

---

# File system distribuito transazionale con replicazione

---

Progetto del corso di  
Sistemi Distribuiti  
Prof.ssa Cardellini

---

Alessandro Pacca 0143230  
Marina Dorelli 0127844  
Vienna Codeluppi 0130452

---

## Sommario

1	Introduzione .....	4
2	Analisi del dominio .....	6
2.1	Protocollo per la consistenza.....	6
2.2	Proprietà ACID .....	6
2.3	Mutua esclusione in ambito distribuito .....	6
2.3.1	Algoritmo Ricart-Agrawala .....	7
2.4	Bilanciamento del carico .....	8
2.5	Tolleranza alle failure ed ai guasti .....	9
2.5.1	Failure per il client .....	9
2.5.2	Failure per il server .....	10
3	Implementazione.....	12
3.1	Server.....	12
3.1.1	Caratteristiche principali del server .....	12
3.1.2	Pacchetto Applicativo .....	13
3.2	Servizi offerti dal server.....	15
3.2.1	Servizi offerti ai server: Lista File .....	16
3.2.2	Servizi offerti ai server: Richiesta di commit .....	16
3.2.3	Servizi offerti ai server: Aggiorna File .....	17
3.2.4	Servizi offerti ai server: Copia File .....	18
3.2.5	Scrittura e modifica dei file.....	18
3.2.6	Sincronizzazione del file system .....	20
3.2.7	Servizi offerti ai server: Uscita .....	21
3.3	Realizzazione del DNS.....	21
3.3.1	Caratteristiche principali del DNS.....	21
3.4	Realizzazione del client.....	27
3.5	Implementazione del server.....	28
3.5.1	Trasferimento di file tra un server e un client o tra server e server .....	28

3.5.2	Comunicazione tra processi differenti e Ricart-Agrawala .....	30
3.5.3	Spedizione degli aggiornamenti agli altri server .....	33
3.5.4	Richiesta degli IP al DNS .....	34
3.5.5	Scrittura del file di log.....	35
3.5.6	File di configurazione.....	36
4	Testing .....	38
4.1	Testing senza failure .....	38
4.1.1	Scrittura di un file presso un server e propagazione del suo aggiornamento.....	38
4.1.2	Scritture concorrenti .....	48
4.2	Testing in caso di failure .....	55
4.2.1	Failstop .....	55
4.2.2	Guasti bizantini .....	56
4.2.3	Omissioni .....	57
5	Conclusioni .....	59

# 1 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 sistema distribuito che si propone come deposito 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 round robin da parte del DNS.
- 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.

Il capitolo seguente illustra brevemente l'analisi del dominio relativa alle tecnologie e scelte implementative effettuate, il capitolo 3 illustra la realizzazione effettiva del lavoro e illustra nei dettagli il funzionamento del sistema. Il capitolo 4 è dedicato alla fase di testing ed infine nel capitolo 5 sono illustrate brevemente le conclusioni tratte dal lavoro svolto.

## 2 Analisi del dominio

Di seguito sono introdotte alcune tecnologie adoperate per la realizzazione del sistema e vengono spiegate brevemente le problematiche incontrate durante lo sviluppo del lavoro presentato. Il sistema, infatti, deve essere in grado di fornire alcune garanzie illustrate nei prossimi paragrafi.

### 2.1 Protocollo per la consistenza

Nell'applicazione in questione, un client invia le operazioni di scrittura ad un file-server che poi si preoccupa, una volta ottenuto il consenso inoltra l'aggiornamento ai rimanenti file-server del sistema. Il protocollo di consistenza adoperato, quindi, è di tipo primary-based, dove le operazioni di scrittura vengono eseguite su una sola replica, che successivamente assicura che gli aggiornamenti siano opportunamente ordinati ed inoltrati alle altre repliche.

### 2.2 Proprietà ACID

Il sistema deve essere in grado di fornire operazioni di tipo ACID:

- 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 transazione 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.

### 2.3 Mutua esclusione in ambito distribuito

Quando si hanno processi concorrenti che accedono ad una risorsa condivisa nasce il bisogno di sincronizzarli in modo tale che tale risorsa sia assegnata ad un processo alla volta. Questo problema va sotto il nome di mutua esclusione. Dal punto di vista astratto il problema può essere formulato come segue. Ci sono N processi ognuno dei quali ripete la seguente sequenza di passi di programma

*<non in sezione critica>*

*<trying protocol>*

*sezione critica*

*<exit protocol>*

*<non in sezione critica>*

Una volta uscito dall'exit protocol il processo può rientrare infinite volte nel trying protocol. Questo problema è stato definito per la prima volta da Dijkstra nel 1965, il quale fornisce anche una soluzione (istanza cioè il trying protocol e l'exit protocol) in un modello di sistema concorrente. In un sistema concorrente esiste uno "scheduler" centralizzato che permette ad un solo processo alla volta di entrare in esecuzione e quindi di evolvere secondo il suo codice. Di fatto quindi lo scheduler esegue una linearizzazione di tutte le istruzioni elementari effettuate dai vari processi. La linearizzazione creata dalla singola esecuzione dell'algoritmo è chiamata "schedule".

Il discorso, però, si complica nel caso di processi concorrenti su sistemi distribuiti. Esistono diversi algoritmi per la gestione dell'accesso a risorse condivise e l'algoritmo da noi scelto per questo lavoro è quello di Ricart-Agrawala.

### **2.3.1 Algoritmo Ricart-Agrawala**

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.

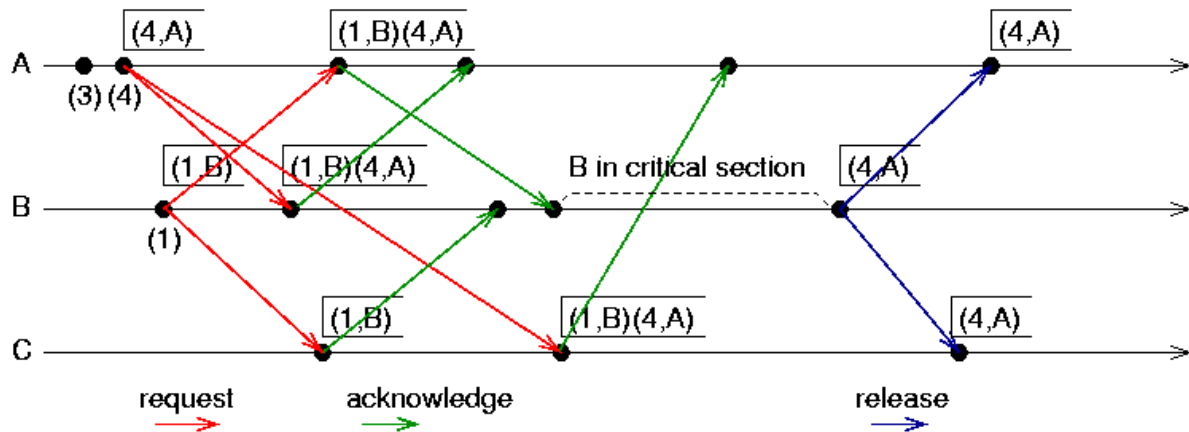


Figura 1 - Esempio di scambio messaggi secondo l'algoritmo di Ricart-Agrawala

## 2.4 Bilanciamento del carico

In base all'algoritmo Round Robin 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.

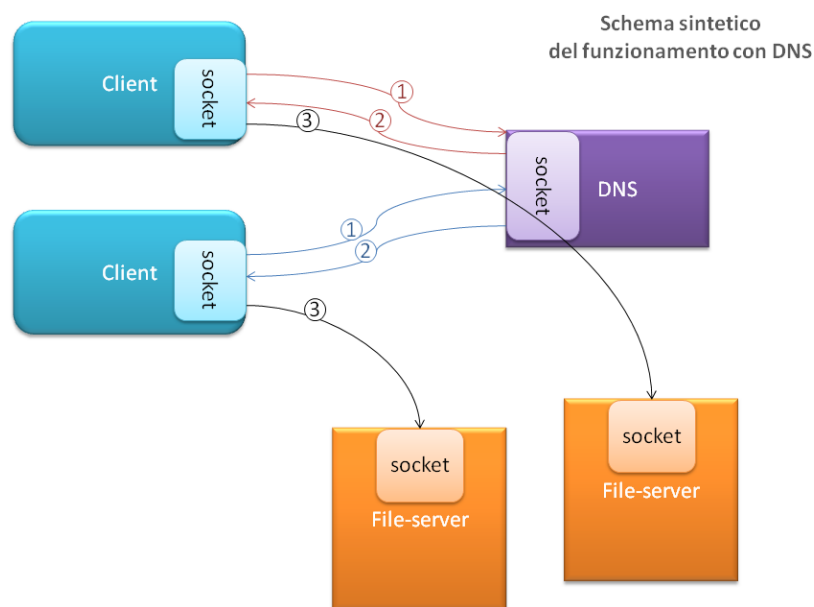


Figura 2 - Architettura generica del sistema distribuito



## 2.5 Tolleranza alle failure ed ai guasti

Il lavoro presentato è in grado di tollerare alcuni tipi di failure. La failure si verifica quando il comportamento di un componente non è conforme alle sue specifiche.

Esse sono di seguito brevemente presentate e verranno illustrato le scelte implementative eseguite per la loro soluzione.

### 2.5.1 Failure per il client

Per quanto riguarda il lato client abbiamo:

- Failure di tipo bizantino – questo tipo di failure sono dovute al malfunzionamento del client, che inizia a rispondere e inviare messaggi in maniera non conforme all’algoritmo specificato. Questo può comportare, ad esempio l’invio di un pacchetto applicativo compilato con campi mancanti o con dati errati. Il sistema deve essere in grado, quindi, di identificare questi problemi e non eseguire commit. Infatti, il server durante le transazioni tra client e server effettua un controllo su quei campi di cui è certo del contenuto. Ad esempio, durante una fase di scambio pacchetti per la modifica file, la transazione tra client e server viene identificata dall’ID pseudo casuale generato, che viene sempre inserito all’interno del pacchetto. Il processo server, quindi, controlla l’ID per verificare che non sia diverso, in questo caso invia un messaggio di errore al client e interrompe la transazione. Un altro campo che viene controllato durante le varie operazioni tra client e server è quello di *TipoOperazione*, se questo non risulta conforme all’operazione che si sta eseguendo o ad una operazione eseguibile, viene restituito un messaggio di errore.



Figura 3 - Failure bizantina

- Omissioni – si verifica quando il client perde alcuni pacchetti. Occorre creare un meccanismo di ritrasmissione per recuperare i pacchetti persi. Nel caso di mancata ricezione di un messaggio il protocollo di trasporto adoperato, TCP, fornisce un meccanismo di ritrasmissione. Può accadere

comunque, che alcuni pacchetti vadano persi o arrivino in ordine invertito. Il client deve supportare una procedura di controllo su questo tipo di failure.

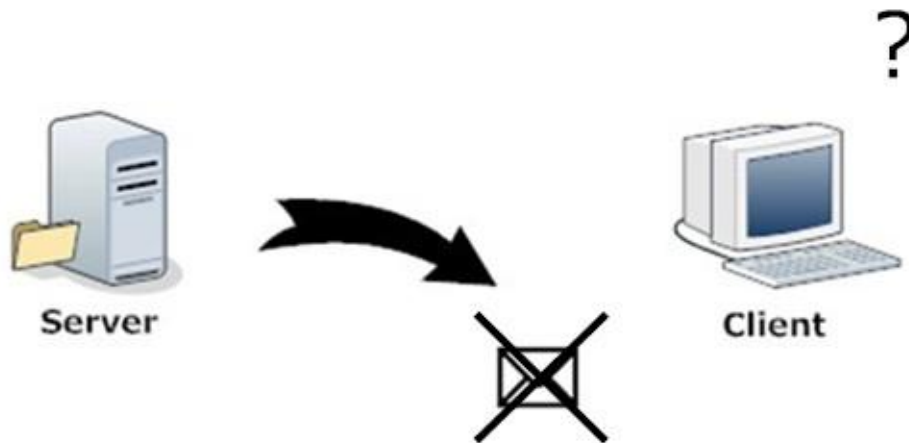


Figura 4 - Omissione del pacchetto

- Failstop – quando si verifica il crash da parte del server il client non deve attenderlo per un tempo indefinito.

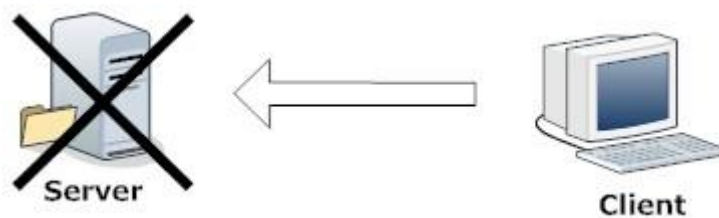


Figura 5 - Crash del server

### 2.5.2 Failure per il server

Il sistema, lato server, deve inoltre essere tollerante ad una failure di tipo:

- Failstop - quando si verifica il crash da parte del client il server non deve attenderlo per un tempo indefinito.

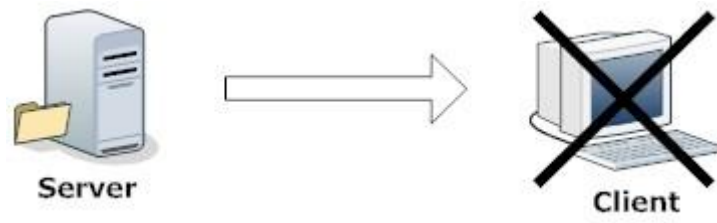


Figura 6 - Crash del client

## 3 Implementazione

### 3.1 Server

#### 3.1.1 Caratteristiche principali del server

Il Server da noi implementato è di tipo non bloccante, perciò è in grado di gestire più richieste provenienti da client o server diversi. Ciò avviene grazie all'utilizzo di un'architettura multi-processo. All'avvio il server crea tre processi: il primo è in attesa di richieste da parte dei Client, il secondo è in attesa di richieste da parte dei server e un terzo processo è utilizzato per attuare l'algoritmo di Ricart-Agrawala (2.3.1.)

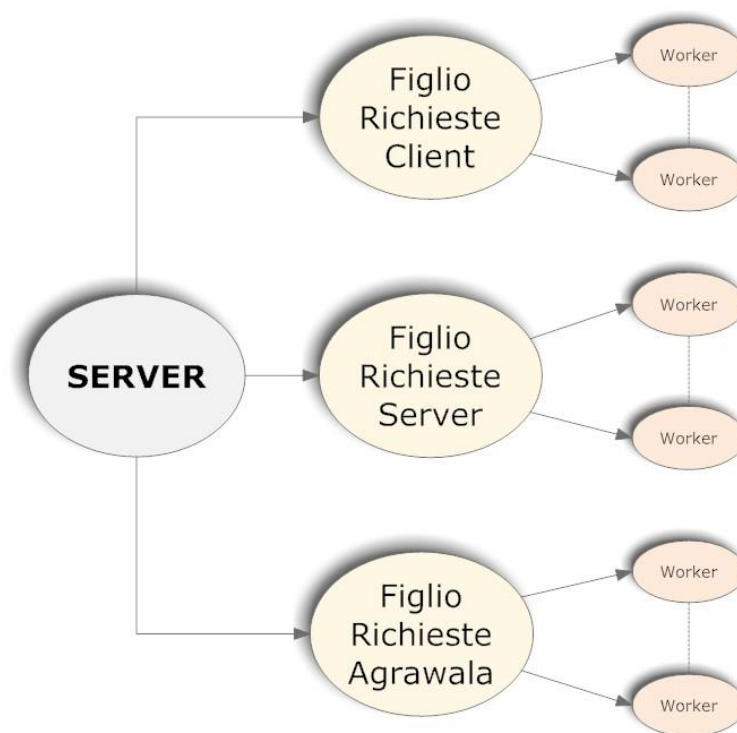


Figura 7 Figli creati dal server durante la sua esecuzione

I processi in attesa di richieste da parte di client o server, creano ognuno un figlio nel momento in cui ne arriva una. Dopo aver creato un figlio, esso si occuperà di gestire la richiesta e il processo padre, che era in ascolto, torna a soddisfare nuovamente le richieste di altri client e a creare se necessario altri figli. Al processo figlio verrà passato il descrittore del socket creato dalla *accept()* e il processo padre, prima di tornare in ascolto, chiude il socket creato dalla *accept()* per risparmiare le risorse del server. Il figlio difatti, essendo un processo, ha una copia di questo socket e lavorerà tramite esso.

Il numero massimo di figli creabili da entrambi i padri in attesa di richieste è 50. Questo per evitare che il server si sovraccarichi di richieste e non riesca a gestirle tutte.

Una delle operazioni effettuate durante l'avvio del server è la sincronizzazione con gli altri server. Durante la fase di avvio verranno contattati gli altri server per ricevere i file presenti sul file system distribuito. In questo modo, sia che si tratti del primo avvio, sia che si tratti di un avvio a seguito di un crash, il server avrà la stessa versione dei file presenti sul file system distribuito

I file che saranno resi disponibili sul file system distribuito sono salvati in una cartella definita nel file di configurazione.

Altri file utilizzati dal server sono:

- Il file di configurazione, *configurazioneServer.cfg*, che contiene una serie di specifiche che permettono di avviare il server secondo una data configurazione.
- Un file di Log, *server Replica.log*, che contiene tutte le operazioni che il server sta svolgendo durante la sua esecuzione.
- Vari file di PIPE utilizzati per implementare l'algoritmo di Ricart-Agrawala. Essi sono salvati nella cartella /tmp di Unix
- Altri file temporanei creati mentre il client sta effettuando l'operazione di scrittura file ma ancora non ha effettuato il commit

Per poter attuare il meccanismo di Ricart-Agrawala e poter fare in modo che ogni processo del server possa conoscere la lista dei file aperti dal server in un dato istante, viene utilizzata la memoria condivisa. Nella memoria condivisa è presente un array che contiene la lista dei file attualmente aperti dal server e in uso. L'utilizzo di questa lista è spiegato nel capitolo 3.2.2.

Il server, riceve e invia gli aggiornamenti relativi ai file presenti nel proprio file system distribuito in modalità push. Nel momento in cui un client effettua una modifica su un file presente nel file system distribuito, essa verrà inviata agli altri server nel momento del commit. Grazie a questa scelta possiamo avere un grado di consistenza molto alto e garantire che i server abbiano la stessa versione del file quasi nello stesso istante.

### 3.1.2 Pacchetto Applicativo

Per permettere lo scambio di dati tra client e server e tra i vari server è stato creato un particolare tipo di pacchetto in modo da ridurre il numero di messaggi scambiati tra i vari client e server e avere con un solo

pacchetto tutta una serie di informazioni utili all'esecuzione di una data operazione. Il pacchetto creato è così composto:

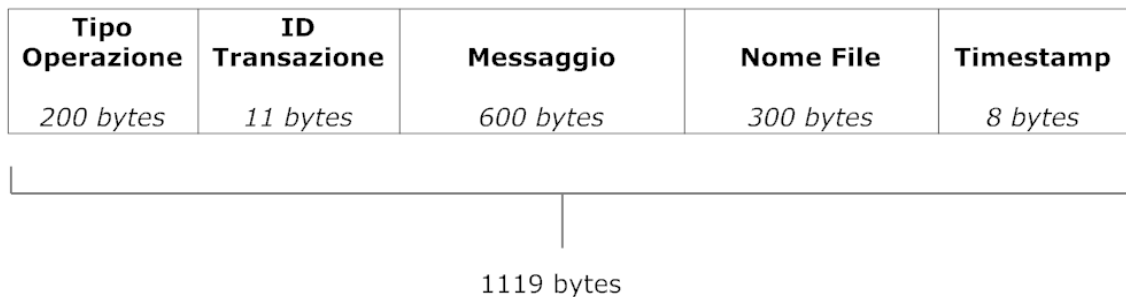


Figura 8 – Struttura del pacchetto applicativo

I campi sono utilizzati nel seguente modo:

- *Tipo Operazione (200 bytes)*: Contiene una stringa con il tipo di operazione che dovrà essere effettuata dal server o dal client. Può contenere sia richieste che conferme da parte di entrambi le parti. E' il primo campo che viene analizzato per verificare il tipo di operazione da attuare da parte del client o del server. E' usato anche per controllare che il comportamento del client o del server non sia di tipo bizantino.
- *ID Transazione (11 bytes)*: Contiene un ID transazione generato pseudocasualmente durante la copia di un file o l'esecuzione di una transazione da parte di un client. Nel caso in cui l'operazione richiesta non necessiti di un ID transazione (ad esempio, nel caso di operazione di uscita), questo campo può essere vuoto. È, inoltre, utilizzato per controlli su failure di tipo bizantino.
- *Messaggio (500 bytes)*: Contiene il messaggio complementare al tipo di operazione richiesta. E' il corpo del pacchetto. A seconda del tipo di operazione in atto può contenere sequenze di byte (per la copia di file), o stringhe di testo. Se non utilizzato può essere vuoto.
- *Nome File (350 bytes)*: Contiene l'eventuale nome del file che dovrà essere scritto o letto a seconda del tipo di operazione richiesta. Se non utilizzato può essere vuoto.
- *Timestamp (8 bytes)*: Contiene un timestamp associato al tipo di operazione in atto. Il timestamp è generato numerando i pacchetti scambiati in modo sequenziale.

### 3.2 Servizi offerti dal server

Per poter realizzare un file system distribuito il server fornisce vari servizi sia ai client sia agli altri server facenti parte del sistema distribuito. Durante l'esecuzione, il server attende le richieste su due porte diverse, una dedicata ai server e l'altra dedicata alle richieste provenienti dai client.

