Analiza zmogljivosti oblačnih in strežniških storitev

Uredil prof. dr. Miha Mraz

Maj 2016

Kazalo

Predgovor

Pri?ujo?e delo je razdeljeno v devet poglavij, ki predstavljajo razli?ne analize zmogljivosti nekaterih obla?nih izvedenk ra?unalni?kih sistemov in njihovih storitev. Avtorji posameznih poglavij so slu?atelji predmeta *Zanesljivost in zmogljivost ra?unalni?kih sistemov*, ki se je v ?tud.letu 2015/2016 predaval na 1. stopnji univerzitetnega ?tudija ra?unalni?tva in informatike na Fakulteti za ra?unalni?tvo in informatiko Univerze v Ljubljani. Vsem ?tudentom se zahvaljujem za izkazani trud, ki so ga vlo?ili v svoje prispevke.

prof. dr. Miha Mraz, Ljubljana, v maju 2016

Poglavje 1

Zmogljivostna analiza Wiki aplikacije

Jan Robas, Žiga Pirih, Rok Oblak

1.1 Opis problema

Vsled zmogljivostne analize smo ustvarili Wiki aplikacijo, kjer ima vsak uporabnik dostop do štirih osnovnih operacij, in sicer dodajanje strani, brisanje strani, pregled poljubne strani in urejanje poljubne strani. Pri pregledovanju uporabnik zahteva določeno vsebino strani, medtem ko pri urejanju posamezno vsebino prepiše z novo. Uporabnik lahko išče po naslovih strani. Vsaka operacija terja vhodno-izhodne operacije na bazi podatkov.

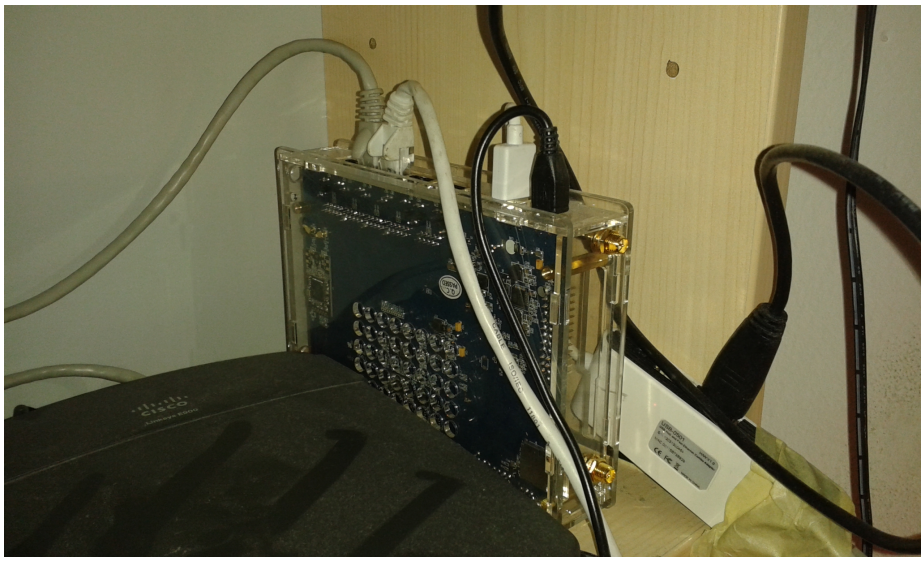
V ta namen smo postavili strežnik, na katerem teče spletna aplikacija, ki obdeluje zahteve in upravlja z dokumentno usmerjeno podatkovno bazo. Za odjemalca smo napisali ločeno aplikacijo, ki ustvarja zahtevke za primere različnih situacij. Tretji člen v shemi je predpomnilniški strežnik oziroma predpomnilniška storitev [1] (angl. *caching service*), ki predpomni zahteve odjemalca in v primeru večjega števila enakih zahtevkov ne obremenjuje po nepotrebnem aplikacijskega strežnika. Testne scenarije smo preizkusili z in brez omenjenega predpomnilniškega strežnika in s tem ovrednotili uporabnost le-tega.

1.2 Rešitev problema

V tem razdelku predstavimo uporabljene programske tehnologije in infrastrukturo, ki smo jih uporabili.

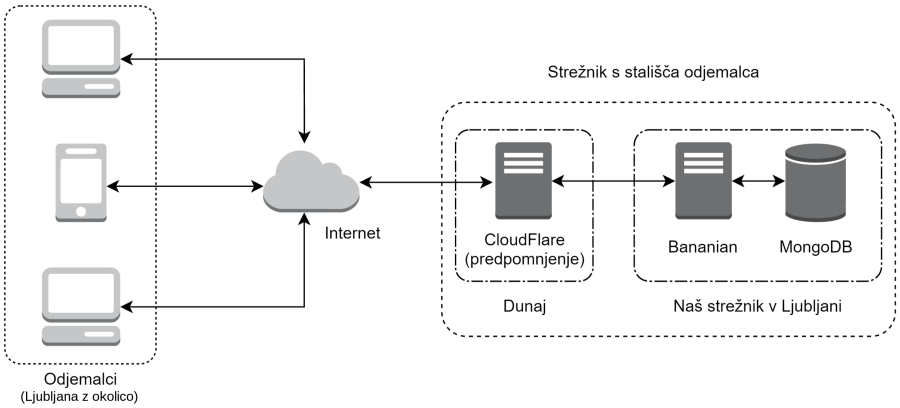
1.2.1 Izbira strežnika in ponudnika oblačnih storitev

Aplikacija teče na lastnem strežniku. Strežnik je Banana Pi R1 [2]. Ima dvojedrni procesor ARM Cortex-A7 s taktom 1 GHz ter 1 GB pomnilnika. Na njem teče operacijski sistem Bananian (različica Debiana, prilagojena za računalnike Banana Pi). Kot spletni strežnik uporabljamo nginx nginx, s katerim prihajajoči promet preusmerimo iz vrat 80 na tista vrata, ki jih uporablja aplikacija. Strežnik se nahaja v enem izmed študentskih domov v Ljubljani. Hitrost internetne povezave je na papirju 100 Mb/s v vsako stran, v praksi pa je bila največja izmerjena hitrost prenosa v smeri proti internetu zgolj med 45 - 65 Mb/s. Količina prenosa, ki ga lahko opravimo od strežnika proti internetu, je omejena s 100 GB na teden. Od tega ostale aktivnosti na strežniku zahtevajo 10 - 20 GB, vendar je popačenje rezultatov ob zadostnih ponavljanjih testov zanemarljivo.



Slika 1.1: Naš strežnik na lokaciji v Ljubljani.

