Piattaforma di file sharing peer to peer

Lazzari Matteo
matteo.lazzari7@studio.unibo.it

September 28, 2022

Indice

1 Obiettivi **4**
1.1 Scenari di utilizzo 4

2 Analisi dei requisiti **6**
2.1 Funzionali 6
2.2 Non funzionali 7
2.3 Architetturali 7

3 Progettazione **8**
3.1 Considerazioni iniziali sulla struttura 8
3.2 Progettazione delle componenti 9
3.3 Struttura delle iterazioni 10
3.3.1 User 11
3.3.2 Server 13
3.3.3 Orchestratore 14

4 Dettagli funzionamento **15**
4.0.1 Orchestratore 15
4.0.2 Server 16
4.0.3 User 17

5 Test **18**

6 Utilizzo **19**

7 Conclusioni **20**
7.1 Progetti futuri 20
7.2 Cosa ho imparato 20

Introduzione

L'idea alla base del progetto è quella di sviluppare una rete di file sharing scalabile, in modo tale da poter funzionare in qualsiasi ambiente essa venga posta, da reti di piccole dimensioni a grandi.
La rete sviluppata si ispira a reti attualmente esistenti, quali BitTorrent. Infatti, come trattato successivamente, alcuni meccanismi di consivisione hanno avuto come progettazione iniziale una visione del funzionamento dello stesso meccanismo sulla rete BitTorrent.
Alla base del progetto si è tenuto in considerazione la nacessità di garantire l'anonimato delle risorse pubblicate, facendo si di mascherare le risorse di cui non si hanno interesse.

1 Obiettivi

Fornire agli utilizzatori una rete stabile in grado di gestire la condivisione all'interno della rete, garantendo la persistenza delle risorse anche con cambiamenti all'interno della rete e bilanciando il carico delle risorse sui server della rete.

1.1 Scenari di utilizzo

Lo user può eseguire azioni di:

- Download di risorse;
- Condivisione di risorse;
- Terminare la condivisione.

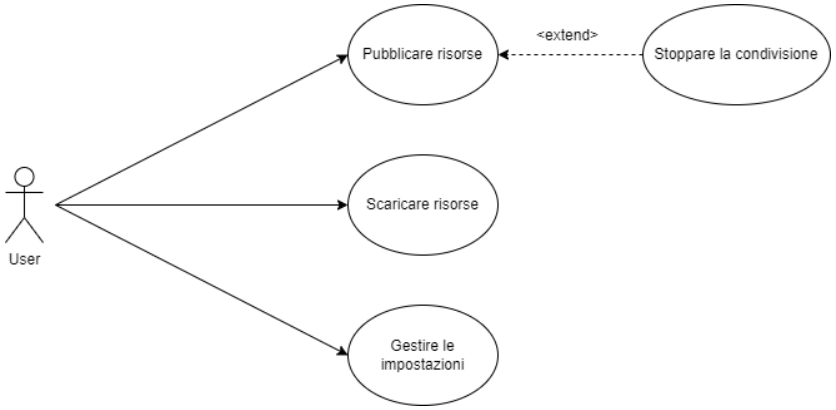

Figure 1: Azioni disponibili per gli utenti

Il server può eseguire azioni di:

- Visualizzazione di informazioni;
- Terminare l'esecuzione.

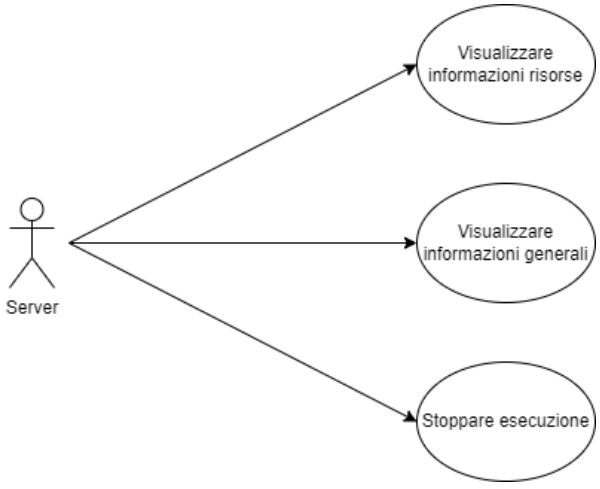

Figure 2: Azioni disponibili per il gestore del server

L'orchestratore può eseguire azioni di:

- Visualizzazione di informazioni;
- Terminare l'esecuzione (la rete andrà offline).

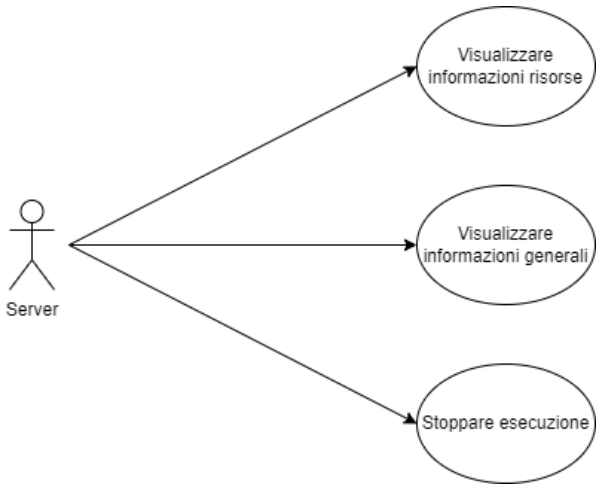

Figure 3: Azioni disponibili per il gestore dell'orchestratore

2 Analisi dei requisiti

Nella fase iniziale di progettazione del progetto, sono stati analizzati quali sono i principali requisiti funzionali e non funzionali che la rete deve avere al termine dello sviluppo.
Successivamente sono elencati tutti i requisiti identificati per le varie componenti della rete. Di questi poi verrà descritta l'implementazione successivamente.

