Tesina Distributed FS (main)

alessiocorrado99

September 2020

1 Introduzione

Lo scopo del progetto è di realizzare un file system distributito. Le caratteristiche che deve avere sono: efficienza, accesso concorrente, replicazione dei dati (files), replicazione dei metadati (struttura del fs).

Il numero di repliche di un file è proporzionale all'importanza di esso: file con dati critici hanno numero di copie maggiore, per garantire maggiore tolleranza ai guasti e tempo di accesso minore.

Viene effettuato anche un bilanciamento del carico: i file vengono copiati o spostati in modo da non sovraccaricare un singolo server.

2 Architettura del sistema

Verranno analizzate diverse possibili architetture del sistema.

2.1 Single-server

La prima idea è quella di avere un singolo server, il quale memorizza sia il fs che i dati. Il client interagisce direttamente con esso sia per l'esecuzione di comandi sul fs che per la memorizzazione dei dati.

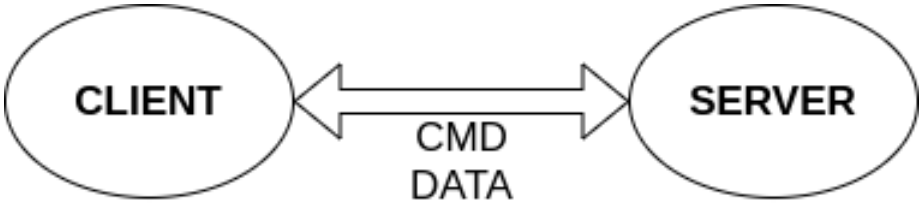
Il fs viene memorizzato direttamente a partire da una directory del server, oppure in una partizione separata. In questo modo il sistema operativo si occupa direttamente della sua gestione.

La replicazione dei dati avviene mediante il salvataggio in diversi dispositivi di memorizzazione (es HDD). Per far ciò la scelta più consona è di utilizzare un sistema RAID 10: i dati vengono salvati in copia (RAID 1) per avere tolleranza ai guasti e in striping (RAID 0) per aumentare la velocità di lettura.

Con schede di rete (e connessioni) multiple è possibile aumentare la tolleranza sia ai guasti che alle congestioni di rete.

Vantaggi: semplice da implementare. economico. il fs è gestito direttamente dal sistema operativo. buona sicurezza contro attacchi hacker. con gli accorgimenti sopra descritti si ha già una buona robustezza.

Svantaggi: single point of failure per quanto riguarda il server (i dati sono parzialmente protetti dal meccanismo RAID). nessuna protezione per eventi eccezionali (eg. catastrofe naturale, incendio nell'edificio, blocco della rete...).



2.2 Virtual server

La soluzione single-server può essere migliorata aggiungendo un livello di virtualizzazione. Il sistema operativo non viene quindi eseguito direttamente sull'hardware ma all'interno di un container/VM.

La macchina virtuale viene periodicamente copiata (interamente o in modo incrementale) e salvata in un dispositivo di memorizzazione dedicato.

Nel caso ci fosse un crash o errore critico del sistema operativo, basta ricaricare la macchina virtuale più recente. La perdita di dati è limitata all'età dell'ultimo backup.

Vantaggi: recovery buona e in tempi brevi in caso di crash.

Svantaggi: durante il periodo di transizione il sistema non risponde. prestazioni leggermente ridotte a causa della virtualizzazione.

2.3 Cloud backup

Il backup può essere effettuato nel cloud, al posto che in un dispositivo dedicato all'interno del datacenter. Inoltre il backup può essere esteso anche ai dati, oltre che al fs.

Vantaggi: protezione in caso di eventi eccezionali.

