



Universidad Europea de Andalucía

Máster Universitario en Análisis de Grandes Cantidades de Datos (Big Data)

Estudiante: Boris Paternina Pérez

Tarea 1 - Computación en Sistemas Distribuídos

Apache Hadoop, HDFS y MapReduce

Desarrollo y ejecución de un ejercicio básico de WordCount sobre un fichero de 1 GB:

Este ejercicio se desarrolló en una distribución Ubuntu 24.04 LTS bajo wsl2, y java development kit OpenJDK 64-Bit version 1.8.0_472. Las instrucciones para la instalación y configuración de Hadoop y YARN en modo de operación pseudo-distribuída, fueron tomadas de [Hadoop: Setting up a Single Node Cluster](#).

Adicional a estas instrucciones, se establecieron las siguientes variables en el archivo `.bashrc`:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_HOME=/home/hadoop/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
# variables adicionales para compilar WordCount
export PATH=${JAVA_HOME}/bin:${PATH}
export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar
```

Se estableció la siguiente configuración en el archivo `$HADOOP_HOME/etc/hadoop/core-site.xml`:

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

Y se estableció la siguiente configuración en el archivo `$HADOOP_HOME/etc/hadoop/hdfs-site.xml`:

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:///home/hadoop/hadoopdata/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///home/hadoop/hadoopdata/hdfs/datanode</value>
  </property>
</configuration>
```

Se comprobó la correcta instalación mediante los siguientes comandos:

```
ssh localhost
start-dfs.sh
start-yarn.sh
jps
```

Al haber comprobado que están activos los [daemon](#) de OpenSSH, Hadoop y YARN, y el clúster HDFS se inició correctamente, procedemos a ejecutar el ejercicio de WordCount.

1. Descarga del archivo [big-quijote.txt](#):

Ejecutamos el siguiente comando dentro del directorio local del proyecto [ueaMasterBD/input/](#):

```
wget https://objectstorage.eu-paris-1.oraclecloud.com/n/emeasespainsandbox/b/hadoop/o/big-quijote.txt
```

2. Creación del directorio y subida del archivo al clúster HDFS:

Antes de subir el archivo para su análisis en el framework MapReduce, debemos crear el directorio [input](#) dentro del clúster de HDFS:

```
hdfs dfs -mkdir -p /user/hadoop/input
```

Ejecutamos el siguiente comando para subir el archivo `big-quijote` desde el directorio local del proyecto, hacia el directorio [input](#) creado anteriormente:

```
hdfs dfs -put ueaMasterBD/input/* input
```

3. Creación de la aplicación WordCount:

Se creó el archivo `WordCount.java` dentro del directorio local `wordCount/src/`, con el [código suministrado](#) en el tutorial MapReduce.

Dentro del directorio `wordCount/src/`, se ejecuta el siguiente comando para crear el archivo `.class`:

```
hadoop com.sun.tools.javac.Main WordCount.java
```

Y el siguiente comando para crear el archivo ejecutable `wc.jar` a partir del archivo `.class`:

```
jar cf wc.jar WordCount*.class
```

Los procedimientos descritos en esta sección han creado los siguientes archivos:

```
'WordCount$IntSumReducer.class'  
'WordCount$TokenizerMapper.class'  
WordCount.class  
WordCount.java  
wc.jar
```

4. Ejecución de la aplicación WordCount:

- El siguiente comando ejecuta la aplicación `WordCount` sobre todo archivo almacenado en el directorio `input` del clúster HDFS.
- En nuestro caso, creando un archivo con el conteo de todas las palabras presentes en el archivo `big-quijote.txt`.
- La aplicación `WordCount` creará de forma automática el directorio `output` dentro del clúster, conteniendo el archivo con el resultado:

```
hadoop jar wc.jar WordCount /user/hadoop/input /user/hadoop/output
```

Para ver el contenido del directorio `output` ejecutamos el siguiente comando:

```
hadoop dfs -ls /user/hadoop/output
```

5. Análisis del resultado de WordCount. Resultado de las 10 palabras más repetidas en el fichero analizado.

Podemos ejecutar el siguiente comando si queremos ver el resultado generado por la aplicación [WordCount](#)

```
hdfs dfs -cat /user/hadoop/output/part-r-00000
```

No obstante, el tamaño del archivo (39.755 líneas), no hace práctico su análisis de esta forma.

