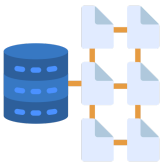
Sistemas Operativos y Distribuidos

Laboratorio Nº 4



Distributed file system (DFS).

Un DFS es un sistema de archivos cuyos componentes (clientes, servidores y dispositivos de almacenamiento) **están dispersos en diferentes máquinas de una red**. A diferencia de un sistema centralizado, el almacenamiento es múltiple e independiente, y todas las operaciones se realizan a través de la red.
Su meta ideal es ser **transparente** para el usuario. Esto significa que debe parecer y funcionar exactamente como un sistema de archivos local y convencional, ocultando la complejidad de acceder a archivos remotos. El sistema se encarga de localizar los archivos y transportar los datos automáticamente.

Componentes:

- **Servicio:** La función de proporcionar acceso a los archivos.
- **Servidor:** El *software* que provee el servicio, ejecutándose en una máquina.
- **Cliente:** Un proceso que solicita operaciones sobre los archivos (crear, leer, escribir) a través de una interfaz.

Características:

- **Multiplicidad y Autonomía:** Hay múltiples clientes y servidores independientes en el sistema.
- **Movilidad:** Un DFS transparente permite a un usuario acceder a su entorno (como su directorio personal) desde cualquier punto de acceso en la red.

Rendimiento:

El rendimiento se mide por el tiempo para satisfacer una solicitud. En un DFS, este tiempo es mayor que en un sistema local debido a la **sobrecarga de la red** (tiempo de envío de solicitudes/respuestas y procesamiento de protocolos de comunicación). El ideal es que su rendimiento se acerque lo más posible al de un sistema de archivos convencional.

Modelos Arquitectónicos Comunes:

1. **Cliente-Servidor:** Diseñado para compartir archivos de forma transparente entre clientes, como si estuvieran almacenados localmente. **Ejemplos:** NFS y Open AFS.
2. **Basado en Clústeres:** Optimizado para ejecutar aplicaciones en paralelo sobre grandes volúmenes de datos, con alta disponibilidad y escalabilidad. **Ejemplos:** Google File System y Hadoop HDFS.

Ejercicio

El objetivo de este laboratorio es desarrollar un **sistema de archivos distribuido (DFS)**. Este sistema está compuesto por varios **nodos de almacenamiento** que guardan partes de los archivos (bloques), un **servidor central** que gestiona la ubicación y recuperación de esos bloques, y **clientes** que ejecutan los comandos para administrar los archivos.

El desarrollo debe realizarse en el lenguaje de programación **Go**, utilizando **sockets** para la comunicación.

Descripción de los componentes

1. Namenode (servidor central)

- Mantiene un **índice** que asocia **nombre de archivo, lista de bloques** y **ubicación** (qué **datanodes** los tienen).
- Guarda su **metadata** en un archivo **JSON**.
- El **namenode** **no almacena los archivos**, pero sabe qué **datanodes** los contienen.
- Es el coordinador del sistema.
- Responde a los pedidos del **cliente** relacionados con:
  - Quién tiene un archivo (`get`).
  - Dónde guardar un archivo (`put`).
  - Qué archivos existen (`ls`).
- **Ejemplo de metadata (`metadata.json`):**

```
{
  "archivo1.txt": [
    {"block": "b1", "node": "10.0.0.2:5000"},
    {"block": "b2", "node": "10.0.0.3:5000"}
  ],
  "archivo2.txt": [
    {"block": "b1", "node": "10.0.0.3:5000"}
  ]
}
```

2. Datanodes (nodos de almacenamiento)

- Cada **datanode** es un **proceso** que guarda físicamente los bloques.
- Su **IP y puerto** están definidos en el código del **namenode** (en caso de modificarse, se edita el código fuente).
- Es el **nodo trabajador** encargado del almacenamiento.
- Cada **datanode** escucha en su propio **puerto TCP** (por ejemplo, `5000`).
- Puede manejar comandos simples:
  - `store <block_id>` → guarda un bloque recibido.

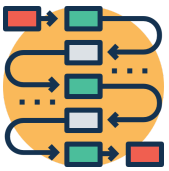
- `read <block_id>` → envía un bloque al **`cliente`**.
- Cada bloque se almacena como un archivo individual dentro de una carpeta `blocks/`.