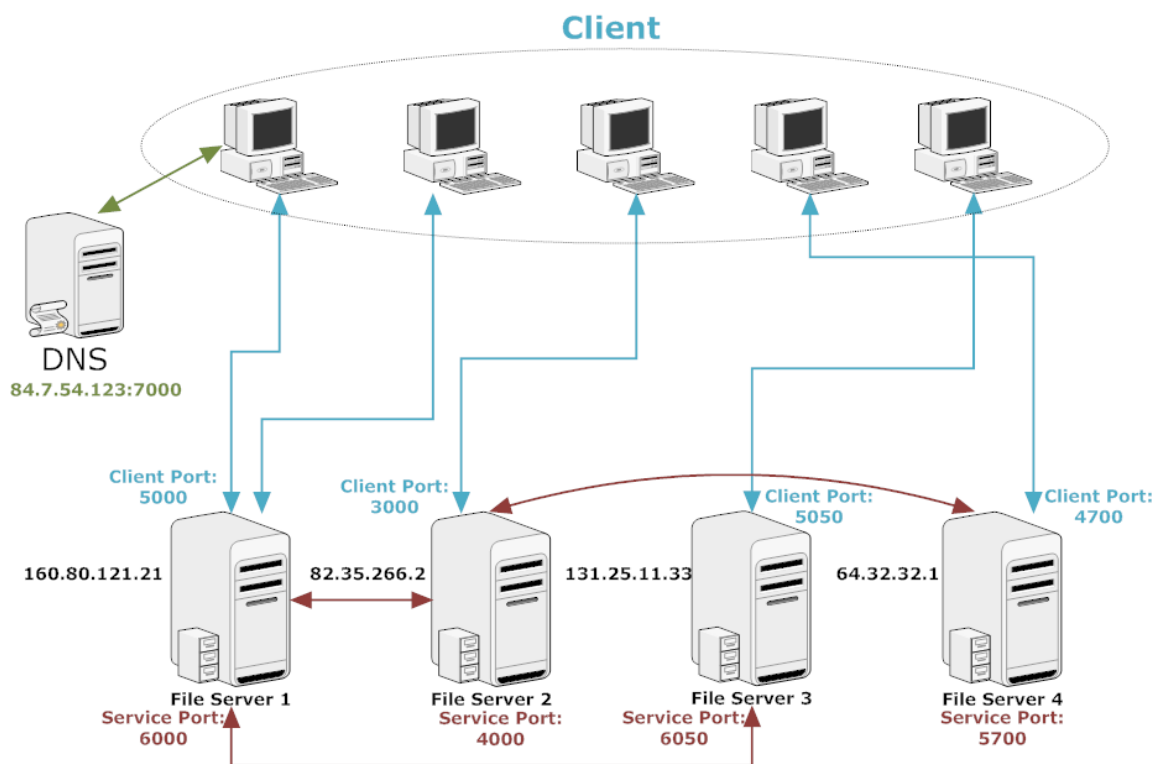


Figura 9 Struttura del file system distribuito

Le operazioni invocabili e rese disponibili al client sono:

- *Lista File*: Il server invia al client la lista dei file presenti sul file system distribuito.
- *Leggi file*: Il server invia al client un intero file presente sul file system distribuito.
- *Scrivi file*: Il server permette al client di effettuare una sequenza di operazioni di scrittura su di un file che sarà visibile all'interno del file system distribuito solo al termine della sequenza di operazioni e tramite l'operazione di commit
- *Uscita*: Il server termina la connessione con il client

Le operazioni invocabili e rese disponibili invece agli altri server sono:

- *Lista File*: Come nel caso del client, invia ad un altro server la lista dei file presenti sul file system distribuito
- *Richiesta di commit*: Questa operazione permette, ad un altro server, di sapere se può effettuare il commit in base all'algoritmo di Ricart-Agrawala.
- *Aggiorna file*: Permette al server aggiornare un file presente sul file system distribuito.
- *Copia File*: Invia un file ad un altro server. Il file è spedito per intero.
- *Uscita*: Permettere di chiudere la connessione con il server

Il funzionamento di ogni operazione è spiegato dettagliatamente nei paragrafi successivi.

La porta di ascolto per i server è calcolata prendendo il numero di porta che i client possono contattare e aggiungendo mille unità. Ad esempio, se il server è pronto a servire i client sulla porta 5000, la porta di ascolto per gli altri server, sarà 6000. In questo modo la configurazione del server sarà più semplice da realizzare e vedremo, nel capitolo 3.3, che grazie a questa soluzione il DNS fornirà una sola porta di ascolto, quella dei client. La porta di servizio, se necessaria, potrà essere calcolata a tempo di runtime dal server.

### 3.2.1 Servizi offerti ai server: Lista File

Questa operazione permette di inviare la lista completa dei file presenti sul file system distribuito, ad altri server.

Il figlio di servizio che ha preso in consegna la richiesta da parte di un altro server, per poter inviare la lista file, dovrà ricevere un pacchetto applicativo in cui il tipo operazione sia uguale a "*lista file*". Dopo aver effettuato questo controllo, provvederà ad effettuare una scansione della cartella locale dove il server contiene i file presenti nel file system distribuito. Preparerà, quindi, un pacchetto applicativo con il tipo di operazione sempre di tipo "*lista file*" e all'interno, nel campo *messaggio*, inserirà la lista dei file. Se la dimensione della lista dei file supera i 500byte, verranno spediti più pacchetti applicativi.

### 3.2.2 Servizi offerti ai server: Richiesta di commit

Tramite questa operazione, un qualsiasi altro server richiede al server se può effettuare il commit in base alla precedenza assegnata tramite il meccanismo di Ricart-Agrawala.

In questo paragrafo, il server che riceve la richiesta e che deve fornire il servizio verrà chiamato server A mentre colui che effettua la richiesta verrà chiamato server B.

Il figlio di servizio del server A che ha preso in consegna la richiesta da parte del server B controllerà che il tipo di operazione ricevuto sia "*chiedo di fare commit*". Il pacchetto applicativo ricevuto conterrà inoltre



anche il nome del file di cui il server B intende effettuare il commit e l'ID. Il server A procederà a controllare nella memoria condivisa, che il file non sia presente nell'array della lista dei file aperti. Le possibili situazioni che si possono verificare sono due:

- Se il file non è presente nella lista dei file aperti, il server A invierà al server B un pacchetto applicativo con il tipo operazione uguale a *“conferma per il commit”* e nel messaggio inserirà *“ok”*. Se il file non è presente nell'array difatti, siamo nella situazione in cui il server A non ha nessun client che sta effettuando operazioni di scrittura su quel file e quindi può inviare senza problemi la conferma al server B.
- Nel caso in cui invece, il file è presente nella lista dei file aperti, il server A deve controllare chi ha la precedenza per effettuare il commit. In questo caso la precedenza per il commit è data al server che ha l'ID più basso. Possiamo quindi distinguere due ulteriori situazioni possibili:
  - Il server A ha ID minore del server B: In questo caso non invierà nessuna conferma al server B che rimarrà in attesa di una risposta. La risposta verrà difatti inviata solo nel momento in cui il server A vedrà sparire dall'array della lista dei file, il file di cui il server B intende effettuare il commit. Il server A difatti, avendo ID minore, ha la precedenza sull'effettuare il commit rispetto al server B. Una volta effettuato il commit può sbloccare il server B e permettergli di effettuare il commit.
  - Il server A ha ID maggiore del server B: In questo caso il server A spedirà subito una conferma al server B in quanto B ha la precedenza nell'effettuare il commit. Nel momento in cui A dovrà effettuare il suo commit, procederà ad attuare il meccanismo di Agrawala.

Si ricorda che, secondo l'algoritmo di Agrawala, il server B per poter effettuare il commit non deve ricevere la conferma solo dal server A ma da tutti i server facenti parte del file system distribuito.

### 3.2.3 Servizi offerti ai server: Aggiorna File

Questa operazione effettua l'aggiornamento di un file presente nella cartella del file system distribuito. Una volta che un server ha ricevuto la conferma per il commit, provvederà ad aggiornare il proprio file locale e spedirà l'aggiornamento agli altri server.

In questo paragrafo il server che vuole propagare l'aggiornamento è chiamato server A mentre il server che riceve la richiesta di aggiornamento è chiamato server B.

Il server A invierà, perciò, a tutti gli altri server, un pacchetto applicativo con tipo operazione uguale a *“aggiorna file”*, un ID transazione generato pseudo casualmente e, nel campo *nomefile*, il nome del file che dovrà essere aggiornato. Il server A effettua l'aggiornamento, inviando al server B solamente i dati

aggiuntivi e non tutto il contenuto del file. Questo onde evitare uno scambio di dati troppo elevato. Scendiamo ora nei dettagli. Una volta che il server B riceve una richiesta di aggiornamento file, creerà un file temporaneo il cui nome sarà uguale all'ID transazione. Il file temporaneo è creato nella cartella /tmp di Unix.

Una volta creato il file temporaneo, il server B avvisa il server A che è pronto a ricevere il file con i dati da aggiornare. Il server A procederà, perciò, a inviare al server B il file(3.2.4). Una volta ricevuto il file per intero, il server B procederà ad apporre i nuovi dati nel file da aggiornare. La scrittura del file è eseguita effettuando un lock sul file. Questo per evitare che sul server B ci sia un altro client che effettui un'operazione di commit nello stesso istante sullo stesso file.

### 3.2.4 Servizi offerti ai server: Copia File

Questa operazione permette ad un server di inviare un file ad un altro server. In questo paragrafo, il server che effettua la richiesta di un file è il server A, mentre colui che riceve la richiesta e che dovrà spedire il file è il server B. Il server A spedisce al server B un pacchetto applicativo, sulla porta di servizio, con il tipo operazione settato a *"copia file"*. All'interno del campo *nomefile*, inserirà il nome del file che vuole ricevere e nel campo *IDtransazione*, inserirà un ID generato pseudo casualmente. Il server B controllerà che il file richiesto dal server A esista e procederà ad inviare al server A un pacchetto applicativo con tipo operazione settata a *"copia file, pronto"*. In questo modo avvisa A che il file esiste e che B è pronto a inviarlo. Il server A risponderà con un pacchetto applicativo il cui tipo operazione è settata a *"copia file, pronto a ricevere"* che serve ad informare B che può cominciare a inviare il file ad A. Terminato l'invio del file, il server B si rimetterà in ascolto di una nuova eventuale operazione da parte di A. Nel caso in cui A non sia interessato a effettuare una nuova operazione, la connessione sarà chiusa tramite l'operazione *"Uscita"*. Se il file richiesto da A non esiste, verrà inviato un pacchetto applicativo con tipo operazione *"copia file, non trovato"* in modo tale da informare A della non presenza del file. Il server B si rimetterà in ascolto di eventuali altre richieste da parte di A.

### 3.2.5 Scrittura e modifica dei file

Una delle funzionalità offerte dal sistema realizzato consiste nella possibilità di scrivere file modificandone il contenuto.

L'utente dopo aver fatto una richiesta di scrittura può accedere ad un file esistente, oppure crearne uno nuovo se questo non esiste e sottomette al server le modifiche che desidera apportare.

Una volta finita l'operazione l'utente attraverso la digitazione del comando "*commit*" sul terminale ordina alla macchina di salvare il contenuto. A questo punto la macchina remota si occuperà di effettuare le modifiche e invierà un messaggio di conferma all'utente, preparandosi poi a ricevere una nuova richiesta di operazione da effettuare. Se, invece, l'utente decidesse di annullare le modifiche scritte, esse verranno eliminate senza lasciare traccia sui file contenuti nel file system. Anche in questo caso la macchina confermerà di aver annullato l'operazione intrapresa e sarà pronta per una nuova operazione.

Per poter richiamare questa funzionalità l'utente deve semplicemente inviare una richiesta di operazione "*scrivi file*", digitandola a console. Il client, riconosciuta l'operazione che si intende effettuare, si occupa di chiedere anche il nome del file che si vuole modificare o scrivere, dopodiché sottomette queste informazioni e invia i dati necessari attraverso il pacchetto applicativo, definito precedentemente, rispettivamente nei campi di *Tipo operazione*, inserendo la stringa "*scrivi file*", e nel campo *Nome File* dove invece viene inserito il nome digitato dall'utente comprensivo di estensione.

In questo modo il server dispone delle informazioni necessarie per poter richiamare la funzione associata, *richiestaScritturaFile()*. In primo luogo viene creato un file temporaneo che conterrà le aggiunte apportate dall'utente, poi viene ricercato il file selezionato, o creato, a seconda delle condizioni sopra descritte. Una volta preparati gli elementi necessari alla memorizzazione delle modifiche, viene creato un ID pseudocasuale che permette di identificare lo scambio di messaggi durante le varie fasi.

Il server invia all'utente il messaggio di notifica che è pronto a ricevere, attraverso il pacchetto applicativo. Il client e il server si scambiano questi pacchetti contenenti nel campo messaggi le modifiche che si vogliono effettuare. Il pacchetto viene inizializzato con i dati relativi all'ID di transazione, al tipo di operazione e ai messaggi inviati. Si è scelto di inviare di volta in volta le modifiche che l'utente scrive sul terminale e memorizzarle, in modo da nascondere un'eventuale latenza dovuto all'invio di messaggi di testo troppo grandi. Quando l'utente digita il comando di sottomissione delle variazioni effettuate, esse sono in realtà già state inviate e pronte per essere memorizzate.

Quindi, lo scambio di messaggi continua inserendo di volta in volta le modifiche nel file temporaneo fino a quando l'utente non sottomette un messaggio contenente il comando "*commit*" od il comando "*abort*".

1. Nel primo caso la macchina deve rendere effettive le modifiche, prima che ciò avvenga però è necessario che il file in uso non sia in conflitto con altre macchine (ossia nessuno voglia modificare lo stesso file). Per questo motivo viene inviata una richiesta di permesso in scrittura del file a tutte le altre macchine presenti nel sistema.

Ogni server ospita più processi, quindi occorre tenere in considerazione l'eventualità in cui due utenti diversi, serviti dallo stesso nodo, stiano entrambi lavorando su uno stesso file.

A tale scopo tutti i processi gestiscono una lista di file che sono in uso dalla macchina, ogni processo in esecuzione utilizza un meccanismo di comunicazione tra processi ad area di memoria condivisa dove vengono inseriti i nomi dei file. Dopo aver inizializzato questo array di stringhe, se si verifica un conflitto sulla scrittura di un file in locale viene effettuato un accesso in scrittura atomico, mediante funzioni di locking delle librerie POSIX, in modo da garantire mutua esclusione per le modifiche. Una volta completata questa procedura, viene avviata una richiesta di scrittura verso tutte le altre macchine del sistema. I meccanismi che regolano la sincronizzazione tra i vari processi di modifiche sono trattati in un'altra sezione [3.2.2].

Nel caso in cui il server che riceve la richiesta non stia lavorando sullo stesso file invia un messaggio di ok al richiedente, altrimenti vengono innescati degli algoritmi che permettano la scrittura in maniera ordinata del file in base all'algoritmo di Ricart-Agrawala (che viene descritta in maniera più completa e dettagliata nel paragrafo apposito).

L'algoritmo di Ricart-Agrawala, quindi, permette al server di effettuare le operazioni di scrittura in modalità atomica. Una volta ricevute le conferme da parte di tutti gli altri server, il file temporaneo viene reso permanente e cancellata la vecchia versione, dopodiché il file aggiornato viene inviato a tutte le altre macchine, in modo da non avere incoerenze nel file system.

Una volta terminato il processo il server invia una conferma dell'avvenuta operazione e l'utente sarà in grado di procedere con nuove operazioni.

2. Nel caso di abort, invece, il server si occuperà semplicemente di cancellare ogni traccia del file temporaneo contenenti tutte le modifiche inserite fino a quel momento, ed invia un messaggio di conferma di annullamento al client.

### 3.2.6 Sincronizzazione del file system

Nel momento in cui un server viene avviato, in prima istanza deve sincronizzare il contenuto del suo file system con quello delle macchine già presenti, in modo che, dopo la fase di avvio, esso si trovi in uno stato coerente e aggiornato. Quindi, per prima cosa richiede al DNS gli indirizzi degli altri server e si mette in contatto con un altro server attivo al quale richiede tutti i dati presenti nel suo file system e li copia nella sua cartella locale.

Si è deciso di operare questa strategia perchè nonostante possa diventare oneroso con grandi quantità di file, questo tipo di operazione avviene solo nella fase di avvio, durante l'esecuzione dei processi avverranno solo degli aggiornamenti e questo giustifica l'onere iniziale del trasferimento dei vari file. In alternativa, per ottimizzare questo processo, il server avrebbe dovuto richiedere agli altri server solo gli aggiornamenti dei file effettivamente modificati rispetto a quelli presenti nel proprio file system locale, durante il periodo di inattività.

Se nessun server è attivo la sincronizzazione non avviene, altrimenti, dopo essersi connesso ad un'altra macchina, invia un messaggio di richiesta di aggiornamento attraverso il pacchetto applicativo. A questo punto il server contattato si occupa di inviare tutti i file presenti nella sua memoria locale. Una volta conclusa l'operazione la comunicazione viene terminata.

### 3.2.7 Servizi offerti ai server: Uscita

Questa operazione permette ad un server (A) di chiudere la connessione stabilita con il server (B). Il server A invierà un pacchetto applicativo con tipo operazione settato a *"Uscita"*. Il server B manderà un messaggio di conferma ad A con il tipo operazione settata a *"arrivederci"*. Dopodiché procederà a terminare il figlio che serviva le richieste provenienti da A. Il server A, ricevuto il messaggio di conferma chiusura, procederà anche lui a chiudere la connessione e ad effettuare la terminazione del figlio che connesso al server B.

## 3.3 Realizzazione del DNS

### 3.3.1 Caratteristiche principali del DNS

Un DNS (Domain Name System) è un sistema gerarchico e distribuito, utilizzato in ambiente TCP/IP per creare una corrispondenza tra un indirizzo numerico ed un nome leggibile e memorizzabile a livello umano, da assegnare ad una macchina.

Il DNS che opera all'interno del sistema realizzato, è un'entità unica e centralizzata, che serve le richieste dei client assegnando loro l'indirizzo IP di uno dei file-server da contattare. Tale scelta è stata possibile tenendo conto delle ridotte dimensioni della rete in termini di numero di client e server che la compongono. In reti di vaste dimensioni avrebbe altrimenti rappresentato un possibile collo di bottiglia per il sistema, in quanto tutti i client si sarebbero rivolti ad una sola macchina, causandone un overflow di richieste. Per fronteggiare un eventuale crash del server DNS, questo elemento può essere facilmente replicato.

I client conoscono a priori l'indirizzo del DNS. Il flusso di lavoro è composto dai seguenti passi:

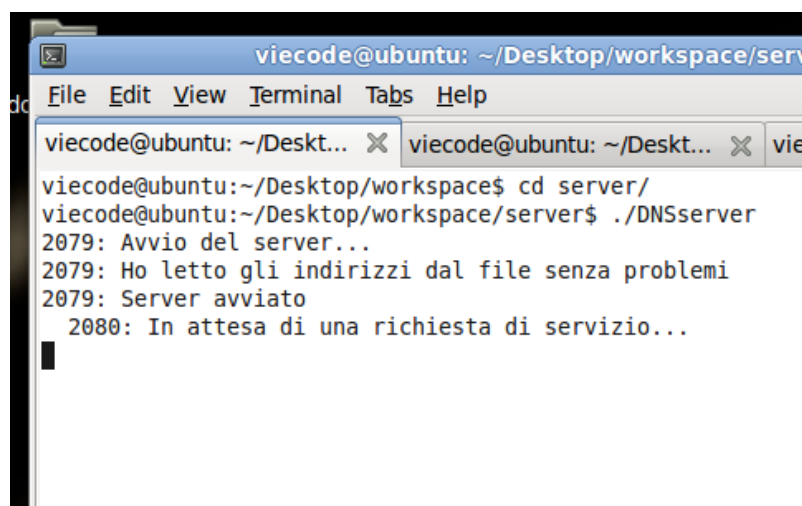
1. Definisce una porta di servizio con quale si pone in attesa delle richieste dei server replica (i file-server)
2. Sono dichiarati due socket: uno, detto di servizio, per stabilire una comunicazione con i server replica, uno detto *normale* per i client.
3. Prepara una struttura dati locale per memorizzare la lista di server replica al suo interno

4. Preleva l'elenco di indirizzi dei server da un file di configurazione testuale e li salva nella struttura locale
5. Inizializza il socket di servizio e si pone in attesa di richieste da parte dei server.
6. Utilizza la funzione fork per creare dei server DNS figli in modo da poter gestire in parallelo ulteriori richieste di servizio.
7. In caso di errore lo segnala e si arresta.
8. Inizializza una seconda struttura socket per ascoltare le richieste dei client.
9. Crea, sempre con il metodo della fork, nuovi eventuali DNS figli in caso di ulteriori richieste dei client

Ad ogni richiesta ricevuta da un client legge un indirizzo IP di un server replica dopo aver fatto partire, un'unica volta, un algoritmo ad anello per assegnare le repliche in modo sequenziale circolare

Può essere terminato da console utilizzando la combinazione CTRL+C, operazione che porterà alla terminazione da parte del DNS di tutti i suoi processi figli attualmente in corso.

