Balansiranje optećerenja u web poslužiteljima

Krušec, Josip

Master's thesis / Diplomski rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:052373

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

BALANSIRANJE OPTEREĆENJA U WEB POSLUŽITELJIMA

Diplomski rad

Josip Krušec

Osijek, 2016.

Sveučilište Josipa Jurja Strossmayera u Osijeku
ETFOS
ELEKTROTEHNIČKI FAKULTET OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek,

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Josip Krušec
Studij, smjer:	Diplomski studij, Procesno računarstvo
Mat. br. studenta, godina upisa:	D-723 R
Mentor:	doc. dr. sc. Irena Galić
Sumentor:	mag. comp. Hrvoje Leventić
Predsjednik Povjerenstva:	
Član Povjerenstva:	
Naslov diplomskog rada:	Balansiranje opterećenja u web poslužiteljima
Primarna znanstvena grana rada:	
Sekundarna znanstvena grana (ili polje) rada:	
Zadatak diplomskog rada:	
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: Postignuti rezultati u odnosu na složenost zadatka: Jasnoća pismenog izražavanja: Razina samostalnosti:

Potpis sumentora:	Potpis mentora:

Dostaviti:

1. Studentska služba

U Osijeku, godine Potpis predsjednika Odbora:

Sveučilište Josipa Jurja Strossmayera u Osijeku

ETFOS ELEKTROTEHNIČKI FAKULTET OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek,

Ime i prezime studenta:	Josip Krušec
Studij :	Diplomski studij, Procesno računarstvo
Mat. br. studenta, godina upisa:	D-723 R

Ovom izjavom izjavljujem da je rad pod nazivom**: Balansiranje opterećenja u web poslužiteljima**

izrađen pod vodstvom mentora

doc. dr. sc. Irene Galić

i sumentora

mag. comp. Hrvoja Leventića

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD....1
2. KORIŠTENE TEHNOLOGIJE....2
2.1. *Linux Ubuntu*....2
2.1.1. Povijest....2
2.1.2. Karakteristike....3
2.1.3. *Ubuntu Server*....4
2.2. *Ruby*....4
2.2.1. Povijest....5
2.2.2. Karakteristike....5
2.3. *Rails*....6
2.3.1. Povijest....7
2.3.2. *Rails* programerska načela....8
2.2.3. Karakteristike....9
2.4. *NGINX*....10
2.4.1. Karakteristike....10
2.4.2. Konfiguriranje *NGINXA*....11
2.5. *Passenger*....*13*
3. BALANSIRANJE OPTEREĆENJA....16
3.1. Balansiranje opterećenja....16
3.1.1. *HTTP* zahtjevi....16
3.1.2. Princip rada balansiranja opterećenja....17
3.2. *NGINX* kao balanser opterećenja....18
3.3. *HAProxy*....20
4. KONFIGURACIJA BALANSIRANJA OPTEREĆENJA I REZULTATI....22
4.1. Inicijalne postavke poslužitelja....22
4.1.1. Osiguravanje poslužiteljskih resursa....22
4.1.2. Dodavanje novog korisnika, privilegija i *SSH* zaštite....23
4.2. Instaliranje programske podrške....25
4.2.1. Instaliranje *Ruby*-a....25
4.2.2. Instaliranje *NGINXA* i *Passengera*....26

4.2.3. Postavljanje web poslužitelja i postavljanje aplikacije na poslužitelj bez balansiranja opterećenja....27
4.2.4. Postavljanje web poslužitelja i postavljanje aplikacije na poslužitelj sa balansiranjem opterećenja....29
4.3. Mjerenje opterećenja s i bez balansiranja opterećenja....30
4.3.1. *Apache Benchmark* program za mjerenje opterećenja....30
4.3.2. Rezultati mjerenja....31
4.3.3. Komentar na dobivene rezultate....32
5. ZAKLJUČAK....34
LITERATURA....35
SAŽETAK....39
ABSTRACT....39
ŽIVOTOPIS....40
PRILOZI....41
P.1. Detaljniji pregled rezultata mjerenja bez balansiranja opterećenja....41
P.2. Detaljniji pregled rezultata mjerenja s balansiranjem opterećenja....46

1. UVOD

U današnje vrijeme svjedoci smo sve veće ekspanzije veličine i mogućnosti globalne mreže internet. Tu činjenicu potvrđuje podatak da određene web stranice imaju po stotine tisuća, pa čak i milijun istovremenih zahtjeva na web poslužitelje prema literaturi [1]. Za poslužitelje navedeni zahtjevi predstavljaju radno opterećenje tj. teret. Da bi bilo moguće odgovoriti velikom broju zahtjeva u što kraćem roku potreban je sustav koji čine računala, nakupine računala ili grozdovi računala. Kod navedenih sustava potrebno je što ravnomjernije opteretiti dane resurse kako bi se oni koristili što bliže optimalnom. Slična se problematika pojavljuje kada je potrebno skalirati aplikaciju od aplikacije sa malo zahtjeva i malo resursa do aplikacije sa puno zahtjeva i puno resursa. Kako bi se navedi problem riješio uvodi se balansiranje opterećenja (*engl. Load balancing*), što je ujedno i tema ovog diplomskog rada.

Cilj ovog rada je mjeriti odziv sustava od jednog poslužitelja te ga usporediti sa odzivom većeg sustava koji se sastoji od 2 poslužitelja i koristi balansiranje opterećenja.

Da bi bilo moguće mjeriti opterećenje potrebno je osigurati poslužiteljske resurse, instalirati potrebne programe kako bi se resursi mogli postaviti za posluživanje web aplikacije pisane u *Ruby on Rails* programskom jeziku. Nakon toga potrebno je izvršiti mjerenja opterećenja pomoću A*pache Benchmark* programa za testiranje opterećenja, usporediti i komentirati rezultate.

Rad je pisan u četiri poglavlja. U drugom poglavlju su opisane tehnologije koje je potrebno primijeniti kako bi aplikacija bila dostupna klijetima putem interneta. U trećem poglavlju se govori općenito o balansiranju opterećenja, na koji način ono radi i kako ga realizirati u konkretnim slučajevima. Četvrto poglavlje sadrži mjerenja odziva konfiguracije poslužitelja s korištenjem balansiranja opterećenja i bez balansiranja opterećenja. Također, četvrto poglavlje sadrži i komentare i objašnjenja rezultata mjerenja.

2. KORIŠTENE TEHNOLOGIJE

2.1. *Linux Ubuntu*

Ubuntu je distribucija operativnog sustava *Linux* sličnih *Unixu* koji se koristi kao platforma na stolnim računalima, prijenosnim računalima, internet poslužiteljima, tabletima i televizorima visoke rezolucije. Nastao je iz *Debian GNU/Linuxa* i pripada skupini operativnih sustava otvorenog koda kako stoji u [2]. Ubuntu za osobna računala standardno sadrži grafičko korisničko sučelje, što nije slučaj u svim *Linux* distribucijama. *Ubuntu* projekt vodi tvrtka Canonical Ltd., ali zbog činjenice da je operativni sustav otvorenog koda, u njegovom razvoju sudjeluju mnogi programeri koji čine zajednicu *Ubuntu* (*engl. Ubuntu community*). *Ubuntu* je naziv afričkog filozofskog koncepta koji znači humanost prema drugima. Pošto je *Ubuntu* operativi sustav otvorenog koda kao i aplikacije koje se na njemu koriste, proizlazi da je besplatan, svjetski rasprostranjen i uživa veliku podršku altruistične *Ubuntu* zajednice.

2.1.1. Povijest

Prva inačica *Ubuntua* objavljenja je 20. listopada 2004. i nastala je na *Debian* arhitekturi i infrastrukturi prema [3]. *Ubutnu* za instalaciju programa koristi pakete koji se nalaze na udaljenim poslužiteljima i korisnik jednostavnim pozivom željenog paketa preko naredbenog retka instalira željenu aplikaciju. Prilikom instalacije aplikacija koriste se sustavi za upravljanje paketima *APT* (*engl. Advanced Packaging Tool*), preuzet iz *Debiana* te *Ubuntu* centar za programsku podršku. *Ubuntu* je ostao vjeran filozofiji otvorenog koda pa sadrži uglavnom programsku podršku te vrste, za razliku od nekih izvedenica *Debiana*.

Ubuntu projekt financira firma Canonical Ltd., koja je ujedno osnovala i *Ubuntu* zakladu te dala početni kapital od 10 milijuna dolara. Cilj Canonical Ltd.-a nije financijska korist, već osigurati financijsku podršku za razvoj *Ubuntu* projekta prema [4].

Nova inačica *Ubuntua* izlazi svakih šest mjeseci i nosi broj godina umanjena za 2000, zatim ide točka te mjesec izdanja. Osim broja inačica sadrži tzv. kodno ime

koje se sastoji od 2 riječi sa istim početnim slovom, a ime svake nove inačice je slijedeće slovo engleske abecede.

Postoje dvije vrste distribucija koje se izdaju, a to su normalna i *LTS* (*engl. Long Term Support*). Za razliku od normalne, *LTS* verzija izlazi svake dvije godine. Razlika je u tome što *LTS* verzije imaju dulju podršku od normalnih. Podrška podrazumijeva izdavanje zakrpa i popravljanje određenih *bugova* prijašnje verzije, dostupnost komercijalnih ugovora tvrtke Canonical te podršku *Landscapea*, koji podrazumijeva set alata za održavanje poslužitelja.

Osim distribucija koje se numeriraju po kronološkom redoslijedu nastanka, svaka distribucija dolazi u dvije verzije ovisno o uređaju na kojem se *Ubuntu* planira koristiti. Postoje verzije *Ubuntu Desktop* za stolna i prijenosna računala, *Ubuntu Server* koji se koristi kao operacijski sustav na udaljenim poslužiteljima, *Ubuntu Phone* za pametne telefone, *Ubuntu Tablet* za tablete te *Ubuntu IoT* namijenjen za korištenje na autonomnim strojevima i drugim digitalnim uređajima koje se mogu povezati na internet.

2.1.2. Karakteristike

Ubuntu, za razliku od velikog dijela starijih *Linux* distribucija, pruža relativno jednostavno, sigurno i stabilno korisničko sučelje za stolna računala, web poslužitelje, prijenosna računala, tablete, pametne telefone i televizore visoke rezolucije.

Instalacija se vrši u samo par koraka. Potrebno je skinuti .iso sliku sa službene stranice *Ubuntua*. Nakon toga potrebno je napraviti *bootabilni* CD/DVD/USB pomoću *UnetBotin* ili sličnog programa za pravljenje *live* CD/DVD/USB sustava. Nakon toga potrebno je ponovo pokrenuti sustav i u *BIOS*-u (enlg. *Basic Input Output System*) postaviti sa kojeg se medija želi pokrenuti instalaciju. Osim instalacije *live* CD/DVD/USB-a nudi mogućnost pokretanja *Ubuntua* bez instalacije kako bi se sustav isprobao ili provjerila kompatibilnost sa sklopovljem.

Za razliku od *Ubuntua*, velik dio *Linux* distribucija dolazi sa minimalnim vizualnim korisničkim sučeljem ili bez njega, pa je korisnik osuđen na učenje naredbi komandne linije, što je zamoran i dugotrajan proces za prosječnog korisnika. *Unity* omogućava jednostavno grafičko korisničko sučelje koje daje aplikacijama više prostora na radnoj površini. Također, jednostavnosti pridonosi i to da su aplikacijski paketi spremljeni u repozitorijima na udaljenim poslužiteljima, kojima se pristupa

jednom linijom u naredbenom retku. Na taj način se izbjegava traženje po internetu koje ponekada zna biti dugotrajno.

Važno je naglasiti i sigurnost *Ubuntu* operacijskog sustava. Sigurnost je riješena sa više mehanizama. Glavni mehanizam je izvođenje korisničkih programa sa niskim privilegijama. Kako bi se ostvarile više privilegije potrebno je unijeti lozinku super korisnika tj. *roota*, što onemogućava programima mijenjanje bitnih postavki. Većina mrežnih portova su standardno zatvoreni kako bi se izbjeglo neželjeno povezivanja od neke treće strane. Osim navedenog, sigurnosti pridonose i ugrađeni vatrozid i zaštita od virusa, a *LTS* inačice daju sigurnosne zakrpe i nadogradnje kroz 5 godina.

