FED-FT
(FEDerated File Transfer)

INTRODUZIONE

Questo lavoro è stato messo in opera con lo scopo di avvicinarsi ad alcuni temi caratteristici che sorgono quando entità diverse intendono interagire in un ambiente non a memoria condivisa, in un ambiente in cui la comunicazione fra tali entità deve essere gestita esplicitamente. In Particolare in questo lavoro i temi maggiormente studiati sono stati la interazione basata sul modello cliente/server, la gestione di risorse replicate, e i protocolli di comunicazione delle informazioni. Il metodo di lavoro seguito può essere schematizzato con la seguente catena di attività:

1) Messa a fuoco degli obiettivi;
2) Stesura delle specifiche Utente;
3) Stesura delle specifiche sistema Client/Server;
4) Analisi individuazione e scelta delle possibili soluzioni;
5) Implementazione

Con evidenti feed-back interstadio.
Il modello di interazione proposto è del tutto generale: si è scelto come entità oggetto dello scambio informativo client/server e server/server il file solo per comodità, essendo il file un concetto che può essere esteso a qualunque gruppo di dati, ed essendo il carattere di questo lavoro principalmente esplorativo.

1.Messa a fuoco degli obiettivi astratti

All'utente deve essere fornito un livello minimo di trasparenza alla allocazione sia degli oggetti (file) a cui egli fa riferimento, sia dei fornitori dei servizi che esso richiede, i Server. L'interazione dell'utente con il sistema FED-FT deve avvenire esclusivamente sulla base dei comandi messi a disposizione dai processi Client. Le uniche informazioni che l'Utente deve interpretare sono:

1) Lista dei file correntemente allocati nel sistema (risorse disponibili)
2) Stato della risorsa: accessibile (unlocked) o inaccessibile (locked)
3) Risposte alle richieste di servizio bipolari: o SI o NO.

Ogni tipo di riferimento alla allocazione fisica della risorsa deve rimanere confinata all'interno del sistema, nell'interazione C/S e S/S.

2.COMANDI UTENTE

Di seguito è riportata il set minimo di comandi che si prevede debbano essere forniti all'Utente. Tale set può essere soggetto ad ampliamento con successive versioni del sistema.

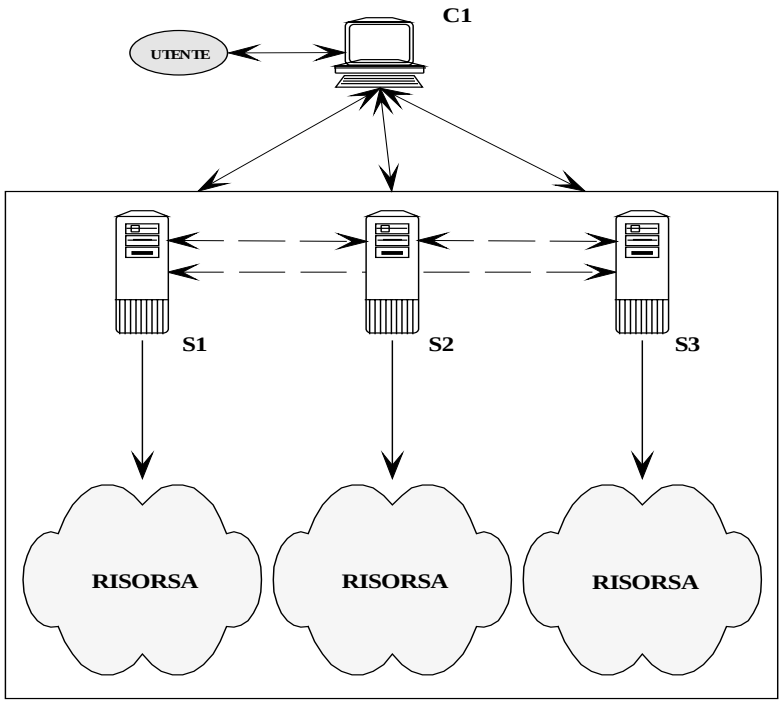
Nome Comando	Azione	Parametri
LIST	Richiesta lista aggiornata file disponibili	Nessuno
SHOWLIST	Stampa a video la lista dei file disponibili	Nessuno
HELP	Stampa l'help sul comando specificato	<nome_comando>
HELP	Stampa l'elenco dei comandi disponibili	Nessuno
GET	Richiesta di download della risorsa	<local_file> <remote_file>
UPDATE	Richiesta di aggiornamento sul server del file	<remote_file>
LOCK	Richiesta di eseguire un lock sul file	<remote_file>
DELETE	Richiesta di eliminare dal sistema il file	<remote_file>
PUT	Richiesta di eseguire un upload del file	<remote_file> <local_file>
DISCONNECT	Richiesta di disconnessione dal sistema	nessuno

Per i comandi LIST, SHOWLIST, GET, UPDATE, LOCK, DELETE e PUT sono previsti due soli risultati finali: successo o fallimento, secondo il paradigma "**tutto o niente"**. Per i restanti comandi HELP e DISCONNECT si considerano sempre terminati con successo.

Poiché l'obiettivo di tale lavoro è analizzare l'interazione fra le entità coinvolte a realizzare il sistema, vale a dire l'interazione fra processi Client e processi Server, si ritiene sufficiente la tabella suesposta per definire le specifiche a livello di Utente.

3. FEDerated File Transfer: analisi del sistema e scelte architetturali

Nella sottostante figura è riportato il modello di riferimento del sistema: il sistema è costituito



da una **costellazione** di entità Server che interagiscono fra di esse per fornire all'Utente delle entità Client l'astrazione di un insieme unico delle risorse esportate. Ciascuna entità Server agisce su risorse possedute localmente, la cui unione con le risorse possedute dagli altri Server della costellazione concorre a formare l'insieme delle risorse esportabili. A livello utente quindi si realizza una sorta di trasparenza all'allocazione, trasparenza che solo parzialmente è mantenuta a livello di processo Client, il quale è di volta in volta connesso con un solo Server della costellazione.

L'architettura proposta è fondata sul classico modello di interazione Client/Server, e tale modello è stato adottato anche per le interazioni fra Server e Server.

