Part2Part – Aplicație CLI pentru Peer2Peer File Sharing

Antohi Robert
FII,UAIC

Abstract. Tema 2 Antohi Robert. Descrierea implementării proiectului Part2Part.

Keywords: TCP, P2P, File Sharing, Client-Server, SQLite3, Cache, LRU

1 Introduction

Part2Part este o aplicație de File Sharing peer2peer, formată dintr-un server și mai mulți clienți. Fiecare client poate face share la unul sau mai multe fișiere sau căuta un fișier, să vadă dacă este disponibil pe rețea. Serverul reține fișierele partajate, împreună cu cine partajează respectivul fișier, într-o bază de date sqlite3. Fișierele nu trec prin server, ci sunt transmise direct de la client la client.

2 Detalii despre tehnologiile și conceptele folosite

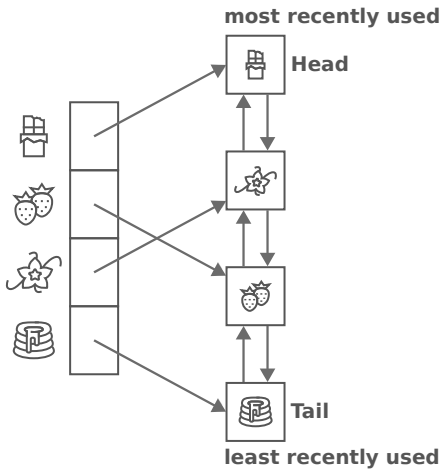
A fost ales protocolul TCP/IP pentru transferul de fișiere(de la client la client), întrucât octeții dintr-un fișier trebuie primiți în ordinea în care au fost trimiși, iar latency-ul nu este de importanță critică (cum e cazul la aplicațiile de videoconferință/VoIP). Deși reconstrucția octeților primiți se poate face și în cazul folosirii UDP, ar adăuga un strat de complexitate în plus și nu ar aduce beneficii.

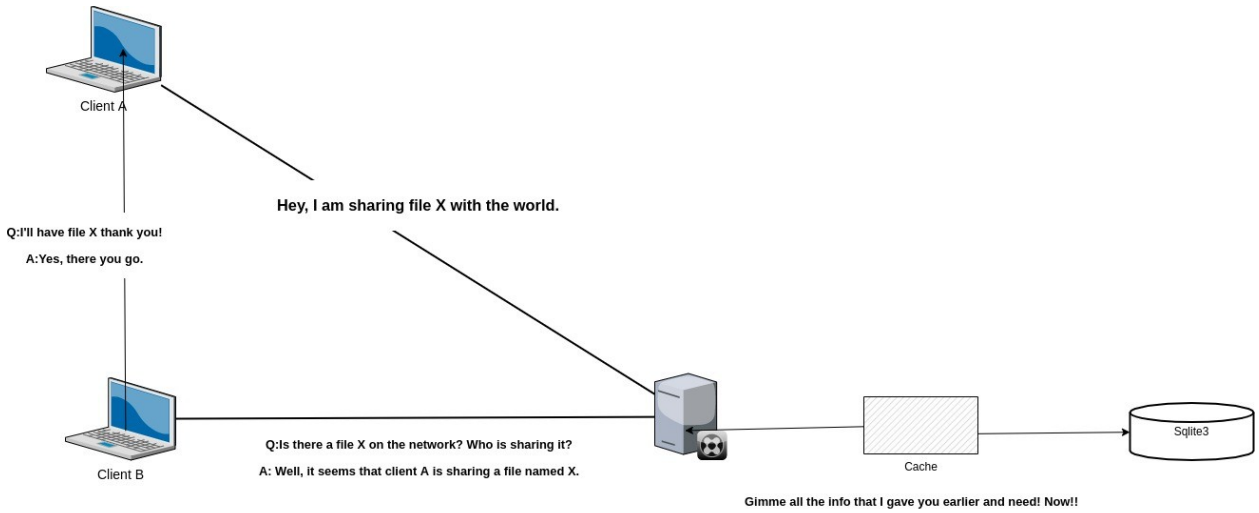
Deși în prototip s-a folosit stocarea tuturor informațiilor de către server in-memory, numărul fișierelor partajate poate ajunge destul de mare, astfel încât stocarea in-memory să nu fie satisfăcătoare. Astfel, am apelat la o bază de date sqlite3, care este perfectă pentru prototipuri(mai avansate!) ale proiectelor, și asigură o tranziție ușoară către SGBD avansate. Conexiunea la această bază de date se va face folosind o librărie a C++(numită, convenient, sqlite3).

