Otkrivanje i uravnoteživanje opterećenja web usluga

Tuđan, Jan

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

JAN TUĐAN

OTKRIVANJE I URAVNOTEŽIVANJE OPTEREĆENJA WEB USLUGA

Diplomski rad

Pula, rujan 2023. godine

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

JAN TUĐAN

OTKRIVANJE I URAVNOTEŽIVANJE OPTEREĆENJA WEB USLUGA

Diplomski rad

JMBAG: 0303085614, redoviti student

Studijski smjer: Sveučilišni diplomski studij Informatika

Predmet: Izrada informatičkih projekata

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacije znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: doc. dr. sc. Nikola Tanković

Pula, rujan 2023. godine

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, **Jan Tuđan** dajem odobrenje Sveučilištu JurjaDobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom

"Otkrivanje i uravnoteživanje opterećenja web usluga"

koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 10. rujna 2023.

Potpis

Jan Tuđan

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani **Jan Tuđan**, kandidat za magistra **informatike** ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljeni način, odnosno daje prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Jan Tuđan

U Puli, 10. rujna 2023.

OTKRIVANJE I URAVNOTEŽIVANJE OPTEREĆENJA WEB USLUGA

Jan Tuđan

Sažetak: U ovom diplomskom radu istražuje se problem otkrivanja i upravljanja opterećenja web usluga u kontekstu modernih informacijskih sustava. Ključno za znati je zašto je otkrivanje usluge važno u kontekstu uravnoteživanja opterećenja i mogu li funkcionirati jedno bez drugoga. Cilj je proći kroz detalje i prokomentirati sve algoritme koji su jedni od najkorištenijih u uravnoteživanju opterećenja. Isto tako napomenut će se horizontalna i vertikalna skalabilnosti za poboljšavanje radnih karakteristika. Detaljno će se pokazati koje mjere postoje u analizi radnih karakteristika. Zbog svega toga, ovo istraživanje teži prema poboljšanju u razvijanju tehnika za ravnotežu u opterećenju usluga. Napravit će se detaljna analiza i usporedba između dva različita načina za uravnoteživanje usluga, s time da je jedan napravljen od nule, a drugi se koristi se kao već gotovo rješenje.

Ključne riječi: otkrivanje usluga, uravnoteživanje opterećenja, raspoređivač prometa, dostupnost i pouzdanost, promet, automatsko skaliranje, resursi, horizontalno i vertikalno skaliranje, mjere

LOAD BALANCING AND SERVICE DISCOVERY

Jan Tuđan

Abstract: In this thesis, we are researching the problem of detecting and managing the load of web services in the context of modern information systems. It is crucial to understand the significance of service discovery in the context of load balancing and whether they can coexist. The main purpose is to go over all of the specifics and comment on all of the methods that are commonly employed in load balancing. Horizontal and vertical scalability to improve performance will also be noted. In the performance analysis, the measurable values will be detailed. Because of all this, this research tends towards improvement in developing techniques for load balancing among the services. A full examination and comparison of two methods for standardizing services will be performed, with one constructed from scratch and the other using an off-the-shelf solution.

Keywords: service discovery, load balancing, load balancer, availability and reliability, traffic, automatic scaling, resources, horizontal and vertical scaling, metrics

Sadržaj

1. Uvod ... 9
2. Web usluge i njihovi izazovi ... 10
   2.1. Otkrivanje usluge ... 11
   2.2. Osjetljivost na prometne fluktuacije i automatsko skaliranje resursa ... 12
   2.3. Dostupnost i pouzdanost web usluga ... 14
   2.4. Upravljanje prometom za optimizaciju radnih karakteristika ... 15
3. Strategije uravnoteživanja opterećenja ... 16
   3.1. Statičko uravnoteživanje opterećenja ... 18
   3.2. Dinamičko uravnoteživanje opterećenja ... 19
   3.3. Algoritmi za raspoređivanje zahtjeva ... 20
      3.3.1. Kružno usmjeravanje ... 22
      3.3.2. Težinsko kružno usmjeravanje ... 23
      3.3.3. IP hash ... 25
      3.3.4. URL hash ... 26
      3.3.5. Algoritam nasumičnosti ... 26
      3.3.6. Metoda najmanje konekcija ... 27
   3.4. Vertikalna skalabilnost ... 28
   3.5. Horizontalna skalabilnost ... 30
4. Mjere za procjenu opterećenja ... 32
   4.1. Količina aktivnih veza i broj zahtjeva ... 35
   4.2. Trajanje obrade zahtjeva ... 38
   4.3. Iskorištenost resursa ... 39
5. Implementacija korištenog raspoređivača prometa ... 39
   5.1. Prednosti Golang jezika za raspoređivanje prometa ... 41
6. Usporedba radnih karakteristika prilagođenog i Nginx raspoređivača prometa ... 42
   6.1. Propusnost ... 42
   6.2. Ukupno korisničko i sistemsko CPU vrijeme ... 44
   6.3. Alokacija memorijske potrošnje ... 46
   6.4. Zauzimanje fizičke memorije ... 48
7. Zaključak ... 51
Literatura ... 53

Popis slika 56
Popis tablica 57
Popis kodova 58

GitHub *Go* raspoređivač prometa: https://github.com/jantudjan99/loadbalancer

GitHub *Nginx* raspoređivač prometa: https://github.com/jantudjan99/nginxLB

1. Uvod

U današnjem sve povezanijem svijetu, web aplikacije ili stranice koje zahtijevaju visok broj korisnika suočavaju se s problemom u preopterećenju zbog gustog prometa u svojim sustavima. Ukratko, uravnoteživanje opterećenja koristi se posvuda u umrežavanju između čvorova ili poslužitelja. U kontekstu web usluga, susrećemo se s njihovim otkrivanjem u opterećenju te kasnije s algoritmima koji su zaduženi za uravnoteživanje opterećenja među poslužiteljima.

Kroz ovu studiju, analizirat ćemo najkorištenije algoritme i strategije za ravnotežu u opterećenju te istražiti njihove prednosti i nedostatke. Također ćemo reći koji je smisao uravnoteženja u opterećenju i koji su razlozi da ga uopće koristimo. Međutim, odluka o korištenju algoritma leži u strukturi i namijeni same aplikacije. Osim algoritama, predstavit ćemo što je horizontalno i vertikalno skaliranje, navesti što ih čini drugačijima i objasniti kada je koje skaliranje bolje za upotrebu. Ovo istraživanje teži doprinosu u boljem razumijevanju i korisnoj primjeni tehnika uravnoteživanja opterećenja u suvremenom digitalnom okruženju. Uz sve navedeno, razmotrit ćemo i moguće izazove te prepreke koje mogu nastati tijekom implementacije ovih metoda u stvarnom svijetu.

Prikazat ćemo implementaciju koda od kojeg je napravljen raspoređivač prometa i pojasniti kako se poslužitelji drže zajedno kao spremnici. Reći ćemo koje tehnologije su korištene i zašto je jezik od napisanog koda važan za uravnoteživanje usluga.

Cilj je posvetiti se istraživanju svih aspekata koji pridonose poboljšanju ravnoteže opterećenja i analizirati ponašanje kod resursa u tom kontekstu. Provest ćemo usporedbu radnih karakteristika i kvalitete našeg raspoređivača prometa, ali i već postojećeg sustava za uravnoteživanje opterećenja u stvarnom okruženju. Kako bismo napravili detaljnu usporedbu, izmjerit ćemo jedne od važnijih karakteristika u ponašanju pri ravnoteži opterećenja kao što su: vrijeme obrade zahtjeva, rad s memorijom, vrijeme rada procesora. Očekujemo da će ovo istraživanje pružiti uvid u razvoj kvalitetnih raspoređivača prometa i njihove radne karakteristike na koje treba obratiti pažnju.

2. Web usluge i njihovi izazovi

Web usluge mogu se odnositi na dva ili više poslužitelja koji koriste standardizirane protokole za razmjenu podataka, kao što su XML-RPC, REST, SOAP i slični [16]. Web usluge korištene su u raznim vrstama web aplikacijama, a to su:

- e-trgovina – razmjena podataka u izvođenju narudžbi i transakcija
- bankarstvo i financije – sigurnosna razmjena podataka između bankovnih računa
- zdravstvo – prikupljanje medicinskih podataka poput dijagnoza, terapija i slično
- medijske platforme – interakcija korisnicima i pristup sadržaju
- logistika – informacije o statusu isporuka, zalihama, narudžbama, transportu

Prednost im je da web aplikacijama ili organizacijama omogućuju sigurno dijeljenje osjetljivih podataka te da isti ne završe dostupni na globalnoj razini. Budući da ih možemo smatrati i dinamičkima, možemo reći i da su automatski skalabilne, što ih čini fleksibilnima te tako smanjuju troškove integracije. One također omogućuju nove poslovne modele i pojednostavljuju Business to Business (B2B) integraciju [16]. Web usluga sama po sebi zahtjeve šalje na više različitih načina poput metoda, a jedne od najpoznatijih su: POST, GET, PUT, PATCH.

U svijetu web usluga, uravnoteživanje opterećenja je zahtjevan atribut koji može imati mnogo zapreka, a one mogu rezultirati neuspjelom komunikacijom sa željenim poslužiteljem. Kao primjer, možemo navesti problematiku gdje je raspoređivač prometa potpuno popunjen, a korisnici bi greškom bili navedeni na popunjene poslužitelje što dovodi do greške 503, koji pak ukazuje na to da je poslužitelj zauzet ili zahtjeva dublju analizu. Također postoje problemi kada je uravnoteženje nejednako raspoređeno, a na to može utjecati poprilično puno faktora. Jedan od njih je neravnoteža sustava na koju utječu resursi poslužitelja. Oni mogu biti različito konfigurirani, a na neravnotežu između poslužitelja mogu utjecati i visoki kapacitet procesora ili više memorije. Problemi s mrežom te pad nekih poslužitelja može preopteretiti druge poslužitelje te negativno utjecati na cijeli sustav. Navedeni problemi mogu biti riješeni korištenjem algoritma za dinamičko uravnoteživanje opterećenja te praćenjem statistike koja prati ponašanje raspoređivača prometa. Isto tako, osigurava se visoka dostupnost i omogućava se fleksibilnost oduzimanja ili dodavanja poslužitelja koji su upućeni prema zahtjevima [3].

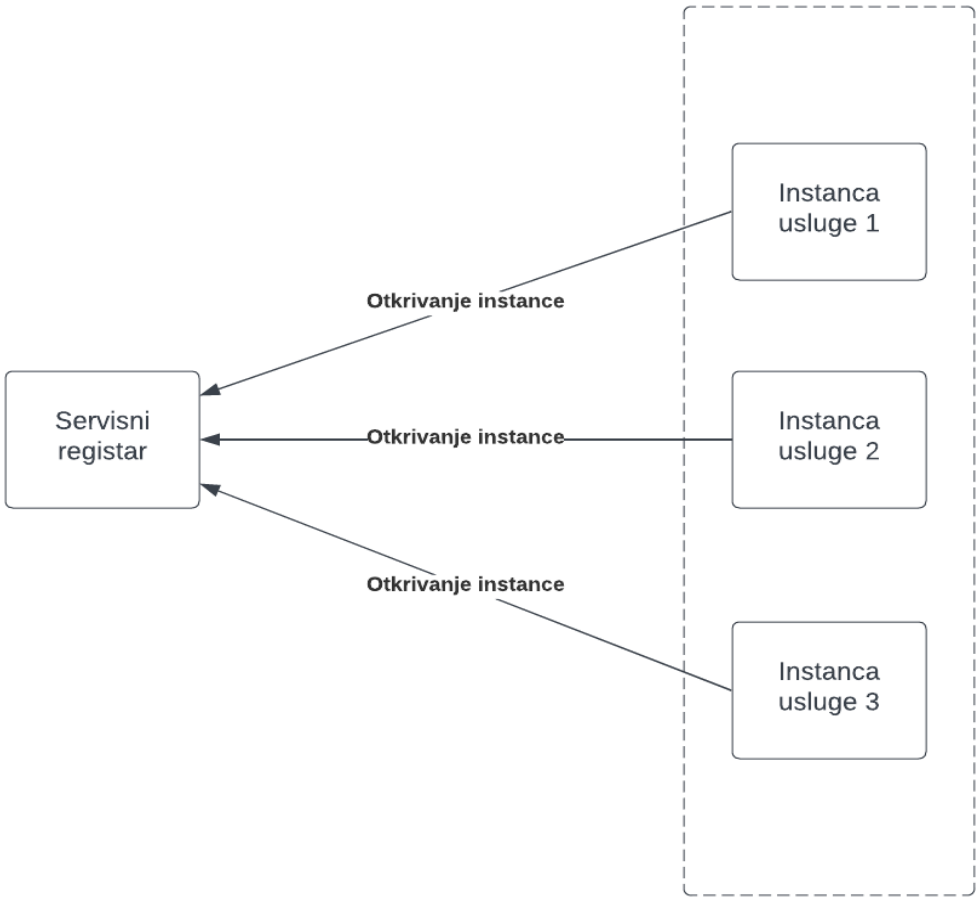
2.1. Otkrivanje usluge

Otkrivanje usluge (eng. service discovery) je proces identificiranja i lociranja bilo kakvih komponenti ili usluga unutar mreže ili distribuiranog sustava u računalstvu. Zbog toga nije potrebno raditi dugi postupak u postavljanju konfiguracije, nego se omogućuje povezivanje uređaja ili usluga bez ručnog konfiguriranja [13]. Otkrivanje usluga je ključna komponenta u mikroservisnoj arhitekturi, oblaku ili nekim sličnim distribuiranim aplikacijama. U kontekstu uravnoteživanja opterećenja, otkrivanje usluga odnosi se na detekciju različitih instanci pojedinih poslužitelja. Upravo to je potrebno kako bi opterećenje bilo bolje optimizirano. Pa na taj način pomoću identifikacije dostupnih usluga, omogućuje algoritmima za ravnotežu opterećenja za rad s tim uslugama. Otkrivanje usluga može identificirati aplikacije na strani poslužitelja (eng. server-side) i aplikacije na strani klijenta (eng. client-side) [13]. **Tablica 1** prikazuje na koje sve načine se usluge mogu otkrivati i koji aspekti su važni u tom procesu.

Načini otkrivanja usluga	**Opis**
Servisni registar (eng. service registry)	Odnosi se na bazu podataka koja sadrži informacije o dostupnosti usluga [13]. Te informacije mogu biti: IP adrese, portovi, aktivnost poslužitelja i slično.
Pružatelj usluga	Prijavljuje se u registar prilikom ulaska i napuštanja sustava [13]. Pružatelji usluge su odgovorni za dostupnost usluge prema drugim korisnicima ili entitetima. Pružatelj usluge može biti pojedinac, neka organizacija, kompanija i slično.
Korisnik usluge	Dostupna mu je lokacija iz servisnog registra, a nakon toga povezuje se sa pružateljem usluga [13].
DNS otkrivanje	Resursi se otkrivaju pomoću DNS upita koji mogu biti traženi putem imena od domene. Ukratko, putem DNS-a dobivene su povratne informacije o resursima.

