Algoritmi Distribuiti - prova pratica

Carlo Uguzzoni

01/2025

Indice

1 Introduzione

Il progetto consiste di un prototipo di file system distribuito (FSD), principalmente basato sull'uso di remote procedure calls (RPC).
L'architettura del sistema si concentra attorno a 3 attori fondamentali:

- Name server: costituisce il fulcro del FSD. Regola le comunicazioni tra clients e file servers, gestisce un database centralizzato contenente i dati dell'intero sistema ed avvia periodicamente task che permettono un funzionamento corretto ed efficiente del sistema
- File servers: si occupano dello storage vero e proprio dei files
- Clients: utenti del FSD. Gestiscono attivamente i propri files

Sebbene il name server costituisca un nodo centrale che rappresenta single point of failure (SPOF), il sistema è sviluppato per garantire gestione decentralizzata dei files, scalabilità, replicazione e consistenza.

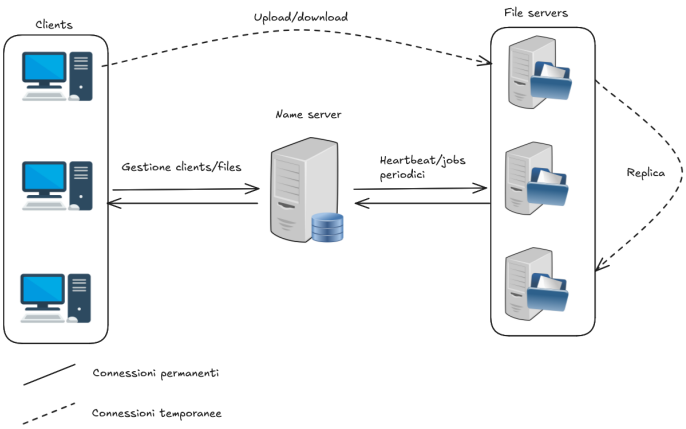

Figura 1.1: Schema architetturale del sistema

Il progetto è realizzato in Python, utilizzando la libreria `RPyC` per l'implementazione delle RPC. Il database del name server viene gestito tramite `SQLite`, utilizzando l'API di `sqlite3`.

2 Architettura

Name server Il name server è senz'altro l'entità più importante e complessa del sistema. Si tratta di un oggetto `rpyc.Service` che fornisce le RPC per rispondere alle richieste dei clients e coordinare le operazioni con i file servers. Queste RPC sono definite con il prefisso `exposed_` lato-name server e costituiscono gli unici metodi di quest'entità fruibili alle altre.
Per avviare il name server viene usato un `ThreadedServer`, che consente di gestire le connessioni simultanee di più clients al servizio. Ogni connessione viene gestita in un thread separato, permettendo il parallelismo e quindi la gestione di richieste concorrenti.
All'avvio del name server vengono anche inizializzati e lanciati tasks periodici, gestiti da un `BackgroundScheduler` di `apscheduler`, per le attività di:

- Replica dei files nel DFS: ha lo scopo di mantenere K repliche attive (ovvero in hosting da parte di file servers online) in ogni momento
- Check di consistenza dei files: consiste nell'invio ai file servers di una lista dei files loro assegnati e dei relativi checksum, affinché essi possano verificarne la correttezza
- Garbage collection: consiste nell'invio ai file servers di una lista dei files loro assegnati, affinché essi possano cancellare dal proprio storage quelli non più necessari
- Check di attività delle entità nel DFS: controlla se le entità connesse al name server hanno inviato un heartbeat, che confermasse la propria attività, entro un periodo di tempo fissato. Questo meccanismo ha lo scopo di rilevare e sistemare automaticamente disconnessioni che non seguono un protocollo corretto

Il name server è un'entità concepita per essere unica all'interno del sistema. Questa proprietà è garantita tramite l'uso del design pattern "Singleton" e della gestione di un file di lock.
Il progetto supporta un sistema di registrazione e logging sia per i clients che per i file servers. Il file server mette a disposizione tutte le RPC per la gestione delle entità:

- `exposed_create_user` / `exposed_create_file_server`
- `exposed_authenticate_user` / `exposed_authenticate_file_server`
- `exposed_delete_user`
- `exposed_logout_user` / `exposed_logout_file_server`

All'atto di login delle varie entità presso il name server, viene fornito loro un token di autenticazione, firmato con una chiave privata nota solo al name server stesso. Il payload del token contiene nome e ruolo dell'entità che esegue il login. Il suo scopo è quello di implementare un sistema di ruoli, permessi ed autenticazione a cui sottoporre chi chiama le RPC critiche per la sicurezza. Ad esempio, senza questo meccanismo un client regolare potrebbe richiamare le RPC di un amministratore, o peggio quelle che regolano i meccanismi di replica dei files. La chiave pubblica con la quale decriptare il payload è inoltrata ai file servers che eseguono il login, affinché anche questi possano sottoporre a controllo le chiamate in ingresso. Nello specifico, i token sono JSON Web Tokens (JWT), firmati con chiavi RSA da 2048 bit.
Un elemento fondamentale del name server è il database centrale, che contiene tutte le informazioni necessarie al corretto funzionamento del sistema. In esso viene conservato lo stato di clients, file servers, files e repliche.
Il file server attinge alle informazioni nel database per convalidare login/logout delle entità, portare a termine la loro creazione/cancellazione, eseguire i tasks periodici, indirizzare la comunicazione tra clients e file servers, e molto altro. Il database centrale consiste in un file SQLite, che viene gestito tramite l'API di `sqlite3`.