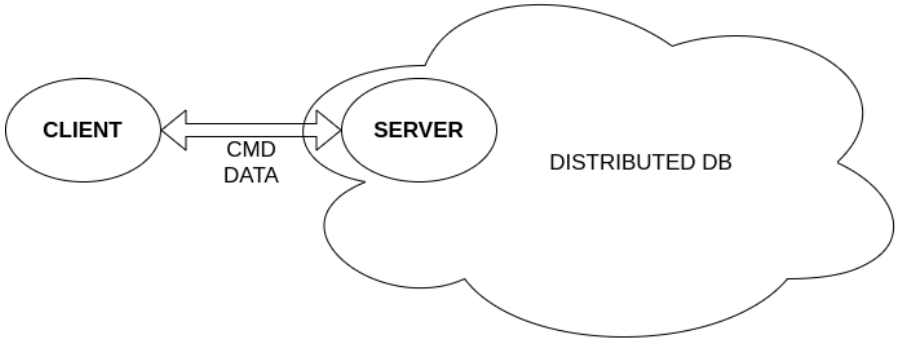
Svantaggi: in generale effettuare un backup nel cloud, anche se solo incrementale, richiede un elevato utilizzo di banda e tempi maggiori. costi maggiori.

2.4 Server as distributed database

Al posto di utilizzare il file system del sistema operativo, è possibile utilizzare un database distribuito. Sia metadati che dati vengono salvati automaticamente nei nodi, garantendo consistenza e fault tolerance.

Vantaggi: è possibile utilizzare un dbms in commercio, beneficiando dall'avere tutto già pronto e costantemente aggiornato.

Svantaggi: la connessione avviene tra il client ed un nodo, il quale poi distribuisce i dati agli altri nodi. se questo nodo dispone di una banda limitata, ciò può rappresentare un bottleneck dal punto di vista delle prestazioni.



2.5 Meta+Data server

L'idea è di far comunicare direttamente il client con i server in cui verranno memorizzati i dati.

Un primo sviluppo è quello di separare i compiti: si hanno quindi un "meta server" e uno o più "data server".

Il meta server si occupa di memorizzare e gestire il file system (metadati), mentre i data server memorizzano soltanto i dati conenuti nei file.

Per effettuare una qualsiasi operazione su un file il client si rivolge inizialmente al meta server, dal quale ottiene la configurazione per contattare il giusto data server e trasferire i dati.

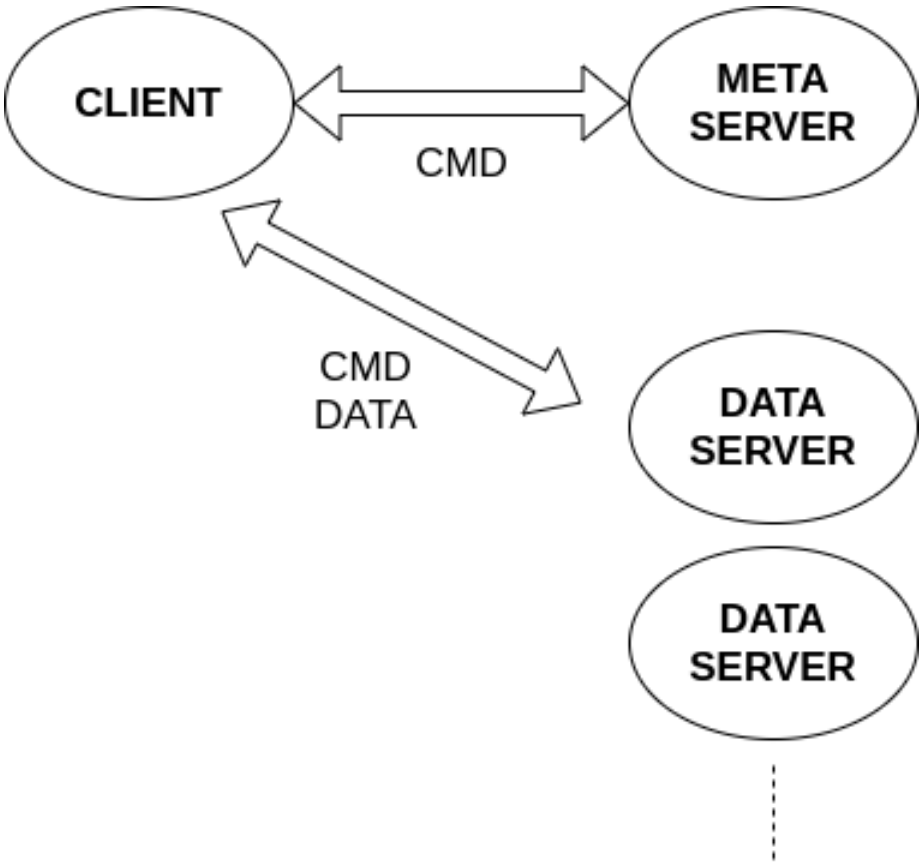
In questo modo ci sono due comunicazioni bidirezionali: client-meta, per la gestione del fs; client-data, per il trasferimento dei dati.