Tablica 1: Komponente za otkrivanje usluga

Uravnoteživanje opterećenja i otkrivanje usluga dvije su zasebne komponente te mogu funkcionirati jedna bez druge. Međutim, takav slučaj može dovesti do većih problema kao što su nedostatak automatizacije, nedostatak skalabilnosti, slaba dinamička prilagodljivost i slično. Ovi nedostaci mogu zahtijevati statičko konfiguriranje što je zapravo i velik izazov upravo zbog dužeg i zahtjevnijeg ručnog kodiranja i prilagođavanja. Zato bismo mogli reći da je otkrivanje usluga jako važna komponenta koja ravnotežu opterećenja u uslugama čini više automatiziranom i efikasnom.



Slika 1: Otkrivanje usluge u servisnom registru

2.2. Osjetljivost na prometne fluktuacije i automatsko skaliranje resursa

Jedna od opasnosti u prometu kod web servisa je neočekivana opterećenost pri resursima. Takva promjena može uzrokovati usporene radne karakteristike kao što su

zauzeće memorije određenog poslužitelja, manjak procesorske snage ili pad kvalitete usluge [18]. Iznenadni porast broja zahtjeva prema web aplikaciji česta su pojava i zahtijevaju brzu prilagodbu resursa, jer mogu izazvati prekide u izvođenju web usluga. Takvi događaji su neželjeni i neočekivani, pa je za njihovo izbjegavanje potrebno koristiti automatsko skaliranje resursa koje se prilagođava stvarnom vremenu, kako bi prema dostupnim servisima ispravno raspodijelilo stabilnost i kvalitetu usluga.

Prije nego što je automatsko skaliranje postojalo, skaliranje u radnim opterećenjima bilo je izazovno. Ručno dodjeljivanje resursa u radnom opterećenju poprilično je sklono greškama jer je teško biti precizan u predviđanju promjena ili unaprijed znati koliko resursa je potrebno za prilagodbu tih promjena [17]. Cilj automatskog skaliranja je postići da poslužitelji imaju dovoljno resursa za postizanje daljnjih zadataka. Velik doprinos samog skaliranja je optimizacija korištenja resursa i minimiziranje troškova.

Prednosti	**Nedostatci**
Osigurava se da poslužitelj ima dovoljno resursa pri realizaciji izvedbe	Visoko povećanje troškova, pogotovo pri neprestanom skaliranju sustava
Neprekinuta dostupnost i pouzdanost poslužitelja, brza prilagodba resursa prema promjenama	Složenost sustava koja utječe na upravljanje i rješavanje problema
Prilikom dodavanja resursa pomaže se minimizirati otpad, što ispada pogodno i isplativo za korištenje	Skaliranje resursa je vremenski zahtjevno i može dovesti do kašnjenja odgovora u vremenu
Dodavanje ili uklanjanje resursa tijekom povećanog ili smanjenog prometa, poboljšanje radnih karakteristika poslužitelja prema korisnicima je povećano	Pogrešna konfiguracija može dovesti do nepravilne dodjele resursa i smanjuje učinkovitost

Tablica 2: Prednosti i nedostaci automatskog skaliranja resursa

Korištenje automatskog skaliranja rezultira vrlo dobrim korisničkim iskustvom, zbog osiguranog opstanka aktivnih usluga, bez obzira na ikakve promjene. Iskustvo korisnika je zapravo najvažnije mjerilo po kojem najčešće organizacije mjere radne karakteristike u

aplikacijama. U organizacijama, automatsko skaliranje u potrošnji resursa pomaže u prilagodbi prema korisničkom prometu na web poslužiteljima [17]. Također je omogućeno i zadržavanje niskih troškova, čak i ako se opterećenost radnih resursa u organizaciji iznenadno poveća.

2.3. Dostupnost i pouzdanost web usluga

Dvije vrlo značajne metrike u web uslugama - dostupnost i pouzdanost, predstavljaju veliku značajnost u korisničkom iskustvu [2]. Očekivanja korisnika uvijek odgovaraju minimalnom vremenskom čekanju pri izvođenju usluga te njihovoj pristupačnosti. Ciljevi dostupnosti i pouzdanosti se razlikuju te mogu utjecati na to koliko košta održavanje određenih kriterija razine usluge [2].

Dostupnost (eng. availability) u web uslugama znači postotak vremena u kojem aplikacija ili izvođenje zahtjeva ostaju operativni u normalnim okolnostima kako bi mogli služiti svojoj svrsi [2]. Za organizacije dostupnost znači vrijeme u kojem je poslužitelj dostupan ili dostupnost samog zahtjeva od određene web usluge.

Matematička formula za izračunavanje dostupnosti glasi:

$$\text{Razina dostupnosti} = \frac{\text{ukupno proteklo vrijeme} - \text{zbroj vremena prekida}}{\text{ukupno proteklo vrijeme}} \quad [2]$$

Razina dostupnosti web usluge	**Vrijeme izvan usluge u godini**	**Vrijeme izvan usluge u mjesecu**	**Vrijeme izvan usluge u tjednu**
90%	36.5 dana	73 sata	16.79 sati
95%	18.25 dana	36.48 sati	8.39 sati
99.1%	3.65 dana	7.29 sati	1.68 sati
99.99%	52.56 minuta	4.32 minuta	59.61 sekundi
99.999%	5.26 minuta	25.9 sekundi	5.96 sekundi
99.9999%	31.5 sekundi	2.59 sekundi	590 milisekundi

Tablica 3: Pripadajuće vrijeme izvan usluge za određenu razinu dostupnosti web usluge

Tablica 3 dokazuje koliko je razina dostupnosti web usluge važna za organizacije koje posjeduju određene web usluge. Možemo pretpostaviti da je razina dostupnosti od

99.99% na dalje idealna te jedva zamjetna za korisničko iskustvo. S druge strane, u razini dostupnosti od 90%, vrijeme nedostupnosti usluge u jednom danu može biti 2 sata i 24 minute, a to je vrlo loš rezultat.

Pouzdanost (eng. reliablity), s druge strane, se odnosi na vjerojatnost pri ispunjavanju zadanih radnih karakteristika, što direktno utječe na točnost rezultata [2]. Pouzdanost može biti dobra za razumijevanje o tome koliko će uspješno usluga biti dostupna u scenarijima različitih uvjeta u stvarnom svijetu. Slično kao i dostupnost, pouzdanost u sustavu jednako je teško izmjeriti [2]. Ima više načina za mjerenje vjerojatnosti kvara sustavnih komponenti koje imaju utjecaj na dostupnost sustava [2]. Postoji mjera pomoću koje se računa srednje vrijeme između kvarova, a njezina matematička formula glasi:

$$\text{MTBF} = \frac{\text{ukupno proteklo vrijeme} - \text{zbroj zastoja}}{\text{broj kvarova}}$$ [19]

Razina pouzdanosti web usluge	**Ukupno proteklo vrijeme**	**Zbroj zastoja**	**Broj kvarova**	**MTBF**
90%	8760 sati	876 sati	5	1576.8
95%	8760 sati	438 sati	5	1664.4
99.1%	8760 sati	87.6 sati	5	1734.5
99.99%	8760 sati	0.87 sati	5	1751.8
99.999%	8760 sati	0.08 sati	5	1751.98

Tablica 4: Srednje vrijeme između kvarova jednoj godini

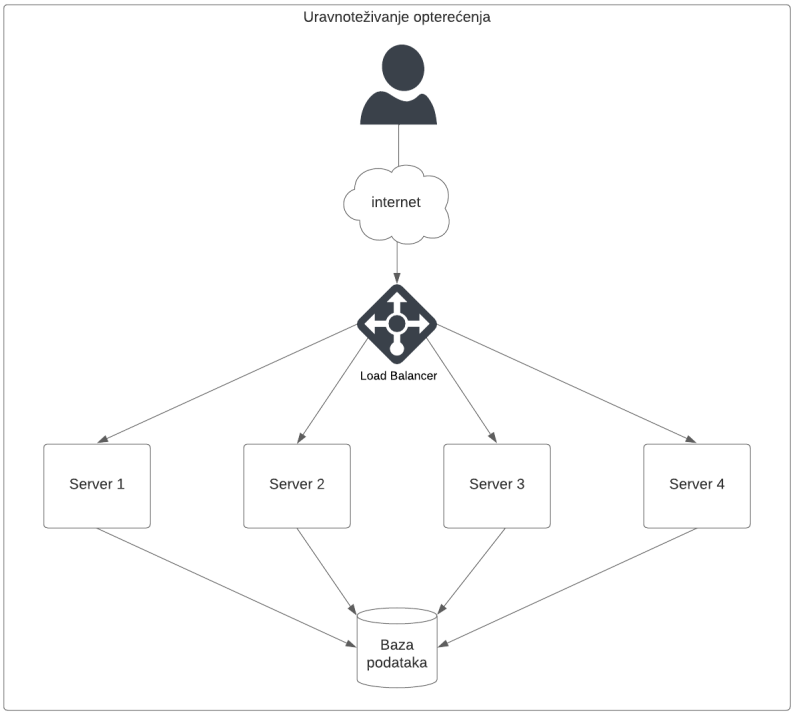
Prema tablici **Tablica 4** možemo vidjeti vremensko trajanje između kvara za komponente sustava. Poput ove formule, postoji i formula za srednje vrijeme do popravka (MTTR), a ta mjera predstavlja vremensko odvijanje popravka određene komponente u sustavu i na taj način cijeli sustav čini dostupnim, prema dogovorenoj obvezi ugovora o razini usluge (SLA) [2]. Postoje i drugi načini mjerenja pouzdanosti poput razina tolerancija na pogreške sustava. Na kraju, može se reći da, što je veća tolerancija greške za određenu komponentu sustava, to će cjelokupan sustav imati manju šansu na osjetljivost od poremećaja pod promjenljivim uvjetima u stvarnom svijetu [2].

2.4. Upravljanje prometom za optimizaciju radnih karakteristika

Upravljanje prometom u web uslugama ili aplikacijama podrazumijeva tehnike za analiziranje, presretanje, dekodiranje i usmjeravanje prometa na webu prema optimalnim resursima koji se temelje na specifičnim pravilima [1]. To se sve zove upravljanje mrežnim prometom, a na taj način se administratorima omogućuje drastično povećanje ukupnih radnih karakteristika. Također, jedno od bitnih svojstava u upravljanju prometom je nadzor svih problema te njihova kontrola u povezivanju i provjeri dostupnosti poslužitelja. Kontroleri u isporukama usluga upravljanju prometom pružaju brzu optimizaciju isporuke i usmjeravanje podataka koji su određenih vrsta. Dijele se na: interaktivni promet, promet osjetljiv na kašnjenje, prometnu gužvu i na promet koji nije u stvarnom vremenu. Interaktivni promet prisutan je u stvarnom vremenu te uključuje zahtjeve i odgovore, a može ga imati web trgovina ili neka interaktivna web aplikacija. Promet osjetljiv na kašnjenje odnosi se na prijenose uživo kao što su prijenos TV programa na poslužitelju, video konferencije, usluge za direktan prijenos video igara i slično. Prometna gužva može predstavljati veliki problem prilikom velikih potrošnja resursa kao što je preuzimanje velikih datoteka, koje može dodatno iscrpljivati dostupnost određene usluge [1]. Promet koji nije u stvarnom vremenu odnosi se na E-poštu i aplikacije za skupnu obradu, a može utjecati i na samu organizaciju prometa [1].

3. Strategije uravnoteživanja opterećenja

Kako bismo korisnicima pružili sigurno i efikasno slanje te primanje informacija, potrebno je razmisliti o strategijama koje pomažu u uravnoteživanju opterećenja. Pri izbjegavanju neispravnosti ili rušenju web poslužitelja potrebni su nam algoritmi za uravnoteživanje opterećenja. Oni omogućuju usmjeravanje svih korisnika prema resursima na uravnotežen način [8]. Dijelimo ih na statično i dinamičko uravnoteživanje opterećenja. Važnost kvalitete raspoređivača prometa je velika jer utječe na postizanje stabilnosti, skalabilnosti i pouzdanosti web usluga. Preko njih možemo mjeriti njihove radne karakteristike i ponašanje uravnoteženih poslužitelja. Zbog toga korisnicima pružamo optimalno iskustvo i smanjujemo red čekanja, što dokazuje samu važnost dobrih i preciznih implementacija za uravnoteženje opterećenja nad resursima.



Slika 2: Uravnoteživanje opterećenja

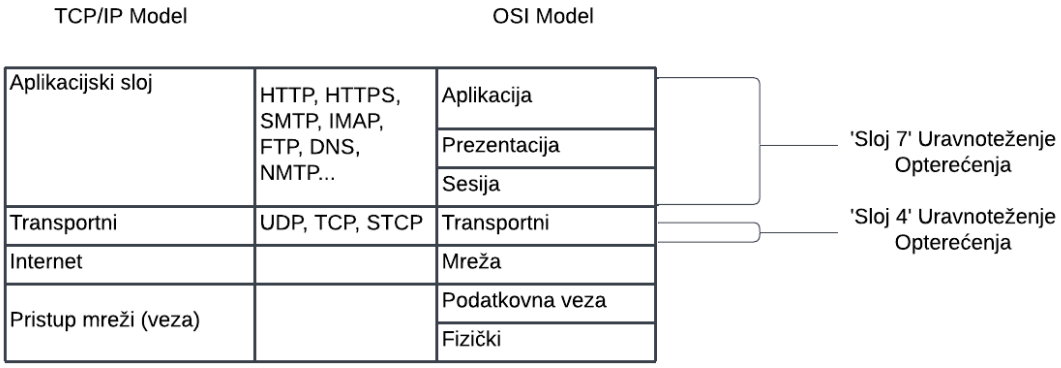
Prema slici **Slika 2** vidljivo je jednostavno uravnoteživanje opterećenja između korisnika, koje je prema algoritmima za uravnoteživanje opterećenja preusmjereno na jednog od poslužitelja, koje također možemo nazivati skup poslužitelja ili farma poslužitelja [3]. Moglo bi se i reći da raspoređivač prometa između tih poslužitelja čeka i usmjerava zahtjeve na način kako bi se brzina i kapacitet ravnomjerno optimizirali. Osim toga, osigurava preopterećenje koje bi moglo vrlo loše utjecati na radne karakteristike. Pri padu jednog poslužitelja, opterećenje je i dalje raspoređeno prema preostalim aktivnim poslužiteljima.

OSI model sastoji se od 7 slojeva u mrežnoj arhitekturi, no u uravnoteživanju opterećenja najviše se ističu transportni i aplikacijski sloj.

Transportni sloj (eng. transport layer) ili sloj 4 značajan je sloj u uravnoteživanju usluga zbog njegove razine prijenosa, zato što upravlja prometom samo na temelju mrežnih informacija poput portova i protokola, bez uvida u sadržaj informacija [4]. Važno je naglasiti da u prosljeđivanju nema dekriptiranja niti nikakvih provjera, nego je sam prijenos efikasan i siguran. Budući da se na ovom sloju odluke na temelju zahtjeva ne mogu

izvršavati, podržan je promet jednostavnijih algoritama kao što je kružno usmjeravanje (eng. round-robin) [4].

