

```
it: map 0% reduce 0% Job_2
it: map 4% reduce 0%
it: map 9% reduce 0%
it: map 14% reduce 0%
it: map 19% reduce 0%
it: map 19% reduce 0%
it: map 23% reduce 6%
it: map 28% reduce 6%
it: map 28% reduce 6%
```

Big Data y Minería de Datos Geoespaciales

Práctica con Hadoop

Agosto 2024
Ángel Martín Furones

Este documento se ha escrito usando la versión 3.3.6 de Hadoop, aunque la versión actual a fecha de agosto de 2024 sea la 3.4.0 los comandos y resultados deben ser los mismos.

HDFS y MapReduce de Hadoop V.3

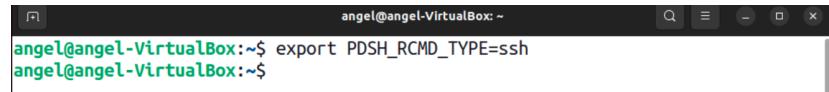
Después de instalar Hadoop de forma pseudo-distribuida, se ha ejecutado un primer trabajo MapReduce, el cálculo del número pi, para ello no hemos necesitado datos extra ni el uso de HDFS.

Vamos a ejecutar otro trabajo, el clásico contador de palabras de un texto (wordcount), para lo cual necesitaremos un fichero de datos y distribuirlo dentro del clúster mediante HDFS.

Para ello vamos a usar el libro de Frank Kane “Building an Internet Business”, que ya tratamos en la práctica de MapReduce y cuyo fichero es “Book.txt”:

Desde la terminal de Ubuntu, para evitar que pdsh y ssh entren en conflicto cuando lancemos los diferentes procesos (daemons) de Hadoop, ejecutamos el comando:

```
$ export PDSH_RCMD_TYPE=ssh
```

A screenshot of a terminal window titled "angel@angel-VirtualBox: ~". The window shows the command "\$ export PDSH_RCMD_TYPE=ssh" being typed by the user "angel". The terminal has a dark theme with white text and a black background.

Vamos al directorio *bin* de hadoop y formatearemos el namenode:

```
$ hdfs namenode –format
```

Vamos al directorio *sbin* de hadoop y arrancamos todos los procesos (daemons) ejecutando las órdenes:

```
./start-dfs.sh  
./start-yarn.sh
```

A partir de este momento se puede monitorizar el namenode desde el localhost 9870:

<http://localhost:9870>

y el ResourceManager desde el localhost 8088:

<http://localhost:8088>

Volvemos al directorio *bin* de hadoop y creamos el directorio input dentro de HDFS:

```
$ hdfs dfs –mkdir /input
```

Copiaremos el fichero Book.txt al directorio input de hdfs:

```
$ hdfs dfs -put /home/angel/Documents/BigData/mapreduce/WordFrequency/Book.txt /input
```

```
angela@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/bin$ hdfs dfs -put /home/angel/Documents/BigData/mapreduce/WordFrequency/Book.txt /input
```

Para ver todos los directorios que hay en HDFS se puede usar la orden:

```
$ hdfs dfs -ls /
```

y para ver lo que hay en un directorio en concreto:

```
$ hdfs dfs -ls /input
```

```
angela@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/bin$ hdfs dfs -ls /input
Found 1 items
drwxr-xr-x - angel supergroup          0 2023-08-08 16:18 /input
angela@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/bin$ hdfs dfs -ls /input
Found 1 items
-rw-r--r-- 1 angel supergroup 264875 2023-08-08 16:18 /input/Book.txt
angela@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/bin$
```

Número de réplicas configuradas
en el fichero hdfs-site.xml

Tamaño del fichero

Para borrar un directorio de forma recursiva (con todos los ficheros que hay en su interior), ejecutar la orden:

```
$ hdfs dfs -rmr /nombre del directorio dentro de HDFS
```