Il processo Client deve interpretare e soddisfare, se possibile, i comandi dell'utente, fornendo una succinta spiegazione dei motivi che hanno portato all'eventuale fallimento del comando.

Ogni processo Client deve poter accedere a qualunque Server che compone la costellazione, purché tale Server sia attivo. Non esiste a priori una gerarchia prefissata sui Server accessibili, la priorità di accesso sarà definita dinamicamente sulla base di:

1) frequenza di accesso alle risorse possedute localmente dal Server;
2) tempi di risposta del Server ai messaggi del Client (turnaroud-time),

REDIRECTION
Lo schema fondamentale di interazione Client/Server è quindi definito da:

1) il Server fatto oggetto della richiesta del servizio che è in grado di soddisfarla localmente è tenuto a soddisfare la richiesta localmente, compatibilmente con altri vincoli imposti sulla risorsa(lock).
2) il Server che non possiede localmente la risorsa o non detiene sufficienti diritti in funzione del tipo di servizio richiesto è tenuto a comunicare al Client l'indirizzo dell'ultimo possessore della risorsa, noto al Server stesso.

Alla luce delle precedenti specifiche si hanno due tipi di **redirection**: 1) ciascun Server ha la facoltà di reindirizzare temporaneamente (per la sola durata di un servizio); 2) ciascun Server ha la facoltà di reindirizzare a tempo indefinito (interazione stabile per più servizi) il Client, qualora ne sussistano le condizioni.

Sulla scorta del modello di interazione Client/Server proposto sono sorte delle considerazioni sull'efficienza di tale modello nel soddisfare le esigenze degli utenti, obiettivo primo del sistema.
In particolare l'adozione del modello fa sorgere i seguenti problemi:

1) Redirection multipla chiamata in questa sede **Address Bouncing**;
2) Approvvigionamento delle risorse

Il punto 1) discende immediatamente dal meccanismo adottato: nel caso un Server non possieda la risorsa richiestagli emette un messaggio di redirection al Client, costringendo quest'ultimo a connettersi con il nuovo Server, che potrebbe nel frattempo aver ceduto a un terzo Server il possesso della risorsa. Questo meccanismo potrebbe iterarsi per tutta la durata della ricerca del Server possessore.

Il punto 2) dal contrasto fra gli effetti della redirection (migrazione del'owner della risorsa) e dai 2 requisiti secondo i quali deve essere stabilita la connessione fra Client e Server: requisiti che in sintesi dicono che il Client si deve connettere al Server che risponde più velocemente alle sue richieste. Nasce quindi la necessità di **replicare le risorse**.

REPLICAZIONE RISORSE

Come si è visto, si rende necessario approntare un meccanismo di gestione della replicazione delle risorse: **caching dei file**. Conseguenza del caching e delle necessarie politiche di aggiornamento delle copie si è pensato di introdurre il conceto **di ownership della risorsa**.

OWNERSHIP

Nel sistema possono coesistere più copie dello stessa risorsa. Si introduce la distinzione fra copia attiva e copia passiva: **copia attiva** è quella copia che può essere modificata; **copia passiva** è quella copia su cui si possono eseguire solo operazioni di lettura (read only).

Nel sistema può essere presente **una sola copia attiva**, e il Server che possiede localmente tale copia è detto Server **owner della risorsa**. Un Server **non owner** può possedere solo copie passive: si dice allora che il Server possiede la risorsa nello stato **cached**;

L'introduzione del concetto di ownership introduce un elemento di centralizzazione nel sistema a costellazione, imponendo sui Server una gerarchia , relativamente a una data risorsa.

Con l'introduzione del meccanismo di caching sorgono tutte le tipiche tematiche relative alla gestione della cache. In particolare bisogna decidere:

1) quando un Server deve richiedere il caching di un file,
2) come il Server debba acquisire la copia
3) come gestire eventuali richieste di updating
4) come gestire la coerenza fra le copie

Possibili soluzioni e scelta effettuata.

1)Per quanto riguarda questo punto le soluzioni possibili considerate sono:

- 1.1) Richiedere il caching del file ogniqualvolta il Client richieda un servizio sul file stesso.
- 1.2) Richiedere il file solo quando l'ammontare degli accessi al file supera una certa soglia

Si è scelto come soluzione la seconda, in quanto con la prima si ritiene si possa generare troppo traffico sulla rete, traffico spesso generato per mettere in cache un file poi non più acceduto dal Client (caching inutile). Tale soluzione inoltre genererebbe una notevole infiorescenza di copie passive, generando ulteriore traffico per gestire il processo di invalidazione.

Soluzione adottata: richiedere il caching del file solo quando il numero di accessi ha superato una soglia prefissata.

2)La scelta è univoca: il Server invia una richiesta di trasferimento al Server owner della risorsa. Necessità di risolvere l'Address Bouncing;

3) e 4)A tale scopo è stato messo a punto un opportuno protocollo di updating (vedere apposito paragrafo) che si fonda sul concetto di **copia cached valida e copia cached non-valida**.

OWNERSHIP: Necessità di trasferirla da un Server all'altro.

L'ownership rappresenta, in sostanza, la responsabilità di un Server sullo stato della risorsa. È la copia locale a un Server che ne detiene l'ownership che costituisce il riferimento ultimo nei casi di inconsistenza delle copie. Il carattere dinamico del sistema potrebbe portare alla seguente situazione:

un Client accede frequentemente in update a una risorsa che è solo mantenuta **cached** sul Server connesso S1, mentre il Server che ne detiene l'ownership, S0, registra scarsi accessi alla risorsa.

È evidente che una tale situazione penalizza fortemente il sistema, il quale a ogni update su S1 dovrebbe avviare il meccanismo di cache invalidate e seguente redirezione del Client su S1, che si presume "più lontano" (nel senso descritto precedentemente) rispetto a S1.
Per ovviare a questo punto di inefficienza si è introdotto il protocollo di passaggio dell'ownership, chiamato in questa sede, **Ownership Passing Protocol: OPP**.