Aplikacijski sloj (eng. application layer) ili sloj 7 radi na razini aplikacije i koristi protokole poput HTTP-a i SMTP-a, kako bi se odluke donosile na temelju stvarnog sadržaja u poruci [4]. Za razliku od sloja 4, sloj 7 prekida mrežni promet, izvodi dešifriranje ako je to potrebno i pregledava poruke. Negativna strana aplikacijskog sloja je ta da potreba za enkripcijom može dovesti do smanjenja radnih karakteristika za obradu, ali to se može riješiti upotrebom SSL funkcije. Ovo umrežavanje možemo nazvati „svjesnim“, što donosi inteligentne odluke i dobru optimizaciju sadržaja [4]. Još jedna dinamična karakteristika u ovom sloju je aktivno ubacivanje kolačića. U tom slučaju algoritmi imaju mogućnost za otkrivanje jedinstvenih sesija za klijente, što dodatno osigurava postojanje poslužitelja i ostvaruje veću učinkovitost. Aplikacijski sloj sve češće koriste organizacije koje očekuju ogroman broj korisnika u isto vrijeme. Pomoću tog sloja, protokoli šalju više zahtjeva na jednu vezu i čine optimizaciju prometa i smanjenje opterećenja, što zapravo čini ovaj sloj traženijim za korištenje.



Slika 3: Transportni i aplikacijski sloj u uravnoteženju opterećenja [10]

3.1. Statičko uravnoteživanje opterećenja

Statičko uravnoteživanje opterećenja je pristup ravnoteži opterećenja koje se temelji na već zadanim pravilima za raspodjelu prometa i upravljanju između više resursa [20].

Raspodjela opterećenja ostaje ista tijekom vremena, a jedini način za promjenu je ručno konfiguriranje prema potrebama raspoređivača prometa. Koriste se za jednostavnije web lokacije.

Statičko raspoređivanje zahtijeva manje konekcija, a to smanjuje vrijeme izvršavanja i znatno poboljšava radne karakteristike. Zbog toga je kašnjenje u komunikaciji smanjeno, što dodatno poboljšava iskustvo korisnika. Statičan algoritam pri njegovoj raspodjeli opterećenja gotovo uopće ne uzima u obzir učinkovitosti komponenti, kao što su procesor, veličina radne memorije i propusnost veze [20]. Takav pristup rezultira s malim troškovima i jednostavnom implementacijom te vrlo dobro radi s homogenim poslužiteljima.

S druge strane, jedan od nedostataka statičkog uravnoteživanja opterećenja je težak izračun vremena izvršavanja unaprijed (a priori) i zbog toga dodjela procesa ostaje nepromjenjiva tijekom izvršenja. Kao rezultat toga, komunikacija povremeno može doživjeti nepredvidiva i nekontrolirana kašnjenja. Budući da statičko raspoređivanje ovisi o unaprijed određenim principima, doživljavamo neočekivane skokove u većim preopterećenjima koje je nemoguće predvidjeti u stvarnom vremenu.

3.2. Dinamičko uravnoteživanje opterećenja

Za razliku od statičkog uravnoteživanja opterećenja, u dinamičkom se odluke o usmjeravanju zahtjeva donose u stvarnom vremenu, ovisno u kakvom stanju je trenutni resurs i kakvo je opterećenje u sustavu [21]. Drugim riječima, ovi algoritmi gledaju na stanje pozadinskog poslužitelja i odlučuju o opterećenju poslužitelja prilikom obrade zahtjeva. Algoritmi su vrlo efikasni te donose dobru učinkovitost, skalabilnost i pouzdanost sustava.

Algoritmi za dinamičko uravnoteživanje opterećenja daju više prednosti za razliku od statičkih algoritama. Oni znatno utječu na povećanje radnih karakteristika i učinkovitosti paralelnih programa, tako da se vrijeme izvršenja minimizira, a korištenje resursa maksimizira [21]. Također, bitna komponenta je prilagodba promjenama u radnom opterećenju kojom se izbjegavaju bilo kakve nepravilnosti, nepredvidivost ili heterogenost. Važno je naglasiti i da je raspodjela poslužitelja dosta uravnotežena u pogledu s njihovim resursima, što sprječava da neki resursi ostanu neiskorišteni. Optimizacija tih algoritama

dovoljno je automatizirana, što olakšava ručne izmjene i sprječava vrijeme za otklanjanje grešaka.

Međutim, postoje i negativne strane u dinamičkom uravnoteživanju opterećenja. Jedna od njih je korištenje puno resursa. Zbog njih dolazi do preopterećenja radne memorije, naglog rasta kapaciteta procesora te intenzivne potrošnje na disku. Taj problem zahtjeva detaljno praćenje resursa poput alata za kontrolu mjera radnih karakteristika poslužitelja. Također, latencija u obradi zahtjeva može predstavljati izazov, zato zahtjeva veliku pozornost kako bi se povećala brzina usluge, kako ne bi došlo do nepravilnosti u isporuci zahtjeva.

3.3. Algoritmi za raspoređivanje zahtjeva

Postoje različiti algoritmi za raspoređivanje korisnicima prema resursima. Međutim, svaki od njih ima neke izazove, prednosti i nedostatke. Jedna od prepreka s kojom se suočavamo su kriteriji odabira algoritma za raspoređivanje zahtjeva. Sami kriteriji ovise o tipu web usluge, koji također mogu značajno utjecati na odabir algoritma. U obzir bi se trebali uzeti zahtjevi radnih karakteristika, o kojima također ovisi koji algoritam će se koristiti – jer se svaki algoritam uklapa prema određenim radnim karakteristikama. Periodičko susretanje s fluktuacijom velik je izazov za neke algoritme, ali dobar za one koji su efikasni u rješavanju problema u opterećenju tijekom vremena.

Algoritam	**Funkcija**	**Prednosti**	**Nedostaci**
Kružno usmjeravanje	Ciklička distribucija po zahtjevu klijenata	• jednostavna implementacija • manja kompleksnost • ravnomjerna raspodjela	• neprepoznatljivost stvarnog opterećenja resursa • potencijalno opterećenje • predvidljivost

Težinsko kružno usmjeravanje	Ciklička težinska distribucija po zahtjevu klijenata	• prilagodljiva raspodjela opterećenja • optimizacija resursa	• neprepoznatljivost stvarnog opterećenja resursa • statička konfiguracija težina
IP Hash	Odlučivanje preusmjeravanja preko hash funkcije na temelju IP adrese korisnika	• usmjeravanje istog korisnika na isti resurs • lako održavanje i mali troškovi • robusnost	• neprilagodljivost • problemi s NAT-om • ovisnost o isključivo IPv4 adresama
URL Hash	Kreirani hash se temelji na URL-u od korisnikova zahtjeva	• usmjeravanje prema istom poslužitelju • skalabilnost • održavanje stanja korisnika	• ovisnost o URL-ovima • nedosljednost u slučaju preusmjeravanja
Algoritam nasumičnosti	Raspoređivanje korisničkih zahtjeva prema generatoru slučajnih brojeva (eng. random number generator)	• ravnomjerna raspodjela • brzo reagiranje na promjene • smanjenje predvidljivosti	• zahtijevanje nad resursima (povećanje troškova i kompleksnosti implementacije) • nepredvidljivo ponašanje
Metoda najmanje konekcija	Preusmjeravanje prema poslužitelju s najmanjim brojem korisnika	• optimalna raspodjela • prilagodljivost promjenama • dobro iskorištavanje resursa	• nedeterministički, nepredvidljiv • moguća neravnoteža

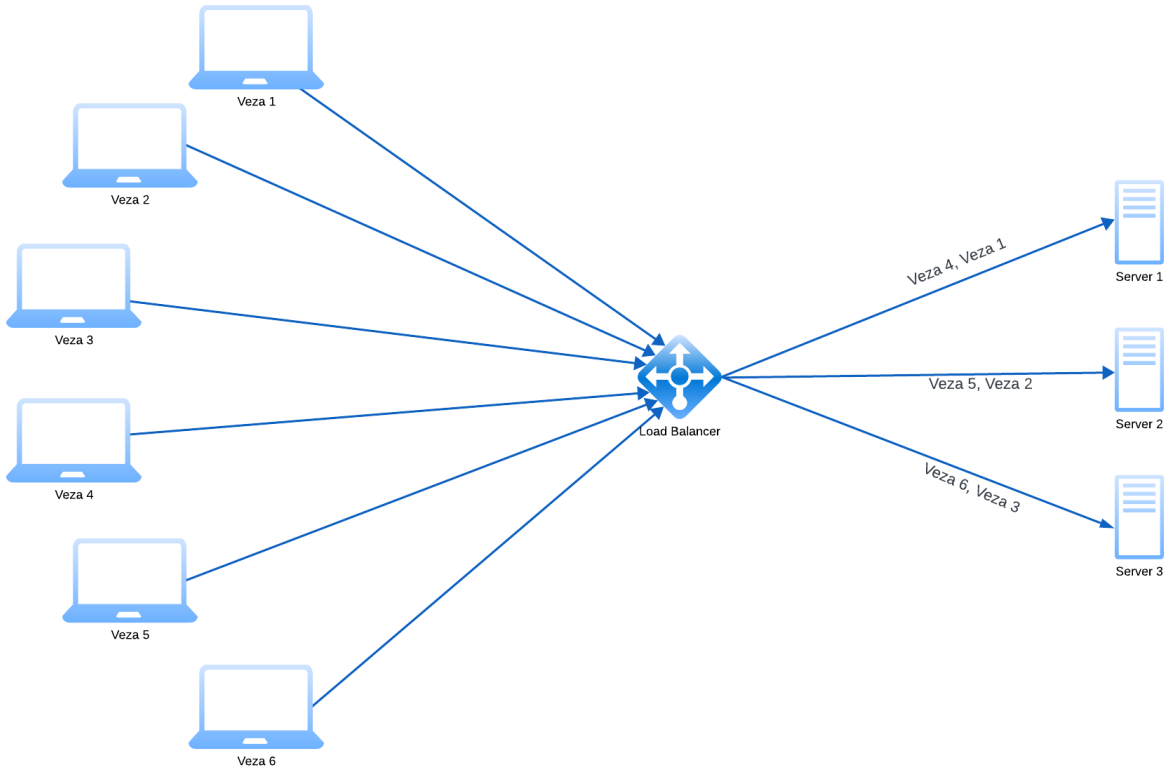
Tablica 5: Vrste algoritama za uravnoteživanje opterećenja

3.3.1. Kružno usmjeravanje

Algoritam često nazivan kao i *Round Robin* jedan je od jednostavnijih i najkorištenijih algoritama za uravnoteživanje opterećenja, koji distribuira zahtjeve u rotacijama, počevši od prvog poslužitelja pa tako po redu do zadnjeg s cikličkim ponavljanjem [5]. Pomoću ove metode, poslužiteljima je omogućeno da rukuju otprilike s istim brojem veza. Ovaj način raspoređivanja opterećenja najbolji je za korištenje, kada poslužitelji imaju gotovo identične tehničke karakteristike, resurse i kapacitet pohrane. Kružno usmjeravanje smatramo statičnim jer nema uvid u trenutno vremensko stanje na poslužiteljima, što ga zapravo čini predvidivim i nije u mogućnosti prepoznati stvarno opterećenje resursa. Ako dođe do promjene u resursima, ovaj algoritam izazivat će djelomično opterećenje.

```
Postavi odabrani server na vrijednost null
Za svaki server u listi svih servera:
     Ako je server aktivan:
          Ako odabrani server nije postavljen(null):
               Postavi server kao odabrani i nastavi na sljedeći server
          Ako je server odabrani:
               Nastavi na sljedeći server
          Ako je server posljednji puta korišten prije nego odabrani:
               Označi server kao odabrani
Vrati odabrani server
```

Slika 4: Pseudokod algoritma za kružno usmjeravanje



Slika 5: Ilustracija kružnog usmjeravanja

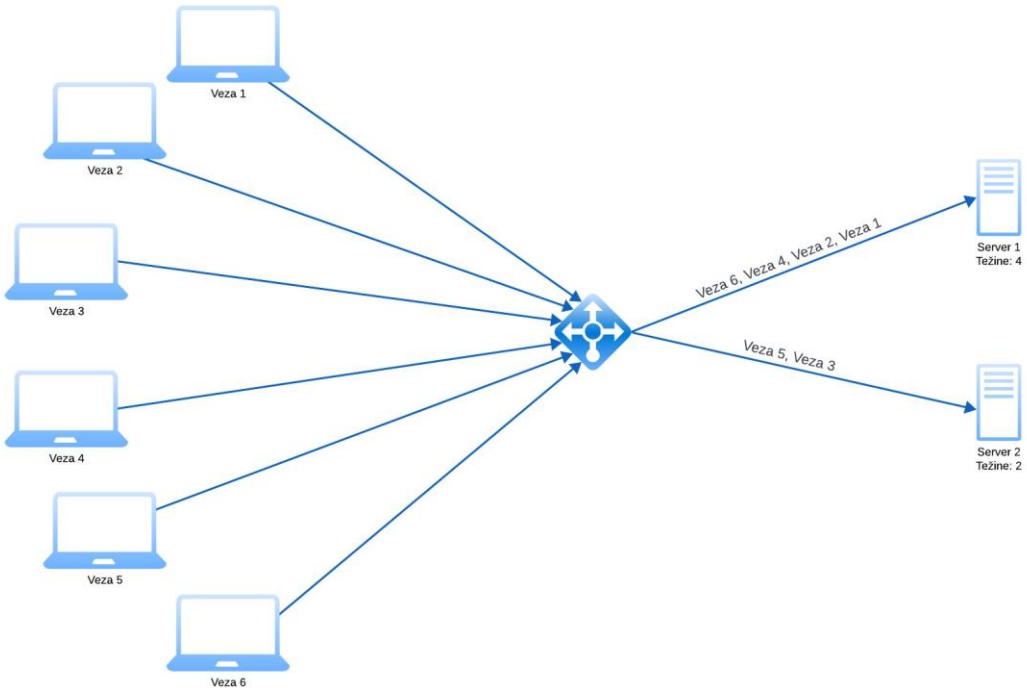
3.3.2. Težinsko kružno usmjeravanje

Ovaj algoritam isti je kao i prethodni, samo što je drugačiji po tome što se odabir poslužitelja razvrstava prema težinama. Oni poslužitelji s najviše težina, preusmjeravat će više korisnika nego oni s manje. Primjerice, ako težina jednog servera iznosi 2, a težina trećeg servera iznosi 6, to znači da će taj treći server primati trostruko više korisnika nego prvi [8]. Tehnike s težinama omogućuju prilagodljivu raspodjelu u opterećenju i opterećenje je kontrolirano više nego u običnom kružnom usmjeravanju. Optimizacija resursa jedno je od bitnijih svojstava u težinskom kružnom usmjeravanju i na taj način poslužiteljima koji zahtijevaju više resursa omogućuje prilagodbu za slanje i zaprimanje više zahtjeva u odnosu na one s manjim kapacitetom resursa. Jedan od glavnih izazova težinskog kružnog usmjeravanja je složenost određivanja težina koje bi bile idealne za određene poslužitelje. Drugim riječima, radi se o ručnom konfiguriranju težina koje može biti zahtjevno i promjenjivo u mrežama s puno poslužitelja. Razlog korištenja ovog

algoritma je raspoređivanje web prometa na poslužiteljima, na temelju njihovog kapaciteta, geografskog položaja ili brzine.

```
Postavi odabrani server na vrijednost null
Postavi ukupne težine na vrijednost 0
Za svaki server u listi svih servera:
     Ako je server aktivan:
          Dodaj težinu u ukupne težine
          Povećaj trenutni server za broj njegovih težina
          Ako odabrani server nije postavljen (null) ili ako trenutni server
ima veću težinsku vrijednost:
               Postavi server kao odabrani
Ako je odabrani server različit od null:
     Smanji trenutni server za ukupnu težinsku vrijednost
     Ako je trenutni server manji od vrijednosti 0
          Postavi ga na vrijednost 0
Vrati odabrani server
```

