**Projekat RAFBook - dokumentacija**

Chord predstvalja implementaciju distribuirane hash tabele. Poenta Chord-a je da se podaci cuvju distribuirano, na vise cvorova I da budu uniformno rasporedjeni. Uniformna rasporedjenost se postize funkcijom hesiranja koja hesira kljuceve uniformno. Npr ako imamo sistem od 64 cvora, funkcija hesiranja ce davati priblizno podatke kao sto su (2, 8 , 16 , 25, 36, 48,..64). Takodje jos jedna bitna odlika Chorda jeste sto ima logaritamsko pretrazivanje.

Mutex – za implementaciju Distribuiranog mutexa sam koristio fer algoritam- Suzuki Kasami.

**Dodavanje prijatelja**

Kada se pozove komanda add\_friend, cvor salje FRIEND\_REQUEST message ka cvoru ciji je IP I PORT naveden u komandi. Poruka putuje preko tabele sledbenika do trazenog cvora, gde on dodaje kod sebe u listu prijatelja cvor koji je zahtevao prijateljstvo, nakon cega mu on posalje FRIEND\_REPLY poruku, gde kada nju primi cvor koji je zahtevao prijateljstvo, ubacuje njega u listu prijatelja. Ceo tok od slanja zahteva, do odgovora je pod distribuiranim mutexom.  
Cvor A koji salje zahtev ka cvoru B. uzima lock, I otpusta lock tek kada mu stigne poruka od cvora B. Dodavanje prijatalje u O = log(n);

**Dodavanje fajlova**  
  
Zamislimo da cvor A hoce da upise fajl.Pozivom komadne add\_file “putanja.txt” cvor A uzme lock nad celom operacijom dodavanja fajla. Putanja fajla se hesira, dobija se id cvora koji ce biti zaduzen za njega I taj fajl preko poruke ADD\_FILE putuje do onog cvora koji je zaduzen da ga ubaci. Onda kada stigne do cvora koji je zaduzen da ga ubaci, on ga upisuje kod sebe, nakon cega salje TELL\_ADD\_FILE poruku onom cvoru koji je inicirao dodavanje fajlova. Takodje, prilikom upisivane podataka kod sebe, on salje BACKUP\_MESSAGE ka svom sledbeniku, gde mu prosledju svoje podatke I informacije neophodne za oporavak sistema. Cim cvor inicijator primi poruku TELL\_ADD\_FILE, on otpusta lock. Dodavanje fajlova je u O =log(n)

**Brisanje fajlova**

Zamislimo da cvor A hoce da upise fajl. Pozivom komadne add\_file “putanja.txt” cvor A uzme lock nad celom operacijom brisanja fajla. Putanja fajla se hesira, dobija se id cvora koji ce biti zaduzen za njega I taj fajl putuje preko poruke REMOVE\_FILE do onog cvora koji je zaduzen da ga izbrise. Onda kada stigne do cvora koji je zaduzen da ga ubaci, on ga upisuje kod sebe, nakon cega salje TELL\_REMOVE\_FILE poruku onom cvoru koji je inicirao dodavanje fajlova. Cim cvor inicijator primi poruku TELL\_REMOVE\_FILE, on otpusta lock. Brisanje fajlova je u O = log(n).

**Pregled fajlova**

Komandom view\_file localhost port navodi se zahtev da recimo cvor A hoce da dobije podatke od cvora B. Hesira se port iz komadne I onda preko chord pretrage se traze zahtevani cvor (u tom momentu se uzima lock). Zahtev putuje preko poruke VIEW\_FILE. Kada ta poruka konacno stigne do cvora koji od kog se zahteva pregled fajlova, on potom vraca preko poruke TELL\_VIEW poruke nazad ka cvoru koji je zahtevao poruke. Bitna napomena je da cvor koji je trazio fajlove dobija sve fajlove od cvora od kog je trazio ukoliko mu je prijatelj, ukoliko nije, onda dobija samo njegove public fajlove. Pregled fajlova je u log(n).

**Uredno gasenje cvorova**

U kodu sa vezbi, gasenje cvora je bilo tako da se oni jednostavno ugase, bez bilo kakve odgovornosti. U mom projektu je ubaceno da se oni uredno gase, tj da kada se je jedan cvor ugasi, da se sacuva cela hijerarhija sistema. To radi na sledeci nacin. Cvor koji hoce da se ugasi, uzima lock onda on salje QUIT\_NODE poruku ka svom sledbeniku gde mu prosledjuje svoje podatke kao I informaciju o tome ko mu je prethodnik. Kada njegov sledbenik primi QUIT\_NODE poruku, azurira svog prethodnika, tako mu cvor koji se gasi nije vise prethodnik, nego njegov prethodnik.  
Potom on salje REMOVE\_UPDATE poruku dalje svom sledbeniku I to tako putuje sve do cvora koji je prethodnih cvoru koji hoce da se ugasi. Posto je njemu azurirana tabela sledbenika, iz REMOVE\_UPDATE poruke uzima informaciju od cvora koji je inicirao gasenje I njemu salje poruku TELL\_QUIT\_NODE. Kada cvor koji je inicirao svoje gasenje primi poruku TELL\_QUIT\_NODE, otpusta lock I gasi se.

**Ulazak novog cvora u sistem**

Kada se jedan cvor prikljucuje sistemu, on se prvo obrati Bootstrap serveru, koji mu dodeljuje neki random chorId iz liste aktivnih cvorova. Nakon toga on se javlja tom novu cvoru putem NEW\_NODE poruke I pamti njegov chordId za potrebe unlocka. Kada do tog cvora stigne NEW\_NODE poruka, on uzima lock, I dalje prosledjuje update poruku za potrebu azuriranja tabela sledbenika I vrednosti salje se update poruka. Ta poruka putuje kruzno I kada stigne do cvora koji se ukljucuje, on salje TELL\_WELCOME poruke cvoru koji je uzeo lock I koji je zapoceo dodavanje novog cvora. Nakon sto on primi TELL\_WELCOME poruku, onda otpusta lock.

**BuddySistem za otkaze I Backup**

Kako bih ovo najlakse objasnio, navescu to kroz jedan primer kako to sve izgleda.  
Recimo da imamo cvorove A, B, I C. A je sledbenik od B, a C je prethodnik od B.  
Cvor B proverava da li je C ziv na sledeci nacin. Na svakih 4 sekunde, cvor B cvoru C salje poruku PING, a ovaj mu odgovara porukom PONG. Ukolko za tih 4 sekunde(slaba granica otkaza) ne dobije PONG poruku, onda on njega oznacava kao sumnjivog, I salje poruku BUDDY svom sledbeniku, cvoru A, gde mu prosledjuje informaciju o tome ko treba da se proveri. Nakon sto primi Buddy poruku, on salje poruku IS\_ALIVE ka cvoru C I ceka 10 sekundi(jaka granica otkaza). Ukoliko se ne desi da primi poruku ALIVE od cvora C, cvor A salje poruku CONFIRMATION cvoru B da je cvor C mrtav. Posto je B sledbenik cvora C, on kod sebe vec ima sve backupovane podatke cvora C. Nakon prijema poruke CONFIRMATION, zapocinje proces restruktuiranja koji je maltene slican kao kod gasenja cvora, sa jednim izuzetkom, da je ovog puta inicijator restruktuiranja sledbenik od cvora koji se ugasio, za razliku od procesa urednog gasenja cvora. Kada se zavrti ceo krug, onda cvor B koji je inicijator pusti lock I sistem nastavlja da radi neometano bez cvora C.