2.1.3. *Ubuntu Server*

Ubuntu Server je verzija *Linux Ubuntua* namijenjena za korištenje kao operacijski sustav na udaljenim poslužiteljima. Do verzije 14.04. razlika između *Ubuntu Desktopa* i *Ubuntu Servera*, osim u instalacijskim paketima koji dolaze sa operativnim sustavom, bila je i u jezgri operacijskog sustava (*engl. Kernel*). Nakon verzije 14.04. razlika je samo u paketima koji dolaze sa operacijskim sustavom. Standardno *Ubuntu Server* nema grafičko korisničko sučelje, ali sadrži mnoge pakete kao što su: *OpenStac*k – platforma otvorenog koda namijenjena za računarstvo u oblacima, *Puppet* – sustav za upravljanje konfiguracijama otvorenog koda, *Xen* – sustav za motrenje virtualnog stroja otvorenog koda, *Ceph* – pruža platformu za spremanje objekata na jednom distribuiranom grozdu računala i mnogo mnogo drugih. Osim paketa za upravljanje i konfiguraciju infrastrukture *Ubuntu Server* sadrži mnoge poslužitelje specifične namjene kao što su: *MySQL* – poslužitelj za upravljanje, stvaranje i dohvaćanje podataka sa baze podataka, *Apach*e – web poslužitelj, točnije, poslužitelj koji rukuje *HTTP* zahtjevima, *NGINX – HTTP* poslužitelj i obrnuti *proxy* poslužitelj, kao i *IMAP/POP3 proxy* poslužitelj i još puno drugih.

2.2. *Ruby*

Ruby je objektno-orijentiran skriptni programski jezik opće namjene. Prva verzija *Ruby* 0.95 objavljena je u Japanskim *news* grupama 21. prosinca 1995, kako stoji u litaraturi [5]. *Ruby* programski jezik je dizajnirao i razvio Yukihiro

Matsumoto, pod utjecajem *Peara*, *SmallTalka*, *Eifelaa*, *Adae* i *Lispa*. Matsumoto je prilikom razvoja *Rubya* pokušao u njega utjeloviti, po njemu, dobre karakteristike navedenih jezika i dodati nove karakteristike koje je smatrao zanimljivim i korisnim. Ideje koje stoje iza *Rubya*, po riječima autora, su: "Pokušavam napraviti *Ruby* prirodnim, ne jednostavnim. Na takav način da reflektira život.", "*Ruby* je jednostavan u izgledu, ali kompleksan iznutra, kao ljudsko tijelo.", "Zapravo, trebamo se fokusirati na ljude, na to kako ljudi žele programirati aplikaciju za stroj. Mi smo gospodari, oni su sluge.", više informacija na [6].

2.2.1. Povijest

Nakon izdanja prve verzije *Rubya* 1995. godine, uslijedila su nova izdanja 25. prosinca 1996. verzija *Ruby* 1.0, prosinac 1998. verzija *Ruby* 1.2, kolovoz 1999. verzija *Ruby* 1.4, rujan 2000. verzija *Ruby* 1.6, kolovoz 2003. verzija *Ruby* 1.8, prosinac 2007. verzija *Ruby* 1.9, veljača 2013. verzija *Ruby* 2.0, 25. prosinca 2013. verzija *Ruby* 2.1, 25. prosinca 2014. verzija *Ruby* 2.2, 25. prosinca 2015. verzija *Ruby* 2.3, koja je ujedno i najnovija.

Prva verzija sadržavala je objektno-orijentirani dizajn, klase sa nasljeđivanjem, rukovanje iznimkama, gospodarenje otpadom, *mixine*, iteratore i djelokruge u smislu dostupnosti varijable s obzirom na mjesto pozivanja.

U svaku slijedeću verziju implementirane su zakrpe prijašnjih *bugova*, sigurnosne zakrpe kao i neke nove karakteristike i funkcionalnosti.

Do 1999. godine *Ruby* je bio rasprostranjen isključivo unutar Japana i *Ruby* zajednica postojala je jedino na Japanskim *mailing* listama. Nakon 1999. diskusije o *Rubyu* postaju dostupne i na *mailing* listama na Engleskom jeziku u obliku *Ruby-talksa*. U 2015. godini *Ruby* je u top 12 tehnologija koje se koriste u programiranju zauzeo 11. mjesto, prema istraživanju [7].

2.2.2. Karakteristike

Ruby je izrazito objektno-orijentiran jezik. Izrazito zbog toga što su i određeni primitivni tipovi podataka poput brojeva ili logičkih tipova objekti, dok u drugim objektno-orijentiranim jezicima to nije slučaj. *Ruby* je besplatan i može se besplatno skinuti, koristiti i modificirati.

Važna karakteristika *Rubya* je njegova fleksibilnost. Dopušta programeru da mijenja kod kako njemu to odgovara. Npr. ako želite umjesto operatora za zbrajanje + koristiti riječ plus, potrebno je u klasi *Numeric* definirati metodu plus i koje parametre prima, uz primjer iz [6].

Ruby podržava *mixine*. *Mixin* podrazumijeva poseban slučaj višestrukog nasljeđivanja, odnosno dodavanje nekoj klasi određen broj ili set dodatnih metoda, koje su naslijeđene iz neke druge klase.

Postoje 4 vrste djelokruga varijable, a to su: globalni, klasni, objektna(instancni) i lokalni. Razlikuju se po znakovima ispred njih ili ako nema nikakvog znaka. Uzmimo za primjer varijablu i. Kada bi napisali i, to znači lokalna varijabla i, @i bi bila objektna varijabla i, $i bi bila globalna varijabla, a @@i bi bila klasna varijabla. U ostalim programskim jezicima postoje najčešće 3 djelokruga varijabli, a to su *public*, *static* i *private*.

U *Ruby*-u je dopuštena literalna notacija prilikom stvaranja nizova, *hasheva*, regularnih izraza i simbola. Ostali programski jezici najčešće zahtijevaju stvaranje instanci navedenih tipova podataka pomoću konstruktora *new*.

Ruby sadrži mehanizam za gospodarenje otpadom. Kada je napravljen novi objekt, on zauzima određeni alocirani prostor u memoriji. Ako se taj objekt iskoristi i u daljnjem kodu više nije potreban, zauzeti prostor ostaje zauzet. U nekim jezicima poput *C++*-a taj problem se mora rješavati ručno i to destruktorima. Kod *Rubya* je taj problem riješen automatski, mehanizmom za gospodarenje otpadom.

Jedna od glavnih prednosti *Rubya* u odnosu na ostale programske jezike je mnoštvo biblioteka i dodatnih modula različite namjene poput: *YAML*, *JASON, XML, CGI, OpenSSL, HTTP, FTP, RSS, curses, zlib* i *Tk*. Osim navedenih bitni su i *Ruby Gem* paketi koji su koncentrirani na jednom mjestu na internetu https://rubygems.org/ , odakle ih je vrlo jednostavno skinuti na računalo i implementirati u projekt.

Iz razloga što je potpuno besplatan, *Ruby*, također, uživa veliku podršku brojne zajednice.

2.3. *Rails*

Ruby on Rails ili skraćeno *Rails* je biblioteka, točnije *Ruby Gem*, koja proširuje *Ruby* i pruža programski okvir za izradu web aplikacija. *Rails* sa svojim *API*-ima (*engl. Application Programming Interface*) postavlja konvencije za lakše stvaranje, održavanje i suradnju prilikom razvoja kompleksnih web aplikacija. *Rails* kombinira *HTML, JavaScript* i *CSS* kao korisničko sučelje (*engl. Front End*) i *Ruby* kao pozadinski dio, skriven od korisnika (*engl. Back end*). U širem smislu *Rails* označava zajednicu *Ruby* entuzijasta koji izmjenjuju svoja iskustva putem mreža. Karakteristično za programsku podršku otvorenog koda, *Rails* je zajednica u kojoj se lako može doći do rješenja specifičnih problema koje pružaju iskusniji programeri. Prednost *Railsa* svakako čine *Gemovi*, tj. programski paketi napisani za rješavanja specifičnih problema. *Rails* zajednica funkcionira na način da netko tko se susreo sa određenim problemom i pojednostavnio ga objavi svoje rješenje u obliku *Gema* na https://rubygems.org/. Nakon što je objavljen *Gem* je dostupan svima i na taj način profitira cijela zajednica, više informacija u [8].

2.3.1. Povijest

Rails je napisao David Heinemeier Hansson u *Rubyu* kao dio projekta web aplikacije *Base Camp* istoimene firme. Prvi puta je izdan u lipnju 2004. godine kao projekt otvorenog koda. Tek u veljači 2005. godine Hansson je omogućio entuzijastima mijenjanje i nadogradnju njegovog prvotnog koda.

Verzija *Railsa* 2.3 izašla je 15. ožujka 2009. sa razvijen novim karakteristikama kao što su predlošci (*engl. Template*), *enginei*, *Rack* i ugniježđene forme modela. Predlošci omogućuju programeru generiranje kostura aplikacije sa određenim *Gemovima* i postavkama. *Engieni* omogućuju ponovnu upotrebu određenih dijelova aplikacije. *Rack* web poslužiteljsko sučelje omogućava optimiziranje koda koji se usmjerava preko *Action Controllera*, prema [9].

23. prosinca 2008. pokrenut je *Merb* projekt koji bi treba biti novi programski okvir za *Ruby*. Rezultat je bio da su se *Rails* i *Merb* spojili u jedan projekt u *Rails* 3.0 verziji, a više informacija je dostupno na [10].

31. kolovoza 2011. izašla je verzija *Rails* 3.1. koja je imala nove karakteristike kao što su migriranje baza podataka, *asset* cjevovodi, *jQuerry* kao standardnu biblioteku za *Javascript* te nove tehnologije *CoffeeScript* i *Sass*.

Svaka slijedeće godine izlazila je nova verzija *Railsa*. 2012. verzija 3.2. sa *Automatic Querry Explin, Tagged Login* kao i bržim opcijama razvijanja i boljim *engineima* za usmjeravanje, prema [11]. 2013. godine izašla je verzija *Rails* 4.0 sa karakteristikama : *Russan Doll Caching, Turbolinks, Live Streaming, Active Resource* i *Active Record Observer*, više karakteristika na [12]. U 2014. izašle su 2 verzije i to u travnju 4.1. i u prosincu 4.2. U verziji 4.1. novine su bile: *Spring, Variants, Enmus, Mailer previews,* prema literaturi [13]. Dok u 4.2. koja je trenutno najnovija novine su: *Active Job*, asinkrona elektronička pošta, *Adequate Record*, web konzola i strani ključevi, više informacija na [14].

2.3.2. *Rails* programerska načela

Rails je samouvjeren(*engl. Rails Is Opinionated*) je načelo u kojem postoji najbolji način rješavanja nekog problema(*engl. Rails way*). Slijeđenje dogovorenih konvencija rezultira sa donošenjem manje odluka te da se dobar dio kodiranja automatizira. Također, rezultat je i lakša suradnja, održavanja i razvoj aplikacija.

Rails je *omakase* (*engl. Rails Is Omakase*) načelo je u kojem se način rješavanja određenog problema tako da se u obzir uzimaju rješenja iskusnijih programera. U japanskom jeziku se *omakase* odnosi na situaciju u kojoj gost u restoranu umjesto da sam odluči što će jesti sa jelovnika, odluku prepusti šefu kuhinje. Forumi preko kojih komunicira *Rails* zajednica vrve različitim mišljenjima iskusnih programera koji razvijaju *Rails*. Najbolje je slijediti njihova mišljenja.

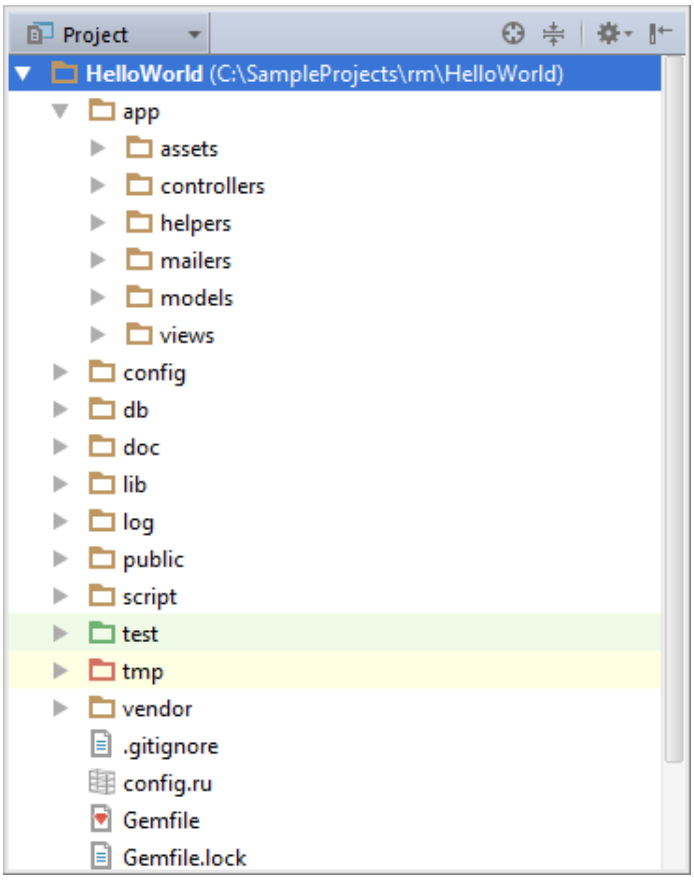
Konvencija iznad konfiguracije (*engl. Convention Over Configuration*) je načelo koje je primjer *Rails* je samouvjeren načela. U različitim programskim jezicima često su potrebne mnoge konfiguracijske datoteke. *Rails* radi pretpostavke. Ako programer napravi model objekta korisnik i nazove ga *User*, *Rails* će pretpostaviti da je potrebna baza podataka u koju će se spremati korisnici i automatski, bez potrebne konfiguracije, podatke će spremati u tablicu *users.* Smisao ovog načela je praćenje konvencija, kako bi *Rails* mogao učiniti neke stvari za programera, a samim time se postiže i pregledniji kod i lakša suradnja.

