Implementación y Análisis de un Clúster de Almacenamiento Distribuido con Hadoop HDFS

Sebastián Acosta Lasso, Sebastián Fanchi López, Sergio Ernesto Mesa Bernal,
Juan Diego Peña Mayorga, Lizeth Portela Lozano, Jean Paul Rodríguez Moreno
Computación de Alto Desempeño
Pontificia Universidad Javeriana, Bogotá, Colombia

***Abstract*—Este informe detalla el proceso de diseño, implementación y evaluación de un clúster de almacenamiento distribuido utilizando el Sistema de Ficheros Distribuido de Hadoop (HDFS). El objetivo principal, definido en [1], es comprender la arquitectura maestro/esclavo de HDFS, configurando un entorno funcional sobre un clúster de máquinas Rocky Linux, y validar su operación a través de comandos estándar de gestión de datos. El análisis se centra en la descripción del proceso, la demostración del funcionamiento y las conclusiones extraídas de la experiencia práctica.**

***Index Terms*—Hadoop, HDFS, Clúster, NameNode, DataNode.**

I. Descripción general de HDFS

A. Introducción

Un Sistema de Ficheros Distribuido de Hadoop (HDFS) es una tecnología fundamental en ecosistemas enfocados a Big Data, diseñada para almacenar enormes volúmenes de datos a través de clústeres. La característica principal de HDFS es la excelente tolerancia a fallos, que se logra mediante la replicación de bloques de datos en múltiples nodos, lo que asegura la disponibilidad de los datos incluso si algunos nodos del clúster fallan.

Otras características principales de HDFS son:

- **Modelo de coherencia simple:** Significa que esta optimizado para un modelo de escritura única, lecturas múltiples (WORM), por lo que un archivo, una vez cerrado, es generalmente inmutable.
- **Optimización para archivos grandes:** HDFS está diseñado para manejar ficheros de tamaño de gigabytes a terabytes, evitando la sobrecarga de gestionar metadatos para millones de archivos pequeños, esta característica a su vez depende de las maquinas usadas en el cluster.
- **Procesamiento por lotes (Batch):** Prioriza el alto rendimiento en el procesamiento de grandes volúmenes de datos (throughput) sobre la baja latencia, haciéndolo ideal para aplicaciones de análisis de datos a gran escala.

B. Arquitectura maestro/esclavo

HDFS opera bajo una arquitectura maestro/esclavo [2], como se ilustra en la Fig. 1.

Los componentes principales son:

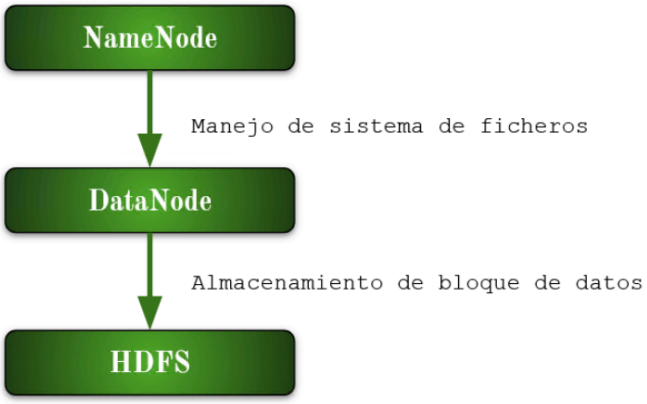

Fig. 1. Arquitectura conceptual de HDFS.[1]

- **NameNode o el maestro:** Es el cerebro del sistema, pues gestiona el espacio de nombres del sistema de ficheros (la estructura de directorios) y mantiene los metadatos de todos los archivos, incluyendo la ubicación de los bloques de datos ocupados en los DataNodes.
- **DataNodes o los esclavos:** Son los trabajadores pues almacenan físicamente los bloques de datos de los archivos y responden a las solicitudes de lectura/escritura de los clientes, siguiendo las instrucciones del NameNode.

II. Implementación del Clúster HDFS

El clúster fue configurado sobre un conjunto de 6 máquinas con sistema operativo Rocky Linux. Un nodo fue designado como **Master** (`cadhead01`, ejecutando el NameNode) y los restantes como **Workers**, como se muestra en la Fig. 2.

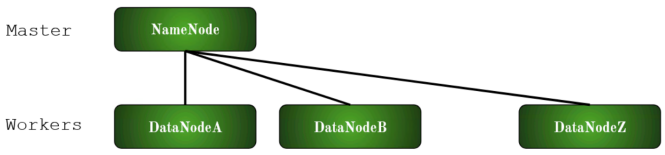

Fig. 2. Componentes del clúster HDFS implementado [1].

