Client Server Distributed Virtual File System (CSDFS)

Russo Antonio

2025-09-22

Introduzione

Il **Distributed Virtual File System (DVFS)** è un progetto che implementa un file system distribuito secondo un modello **client-server**. L'idea di base è permettere a più client di accedere a un file system remoto come se fosse locale, con un'interfaccia semplice e coerente. Il sistema è sviluppato in **Java** ed utilizza **RMI (Remote Method Invocation)** come meccanismo di comunicazione, garantendo trasparenza delle invocazioni e modularità.

Il DVFS offre funzionalità classiche di un file system (creazione, lettura, scrittura, navigazione) e introduce una politica di **write-through**, che assicura che ogni modifica in memoria venga immediatamente riflessa anche sul file system reale montato sul server.

Architettura

L'architettura segue il modello **client-server centralizzato**:

- **FileSystem**: cuore del sistema, un file system virtuale in memoria strutturato come un albero. Ogni nodo può rappresentare directory, file o symlink. Le operazioni in memoria vengono sincronizzate su disco tramite write-through.
- **RemoteFileSystem**: oggetto RMI che funge da "ponte" tra i client e il VFS locale. Implementa l'interfaccia remota e inoltra le richieste al FileSystem.
- **FileSystemServer**: avvia e monta il VFS da una directory reale, pubblica lo stub RMI e resta in ascolto delle richieste.
- **FileSystemClient**: applicazione a riga di comando che permette di interagire col file system remoto. Supporta comandi familiari (**mkdir**, **ls**, **read**, **write**) e funzionalità avanzate come **edit**, che scarica un file remoto in un editor locale e lo risincronizza al termine della modifica.

Questa separazione isola le responsabilità: i client gestiscono l'interazione con l'utente, mentre il server centralizza la logica del file system e garantisce consistenza tra più richieste concorrenti.

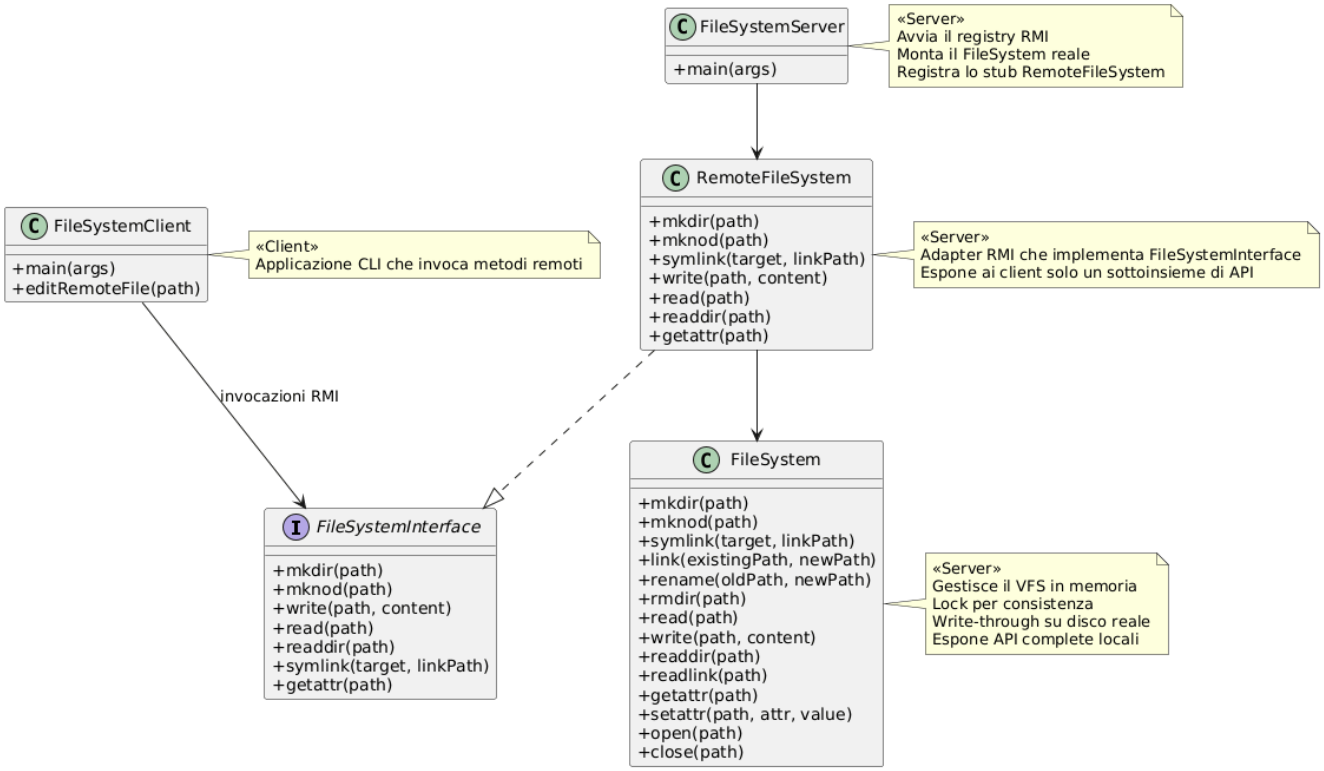

Figure 1: Architettura client-server DVFS

Consistenza

La consistenza è garantita dal **file system**. Ogni operazione che modifica lo stato (scrittura, rinomina, rimozione) viene protetta da lock a livello di path (**ReentrantReadWriteLock**).

- Più client possono leggere contemporaneamente senza conflitti.
- Le scritture sono serializzate, impedendo **race condition**.
- Ogni modifica avviene in due fasi: aggiornamento in memoria e write-through su disco.

