Dokumentacija projekta: Load Balancer

Uvod

Projekat predstavlja simulaciju distribuiranog sistema za skladištenje podataka, u kojem se koristi Load Balancer (LB) komponenta za raspodelu zahteva i više Worker (WR) komponenti za skladištenje i sinhronizaciju podataka. Problem koji se rešava jeste ravnomerna raspodela opterećenja i obezbeđivanje dostupnosti svih podataka u sistemu, čak i kada se neki od Workera isključe.

Cilj zadatka je:

- Implementacija distribuiranog sistema za skladištenje podataka.
- Korišćenje TCP/IP mrežne komunikacije između komponenti.
- Implementacija mehanizma za ravnomernu raspodelu podataka i njihovu sinhronizaciju između WR-ova.
- Obezbeđenje otporne i stabilne komunikacije uz logovanje i nadgledanje opterećenja.

Dizajn

Sistem se sastoji od tri glavne komponente:

- **Load Balancer**: centralna tačka koja prima zahteve od klijenata i prosleđuje ih Workerima sa najmanjim opterećenjem. Takođe obrađuje registracije i odjave Workera.
- **Worker**: skladišti podatke koje primi i sinhronizuje ih sa ostalim Workerima.
- **Klijent**: šalje zahteve za skladištenje podataka.

Razlozi za dizajn:

- Load Balancer bira Workera sa najmanje podataka (na osnovu lokalno čuvane mape opterećenja).
- Svaki Worker sinhronizuje nove podatke sa ostalima pomoću SYNC poruka.
- Prilikom gašenja Workera, podaci koje je on čuvao se redistribuiraju na ostale dostupne Workere.

Strukture podataka

- **Lista (`std::vector`)**: koristi se za lokalno skladištenje podataka u svakom Workeru.
- **Red (`std::queue`)**: koristi se za komunikacioni red za obradu SYNC poruka u posebnoj niti.
- **Mapa (`std::unordered_map`)**: koristi se u Load Balanceru za praćenje broja podataka po Workeru.
- **Set (`std::set`)**: koristi se za evidenciju poznatih portova Workera.
- **Mutex-i i condition variable**: sinhronizacija pristupa deljenim strukturama.

Semantika:

- Svaki podatak je `std::string`, a strukture omogućavaju detekciju duplikata, konkurentno dodavanje i sinhronizaciju.

Rezultati testiranja

Opis testova:

1. **Test registracije**: Proverava da li se Worker registruje pri pokretanju.
2. **Test usmeravanja**: Proverava da li LB šalje STORE zahtev Workeru sa najmanjim opterećenjem.
3. **Test sinhronizacije**: Proverava da li svi WR-ovi imaju identične podatke nakon STORE zahteva.
4. **Test odjave Workera**: Proverava da li se podaci pravilno redistribuiraju pri gašenju Workera.

Rezultati:

- Pravilna raspodela: svi Workeri imaju jednak broj podataka.
- Pravilna sinhronizacija: svi Workeri imaju sve podatke.
- RAM potrošnja se meri u KB i prikazuje u logu.
- Vreme obrade klijenata meri se u mikrosekundama i loguje.

Zaključak

Implementacija je uspešna. Sistem raspoređuje zahteve ravnomerno, sinhronizuje podatke između Workera i reaguje na promene u mreži (gašenje WR).

- Rezultati su bolji od očekivanih jer je sinhronizacija implementirana efikasno, bez korišćenja eksterne baze.
- Memorija se koristi efikasno (nema curenja), a sistem je stabilan i pod stresom.

Potencijalna unapređenja

- Implementacija replikacije sa detekcijom neuspeha (fallback mehanizam).
- Vizuelni interfejs za praćenje stanja LB i WR komponenti.
- Napredniji algoritam za balansiranje
- Korisnički interfejs za slanje zahteva i pregled stanja sistema.
- Zamena STL struktura korisničkim implementacijama (to je već urađeno za queue i delimicno za hashmap).