Prezentarea proiectului Part2Part

Aliciuc Alexandru-Gabriel, 2B2

January 9, 2023

1 Introducere

Peer-to-peer (P2P), în traducere liberă "de la egal la egal", este o arhitectură de rețea pentru aplicațiile distribuite care împarte sarcinile la mai mulți parteneri. Aceasta este utilizată în principal pentru partajarea resurselor între noduri din rețea (peers), fiecare putând juca atât rol de client cât și de server pentru celelalte noduri. Proiectul *part2part* impune realizarea unei aplicații de acest tip, utilizată pentru partajare de fișiere, putând fi rulat pe mașini diferite.

Am ales acest proiect din dorința de a experimenta arhitecturi noi diferite de clasicul model client/server și deoarece conceptul de P2P este foarte bogat, cu diverse implementări utilizate în practică.

Totuși, nu ne vom detașa complet de acest model deoarece rețeaua implementată este de tip centralizat, având un server central ce facilitează găsirea unui peer ce conține un fișier căutat de alt peer, ulterior transferul de fișiere fiind realizat independent între peers, fără intervenția unui server.

2 Tehnologii utilizate

Pentru transferul de fișiere între peers este folosit la nivelul transport protocolul de comunicare TCP datorită siguranței oferite de acesta, cu toate că este mai lent decât UDP. Este foarte important ca fișierele transferate să nu fie incomplete sau corupte, astfel că, folosind TCP, garantăm că datele vor ajunge la destinație și vor fi primite în ordinea corectă. Similar, pentru comunicarea dintre un peer și serverul central vom folosi același protocol pentru a realiza conexiunea dintre acestea și a garanta siguranța datelor trimise (request-uri de căutare, download, publicare, etc. + răspunsul de la server).

Pentru ca serverul să mențină evidența fișierelor partajate de către utilizatorii rețelei și pentru a face direcționarea către nodurile corecte ce conțin un fișier căutat, acesta este nevoit să stocheze diverse date într-un mod relațional (nume de fișiere, adrese ip + port, numele utilizatorilor, caracteristici ale fișierelor). Pentru a face acest lucru utilizăm o bază de date de tip SQL, având drept SGBD MySQL. Am ales MySQL deoarece aduce viteză și performanță sporită, scalabilitate pentru baze de date mari și suport pentru acces concurent, având de asemenea un API ușor de utilizat pentru limbajul C.

3 Arhitectura aplicației

După cum a fost menționat, aplicația face parte dintr-o rețea P2P de tip centralizat, în care un server principal preia cererile unui peer și se ocupă de găsirea altui peer ce îl poate ajuta. Atât serverul central cât și peer-ii pot prelua cereri în mod concurent.

Un peer este astfel împărțit în două componente:

- Componenta "client" - un thread ce realizează interacțiunea dintre utilizatorul uman și aplicația, prin intermediul căreia introduce diverse comenzi
- Componenta "server" - o serie de thread-uri prin care peer-ul servește cererile "colegilor" de rețea pentru transferul de fișiere

Când utilizatorul introduce o comandă în aplicație, spre exemplu o cerere de desărcare a unui fișier, aceasta este procesată inițial după care este trimisă la serverul central care interoghează baza de date, returnând către peer datele necesare conectării (adresa IP + port) la "colegul" ce are fișierul dorit.