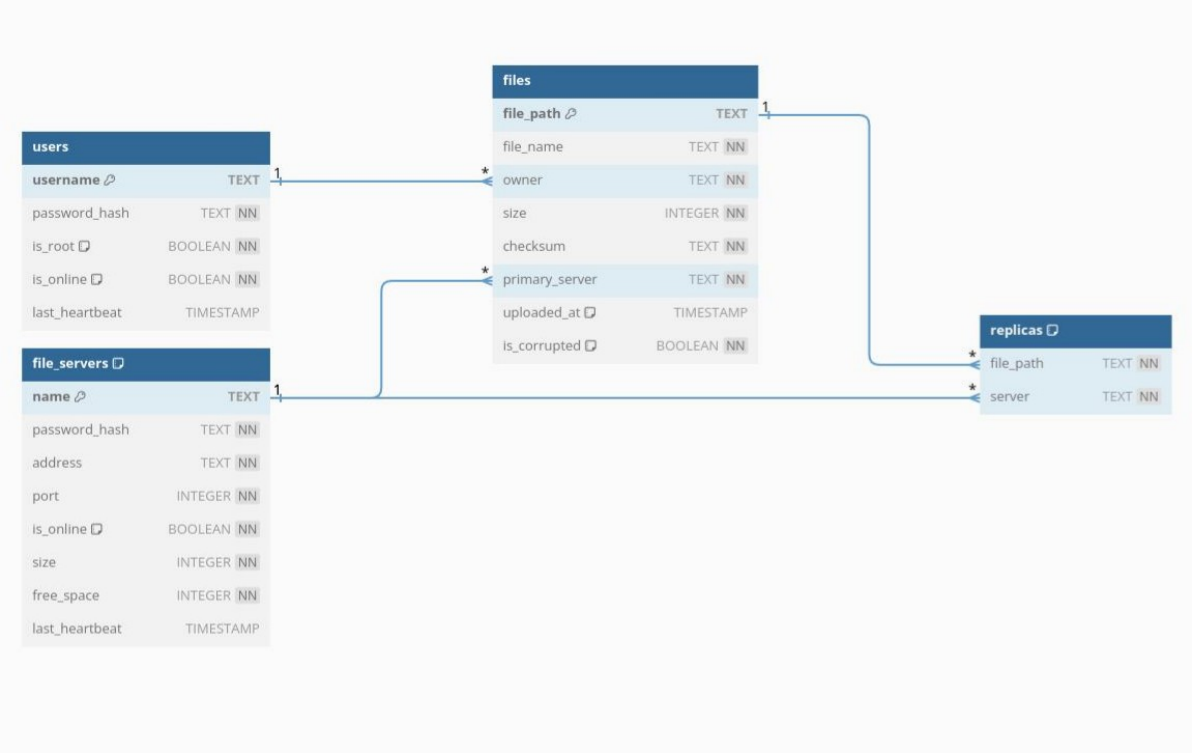

Figura 2.1: ERD del database centrale

Il name server interagisce con i clients tramite diverse RPCs:

- `exposed_get_user_files`: restituisce la lista dei files di un utente

- `exposed_get_file_server_upload`: restituisce le coordinate (`ip`, `port`) di un file server per l'upload di un file, scegliendolo con il metodo "K-least loaded". Il file server su cui viene caricato il file verrà marcato come "primary" per tale file
- `exposed_get_file_server_download`: restituisce le coordinate (`ip`, `port`) di un file server per il download di un file, dando priorità al file server marcato come "primary" per tale file
- `exposed_delete_file`: cancella un file dal sistema. La cancellazione avviene solo nel database, il task di garbage collection provvederà all'eliminazione effettiva del file dal FSD
- `exposed_get_file_server_update`: restituisce le coordinate (`ip`, `port`) di un file server per l'aggiornamento di un file. L'update di fatto consiste in una sequenza di cancellazione ed upload, per garantire immediata disponibilità del servizio più replicazione e consistenza alla prima chiamata dei tasks periodici
- `exposed_list_all_files`: esclusiva del client amministratore, restituisce la lista di tutti i files e le relative repliche presenti nel sistema
- `exposed_list_all_clients`: esclusiva del client amministratore, restituisce la lista di tutti i clients nel sistema ed il loro stato
- `exposed_list_all_file_servers`: esclusiva del client amministratore, restituisce la lista di tutti i file servers nel sistema ed il loro stato

Task periodici e altre funzioni di handling vengono approfonditi e descritti nel seguito.

File server I file servers realizzano lo storage vero e proprio nel FSD. Essi ereditano a loro volta da `rpyc.Service` e vengono lanciati da un `ThreadedServer`.
Per i file servers è prevista un'interfaccia a riga di comando che consente la registrazione di un nuovo file server od il login di un file server esistente.
Il sistema, pur essendo scalabile di default, è concepito per realizzare un FSD di piccole dimensioni. Per questo motivo non è possibile la cancellazione di un file server esistente, come avviene per i clients. La sua implementazione sarebbe in realtà un task relativamente semplice, che implicherebbe la rimozione del file server da ruolo di "primary" per tutti i files attualmente assegnati e la successiva ri-assegnazione degli stessi ad uno nuovo. Infine, si dovrebbero cancellare tutte le repliche assegnate al file server ed il suo record dalle tabelle del database.
La connessione con il name server avviene all'avvio. Devono essere forniti come parametri `host` e `port` del name server allo script di lancio.
Nonappena avviene con successo il login, il file server avvia il loop principale e diventa disponibile per le richieste. Lo spegnimento deve avvenire tramite un segnale di terminazione; viene gestita l'uscita in modo polite con `SIGINT`.
I file servers ineragiscono con i clients tramite le RPCs:

- `exposed_store_file`
- `exposed_send_file`

che ricevono/inviano rispettivamente un file. Come già indicato, i clients vengono appositamente indirizzati dal name server, prima di accedere a queste RPC.
Il salvataggio ed il reperimento dei files avvengono sullo stesso host del file server in una directory predefinita. La directory di base assegnata a ciascun client coincide con il suo `username` nel database centrale.
L'interazione tra file servers e name server è invece più complessa. Abbiamo:

- `exposed_send_file_replicas`: chiamata dal job di replicazione periodica, si occupa di inviare repliche di un file ad una lista di altri file servers
- `exposed_garbage_collection`: chiamata dal job di garbage cleaning periodico, elimina files non più presenti sul database del name server e tutte le directory vuote
- `exposed_consistency_check`: chiamata dal job di consistency check periodico, controlla la consistenza dei file presenti sul database del name server. Se riconosce un'incongruenza, chiama una speciale RPC del name server per risolverla

In ultimo, i file servers, quando attivi, inviano periodicamente un heartbeat al name server per confermare la loro attività. Lo heartbeat è schedulato come job periodico in un `BackgroundScheduler`.