Nemoj se ponavljati (*engl. Don't Repeat Yourself*) načelo koje podrazumijeva izbacivanje pisanja dupliciranog koda kako bi sam kod bio čitljiviji i manje sklon *bugovima*. *Rails* iskorištava *Rubyevo* svojstvo meta programiranja, pa se na taj način može još više eliminirati kod koji nije potreban.

Programiranje pogonjeno testiranjem (*engl. Test Driven Development*) je načelo u kojem se pišu testni kodovi prije ili za vrijeme dok se piše aplikacija, a ne nakon što je pisanje završeno. Uz ovo načelo se jako dobro uklapa i *DRY* načelo zbog toga što za mnoge situacije *Rails* automatski generira testne slučajeve, pa se tako piše minimalna količina koda.

2.2.3. Karakteristike

Rails koristi *MVC* (*engl. Models-Views-Controllers*) koncept. *MVC* je visoka razina apstrakcije u kojoj je za korisničko sučelje odgovoran *Views*, za podatkovne strukture *Models*, a za komunikaciju između njih i pozadinsku logiku *Controllers*.



Slika 2.1. Izgled datoteka Rails projekta [15]

U mapi app nalaze se mape *assets* u kojoj su spremljene *CSS* i *Javascript* datoteke te slike. U mapi app/controllers *Rails* nalazi kontrolere i pomoću njih rukuje zahtjevima. App/helpers mapa sadrži pomoćne klase koje pomažu klasama kontrolera, modela i prikaza. Unutar mape app/models mogu se naći modeli koji omotavaju podatke spremljene u bazama. Unutar mape app/views sadržani su *HTML* predlošci na osnovu kojih će se generirati *View* i biti poslan korisniku. U mapi config nalaze razne konfiguracijske datoteke koje služe za postavljanje baze podataka, za postavljanje ruta i sl. Unutar mape db nalaze se tablice baza podataka. Unutar mape doc nalazi se dokumentacija o projektu koja se može generirati programskim okvirom unutar *Rubya* koji se zove *RubyDoc*. U mapi lib se nalaze biblioteke, osim ako eksplicitno nije naglašeno da pripadaju negdje drugdje. Unutar mape log mogu se naći bilješke o pogreškama. Unutar datoteke public nalaze se datoteke koje se ne mijenjaju i njihov prikaz se omogućava korisniku (npr. *JavaScript* datoteke). Unutar script mape nalaze se različite skripte za pokretanje alata koje će se eventualno koristiti sa *Railsom*. Svi testovi koje *Rails* sam generira ili koje programer napiše nalaze se u mapi test. Temp mapa namijenjena je za privremene datoteke. U mapi vendor nalaze se biblioteke od drugih prodavača programske podrške.

2.4. *NGINX*

NGINX je *HTTP* i obrnuti *prox*y poslužitelj te općeniti *TCP/UDP proxy* poslužitelj, besplatan i otvorenog koda. Osim što se koristi za služenje *HTTP* zahtjeva može se koristiti i za slanje i primanje elektroničke pošte preko *SMTP/POP3* protokola. Najvažnija karakteristika *NGINXA*, s aspekta ovog diplomskog rada, je da se može koristiti za balansiranje opterećenja na sustav. *NGINX* odlikuju stabilnost, skalabilnost, visoke performanse, jednostavno postavljanje i mala potrošnja resursa.

NGINX je napisan kako bi se mogao nositi sa *C10K* problemom. *C10K* problem podrazumijeva 10 000 i više korisničkih zahtjeva na server u isto vrijeme. Za rukovanje zahtjevima *NGNIX* se oslanja na događajima pogonjenu arhitekturu, koja rezultira korištenjem manjih i predvidljivih količina memorije. U pogodnostima *NGNIXA* uživaju kako virtualni privatni poslužitelji, tako i grozdovi poslužitelja koji na sebi nose robusne i kompleksne aplikacije.

Primjeri popularnih stranica koje koriste *NGINX* su: *Netflix, WordPress.com, GitHub, SoundCloud, AirBbnb*, za još stranica pogledati [15].

2.4.1. Karakteristike

Od osnovnih *HTTP* karakteristika *NGINX* odlikuju : posluživanje statičkih i *index* datoteka, ubrzani obrnuti *proxy* sa keširnajem, balansiranje opterećenja i toleriranje pogrešaka, ubrzana podrška sa keširanjem za *FasdCGI*, *uwsgi, SCGI* i *memcached* poslužitelj, modularna arhitektura, filtri koji uključuju kompresiju *gzippom*, raspone bajtova, *XSLT, SSI* i filtri za transformiranje slika te *SSL* i *TLS SNI* podrška. Ostale karakteristike *HTTP* poslužitelja su: virtualni poslužitelji bazirani na imenu i *IP*-u, podrška za perzistentne i cjevovodne konekcije, rekonfiguracija i podizanje izvršnih datoteka bez ometanja rada poslužitelja, vođenje dnevnika o pristupima sustavu, preusmjeravanje u slučaj 3xx-5xx pogrešaka, mijenjanje *URI*-a korištenjem regularnih izraza, izvršavanje različitih funkcija ovisno o klijentovoj adresi, pristup baziran na klijentovoj *IP* adresi, lozinki i rezultatu podzahtjeva, podržava *PUT, DELETE, MKCOL, COPY* i *MOVE* metode, FLV i *MP4* streamanje, ograničavanje vremena odgovora, ograničavanje zahtjeva i konekcija koje dolaze sa jedne adrese te ugrađeni *Pearl*.

Kada se koristi kao *mail proxy* poslužitelj ima slijedeće karakteristike: preusmjeravanje korisnika na *IMAP* ili *POP3* preko vanjskog *HTTP* poslužitelja za autentikaciju, autentikacija korisnika preko vanjskog *HTTP* poslužitelja i preusmjeravanje na unutrašnji *SMTP* poslužitelj, razne vrste autentikacijskih metoda za *POP3/IMAP/SMTP*, podrška za *SSL*, kao i podrška za *STARTTLS i STLS*.

Kao *TCP/UDP proxy* poslužitelj ima slijedeće karakteristike: općenito *proxyranje TCP*-a i *UDP*-a, *SSL* terminaciju za *TCP*, balansiranje opterećenja i toleriranje pogrešaka, kontrolu pristupa baziranu na klijentskoj adresi, ograničavanje broja istovremenih konekcija od strane jedne adrese.

Testiran je na slijedećim operativnim sustavima i platformama: *FreeBSD 3-10/i386*, *FreeBSD 5-10/amd64*, *Linux 2.2-4/i386*, *Linux 2.6-4/amd64*, *Linux 3-4/armv6l*, *amrv7l* i *aarch64*, *Solaris 9/i386*, *Solaris 10/i 386*, *amd64*, s*un4v, AIX 7.1/powerpc, HP-UX 11.31/ia64, Mac OS X/ ppc, i386, Windows XP, Windows Server 2003*.

2.4.2. Konfiguriranje *NGINXA*

Kako bi *NGINX* radio ispravno potrebno ga je instalirati i pravilno konfigurirati, tj. postaviti parametre. *NGINX* procesira zahtjeve na način da postoji jedna nit koja je *master* i nekoliko niti koje su radnici. Nit *master* učitava postavke i koordinira radnicima. Niti radnici se koriste za procesiranje zahtjeva od strane klijenta. Broj niti radnika se definira u konfiguracijskoj datoteci, a može postaviti automatski, ovisno o raspoloživoj računalnoj moći (broj jezgri *CPU*-a). Uobičajeno konfiguracijska datoteka se zove nginx.conf i nalazi se u direktoriju usr/local/nginx/conf, /etc/nginx ili /usr/local/etc/nginx.

Kako bi se *NGINX* pokrenuo, potrebno je pokrenuti izvršnu datoteku. Kada je *NGINX* pokrenut, kontrolira se preko komandne linije koristeći slijedeću sintaksu:

```
nginx -s signal
```

Kod 2.1. Sintaksa kontrole *NGINX* web poslužitelja

gdje signal može biti:
- *stop* – brzo gašenje,
- *quit* – zahvalno gašenje,
- *reload* – ponovno otvaranja konfiguracijske datoteke,
- *reopen* – ponovno otvaranje zabilješki.

Ponašanje *NGINX*-a se definira preko direktiva koja se nalaze u konfiguracijskoj datoteci. Direktive se dijele u jednostavne direktive koje završavaju za točkom zarezom(;) ili blokove direktiva koje se nalaze unutar vitica({}). Ako blok direktiva unutar sebe sadrži jednostavne direktive, onda se to smatra kontekstom. Direktive koje se nalaze u konfiguracijskoj datoteci i nisu u nekom kontekstu smatraju se glavnim kontekstom. Dio koda koji se nalazi u redu iza znaka # se zanemaruje, pa se na taj način mogu komentirati direktive.

Kako bi se *NGINX* postavio kao poslužitelj statičkog sadržaja (hipertekst i slike), potrebno napraviti *HTML* datoteku index.html i postaviti ju u mapu /data/www. Slike je potrebno staviti u mapu /data/images. Slijedeće korak je otvoriti konfiguracijsku datoteku i u nju upisati slijedeće direktive:

```
http {
    server {
        location / {
```

```
            root data/www;
        }
        location /images/ {
            root /data;
        }
    }
}
```

Kod 2.2. Direktive potrebne za rad *NGINX* web poslužitelja

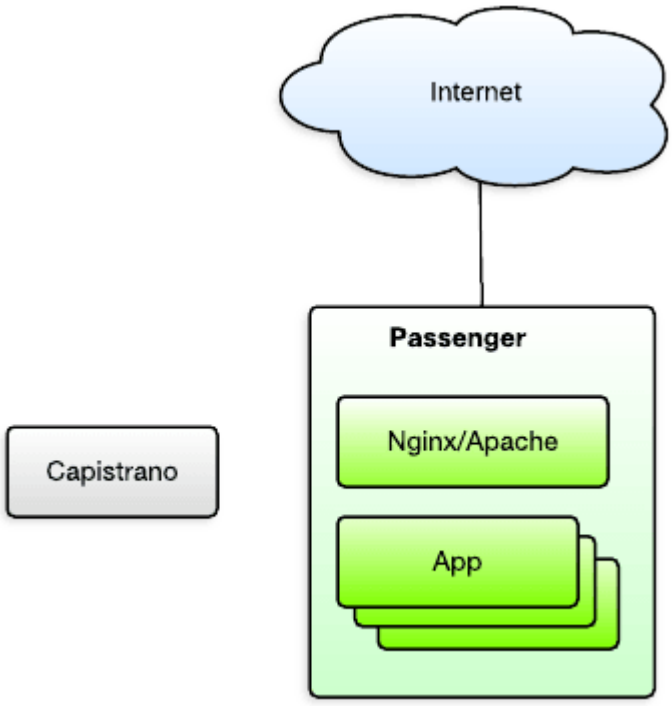
NGINX pokreće *HTTP* poslužitelj i iako port nije zadan eksplicitno, sluša na standardnom *HTTP* portu 80. Na lokalnom računalu sadržaj će biti dostupan na http://localhost/. *Location* blok direktiva preusmjerava zahtjeve sa http://localhost/primjer.html na data/www/primjer.html i vraća klijentu primjer.html datoteku. U slučaju da navedena datoteka ne postoji *NGINX* vraća pogrešku 404. Slično vrijedi i za slike. Ako klijent zatraži sliku sa http://localhost/images/slika.jpg , *NGINX* ga preusmjerava na data/images/slika.jpg i u slučaju da slika na navedenoj lokaciji ne postoji vraća pogrešku 404. Kako bi se primijenile konfiguracijske promijene potrebno je pokrenuti slijedeću naredbu prema primjeru iz [16]:

```
ngnix -s reload
```

Kod 2.3. Ponovo pokretanje *NGINX*-a

2.5. *Passenger*

Passenger je web aplikacijski server otvorenog koda koji rukuje *HTTP* zahtjevima, procesira poruke te omogućava administraciju, motrenje i dijagnosticiranje problema. *Passenger* je jednostavan za korištenje i olakšava postavljanje aplikacije na server , puštanje u produkciju i skaliranje. Može se koristiti samostalno ili kao modul unutar web poslužitelja kao što su *NGINX* ili *Apache*. U ovom diplomskom radu *Passenger* je korišten kao modul unutar *NGINX* web poslužitelja. Također omogućava korištenje aplikacije u više okruženja – produkcijskom i razvojnom i to istovremeno. Dakle, moguće postaviti stranicu u produkcijsko okruženje i istovremeno na njoj razvijati dodatne karakteristike ili popravljati stare u razvojnom okruženju.