Za predpomnenje (angl. *caching*) smo uporabili storitev CloudFlare. CloudFlare smo izbrali, ker je brezplačen in ker nam lahko izpiše grobe podatke o številu predpomnjenih in nepredpomnjenih zahtev. CloudFlare izvaja predpomnjenje na svojih strežnikih, ki so optimalno blizu tako uporabniku kot strežniku. Nam najbližji strežnik ponudnika CloudFlare, preko katerega gre ves promet do našega strežnika (brez izjem), je na Dunaju. Za primerjavo smo uvedli tudi dostopno točko, kjer gre promet preko CloudFlare strežnikov, brez da bi se izvajalo predpomnjenje in dostopno točko, ki gre neposredno do našega strežnika.



Slika 1.2: Diagram strežniške arhitekture naše aplikacije.

1.2.2 Izbira tehnologij

Node.js

Strežniška aplikacija je napisana v jeziku JavaScript in teče v okolju Node.js. Node.js omogoča asinhrone vhodno-izhodne operacije s pomočjo dogodkovno gnanega programiranja (angl. *event-driven programming* [3]) in posledično ponuja visoko propustnost in skalabilnost realno-časovnih spletnih aplikacij. Zaradi svoje razširjenosti in velike izbire knjižnic omogoča preprosto razvijanje in nameščanje na strežnik.

MongoDB

Za shranjevanje podatkov smo uporabili dokumentno usmerjeno bazo MongoDB. Vsak zapis v bazi je shranjen kot objekt v obliki, podobni JSON (*JavaScript Object Notation*). Wiki stran bi na primer lahko vsebovala tako kraje kot ljudi. Ljudje bi imeli na primer podatek o datumu rojstva, kraji pa bi imeli podatek o nadmorski višini. Pri dokumentno usmerjeni bazi nam ni potrebno vnaprej določiti strukture, tako kot moramo to narediti pri uporabi relacijske baze. To nam omogoča veliko prostora za razširitve podatkov. Dodatna prednost, predvsem v našem primeru pa je, da je baza MongoDB zelo dobro podprta v Node.js.

1.2.3 Implementacija aplikacije

Programski vmesnik deluje po principih REST [4]. Ti principi so smernice za arhitekturo spletnih aplikacij. Po principu REST uporabljamo standardne HTTP metode (GET, POST, DELETE, itd.), ki predstavljajo operacije kot so branje, dodajanje in brisanje. Vsaka stran ima svoj URL naslov v obliki */wiki/ime_strani*, kjer je *ime_strani* nadomestna beseda (angl. *placeholder*)

za poljubno ime strani. Vmesnik je brezstanjski (angl. *stateless*), kar pomeni da ne vodimo evidence o odjemalcih med zahtevki. Vsak zahtevek vsebuje vse potrebne informacije za izvršitev željene operacije. V odgovor strežnika prav tako dodamo še informacije za predpomnilniški strežnik (največja starost strani). Vse to nam je zelo olajšalo testiranje.

Naš vmesnik podpira naslednje operacije:

- dodajanje / spreminjanje strani (POST zahtevek z vsebino na URL že obstoječe ali nove strani s poljubnim imenom),
- brisanje strani (DELETE zahtevek na URL strani),
- pregled strani (GET zahtevek na URL strani).

Vsaka od teh funkcij terja vhodno-izhodno operacijo na bazi podatkov.

1.2.4 Analiziranje zmogljivosti aplikacije

Analiziranje zmogljivosti aplikacije smo izvedli iz dveh vidikov:

- odjemalčeva stran (na različnih odjemalcih),
- strežnikova stran.

Za vnaprej določene scenarije smo generirali testne nabore zahtevkov, v katerih so nastopale različne operacije vmesnika z različnimi parametri. Beležili smo čas odgovora na posamezen zahtevek, povprečni čas odgovora na zahtevek in skupni čas vseh zahtevkov.

S primerjanjem testnega scenarija in analizo obremenjenosti strežnika ter odzivnega časa za odgovor na zahtevek odjemalca smo naredili zaključke o samem delovanju celotne sheme ter o učinkovitosti predpomnilniškega strežnika.

1.2.5 Testiranje

Zavoljo testiranja smo pripravili skripto, ki prejme sledeče parametre:

- URL - dostopna točka, na kateri teče naša Wiki aplikacija,
- število različnih strani, med katerimi skripta naključno izbira,
- število zahtevkov, ki jih skripta izvede,
- čas čakanja med posameznimi zahtevki (v sekundah; kot parameter lahko nastopa tudi necelo število),
- za kakšne zahtevke gre: GET ali POST,
- dolžina vsebine v znakih (v primeru POST zahtevkov, ko urejamo oziroma dodajamo nove strani).

Parameter za spanje med pošiljanjem posameznih zahtevkov je uporaben v primeru, ko hočemo dati strežniku nekaj časa med posameznimi zahtevki, sicer pa je lahko nastavljen na 0 in se zahtevki izvajajo v normalnem, zaporednem načinu.

Skripta vsak zahtevek naslovi na eno izmed strani, katerih število je določeno s prvim argumentom. S tem je mogoče določati porazdelitev zahtevkov med stranmi - veliko razpoložljivih strani in majhno število zahtevkov pomeni veliko verjetnost, da je posamezen zahtevek vedno naslovljen na unikatno stran, medtem ko v obratni situaciji več zahtevkov pade na isto stran. V tem primeru lahko vidimo učinke predpomnjenja v primeru GET zahtevkov.

Na strežniški aplikaciji smo implementirali tudi števec zahtevkov. Z dostopom na URL *wiki/stevec* pridobimo število zahtevkov pred dostopom in števec ponastavimo na 0. Tako smo lahko prešteli, koliko zahtevkov je bilo odgovorjenih s strani predpomnilniškega strežnika. S tem smo dobili dodatno metriko za učinkovitost predpomnjenja - delež zahtevkov, ki je bil predpomnjen oziroma delež zahtevkov, ki jih je moral servirati naš aplikacijski strežnik.

Pripravili smo dve dostopni točki - eno, ki gre preko predpomnilnika in eno, ki predpomnjenja nima. Poleg primerjave učinkovitosti predpomnjenja pri različnih testnih scenarijih smo tako lahko primerjali tudi testni scenarij brez predpomnilniškega strežnika.

1.2.6 Testni scenariji

Ker aplikacija predstavlja Wiki stran, kjer lahko vsak uporabnik pregleduje, spreminja in briše posamezne strani oziroma dokumente v bazi, smo definirali nekaj testnih scenarijev, ki posnemajo realno uporabo take strani. Testni scenariji so sestavljeni iz različnega tipa in števila zahtevkov ter različne velikosti podatkov (velikosti vsebin strani), ki se prenašajo z zahtevki. Tako smo spreminjali velikost bremena in s tem izvedli t. i. *test to pass* in *test to fail*. Vsled tega smo izbrali smiselno zgornjo mejo odziva strežnika na zahtevek. Če operacija traja dlje časa od naše zgornje meje, določimo da je test neuspešen, saj s predolgim čakanjem izgubljamo uporabnike [5]. Testni scenariji so potekali po naslednjem vzorcu:

- Zaporedje zahtevkov za pregled strani (GET): Strežniku posredujemo n zahtevkov za pregled strani, kjer vsak zahtevek naslavlja eno od m strani. Parametra n in m spreminjamo in tako izvedemo različne vrste testov.
- Zaporedje zahtevkov za spreminjanje strani (POST): Strežniku posredujemo n zahtevkov za spreminjanje strani, kjer ima vsak zahtevek naključno vsebino dolžine l in naslavlja eno od m strani. Parametre n, l in m spreminjamo.
- Mešano zaporedje: Sestavljeno je iz zgornjih dveh zaporedij.

