num_queries = FOREACH grouped_hour GENERATE $0 AS hour, COUNT($1) AS count;
15 # Escribir
16 DUMP num_queries;
17 # Combinamos
18 final_result = JOIN houred BY hour, num_queries BY hour;
19 # Guardamos
20 STORE final_result INTO 'num_queries' USING PigStorage(',');
[15] 1 !pig -f practico_pig.pig
```

r. Ayudantía 1:

SETUP

Instalar Haddop y cargar dataset

```
[1] 1 from google.colab import drive
2 drive.mount('/content/drive')
[2] 1 from google.colab import files
2
3 uploaded = files.upload()
4
5 for fn in uploaded.keys():
6   print('User uploaded file "{name}" with length {length} bytes'.format(
7     name=fn, length=len(uploaded[fn])))
[3] 1 exec(open('hadoop_colab_installer.py').read())
```

```
1 from google.colab import files
2
3 uploaded = files.upload()
4
5 for fn in uploaded.keys():
6     print('User uploaded file "{name}" with length {length} bytes'.format(
7         name=fn, length=len(uploaded[fn])))
```

Crear directorios

```
[5] 1 !hdfs dfs -mkdir ramen-ratings
[6] 1 !hdfs dfs -put ramen-ratings.csv ramen-ratings
[7] 1 !hdfs dfs -ls ramen-ratings
[8] 1 !hdfs dfs -cat ramen-ratings/ramen-ratings.csv
```

Apache Hive

```
[9] 1 !head ramen-ratings.csv
```

Crear tabla en hdfs

```
[10] 1 %%writefile create_tables.sql
2 create external table ramen_ratings (review INTEGER, brand STRING, variety STRING, style STRING, country STRING, stars FLOAT)
3 row format delimited
4 fields terminated by '\t'
5 location '/user/root/ramen-ratings'
[11] 1 !hive -f create_tables.sql
```

Ramen japones

```
[12] 1 !hive
[13] 1 %%writefile japanese_ramen.sql
2 SELECT * FROM ramen_ratings WHERE country == 'Japan';
[14] 1 !hive -f japanese_ramen.sql
[15] 1 %%writefile japanese_ramen_by_brand.sql
2 SELECT brand, style, AVG(stars) AS stars_avg FROM ramen_ratings WHERE country == 'Japan' GROUP BY brand, style
[16] 1 !hive -f japanese_ramen_by_brand.sql
[17] 1 %%writefile japanese_ramen_by_brand_sorted.sql
2 SELECT brand, AVG(stars) AS stars_avg FROM ramen_ratings WHERE country == 'Japan' GROUP BY brand ORDER BY stars_avg DESC LIMIT 10
[18] 1 !hive -f japanese_ramen_by_brand_sorted.sql
```

Apache Pig

UDF

Se crea una UDF que va a servir para concatenar 2 campos con un -

```
[25] 1 %%writefile my_udfs.py
2
3 from pig_util import outputSchema
4 import array
5
6 @outputSchema('concat:STRING')
7 def concat(string1, string2):
8     if string1 == None:
9         string1 = 'None'
10    else:
11        string1 = string1.tostring()
12    if string2 == None:
13        string2 = 'None'
14    else:
15        string2 = string2.tostring()
16    return string1 + '-' + string2
```

Mejores cups

- Registramos la udf para poder usarla.
- Cargamos los datos para procesarlo.
- Filtramos por los ratings que corresponden a cups.
- procesamos cada entrada con nuestra función.
- Ordenamos de forma descendente.
- Limitamos la cantidad a los primeros 20.
- Guardamos la información en un directorio usando tabs (\t) como separadores.

```
[36] 1 %%writefile join_brand_variety.pig
2
3 REGISTER 'my_udfs.py' USING python AS my_udfs;
4 raw = LOAD 'ramen-ratings/ramen-ratings.csv' USING PigStorage(',') AS (review, brand, variety, style, country, stars);
5 cups = FILTER raw BY style == 'Cup';
6 joined = FOREACH cups GENERATE my_udfs.concat(brand, variety), style, country, stars;
7 ordered = ORDER joined BY stars DESC;
8 top_20 = LIMIT ordered 20;
9 STORE top_20 INTO 'ramen-top-20' USING PigStorage('\t');
```

```
1 !pig join_brand_variety.pig
```

```
1 !hdfs dfs -ls ramen-top-20
2
```

```
1 !hdfs dfs -cat ramen-top-20/part-r-00000
```

s. Apache Spark:

i. ¿Qué es?

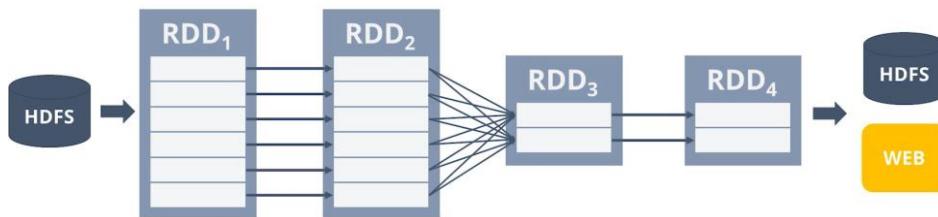
1. Es un ejecutor de procesos de propósitos generales (escalable, robusto y simple).

ii. Lenguajes: Escrito en Java y puede ser ejecutado en múltiples lenguajes (Java, Scala, Python y R).

iii. ¿Por qué usarlo?

1. Uso interactivo y aplicaciones.
2. Rápida integración.
3. Abstracción de hardware.
4. Uso intensivo de RAM de máquinas.
5. Aplicaciones Near Real Time (NRT).

iv. Resilient Distributed Dataset (**RDD**):



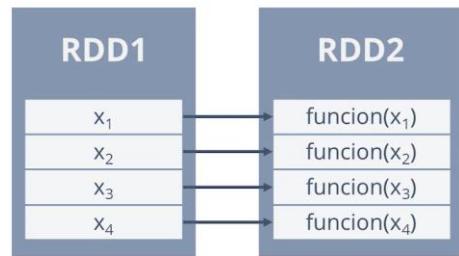
v. Abstracciones de datos en Spark. Los RDD son:

1. Contenedor de datos.
2. Tolerante a fallas (resiliente).
3. Distribuido.
4. Inmutable.
5. Lleva la historia (lineage).
6. Programa de Spark: **workflow sobre transformaciones de RDD**.

vi. Transformaciones y acciones:

1. Transformaciones:

- a. Transforma un RDD en un nuevo RDD.
- b. La transformación es una función de Spark.
- c. Lo que haga la función a cada uno de los datos **no** es de Spark.
- d. Nada se ejecuta hasta una acción (Lazy Execution).



2. Acciones:

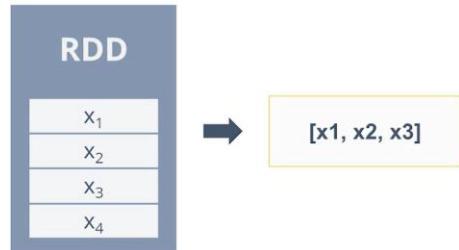
a. Ejecuta el workflow de transformación sobre RDD.

b. Posibles acciones:

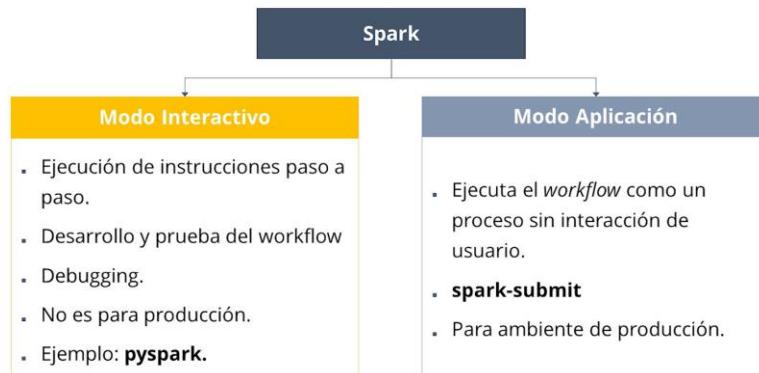
i. Retornar un valor.

ii. Escribir en disco.

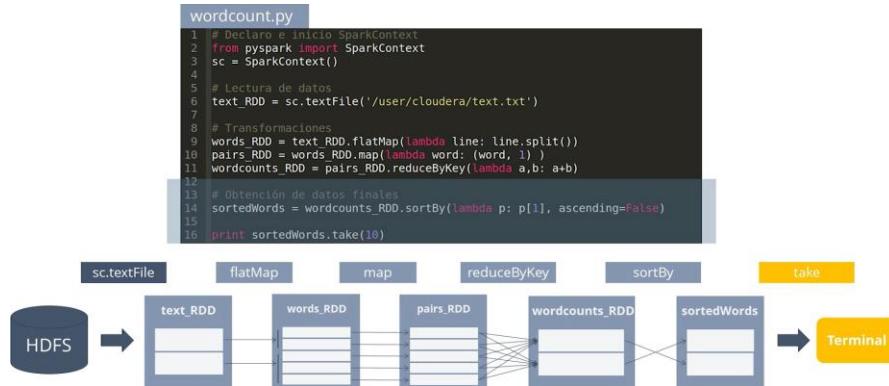
iii. Calcular el resultado de workflow en general.



vii. Modos de ejecución de Spark:



viii. Ejemplo de ejecución de programa en Spark:



ix. Práctica Apache Spark:

1. Para realizar operaciones Modelo de abstracción de datos: RDDs y Dataframes.

Instalación Hadoop y Spark

```
[1] 1 from google.colab import drive
2 drive.mount('/content/drive')

Mounted at /content/drive

[2] 1 from google.colab import files
2
3 uploaded = files.upload()
4
5 for fn in uploaded.keys():
6   print('User uploaded file "{name}" with length {length} bytes'.format(
7     name=fn, length=len(uploaded[fn])))

[Elegir archivos] spark_colab_installer.py
• spark_colab_installer.py(text/x-python) - 14696 bytes, last modified: 13-06-2022 - 100% done
Saving spark_colab_installer.py to spark_colab_installer.py
User uploaded file "spark_colab_installer.py" with length 14696 bytes

[3] 1 exec(open('spark_colab_installer.py').read())
```

Preparación de ambiente

```
● 1 from google.colab import files
2
3 uploaded = files.upload()
4
5 for fn in uploaded.keys():
6   print('User uploaded file "{name}" with length {length} bytes'.format(
7     name=fn, length=len(uploaded[fn])))

[Elegir archivos] novela.txt
• novela.txt(text/plain) - 49906 bytes, last modified: 13-06-2022 - 100% done
Saving novela.txt to novela.txt
User uploaded file "novela.txt" with length 49906 bytes

[5] 1 !hdfs dfs -mkdir my_text_files
2 !hdfs dfs -put novela.txt my_text_files

[6] 1 !hdfs dfs -ls my_text_files
```

1. Se usara la interfaz de programación spark en leguaje python.
2. Esto es provisto por el módulo pyspark, hay que especificar el directorio donde se encuentra el módulo para que el interprete sepa cargarlo.
3. Para facilitar esto se usará el módulo findspark y su función init() que registrara la ruta de pyspark.