Slika 2.2. Integracija *Passengera* kao modula sa *NGINXOM* [19]

Prema slici 2.2. *Passenger* je integriran u ostale komponente programskog stoga za potrebe kasnijeg mjerenja opterećenja. *NGINX* je web poslužitelj i koristi se za posluživanje *HTTP* zahtjeva. Sam za sebe ne može pogoniti *Ruby* aplikaciju te se zbog toga kombiniraju web i aplikacijski poslužitelji. Web poslužitelji omogućuju aplikaciji da komunicira sa klijentom putem *HTTP* protokola, rukuje *I/O* sigurnosti, istekom vremena konekcije, dok aplikacijski server pogoni samu aplikaciju.

Prilikom korištenja *Passengera* kao modula u kombinaciji sa *NGINXOM* *Passenger* se postavlja preko *NGINX* konfiguracijske datoteke. Na taj način se umjesto sa dvije upravlja sa jednom datotekom.

Capistrano je bitna komponenta programskog stoga koja omogućava automatizaciju izdavanja nove verzije aplikacije. Prilikom izdavanja nove verzije aplikacije potrebno je podići kod nove aplikacije na poslužitelj, pokrenuti komandu za instaliranje *gemova* preko *bundlera*, ponovno pokretanje procesa, a *Capistrano* navedene radnje automatizira.

U praksi se pokazalo kako su performanse u samostalnom načinu rada ili u kombinaciji sa *NGINXOM* nešto bolje. Pokazalo se kako je samostalni način rada

bolji za razvojni način, a integrirani za produkcijski. Također, ako se *Passenger* integrira sa *NGINXOM* ili sa *Apacheom* moguće je postaviti više aplikacija koje će raditi istovremeno.

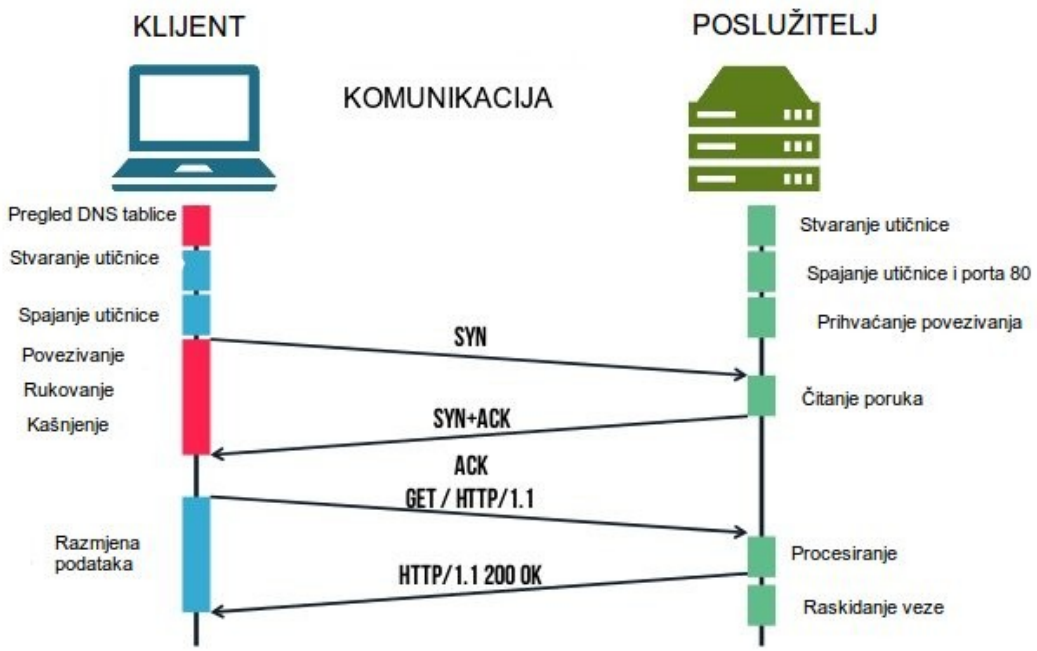
3. BALANSIRANJE OPTEREĆENJA

3.1. Balansiranje opterećenja

3.1.1. *HTTP* zahtjevi

HTTP (engl. *Hypertext Transfer Protocol*) je aplikacijski protokol koji omogućava razmjenu i prijenos hiperteksta i hipermedije. Hipertekst je logički strukturirani tekst koji sadrži logičke veze između čvorova koji sadrže tekst, a hipermedija može sadržavati i grafiku, video te zvuk, prema [20].

HTTP je jedan od važnijih protokola na kojima počiva internet i koristi se u klijent-poslužitelj mrežama. Komunikacija se vrši na način kako je pokazano na slici 3.1..



Slika 3.1. Uspostavljanje *HTTP* komunikacije [21]

Razmjena podataka sa slike 3.1. može izgledati i drugačije. Osim metode *GET* za dohvaćanje sadržaja sa određenog *URI*-a (engl. *Uniform Resource Identifier*) klijent može koristiti i :

- *HEAD* – slična metoda kao i *GET*, samo što zahtjeva sadržaj html datoteke koji se nalazi unutar <head> </head> tagova. Ova metoda se koristi za dohvaćanje raznih meta podataka.
- *POST* – ovom metodom se zahtjeva da web poslužitelj prihvati i spremi podatke koji se nalaze unutar zahtjeva.
- *PUT* – ovom metodom se zahtjeva da se podaci koji se nalaze u zahtjevu spreme unutar *URI*-a koji server ponudi.
- *DELETE* – pomoću ove metode se briše specificirani resurs.
- *TRACE* – ova metoda prosljeđuje klijentu primljeni zahtjev kako bi mogao vidjeti postoje li kakve promjene napravljene od strane posredničkih servera.
- *OPTIONS* – ova metoda vraća *HTTP* metode koje je moguće primijeniti za specificirani *URL* (engl. *Uniform Resource Locator*).
- *CONNECT* – ova metoda pretvara zahtjev u *TCP/IP* tunel, kako bi se olakšala *SSL* (engl. *Secure Sockets Layer*) kriptirana komunikacija kroz nekriptirani *HTTP Proxy*.
- *PATCH* – pomoću ove metode se vrše djelomične promjene na resursu, prema [22].

3.1.2. Princip rada balansiranja opterećenja

Balansiranje opterećenja je distribuiranje posla koje računalo mora obaviti između dva ili više računala kako bi se obavilo što više posla u vremenu i kako bi svi klijenti bili što brže posluženi. Balansiranje opterećenja je ujedno i razlog za nastajanje nakupina računala.

Balanser opterećenja je računalo koje je postavljeno ispred ostalih računala i brine se o tome da se posao jednoliko distribuira kako bi iskorištenje resursa bilo što bliže optimumu, iz čega proizlaze i optimalne performanse same aplikacije.

Postoje različiti algoritmi prema kojima se vrši balansiranje opterećenja i odabir algoritma ovisi o samoj namjeni sustava. Najčešće korišteni algoritmi su :

- *Round Robin* – zahtjevi se distribuiraju na grupu poslužitelja sekvencijalno.
- *Least Connections* – novi zahtjev se šalje poslužitelju koji je trenutno povezan sa najmanje klijenata. Relativni računalni kapacitet svakog poslužitelja određuje koji poslužitelj ima najmanje konekcija.

- *IP Hash* – *IP* adresa klijenta se koristi za određivanje koji server prima zahtjev.

Balanser opterećenja posjeduje funkcionalnost održavanja perzistentne konekcije što je u određenim slučajevima potrebno kako bi se izbjegle greške transakcije i problemi sa performansama (npr. kupovina preko interneta općenito).

Balansiranje opterećenja je jako koristan alat koji se koristi kod aplikacija koje brzo rastu. Karakteristično za *Amazon Elastic Computer Cloud* koji naplaćuje korisniku kapacitet resursa koji koristi. U takvom okruženju blanser opterećenja olakšava skaliranje aplikacije dodavanjem/uklanjanjem poslužiteljskih resursa, više informacija na [23].

3.2. *NGINX* kao balanser opterećenja

Balansiranje opterećenja se može vršiti preko sklopovlja ili preko programske podrške. U ovom radu korišten je *NGINX* kao programska podrška za distribuciju opterećenja. Balansiranje opterećenja se može vršiti na aplikacijskom sloju *TCP/IP* modela (koristi se *HTTP* protokol) ili na transportnom sloju istog modela (koristi se *TCP* protokol). Za potrebe ovog diplomskog rada koristi će se balansiranje opterećenja na aplikacijskom sloju (balansiranje *HTTP* zahtjeva).

Kao što je objašnjeno u potpoglavlju 2.4.2. *NGINX* se konfigurira na način da se u konfiguracijsku datoteku unose direktive. Najjednostavnija konfiguracija za balansiranje opterećenja izgleda kao u tablici ispod.

```
http {
   upstream myapp1 {
      server srv1.example.com;
      server srv2.example.com;
      server srv3.example.com;
   }

   server {
      listen 80;
```

```
    location / {
        proxy_pass http://myapp1;
    }
  }
}
```

Kod 3.1. Najjednostavnija realizacija balansiranja opterećenja koristeći *NGINX*, prema [24]

Za kod iz tablice iznad vrijedi da se tri instance iste aplikacije pogone na poslužiteljima serv1, serv2 i serv3. Kada nije specificiran, korišteni algoritam za balansiranje je *Round Robin*. Svi zahtjevi se putem *proxy* poslužitelja prosljeđuju grupi poslužitelja mypp1 i *NGINX* primjenjuje balansiranje opterećenja.

Ako bi se koristio neki drugi algoritam za balansiranje opterećenja, npr. *Least Connections*, potrebno je dodati direktive kao što je pokazano u tablici 3.2.

```
upstream myapp1 {
    least_conn;
    server srv1.example.com;
    server srv2.example.com;
    server srv3.example.com;
}
```

Kod 3.2. Realizacija balansiranja opterećenja koristeći *Least Connections* algoritam, prema [24]

Potrebno je imati na umu činjenicu kako navedena 2 algoritma ne garantiraju da će svaki zahtjev istog klijenta poslužiti isti poslužitelj. Perzistentna konekcija se može postići na način da se koristi *IP Hash* alogirtam i to na način prikazan u tablici 3.3.

```
upstream myapp1 {
    ip_hash;
    server srv1.example.com;
    server srv2.example.com;
    server srv3.example.com;
```

```
}
```
Kod 3.3. Realizacija balansiranja opterećenja koristeći *IP Hash* algoritam, prema [24]

Balansiranje opterećenja se može optimizirati korištenjem faktora težine. Poslužitelju ili više njih se dodjeli parametar težine i na taj način određeni poslužitelj može posluživati više tj. manje zahtjeva.

```
upstream myapp1 {
    server srv1.example.com weight = 3;
    server srv2.example.com;
    server srv3.example.com;
}
```
Kod 3.4. Realizacija balansiranja opterećenja sa faktorom težine

Prema direktivama iz tablice 3.4. poslužitelju srv1 će biti dodijeljena težina 3. Konkretno, ako bi sustav dobio 5 zahtjeva u određenom trenutku, prema poslužitelju srv1 biti će usmjerena 3, prema srv2 i srv3 po jedan zahtjev. Na ovaj način se može upravljati distribucijom opterećenja ako poslužitelji posjeduju različit poslužiteljski kapacitet.

3.3. *HAProxy*

HAProxy (engl. *High Availability Proxy*) je besplatna programska podrška otvorenog koda koja se koristi za balansiranje opterećenja na internet poslužiteljima kao alternativa *NGINXU*. Pošto je u ovom radu korišten *NGINX,* ovo je samo pregled alternativne solucije.

HAProxy se koristi na operativnim sustavima *Linux*, *Solaris* i *FreeBSD*, a za visoku dostupnost i balansiranje opterećenja koriste ga mnoge popularne web aplikacije poput *GitHub*-a, *Imgur*-a, *Instagram*-a i *Twitter-a*, prema [25].

Glavna razlika između *NGINXA* i *HaProxya* je u tome što je *HaProxy* samo balanser opterećenja i obrnuti proxy, dok je *NGINX* web poslužitelj koji može ostvariti navedene funkcije. *HAProxy* može balansirati opterećenje na aplikacijskom sloju koristeći *HTTP* protokol i na transportnom sloju koristeći *TCP* protokol.

U istraživanjima provedenim 2008. godine pokazalo se da *HAProxy* pokazuje bolje performanse pri manjem opterećenju(3 istovremene konekcije i manje), dok se

NGINX pokazao kako se bolje nosi sa većim opterećenjem(10 do 30+ istovremenih konekcija).

