|  |  |
| --- | --- |
|  | **2010** |
|  | Corso di Sistemi Distribuiti Prof.ssa Valeria Cardellini  Alessandro Pacca 0143230 Marina Dorelli 0127844 Vienna Codeluppi xxx |

|  |
| --- |
| **[File system distribuito transazionale con replicazione ]** |
| [Digitare qui il sunto del documento. Di norma è una breve sintesi del contenuto del documento. [Digitare qui il sunto del documento. Di norma è una breve sintesi del contenuto del documento.] |

# Introduzione

In ambiente client-server, le risorse si trovano spesso ad essere condivise tra i vari client all'interno della rete. L'accesso a tali risorse deve avvenire in modo trasparente alla concorrenza, un compito non semplice da gestire ed implementare all'interno di ciascuno dei singoli server che rendono usufruibile il servizio comune.

Un filesystem distribuito, o più brevemente DFS, prevede che siano presenti uno o più *file-server*, atti a memorizzare file che possono essere letti e scritti (od anche essere inviati, ricevuti, creati) da più client contemporaneamente, e che sono inoltre organizzati e sincronizzati nelle loro molteplici copie, in modo che tutto il sistema venga percepito dall'esterno come una struttura singola ed affidabile.

Lo scopo di un DFS così definito è dunque quello di far beneficiare gli utenti di una maggiore disponibilità dei dati, e gli amministratori del sistema di un miglior bilanciamento del carico.

Un DFS, rispetto ad un filesystem singolo richiede algoritmi più complessi per la sua gestione. Con la replicazione dei server si aggiunge l’elemento di complessità dell’accesso concorrente alle risorse, che essendo distribuite, introducono il problema della mutua esclusione, che diventa anch’essa pensata in modo distribuito. Occorre poi tener conto dell’integrità nel funzionamento delle macchine server, che potrebbero andare incontro a delle failure, causa di incongruenze e malfunzionamenti.

A tal fine, è stato progettato ed implementato per il corso di Sistemi Distribuiti un file system distribuito che si propone come deposito distribuito di file testuali il cui contenuto è reso disponibile e modificabile a tutti i client nella rete.

Il sistema realizzato è in grado di garantire un ambiente transazionale che soddisfi le proprietà ACID per quanto concerne le operazioni sui file, ed il mantenimento integro degli stessi all’interno della rete di file-server, mediante un meccanismo che garantisce la resistenza ad una serie di failure coinvolgenti sia i client che i server. Per garantire l’accesso esclusivo ai file dei server da parte dei client in modo che più client possano scrivere sullo stesso file uno alla volta, è stato scelto di:

* rendere la lettura dei file sempre disponibile,
* Per le operazioni di scrittura e la loro relativa sincronizzazione tra server è stato scelto l’algoritmo distribuito per la mutua esclusione Ricart-Agrawala,
* Il bilanciamento del carico dei file-server è stato gestito tramite assegnamento degli stessi ai client con un algoritmo di roud-robin da parte del DNS.

Nella fattispecie delle proprietà garantite dal sistema si hanno:

* Protocollo per la consistenza
  + Primary-based – nell’applicazione in questione, un client invia le operazioni di scrittura ad un file-server che poi si preoccupa, una volta ottenuto il consenso tramite l’algoritmo Ricart-Agrawala (descritto in seguito), di inoltrare l’aggiornamento ai rimanenti file-server del sistema
* Operazioni transazionali sui file
  + Atomicità – non è ammessa l’esecuzione parziale, ad esempio una lettura parziale del contenuto di un file presente nel sistema
  + Consistenza – un file-server del sistema si troverà in uno stato consistente (non violerà vincoli di integrità e non si contraddirà con gli altri file-server) dal momento in cui una transazione inizia fino a che essa termina
  + Isolamento – ciascuna transazione in un file-system deve essere eseguita in modo isolato ed indipendente senza interferire con le altre in esecuzione
  + Durabilità – una volta che una transizione effettua una richiesta di commit, i cambiamenti apportati non andranno più persi. Se si verifica un malfunzionamento del server durante la scrittura delle modifiche, è presente un file di log dove sono registrate le operazioni effettuate fino al momento che precede il guasto.
* Tolleranza alle failure ed ai guasti
  + Per il server
    - failstop – il server subisce un crash e smette di produrre output, ma riesce ad inviare un messaggio in cui comunica che non è più attivo.
  + Per il client
    - failstop – il server subisce un crash ed il client lo considera inattivo perché non ricevendo più risposte avvia un timeout oltre il quale ricontatta il DNS per ricevere l’indirizzo IP di un nuovo file-server (ed è questa la soluzione adottata nel progetto realizzato per gestire crash del server)
    - omissioni – il client non ha ricevuto uno tra i messaggi che gli ha inviato il server. Come soluzione è stato fatto in modo che il server inserisca un numero sequenziale all’interno dei messaggi verso ciascun client
    - “guasti bizantini” – il server continua a funzionare ma esibisce un comportamento arbitrario o invia messaggi non congrui.
* Mutua esclusione in ambito distribuito
  + Algoritmo Ricart-Agrawala – una premessa essenziale da fare è che in un sistema distribuito non esiste un tempo globale, ma bisogna stabilire delle relazioni di ordinamento temporale fra eventi tramite lo scambio di messaggi. L’algoritmo utilizzato nel progetto, Ricart-Agrawala, è un algoritmo che consente di implementare la mutua esclusione sulle scritture dei file mediante il multicast di messaggi: il client che desidera registrare le proprie modifiche su un file invia la richiesta di commit al file-server che gli è stato assegnato, che a sua volta si occupa di inviarla agli altri server presenti nel sistema. I server del sistema risponderanno al server chiamante in modo affermativo se e solo se non hanno assegnati client che desiderano modificare la risorsa interessata (il file testuale), oppure se hanno un client interessato a compiere tali registrazioni ma ha inviato a questi una richiesta di commit ed inoltre il server in questione ha un ID maggiore (e quindi minore priorità rispetto al server interessato). Una volta ricevuti tutti i messaggi di reply e registrate le modifiche, un file-server potrà rispondere a eventuali richieste che gli erano arrivate da parte di altri server con ID maggiore.
* Bilanciamento del carico
  + Algoritmo Round Robin – con questo algoritmo circolare il DNS serve le richieste dei client, scorrendo in modo sequenziale la lista di file-server che possiede, ed assegnando le macchine ad i client che lo hanno contattato in modo FCFS. Grazie a questa scelta, riusciamo ad ottenere un carico di lavoro abbastanza equo tra i vari server ed evitare che ci siano server sovraccarichi di richieste e server senza richieste nello stesso momento.

