УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

PROJEKTNI ZADATAK-LOAD BALANCING

Mentor:
Saša Tošić

Tim:
Emilija Balaž PR140-2019
Cvijetin Glišić PR137-2019

Novi Sad, 22.01.2023.

Sadržaj

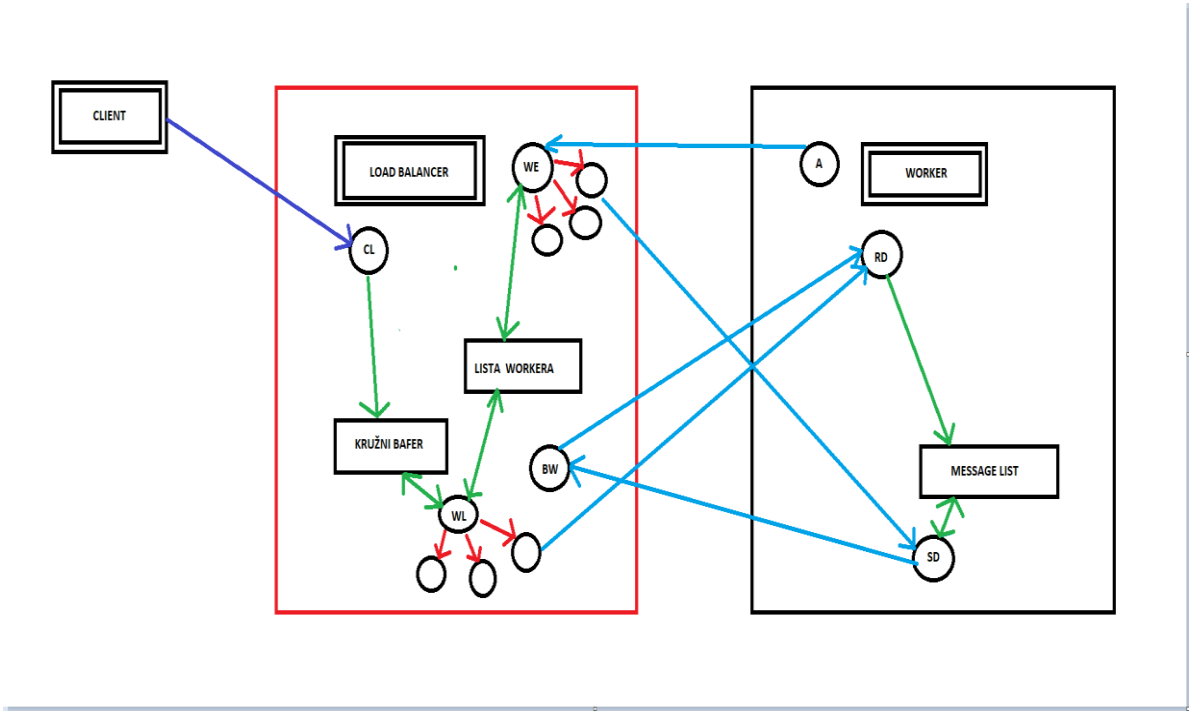
1.UVOD

Projekat Load Balancing se sastoji od tri glavne komponente: Load Balancera, Servera i Klijenta. Svaka komponenta ima drugačiju funkcionalnost i zadatak, ali su međusobno povezane i čine jedinstven sistem.

'Podela opterećenja' je odličan za raspodelu odgovornosti i povećanje performansi u sistemu. Ponaša se kao 'saobraćajni policajac' koji usmerava zahteve pristigle od klijenta serverima koji mogu da ispune te zahteve. Na ovaj način maksimizira brzinu i iskorišćenost kapaciteta i uspešno obezbeđuje da ni jedan server ne bude preopterećen. U slučaju da se neki od server ugasi ili postane nedostupan iz nekog razloga, Load Balancer preusmerava zahteve na servere koji su spremni da opsluže iste. Kada se javi novi server tj.u našem projektu Worker, Load Balancer distribuira podatke odmah i na njega – smanjuje opeterćenje ostalih servera.