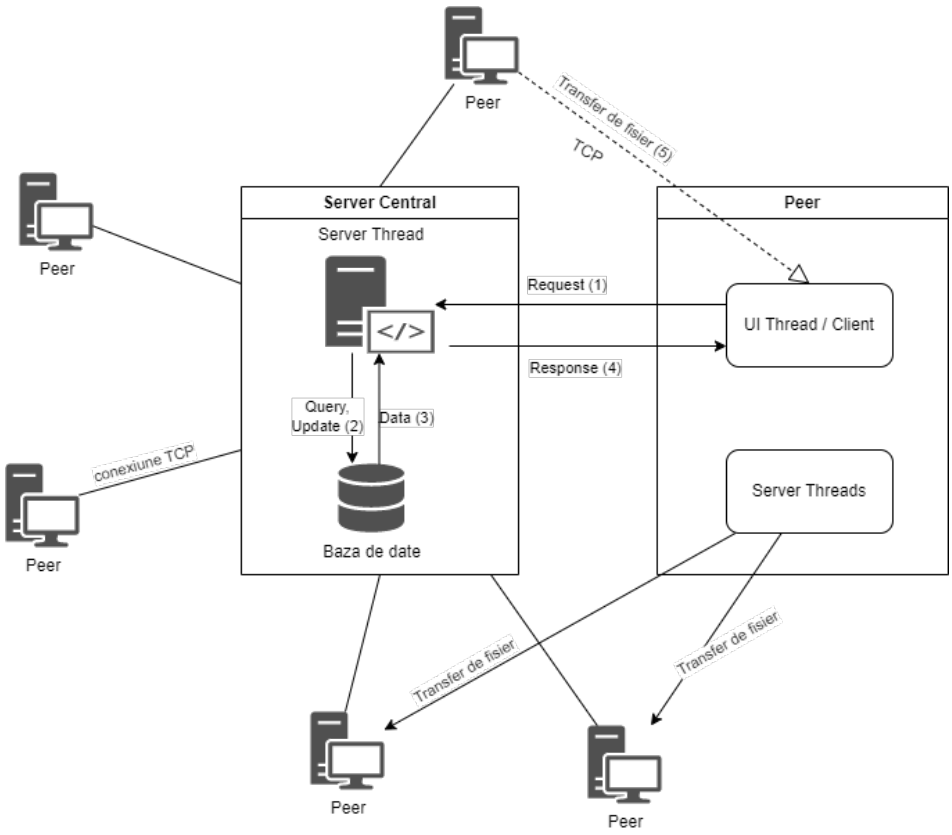

Figure 1: Arhitectura rețelei

Ulterior se inițiază conexiunea de tip TCP după care se realizează transferul fișierului (vezi Figura 1). Funcționalitatea este similară și pentru celelalte tipuri de comenzi, detaliate în secțiunea următoare.

Acest tip de arhitectură vine cu diverse avantaje și dezavantaje, remarcându-se în primul rând prin eficiență și viteză deoarece legătura între nodurile din rețea se face rapid prin intermediului serverului central (fără a fi nevoie sa facem flooding rețelei ca în cazul arhitecturii pur decentralizate), dar și prin faptul că eliminarea unor noduri nu afectează foarte mult rețeaua, deci vom fi mai puțin predispuși la erori de acest tip. Pe de altă parte, serverul principal reprezintă un "single point of failure" de care depinde toată rețeaua, iar această opțiune nu este cea mai scalabilă pentru un număr extrem de mare de utilizatori, administrarea serverului fiind costisitoare.

4 Implementare

Programele (peer + server central) sunt implementate în limbajul C/C++, comunicarea dintre acestea fiind realizată prin socket-uri. Atât programul peer cât și cel server suportă acces concurent prin multithreading (pentru fiecare conectare diferită se creează un thread ce gestionează comunicarea ulterioară), utilizând primitivele din standardul POSIX (pthread.h). Inițial utilizatorul ce folosește aplicația peer trebuie să introducă un username (pentru vizualizare mai ușoară) și un port valabil (pentru a primi cereri de transfer de la alți utilizatori), după care se creează thread-urile necesare celor două componente ale aplicației.

Protocolul pentru interacțiunea utilizatorului cu aplicația este următorul:

1. ***search*** *nume_fisier -type:<text,video,audio,game,software,other> -maxsize:<real_value (MB)> -minsize:<real_value (MB)>*

Peer-ul trimite o cerere de căutare a unui fișier către server, după care primește un răspuns cu

toate intrările găsite alături de atributele corespunzătoare (tipul fișierului, dimensiune, hash-ul acestuia, numele user-ului deținător), fiecare pe o linie.

În loc de *nume_fisier* poate fi specificat caracterul '*', afișând toate intrările ce respectă argumentele date ca input.

Atributele *type*, *maxsize* și *minsize* sunt opționale, semnificația acestora fiind de la sine înțeleasă. Tipul fișierului este stabilit opțional de către peer-ul care încarcă fișierul prin comanda **publish**.

Atributele *maxsize* și *minsize* primesc o valoare de tip real (tipurile float/double în C), specificând dimensiunea în MB.

2. ***download*** *nume_fisier -type:<text,video,audio,game,software,other> -maxsize:<real_value (MB)> -minsize:<real_value (MB)> -hash:<MD5 hash>*

   Peer-ul trimite o cerere de căutare a unui fișier către server similar comenzii search, dar de această dată serverul trimite drept răspuns adresa IP + port-ul unui alt peer ce conține fișierul căutat (sau un "Peer not found" dacă nu este găsit).

   Atributele au aceeași semnificație ca la comanda search, la care se adaugă unul nou: hash-ul fișierului, de tip MD5. Acesta este calculat automat în programul peer în momentul publicării unui fișier prin comanda **publish**, ajutând la identificarea fișierelor cu un conținut identic. Într-un exemplu de utilizare, un utilizator caută un fișier prin comanda **search**, după care alege specific fișierul pe care dorește să-l descarce (din cele vizualizate) specificând hash-ul acestuia. Dacă există mai multe fișiere ce respectă condițiile din această comandă, atunci se alege pentru download primul găsit.

