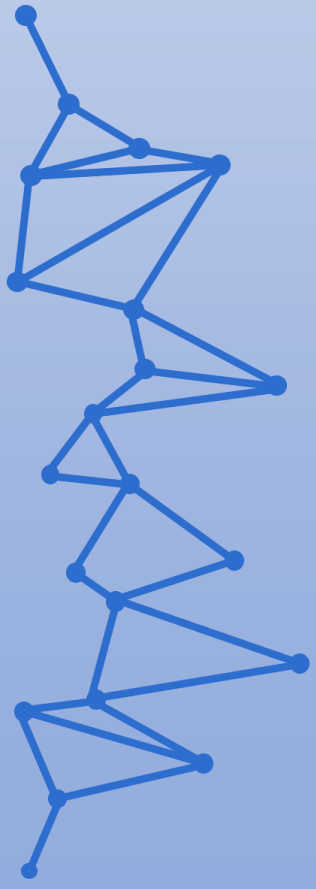
Curso de Especialización de
Inteligencia Artificial y Big Data (IABD)



Big Data Aplicado

UD02. Almacenamiento y procesamiento en Hadoop.
Resumen.

JUAN ANTONIO GARCIA MUELAS

Core de Hadoop.

Su base está formada por **HDFS**, **YARN** y **MapReduce**:

- ✓ **HDFS**, que es la **capa de almacenamiento**. Es un sistema de almacenamiento **distribuido**, lo que significa que las **operaciones no las realiza un único servidor**, sino que múltiples servidores trabajan coordinados para almacenar u ofrecer los datos.
  Como sabes, **HDFS** (Hadoop Distributed File System) **se inspiró en** el paper de Google denominado **Google File System**, donde se explicaba como resolvió el problema de almacenar "todo internet".

  Sus **características** son:

  - **Es un sistema de ficheros distribuido**, es decir, se **ejecuta sobre diferentes nodos** que trabajan en conjunto ofreciendo a los usuarios y aplicaciones que utilizan el sistema, un interfaz como si sólo hubiera un único servidor por detrás. Es decir, para los usuarios de HDFS, **es transparente** su modelo distribuido, no teniendo que conocer su funcionamiento interno.
  - Está **diseñado** para ejecutarse sobre **hardware commodity**, es decir, no requiere unos servidores específicos o costosos. Esto conlleva la necesidad de poder sobreponerse a los fallos que pudieran tener los servidores o algunas partes de los servidores.
  - Está **optimizado** para almacenar **ficheros de gran tamaño y** para hacer **operaciones** de lectura o escritura **masivas**. Su objetivo es cubrir los casos de uso de analítica masiva, no los casos de uso que dan soporte a las operaciones de las empresas.
  - Tiene capacidad para **escalar horizontalmente** hasta volúmenes de Petabytes y miles de nodos, y está diseñado para poder dar soporte a **múltiples clientes** con acceso concurrente. La **escalabilidad se consigue añadiendo más servidores**, lo cual es una operación relativamente sencilla, y en cuanto a la posibilidad de dar **soporte a múltiples clientes,** es una característica importante, ya que existen sistemas de almacenamiento masivo que no permiten el acceso concurrente de más de un cliente, o si lo soportan, su rendimiento decrece en gran medida (por ejemplo, los sistemas de almacenamiento basados en cinta).
  - No establece **ninguna restricción sobre los tipos de datos** que se almacenan en el sistema, ya que éstos pueden ser estructurados, semiestructurados o no disponer de ninguna estructura, como el caso de imágenes o vídeos.
  - HDFS tiene una orientación "**write-once, read many**", que significa "se escribe una vez, se lee muchas veces", es decir, asume que un archivo una vez escrito en HDFS no se modificará, aunque se puede acceder a él muchas veces.

    Permite guardar y recuperar ficheros. Tienen un nombre y una extensión, distinguiendo entre mayúsculas y minúsculas, es decir, para HDFS, el fichero “video.mov” y “1ideo.mov” son diferentes.

    Para saber más

    El sistema de permisos que utiliza HDFS es el mismo que se utiliza en sistemas Unix. Este sistema define los siguientes permisos para cada fichero o directorio:

    - Lectura (r): el fichero o el directorio se puede leer pero no modificar.
    - Escritura (w): el fichero se puede modificar. En el caso de los directorios, permite añadir o borrar ficheros.
    - Ejecución (x): el fichero se puede ejecutar.