2.1 Funzionali

Per quanto riguarda l'utente sono stati evidenziati 5 punti principali:

- L'utente finale deve avere la possibilità di scaricare risorse dalla rete, specificando semplicemente il nome della risorsa. Questi, per questioni di privacy e funzionamento, saranno criptate, per poi essere salvate sui server che gestiranno la relativa risorsa;
- L'utilizzatore della rete (peer) sarà in grado di condividere risorse che sono disponibili sul computer. Questo requisito non prevedere il versioning delle risorse (non gestito in questo progetto), ma solo la possibilità di pubblicare la risorsa e il download di esso da parte di altri peer che la richiedono. Inoltre, il programma deve essere in grado di verificare se le risorse che sono state pubblicate siano ancora reperibili all'interno del dispositivo o sono state spostate/rimosse;
- Deve essere fornita la possibilità allo user che pubblica una risorsa di terminare la condivisione delle risorse conviside a lui desiderate;
- Deve essere gestita la persistenza dei dati pubblicati. Questo prevede l'automatica ripubblicazione delle risorse condivise quando l'utente si riconetterà alla rete dopo essersi disconnesso;
- Modificare le impostazioni di base dell'applicativo (es. path del download del file).

I server all'interno della rete sono responsabili di tenere traccia delle risorse. per questo sono stati pensati 3 punti principali nello sviluppo:

- Il server deve tenere traccia delle risorse che sono state pubblicate nella rete e sono salvate nel server. Queste risorse contengono il nome della risorsa pubblicata (il nome è criptato) e tutti i relativi peer che la condividono;
- Il server deve verificare periodicamente che le risorse pubblicate sul server siano ancora reperibili. Per questo contatterà i peer per verificare se sono ancora connessi e la risorsa è ancora disponibile;
- Visualizzare le informazioni generali del server (es. porta di ascolto, numero risorse pubblicate...).

L'orchestratore all'interno della rete ha un ruolo di gestore dei server. Per questo ha solo 2 funzioni generali:

- Tenere traccia di tutte le inforamzioni dei server che sono connessi alla rete;
- Assegnare ad ogni server gli indici (lettere) delle risorse che deve gestire.

2.2 Non funzionali

Il sistema prevede un'interfaccia semplice per tutte 3 le entità e funzionalmente completa basata su prompt dei comandi.
L'architettura del sistema deve essere scalabile al fine di adattarsi ai possibili cambiamenti della rete.
Il sistema infine deve essere sicuro, infatti nessuno deve avere la possibilità di conoscere o analizzare quali sono i file che sono condivisi nel network.

2.3 Architetturali

Il sistema è basato su un'architettura peer to peer, costituita da 3 entità differenti:

1. Orchestratore: Ne è presente soltanto uno all'interno della rete ed è gestito da chi mette a disposizione il network. Tiene traccia di tutti i server connessi e gestisce gli indici assegnati ai server;

2. Server: Possono essere avviati più server all'interno della rete e tutti comunicano con l'orchestratore per ottenere le risorse da gestire. Tengono traccia delle risorse che sono disponibili nel network e quali peer le condividono;

3. Peer: Viene gestito dall'utente finale, ha la possibilità di interfacciarsi alla rete per eseguire le azioni descritte nei requisiti funzionali. È direttamente responsabile del download delle risorse, in quanto viene instaurata una comunicazione diretta tra i due peer, rendendo i server una entità keyhandler.

3 Progettazione

I componenti all'interno della rete eseguono azioni in automatico oppure in base all'input dell'utente. Le figure 4/5/6 mostrano a livello generale le iterazioni che avvengono tra le entità della rete quando vengono eseguite determinate azioni, seguite successivamente da una descrizione dello scambio di messaggi e della loro sintassi. Le azioni possono essere di 2 tipi:

- Attive (pallino pieno): Sono azioni che vengono eseguite manualmente dall'utente, tramite l'utilizzo dei comandi (es. pubblicazione di una risorsa, rimozione della condivisione di un elemento);
- Passive (pallino vuoto): Sono azioni che vengono eseguite automaticamente dall'applicazione (es. verifica da parte del server che un peer è ancora connesso).

3.1 Considerazioni iniziali sulla struttura

Nella fase iniziale del progetto sono state analizzate reti con funzionamento simile a quello desiderato (es. BitTorrent, Freenet). Da questa analisi si sono prese importanti decisioni sul progetto, quali l'utilizzo di server come keyholder e la necessità di criptare il nome della risorsa per mascherare il contenuto della rete (anche se l'ip del peer viene memorizzato in paintext).

La progettazione ha incluso anche il determinare quali azioni l'orchestratore, i server e i peer possono fare nella rete, arrivando alla conclusione che soltanto i peer possono eseguire azioni sulle risorse del network. Inoltre è stata presa la decisione che l'orchestratore deve rimanere sempre online, essendo il punto di accesso alla rete per determinare i server online. Se l'orchestratore dovesse spegnersi, la rete risulterebbe offline.
Su quest'ultimo punto è stato pensato anche di implementare un sistema di election all'interno della rete, per far si che un server possa trasformarsi in orchestratore ed evitare che la rete andasse offline.

È stato deciso di far gestire ad ogni server solo un numero limitato di risorse, in modo da dividere il carico tra i vari server che sono connessi alla rete. Questo fa si che i server devono comunicare tra di loro nel caso in cui si ha un cambiamento all'interno della rete (disconnessione/connessione di un server), in quando le risorse da gestire di un determinato server cambiano al cambiare della rete. Ciò porterebbe alla necessità di inviare ad altri server le risorse che non sono più in gestione del server attuale.
In reti di dimensioni maggiori si ha un minor carico per ogni server online, in quando all'aumentare del numero dei server, ognuno di essi deve gestire un numero di risorse inferiore.