Por tanto, podemos ejecutar el siguiente script en [bash](#) que nos permita obtener las 10 palabras más repetidas en el archivo [big-quijote.txt](#) sin necesidad de copiar el archivo al sistema local:

```
hdfs dfs -cat /user/hadoop/output/part-00000 | sort -n -r -k2,2 | head
```

El cual nos da el siguiente resultado:

que	10007552
de	9284096
y	8179712
la	5288448
a	4929024
el	4100608
en	4065792
no	2878464
se	2432512
los	2406912

6. ¿Cómo se ha almacenado la información en HDFS?

HDFS es la capa de almacenamiento del framework Apache Hadoop. Toda información (ficheros) en un clúster HDFS se dividen y almacenan en bloques fijos de 128MB por defecto en los DataNodes.

Normalmente un DataNode corresponde a un nodo o servidor en el clúster.

Debido a que los fallos de los servidores (nodos) en un sistema distribuído es una presunción por defecto en el framework Hadoop, cada bloque de 128MB es replicado por un factor de 3, almacenados en 3 diferentes nodos, durante la fase de creación del fichero. La replicación sucede en tiempo real, o asíncrona en escala mínima. HDFS almacena de forma típica 1 bloque en el nodo de un rack local, y los otros 2 bloques en 2 nodos de un rack remoto. De esta forma se conserva la tolerancia a fallos sin afectar la velocidad de las operaciones de lectura y escritura.

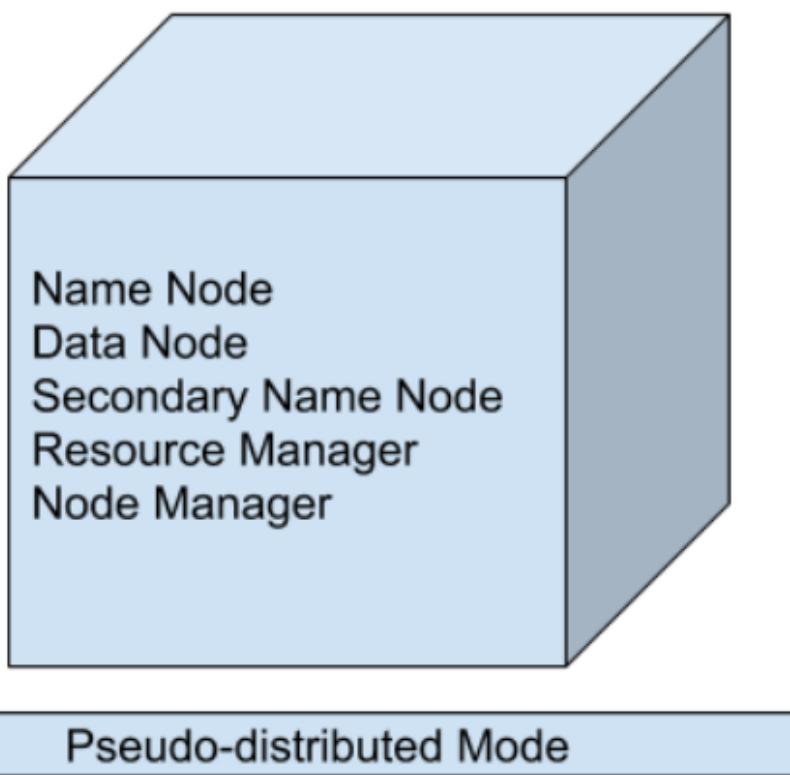
HDFS puede almacenar cualquier tipo de datos estructurados o no estructurados. No obstante, para optimizar el performance de los análisis de los datos almacenados, se emplean formatos basados en filas (e.g. Avro, csv), o columnares (e.g. Parquet).

7. ¿Qué tipo de sistema distribuido es?

Un clúster de HDFS es un sistema distribuído de archivos tipo Master/Slave, donde un Master Server (HDFS NameNode) administra y regula el acceso del sistema de archivos a los Slaves (HDFS DataNodes).

8. ¿Cuál es el factor de replicación del fichero?

El presente ejercicio de WordCount se ejecutó en un clúster pseudo distribuído. Debido a esto, los procesos (daemons) de Hadoop fueron ejecutados en máquinas virtuales java (JVM) independientes dentro de una misma computadora.