OPP

L'OPP, viene attivato nelle condizioni determinate nel successivo paragrafo dedicato all'updating. Prevede lo scambio fra i due Server coinvolti delle informazioni relative al numero di accessi a cui correntemente il file in esame è soggetto su entrambi i Server. L'ownership viene di fatto passata solo quando il numero di accessi del Server richiedente supera di un certo delta quello del Server owner. Poiché l'ownership è una informazione estremamente importante per l'integrità e coerenza del sistema si è pensato di stendere un protocollo piuttosto pesante, per allontanare il più possibile la possibilità che i due database dei file esportati sui due Server coinvolti possano rimanere in uno stato inconsistente (il file rimane con due owner o, peggio senza owner, perché errori o malfunzionamenti sono incorsi durante il protocollo).

Durante la fase di gestazione del protocollo si sono presentate due possibilità:

1) OPP con Broadcast
2) OPP con Bouncing

La soluzione 1) prevedeva di inviare a tutta la costellazione, in modalità broadcast, un messaggio del tipo OPP_REQ, a cui il solo Server owner avrebbe dovuto rispondere in maniera positiva (o con un ACKnowledge o con un DENY). Questo meccanismo richiama

esplicitamente il protocollo ARP per l'address resolving su reti IP. Come estensione si poteva implementare un reply da parte dell'owner in modalità broadcast in modo che tutte le tabelle

Vantaggi: Tutte i database locali sono aggiornati in caso di OPP, quindi riduzione del livello di redirezione a due soli stadi mediamente, salvo errori di comunicazione. Facilità di implementazione rinunciando esplicitamente all'Address Bouncing sull' OPP;
Svantaggi: traffico generato sulla rete e difficoltà di gestire il pesante protocollo in modalità broadcast. Possibili prolungati blocchi se si decidesse di implementare il broadcast in maniera reliable, aspettando gli ACKnowledge di tutte le entità coinvolte.

La soluzione 2) prevede di la connessione diretta 1 a 1 fra Server richiedente e Server owner, rendendo necessaria la fare di address resolving risalendo la catena. Una volta stabilita la connessione la OPP è cosa privata fra i due Server.

Vantaggi: riutilizzo dello stesso meccanismo adottato per la gestione delle copie. Coinvolgimento delle sole parti terminali interessate. Il protocollo è confinato solo nei due Server coinvolti.
Svantaggi: il meccanismo è esposto ai problemi tipici dell'Address Bouncing: necessità di risalire la catena di redirezioni, necessità di mantenere l'integrità della catena.

Scelta effettuata.
OPP con bouncing è sicuramente più difficile e delicato da implementare rispetta alla alternativa con broadcast, ma tenendo presente che già da ora si era scelto TCP/IP come sottosistema di comunicazione e tenendo presente l'opportunità di riutilizzare meccanismi di basso livello implementati per le funzioni di caching, si è optato per la OPP con Bouncing BOPP.

BOPP si basa su 4 fasi distinte (si veda al proposito lo schema di interazione proposto nella sezione "Schemi di Interazione Client/Server/Server"):

Fase 1): Invio da parte del Server richiedente Sr di un PSO_MSG al Server che si crede owner della risorsa. Risalita della catena di redirezioni fino a che si ricevono REDIR dai Server interpellati. La fase uno termina quando si raggiunge l'effettivo owner della risorsa.In questa fase i database dei file non sono modificati in alcun Server interpellato

Fase 2): Il Server owner So riceve il PSO_MSG, vaglia la fattibilità del passaggio e risponde con DENY o con un ACK1. Nel primo caso il BOPP lato So termina. Se Sr riceve correttamente il DENY il BOPP termina anche lato Sr. Se DENY va perso in questa fase, il BOPP deve essere riattivato dall'inizio, ma non si ripercorre la catena interamente, si riparte dall'ultimo Server (So nel caso) in esame.Lo stesso accade se si perde un REDIR nei passaggi intermedi.

Fase 3) Supponiamo che So abbia inviato ACK1 e che Sr lo abbia ricevuto correttamente. A questo punto Sr **modifica il database impostando la proprio ownership sulla risorsa**. I database di So e Sr sono ora globalmente inconsistenti: la risorsa ha due owner. Sr emette allora ACK2 e si pone in attesa di ACK3. Se So non riceve ACK2 termina il BOPP conservando le vecchie informazioni nel database. In questo caso se i successivi resume del BOPP da parte di Sr falliranno, si renderà necessaria una operazione di rollback sul database di Sr.

Fase 4):Supponiamo che **So** riceva ACK2: Esso allora applica le modifiche al proprio database impostando il nuovo owner: ora i due database sono nuovamente coerenti. **So** invia ACK3 e termina definitivamente il BOPP lato Sr. Se ora Sr NON riceve ACK3, non sa se **So** abbia applicato o meno le modifiche. **Sr** fa un resume di BOPP inviando in questa evenienza un messaggio NOT_OWN a cui esso è tenuto a rispondere con un nuovo NOT_OWN.

GESTIONE DELLA CACHE

Per gestire la cache come meccanismo fondamentale si è scelto quella già esposto della fetch on demand con soglia nel caso di accessi in lettura (si rinuncia al readhaed, essendo troppo costoso in un sistema interconnesso quale quello che stiamo considerando).
Nel caso di accessi in update si applicano le seguenti due politiche:

1) Fetch On Write se il file è in uno stato valid cached e il numero di accessi supera una determinata soglia superiore a quella del semplice fetch on demand. Questa modalità implica necassariamente l'attivazione del BOPP.
2) ReadHaed nel caso giunga un update verso un file non cached (o non valid cached) e contemporaneamente si supera la soglia del fetch on demand (in questo caso si adotta una politica speculativa: se il Client aggiorna il file è probabile che ne richieda un accesso in tempi brevi). L'ownership non passa in alcun caso di mano.

In Ogni caso quando è garantito un accesso in update ad una risorsa, deve essere attivato il Cache Invalidate Protocol CIP, che a discapito del nome è piuttosto semplice. Il **Server owner** della risorsa aggiornata si impegna ad emettere broadcast un messaggio CAC_INV e ad attendere che tutti i Server della costellazione rispondano al messaggio con un ACK, sia che essi possiedano la risorsa valid cached, sia che la possiedano invalid cached o sia che non la possiedano affatto.