Infine, è stata presa la decisione di non gestire il versioning delle risorse all'interno della rete. Infatti, ogni risorsa pubblicata all'interno della rete è a sè stante.

3.2 Progettazione delle componenti

All'interno della rete ogni entità è stata progettata per con le proprie funzionalità e specifiche di comunicazione tra di esse:

- Orchestratore: viene utilizzato come punto di accesso alla rete. L'orchestratore ha come unico ruolo tenere traccia e gestire i server. Infatti l'orchestratore è il responsabile dell'assegnare quale risorsa ogni server gestirà. Se l'orchestratore si dovesse spegnere o crashare, allora tutta la rete andrà offline.
  L'orchestratore tiene aperte le N connessioni con gli N server che sono online in un determinato momento. In questo modo:
  - Se avviene una chiusura di una connessione, l'orchestratore sa che un server è andato offline e quindi avvierà la procedura di assegnamento delle risorse ai server. Queste poi saranno inviate tramite i canali ancora aperti;
  - Se arriva un messaggio di connessione da parte di un nuovo server, l'orchestratore aggiungerà la nuova connessione col server a tutte le connessioni esistenti ed avvierà la procedura di assegnamento delle risorse ai server. Queste poi saranno inviate tramite i canali ai vari server;

  Lo scambio di messaggi tra le varie entità verrà descritto successivamente nella sezione dedicata;

- Server: è un peer speciale della rete che viene utilizzato come keyholder delle risorse pubblicate. Ogni server può gestire solo delle determinate risorse (che vengono specificate dall'orchestratore alla connessione della rete e durante il periodo online)
  Il server, quando viene avviato, apre una comunicazione con il server, che manterrà sempre aperta per tutto il tempo in cui il server resta online (se questa comunicazione si chiude per qualsiasi motivo, il server viene chiuso automaticamente).
  Oltre alla connessione con l'orchestratore, il server apre una comunicazione per ogni utente che si connette ad esso. Questa però resta aperta solamente per il periodo in cui avviene lo scambio di messaggi tra le due parti, per poi essere chiusa.
  Lo scambio di messaggi tra le varie entità verrà descritto successivamente nella sezione dedicata;

- User: peer generico della rete. Può eseguire le azioni che sono descritte descritte nei requisiti funzionali e interagisce con l'orchestratore quando si connette per la prima volta alla rete oppure quando individua una desincronizzazione delle risorse dei server.
  Infatti, ogni volta che viene mandato un messaggio di ottenere/rimuovere/pubblicare una risorsa su un server, quest'ultimo può rispondere con il contenuto richiesto o con un messaggio di fallimento. In caso di fallimento, il peer aspetta 1 secondo prima di riprovare nuovamente il messaggio. Dopo X fallimenti (specificato nel codice, di default: 20), il peer si considera desincronizzato e contatta l'orchestratore per ottenere nuovamente la lista dei server.
  Il peer non ha nessuna connessione aperta fino al momento in cui contatta il server o l'orchestratore. In entrambi i casi, quando poi lo scambio di messaggi è completo, viene chiusa la connessione.

Lo scambio di messaggi tra le varie entità verrà descritto successivamente nella sezione dedicata.

3.3 Struttura delle iterazioni

In seguito sono riportate le iterazioni che avvengono tra le entità della rete ogni volta che viene eseguita una delle azioni mostrate.
Sono anche descritti i messaggi e le relative sintassi che vengono utilizzate per ogni tipo di iterazione che avviene con le altre entità della rete.
È importante sottolineare che le azioni svolte dai 3 componenti della rete si dividono in 2 tipi:

- Attive (eseguite manualmente dall'utente), che sono visualizzate con un pallino pieno;
- Passive (eseguite automaticamente dal programma), che sono visualizzate con un pallino vuoto.

3.3.1 User

Nella Figura 4 sono mostrate le iterazioni che l'applicazione dello user esegue quando l'utente fa delle operazioni.