Slika 6: Pseudokod algoritma za težinsko kružno usmjeravanje



Slika 7: Ilustracija kružnog usmjeravanja s težinama

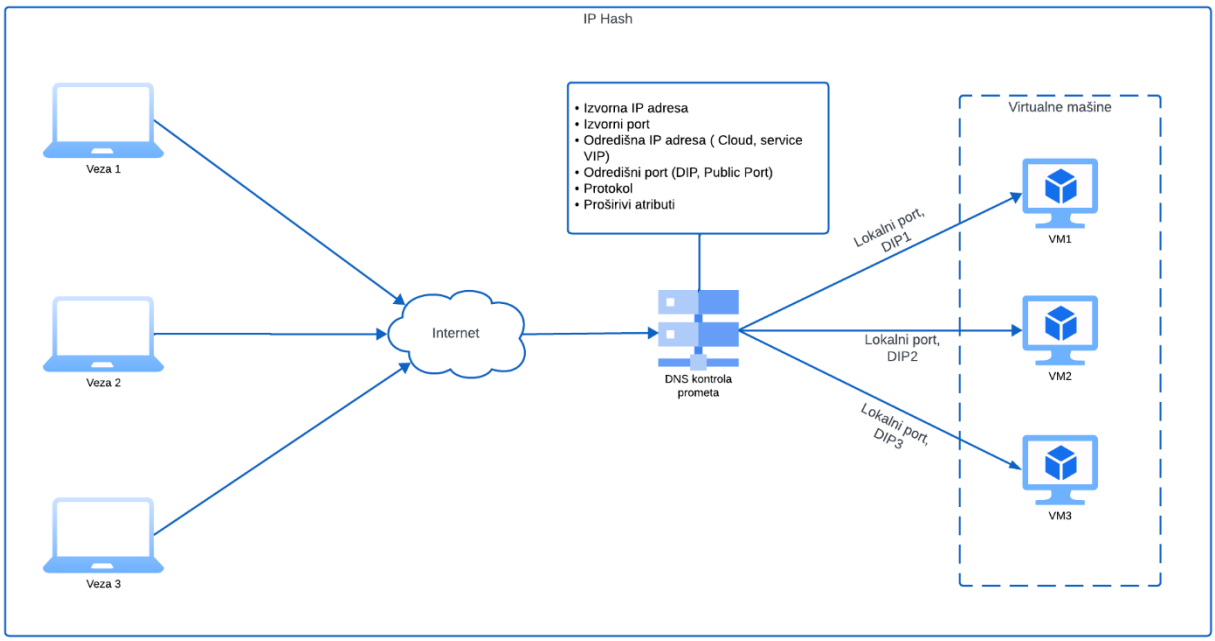
3.3.3. IP hash

Ovakav način uravnoteživanja opterećenja koristi se u sljedećim slučajevima: u aplikacijama temeljenima na sesiji, u scenarijima predmemoriranja, u mikroservisima sa stanjem, u aplikacije s prikazom stanja, u dijeljenju baze podataka i slično. Namjena ovog algoritma je generiranje jedinstvenog hash ključa na temelju IP adrese klijenta i poslužitelja [6]. Pomoću ključa, događa se usmjeravanje zahtjeva i omogućuje se nastavak veze na određenom poslužitelju. Stoga, možemo reći da je ravnomjerna raspodjela IP adresa ključna kako bi se postiglo uravnoteženo opterećenje, a neujednačena raspodjela zato dovodi do neuravnoteženog opterećenja.

```
Stvori novu instancu hash funkcije
Izračunaj hash za korisnikovu IP adresu:
     Postavi korisnikovu IP adresu u hash funkciju h
     Dohvati rezultat hash funkcije kao 32-bitni cijeli broj
Izračunaj indeks servera:
     Izračunaj ostatak pri dijeljenju hash-a sa brojem servera kako bi se
dobio indeks
Vrati server na poziciji index iz liste za servere
```

Slika 8: Pseudokod algoritma za IP hash

Kako bi se koristio algoritam za hashiranje, potrebno je koristiti funkciju za hashiranje. Jedna od njih prigodna za ovaj algoritam je hash funkcija pod nazivom FNV-1a, čija distribucija je izvanredna i ima vrlo nisku stopu sudara. Njezina brzina omogućuje brzo raspršivanje za puno podataka, održavajući razumnu stopu sudara. Pri dobitku izračunate hash vrijednosti, važno je koristiti sumiranje. Koristi se operator za dobivanje ostatka vrijednosti „%“ i omogućuje da indeks bude unutar raspona niza, što svaku IP adresu čini jedinstvenom.



Slika 9: Ilustracija IP hash algoritma

3.3.4. URL hash

Sličan je IP hashu, ali se razlikuju po tome što je URL hash jedinstven, jer osigurava da svi zahtjevi od klijenta za određeni URL uvijek budu preusmjereni na istom poslužitelju.

```
Stvori novu instancu hash funkcije h
Izračunaj hash za URL:
     Postavi URL u hash funkciju h.
     Dohvati rezultat hash funkcije kao 32-bitni cijeli broj
Izračunaj ostatak pri dijeljenju hash-a sa brojem servera kako bi se dobio
indeks
Vrati server na poziciji index iz liste za servere
```

Slika 10: Pseudokod algoritma za URL hash

3.3.5. Algoritam nasumičnosti

S druge strane, nasumični algoritam za uravnoteženje opterećenja omogućuje nasumičnu distribuciju zahtjeva prema poslužiteljima, pomoću generatora za slučajne brojeve [6]. Pri primanju zahtjeva, ovaj algoritam ih nasumično, ali ravnomjerno raspoređuje prema

poslužiteljima. Kao i kružni algoritam, dobar je za rad s grupnim poslužiteljima koji imaju slične konfiguracije.

```
Postavi početnu vrijednost za generator slučajnih brojeva na trenutni trenutak
u nanosekundama
Postavi slučajni indeks pomoću funkcije za generiranje nasumičnih cijelih
brojeva u rasponu od vrijednosti 0 i broja servera - 1servera - 1)
Vrati server na poziciji index iz liste za servere
```

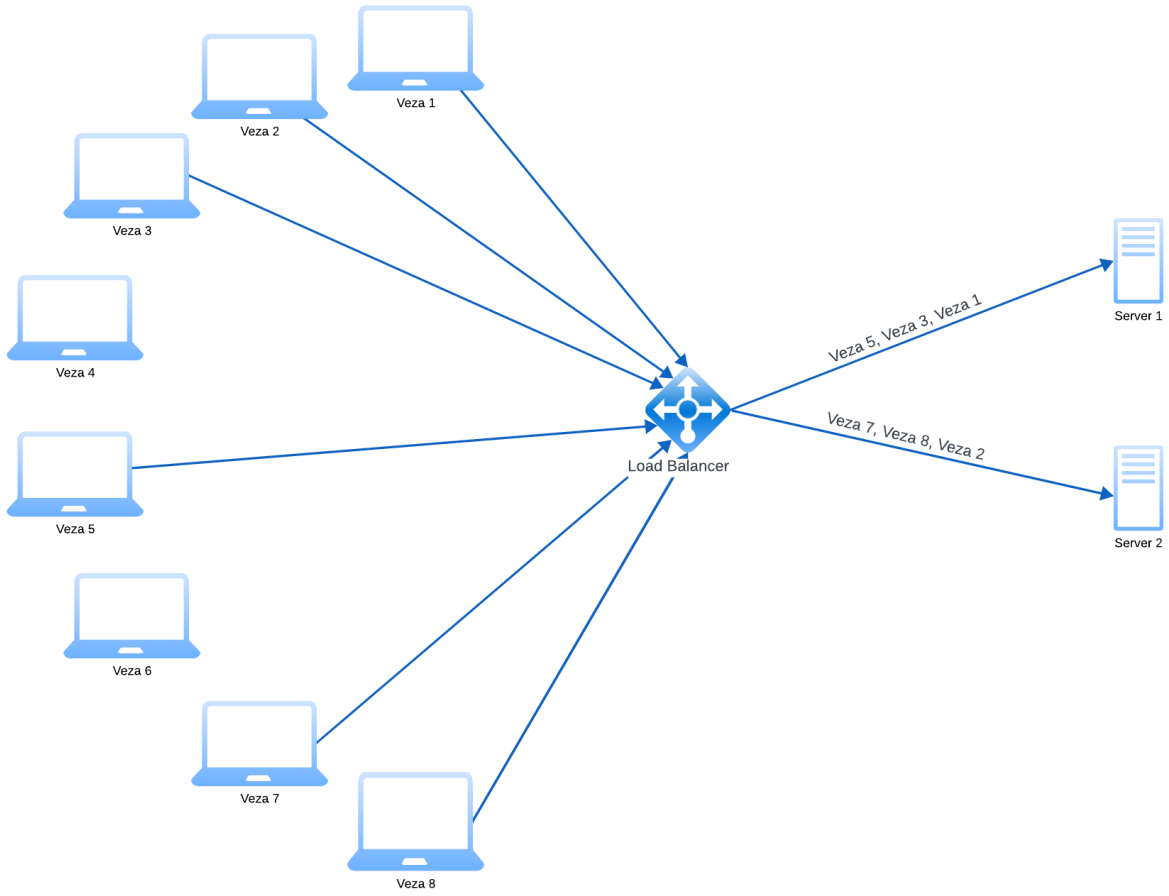
Slika 11: Pseudokod algoritma nasumičnosti

3.3.6. Metoda najmanje konekcija

Tehnika s najmanje konekcija (eng. least connections) uzima u obzir trenutni broj veza na svim poslužiteljima, što drugim riječima znači da radi u stvarnom vremenu pa tako spada u dinamičko uravnoteženje opterećenja [7]. Raspoređivač prometa korisniku šalje zahtjev čiji poslužitelj trenutno ima najmanje aktivnih veza. Jedna od najvećih prednosti ovog algoritma je minimalistička šansa za opterećenjem [7], upravo zbog prijašnje navedenog spajanja prema poslužiteljima s najmanje trenutnih veza .

```
Postavi odabrani server na null
Postavi varijablu za najmanji broj konekcija na – 1 ili neku inicijalnu
negativnu vrijednost
Za svaki server u listi svih servera:
     Ako je vrijednost za najmanji broj konekcija jednaka vrijednosti -1 ili
ako je broj aktivnih konekcija mani od vrijednosti za najmanji broj konekcija:
          Postavi server kao odabrani
          Postavi konekcije servera kao najmanji broj konekcija
Ako odabrani server nema vrijednost null:
     Povećaj broj veza na odabranom serveru za vrijednost 1
Vrati odabrani server
```

Slika 12: Pseudokod algoritma za spajanje prema poslužitelju s najmanje konekcija



Slika 13: Ilustracija algoritma s najmanjim brojem aktivnih veza

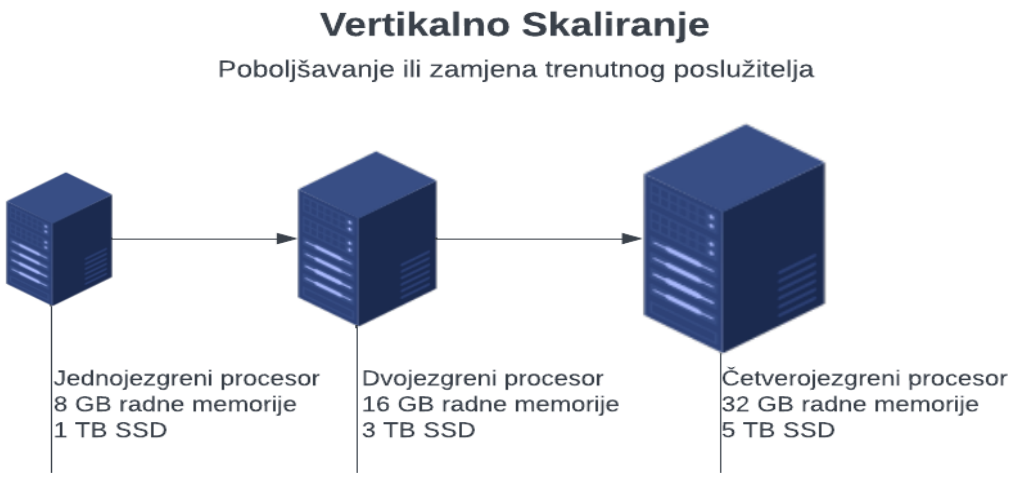
Pretpostavimo da se prema slici **Slika 13** na 2 poslužitelja spojilo 8 veza redom od prve pa sve do osme. U trenutku kod 8 aktivnih veza imamo sljedeći poredak za *Server 1*: *Veza 7*, *Veza 5*, *Veza 3*, *Veza 1*. S druge strane poredak za *Server 2* glasi: *Veza 8*, *Veza 6*, *Veza 4*, *Veza 2*. Pretpostavimo da su se u drugom serveru isključile *Veza 4* i *Veza 6*, pa se na taj način *Veza 7* pridružila u *Server 2* koji je u tom trenutku imao 2 veze manje. Ali zato algoritam s najmanjim brojem aktivnih veza rješava ravnotežu u takvom slučaju opterećenja.

3.4. Vertikalna skalabilnost

Vertikalno skaliranje (često nazivano kao skaliranje prema gore/dolje) koristi se kako bi se povećalo stanje resursa kao što su CPU, RAM, diskovni kapacitet i slično [14]. To se odnosi na uklanjanje ili nadogradnju radne memorije, na zamjenu procesora s boljim radnim karakteristikama i na nadogradnju diskovnog kapaciteta. Takvi resursi mogu biti

dobro skalirani što se odnosi na povećanje resursa na određenoj fizičkoj ili virtualnoj mašini. Vertikalno skaliranje također može biti potpuna zamjena poslužitelja ili prebacivanje radnog opterećenja na druge poslužitelje kako bi resursi bili bolje optimizirani [14]. Još jedan način za dobru vertikalnu skalabilnost je dinamičko prilagođavanje što se odnosi na automatizirano ponašanje prema resursima što smanjiva praćenje i ručno podešavanje.

Skalabilnost je dobro provođena na virtualnim strojevima i njihovim virtualnim operativnim sustavima. Sve u svemu, možemo reći da je vertikalno skaliranje isplativo jer nadogradnja postojećeg poslužitelja profitabilnija je od dodavanja novog [15]. Isto tako održavanje je manje složeno upravo zbog čvorova koji zahtijevaju upravljanje [15]. Ali skalabilnost također ima i loše strane, a jedna od njih je ograničenje resursa u poslužiteljima ili fizičkim hardverima ili povećavanje do te mjere kad se počinje gubiti ravnoteža između poslužitelja. Također takav način može izazvati ogromne troškove i može ispasti poprilično neprofitabilno. Ako se radi o manjem posjedovanju poslužitelja (sa boljim radnim karakteristikama) može dovesti do veće mogućnosti zastoja zbog preopterećenja resursa ukoliko ne postoji rezervni poslužitelj koji će pretrpjeti to opterećenje [15].



