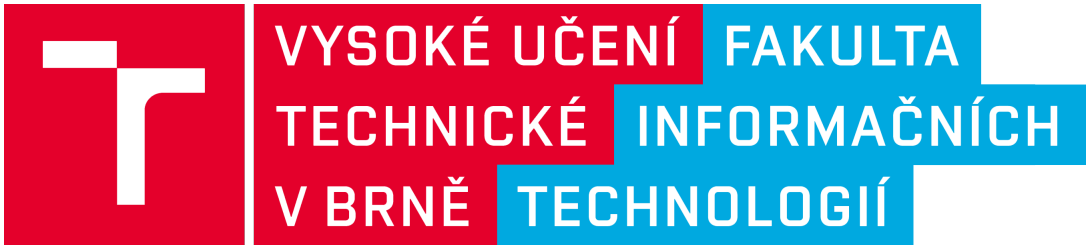
Vysoké učení technické v Brně
Fakulta informačních technologií
Ústav informačních systémů



PDS - Přenos dat, počítačové sítě a protokoly

Hybridní chatovací P2P síť
2018/2019

Maroš Vasilišin (xvasil02)

27.04.2019

Obsah

1 Úvod

V tejto dokumentácii sa nachádza popis riešenia projektu do predmetu PDS v akademickom roku 2018/2019 na FIT VUT. Projekt je zameraný na vytvorenie hybridnej chatovacej peer-to-peer siete a otestovanie kompatibility s niekoľkými spolužiakmi.

V druhej kapitole sa nachádza krátky úvod do peer-to-peer problematiky a sú v nej popísané rozdiely oproti iným architektúram. V tretej kapitole je popis implementácie a sú v nej bližšie popísané zaujímavejšie časti aplikácie. Vo štvrtej kapitole je popísaný priebeh testovania aplikácie.

2 Sieťové architektúry

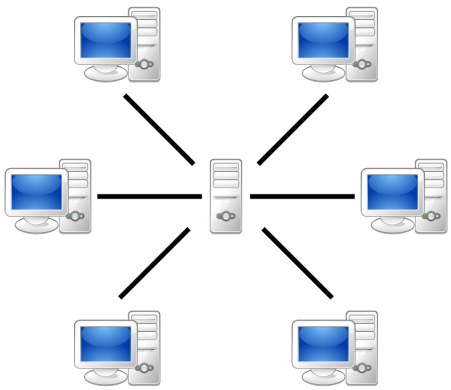
Táto kapitola popisuje jednotlivé sieťové architektúry, ukazuje ich výhody a nevýhody a popisuje rozdiely medzi nimi.

2.1 Klient-server

Klient server architektúra [1] je založená na princípe, kde sa uzly(počítače) v sieti rozdeľujú na dve kategórie. Uzly nazývame serverami, ak poskytujú svoje služby. Ak uzly služby využívajú, nazývame ich klientami. Príkladom služby môže byť www, email a podobne.

Existuje viacero typov serverov, podľa služby, ktorú poskytujú. Poznáme databázové, webové či iné servery. Niektoré typy serverov poskytujú viacero služieb súčasne. Na jednom počítači môže byť súčasne spustených viacero serverov.

Komunikácia funguje na princípe dotaz-odpoveď. Klienti si vyžiadajú nejaké dáta, prípadne vykonanie operácie na serveri, a server na ten dotaz odpovedá. Aby si uzly v sieti rozumeli, musia používať rovnaký protokol.