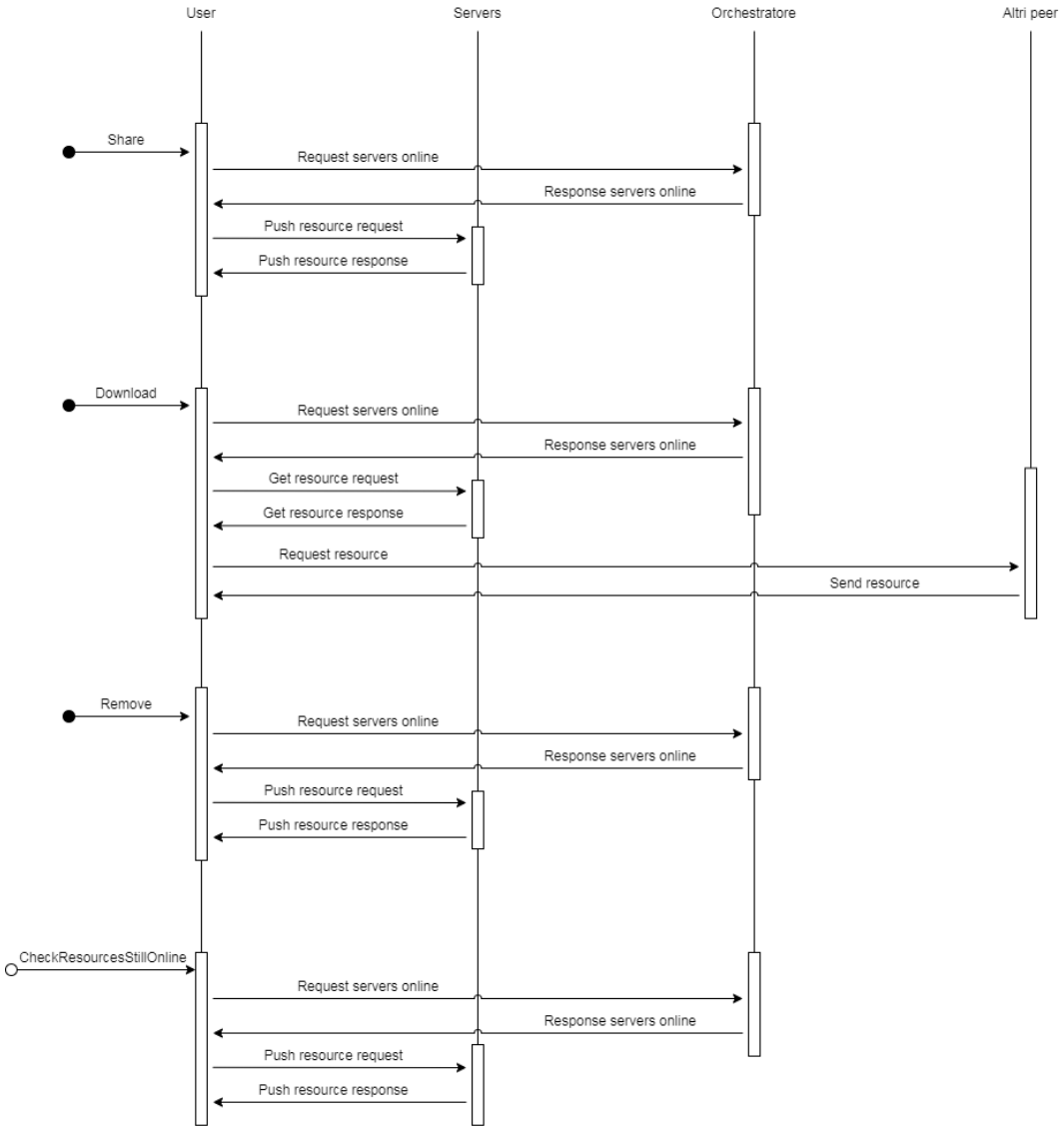

Figure 4: Struttura delle comunicazioni per le azioni dello user.

Nella Figura 4 è mostrato lo scambio di messaggi che avviene tra lo user e le entità della rete quando vengono utilizzate le funzionalità dello user. Qui sotto è riportata la lista della sintassi di ogni messaggio che viene scambiato. La descrizione del funzionamento sarà trattata successivamente nella descrizione dell'utente:

- Request/response servers online: Questa azione viene eseguita dal peer aprendo una connessione con l'orchestratore, inviando un messaggio specificando di essere un peer. Il server risponderà al peer mandando un messaggio con la seguente sintassi:
  > [risorsa gestita],[indirizzo ip]-[porta];[risorsa gestita],[indirizzo ip]-[porta]; ...
  che tramite un opportuno parse, è in grado di ottenere tutte le informazioni necessarie. Questo meccanismo viene utilizzato per tutte le azioni.

- (SHARE) push resource request/response: Il peer manda un messaggio al rispettivo server con la sintassi

> [publish]-[resource name crypted]-[peer port]
Il server poi risponderà al peer con un semplice messaggio di successo/errore dell'azione.

- (DOWNLOAD) get resource request/response: Il peer invia al server il messaggio strutturato come
> [getResource]-[resource name crypted]
Il server poi risponderà al peer in diversi modi:

  1. notMine: Le informazioni sui server sono sbagliate ed è necessario refreshare le informazioni dei server;
  2. notThere: Il server non ha informazioni sulla risorsa richiesta (anche se gestisce quell'indice);
  3. altro: Il server ha appena inviato l'indirizzo di tutti i peer che gestiscono l'informazione. Questo messaggio è nel formato: > [Indirizzo ip]-[Porta]=[Indirizzo ip]-[Porta]= ...

- (DOWNLOAD) request/send resource: Avviene il vero e proprio scambio di file all'interno di questa transizione. Il peer manda una richiesta nel formato
> Peer%[Nome risorsa criptato]
e l'altro peer risponderà con un messaggio

  1. gotIt: Il peer ha la risorsa e la condivide. Avvia lo scambio di messaggio.
  2. missing: Il peer non condivide la risorsa / non ce l'ha e c'è stato desync del server.

- (REMOVE) push resource request/response: La sequenza di messaggi scambia le informazioni per eliminare l'indentificativo del peer dalla risorsa pubblicata. In particolare, il peer invia
> removeResource-[Nome risorsa criptato]
come richiesta, mentre il server risponderà con un messaggio di:

  – done: Rimozione dell'ip dalla risorsa avvenuto;
  – notMine: Le informazioni sui server sono sbagliate ed è necessario refresharle.

