IKP
projektni zadatak - Load Balancing

DOKUMENTACIJA

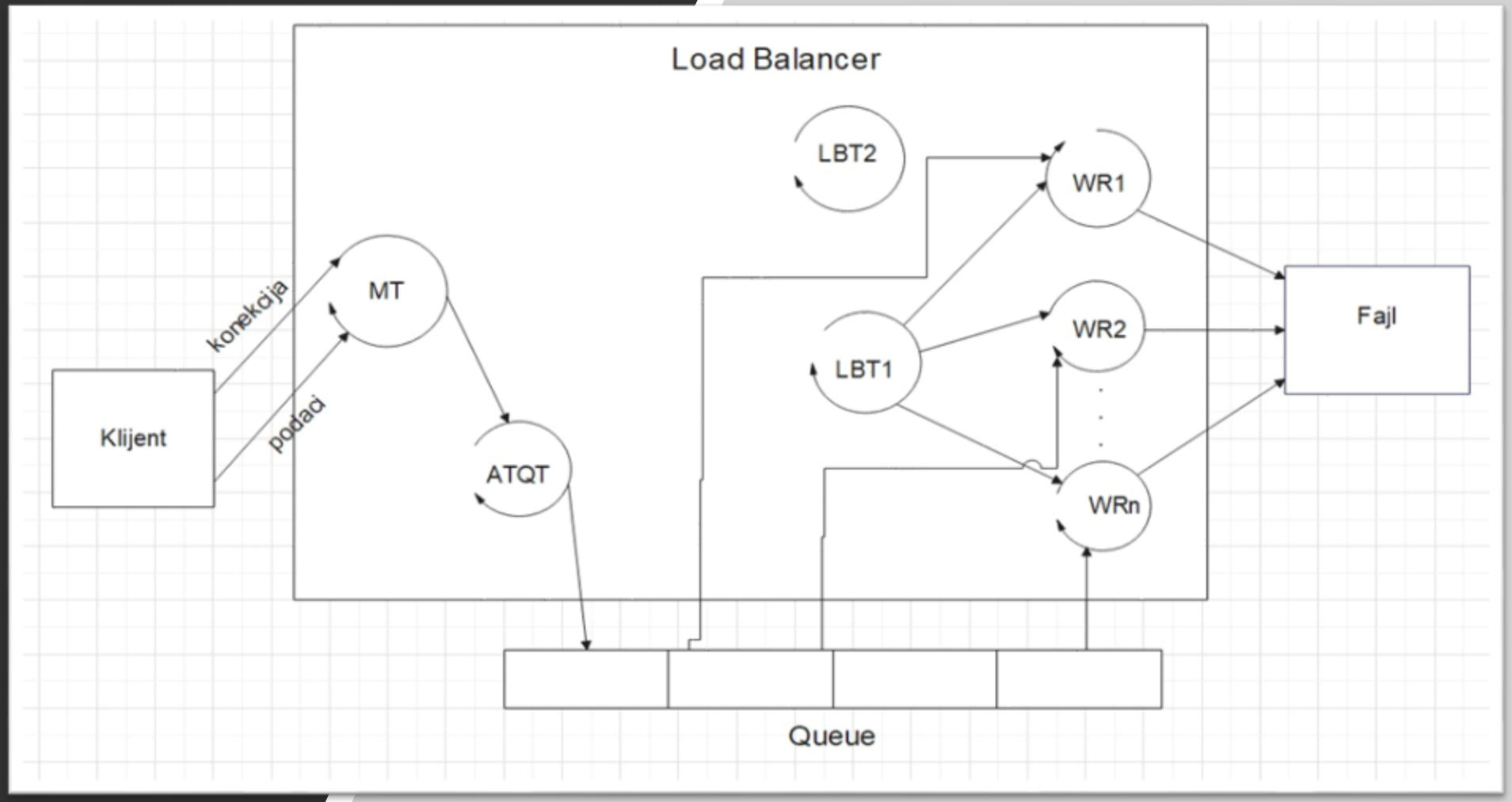
Unkovic Vasilije PR99/2018

Brdarevic Bojan PR156/2018

Uvod

- Zadatak na ovom projektu je da se uspostavi sistem koji vrsi dinamicko balansiranje opterecenja, tako sto ce postojati 2 tipa procesa: Load Balancer i vise Worker Role-a. Load Balancer sluzi da vodi racuna koja Worker Role-a ce da preuzme podatak I izvrsi obradu istog. Broj Worker Role-a nije isti, on varira od broja zahteva koje je potrebno obraditi.
- Zadatak je implementiran prema klijent-server arhitekturi gde je zamisljeno da se sa klijentske strane salju genericki zahtevi za obradu. Server prima te zahteve u vidu integer podataka, skladisti ih u red odakle se oni preuzimaju I Load Balancer procenjuje kojoj Worker Roli ih salje na obradu. Worker Role obradjuju podatke I upisuju ih u file pomocu metode WriteInFile.
- Broj Worker Role instanci nije fiksan, on se menja u zavisnosti od popunjenosti reda, tako da je potrebno implementirati logiku koja ce se baviti ovim problemom. Plan je da postoji nit koja se konstantno izvrsava I proverava popunjenost reda. Ukoliko je popunjenost ispod 30% broj worker role-a se smanjuje dok je obrnuta situacija kada je red popunjen preko 70%.
- Slanje generickih zahteva se vrsi sve dok red nije popunjen, kada se red popuni prestaje se sa slanjem podataka od strane klijenta.

Dizajn


Load Balancer
LBT2
WR1
MT
konekcija
podaci
Klijent
LBT1
WR2
Fajl
ATQT
WRn
Queue

Dizajn

- Dizajn zadatka je formiran na sledeci nacin:
  - Klijent predstavlja komponentu "Klijent" cija je uloga da salje genericke zahteve (int podatke) ka serveru. Na serverskoj strani kada se primi podatak, aktivira se AddToQueueSemaphore koji daje dozvolu niti AddToQueue da primljen podatak s klijentske strane doda u red Q. Uspostavljanje konekcije kao I prijem podataka vrsi glavna main nit.
  - Nakon dodavanja podatka u red, aktivira se semafor LoadBalancerThread1Semaphore koji ce omoguciti niti LoadBalancerThread1 da se izvrsi. Njen zadatak je da odabere worker role-u koja ce obraditi sledeci podatak. Ona to cini tako sto pronalazi prvu slobodnu worker role-u I aktivira njen semafor (ThreadPoolSemaphore).
  - Kada se aktivira semafor neke worker role, ulazi se u kriticnu sekciju gde se preuzima podatak iz reda. Nakon toga sledi obrada podatka posle koje se on prosledjuje metodi WriteInFile u kojoj se podatak zapisuje u neki txt file. Ta worker rola se posle ovoga oslobadja.
  - LoadBalancerThread2 je nit koja se konstantno izvrsava I sluzi da proverava popunjenost reda. Ukoliko je popunjenost preko 70% kreiraju se nove worker role, dok ako je popunjenost ispod 30% aktivira se semafor koji ce omoguciti brisanje odredjene worker role.