A. Configuración del Entorno

Se configuró un usuario `estudiante` y un grupo `conda` en todas las máquinas. Un paso muy crítico es establecer autenticación SSH sin contraseña desde el Master a los Workers, utilizando `ssh-keygen` y `ssh-copy-id`, para permitir la gestión remota de los demonios de Hadoop.

B. Configuración de Archivos Hadoop

Se editaron ficheros clave para definir la arquitectura. A continuación se explica el propósito y la configuración de cada uno:

- **hadoop-env.sh:** Este archivo permite definir variables de entorno específicas para Hadoop. Se configuró `JAVA_HOME` para apuntar a OpenJDK 8, versión estable y probada con Hadoop 3.x. `HADOOP_PID_DIR` especifica dónde se guardarán los archivos de identificación de proceso, y `HDFS_NAMENODE_OPTS` asignó un Heap de 4GB al NameNode, optimizando su rendimiento para gestionar metadatos con el recolector de basura ParallelGC.

Listing 1. Ajustes en hadoop-env.sh

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-
    openjdk
export HADOOP_PID_DIR=/mnt/sda1/Cluster/
    pid
#Configuración NameNode para usar
    parallelGC con 4GB Java Heap
export HADOOP_OPTS="-Djava.library.path=$
    HADOOP_HOME/lib/native"
export HADOOP_COMMON_LIB_NATIVE_DIR="$
    HADOOP_HOME/lib/native/"
export HDFS_NAMENODE_OPTS="-XX:+
    UseParallelGC -Xmx4g"
```

- **core-site.xml:** Actúa como el archivo de configuración central, donde se parametriza `fs.defaultFS`, que define el URI del NameNode (‘hdfs://IPMASTER:9000‘), con esto se le indica a todos los clientes y demonios del clúster cuál es el punto de entrada al sistema de ficheros distribuido.

Listing 2. Ajustes en core-site.xml

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://cadhead01:9000</
            value>
    </property>
    <property>
        <name>dfs.permissions</name>
        <value>false</value>
    </property>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/mnt/sda1/hadoop/tmp</value
            >
    </property>
    <property>
        <name>hadoop.native.lib</name>
        <value>true</value>
    </property>
</configuration>
```

- **hdfs-site.xml:** Contiene las propiedades específicas de HDFS. Se configuró `dfs.namenode.name.dir` y `dfs.datanode.data.dir` para indicar las rutas de almacenamiento local de metadatos y datos, respectivamente, ademas, una propiedad clave es `dfs.replication` que se estableció en ‘3‘, lo que instruye a HDFS a crear tres réplicas de cada bloque de datos para garantizar una alta tolerancia a fallos.

Listing 3. Ajustes en hdfs-site.xml

```
<configuration>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:/mnt/hadoop/NameNode</
            value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:/mnt/hadoop/DataNode</
            value>
    </property>
    <property>
        <name>dfs.replication</name>
        <value>3</value>
    </property>
    <property>
        <name>dfs.permission</name>
        <value>false</value>
    </property>
    <property>
        <name>dfs.webhdfs.enabled</name>
        <value>true</value>
    </property>
    <property>
        <name>ipc.maximum.data.length</
            name>
        <value>134217728</value>
    </property>
</configuration>
```

- **workers:** Es un simple archivo de texto que lista, uno por línea, los hostnames de todas las máquinas que deben actuar como DataNodes, utilizado por los scripts de arranque para saber a qué nodos conectarse y levantar los servicios correspondientes.

Listing 4. Ajustes en workers

```
cad01-nfs01
cadcliente01
cad01-w000
cad01-w001
cad01-w002
```

- **.bashrc:** En todos los nodos, se exportaron las variables de entorno de Hadoop para integrar sus comandos al `PATH` del sistema.

Listing 5. Ajustes en .bashrc

```
# VARIABLES DE ENTORNO DE HADOOP
# AJUSTADO para Java 8
export JAVA_HOME=/usr/lib/jvm/jre-1.8.0-
    openjdk
```

```
export HADOOP_HOME=~/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=$HADOOP_HOME
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/
    hadoop
export PATH=$PATH:$HADOOP_HOME/sbin:$
    HADOOP_HOME/bin
export HDFS_NAMENODE_USER="estudiante"
export HDFS_DATANODE_USER="estudiante"
export HDFS_SECONDARYNAMENODE_USER="
    estudiante"
```

C. Retos técnicos durante la configuración

El montaje del clúster presentó varios desafíos prácticos, cuya resolución fue clave para el aprendizaje.

