

UNIVERSITÀ DI PADOVA

SCUOLA DI INGEGNERIA

LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

NOME DEL CORSO: SISTEMI DISTRIBUITI

ANNO ACCADEMICO 2017-2018

Documento di progetto: File System distribuito

Studenti:

Marco Gomiero

Luca Rossi

Davide Storato

Andrea Ziggiotto

Numero di Matricola:

1134728

1141231

1153692

1140706

3 giugno 2018

1 Obiettivo

L'obiettivo del progetto consiste nella progettazione e nell'implementazione di un File System distribuito con requisito principale la *Single Fault Tolerance*, ovvero l'abilità del sistema a tollerare la caduta o l'uscita dal sistema di un nodo, impedendo in questo modo la perdita dell'informazione.

2 Descrizione del Sistema

Il sistema è stato progettato suddividendolo in livelli, come riportato in Figura 1, in modo tale da permettere uno sviluppo più agevole e per meglio garantire una corretta suddivisione dei ruoli per ogni livello. I livelli progettati sono i seguenti:

- Livello di interfaccia utente;
- Livello del File System;
- Livello di connessione.

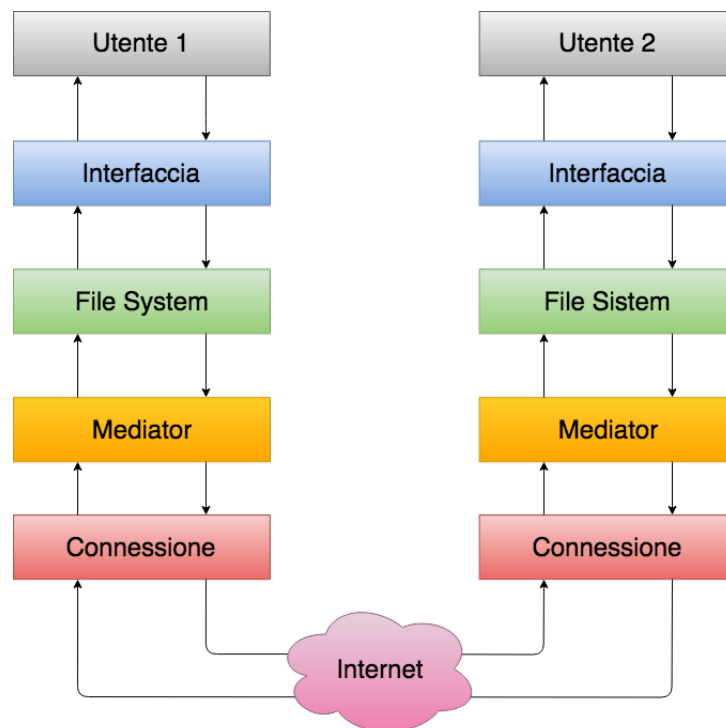


Figura 1: Rappresentazione del sistema sviluppato.

2.1 Livello di connessione

Il livello di connessione si occupa della comunicazione fra i nodi e la creazione o l'accesso ad un specifico File System distribuito. La rete di nodi, che si viene a creare, ha un'architettura logica simile a quella definita nel modello *peer-to-peer (P2P)*, in quanto non esiste una gerarchia tra i nodi. Questa architettura è sviluppata utilizzando la libreria *Java Remote Method Invocation (RMI)*.

Alla creazione di un nodo è necessario fornire alcuni parametri, nel caso fosse già presente un File System (FS) bisogna conoscere l'indirizzo IP, il numero di porta ed il nome del servizio RMI di uno dei nodi già connessi, mentre nel caso il nodo voglia creare un nuovo FS si dovrà solamente avviare la procedura di *Binding* prevista dall'infrastruttura RMI.

In entrambi i casi il nodo deve esporre la propria interfaccia RMI in rete creando un'istanza dell'interfaccia *NetNode*, che nel sistema gestisce la comunicazione tra nodi. L'istanza creata viene resa accessibile all'esterno attraverso il metodo *bind()* della classe *Registry*.

Quando ci si connette ad un nodo, esso fornisce la lista dei nodi presenti nel sistema al nodo che si è connesso, il quale a sua volta informerà della sua connessione i nodi restanti. Per tenere sempre aggiornata la lista dei nodi connessi al sistema ogni nodo lancerà una routine che periodicamente ne verifica la raggiungibilità e in caso di nodi non raggiungibili, li rimuove dalla lista. A connessione avvenuta il nuovo nodo riceverà dal nodo a cui si è connesso tutte le strutture dati aggiornate che descrivono lo stato in cui si trova il sistema.

2.1.1 Replicazione

Per garantire la *Single Fault Tollerance* il sistema implementa una routine di servizio che gestisce la replicazione dei file nella rete. La replica del file viene salvata su un altro nodo appartenente alla rete in base ai seguenti criteri:

- Stabilità e affidabilità: vengono considerati i nodi che sono connessi da più tempo.
- Numero di File: si cerca di distribuire uniformemente i file, in modo da non avere nodi contenenti un maggior numero di repliche aumentando così considerevolmente la perdita d'informazione in caso di caduta del nodo.