Totuși, pentru că a verifica datele in-memory durează mult mai puțin decât a face query pentru ele în baza de date, s-a implementat un in-memory cache, cu politica LRU (Least Recently Used), folosind structurile de date std::list(doubly linked list) și std::unordered_map(hash map). Astfel, obținem o complexitate O(1) atât pe partea de update, cât și pe partea de query. Singura inconvenientă este complexitatea spațiu O(n), care poate fi însă ajustată prin limitarea numărului maxim de elemente din cache.

Pentru a trimite efectiv un fișier, un client se va transforma el însuși în server. Aplicația, este, astfel, parțial descentralizată, serverul fiind necesar numai pentru a afla de la ce client trebuie luat fișierul.

3 Arhitectura aplicației





4 Detalii de implementare

Librăria sqlite3 se include ca header “sqlite3.h”, și a fost link-uită la compilare(fiind descărcată de pe internet).
Se va deschide conexiunea către baza de date, folosind funcția sqlite3_open.

```
#include <sqlite3.h>
```

```
sqlite3 *db;
int res = sqlite3_open("database.db", &db);
```

Pentru conexiunea TCP/IP (socket, listen, etc) am folosit codul cu care am lucrat la laborator.
Pentru a asigura caracterul concurent al serverului am folosit primitiva fork().

Structurile de date folosite pentru Cache tip LRU.

```
unordered_map<string, pair<string,int>> m;
list<tuple<string,string,int>> l;
```

Fișierele vor fi citite și trimise folosind un buffer, întrucât e imposibil să trimitem(și/sau citim) fișierele mari dintr-o dată. **code shamelessly taken from stackoverflow(see ref. 4)**

```
std::ofstream outfile(collector, std::ofstream::binary);
char buffer[MAX];
int bytesread; //how much data was read? recv will return -1 on error

while (size)
{ //collect bytes
    bytesread = recv(sock_CONNECTION, buffer, MAX > size ? size : MAX, 0);
    // MAX>size?size:MAX is like a compact if-else: if (MAX>size){size}else{MAX}
    if (bytesread < 0)
    {
        //handle error.
    }
    outfile.write(buffer, bytesread);
    size -= bytesread; //decrease
    std::cout << "Transferring.." << std::endl;
}
```

Clientul poate trimite următoarele comenzi:
“start share port” – pornește share-ul de fișiere, transformându-se in clientminiserver (similar cu server).
“share file.txt” – partajează fișierul file.txt
“stop share” – oprește funcția de share
“search file.txt” – caută dacă există în rețea un fișier cu numele file.txt. Dacă da, va primi adresa ip și portul la care este accesibil.
“get file.txt ip port” – încearcă să descarce file.txt de la adresa ip menționată, de la portul port
Pentru a asigura caracterul concurent al serverului am folosit primitiva fork(). Și clientul va servi ceilalți clienți tot cu ajutorul primitivei fork().

5 Concluzii

Arhitectura implementată asigură că odată ce clienții știu de unde să primească fișierele, nu mai au nevoie de server. Serverul funcționează, deci, doar ca un lookup table. Evident, pentru ca clienții să se poată conecta inițial, trebuie conexiunea la server. Poate fi transformată în aplicație de tip torrent, unde detaliile de share să fie făcute public pe mai multe website-uri, pentru a avea cu adevărat un caracter decentralizat.

Fork() nu este cel mai eficient mod posibil, întrucât se așteaptă query-urile de la BD- proces blocant. Se poate proceda ca în Lab 9, cu multiplexare I/O.

Nu în ultimul rând, putem folosi această oportunitate pentru a ne gândi la decentralizarea serverelor/edge computing/cloud. Uber a transformat fiecare om/mașină în potențial șofer/taxi, AirBnB a transformat fiecare apartament în potențial hotel. Ne-am putea gândi la fiecare calculator/device(eventual și smartphone) ca potențial server și la fiecare proprietar de device ca potențial cloud provider– majoritatea calculatoarelor stau închise măcar 50% din timp, și au o rată de folosire a puterii de calcul<10% în medie - > 95% untapped potential, which is huge.

References

1. https://www.interviewcake.com/concept/java/lru-cache
2. https://www.gigaspaces.com/blog/in-memory-cache/
3. https://profs.info.uaic.ro/~computernetworks/index.php
4. https://stackoverflow.com/questions/30877182/sending-files-in-socket-programming-tcp
5. https://videlais.com/2018/12/12/c-with-sqlite3-part-1-downloading-and-compiling
6. http://www.rogerclarke.com/EC/P2POview.html