Strukture podataka

- Red je strukrura podataka koja je koriscena u zadatku, I u njoj se skladiste podaci poslati od strane klijenta. Prilikom kreiranja reda definisan mu je kapacitet. Prilikom svakog dodavanja podatka u red proverava se da li je popunjen tako sto se porede popunjenost reda (size) I kapacitet reda (capacity). Prilikom svakog preuzimanja podatka iz reda proverava se da li je prazan (provera da li je size 0).
- Zamisljeno je da podaci koji se nalaze u redu budu tipa integer, I oni predstavljaju random generisane brojeve poslate sa klijentske strane ka serveru.
- Pored queue strukture, projekat poseduje I niz niti kao I semafora, koji su korisceni za implementaciju Thread Pool mehanizma konkurentnog programiranja. Zamisljeno je da worker role u stvari predstavljaju niz niti I semafora koje se koriste po potrebi, gde se takodje u odredjenom momentu tim nizovima dodaju nove niti I semafori, ili brisu isti.

Implementacija zadatka

- Za implementaciju zadatka koriscena je TCP konekcija u neblokirajucem rezimu, tako da je aplikacija sadrzi klijentsku I serversku stranu.
- Klijent: na klijentskoj strani, osim logike za uspostavljenje TCP konekcije nije nista dodatno implementirano. Kreiran je connectSocket za uspostavljanje konekcije, kreirana je takodje I adresa na koju se salju podaci, kao I logika za automatsko generisanje random broja koji ce predstavljati podatak koji se salje serveru.
- Server: sto se servera tice, takodje je implementiran serverski deo za uspostavljanje konekcije, u vidu kreiranja listenSocketa I acceptedSocketa gde jedan sluzi za osluskivanje I cekanje zahteva a drugi da prihvati zahtev I omoguci serveru manipulaciju istim.
- Pored pomenutog mreznog dela implementirana je I logika konkurentnog programiranja u vidu niti, semafora kao I kriticnih sekcija. Kreirane su niti za postavljanje podatka u red – addToQueueThread (ova nit se na dozvolu semafora izvrsava, I njen zadatak je da preko metode 'enqueue' doda podatak primljen od klijenta u red Q). Nakon ovog koraka, aktivira se sledeci semafor koji sluzi da da dozvolu niti loadBalancerThread1 koja ima ulogu LoadBalancer-a (balansira opterecenje worker role-a tj bira kojoj worker role-I prosledjuje podatak). Za potpunu implementaciju zadatka potrebna je jos jedna nit, cija ce uloga biti da konstantno proverava popunjenost reda. Ukoliko je popunjenost ispod 30% aktivirace se Finish signal koji predstavlja semafor koji dozvoljava brisanje worker role-a. Ako je popunjenost reda preko 70% vrsi se kreiranje novih worker role-a. Kada je popunjenost reda izmedju 2 pomenute vrednosti broj rola je 3.

