RPCall

Il seguente elaborato presenta un'implementazione di un modello di rete P2P ibrido per la realizzazione di un'applicazione di chiamata a procedura remota (remote procedure call).

L'applicazione comprende:

. Randezvous: svolge il ruolo di intermediario tra i partecipanti della rete affinché essi possano trovarsi. Tutto ciò che fa è accettare nuove connessioni e salvare i dettagli dei vari endpoint messi a disposizione dai partecipanti fornendoli di volta in volta ai nuovi.

Al fine di rendere l'applicazione fruibile anche da coloro che non hanno metodi da mettere a disposizione, è stato scelto di implementare una differenziazione fra i tipi di peer per cui, all'avvio dell'applicazione, verrà chiesto all'utente il ruolo con cui loggarsi, che può essere di due tipi:

. Nodo: i peer, per diventare dei nodi attivi, dovranno registrarsi al randezvous server ottenendo i dati sui vari partecipanti e fornendogli le informazioni sui propri endpoint dove verranno hostate le relative procedure. Ognuno di essi potrà fare uso delle procedure messe a disposizione degli altri nodi.

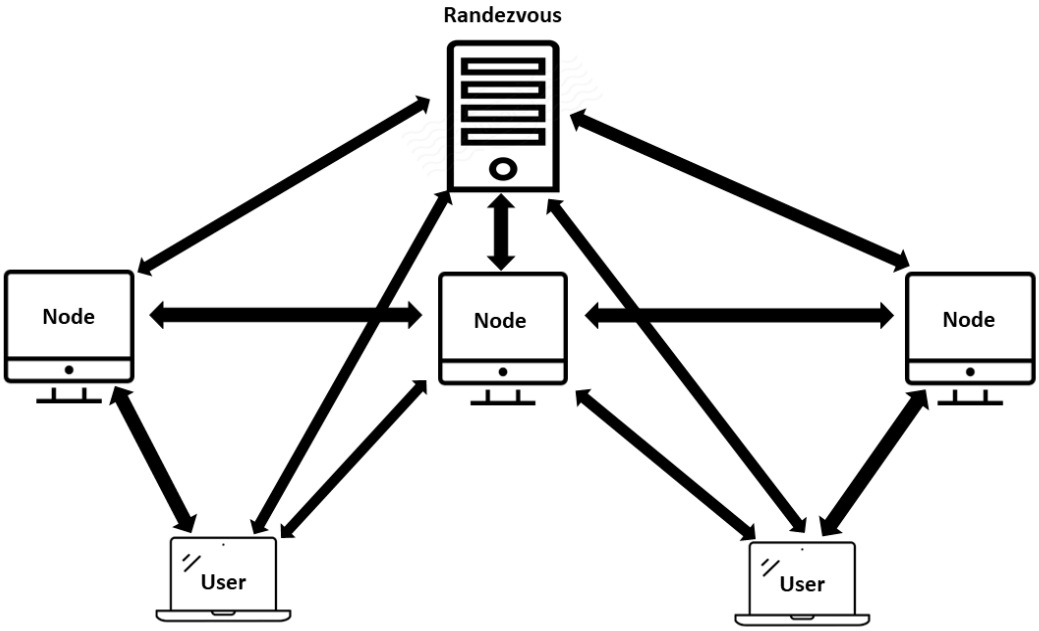
. User: parimenti ai nodi, anche essi effettueranno una connessione al randezvous server per ottenere la lista degli endpoint con relative procedure offerte ed utilizzarle. Tuttavia, non diventeranno partecipanti attivi della rete e le loro informazioni verranno tralasciate.

Traccia originale:

Progettare ed implementare un'applicazione p2p per realizzare il servizio di chiamata a procedura remota (Remote Procedure Call - RPC). Ogni peer mette a disposizione degli altri peer della rete un certo numero (> 2) di procedure che possono essere richiamate da remoto. Per entrare nella rete, ogni peer deve registrarsi presso un server inviandogli la lista delle procedure che mette a disposizione, gli argomenti di ogni procedura ed il numero di porta su cui è disponibile ogni procedura. Ogni peer permette ad un utente di richiamare una procedura remota messa a disposizione da un altro peer. A tal fine, recupera dal server la lista delle procedure remote disponibili nella rete ed in base alla scelta dell'utente, si collega al peer che mette a disposizione la procedura remota selezionata, sulla porta indicata ed inviando gli argomenti richiesti. Il peer esegue la procedura richiesta e

restituisce il risultato, che viene mostrato all'utente.

Architettura di rete



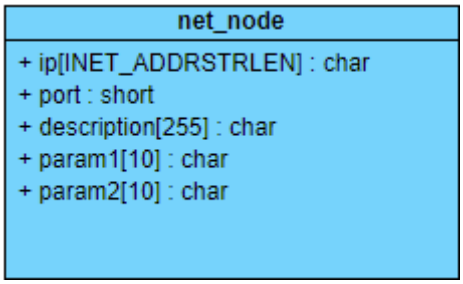
L’architettura di rete è basata su un modello P2P ibrido che prevede la presenza di un server centrale che faciliti lo scambio di endpoint tra i partecipanti. In questo caso, sarà il randezvous server a svolgere il ruolo di punto d’incontro per i vari peer, e, sebbene rappresenti un SPF (Single point of failure) in caso di crash o di disconnessione improvvisa, i partecipanti potranno continuare ad offrire/utilizzare le procedure remote in modo completamente decentralizzato, ma non sarà possibile unirsi alla rete.