Con l'obiettivo di evitare che la caduta di un nodo comporti la presenza di una sola copia del file, ogni nodo possiede una routine che ad intervalli regolari verifica il numero di copie per i file presenti nel File System. In tal modo se il numero di copie è inferiore alla soglia desiderata a causa della caduta di un nodo, la routine provvede a ricreare su altri nodi le copie necessarie per ristabilire la fault tollerance.

2.2 Livello del File System

Tale livello, che si occupa della gestione dei file e delle directory, si suddivide essenzialmente in due sotto livelli:

- *FileService*: si occupa della creazione, dell'eliminazione e della modifica dei file e degli attributi ad esso legati. Durante la creazione di un file viene ad esso assegnato un *UFID* univoco che è successivamente memorizzato all'interno di una cartella comune del File System del PC. Il loro posizionamento all'interno delle varie cartelle è gestito dall'interfaccia utente descritta nella Sezione 2.3.
- *DirectoryService*: si occupa della gestione dei file all'interno delle varie directory permettendone la creazione e la rimozione. Questo funzionamento viene gestito attraverso la presenza di un file *JSON* in cui è possibile salvare l'albero delle directory che rappresenta il File System distribuito completo.

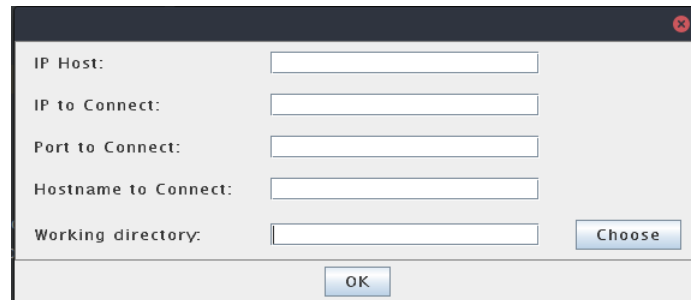
Il livello File System deve poter accedere ai file presenti in rete, in lettura e scrittura, quindi deve poter interagire con il *Mediator* che gestisce la comunicazione con i nodi. Al fine di ridurre il numero di tali comunicazioni è stato implementato nei nodi un sistema di *Caching* che, conservando una copia degli ultimi file letti, permette di velocizzare e ridurre le operazioni per la lettura. Per evitare di conservare un file non aggiornato, quando viene modificato da un qualunque nodo, tutte le copie presenti nelle Cache del sistema sono invalidate.

I livelli di File System di ogni nodo possiedono inoltre una struttura dati condivisa, ovvero ogni struttura sarà la copia esatta delle altre, che per ogni file del sistema indica la possibilità di accedere in scrittura in quel momento ad un determinato file, questo per evitare collisioni ed episodi di inconsistenza nel caso due nodi provino ad modificare contemporaneamente il file. Ogni volta che un nodo vuole modificare uno specifico file, prende i diritti per la modifica e disabilita questo privilegio per tutti gli altri nodi della rete.

2.3 Livello di Interfaccia Utente

Il livello di Interfaccia Utente permette all'utente di eseguire tutte le funzioni sviluppate senza l'utilizzo della riga di comando. All'avvio dell'applicazione viene mostrato un Form, come quello mostrato in Figura 2, che permette d'inserire il proprio IP e la directory del File System dove salvare i file. Nel caso il nodo non fosse il creatore di un nuovo FS distribuito, è necessario compilare gli altri campi con le coordinate del nodo a cui connettersi.

Una volta eseguita questa operazione viene mostrata l'interfaccia principale dell'applicazione Figura 3 dove è possibile utilizzare le funzionalità offerte dal sistema. La barra del titolo riporta le coordinate del proprio nodo per permettere una lettura più agevole dei dati.



A dialog box titled "Accesso al File System" with a close button (X) in the top right corner. It contains five input fields: "IP Host:", "IP to Connect:", "Port to Connect:", "Hostname to Connect:", and "Working directory:". The "Working directory:" field has a "Choose" button next to it. At the bottom center is an "OK" button.

Figura 2: Schermata di accesso al File System.