Regular client I clients regolari rappresentano gli utenti standard del FSD. Le loro facoltà sono principalmente quelle di eseguire upload/download/listing di files sul FSD tramite le RPC messe a disposizione dalle entità server. La connessione con il name server avviene all'avvio.
L'interfaccia da riga di comando consente ai regular clients di eseguire liberamente login/logout e di autenticarsi con diversi profili. I files sono salvati sullo stesso host dal quale i clients si autenticano, in una directory predefinita. La directory di base di ciascun utente coincide con il suo `username` nel database centrale.
All'upload di un file è richiesto al client di specificare il percorso assoluto del file da caricare e la directory di destinazione all'interno del FSD. In questo modo è possibile caricare nel FSD un file da qualsiasi directory locale.
Nel download si richiede il solo percorso del file nel FSD. Il file viene scaricato nella base directory locale del client autenticato, assieme a tutta la struttura di directories da cui discende nel FSD.

Root clients Il client root dispone di tutte le funzionalità garantite ai clients di regolari (cioè quelle di base), infatti entrambi ereditano dalla classe astratta `BaseClient`. Inoltre, può richiamare RPC esclusive per visualizzare lo stato di tutte le entità ed i files nel FSD. Queste funzionalità assolvono principalmente alla funzione di amministrazione e

manutenzione del FSD.
Per ora al client root non sono garantiti accesso, cancellazione ed upload di files dei clients regolari, in quanto queste azioni sono critiche dal punto di vista della privacy degli utenti. Si potrebbero implementare in futuro, per esempio a patto che il loro uso venga rigorosamente registrato da un apposito sistema di logging.
Il client root è pensato per essere unico nel FSD, che come abbiamo detto è concepito per avere modeste dimensioni. Come per il name server, si è adottato il design pattern Singleton, con annesso meccanismo basato su file di lock. È dunque possibile effettuare una sola volta la registrazione per questo ruolo ed il login è statico.
Come i file servers, tutti i clients (anche quelli regolari) inviano periodicamente un heartbeat al name server per confermare la loro attività.

3 Interazioni

Upload e download Queste sono senz'altro le due interazioni più comuni nell'uso del FSD che coinvolgono direttamente ed attivamente i clients e le entità server.
Nel primo caso, il client chiama il metodo RPC `exposed_get_file_server_upload` del name server. Quest'ultimo interroga il database centrale e restituisce le coordinate di un file server online in grado di ricevere il file. Come abbiamo detto, tale file server viene marcato come "primary". Ciò significa che sarà il primo a ricevere il file, il punto di riferimento per le repliche e la prima scelta in caso di download. A questo punto, il client si connette direttamente al file server indicato, richiamando la RPC `exposed_store_file`, che provvede a trasferire il file generando, se necessario, tutta la struttura di directories. La procedura di download segue una dinamica del tutto analoga. Il client si rivolge al name server con la RPC `exposed_get_file_server_download`, che fornisce le coordinate del primary server del file specificato, o quelle di un altro file server se questo non è online. Il client si connette al file server, richiamando la RPC `exposed_send_file` per ottenere il file.