GESTIONE UPDATE delle risorse

Si sono riconosciuti 4 casi diversi di gestione dell'update. I parametri che hanno portato all'individuazione dei caso sono: possesso o meno dell'ownership, stato dell'evetuale copia cached, (valid cached, invalid cache, non cached), stato dei contatori di accesso. Nei seguenti casi si considera S1 come il Server correntemente connesso con il Client e S2, S3… come gli altri Server della costellazione non direttamente connessi con il Client

A) S1 è owner della risorsa.
B) S1 **non** è owner e **non** possiede valid cached il file,e il numero di accessi al file è scarso (**inferiore** alla soglia di fetch on demand)
C) S1 **non** è owner e possiede valid cached il file e il numero di accessi è **maggiore** alla soglia di fetch on write
D) Si **non** è owner della risorsa e **non** la possiede valid cached, **ma** il numero di accessi è maggiore della soglia di fetch on demand.

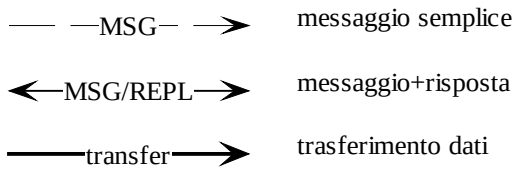
Nell ípotesi che non sussistano vincoli sulle risorse (lock) si adottano le seguenti soluzioni:

A) S1 accetta la richiesta di update, avvia il CIP al termine del quale applica le modifiche al file e invia a C l'ACKnowledge dell'operazione. **N.B.** Il fatto di differire l'applicazione delle modifiche e dell'invio dell'ACK a C **solo dopo** la conclusione del CIP è una precisa scelta, attuata nell'ottica di garantire il più possibile la coerenza delle copie. In alternativa si sarebbe potuto adottare una politica asincrona: S1 invia l'ACK e applica le modifiche **contemporaneamente** alla esecuzione del CIP.
B) In questo caso S1 svolge solo una funzione passiva:rinvia C all'effettivo owner ed attende l'arrivo di un eventuale CAC_MSG dall'owner remoto
C) S1 tenta di acquisire l'owership attraverso BOPP. In caso positivo, al termine del BOPP si ricade nel caso A.Nel caso BOPP si concluda con DENY, S1 invia un REDIR a C e invalida la sua copia, e ricade nel caso D;
D) S1 risponde con REDIR alla richiesta di C e si pone in attesa di un secondo messaggio da parte di C che lo informa sull'avvenuto o meno updating presso il Server owner remoto. In caso riceva ACK da C avvia la procedura di caching utilizzando l'indirizzo fornitogli da C con l'ACK. In caso di DENY termina la fase.

SCHEMI DI INTERAZIONE Client/Server/Server

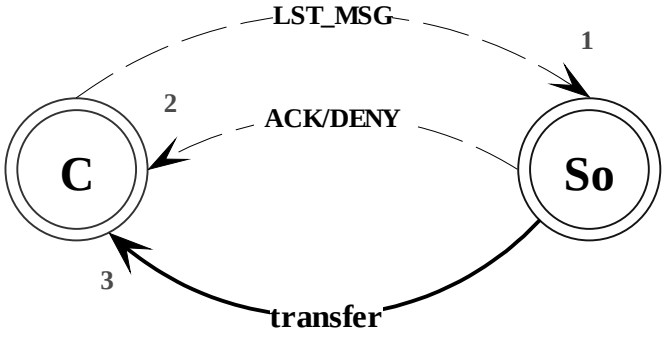
Come si è anticipato in precedenza l'interazione fra le varie entità avviene sulla base dello scambio di messaggi, della loro interpretazione e delle azioni che ne seguono. Per ogni attività è stato definito sia un set di messaggi scambiabili (ammessi) che il set di modalità con cui devono essere scambiati (il protocollo).
Di seguito sono riportati gli schemi di interazione (messaggi, protocollo) fra le varie entità coinvolte nel sistema, per le attività più importanti e peculiari del sistema .

Negli schemi sono riportati:
nome attività, messagi scambiati e loro eventuali parametri, schema che esemplifica il protocollo, ed eventuale spiegazione a parole della successione di interazione.



Legenda: 1

COMANDO LIST

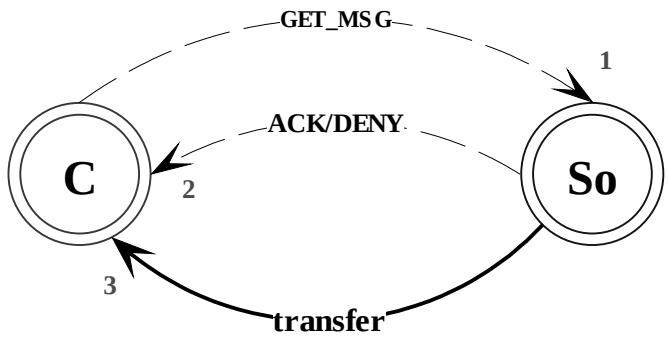


MESSAGGI AMMESSI

C->S:LIST_MSG.
S->C: ACK<port><filelen>.
S->C: DENY:

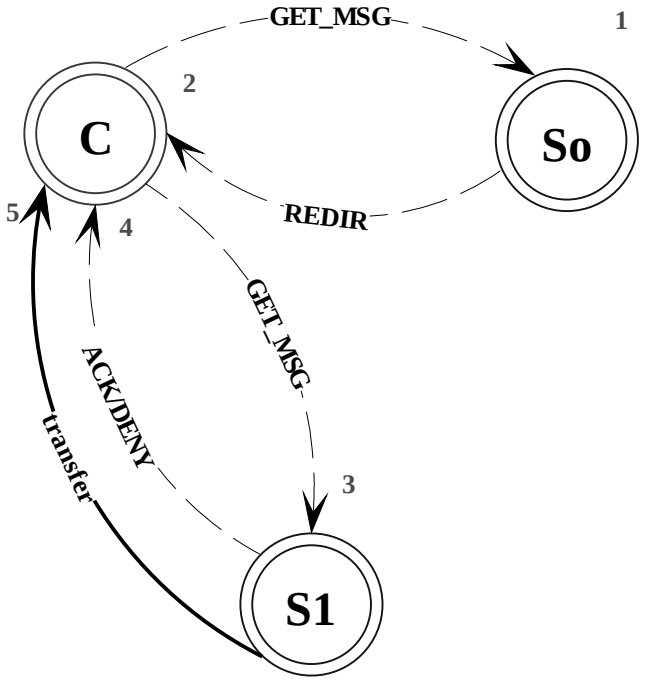
Il Client invia un messsaggio di LST_MSG (1) con il quale richiede la lista dei file esportati. Il Server controlla che la lista sia disponibile o meno e invia, rispettivamente un messaggio di ACK o di DENY (2). Successivamente il Server attiva la procedura di trasferimento in attesao che il Client si connetta e incominci a fare il download.(3).