```
[7] 1 import findspark
2 findspark.init()
```

Operaciones básicas con RDDs

1. Hay que crear un objeto tipo SparkContext que es el punto de entrada de conexión para utilizar Spark.
2. getOrCreate crea el objeto tipo singleton si es que no existe.

```
[8] 1 from pyspark.context import SparkContext
2 sc = SparkContext.getOrCreate()
3 rdd_textfile = sc.textFile('my_text_files/novela.txt')
```

```
[9] 1 print(type(rdd_textfile))
<class 'pyspark.rdd.RDD'>
```

Acciones

1. Count(): Obtiene el número de elementos (cada línea es un número de elementos).
2. First(): Obtiene el primer elemento.
3. take: Arreglo con la cantidad de elementos.

```
[10] 1 rdd_textfile.count()
500
```

```
[11] 1 rdd_textfile.first()
'Harry Potter and the Sorcerer's Stone '
```

```
[12] 1 rdd_textfile.take(10)
```

Transformaciones

1. Filter: recibe una función y retorna un nuevo rdd.
2. Otra opción es usar las funciones lambda.
3. Map: transforma los datos originales generando un nuevo rdd.
4. Reduce: reduce los elementos del mapeo.

```
[13] 1 def has_harry(line):
2     return 'harry' in line.lower()
```

```
[14] 1 pipelined_rdd = rdd_textfile.filter(has_harry)
```

```
[15] 1 pipelined_rdd = rdd_textfile.filter(lambda line: 'harry' in line.lower())
```

```
[16] 1 pipelined_rdd.count()
```

```
[17] 1 pipelined_rdd.take(81)
```

```
[18] 1 line_lengths = rdd_textfile.map(lambda line: len(line.split()))
```

```
[19] 1 line_lengths.take(10)
```

```
[6, 0, 2, 0, 4, 0, 45, 0, 85, 0]
```

```
[20] 1 line_lengths.reduce(lambda a,b: a if(a>b) else b)
```

Operaciones básicas con Dataframe

```
[23] 1 from pyspark.sql import SparkSession  
  
[24] 1 spark = SparkSession.builder.getOrCreate()  
2 df_textfile = spark.read.text('my_text_files/novela.txt')  
  
[25] 1 type(df_textfile)  
pyspark.sql.dataframe.DataFrame
```

Acciones

```
1 df_textfile.show()  
  
[27] 1 df_textfile.count()  
500  
  
[30] 1 df_textfile.first()  
Row(value="Harry Potter and the Sorcerer's Stone ")  
  
[31] 1 df_textfile.take(10)  
  
1 from google.colab import files  
2  
3 uploaded = files.upload()  
4  
5 for fn in uploaded.keys():  
6   print('User uploaded file "{name}" with length {length} bytes'.format(  
7       name=fn, length=len(uploaded[fn])))  
  
[Elegir archivos] chilean_bands_spark.txt  
• chilean_bands_albums_spark.txt(text/plain) - 500 bytes, last modified: 13-06-2022 - 100% done  
Saving chilean_bands_albums_spark.txt to chilean_bands_albums_spark.txt  
User uploaded file "chilean_bands_albums_spark.txt" with length 500 bytes  
  
1 !cat chilean_bands_albums_spark.txt  
  
1 !hdfs dfs -mkdir database  
  
1 !hdfs dfs -put chilean_bands_albums_spark.txt database  
mkdir: `database': File exists  
  
1 df_albums = spark.read.csv('database/chilean_bands_albums_spark.txt', header=True)  
1 df_albums.show()
```

Transformaciones

```
[1] 1 df_filtered = df_textfile.filter(df_textfile.value.contains('Harry'))

[39] 1 df_filtered.count()
81

[41] 1 df_filtered.take(81)

[41] 1 from pyspark.sql.functions import split, size, max as spark_max, col
2
3 df_numwords = df_textfile.select(size(split(col('value'), '\s+')).name('num_words'))
4 df_numwords.show()

1 df_max = df_numwords.agg(spark_max(col('num_words')))
2 df_max.show()

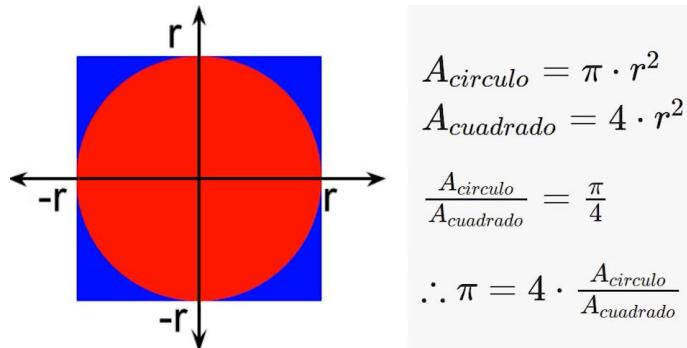
+-----+
|max(num_words)|
+-----+
|          168|
+-----+

1 df_max.collect()
[Row(max(num_words)=168)]

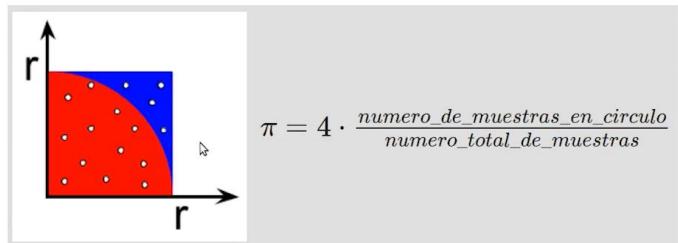
1 df_textfile.select(size(split(df_textfile.value, "\s+")).name('num_words')).agg(spark_max(col('num_words'))).collect()
[Row(max(num_words)=168)]
```

Calculo aproximado de π a través de MapReduce

Círculo circunscrito en cuadrado



Muestreo aleatorio sobre área



```

1 from random import random
2
3 # Mapper
4 def sample(p):
5     x = random()
6     y = random()
7     dist = x*x + y*y
8     in_circle = 1 if dist < 1 else 0
9     return in_circle
10
11 # Reduce
12 def sum_counts(a, b):
13     return a + b

1 from pyspark.context import SparkContext
2
3 NUM_SAMPLES = 100000
4
5 sc = SparkContext.getOrCreate()
6 count = sc.parallelize(range(0, NUM_SAMPLES)).map(sample).reduce(sum_counts)
7 print("PI is roughly %f" % (4.0 * count / NUM_SAMPLES))

```

x. Input de datos:

1. Métodos de SparkContext.

```

1 from pyspark import SparkContext
2 sc = SparkContext()

```

sc.textFile(file)

text_RDD = sc.textFile('documentos/*.txt')

sc.wholeTextFiles(dir)

docs_RDD = sc.wholeTextFiles('documentos')

sc.sequenceFile(file)

dict_RDD = sc.sequenceFile('data/*.seq')

sc.parallelize(list)

list_RDD = sc.parallelize([1,2,3,4,5,6])

2. Ejemplo, textFile:

```
text_RDD = sc.textFile('lorem_ipsum.txt')
```

lorem_ipsum.txt

 Lorem ipsum dolor
 sit amet, consectetur
 adipiscing elit, sed do
 eiusmod tempor



text_RDD

Lorem ipsum dolor
sit amet, consectetur
adipiscing elit, sed do
eiusmod tempor

3. Transformaciones:

- a. Es un método de un RDD de Spark.

RDD2 = RDD1.transformacion(función)

- b. Función es una función de Python (anónima o definida).
- c. Principales transformaciones locales:

<code>map(funcion)</code>	Mapea <code>funcion</code> de 1 elemento a 1 elemento.
<code>flatMap(funcion)</code>	Mapea <code>funcion</code> de 1 elemento a 0,1,...,N elementos.
<code>filter(funcion)</code>	Filtrá elementos según <code>funcion</code> , que debe retornar True o False.
<code>sortBy(funcion)</code>	Ordena elementos según <code>funcion</code> , que debe retornar key.
<code>join(RDD)</code>	Genera join de dos RDD de forma (key, value).
<code>coalesce(numParticiones)</code>	Balancea RDD localmente en numParticiones.

- d. Tipo de dato key-value: (key, value), donde key (número, texto, tupla) y value (cualquier tipo de valor, solo un elemento, puede ser lista).

- e. Principales transformaciones globales:

<code>reduceByKey(funcion)</code>	Reduce RDD según <code>funcion</code> independiente para cada key.
<code>groupByKey(funcion)</code>	Agrupa elementos según key definida por <code>funcion</code> .
<code>groupByKey()</code>	Agrupa elementos según key.
<code>repartition(numParticiones)</code>	Balancea RDD globalmente en numParticiones.

4. Acciones:

- a. Retorna valores desde RDD.
- b. Acción es un método de RDD de Spark.
- c. Valor = RDD.acción(argumento).

d. Valor es una variable en memoria RAM o escrita a almacenamiento. No RDD.

e. Principales acciones:

reduce(funcion)	Reduce el RDD según funcion a un valor.
countByKey()	Cuenta elementos tipo (key,value) retornando una lista (key, conteo).
max(), min(), mean(), stdev()	Estadísticos para RDD numéricos.
collect()	Retorna una lista completa a partir de los datos del RDD.
take(n)	Retorna una lista de primeros n elementos de los datos del RDD.
saveAsTextFile(path)	Almacena los datos del RDD en path.

xi. Persistencia:

1. Cache vs persist:

RDD.cache()	RDD.persist(storageLevel)
<ul style="list-style-type: none">Declaración de que cuando se calcule RDD, quedará en memoria (si es posible).Optimizaciones en datos requeridos muchas veces.Posibilita código iterativo.Análogo a RDD.persist(MEMORY_ONLY).	<ul style="list-style-type: none">Genérica.storageLevel:<ul style="list-style-type: none">MEMORY_ONLYDISK_ONLYMEMORY_AND_DISK

2. Ejemplo de cómo opera cache:

RDD.cache()	
<ul style="list-style-type: none">En general, RDD no se almacenan en memoria.Cada acción hace recalcular todas las transformaciones.Con cache(), se calcula solo una vez.	<pre>1 textData = sc.textFile('text.txt') 2 textData.count() 3 textData.count() 4 5 ----- 6 7 textData = sc.textFile('text.txt') 8 textData.cache() 9 textData.count() 10 textData.count()</pre>

xii. Acciones de Output de datos:

1. Siempre son acciones.

RDD.**saveAsTextFile**(file)

RDD.saveAsTextFile('rdd_text.txt')

RDD.**saveAsSequenceFile**(dir)

RDD.saveAsSequenceFile('rdd.seq')

RDD.**foreach**(function)

response = RDD.foreach(send_to_dashboard)

RDD.**collect**()

valores = RDD.collect()

- **Spark** provee de funciones para input de datos, transformaciones para procesar los RDD, y acciones para output de datos.
- Las **funciones** que son definidas para las transformaciones dan la lógica del código.
- Las declaraciones de persistencia como **cache()** permiten escalar a velocidades de procesamiento mucho mayores que otros frameworks.
- **Spark** se usa como un módulo o librería. A diferencia de otros frameworks, no hay que aprender un lenguaje particular y además usa toda su potencia.

xiii. Práctica de código en Spark:

Sentiment140

- 1.600.000 de tweets.
- Rotulados negativo/positivo.
- En inglés.

T-Drive Taxi Trajectories

- Más de 10.000 taxis y 17.000.000 de puntos GPS registrados.
- En Beijing.

Labeled Faces on the Wild

- 13.000 imágenes
- 1680 personas
- Fines académicos

Preparación del entorno

Instalación de Spark

```
[4] 1 from google.colab import files  
2  
3 uploaded = files.upload()  
4  
5 for fn in uploaded.keys():  
6   print('User uploaded file "{name}" with length {length} bytes'.format(  
7     name=fn, length=len(uploaded[fn])))  
  
Elegir archivos | spark_colab_installer.py  
• spark_colab_installer.py(text/x-python) - 14696 bytes, last modified: 13-06-2022 - 100% done  
Saving spark_colab_installer.py to spark_colab_installer (1).py  
User uploaded file "spark_colab_installer.py" with length 14696 bytes  
  
[5] 1 exec(open('spark_colab_installer.py').read())
```

Montar Google Drive, copiar datasets y descomprimirlos en Colaboratory

```
[3] 1 from google.colab import drive  
2 drive.mount('/content/drive', force_remount=True)  
3 !cp /content/drive/MyDrive/miDrive/datasets_spark.zip /content/  
4 !unzip -q datasets_spark.zip  
  
Mounted at /content/drive
```

Librerías de Python adicionales y preparación de datos

- numpy: manejo de matrices.
- nltk: Natural Language Toolkit. Manejo de texto.
- folium: manejo de mapas y coordenadas en mapa.
- opencv: manejo de imágenes.
- matplotlib: gráficos y visualizaciones en Python.

```
[6] 1 !pip -q install numpy nltk folium opencv-python opencv-contrib-python matplotlib  
2  
3 import findspark  
  
[6] 4 findspark.init()
```

Práctica de código en Spark

Sentiment140

Base de datos tweets rotulados, con categoría '0' para tweets negativos y '4' para positivos. 1.600.000 registros.

Elementos a destacar:

- Uso de funciones del módulo NLTK.
- Manejo de CSV.
- Wordcount en Spark.

```
▶ 1 from pyspark import SparkContext  
2 import csv  
3 import nltk  
4 from nltk.corpus import stopwords  
5  
6 sc = SparkContext.getOrCreate()
```

```

1 # Funciones de "tokenización"
2
3 # Uso una lista genérica de stopwords
4 nltk.download('stopwords')
5 swords = stopwords.words('english')
6
7 # Se excluyen todos los signos de puntuación
8 tokenizer = nltk.RegexpTokenizer(r"\w+")
9
10 # El formato de este dataset es csv, por lo que se
11 # usa el módulo csv para leer cada línea y transformarla
12 # en una lista.
13
14 def getSentimentData(line):
15     elements = list(csv.reader(line))
16     return int(elements[0][0]), elements[-1][0]
17
18 def vectorize_and_clear_words(data):
19     new_line = tokenizer.tokenize(data[1].lower())
20     return data[0], [w for w in new_line if w not in swords]

```

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.

```

1 # Sentiment140 dataset
2 # Se usan versiones locales de los dataset
3
4 sentiment140 = sc.textFile('file:///content/datasets_spark/2477_4140_bundle_archive/training.1600000.processed.noemoticon.csv')
5
6 print("Sample")
7 print(sentiment140.take(2))
8 print("Count")
9 print(sentiment140.count())

```

Sample
["0","1467810369","Mon Apr 06 22:19:45 PDT 2009","NO_QUERY","_TheSpecialOne_","@switchfoot <http://twitpic.com/2y1zl> - Awww, that's
Count
1600000

```

1 # A partir de las líneas de texto, se tokenizan los tweets
2
3 data = sentiment140.map(getSentimentData).map(vectorize_and_clear_words)
4 data.take(2)

```

```

1 # wordcount para obtener palabras comunes de tweets negativas
2
3 wcount_neg = data.filter(lambda x: x[0] == 0)
4 print(wcount_neg.take(2))

```

[('0', ['switchfoot', 'http', 'twitpic', 'com', '2y1zl', 'awww', 'bummer', 'shoulda', 'got', 'david', 'carr', 'third', 'day']), ('0',

```

1 wcount_neg_dos = wcount_neg.flatMap(lambda x: x[1])
2 print(wcount_neg_dos.take(20))

```

['switchfoot', 'http', 'twitpic', 'com', '2y1zl', 'awww', 'bummer', 'shoulda', 'got', 'david', 'carr', 'third', 'day', 'upset',

```

1 wcount_neg_tres = wcount_neg_dos.map(lambda x: (x,1))
2 print(wcount_neg_tres.take(3))

```

[('switchfoot', 1), ('http', 1), ('twitpic', 1)]

```

1 wcount_neg_cuatro = wcount_neg_tres.reduceByKey(lambda a,b: a+b)
2 print(wcount_neg_cuatro.take(4))

```

[('facebook', 2230), ('cry', 3920), ('result', 317), ('school', 12886)]

```

1 wcount_neg_cinco = wcount_neg_cuatro.filter(lambda x: x[1]>500)
2 print(wcount_neg_cinco.take(3))

```

[('facebook', 2230), ('cry', 3920), ('school', 12886)]

```

1 wcount_neg_sorted = wcount_neg_cinco.sortBy(lambda x: x[1], ascending=False)
2 wcount_neg_sorted.take(20)

```

```

1 # Completo - wordcount para obtener palabras comunes de tweets positivos
2
3 wcount_pos = data.filter(lambda x: x[0]==4).flatMap(lambda x: x[1]).map(lambda x: (x,1)).reduceByKey(lambda a,b: a+b).filter(lambda x: x[1]>500)
4 wcount_pos_sorted = wcount_pos.sortBy(lambda x: x[1], ascending=False)
5 wcount_pos_sorted.take(20)

```

```

1 # Obtengo las palabras con la mayor relacion negativa/positiva
2
3 words_joined = wcount_pos.join(wcount_neg_cinco)
4 most_neg_relative = words_joined.map(lambda x: (x[0], x[1][1]/x[1][0])).sortBy(lambda x: x[1], ascending=False)
5 most_neg_relative.take(30)

```

Taxis en Beijing

Dataset de coordenadas de viajes de cerca de 10000 taxis en Beijing, obtenidas por una semana.

*** Su uso académico es libre, no usar para fines comerciales. Elementos a destacar:

- Miles de archivos de texto.
- Datos numéricicos de latitud/longitud.
- Manejo de mapa en Python.