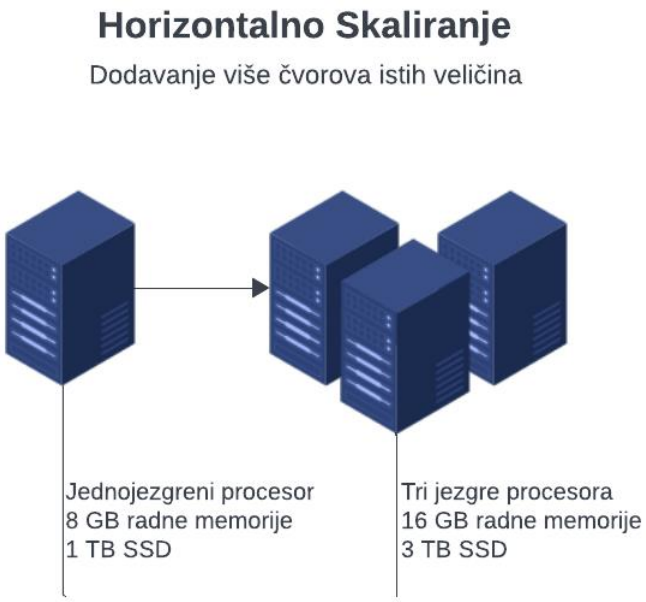
Slika 14: Prikaz vertikalne skalabilnosti [15]

Prema slici **Slika 14** može nam biti jasno što je najbitnije kod vertikalne skalabilnosti. Mogli bismo reći da što su radne karakteristike poslužitelja bolje, to će više korisnika biti na tim poslužiteljima.

3.5. Horizontalna skalabilnost

Horizontalno skaliranje (isto nazivano kao skaliranje lijevo/desno) koristi se kako bi se dodali dodatni čvorovi ili strojevi u infrastrukturu kako bi nošenje sa zahtjevima bilo jednostavnije [15]. Znači da za razliku od vertikalne skalabilnosti, koja se fokusira na poboljšanim radnim karakteristikama jednog poslužitelja, horizontalna skalabilnost fokusira se na rad s više mašina koje imaju nešto manje resurse kao što je RAM, CPU, diskovni kapacitet i tako dalje. S obzirom da horizontalno skaliranje radi s više manjih poslužitelja, mogli bismo reći da je ono jako dobro optimizirano u održavanju ravnoteže između poslužitelja. Znači kao što smo ranije naveli, glavna svrha horizontalnog skaliranja bila bi paralelna obrada sekvencijalnog dijela logike, drugim riječima to skaliranje je namijenjeno za razbijanje u sitnije dijelove i za raspodjelu logike na nove strojeve.

Horizontalna skalabilnost uklanja težnju prema analizi radnih karakteristika koje bi trebale biti za nadogradnju sustava [15]. Još jedna prednost bila bi povećana otpornost na kvarove zato što služi za dodavanje više čvorova ili poslužitelja te time smanjuje mogućnost preopterećenja. Jer ako je jedan poslužitelj pao, ostali će preuzet njegovo opterećenje i to predstavlja dobru optimalnost. Ona je također idealna u mikroservisnoj arhitekturi gdje je više servisa, pa je zbog poželjno koristiti više poslužitelja koji će se međusobno skalirati. Kod horizontalne skalabilnosti također postoje i negativne strane, a jedna od njih je velika složenost u održavanju i kontroliranju, pa bismo i mogli zaključiti da je više poslužitelja puno teže za održavati nego jedan [15]. Također pri velikom broju poslužitelja, ispada neisplativo u početnim troškovima jer je rad s novim poslužiteljima znatno skuplji od onih starijih.



Slika 15: Prikaz horizontalne skalabilnosti [15]

Prema slici **Slika 15** možemo vidjeti kako se podjela resursa prema više poslužitelja ili čvorova radi na uravnoteženi način. Ovakvu strukturu u horizontalnom skaliranju bismo mogli nazvati bazenom poslužitelja koji uravnotežuju opterećenje.

```
  server1:
    build: ./server1
    ports:
      - 192.168.1.6:8001:8001
    networks:
      - lb-net
    deploy:
      replicas: 3
      resources:
        limits:
          cpus: '2'
          memory: '1GB'

  server2:
    build: ./server2
    ports:
      - 192.168.1.6:8002:8002
    networks:
```

```
      - lb-net
    deploy:
      replicas: 3
      resources:
        limits:
          cpus: '2'
          memory: '1GB'

  server3:
    build: ./server3
    ports:
      - 192.168.1.6:8003:8003
    networks:
      - lb-net
    deploy:
      replicas: 3
      resources:
        limits:
          cpus: '2'
          memory: '1GB'
```

Isječak koda 1: Horizontalno skaliranje u docker-compose *konfiguraciji*

U prikazanom kodu možemo vidjeti kako je u *docker-compose* konfiguracijsko okruženje implementirano horizontalno skaliranje. Dodali smo replike (eng. replicas) kako bi svaki server bio kloniran u određenom broju. Također limitirali smo korištenje resursa svakog servera na 2 CPU jezgre i na 1 GB RAM-a (po replici). Kao što smo i prije rekli, korištenje više servera i limitiranje njihovih resursa nam omogućava horizontalno skaliranje pomoću kojeg ostvarujemo bolje rezultate u mjerenju radnih karakteristika servera.

4. Mjere za procjenu opterećenja

Značajna komponenta u uravnoteživanju opterećenja je praćenje karakteristika svih upravljanih poslužitelja. Mjere u kontekstu procjene opterećenja omogućuju bolje snalaženje te bolju optimizaciju funkcioniranja raspoređivača prometa. One također uočavaju preopterećenje u ranim fazama prije dolaska do ozbiljnih problema ili pada sustava. Generalno gledano, metrike mjere zdravlje, kapacitet i adaptaciju u uravnoteženju opterećenja [10]. Korištenje mjera dovodi do donošenja informiranih i pravilnih odluka kako bi opterećenje zahtjeva između raznih poslužitelja bilo više

uravnoteženo. Sve u svemu, mjere igraju veliku ulogu u poboljšavanju korisničkog iskustva.

Razlozi korištenja	**Objašnjenje**
Prikaz stanja u realnom vremenu	Ovo svojstvo jedno je od najvažnijih razloga za korištenje mjera. Nadziranje nad resursima znatno smanjuje šanse za rušenje određenih poslužitelja.
Optimizacija resursa	Mogućnost uočavanja resursa koji su manje iskorišteni, više iskorišteni ili potpuno nepotrebni. Ovo utječe na bolju iskorištenost resursi što dovodi do kvalitetne optimizacije resursa.
Kontinuirano praćenje zdravlja	Naglasak je na održavanju i nadziranju zdravlja poslužitelja. To se odnosi na brzinu njihovih odgovora, njihovu dostupnost i stanje resursa.
Prilagodba Geo-lokacije	Omogućeno praćenje podataka (ukoliko je uključeno) koji pružaju informacije o geografskom položaju poslužitelja i slično. Također omogućeno je vidjeti zahtjeve korisnika i na koji poslužitelj su preusmjereni po geografskom opredjeljenju. Geo-lokacija je najčešće korištena za aplikacije na globalnim razinama koje zahtijevaju visok broj korisnika u isto vrijeme.
Otkrivanje anomalija	Utječe na uočavanje naglog skoka u prometu, odnosno neuobičajenog i naglog rasta zahtjeva po sekundi u prometu. Detekcija anomalije može biti veliki indikator na DDoS napad, tj. probijanje sigurnosti [24]. Drugi razlozi za otkrivanje anomalija bi bili tehnički kvarovi u sustavu ili nagla promjena u ponašanju i u broju korisnika.
Automatska skalabilnost	Mjere omogućuju automatsko reagiranje sustava što stvara dinamičku skalabilnost [17]. Ovo drugim

	riječima pokazuje koliko implementacija automatskog skaliranja dobro ili loše automatski skalira resurse.

Tablica 6: Razlozi korištenja mjera za procjenu opterećenja

Kod praćenja mjera za procjenu opterećenja vrlo je važno imati cilj i nekakvu razgrađenu strategiju o željenom stanju radnih karakteristika. Takva metoda je u praksi vrlo pogodna za korisnika jer omogućuje vrlo dobru i kvalitetnu usporedbu sa trenutnim stanjem radnih karakteristika. To definitivno poboljšava jednostavniju detekciju grešaka i brži te kvalitetniji razvoj uravnoteživanja opterećenja između poslužitelja. Prepoznavanje lošeg ponašanja resursa, kao što su povećanje latencije ili niska razina dostupnosti, omogućuje kontinuirano prilagođavanje i optimizacija resursa. Korištenje mjera također pomaže u uočavanju uskih grla, a šansa za pojavu uskih grla pri povećavanju skale postaje sve veća. Ukoliko dođe do pojave uskih grla, dolazi do težnje za upotrebom više raspoređivača prometa kako bi se zagarantirala dostupnost [11].

```
process_virtual_memory_max_bytes 1.8446744073709552e+19
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes being served.
# TYPE promhttp_metric_handler_requests_in_flight gauge
promhttp_metric_handler_requests_in_flight 2
# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP status code.
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 4315
promhttp_metric_handler_requests_total{code="404"} 1
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
# HELP server_current_connections Current number of connections per server
# TYPE server_current_connections gauge
server_current_connections{server_name="Mock server 1",server_url="http://192.168.1.2:8004"} 0
server_current_connections{server_name="Mock server 2",server_url="http://192.168.1.2:8005"} 3
server_current_connections{server_name="Mock server 3",server_url="http://192.168.1.2:8006"} 2
server_current_connections{server_name="Server 1",server_url="http://192.168.1.2:8001"} 3
server_current_connections{server_name="Server 2",server_url="http://192.168.1.2:8002"} 0
server_current_connections{server_name="Server 3",server_url="http://192.168.1.2:8003"} 9
# HELP server_memory_usage_bytes Memory usage of the server
# TYPE server_memory_usage_bytes gauge
server_memory_usage_bytes{server_name="Server 1",server_url="http://192.168.1.2:8001"} 266160
server_memory_usage_bytes{server_name="Server 2",server_url="http://192.168.1.2:8002"} 342648
server_memory_usage_bytes{server_name="Server 3",server_url="http://192.168.1.2:8003"} 306872
# HELP server_requests_by_route_total Total number of requests by route
# TYPE server_requests_by_route_total counter
server_requests_by_route_total{route="/",server_name="Mock server 1",server_url="http://192.168.1.2:8004"} 2
server_requests_by_route_total{route="/",server_name="Mock server 2",server_url="http://192.168.1.2:8005"} 6
server_requests_by_route_total{route="/",server_name="Mock server 3",server_url="http://192.168.1.2:8006"} 10
server_requests_by_route_total{route="/",server_name="Server 1",server_url="http://192.168.1.2:8001"} 4
server_requests_by_route_total{route="/",server_name="Server 2",server_url="http://192.168.1.2:8002"} 8
server_requests_by_route_total{route="/",server_name="Server 3",server_url="http://192.168.1.2:8003"} 13
server_requests_by_route_total{route="/favicon.ico",server_name="Mock server 3",server_url="http://192.168.1.2:8006"} 1
# HELP server_requests_total Total number of requests received by each server
# TYPE server_requests_total counter
server_requests_total{server_name="Mock server 1",server_url="http://192.168.1.2:8004"} 2
server_requests_total{server_name="Mock server 2",server_url="http://192.168.1.2:8005"} 6
server_requests_total{server_name="Mock server 3",server_url="http://192.168.1.2:8006"} 11
server_requests_total{server_name="Server 1",server_url="http://192.168.1.2:8001"} 4
server_requests_total{server_name="Server 2",server_url="http://192.168.1.2:8002"} 8
server_requests_total{server_name="Server 3",server_url="http://192.168.1.2:8003"} 13
# HELP server_response_status_codes_total Total number of requests by response status codes for each server
# TYPE server_response_status_codes_total counter
server_response_status_codes_total{server_name="Mock server 1",server_url="http://192.168.1.2:8004",status_code="200"} 2
server_response_status_codes_total{server_name="Mock server 2",server_url="http://192.168.1.2:8005",status_code="200"} 6
server_response_status_codes_total{server_name="Mock server 3",server_url="http://192.168.1.2:8006",status_code="200"} 11
server_response_status_codes_total{server_name="Server 1",server_url="http://192.168.1.2:8001",status_code="200"} 4
server_response_status_codes_total{server_name="Server 2",server_url="http://192.168.1.2:8002",status_code="200"} 8
server_response_status_codes_total{server_name="Server 3",server_url="http://192.168.1.2:8003",status_code="200"} 13
```

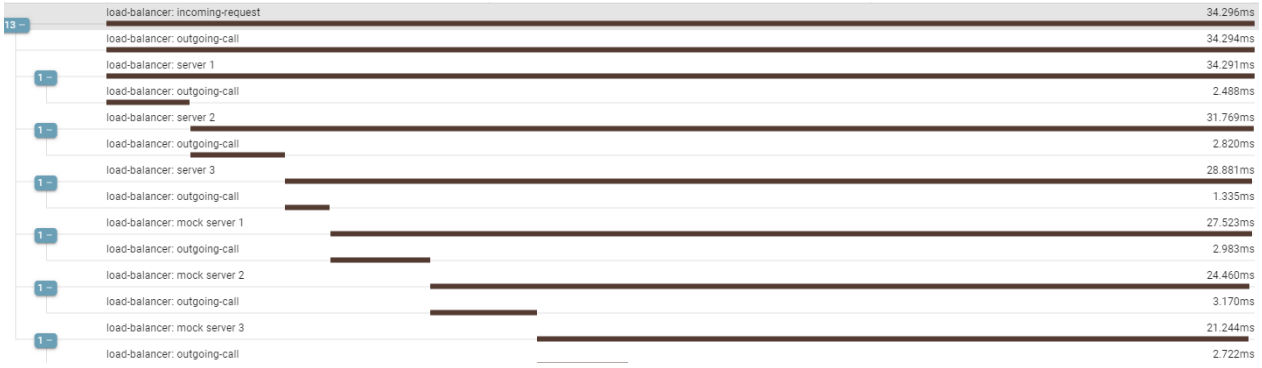
Slika 16: Prikaz nekih ključnih metrika

4.1. Količina aktivnih veza i broj zahtjeva

Mjere za aktivne veze prikazuju aktivnost između klijenta i ciljnog poslužitelja. Ta mjera služi za lakše razumijevanje o tome je li opterećenje ravnomjerno raspoređeno po servisima u klasteru [9]. Broj aktivnih veza korisnika na svakom serveru razlikuje se ovisno o korištenom algoritmu za uravnoteživanje opterećenja.