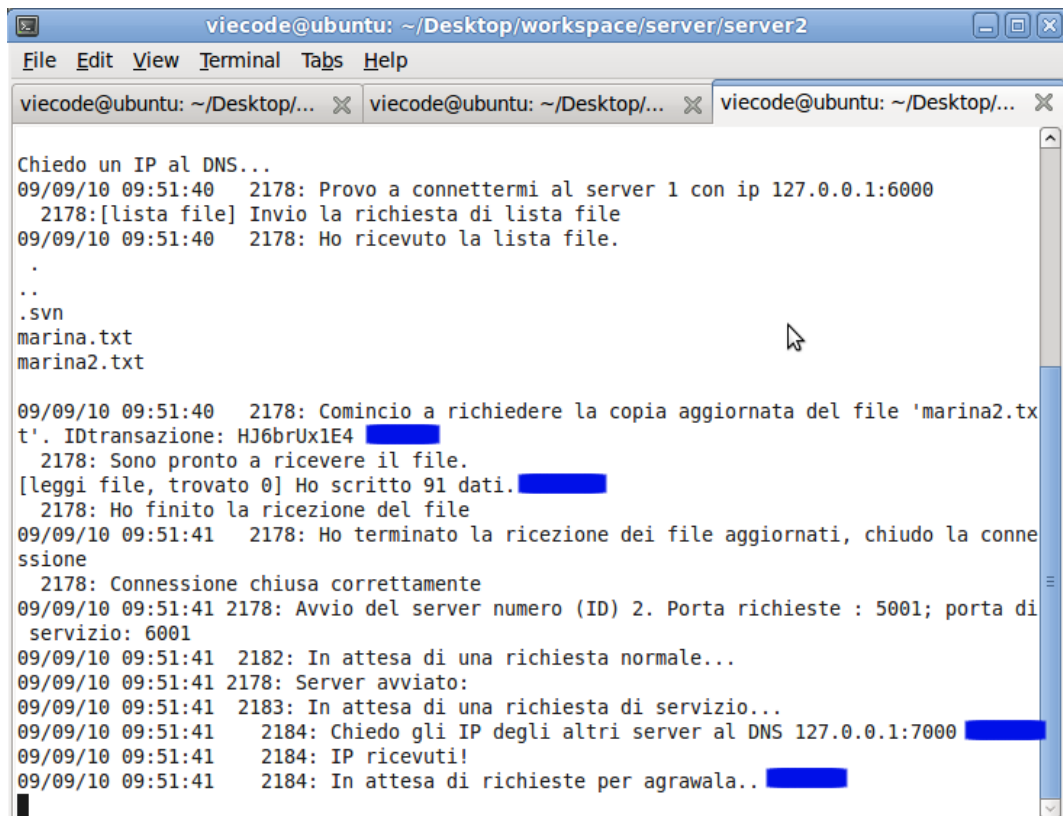
Per avviare l'esecuzione del DNS, come primo passo occorre lanciare l'eseguibile DNSServer da terminale. Questo a sua volta aprirà un file in cui sono registrati i server-file replica del sistema e le loro rispettive porte di ascolto. Una volta fatto ciò, verrà stampato a schermo un messaggio di successo (o di arresto in caso di errore) seguito dalla conferma di avvio del DNS.



```
viocode@ubuntu: ~/Desktop/workspace/server
File Edit View Terminal Tabs Help
viocode@ubuntu: ~/Desktop... X viocode@ubuntu: ~/Desktop... X vie
viocode@ubuntu:~/Desktop/workspace$ cd server/
viocode@ubuntu:~/Desktop/workspace/server$ ./DNSServer
2079: Avvio del server...
2079: Ho letto gli indirizzi dal file senza problemi
2079: Server avviato
2080: In attesa di una richiesta di servizio...
```

Figura 10 - Avvio del DNS

Nel frattempo, i vari server replica sono avviati dalle loro postazioni. Contatteranno il DNS per ricevere la lista aggiornata di file-server (i rettangoli blu nelle figure evidenziano i passi fondamentali).



```
viocode@ubuntu: ~/Desktop/workspace/server/server2
File Edit View Terminal Tabs Help
viocode@ubuntu: ~/Desktop/... X viocode@ubuntu: ~/Desktop/... X viocode@ubuntu: ~/Desktop/... X

Chiedo un IP al DNS...
09/09/10 09:51:40 2178: Provo a connettermi al server 1 con ip 127.0.0.1:6000
2178:[lista file] Invio la richiesta di lista file
09/09/10 09:51:40 2178: Ho ricevuto la lista file.
.
..
.svn
marina.txt
marina2.txt

09/09/10 09:51:40 2178: Comincio a richiedere la copia aggiornata del file 'marina2.txt'. IDtransazione: HJ6brUx1E4 [redacted]
2178: Sono pronto a ricevere il file.
[leggi file, trovato 0] Ho scritto 91 dati. [redacted]
2178: Ho finito la ricezione del file
09/09/10 09:51:41 2178: Ho terminato la ricezione dei file aggiornati, chiudo la connessione
2178: Connessione chiusa correttamente
09/09/10 09:51:41 2178: Avvio del server numero (ID) 2. Porta richieste : 5001; porta di servizio: 6001
09/09/10 09:51:41 2182: In attesa di una richiesta normale...
09/09/10 09:51:41 2178: Server avviato:
09/09/10 09:51:41 2183: In attesa di una richiesta di servizio...
09/09/10 09:51:41 2184: Chiedo gli IP degli altri server al DNS 127.0.0.1:7000 [redacted]
09/09/10 09:51:41 2184: IP ricevuti!
09/09/10 09:51:41 2184: In attesa di richieste per agrawala.. [redacted]
```

Figura 11 - Il server replica riceve la lista degli IP dal DNS

Il DNS risponde dunque alle richieste dei server replica (figura in basso) tramite la porta di servizio (la porta 7000, configurabile da file), inviando la lista degli IP dei server registrati nel sistema.

```
viecode@ubuntu: ~/Desktop/workspace/server
File Edit View Terminal Tabs Help
viecode@ubuntu: ~/Desktop/... X viecode@ubuntu: ~/Desktop/... X viecode@ubuntu: ~/Desktop/... X
2122: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1 : 47636. Elaboro la richiesta di servizio...
2122: Invio la lista degli indirizzi
2122: Ho letto gli indirizzi dal file senza problemi
2122: Richiesta elaborata!

2080: Creazione di un figlio di servizio in corso...
2179: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1 : 48159. Elaboro la richiesta di servizio...
2179: Invio l'IP 127.0.0.1:5002:3

2179: Richiesta elaborata!

2080: Creazione di un figlio di servizio in corso...
2180: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1 : 48161. Elaboro la richiesta di servizio...
2180: Invio l'IP 127.0.0.1:5000:1

2180: Richiesta elaborata!

2080: Creazione di un figlio di servizio in corso...
2185: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1 : 48163. Elaboro la richiesta di servizio...
2185: Invio la lista degli indirizzi
2185: Ho letto gli indirizzi dal file senza problemi
2185: Richiesta elaborata!
```

Figura 12 - Il DNS invia la lista degli IP ad un server replica

Dall'altra parte, i vari client, avviati lanciando l'eseguibile client, contatteranno anch'essi il DNS per ricevere un indirizzo IP di una replica cui richiedere i file di testo. Nella figura in basso è possibile vedere come, quando il client riceve un indirizzo, se questo punta ad un server replica spento il client ricontatta il DNS, che fornirà al client l'IP di un secondo server replica su cui potrà provare ad effettuare una nuova richiesta di connessione.

```
viecode@ubuntu: ~/Desktop/workspace/client
File Edit View Terminal Help
viecode@ubuntu:~/Desktop/workspace$ cd client/
viecode@ubuntu:~/Desktop/workspace/client$ ./client

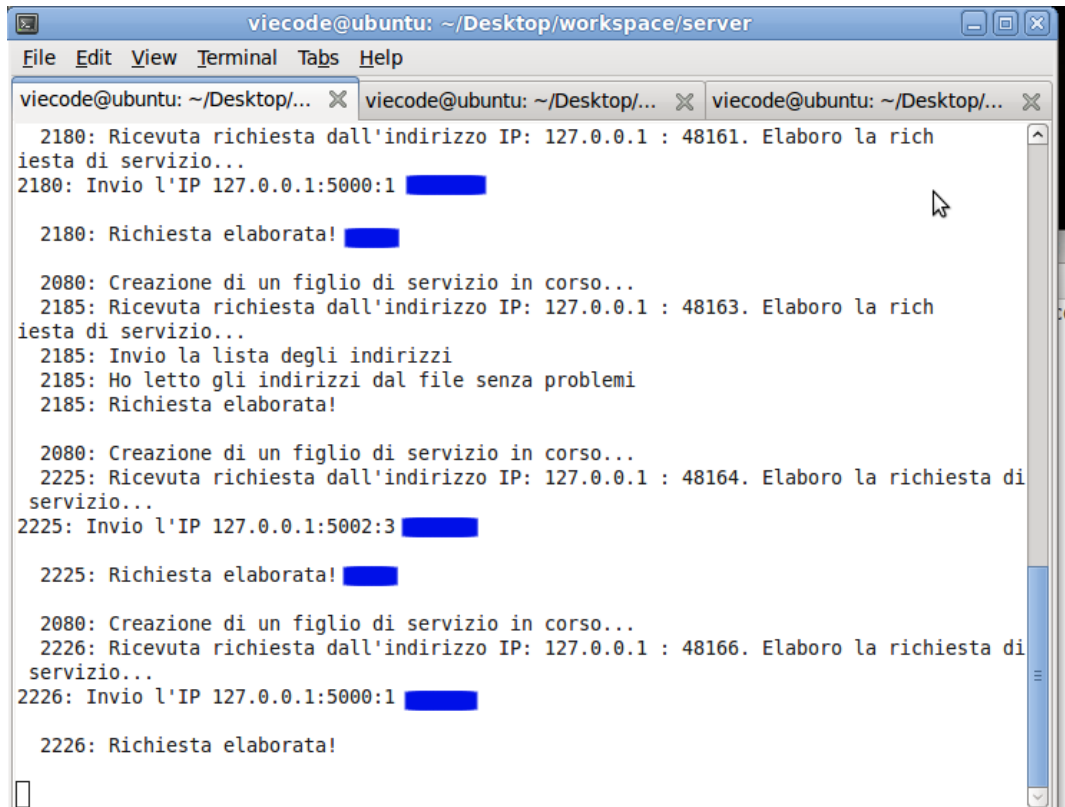
Chiedo un IP al DNS...
Provo a connettermi al server 3 con ip 127.0.0.1:5002
Errore nell'apertura della connessione: Connection refused

Chiedo un IP al DNS...
Provo a connettermi al server 1 con ip 127.0.0.1:5000
Operazione da eseguire:
```

Figura 13 - Il client chiede più volte un IP al DNS



Il comportamento senza errori del DNS alle richieste “normali” dei client è evidenziato in blu. Vengono creati dei processi figli che si occupano di inoltrare un indirizzo per ciascuna richiesta, il tutto secondo un algoritmo ad anello circolare.



```
viocode@ubuntu: ~/Desktop/workspace/server
File Edit View Terminal Tabs Help
viocode@ubuntu: ~/Desktop/... X viocode@ubuntu: ~/Desktop/... X viocode@ubuntu: ~/Desktop/... X
2180: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1 : 48161. Elaboro la rich
iesta di servizio...
2180: Invio l'IP 127.0.0.1:5000:1 [redacted]
2180: Richiesta elaborata! [redacted]
2080: Creazione di un figlio di servizio in corso...
2185: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1 : 48163. Elaboro la rich
iesta di servizio...
2185: Invio la lista degli indirizzi
2185: Ho letto gli indirizzi dal file senza problemi
2185: Richiesta elaborata!
2080: Creazione di un figlio di servizio in corso...
2225: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1 : 48164. Elaboro la richiesta di
servizio...
2225: Invio l'IP 127.0.0.1:5002:3 [redacted]
2225: Richiesta elaborata! [redacted]
2080: Creazione di un figlio di servizio in corso...
2226: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1 : 48166. Elaboro la richiesta di
servizio...
2226: Invio l'IP 127.0.0.1:5000:1 [redacted]
2226: Richiesta elaborata!
```

Figura 14 - IL DNS invia gli IP ai client

Nel caso in cui il client è avviato, ma trova il DNS fuori servizio (nella finestra in background, a sinistra, è stato arrestato da terminale di proposito), esso si arresterà e sarà necessario lanciarlo in un secondo momento, per vedere se il DNS è in funzione. .

```
2080: Creazione di un figlio di servizio in corso...
2185: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1 :
iesta di servizio...
2185: Invio la lista degli indirizzi
2185: Ho letto gli indirizzi dal file senza problemi
2185: Richiesta elaborata!

2080: Creazione di un figlio di servizio in corso...
2225: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1 :
servizio...
2225: Invio l'IP 127.0.0.1:5002:3

2225: Richiesta elaborata!

2080: Creazione di un figlio di servizio in corso...
2226: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1 :
servizio...
2226: Invio l'IP 127.0.0.1:5000:1

2226: Richiesta elaborata!

^C2079: Il server è stato terminato da console
viocode@ubuntu:~/Desktop/workspace/server$
```

```
viocode@ubuntu: ~/Desktop/workspace/client
File Edit View Terminal Help
viocode@ubuntu:~/Desktop/workspace/client$ ./client
Errore nell'apertura della connessione: Connection refused

Chiedo un IP al DNS...
viocode@ubuntu:~/Desktop/workspace/client$
```

Figura 15 - Il client non trova il DNS attivo

Se sono inattivi tutti i server replica, il client contatterà il DNS un numero di volte pari a quanti sono i server nella rete. Non trovando alcun server, esso si arresterà ed anche in questo caso, come nel caso di DNS inattivo, sarà necessario che l'utente lo lanci nuovamente.

```
viocode@ubuntu: ~/Desktop/workspace/client
File Edit View Terminal Help
viocode@ubuntu:~/Desktop/workspace/client$ ./client
Errore nell'apertura della connessione: Connection refused

Chiedo un IP al DNS...
viocode@ubuntu:~/Desktop/workspace/client$ ./client

Chiedo un IP al DNS...
Provo a connettermi al server 2 con ip 127.0.0.1:5001
Errore nell'apertura della connessione: Connection refused

Chiedo un IP al DNS...
Provo a connettermi al server 3 con ip 127.0.0.1:5002
Errore nell'apertura della connessione: Connection refused

Chiedo un IP al DNS...
Provo a connettermi al server 1 con ip 127.0.0.1:5000
Errore nell'apertura della connessione: Connection refused
Non risulta nessun server attivo! :(
viocode@ubuntu:~/Desktop/workspace/client$
```

Figura 16 - Il client non riesce a connettersi a nessun server ricevuto dal DNS

Il funzionamento del DNS in questo caso è continuo. Esso continuerà a servire i client indipendentemente dal fatto che i server registrati nel sistema siano attivi o meno.

### 3.4 Realizzazione del client

Il client, una volta avviato, contatta il DNS, che si occupa di smistare le varie richieste ai vari server disponibili, in base all'algoritmo di tipo Round Robin. La funzione del DNS consiste semplicemente nel fornire l'indirizzo e la porta del server remoto da contattare, quindi il client inoltrerà una nuova richiesta di connessione al server indicato dal DNS. Può accadere però che il server non sia disponibile, per questo motivo è stato settato un tempo di timeout per connettersi al server. Se questo timeout scade il client provvede a fare una nuova richiesta al DNS. Se tutti i server sono irraggiungibili, il client notifica l'utente dell'impossibilità a procedere.

Una volta connessosi ad un server disponibile, potrà effettuare diversi tipi di operazione. Il client può chiedere di:

- *visualizzare la lista dei file presenti nella macchina remota*: attraverso il comando "*lista file*" il client inoltra una richiesta al server per poter visionare un elenco dei file presenti nel file system distribuito. Dopo aver inserito nel campo *TipoOperazione* del pacchetto applicativo "*lista file*", esso viene spedito al server, il quale, riconosciuta l'operazione, recupera la lista dei file contenuti nella directory del file system, ed inserisce i loro nomi, comprensivi di estensione, in un nuovo pacchetto applicativo, nel suo campo dati. Una volta ricevuto il pacchetto, il client mostra a video la stringa contenuta nel messaggio.
- *visualizzare un file*: l'utente può richiedere di visualizzare un file, indicando il suo nome comprensivo di estensione (per evitare casi di "omonimia"). L'applicazione client, quindi, si occupa di inizializzare il messaggio inserendo il tipo di operazione da effettuare e il nome del documento prescelto. In questo modo il server che ha intercettato la richiesta verifica la sua presenza nel file system. Se questo non è presente dovrà inviare una risposta con esito negativo al client, che a sua volta chiede all'utente di inserire di nuovo il nome del file. Se esso viene trovato, invece, il server impacchetta il contenuto nel pacchetto applicativo e inoltra il messaggio al client. Se le dimensioni del file sono troppo grandi, esso viene partizionato in più messaggi. Solo una volta che tutte le parti raggiungono il client, viene creata una copia in locale del documento che diventa disponibile per la lettura. Un messaggio di avvenuta operazione viene mostrata a video come conferma.

- *modificare un file*: attraverso il comando “*scrivi file*” l’utente può richiedere di modificare in scrittura un documento. L’applicazione client provvede a richiedere il nome del file e invia le informazioni necessarie al server. Il server, a sua volta, controlla se il file è presente nel file system e se ciò è verificato, l’utente può sottomettere le sue modifiche. Una volta terminato il lavoro il client fornisce un messaggio a video per confermare se la modifica ha avuto esito negativo o meno. I dettagli sono stati visti nel paragrafo [Scrittura e modifica dei file].
- *richiedere di uscire dall'applicazione*: inserendo la richiesta di chiudere l’applicazione tramite il comando “*uscita*”, il client effettua la disconnessione dal server e termina l’applicazione.

### 3.5 Implementazione del server

Verranno ora illustrate in che modo sono state implementate le funzionalità offerte dai vari server e quali sono state le varie problematiche affrontate e le scelte attuate per la risoluzione dei problemi riscontrati.

#### 3.5.1 Trasferimento di file tra un server e un client o tra server e server

Il trasferimento di file tra due parti connesse è realizzato tramite due funzioni, *riceviFile* e *spedisciFile*. Il mittente chiamerà la funzione *spedisciFile* mentre il destinatario chiamerà la funzione *riceviFile*. Per comodità, chiameremo con A la parte che si preoccupa di inviare il file e con B la parte che intende ricevere il file.

##### 3.5.1.1 Spedisci File

La funzione *spedisciFile* prende in ingresso il socket connesso, l’ultimo pacchetto ricevuto e il descrittore del file da spedire. Il controllo sull’esistenza del file da spedire è effettuato prima di richiamare questa funzione. In caso il file non esista, la funzione *spedisciFile* non sarà richiamata e il mittente spedisce un pacchetto applicativo con all’interno un messaggio che avverte il destinatario che il file non è stato trovato. In caso in cui invece, il file esista, la funzione *spedisciFile* controllerà la dimensione del file da inviare e spedirà al destinatario un pacchetto applicativo con il tipo operazione settata a “leggi file, trovato”. Nel pacchetto applicativo verrà inserita anche la dimensione del file che dovrà spedire. In questo modo, il destinatario potrà effettuare tutta una serie di operazioni, che vedremo nella descrizione della funzione *riceviFile*, che gli permetterà di ricevere correttamente il file. Il mittente, spedito il pacchetto, calcolerà il numero di pacchetti applicativi necessari ad inviare il file e procederà ad inviarli. Il file è quindi suddiviso e inviato in tante parti. Per controllare che i pacchetti arrivino con lo stesso ordine al destinatario, ogni pacchetto contiene il numero di parte di file inviato. In questo modo a destinazione è possibile sapere se si sta ricevendo la parte corretta del file o se si è perso qualche pacchetto. Una volta terminato l’invio del file,

la funzione ritorna il valore 1 se è riuscita ad inviare il file o 0 se il destinatario ha riscontrato problemi durante la ricezione del file.



Figura 17 – Invio e ricezione di un file

### 3.5.1.2 Ricevi File

La funzione *riceviFile* permette al destinatario di ricevere un file spedito da un mittente. Questa funzione prende in ingresso il socket connesso al mittente, il pacchetto ricevuto e il percorso, compreso di nome del file, dove andrà scritto il file. Questa funzione gestisce soltanto la ricezione del file, quindi il destinatario, prima di richiamarla, dovrà avvisare il mittente che intende ricevere un file. Per farlo, invierà un pacchetto applicativo con il tipo operazione settata a “*leggi file*” e il campo *nomefile* contenente il nome del file che intende ricevere. Se il mittente troverà il file, come visto in precedenza, spedirà un pacchetto applicativo con il tipo operazione settata a “*leggi file, trovato*”. Il destinatario, ricevuto questo pacchetto, provvederà a richiamare la funzione *riceviFile*.

La funzione, procederà mettendosi in ascolto di un nuovo pacchetto che sarà spedito dal mittente, contenente la dimensione del file che si sta per ricevere. In questo modo, il destinatario può calcolare il numero di parti che dovrà ricevere e controllare se il file ricevuto è stato ricevuto interamente. Dopo aver calcolato il numero di parti, la funzione *riceviFile* prosegue cominciando a ricevere le parti del file. Durante la ricezione viene controllato che il tipo di operazione sia sempre uguale a “*leggi file, trovato*” e il numero del pacchetto ricevuto sia sequenziale. In questo modo si evitano comportamenti bizantini da parte del mittente e si evita che il destinatario si perda qualche parte del file.

Entrambi le funzioni, spediscono il file inserendolo nel corpo del messaggio. Il file è suddiviso in tante parti da 600 bytes, ovvero la dimensione massima del corpo del messaggio. L'ultima parte del file può non essere grande quanto la dimensione massima del corpo del messaggio.

Durante l'implementazione, prima di inserire un controllo sull'ordinamento dei pacchetti ricevuti, abbiamo riscontrato problemi di corruzione dei file. I file, giungevano difatti a destinazione ma risultavano corrotti

anche se la dimensione del file ricevuto coincideva con quella del file inviato. Un caso particolare è stato quello di immagini Jpeg scambiate tra i due interlocutori. L'immagine ricevuta a destinazione veniva visualizzata con delle aberrazioni di colore o disturbi. Per quanto riguarda i file di testo invece, essendo di dimensione molto piccola, non si riscontravano problemi di lettura dei file.

### 3.5.2 Comunicazione tra processi differenti e Ricart-Agrawala

Per attuare l'algoritmo di Ricart-Agrawala abbiamo dovuto scegliere e implementare un meccanismo che permetta a due processi diversi di comunicare tra di loro. Come spiegato nel capitolo 3.1.1 difatti, il meccanismo di Agrawala è eseguito da un processo diverso dai processi che si occupano di gestire le connessioni. In questo modo, il figlio che ha preso in carico una richiesta da parte di un client può rimanere in ascolto delle richieste del client mentre il processo di Ricart-Agrawala si occuperà di contattare tutti gli altri server.

In questo paragrafo, chiameremo con Agra, il figlio dedicato alle richieste di Agrawala, worker, il figlio che ha preso in carico una richiesta proveniente da un client, servant, il figlio che prende in carico le richieste provenienti da altri server sulla porta di servizio.