Per migliorare le prestazioni, soprattutto nel caso in cui il client abbia una banda più ampia rispetto ai data server, i file di grandi dimensioni vengono spezzati in chunks e distribuiti.

Come prima il meta server è unico, virtualizzato e sottoposto a backup periodico.

Vantaggi: connessioni a data server multipli permettono di migliorare le prestazioni nel caso in cui il client abbia una banda elevata. un unico meta server permette di mantenere in modo semplice la consistenza del file system.

Svantaggi: la frequenza dei backup del meta server è fondamentale per minimizzare la perdita di dati in caso di fault. tutti i data server devono essere raggiungibili dal client (maggiore attenzione alla sicurezza e struttura della rete).



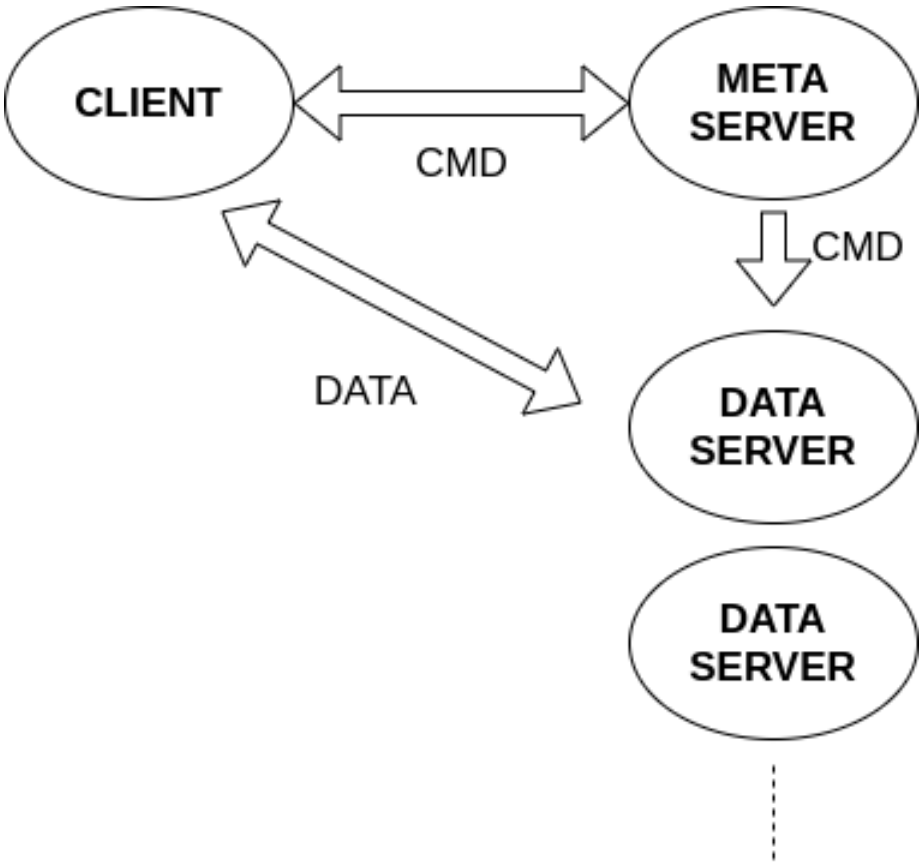
2.6 Meta+Data server, direct config

Facendo dialogare meta e data server, è possibile: spostare l'onere di configurazione dei data server da client a meta server; gestire in qualsiasi momento la replicazione dei dati, per bilanciare il carico o far fronte alla perdita di un nodo;

Il client quindi si limita a trasferire i dati da/verso data server utilizzando token forniti dal meta server.