Slika 17: Grafikon aktivnih veza

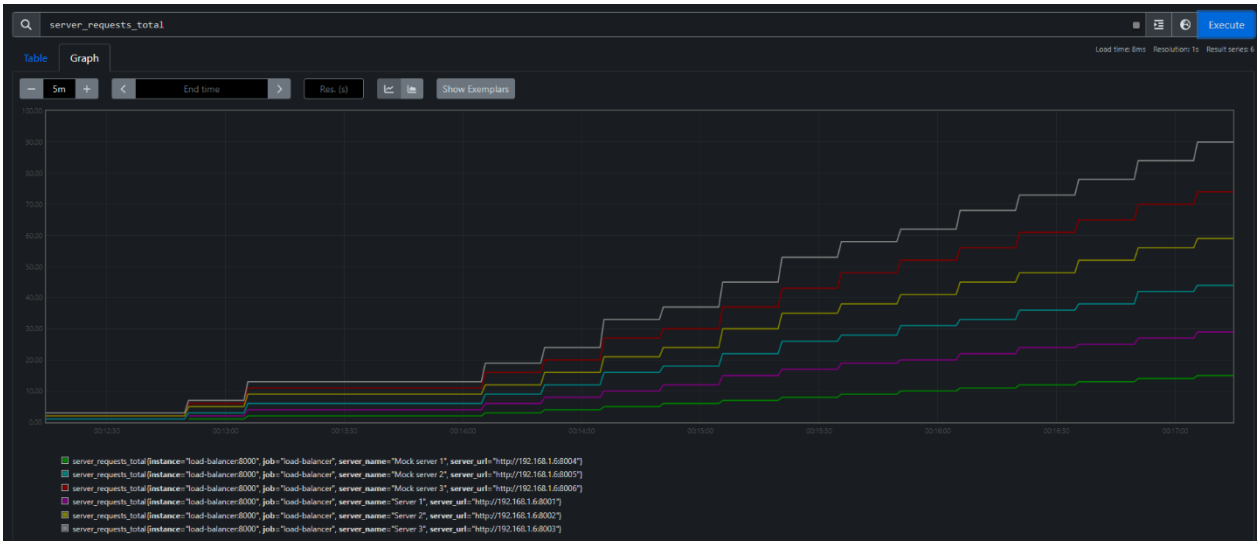
Na slici **Slika 17** vidljiv je prikaz trenutnih korisnika na svakom od servera. U sredini vidi se nagli pad aktivnih veza koji se desio zbog isključivanja svih servera. Svaki od njih ima različit broj poslužitelja, a razlog tomu je uravnoteživanje prema serveru s najvećim brojem težina.



Slika 18: Broj zahtjeva prema poslužiteljima

Na slici **Slika 18** možemo vidjeti ukupno vrijeme izvršavanja zahtjeva na svaki poslužitelj. Za prikaz svih zahtjeva prema serverima koristili smo distribuirani sustav praćenja zvan *Zipkin*. Možemo vidjeti da se radilo ukupno 6 zahtjeva prema poslužiteljima i za svaki od njih pokazano je vrijeme izvršavanja tog zahtjeva u milisekundama. Prikazana su dva vremenska načina slanja zahtjeva: vrijeme izvođenja zahtjeva prema poslužitelju i kompletno vrijeme obrade zahtjeva. Vrijeme izvođenja zahtjeva prema poslužitelju je

samo vrijeme obrade slanja zahtjeva do poslužitelja. S druge strane, kompletno vrijeme obrade zahtjeva tiče se zahtjeva prema poslužitelju, ali i cijelo vrijeme reagiranja servera.



Slika 19: Ukupan broj zahtjeva prema poslužiteljima

Na grafikonu za mjerenje radnih karakteristika možemo vidjeti ukupne zahtjeve prema poslužiteljima od strane korisnika. Možemo uočiti kako svaki poslužitelj ima različit broj zahtjeva, a razlog tome je što algoritam u ovom slučaju raspoređuje na poslužitelje po njihovim težinama, pa isto tako možemo zaključiti da svaki od ovih servera ima različit broj težina. Također, vidljivo je vremensko razdoblje kada ponašanje u grafu miruje, a to znači da u tom momentu nema upućenih zahtjeva prema poslužitelju. S druge strane, skokovi predstavljaju određeni broj zahtjeva usmjerenih prema poslužitelju.

HTTP statusni kodovi	**Broj zahtjeva**	**Opis**
200	507	Statusni kod 200 znači da je zahtjev uspješno proveden
500	0	Statusni kod 500 implicira da se radi o greški u vezi poslužitelja
502	1	Statusni kod 502 znači da je *proxy* ili *gateway* primio nevažeći odgovor od poslužitelja

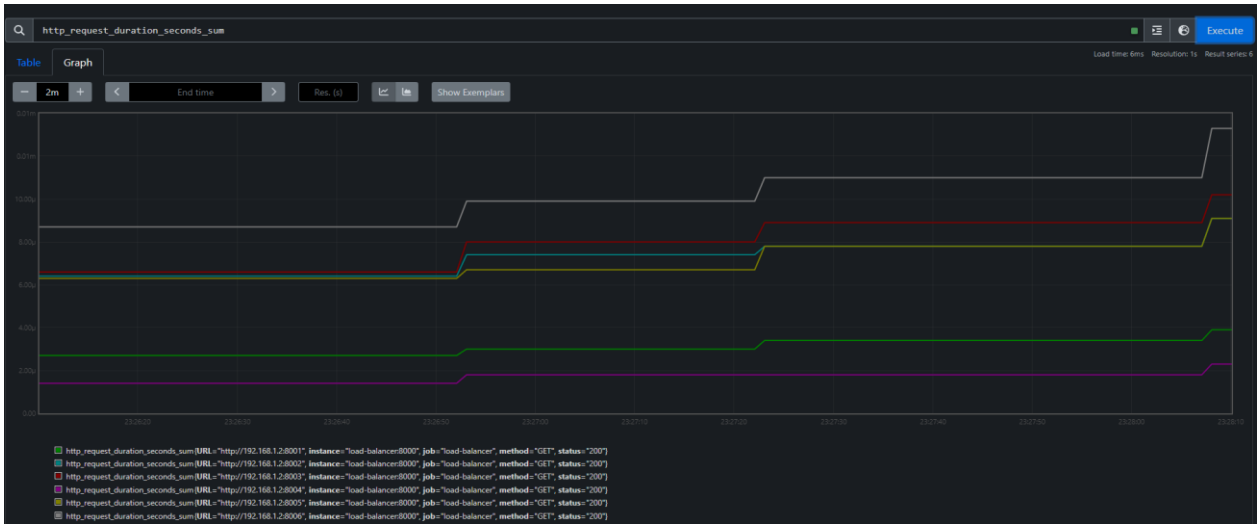
503	0	Statusni kod 503 odnosi se na to da web poslužitelj radi ispravno ali da nije u trenutnoj mogućnosti za obradu zahtjeva

Tablica 7: Rezultati zahtjeva prema HTTP statusni kodovima

U prikazanoj tablici, vidljivo je da su zahtjevi većinom uspješno obrađeni, ali s iznimkom koja prikazuje nevažeći odgovor od poslužitelja. Mjere za prikaz broja zahtjeva u HTTP statusnim kodovima jako su korisne za dobar uvid u ponašanju zahtjeva prema svim poslužiteljima.

4.2. Trajanje obrade zahtjeva

Jedna od važnijih mjera je trajanje obrade zahtjeva, pa možemo pratiti vrijeme da bismo ubuduće mogli optimizirati naš raspoređivač prometa. Trajanje zahtjeva ovisi o puno faktora, a jedan od važnijih je broj istovremenih zahtjeva, pri kojem više korisnika može produljiti obradu zahtjeva. Resursi poslužitelja kao što su memorija ili CPU također mogu biti značajan faktor za usporavanje obrade zahtjeva. Rješenje tog problema bilo bi skaliranje (horizontalno i vertikalno), koje omogućuje povećanje resursa na postojećim instancama, što istovremeno znači i da će se vrijeme obrade zahtjeva smanjiti. Baza podataka web aplikacije također značajno utječe na brzinu slanja zahtjeva, stoga je pametno upotrijebiti indeksiranje tablica, optimizaciju upita i prave baze podataka u određenim situacijama, kako bismo skratili to vrijeme.



Slika 20: Grafikon vremena obrade svih zahtjeva na pojedinom serveru

U primjeru koji se nalazi na slici **Slika 20**, vrijeme obrade svih zahtjeva na pojedinom serveru iznosi između jedne do deset mikrosekundi. Možemo pretpostaviti da je vrijeme izvođenja na svakom zahtjevu fleksibilno.

4.3. Iskorištenost resursa

Mjera za iskorištenost resursa jedna je od ključnih mjera koja nam omogućuje da znatno optimiziramo naš raspoređivač prometa. Pruža uvid u korištenje pojedinačnih resursa, a to omogućuje analizu opterećenja i radnih karakteristika za lakše uočavanje problema. Postoji više vrsta iskorištenosti:

- CPU iskorištenost – prati se koliko procesorske snage troši svaki server ili instanca
- Memorijska iskorištenost – prati se koliko memorije troši svaki server ili instanca
- Mrežna iskorištenost – prati se koliko mrežnog prometa je generirano od svakog servera ili instance
- Diskovna iskorištenost – prati se koliko diska je iskorišteno od svakog servera ili instance

Detaljna analiza mjera za prikaz iskorištenosti resursa pomaže u prepoznavanju uskih grla u klasteru. To ostvarujemo pomoću nadziranja iskorištenosti svih navedenih resursa kako bi problem bilo moguće ukloniti ili razmotriti o korištenju skalabilnosti.

5. Implementacija korištenog raspoređivača prometa

Ključna faza projekta bila je implementacija korištenog raspoređivača prometa. Tijekom razvoja koristili smo jezik *Golang* i izradili aplikaciju koja se sastoji od više spremnika, pri čemu su spremnici konfigurirani i upravljani pomoću softverske platforme *Docker*. Za zajedničku konfiguraciju spremnika koristili smo *Docker* kompoziciju koja je definirana u YAML jeziku koji je namijenjen za upravljanje kontejnerima.

```
version: '3'
services:
load-balancer:
    build: .
    ports:
      -192.168.1.6:8000:8000
    networks:
-lb-net
    depends_on:
```

```
- server1
- server2
- server3
- mock-server1
- mock-server2
- mock-server3
    environment:
-ZIPKIN_SERVER_URL = http://192.168.1.6:9411/api/v2/spans
```

Isječak koda 2: Konfiguracija Docker kompozicije za raspoređivač prometa

U isječku koda **Isječak koda 2** prikazana je *Docker* kompozicija koja prikazuje konfiguraciju raspoređivača prometa i njegove međuodnose sa drugim serverima. Možemo vidjeti da je raspoređivač prometa ovisan o serverima, a njegova ključna uloga je efikasno raspoređivanje prometa među njima kako bi se osigurala optimalna isporuka usluga.

Za pravilan razvoj raspoređivača prometa bilo je potrebno definirati strukture koje sadrže ključne informacije kao što su: URL, ime, težine, trenutačan broj veza i maksimalan broj veza. Također potrebne su i relevantne informacije o raspoređivaču prometa kako bi se ispravno stvarala i održavala ravnoteža. Generalno gledano, ovo su nekakvi osnovni podaci da bi algoritmi za uravnoteživanje mogli raspolagati s tim informacijama koje su esencijalne za opću mogućnost u ravnoteži opterećenja. Ključna je i funkcija koja služi za izradu servera kako bi serveri pomoću podataka iz strukture s tim podacima bili spremni za raspolaganje sa uravnoteženjem.

```
type MockServer struct {
      URL      string
      Name     string
      Weight   int
      Current  int
      MaxConns int
}
type Server struct {
      URL      string
      Name     string
      Weight   int
      Current  int
      MaxConns int
}
type LoadBalancer struct {
      servers[]     *Server
```

```
	mutex         sync.Mutex
	activeServers map[string]bool
	mockServers   []*MockServer
}
func(lb * LoadBalancer) AddServer(name, url string, weight, maxConns int) {
	lb.mutex.Lock()
	defer lb.mutex.Unlock()

	if lb.activeServers == nil {
		lb.activeServers = make(map[string]bool)
	}

	lb.servers = append(lb.servers, &Server{
	URL: url,
		Name: name,
		Weight: weight,
		Current: 0,
		MaxConns: maxConns,
	})
	lb.activeServers[url] = true
}
```

Isječak koda 3: Struktura i konfiguracija raspoređivača prometa

Korištena su dva različita alata kojima je zajednička namjena nadzor i praćenje radnih karakteristika aplikacije. Jedan od njih je *Prometheus* koji se koristi za praćenje vremenskih serija podataka o mjerama radnih karakteristika, izvođenja vremena, ukupan zbroj zahtjeva i slično. Taj alat od značajne je važnosti jer inženjerima i administratorima aplikacija omogućava lakše uočavanje problema preopterećenja. S druge strane, *Zipkin* se koristi za analizu i praćenje zahtjeva koji prolaze prema različitim poslužiteljima.

Go raspoređivač prometa: https://github.com/jantudjan99/loadbalancer

Nginx raspoređivač prometa: https://github.com/jantudjan99/nginxLB

5.1. Prednosti Golang jezika za raspoređivanje prometa

Ovaj jezik preporučljivo je koristiti za održavanje ravnoteže u opterećenju zbog više razloga. Jedan od njih je izvanredna brzina u izvođenju koju pruža *Go* jezik, što je ključno za usmjeravanje prometa u niskim latencijama [22]. Osim toga koristi se takozvani sakupljač smeća (eng. garbage collector) koji automatski odbacuje memoriju koja se više ne koristi i tako omogućuje bržu optimizaciju. Izvršne niti (eng. goroutines) također su isplative za korištenje zato što su njihovi troškovi izrazito niski [23]. One su isto tako dobre za paralelno programiranje, što se u kontekstu raspoređivanja prometa koristi za obradu

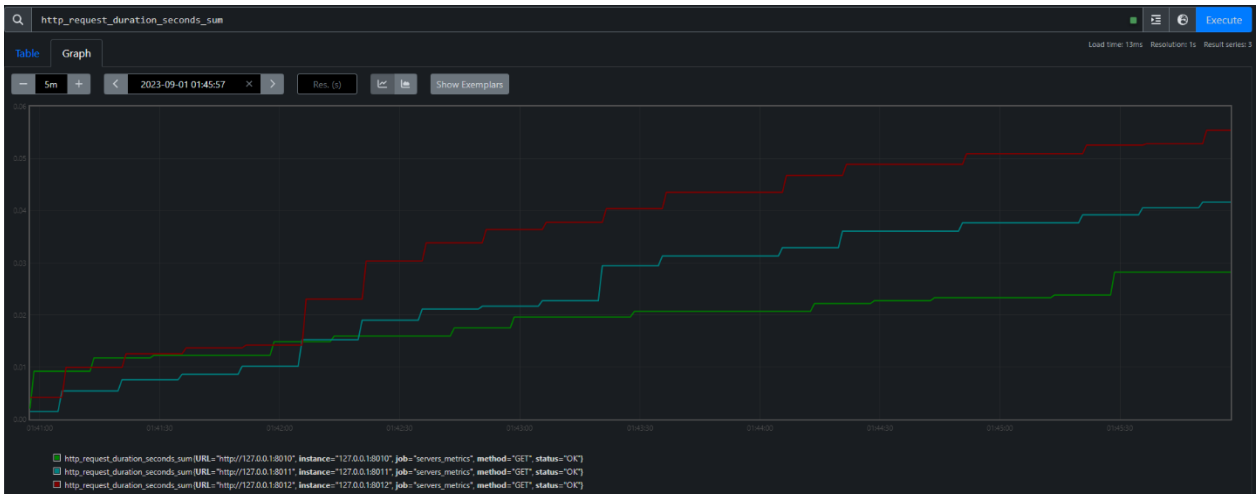
više zahtjeva istovremeno, a to je česta potreba u problemima kod uravnoteživanja opterećenja. *Go* omogućuje efikasno rukovanje s greškama što je isto bitno za raspoređivač prometa.