1) **Conectividad y versión de Java:** Mas halla de los problemas de conexión SSH, nuestro mayor reto aquí fue elegir la versión Java y se validó con `java -version` que la versión de Java activa era la esperada por Hadoop, ajustando la variable `JAVA_HOME` en todos los nodos.
2) **Gestión de almacenamiento y permisos:** Uno de los errores más frecuentes fue "Permission Denied" al iniciar los DataNodes. La solución consistió en crear manualmente las carpetas de datos en cada nodo y aplicarles los permisos correctos, asignando la propiedad al usuario y grupo adecuados (`estudiante:conda`) y concediendo acceso total (‘chmod 777‘). La verificación se realizó con el comando ‘ls -l‘ como se muestra en la Fig. 3. Usamos los comandos mostrados a continuacion para esta certificación.

Listing 6. Comandos para carpetas y permisos

```
# MASTER
$ sudo mkdir -p /mnt/hadoop
$ sudo mkdir -p /mnt/hadoop/NameNode
$ sudo chown -R estudiante:conda /mnt
$ sudo chmod -R 777 /mnt
$ sudo chown -R estudiante:conda /mnt/
    hadoop
$ sudo chmod -R 777 /mnt/hadoop
$ sudo chown -R estudiante:conda /mnt/
    hadoop/NameNode
$ sudo chmod -R 777 /mnt/hadoop/NameNode

# En cada WORKER
$ sudo mkdir -p /mnt/hadoop
$ sudo mkdir -p /mnt/hadoop/DataNode
$ sudo chown -R estudiante:conda /mnt
$ sudo chmod -R 777 /mnt
$ sudo chown -R estudiante:conda /mnt/
    hadoop
$ sudo chmod -R 777 /mnt/hadoop
$ sudo chown -R estudiante:conda /mnt/
    hadoop/DataNode
$ sudo chmod -R 777 /mnt/hadoop/DataNode
```

3) **Verificación de demonios con JPS:** Inicialmente, los procesos ‘DataNode‘ no aparecían en el resultado de `jps` en algunos workers, por lo que realizamos multiples depuraciones analizando las variables en `.bashrc`, los archivos de configuración y los permisos de las carpetas en cada nodo, hasta lograr que todos los demonios se ejecutaran correctamente.

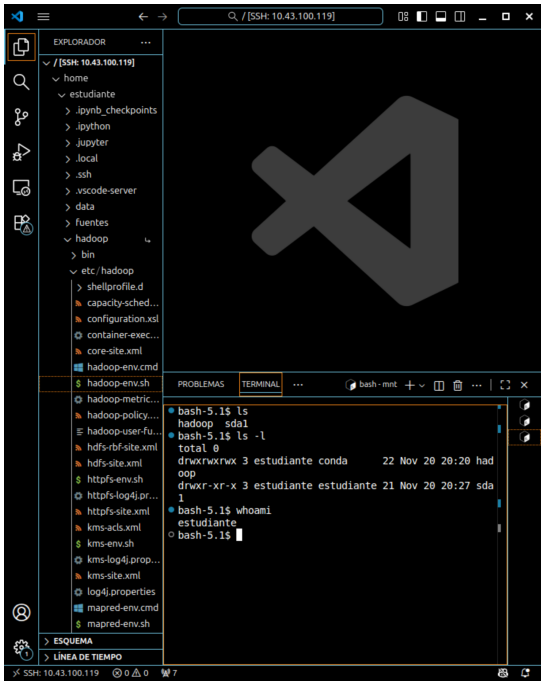

Fig. 3. Verificación de permisos sobre el directorio `/mnt`.

D. Puesta en marcha

Finalmente, el clúster fue inicializado desde el nodo Master con los comandos: `hdfs namenode -format` (que se debia ejecutar una sola vez [y por ejecutarlo mas de una tuvimos los retos tecnicos expuestos anteriormente]) y `start-dfs.sh`.

La correcta operación fue validada con `jps` en todos los nodos y accediendo a la interfaz web de HDFS, (Fig. 4). En esta interfaz, se pudo confirmar la observación central del despliegue: cada uno de los 5 DataNodes trabajadores aportaba aproximadamente 20 GB de espacio de almacenamiento al sistema de ficheros distribuido, lo que resulta en una capacidad bruta total configurada de aproximadamente 120 GB para el clúster.