Vantaggi: il sistema può riconfigurarsi in ogni momento per far fronte a qualsiasi evenienza. ruolo del client semplificato. maggiore sicurezza perchè il client non può inviare comandi ai data server.

Svantaggi: maggiore complessità nel gestire operazioni concorrenti e asincrone.



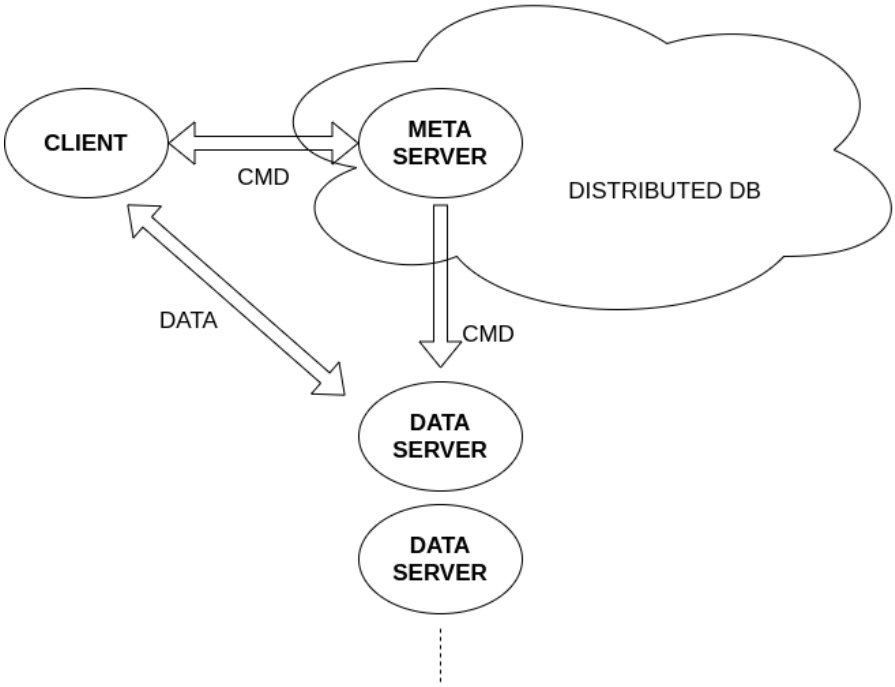
2.7 Meta server as distributed system

In questa soluzione non è presente un unico meta server, ma viene utilizzato un sistema distribuito. Approcci possibili sono utilizzare database distribuiti, blockchain o implementare a basso livello meccanismi di elezione e mantenimento della consistenza.

Come dimostrato dal teorema CAP bisogna rinunciare alla disponibilità per garantire la coerenza del fs.

Vantaggi: possibilità di utilizzo di software in commercio. maggiore consistenza e fault tolerance dei metadati.

Svantaggi: maggiore difficoltà nell'implementazione dei meta server e dei protocolli di comunicazione tra i vari componenti.