6. Usporedba radnih karakteristika prilagođenog i Nginx raspoređivača prometa

Važno je imati cilj za unaprjeđenje ravnoteže opterećenja prema poslužiteljima, u ovom poglavlju razmotrit ćemo evaluaciju dvaju tipova raspoređivača prometa te vidjeti koji od njih je bolji u radnim karakteristikama i zašto. Kako bismo izmjerili te radne karakteristike koristit ćemo sustav za praćenje *Prometheus*. Kao mehanizam za postizanje opterećenja prema više poslužitelja, oba analizirana pristupa koriste algoritam za težinsko kružno usmjeravanje – prilagođeno rješenje i *Nginx* rješenje. Pri razvoju strategije za postizanje ravnoteže prema uslugama, uvijek je najbolje mjeriti se prema onima koji su predstavljeni kao vodeći u tom području. Kao što smo već prije naveli, ovo će nam poboljšati njihovu funkciju, efikasnost i iskorištavanje resursa na najbolji mogući način.

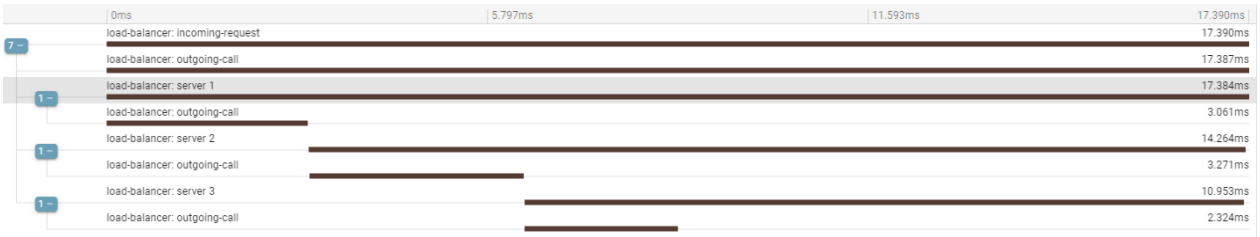
6.1. Propusnost

Jedna od ključnih mjera koju je važno za spomenuti je sposobnost poslužitelja u obradi zahtjeva za određeni vremenski period. Sporiji zahtjevi prema poslužitelju mogu imati vrlo negativan utjecaj prema kvaliteti uravnoteženja opterećenja [9]. Izmjerit ćemo kolika je suma u vremenskom izvršavanju zahtjeva na svakom poslužitelju u odnosu na broj zahtjeva prema poslužiteljima.



Slika 21: Grafikon ukupnog izvršavanja zahtjeva prema poslužiteljima tijekom uravnoteženja opterećenja u *Nginx*-u

Prema slici **Slika 21** možemo vidjeti rast ukupnog izvođenja zahtjeva prema svakom poslužitelju. Možemo vidjeti da *server 1* ima najmanje vrijeme, dok s druge strane *server 3* ima najduže vrijeme izvođenja. Razlog je zato što *server 3* ima najveću težinu, a *server 1* najmanju.



Slika 22: vremenski prikaz izvršavanja zahtjeva prema poslužiteljima u prilagođenom raspoređivaču prometa

Prosječno vrijeme izvršavanja zahtjeva u Nginx raspoređivaču prometa		
Server 1	**Server 2**	**Server 3**
28 ms / 68 zahtjeva = 0.41 ms brzine po zahtjevu	41 ms / 113 zahtjeva = 0.36 ms brzine po zahtjevu	55 ms / 168 zahtjeva = 0.33 ms brzine po zahtjevu
Vrijeme izvršavanja zahtjeva u prilagođenom raspoređivaču prometa		
Server 1	**Server 2**	**Server 3**

10.953 ms	14.264 ms	17.384 ms

Tablica 8: Prosječno vrijeme izvršavanja zahtjeva u *Nginx* i prilagođenom raspoređivaču prometa

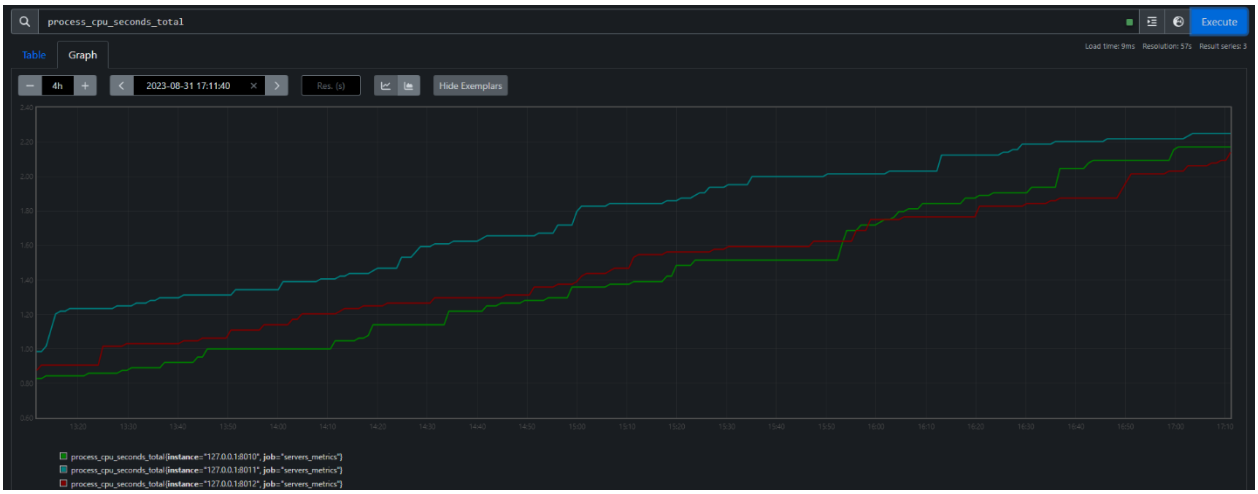
U iznad navedenoj tablici za vrijeme izvršavanja zahtjeva u *Nginx* raspoređivaču prometa možemo vidjeti da je vrijeme izvršavanja po jednom zahtjevu izuzetno malo. Možemo vidjeti da je rad *Nginx* raspoređivača prometa vrlo impresivan jer za svaki poslužitelj vrijeme izvršavanja iznosi manje od pola milisekunde. S druge strane, prilagođeni raspoređivač prometa također ima solidno vrijeme izvršavanja zahtjeva, ali u interesu je uvijek da je vrijeme izvršavanja prema zahtjevima što kraće.

6.2. Ukupno korisničko i sistemsko CPU vrijeme

Mjerenje upotrebe CPU-a također je izuzetno važno za analizu radnih karakteristika i omogućuje prilagodljivo korištenje resursa prema potrebama sustava. Ovdje ćemo obratiti pažnju na oscilacije i promjene u CPU vremenu. Putem pravilnog praćenja ovih oscilacija, administratori sustava mogu donositi ispravnije odluke o optimizaciji, skaliranju i pravilnom raspoređivanju resursa kako bi radne karakteristike sustava bile dobro održane i dobro raspoređene. Na taj način izbjegli bi izlaganje riziku preopterećenja i nepotrebnog trošenja resursa u razdobljima s izuzetno malom potrošnjom. Mjerne jedinice s kojima ćemo mjeriti vrijeme biti će sekunde.

Slika 23: Grafikon dinamike CPU-a tijekom uravnoteženja opterećenja u prilagođenom raspoređivaču prometa unutar intervala od 30 minuta

Na slici **Slika 23** možemo vidjeti vremenske skokove u vremenskom radu procesora, ovi skokovi nastaju zbog preusmjeravanje prema poslužiteljima pa se isključivo samo u tim trenucima mjeri i obilježava procesorsko vrijeme. Možemo vidjeti da je to vrijeme vrlo visoko i ne predstavlja dobre radne karakteristike s vremenskim mjerenjem CPU-a. Ovi manji skokovi također su nastali zbog zahtjevima prema poslužiteljima, ali u bržem vremenskom periodu pa je zato vrijeme samo malo veće. Isto tako, vidimo da vrijeme izvođenja ne počinje od 0, nego za prvi server počinje sa 53.42 s, drugi počinje sa 53.23 s, a treći sa 37.29 s.



Slika 24: Grafikon dinamike CPU-a tijekom uravnoteženja opterećenja u Nginx-u unutar intervala od 4 sata

Prema slici **Slika 24** možemo vidjeti vremenski porast u potrošnji vremena koji je vidljiv u vremenskom razdoblju od 4 sata. Vidi se da u početku rada za ravnotežu opterećenja nad poslužiteljima počinje s vremenskim vrijednostima koje su veće od nule, a to su: 0.83 s sekundi, 0.98 s i 0.875 s. Ovako dolazimo do zaključka da CPU počinje raditi s vremenskom potrošnjom od nekoliko stotinjaka milisekundi. Važno je za naglasiti da vremenska potrošnja CPU-a raste kroz vrijeme, čak raste i kad nema zahtjeva prema poslužiteljima. Prijetnje u vremenskom porastu CPU-a mogu uzrokovati vremenski porast, lošija optimizacija koda, loše skaliranje i slično.

Ukupno sistemsko CPU vrijeme u Nginx raspoređivaču prometa za 4 sata		
Server 1	**Server 2**	**Server 3**

2.17 s	2.25 s	2.14 s
Ukupno sistemsko CPU vrijeme u prilagođenom raspoređivaču prometa za 30 minuta		
Server 1	**Server 2**	**Server 3**
1819.49 s	1819.17 s	1803.92 s

Tablica 9: Sistemsko CPU vrijeme u Nginx i prilagođenom raspoređivaču prometa

Prema prikupljenim podacima možemo vidjeti veliku vremensku razliku između 2 različita raspoređivača prometa. Prema *Nginx*-u možemo vidjeti kvalitetnu optimizaciju te izuzetno nisko vrijeme za vremensko izvođenje 4 sata što predstavlja vrlo dobar rezultat i efikasno uravnoteživanje opterećenja. S druge strane, u prilagođenom raspoređivaču prometa, vrijeme izvođenja u samo 30 minuta je skoro 900 puta veće nego u *Nginx*-u što je jako velika razlika. Ovakve radne karakteristike mogu ukazivati na neefikasnost u uravnoteživanju opterećenja ili čak na neprimjerenu konfiguraciju sustava. U ovim radnim karakteristikama definitivno možemo reći da *Nginx* rješenje puno bolje funkcionira.

6.3. Alokacija memorijske potrošnje

Memorija je snažan i bitan resurs u ravnoteži opterećenja, a njezino korištenje odnosi se na količinu memorije koju koristi web aplikacija ili stranica. Da bi bilo jasnije, jedinice za mjerenje označavaju količinu memorije koja je alocirana (ili rezervirana) za upotrebu unutar izvođenja procesa [12]. Ukoliko nam analiza govori da je kapacitet memorije veći od očekivanog, dobra odluka bila bi dodjela dodatnih usluga ili nadogradnja maksimalnog kapaciteta memorije [9]. Uz to, detaljno praćenje korištenja memorije omogućuje bolju optimizaciju radnih karakteristika sustava. Ako se memorija koristi ispod očekivanja, dobro rješenje bilo bi korištenje dostupnih resursa za izvođenje drugih zadataka ili povećavanje broja instanci aplikacije.

Slika 25: Ponašanje alocirane memorije u prilagođenom raspoređivaču prometa

Na slici **Slika 25** možemo vidjeti kako sva 3 servera alociraju sličnu količinu memorije te također možemo primijetiti približno, ali ne i identično ponašanje. S obzirom da su ovi serveri u lokalnom okruženju i nisu puni podataka, ovo bi i trebalo biti očekivano ponašanje, ali svakako ovakva poprilično niska alokacija memorije poželjna je za korištenje u web stranicama ili web aplikacijama.

Slika 26: Ponašanje alocirane memorije u Nginx-u

Na slici **Slika 26**, možemo vidjeti potencijalne padove i skokove u memoriji. Promjena memorije događa se u trenutcima izvođenja zahtjeva prema poslužitelju ili u trenucima primanja odgovora od poslužitelja. Možemo vidjeti da je server 3 imao nagli pad u memoriji za čak 2 MB pri zahtjevu prema poslužitelju, što se smatra velikom promjenom.

Prosječna alokacija memorijske potrošnje u Nginx raspoređivaču prometa		
Server 1	**Server 2**	**Server 3**
5.71 MB	4.05 MB	3.97 MB
Prosječna alokacija memorijske potrošnje u prilagođenom raspoređivaču prometa		
Server 1	**Server 2**	**Server 3**
2.33 MB	2.11 MB	2.24 MB

Tablica 10: Alokacija memorijske potrošnje u Nginx i prilagođenom raspoređivaču prometa

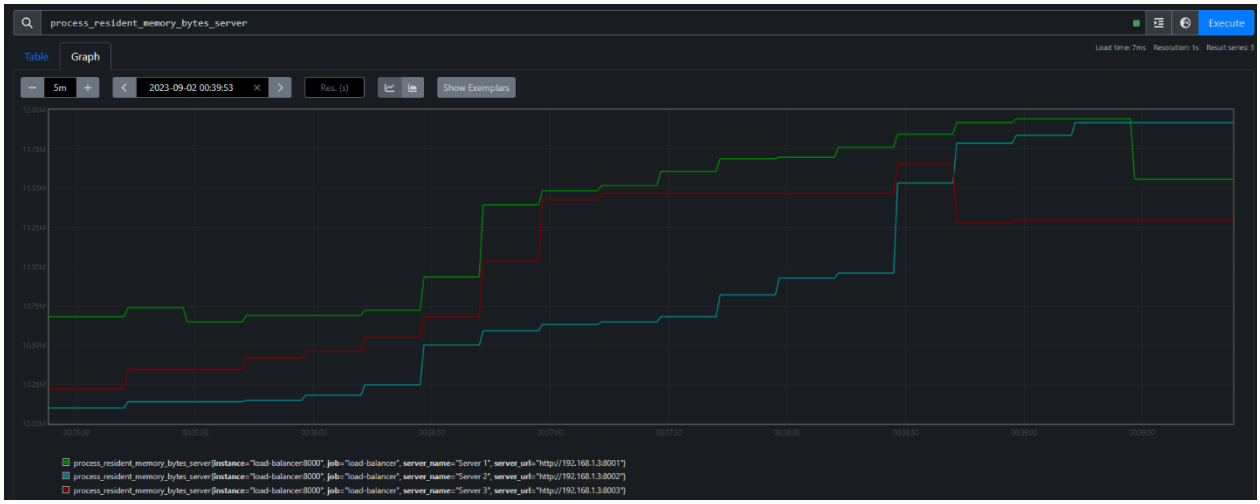
Možemo vidjeti da prilagođeni raspoređivač prometa, za razliku od *Nginx*-ovog, ima otprilike duplo manju alokaciju u memorijskoj potrošnji što je poprilično bolji rezultat zbog manjeg alokacijskog zauzimanja prostora. Na temelju određenog analiziranog vremena moglo bi se zaključiti da *Nginx* rješenje ima više skokova u memoriji za razliku od prilagođenog rješenja. U pravom prometu sa značajno većim brojem korisnika oba raspoređivača prometa imala bi puno veći izazov jer svaka nova konekcija na poslužitelju izaziva veću alokaciju.