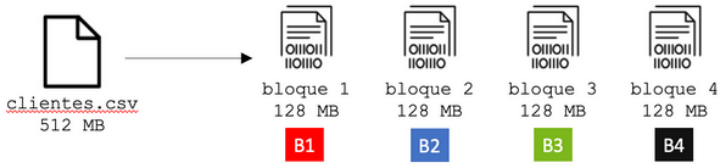
Para cada fichero, se le da permiso de lectura, escritura o ejecución a tres grupos de usuarios:

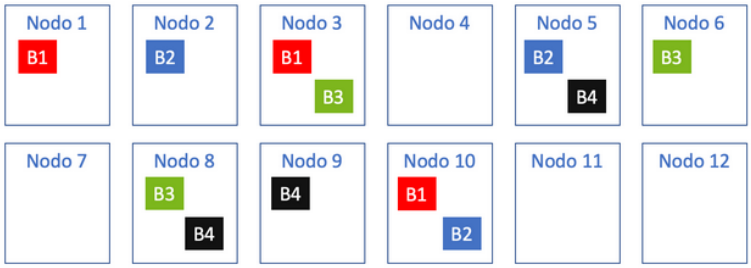
- Al propietario del fichero.
- Al grupo de usuarios al que pertenece el usuario.
- Al resto de usuarios.

Por lo tanto, cuando veas los permisos de un fichero, podrás encontrar una cadena de caracteres como éstas:

- rw-r--r-- : permiso de lectura y escritura para el propietario y de lectura para el grupo al que pertenece el propietario y para el resto de los usuarios.
- rwxrwxrwx : permiso de lectura, escritura y ejecución para todos los usuarios

Los ficheros **se dividen en bloques de 128 megabytes por defecto** (muy grandes respecto a otros sistemas), aunque el fichero original sea de 1kilobyte y por eso está orientado a ficheros y lecturas masivas. Luego, los bloques se almacenan en distintos nodos (**replicación x 3**), permitiendo la tolerancia a fallos, y **que no se pierdan datos**, por una caída o error.





Su **arquitectura** consta de 3 tipos de servicios y nodos:

Namenode	• **Coordina** el trabajo de los **`Datanodes`.** • **Almacena** toda la **información** sobre los ficheros, los bloques y los **`Datanodes`** (actúa de **maestro**). • **Verifica** que los **`Datanodes`** están **activos**. • Es el **único punto de fallo** de HDFS. • Suelen tener mecanismos de **tolerancia a fallos**: redundancia de discos, etc.
Secondary Namenode	• **Facilita** el proceso de **arranque** de un **`Namenode`** en caso de caída (**no es un nodo de respaldo**). • **Almacena** el **estado** de HDFS mediante dos ficheros: **`FsImage`** (instantánea de los metadatos) y **`EditLog`** (registro de cambios en los metadatos. Se borra con cada nueva instantánea). • Se **ejecuta en un nodo diferente al `Namenode`**.

Datanode	• **Almacena y lee los bloques** de los ficheros almacenados en HDFS. • **No** dispone de **información de** los **ficheros** o estructura de directorios de HDFS. • **Envía** un **mensaje** al **Namenode** para avisar que se encuentra activo. • En caso de **caída, no se pierden datos** y HDFS sigue funcionando correctamente. • Se ejecuta en nodos **hardware commodity**, habitualmente con muchos discos.

Los **datos** que se escriben **en HDFS son inmutables** (no pueden modificarse).

Los **datos** que se escriben **en HDFS son inmutables** (no pueden modificarse). El cliente se comunica directamente con los **Datanodes**, sin que los datos pasen por el **Namenode**, **evitando** que este no sea un **cuello de botella**. El cliente tiene que tener conectividad con todos los **Datanodes**.

Los **principales interfaces** con los que poder usar el sistema de ficheros, son los mencionados a continuación:

- **Cliente de línea de comandos**: HDFS dispone de un amplio número de comandos que pueden ser ejecutados en consola. Estos comandos representan la práctica totalidad de las operaciones que se pueden realizar con HDFS.

  hadoop **fs** nos proporcionará todas las **funcionalidades** sobre HDFS.

  mkdir, se usa para **crear directorios** en HDFS.

  Sintaxis:
  - `hadoop fs -mkdir [-p] <path>`

  Ejemplos:
  - `hadoop fs -mkdir /tmp/BDA01`
  - `hadoop fs -mkdir /tmp/BDA01/hadoop-core`

  ls, se usa para **listar directorios** en HDFS, es decir, **para mostrar su contenido**. La opción **-R** se puede utilizar para hacer un **listado recursivo**, es decir, para mostrar el contenido de los subdirectorios que están dentro del directorio que vamos a listar.

  Sintaxis:
  - `hadoop fs -ls [-d] [-h] [-R] <path>`

  Ejemplos:
  - `hadoop fs -ls /tmp/`
  - `hadoop fs -ls -R /tmp`

  put, **copia** archivos del sistema de **archivos local a HDFS**.