COMANDO GET



MESSAGGI AMMESSI
C->S: GET_MSG<filename>.
S->C: ACK<port><len>.
S->C: DENY.
S->C:REDIR<ipaddr><port>.

Get con connessione diretta al server owner del file.
Il C invia un messaggio di get con incluso il nome del file che vuole acquisire(1). Il S controlla la disponibilità del file localmente (cache valida o ownership) e risponde di conseguenza con ACK o DENY(2). In caso di ACK il S comunica la porta alla quale il C dovrà collegarsi e la dimensione del file in richiesto. C e S attivano quindi la comunicazione per il download del file.



In questo caso il S0 non possiede né la ownership del file né una copia valida. Ciò attiva il protocollo di redirezione della richiesta (**Address Bouncing**). Il Server interroga il database dei file e il database dei Server, comunica al Client l'indirizzo dell'ultimo Server S1 conosciuto come l'owner del file. Il Client conseguentemente riattiva la procedura di get sul nuovo Server S1 il quale a sua volta potrebbe aver ceduto l'ownership a un terzo Server senza possedere una copia valida. Si estrinseca così la tecnica di recupero dell'indirizzo dell'owner attraverso la catena di reindirizzamenti, tecnica che qui è stata arbitrariamente chiamata Address Bouncing

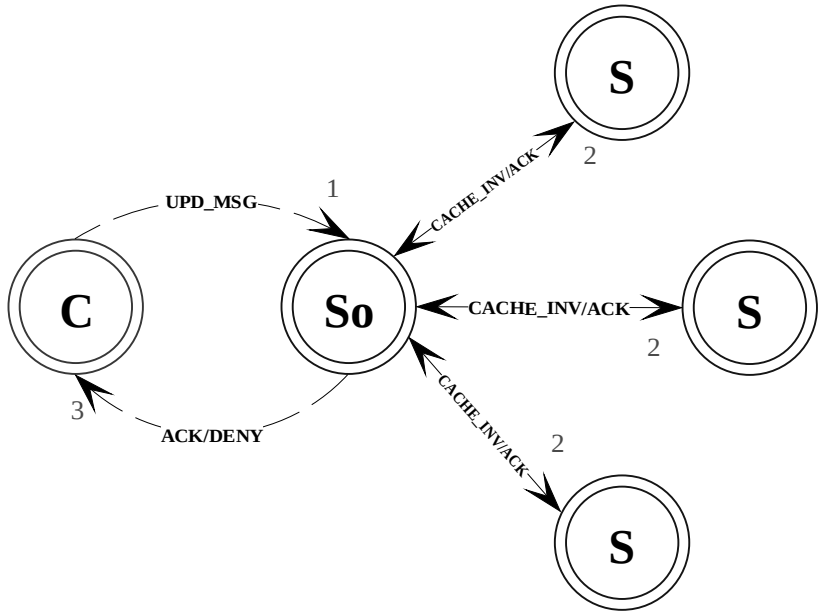
COMANDO DI UPDATE

La complessità del gestire il comando di update deriva dai numerosi side effect di cui si è precedentemente parlato in occasione delle politiche di caching. Qui brevemente riportiamo gli schemi dei quattro casi in cui si ricade in occasione del comando di update e della sequenza di comandi/messaggi/azioni che si sono adottati:

Il sommario dei segnali coinvolti in una generica operazione di update è il seguente:

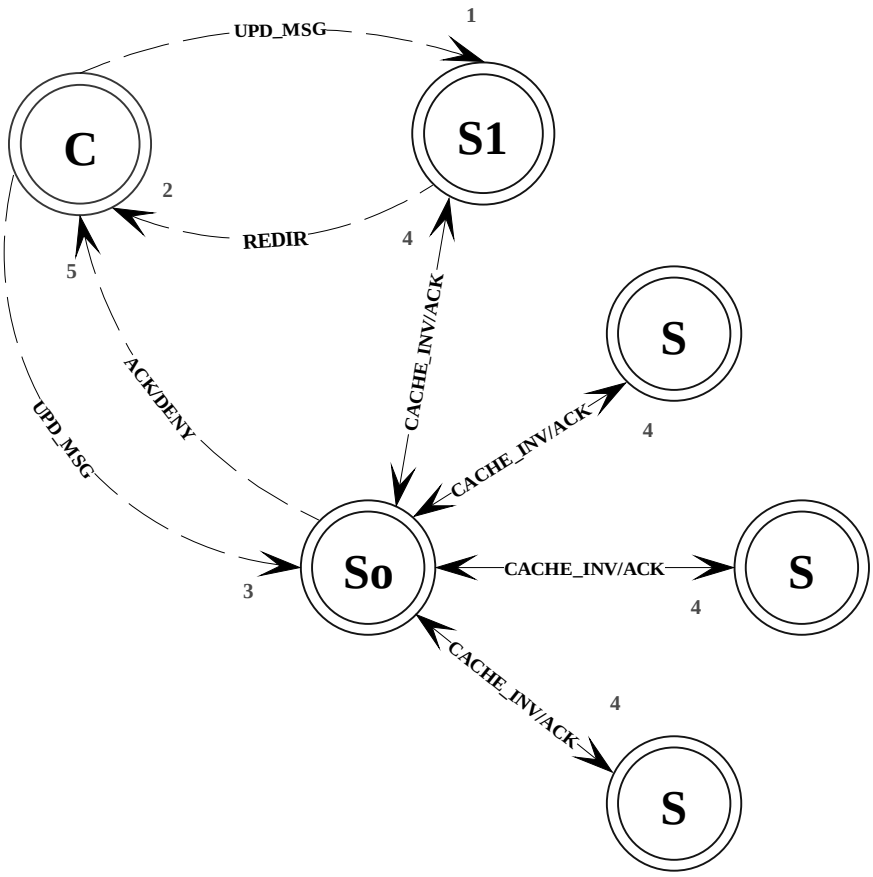
C->S: UPD_MSG<filename>	**S->S: REDIR**
S->C: DENY	**S->S: CAC_INV**
S->C: ACK	**S->S: PSO_OWN**
C->S: ACK	**S->S: ACK**
S->C: REDIR	**S->S: NOT_OWN**