```

[30] 1 puntos = sc.textFile('file:///content/datasets_spark/T-drive Taxi Trajectories/release/taxi_log_2008_by_id/*.txt')

[31] 1 print("Sample")
2 print(puntos.take(1))
3 print("Count")
4 print(puntos.count())

Sample
['1,2008-02-02 15:36:08,116.51172,39.92123']
Count
17662984

```

```

1 import folium
2
3 def get_data(line):
4     d = line.split(',')
5     return int(d[0]), d[1], float(d[2]), float(d[3])
6

```

```

1 data = puntos.sample(False, 0.01).map(get_data)

1 mean_long = data.map(lambda x: x[2]).mean()
2 mean_lat = data.map(lambda x: x[3]).mean()

1 m = folium.Map(location=[mean_lat, mean_long])
2 m

```

```

1 delta = 1/3600/3 # 10 metros aproximadamente

1 most_visited = data.map(lambda x: ((int(x[3]/delta), int(x[2]/delta)), 1)).reduceByKey(lambda a,b: a+b).sortBy(lambda x: x[1], ascending=False)

1 value_most_visited = most_visited.map(lambda x: (x[0][0]*delta + delta/2, x[0][1]*delta + delta/2)).take(50)

1 for tip in value_most_visited:
2     folium.Marker(tip).add_to(m)
3 m

```

Manejo de imágenes

Dataset "Labeled Faces on the Wild", muy usado para reconocimiento de rostros.

Elementos a destacar:

- Uso de archivos binarios.
- Manejo de imágenes dentro de Spark.

```

[41] 1 images = sc.binaryFiles('file:///content/datasets_spark/lfw/*/*.jpg')

```

```

1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from os.path import basename
5 from os.path import dirname
6
7 # Esta función es solo para obtener el nombre de la persona de la foto como una categoría
8 def get_name(x):
9     return basename(dirname(x))
10
11 # Conversor de datos binarios leidos por Spark en imagen de OpenCV.
12 # Depende del módulo de manejo de imágenes a usar, en este ejemplo
13 # se usa OpenCV pero puede ser Pillow, scikit-image, etc.
14 def binary2image(bytes_image):
15     return cv2.imdecode(np.asarray(bytarray(bytes_image)), dtype=np.uint8), cv2.IMREAD_COLOR)
16
17 Icolor = images.map(lambda x: (get_name(x[0]), binary2image(x[1])))
18 Igray = Icolor.map(lambda I: (I[0], cv2.cvtColor(I[1], cv2.COLOR_BGR2GRAY)))

```

```

1 samples = Igray.take(5)
2 for img in samples:
3     plt.imshow(img[1], cmap='gray')
4     plt.title(img[0])
5     plt.show()

```

```

1 # Solo para demostración, obtenemos un descriptor HoG de la imagen,
2 # en la práctica no es bueno para el reconocimiento de rostros
3 def get_hog(img):
4     return img[0], cv2.HOGDescriptor().compute(img[1]).ravel()
5
6 hog_descr = Igray.map(get_hog)

```

```

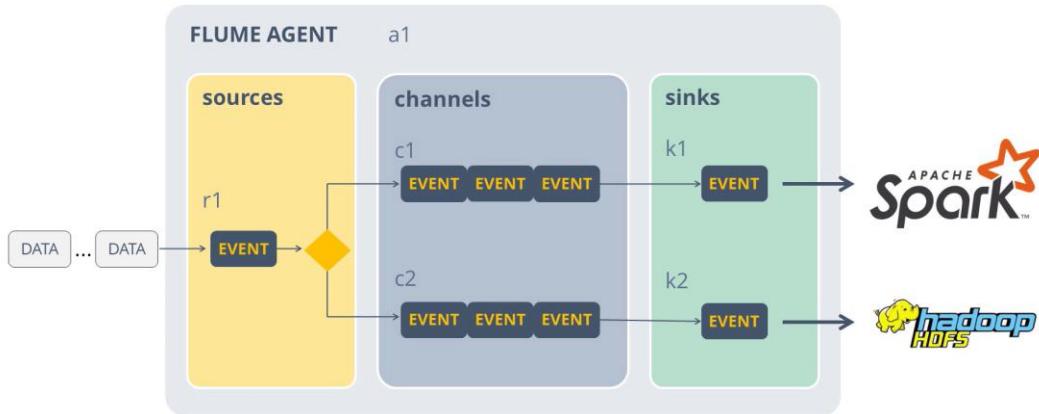
1 hogs = hog_descr.take(5)
2 for h in hogs:
3     plt.plot(range(len(h[1])),h[1])
4     plt.title(h[0])
5     plt.show()

```

t. Apache Flume:

¿Qué es?	¿Utilidad?	¿Cómo se usa?
<p>Software para transmisión de logs y eventos:</p> <ul style="list-style-type: none"> ▪ Escalable horizontalmente ▪ Robusto ▪ Simple 	<p>En logs y eventos:</p> <ul style="list-style-type: none"> ▪ Tamaño “pequeño” ▪ Transmitir a un cluster ▪ Agrupar y almacenar ▪ Enrutamiento 	<ul style="list-style-type: none"> ▪ <i>Daemon</i> ▪ flume-ng ▪ Archivo de configuración: <ul style="list-style-type: none"> ▪ Árbol ▪ Tipos predeterminados

Estructura de Agente



Event

Unidad básica: EVENT

- Cada dato recibido se transforma a EVENT
- Unidad de datos de Flume.
- Similar a *e-mail*:
 - **Header** => metadata
 - **Body** => datos

EVENT

HEADER

host: big.data.com
timestamp: 154343232.23

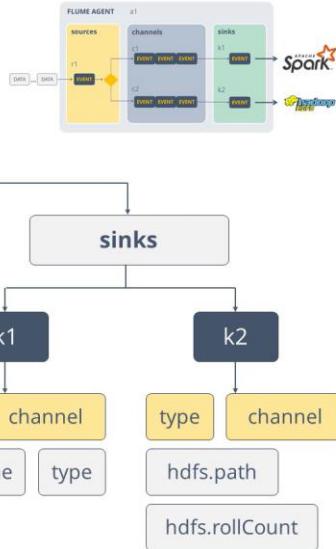
BODY

"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor"

Ejemplo de estructura en archivo de configuración



Estructura de Agente



i. Ver PDF.

u. Spark Streaming:

¿Qué es?	¿Cómo opera?	¿Por qué usarlo?
<p>Módulo de procesamiento de <i>streams</i> de datos de Spark:</p> <ul style="list-style-type: none"> ▪ <i>Near Real Time (NRT)</i> ▪ Robusto ▪ Simple 	<ul style="list-style-type: none"> ▪ Inputs: <ul style="list-style-type: none"> ▪ Socket ▪ Directorio ▪ Conectado a Flume ▪ Secuencia de RDD: <ul style="list-style-type: none"> ▪ DStream 	<ul style="list-style-type: none"> ▪ Aplicaciones <i>NTR</i> ▪ Código simple ▪ Extiende Spark añadiendo pocas líneas de código.

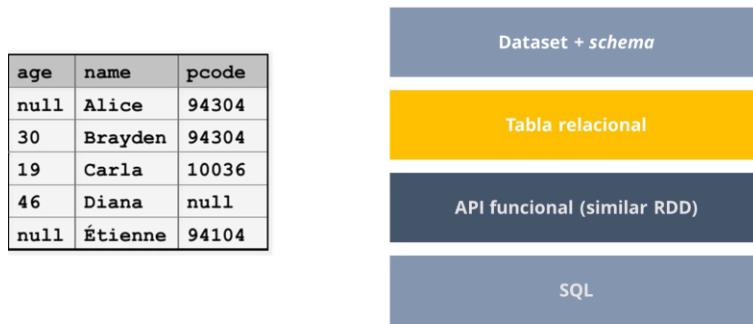
i. Ver PDF.

v. Practica Apache Flume: Validar.

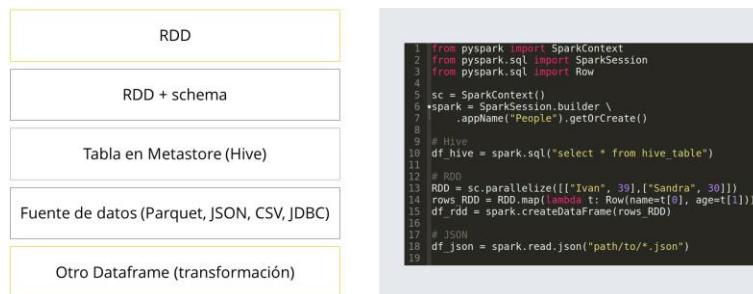
w. Spark SQL:

¿Qué es?	¿Cómo opera?	¿Por qué usarlo?
<p>Módulo de procesamiento de datos estructurados de Spark:</p> <ul style="list-style-type: none"> ▪ Versátil ▪ Eficiente ▪ Robusto ▪ Simple 	<ul style="list-style-type: none"> ▪ Dataset: <ul style="list-style-type: none"> ▪ Datos estructurados ▪ Dataframe: <ul style="list-style-type: none"> ▪ Dataset + schema ▪ API funcional y/o SQL. 	<ul style="list-style-type: none"> ▪ Simplifica ETL (o ELT) de datos estructurados. ▪ Eficiencia. ▪ Tendencia a operar Spark con Dataframes/Datasets.

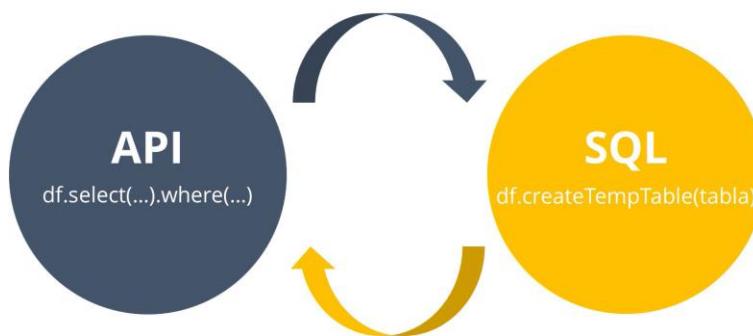
- i. Permite manejar datos estructurados de una manera eficiente.
- ii. Dataframe:



iii. Creación:



iv. Operatoria:



v. API: Métodos de Dataframe

cache()	describe(*cols)	join(other, on=None, how=None)
coalesce(numPartitions)	distinct()	limit(num)
collect()	drop(*cols)	orderBy(*cols, **kwargs)
count()	filter(condition)	select(*cols)
createTempView(name)	foreach(f)	show(n=20)
crossJoin(other)	groupBy(*cols)	toPandas()
cube(*cols)	intersect(other)	union(other)
where(condition)	unpersist(blocking=False)	fillna(value, subset=None)