2.DIZAJN

Klijent posalje podatak na Load Balancer koji ga trenutmo skladisti u svom kruznom baferu. Iz kruznog bafera preko niti WL koja pravi thread pool poruka se salje na Worker. Poruka se preko RD smesti u listu na workeru. (Message List). Nit A služi da se javi load balanceru da je aktivan, koja ga preko niti WE smesta u listu workera da bi vodili racuna o tome koliko imamo worker rola i ko se javio od njih.Distribucija podataka se obavlja preko thread poola koju je napravala nit WE, da bi se podaci ravnomerno skladistili. Pomocu SD niti poruka se vraca nazad na BW nit gde BW nit sluzi za to da bi uzeo podatak od workera i odmah ga posalje odgovarajucem worker (redistribucija). SD se pokrece samo kad mu WE javi da treba da vrati poruke. WE ce mu javiti ako vidi da je razlika izmedju datacount od workera veca od 1.(WE proveri da li treba redistribuirati podatke, ako treba onda pokrece nit iz poola koje javljaju workerima da imaju vise poruka nego sto treba da ih vrate nazad. Ako ima vise poruka salje se na load balancer, koji ih prosledi Workeru gde preko niti RD razmesti te podatke ravnomerno u message list).

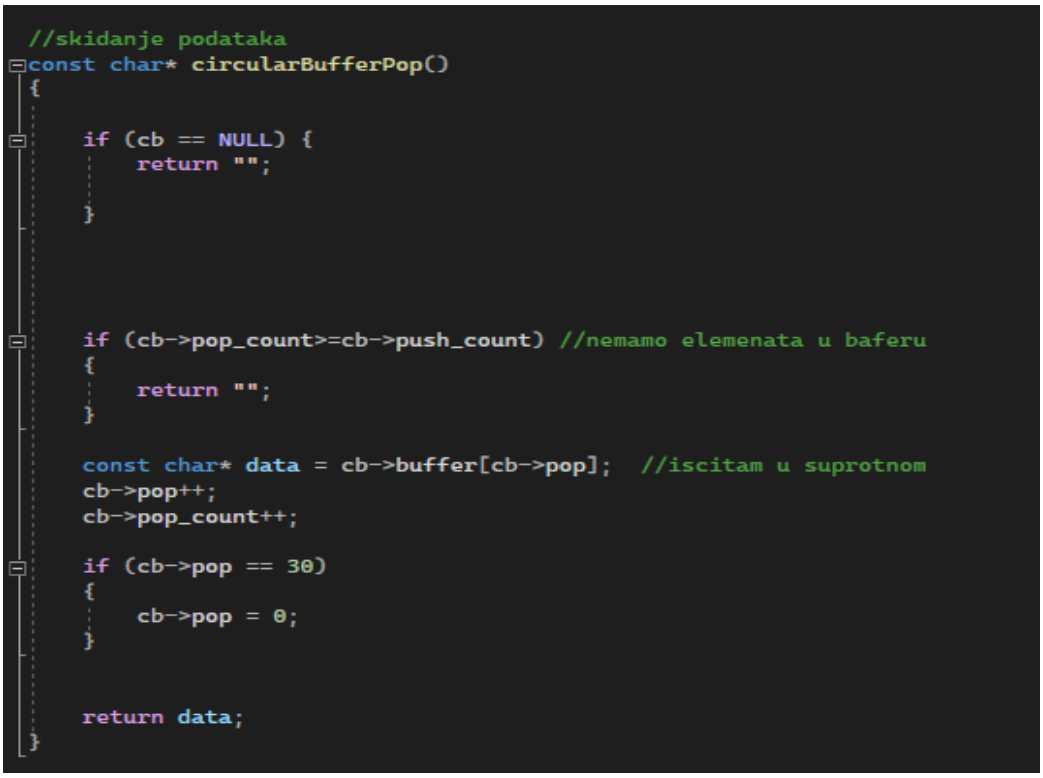


3. STRUKTURE PODATAKA

Projekat sadrži dve liste, kružni bafer i dva pool thread-a. Jedna lista je da se beleže Workeri koji su aktivni i ona je implementirana na Load Balanceru, dok se druga nalazi na Workeru i služi za zapis poruka. Kružni bafer je koristan jer pruža brz način za skladištenje podataka i efikasno koristi memoriju. Pritom mu je implementacija veoma jednostavna. Za implementaciju se koristi dva pokazivača, za upis i za čitanje podataka. Pokazivač upisa pokazuje na lokaciju gde će se upisati podatak, dok pokazivač čitanja pokazuje na najstariju nepročitanu vrednost u baferu. Pokazivač pisanja se uvećava nakon svake izvršene operacije pisanja, dok pokazivač čitanja nakon svake operacije čitanja. Pokazivač se uvećava dok ne stigne do kraja, tj.dok ne pokaže na poslednju alociranu lokaciju u memoriji. U narednom pokušaju pokazivaču će bitit dodeljena vrednost adrese prve lokacije u alociranom baferu. Samim tim u ovom projektu se poruke klijenta vrte u kružnom baferu i ubrzava se čitav process rada sa istim.

```
int circularBufferPush(const char* data)
{




    if (cb == NULL) {
        cb = (struct circular_buffer*)malloc(sizeof(struct circular_buffer));

        cb->push = 0;
        cb->pop = 0;
        cb->push_count = 0;
        cb->pop_count = 0;

    }
    if (cb->push == 30) //kruzni bafer je pun
    {
        cb->push = 0;    //sada je pokazivac opet na nultom mesto, jer je stigao do kraja
    }


    strcpy_s(cb->buffer[cb->push], data);  //stavi se podatak u kruzni bafer
    cb->push++;  //pokazivac se pomeri za +1
    cb->push_count++;



    return cb->push_count;
       //upisan je podatak, counter se poveca za jedan
}
```