(imagen de <https://www.geeksforgeeks.org/data-engineering/hadoop-different-modes-of-operation/>)

Debido a que existe un solo DataNode en un sistema HDFS pseudo distribuído, el factor de replicación se establece típicamente en 1.

Lo anterior podemos corroborarlo examinando la configuración establecida en el archivo `$HADOOP_HOME/etc/hadoop/hdfs-site.xml`:

```
<property>
    <name>dfs.replication</name>
    <value>1</value>
</property>
```

9. ¿En cuántos bloques se ha dividido?

En el presente ejercicio, de la ejecución del comando

```
hadoop jar wc.jar WordCount /user/hadoop/input /user/hadoop/output
```

Podemos observar la siguiente información en el CLI:

```
hadoop jar wc.jar WordCount /user/hadoop/input /user/hadoop/output
2026-01-08 00:14:50,481 INFO client.DefaultNoHARMFailoverProxyProvider:
Connecting to ResourceManager at /0.0.0.0:8032
2026-01-08 00:14:50,780 WARN mapreduce.JobResourceUploader: Hadoop command-
line option parsing not performed. Implement the Tool interface and execute
your application with ToolRunner to remedy this.
2026-01-08 00:14:50,807 INFO mapreduce.JobResourceUploader: Disabling
Erasure Coding for path: /tmp/hadoop-
yarn/staging/hadoop/.staging/job_1767827162860_0001
2026-01-08 00:14:51,536 INFO input.FileInputFormat: Total input files to
process : 1
2026-01-08 00:14:52,417 INFO mapreduce.JobSubmitter: number of splits:9
```

Por tanto podemos decir que el fichero **big-quijote.txt** fue dividido en 9 bloques o **splits**.

10. Si ejecuto de nuevo el mismo comando para cargar el fichero en HDFS, ¿qué ocurre? Explica el resultado obtenido.

Al ejecutar el mismo comando **-put** empleado anteriormente:

```
hdfs dfs -put ueaMasterBD/input/* input
```

Obtenemos la siguiente respuesta:

```
put: `input/big-quijote.txt': File exists
```

Esto es debido a que Hadoop sigue la regla de "Write-Once-Read-Many" o WORM, por lo que no sobre escribirá ficheros existentes para prevenir pérdidas de datos accidentales.

Esta misma regla se aplica para escribir los resultados de MapReduce en **output**, donde abortará el proceso si ya existe el directorio previamente.

11. ¿Cómo se ha ejecutado el WordCount utilizando el framework de MapReduce?

El framework de Apache Hadoop aplica la estrategia de llevar las tareas computacionales a donde están almacenados los datos, siendo estas tareas los trabajos (jobs) de MapReduce. Por tanto, el nodo de

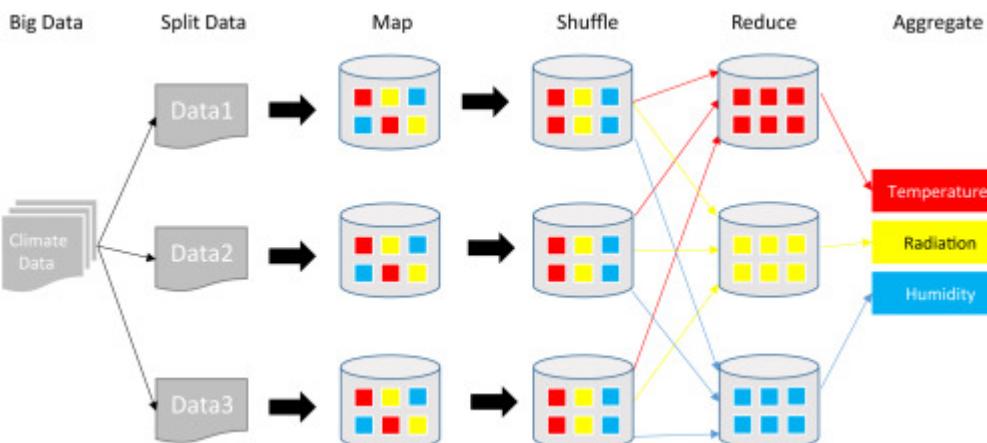
computación y el de almacenamiento son el mismo.