Pri testiranju smo obravnavali več različnih scenarijev, kar smo dosegli s spreminjanjem spremenljivk. Spreminjali smo tip zahtevkov - POST in GET,

pri POST zahtevkih pa tudi količino informacije, ki se je prenesla na strežnik. Poskušali smo s primerno velikostjo Wiki strani, nekje med nekaj kilobajti do nekaj deset kilobajtov, z namenom da pride strežniško predpomnjenje do izraza. Za podatke se jemljejo naključni nizi iz */dev/random*, saj njihova vsebina ni pomembna. Spreminjali smo tudi število obravnavanih strani. Kot hipotezo smo vzeli, da je predpomnjenje bolj učinkovito pri velikem številu zahtevkov na majhno število strani.

Strežnik smo preizkušali tako, da smo ga iz ene strani čim bolj obremenili z velikim številom hkratnih zahtevkov, medtem pa iz druge strani izvajali zahtevke in merili njihov povprečni čas v primerjavi s situacijo, kjer strežnik ni obremenjen iz druge strani.

1.3 Metrike

Pri testnih scenarijih smo obravnavali različne metrke, s katerimi smo določali parametre testov ali pa jih merili v rezultatih testov. Glavne izmed teh metrik so naslednje:

- Število predpomnjenih in nepredpomnjenih strani: pri testih z večjim številom zahtevkom smo opazovali absolutno število strani oziroma zahtevkov, ki so bile predpomnjeni in jih je zato namesto aplikacijskega strežnika stregel strežnik CloudFlare.

- Delež predpomnjenih in nepredpomnjenih zahtevkov: opazovali smo deleže vseh zahtevkov, ki so bili predpomnjeni na strežniku CloudFlare od skupine več izvedenih zahtevkov v enem kosu.

- Povprečni čas posameznega zahtevka: ker zaradi velikega števila različnih dejavnikov čas za izvedbo zahtevkov niha, smo povprečili čase preko večjega števila zahtevkov in to večkrat ponovili.

- Nihanje časa za izvedbo zahtevkov v krajših časovnih in daljših časovnih intervalih: na krajših intervalih čas niha zaradi manjših okoljskih in fizičnih dejavnikov na linijah, na daljših pa ima večji vpliv sama obremenjenost linij zaradi različne prisotnosti drugih uporabnikov spleta.

- Število zaporednih zahtevkov v enem testiranju z uporabo testne skripte: s testno skripto smo pošiljali določeno število zaporednih zahtevkov, kar je vplivalo na dolžino časa, ko je bil strežnik zaposlen s streženjem.

- Število hkratno naslovljenih zahtevkov iz enega odjemalca: s skripto smo generirali tudi posamezne kose hkratno naslovljenih zahtevkov in jih zaporedno ponavljali; to je poleg dolžine časa, ki je bil porabljen za zahtevo, odgovor in prenos, določalo tudi nivo obremenitve strežnika.

- Količina prenesenih podatkov pri vsakem zahtevku: ker smo pri zahtevkih tipa POST lahko spreminjali količino naključno generirane vsebine,

ki se je zapisala na Wiki stran, smo tako lahko določali količino prenešenih podatkov. To je posledično vplivalo na količino prenešenih podatkov pri sledečih GET zahtevkih po zapisanih straneh. Pri posameznem testu smo obravnavali strani enake dolžine, tako da je bila količina prenesenih podatkov sorazmerna s količino zahtevkov. Količina prenesenih podatkov od aplikacijskega strežnika je sorazmerna s količino nepredpomnjenih zahtevkov.

- Pot zahtevkov oz. končna točka za zahtevke (angl. *endpoint*): glede na določen naslov končne točke smo določali, po kateri poti potujejo zahtevki in ali se v primeru potovanja preko strežnika CloudFlare za njih izvaja predpomnjenje. Ta naslov je sicer iz stališča uporabnika deloval enako v vseh primerih, vplival je le na čas za izvedbo zahtevkov.

1.4 Izvedba meritev

1.4.1 Veliko število zahtev za branje po majhnem številu različnih strani

Pri tem testu gre za izvedbo 1000 zahtev za branje (GET) po 100 različnih straneh. Testni scenarij je bil izveden s predpomnjenjem in brez predpomnjenja. Za namene testiranja smo vzpostavili 3 različne dostopne točke, ki kažejo na isti strežnik, njihova pot pa je različna:

- navadna: zahtevki gredo preko CloudFlare, kjer se izvaja predpomnjenje,
- nocache: zahtevki gredo preko CloudFlare, kjer pa se predpomnjenje preskoči,
- source: zahtevki ne gredo preko CloudFlare, temveč gredo neposredno do strežnika.

Hipoteza

Naša hipoteza je, da bo zaradi ponavljajočih se zahtev na veliko zahtev odgovoril predpomnilniški strežnik in s tem razbremenil aplikacijski strežnik. Strani so dolge 5000 znakov, tako da so zadosti velike, da je predpomnjenje sploh smiselno in da se pozna njegov učinek.

Meritve

Vsaka meritev predstavlja 1000 zahtev. Meritve so bile izvedene v petek zvečer, 8.4.2016 in ponovljene v soboto popoldne, 9.4.2016. V tabeli 1.2 so rezultati petkovih meritev. Število zahtevkov do aplikacijskega strežnika ni odvisno od razmer v omrežju, saj tako lokalni predpomnilnik kot CloudFlare ob drugi zahtevi za isto stran vrne predpomnjeno vsebino. Povprečen porabljen čas se je razlikoval za nekaj milisekund (največ 5 ms), vendar so razmerja, s katerimi

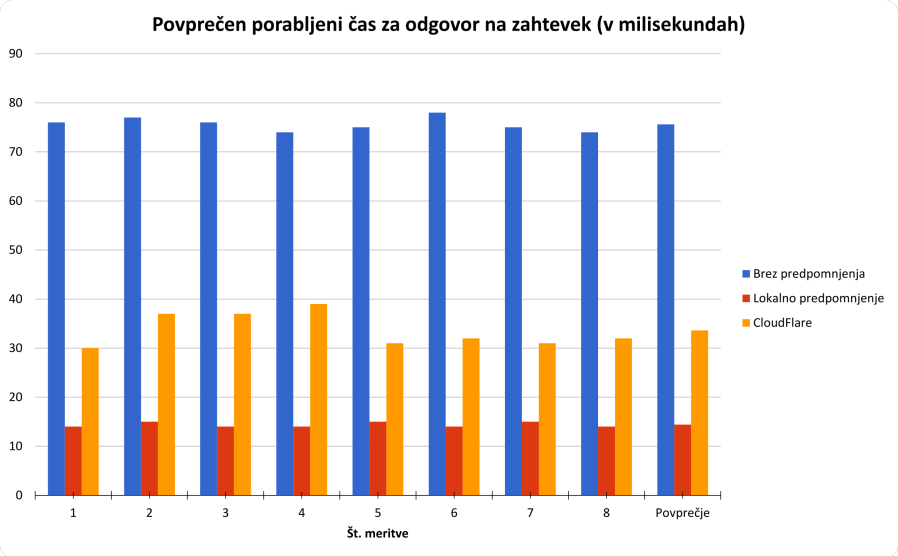
dokazujemo smiselnost uporabe spletnega predpomnilniškega strežnika, ostala podobna.