3 Comandi

3.1 Comandi del client

list <path?> <-r> <-meta> <-format=json, csv, default>: lista il contenuto di una cartella. output per linea dir=path assoluto. se assente root. -r=ricorsivo -meta=includi metadati -format=formato output

mkdir <path> crea una cartella
meta <path> metadati di un path
get <path> download di un file
push <path> upload di un file
rem <path> delete di un path
link <file path> <link path> crea link

3.2 Metadati dei path

I metadati associati ad un path sono:

uid = usato come identifcatore univoco anche per versioni dello stesso path
name
creation date
size
owner = chi l'ha creato è owner

visible = visibile solo da owner o pubblico
ncopies = numero copie minimo desiderato

4 Protocollo comunicazione

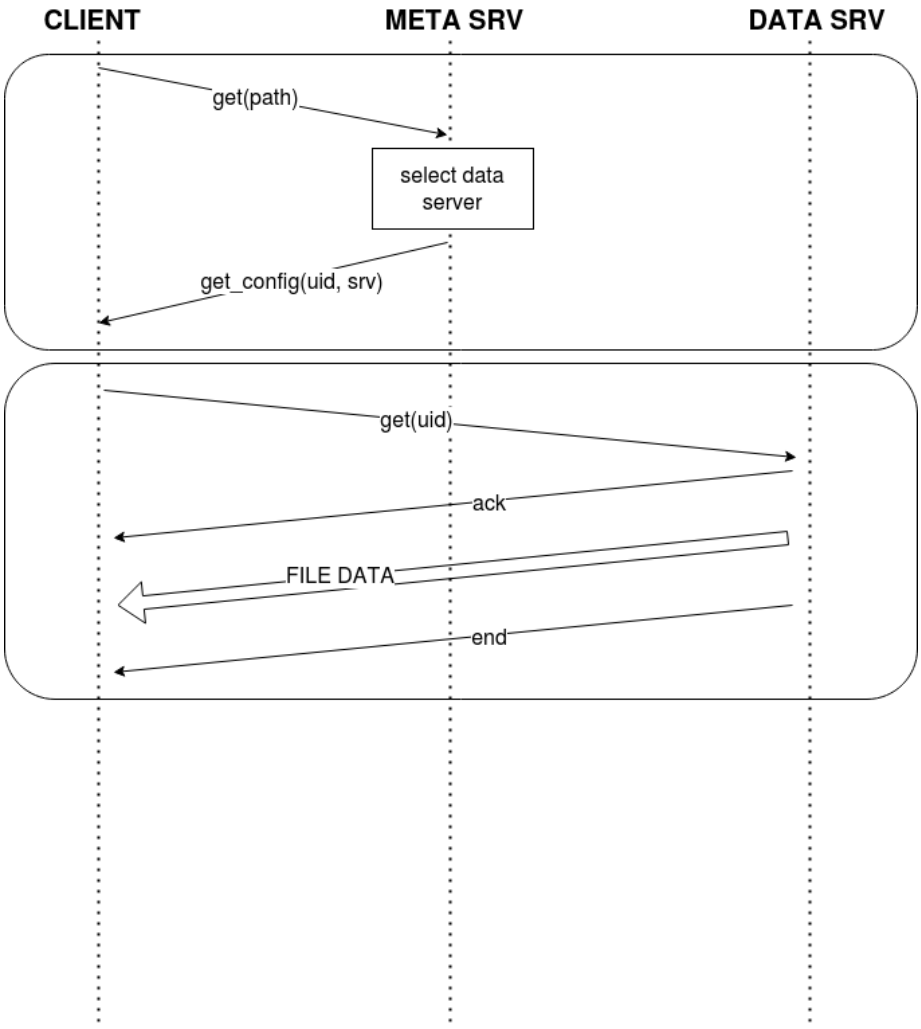
Di seguito i flussi di dati coinvolti nelle diverse tipologie di richieste.

4.1 get

La richiesta di tipo *get(path)* richiede il trasferimento in entrata di un file.

Inizialmente il *client* contatta il *meta server*, richiedendo un particolare *path*. Il *meta server* seleziona il *data server* ottimale per gestire la richiesta (deve contenere la risorsa e non essere sovraccarico). Il *meta server* risponde quindi al client specificando l'uid della risorsa (richiesto per identificarla) e l'indirizzo del *data server* a cui collegarsi. Il *client* si disconnette dal *meta server*.