I processi, possono comunicare tra di loro tramite varie modalità; attraverso lo scambio di messaggi con *send()* e *receive()*, attraverso una memoria condivisa o tramite pipe.

Con la prima modalità ogni processo, per poter comunicare con un altro, deve essere a conoscenza del PID dell'altro processo. Questa modalità non è stata adottata in quanto, l'unico processo che può conoscere i PID degli altri processi generati a runtime è il padre. Agra inoltre, è in ascolto di richieste provenienti da molti processi e non da un solo processo.

L'altra modalità prevede che i processi possano accedere ad un'area di memoria condivisa, che è letta da qualsiasi processo e può essere scritta da qualsiasi processo. Questa modalità è stata adottata, come vedremo in seguito, per fare in modo che un worker possa notificare ad Agra che deve cominciare a effettuare il meccanismo di Agrawala.

Un'altra modalità prevede che i processi possano comunicare tra di loro tramite la pipe. La pipe è simile alla memoria condivisa. E' un descrittore di file, presente sul file system, che può essere letto e scritto da due processi. Vedremo che la Pipe è utilizzata per fare in modo che Agrawala possa informare il worker che il commit può essere effettuato.

Il server, all'avvio, prima di creare e avviare i vari processi, tra cui Agra, crea una memoria condivisa inserendo all'interno un array di 50 posizioni contenente una struct. La struct contiene al suo interno una stringa di 100 bytes e un'altra da 11 bytes per l'ID transazione.

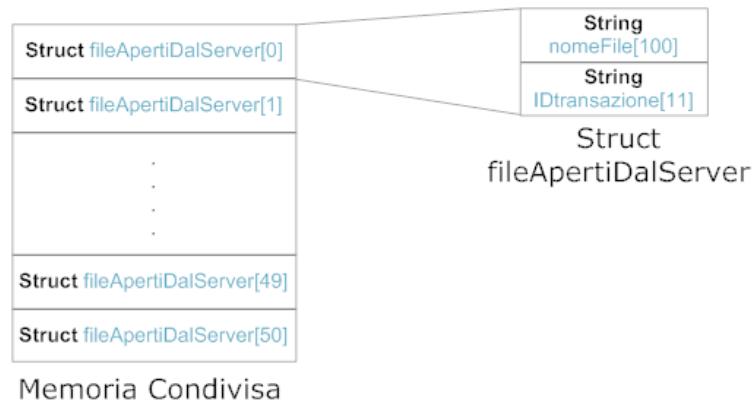


Figura 18 – Struttura della memoria condivisa

L'array ha una dimensione di 50 posizioni in quanto, il numero massimo di client connessi al server è 50. Nella peggiore delle ipotesi quindi, il server si ritroverà ad avere 50 file aperti in attesa di avere una conferma di commit.

Dopo aver creato la memoria condivisa, il server procede a creare il figlio Agra che si occuperà di gestire Agrawala. Agra, procederà a preparare le strutture per leggere la memoria condivisa. Dopodichè contatta il DNS per avere gli indirizzi degli altri server che sono presenti nel sistema e che deve contattare in caso di richieste di Agrawala da attuare. Dopo aver ottenuto gli IP dei server, Agra calcola qual è la porta di servizio su cui può contattare i server, aggiungendo mille unità alla porta comunicata dal DNS, questo in quanto il DNS gli fornisce le porte che i client devono contattare per avere i servizi dal server. Agrawala invece deve contattare il server sulla porta di servizio, che è quella su cui il server è in ascolto in attesa di richieste provenienti da altri server.

Dopo aver effettuato il calcolo entra in un ciclo in cui controlla la presenza di una posizione non vuota all'interno della memoria condivisa. Infatti, quando un worker vuole effettuare Agrawala, per avvisare Agra deve inserire all'interno della memoria condivisa, nella prima posizione libera, il nome del file di cui intende effettuare il commit e l'ID transazione. Agra, non appena si ritrova una posizione non vuota, procede a contattare gli altri server con un pacchetto applicativo il cui campo tipo operazione è settato a "*chiedo di fare commit*", nel corpo del messaggio il proprio ID e nel campo nomefile, il nome del file di cui il client intende fare il commit.

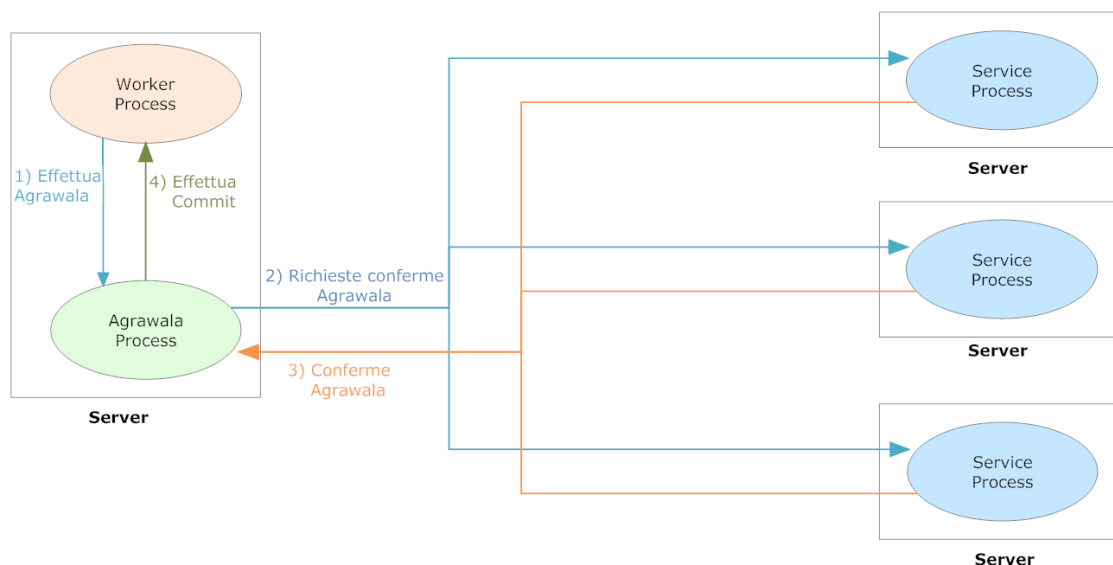


Figura 19 – Esecuzione di Agrawala

Il server che riceve sulla porta di servizio la richiesta di commit, andrà a controllare nella propria memoria condivisa la presenza del file. Se il nome del file non è presente nella memoria condivisa, siamo nel caso in cui il server contattato non ha nessun client che sta utilizzando il file richiesto dall'altro server e quindi, invia subito un pacchetto applicativo con tipo operazione uguale a *"conferma per il commit"* al server richiedente.

Nel caso in cui invece, il nome del file è presente nell'array condiviso, il server che a ricevuti la richiesta controlla l'ID del server richiedente con il proprio. Se il proprio ID è minore, ha la precedenza nel commit e quindi, prima di inviare la conferma al server richiedente, effettuerà lui il commit e poi spedisce la conferma. Altrimenti, spedisce subito la conferma, e nel momento in cui lui dovrà effettuare il commit, avvierà una nuova procedura di Agrawala.

Ritornando al server richiedente, Agra potrà effettuare il commit una volta che ha ricevuto le conferme da tutti i server comunicatigli dal DNS. Nel caso in cui un altro server stia effettuando il commit dello stesso file e ha la precedenza rispetto a lui, Agra rimarrà bloccato fino a che l'altro server non avrà terminato di effettuare il commit.

Una volta ricevute tutte le conferme da tutti i server, Agra per avvisare il worker che ha preso in consegna la richiesta di commit, scriverà nella pipe il messaggio di conferma. La pipe è creata dal worker prima di scrivere nella memoria condivisa il file di cui si intende effettuare il commit. Per fare in modo che solo i due processi leggano la stessa pipe, essa ha come nome l'ID transazione utilizzato dal worker per scambiare dati



con il client. Agra è a conoscenza dell'ID transazione grazie alla struct presente nella memoria condivisa. La struct contiene infatti, oltre al nome del file di cui si intende fare il commit, anche l'ID transazione associato ad esso e può quindi facilmente sapere quale sarà il file di pipe che dovrà aggiornare per avvisare il worker.

Il worker, dopo che ha scritto nella memoria condivisa, si trova in un ciclo in cui controlla in continuazione il file di pipe per avere la conferma da Agra. Dopo aver ricevuto la conferma, il worker procede a scrivere il file nel file system del server e invierà l'aggiornamento agli altri server. Quando tutti i server hanno ricevuto l'aggiornamento, procede ad inviare una conferma al client.

### 3.5.3 Spedizione degli aggiornamenti agli altri server

Dopo che il worker ha ricevuto la conferma di via libera per il commit, dopo aver effettuato la scrittura del file in locale, prima di confermare al client il buon esito dell'operazione deve spedire l'aggiornamento agli altri server in modo tale da avere preservare la consistenza del file system distribuito. L'operazione di aggiornamento è effettuata tramite la funzione *spedisciAggiornamentiAiServer()* che prende in ingresso il descrittore del file contenente gli aggiornamenti da inviare, il nome del file da aggiornare, l'id del server che vuole effettuare l'aggiornamento e l'id transazione. La funzione procede nel richiedere al DNS gli indirizzi dei server facenti parte del file system distribuito. Anche in questo caso, la porta che utilizzerà per contattare i server sarà quella di servizio, quindi anche in questo caso, dopo aver ricevuto la porta dal DNS, calcoleremo quella di servizio aggiungendo mille unità. Dopo aver terminato il calcolo, la funzione comincia a spedire gli aggiornamenti ad un server alla volta. Per poter spedire un aggiornamento, il server crea un pacchetto applicativo con il tipo operazione settato a *"aggiorna file"*, il campo *nomefile* contenente il nome del file di cui si deve effettuare l'aggiornamento, il campo *ID transazione* con l'ID transazione creato in precedenza dal worker. Il figlio di servizio del server contattato, risponderà con un pacchetto applicativo contenente all'interno il tipo operazione settato a *"aggiorna file, pronto"*. Da questo punto in poi, l'aggiornamento è inviato secondo le modalità descritte nel capitolo 3.5.1.

Dal lato del server che ha preso in consegna la richiesta di servizio e che ha ricevuto il pacchetto applicativo con tipo operazione settato a *"aggiorna file"*, dopo aver ricevuto il file, provvederà ad unire gli aggiornamenti ricevuti con il contenuto del file già presente nel proprio file system. Prima di effettuare l'unione con gli aggiornamenti ricevuti, il server effettua un lock del file che intende aggiornare. In questo modo si evita che un altro processo, che ha ricevuto le conferme per il commit, scrivi lo stesso file. E' così preservata l'integrità del file e l'ordinamento delle scritture sul file del file system distribuito.

### 3.5.4 Richiesta degli IP al DNS

Nei paragrafi 3.5.2 e 3.5.3 abbiamo illustrato come, il figlio di servizio e di Agrawala, per ottenere gli indirizzi degli altri server facenti parte del file system distribuito, procedono a contattare il server DNS. Illustreremo ora come viene contattato il server DNS e quali sono le operazioni che vengono effettuate dopo aver ricevuto la lista.

La richiesta degli ip da parte dei processi è effettuata tramite la funzione *chiediTuttiGliIPalDNS()* che prende in ingresso un array di struct di tipo *sockaddr\_in* dove verranno salvati gli indirizzi ricevuti dal DNS, l'IP e la porta del DNS da contattare e l'ID del server che sta effettuando la richiesta. La funzione procede a allocare

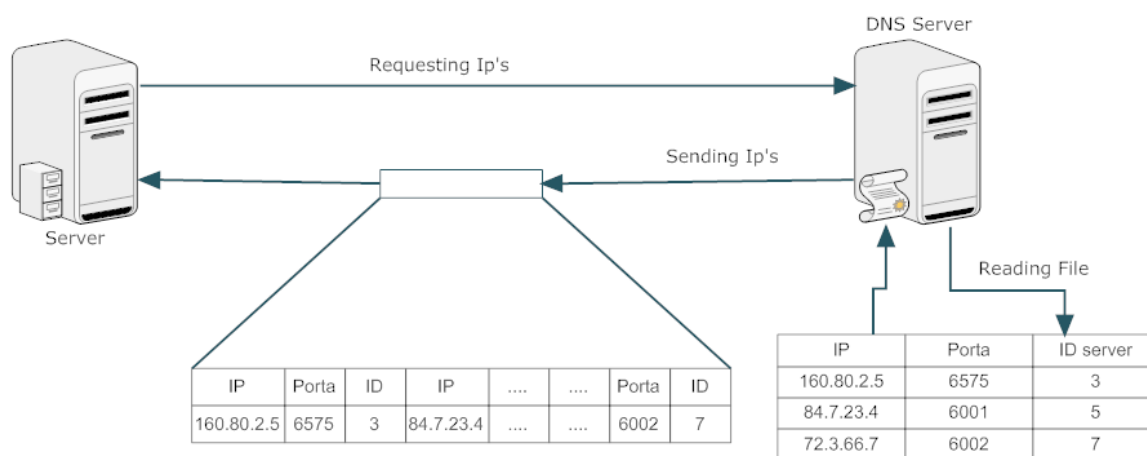


Figura 20 - Richiesta di IP al DNS server

le strutture necessarie a ricevere e a processare gli indirizzi DNS ricevuti dal server DNS. Successivamente procederà ad inviare un pacchetto applicativo con il tipo operazione settato a "indirizzi server" al server DNS.

Il server DNS, ricevuta la richiesta, procederà a leggere un file contenente gli indirizzi, le porte e gli ID dei server attivi e li serializzerà all'interno di una stringa. La stringa verrà inserita all'interno del campo messaggio del pacchetto applicativo e verrà spedita al server richiedente gli indirizzi. Il tipo operazione settato è "indirizzi server". Il server, ricevuta la risposta, controllerà che il tipo operazione ricevuta dal DNS sia "indirizzi server", questo per evitare comportamenti bizantini da parte del DNS. Se il tipo di operazione è quello desiderato, procederà nel leggere il campo messaggio del pacchetto applicativo, che conterrà gli indirizzi IP serializzati. Effettuerà dunque una deserializzazione salvando IP e porta in una stringa e l'ID in una variabile di tipo int. Dopo questo procedimento, procederà a salvare nella struct *sockaddr\_in* ricevuta come parametro di ingresso, gli indirizzi IP e le porte dei server attivi. Durante la fase di salvataggio controllerà gli ID dei server corrispondenti agli IP e scarnerà quello corrispondente al proprio ID. Questo

perché al server interessa conoscere tutti gli IP dei server presenti, escluso il proprio, perché non è interessato a contattare se stesso su un'altra porta. Durante le fasi di debug ci è capitato di tralasciare questo dettaglio e l'effetto ottenuto è stato un comportamento anomalo del server durante le fasi di Agrawala o di aggiornamento dei file. In particolare, durante Agrawala, il server rimaneva appeso in attesa di risposte da lui stesso. Difatti, quando veniva effettuata la richiesta a lui stesso, il processo di servizio andava a controllare l'array dei file condivisi, in cui risultava che il file richiesto era già in uso. Si otteneva così una situazione di deadlock in quanto entrambi i figli, sia quello di Agrawala, sia quello di servizio, rimanevano in attesa che l'altro inviasse una conferma per sbloccare la situazione.

### 3.5.5 Scrittura del file di log

Durante la sua esecuzione il server scrive alcune delle proprie operazioni su un file di log, in modo che, se subisce un crash, si può vedere cosa l'ha provocato o qual è stata l'ultima operazione svolta dal server prima di esso.

Il file di log è scritto da molti processi nello stesso istante e per permettere una scrittura in mutua esclusione, essa avviene tramite un meccanismo di locking. Grazie alla mutua esclusione e ai meccanismi di locking, un processo che intende scrivere il file, entra nella zona di trying protocol, la fase che precede l'accesso alla sezione critica, e attende l'accesso alla sezione critica. Se nessun processo è all'interno della mutua esclusione, accederà subito, scriverà il file e uscirà dalla sezione critica. Dopo essere uscito dalla

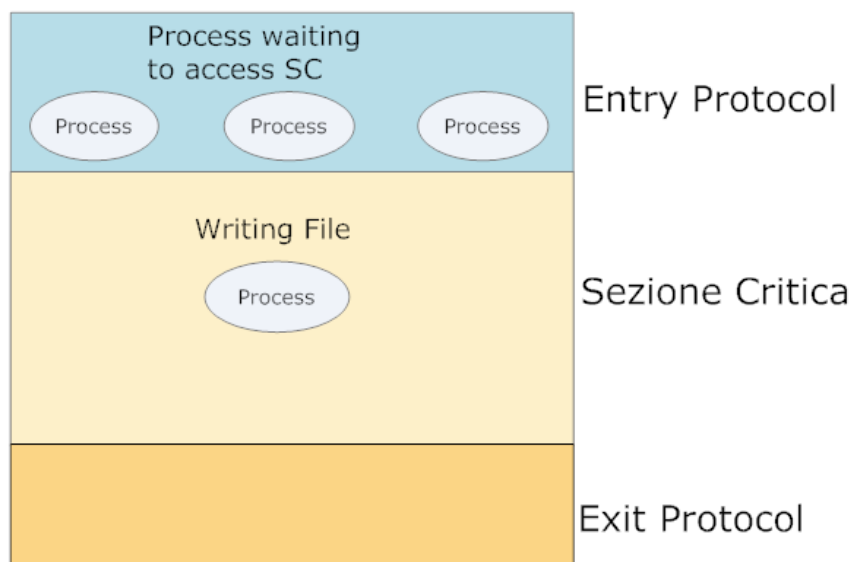


Figura 21 - Accesso alla sezione critica

sezione critica entra nella zona di exit protocol, zona in cui il processo alloca le variabili in modo tale da permettere l'accesso alla sezione critica ad altri processi.

Altrimenti, attenderà fino a che il processo che lo precede non abbia terminato la scrittura del file.

La scrittura del file di log avviene tramite la funzione *writeFileWithLock*. Questa funzione viene richiamata anche durante la fase di commit in quanto permette di scrivere un file in mutua esclusione. La firma prevede in ingresso un descrittore del file da scrivere, il contenuto da scrivere nel file e altri parametri usati per l'eventuale stampa a video dell'informazione che si sta scrivendo sul file. Il locking del file si ottiene grazie alla chiamata di sistema *fcntl* e l'unlocking è effettuato tramite la stessa chiamata. All'interno di questa chiamata vengono passate le modalità in cui si intende accedere al file, in questo caso solo scrittura.

Di seguito è riportato un esempio di file di log.

```
25/06/10 12:02:16 15989 : Avvio la sincronizzazione del file system..
25/06/10 12:02:16 15989: Provo a connettermi al server 2 con ip 127.0.0.1:6001
25/06/10 12:02:16 15989: Provo a connettermi al server 3 con ip 127.0.0.1:6002
25/06/10 12:02:16 15989: Provo a connettermi al server 1 con ip 127.0.0.1:6000
25/06/10 12:02:16 15989: Non risulta nessun server attivo! :(
25/06/10 12:02:16 15989: sbagliato qualcosa, o sono l'unico superstite! Aiuto,
non lasciatemi solo!
25/06/10 12:02:16 15989: Avvio del server numero (ID) 1. Porta richieste : 5000;
porta di servizio: 6000
25/06/10 12:02:16 15989: Server avviato:
25/06/10 12:02:16 15994: In attesa di una richiesta di servizio...
25/06/10 12:02:16 15993: In attesa di una richiesta normale...
25/06/10 12:02:16 15995: Chiedo gli IP degli altri server al DNS
127.0.0.1:7000
25/06/10 12:02:16 15995: IP ricevuti!
25/06/10 12:02:16 15995: In attesa di richieste per agrawala..
```

La scrittura dell'operazione nel file di log viene effettuata dopo che l'operazione è stata eseguita con successo dal server.

### 3.5.6 File di configurazione

Nel paragrafo 3.1.1 abbiamo accennato alla lettura di un file di configurazione durante la fase di avvio del server. Il file di configurazione è stato creato per fare in modo che il server possa avviarsi secondo una

determinata configurazione che può essere modificata senza dover per forza ricompilare i sorgenti del server.

Di seguito è riportato un file di configurazione:

```
ID:1
Porta:5000
cartella file condivisi:fileCondivisi/
IPServerDNS:127.0.0.1
PortaDNS:7000
```

I campi di configurazione sono di facile interpretazione, in particolare:

- **ID:** Contiene l'ID numerico del server che si vuole avviare. E' scelto dall'amministratore di rete e deve essere univoco all'interno del file system distribuito. Serve a far sapere al server qual è il proprio ID che comunicherà agli altri server e che userà per effettuare Agrawala.
- **Porta:** Indica la porta sul quale verranno accettate le richieste di connessione da parte dei client. Il server, dopo aver letto questo parametro, procederà a calcolare la porta di servizio aggiungendo mille unità a questo valore, come specificato nel paragrafo 3.2.
- **Cartella File Condivisi:** E' la cartella dove il server memorizza i file presenti nel file system distribuito. La cartella deve essere scrivibile e leggibile dall'utente che avvia il server.
- **IPServerDNS:** Contiene l'IP del server DNS da contattare per avere informazioni sugli altri server attivi
- **PortaDNS:** Porta su cui il DNS è in attesa di richieste.

Il file di configurazione è letto solo durante la fase di avvio del server per cui, nel caso in cui la configurazione del server venga cambiata, il server dovrà essere riavviato per permettere una nuova lettura del file. I parametri del file di configurazione devono essere scritti con l'ordine con cui sono stati presentati. Il server infatti, durante la lettura del file, non controlla la stringa che precede il parametro di configurazione. Questa scelta è stata effettuata per velocizzare le operazioni di implementazione considerando anche il livello accademico a cui il progetto è riferito.

## 4 Testing

Il seguente capitolo si distingue in due sottoparagrafi, il primo si occupa della verifica del corretto funzionamento in situazioni normali, nel secondo, invece, viene illustrata la capacità del sistema di tollerare i vari casi di failure.