Številka meritve	1	2 - 7	8	povprečje
Brez predpomnjenja	1000	1000	1000	1000
S predpomnjenjem	100	100	100	100

Tabela 1.1: Število zahtevkov do aplikacijskega strežnika.

Številka meritve	1	2	3	4	5	6	7	8	povprečje
Brez predpomnjenja	76	77	76	74	75	78	75	74	75,6
Lokalno predpomnjenje	14	15	14	14	15	14	15	14	14,4
CloudFlare	30	37	37	39	31	32	31	32	33,6

Tabela 1.2: Povprečen porabljen čas na odgovor 1 zahtevka (v milisekundah).



Slika 1.3: Graf rezultatov prvega scenarija.

Komentarji

Kot je prikazano v tabeli 1.1, je brez predpomnjenja število zahtevkov do aplikacijskega strežnika, jasno, 1000, ob uporabi predpomnilnika pa se to število zmanjša na 100, saj pri vsakem zahtevku zahtevamo eno od 100 strani, ob ponovnem zahtevku strani pa le-to servisira predpomnilniški strežnik. Pri tem scenariju je uporaba predpomnilniškega strežnika zelo učinkovita. Aplikacijski strežnik servira le 10% zahtevanih strani.

Na tabeli 1.2 je za vsako meritev prikazan povprečni čas odgovora na zahtevek. Lokalno predpomnjenje je realizirano tako, da je na odjemalcu naložen strežnik nginx, ki je nastavljen kot predpomnilniški strežnik [6]. Povprečen čas odgovora je v primeru lokalnega predpomnjenja najkrajši, saj na večino (90%) zahtevkov odgovori program, ki je na samem odjemalcu. Ugotovili smo, da CloudFlare predpomni na enak način kot lokalni odjemalec, le da je čas odgovora primerno večji, ker se predpomnilniški strežnik v našem primeru nahaja na Dunaju.

Opisan scenarij odraža precej realno uporabo Wiki strani. Iz danih rezultatov je razvidno, da je v našem primeru uporaba spletnega predpomnilniškega strežnika zelo smiselna.

1.4.2 Majhno število zahtev za branje po velikem številu različnih strani

Pri tem testu gre za izvedbo 100 zahtev za branje (GET) po 1000 različnih straneh. Strani so velike 5 kB, beležili pa smo čas odzivanja na zahtevek. Predpomnilnik smo med poskusi pobrisali, za dostop pa smo uporabljali dve vrsti dostopov: s predpomnjenjem in brez predpomnjenja. Obe povezavi sta šli preko CloudFlare strežnikov. Test smo izvedli osemkrat. Test smo izvedli v noči iz 16. na 17. april. Izvedli smo ga osemkrat. Začeli smo ob 0:05, vsak test je trajal okoli 3 minute, med ponovitvami testa pa je pretekla točno ena ura.

Hipoteza

Ker izvajamo majhno število zahtev na velikem številu strani, predvidevamo, da ne bo kakšne razlike med tem, če uporabljamo predpomnjenje ali ne. Prav tako predvidevamo, da predpomnjenja v tem primeru skorajda sploh ne bo, saj se zahteva shrani v predpomnilnik šele po najmanj enem dostopu, verjetnost za to pa je sorazmerno majhna: v teoriji bi pričakovali tam okoli 5 dostopov do strani, do katerih smo tekom poskusa že dostopali.

Rezultati

Številka meritve	1	2	3	4	5	6	7	8	Povprečje
Brez predpomnjenja	130	128	121	130	144	133	122	117	128,12
S predpomnjenjem	121	133	133	133	127	122	129	148	130,75
Št. predpomnjenih zah.	2	1	2	3	3	2	5	4	2,75

Tabela 1.3: Tabela povprečnih časov odgovora ter število predpomnjenih zahtevkov.

Komentarji

Kot je razvidno iz tabele 1.3, so dostopni časi približno enaki, kakšnih večjih razlik pa ni. V povprečju je čas dostopa brez predpomnjenja malo manjši kot čas dostopa preko povezave s predpomnjenjem. To je rahlo presenetljivo, vendar pa je razlika zadosti majhna, da bi jo lahko razložili kot šum v podatkih.

Prav tako vidimo, da je bilo predpomnjenih dostopov nekaj, vendar pa ne toliko, kot smo pričakovali. Iz tega lahko sklepamo, da Cloudflare pri majhni količini prometa čez večje število strani ne pomni vsakega zahtevka (seveda pa je možno tudi, da smo se ušteli pri teoretičnem izračunu).

1.4.3 Test z večjo obremenitvijo do odpovedi

Pri tem testu smo iz določenih strani (odjemalcev) poskušali strežnik čim bolj obremeniti, medtem pa smo poskušali s strani tretjega odjemalca izvajati zahtevke in beležiti, kako dobro mu je strežnik še sposoben servisirati pod veliko obremenitvijo.

Primerjali smo povprečne rezultate sto zahtevkov pri različni obremenjenosti strežnika in pri dveh načinih dostopa do strežnika - najprej po neposredni povezavi, ki zaobide CloudFlare in s tem predpomnjenje, nato pa še preko CloudFlare.

Strežnik smo obremenili tako, da smo napisali in izvajali posebno skripto, ki je hkrati (tj. vzporedno v različnih podprocesih) generirala določeno število GET zahtevkov in to ponavljala za čas testiranja. Zahtevki so bili izvedeni vzporedno zato, ker bi se sicer v skripti izvajali zaporedno in bi jih strežnik izvajal sinhrono, enega za drugim, ker pa procesiranje zahtevka na strežniku traja zgolj določen čas celotnega časa izvedbe, tako ne bi imeli zagotovitve, da je strežnik obremenjen celotni čas oziroma da ima v vrsti še čakajoče, neobdelane zahtevke. Skripta je poleg tega beležila še število izvedenih zahtevkov na sekundo oziroma minuto.