Tutte le comunicazioni sfruttano il protocollo TCP e socket connesse per una maggiore affidabilità.

Protocollo applicazione

All'avvio di un peer bisognerà fornirgli un intero (0 o 1) per scegliere la modalità di accesso: 0 per la modalità Node, 1 per la modalità utente ed infine l'indirizzo ip del server.

L'identificazione dei nodi verrà effettuata tramite la seguente struct net_node:



Fase di join della rete

Caso modalità Node:

Il nodo provvederà a riempire i campi delle proprie struct net_node rappresentanti gli endpoint e le informazioni sui metodi messi a disposizione. In particolare, verranno settati numero di porta, nome, descrizione del metodo e, assumendo che tutte le procedure prendano in input due argomenti, due stringhe rappresentanti il tipo di parametri accettati dalla procedura. Il campo riservato all'indirizzo ip, invece, verrà riempito dal server nel caso della registrazione o da un peer ricevente una richiesta da un nodo a lui sconosciuto (entrato in rete in un secondo momento e non presente nella lista fornita dal server).

Il nodo effettuerà in maniera del tutto sequenziale la connessione al randezvous server fornendogli il tipo di modalità che, in questo caso, sarà 0 affinché il server sappia che si tratti di un nodo e non di un utente, e, se bisognerà aggiungerlo alla lista, questo avverrà solo nel caso in cui il peer sia un nodo e l'indirizzo ip sia sconosciuto al server stesso, altrimenti si passerà direttamente all'invio della lista.

Una volta appurata la natura del peer, il server, qualora vi siano già altri nodi, fornirà un intero rappresentante la lunghezza della lista affinché il peer sappia per quante volte iterare la lettura del pacchetto di tipo net_node. In caso contrario, non avendo endopoint in lista, il server invierà un intero =0, cosicché il nodo rimanga in attesa di nuove richieste.

Terminato questo step, il peer invierà le proprie informazioni tramite tre pacchetti di tipo net_node (uno per ogni procedura offerta) che il server aggiungerà alla propria lista. Nel caso in cui il peer si connetta nuovamente al server in seguito ad una disconnessione, il server rileverà la presenza del peer nella propria lista e provvederà

a crearne una versione temporanea in cui quest'ultimo non sia presente e, seguendo un approccio analogo al caso precedente, comunicherà al peer un intero rappresentante la lunghezza della nuova lista temporanea, replicando l'invio della stessa e distruggendola in seguito.

A questo punto il peer è a tutti gli effetti un partecipante attivo della rete, che può usare le procedure offerte dagli altri partecipanti ed offrire le proprie, comunicando in modo del tutto decentralizzato con altri nodi.

Caso modalità user:

L'user svolgerà un ruolo di partecipante passivo in quanto non avrà nulla da offrire alla rete. In tal caso, infatti, la fase di registrazione al server differirà da quella del nodo solo per quanto concerne l'invio delle proprie informazioni, che verranno tralasciate. Dopo aver ottenuto dal randezvous server gli endpoint dei partecipanti, l'user potrà utilizzare tutte le procedure a lui note e disponibili in quel momento.

Hosting dei metodi

Ognuno dei nodi metterà a disposizione della rete 3 procedure, ognuna delle quali rispettivamente sulle porte 4443, 4444, 4445.

Quando vi è una richiesta, il nodo controllerà la natura del peer chiamante verificando se si tratti di un altro nodo o un utente.

Nel caso si tratti di un utente fornirà un intero = 1 per comunicargli che può inviare i parametri che saranno passati al metodo d'interesse. Detti parametri saranno passati sottoforma di stringhe, questo perché il C non supporta tipi di dati dinamici e, al fine di una migliore modularità, le stringhe offrono un ottimo compromesso, sarà poi il nodo a convertire le stringhe nel giusto tipo. Una volta ricevuti e convertiti, i parametri saranno passati come argomenti al metodo e il risultato verrà scritto in una stringa rappresentante l'output della procedura che il ricevente convertirà nel giusto tipo, come avviene per la ricezione dei parametri. Infine, verrà mostrato il valore ottenuto dal nodo a schermo.

Nel caso la richiesta arrivi da un altro nodo possono verificarsi due scenari:

- se il nodo è sconosciuto oppure è il primo a contattarci, gli passeremo un intero = 0, questo indicherà al nodo che avremo bisogno delle sue informazioni per popolare la lista. Solo successivamente si potrà passare alla fase di ricezione dei parametri ed invio del risultato.

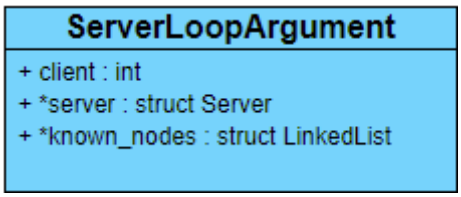
- Se il nodo è già nella lista gli passeremo, come nel caso dell'utente, un intero = 1 che indicherà la disponibilità a passare direttamente alla fase di invio dei parametri.