### 4.1 Testing senza failure

Il flusso di eventi che caratterizzano il sistema secondo un funzionamento senza errori necessita di particolare attenzione verso le diverse fasi di scrittura dei file: un file modificato su un server replica deve essere inoltrato agli altri file-server della rete, oppure, uno stesso file modificato da due client distinti presso due o più file-server distinti deve, al termine delle operazioni, risultare identico in tutte le sue copie esistenti presso tali file-server. Verranno qui di seguito illustrati passo-passo entrambi i due casi.

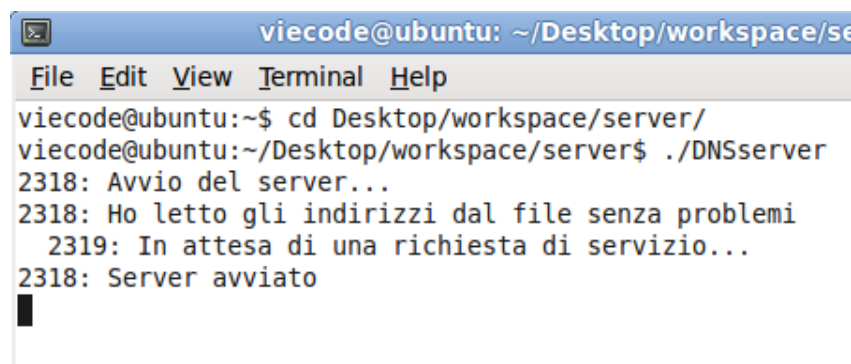
Si considera il sistema composto da un DNS, tre server replica, due client connessi a due server replica differenti.

Nello specifico, il primo client, che verrà chiamato Client1, è connesso al server replica con ID 3, mentre il secondo client, analogamente Client2, è connesso al server replica con ID 2.

Come precedentemente specificato, la priorità nelle commit di scrittura, ordinate dai client ai server, segue uno schema prioritario basato sull'ID dei server: minore sarà l'ID, maggiore sarà la priorità.

#### 4.1.1 Scrittura di un file presso un server e propagazione del suo aggiornamento

Come primo passo viene avviato il server DNS.

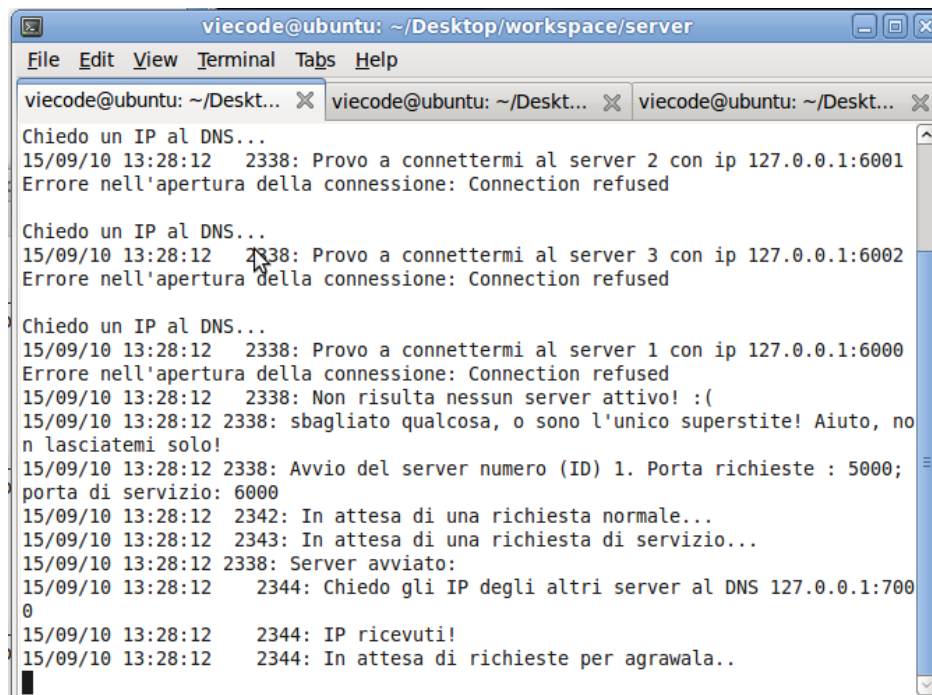


```
viocode@ubuntu: ~/Desktop/workspace/se
File Edit View Terminal Help
viocode@ubuntu:~$ cd Desktop/workspace/server/
viocode@ubuntu:~/Desktop/workspace/server$ ./DNSserver
2318: Avvio del server...
2318: Ho letto gli indirizzi dal file senza problemi
2319: In attesa di una richiesta di servizio...
2318: Server avviato
```

Figura 22 - Avvio del server DNS

Successivamente vengono avviati i file-server replica.

Il primo server avviato, server replica ID 1, riceverà dal DNS la lista degli altri server presenti nella rete e proverà a contattarli. Nel caso specifico i server con ID 2 e ID 3 non sono ancora avviati. Fatto ciò il sistema si pone in attesa di richieste di agrawala per le operazioni sui file.



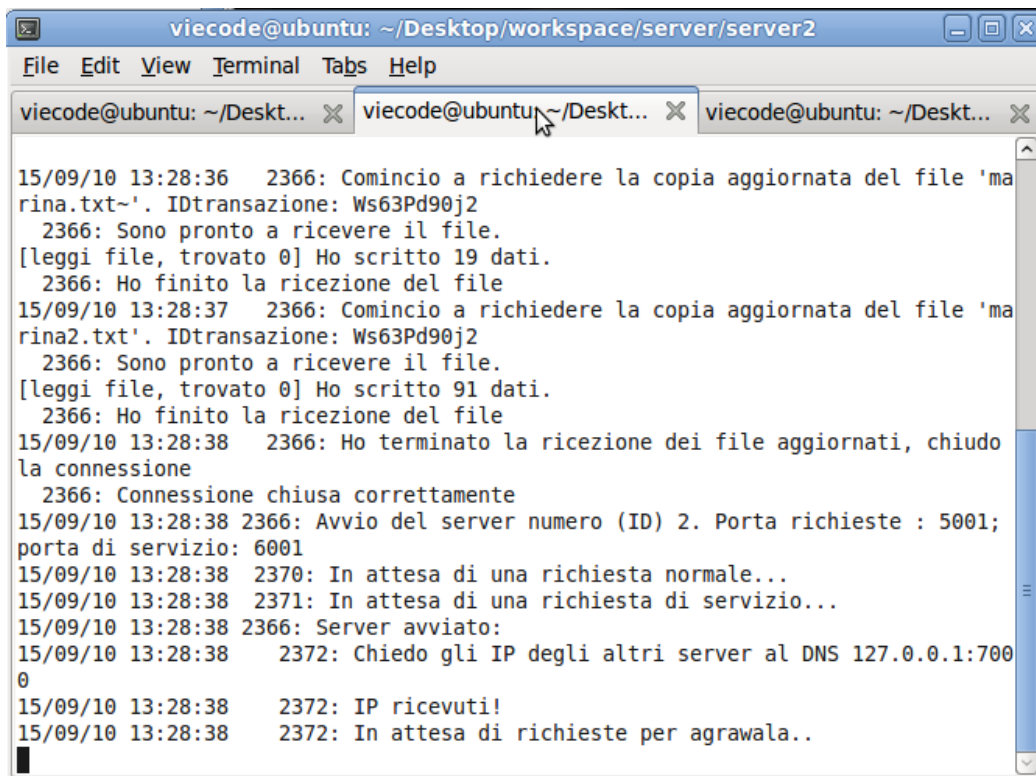
```
viocode@ubuntu: ~/Desktop/workspace/server
File Edit View Terminal Tabs Help
viocode@ubuntu: ~/Deskt... X viocode@ubuntu: ~/Deskt... X viocode@ubuntu: ~/Deskt... X
Chiedo un IP al DNS...
15/09/10 13:28:12 2338: Provo a connettermi al server 2 con ip 127.0.0.1:6001
Errore nell'apertura della connessione: Connection refused

Chiedo un IP al DNS...
15/09/10 13:28:12 2338: Provo a connettermi al server 3 con ip 127.0.0.1:6002
Errore nell'apertura della connessione: Connection refused

Chiedo un IP al DNS...
15/09/10 13:28:12 2338: Provo a connettermi al server 1 con ip 127.0.0.1:6000
Errore nell'apertura della connessione: Connection refused
15/09/10 13:28:12 2338: Non risulta nessun server attivo! :(
15/09/10 13:28:12 2338: sbagliato qualcosa, o sono l'unico superstite! Aiuto, non lasciatemi solo!
15/09/10 13:28:12 2338: Avvio del server numero (ID) 1. Porta richieste : 5000;
porta di servizio: 6000
15/09/10 13:28:12 2342: In attesa di una richiesta normale...
15/09/10 13:28:12 2343: In attesa di una richiesta di servizio...
15/09/10 13:28:12 2338: Server avviato:
15/09/10 13:28:12 2344: Chiedo gli IP degli altri server al DNS 127.0.0.1:7000
15/09/10 13:28:12 2344: IP ricevuti!
15/09/10 13:28:12 2344: In attesa di richieste per agrawala..
```

Figura 23 -Avvio del server 1

Viene avviato il secondo server, server replica ID 2, che come il primo, contatta e riceve dal DNS la lista dei server registrati. Oltre a questo, trovando il server ID 1 attivo, richiederà la versione più aggiornata di un file testuale presente in entrambi, "marina.txt". La transazione avrà un id univoco.



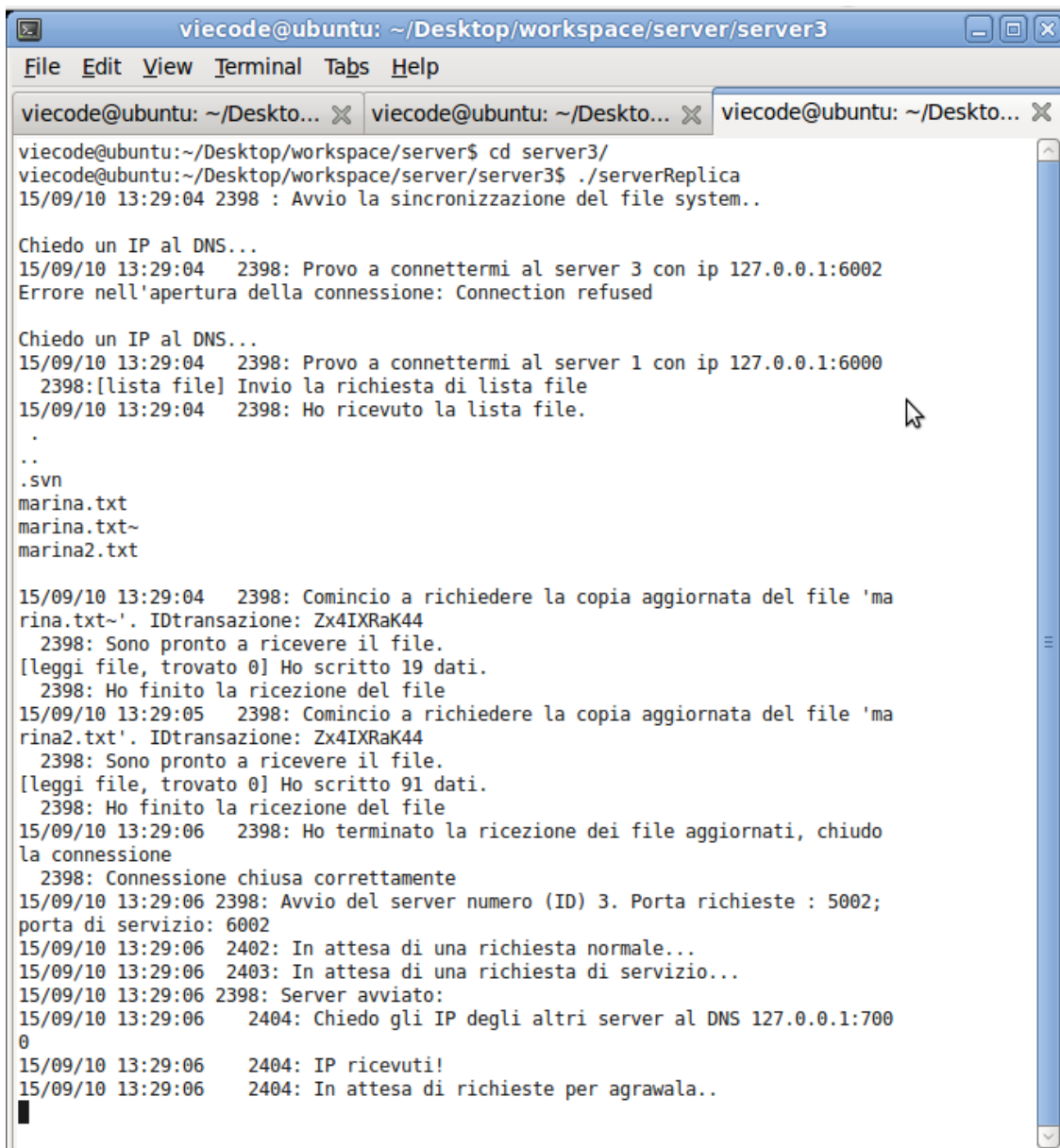
```
viecode@ubuntu: ~/Desktop/workspace/server/server2
File Edit View Terminal Tabs Help
viecode@ubuntu: ~/Deskt... viecode@ubuntu: ~/Deskt... viecode@ubuntu: ~/Deskt...

15/09/10 13:28:36 2366: Comincio a richiedere la copia aggiornata del file 'ma
rina.txt~'. IDtransazione: Ws63Pd90j2
2366: Sono pronto a ricevere il file.
[leggi file, trovato 0] Ho scritto 19 dati.
2366: Ho finito la ricezione del file
15/09/10 13:28:37 2366: Comincio a richiedere la copia aggiornata del file 'ma
rina2.txt'. IDtransazione: Ws63Pd90j2
2366: Sono pronto a ricevere il file.
[leggi file, trovato 0] Ho scritto 91 dati.
2366: Ho finito la ricezione del file
15/09/10 13:28:38 2366: Ho terminato la ricezione dei file aggiornati, chiudo
la connessione
2366: Connessione chiusa correttamente
15/09/10 13:28:38 2366: Avvio del server numero (ID) 2. Porta richieste : 5001;
porta di servizio: 6001
15/09/10 13:28:38 2370: In attesa di una richiesta normale...
15/09/10 13:28:38 2371: In attesa di una richiesta di servizio...
15/09/10 13:28:38 2366: Server avviato:
15/09/10 13:28:38 2372: Chiedo gli IP degli altri server al DNS 127.0.0.1:700
0
15/09/10 13:28:38 2372: IP ricevuti!
15/09/10 13:28:38 2372: In attesa di richieste per agrawala..
```

Figura 24 - Avvio del server 2

L'ultimo ad essere avviato è il server ID 3, che troverà sia ID 1 che ID 2 operanti.





```
viecode@ubuntu: ~/Desktop/workspace/server/server3
File Edit View Terminal Tabs Help
viecode@ubuntu: ~/Deskto... X viecode@ubuntu: ~/Deskto... X viecode@ubuntu: ~/Deskto... X
viecode@ubuntu:~/Desktop/workspace/server$ cd server3/
viecode@ubuntu:~/Desktop/workspace/server/server3$ ./serverReplica
15/09/10 13:29:04 2398 : Avvio la sincronizzazione del file system..

Chiedo un IP al DNS...
15/09/10 13:29:04 2398: Provo a connettermi al server 3 con ip 127.0.0.1:6002
Errore nell'apertura della connessione: Connection refused

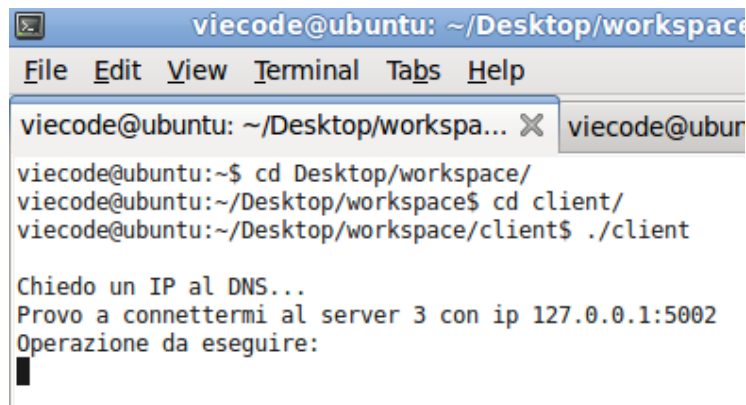
Chiedo un IP al DNS...
15/09/10 13:29:04 2398: Provo a connettermi al server 1 con ip 127.0.0.1:6000
2398:[lista file] Invio la richiesta di lista file
15/09/10 13:29:04 2398: Ho ricevuto la lista file.
.
..
.svn
marina.txt
marina.txt~
marina2.txt

15/09/10 13:29:04 2398: Comincio a richiedere la copia aggiornata del file 'ma
rina.txt~'. IDtransazione: Zx4IXRaK44
2398: Sono pronto a ricevere il file.
[leggi file, trovato 0] Ho scritto 19 dati.
2398: Ho finito la ricezione del file
15/09/10 13:29:05 2398: Comincio a richiedere la copia aggiornata del file 'ma
rina2.txt'. IDtransazione: Zx4IXRaK44
2398: Sono pronto a ricevere il file.
[leggi file, trovato 0] Ho scritto 91 dati.
2398: Ho finito la ricezione del file
15/09/10 13:29:06 2398: Ho terminato la ricezione dei file aggiornati, chiudo
la connessione
2398: Connessione chiusa correttamente
15/09/10 13:29:06 2398: Avvio del server numero (ID) 3. Porta richieste : 5002;
porta di servizio: 6002
15/09/10 13:29:06 2402: In attesa di una richiesta normale...
15/09/10 13:29:06 2403: In attesa di una richiesta di servizio...
15/09/10 13:29:06 2398: Server avviato:
15/09/10 13:29:06 2404: Chiedo gli IP degli altri server al DNS 127.0.0.1:700
0
15/09/10 13:29:06 2404: IP ricevuti!
15/09/10 13:29:06 2404: In attesa di richieste per agrawala..
```

Figura 25 - Avvio del server 3

Vengono lanciati i due client:

Il client 1, che stabilisce una connessione con il server ID 3, assegnatogli dal DNS.

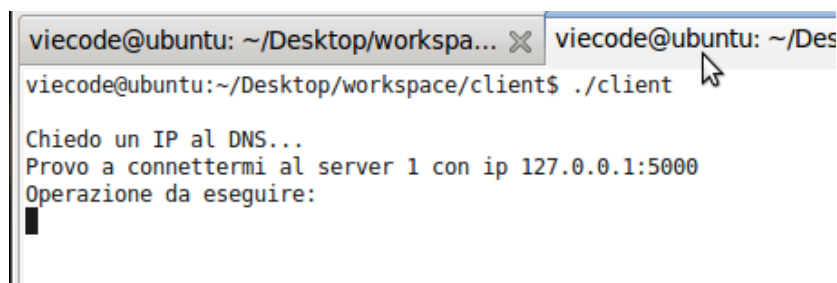


```
viecode@ubuntu: ~/Desktop/workspace
File Edit View Terminal Tabs Help
viecode@ubuntu: ~/Desktop/workspa... X viecode@ubun
viecode@ubuntu:~$ cd Desktop/workspace/
viecode@ubuntu:~/Desktop/workspace$ cd client/
viecode@ubuntu:~/Desktop/workspace/client$ ./client

Chiedo un IP al DNS...
Provo a connettermi al server 3 con ip 127.0.0.1:5002
Operazione da eseguire:
█
```

Figura 26 - Avvio del primo client

Il client2, che stabilisce una connessione con il server replica ID 1, assegnatoli dal DNS.



```
viecode@ubuntu: ~/Desktop/workspa... X viecode@ubuntu: ~/Des
viecode@ubuntu:~/Desktop/workspace/client$ ./client
Chiedo un IP al DNS...
Provo a connettermi al server 1 con ip 127.0.0.1:5000
Operazione da eseguire:
█
```

Figura 27 - Avvio del client 2

Il client 1 richiede al server ID 3 di poter effettuare una scrittura su un file inizialmente vuoto, "marina.txt", digitando il comando "scrivi file". Ricevuto un messaggio di conferma dal server replica, immetterà nel terminale la stringa "stringa 1", darà invio ed al messaggio successivo del server che ha preso in consegna la stringa, risponderà con "commit" per chiedere a questo di salvare le modifiche.

Successivamente, con il comando "uscita", terminerà la connessione col server ID 3.

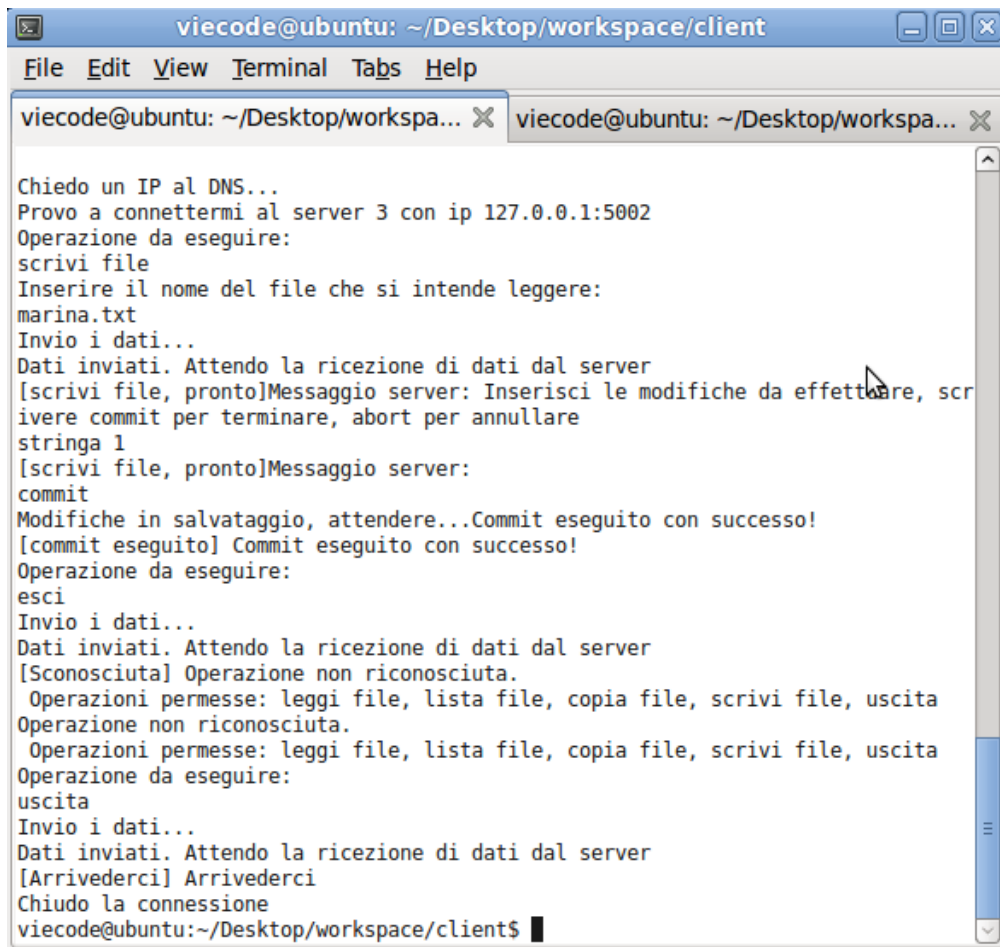


Figura 28 - Scrittura di un file e uscita con il client 1

Nel server ID 3 è ora presente la versione aggiornata del file.