I client non gestiscono lock: tutta la concorrenza viene risolta dal server, che possiede l'unica copia "autorevole" dello stato.

Montaggio da directory reale

Il sistema può partire da zero o essere montato da una directory esistente. In questo caso, il contenuto viene caricato ricorsivamente:

- Directory → DirectoryNode.
- File → FileNode (contenuto letto in memoria).
- Symlink → SymlinkNode (target salvato).

La root del VFS viene rinominata "/", e ogni operazione successiva (scrittura, rinomina, rimozione) viene riflessa anche sulla directory reale tramite write-through.

Funzionalità

Il DVFS mette a disposizione un set completo di operazioni:

- **Creazione**: mkdir, mknod, symlink, link.
- **Navigazione**: lookup, readdir, readlink.

- **Manipolazione**: read, write, rename, rmdir.
- **Gestione attributi**: getattr, setattr.
- **Gestione apertura/chiusura**: open, close.

Inoltre, lato client è disponibile il comando **edit**, che consente di modificare un file remoto con un editor locale in maniera trasparente.

Protocolli

La comunicazione tra client e server avviene tramite **Java RMI**. Le invocazioni remote sono trasparenti: il client invoca metodi sull'interfaccia **FileSystemInterface**, che vengono eseguiti dal server sul VFS locale.

Flusso tipico di un'operazione

1. Il client invia una richiesta remota (es. `write("/foo", data)`).
2. Lo stub RMI inoltra la chiamata a RemoteFileSystem sul server.
3. RemoteFileSystem chiama il metodo corrispondente di FileSystem.
4. FileSystem acquisisce il lock, aggiorna lo stato in memoria e riflette la modifica su disco.
5. Il risultato viene restituito al client.

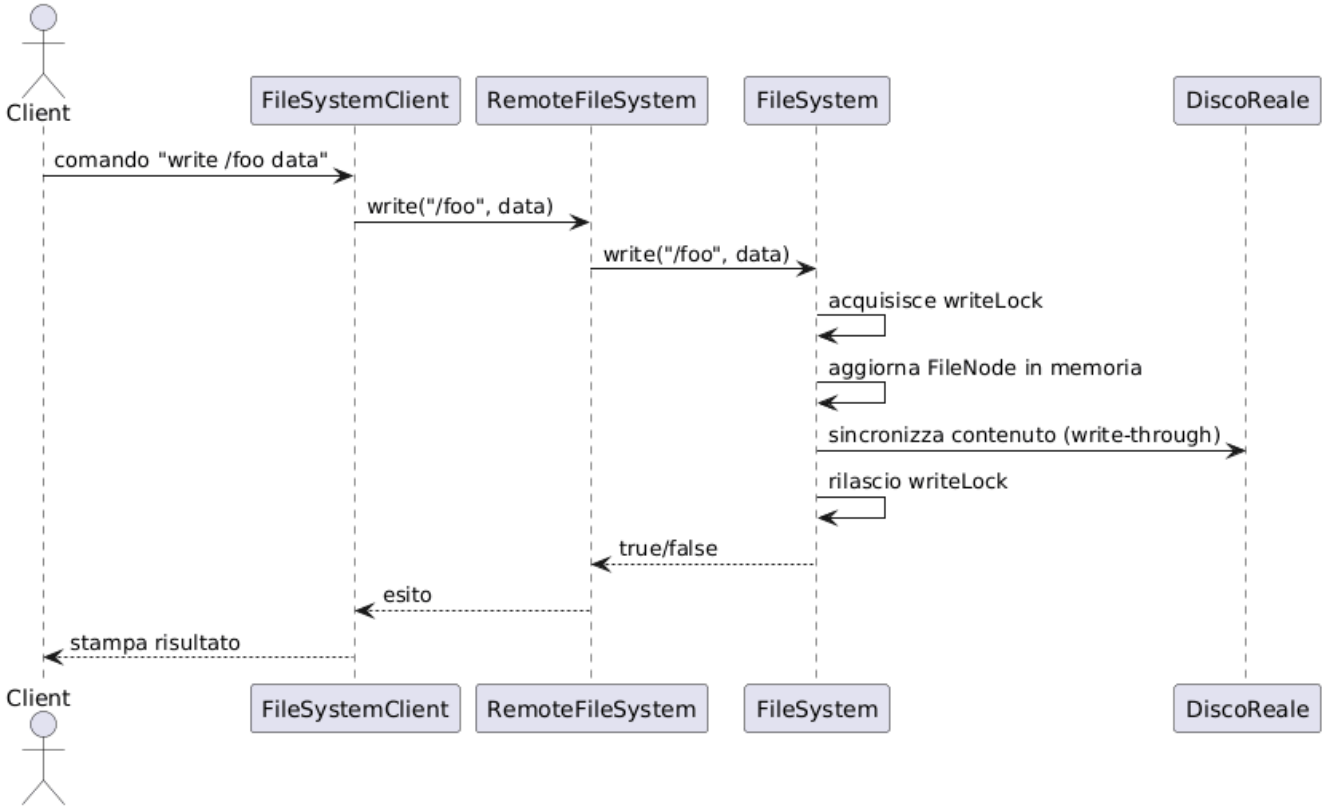

Figure 2: Flusso di una richiesta write

Sicurezza ed error handling

- Durante la risoluzione dei path, il server impedisce accessi fuori dalla root montata (protezione da path traversal).
- In caso di errori I/O durante il write-through, l'operazione resta valida in memoria, evitando perdita di dati.
- Gli errori lato server vengono propagati al client come eccezioni RMI.