Es crucial destacar la diferencia entre la capacidad bruta y la capacidad útil debido al factor de replicación, dado que para esta prueba se estableció `dfs.replication` en 3, cada bloque de datos escrito en HDFS se replica tres veces a través de los DataNodes para garantizar la tolerancia a fallos, por lo que en consecuencia, la capacidad útil real del clúster para almacenar datos únicos es:

$$CapacidadÚtil = \frac{CapacidadBruta}{FactordeReplicación} = \frac{120GB}{3} = 40GB$$

Esto demuestra uno de los principios de diseño más importantes de HDFS, el compromiso entre la resiliencia de los datos y el costo de almacenamiento, así mismo logramos evidenciar otro principio intrínseco que es el principio de escalabilidad horizontal, que sirve para aumentar la capacidad útil del clúster, bastaría con añadir más nodos DataNode al archivo `workers`, aplicar los comandos de carpetas y permisos, y por ultimo reiniciar los servicios.

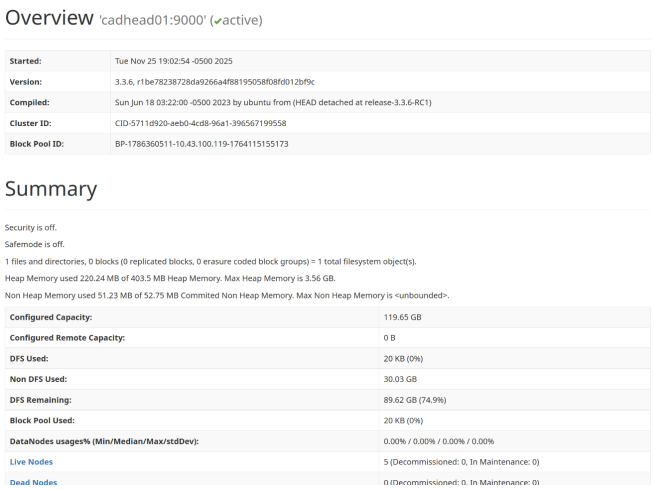

Fig. 4. Interfaz web del NameNode mostrando el estado del clúster.

III. Análisis de Comandos y Actividades

A continuación, ejemplificamos el uso de los comandos básicos de HDFS para demostrar el funcionamiento del clúster, para la secuencia de operaciones sigue un flujo lógico: crear una estructura de directorios, subir y verificar datos, manipularlos y finalmente, analizarlos. Para ello, creamos dos directorios base (`/memoria` y `/cuento`) y dos archivos locales (`recuento.txt` y `CHARLIE.txt`) con contenido textual.

1) ***hdfs dfs -mkdir:*** **Descripción:** Este comando, análogo al de Linux, crea un nuevo directorio en la ruta especificada dentro de HDFS. Es el primer paso para organizar el espacio de nombres.

Ejemplo 1 y 2: Creamos los directorios raíz para nuestro ejercicio, `/memoria` y `/cuento`, para separar los datos temáticamente.

```
$ hdfs dfs -mkdir /memoria
$ hdfs dfs -mkdir /cuento
```

Como es habitual en comandos Unix exitosos, no producen ninguna salida si la operación se completa correctamente.

2) ***hdfs dfs -ls:*** **Descripción:** Lista el contenido de un directorio en HDFS, mostrando permisos, factor de replicación, propietario, grupo, tamaño y nombre.

Ejemplo 1: Verificamos la creación de los directorios listando el contenido de la raíz (`/`).

```
$ hdfs dfs -ls /
Found 2 items
drwxr-xr-x - estudiante supergroup 0 2025-11-25 20:26 /
    cuento
drwxr-xr-x - estudiante supergroup 0 2025-11-25 20:26 /
    memoria
```

3) ***hdfs dfs -put:*** **Descripción:** Transfiere archivos desde el sistema de ficheros local al sistema de ficheros HDFS. Es el método principal para ingestar datos al clúster.

Ejemplo 1: Se sube el archivo local `recuento.txt` al directorio `/memoria` en HDFS.

```
$ hdfs dfs -put recuento.txt /memoria/
```

Ejemplo 2: Se sube el archivo local `CHARLIE.txt` al directorio `/cuento`.

```
$ hdfs dfs -put CHARLIE.txt /cuento/
```

4) ***hdfs dfs -cat:*** **Descripción:** Concatena y muestra el contenido de uno o más archivos de HDFS en la salida estándar. Es ideal para una inspección rápida del contenido sin necesidad de descargar el fichero.