6.4. Zauzimanje fizičke memorije

Za razliku od mjera koje se bave alociranom memorijom čiji kapacitet je predodređen za korištenje, zauzimanje fizičkog RAM-a odnosi se na stvarnu potrošnju memorije koja se nalazi na određenom poslužitelju. Mjera za zauzimanje fizičke memorije je vrlo značajna za buduće planiranje o promjeni opterećenja ili za povećanje maksimalnog kapaciteta memorije u poslužitelju. Moglo bi se pretpostaviti da je zauzimanje fizičke memorije, za razliku od alokacije memorije, značajno većeg kapaciteta. Pomoću analize korištenja ovih mjera, može se napraviti odluka o dodjeljivanju više usluga s manjom upotrebom memorije, ili s druge strane odluka o nadogradnji memorije virtualne mašine koja koristi više memorije [12].

Slika 27: Ponašanje fizičke memorije u Nginx raspoređivaču prometa

Na slici **Slika 27** vidljivo je konstantno variranje fizičke memorije, koja za razliku od alocirane memorije nije promjenjiva samo tijekom zahtjeva, već se mijenja i tijekom aktivnih poslužitelja. Također vidljivi su veći skokovi i padovi u određenim trenucima na što može utjecati puno faktora. Jedni od tih slučajeva su: velika količina zahtjeva u kratkom vremenskom periodu, memorijsko curenje (eng. memory leak), velike izmjene u aplikaciji, loše upravljanje memorijom i slično [12].



Slika 28: Ponašanje fizičke memorije u prilagođenom raspoređivaču prometa

Prosječna memorijska potrošnja u Nginx raspoređivaču prometa		
Server 1	**Server 2**	**Server 3**
19.46 MB	17.53 MB	19.43 MB

Prosječna memorijska potrošnja u prilagođenom raspoređivaču prometa		
Server 1	**Server 2**	**Server 3**
11.56 MB	11.92 MB	11.3 MB

Tablica 11: Zauzimanje fizičke memorije u Nginx i prilagođenom raspoređivaču prometa

Sad kad smo usporedili dva raspoređivača prometa možemo izvesti nekoliko zaključaka. Očigledno je da *Nginx* rješenje za razliku od prilagođenog rješenja ima visoke i niske skokove, ali s druge strane u prilagođenom rješenju možemo vidjeti skoro duplo manju memorijsku potrošnju. Naravno, poželjnije bi bilo izabrati manju memorijsku potrošnju, no i dalje ne bismo smjeli biti sigurni da će i se ovakvo ponašanje pojavljivati u zahtjevnijim situacijama gdje se troši više resursa ili se nalazi više korisnika koji podnose znatno veći broj zahtjeva. Skalabilnost također može značajno utjecati na ovu mjeru, jer ako je skalabilnost loša moglo bi doći do velikih problema u curenju memorije i do preopterećenja u poslužitelju.

7. Zaključak

U ovom diplomskom radu, spomenuta je važnost korištenja algoritama za uravnoteženje opterećenja između web usluga. S obzirom na velik broj različitih algoritama, rečeno je da je njihov odabir relevantan prema potrebama web uslugama s kojima se radi. Može se zaključiti koliko je lakše koristiti otkrivanje usluga u kontekstu ravnoteže između usluga, ali isto tako da te dvije tehnike mogu raditi jedna bez druge, samo na teži način. Također, bitno je uzet u obzir koliko su dostupnost i pouzdanost važne za kvalitetnu ravnotežu između usluga. Uz to, bitno je bilo za naglasiti koliko je bitan nadzor u prometu kako bi se uspjelo doći do poboljšanja radnih karakteristika.

Predstavljene su tehnike u uravnoteživanju, poput dinamičkog i statičkog uravnoteživanja. Svaka od tih tehnika ima svoje prednosti i mane, ali moglo bi se reći da je tehnika za statičku ravnotežu između usluga vrlo dobra u jednostavnijim aplikacijama za koje se može pretpostaviti da ne prelaze određenu granicu u trošenju resursa. A s druge strane, rekli smo da je tehnika za dinamičku ravnotežu povoljnija za složenije aplikacije koje očekuju ogroman broj korisnika i predvidivo je da će za vrijeme aplikacije biti događaja koji zahtijevaju promjenjivo održavanje. Neki važniji algoritmi za držanje ravnoteže među uslugama spomenuti su te su navedene njihove prednosti i nedostaci. Preko nekih ilustracija spomenute su njihove ključne značajke i uz sve to, prikazane su njihove funkcionalnosti. Naglašeno je što su vertikalna i horizontalna skalabilnost. Poučno je kako horizontalna skalabilnost skalira prema više poslužiteljima na način dodjeljivanja manjeg kapaciteta resursa, a s druge strane vertikalna skalabilnost skalira prema manjem broju poslužiteljima ali omogućava im veći broj resursa i sprječava ih od lakšeg preopterećenja.

Prikazana je detaljna implementacija raspoređivača prometa i navedene su korištene tehnologije. Isto tako objasnilo se zašto je dobro koristiti *Golang* u uravnoteživanju opterećenja te je spomenuto kako taj jezik utječe na radne karakteristike sustava, uključujući iskorištenost memorije, latenciju i pouzdanost.

Zatim je slijedio uvid u mjere kako bi se pojasnilo koje su ključne i najkorisnije za praćenje. Ispostavlja se da je važno nadzirati broj veza između svih poslužiteljima, koliko

vremena je klasično za obradu zahtjeva i koji resursi su najvažniji za praćenje. To je sve potkrijepljeno primjerima i kako zapravo to izgleda u testiranom okruženju.

Za kraj smo ostavili usporedbu radnih karakteristika ručno napravljenog raspoređivača prometa i postojećeg *Nginx* raspoređivača prometa. Jasno je zaključeno kako je prilagođeni raspoređivač prometa štedljiviji u nekim radnim karakteristikama kao što su memorijska potrošnja, ali se zato *Nginx* raspoređivač prometa iskazao boljim u propusnosti te u ukupnom CPU vremenu što obuhvaća i korisnika i sustav. To se sve postiglo preko alata za mjerenje kao što su *Prometheus* (u većini slučajeva) i *Zipkin* te smo pomoću uspoređivačkih tablica i slikovnih primjera za obje aplikacije uspjeli prikazati ključnu razliku.

Moglo bi se reći kako je važno imati na umu da aplikacija sa uravnoteživanjem opterećenja između usluga nikad ne može biti savršena, ali kroz detaljnu analizu radnih karakteristika uvijek je poželjno težiti prema boljemu i izbjegavati smetnje kao što su uska grla.

Literatura

[1] Avi Networks, »Application Traffic Management Definition« Dostupno na: https://avinetworks.com/glossary/application-traffic-management/. [Mrežno; pristupljeno 2023.].

[2] M. Raza, »Reliability vs Availability: What's The Difference?« 13. svibanj 2020. Dostupno na: https://www.bmc.com/blogs/reliability-vs-availability/. [Mrežno; pristupljeno 2023.].

[3] Nginx, »What Is Load Balancing?« Dostupno na: https://www.nginx.com/resources/glossary/load-balancing/. [Mrežno; pristupljeno 2023.].

[4] A10, »How do Layer 4 Load Balancing and Layer 7 Load Balancing Differ?« Dostupno na: https://www.a10networks.com/glossary/how-do-layer-4-and-layer-7-load-balancing-differ/. [Mrežno; pristupljeno 2023.].

[5] Avi Networks, »Round Robin Load Balancing « Dostupno na: https://avinetworks.com/glossary/round-robin-load-balancing/. [Mrežno; pristupljeno 2023.].

[6] EnjoyAlgorithms, »Different types of Load Balancing Algorithms« Dostupno na: https://www.enjoyalgorithms.com/blog/types-of-load-balancing-algorithms. [Mrežno; pristupljeno 2023.].

[7] A. Arshad, »What Is the Least Connections Load Balancing Technique?« 2023. Dostupno na: https://www.educative.io/answers/what-is-the-least-connections-load-balancing-technique. [Mrežno; pristupljeno 2023.].

[8] Progress Kemp, »Load Balancing Algorithms and Techniques« Dostupno na: https://kemptechnologies.com/load-balancer/load-balancing-algorithms-techniques. [Mrežno; pristupljeno 2023.].

[9] P. Shibu, » 11 Load Balancing Metrics You Need to Measure and Why It Matters« 21. ožujak 2022. Dostupno na: https://arraynetworks.com/load-balancer-important-metrics-impacting-performance/. [Mrežno; pristupljeno 2023.].

[10]Progress Kemp, »Load Balancing Layer 4 and Layer 7« Dostupno na: https://freeloadbalancer.com/load-balancing-layer-4-and-layer-7. [Mrežno; pristupljeno 2023.].

[11]Exponent, » How to Cover Load Balancing in a System Design Interview« Dostupno na: https://www.tryexponent.com/courses/fundamentals-system-design/load-balancers. [Mrežno; pristupljeno 2023.].

[12]AlgoDaily, »What are metrics and why do they matter?« Dostupno na: https://algodaily.com/lessons/application-performance-metrics. [Mrežno; pristupljeno 2023.].

[13]Avi Networks, »Service Discovery« Dostupno na: https://avinetworks.com/glossary/service-discovery/. [Mrežno; pristupljeno 2023.].

[14]nOps, »Horizontal vs Vertical scaling: An in-depth Guide« 6. prosinac 2022. Dostupno na: https://www.nops.io/blog/horizontal-vs-vertical-scaling/. [Mrežno; pristupljeno 2023.].

[15]C. Slingerland, »Horizontal vs Vertical scaling: An in-depth Guide« 5. svibanj 2023. Dostupno na: https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling. [Mrežno; pristupljeno 2023.].

[16]S. J. Bigelow, »What are the types of APIs and their differences?« 10. siječanj 2023. Dostupno na: https://www.techtarget.com/searchapparchitecture/tip/What-are-the-types-of-APIs-and-their-differences. [Mrežno; pristupljeno 2023.].

[17]Datadog, »Auto-scaling Overview« Dostupno na: https://www.datadoghq.com/knowledge-center/auto-scaling/. [Mrežno; pristupljeno 2023.].

[18]S. Rahle, » Server performance issues: Detection and causes« 9. svibanj 2016. Dostupno na: https://www.syskit.com/blog/server-performance-issues-detection-causes/. [Mrežno; pristupljeno 2023.].

[19]Infraspeak Team, » Server performance issues: Detection and causes« 1. siječanj 2023. Dostupno na: https://blog.infraspeak.com/mtbf-mean-time-between-failures/. [Mrežno; pristupljeno 2023.].

[20]Avi Networks, »Round Robin Load Balancing« Dostupno na: https://avinetworks.com/glossary/static-load-balancing/. [Mrežno; pristupljeno 2023.].

[21]B. Hendrickson, K. Devine, »Dynamic load balancing in computational mechanics« 23. ožujak 2000. Dostupno na: https://www.sciencedirect.com/science/article/abs/pii/S0045782599002418. [Mrežno; pristupljeno 2023.].

[22]R. Hudson, »Go GC: Prioritizing low latency and simplicity« 31. kolovoz 2015. Dostupno na: https://go.dev/blog/go15gc. [Mrežno; pristupljeno 2023.].

[23]Educative, »What is a goroutine?« Dostupno na: https://www.educative.io/answers/what-is-a-goroutine. [Mrežno; pristupljeno 2023.].

[24]K. B. Abdedeji, A. M. Abu-Mahfouz, Anish M. Kurien, »DDoS Attack and Detection Methods in Internet-Enabled Networks: Concept, Research Perspectives, and Challenges« 6. srpanj 2023. Dostupno na: https://www.mdpi.com/2224-2708/12/4/51. [Mrežno; pristupljeno 2023.].

Popis slika

Slika 1: Otkrivanje usluge u servisnom registru 12
Slika 2: Uravnoteživanje opterećenja 17
Slika 3: Transportni i aplikacijski sloj u uravnoteženju opterećenja [10] 18
Slika 4: Pseudokod algoritma za kružno usmjeravanje 22
Slika 5: Ilustracija kružnog usmjeravanja 23
Slika 6: Pseudokod algoritma za težinsko kružno usmjeravanje 24
Slika 7: Ilustracija kružnog usmjeravanja s težinama 24
Slika 8: Pseudokod algoritma za IP hash 25
Slika 9: Ilustracija IP hash algoritma 26
Slika 10: Pseudokod algoritma za URL hash 26
Slika 11: Pseudokod algoritma nasumičnosti 27
Slika 12: Pseudokod algoritma za spajanje prema poslužitelju s najmanje konekcija 27
Slika 13: Ilustracija algoritma s najmanjim brojem aktivnih veza 28
Slika 14: Prikaz vertikalne skalabilnosti [15] 29
Slika 15: Prikaz horizontalne skalabilnosti [15] 31
Slika 16: Prikaz nekih ključnih metrika 35
Slika 17: Grafikon aktivnih veza 36
Slika 18: Broj zahtjeva prema poslužiteljima 36
Slika 19: Ukupan broj zahtjeva prema poslužiteljima 37
Slika 20: Grafikon vremena obrade svih zahtjeva na pojedinom serveru 38
Slika 21: Grafikon ukupnog izvršavanja zahtjeva prema poslužiteljima tijekom uravnoteženja opterećenja u Nginx-u 43
Slika 22: vremenski prikaz izvršavanja zahtjeva prema poslužiteljima u prilagođenom raspoređivaču prometa 43
Slika 23: Grafikon dinamike CPU-a tijekom uravnoteženja opterećenja u prilagođenom raspoređivaču prometa unutar intervala od 30 minuta 44
Slika 24: Grafikon dinamike CPU-a tijekom uravnoteženja opterećenja u Nginx-u unutar intervala od 4 sata 45
Slika 25: Ponašanje alocirane memorije u prilagođenom raspoređivaču prometa 47
Slika 26: Ponašanje alocirane memorije u Nginx-u 47
Slika 27: Ponašanje fizičke memorije u Nginx raspoređivaču prometa 49
Slika 28: Ponašanje fizičke memorije u prilagođenom raspoređivaču prometa 49

Popis tablica

Tablica 1: Komponente za otkrivanje usluga 11
Tablica 2: Prednosti i nedostaci automatskog skaliranja resursa 13
Tablica 3: Pripadajuće vrijeme izvan usluge za određenu razinu dostupnosti web usluge 14
Tablica 4: Srednje vrijeme između kvarova jednoj godini 15
Tablica 5: Vrste algoritama za uravnoteživanje opterećenja..... 21
Tablica 6: Razlozi korištenja mjernih vrijednosti za procjenu opterećenja 34
Tablica 7: Rezultati zahtjeva prema HTTP statusni kodovima..... 38
Tablica 8: Prosječno vrijeme izvršavanja zahtjeva u Nginx i prilagođenom LB-u 44
Tablica 9: Sistemsko CPU vrijeme u Nginx i prilagođenom LB-u 46
Tablica 10: Alokacija memorijske potrošnje u Nginx i prilagođenom LB-u..... 48
Tablica 11: Zauzimanje fizičke memorije u Nginx i prilagođenom LB-u..... 50

Popis kodova

Isječak koda 1: Horizontalno skaliranje u docker-compose konfiguraciji 32
Isječak koda 2: Konfiguracija Docker kompozicije za raspoređivač prometa 40
Isječak koda 3: Struktura i konfiguracija raspoređivača prometa 41