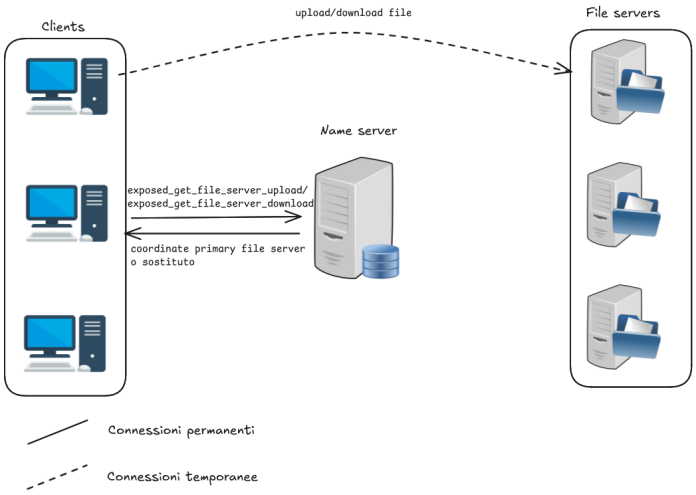

Figura 3.1: Interazioni clients/servers per upload/download

Tasks periodici

Replicazione La replicazione riguarda tutti quei files che hanno meno di K repliche attive, ovvero presenti su meno di K file servers online. Questi vengono selezionati dal name server, assieme alle coordinate (ovvero la coppia `host`, `port`) del relativo primary

server. Il name server chiama quindi la RPC `exposed_send_file_replicas`, allegando le coordinate di tutti i file servers online che non ospitano già una replica del file, per raggiungere un massimo di K in totale.
A questo punto, il primary file server, iterativamente, invia il file ai nuovi file servers tramite la RPC `exposed_store_file`.
Una volta restituita al chiamante la prima RPC, vengono create le entry nella tabella `replicas` del database centrale. Repliche non andate a buon fine vengono successivamente eliminate dai tasks di consistency check e garbage cleaning.

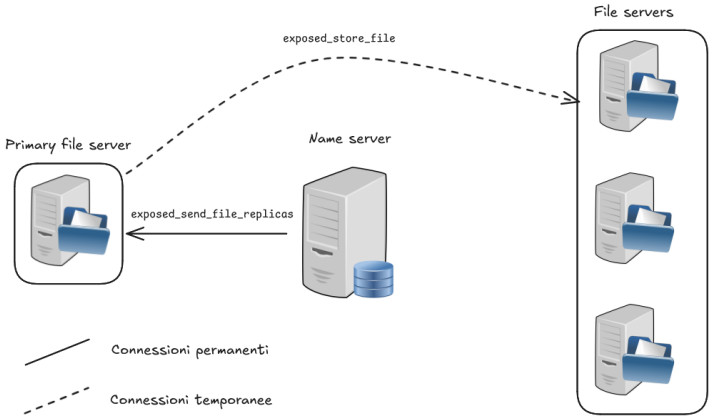

Figura 3.2: Interazioni name server/file servers per replicazione

Check di consistenza Il name server invia a tutti i file servers i nomi dei files in essi salvati ed i relativi checksums, così che questi possano controllarne l'integrità. Questo processo avviene tramite la chiamata della RPC `exposed_consistency_check`.
Nel caso il checksum calcolato sul file nello storage del file server non corrisponda a quello fornito dal name server, oppure il file non venga trovato, il file server chiama una apposita RPC del name server per gestire l'inconsistenza operando sul database centrale. Questa RPC è `exposed_handle_inconsistency`.
Nel caso di inconsistenza, il name server cancella la replica sul file server chiamate, così che questa venga rimossa dal successivo garbage cleaning. Dopodiché, se si trattava di un file server primario ne assegna uno nuovo al file, dando priorità a quelli online che ne ospitano una copia. In caso contrario, la procedura termina.
Se non viene trovato alcun file server contenente una replica per il file, esso viene marcato come `corrupted`. I files corrotti non possono più essere reperiti dai clients, ma solo cancellati. I clients possono conoscere lo stato di ogni file di loro proprietà tramite la lista dei propri files sul FSD.