O:

```
$ hdfs dfs -rm -r /nombre del directorio dentro de HDFS
```

Si deseamos mover varios ficheros a HDFS, por ejemplo, renombramos el fichero Book.txt como Bookb.txt y hacemos una copia como Bookc.txt, los dos dentro del directorio local /wordFrequency, se puede hacer, en este caso, ejecutando la orden:

```
$ hdfs dfs -put /home/angel/Documents/BigData/mapreduce/WordFrequency/*.txt /input
```

Si queremos ver únicamente el tamaño de los ficheros, ejecutar

```
$ hdfs dfs -du /input
```

```
angela@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/bin$ hdfs dfs -du /input
264875 264875 /input/Book.txt
264875 264875 /input/Bookb.txt
264875 264875 /input/Bookc.txt
angela@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/bin$
```

Si deseamos ver el tamaño de un único archivo:

```
$ hdfs dfs -du /input/Book.txt
```

```
angel@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/bin$ hdfs dfs -du /input/Book.txt
264875 264875 /input/Book.txt
angel@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/bin$
```

Y si únicamente queremos ver el tamaño total de un directorio:

```
$ hdfs dfs -du -s /input
```

También es posible ejecutar opciones de Linux mediante tuberías, por ejemplo, ver únicamente las primeras 20 líneas del fichero:

```
$ hdfs dfs -text /input/Book.txt | head -n 20
```

```
angel@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/bin$ hdfs dfs -text /input/Book.txt | head -n 20
Self-Employment: Building an Internet Business of One
Achieving Financial and Personal Freedom through a Lifestyle Technology Business
By Frank Kane

Copyright © 2015 Frank Kane.
All rights reserved worldwide.

CONTENTS
Disclaimer
Preface
Part I: Making the Big Decision
Overcoming Inertia
Fear of Failure
Career Indoctrination
The Carrot on a Stick
Ego Protection
Your Employer as a Security Blanket
text: Unable to write to output stream.
angel@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/bin$
```

La orden:

```
$ hdfs dfs --help
```

Muestra todas las opciones disponibles del comando hdfs dfs:

```
angel@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/sbin$ hdfs dfs -help
Usage: hadoop fs [generic options]
      [-appendToFile [-n] <localsrc> ... <dst>]
      [-cat [-ignoreCrc] <src> ...]
      [-checksum [-v] <src> ...]
      [-chgrp [-R] GROUP PATH...]
      [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
      [-chown [-R] [OWNER][:GROUP]] PATH...
      [-concat <target path> <src path> <src path> ...]
      [-copyFromLocal [-f] [-p] [-l] [-d] [-t <thread count>] [-q <thread pool queue size>] <localsrc> ... <dst>]
      [-copyToLocal [-f] [-p] [-crc] [-ignoreCrc] [-t <thread count>] [-q <thread pool queue size>] <src> ... <localdst>]
      [-count [-q] [-h] [-v] [-t [<storage type>]] [-u] [-x] [-e] [-s] <path> ...]
      [-cp [-f] [-p] [-topax]] [-d] [-t <thread count>] [-q <thread pool queue size>] <src> ... <dst>]
      [-createSnapshot <snapshotDir> [<snapshotName>]]
      [-deleteSnapshot <snapshotDir> <snapshotName>]
      [-df [-h] <path> ...]
      [-du [-s] [-h] [-v] [-x] <path> ...]
      [-expunge [-immediate] [-fs <path>]]
      [-find <path> ... <expression> ...]
      [-get [-f] [-p] [-crc] [-ignoreCrc] [-t <thread count>] [-q <thread pool queue size>] <src> ... <localdst>]
      [-getfacl [-R] <path>]
      [-getattr [-R] {-n name | -d} [-e en] <path>]
      [-getmerge [-nl] [-skip-empty-file] <src> <localdst>]
      [-head <file>]
      [-help [cmd ...]]
      [-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [-e] [<path> ...]]
```