Prema istom istraživanju pokazano je kako *HAProxy* ne podržava *SSL* (engl. *Secure Sockets Layer*) i da je *NGINX* u tom slučaju bolja opcija, prema [26].

Gerealni zaključak istraživanja bio bi kako su i *HAProxy* i *NGINX* kvalitetni alati za balansiranje opterećenja gdje svaki ima svoje prednosti i nedostatke te odluka koji koristiti najviše ovisi o specifičnoj aplikaciji i osobnim preferencijama.

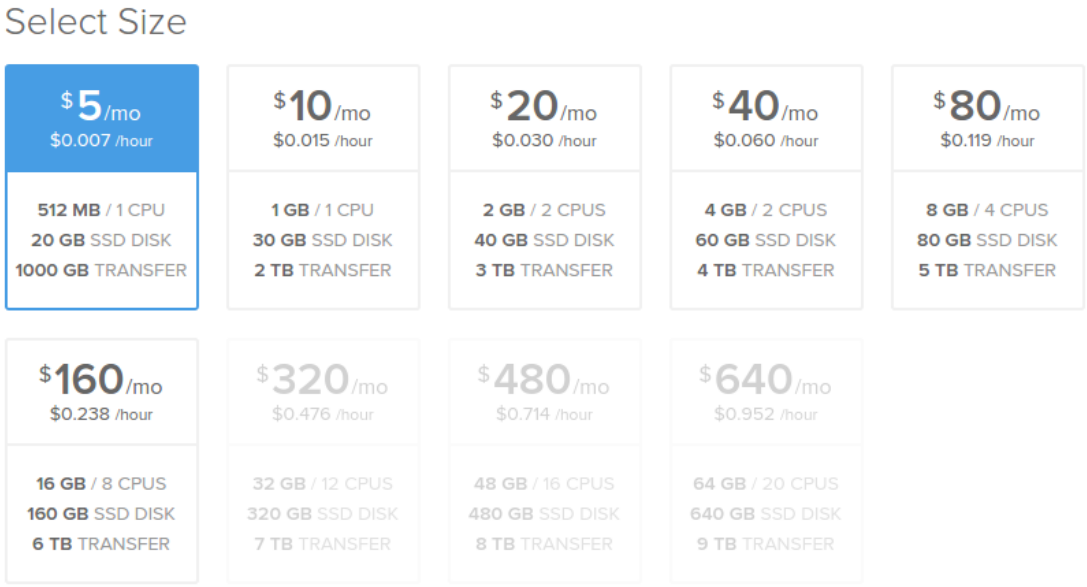
4. KONFIGURACIJA BALANSIRANJA OPTEREĆENJA I REZULTATI

4.1. Inicijalne postavke poslužitelja

4.1.1. Osiguravanje poslužiteljskih resursa

Kako bi web aplikacija bila dostupna klijentima preko web poslužitelja prvi korak je osigurati potrebne poslužiteljske resurse (*CPU, RAM* memoriju i diskovni prostor). Postoje razni servisi koji nude navedene usluge. Za potrebe ovog diplomskog rada korišteni su *Digital Ocean* za postavljanje na poslužitelj aplikacije bez balansiranja opterećenje i *Vultr* za postavljanje na poslužitelj aplikacije s balansiranjem opterećenja.

Bilo da se koristi *Digital Ocean* ili *Vultr* procedura je ista, tj. prvo se mora odabrati količina resursa koje ćemo zakupiti. Iz razloga što aplikacija koju postavljamo na server nije zahtjevan, zakupljena je 1 jezgru *CPU*-a, 512 MB radne memorije i 20 GB diskovnog prostora. Osim potrebnih resursa potrebno je izabrati i operacijski sustav na kojem će počivati programska podrška. Odabrana je *Ubuntu* inačica 16.04.1 *LTS* koja je u vrijeme odabira bila najaktualnija.



Slika 4.1. Izgled izbornika za odabir poslužiteljskih resursa [27]

4.1.2. Dodavanje novog korisnika, privilegija i *SSH* zaštite

Kako bi se konfigurirao poslužitelj, instalirala programska podrška i unosile promjene, potrebno se povezati sa poslužiteljom koristeći *SSH* (engl. *Secure Shell*). Povezivanje sa poslužiteljem se može realizirati koristeći neki od programa npr. *Putty* (češće se koristi u *Windows OS*) ili preko naredbenog retka (češće se koristi kod *Linux OS*). Osobno sam koristio drugu soluciju iz razloga što koristim *Linux OS*. Spajanje sa poslužiteljem se vrši po sintaksi prikazanoj u tablici 4.1.

```
$ ssh root@SERVER_IP_ADDRESS
```

Kod 4.1. Sintaksa naredbe za spajanje sa poslužiteljem preko *SSH*

Jedini korisnik koji trenutno postoji na našem poslužitelju je *root*. *Root* je administrativni korisnik širokih privilegija, koji može napraviti destruktivne promjene, pa čak i slučajno. Kako bi se izbjegao destruktivni scenario obično se dodaje još korisnik/a za svakodnevnu upotrebu. Kada smo prijavljeni kao *root* novi korisnik se dodaje naredbom iz tablice 4.2.

```
$ adduser USER_NAME
```

Kod 4.2. Sintaksa naredbe za dodavanje novog korisnika

Sada imamo novog korisnika koji ima standardne privilegije. Ako bi htjeli koristiti naredbe koje traže *root* privilegije, morali bi se odjaviti sa novim korisnikom i ponovo prijaviti sa *root* korisnikom. Kako bi izbjegli stalno prijavljivanje i odjavljivanje kada su potrebne više privilegije možemo novog korisnika dodati u grupu super korisnika. Svaki super korisnik ima standardne privilegije, a kada se žele postići veće privilegije potrebno je ispred željene naredbe dodati *sudo* (engl. *Super Do*). Kako bi dodali novog korisnika u *sudo* grupu potrebno je kao *root* pokrenuti naredbu iz tablice 4.3.

```
$ gpasswd -a USER_NAME sudo
```

Kod 4.3. Naredba za dodavanje novog korisnika u *sudo* grupu

Kako bi se korisnik mogao prijaviti na poslužitelj preko *SSH* koristi se zaštita pomoću korisničkog imena i zaporke. U praksi, kako bi se dodatno povećala

sigurnost, koristi se autentikacija pomoću privatnog i javnog ključa. Da bi se zaštita pomoću privatnog i javnog ključa omogućila potrebno je proći slijedeće korake :

1. Generirati privatni i javni ključ prema naredbi iz tablice 4.4. i spremiti generirani ključ u mapu prema tablici 4.5.

```
$ ssh-keygen
```

Kod 4.4. Naredba za generiranje privatnog i javnog ključa

```
/Users/USER_NAME/.ssh/id_rsa
```

Kod 4.5. Mapa u koju je potrebno spremiti generirane ključeve

2. Nakon što su ključevi generirani, potrebno je kopirati generirani javni ključ na željeni poslužitelj što je vrlo jednostavno koristeći *ssh-copy-id* skriptu, prema tablici 4.6.

```
$ ssh-copy-id USER_NAME@SERVER_IP_ADRESS
```

Kod 4.6. Kopiranje javnog ključa na poslužitelj

3. Kako bi se novom korisniku omogućila autentikacija koristeći privatni i javni ključ, potrebno je javni ključ na poslužitelju premjestiti u posebnu datoteku unutar korisnikovog *home* direktorija.

```
$ su – USER_NAME
$ mkdir .ssh
$ chmod 700 .ssh
$ nano  .ssh/authorized_keys
$ chmod 600 .ssh/authorized_keys
$ exit
```

Kod 4.7. Naredbe za dodavanje autentikacije pomoću privatnog i javnog ključa za novog korisnika

Naredbom $ su – USERNAME ulazimo u korisnikov *home* direktorij. Naredbom $ mkdir .ssh se pravi nova datoteka .ssh. Naredbom $ chmod 700 .ssh se datoteci .ssh se ograničavaju prava pristupa. Naredbom $ nano .ssh/authorized_keys otvara se .ssh datoteka u programu za obradu teksta *nano* i tu je potrebno kopirati

generirani javni ključ. Naredbom $ chmod 600 .ssh/authorized_keys se ponovo ograničavaju prava pristupa i $ exit za odjavu *root* korisnika.

4. Dodatno povećanje sigurnosti omogućava se zabranom prijave kao *root* korisnik preko *SSH*. Naredbom $ nano /etc/ssh/sshd_config otvara se konfiguracijska datoteka za postavljanje *SSH* usluge. U navedenoj datoteci je potrebno promijeniti liniju PermitRootLogin yes u PermitRootLogin no i ponovo pokrenuti *SSH* uslugu naredbom $ service ssh reload, više informacija na [28].

4.2. Instaliranje programske podrške

4.2.1. Instaliranje *Ruby*-a

Postoji više načina kako instalirati *Ruby*. Prilikom realizacije diplomskog rada korištena je metoda skidanja i prevođenja izvornog koda. Prije nego što se krene s instalacijom potrebno je dohvatiti sve pakete koji su nam dostupni. Navedena akcija se realizira naredbom iz tablice 4.8.

```
$ sudo apt-get update
```

Kod 4.8. Skidanje dostupnih paketa za unapređivanje sustava

Slijedeći korak je instalirati određene zavisnosti koje bi trebale olakšati instalaciju *Rubya* što je više moguće.

```
sudo apt-get install build-essential libssl-dev libyaml-dev libreadline-dev openssl curl git-core zlib1g-dev bison libxml2-dev libxslt1-dev libcurl4-openssl-dev nodejs libsqlite3-dev sqlite3
```

Kod 4.9. Instaliranje potrebnih paketa za instaliranje *Ruby*-a

Nakon instaliranja zavisnosti, potrebno je napraviti novi direktorij naredbom $ mkdir ~/ruby, pomaknuti se u novi direktorij naredbom $ cd ~/ruby, skinuti aktualnu verziju *Rubya* naredbom $ wget http://cache.ruby-lang.org/pub/ruby/2.3/ruby-2.3.2.tar.gz , raspakirati skinutu datoteku naredbom $ tar -xzf ruby-2.3.2.tar.gz , premjestiti se u direktorij u kojem smo raspakirali datoteku naredbom $ cd ruby-2.3.2. , pokrenuti skriptu koja će prevesti izvorni kod naredbom $ ./configure , pokrenuti

Makefile skriptu koja će napraviti egzekucijsku datoteku naredbom $ make, pokrenuti istu naredbu sa install parametrom $ make install te ukloniti privremenu datoteku u kojoj se nalazio *Ruby* naredbom $ rm -rf ~/ruby.

4.2.2. Instaliranje *NGINXA* i *Passengera*

Postoji više načina instaliranja *NGINXA* i *Passenger*a. Odlučio sam se za instalaciju korištenjem *APT*-a (engl. *Advance Packaging Tool*), iz razloga što je sama instalacija jednostavna i što je ažuriranje također jednostavno.

Prvi korak je instalirati *PGP* ključ koji kriptografsku privatnost i autentikaciju prilikom razmjene podataka. Navedena akcija se realizira naredbom iz tablice 4.10.

```
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 561F9B9CAC40B2F7
```

Kod 4.10. Instaliranje *PGP* ključa

Slijedeći korak je napraviti *APT* izvornu datoteku koristeći naredbu iz tablice 4.11. i unutar nje unijeti liniju iz tablice 4.12.

```
$ sudo nano /etc/apt/sources.list.d/passenger.list
```

Kod 4.11. Kreiranje *APT* izvorne datoteke

```
deb https://oss-binaries.phusionpassenger.com/apt/passenger xenial main
```

Kod 4.12. Linija koja usmjerava *APT* na *Passenger APT* direktorij

Nakon koraka iznad potrebno je promijeniti vlasnika novo stvorene datoteke i prava pristupa, ažurirati *APT cache* i instalirati *Passenger* s *NGINXOM* naredbama iz tablice 4.13.

```
$ sudo chown root: /etc/apt/sources.list.d/passenger.list
$ sudo chmod 600 /etc/apt/sources.list.d/passenger.list
$ sudo apt-get update
$ sudo apt-get install nginx-extras passenger
```

Kod 4.13. Ograničenje prava pristupa i instalacija *NGINXA* i *Passengera*

4.2.3. Postavljanje web poslužitelja i postavljanje aplikacije na poslužitelj bez balansiranja opterećenja

Pošto je *Passenger* instaliran kao modul unutar *NGINXA*, da bi se oba poslužitelja mogla postavljati preko *NGINX* konfiguracijske datoteke potrebno je otvoriti konfiguracijsku datoteku od *NGINXA* i u *HTTP* bloku odkomentirati linije koje govore gdje se nalaze *Passenger* i *Ruby* prema tablici 4.13.

```
$ sudo nano /etc/nginx/nginx.conf
# passenger_root /usr/lib/ruby/vendor_ruby/phusion_passenger/locations.ini;
# passenger_ruby /usr/bin/ruby;
```

Kod 4.14. Postavljanje rada *Passengera* kao modula unutar *NGINX*A

Komentari u poslužiteljskim direktivama se označavaju znakom # i kako bi se direktiva odkomentirala potrebno je ukloniti znak #.