Implementacija zadatka

- Worker role su drugi tip procesa u ovoj aplikaciji, njihov zadatak je da obrade podatke dobijene od strane klijenta. Broj njihovih instanci treba da varira u zavisnosti od popunjenosti reda, tako da je ovaj problem resen implementiranjem niza niti I semafora sto predstavlja thread pool mehanizam. Niz niti se zove ThreadPoolThread, I funkcija koja se izvrsava u toku rada ovih niti (worker role-a) je workerRole funkcija. Ova funkcija poseduje 2 tipa semafora I ako je aktiviran Finish semafor, onda se vrsi brisanje odredjene worker role, dok ukoliko se aktivira ThreadPoolSemaphore vrsi se preuzimanje podatka iz reda, njegova obrada I upisivanje u txt file pomocu metode WriteInFile.
- U txt fajlu se nakon izvrsavanja ove metode nalazi podatak koji je obradjen, I ID worker role koja ga je obradila, te se na osnovu toga moze ispratiti rad worker role-a.

Rezultati testiranja

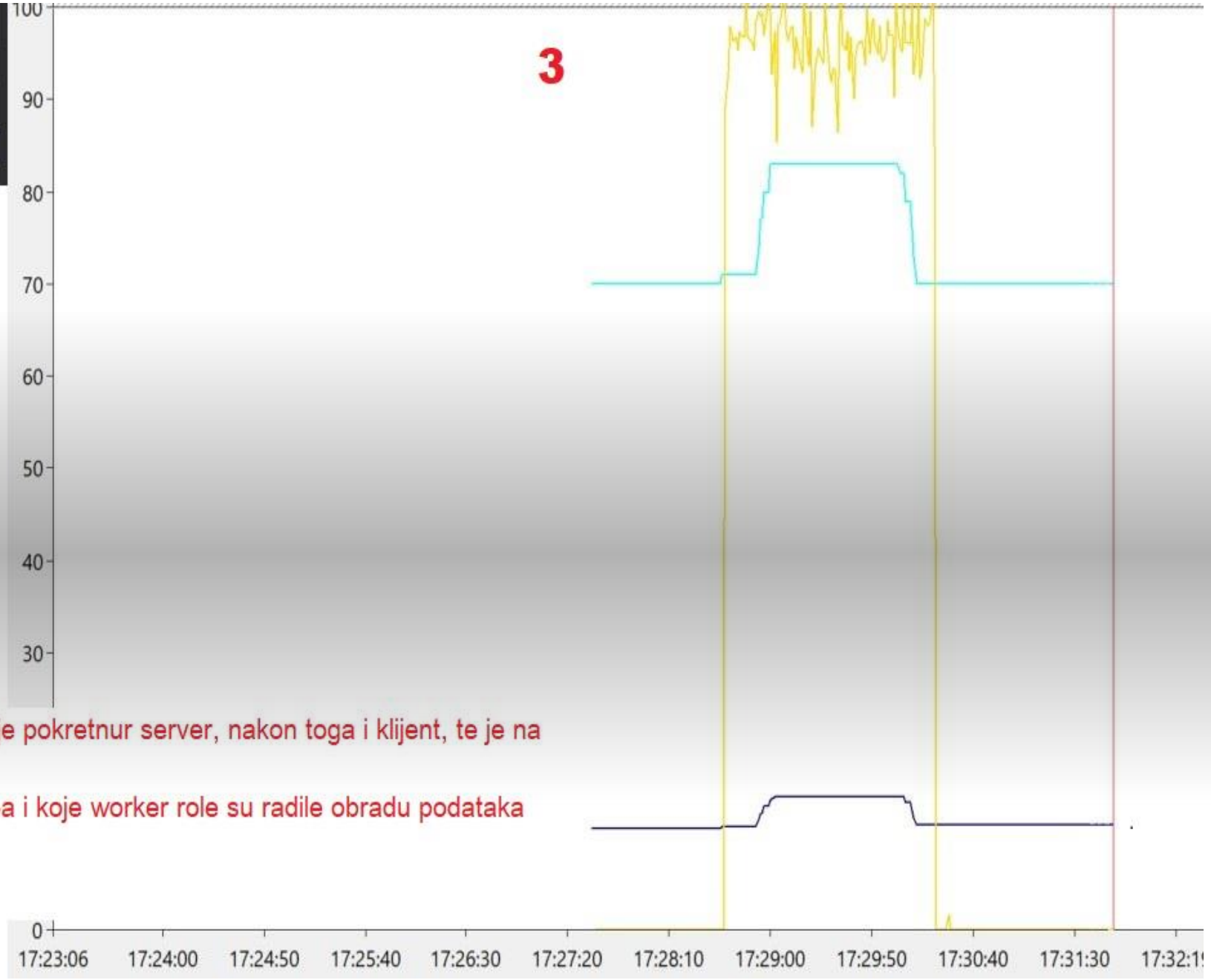
- Projektni zadatak je testiren putem 2 stress testa:
  - Prvi test predstavlja "laganiji" test, u kom nije cilj bio preopterecenje procesora nego pracenje opterecenja prilokom rada aplikacije u normalnim uslovima. Podaci s klijenta su se slali na svakih 1.5 sekundi dok su worker role vrsile obradu na svakih 7 sekundi.
  - Drugi test vazi za test jaceg opterecenja I razlikuje se od prvog u tome sto su se podaci sa klijentske strane slali dosta cesce, na svakih 0.1 sekunde sto je dodatno opteretilo procesor,
  - Slikovit prikaz rezultata testova se nalazi na sledecim slajdovima