Il *client* si connette al *data server* inviando una richiesta *get(uid)*. Se la risorsa è disponibile (come dovrebbe essere) il *data server* invia un *ack* seguito dal flusso di dati del file. (TODO implementare lastpos) Altrimenti risponde con *err* seguito dai dettagli.



4.2 push

La richiesta di tipo *push(path, data)* richiede il trasferimento in uscita di un file da parte del client.

Inizialmente il *client* contatta il *meta server*, inviando *path* e *size*. Il *meta server* seleziona il *data server* ottimale per gestire la richiesta (deve poter contenere la risorsa e non essere sovraccarico). Il *meta server* si occupa anche di generare un nuovo *uid* da assegnare alla risorsa.

Il *meta server* si collega al *data server* dove verrà inizialmente salvata la risorsa, inviando un comando *create(uid, size)*. Questo comando predispone il *data server* per ospitare un nuovo documento con tale uid. Se non ci sono errori, il *data server* risponde al *meta server* con *ack*. Altrimenti con *err* seguito dai dettagli.

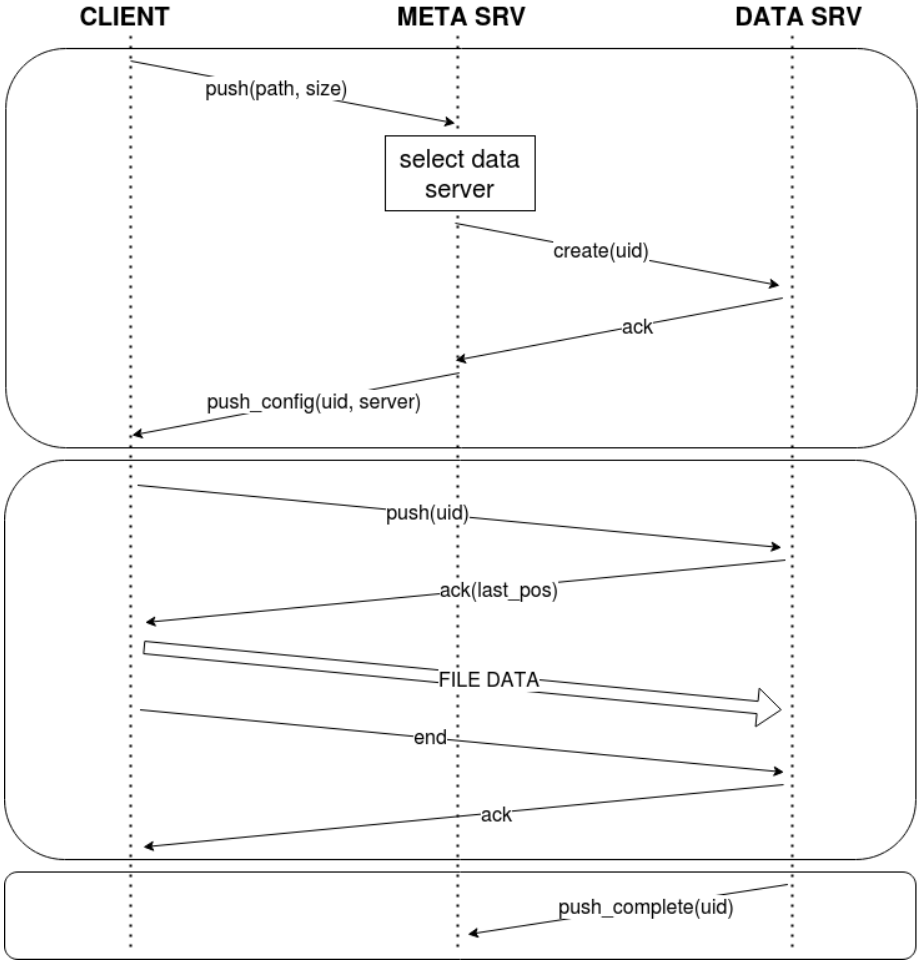
(TODO aggiungi size a create) (TODO aggiungi richieste eliminazione)

Il *meta server* risponde quindi al client specificando l'uid della risorsa (richiesto per identificarla) e l'indirizzo del *data server* a cui collegarsi. Il *client* si disconnette dal *meta server*.