Dettagli implementativi del server

All'avvio il randezvous inizializza un oggetto di tipo server che rappresenta un wrapper per una Listen Socket che verrà utilizzata per accettare nuove connessioni, caricherà il file peers.dat che altro non è che una copia di backup della lista degli endpoint della rete, e, qualora ve ne fossero salvati da un'esecuzione precedente, provvederà ad inizializzare una LinkedList con essi; in caso contrario, la lista sarà creata e si attenderà una nuova connessione per popolarla.

È stato scelto di implementare una LinkedList di puntatori a void al fine di avere una sorta di template di array dinamico che supportasse metodi di inserimento, estrazione, rimozione e un indice interno alla struttura aggiornato a runtime che ne rappresenti la dimensione.

Il main thread sarà dedicato esclusivamente all'accettazione delle connessioni da parte dei peer e, ad ogni nuova connessione, provvederà a riempire i campi della struct ServerLoopArgument:



I quali rappresentano rispettivamente:

- client: il descrittore ritornato dall'accept().
- *server: puntatore alla struct server inizializzata all'avvio del randezvous da cui reperire l'indirizzo ip del peer che ci ha contattato.
- *known_nodes: puntatore alla LinkedList dove andremo a salvare le informazioni dei vari peer.

Fatto ciò, l'oggetto di tipo ServerLoopArgument sarà passato come parametro al thread incaricato di gestire la richiesta. È stata preferita l'implementazione di un modello multithreaded, al posto di un modello multi-processo, in quanto i thread condividono la memoria e, in questo caso, non vi è la necessità di sincronizzare gli accessi dato che non vengono usate variabili o strutture dati globali, fatta eccezione per il file peers.dat; al contrario, un nuovo processo avrebbe ricreato una copia di ogni

variabile utilizzata, con maggior utilizzo di memoria e conseguente overhead, nonché rischi di race conditions.

Dettagli implementativi dei peer

Come già descritto in precedenza, il peer all'avvio necessiterà di un intero (0 o 1) passato come argomento da riga di comando: questo servirà a differenziare i nodi dagli utenti, rendendo la rete aperta anche a coloro che non hanno un metodo da mettere a disposizione.

La routine dell'utente prevede l'inizializzazione della LinkedList, una prima connessione al server, il reperimento degli endpoint qualora vi fossero, altrimenti bisognerà provare a riconnettersi in un secondo momento; dopodiché, l'utente potrà usare una qualsivoglia delle procedure messe a disposizione dai nodi a lui noti effettuando una richiesta (user_request()), il tutto in maniera sequenziale dal momento che sarà egli stesso a contattare gli altri e mai ad essere contattato.

Per quanto riguarda i nodi verrà inizializzata una LinkedList che ospiterà gli endpoint ricevuti dal server e dai nodi sconosciuti che ci hanno contattato, dopodichè verranno creati 3 thread ognuno dei quali sarà incaricato di accettare una connessione ed in modo simile a quanto avviene con l'I/O multiplexing creando un nuovo thread passandogli la struct ServerLoopArgument per gestire tale richiesta solo quando necessario, in modo del tutto analogo a quanto accadeva nel server.

Il main thread si occuperà di tutta la parte relativa all'utilizzo di metodi remoti e invio dei parametri. Per cui, verranno stampati a video tutti gli endpoint presenti in lista e sarà possibile scegliere uno di essi, estrarlo dalla LinkedList per ottenere i dati ed effettuare una richiesta (node_request()).

Manuale utente

Guida alla compilazione

Per la compilazione sarà sufficiente dirigersi nella directory */rpCall* e lanciare il comando *make*, verranno così generati gli eseguibili da lanciare con *./randezvous* e *./peer*.

All'avvio del peer bisognerà fornirgli un intero (0 o 1) per scegliere la modalità di accesso, ricordiamo, 0 = node_mode e 1 = user_mode, seguiti dall'indirizzo ip del randezvous.

Esecuzione

Per eseguire l'elaborato verrà utilizzato katharà, un emulatore di ambiente di rete che richiede i seguenti step per l'installazione:

1- Installare Docker (https://docs.docker.com/engine/install/).
2- Installare xterm terminal emulator (sudo apt install xterm or sudo yum install xterm or sudo pacman -S xterm).
3- Installare la giusta versione di Kathara (https://github.com/KatharaFramework/Kathara/wiki/Installation-Guides).

Una volta installate tutte le componenti, basterà dirigersi nella cartella */lab* e lanciare il comando *kathara lstart.* Creati i device, da ognuno di essi, spostarsi nella directory */shared/project* dove saranno presenti i sorgenti.

Lanciare il comando *make* dal device randezvous e, successivamente, *./randezvous,*

dagli altri device *./peer* <modalità di utilizzo> <ip del randezvous>.

Per chiudere il laboratorio e relativi device sarà sufficiente lanciare il comando *kathara wipe* .