- (CHECKRESOURCESSTILLONLINE) push resource request/response: Esegue le stesse azioni (e scambio di messaggi) del caso (SHARE)

Per il peer è sempre importante avere una visione aggiornata della rete. Per evitare spreco di risorse e sovraccarico, quest'ultima viene aggiornata solamente nel momento in cui il peer esegue un'azione attiva sulla rete (è inutile aggiornare in modo continuo ogni X secondi i server nella rete, se poi non si eseguono azioni. Porta solo ad un maggior carico inutile sulla rete).
Nel caso in cui si abbia un cambiamento della rete durante una condivione/download e non venga trovata la risorsa, il sistema è progettato per rieseguire la relativa azione per un numero determinato di volte (di default 20), prima di considerare l'azione non andata a buon fine. Questa automatica riesecuzione dell'azione avviene in background e non mostra nessun avviso all'utente (se non notare un ritardo nella ricezione di un feedback positivo/negativo del risultat dell'esecuzione) e viene reseguita ogni 1s dal fallimento della chiamata precedente. Questo delay è stato impostato per aumentare la probabilità di successo dell'azione (se eseguita in modo continuo senza delay, non si da tempo alla rete di aggiornarsi ad un possibile cambiamento interno, andando sicuramente a fallire in quanto le risorse non sono disponibili dove ce le si aspetta).

3.3.2 Server

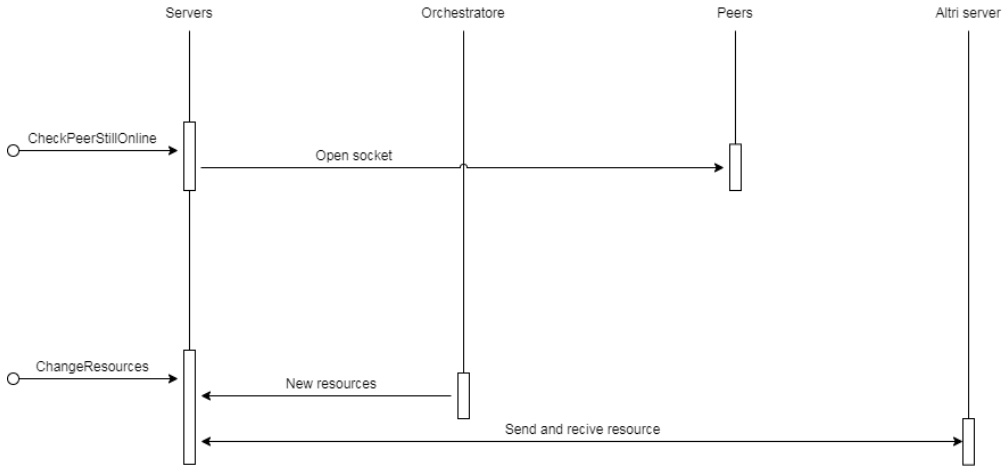

Figure 5: Struttura delle comunicazioni per le azioni del server.

Il server è stato progettato per non avere comportamenti attivi sulla rete, quali rimozione di risorse. Questo fa si che le uniche azioni possibili tra componenti interni alla rete sono eseguite automaticamente in background.
Successivamente è riportata la sintassi dei messaggi che vengono scambiati con le altre componenti della rete. I dettagli di funzionamento saranno descritti successivamente nella descrizione del server.

- (CHECKPEERSILLONLINE) openSocket: Il server apre una socket con il peer per verificare se è ancora online. In caso di successo, viene inviato un semplice messaggio
  > Server
  per specificare l'entità che lo sta contattando.

- (CHANGERESOURCES) new resources: Viene ricevuto in qualsiasi momento dall'orchestratore e prevede una sintassi del tipo
  > [indici risorse]

- (CHANGERESOURCES) send and recive resources: Il server invia e riceve agli altri server della rete messaggi del tipo
  > handleThisResource-[resource name][[peer ip]![peer port]=[peer ip]![peer port]= ...]

3.3.3 Orchestratore

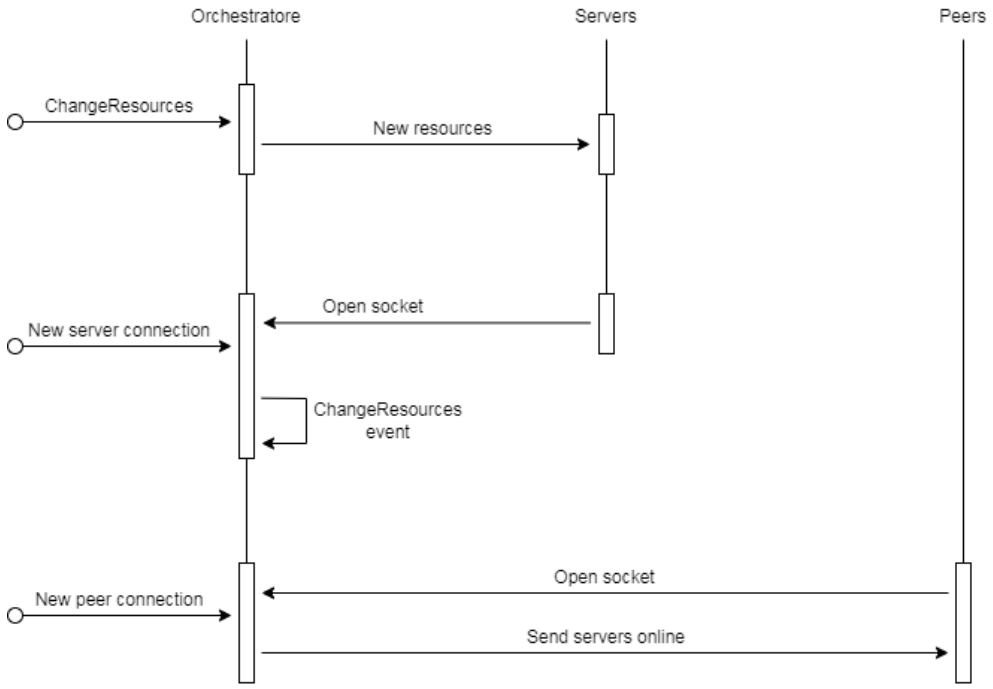

Figure 6: Struttura delle comunicazioni per le azioni dell'orchestratore.