Inizia quindi il caricamento del documento. Il *client* si collega al *data server* e invia una *push(uid)*. Se non ci sono errori, il *data server* risponde con *ack* e invia la posizione (indice 0-based) *last_pos* del primo byte non trasferito (equivalente al numero di byte già trasferiti). A quel punto il *client* trasferisce la porzione rimanente di documento. Al termine del trasferimento, se non ci sono errori, il *data server* risponde con *ack*. La connessione viene chiusa.

Quando il trasferimento è completato con successo, inizia la terza fase. Il *data server* invia un messaggio del tipo *push_complete(uid)* al *meta server*. Il meta server inizia quindi ad inviare ai *data server* interessati le richieste di trasferimento, per raggiungere il grado di replicazione voluto.

(TODO aggiungi richieste trasferimento)

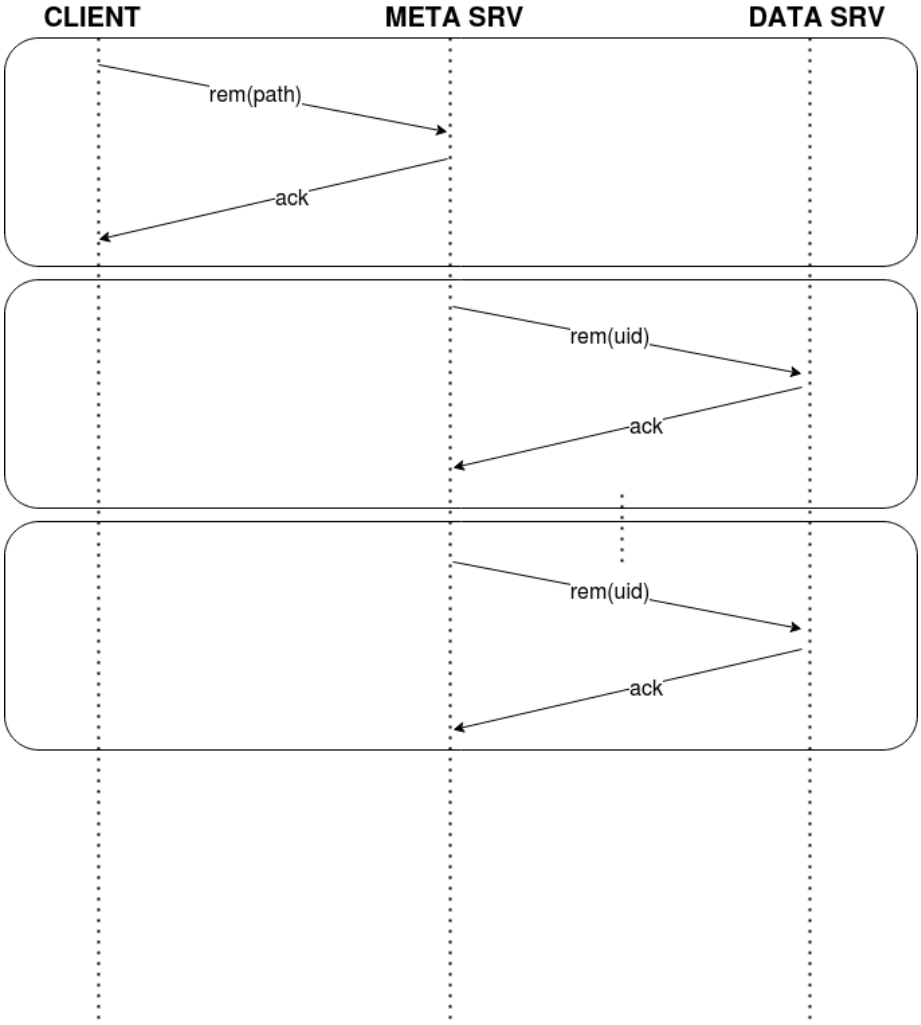


4.3 rem

La richiesta di tipo *rem(path)* richiede l'eliminazione di un file/directory dallo storage.