Hipoteza

Hipoteza pri tem testu je, da bo strežnik do določene obremenitve iz ene strani z vidika uporabnika deloval kot nemoten, od tam naprej pa se bo obremenjenost že poznala s strani uporabnika v obliki daljšega (in eventuelno predolgega [5]) časa za izvedene zahtevke. Prav tako je za pričakovati, da bo v primeru preobremenjenosti strežnika uporabnikovo čakanje na zahtevke hitro poskočilo.

Z uporabo predpomnjenja je pričakovati, da bo strežnik precej bolj odporen na obremenitev in bo zato vpliv oremenitve na uporabnika manjši, saj se bo razmeroma majhen del zahtevkov moral dejansko izvajati na strežniku, večino pa bo stregel CloudFlare.

Meritve

Pri meritvah smo pri različnem številu hkratno izvedenih zahtevkov poleg povprečnega porabljenega časa za zahtevek uporabnika beležili še število izvedenih

zahtevkov na minuto s strani odjemalca.

Meritve so bile izvedene v petek, 15. aprila, med 17. in 20. uro. Skupno 2012 zahtevkov je za izvršitev potrebovalo 3304 sekund. Velikosti strani so znašale med 100 in 10000 znakov naključno generirane vsebine.

Hkratnih	0	5	10	15	25	40	50	60	70	80	100	150	200	250
Hkratnih/min	0	540	660	720	840	840	840	840	900	900	960	960	900	900
Čas v ms	299	415	623	639	838	1052	1385	1935	2136	2185	2253	2324	3193	3836

Tabela 1.4: Povprečen porabljen čas na odgovor 1 zahtevka (v milisekundah) brez uporabe predpomnjenja.

Hkratnih	0	5	10	15	25	40	50	60	70	80	100	150	200	250
Hkratnih/min	0	240	420	480	600	840	780	780	840	840	840	900	840	840
Čas v ms	323	320	341	352	362	417	434	439	437	477	547	519	578	565

Tabela 1.5: Povprečen porabljen čas na odgovor 1 zahtevka (v milisekundah) z uporabo predpomnjenja.

Komentarji

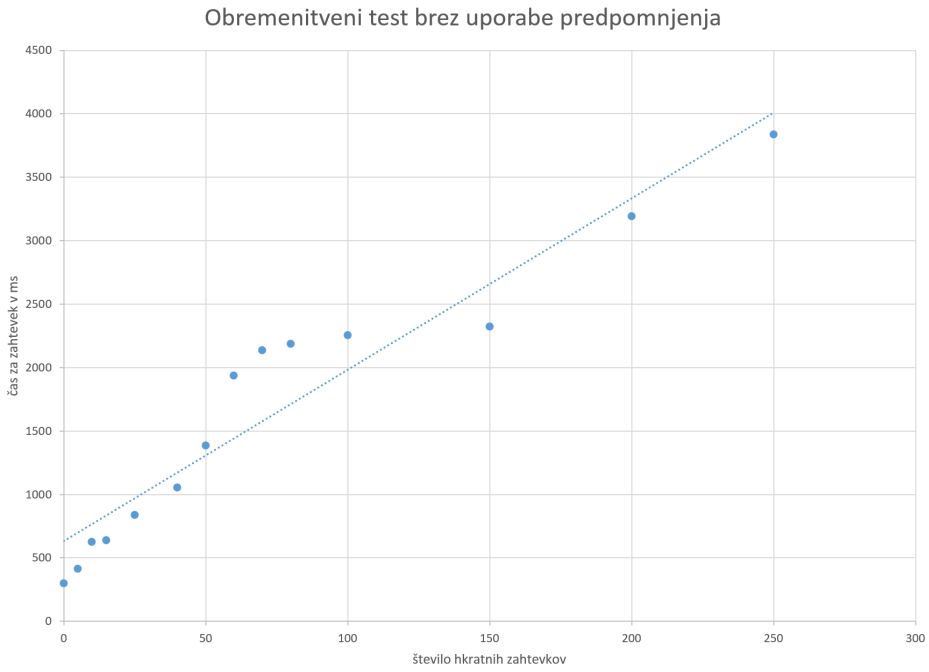
Za prvi preizkus hipoteze smo izvedli poskusen test, kjer smo strežnik preobremenili s 1000 hkratnimi zahtevki (izvedeni so bili paralelno), kar je simuliralo manjši DOS napad (angl. *denial of service*) in kot pričakovano je strežnik zares potreboval precej dlje, da se je odzval na zahtevek - namesto nekaj sto milisekund se je čas povečal na več kot 4 sekunde, kar je za uporabnika že moteče.

Za glavni test smo izvedli scenarij, kjer smo spreminjali število vzporedno izvedenih zahtevkov, začenši z 0 (uporabnik je bil edini, ki je takrat dostopal do strežnika) in vse do 250. Test smo izvedli s predpomnjenjem in brez njega. Da bi simulirali situacijo Wiki strani, smo test omejili na 1000 različnih vnosov - šlo je torej za 1000 različnih zahtevkov GET. Po vsakem testu smo izpraznili predpomnilnik, tako da so posamezni testi ostali neodvisni od prejšnjih.

V tabelah 1.4 in 1.5 je razviden vpliv števila hkratno izvedenih zahtevkov iz tretje strani na povprečni čas enega zahtevka na uporabnikovi strani. Vidno je, kako v primeru neuporabe predpomnjenja ta čas hitro zraste na za uporabnika neugodne vrednosti, medtem pa z uporabo predpomnjenja te vrednosti ostanejo sprejemljive tudi pri vseh testiranih vrednostih hkratno izvedenih zahtevkov. Pri večjih vrednostih je za pričakovati, da bo velik del zahtevkov že v predpomnilniku, zato čas za uporabnika ne bo nadalje precej rastel, kar je iz rezultatov razvidno.

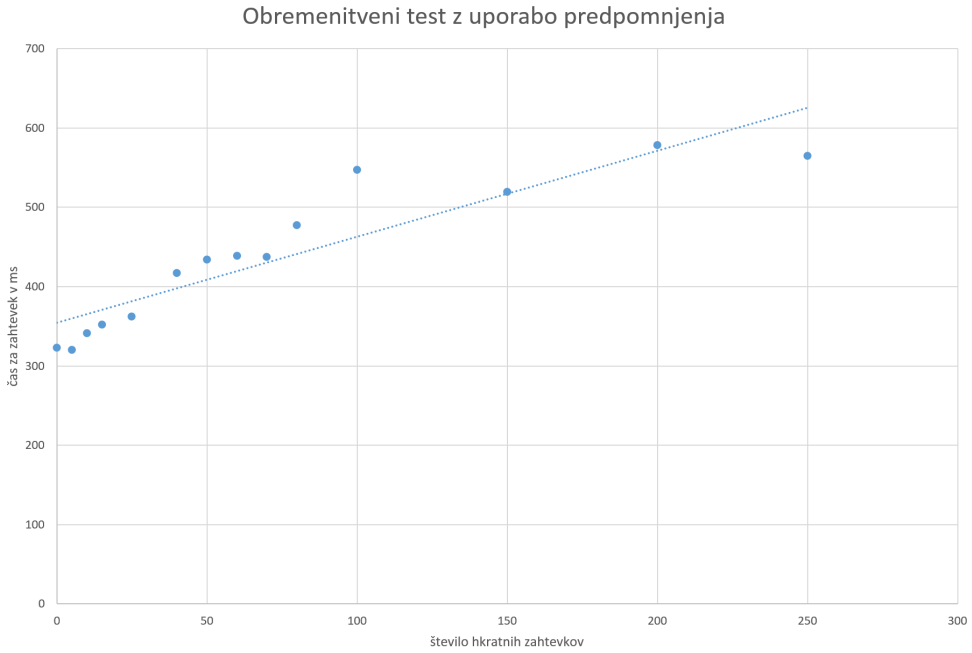
Ta test je služil tudi kot test do odpovedi (angl. *test to fail*), saj vrednosti nad nekaj sekund lahko smatramo kot odpoved sistema s perspektive ugodne uporabniške izkušnje. S predolgim čakanjem na vsebino namreč izgubljamo stranke in s tem profit, ki nam omogoča nadaljnje delovanje [5].