L'orchestratore è stato progettato e sviluppato senza la possibilità di interagire con le altre entità della rete, in quanto il suo unico scopo è quello di tenere traccia dei server connessi e delle loro informazioni. Per questo tutte le attività sono eseguite automaticamente dal programma quando si hanno nuove connessioni da parte di server e peers.
In seguito è descritta la sintassi dei messaggi che sono scambiati con i server e i peer. Una loro maggior descrizione sarà trattata successivamente nella descrizione dell'orchestratore.

- (CHANGERESOURCES) New resources: L'orchestratore invia ad ogni server un messaggio con i corrispettivi nel formato
  > [indici risorse]
  e una volta inviate tutte le risorse, manda un altro messaggio nel formato
  > [Risorsa server]:[Server ip]:[Server port]%[Risorsa server]:[Server ip]:[Server port]% ...
  per far attivare i server nello scambio di risorse;
- (NEWSERVERCONNECTION): Utilizza la stessa sintassi dei messaggi di (CHANGERESOURCES);
- (NEWPEERCONNECTION): Tutto parte da un'apertura della socket e dall'invio del peer di un messaggio
  > Peer
  L'orchestratore poi invierà al peer le informazioni dei server nel formato
  > [Risorsa server]:[Server ip]:[Server port]%[Risorsa server]:[Server ip]:[Server port]% ...

4 Dettagli funzionamento

La descrizione dei dettagli sul funzionamento del progetto seguirà il seguente ordine:

1. Descrizione dell'orchestratore
2. Descrizione del server;
3. Descrizione dello user;

4.0.1 Orchestratore

L'orchestratore è l'entità centrale della rete. Infatti, nel caso in cui l'orchestratore risulti offline oppure diventi offline, l'intera rete si spegnerà in quanto non sarà più possibile tenere traccia della struttura interna del network. Questo ha fatto si di impostare l'orchestratore come punto fisso della rete, impedendo di modificare la porta su cui è in ascolto (porta 10000).
L'orchestratore ha il solo ed unico compito di tenere traccia di quali risorse i server devono gestire (è presente una lista che tontiene un riferimento serverIP-risorse). Non tiene traccia delle risorse pubblicate sui server, ma solamente delle lettere generate per i server (chiamate successivamente indici), i quali accetteranno solo le risorse che iniziano per tali indici.

```
Calculate number of indexes to be assigned for each server
for each server, do:
    Get first available letters, using a counter as index pointer of the letter
for each server, do:
    Send the new indexes to the corresponding servers
    Attach to a string all servers information, based on the structure   R%R%R.... with each R be [indexes]:[ip]:[port]
for each server, do:
    Send the string with all servers information
```

Figure 7: Pseudocodice del calcolo dei nuovi indici e invio di essi.

Questo fa si che non esistano entità all'interno della rete che sono in grado di conoscere tutte le risorse pubblicate in un determinato momento (ad eccezione del caso in cui si ha solo 1 server connesso che gestice tutte le risorse).
L'orchestratore conosce solo determinate informazioni della rete, quali:

- Numero di server connessi alla rete;
- Informazioni di ogni server. Queste informazioni sono composte da:
  - Socket del server, usata per la comunicazione e per determinare quando un server si disconnette;
  - Porta di ascolto del server;
  - Indici delle risorse gestite dal server.

All'interno dell'orchestratore non sono presenti informazioni di alcun tipo riguardanti i peer connessi.

L'orchestratore all'interno della rete ha come unica funzione accettare nuove connessioni e restituire le informazioni che sono richieste.
Ogni volta che un server si connette/disconnette dalla rete, vengono ricalcolati tutti gli indici in base a quanti server sono ancora connessi. Questa procedura richiede più esecuzioni nel momento in cui si ha la connessione in simultanea di più server (evento raro, ma non impossibile, come avviene nel caso di avvio tramite script).
Nel caso di connessione di un peer, l'orchestratore restituisce l'elenco di tutti i server che sono presenti nella rete, con le relative risorse (è possibile visionare la sintassi dei messaggi nella sezione dedicata.

4.0.2 Server

Il server è una entità "di passaggio" all'interno della rete. Infatti l'unico ruolo del server è quello di mantenere una lista chiave-valore di tutte le risorse che gestisce all'interno della rete. Per gestire ciò, all'interno del server è presente una lista contenente tutte le risorse pubblicate sul server e, insieme a ciascuna di esse, è presente un'altra lista contenente tutti gli indirizzi ip dei peer che la condividono. Quando un server viene avviato, come prima cosa contatta l'orchestratore della rete (è possibile visionare la sintassi dei messaggi nella sezione dedicata). Questo passaggio permette di ricevere i nuovi indici che saranno usati per validare le risorse che vengono inviate al server e anche per verificare che la rete sia attiva. Infatti, se al momento della connessione, l'orchestratore risultasse offline (o andasse offline nel tempo), il server termina l'esecuzione immediatamente.

Come detto precedentemente nella descrizione dell'orchestratore, ad ogni server sono associati degli indici che corrispondono all'iniziale delle risorse che il server deve gestire (es. se ad un server sono associate le lettere abc, il server gestirà solamente le risorse che iniziano per a/b/c). Gli indici sono calcolati dall'orchestratore e non possono essere modificati in alcun modo, come le risorse che sono salvate all'interno del server.

Il server non ha funzioni attive nella rete (es. rimozione di una risorsa pubblicata, disconnessione di peer) in quando ha solo il ruolo di key-holder. Tutti i comandi che può eseguire mostrano solamente informazioni riguardanti lo stato del server e delle risorse.