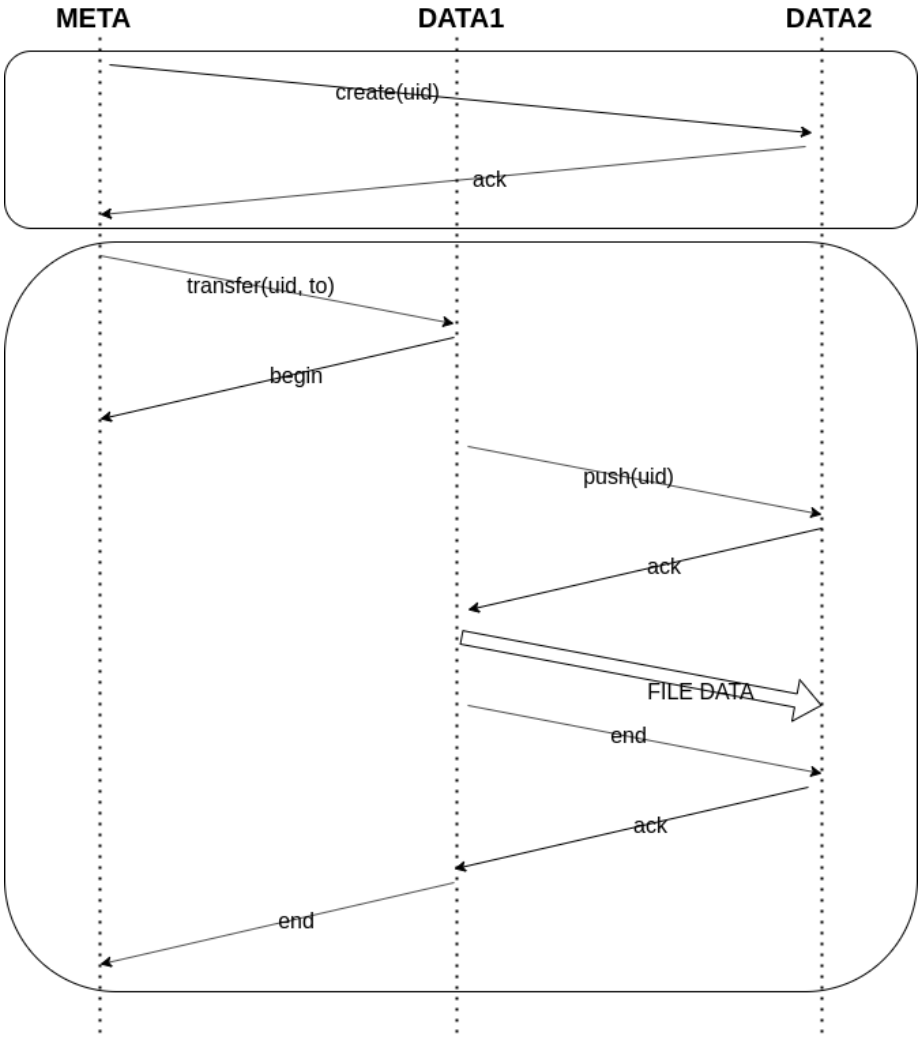
Il *client* invia al *meta server* una richiesta *rem(path)*. Se non ci sono errori, il *meta server* risponde con *ack* e chiude la connessione.

Il *meta server* contatta separatamente tutti i *data server* contenenti dati relativi a quel particolare *uid* e invia una richiesta *rem(uid)*.



4.4 transfer

(TODO fai tutto)



5 Logging

Tutte le operazioni prima di essere eseguite vengono registrate in appositi database di log. In questo modo, se un'operazione viene interrotta per qualsiasi motivo, può essere analizzata e rieseguita, in modo da non lasciare spazzatura.

References