ID	Time	Allocations (Diff)	Heap Size (Diff)
1	22.53s	247 (n/a)	98.28 KB (n/a)
2	268.79s	264 (+17 ↑)	105.43 KB (+7.15 KB ↑)
3	363.03s	246 (-18 ↓)	97.54 KB (-7.90 KB ↓)

1

```
Worker role 4: 42
Worker role 1: 43
Worker role 2: 44
Worker role 3: 45
Worker role 0: 46
Worker role 4: 47
Worker role 1: 48
Worker role 2: 49
Worker role 3: 50
Worker role 0: 51
Worker role 4: 52
Worker role 1: 53
Worker role 2: 54
Worker role 0: 55
Worker role 0: 56
Worker role 0: 57
```

2



1 - prikaz zauzetosti heap-a u 3 slucaja (prvobitno je pokretnur server, nakon toga i klijent, te je na kraju ugasen klijent)

2 - prikaz txt file-a u kom vidimo koliko worker role-a i koje worker role su radile obradu podataka

3 - prikaz stanja u performance monitor-u

ID	Time	Allocations (Diff)	Heap Size (Diff)
1	25.05s	247 (n/a)	98.28 KB (n/a)
2	150.47s	501 (+254)	218.22 KB (+119.94 KB)
3	217.61s	247 (-254)	97.63 KB (-120.60 KB)

1

```
Worker role 64: 305
Worker role 25: 306
Worker role 39: 307
Worker role 53: 308
Worker role 12: 309
Worker role 0: 310
Worker role 26: 311
Worker role 40: 312
Worker role 54: 313
Worker role 13: 314
Worker role 27: 315
Worker role 41: 316
Worker role 55: 317
Worker role 14: 318
Worker role 65: 319
Worker role 28: 320
Worker role 42: 321
Worker role 1: 322
Worker role 56: 323
Worker role 15: 324
Worker role 29: 325
Worker role 43: 326
Worker role 2: 327
```

2

Zakljucak

- Nakon testiranja aplikacije putem stress testova, zakljucuje se da nema curenja memorije ni prilikom prvog ni prilikom drugog merenja sto se moze videti na slikama prikazanim na prethodnim slajdovima. Ponasanje procesora se odvija ocekivano.
- Sto se same aplikacije tice zakljucak je da je ispostovan dizajn iste, da su problemi reseni skoro u celosti I da bi uz par unapredjenja u potpunosti ispostovala zahteve.

Potencijalna unapredjenja

- Kako je u zakljucku dokumentacije navedeno, aplikaciju je moguce unaprediti dodatnim implementacijama.
    1. Thread Pool mehanizam implementiran na bolji nacin od postojeceg bi unapredio program, ravnopravnije regulisanje broja worker role instanci je ono sto nedostaje aplikaciji.
    2. Takodje moguce je prestanak slanja podataka s klijenta implementirati na bolji nacin od uradjenog. Nacin na koji je uradjeno je da kada se red napuni, nit addToQueue prestane sa radom, a automatski I nit loadBalancerThread1. Bolje resenje bi bilo prekidanje konekcije sa klijentom gde bi do kraja rada programa samo worker role i load balancer radili svoj posao.