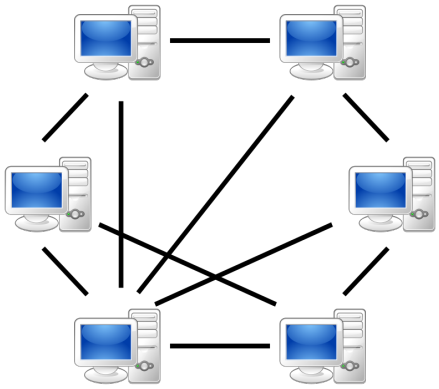
Obr. 1: Klient-server architektúra

2.2 Peer-to-peer

Peer-to-peer architektúra [2], nazývaná aj P2P, je taká architektúra, kde sú všetky zariadenia na rovnakej úrovni. Dokážu medzi sebou priamo komunikovať, bez použitia centrálneho administratívneho systému. V čistých P2P sieťach si uzly, nazývane peeri, medzi sebou zdieľajú zdroje, napríklad výpočtový výkon, databáze a podobne. Samotnú správu zdrojov si riadia peeri sami. Peeri sú schopní zdroje poskytovať a aj využívať. Peeri teda plnia funkciu klienta aj serveru súčasne.

Existujú dva princípy ako vytvoriť P2P sieť. Prvý je neštrukturovaný princíp, kde sa spojenia medzi peermi tvoria relatívne náhodne. Tento princíp sa využíva napríklad v aplikáciách ako Gnutella alebo Kazaa. Výhodou tohto princípu ke rýchlosť vytvárania a odoberani uzlov v sieti. Nevýhodou je samotné využívanie siete na komunikáciu. Neštrukturalizovanosť spôsobuje, že uzol musí posielať požiadavku na zdroje na všetkých peerov, kým nenájde požadované dáta. Tento typ siete sa vyznačuje vysokým využitím procesorov a veľkým množstvom správ.

Druhý princíp je štrukturovaný, kde sa peeri spájajú do určitej hierarchie. Výhodou je efektívnosť pri vyhľadávaní zdrojov. Najčastejšie sa implementuje pomocou distribuovaných hashovacích tabuliek, do ktorých sa zapisuje, ktorý zdroj je uložený na ktorom uzle v sieti. Peeri si udržiavajú informácie o svojich susedoch v sieti. Príkladom aplikácií, ktoré tento princíp využívajú sú Tixati, Kad, Storm botnet a iné.



Obr. 2: Peer-to-peer architektúra

2.3 Hybridné modely

Existujú aj takzvané hybridné modely, ktoré kombinujú P2P a klient-server architektúry. Najčastejšie sú modelované tak, že hlavný server sa využíva na to, aby sa peeri mohli medzi sebou efektívne vyhľadávať. Vo zvyšku siete je potom zachovaná P2P architektúra. Tento model využívala v začiatkoch firma Spotify, dnes je to už inak [3]. Hybridné modely sú považované za najefektívnejšie zo všetkých spomenutých.

3 Popis implementácie

V tejto kapitole bude popísaná hlavná funkcionalita aplikácie. Podrobnejšie budú rozobrané zaujímavejšie časti implementácie. Pre implementáciu bol zvolený jazyk Python, pre jeho flexibilnosť a rýchlu učiacu krivku.

3.1 Časti aplikácie

Aplikácia sa skladá z troch častí: uzlov, peerov a rpc.

Uzly aj peeri fungujú ako falošní démoni. Pri spustení sa inicializujú s parametrami danými ako argumenty spustenia skriptu. Po spustení čakajú na príkazy od rpc. Každý uzol aj peer sa inicializuje s unikátnym identifikátorom. Zadávanie príkazov pre všetky prvky v sieti (uzly aj peeri), je založení na tom, že každý prvok si pri vytvorení vytvorí dočasný súbor, ktorý je pomenovaný identifikátorom typu prvku a identifikátorom daného prvku samotného. Do tohto súboru je možné zadávať príkazy v presne špecifikovanom formáte, ktorému prvky rozumejú. Formát je vlastne zjednodušeným formátom, ktorým sa zadávajú rpc príkazy, kde sú odstránené názvy argumentov. Tieto dočasné súbory sa pri ukončení peera alebo uzlu automaticky odstránia.