  Es similar a **-copyFromLocal**

  Sintaxis:
  - `hadoop fs -put [-f] [-p] <localsrc>…<dst>`

  Ejemplo:
  - `hadoop fs -put /tmp/hadoop-state-pusher.config/tmp/BDA01`

  get, **copia archivos de HDFS al** sistema de archivos **local**.

Es similar al comando **`-copyToLocal`**

Ejemplo:

- `hadoop fs -cp /tmp/BDA01/hadoop-state-pusher.config/tmp/`

`cat`, se usa para **mostrar el contenido** de un archivo.

Ejemplo:

- `hadoop fs -cat /tmp/BDA01/hadoop-state-pusher.config`

`cp`, **copia archivos de un directorio a otro** dentro de HDFS.

Ejemplo:

- `hadoop fs -cp /tmp/BDA01/hadoop-state-pusher.config/tmp/`

`rm` , se usa para **eliminar un archivo** de HDFS. La opción **`-R`** se puede usar para la **eliminación recursiva**, es decir, para borrar además los **subdirectorios** que están en el directorio indicado.

Ejemplo:

- `hadoop fs -rm /tmp/BDA01/hadoop-state-pusher.config`

`mv`, mueve (no copia, como cp) archivos de HDFS de una ruta a otra

Ejemplo:

- `hadoop fs -mv /tmp/BDA01/hadoop-state-pusher.config /tmp/`

`setrep`, modifica el **factor de replicación** de un fichero o un directorio. Ya sabes que el factor de replicación **por defecto es 3**. Con este comando se puede modificar para un fichero o directorio concreto.

En HDFS, lo ideal es poner un factor de replicación alto a los ficheros que son críticos o muy importantes.

Ejemplo:

- `hadoop fs -setrep 6 /tmp/BDA01/hadoop-state-pusher.config`

Cuando accedemos por terminal a una máquina, que suele ser la **máquina frontera**, y navegamos por su sistema de ficheros, lo estaremos haciendo sobre el disco o los discos que tiene esa máquina. Cuando ejecutamos el comando hadoop **`fs`** , éste se ejecutará sobre el sistema de ficheros de HDFS, que es diferente al de la máquina en la que estamos.

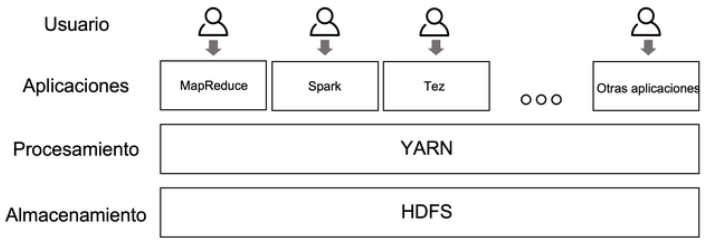
Cuando queremos **subir un fichero a HDFS, lo habitual es copiarlo primero en la máquina frontera, y** posteriormente **subirlo a Hadoop con** el comando **`put`**.

- **Java API**: HDFS está escrito en **Java de forma nativa** y ofrece un API que puede ser utilizado por aplicaciones con el mismo lenguaje.
- **RestFul API(WebHDFS)**: para poder **utilizar HDFS desde otros lenguajes**, ofrece su funcionalidad **mediante un servicio HTTP**, mediante el **protocolo WebHDFS.** Ofrece un **rendimiento inferior al API de Java** al utilizar HTTP como capa de transporte, por lo que no debería utilizarse para operaciones masivas o con alto volumen de datos.
- **NFS interface (HDFS NFS Gateway)**: es posible montar HDFS en el sistema de archivos de un cliente local utilizando la puerta de enlace NFSv3 de Hadoop. Luego puede usar las utilidades de Unix (como `ls` y `cat`) para interactuar con el sistema de archivos, cargar archivos y, en general, usar bibliotecas POSIX para acceder al sistema de archivos desde cualquier lenguaje de programación. Agregar a un archivo funciona,

pero las modificaciones de un archivo no, ya que HDFS sólo puede añadir datos a un archivo.

- **Librería C**: HDFS ofrece una librería escrita en C, llamada **`libhdfs`**, que tiene un buen rendimiento, pero que no suele ofrecer toda la funcionalidad del API Java.

✓ **YARN**, que es el **gestor de los procesos** (recursos) que se ejecutan **en el clúster**. Haciendo un símil, YARN (Yet Another Resource Negotiator) es el sistema operativo de Hadoop, pero sin gestionar el almacenamiento.