Opazno je torej, da ima vključitev predpomnjenja velik pozitivni učinek pri takem scenariju kljub temu, da povezava ni neposredna (v tem primeru je

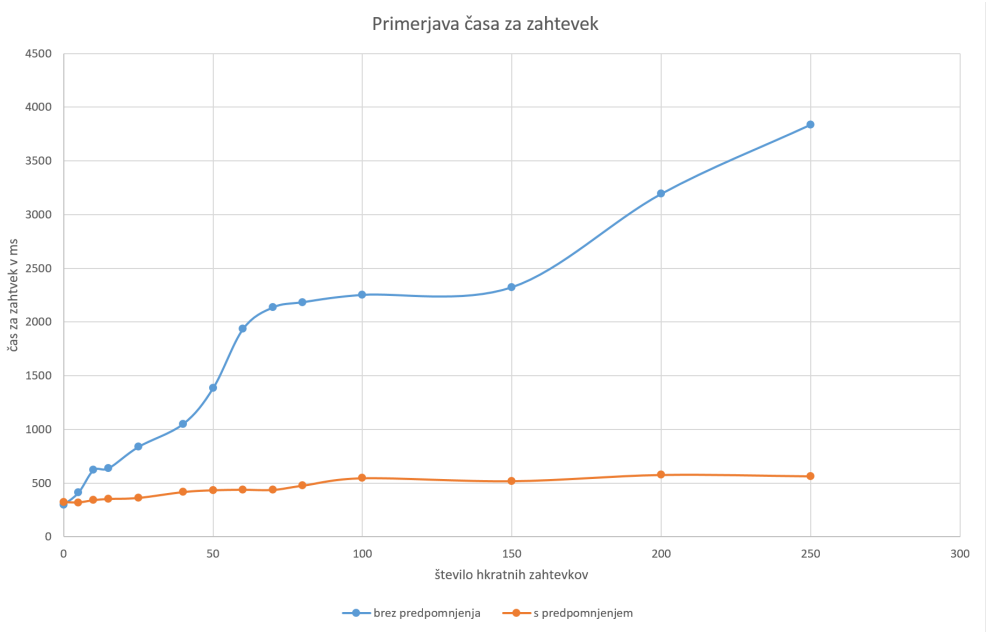


Slika 1.4: Graf rezultatov scenarija brez uporabe predpomnjenja.

neposredna povezava krajša). Za boljšo predstavo naraščanja časa za zahtevek in primerjave obeh scenarijev so vključeni grafi na slikah 1.4, 1.5 in 1.6.



Slika 1.5: Graf rezultatov scenarija z uporabo predpomnjenja.



Slika 1.6: Graf primerjave rezultatov obeh scenarijev.

1.4.4 Učinkovitost predpomnjenja pri pisanju

Pri tem testu smo merili učinkovitost predpomnjenja pri mešanih zahtevah. Izvedli smo več poskusov z različno velikostjo strani. Strani so bile velike 5, 10, 20 ali 50 kB. Vsak zahtevek je imel naključno možnost (5%), da bo spremenil stran — pri taki priložnosti smo na strežnik najprej naslovili zahtevek GET, zatem pa še zahtevek POST. Izvedli smo 10000 zahtevkov po 100 straneh. Posamezna stran po vpisu v predpomnilnik tam ostane tri minute.

Hipoteza

Hipoteza pri tem testu je, da bo zaradi naključnih POST zahtevkov predpomnilnik manj učinkovit v primerjavi s primerom, ko smo uporabljali le GET zahteve. Ker je največja starost predpomnilnika manj, kot pričakujemo, da bo test tekel, pričakujemo tudi, da bo pri večjih zahtevah predpomnilnik manj učinkovito uporabljen.

Meritve

Najprej smo zagnali meritve z mešanim prometom, zatem pa smo za primerjavo izvedli še meritve brez POST zahtevkov. Predpomnilnik smo izpraznili pred vsakim izvajanjem testa, zahteve pa so v predpomnilniku ostale le tri minute. Teste smo izvajali zaporedno, po vrsti od najmanjše velikosti strani do največje. Test smo izvedli osemkrat: za vsako velikost strani po dvakrat — enkrat z mešanimi zahtevami, enkrat pa samo z zahtevki tipa GET. Vseh osem poskusov smo izvedli enkrat brez premorov v noči iz 1. na 2. maj (nedelja, ponedeljek) med 22. in 2. uro zjutraj. Rezultati meritev so vidni v tabeli 1.6.

Velikost zahtevka		5	10	20	50
Mešani zahtevki	Št. POST zah.	471	498	504	528
	Št. GET zah. na strežniku	1044	1017	1183	1600
	Delež predpom. zahtev	89,56%	89,83%	88,17%	84,00%
Samo GET	Št. zah. na strežniku	412	447	600	791
	Delež predpom. zahtev	95,88%	95,53%	94,00%	92.09%

Tabela 1.6: Delež predpomnjenih zahtev pri mešanih zahtevah različnih velikosti. Za primerjavo smo izvedli tudi poskus, v katerem smo imeli le zahteve tipa GET.

Komentarji

Vidimo, da se ob 5% deležu POST zahtev število zahtevkov na strežniku približno podvoji. To pomeni, da je učinkovitost predpomnilnika v takih primerih manj učinkovita v primerjavi s primerom, da na strežnik pošiljamo samo GET zahteve — tako, kot smo pričakovali.

Zanimivo je, da učinkovitost predpomnilnika ne pada z faktorjem povečanja velikosti, čeprav se je trajanje testov podaljševalo približno s faktorjem velikosti.