Si deseamos conocer un comando específico (por ejemplo *du*):

```
$ hdfs dfs -help du
```

```
angel@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/bin$ hdfs dfs -help du
-du [-s] [-h] [-v] [-x] <path> ... :
  Show the amount of space, in bytes, used by the files that match the specified
  file pattern. The following flags are optional:
  -s Rather than showing the size of each individual file that matches the
        pattern, shows the total (summary) size.
  -h Formats the sizes of files in a human-readable fashion rather than a number
        of bytes.
  -v option displays a header line.
  -x Excludes snapshots from being counted.

  Note that, even without the -s option, this only shows size summaries one level
  deep into a directory.

  The output is in the form
    size   disk space consumed      name(full path)
angel@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/bin$
```

Como ya se ha dicho, el estado del NameNode también se puede ver desde el localhost 9870, pero, además, si queremos ver el estado del clúster y obtener detalles sobre los nodos que están soportando el sistema de archivos HDFS desde la terminal se puede ejecutar la orden:

```
$ hdfs dfsadmin -report
```

```
angel@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/bin$ hdfs dfsadmin -report
Configured Capacity: 30995845120 (28.87 GB)
Present Capacity: 8938004480 (8.32 GB)
DFS Remaining: 8937156608 (8.32 GB)
DFS Used: 847872 (828 KB)
DFS Used%: 0.01%
Replicated Blocks:
  Under replicated blocks: 0
  Blocks with corrupt replicas: 0
  Missing blocks: 0
  Missing blocks (with replication factor 1): 0
  Low redundancy blocks with highest priority to recover: 0
  Pending deletion blocks: 0
Erasure Coded Block Groups:
  Low redundancy block groups: 0
  Block groups with corrupt internal blocks: 0
  Missing block groups: 0
  Low redundancy blocks with highest priority to recover: 0
  Pending deletion blocks: 0
-----
Live datanodes (1):
Name: 127.0.0.1:9866 (localhost)
Hostname: angel-VirtualBox
Decommission Status : Normal
Configured Capacity: 30995845120 (28.87 GB)
DFS Used: 847872 (828 KB)
Non DFS Used: 20457402368 (19.05 GB)
DFS Remaining: 8937156608 (8.32 GB)
DFS Used%: 0.00%
DFS Remaining%: 28.83%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
```

Donde podemos ver la cantidad de espacio libre que queda en cada nodo, por ejemplo.

Una vez tenemos los ficheros en HDFS procedemos a ejecutar el trabajo MapReduce wordcount, para ello Hadoop viene con una serie de rutinas de ejemplo ya listas para ser ejecutadas (como el cálculo del número pi ya ejecutado cuando se instaló hadoop), para ello vamos al directorio /share/hadoop/mapreduce dentro de hadoop y ejecutamos:

```
$ yarn jar hadoop-mapreduce-examples-3.3.6.jar wordcount /input /output
```

Si lo ejecutamos así se efectuará el proceso wordcount sobre todos los ficheros que se encuentren dentro del directorio /input de HDFS (tres en este caso), los resultados los

guardará en el directorio /output dentro de HDFS (se puede poner el nombre que se quiera pero no debe estar creado el directorio en HDFS, es un mecanismo de seguridad de Hadoop para asegurar que no se borran resultados de trabajos anteriores, si se pone el nombre de un fichero que ya existe en HDFS la ejecución del trabajo MapReduce fallará).

Ejecutado el programa podemos ver los ficheros que ha generado el proceso dentro del directorio /output, para ello volvemos al directorio bin y ejecutamos desde la terminal:

```
$ hdfs dfs -ls /output
```

```
angel@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/share/hadoop/mapreduce$ hdfs dfs -ls /output
Found 2 items
-rw-r--r-- 1 angel supergroup      0 2023-08-08 16:28 /output/_SUCCESS
-rw-r--r-- 1 angel supergroup 73809 2023-08-08 16:28 /output/part-r-00000
angel@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/share/hadoop/mapreduce$
```

Se ha generado un fichero vacío llamado _SUCCESS que únicamente informa de que el proceso se ha ejecutado con éxito y el fichero con la solución: part-r-00000.