CASO A: S0 OWNER del file



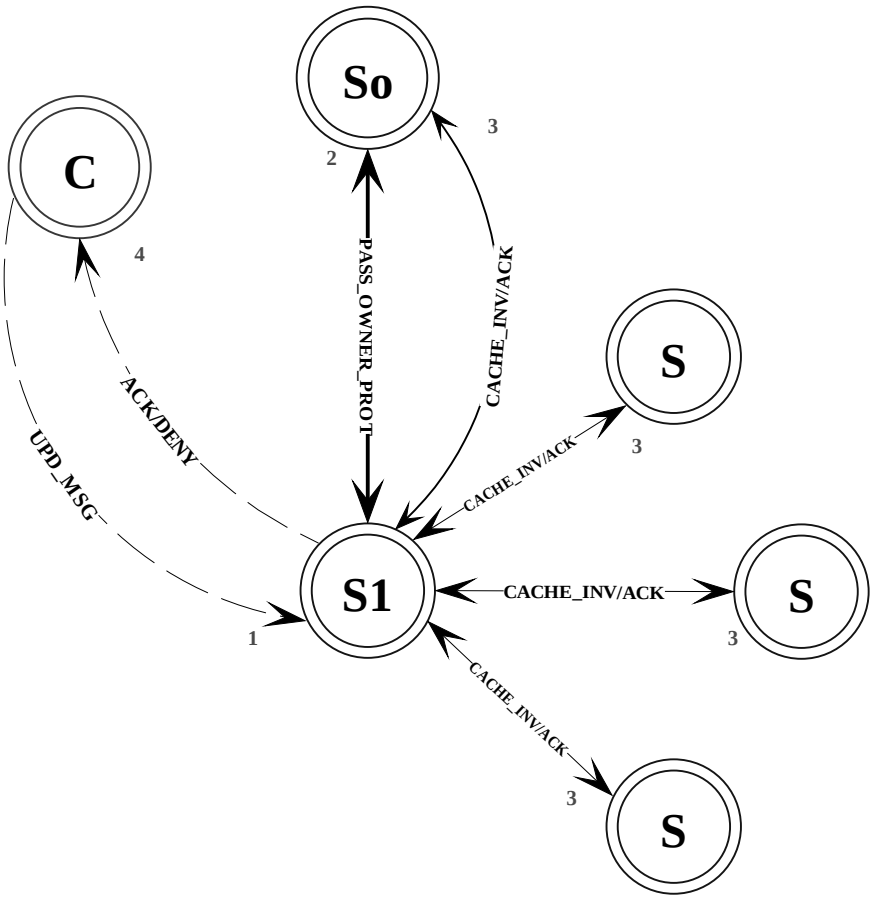
C->S0: UPD-MSG<filename>.
IF(filename is locked) THEN S0->C: DENY. (fine)
ELSE
S0->S: CACHE_INV<filename>. (messaggio broadcast)
WHEN (ALL S->S0: ACK. OR TIMEOUT)
LOCAL_UPDATE(filename);
S0->C: ACK.
ENDIF.

CASO B: S0 NON OWNER, filename NON CAHED, NO FETCH



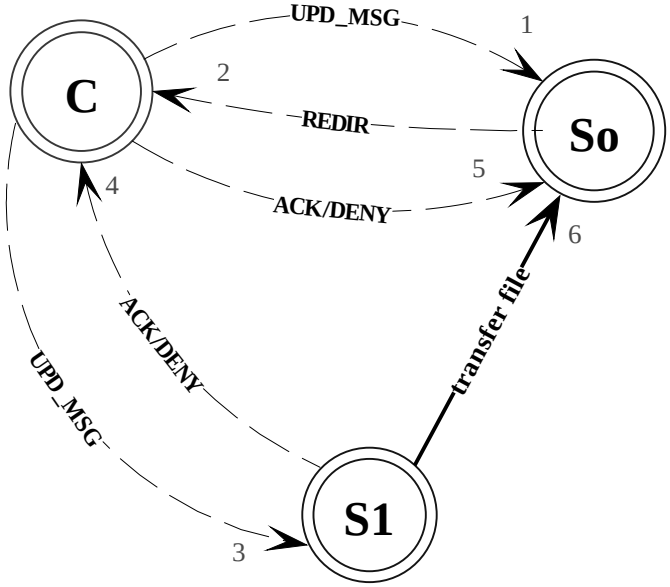
C->S0: UPD_MSG<filename>.
IF(filename is locked) THEN S0->C: DENY. (fine)
ELSE
S0->C: REDIR<remoteserver>.)
C->S1: UPD_MSG<filename>.
IF(filename is locked) THEN S1->C: DENY.
ELSE CIP da S1.
ENDIF
ENDIF

CASO C: S1 NON OWNER, copia CACHE VALID, FETCH ON WRITE



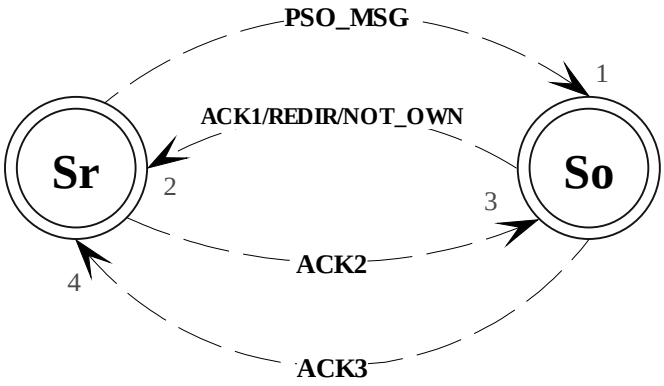
C->S1: UPD_MSG<filename>.
S1->S0: BOPP
IF(OK) THEN
	CIP da S1
	Aggiorna filename;
	S1->C: ACK;
ELSE
	Inavlida la copia locale
	S1->C: REDIR<owner addr>
ENDIF

CADO D: S1 NON OWNER, file NON CACHED: READ AHEAD.