Natančnih meritev za trajanje nismo opravili, vendar pa smo trajanje približno ocenili na podlagi opazovanja ter zahtevkov, ki so bili v določenem časovnem obdobju izmerjeni na CloudFlare. Čeprav se je test z zahtevki velikosti 50kB izvajal okoli ene ure, je bilo zahtevkov na strežniku le 791, pričakovali pa bi jih tam okoli 2000 (število strani * (čas izvajanja / dolžina veljavnosti strani v predpomnilniku)). Medtem pa je pri testu z najmanjšo velikostjo strani nepredpomnjenih zahtevkov vsaj toliko (ali pa celo več), kot bi jih glede na dane pogoje pričakovali. Iz tega lahko sklepamo dve stvari: prva je, da Cloudflare raje predpomni večje strani, druga pa, da nastavitev za največjo starost strani v predpomnilniku pri tako majhnih vrednostih ne deluje najbolj zanesljivo. To pomeni, da se lahko (če je največja dovoljena starost majhna) stran v predpomnilniku nahaja dlje časa, kot smo določili.

A če od deleža nepomnjenih zahtev pri poskusu z mešanimi zahtevami odštejemo delež nepomnjenih zahtev pri poskusu samo z zahtevki tipa GET (torej, odstranimo vpliv poteka predpomnilnika, tako da nam ostane samo vpliv invalidacije predpomnilnika zaradi POST zahtev) ugotovimo, da je pri večjih straneh vpliv invalidacije na predpomnjenje večji kot pri manjših straneh. Na prvi pogled se torej zdi, da predpostavka o tem, da Cloudflare raje pomni večje strani, ne drži. To neskladje se lahko v meri razloži z nesimetrično internetno povezavo: ker je bil prenos proti strežniku med tem poskusom okoli 6-krat manjši kot prenos v obratni smeri, zahtevki pa niso bili vzporedni, so lahko imeli POST zahtevki večji vpliv na potek veljavnosti strani v predpomnilniku, kot zahtevki tipa GET. Ta vpliv pa je bil spet večji pri straneh z večjo velikostjo.

1.4.5 Primerjava zahtevnosti pisanja in branja brez predpomnilnika

Pri tem testu smo primerjali povprečen čas odgovora na zahtevek pri 200 urejanjih (POST) po 100 straneh s povprečnim časom odgovora na zahtevek pri 200 branjih (GET) po 30 straneh brez vključenega predpomnilnika. Zahtevki tipa POST so na strani zapisali naključno generirano vsebino, dolgo 1000 znakov.

Hipoteza

Naša hipoteza pri tem testu je, da so pisanja (POST zahteve) malo bolj zahtevna kot branja in je zato povprečen čas odgovora pri pisanju malo daljši.

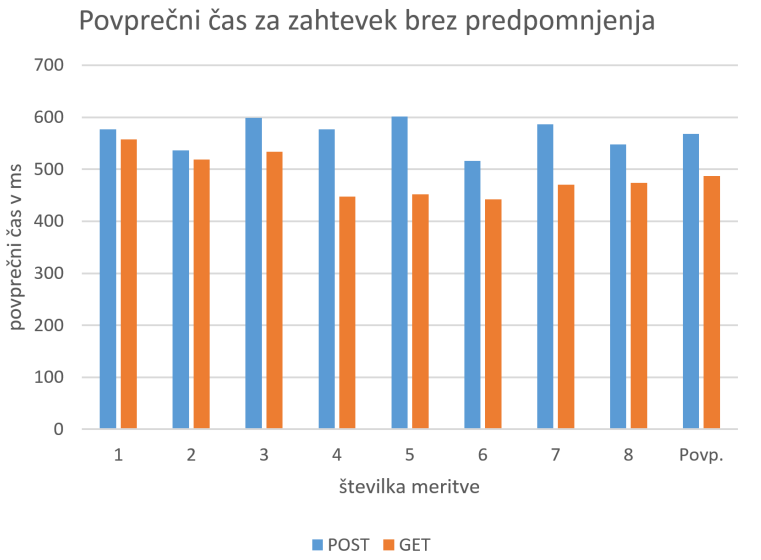
Merive

Pri merjenju povprečnega časa smo izvedli osem zaporednih meritev za oba tipa zahtevkov in beležili povprečje. Zahtevki v tem primeru niso šli preko strežnika CloudFlare in so tako prepotovali najkrajšo možno pot.

Meritve so bile izvedene v sredo, 27. 5. med 13. in 18. uro. Skupno je bilo poslanih 1600 zahtevkov, skupno trajanje vseh zahtevkov pa je znašalo 1687 sekund. Predstavljene so v tabeli 1.7 in v grafični obliki na sliki 1.7.

Številka meritve	1	2	3	4	5	6	7	8	Povprečje
POST	577	536	599	577	601	516	586	548	567.5
GET	557	519	534	447	452	442	470	474	486.875

Tabela 1.7: Tabela povprečnih časov odgovora glede na tip zahtevka.



Slika 1.7: Graf scenarija brez uporabe predpomnilnika.

Komentarji

Iz rezultatov je razvidno, da je povprečni čas za oba tipa zahtevkov podoben, a konsistentno večji za zahtevke tipa POST, kar je skladno z našo hipotezo. Zahtevki tipa POST so v tem primeru za izvršitev potrebovali okoli 16% več časa.

1.4.6 Primerjava zahtevnosti pisanja in branja s predpomnilnikom

Pri tem testu smo primerjali povprečen čas odgovora na zahtevek pri 200 urejanjih (POST) po 30 straneh s povprečnim časom odgovora na zahtevek pri 200 branjih (GET) po 30 straneh z vključenim predpomnilnikom. Zahtevki tipa POST so na strani zapisali naključno generirano vsebino, dolgo 1000 znakov.

Hipoteza

Naša hipoteza pri tem testu je, da pisanja zahtevajo veliko dlje časa kot branja, saj pri 200 zahtevkih za branje po 30 straneh na večino zahtevkov (90%)

odgovori predpomnilniški strežnik, tako kot pri našem prvem testu.

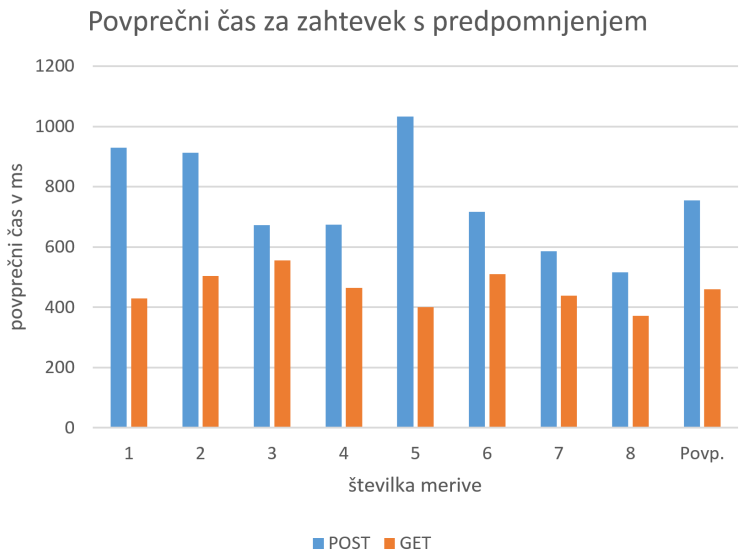
Meritve

Tudi pri tem scenariju smo zagnali zaporedje osmih meritev in beležili povprečje. Ker so zahtevki potovali preko strežnika CloudFlare, ki je izvajal predpomnjenje, so prepotovali daljšo pot. Po vsakem izmed osmih testov smo izpraznili predpomnilnik in tako zagotovili njihovo medsebojno neodvisnost.