```
df_in = spark.read.json("path/to/*.json")
df_in.createTempTable("tabla_ejemplo")
df_out = spark.sql("SELECT id, col1 FROM tabla_ejemplo WHERE col3 < 10 ORDER BY col1 LIMIT 30")
```

```
df_in = spark.read.json("path/to/*.json")
df_out = df_in.select(df_in['id'], df_in['col1']).where(df_in['col3'] < 10).orderBy(df_in['col1']).limit(30)
```

vi. Operatoria con Spark SQL:

Dataframes comparten características con RDD	Consideraciones de rendimiento
<ul style="list-style-type: none"> Immutable <i>Lazy execution</i> <i>Lineage</i> Esquema de transformaciones/acciones cache(), persist(...) 	<ul style="list-style-type: none"> Más rápido que RDD con mismos datos. Optimiza orden de ejecución. API o SQL mismo rendimiento (comparten mismo flujo). IF (datos estructurados) THEN (usar Spark SQL).

vii. Ejemplo básico:

```

1 from google.colab import drive
2 drive.mount('/content/drive', force_remount=True)

mounted at /content/drive

1 from google.colab import files
2
3 uploaded = files.upload()
4
5 for fn in uploaded.keys():
6   print('User uploaded file "{name}" with length {length} bytes'.format(
7     name=fn, length=len(uploaded[fn])))

Elegir archivos Ninguno archivo selec. Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
saving spark_colab_installer.py to spark_colab_installer.py
User uploaded file "spark_colab_installer.py" with length 14696 bytes

1 from google.colab import files
2
3 uploaded = files.upload()
4
5 for fn in uploaded.keys():
6   print('User uploaded file "{name}" with length {length} bytes'.format(
7     name=fn, length=len(uploaded[fn])))

Elegir archivos Ninguno archivo selec. Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
saving chilean_bands.csv to chilean_bands.csv
User uploaded file "chilean_bands.csv" with length 53 bytes

1 from google.colab import files
2
3 uploaded = files.upload()
4
5 for fn in uploaded.keys():
6   print('User uploaded file "{name}" with length {length} bytes'.format(
7     name=fn, length=len(uploaded[fn])))

Elegir archivos Ninguno archivo selec. Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
saving chilean_bands_albums.csv to chilean_bands_albums.csv

1 exec(open('spark_colab_installer.py').read())

Active services:
2194 NameNode
2261 DataNode
2616 JobHistoryServer
2108 ResourceManager
2671 Jps
2559 NodeManager

Apache Spark installed

1 import findspark
2 findspark.init()
3
4 from pyspark.sql import SparkSession
5
6 spark = SparkSession.builder.appName("ChileanBands").getOrCreate()

1 df_artist = spark.read.csv('file:///content/chilean_bands.csv', header=True)
2 df_albums = spark.read.csv('file:///content/chilean_bands_albums.csv', header=True)

1 df_artist.show()

+-----+-----+
|artist_id|      name|
+-----+-----+
|      1|    La Ley|
|      2|    Los Tres|
|      3|Los Prisioneros|
+-----+-----+

```

```

1 df_albums.show()

+-----+-----+-----+
|album_id|           name|year|artist_id|
+-----+-----+-----+
|    1|Desiertos|1990|      1|
|    2|Doble opuesto|1991|      1|
|    3|La Ley|1993|      1|
|    4|Invisible|1995|      1|
|    5|Vertigo|1998|      1|
|    6|Uno|2000|      1|
|    7|Libertad|2003|      1|
|    8|Adaptacion|2016|      1|
|    9|Los Tres|1991|      2|
|   10|Se remata el siglo|1993|      2|
|   11|La espada & la pared|1995|      2|
|   12|Fome|1997|      2|
|   13|La sangre en el c...|1999|      2|
|   14|Hágalo usted mismo|2006|      2|
|   15|Coliumo|2010|      2|
|   16|La voz de los 80|1984|      3|
|   17|Pateando piedras|1986|      3|
|   18|La cultura de la ...|1987|      3|
|   19|Corazones|1990|      3|
|   20|Los Prisioneros|2003|      3|
+-----+-----+-----+
only showing top 20 rows

```

```

1 new_albums = df_albums.orderBy(df_albums['year'].desc()).limit(5).join(df_artist, on='artist_id')
2 new_albums.show()

+-----+-----+-----+-----+
|artist_id|album_id|           name|year|           name|
+-----+-----+-----+-----+
|    1|     8|Adaptacion|2016|La Ley|
|    2|    15|Coliumo|2010|Los Tres|
|    2|    14|Hágalo usted mismo|2006|Los Tres|
|    3|    21|Manzana|2004|Los Prisioneros|
|    1|     7|Libertad|2003|La Ley|
+-----+-----+-----+-----+

```

```

1 df_albums.createOrReplaceTempView('albums')
2 df_artist.createOrReplaceTempView('artist')
3 new_albums_sql = spark.sql('select A.* , B.name from albums A join artist B on A.artist_id = B.artist_id order by A.year desc limit 5')
4 new_albums_sql.show()

+-----+-----+-----+-----+
|album_id|           name|year|artist_id|           name|
+-----+-----+-----+-----+
|    8|Adaptacion|2016|      1|La Ley|
|   15|Coliumo|2010|      2|Los Tres|
|   14|Hágalo usted mismo|2006|      2|Los Tres|
|   21|Manzana|2004|      3|Los Prisioneros|
|    7|Libertad|2003|      1|La Ley|
+-----+-----+-----+-----+

```

```

1 from pyspark.sql.functions import lower
2 albums_with_la = df_albums.select(df_albums['artist_id'], df_albums['name']).where(lower(df_albums['name']).contains('la'))
3 albums_with_la.show()

+-----+-----+
|artist_id|           name|
+-----+-----+
|      1|La Ley|
|      2|La espada & la pared|
|      2|La sangre en el c...|
|      3|La voz de los 80|
|      3|La cultura de la ...|
+-----+-----+

```

viii. Practica Spark SQL:

Preparamos el entorno

```
[1] 1 from google.colab import drive  
2 drive.mount('/content/drive')  
  
Mounted at /content/drive  
  
[2] 1 from google.colab import files  
2  
3 uploaded = files.upload()  
4  
5 for fn in uploaded.keys():  
6   print('User uploaded file "{name}" with length {length} bytes'.format(  
7     name=fn, length=len(uploaded[fn])))  
  
Elegir archivos spark_colab_installer.py  
• spark_colab_installer.py(n/a) - 14696 bytes, last modified: 18/6/2022 - 100% done  
Saving spark_colab_installer.py to spark_colab_installer.py  
User uploaded file "spark_colab_installer.py" with length 14696 bytes  
  
[3] 1 !cp /content/drive/MyDrive/datasets_spark.zip /content/  
2 !unzip -q datasets_spark.zip  
3  
  
[4] 1 exec(open('spark_colab_installer.py').read())  
  
Active services:  
2418 NodeManager  
2515 Jps  
1972 ResourceManager  
2056 NameNode
```

Inicializamos Spark y creamos un SparkSession

```
[13] 1 !pip -q install numpy nltk  
2  
3 import findspark  
4 findspark.init()  
5  
6 import csv  
7 import json  
8 import os  
9 import numpy  
10  
11 import nltk  
12 from nltk.corpus import stopwords  
13 nltk.download('stopwords')  
14 swords = stopwords.words('english')  
15 tokenizer = nltk.RegexpTokenizer(r"\w+")  
16  
17 def vectorize_and_clean_words(data):  
18   new_line = tokenizer.tokenize(data.lower())  
19   return [w for w in new_line if w not in swords]  
20  
21 from pyspark.sql import SparkSession  
22 spark = SparkSession.builder.appName("People").getOrCreate()  
  
[nltk_data] Downloading package stopwords to /root/nltk_data...  
[nltk_data] Package stopwords is already up-to-date!
```

Carga de CSV

```
[6] 1 df_tweets = spark.read.csv('file:///content/datasets_spark/2477_4140_bundle_archive/training.1600000.processed.noemoticon.csv')
2 print(df_tweets.count())
3 df_tweets.show(5)

[7] 1 # rdd
2 tweets = spark.sparkContext.textFile('file:///content/datasets_spark/2477_4140_bundle_archive/training.1600000.processed.noemoticon.csv')
3 print(tweets.count())
4 tweets.take(5)

[8] 1 # Contamos solo palabras positivas (c0 == 0) con SQL
2 df_tweets.select(df_tweets._c0).where(df_tweets._c0==0).count()

[9] 1 # Lo mismo con rdd
2 tweets.map(lambda x: [1[i] for i in list(csv.reader(x)) if len(i[0])]).filter(lambda x: x[0] == '0').count()

[10] 1 df_tweets.printSchema()

root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: string (nullable = true)

[11] 1 df_tweets.createOrReplaceTempView('sentiment')
2 df_positivos = spark.sql("select _c0, _c5 from sentiment where _c0='4' and _c5 like '%welcome%'")
3 df_positivos.show(5)

+---+-----+
| _c0|      _c5|
+---+-----+
| 4|@isdomous you're...|
| 4|@zinedistro Your ...|
| 4|@jeanchia you're ...|
| 4|@dj_bubble ur wel...|
| 4|@point_moot You'r...|
| 4|@surana you're w...|
| 4|@anita_0517 Ani!...|
| 4|@Hetty4Christ yea...|
| 4|@sarahatwood you'...|
| 4|@Amy440 well well...|
```