C->S0: UPD_MSG<filename>
S0->C: REDIR<newowner>
C->S1: UPD_MSG<filename>
IF(file is locked) THEN
    S1->C: DENY
    C->So: DENY
ELSE
    S1->C: ACK;
    C->S0: ACK<newowner>;
    S1->So: transferfile;
ENDIF

BOUNCING OWNERSHIP PASSING PROTOCOL



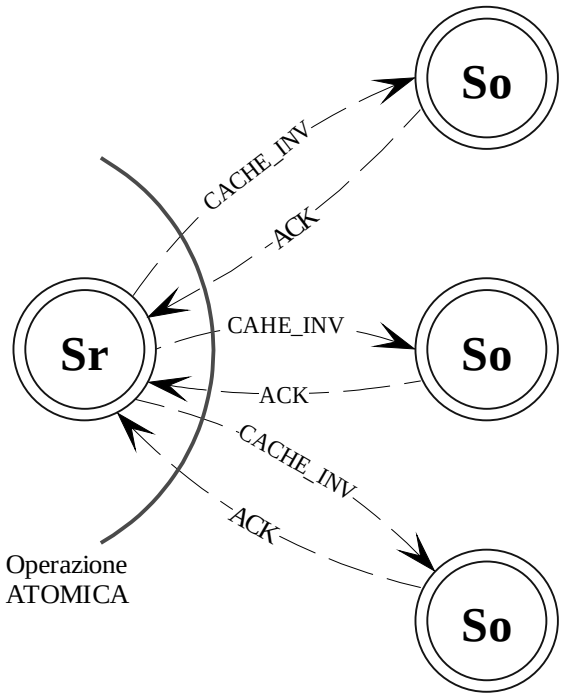
In questo frangente spiegheremo solo l'OPP, considerando già conclusa la fase si address bouncing. (1) Il Server richiedente, Sr, inoltra presso il Server owner, So, un PSO_MSG con annessi come parametri il nome del file di cui si vuole acquisire l'ownership e il numero attuale di accessi cui il file è stato fatto oggetto sul Server Sr. Ricevuto il messaggio, So confronta il numero fonitogli con il numero di accessi cui il file è fatto oggetto localmente. Sulla base del confronto So stabilisce se cedere o meno l'Ownership. In caso positivo esso invia il messaggio ACK1 (2) a Sr. Sr ricevuto e correttamente decodificato ACK1, provvede a modificare permanentemente il database dei file, attribuendo a se stesso l'ownership del file. Terminata l'operazione invia a So un nuovo messaggio di ACK (3) che provocherà l'aggiornamento del database dei file anche su So stesso, il quale inserirà le informazioni relative a Sr. A questo punto a So non rimane che inviare a Sr il messaggio che chiude definitivamente il protocollo.

Comportamento in caso di perdita di messaggi.

Perdita del messaggio (1) e (2): Sr attende lo scadere del timeout al termine del quale provvederà a inviare ancora il messaggio (1). I due database sono non modificati e quindi non inconsistenti Al termine di un numero prefissati di tentativi Sr abortirà OPP dichiarando impossibile la cessione dell'ownership.

Perdita del messaggio (3): So non aggiorna il proprio DB, ma Sr non sa esattamente quale sia lo stato di So. Allo scadere del timeout, Sr esegue un resume di OPP inviando a So un messaggio NOT_OWN <filename>, con il quale comunica a So di essere nella seconda fase del protocollo. A tale messaggio So risponde riprendendo il protocollo dall'invio di un nuovo messaggio di NOT_OWN con il quale informa Sr di aver già aggiornato il DB.Quando Sr riceve un messaggio di NOT_OWN il protocollo termina. Se anche i successivi messaggi NOT_OWN di Sr non ricevono risposta, Sr esegue rollback e ritorna un fallimento su OPP.

COMANDO CACHE INVALIDATE



Si è pensato di utilizzare un modello di comunicazione a broadcast per inviare i comandi di cache invalidate. Teoricamente si vorrebbe l'intera operazione di invio e raccolta degli acknowledgement come atomica: l'operazione termina con successo **se e solo se** tutti gli ACK in risposta al messaggio CAC_INV sono stati raccolti, in caso contrario l'operazione ritornerebbe con un fallimento. Benché questa sia una situazione desiderabile e auspicabile non è detto in realtà che si presenti in tutte le occasioni, dato il sottosistema di comunicazioni e dato il modello di interazione fra i Server della costellazione.

Il trade-off che si presenta è fra: la necessità di coerenza delle copie, e la necessità di non interrompere il servizio ogni qual volta un Server non si rende disponibile. Infatti se un Server per cause indipendenti dalla costellazione si trova in uno stato di irraggiungibilità o di failure ciò pregiudicherebbe il caching di qualsiasi file nel sistema, penalizzando eccessivamente le prestazioni. Si tollera, quindi, che una azione di cache invalidate possa terminare con successo anche quando uno o più Server non rispondono al comando, tollerando di conseguenza un certo grado di inconsistenza nelle copie del sistema.

BREVI CENNI SULLA IMPLEMENTAZIONE

Il sistema è costituito da un programma Server e da un programma Client. Il Server era stato originariamente pensato per funzionare come daemon, quindi come un processo che si instaura permanentemente nel sistema e sovrintende alle risorse destinateli. Originariamente il Server dovrebbe eseguire un binding sulle TCP port come se ad esso fosse stata concessa una well-known port, ma per la necessità di far funzionare il sistema anche con privilegi utente si è ricorsi a una modalità manuale di assegnamento delle porte (si vedano le avvertenze per lanciare il sistema allegate nel tar dei file sorgenti).