Il server è responsabile di verificare che i peer che pubblicano una risorsa siano ancora disponibili e raggiungibili all'interno della rete. Infatti, ogni X secondi (definito nel codice), il server apre una connessione con ogni singolo peer di ogni risorsa per verificare che il peer sia ancora connesso. Se l'apertura della connessione va a buon fine, allora il peer è ancora connesso, altrimenti in caso di errore, il peer viene eliminato dalla lista.

Ogni volta che si ha un cambiamento nella rete, il server riceve dall'orchestratore una lista contenente l'indirizzo di tutti gli altri server online e gli indici delle risorse che essi gestiscono. Infatti, ogni volta che gli indici cambiano, il server deve controllare se ci sono risorse pubblicate che non vengono più gestite (la lettera iniziale della risorsa non rientra tra gli indici del server) e inviarle ai server che le gestiranno.

Ogni volta che il server riceve un messaggio di pubblicazione, il server controlla l'iniziale del nome della risorsa (indifferentemente che sia un peer o un altro server ad avergliela mandata). Nel caso in cui la lettera iniziale non coincida con una delle lettere gestite dal server, la richiesta viene scartata e rinviato un messaggio di errore. Altrimenti, viene verificato se la risorsa è già stata pubblicata. Nel caso sia già presente, aggiunge i peer mandati alla lista dei peer già presenti, altrimenti aggiunge la risorsa alla lista di tutte le risorse.

4.0.3 User

Lo user (o peer) è l'unica entità funzionale all'interno della rete. Solamente lui ha la possibilità di pubblicare/rimuovere risorse all'interno della rete.

Il peer, quando si connette alla rete, richiede all'orchestratore tutti i server che sono attualmente online, per crearsi una lista di contatti interna (è possibile visionare la sintassi dei messaggi nella sezione dedicata). Come per il server, se il peer non è in grado di contattare l'orchestratore o diventasse offline nel tempo, il peer viene terminato immediatamente.

Il peer, all'avvio, imposta i settings che sono presenti nel file config.txt, permettendo di usare impostazioni personalizzate. Inoltre, quando il peer ha caricato le impostazioni, vengono automaticamente pubblicate tutte le risorse che aveva scaricato e che nel momento della precedente chiusura, erano rimaste pubblicate. Questo sistema è stato implementato per aumentare la sopravvivenza di una determinata risorsa nella rete. Insieme a questo meccanismo (e per lo stesso scopo), tutte le risorse che sono state scaricate vengono automaticamente pubblicate. Nel caso lo user decidesse di terminare la condivisione della risorsa, questa non verrà più automaticamente caricata all'avvio del programma.
Le operazioni che possono essere eseguite nella rete sono:

- Pubblicare una risorsa: Ogni volta che il peer vuole pubblicare una risorsa, viene contattato l'orchestratore per ottenere i server attualmente online. Essendo la rete dinamica, è possibile che i server hanno cambiato le risorse e il client abbia ancora una visione deprecata della rete. Ottenuti i server, il peer contatta tutti i server che gestiscono la chiave per pubblicarla.

- Download di una risorsa: Nel caso di download di una risorsa, come nel caso della pubblicazione, il peer contatta l'orchestratore per ottenere tutti i server con le informazioni aggiornate. Successivamente, vengono tontattati tutti i server che gestiscono la chiave della risorsa. Nel caso nessun server avesse la risorsa, allora essa non è presente nella rete. Altrimenti, appena si trova un server che la gestisce, il server invia la lista degli indirizzi dei peer da contattare che pubblicano la risorsa. Il peer riceve la lista di tutti i peer e ne randomizza l'ordine, per far si che non venga contattato sempre il 1° peer che ha pubblicato la risorsa.
  È stata adottato anche una politica che lo user, dopo aver completato il download della risorsa, diventa anche un peer che la pubblica. Infatti, per aumentare la soppravvivenza delle risorse sulla rete, appena viene scaricata la risorsa, viene anche inviato un comando di pubblicazione di essa sulla rete

- Rimozione di una risorsa: Quando il peer decide di rimuovere una risorsa da lui pubblicata, il peer contatta l'orchestratore per ottenere tutti i server della rete con le risorse aggiornate. A questo punto, a tutti i server che gestiscono la chiave della riorsa viene inviato un messaggio di rimozione. Il server poi proseguirà nel rimuovere il peer dalla lista dei peer che condividono la risorsa, eliminandola dalle risorse disponibili se nessun altro peer la pubblica.

Nel caso il peer contattasse il server, ma questo non gestisse l'indice della risorsa (che è differente da non essere presente!), il peer ripeterebbe l'intero processo descritto sopra, fino ad un massimo di 20 volte. Se per tutte le volte non riuscisse a contattare il server corretto, allora l'operazione viene considerata fallita.

5 Test

Durante lo sviluppo sono stati eseguiti 2 tipi di test:

- Automatici: i test automatici sono stati utilizzati principalmente in tutte le fasi dello sviluppo, per verificare il continuo e corretto funzionamento della rete durante lo svilupo e attività di refactoring del codice. Questi test coprivano tutte le principali attività della rete;
- Manuali: i test manuali sono stati utilizzati per verificare le situazioni più particolari e complesse, di difficile emulazione con i test automatici (es. verifica di corretta eliminazione del file dalla rete se questo viene cancellato mentre è condiviso).

Tramite i test automatici sono state testate tutte le funzionalità attive del client, la genstione delle risorse del server e dell'orchestratore. Nello specifico, per il client sono stati verificati tutti quei metodi che vengono utilizzati per la comunicazione con il server e l'orchestratore, quali pubblicazione di una risorsa, l'ottenimento di essa e la sua rimozione.