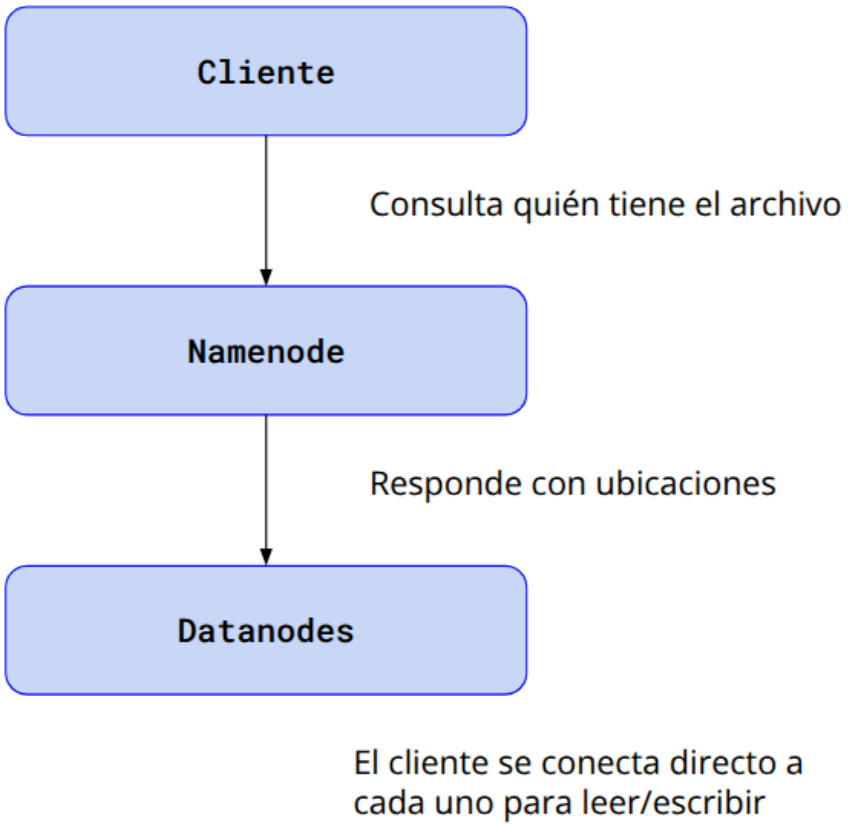
3. Cliente

- Es la **interfaz del usuario** con el sistema.
- Se conecta primero al **namenode** para consultar la ubicación de los datos.
- Luego se conecta directamente a los **`datanodes`** para transferir archivos.
- El **`cliente`** **ya conoce** la dirección IP y el puerto del **namenode** (el servidor central).
- **Comandos disponibles:**
  - `put <archivo_local>` → **subir archivo al DFS**
  - `get <nombre_archivo>` → **descargar archivo del DFS**
  - `ls` → **listar archivos disponibles**
  - `info <nombre_archivo>` → **mostrar información de un archivo**

En el caso de **PUT**, el **`cliente`** envía el archivo al **namenode**, que decide cómo fragmentarlo y en qué **`datanodes`** almacenarlo.

Flujos de cada operación del sistema





> put - Subir un archivo al DFS

El **Cliente**:

1. Abre el archivo local.
2. Lo parte en bloques de 1 KB.
3. Consulta al **Namenode** dónde guardar cada bloque.
4. Recibe la lista de **Datanodes** asignados.
5. Envía cada bloque al **Datanode** correspondiente por socket.
6. Informa al **Namenode** que la carga finalizó correctamente.

> get - Descargar un archivo del DFS

El **Cliente**

1. Solicita al **Namenode** el archivo deseado.
2. Recibe del **Namenode** la lista de bloques y sus ubicaciones.
3. Se conecta a cada **Datanode** para pedir los bloques.
4. Recibe los datos y reconstruye el archivo en orden.
5. Guarda el archivo completo en el sistema local.