```

1 df_positivos_api = df_tweets.select([df_tweets['_c0'], df_tweets['_c5']]).filter((df_tweets['_c0'] == '4') & (df_tweets['_c5'].contains('welcome')))
2 df_positivos_api.show()

+---+-----+
| _c0 |      _c5 |
+---+-----+
| 4|@wisdomous you're...|
| 4|@zinedistro Your ...|
| 4|@jeanchila you're ...|
| 4|@dj_bubble ur wel...|
| 4|@point_moot You'r...|
| 4|@asurana you're w...|
| 4|@anita_0517 Ani...|
| 4|@Hetty4Christ yea...|
| 4|@sarahatwood you'...|
| 4|@Amy440 well well...|
| 4|@mrgarylee Aww t...|
| 4|@markress Mucvhly...|
| 4|@m_csquare you're...|
| 4|@saulashby you're...|
| 4|@ladydoc_1988 Hey...|
| 4|@boomerjack coffe...|
| 4|@appy_ro welcome...|
| 4|@RosalieHale you...|
| 4|@SexySubKaylee Oh...|
| 4|@plannersusanna T...|
+---+-----+
only showing top 20 rows

```

```

1 # Pasar df a rdd
2 from pyspark.sql import Row
3
4 df_features = spark.createDataFrame(df_tweets.rdd.map(lambda row: Row(label=float(row['_c0']), clean_words=vectorize_and_clean_words(row['_c5']))))
5 df_features.show(100)

+-----+-----+
|   clean_words|label |
+-----+-----+
|[switchfoot, http...|  0.0|
|[unset, unset, f...|  0.0|

```

Carga de JSON

Creación de archivos simulados

```

[17] 1 from random import random
2 if not os.path.exists('json_files'):
3     os.makedirs('json_files')
4
5 for nf in range(100):
6     fid = open(os.path.join('json_files', 'json_lines_%03d.txt'%nf), 'w')
7     for l in range(200):
8         fid.write('%s\n'%json.dumps({'a': random(), 'b': 3*random(), 'c': 4*random()}))
9         if random() > 0.8:
10             fid.write('%s\n'%json.dumps({'a': random(), 'b': 3*random(), 'c': -4*random(), 'd': 10*random()}))
11 fid.close()

```

```

[18] 1 df_jsons = spark.read.json('file:///content/json_files/*')
2 df_jsons.show()
3 df_jsons.count()

```

a	b	c	d
0.45986020467150157	0.2736976034520724	2.4835146652113855	null
0.6632020477950775	0.152156658954956	-1.0732701233496598	8.718071969870772

```
[19] 1 df_jsons.printSchema()
```

```
root
|-- a: double (nullable = true)
|-- b: double (nullable = true)
|-- c: double (nullable = true)
|-- d: double (nullable = true)
```

Creación de Dataframe desde memoria

```
[20] 1 from pyspark.sql import Row
2
3 row_list = []
4 for nf in range(100):
5     for l in range(200):
6         row_list.append(Row(a=random(), b=3*random(), c=4*random()))
7
8 df_numbers = spark.createDataFrame(spark.sparkContext.parallelize(row_list))
9 df_numbers.show()
10 df_numbers.count()

+-----+-----+-----+
|      a|      b|      c|
+-----+-----+-----+
| 0.7584334158602335| 0.29830243367093656| 1.1945027847120038|
| 0.17945316417843904| 0.752739159680045| 2.085543093111765|
| 0.82378812706877| 0.04707636275459215| 1.864826263899051|
```

Taxis (multiples CSV)

```
[21] 1 df_taxis = spark.read.csv('file:///content/datasets_spark/T-drive Taxi Trajectories/release/taxi_log_2008_by_id/*.txt')
2 df_taxis.count()
```

```
17662984
```

```
[22] 1 df_taxis.printSchema()
```

```
root
|-- _c0: string (nullable = true)
|-- _c1: string (nullable = true)
|-- _c2: string (nullable = true)
|-- _c3: string (nullable = true)
```

```
[26] 1 # Cambiamos nombre y tipo de columna
2 from pyspark.sql.types import DoubleType, IntegerType
3
4 df_taxis_ok = df_taxis.withColumnRenamed('_c0', 'id')\
5             .withColumnRenamed('_c1', 'timestamp')\
6             .withColumnRenamed('_c2', 'lon')\
7             .withColumnRenamed('_c3', 'lat')
8 df_taxis_ok = df_taxis_ok.withColumn('id', df_taxis_ok['id'].cast(IntegerType()))\
9             .withColumn('lon', df_taxis_ok['lon'].cast(DoubleType()))\
10            .withColumn('lat', df_taxis_ok['lat'].cast(DoubleType()))
11
12 df_taxis_ok.show()

+---+-----+-----+-----+
| id| timestamp|   lon|    lat|
+---+-----+-----+-----+
|6275|2008-02-02 13:30:44|116.36838|39.90484|
```

```

1 df_taxis_ok.printSchema()

root
 |-- id: integer (nullable = true)
 |-- timestamp: string (nullable = true)
 |-- lon: double (nullable = true)
 |-- lat: double (nullable = true)

1 df_taxis_ok.createOrReplaceTempView('taxis')

1 df_result = spark.sql("select id, lat, lon from taxis where id = 3000 and timestamp like '2008-02-06%' order by timestamp desc")
2 df_result.show()
3 df_result.count()

+---+-----+-----+
| id|    lat|    lon|
+---+-----+-----+
|3000|39.94757|116.54482|
```

```

1 # JSON
2 ret = df_result.write.json('taxi_3001.json')

----- ◆ 3 frames -----
/content/spark/python/lib/py4j-0.10.7-src.zip/py4j/protocol.py in get_return_value(answer, gateway_client, target_id, name)
326         raise Py4JJavaError(
327             "An error occurred while calling {0}{1}{2}.\\n".
--> 328             format(target_id, ".", name), value)
...--
```

```

1 # HIVE
2 ret = df_result.write.saveAsTable('table_300')

1 !hdfs dfs -ls

Found 2 items
drwxr-xr-x  - root supergroup      0 2022-06-19 02:00 taxi_300.json
drwxr-xr-x  - root supergroup      0 2022-06-19 02:04 taxi_3001.json
```

```

1 df_new_result = spark.sql("SHOW TABLES").show()

+-----+-----+
|database|tableName|isTemporary|
+-----+-----+
| default|table_300|     false|
|        |sentiment|     true|
|        |taxis|     true|
+-----+-----+
```

```

1 df_new_result = spark.sql("select * from table_300")
2 df_new_result.show()

+---+-----+-----+
| id|    lat|    lon|
```

x. Spark MILLIB: Algoritmos supervisados

- i. Debe haber un dato supervisado.
- ii. Spark MILLIB:
 1. Se converge a usar Dataframe.
 2. Se usan datos distribuidos.

3. Posee múltiples algoritmos.
4. Se recomienda usar **pyspark.ml** de alto nivel.
5. Continua con el modelo fit/transform.

pyspark.ml

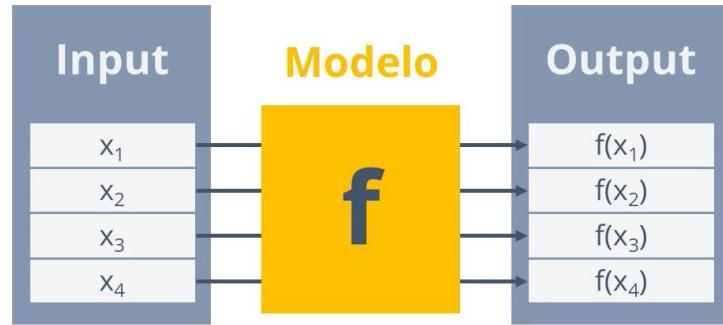
"Alto" nivel

- Transformador / Pipelines
- Inputs:
 - Dataframe (Spark SQL)
 - Foco de Spark en ML
 - Paradigma "Python" (scikit-learn)

6. Algoritmos ML (pyspark.ml) son, clasificación, regresión, agrupamiento, reducción dimensionalidades, otros (collaborative, filtering, frequent, pattern mining).
7. Modo de uso:
 - a. Entrenamiento: Datos (DF) -> Algoritmo (estimador) -> Modelo (transformer).
 - b. Predicción: Datos (DF) -> Modelo (transformer) -> Predicción (DF).

iii. Algoritmos supervisados:

1. Necesita un target a predecir.
2. Algoritmo de clasificación (predice una categoría o clase, ej. Estafa o no estafa, reconocimiento facial).
3. Algoritmo de regresión (Predice un número real, ej. Valor de una acción, temperatura, probabilidad de lluvia).
4. Para entrenamiento hay que ajustar un modelo, se necesita un Input y un Output.



5. Algoritmos supervisados en Spark:

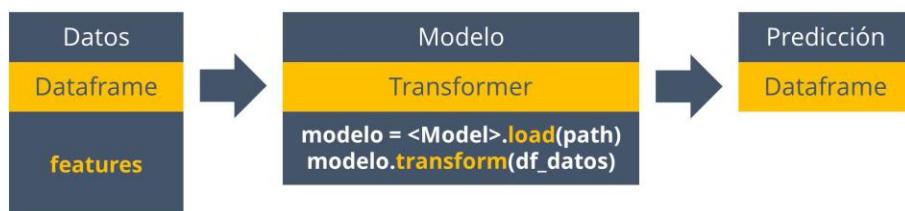
- a. Clasificación: Regresión logística, Árbol de decisión / Random Forest / Gradient Boosted Trees (GBT), Multilayer Perceptron (MLP), SVM lineal, Naive Bayes, Multiclasa: OneVsRest.
- b. Regresión: Regresión lineal, Regresión Árbol de decisión/Random Forest/Gradient Booted Trees (GBT), Sirvival regression, Isotonic regression.

6. Variable de entrada siempre es un Dataframe.

7. Entrenamiento:



8. Predicción:



9. El Dataframe de la predicción va a contener labels.

10. Ejemplo SVM lineal:

```

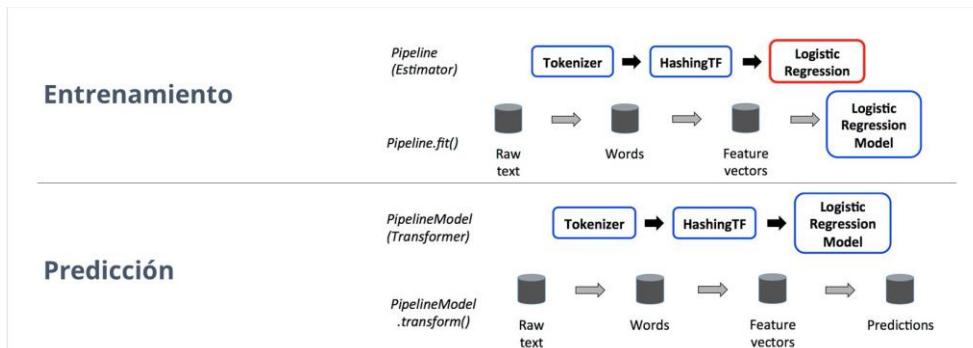
1  from pyspark.sql import SparkSession
2  from pyspark.ml.classification import LinearSVC, LinearSVCModel
3
4  spark = SparkSession.builder \
5      .appName("SVM").getOrCreate()
6
7  # Carga de datos como Dataframe
8  data = spark.read.format("libsvm")\
9      .load("data/mllib/sample_libsvm_data.txt")
10
11 # Datos de training y testing
12 training, testing = data.randomSplit([0.8, 0.2])
13
14 lsvc = LinearSVC(maxIter=10, regParam=0.1)
15
16 # Entrenamiento
17 lsvcModel = lsvc.fit(training)
18
19 # Se guarda modelo
20 lsvcModel.save('model_SVM.pkl')
21
22 # Prediccion
23 lsvcSavedModel = LinearSVCModel.load('model_SVM.pkl')
24 predictions = lsvcSavedModel.transform(testing)

```

11. Estimadores y modelos (clasificación):

SVM	LinearSVC LinearSVCModel
Random Forest	RandomForestClassifier RandomForestClassificationModel
MLP	MultilayerPerceptronClassifier MultilayerPerceptronClassificationModel
Regresión logística	LogisticRegression LogisticRegressionModel
Naive Bayes	NaiveBayes NaiveBayesModel

12. Pipelines:



13. Práctica Spark Mllib:

```
1 from google.colab import drive
2 drive.mount('/content/drive')

Mounted at /content/drive

1 from google.colab import files
2 uploaded = files.upload()

Elegir archivos spark_colab_installer.py
• spark_colab_installer.py (text/x-python) - 14696 bytes, last modified: 18-06-2022 - 100% done
Saving spark_colab_installer.py to spark_colab_installer.py

1 exec(open('spark_colab_installer.py').read())
2 !cp /content/drive/MyDrive/miDrive/datasets_spark.zip /content/
3 !unzip -q datasets_spark.zip

Active services:
2512 JobHistoryServer
2546 Jps
2453 NodeManager
2006 ResourceManager
2093 NameNode

Apache Spark installed

1 !pip -q install nltk
2
3 import findspark
4 findspark.init()
5
6 import csv
7 import nltk
8 from nltk.corpus import stopwords
9 from nltk.corpus import words
10 from nltk.stem import PorterStemmer
11
12 nltk.download('stopwords')
13 nltk.download('words')
14 swords = stopwords.words('english')
15 realwords = words.words()
16 stemmer = PorterStemmer()
17
18 tokenizer = nltk.RegexpTokenizer(r"\w+")
19 realwords = {stemmer.stem(w): True for w in words.words()}
20
21 def vectorize_and_clean_words(data):
22     word_list = tokenizer.tokenize(data.lower())
23     stemmed_words = [(w,stemmer.stem(w)) for w in word_list]
24     return [w[1] for w in stemmed_words if w[1] in realwords]
25
26 from pyspark.ml.feature import Word2Vec
27 from pyspark.sql import Row
28 from pyspark.sql import SparkSession
29
30 spark = SparkSession.builder.appName("People").getOrCreate()
```

Sentiment140

Usar modelos de ML según la documentación:

```
class pyspark.ml.classification.LinearSVC (featuresCol='feature', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, tol=1e-06, rawPredictionCol='rawPrediction', fitIntercept=True, standardization=True, threshold=0.0, weightCol=None, aggregationDepth=2)

[12] 1 # Importamos librerías de clasificación
2 from pyspark.ml.classification import LinearSVC, RandomForestClassifier, GBTClassifier
3
4 # Se ocupa un sample de 1 de cada 100 datos
5 df_tweets = spark.read.csv('file:///content/datasets_spark/2477_4140_bundle_archive/training.1600000.processed.noemoticon.csv').sample(fraction=0.01)
6 df_tweets.show()
7 print("Count: {}".format(df_tweets.count()))
```

```

1 df_words = spark.createDataFrame(df_tweets.rdd.map(lambda row: Row(label=float(row['_c0'])=='4'), clean_words=vectorize_and_clean_words(row['_c5']), tweet=row['_c5'])))
2
3 # Agrego un indice a cada tweet
4 from pyspark.sql.functions import monotonically_increasing_id
5 df_words = df_words.withColumn('index', monotonically_increasing_id())
6
7 # El False muestra completo el dataframe, no lo resume
8 df_words.show(20, False)
9
10 # Training/testing split
11 training, testing = df_words.randomSplit([0.95, 0.05])
12
13 # Modelo word2vec
14 model_w2v = Word2Vec(vectorSize=32, minCount=0, inputCol="clean_words", outputCol="features").fit(training)
15
16 # Dataframe de [word, vector]
17 w2v_vectors = model_w2v.getVectors()
18 w2v_vectors_real = w2v_vectors.rdd.map(lambda row: [row['word'], row['vector']]).collect()
19
20 # Diccionario word:word2vec en variable "broadcast" - variables globales distribuida
21 dict_w2v = spark.sparkContext.broadcast([w[0]:w[1] for w in w2v_vectors_real])
22

```

```

1 # Set entrenamiento
2
3 train_flat_features = training.rdd.flatMap(lambda x: [(x['index'], x['label']), w) for w in x['clean_words']] ) \
4     .mapValues(lambda x: dict_w2v.value[x])
5
6 test_flat_features = testing.rdd.flatMap(lambda x: [(x['index'], x['label']), w) for w in x['clean_words']] ) \
7     .filter(lambda x: x[1] in dict_w2v.value) \
8     .mapValues(lambda x: dict_w2v.value[x])
9
10 print("Train flat features")
11 print(train_flat_features.take(1))
12
13 mean_train_features = train_flat_features \
14     .mapValues(lambda v: (v, 1)) \
15     .reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1])) \
16     .mapValues(lambda v: v[0]/v[1])
17
18 mean_test_features = test_flat_features \
19     .mapValues(lambda v: (v, 1)) \
20     .reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1])) \
21     .mapValues(lambda v: v[0]/v[1])
22
23 print("Mean train features")
24 print(mean_train_features.take(1))
25
26 print("Mean test features")
27 print(mean_test_features.take(1))
28
29 train_features = spark.createDataFrame(mean_train_features.map(lambda x: Row(index=x[0][0], label=x[0][1], features=x[1])))
30 test_features = spark.createDataFrame(mean_test_features.map(lambda x: Row(index=x[0][0], label=x[0][1], features=x[1])))
31
32 train_features.show()

```

```

1 # Entrenamiento
2 algorithm = LinearSVC(maxIter=10, regParam=0.1)
3 # algorithm = RandomForestClassifier()
4 # algorithm = GBTClassifier()
5
6 model = algorithm.fit(train_features)
7
8 # Predicción
9 test_features = model_w2v.transform(testing)
10 train_pred = model.transform(train_features)
11 test_pred = model.transform(test_features)
12
13 test_pred.show(20, False)
14
15 # Métrica de rendimiento
16 train_accuracy = train_pred.select(['label', 'prediction']).where(train_pred['label']==train_pred['prediction']).count()/train_pred.count()
17 print("TRAIN DATASET ACCURACY: " + str(train_accuracy))
18
19 test_accuracy = test_pred.select(['label', 'prediction']).where(test_pred['label']==test_pred['prediction']).count()/test_pred.count()
20 print("TEST DATASET ACCURACY: " + str(test_accuracy))

```