Il sistema è stato realizzato in linguaggio C, mediante l’impiego delle funzioni socket C di Berkeley. Essendo delle API (Application Program Interface), si presentano come un insieme di funzioni che le applicazioni possono invocare per richiedere il servizio desiderato. Attraverso di esse un’applicazione può ricevere o inviare dei dati, sia verso un’applicazione sulla stessa macchina, sia verso un’applicazione situata su una macchina distinta. Per permettere la comunicazione del secondo tipo, è necessario che l’applicazione chiamante conosca l’indirizzo dell’applicazione chiamata, compito assolto dal sistema attraverso l’utilizzo di un server DNS (Domain Name System) centralizzato che resta in attesa delle chiamate dei client e fornisce loro gli indirizzi IP dei file-server secondo un algoritmo ad anello circolare. Chiaramente tutti i client conoscono a priori l’indirizzo del DNS. Tale sistema verrà comunque discusso ed approfondito in seguito.

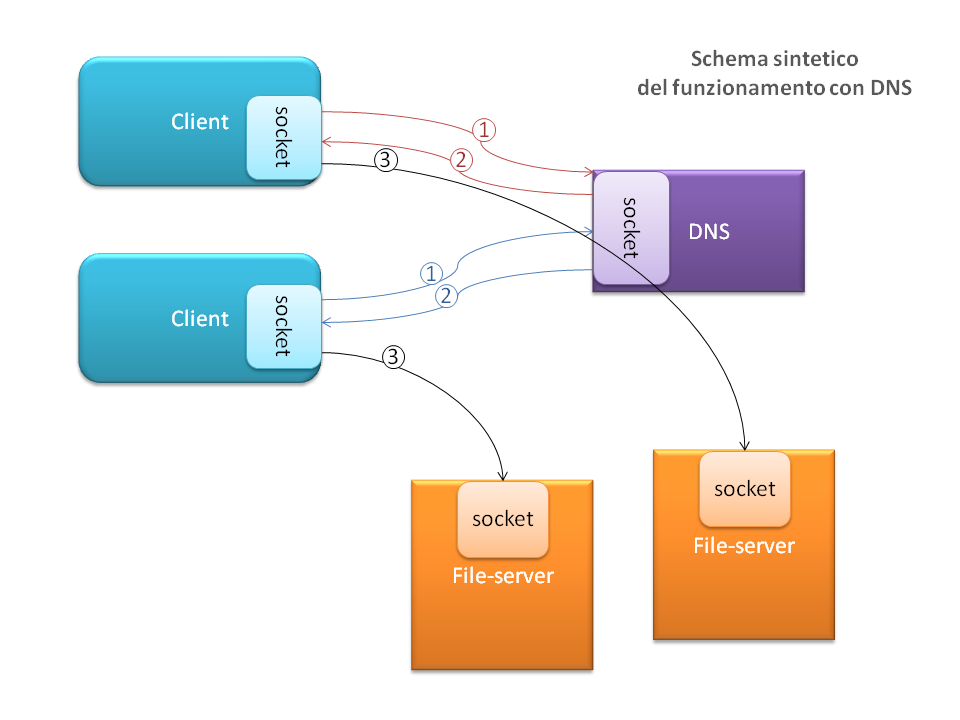


Figura - Architettura generica del sistema distribuito