Cada bloque de 128MB del fichero es procesado en simultáneo. La aplicación WordCount fue ejecutada de la siguiente forma:

- Cada bloque de 128MB es asignado a un Mapper. Este lee la data, y aplica la función definida en la aplicación para transformarla en un conjunto de pares `<key, value>`. Es necesario que los datos producidos sean del tipo `<key, value>` porque MapReduce opera exclusivamente con este tipo de datos.
- Los pares `<key, value>` de todos los nodos son redistribuidos de acuerdo al valor `key`. Una función hash determina cuál nodo obtiene cuál conjunto de pares, para asegurar una carga de trabajo balanceada.
- Cada nodo que ejecutó la función Mapper ahora ejecuta la función Reducer, mediante una agregación de datos según el valor `key`.
- Una vez ejecutada la función Reducer por todos los nodos, se consolidan los resultados en un archivo en HDFS.

En resumen, el proceso se compone de las siguientes fases:

- Input = lectura del fichero
- Splitting = división en bloques
- Mapping = creación de conjuntos `<key, value>`
- Shuffling = agrupado por `key` y redistribución
- Reducing = agregación de datos de cada conjunto
- Resultado final de las agregaciones en un solo archivo



(imagen de <https://www.sciencedirect.com/topics/engineering/mapreduce>)

12. (Opcional) Crea una nueva versión del código Java basado en MapReduce para resolver la siguiente pregunta: ¿Cuáles son las 10 primeras palabras sin contar preposiciones?

Se decidió emplear la utilidad [Hadoop Streaming](#) para crear una nueva aplicación MapReduce empleando python, que descartará las preposiciones durante la ejecución del WordCount.

Los scripts iniciales en python para el Mapper y el Reducer fueron tomados del tutorial [Hadoop Streaming Using Python - Word Count Problem](#).

Estos scripts fueron modificados para descartar [las 23 preposiciones del español](#), y hacerlos compatibles con la versión mas moderna de python.

[python-no-prepocisiones/mapper.py](#)

[python/reducer.py](#)

Deben ejecutarse los comandos `chmod 777 mapper.py` y `chmod 777 reducer.py` para cambiar los permisos a `read, write, execute`.

La versión mas actualizada del `hadoop-streaming.jar` fue descargada del repositorio de maven [org/apache/hadoop/hadoop-streaming/3.4.2](#).

El siguiente comando ejecutará los script de python y guardará el resultado en el directorio `/user/hadoop/output3` del clúster HDFS:

```
hadoop jar /home/hadoop/ueaMasterBD/hadoop-streaming-3.4.2.jar \
-input /user/hadoop/input/* \
-output /user/hadoop/output3 \
-mapper /home/hadoop/ueaMasterBD/tarea-1-sist-dis/python-no-
prepocisiones/mapper.py \
-reducer /home/hadoop/ueaMasterBD/tarea-1-sist-dis/python/reducer.py
```

Dando como resultado las 10 primeras palabras sin contar preposiciones:

que	10007552
y	8179712
la	5288448
el	4100608
no	2878464
se	2432512
los	2406912
las	1761280
lo	1750016
le	1743360

13. (Opcional) Crea una nueva versión del código Java basado en MapReduce para resolver la siguiente pregunta: ¿Cuáles son las 10 letras más repetidas?

Para responder esta pregunta, nuevamente se decidió emplear python y Hadoop Streaming. Se creó un nuevo script para el Mapper tomando como base el script del ejercicio anterior, esta vez incluyendo una función en regex que genera un `True` or `False` al comparar cada token con las letras del abecedario.

[python-single-letter/mapper.py](#)

El siguiente comando ejecutará los script de python guardando el resultado en el directorio **/user/hadoop/output5** del clúster HDFS:

```
hadoop jar /home/hadoop/ueaMasterBD/hadoop-streaming-3.4.2.jar \
-input /user/hadoop/input/* \
-output /user/hadoop/output5 \
-mapper /home/hadoop/ueaMasterBD/tarea-1-sist-dis/python-single-
letter/mapper.py \
-reducer /home/hadoop/ueaMasterBD/tarea-1-sist-dis/python/reducer.py
```

Como nota curiosa, el resultado arrojó solo 9 letras repetidas en el archivo **big-quijote.txt**:

y	8179712
a	4929024
o	595456
Y	343040
A	139264
e	36864
O	3584
i	512
X	512