> info - Consultar información de un archivo

El **Cliente**

1. Envía al **Namenode** el comando INFO <nombre_archivo>.
2. El **Namenode** busca la metadata correspondiente.
3. Devuelve la lista de bloques y **Datanodes** que lo almacenan.

> ls - Listar archivos disponibles en el DFS

El **Cliente**

1. Envía al **Namenode** el comando ls
2. El **Namenode** consulta su tabla de metadata
3. Devuelve la lista de archivos registrados en el sistema

Consideraciones

- Subir los archivos al servidor de desarrollo en **`Programs/Go/Labo4`**
- Logs por todos lados.
  - Guardar en un archivo .txt cada paso realizado.
  - Mostrar por pantalla todo lo que pasa, todo es todo.
- Suma de puntos (eliminación de grises).
  - Mejoras:
    - Replicación automática (tolerancia a fallos).
    - Balanceo de carga (elegir **`datanodes`** con menos bloques).
    - Comandos adicionales (**`rm`**, **`stat`**, **`cat`** remoto).
    - Interfaz **`web`** o **`CLI`** amigable.
    - Autenticación básica entre **`nodos`**.

Nuevas consideraciones sobre el laboratorio 4 🔊 dadas el 17/11:

- 🧠 Al momento de presentar el laboratorio **debe entender lo que está haciendo**. Durante la corrección se realizarán preguntas sobre el código y sobre el funcionamiento general del sistema. En la corrección preliminar del viernes se observó que algunos alumnos no comprendían su propio trabajo; **en esos casos, el laboratorio será desaprobado**.
- 💾 Se modificó el **formato de corrección**: ahora será **individual** y se realizará **en varias máquinas del laboratorio**. Cada estudiante deberá desplegar y ejecutar su sistema en **al menos cuatro computadoras**, utilizando las **direcciones IP** que se proporcionarán en el momento. Asegúrese de que su solución sea portable y reproducible.
- 🖥️ Las computadoras del laboratorio cuentan con **Fedora 43 y Go 1.25.4**. Verifique compatibilidades antes de asistir a la corrección.
- ✨ Si su desarrollo supera el sistema básico, incorporando "**mejoras considerables**" que realmente llamen la atención del corrector, podrán sumarse puntos adicionales. Esta constituye la **tercera instancia** en la que se pueden sumar puntos **(Si te quejás, es de lleno, daale.)**.
- ⏰ Dado este nuevo formato, las correcciones demandarán más tiempo. A partir de esta semana, podrá comenzar la corrección en **horario de consulta** o coordinar otro horario adicional. También podrán realizarse correcciones **por la mañana** en los laboratorios de Alem, previo acuerdo.

- 🗓️ El **martes 25/11** se dará por finalizado el período de correcciones. De manera excepcional, alguna presentación podrá reprogramarse para el **viernes 28/11**. Organice su entrega con anticipación; recuerde, como dice el dicho: **su falta de planificación no es mi emergencia**.

Observaciones y entrega

- El código debe estar debidamente indentado y comentado, esto suma mucho.
- Se deben respetar todos los nombres de archivo solicitados.
- Se deben utilizar solo comandos **`Linux`** en la consola **`Linux`**. Debe funcionar en el servidor de desarrollo **`Fedora Linux 43`**.
- No se debe copiar y pegar el código de internet. Tener en cuenta que **`cada línea del código entregado puede ser evaluada`**.
- El laboratorio no debe tener errores de compilación; se debe compilar con el parámetro **`“-Wall”`** y solucionar cualquier ***`warning`*** o ***`error`*** que aparezca.
- Si utiliza comandos particulares para resolver los ejercicios, éstos deben quedar por escrito en un archivo denominado **`“resolución_Labo4.txt”`** en el directorio **`”Laboratorio_4”`**, así como cualquier otra consideración que se haya tomado o respuesta que tenga que dar.
- El laboratorio no tiene entrega de archivos; una vez finalizado se debe informar a la cátedra por mail (soyd-2025@cs.uns.edu.ar), la cual se encargará de corregirlo directamente en su directorio **`”home”`**. El asunto del correo debe ser: **`"SOYD: Entrega laboratorio 4"`**. En el cuerpo del mensaje, incluir **`nombre completo y número de libreta universitaria (L.U.)`**.