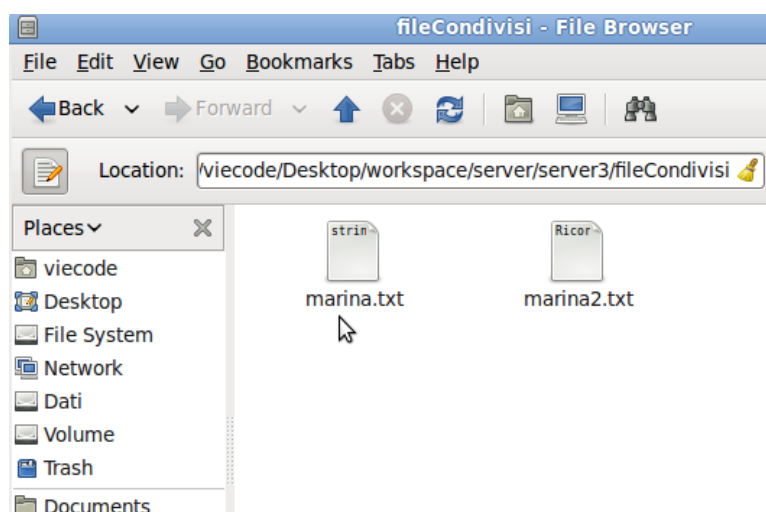


Figura 29 - File marina.txt presente sul file system

Il file “marina.txt” è stato tenuto aperto, per comodità, da un editor testuale che infatti mostra che il file ha subito delle modifiche da parte di un’applicazione (i server replica in esecuzione sulla macchina) ed occorre farne il reload per poterne visualizzare il contenuto aggiornato.

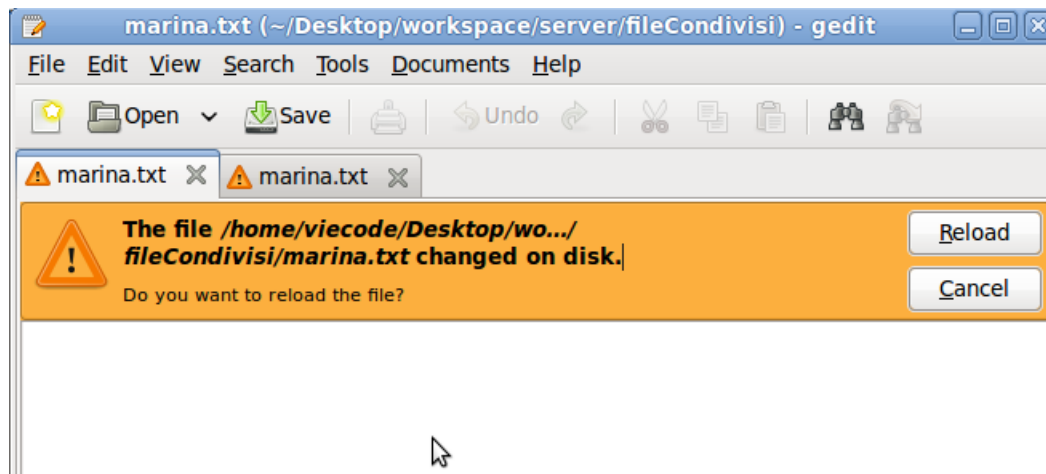


Figura 30 - Gedit notifica la scrittura del file da parte di un altro processo

Le modifiche risuntano infatti esser state propagate dal server ID 3 agli altri due server replica:

Il server ID 1

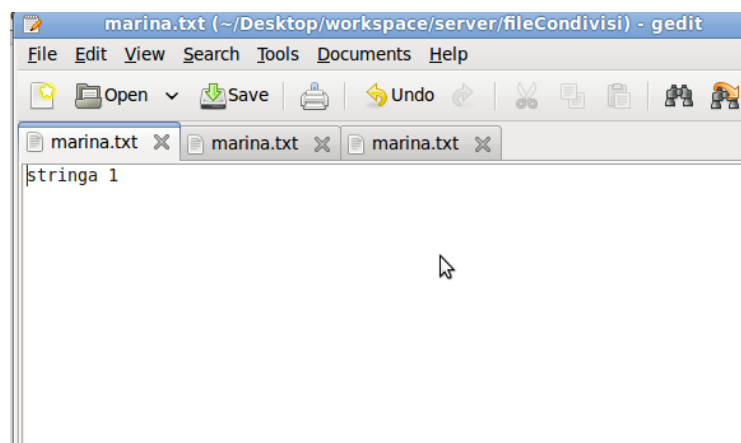


Figura 31 - Contenuto del file presente nel file system del server 1

Il server ID 2

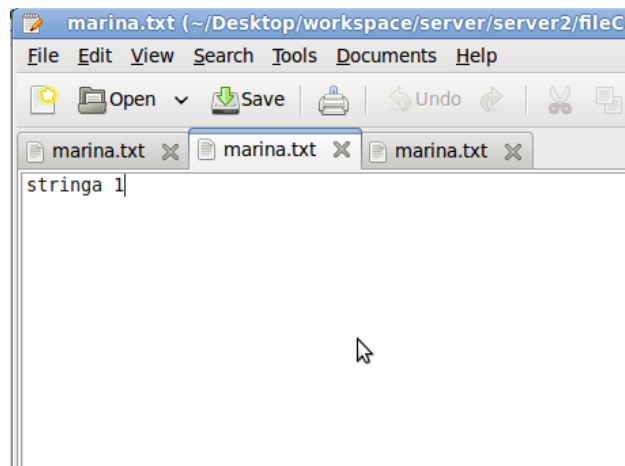
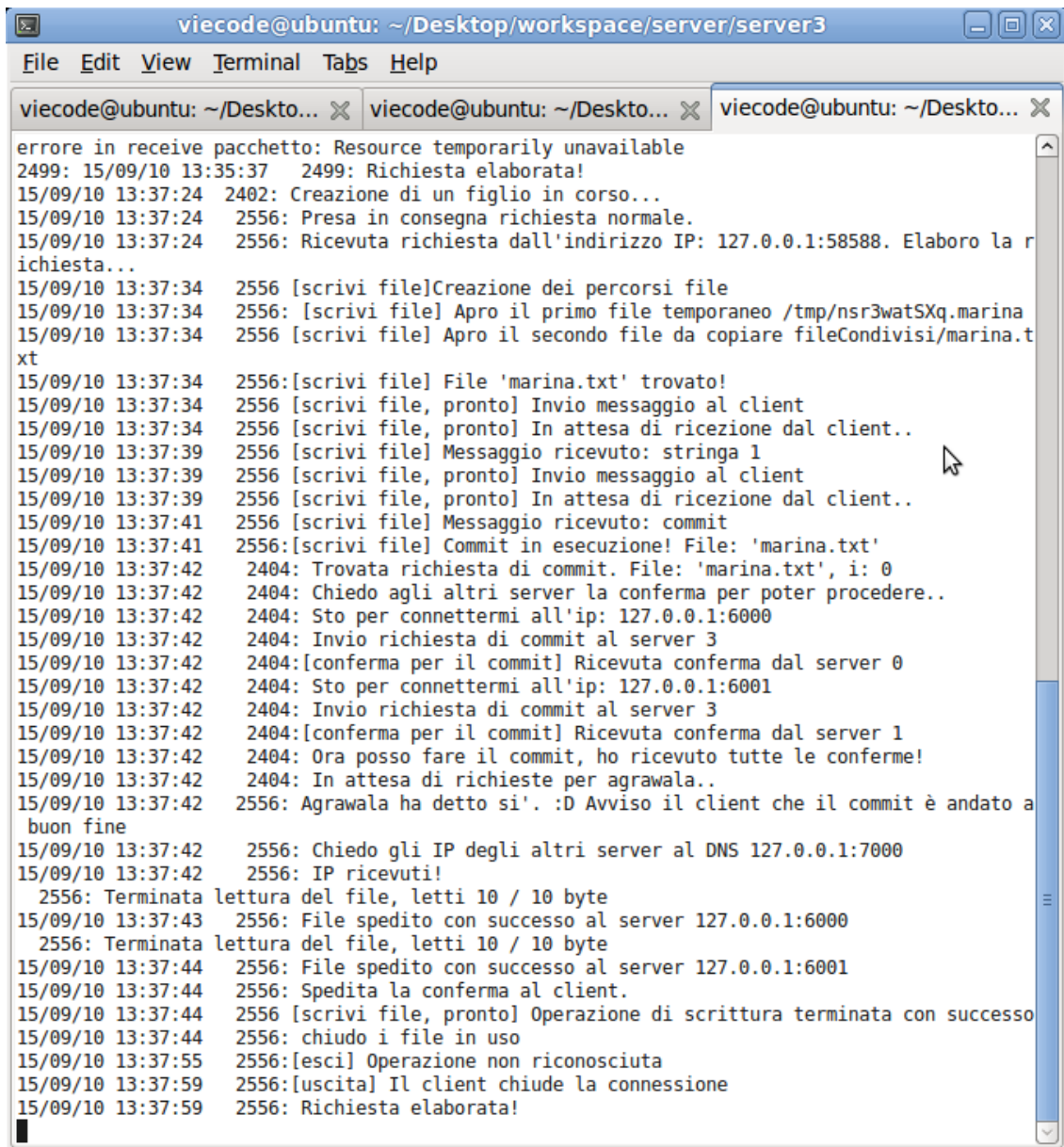


Figura 32 - Contenuto del file presente nel file system 2

Dal punto di vista del server ID 3, sono visibili in shell le operazioni di ricezione della stringa testuale, ricezione della richiesta di salvataggio, avvio dell'algoritmo di agrawala per stabilire delle priorità (in questo caso nessun altro server era interessato ad effettuare il commit), ricezione delle reply dagli altri server e propagazione del file.

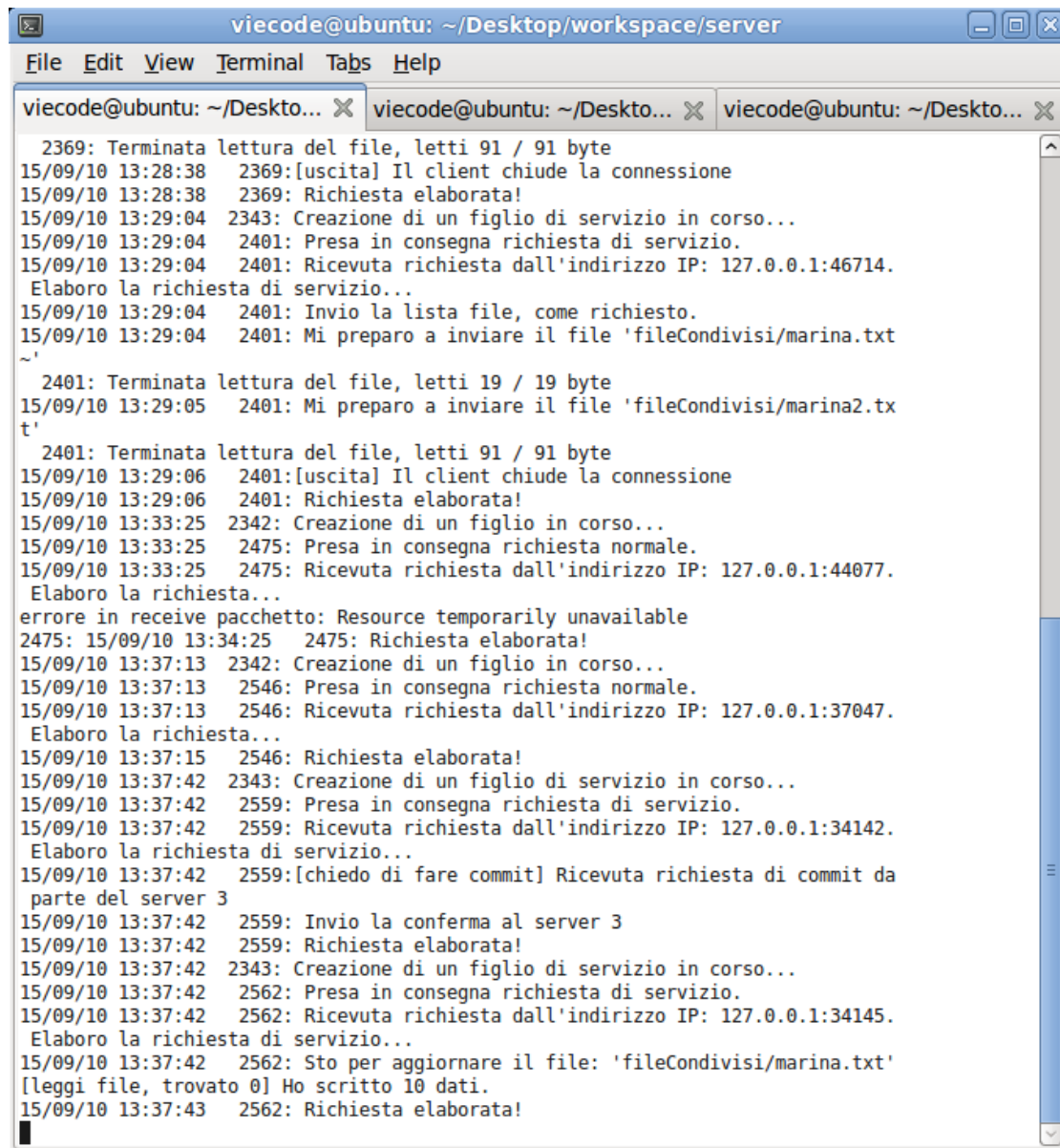


```
viocode@ubuntu: ~/Desktop/workspace/server/server3
File Edit View Terminal Tabs Help
viocode@ubuntu: ~/Deskto... X viocode@ubuntu: ~/Deskto... X viocode@ubuntu: ~/Deskto... X
errore in receive pacchetto: Resource temporarily unavailable
2499: 15/09/10 13:35:37 2499: Richiesta elaborata!
15/09/10 13:37:24 2402: Creazione di un figlio in corso...
15/09/10 13:37:24 2556: Presa in consegna richiesta normale.
15/09/10 13:37:24 2556: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1:58588. Elaboro la r
ichiesta...
15/09/10 13:37:34 2556 [scrivi file] Creazione dei percorsi file
15/09/10 13:37:34 2556: [scrivi file] Apro il primo file temporaneo /tmp/nsr3watSXq.marina
15/09/10 13:37:34 2556 [scrivi file] Apro il secondo file da copiare fileCondivisi/marina.t
xt
15/09/10 13:37:34 2556:[scrivi file] File 'marina.txt' trovato!
15/09/10 13:37:34 2556 [scrivi file, pronto] Invio messaggio al client
15/09/10 13:37:34 2556 [scrivi file, pronto] In attesa di ricezione dal client..
15/09/10 13:37:39 2556 [scrivi file] Messaggio ricevuto: stringa 1
15/09/10 13:37:39 2556 [scrivi file, pronto] Invio messaggio al client
15/09/10 13:37:39 2556 [scrivi file, pronto] In attesa di ricezione dal client..
15/09/10 13:37:41 2556 [scrivi file] Messaggio ricevuto: commit
15/09/10 13:37:41 2556:[scrivi file] Commit in esecuzione! File: 'marina.txt'
15/09/10 13:37:42 2404: Trovata richiesta di commit. File: 'marina.txt', i: 0
15/09/10 13:37:42 2404: Chiedo agli altri server la conferma per poter procedere..
15/09/10 13:37:42 2404: Sto per connettermi all'ip: 127.0.0.1:6000
15/09/10 13:37:42 2404: Invio richiesta di commit al server 3
15/09/10 13:37:42 2404:[conferma per il commit] Ricevuta conferma dal server 0
15/09/10 13:37:42 2404: Sto per connettermi all'ip: 127.0.0.1:6001
15/09/10 13:37:42 2404: Invio richiesta di commit al server 3
15/09/10 13:37:42 2404:[conferma per il commit] Ricevuta conferma dal server 1
15/09/10 13:37:42 2404: Ora posso fare il commit, ho ricevuto tutte le conferme!
15/09/10 13:37:42 2404: In attesa di richieste per agrawala..
15/09/10 13:37:42 2556: Agrawala ha detto si'. :D Avviso il client che il commit è andato a
buon fine
15/09/10 13:37:42 2556: Chiedo gli IP degli altri server al DNS 127.0.0.1:7000
15/09/10 13:37:42 2556: IP ricevuti!
2556: Terminata lettura del file, letti 10 / 10 byte
15/09/10 13:37:43 2556: File spedito con successo al server 127.0.0.1:6000
2556: Terminata lettura del file, letti 10 / 10 byte
15/09/10 13:37:44 2556: File spedito con successo al server 127.0.0.1:6001
15/09/10 13:37:44 2556: Spedita la conferma al client.
15/09/10 13:37:44 2556 [scrivi file, pronto] Operazione di scrittura terminata con successo
15/09/10 13:37:44 2556: chiudo i file in uso
15/09/10 13:37:55 2556:[esci] Operazione non riconosciuta
15/09/10 13:37:59 2556:[uscita] Il client chiude la connessione
15/09/10 13:37:59 2556: Richiesta elaborata!
```

Figura 33 - Operazioni svolte dal server 3 durante la scrittura del file

Dal punto di vista degli altri due server ID 1 e ID 2 è possibile notare la ricezione della richiesta, l'invio della reply ad ID 3 e l'aggiornamento del file ricevuto.