Le caratteristiche peculiari del Server sono:

1) utilizzo di stream socket per implementare il sistema di comunicazione (TCP)
2) multithreading per gestire i comandi provenienti dai Client e soddisfare le richieste di caching da parte dei Server
3) motore di gestione dei comandi provenienti dai Server seriale

In fase di start-up il main thread crea due diversi thread uno preposto a soddisfare in maniera seriale i comandi provenienti dagli altri Server, e uno preposto a soddisfare le richieste provenienti dai Client. Lo thread preposto alla gestione dei Client a sua volta genera un successivo thread per ogni connessione stabilita con i Client.
Altre caratteristiche salienti del Server sono:

1) Implementazione del broadcast per il cache invalidate in multithreading: ogni connessione verso gli altri Server è gestita autonomamente da uno thread dedicato.
2) Parallelismo fra soddisfacimento di una richiesta di get da parte del Client ed eventuale caching richiesto dal Server. Questa modalità è offerta dall'address bouncing che redirige il Client verso una connessione diretta con il Server owner permettendo quindi una contemporanea e semplice sovrapposizione delle operazioni.
3) Mascheramento da Client di un Server per compiere alcune operazioni: file caching e upload di file dal Client

È da notare che il sistema è stato pensato per essere stabile, quindi la configurazione del sistema è intrinsecamente difficoltosa, basandosi in puri spirito unix su file di configurazione memorizzati in locale. I file di configurazione sono tre: due per ciascun Server e uno comune per tutti i Client.I file di configurazione per i Server sono**: file.db**, il database globale dei file esportati; **server.db** il database di tutti i Server della costellazione.

Di seguito sono riportati due esempi di file.db e Server.db:

Server.db

[Name]	[IP ADDR]	[C-PORT]	[S-PORT]	[STATUS]
serverA	192.168.0.1	45001	50001	1
serverB	192.168.0.1	45002	50002	2
serverC	192.168.0.2	45003	50003	3

file.db

[Name]	[Status]	[Location]	[access time]	[Owner]
file1.txt	80	10	3	serverA
file2.txt	80	10	1	serverA
file3.txt	80	10	6	serverA
file4.txt	80	10	2	serverA
file5.txt	80	10	2	serverA
file6.ttt	60	30	4	serverB
file7.ttt	60	30	2	serverB
file8.ttt	60	30	5	serverB
file9.ttt	60	30	1	serverB
file10.ttt	60	30	2	serverB
file11.txx	60	30	6	serverC
file12.txx	60	30	5	serverC
file13.txx	60	30	5	serverC
file14.txx	60	30	7	serverC
file15.txx	60	30	1	serverC

Lato client è indispensabile configurare il database dei server ragiungibili:

Client_Server.db

[IP Address]	[C-Port]
192.168.0.1	45001
192.168.0.1	45002
192.168.0.2	45003

CENNI SULL"INSTALLAZIONE DEL PROTOTIPO:

Il sistema fornito in allegato è da considerarsi né più né meno che un prototipo, dato che è sprovvisto delle parti necessarie a far si che possa essere sufficientemente autonomo. In particolare l'installazione dei Server deve essere fatta configurando attentamente e correttamente i due file di configurazione, inserendo esattamente indirizzi IP degli altri Server della costellazione e delle porte sulle quali si porranno in ascolto: C-PORT indica la porta a cui dovranno connettersi i Client verso i Server, mentre S-PORT indica la porta sulla quale i sever sono in attsa di messagi provenienti dagli altri Server.

Nel file tar in allegato è contenuta un sistema composto da 3 directory-Server nelle quali dovranno essere copiati gli eseguibili dei Server (/fedserver_1 /fedserver_2 /fedserver_3) e nelle quali i "file.db" sono stati correttamente configurati, mentre vanno compilati i "server.db" Saranno inclusi anche due directory contenenti i "client_Server.db" che dovranno essere configurati coerentemente con i "server.db" dei Server.

Purtroppo il progetto è cresciuto molto più del previsto, e i ristretti limiti di tempo hanno costretto a implementare solo lo stretto numero necessario di comandi utente per far

funzionare il sistema sfruttando tutte le sue caratteristiche peculiari. Inoltre la gestione degli errori, attività estremamente time-consuming è stat in questa sede perseguita per lo stretto necessario.

NB. Il Server deve essere lanciato nella stessa directory dove sono localizzati i file che gestisce. Un Server deve essere lanciato con un comando del tipo:

./fedserver <c-port> <s-port>

dove **c-port** è la porta che si assegna per le connessioni provenienti dai client e **s-port** quella per le connessioni provenienti dagli altri Server.

Esempio: **./fedserver 45001 50001**

NOTE SULLA COMPILAZIONE.

Il progetto è stato realizzato e provato su una minirete casalinga composta da due sistemi Linux nelle distribuzioni RedHat6.0 (kernel 2.2.5) e una Slakware7 (kernel 2.2.13) utilizzando le glibc2.1. e egcs 2.91.66

Nel tar sono inclusi i due comandi **fedclicomp.sh** e **fedservcomp.sh** che lanciano la compilazione e linking di tutti i moduli.

Socket e thread sono stati utilizzati cercando di mantenersi il più possibile POSIX1.G compliant. Il progetto dovrebbe essere compilabile senza problemi anche in ambienti *BSD.

NB: è necessario compilare con l'opzione **–pthread** oppure **–lpthread** per poter includere il supporto ai thread.

CONSIDERAZIONI FINALI

Questo lavoro è il frutto della volontà di addentrarsi in aspetti peculiari della interazione attraverso comunicazione esplicita di entità distinte. Essendo fonte e stimolo di studio è necessariamente incompleto e sicuramente non privo di errori. Dato che le dimensioni totali del lavoro sono cresciute oltre ogni limite previsto, parte delle intenzioni iniziali sono rimaste nel casseto. In particolare, una parte di sicuro interesse che purtroppo è stata sviluppata solo a livello di abbozzo, quindi non degna di essere inclusa in questo lavoro, riguarda tutta la problematica di resume di un Server da crash: recupero delle informazioni sulla composizione della costellazione da almeno un sever attivo (ciò presume che il file dei Server non sia del tutto corrotto); ricostruzione e ripristino della coerenza del database locale (per tutta la durata del periodo di failure la costellazione si è evoluta a insaputa del Server decaduto) tramite appositi comandi di inquiring sui database remoti…