Ejemplo 1 y 2: Verificamos que los archivos se subieron correctamente mostrando su contenido.

```
$ hdfs dfs -cat /memoria/recuento.txt
Fue divertido, de verdad, que buenas risas tuvimos, los 6
    disfrutamos mucho, es especial cuando teníamos que
    perdonar el francés, este segundo módulo sin duda irá
    en nuestra MEMORIA
$ hdfs dfs -cat /cuento/CHARLIE.txt
Un día, la C, celosa de un guiño indecente, partió al señor
     H por la mitad, lo que provocó que la A afilara su vé
    rtice y degollara a la C, instante que aprovechó la R
    para rodar violentamente y triturar hasta el polvo a la
     A; al ver el baño de sangre, el L usó su ángulo recto
    para ahorcar a la R, pero no vio venir al señor I, que
    se lanzó como una jabalina suicida atravesándole la
    garganta, dejando a la pobre E tan traumatizada que
    decidió detonar sus tres patas en una explosión de
    tinta, y justo cuando la última vocal se desintegró, el
     monitor del quirófano pitó en una línea recta
    infinita: Charlie se había ido, dijo el cirujano a las
    4:02 p.m.
```

5) ***hdfs dfs -cp:*** **Descripción:** Copia un archivo o directorio de una ubicación a otra dentro de HDFS.

Ejemplo 1: Creamos una copia de seguridad de `recuento.txt` dentro de su mismo directorio, renombrándola.

```
$ hdfs dfs -cp /memoria/recuento.txt /memoria/
    recuento_copia.txt
```

6) ***hdfs dfs -mv:*** **Descripción:** Mueve (o renombra) un archivo o directorio de una ubicación a otra dentro de HDFS.

Ejemplo 1: Movemos la copia de seguridad recién creada, `recuento_copia.txt`, desde el directorio `/memoria` hacia `/cuento`, demostrando así la capacidad de mover archivos entre carpetas.

```
$ hdfs dfs -mv /memoria/recuento_copia.txt /cuento/
```

7) ***hdfs dfs -rmr:*** **Descripción:** Borra recursivamente un directorio y todo su contenido. Es un comando destructivo que debe usarse con precaución.

Ejemplo 1: Para demostrar la funcionalidad, eliminamos el archivo que acabamos de mover.

```
$ hdfs dfs -rmr /cuento/recuento_copia.txt
rmr: DEPRECATED: Please use '-rm -r' instead.
Deleted /cuento/recuento_copia.txt
```

*8) **hdfs dfs -du** y **-dus**:* **Descripción:** Muestran el uso de disco ('Disk Usage'). `-du` lista el tamaño de cada elemento dentro de un directorio, mientras que `-dus` muestra un resumen (*summarize*) con el tamaño total.

Ejemplo 1 y 2: Comparamos ambas funcionalidades sobre el directorio `/memoria`. `-du` detalla el tamaño del archivo contenido, mientras que `-dus` solo muestra el total del directorio.

```
$ hdfs dfs -du /memoria
182  546  /memoria/recuento.txt
$ hdfs dfs -dus /memoria
182  546  /memoria
```

*9) **hdfs dfs -stat**:* **Descripción:** Muestra estadísticas o metadatos de un archivo o directorio, como la fecha de modificación, el tamaño del bloque o el nombre.

Ejemplo 1: Obtenemos la fecha de última modificación de `recuento.txt`.

```
$ hdfs dfs -stat /memoria/recuento.txt
2025-11-26 01:27:48
```

Ejemplo 2: Obtenemos la fecha de modificación de `CHARLIE.txt`.

```
$ hdfs dfs -stat /cuento/CHARLIE.txt
2025-11-26 01:28:06
```

IV. Modularidad y automatización

Aunque el taller se centró en la configuración manual para fines de aprendizaje, el diseño de la implementación se apoya en los principios de modularidad y automatización que provee el ecosistema Hadoop.