Con YARN, **se divide** la capa de computación en dos:

- Un **gestor de procesos** que se ejecutan en el clúster, para coordinar aplicaciones, asignar recursos y prioridades, permitir su convivencia, etc.
- Las **aplicaciones**, que pueden desarrollarse en un marco de ejecución más ligero, sin un modelo estricto para ejecución y más libertad para desarrollarlas.

YARN, por lo tanto, **realiza las siguientes tareas**:

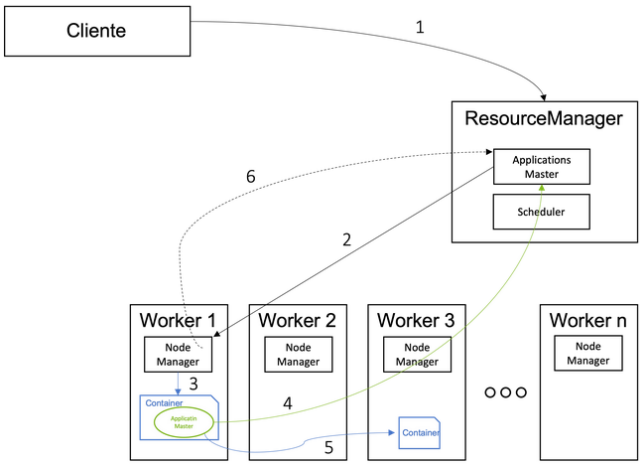
- **Ofrece un API** a las aplicaciones mucho **menos estricto que MapReduce**, no impone la forma en la que hacer el procesamiento de datos. Las operaciones del API de YARN son del tipo:
  - Ejecutar una aplicación en el clúster.
  - La aplicación necesita X recursos de memoria y CPU.
  - Parar la ejecución de una aplicación.
  - etc.
- **Ejecuta las aplicaciones** en el clúster (ejecuta el código en diferentes nodos), les da los recursos de CPU y memoria necesarios, etc.
- **Sincroniza la ejecución simultánea** de las aplicaciones, decidiendo el nivel de **prioridad** en cada aplicación **mediante la configuración del Scheduler**, recursos a asignar cuando compiten por los mismos recursos, etc. Son políticas configuradas por el administrador de YARN.
- **Monitoriza la ejecución** de las aplicaciones, y en caso de error por un fallo de algún nodo, vuelve a lanzar el trabajo, garantizando la tolerancia a fallos.
  Si un nodo falla durante la ejecución de una tarea, YARN se da cuenta y rearranca esa tarea en otro nodo.
- **Gestionar los recursos** del clúster disponibles, vigila los nodos activos, qué capacidad de memoria y CPU tiene cada nodo, etc.

YARN puede ejecutar aplicaciones de procesamiento en streaming.

Todas las tareas de las **aplicaciones YARN se ejecutan en contenedores** (unidad mínima de recursos de ejecución). **Por defecto,** su tamaño es de **8 gigabytes**. La cantidad de tareas está limitada por la cantidad de contenedores del clúster. Ejemplo, en un clúster de 20 nodos, con 256 gigabytes de RAM y 12 cores por nodo, habrá un total de 5TB (256x20) de

RAM y 240 (20x12) cores disponibles. Tendrá un máximo de 160 (20x8) contendores disponibles y por tanto podrá ejecutar como máximo 160 tareas de forma recurrente.

A diferencia de HDFS, donde los clientes que quisieran leer o escribir datos en el sistema, sí debían tener acceso a los nodos worker, **en YARN el cliente sólo interactúa con el `ResourceManager`, que coordina la ejecución de trabajos,** por lo que **no necesita conectividad con los nodos worker.**



- ✓ **MapReduce**, que es un **modelo de programación para desarrollar tareas** de procesamiento de datos masivos. Es un **framework** para escribir fácilmente aplicaciones que procesan **grandes cantidades de datos** en **paralelo** (la ejecución se divide en partes pequeñas y cada parte pequeña se ejecuta en paralelo, lo que **facilita la escalabilidad o la tolerancia a fallos**) en grandes **clústeres** (miles de nodos) de **hardware commodity** de manera **confiable** y **tolerante a fallos**.

  Un trabajo de MapReduce se compone de **cinco etapas** distintas, ejecutadas en orden:

  1. **Envío** del trabajo, aceptación y distribución en el clúster.
  2. **Ejecución** de la fase **`map`**.
  3. **Ejecución** de la fase **`shuffle`**.
  4. **Ejecución** de la fase **`order`**.
  5. **Ejecución** de la fase **`reduce`**.

  De todas estas fases debes saber que **el programador sólo suele programar** la fase **`map`** y **`reduce`**, siendo el resto ejecutadas de forma automática por MapReduce en base a los parámetros de configuración.