Per quanto riguarda il server, sono state testate tutte le funzionalità che rigardano la gestione delle risorse che sono state pubblicate. Queste funzionalità vanno dalla ricezione di una richiesta di aggiunta di una risorsa alla rete, alla richiesta di rimuovere/ottenere la risorsa. Sono state testate le funzionalità di ottenimento degli indici da parte dell'orchestratore e la gestione di risorse che non appartengono più al server (ricondivisione/rifiuto della risorsa).

Per quanto riguarda i test dell'orchestratore, è stato verificata la corretta generazione degli indici da inviare ai server e il corretto tracciamento delle risorse dei server.

I test manuali sono stati utilizzati per verificare i casi più particolari e di difficile realizzazione mediante unit tests. In particolare, sono stati eseguiti test sulla cancellazione di risorse pubblicate o avvio in contemporanea di più server/peer per vedere come la rete rispondeva).

Alla fine, i test hanno verificato il corretto comportamento dei programmi, evidenziandone alcune criticitià che poi sono state risolte con successo (quali connessione in simultanea di più server o cambiamento multiplo delle risorse in contemporanea).

6 Utilizzo

Per eseguire la rete è necessario prima di tutto avviare un orchestratore. L'orchestratore già dall'inizio si metterà in ascolto sulla porta 10000 in attesa di connessioni. Questo è il primo step che deve essere sempre svolto, altrimenti all'avvio di server e/o peer questi si chiuderanno istantaneamente. Per l'avvio dell'orchestratore è possibile trovare un bash startOrchestrator.bat che viene usato per avviare l'orchestratore tramite gradle.

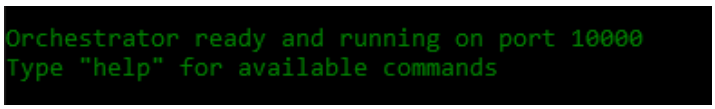

Figure 8: Messaggio di corretto avvio dell'orchestratore

A questo punto è possibile avviare sia peer che server, tramite i bash startPeer.bat e startServer.bat (questi script avviano solamente 1 peer/server per ogni esecuzione). Anche questi script avviano tramite gradle i rispettivi programmi. In entrambi i casi la porta è assegnata randomicamente.
È possibile anche avviare i programmi manualmente, posizionandosi nella rispettiva directory e usando il comando
> gradle run -Pport=[port]
Si noti che il comando non può essere usato con l'orchestratore, che può essere avviato solo tramite il comando
> gradle run
in quando l'orchestratore è preimpostato ad utilizzare la porta 10000.
Sono presenti anche altri bash script per avviare la rete:

- upServers.bat: Viene utilizzato per attivare un determinato numero di server.
- onlyServers.bat: Viene attivata una rete con un orchestratore e un numero di server da inserire.
- upPeers.bat: Viene utilizzato per attivare un determinato numero di peer.

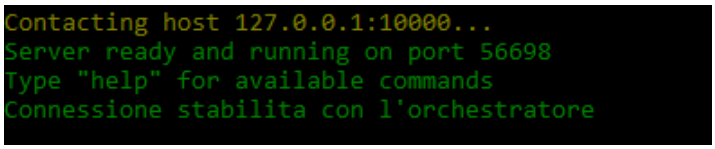

Figure 9: Messaggio di corretto avvio del server

7 Conclusioni

Il progetto è stato portato a termine rispettando i punti che sono stati predisposti nella proposta del progetto. Sono stati lasciati fuori dal progetto i punti che riguardavano la possibilità di utilizzare orchestratori multipli, per evitare la one-point failure (in quanto, se l'orchestratore si disconnette per un possibile malfunzionamento, la rete si spegne).
L'architettura finale che è stata sviluppata si basa sul principio di facilitare la scalabilità della rete. Infatti, grazie alle decisioni prese in fase di progettazione, si è in grado di aggiungere e rimuovere server in modo estremamente dinamico, portando la rete a supportare un elevato numero di server e peer che si connettono/disconnettono dalla rete.

7.1 Progetti futuri

Nel programma si possono determinare diversi punti che possono essere migliorati:

- Nella rete deve essere risolto il problema della one-point failure. Questo problema determina l'effettiva efficacia dell'intera rete. La soluzione principale è quella di aggiungere un algoritmo di election che, nel caso l'orchestratore risulti offline, elegga un server trasformandolo da server ad orchestratore.

- Si può implementare un sistema di cifratura degli ip all'interno dei server, in modo da non poter leggere quali sono i peer che condividono una risorsa. In questo caso, è possibile applicare un algoritmo di cifratura che utilizza una chiave generata random (salvata nel server) per mascherare gli IP. In questo modo non sono visibili e, nel momento di inviare gli ip ad altri componenti, semplicemente si decifra l'ip.
  Questo metodo fa si che server diversi hanno chiavi diverse e risulta impossibile conoscere risorse e ip di chi pubblica.

- Al posto di gestire ogni connessione che si riceve (questo è valido per tutte le entità della rete) creando eseguendo un executor, è possibile cambiare la struttura delle chiamate interne per gestire il tutto mediante l'utilizzo di una bag of work, evitando la necessità di creare n connessioni per n nuovi utenti.

7.2 Cosa ho imparato

Questo progetto mi ha permesso di capire l'importanza dei sistemi distribuiti e di avere una visione dei meccanismi che governano la comunicazione. Inoltre durante lo sviluppo del progetto è stato interessante anche progettare dei meccanismi di sincronizzazione e scambio di messaggi per mantenere la rete aggiornata.
Oltre a concetti lato programmazione/progettazione, il progetto mi ha permesso di approfondire l'utilizzo di tool quali gradle, Junit, git e bash scripting.