A. Scripts de automatización

Si bien no se implementó un `Makefile` personalizado, se utilizaron intensivamente los scripts nativos de Hadoop, que funcionan como una herramienta de automatización para gestionar procesos complejos del clúster. Los más relevantes son:

- `start-dfs.sh`: Este script automatiza el arranque de todo el sistema de ficheros HDFS, loq ue hace es que internamente, lee la lista de nodos del archivo `workers` y utiliza SSH para conectarse a cada uno y lanzar el demonio `DataNode`, en simultaneo, inicia el `NameNode` en la máquina local (maestro), lo que encapsula una serie de operaciones distribuidas en un único comando.
- `stop-dfs.sh`: Realiza el proceso inverso al anterior, automatizando la detención ordenada de todos los servicios hdfs del clúster.
- `hdfs namenode -format`: Aunque es un comando de inicialización, automatiza la creación de la estructura de directorios y metadatos necesarios para el NameNode.

El uso de estos scripts es, en nuestra opinion, equivalente a tener targets como `make start-cluster` o `make stop-cluster`.

B. Modularidad de la configuración

El diseño de Hadoop se basa en un fuerte principio de modularidad, separando la configuración de la lógica de ejecución.

- **Archivos de configuración:** Parámetros críticos como el URI del NameNode (`core-site.xml`), las rutas de almacenamiento y el factor de replicación (`hdfs-site.xml`), y la lista de nodos trabajadores (`workers`) están externalizados permitiendo modificar la topología o el comportamiento del clúster sin alterar ni una sola línea de código en los scripts de operación.
- **Variables de entorno:** Las rutas de instalación (`HADOOP_HOME`) y la versión de Java (`JAVA_HOME`) se gestionaron mediante variables de entorno en el archivo `.bashrc` y en `hadoop-env.sh`. Esta práctica permite que el sistema sea adaptable a diferentes máquinas donde las rutas de instalación podrían variar, manteniendo la configuración del clúster organizada y fácil de mantener.

Esta separación entre "qué hacer" (lógica de los scripts) y "cómo hacerlo" (parámetros de los archivos de configuración) es la esencia de un diseño modular y robusto.

V. Conclusiones

La implementación y operación del clúster HDFS desde cero ha sido una experiencia práctica de gran valor que permitió consolidar los conceptos teóricos de los sistemas de almacenamiento distribuido. A partir de la configuración, operación y análisis del clúster, resaltamos las siguientes conclusiones:

- **El éxito de un sistema distribuido reside en una configuración de entorno impecable, siendo más crítico que el propio software.** Se demostró que los mayores desafíos en el despliegue de HDFS no provienen de la herramienta en sí, sino de la infraestructura subyacente, debido a que la correcta configuración de la resolución de nombres, la autenticación SSH sin contraseña y, sobre todo, la meticulosa gestión de permisos de directorios ('chown', 'chmod') en cada nodo, fueron los factores determinantes para el éxito del clúster, lo que enfatiza que la administración de sistemas y redes es un prerrequisito fundamental y la principal fuente de errores en la computación distribuida.
- **La simplicidad conceptual de HDFS oculta una considerable complejidad operativa que demanda una depuración iterativa y multinodo.** Aunque la arquitectura maestro/esclavo es sencilla, su puesta en marcha requirió una atención minuciosa al

detalle en múltiples archivos y hosts. Problemas como la falta de demonios 'DataNode' en `jps` o fallos por versiones de Java inconsistentes, evidenciaron que la depuración no es lineal, por ende, la solución requirió un ciclo sistemático de: *configurar, arrancar, verificar, detener y corregir*, aplicado de forma consistente en todos los nodos, desarrollando habilidades de debugging esenciales para entornos distribuidos.

- **HDFS es la capa fundamental tangible y verificable que habilita el ecosistema de Big Data.** A través de la configuración práctica del factor de replicación ('dfs.replication') y el análisis de la capacidad con `hdfs dfs -du`, los principios de tolerancia a fallos y escalabilidad horizontal de HDFS dejaron de ser teóricos, pues al finalizar el ejercicio, queda claro cómo HDFS actúa como el cimiento sobre el que operan herramientas de procesamiento como Spark, Hive o HBase. La comprensión de esta capa base esimportante para diseñar y analizar cualquier sistema de procesamiento de alto volumen de datos.

References

[1] J. Corredor, "Sistema de Ficheros Distribuido Hadoop," *Material de clase, Procesamiento de Alto Volúmen de Datos, Pontificia Universidad Javeriana*, Bogotá, Colombia, 2025.

[2] Apache Software Foundation, "HDFS Architecture Guide," *Documentación Oficial de Hadoop 3.3.6*. [En línea]. Disponible en: https://hadoop.apache.org/docs/r3.3.6/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

[3] T. White, "Hadoop: The Definitive Guide," 4th ed., O'Reilly Media, Inc., 2015.