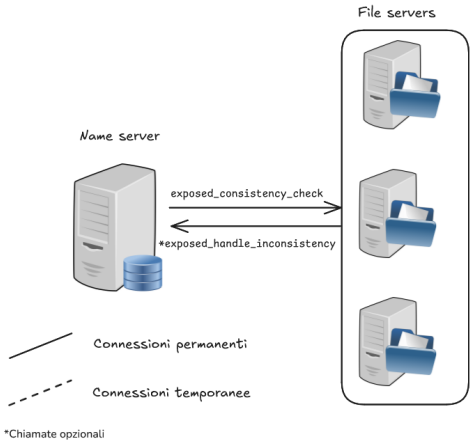

Figura 3.3: Interazioni name server/file servers per check di consistenza

Garbage cleaning Il name server invia a tutti i file servers i nomi dei files in essi salvati. Questi nomi consistono in realtà del campo `file_path` della tabella `files`, ovvero del path assoluto dei files nel FSD.
I file servers rimuovono tutti i files nel proprio storage che non compaiono nella lista ricevuta dal name server, più le directories rimaste vuote a seguito della rimozione.

Eventual consistency Il concetto di eventual consistency è stato formalmente definito da Werner Vogels come segue: *"Se non vengono eseguite nuove operazioni di aggiornamento, dopo un tempo sufficientemente lungo tutte le repliche del sistema convergeranno allo stesso stato"*.
Nel FSD realizzato, il name server, attraverso l'esecuzione dei tasks periodici descritti sopra, garantisce questa proprietà.

- La replica fornisce ridondanza e disponibilità
- Il consistency check realizza sincronizzazione tra i files, prendendo come riferimento privilegiato il file server primario
- Il garbage cleaning rimuove files incongruenti per evitare incoerenza tra le repliche

In questo modo, anche in presenza di fallimenti temporanei o partizioni di rete, tutte le repliche alla fine rifletteranno uno stato coerente con la visione del name server.

4 Conclusioni

In questo progetto è stato realizzato un FSD con un'architettura fortemente centralizzata, in cui il name server gestisce la mappatura dei file, coordina le interazioni tra clients e file servers, e svolge varie operazioni per garantire consistenza.
Tra i propri punti di forza il sistema offre:

- Gestione centralizzata: riduce la complessità delle classi che non sono il name server e la logica delle interazioni tra loro
- Un sistema di tasks periodici, i quali:
  - Delegano per design quanto più lavoro possibile ai file servers, riducendo il carico sul name server e cercando di dare scalabilità al sistema
  - Garantiscono eventual consistency, oltre che tolleranza ai guasti e disponibilità

Tuttavia, sono presenti anche diverse limitazioni:

- SPOF: il funzionamento del name server è essenziale per quello dell'intero sistema, dunque una possibile rottura potrebbe rendere il tutto inutilizzabile
- Scalabilità limitata: un'architettura fortemente centralizzata come quella proposta, con l'aggiunta di un alto numero di nodi potrebbe risentire pesantemente del carico di lavoro sul name server. Sebbene si sia cercato di scaricare il name server da ogni lavoro superfluo e comunque il sistema sia concepito per avere dimensioni limitate, questa problematica rimane una possibilità concreta

Una soluzione possibile a queste problematiche, perlopiù architetturali, potrebbe essere quella rifattorizzare l'intero progetto introducendo una nuova architettura completamente distribuita, ad esempio basata su Distributed Hash Tables (DHT). Anche il modello di consistenza potrebbe essere migliorato, aderendo a standard più potenti, pur incrementando significativamente la complessità del progetto.
In ultimo, se invece si pensasse che l'architettura attuale sia sufficiente per servire allo scopo, una modifica applicabile per migliorare le performances potrebbe essere la parallelizzazione del lancio dei tasks periodici, che nella versione base è sequenziale. Realizzare questa feature per l'hardware sul quale è stato realizzato il progetto non avrebbe avuto molto senso, anche perché i file servers attivi erano tipicamente al massimo 3 (sempre per la natura modesta del sistema).