Slika 2 – Metoda dodavanja podataka u kružni bafer



```
//skidanje podataka
const char* circularBufferPop()
{

    if (cb == NULL) {
        return "";

    }




    if (cb->pop_count>=cb->push_count) //nemamo elemenata u baferu
    {
        return "";
    }

    const char* data = cb->buffer[cb->pop];  //iscitam u suprotnom
    cb->pop++;
    cb->pop_count++;

    if (cb->pop == 30)
    {
        cb->pop = 0;
    }


    return data;
}
```

Slika 3 – Metoda iscitavanja podataka iz kružnog bafera

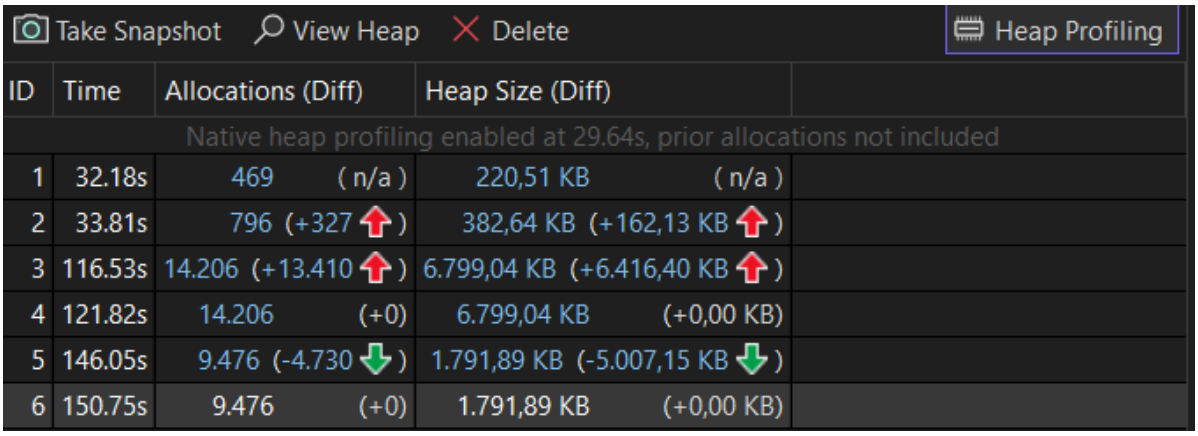
Na slikama 2 i 3 je prikazana implementacija našeg kružnog bafera u projektu.

Osim listi i kružnog bafera implementirano je u kodu sva pool thread-a radi bržeg izvršavanja koda. Ovo je mehanizam koji se koristi radi kreiranja unapred određenog broja potrebnih niti, kako bi se izbeglo kasnije kreiranje zbog toga što je kreiranje niti veoma skupa operacija. Umesto kreiranja niti, uzima se već spremna nit iz Thread pool-a i nakon izvršenog zadatka vraća se u thread pool nazad. Jako je efikasan i veoma ubrzava aplikaciju. U projektu je implementirano dva Thread pool-a, gde svaka sadrži po tri niti.

4.REZULTATI TESTIRANJA

Rezultati testiranja nisu skroz uspešni. Deo memorije je uspešno oslobođen, dok ostatak nije, kao što se može videti na slici.

Memorija je oslobađana u listi poruka na Worker komponenti, kao i u kružnom baferu na Load Balanceru i takođe oslobođena je i lista workera.



Slika 4 - Snapshot

5.ZAKLJUČAK

Zaključak je da Load Balancer na ovaj način efikasno distribuira zahteve pristigle od klijenta ili mrežno opterećenje na više server. Takođe, obezbeđuje visok nivo dostupnosti i fleksibilnost za dodavanje ili gašenje servera.