Entrega #1 GRIDFS

Lorena Goez Ruiz

Samuel Valencia Loaiza

Universidad EAFIT

2025

Objetivo y Marco Teórico

El objetivo de este proyecto es diseñar e implementar un sistema de archivos distribuido inspirado en HDFS, denominado GridFS. Este sistema permite almacenar archivos de gran tamaño de forma distribuida en múltiples nodos, facilitando la tolerancia a fallos y la escalabilidad.

Descripción del servicio y problema abordado

El problema abordado es el almacenamiento y recuperación de archivos grandes de manera eficiente y tolerante a fallos. Un solo servidor resulta insuficiente cuando los volúmenes de datos crecen exponencialmente. GridFS ofrece un servicio donde los archivos se dividen en bloques de 4MB y se distribuyen entre varios DataNodes. El NameNode actúa como coordinador central, manteniendo los metadatos y gestionando la autenticación de usuarios.

Arquitectura del sistema

La arquitectura de GridFS se compone de tres elementos principales:

- Cliente: Interfaz de usuario que permite subir, descargar y gestionar archivos.
- NameNode: Nodo maestro encargado de la autenticación, registro de usuarios, y gestión de metadatos.
- DataNodes: Nodos de almacenamiento que contienen los bloques físicos de los archivos.

El sistema está desplegado en contenedores Docker orquestados mediante Docker Compose, con una red virtual interna y volúmenes persistentes para garantizar la durabilidad de los datos.

Especificación de protocolos y APIs

Los servicios expuestos están implementados con FastAPI y utilizan HTTP como protocolo de comunicación. Los principales endpoints son:

NameNode:

- POST /register: Registrar un nuevo usuario.
- POST /login: Autenticar usuario y emitir token.
- POST /register_file: Registrar metadatos de un archivo.
- GET /ls: Listar archivos registrados.
- GET /file/{filename}: Obtener información de un archivo.
- DELETE /rm/{filename}: Eliminar un archivo.
- GET /file_health/{filename}: Verificar integridad de bloques.
- GET /system_status: Estado del sistema.

DataNode:

- POST /upload_block/{block_id}: Subir un bloque.
- GET /block/{block_id}: Descargar un bloque.
- GET /storage_info: Información de bloques almacenados.
- POST /heartbeat: Mantener registro de disponibilidad del DataNode.

Algoritmos de particionamiento y distribución

Los archivos son divididos en bloques de tamaño fijo de 4MB. El cliente se encarga de fragmentar los archivos y distribuirlos en los DataNodes usando un algoritmo round-robin. De esta forma se balancea la carga entre los diferentes nodos disponibles.

Entorno de ejecución nativo o en Docker

El sistema está implementado en Python y desplegado mediante Docker Compose. Cada servicio (NameNode y DataNodes) se ejecuta en un contenedor independiente conectado a la red interna 'gridnet'. Se emplean volúmenes Docker persistentes para el almacenamiento:

- namenode_data: Almacena metadatos y usuarios.
- datanodeX_data: Almacena bloques físicos de los archivos.

El archivo docker-compose.yml define los servicios, redes, volúmenes y políticas de reinicio.

Pruebas y Análisis de Resultados

Se realizaron diversas pruebas para validar el funcionamiento del sistema:

- Prueba de conectividad: El comando 'ping' desde el cliente confirma la conexión con el NameNode y la disponibilidad de los 4 DataNodes.
- Subida de archivo pequeño: Se creó el archivo 'test.txt' (30 bytes), almacenado en un único bloque.
- Subida de archivo grande: Se creó el archivo 'bigfile100.txt' de 100MB, dividido en 25 bloques distribuidos entre los 4 DataNodes. La distribución se realizó exitosamente.

- Descarga y verificación: El archivo descargado fue idéntico al original, confirmando la correcta reconstrucción.
- Manejo de fallos: Al detener un DataNode, el sistema detecta la pérdida de bloques mediante el comando 'health'.

Estos resultados demuestran que GridFS cumple con los objetivos básicos de un sistema de archivos distribuido: particionamiento, distribución y tolerancia a fallos detectados.

Conclusiones

El proyecto GridFS implementa un sistema de archivos distribuido funcional con autenticación, almacenamiento distribuido en bloques y tolerancia a fallos mediante detección de nodos caídos. El despliegue en Docker facilita la portabilidad y la reproducibilidad del entorno.