Parte implementativa:

Pacca dovrebbe scrivere tutta la parte relativa all’invio e alla ricezione dei file. Come avviene, come è stata implementata. La parte relativa ad Agrawala, la creazione dei vari figli del server. Il compito dei due figli del server, la gestione della memoria dinamica e delle pipe e dove vengono utilizzate (sempre Agrawala). Scrivere come il DNS fornisce gli indirizzi dei server presenti utilizzati per Agrawala. Scrivere come viene scritto il file di configurazione e il file di log con locking.

Marina dovrebbe scrivere tutta la parte riguardante l’operazione scrivi file, e le altre operazioni del server. Cosa avviene sul server nel momento in cui è richiamata l’operazione scrivi file. Come i vari server si scambiano le informazioni sullo stato dei file aggiornati nel momento del loro avvio. E il funzionamento del client. Risoluzione delle failure.

Vedere insieme la parte di testing delle failure.

## 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 [riferimento ad agrawala].

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.

1. 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.

## 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.

## Funzionamento 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.

## Tolleranza alle failure

Il lavoro presentato è in grado di tollerare alcuni tipi di failure. Esse sono di seguito brevemente presentate e verranno illustrato le scelte implementative eseguite per la loro soluzione.

### 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.

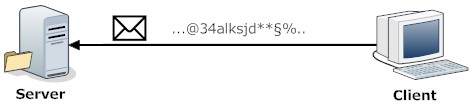


Figura - Failure bizantina

* Omissioni – si verifica quando il client perde alcuni pacchetti. Occorre creare un meccanismo di ritrasmissione per recuperare i pacchetti persi.

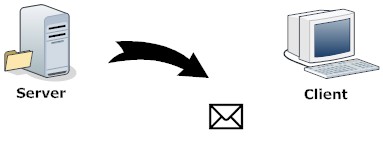


Figura - Omissione del pacchetto

* Failstop – quando si verifica il crash da parte del server il client non deve attenderlo per un tempo indefinito. In questo caso si è scelto di impostare un timeout del socket di trasmissione, di modo che dopo 30 secondi la connessione venga chiusa automaticamente.

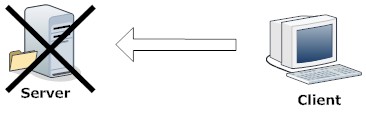


Figura - Crash del server

### 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. Anche in questo caso si è scelto di impostare un timeout del socket di trasmissione, di modo che dopo 30 secondi la connessione venga chiusa automaticamente.

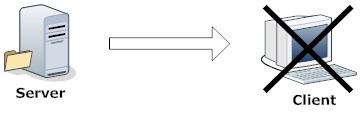


Figura - Crash del client