Za potrebe mjerenja opterećenja koristio sam web aplikaciju koju je napravila kolegica za njezin diplomski rad i zove se Alumni ETFOS. Da bi bilo moguće pogoniti navedenu aplikaciju potrebno je instalirati i *Rails*. Najbrži način instaliranja *Railsa* je kao *gema* i to bez dokumentacije pomoću naredbe iz tablice 4.15.

```
$ sudo gem install --no-rdoc --no-ri rails
```

Kod 4.15. Instalacija *Railsa* bez dokumentacije

Kako bi aplikacija bila na poslužitelju i kako bi ju bilo moguće pogoniti s *NGINXOM* i *Passengerom* potrebno je klonirati repozitorij sa *Github*-a u proizvoljni direktorij na poslužitelju. Kako to napraviti pokazano je u tablici 4.16.

```
$ git clone https://github.com/monikaCiv/alumni_etfos.git
```

Kod 4.16. Kloniranje *Github* repozitorija

Nakon što smo klonirali repozitorij aplikacija se nalazi na poslužitelju. Potrebno je otići u direktorij koji sadrži aplikaciju i pomoću *nano* programa za obradu teksta otvoriti *Gemfile* od aplikacije. Unutar *Gemfilea* potrebno je otkomentirati *gem* 'therubyracer' i uskladiti verziju *Rubya* u *Gemfileu* sa instaliranom. Kako bi navedene promjene imale efekta potrebno je instalirati promijenjene *gemove* naredbom $ bundle install.

Kada na poslužitelju imamo postavljenu aplikaciju sa instaliranim *gemovima* potrebno je konfigurirati web poslužitelj. Cilj je promijeniti standardno ponašanje web poslužitelja i napisati direktive kako bi se poslužitelj ponašao kako mi želimo. Slijedeći korak je otvoriti *NGINX* kofiguracijsku datoteku i zakomentirati linije prema tablici 4.17.

```
$ sudo nano /etc/nginx/sites-available/default
listen 80 default_server; <--- potrebno zakomentirati
listen [::]:80 default_server ipv6only=on; <--- potrebno zakomentirati
```

Kod 4.17. Isključivanje standardnih *NGINX* postavki

Kako bi instruirali *NGINX* da se ponaša kako mi želimo potrebno je napraviti novu konfiguracijsku datoteku naredbom iz tablice 4.18.

```
$ sudo nano /etc/nginx/sites-available/alumni_etfos
```

Kod 4.18. Kreiranje nove konfiguracijske datoteke za *NGINX*

U novu stvorenu datoteku potrebno je upisati poslužiteljske direktive kako je prikazano u tablici 4.19.

```
server {
  listen 80 default_server;
  server_name www.krusec.levara.net;
  passenger_enabled on;
  passenger_app_env production;
  root /home/rails/alumni_etfos/public;
}
```

Kod 4.19. Konfiguracija *NGINXA*

Prva direktiva *listen* govori web poslužitelju da sluša na portu 80. Direktiva *server_name* instrura server da će zahtjevi na aplikaciju dolaziti na postavljenu domenu. Direktiva *passenger_enabled on* govori kako je *Passenger* uključen, *passenger_app_env* govori kako će okolina biti produkcijska, a *root* direktiva pokazuje putanju na našu aplikaciju, prema [29].

4.2.4. Postavljanje web poslužitelja i postavljanje aplikacije na poslužitelj sa balansiranjem opterećenja

Kako bi se postavio poslužitelj koji balansira opterećenje potrebno je provesti korake postavljanja poslužitelja opisane u poglavljima 4.1. do 4.2.3.

Razlika u postavkama je sam servis koji je u prvom slučaju bio *Digital Ocean*, a u slučaju sa balansiranjem opterećenja to je *vultr*. Razlika je i u zakupljenim resursima. Prilikom prvog korištenja *vultr* servisa dobije se besplatnih 20 $, pa su zakupljena 2 *CPU*-a, 2048 MB radne memorije i 3000 GB diskovnog prostora.

Jedina razlika u postavljanju poslužitelja s balansiranjem opterećenja i poslužitelja bez balansiranja je u *NGINX* konfiguraciskoj datoteci. Navedena datoteka sadrži direktive prema tablici 4.20.

```
upstream myapp1 {
 server krusec2.levara.net:8080;
 server krusec3.levara.net;
}



server {
  listen 80;
  server_name krusec.levara.net;
  #passenger_enabled on;
  #passenger_app_env production;
  #root /home/jkrusec/alumni_etfos/public;

       location /{
             proxy_pass http://myapp1;
       }
}
```

Kod 4.20. *NGINX* konfiguracijska datoteka s postavljenim balansiranjem opterećenja

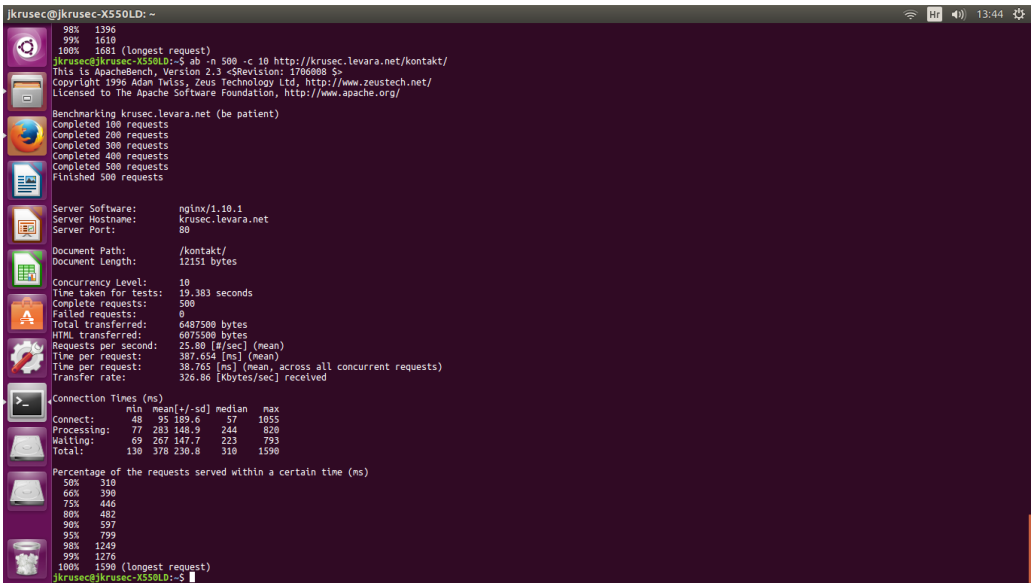
4.3. Mjerenje opterećenja s i bez balansiranja opterećenja

4.3.1. *Apache Benchmark* program za mjerenje opterećenja

Apache Benchmark je program koji generira *HTTP* opterećenje uz zadane parametre *-n* što je broj zahtjeva i *-c* što je maksimalan broj istovremenih zahtjeva. Primjer za njegovo korištenje prikazana je u tablici 4.21.

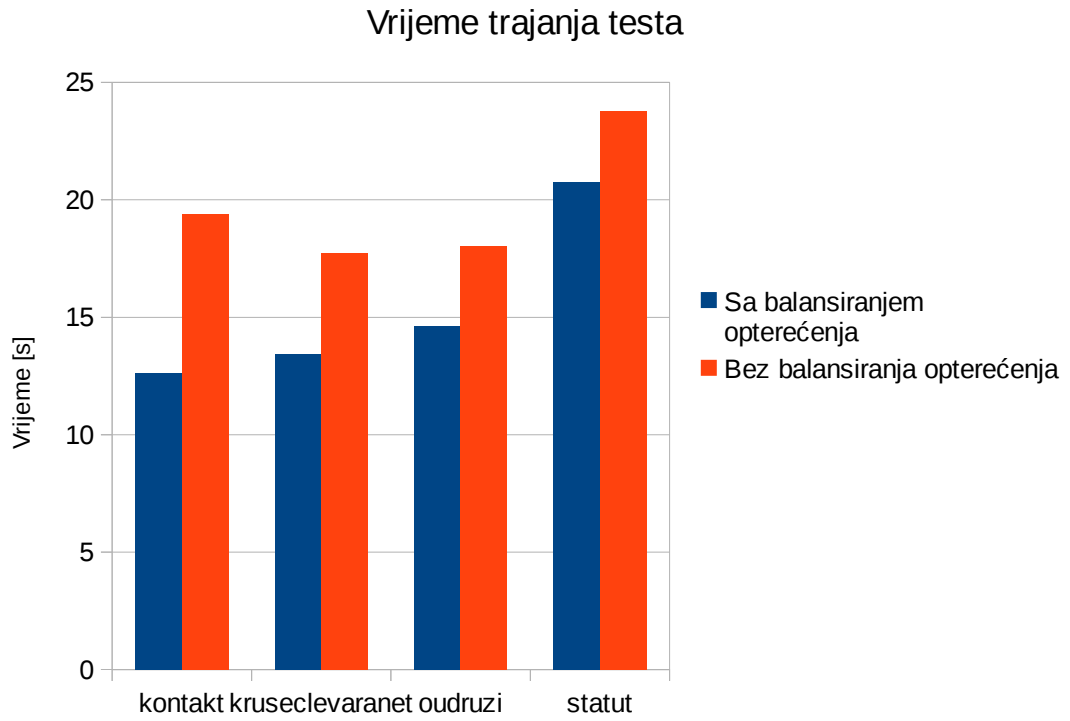
```
$ ab -n 500 -c 10 http://krusec.levara.net/kontakt/
```

Kod 4.21. Korištenje *Apache Benchmark* programa za generiranje 500 zahtjeva od koji 10 mogu biti istovremeni na stranicu http://krusec.levara.net/kontakt/

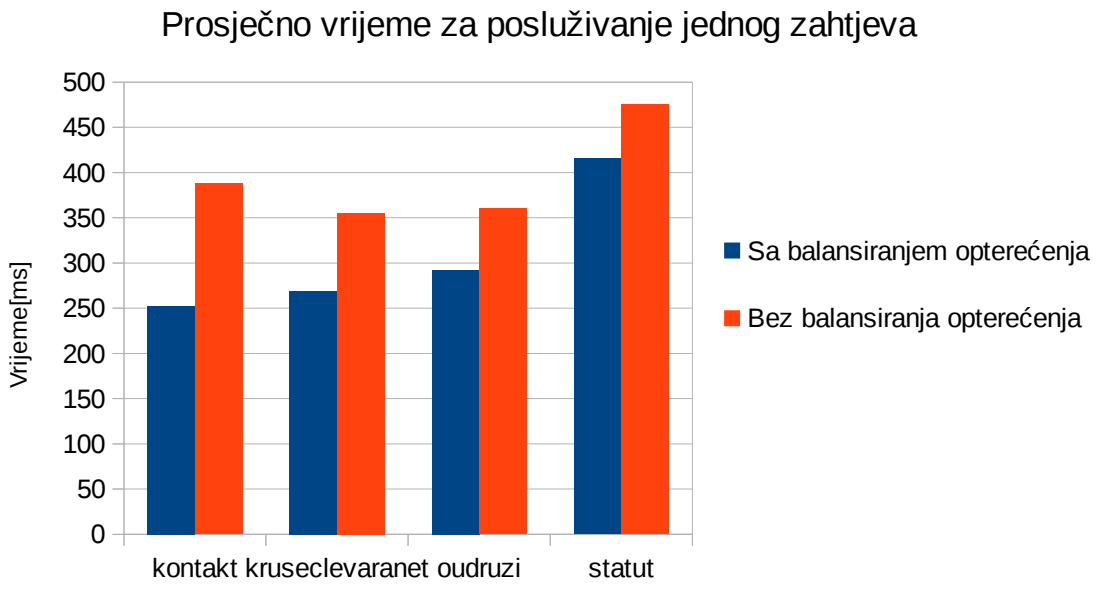


Slika 4.2. Primjer testa za stranicu http://krusec.levara.net/kontakt/ na 500 zahtjeva od kojih 10 mogu biti istovremeni