Output del server ID 1



```
viecode@ubuntu: ~/Desktop/workspace/server
File Edit View Terminal Tabs Help
viecode@ubuntu: ~/Deskto... X viecode@ubuntu: ~/Deskto... X viecode@ubuntu: ~/Deskto... X
2369: Terminata lettura del file, letti 91 / 91 byte
15/09/10 13:28:38 2369:[uscita] Il client chiude la connessione
15/09/10 13:28:38 2369: Richiesta elaborata!
15/09/10 13:29:04 2343: Creazione di un figlio di servizio in corso...
15/09/10 13:29:04 2401: Presa in consegna richiesta di servizio.
15/09/10 13:29:04 2401: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1:46714.
Elaboro la richiesta di servizio...
15/09/10 13:29:04 2401: Invio la lista file, come richiesto.
15/09/10 13:29:04 2401: Mi preparo a inviare il file 'fileCondivisi/marina.txt'
~'
2401: Terminata lettura del file, letti 19 / 19 byte
15/09/10 13:29:05 2401: Mi preparo a inviare il file 'fileCondivisi/marina2.tx
t'
2401: Terminata lettura del file, letti 91 / 91 byte
15/09/10 13:29:06 2401:[uscita] Il client chiude la connessione
15/09/10 13:29:06 2401: Richiesta elaborata!
15/09/10 13:33:25 2342: Creazione di un figlio in corso...
15/09/10 13:33:25 2475: Presa in consegna richiesta normale.
15/09/10 13:33:25 2475: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1:44077.
Elaboro la richiesta...
errore in receive pacchetto: Resource temporarily unavailable
2475: 15/09/10 13:34:25 2475: Richiesta elaborata!
15/09/10 13:37:13 2342: Creazione di un figlio in corso...
15/09/10 13:37:13 2546: Presa in consegna richiesta normale.
15/09/10 13:37:13 2546: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1:37047.
Elaboro la richiesta...
15/09/10 13:37:15 2546: Richiesta elaborata!
15/09/10 13:37:42 2343: Creazione di un figlio di servizio in corso...
15/09/10 13:37:42 2559: Presa in consegna richiesta di servizio.
15/09/10 13:37:42 2559: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1:34142.
Elaboro la richiesta di servizio...
15/09/10 13:37:42 2559:[chiedo di fare commit] Ricevuta richiesta di commit da
parte del server 3
15/09/10 13:37:42 2559: Invio la conferma al server 3
15/09/10 13:37:42 2559: Richiesta elaborata!
15/09/10 13:37:42 2343: Creazione di un figlio di servizio in corso...
15/09/10 13:37:42 2562: Presa in consegna richiesta di servizio.
15/09/10 13:37:42 2562: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1:34145.
Elaboro la richiesta di servizio...
15/09/10 13:37:42 2562: Sto per aggiornare il file: 'fileCondivisi/marina.txt'
[leggi file, trovato 0] Ho scritto 10 dati.
15/09/10 13:37:43 2562: Richiesta elaborata!
```

Figura 34 - Operazioni svolte dal server 1 durante la scrittura

Output del server ID 2, del tutto analogo al server precedente



```
viecode@ubuntu: ~/Desktop/workspace/server/server2
File Edit View Terminal Tabs Help

viecode@ubuntu: ~/Deskto... X viecode@ubuntu: ~/Deskto... X viecode@ubuntu: ~/Deskto... X

15/09/10 13:28:37 2366: Comincio a richiedere la copia aggiornata del file 'marina2.txt'. IDtransazione: Ws63Pd90j2
2366: Sono pronto a ricevere il file.
[leggi file, trovato 0] Ho scritto 91 dati.
2366: Ho finito la ricezione del file
15/09/10 13:28:38 2366: Ho terminato la ricezione dei file aggiornati, chiudo la connessione
2366: Connessione chiusa correttamente
15/09/10 13:28:38 2366: Avvio del server numero (ID) 2. Porta richieste : 5001; porta di servizio: 6001
15/09/10 13:28:38 2370: In attesa di una richiesta normale...
15/09/10 13:28:38 2371: In attesa di una richiesta di servizio...
15/09/10 13:28:38 2366: Server avviato:
15/09/10 13:28:38 2372: Chiedo gli IP degli altri server al DNS 127.0.0.1:7000
15/09/10 13:28:38 2372: IP ricevuti!
15/09/10 13:28:38 2372: In attesa di richieste per agrawala..
15/09/10 13:34:27 2370: Creazione di un figlio in corso...
15/09/10 13:34:27 2495: Presa in consegna richiesta normale.
15/09/10 13:34:27 2495: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1:60409.
Elaboro la richiesta...
15/09/10 13:34:31 2495: Richiesta elaborata!
15/09/10 13:37:17 2370: Creazione di un figlio in corso...
15/09/10 13:37:17 2551: Presa in consegna richiesta normale.
15/09/10 13:37:17 2551: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1:51622.
Elaboro la richiesta...
15/09/10 13:37:20 2551: Richiesta elaborata!
15/09/10 13:37:42 2371: Creazione di un figlio di servizio in corso...
15/09/10 13:37:42 2560: Presa in consegna richiesta di servizio.
15/09/10 13:37:42 2560: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1:43550.
Elaboro la richiesta di servizio...
15/09/10 13:37:42 2560:[chiedo di fare commit] Ricevuta richiesta di commit da parte del server 3
15/09/10 13:37:42 2560: Invio la conferma al server 3
15/09/10 13:37:42 2560: Richiesta elaborata!
15/09/10 13:37:43 2371: Creazione di un figlio di servizio in corso...
15/09/10 13:37:43 2563: Presa in consegna richiesta di servizio.
15/09/10 13:37:43 2563: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1:43553.
Elaboro la richiesta di servizio...
15/09/10 13:37:43 2563: Sto per aggiornare il file: 'fileCondivisi/marina.txt'
[leggi file, trovato 0] Ho scritto 10 dati.
15/09/10 13:37:44 2563: Richiesta elaborata!
```

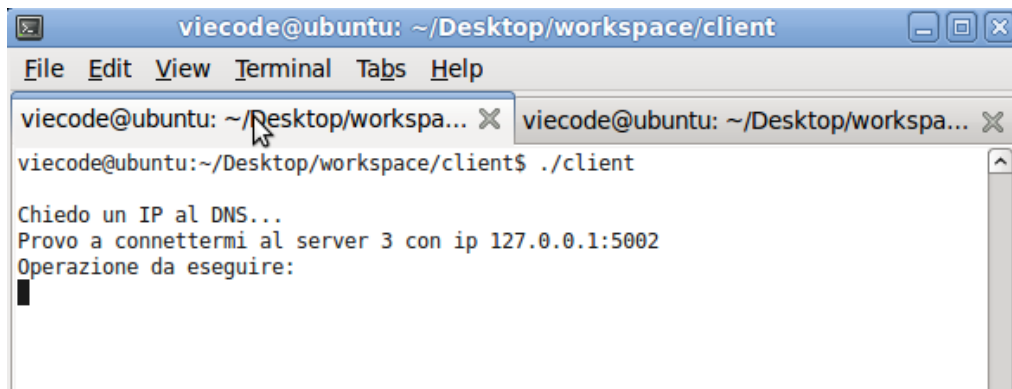
Figura 35 - Operazioni svolte dal server 2 durante la scrittura

#### 4.1.2 Scritture concorrenti

Verrà ora illustrata la gestione delle priorità tra server replica.

Due client, connessi a due server distinti, effettueranno delle scritture concorrenti su due copie dello stesso file. Il file in questione è quello usato nel paragrafo precedente, “marina.txt”. I server sono tutti avviati come nel caso precedente. Il client 1 viene ravviato, e ottiene una connessione con il server ID 3.



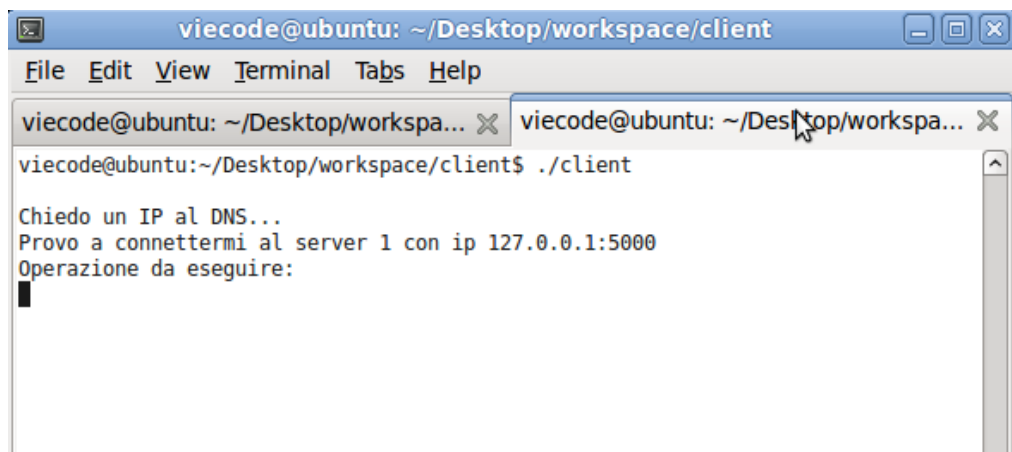


```
viocode@ubuntu: ~/Desktop/workspace/client
File Edit View Terminal Tabs Help
viocode@ubuntu: ~/Desktop/workspa... X viocode@ubuntu: ~/Desktop/workspa... X
viocode@ubuntu:~/Desktop/workspace/client$ ./client

Chiedo un IP al DNS...
Provo a connettermi al server 3 con ip 127.0.0.1:5002
Operazione da eseguire:
█
```

Figura 36 - Avvio del client 1

Il client 2 viene anch'esso avviato e ottiene una connessione con il server replica ID 1, il quale avrà maggiore priorità nelle scritture rispetto al server ID 3, essendo il suo ID minore.



```
viocode@ubuntu: ~/Desktop/workspace/client
File Edit View Terminal Tabs Help
viocode@ubuntu: ~/Desktop/workspa... X viocode@ubuntu: ~/Desktop/workspa... X
viocode@ubuntu:~/Desktop/workspace/client$ ./client

Chiedo un IP al DNS...
Provo a connettermi al server 1 con ip 127.0.0.1:5000
Operazione da eseguire:
█
```

Figura 37 - Avvio del client 2

Tornando al client 1, questo effettuerà presso il server ID 3 le operazioni di “scrivi file”, specifica del nome del file “marina.txt”, immissione di una stringa, “stringa 23”, ma senza effettuare la richiesta di commit.

```
viecode@ubuntu: ~/Desktop/workspace/client
File Edit View Terminal Tabs Help
viecode@ubuntu: ~/Desktop/workspa... X viecode@ubuntu: ~/Desktop/workspa... X
viecode@ubuntu:~/Desktop/workspace/client$ ./client

Chiedo un IP al DNS...
Provo a connettermi al server 3 con ip 127.0.0.1:5002
Operazione da eseguire:
scrivi file
Inserire il nome del file che si intende leggere:
marina.txt
Invio i dati...
Dati inviati. Attendo la ricezione di dati dal server
[scrivi file, pronto]Messaggio server: Inserisci le modifiche da effettuare, scr
ivere commit per terminare, abort per annullare
stringa23
[scrivi file, pronto]Messaggio server:
█
```

Figura 38 - Richiesta scrittura file da parte del client 1

Il client 2, eseguirà contemporaneamente al client 1, le seguenti operazioni: “scrivi file”, specifica del nome del file “marina.txt”, immissione di una stringa “stringa 45”, immissione della richiesta (che verrà accettata) di “commit”.

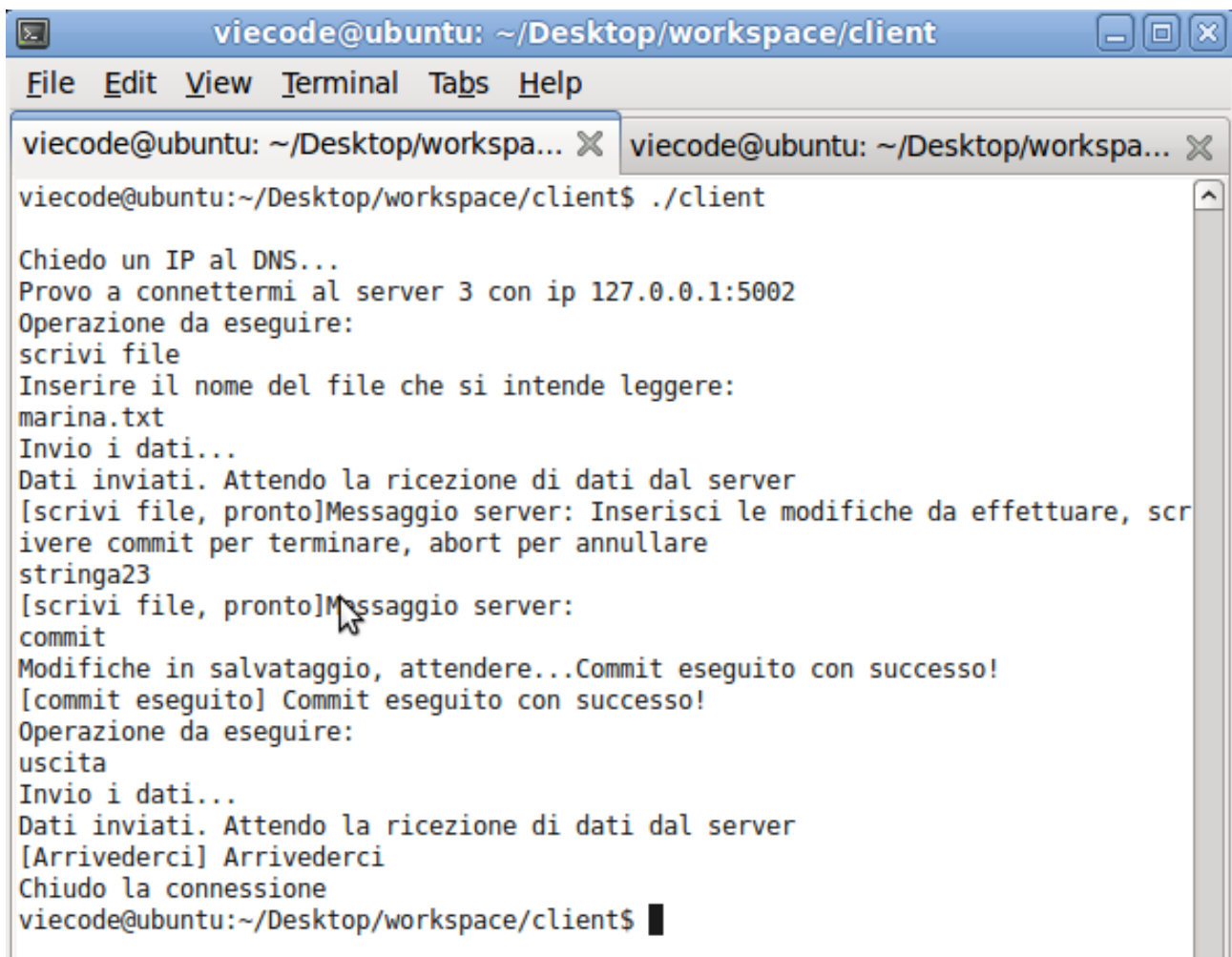
```
viecode@ubuntu: ~/Desktop/workspace/client
File Edit View Terminal Tabs Help
viecode@ubuntu: ~/Desktop/workspa... X viecode@ubuntu: ~/Desktop/workspa... X
viecode@ubuntu:~/Desktop/workspace/client$ ./client

Chiedo un IP al DNS...
Provo a connettermi al server 1 con ip 127.0.0.1:5000
Operazione da eseguire:
scrivi file
Inserire il nome del file che si intende leggere:
marina.txt
Invio i dati...
Dati inviati. Attendo la ricezione di dati dal server
[scrivi file, pronto]Messaggio server: Inserisci le modifiche da effettuare, scr
ivere commit per terminare, abort per annullare
stringa45
[scrivi file, pronto]Messaggio server:
commit
Modifiche in salvataggio, attendere...Commit eseguito con successo!
[commit eseguito] Commit eseguito con successo!
Operazione da eseguire:
█
```

Figura 39 - Scrittura del file e commit da parte del client 2

La richiesta di commit è stata accettata in quanto il server del client 2, server ID 1, ha l'ID minore rispetto agli altri server interessati (ID 3).

A questo punto il client 1 effettuerà la richiesta di commit (accettata perché quella del client 2 è già stata eseguita) e si disconnette.



```
viecode@ubuntu: ~/Desktop/workspace/client
File Edit View Terminal Tabs Help
viecode@ubuntu: ~/Desktop/workspa... X viecode@ubuntu: ~/Desktop/workspa... X
viecode@ubuntu:~/Desktop/workspace/client$ ./client

Chiedo un IP al DNS...
Provo a connettermi al server 3 con ip 127.0.0.1:5002
Operazione da eseguire:
scrivi file
Inserire il nome del file che si intende leggere:
marina.txt
Invio i dati...
Dati inviati. Attendo la ricezione di dati dal server
[scrivi file, pronto]Messaggio server: Inserisci le modifiche da effettuare, scr
ivere commit per terminare, abort per annullare
stringa23
[scrivi file, pronto]Messaggio server:
commit
Modifiche in salvataggio, attendere...Commit eseguito con successo!
[commit eseguito] Commit eseguito con successo!
Operazione da eseguire:
uscita
Invio i dati...
Dati inviati. Attendo la ricezione di dati dal server
[Arrivederci] Arrivederci
Chiudo la connessione
viecode@ubuntu:~/Desktop/workspace/client$
```

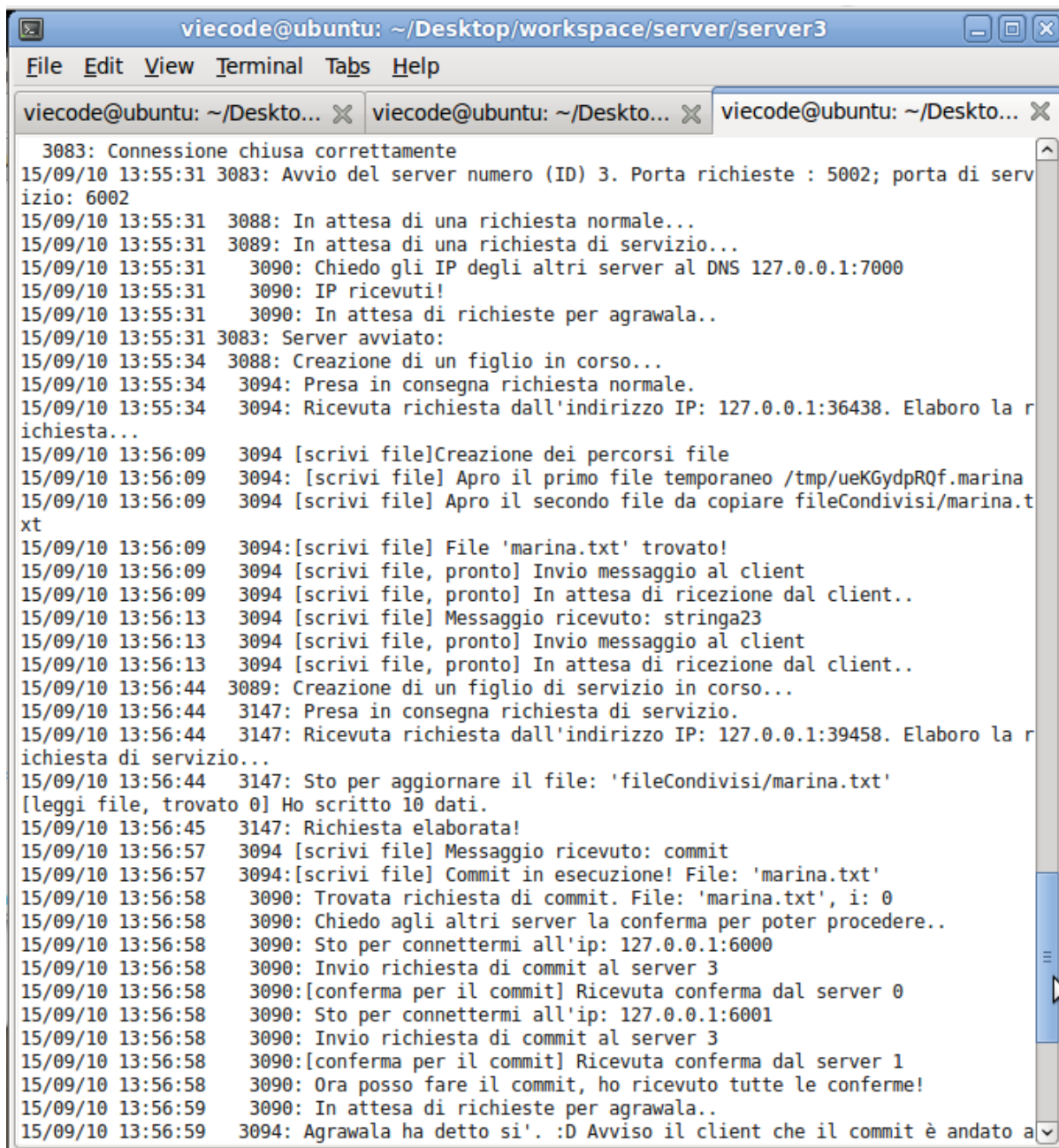
Figura 40 - Commit da parte del client 1

Dal punto di vista dei server replica, è possibile vedere la richiesta presa in consegna dal server ID 1, e l'avvio di agrawala

```
viocode@ubuntu: ~/Desktop/workspace/server
File Edit View Terminal Tabs Help
viocode@ubuntu: ~/Deskto... X viocode@ubuntu: ~/Deskto... X viocode@ubuntu: ~/Deskto... X
15/09/10 13:56:30 3098 [scrivi file, pronto] Invio messaggio al client
15/09/10 13:56:30 3098 [scrivi file, pronto] In attesa di ricezione dal client..
15/09/10 13:56:38 3098 [scrivi file] Messaggio ricevuto: stringa45
15/09/10 13:56:38 3098 [scrivi file, pronto] Invio messaggio al client
15/09/10 13:56:38 3098 [scrivi file, pronto] In attesa di ricezione dal client..
15/09/10 13:56:41 3098 [scrivi file] Messaggio ricevuto: commit
15/09/10 13:56:41 3098:[scrivi file] Commit in esecuzione! File: 'marina.txt'
15/09/10 13:56:42 3072: Trovata richiesta di commit. File: 'marina.txt', i: 0
15/09/10 13:56:42 3072: Chiedo agli altri server la conferma per poter procedere..
15/09/10 13:56:42 3072: Sto per connettermi all'ip: 127.0.0.1:6001
15/09/10 13:56:42 3072: Il server con IP 127.0.0.1:6001 sembra non essere attivo.
15/09/10 13:56:42 3072: Sto per connettermi all'ip: 127.0.0.1:6001
15/09/10 13:56:42 3072: Il server con IP 127.0.0.1:6001 sembra non essere attivo.
15/09/10 13:56:42 3072: Ora posso fare il commit, ho ricevuto tutte le conferme!
15/09/10 13:56:43 3072: In attesa di richieste per agrawala..
15/09/10 13:56:43 3098: Agrawala ha detto si'. :D Avviso il client che il commit è andato a
buon fine
15/09/10 13:56:43 3098: Chiedo gli IP degli altri server al DNS 127.0.0.1:7000
15/09/10 13:56:43 3098: IP ricevuti!
3098: Terminata lettura del file, letti 10 / 10 byte
15/09/10 13:56:44 3098: File spedito con successo al server 127.0.0.1:6001
3098: Terminata lettura del file, letti 10 / 10 byte
15/09/10 13:56:45 3098: File spedito con successo al server 127.0.0.1:6002
15/09/10 13:56:45 3098: Spedita la conferma al client.
15/09/10 13:56:45 3098 [scrivi file, pronto] Operazione di scrittura terminata con successo
15/09/10 13:56:45 3098: chiudo i file in uso
15/09/10 13:56:58 3071: Creazione di un figlio di servizio in corso...
15/09/10 13:56:58 3165: Presa in consegna richiesta di servizio.
15/09/10 13:56:58 3165: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1:55597. Elaboro la r
ichiesta di servizio...
15/09/10 13:56:58 3165:[chiedo di fare commit] Ricevuta richiesta di commit da parte del se
rver 3
15/09/10 13:56:58 3165: Invio la conferma al server 3
15/09/10 13:56:58 3165: Richiesta elaborata!
15/09/10 13:56:59 3071: Creazione di un figlio di servizio in corso...
15/09/10 13:56:59 3168: Presa in consegna richiesta di servizio.
15/09/10 13:56:59 3168: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1:55600. Elaboro la r
ichiesta di servizio...
15/09/10 13:56:59 3168: Sto per aggiornare il file: 'fileCondivisi/marina.txt'
[leggi file, trovato 0] Ho scritto 10 dati.
15/09/10 13:57:00 3168: Richiesta elaborata!
15/09/10 13:57:33 3070: Creazione di un figlio in corso...
15/09/10 13:57:33 3192: Presa in consegna richiesta normale.
```