```

1 # Más positivas según predicción
2 from pyspark.sql.functions import udf
3 from pyspark.sql.types import FloatType
4 firstelement = udf(lambda v: float(v[0]),FloatType())
5
6 clean_results = test_pred.join(df_words, on='index').select(test_pred['label'], df_words['tweet'], test_pred['prediction'], test_pred['rawPrediction'])
7
8 ordered = clean_results.orderBy(firstelement(clean_results['rawPrediction']), ascending=True)
9
10 ordered.show(100, False)

```

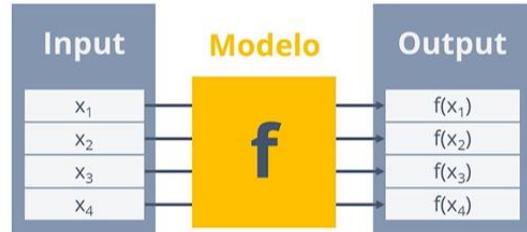
y. **Algoritmos no supervisados:**

- i. Algoritmos de Agrupamiento y Reducción dimensionalidad.
- ii. En general, cualquier algoritmo que no necesite labels o target se conocen como no supervisados.



Entrenamiento

Ajustar un Modelo
(Transformador)

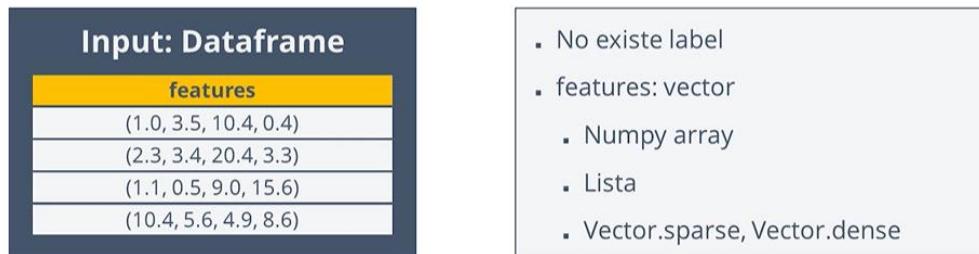


- iii. Algoritmos no supervisados en Spark:



iv. Entrenamiento y predicciones:

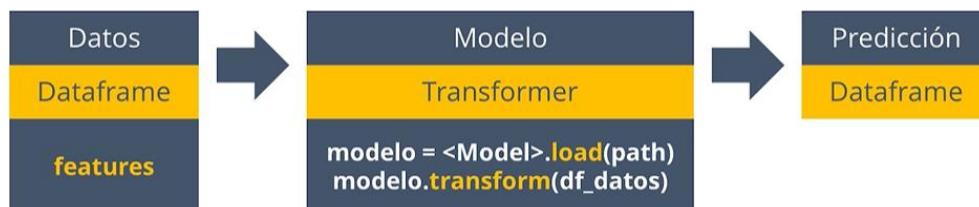
1. Variables de entrada:



2. Entrenamiento:



3. Predicción:



v. Ejemplo K-Means:

```

k_means.py
1  from pyspark.sql import SparkSession
2  from pyspark.ml.clustering import KMeans, KMeansModel
3
4  spark = SparkSession.builder \
5      .appName("KMeans").getOrCreate()
6
7  # Carga de datos
8  training = spark.read.format("libsvm").\
9      load("data/mllib/kmeans_train_data.txt")
10
11 # Entrenamiento
12 kmeans = KMeans().setK(2).setSeed(1)
13 model = kmeans.fit(training)
14 model.save('my_kmeans_model.pkl')
15
16 # Asignacion de grupos encontrados
17 # Carga de datos
18 testing = spark.read.format("libsvm").\
19     load("data/mllib/kmeans_test_data.txt")
20 model_loaded = KMeansModel.load('my_kmeans_model.pkl')
21 predictions = model_loaded.transform(testing)
22
23 # Centros de grupos
24 centers = model_loaded.clusterCenters()
25

```

vi. Otros algoritmos de Spark MLlib:

Frequent Pattern Mining	Frequent Pattern Mining con FP-Growth. pyspark.ml.fpm
Sistema recomendador off-the-shelf	Filtrado colaborativo con ALS. pyspark.ml.recommendation
Estadísticas	Test de hipótesis, estadísticas generales. pyspark.ml.stat

vii. Tips generales:

features	Múltiples transformaciones ya programadas (Tokenizer, OneHotEncoder, PCA, Word2Vec, Normalizer, etc.). pyspark.ml.features
checkpoints	Definir directorio (sc.setCheckpointDir(path)) y declarar checkpoints (RDD.checkpoint)
Image data source	Spark SQL permite cargar directamente imágenes a Dataframe, con tipo de dato "image". Son manipulables con OpenCV como Numpy.ndarray.
Documentación	La mejor fuente de conocimiento es la documentación en http://spark.apache.org

viii. Práctica Algoritmos no supervisados:

```

1 from google.colab import drive
2 drive.mount('/content/drive')

Mounted at /content/drive

1 from google.colab import files
2 uploaded = files.upload()

[Elegir archivos] Sin archivos...lecciónados Upload widget is only available when the cell has been executed in the current b
Saving spark_colab_installer.py to spark_colab_installer.py

1 exec(open('spark_colab_installer.py').read())

Active services:
2115 DataNode
2503 Jps
1960 ResourceManager
2409 NodeManager
2042 NameNode
2463 JobHistoryServer

Apache Spark installed

1 uploaded = files.upload()

[Elegir archivos] Sin archivos...lecciónados Upload widget is only available when the cell has been executed in the current br
Saving bank-additional-full.csv to bank-additional-full.csv

1 # Librerías necesarias
2 !pip -q install numpy matplotlib

1 # Librerías necesarias
2 import findspark
3 findspark.init()
4
5 import os
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9 from pyspark.sql import Row
10 from pyspark.sql import SparkSession
11
12 spark = SparkSession.builder.appName("Bank").getOrCreate()

```

Obtención de grupos de clientes

Una forma de preprocesar los datos es llevar ciertos valores categóricos a valores numéricos.

```

[29] 1 clients = spark.read.csv('file:/content/bank-additional-full.csv', header=True, sep=";")
2
3 clients.show(5, False)
4 print(clients.count())
5 print(clients.printSchema())

```

age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign
56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	261	1
57	services	married	high.school	unknown	no	no	telephone	may	mon	149	1
37	services	married	high.school	no	yes	no	telephone	may	mon	226	1
40	admin.	married	basic.6y	no	no	no	telephone	may	mon	151	1
56	services	married	high.school	no	no	yes	telephone	may	mon	307	1

only showing top 5 rows

```

1 # StringIndexer, toma un string y lo transforma en un índice
2 # Seleccionar columnas y llevarlos a atributos numéricos
3 from pyspark.ml.feature import StringIndexer
4
5 important_columns = ['job', 'marital', 'education', 'housing', 'loan']
6 indexed_important_columns = []
7
8 for col in important_columns:
9     indexer = StringIndexer(inputCol=col, outputCol=col + 'Index')
10    indexed_important_columns.append(col + 'Index')
11    clients = indexer.fit(clients).transform(clients)
12
13 clients.show()
14

+-----+-----+-----+-----+-----+-----+-----+
|age|      job| marital|      education|default|housing|loan| contact|month|day_of_week|duration|
+---+-----+-----+-----+-----+-----+-----+
| 56| housemaid| married| basic.4y| no| no| no|telephone| may| mon| 261|
| 57| services| married| high.school|unknown| no| no|telephone| may| mon| 149|
| 37| services| married| high.school| no| yes| no|telephone| may| mon| 226|
| 40| admin.| married| basic.6y| no| no| no|telephone| may| mon| 151|
| ...| ...| ...| ...| ...| ...| ...| ...| ...| ...| ...| ...

```

```

1 # OneHotEncoderEstimator: vector de n elementos de 0 y un 1 solo en su indice
2 from pyspark.ml.feature import OneHotEncoderEstimator
3
4 vectorized_important_columns = [a.replace('Index', 'vec') for a in indexed_important_columns]
5
6 encoder = OneHotEncoderEstimator(inputCols=indexed_important_columns, outputCols=vectorized_important_columns)
7
8 modelVec = encoder.fit(clients)
9 encoded = modelVec.transform(clients)
10 encoded.show()

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|age|      job| marital|      education|default|housing|loan| contact|month|day_of_week|duration|campaign|pdays|
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 56| housemaid| married| basic.4y| no| no| no|telephone| may| mon| 261| 1| 999|
| 57| services| married| high.school|unknown| no| no|telephone| may| mon| 149| 1| 999|
| 37| services| married| high.school| no| yes| no|telephone| may| mon| 226| 1| 999|
| 40| admin.| married| basic.6y| no| no| no|telephone| may| mon| 151| 1| 999|
| 56| services| married| high.school| no| no| yes|telephone| may| mon| 307| 1| 999|
| ...| ...| ...| ...| ...| ...| ...| ...| ...| ...| ...| ...

```

```

1 # Formamos un vector de features
2 # VecroAssembler: concatena una lista de columnas
3 from pyspark.ml.feature import VectorAssembler
4
5 assembler = VectorAssembler(inputCols=vectorized_important_columns, outputCol="features")
6
7 train_dataset = assembler.transform(encoded)
8 train_dataset.show()
9

+-----+-----+-----+-----+-----+-----+-----+
|age|      job| marital|      education|default|housing|loan| contact|month|day_of_week|duration|features|
+---+-----+-----+-----+-----+-----+-----+
| 56| housemaid| married| basic.4y| no| no| no|telephone| may| mon| :| |
| 57| services| married| high.school|unknown| no| no|telephone| may| mon| :|
| 37| services| married| high.school| no| yes| no|telephone| may| mon| :|
| 40| admin.| married| basic.6y| no| no| no|telephone| may| mon| :|
| 56| services| married| high.school| no| no| yes|telephone| may| mon| :|
| 51| services| married| basic.6y| unknown| no| no|no|telephone| may| mon| :|
| ...| ...| ...| ...| ...| ...| ...| ...| ...| ...| ...

```

```

1 # Entrenamos el clustering con KMeans
2 from pyspark.ml.clustering import KMeans
3 K = 20
4
5 algorithm = KMeans(k=K)
6 model = algorithm.fit(train_dataset)

1 # Transformamos los datos
2 clustered = model.transform(train_dataset)

1 # Mostramos ejemplo de los grupos
2 for n in range(K):
3     print("GROUP " + str(n))
4     filtered = clustered.filter(clustered['prediction'] == n)
5     print("CREDIT PERC: %02.1f"%(filtered.filter(filtered['y'] == 'yes').count()/filtered.count()*100))
6     filtered.select(important_columns).show (6)
7     print("-----")
8

```

GROUP 0
CREDIT PERC: 10.8

job marital education housing loan
hacis aul vaci mol

z. Reducción de dimensionalidad:

i. Ver PDF.

aa. Práctica reducción de dimensionalidad: Ver práctica.

II.