Zadávanie príkazov je možné aj automatizovane, pomocou rpc skriptu. Podľa parametrov predaných skriptu je možné poslať rpc správu ľubovoľnému lokálne dostupnému uzlu či peeru. Cieľ príkazu sa musí nachádzať v rovnakej zložke ako rpc skript. Po poslaní príkazu práca rpc skriptu končí, všetky ďalšie výpisy sa dejú v rámci uzlov a peerov.

3.2 Dostupné funkcie

Uzly v sieti majú dve hlavné funkcie. Udržiavajú si databázu aktuálne pripojených peerov, a udržiavajú informácie o susedných uzloch. Uzly sa vedia pripájať a odpájať k ďalším uzlom v sieti, a týmto spôsobom šíriť informácie o svojich pripojených peeroch. Uzly si v pravidelných intervaloch posielajú správy, v ktorých je aktuálny stav ich databázy. Pre posielanie správ v pravidelných intervaloch bola použitá knižnica *threading*, ktorá pomáha pri vykonávaní opakovanej akcie na pozadí. Je použitá pri posielaní správ medzi uzlami, ako aj medzi uzlom a peermi. Jedno vlákno v uzli je vyčlenené na pravidelné čítanie z vytvoreného dočasného súboru, aby uzol mohol vykonávať požadované príkazy.

Uzly súčasne musia načúvať na správy od peerov a iných uzlov. Na túto funkcionalitu je použitý nekonečný while cyklus. Pri spracovaní správy je buď okamžite spracovaná a výsledok vypísaný, alebo ak by spracovanie trvalo dlhšie, je pre spracovanie založené nové vlákno a hlavný program nie je narušený.

Pre kódovanie správ bola použitá Open Source knižnica bencoder [4]. Knižnica podporuje správy v ASCII kódovaní. Takto zakódované správy sa posielajú medzi všetkými prvkami v sieti. Knižnica využíva štandardný formát Bencode, a je teda zaručená kompatibilita s implementáciami ostatných spolužiakov.

3.3 Zaujímavé časti implementácie

Jednou zo zaujímavých častí bolo zaručenie ukočenia behu peera či uzlu na signál SIGINT. Bola to jedna z viac komplikovaných častí, pretože bolo potrebné zaručiť, že sa všetky vlákna aplikácie ukončia v správnom poradí bez chybových hlásení. Takisto bolo potrebné zaručiť ukončenie okamžite, aby sa nečakalo na ukončenie vlákien. Pri implementácii bolo vyskúšaných viacero princípov, nakoniec sa ako správne ukázalo využitie objektov *Event* z knižnice *threading*.

Ďalšou zaujímavou časťou bolo vymyslieť ako predávať rpc príkazy z rpc skriptu do peerov a uzlov. Zvolené riešenie cez dočasné súbory je popísané v sekcii 3.1.

Z pohľadu testovania aplikácie bola zaujímavá aj požiadavka zadania využiť UDP protokol, ale posielať si vlastné ACK správy na potvrdenie doručenia. Bolo potrebné testovať prípady ak sa ACK správy poslali a doručili, ako aj prípady kedy nedorazili. Pri testovaní aplikácie tým pádom vzniklo veľké množstvo rôznych prípadov a podarilo sa tým vyriešiť veľa okrajových prípadov.

4 Testovanie

V tejto kapitole sa bližšie zameriame na postup a výstupy testovania. Rozdelená bude na dve časti, lokálne testovanie počas vývoja a testovanie s implementáciami spolužiakov na servere merlin. V závere kapitoly budú popísané očakávané výstupy jednotlivých implementovaných príkazov.