Figura 41 - Il server 1 scrive il file effettuato anche Agrawala

L'output generato dal server ID 3 sarà il seguente:



```
3083: Connessione chiusa correttamente
15/09/10 13:55:31 3083: Avvio del server numero (ID) 3. Porta richieste : 5002; porta di servizio: 6002
15/09/10 13:55:31 3088: In attesa di una richiesta normale...
15/09/10 13:55:31 3089: In attesa di una richiesta di servizio...
15/09/10 13:55:31 3090: Chiedo gli IP degli altri server al DNS 127.0.0.1:7000
15/09/10 13:55:31 3090: IP ricevuti!
15/09/10 13:55:31 3090: In attesa di richieste per agrawala..
15/09/10 13:55:31 3083: Server avviato:
15/09/10 13:55:34 3088: Creazione di un figlio in corso...
15/09/10 13:55:34 3094: Presa in consegna richiesta normale.
15/09/10 13:55:34 3094: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1:36438. Elaboro la richiesta...
15/09/10 13:56:09 3094 [scrivi file] Creazione dei percorsi file
15/09/10 13:56:09 3094: [scrivi file] Apro il primo file temporaneo /tmp/ueKGydpRQf.marina
15/09/10 13:56:09 3094 [scrivi file] Apro il secondo file da copiare fileCondivisi/marina.txt
15/09/10 13:56:09 3094:[scrivi file] File 'marina.txt' trovato!
15/09/10 13:56:09 3094 [scrivi file, pronto] Invio messaggio al client
15/09/10 13:56:09 3094 [scrivi file, pronto] In attesa di ricezione dal client..
15/09/10 13:56:13 3094 [scrivi file] Messaggio ricevuto: stringa23
15/09/10 13:56:13 3094 [scrivi file, pronto] Invio messaggio al client
15/09/10 13:56:13 3094 [scrivi file, pronto] In attesa di ricezione dal client..
15/09/10 13:56:44 3089: Creazione di un figlio di servizio in corso...
15/09/10 13:56:44 3147: Presa in consegna richiesta di servizio.
15/09/10 13:56:44 3147: Ricevuta richiesta dall'indirizzo IP: 127.0.0.1:39458. Elaboro la richiesta di servizio...
15/09/10 13:56:44 3147: Sto per aggiornare il file: 'fileCondivisi/marina.txt' [leggi file, trovato 0] Ho scritto 10 dati.
15/09/10 13:56:45 3147: Richiesta elaborata!
15/09/10 13:56:57 3094 [scrivi file] Messaggio ricevuto: commit
15/09/10 13:56:57 3094:[scrivi file] Commit in esecuzione! File: 'marina.txt'
15/09/10 13:56:58 3090: Trovata richiesta di commit. File: 'marina.txt', i: 0
15/09/10 13:56:58 3090: Chiedo agli altri server la conferma per poter procedere..
15/09/10 13:56:58 3090: Sto per connettermi all'ip: 127.0.0.1:6000
15/09/10 13:56:58 3090: Invio richiesta di commit al server 3
15/09/10 13:56:58 3090:[conferma per il commit] Ricevuta conferma dal server 0
15/09/10 13:56:58 3090: Sto per connettermi all'ip: 127.0.0.1:6001
15/09/10 13:56:58 3090: Invio richiesta di commit al server 3
15/09/10 13:56:58 3090:[conferma per il commit] Ricevuta conferma dal server 1
15/09/10 13:56:58 3090: Ora posso fare il commit, ho ricevuto tutte le conferme!
15/09/10 13:56:59 3090: In attesa di richieste per agrawala..
15/09/10 13:56:59 3094: Agrawala ha detto si'. :D Avviso il client che il commit è andato a
```

Figura 42 - Scrittura del file da parte del server 3

L'editor di testo mostra le modifiche correttamente inserite, come nell'ordine delle commit, quindi prima la stringa "stringa 45" e dopo la stringa "stringa 23".

Nel server ID 1

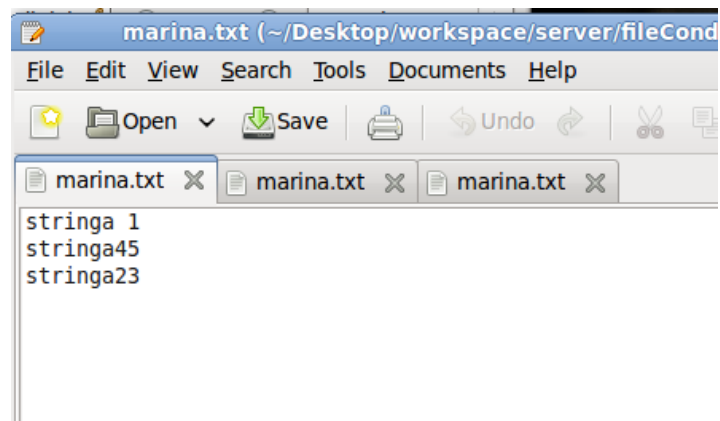


Figura 43 - Server 1, Contenuto del file marina.txt

Nel server ID 2

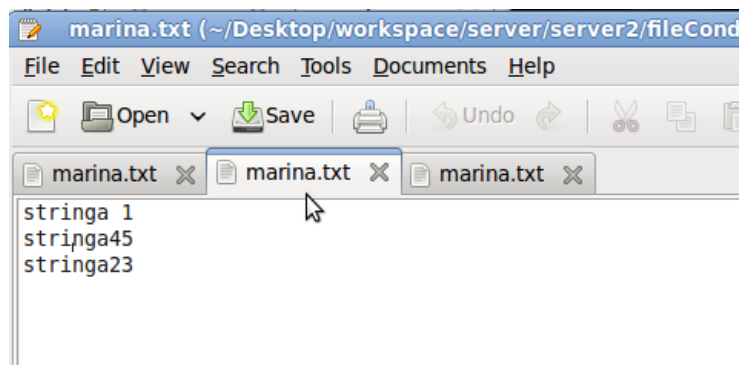


Figura 44 - Server 2, Contenuto del file marina.txt

Nel server ID 3

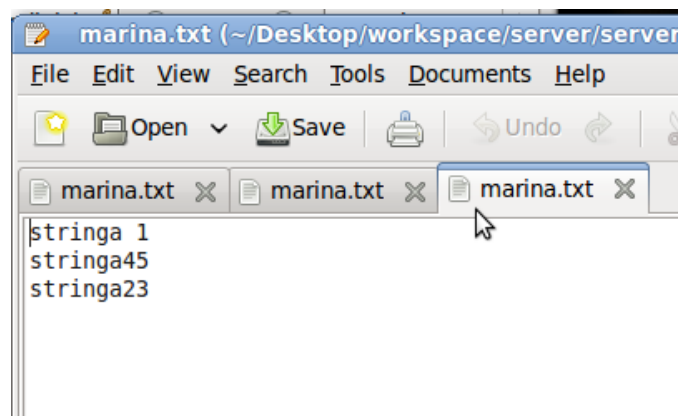


Figura 45 - Server 3, Contenuto del file marina.txt

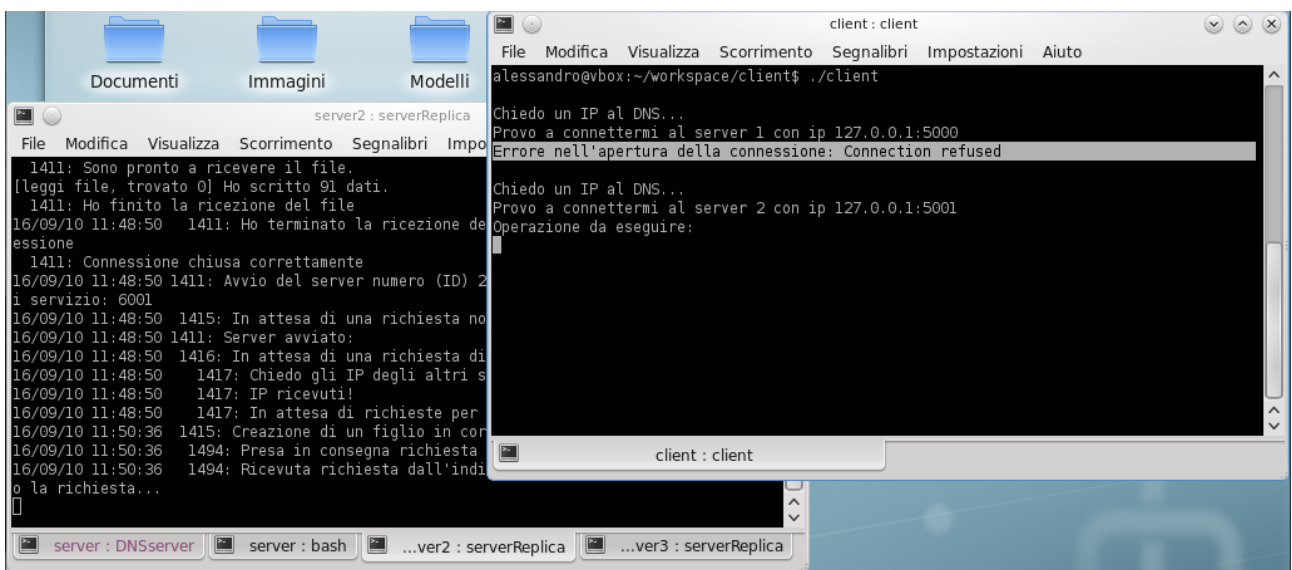


## 4.2 Testing in caso di failure

Vediamo ora il comportamento del server e del client nel caso in cui siano presenti delle failure.

### 4.2.1 Failstop

Il client prova a contattare un server che ha subito un crash. Il client ha un meccanismo che gli permette di capire se il client che sta per contattare è attivo o no. Dopo aver inoltrato una richiesta di connessione il client difatti, attende un timeout, scaduto il quale procederà a contattare un altro server. Nel caso in cui l'indirizzo IP e la porta del server che si sta per contattare risultino irraggiungibili si procede a contattare un nuovo server.



```
client : client
File Modifica Visualizza Scorrimento Segnalibri Impostazioni Aiuto
alessandro@vbox:~/workspace/client$ ./client
Chiedo un IP al DNS...
Provo a connettermi al server 1 con ip 127.0.0.1:5000
Errore nell'apertura della connessione: Connection refused
Chiedo un IP al DNS...
Provo a connettermi al server 2 con ip 127.0.0.1:5001
Operazione da eseguire:
[...]
```

Figura 46 - Failure del client

Il client, prima di contattare il nuovo server, procede a contattare il DNS. Difatti, ricordiamo che il DNS fornisce al client solo un indirizzo IP per volta, secondo un algoritmo di rotazione di tipo round-robin. Il timeout è assegnato al socket tramite la chiamata di sistema `setsockopt()`. In particolare, è stata usata l'opzione `SO_RCVTIMEO`. Se il socket non riesce a contattare il server o se il timeout è scaduto, il socket viene chiuso e la variabile `errno` è settata a 111 e 11 rispettivamente.

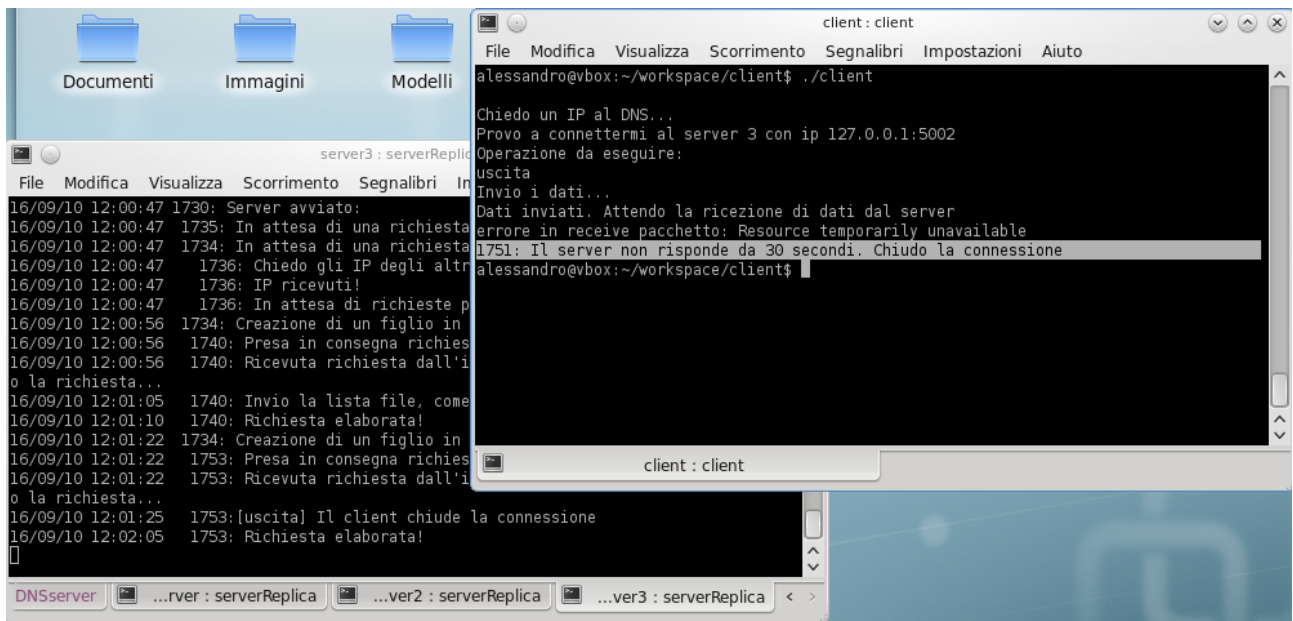


Figura 47 - Crash del server. Timeout scaduto

Grazie a questa scelta, il timeout di attesa vale anche durante tutto il tempo in cui il client è connesso al server. Per cui, se il server subisce un crash durante la connessione con il client, il client chiude a sua volta la connessione e stampa a video un messaggio di errore. Questo tipo di failure è stata simulata inserendo una sleep > di 40 secondi all'operazione di uscita del server. In questo modo, il server attende 40 secondi prima di rispondere, un tempo molto maggiore rispetto al tempo di attesa di risposta da parte del client.

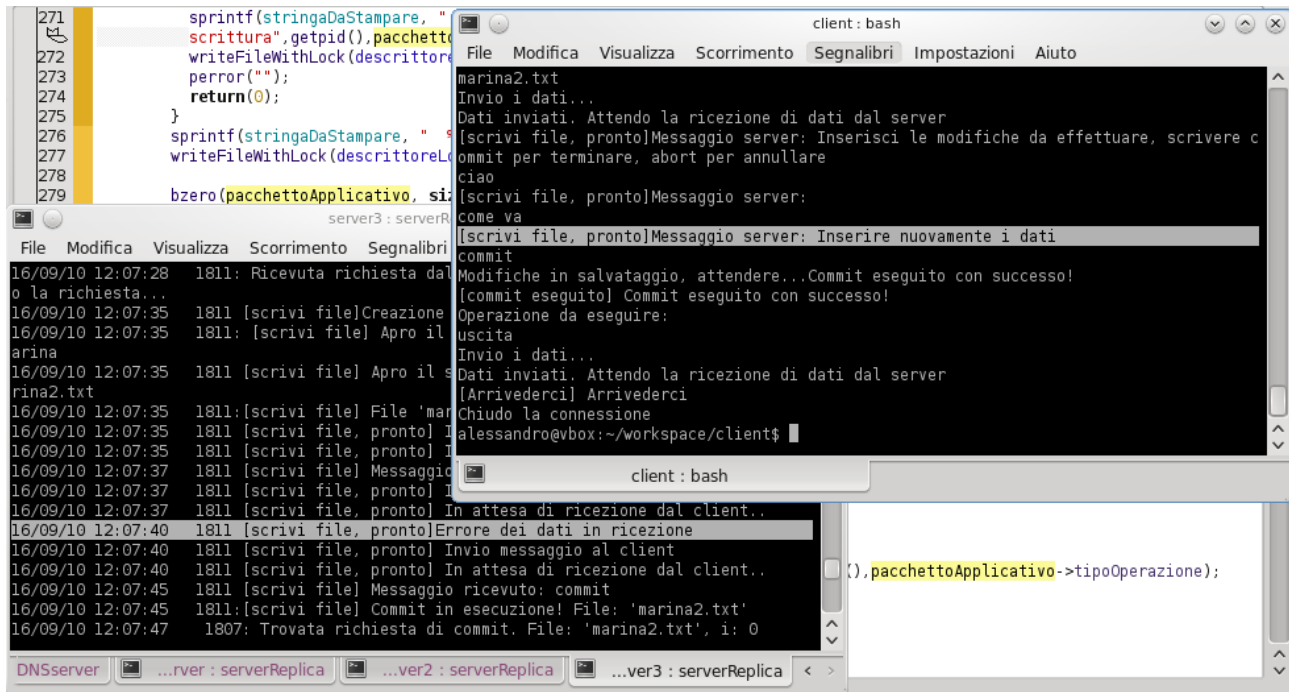
#### 4.2.2 Guasti bizantini

Vedremo ora come il client si comporta in caso di guasti bizantini da parte del server. Abbiamo visto nel capitolo 3.1.2 come durante ogni operazione svolta sia dal client che dal server, il pacchetto applicativo scambiato tra gli interlocutori contiene, nel campo tipo-operazione, l'operazione in atto. Il client e il server controllano difatti ad ogni scambio di messaggi, il tipo di operazione, in modo tale da tutelare i tipi di guasti di tipo bizantino. Difatti, se ad esempio, il server sta inviando al client la lista dei file, ci si aspetta che ogni pacchetto inviato dal server contenga l'operazione "lista file". Il client controlla ad ogni pacchetto ricevuto che il tipo operazione corrisponda, e se ciò non accade interrompe la sua esecuzione, stampa a video un messaggio di errore e termina la propria esecuzione. Lato server invece, il server, passato un timeout, provvederà a chiudere la connessione e a terminare il processo figlio che stava gestendo la richiesta del client.

I guasti bizantini sono anche evitati grazie all'utilizzo di un *IDtransazione* per le operazioni ritenute di maggiore importanza. Durante la copia di un file tra server, o la scrittura di un file da un client a un server,



viene utilizzato una stringa pseudo-casuale di 10 caratteri chiamata *IDtransazione*. Quando un client vuole scrivere un file ad un server, il server provvede a assegnare al client un *IDtransazione*. Esso viene controllato sia dal server, sia dal client e ha validità fino a che non viene effettuato il commit della scrittura del file. Durante la sua validità, sia client che server, controllano che l'*IDtransazione* sia sempre lo stesso.



### Figura 48 - Failure bizantino con IDtransazione modificato

Nel caso in cui l'*IDtransazione* non corrisponda, il server prova a farsi rispedire dal client i dati, in modo tale da non annullare definitivamente la transazione. Nella figura possiamo vedere il caso in cui il server riceva un *IDtransazione* diverso da quello specificato in partenza. Questo tipo di failure è stata simulata inserendo nel server, durante la ricezione del pacchetto applicativo, una stringa diversa da quella dell'*IDtransazione*.

### 4.2.3 Omissioni

Nel caso di omissioni, client e server hanno già stabilito una connessione e si stanno scambiando dei messaggi. Non siamo quindi nel caso in cui il client manda una sola richiesta e il server risponde. Abbiamo già visto nel paragrafo precedente, come l'omissione di una risposta, in questo caso, è risolta con il timeout. Durante lo scambio di messaggi, client e server, oltre a controllare che i campi *tipo operazione* e *IDtransazione* coincidano sempre con quelli aspettati, controllano anche il campo *timestamp*. In questo campo difatti, durante lo scambio di dati, viene salvato, sia dal client che dal server, il numero sequenziale

del pacchetto inviato. In ricezione, entrambi quindi controlleranno che il numero di messaggio ricevuto sia maggiore di una unità rispetto al precedente. Se questa condizione non è verificata siamo nel caso in cui la rete ha perso un messaggio o ha scambiato l'ordine dei messaggi. Durante le fasi di implementazione, uno dei bug di funzionamento del client e del server, era dovuto proprio all'ordine dei messaggi. Durante la fase di scambio file difatti, il file ricevuto dal client risultava corrotto ma con la dimensione file uguale rispetto a quella inviata dal server.

## 5 Conclusioni

Il file system distribuito è stato realizzato per essere eseguito solo per scopi didattici e non professionali. Ci siamo concentrati sulla progettazione e sulla risoluzione delle problematiche relative alla gestione delle failure e della consistenza che ci permettessero di realizzare velocemente il sistema distribuito. Alcune delle scelte progettuali effettuate per il sistema distribuito non possono essere utilizzate in sistemi atti a funzionare su larga scala.

Il DNS è un'entità che dovrebbe essere replicata e non centralizzata. Questo onde evitare eventuali crash del sistema o rallentamenti dovuti al sovraccarico di richieste ad esso pervenute. Inoltre, l'algoritmo Round Robin può essere sostituito con un qualsiasi altro algoritmo di ottimizzazione della distribuzione del carico di lavoro. Ad esempio, i server potrebbero mandare continuamente il proprio stato al DNS in modo tale che, la scelta del server da assegnare al client, possa essere effettuata secondo l'effettivo stato del server e non su base statica come avviene per Round Robin.

L'algoritmo di Ricart-Agrawala utilizzato per garantire la scrittura contemporanea dei file da parte di più client assegna la priorità di scrittura del file in base all'ID del server contattato e non in base a quale client contatta prima il server ed effettua prima il commit. Per avere questa funzionalità si potrebbe utilizzare l'algoritmo di Lamport.

L'utilizzo del linguaggio C ci ha permesso di realizzare funzionalità di basso livello che sono spesso offerte da librerie di sistema integrate nel linguaggio di programmazione in uso e che molto spesso diamo per scontate. Molte problematiche che abbiamo risolto sono state scoperte solo durante l'implementazione del progetto e riguardavano la creazione di algoritmi per funzionalità non offerte dal linguaggio di programmazione C.

Grazie all'utilizzo dell'SVN, un sistema di controllo versione, l'implementazione è stata potuta suddividere in piccole funzionalità anche se erano legate tra di loro, in quanto ogni componente del gruppo aveva in qualsiasi momento la versione aggiornata dei file sorgenti.