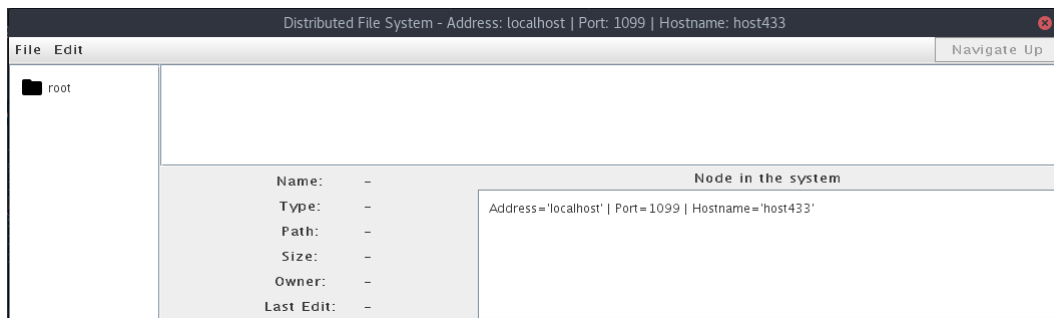


Figura 3: Schermata interfaccia del File System.

Nello spazio centrale della finestra, viene riportato il contenuto della directory selezionata nell'albero rappresentato sulla sinistra. Attraverso i menu a tendina che si aprono, cliccando le voci presenti sulla barra degli strumenti è possibile creare, eliminare e modificare i file e le directory. Nello spazio in basso a sinistra della finestra viene riportata la lista dei nodi presenti nel File Sytem distribuito, in modo tale da verificare quali nodi siano ancora connessi al sistema.

3 Gestione della Concorrenza

La concorrenza è stata gestita, dove possibile, tramite i metodi *synchronized* di *Java*. Tuttavia in due casi particolari è stato necessario fare delle scelte differenti:

- Write: nel caso un file sia aperto da un utente, questo file potrà essere essere aperto in sola lettura dai restanti. Per gestire questo caso si è scelto di creare un *Flag* che viene aggiornato nel momento in cui un utente apre il file.
- Verifica replicazione: se ogni nodo verificasse in maniera autonoma l'esistenza delle repliche e cercasse di replicare un file, due nodi potrebbero creare delle repliche in contemporanea. Inoltre dopo la creazione della replica vi sarebbe concorrenza nell'aggiornamento dell'hashmap `fileNodeList`, contenente nella chiave l'UFID del

file e nel valore le locazioni delle copie. Per ovviare a questo problema ogni nodo controlla e verifica l'esistenza delle repliche dei soli file da esso posseduti in locale. In tal modo, poiché il sistema è progetto per gestire il caso *Single-Fault*, esisteranno solamente due copie in due nodi differenti quindi se ogni nodo controlla i file da lui posseduti, solamente lui cercherà di creare una replica nel caso questa manchi. Inoltre nel momento in cui modificherà l'hashMap, cambierà solamente l'entry del file corrispondente senza entrare in concorrenza con le modifiche degli altri nodi.

4 Limiti del Sistema

Durante la progettazione del sistema si sono evidenziati dei limiti legati ai seguenti aspetti:

- La mancanza di un livello di sicurezza tuttavia non necessario poiché l'applicativo non deve essere commercializzato. Una possibile soluzione risulta essere l'aggiunta di un livello di sicurezza, ossia uno strato che si occupi della codifica delle informazioni secondo i metodi diffusamente spiegati in letteratura come il protocollo di crittografia asimmetrica basato sulla combinazione di chiave pubblica e chiave privata.
- La mancanza di un protocollo di autenticazione. Questo perché si è pensato ad un sistema adatto ad un mondo immaginario non avversario. Nel caso di utilizzo nel mondo reale, bisogna prevedere un sistema di autenticazione basato per esempio su qualche altra rete di comunicazione (SMS, email, internet), poiché la rete da noi sviluppata essendo totalmente P2P, non riesce a garantire un locazione stabile dove salvare in modo sicuro le coppie username e password.
- La necessità di sviluppare il sistema appoggiandosi ad una VPN poiché dopo innumerevoli prove l'utilizzo della rete internet non era possibile a causa della presenza di molteplici vincoli legati alla sicurezza e all'accesso alle porte dei Firewall dei sistemi utilizzati.
- La gestione della concorrenza tramite una variabile *Flag* in un sistema distribuito non è la scelta più efficiente e sicura. Inoltre a causa della difficoltà nel verificare la correttezza della gestione della concorrenza, potrebbero esserci casistiche non gestite.

5 Divisione dei compiti

Nello sviluppo del progetto i compiti svolti dai membri del gruppo sono i seguenti:

- Marco Gomiero: sviluppo dell'interfaccia grafica, gestione delle *Properties*, salvataggio e ripristino della struttura del File System tramite file JSON, gestione della struttura del File System, connessione iniziale tra host, bug fixing.

- Luca Rossi: creazione delle routine per l'aggiornamento delle strutture dati condivise (nodi connessi, JSON, lista dei file con locazione), gestione della concorrenza file condivisi, verifica della replicazione.
- Davide Storato: implementazione dei metodi che attuano la replicazione sia dei file che di tutte le strutture dati utilizzate nel sistema sviluppato, gestione della struttura del File System.
- Andrea Ziggiotto: connessione iniziale tra host, sistema di caching, sviluppo comandi base del File System, implementazione del mediator e creazione gestione dei permessi di scrittura.