Meritve so bile izvedene v sredo, 27. 4. med 13. in 18. uro. Skupno je bilo poslanih 1600 zahtevkov, skupno trajanje vseh zahtevkov pa je znašalo 1942 sekund.

Številka meritve	1	2	3	4	5	6	7	8	Povprečje
POST	930	912	672	674	1032	717	586	516	754.875
GET	429	504	556	464	401	510	438	372	459.25

Tabela 1.8: Tabela povprečnih časov odgovora glede na tip zahtevka.



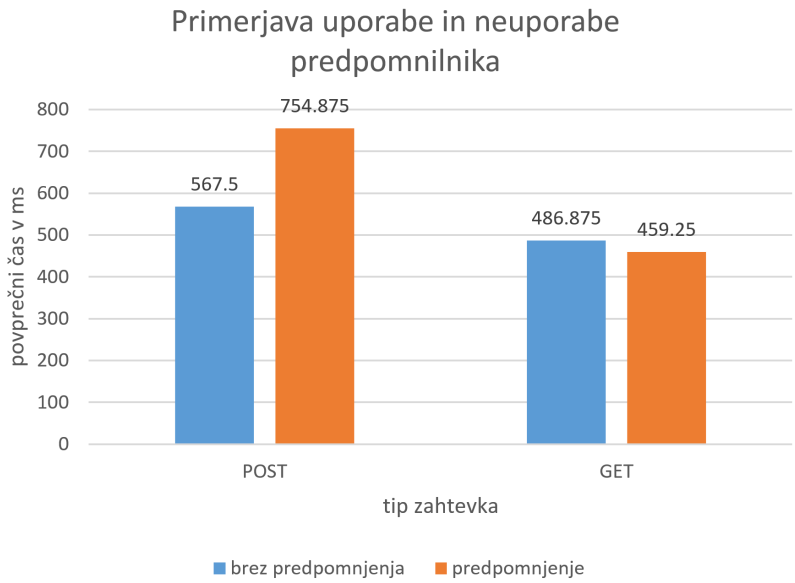
Slika 1.8: Graf scenarija z uporabo predpomnilnika.

Komentarji

Za razliko od scenarija brez vključenega predpomnilnika je tu moč opaziti nekajkrat večje razlike - zahtevki tipa POST so za izvršitev potrebovali okoli 64% dlje. Ker so zahtevki prepotovali daljšo pot, je bilo za pričakovati, da bodo tisti, ki bodo to pot prepotovali v celoti (in ne bodo servirani že s strani strežnika

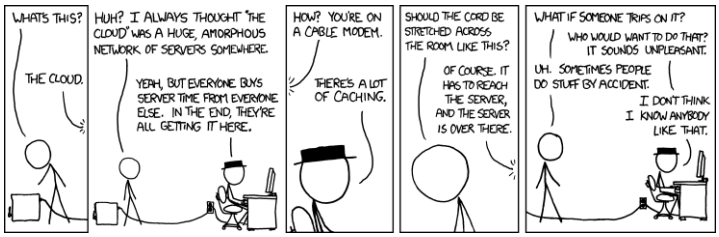
	brez	predp.
POST	567.5	754.875
GET	486.875	459.25

Tabela 1.9: Tabela povprečnih časov odgovora za scenarij brez uporabe predpomnilnika v primerjavi s scenarijem, kjer se ta uporablja.



Slika 1.9: Graf primerjave obeh scenarijev.

CloudFlare), trajali dlje časa kot zahtevki v scenariju brez uporabe predpomnilnika (in zato direktno, krajšo potjo). V tabeli 1.9 je razvidno, da to res velja za zahtevke tipa POST, medtem ko so zahtevki tipa GET kljub daljši prepotovani poti hitrejši.



Slika 1.10: Vir: https://xkcd.com/908/

1.5 Zaključek

V našem delu smo ocenjevali učinkovitost predpomnilniškega strežnika v primeru Wiki aplikacije. Pričakovali smo, da bo predpomnilniški strežnik razbremenil aplikacijski strežnik in zmanjšal čas čakanja na zahteve odjemalca.

Testne scenarije smo pripravili tako, da so v skladu z realno uporabo take strani. Naše hipoteze smo potrdili z meritvami. Izkaže se, da je predpomnilniški strežnik občutno pohitril čas strežbe Wiki strani in s tem omogočil boljšo uporabniško izkušnjo. Pohitritev je bila še posebej očitna v primeru velikega števila hkratnih zahtevkov, ko je bil v primeru neposrednega dostopa (tj. brez predpomnjenja) aplikacijski strežnik preveč obremenjen, da bi v primernem času odgovoril na posamezni zahtevek. Z obremenitvenim testom smo dosegli, da je bil brez predpomnjenja čas čakanja na odgovor okoli 4 sekunde, kar smo obravnavali kot odpoved storitve [5], medtem ko je bil z uporabo predpomilniškega strežnika v enaki situaciji čas čakanja približno pol sekunde. Predpomnilniški strežnik je bil učinkovit samo v zahtevkih za branje, saj je pri zahtevkih za pisanje v vsakem primeru bilo potrebno podatke poslati aplikacijskemu strežniku, da jih je le-ta shranil v bazo. V primeru Wiki strani je v primerjavi z zahtevki za branje zahtevkov za pisanje sorazmerno malo, zato je bila uporaba predpomnilniškega strežnika zelo smiselna. Iz tega lahko sklepamo, da je učinkovitost predpomnilniškega strežnika odvisna od narave aplikacije.

Literatura

[1] "What is a reverse proxy server?." https://www.nginx.com/resources/glossary/reverse-proxy-server/.

[2] "Bananapi - home page." http://bananapi.com/index.php/2-uncategorised/59-what-is-bpi-r1-2.

[3] K. D. Lee, *Python Programming Fundamentals*, ch. Event-Driven Programming, pp. 149–165. London: Springer London, 2011.

[4] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. big'web services: making the right architectural decision," in *Proceedings of the 17th international conference on World Wide Web*, pp. 805–814, ACM, 2008.

[5] F. F.-H. Nah, "A study on tolerable waiting time: how long are web users willing to wait?," *Behaviour & Information Technology*, vol. 23, no. 3, pp. 153–163, 2004.

[6] "Reverse proxy example." https://www.nginx.com/resources/wiki/start/topics/examples/reverseproxycachingexample/.