Para pasar los ficheros de HDFS a un directorio local ejecutar (el directorio de salida es un ejemplo, pero debe ser un directorio que esté creado en local):

```
$ hdfs dfs -get /output/* /home/angel/Documents/BigData/hadoop
```

El proceso también se puede monitorizar desde el localhost 8088.

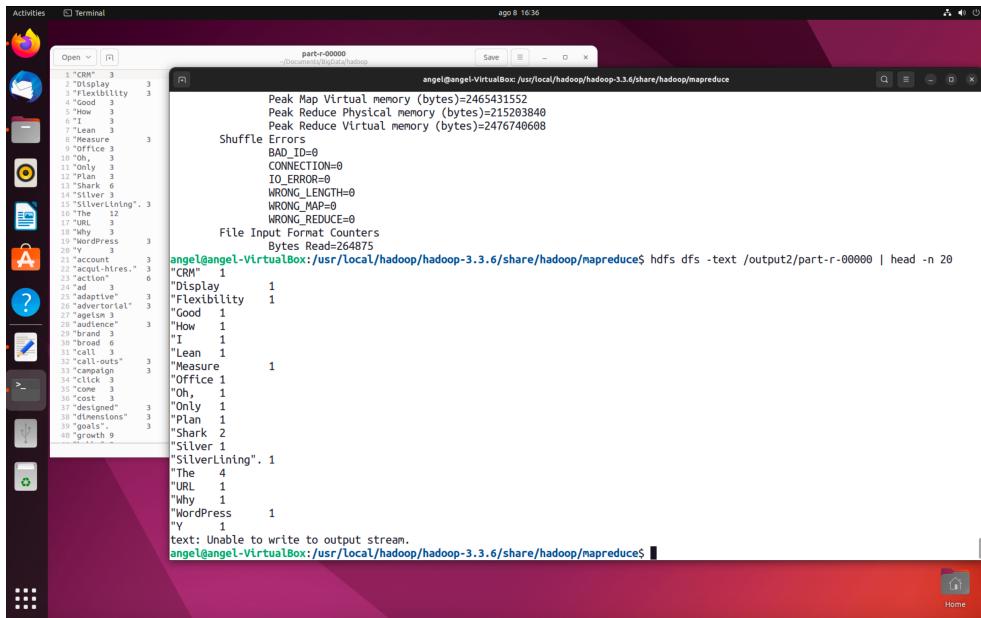
Para comprobar que se han usado los tres ficheros en el proceso MapReduce podemos borrar las dos copias en HDFS con la orden:

```
$ dhfs dfs -rm /input/Bookb.txt
$ hdfs dfs -rm /input/Bookc.txt
```

Si volvemos a ejecutar el proceso wordcount como hacíamos antes (poniendo como directorio de salida /output2 por ejemplo), podemos abrir el fichero resultante con la orden (desde bin de hadoop):

```
$ hdfs dfs -text /output2/part-r-00000 | head -n 20
```

y compararlo con el que habíamos generado anteriormente y guardado en local, si todo ha ido bien el que tenemos guardado en local deberá tener el triple de número de palabras que el fichero resultante del proceso que acabamos de ejecutar:



Los procesos MapReduce que lleva Hadoop instalados como ejemplo para su ejecución son:

```
angel@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/share/hadoop/mapreduce$ yarn jar hadoop-mapreduce-examples-3.3.6.jar
An example program must be given as the first argument.
Valid program names are:
  aggregatewordcount: An Aggregate based map/reduce program that counts the words in the input files.
  aggregatewordhist: An Aggregate based map/reduce program that computes the histogram of the words in the input files.
  bbp: A map/reduce program that uses Bailey-Borwein-Plouffe to compute exact digits of Pi.
  dbcount: An example job that count the pageview counts from a database.
  distbbp: A map/reduce program that uses a BBP-type formula to compute exact bits of Pi.
  grep: A map/reduce program that counts the matches of a regex in the input.
  join: A job that effects a join over sorted, equally partitioned datasets.
  multifilew: A job that counts words from several files.
  pentomino: A map/reduce tile laying program to find solutions to pentomino problems.
  pi: A map/reduce program that estimates Pi using a quasi-Monte Carlo method.
  randomtextwriter: A map/reduce program that writes 10GB of random textual data per node.
  randomwriter: A map/reduce program that writes 10GB of random data per node.
  secondarysort: An example defining a secondary sort to the reduce.
  sort: A map/reduce program that sorts the data written by the random writer.
  sudoku: A sudoku solver.
  teragen: Generate data for the terasort
  terasort: Run the terasort
  teravalidate: Checking results of terasort
  wordcount: A map/reduce program that counts the words in the input files.
  wordmean: A map/reduce program that counts the average length of the words in the input files.
  wordmedian: A map/reduce program that counts the median length of the words in the input files.
  wordstandarddeviation: A map/reduce program that counts the standard deviation of the length of the words in the input files.
angel@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/share/hadoop/mapreduce$
```

Una opción muy útil para guardar ficheros pequeños dentro de uno grande en HDFS (por ejemplo, miles de logs de ejecución e interacción de usuarios con una web en un único fichero dentro de Hadoop) es la opción *-appendToFile*; por ejemplo, podemos ejecutar la orden (desde el directorio bin de Hadoop):

```
$ hdfs dfs -appendToFile /home/angel/Documents/BigData/mapreduce/WordFrequency/Bookb.txt /input/Book.txt
```

y el contenido del fichero local se añadirá al contenido del fichero en HDFS (si el nombre del fichero que ponemos en HDFS no existe, Hadoop creará el fichero), para comprobar que ha funcionado podemos ver que el tamaño del fichero en HDFS ha aumentado su tamaño a, aproximadamente, el doble del original:

```
$ hdfs dfs -ls /input
```

```
angel@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/bin$ hdfs dfs -ls /input
Found 1 items
-rw-r--r-- 1 angel supergroup 529750 2023-08-08 16:39 /input/Book.txt
angel@angel-VirtualBox:/usr/local/hadoop/hadoop-3.3.6/bin$
```

Para terminar, se debe tener en cuenta que la salida de un proceso MapReduce puede generar un fichero grande, en ese caso Hadoop dividirá la salida en varios ficheros del mismo tamaño:

Nombre	Fecha de modificación	Tamaño	Clase
_SUCCESS	13 jun 2016 2:38	0 bytes	Documento
part-00000	13 jun 2016 2:38	3 KB	Documento
part-00001	13 jun 2016 2:38	3 KB	Documento
part-00002	13 jun 2016 2:38	3 KB	Documento
part-00003	13 jun 2016 2:38	3 KB	Documento
part-00004	13 jun 2016 2:38	3 KB	Documento
part-00005	13 jun 2016 2:38	3 KB	Documento
part-00006	13 jun 2016 2:38	3 KB	Documento

Para generar un único fichero de salida en local se puede ejecutar hdfs dfs con la opción `-getmerge`:

```
-getmerge [-nl] <src> <localdst> :
  Get all the files in the directories that match the source file pattern and
  merge and sort them to only one file on local fs. <src> is kept.

  -nl  Add a newline character at the end of each file.
```

En el supuesto de que en el directorio `output` se nos generaran varios ficheros `part-*` con los datos de salida, la orden para unirlos todos en un solo fichero sería:

```
$ hdfs dfs -getmerge -nl /output/part* /DIRECTORIO_LOCAL/salidaTodo.txt
```

Para apagar los procesos en ejecución, vamos al directorio `sbin` de hadoop y ejecutamos:

```
$ ./stop-yarn.sh
$ ./stop-dfs.sh
```

Si los volvemos a encender, la estructura de directorios creada en HDFS permanecerá, es decir, seguiremos teniendo los directorios `/input`, `/output` y `/output2`, ya que el namenode no ha perdido sus vínculos de dirección con el localhost, pero si apagamos la máquina lo perderá y deberemos formatear de nuevo el namenode, con lo que perderemos toda la estructura de directorios generada en trabajos anteriores.