4.1 Lokálne testovanie

Testovanie prebiehalo vo viacerých fázach počas vývoja aplikácie. Neboli vytvorené žiadne automatizované testy. Počas vývoja, ako boli vyvíjané jednotlivé požadované rpc príkazy sa objavovali nové prípady použitia, ktoré boli pravidelne testované. Aplikácia bola vyvíjaná mimo dostupného virtuálneho stroja, ale v pravidelných intervaloch bola na ňom testovaná. Testovanie prebiehalo v rozsahu siete 1-3 uzly, 1-3 peeri. Výstupom testov bola vždy množina prípadov, pre ktore aplikácia nefunguje, ktoré boli v ďalšej verzii opravené.

Príklad testovacieho scenára:

- spustenie uzlu A
- pripojenie peera X k uzlu A
- pripojenie peera Y k uzlu A
- RPC príkaz peers na peer X
- RPC príkaz message z peera A na uzol B

4.2 Implementácie spolužiakov

Testovanie so spolužiakmi prebiehalo formou nahrania a spustenia uzlov a peerov na serveri merlin. S niekoľkými spolužiakmi (xuhlir16, xurban61) prebiehalo vzájomné testovanie častejšie, hlavne v posledných týždňoch vývoja, za účelom nájdenia chýb. Napríklad vzájomné testovanie ukázalo chybu v kódovaní správy UPDATE, ktorú by sa lokálne nepodarilo nikdy objaviť. Takisto vzájomné testovanie pomohlo pri pochopení niektorých nejasných častí zadania.

S ďalšími spolužiakmi (xkiral01, xmatej51) prebiehalo testovanie nepriamo. Niekoľko ich nahralo svoje riešenia na merlina a zverejnili na fóre predmetu PDS či vo facebookovej skupine ročníka IP adresy a porty ich bežiacich uzlov a peerov. V rámci tohto testovania prebiehalo pripojenie a komunikácia samostatne, od spolužiakov bol len zistený jazyk implementácie.

So spolužiakmi boli testované príkazy getlist, peers, message, reconnect, connect, neighbors a disconnect. Zhrnutie testovania je zhrnuté nižšie v Tabuľke 1.

Testovanie so spolužiakmi sa ukázalo ako potrebné, na nájdenie chýb, ktoré by sa lokálne nepodarilo objaviť. Testovanie bolo úspešné, keďže väčšina príkazov fungovala podľa očakávaní.

Tabuľka 1: Výsledky testovania

login	jazyk	problémy
xuhlir16	python	nájdené chyby v benode, ACK, message, dbrecord, v druhom testovaní opravené
xurban61	python	nefungovali správy ACK, UPDATE, DISCONNECT na jeho strane, v druhom testovaní opravené ACK, UPDATE
xkiral01	python	žiadne
xmatej51	python	žiadne

4.3 Výstupy

V Tabuľke 2 sú popísané očakávané výstupy na vybrané príkazy RPC. Ostatné príkazy majú výstupy na pozadí.

Tabuľka 2: Očakávané výstupy

príkaz	výstup
peers	zobrazenie tabuľky peerov na výstup
message	zobrazenie prijatej správy na cieľovom peerovi
database	zobrazenie tabuľky peerov
neighbors	zobrazenie tabuľky susedných uzlov

5 Záver

V tomto projekte bola úspešne implementovaná hybridná chatovacia p2p sieť v jazyku python. Aplikácia bola implementovaná v plnom rozsahu zadania. Bonusové riešenie nebolo implementované.

Literatúra

[1] Wikipedia contributors, "Client–server model — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Client%E2%80%93server_model&oldid=894200374, 2019. [Online; 27-April-2019].

[2] Wikipedia contributors, "Peer-to-peer — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Peer-to-peer&oldid=893904097, 2019. [Online; 27-April-2019].

[3] "Spotify abandoning p2p in favor of a more traditional dedicated architecture." https://siliconangle.com/2014/04/22/spotify-abandoning-p2p-in-favor-of-a-more-traditional-dedicated-architecture/, Apr 2014. [Online; 27-April-2019].

[4] U. Demir, "bencoder." https://github.com/utdemir/bencoder, 2016. [Online; 27-April-2019].