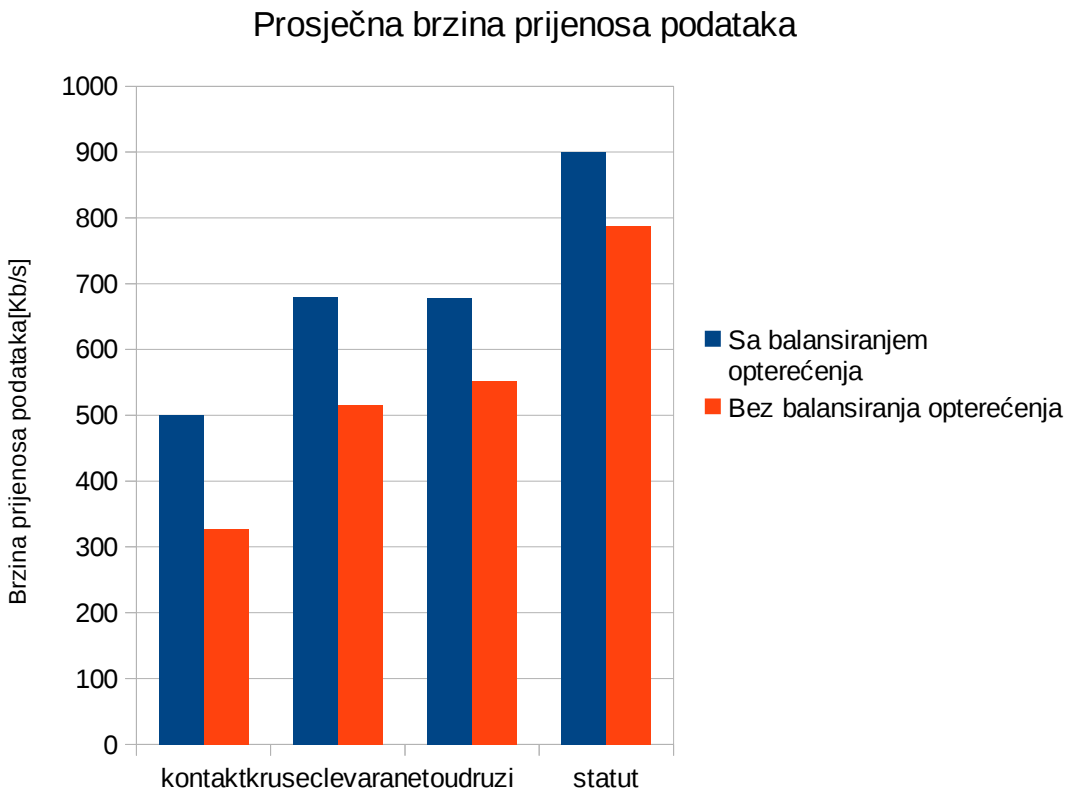
4.3.2. Rezultati mjerenja



Graf 4.1. Vremena trajanja testa sa i bez balansiranja opterećenja za stranice unutar domene krusec.levara.net



Graf 4.2. Vrijeme potrebno za posluživanje jednog zahtjeva za stranice unutar domene



Graf 4.3. Prosječne brzine prijenosa podataka za stranice unutar domene

4.3.3. Komentar na dobivene rezultate

Prvi relevantan parametar je vrijeme trajanja testa. Sve stranice unutar domene krusec.levara.net su testirane su pod istim uvjetima, a to je petsto zahtjeva od kojih deset može biti istovremeno. Usporedimo li vrijeme trajanja testa za stranice bez balansiranja opterećenja i za one sa balansiranim opterećenjem da se primijetiti da kod stranica sa balansiranjem opterećenja testovi traju kraće za od oko 13% pa do oko 44% . Dakle, balansiranjem opterećenja postigli smo brži odziv stranica što nam je i bio cilj.

Slijedeći relevantni parametri su vrijeme potrebno da se posluži zahtjev te vrijeme potrebno da se posluže istovremeni zahtjevi. Kod stranica kod kojih se koristilo balansiranje opterećenja vrijeme za služenje zahtjeva je manje bilo da se radi

o istovremenim ili svakom posebno, a to proizlazi iz kraćeg vremena trajanja testa, tj. iz bržeg posluživanja zahtjeva zbog balansiranja opterećenja.

Ako se pogleda parametar brzine prijenosa, može se uočiti da je balansiranje opterećenja povećalo brzinu prijenosa podataka.

Vremena konekcija su također manja kod stranica sa balansiranjem opterećenja iz razloga što se zahtjev brže posluži, pa je potrebno kraće vrijeme konekcije.

Također, kod stranica sa balansiranim opterećenjem isti postotak zahtjeva je poslužen u kraćem vremenu što je uzrokovano balansiranjem opterećenja.

5. ZAKLJUČAK

U ovom diplomskom radu pokazano je kako se korištenjem balansiranja opterećenja mogu poboljšati performanse na način da poslužitelji koji nose aplikaciju mogu poslužiti jednak broj zahtjeva u kraćem vremenu.

Prilikom realizacije diplomskog rada bili je potrebno zakupiti poslužiteljske resurse na servisima *Digital Ocean* i *vultr*. Zatim je bilo potrebno instalirati programski stog. Programski stog podrazumijeva platformu na kojoj počiva aplikacija. Instalacija stoga se sastoji od instaliranja *Rubya*, instaliranja i konfiguriranja aplikacijskog poslužitelja (*Phusion Passenger*), instaliranja i konfiguriranja web poslužitelja (*NGINX*) i postavljanje na poslužitelj aplikacije koja mi je stavljena na raspolaganje. Nakon što je aplikacija upogonjena i dostupna na internetu bilo je potrebno testirati njezine performanse koristeći alat *Apache Benchmark*. Nakon toga, bilo je potrebno postaviti web poslužitelj kao balanser opterećenja. Nakon što je balansiranje opterećenja uspješno postavljeno, stranica je ponovno testirana programom *Apache benchmark*. Nakon usporedbe rezultata pokazano je, kao što je i pretpostavljeno, kako je balansiranje opterećenja princip koji je neizbježan prilikom optimizacije performansi poslužiteljske infrastrukture koja nosi stranice sa velikim brojem istovremenih zahtjeva.

Zbog financijskih ograničenja resursi su zakupljeni na dva različita servisa iz čega proizlazi da se poslužitelji nalaze na različitim fizičkim lokacijama što povećava put paketa iz čega proizlazi duže vrijeme za posluživanje zahtjeva. Kada bi poslužitelji bili u istom *data* centru vrijeme putovanja paketa bi se smanjilo, a samim time bi se poboljšale performanse aplikacije.

Performanse stranica koje koriste balansiranje opterećenja bi se mogle dodatno optimizirati korištenjem algoritma sa faktorom težine iz razloga što su resursi zakupljeni na *vultr* servisu duplo jači, a opterećuje ih se jednako kao i slabije resurse.

LITERATURA

[1] "What is load balancing?," (Online), https://www.nginx.com/resources/glossary/load-balancing/
pristupljeno: 3. studenog 2016.

[2] "*About the name", Official Ubuntu Documentation. Canonical., (Online) https://en.wikipedia.org/wiki/Ubuntu_%28operating_system%29*
*pristupljen*o: 16. svibnja 2016.

[3] "About Ubuntu. Ubuntu and Debian", Canonical Ltd., (Online) https://hr.wikipedia.org/wiki/Ubuntu,
pristupljeno: 16. svibnja 2016.

[4] "New Ubuntu Foundation Announced", Canonical Ltd., (Online) , https://hr.wikipedia.org/wiki/Ubuntu,
pristupljeno: 31. svibnja 2016.

[5] *"[ruby-talk:00382] Re: history of ruby"* , (Online),
https://en.wikipedia.org/wiki/Ruby_(programming_language)
pristupljeno: 6. lipnja 2016.

[6] "About Ruby" , (Online),
https://www.ruby-lang.org/en/about/
pristupljeno: 6. lipnja 2016.

[7] "Most popular tehnologys", StackOverflow, (Online) *http://stackoverflow.com/research/developer-survey-2015#tech*
pristupljeno: 6. lipnja 2016.

[8] "What is Ruby on Rails", Daniel Kehoe, 2013, (Online)
http://railsapps.github.io/what-is-ruby-rails.html ,
pristupljeno: 8. lipnja 2016.

[9] *Rails 2.3: Templates, Engines, Rack, Metal, much more!, Hansson, David , 2009* , (Online),
https://en.wikipedia.org/wiki/Ruby_on_Rails
pristupljeno: 8. lipnja 2016.

[10] "Ruby on Rails 3.0 Release Notes", (Online), *http://edgeguides.rubyonrails.org/3_0_release_notes.html*

https://en.wikipedia.org/wiki/Ruby_on_Rails
pristupljeno: 8. lipnja 2016.
[11] "Ruby on rails release notes",(Online), *http://guides.rubyonrails.org/3_1_release_notes.html* ,
pristupljeno: 8. lipnja 2016.
[12] "Rails 3.2.x is now compatible with Ruby 2.0", GitHub,2013.,(Online)
Rails 3.2.x is now compatible with Ruby 2.0.0 by sikachu · Pull Request #9406 · rails/rails · GitHub ,
pristupljeno: 8. lipnja 2016.
[13] "Rails 4.0: Final Version released!", 2013., (Online)*http://weblog.rubyonrails.org/2013/6/25/Rails-4-0-final/* ,
pristupljeno: 8. lipnja 2016.
[14] "Rails 4.1.0: Spring, Variants, Enums, Mailer previws, secrets.yml", 2014., (Online),*Rails 4.1.0: Spring, Variants, Enums, Mailer previews, secrets.yml* ,
pristupljeno: 8. lipnja 2016.
[15] *https://www.codeproject.com/KB/applications/575551/mvc.png*,
pristupljeno: 7.prosinac 2016.
[16]*Rails 4.2: Active Job, Asynchronous Mails, Adequate Record, Web Console, Foreign Keys*
pristupljeno: 8. lipnja 2016.
[17] "Welcome to the NGINX Wiki's documentation", (Online), *https://www.nginx.com/resources/wiki/*#
pristupljeno: 10. lipnja 2016.
[18] "Beginner's Guide", (Online),
http://nginx.org/en/docs/beginners_guide.html
pristupljeno: 14. lipnja 2016.
[19] *https://www.phusionpassenger.com/library/walkthroughs/basics/ruby/stack-84d411be.png*
pristupljeno: 3. studenog 2016.
[20] Basics for Ruby developres,(Online), *https://www.phusionpassenger.com/library/walkthroughs/basics/ruby/fundamental_concepts.html* ,

pritupljeno: 3. studenog 2016.
[21]"What is HyperText",(Online),
https://www.w3.org/WhatIs.html ,
pristupljeno: 4. studenog 2016.
[22] *http://image.slidesharecdn.com/http-141014082957-conversion-gate02/95/http-29-638.jpg?cb=1413275552* ,
pristupljeno: 4. studenog 2016.
[23]"Hypertext transfer protocol",(Online),
https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#cite_note-20 ,
pristupljeno: 4. studenog 2016.
[24] "What is load balancing?," (Online),
https://www.nginx.com/resources/glossary/load-balancing/
[25] "Using ngix as HTTP load balancer", (Online),
http://nginx.org/en/docs/http/load_balancing.html ,
pristupljeno: 7. studenog 2016.
[26] "HAProxy",2001.,(Online),
https://en.wikipedia.org/wiki/HAProxy ,
pristupljeno: 7. studenog 2016.
[27] Comparing Load Balancing Options: Nginx vs. HAProxy ,(Online),
http://www.mervine.net/comparing-load-balancing ,
pristupljeno: 8. studenog 2016.
[28] *https://assets.digitalocean.com/articles/Rails_Passenger_Nginx/2.png* ,
pristupljeno: 15. studenog 2016.
[29] "Initial Server Setup with Ubuntu 14.04 ", Justin Ellingwood, 2014, (Online)
https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-14-04
pristupljeno: 15. studenog 2016.

SAŽETAK

Na početku ovog diplomskog rada postavljena je pretpostavka kako balansiranje opterećenja povećava performanse web aplikacije i optimizira iskorištenje poslužiteljskih resursa. Kako bi se to pokazalo identična aplikacija postavljena je na poslužitelj i to sa korištenjem balansiranja opterećenja i bez. Da bi aplikacija bila dostupa korisnicima preko mreže potrebno je zakupiti poslužiteljske resurse i instalirati cijeli programski stog. Pošto je aplikacija pisana u *Ruby on Rails* programskom jeziku programski stog se sastoji od *Linux Ubuntu 16.01. LTS* operacijskog sustava, *Phusion Passenger* aplikacijskog poslužitelja i *NGINX* web poslužitelja. Obje aplikacije počivaju na istom programskom stogu, samo što je u jednoj verziji *NGINX* konfiguriran na način da vrši balansiranje opterećenja. Nakon provedenih testova koristeći program *Apache Benchmark* pokazano je da aplikacija koja koristi balansiranje opterećenja brže odgovara na isti broj zahtjeva te samim time optimizira korištenje raspoložive infrastrukture.