Ejecución de trabajos MapReduce con hadoop-streaming

La herramienta de traducción de código `hadoop-streaming` se encuentra en el directorio de hadoop: `share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar`

Para esta práctica probaremos con el fichero `purchases.txt` y el `mapper.py` y `reducer.py` que ya vimos anteriormente en otra práctica, el proceso es muy simple:

Si empezamos desde cero, es decir, con la máquina virtual apagada, en primer lugar, formatearemos el `namenode` de hadoop e iniciaremos todos los procesos de java (daemons), a continuación, grabaremos el fichero `purchases.txt` en HDFS, por ejemplo en el directorio `/input` (que deberemos crear con `mkdir`).

A continuación, desde la terminal situada en el directorio donde tenemos el fichero `hadoop-streaming-3.3.6.jar` ejecutar la orden:

```
$ hadoop jar hadoop-streaming-3.3.6.jar
-mapper "python mapper.py" → Orden para ejecutar el mapper
-reducer "python reducer.py" → Orden para ejecutar el reducer
-input /input/purchases.txt → Fichero de entrada
-output /output3 → Directorio de salida que no debe existir
-file /path al mapper.py → Ruta completa al fichero mapper
-file /path al reducer.py → Ruta completa al fichero reducer
```

```
angel@angel-VirtualBox:~/Documents/BigData/mapreduce/purchases$ hadoop jar hadoop-streaming-3.3.6.jar -mapper "python3 mapper.py" -reducer "python3 reducer.py" -input /input/purchases.txt -output /output3 -file /home/angel/Documents/BigData/mapreduce/purchases/mapper.py -file /home/angel/Documents/BigData/mapreduce/purchases/reducer.py
```

La solución se encontrará en el fichero `output` dentro de HDFS:

```
angel@angel-VirtualBox:~/Documents/BigData/mapreduce/purchases$ hdfs dfs -ls /output3
Found 2 items
-rw-r--r-- 1 angel supergroup          0 2023-08-08 16:49 /output3/_SUCCESS
-rw-r--r-- 1 angel supergroup 3007 2023-08-08 16:49 /output3/part-00000
angel@angel-VirtualBox:~/Documents/BigData/mapreduce/purchases$ hdfs dfs -text /output3/part-00000 | head -n 20
Albuquerque 1005231.41999998
Anaheim 10076416.35999995
Anchorage 9933580.39999997
Arlington 10072207.969999978
Atlanta 9997146.69999945
Aurora 9992970.920000078
Austin 10057158.89999965
Bakersfield 10031208.919999931
Baltimore 10096521.449999949
Baton Rouge 10131273.229999999
Birmingham 10076606.519999951
Boise 10039166.740000017
Boston 10039473.28000001
Buffalo 10001941.190000022
Chandler 9919559.860000027
Charlotte 10112531.33999989
Chesapeake 10038504.920000048
Chicago 10062522.070000123
Chula Vista 9974951.34000001
Cincinnati 10139505.740000024
angel@angel-VirtualBox:~/Documents/BigData/mapreduce/purchases$
```

Hadoop HDFS y MapReduce con la clase MRJob y Hadoop-Streaming

Es posible ejecutar la clase `MRJob` dentro de Hadoop, pero al contener código python lo debemos hacer con Hadoop-Streaming. Para ello debemos tener activados todos los procesos java de Hadoop.

Por ejemplo, desde la terminal situada en el directorio donde tenemos el fichero `purchases.txt` y el programa `SolMRJob.py` ejecutar:

```
$ python SolMRJob.py
-r hadoop → Ejecución (run) sobre hadoop
--hadoop-streaming-jar="/PATH COMPLETO A hadoop-streaming-3.3.6.jar"
purchases.txt > salidaHadoop.txt
```

```
angel@angel-VirtualBox:~/Documents/BigData/mapreduce/purchases$ python3 SolMRJob.py -r hadoop --hadoop-streaming-jar="/usr/local/hadoop/hadoop-3.3.6/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar" purchases.txt > salidaHadoop.txt
```

La solución se genera temporalmente en HDFS pero, si no se indica lo contrario, se guarda el resultado final en local, en este caso fichero `salidaHadoop.txt` dentro del directorio local donde están los ficheros `purchases.txt` y `SolMRJob.py`.