   În continuare se stabilește conexiunea între peers pentru a iniția transferul.

3. ***publish*** *file_path -type:<text,video,audio,game,software,other>*

   Adaugă un fișier în lista celor partajabile, după care trimite numele și atributele noului fișier serverului care actualizeaza baza de date. Serverul va trimite după un răspuns prin care confirmă daca operația a fost realizată cu succes. Din acest moment fișierul poate fi accesat și descărcat de alți utilizatori ai rețelei.

4. ***unpublish*** *file_path*

   Inversul comenzii publish, scoate fișiere din lista celor ce pot fi partajate.

5. ***disconnect***

   Deconectarea de la rețea. Serverul șterge aparițiile ce corespund acestui peer din baza de date, iar programul este închis alături de thread-ul ce menținea conexiunea în server.

6. ***downlocation*** *directory*

   Singura comandă care se execută local, doar la nivelul programului peer. Setează directorul în care fișierele sunt downloadate.

Toate aceste comenzi vor fi parsate din programul peer pentru a li se valida corectitudinea (clasa $argsParser$ + functia $validateCommand$), după care vor fi procesate pentru a putea fi trimise către server. Fiecarei comenzi îi corespunde un cod unic de tip short int, care va fi trimis către server (*enum cmdType*). Apoi, în funcție de comandă, informatiile fisierului si eventualele argumente vor fi condensate într-o structură ce va fi trimisă la server pentru a putea fi ulterior procesata si a se trimite un raspuns la request (structurile $argsInfo$, $publishedFile$ si $peerInfo$).

Pentru transferul de fișiere, peer-ul client trimite hash-ul fișierului dorit (obtinut prin interogarea serverului central) către peer-ul server, precedat de lungimea acestuia. După ce acesta confirmă existența fișierului, trimite continuu "bucăți" de o anumită dimensiune (să o notăm cu MAX_SIZE)

din fișierul dorit până la finalizarea transferului (ultima bucată are o lungime mai mică decât MAX_SIZE). Se apelează la această metodă deoarece dimensiunea unui fișier poate fi foarte mare, fiind imposibil în acest caz să adăugam într-un buffer din memorie întreg conținutul fișierului pentru a putea face transferul prin socket-uri.

Exemplu de cod C ce realizează acest lucru:

```
void sendFile(int socket, const std::string& path)
{
    int file_fd;
    char buff[MAX_SIZE];
    int chunkSize = MAX_SIZE;

    CHECK_RET(file_fd = open(path.c_str(), O_RDONLY), "Can't open for file transfer");

    while (chunkSize == MAX_SIZE)
    {
        chunkSize = read(file_fd, buff, MAX_SIZE);
        write(socket, buff, chunkSize);
    }

    close(file_fd);
}
```

Cazuri de eroare mai speciale apar atunci când:

- Un peer se deconectează de la rețea fără a anunța acest lucru prin comanda **disconnect** (de ex. o pană de curent) - în acest caz baza de date conține în continuare date despre fișierele acestui peer, deși el nu se mai află în rețea. Trebuie implementat astfel un mecanism care verifică periodic starea conexiunii unui peer la server pentru a putea corecta înregistrările din baza de date. Pentru a face acest lucru, ne folosim de faptul că un thread din serverul central așteaptă în apelul blocant al funcției read primirea unei comenzi de la peer-ul conectat, astfel că se poate identifica deconectarea forțată a peer-ului dacă apar erori de citire.
- Un peer se deconectează de la rețea în timp ce transferă un fișier - peer-ul client notifică serverul central despre deconectarea "colegului" de transfer din rețea și reface conexiunea cu alt peer

5 Concluzii

Orice arhitectură de tip P2P vine cu diverse avantaje și dezavantaje, dar și cu noi provocări și dificultăți față de modelul client/server, alegerea unei arhitecturi potrivite pentru implementare fiind dependentă de scopurile aplicației. În acest caz am ales implementarea unei rețele P2P cu server central, la care se pot adăuga însă îmbunătățiri precum evitarea unui "single point of failure" prin adăugarea unui cluster de servere, transferul simultan de bucăți de fișiere de la mai multe noduri sau adăugarea unor mecanisme de caching pentru deconectarea și reconectarea constantă la rețea a unui peer.

References

[1] *Course&Laboratory - Computer Networks 2021-2022*. URL: https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php.

[2] *Flowchart Maker & Online Diagram Software*. URL: https://app.diagrams.net/.

[3] *Peer-to-peer*. ro. Page Version ID: 15187310. Oct. 2022. URL: https://ro.wikipedia.org/w/index.php?title=Peer-to-peer&oldid=15187310.

[4] *Roger Clarke's 'Overview of P2P'*. URL: http://www.rogerclarke.com/EC/P2POview.html.