Ključne riječi : balansiranje opterećenja, web poslužitelj, aplikacijski poslužitelj, *Ruby on Rails*, *Linux Ubuntu*

ABSTRACT

The task of this thesis was to prove that load balancing increases performances of a web aplication. To prove that two equal applications were used one with use of load balancing other without. To make application available on Internet application stack must be installed. But first we have to buy server resources. The program stack consist of *Linux Ubuntu 16.01 LTS*, *Phusion Passenger* application server, *NGINX* web server. Both applications were deployed on the same application stack, except on one application *NGINX* was configured to perform load balancing. After testing the both applications it was proved that load balancing increases application performance and optimizes usage of given infrastructure.

Key words : load balancing, web server, application server, *Ruby on Rails*, *Linux Ubuntu*

ŽIVOTOPIS

Josip krušec, rođen 26.srpanja 1990. godine u Ljubljani, sin Zvonimira i Stelle. Pohađao osnovnu školu "I. G. Kovačića" u Đakovu te nakon završetka upisao je opću gimnaziju "A. G. Matoš" u Đakovu. Tijekom školovanja razvio je afinitete prema prirodnim predmetima, najviše prema matematici i fizici. Nakon završene srednje škole upisuje Elektrotehnički fakultet u Osijeku, preddiplomski studij smjer računarstvo. Po završetsku preddiplomskog studija upisuje diplomski studij na istome fakultetu, smjer procesno računarstvo.

Potpis : _____

PRILOZI

P.1. Detaljniji pregled rezultata mjerenja bez balansiranja opterećenja

Poslužiteljska programska podrška :	nginx/1.10.1
Poslužiteljski *hostname* :	krusec.levara.net
Poslužiteljski port :	80
Put do dokumenta :	/kontakt/
Dužina dokumenta :	1 2151 b
Razina istovremenosti :	10
Vrijeme trajanja testa :	19.383 s
Zaključeni zahtjevi :	500
Neuspješni zahtjevi :	0
Preneseni promet :	6 487 500 b
HTML promet :	6 075 500 b
Zahtjeva po sekundi :	25.8
Vrijeme po zahtjevu :	387.654 ms (prosjek)
Vrijeme po zahtjevu :	38.765 ms (prosjek za simultane zahtjeve)
Brzina prijenosa :	326.86 Kb/s

Tablica P1.1. Općeniti rezultati testa za stranicu http://krusec.levara.net/kontakt/

	min	prosječni	[+/- sd]	median	max
Povezani :	48	95	189.6	57	1 055
U procesiranju :	77	283	148.9	244	820
Na čekanju :	69	267	147.7	223	793
Ukupno :	130	378	230.8	310	1 590

Tablica P.1.2. Vremena konekcija [ms] za stranicu http://krusec.levara.net/kontakt/

50%	310
66%	390

75%	446
80%	482
90%	597
95%	799
98%	1 249
99%	1 276
100%	1 590 (najdulji zahtjev)

Tablica P.1.3. Postoci zahtjeva posluženih u određenom vremenu [ms] za stranicu http://krusec.levara.net/kontakt/

Poslužiteljska programska podrška :	nginx/1.10.1
Poslužiteljski *hostname* :	krusec.levara.net
Poslužiteljski port :	80
Put do dokumenta :	/
Dužina dokumenta :	17 909
Razina istovremenosti :	10
Vrijeme trajanja testa :	17.747
Zaključeni zahtjevi :	500
Neuspješni zahtjevi :	0
Preneseni promet :	9 366 500
HTML promet :	8 954 500
Zahtjeva po sekundi :	28.17
Vrijeme po zahtjevu :	354.938 ms (prosjek)
Vrijeme po zahtjevu :	35.494 ms (prosjek za simultane zahtjeve)
Brzina prijenosa :	515.41 Kb/s

Tablica P.1.4. Općeniti rezultati testa za stranicu http://krusec.levara.net/

	min	prosječni	[+/- sd]	median	max
Povezani :	48	73	124.9	56	1 059
U procesiranju	127	278	96.2	255	669

:					
Na čekanju :	67	210	97.2	183	613
Ukupno :	185	351	163.1	313	1 154

Tablica P.1.5. Vremena konekcija [ms] za stranicu http://krusec.levara.net/

50%	313
66%	350
75%	383
80%	413
90%	485
95%	538
98%	716
99%	1 349
100%	1 554 (najdulji zahtjev)

Tablica P.1.6. Postoci zahtjeva posluženih u određenom vremenu [ms] za stranicu http://krusec.levara.net/

Poslužiteljska programska podrška :	nginx/1.10.1
Poslužiteljski *hostname* :	krusec.levara.net
Poslužiteljski port :	80
Put do dokumenta :	/o-udruzi
Dužina dokumenta :	19 530 b
Razina istovremenosti :	10
Vrijeme trajanja testa :	18.017
Zaključeni zahtjevi :	500
Neuspješni zahtjevi :	0
Preneseni promet :	10 177 000 b
HTML promet :	9 765 000 b
Zahtjeva po sekundi :	27.75
Vrijeme po zahtjevu :	360.335 ms (prosjek)
Vrijeme po zahtjevu :	36.034 ms (prosjek za simultane zahtjeve)
Brzina prijenosa :	551.62 Kb/sec

Tablica P.1.7. Općeniti rezultati testa za stranicu http://krusec.levara.net/o-udruzi/

	min	prosječni	[+/- sd]	median	max
Povezani :	49	114	276.7	58	3059
U procesiranju :	126	242	106.6	213	646
Na čekanju :	65	176	104.9	143	573
Ukupno :	179	356	300.2	273	3219

Tablica P.1.8. Vremena konekcija [ms] za stranicu http://krusec.levara.net/o-udruzi/

50%	273
66%	320
75%	358
80%	395
90%	510
95%	708
98%	1 381
99%	1 456
100%	3 219

Tablica P.1.9. Postoci zahtjeva posluženih u određenom vremenu [ms] za stranicu http://krusec.levara.net/o-udruzi/

Poslužiteljska programska podrška :	nginx/1.10.1
Poslužiteljski *hostname* :	krusec.levara.net
Poslužiteljski port :	80
Put do dokumenta :	/statut/
Dužina dokumenta :	37 517
Razina istovremenosti :	10
Vrijeme trajanja testa :	23.762

Zaključeni zahtjevi :	500
Neuspješni zahtjevi :	0
Preneseni promet :	19 170 500 b
HTML promet :	18 758 500 b
Zahtjeva po sekundi :	21.04
Vrijeme po zahtjevu :	475.240 ms (prosjek)
Vrijeme po zahtjevu :	47.524 ms (prosjek istovremenih zahtjeva)
Brzina prijenosa :	787.86 Kb/s

Tablica P.1.10. Općeniti rezultati testa za stranicu http://krusec.levara.net/statut/

	min	prosječni	[+/- sd]	median	max
Povezani :	48	106	208.9	59	1 091
U procesiranju :	171	364	141.3	331	1 197
Na čekanju :	223	470	248	401	1 681
Ukupno :	223	470	248.9	401	1 681

Tablica P.1.11. Vremena konekcija [ms] za stranicu http://krusec.levara.net/statut/

50%	401
66%	477
75%	521
80%	544
90%	694
95%	956
98%	1 396
99%	1 610
100%	1 681

Tablica P.1.12. Postoci zahtjeva posluženih u određenom vremenu [ms] za stranicu http://krusec.levara.net/statut/

P.2. Detaljniji pregled rezultata mjerenja s balansiranjem opterećenja

Poslužiteljska programska podrška :	nginx/1.10.1
Poslužiteljski *hostname* :	krusec.levara.net
Poslužiteljski port :	80
Put do dokumenta :	/kontakt/
Dužina dokumenta :	12 151 b
Razina istovremenosti :	10
Vrijeme trajanja testa :	12.640 s
Zaključeni zahtjevi :	500
Neuspješni zahtjevi :	0
Preneseni promet :	6 474 000 b
HTML promet :	6 075 500 b
Zahtjeva po sekundi :	39.56
Vrijeme po zahtjevu :	252.809 ms (prosjek)
Vrijeme po zahtjevu :	25.281 ms (prosjek istovremenih zahtjeva)
Brzina prijenosa :	500.16 Kb/s

Tablica P.2.1. Općeniti rezultati testa za stranicu http://krusec.levara.net/kontakt/

	min	prosječni	[+/- sd]	median	max
Povezani :	49	57	4.4	56	76
U procesiranju :	74	189	169.9	115	1 263
Na čekanju :	67	181	170.0	107	1 256
Ukupno :	126	246	169.8	174	1 326

Tablica P.2.2. Vremena konekcija [ms] za stranicu http://krusec.levara.net/kontakt/

50%	174
66%	242
75%	279

80%	296
90%	372
95%	444
98%	1075
99%	1 214
100%	1 326

Tablica P.2.3. Postoci zahtjeva posluženih u određenom vremenu [ms] za stranicu http://krusec.levara.net/kontakt/

Poslužiteljska programska podrška :	Nginx/1.10.1
Poslužiteljski *hostname* :	krusec.levara.net
Poslužiteljski port :	80
Put do dokumenta :	/
Dužina dokumenta :	17 911 b
Razina istovremenosti :	10
Vrijeme trajanja testa :	13.440 s
Zaključeni zahtjevi :	500
Neuspješni zahtjevi :	250
Preneseni promet :	9 356 000 b
HTML promet :	8 957 500 b
Zahtjeva po sekundi :	37.20
Vrijeme po zahtjevu :	268.810 ms (prosjek)
Vrijeme po zahtjevu :	26.881 (prosjek istovremenih zahtjeva)
Brzina prijenosa :	679.79

Tablica P.2.4. Općeniti rezultati testa za stranicu http://krusec.levara.net/

	min	prosječni	[+/- sd]	median	max
Povezani :	49	91	171.9	59	1 107
U procesiranju :	117	176	63.4	158	559
Na čekanju :	64	114	60.1	97	503
Ukupno :	168	267	185.5	218	1 606

Tablica P.2.5. Vremena konekcija [ms] za stranicu http://krusec.levara.net/

50%	218
66%	229
75%	238
80%	253
90%	351
95%	474
98%	1 209
99%	1 235
100%	1 606

Tablica P.2.6. Postoci zahtjeva posluženih u određenom vremenu [ms] za stranicu http://krusec.levara.net/

Poslužiteljska programska podrška :	Nginx/1.10.1
Poslužiteljski *hostname* :	krusec.levara.net
Poslužiteljski port :	80
Put do dokumenta :	/o-udruzi/
Dužina dokumenta :	19 530 b
Razina istovremenosti :	10
Vrijeme trajanja testa :	14.630 s
Zaključeni zahtjevi :	500
Neuspješni zahtjevi :	0
Preneseni promet :	10 163 500 b
HTML promet :	9 765 000
Zahtjeva po sekundi :	34.18
Vrijeme po zahtjevu :	292.606 ms (prosjek)
Vrijeme po zahtjevu :	29.261 ms (prosjek istovremenih zahtjeva)
Brzina prijenosa :	678.41 Kb/s

Tablica P.2.7. Općeniti rezultati testa za stranicu http://krusec.levara.net/o-udruzi/

	min	prosječni	[+/- sd]	median	max
Povezani :	49	111	219.2	60	1 183
U procesiranju :	118	177	84.8	155	707
Na čekanju :	64	112	82.2	94	651
Ukupno :	169	287	244.2	215	1 726

Tablica P.2.8.Vremena konekcija [ms] za stranicu http://krusec.levara.net/o-udruzi/

50%	215
66%	224
75%	232
80%	244
90%	448
95%	889
98%	1 272
99%	1 361
100%	1 726

Tablica P.2.9. Postoci zahtjeva posluženih u određenom vremenu [ms] za stranicu http://krusec.levara.net/o-udruzi/

Poslužiteljska programska podrška :	nginx/1.10.1
Poslužiteljski *hostname* :	krusec.levara.net
Poslužiteljski port :	80
Put do dokumenta :	/statut/
Dužina dokumenta :	37 517 b
Razina istovremenosti :	10
Vrijeme trajanja testa :	20.774
Zaključeni zahtjevi :	500
Neuspješni zahtjevi :	0
Preneseni promet :	19 157 000
HTML promet :	18 758 500
Zahtjeva po sekundi :	24.07

Vrijeme po zahtjevu :	415.472 ms (prosjek)
Vrijeme po zahtjevu :	41.547 ms (prosjek istovremenih zahtjeva)
Brzina prijenosa :	90057 Kb/s

Tablica P.2.10. Općeniti rezultati testa za stranicu http://krusec.levara.net/statut/

	min	prosječni	[+/- sd]	median	max
Povezani :	49	136	199.5	93	1 122
U procesiranju :	137	276	110.8	264	1 000
Na čekanju :	64	153	92.3	131	766
Ukupno :	191	412	229.1	394	1 592

Tablica P.2.11. Vremena konekcija [ms] za stranicu http://krusec.levara.net/statut/

50%	394
66%	422
75%	433
80%	445
90%	534
95%	908
98%	1 278
99%	1 438
100%	1 592

Tablica P.2.12. Postoci zahtjeva posluženih u određenom vremenu [ms] za stranicu http://krusec.levara.net/statut/