Este proceso usa únicamente el motor MapReduce; también es posible usar el sistema de almacenamiento HDFS (un fichero que esté ya grabado en HDFS) cambiando la ultima línea de la orden anterior de la forma:

```
$ python SolMRJob.py
    -r hadoop
    --hadoop-streaming-jar="/PATH COMPLETO A hadoop-streaming-3.3.6.jar"
        hdfs://input/purchases.txt > salidaHadoop2.txt
```

O si queremos que el fichero de salida se sitúe sobre HDFS (con la estructura por defecto de salida que tiene Hadoop):

```
$ python SolMRJob.py
    -r hadoop
    --hadoop-streaming-jar="/PATH COMPLETO A hadoop-streaming-3.3.6.jar"
    --output-dir=hdfs://output4/
        hdfs://input/purchases.txt
```

La clase *SolMRJob* sobre Hadoop siempre necesitará que el fichero de entrada se le proporcione desde la terminal, tal como se ha hecho en este ejercicio; si introducimos la lectura desde dentro del código del programa *SolMRJob.py* con la orden *sys.stdin=open("fichero","r")* el proceso MapReduce fallará.

Acceso a HDFS desde Python

Existen varias librerías de Python que son capaces de trabajar con HDFS: pydoop, WebHDS o *snakebyte*, son algunas de las más utilizadas, todas ellas presentan algunas limitaciones y las diferentes versiones y actualizaciones de *Hadoop* pueden hacer que alguna de ellas, si no es actualizada convenientemente, llegue a no funcionar correctamente.

Para interactuar con HDFS, todos los procesos (*daemons*) de Hadoop deben estar activos, de manera que la interacción se basa en escribir comandos directamente en la ventana de la terminal. Para ello, la manera más sencilla es usar la librería de Python *Subprocess*.

A través de la librería de Python *subprocess* (que suele venir ya instalada) es posible la interacción con la terminal y el envío y ejecución de órdenes. Esta librería permite generar nuevos procesos, conectarse a sus canales de entrada/salida/error y obtener sus códigos de retorno.

Una estructura de programación simple donde se puedan ver en la terminal de Python los directorios de HDFS, donde se genere un directorio nuevo en HDFS y donde se grabe un fichero de local a HDFS y viceversa se puede ver a continuación (el fichero lo puedes descargar de PoliformaT en el apartado de Hadoop de la sección de prácticas: *ConnectHDFS.py*):

```
File Edit Format Run Options Window Help
1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4Created on Wed Sep 6 09:40:02 2023
5
6@author: angel
7"""
8import subprocess
9
10def run_cmd(args_list):
11    """
12        run linux commands
13    """
14    proc = subprocess.Popen(args_list, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
15    s_output, s_err = proc.communicate()
16    return s_output, s_err
17
18##Run Hadoop -ls / command in Python
19(out, err)= run_cmd(['hdfs', 'dfs', '-ls', '/'])
20print(out)
21
22##Run Hadoop -mkdir command in Python
23(out, err)= run_cmd(['hdfs', 'dfs', '-mkdir', '/input2'])
24
25##Run Hadoop -ls / command in Python
26(out, err)= run_cmd(['hdfs', 'dfs', '-ls', '/'])
27print(out)
28
29##Run Hadoop -put command in Python
30(out, err)= run_cmd(['hdfs', 'dfs', '-put',
31                     '/home/angel/Documents/BigData/mapreduce/WordFrequency/Book.txt', '/input2'])
32
33##Run Hadoop -get command in Python
34(out, err)= run_cmd(['hdfs', 'dfs', '-get',
35                     '/input2/Book.txt', '/home/angel/Documents/BigData/hadoop/'])
36
```

Ln: 13 Col: 11