Analiza zmogljivosti oblačnih in strežniških storitev

Uredil prof. dr. Miha Mraz

Maj 2018

Kazalo

Predgovor

Pričujoče delo je razdeljeno v deset poglavij, ki predstavljajo analize zmogljivosti nekaterih tipičnih strežniških in oblačnih izvedenk računalniških sistemov in njihovih storitev. Avtorji posameznih poglavij so slušatelji predmeta *Zanesljivost in zmogljivost računalniških sistemov*, ki se je v štud.letu 2017/2018 predaval na 1. stopnji univerzitetnega študija računalništva in informatike na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Vsem študentom se zahvaljujem za izkazani trud, ki so ga vložili v svoje prispevke.

prof. dr. Miha Mraz, Ljubljana, v maju 2018

Poglavje 1

Analiza zmogljivosti oblačnih storitev

Edi Čebokli, Rok Grmek, Tadej Škapin

1.1 Opis problema

Za izvedbo analize zmogljivosti oblačnih storitev smo si izbrali problem računanja števila π na oddaljenem strežniku. Računamo ga na dva načina - po Leibnizovi [1] formuli:

$$\frac{\pi}{4} = 1 + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \frac{1}{9} + ... \tag{1.1}$$

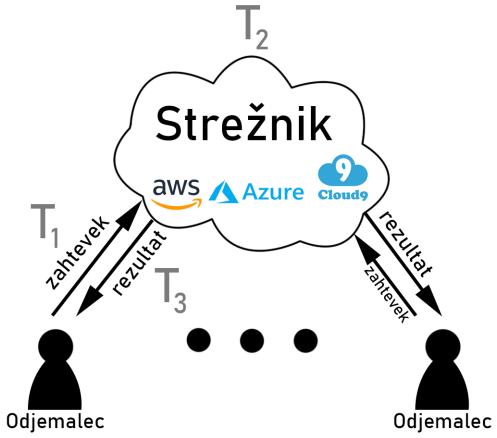
in po Bellardovi [2] formuli:

$$\pi = \frac{1}{2^6} \sum_{n=0}^{\infty} \frac{(-1)^n}{2^{10n}} (-\frac{2^5}{4n+1} - \frac{1}{4n+3} - \frac{2^8}{10n+1} - \frac{2^6}{10n+3} - \frac{2^2}{10n+5} - \frac{2^2}{10n+7} - \frac{1}{10n+9}). \tag{1.2}$$

V obeh primerih je število π predstavljeno kot vsota neskončne vrste, seveda pa v praksi seštejemo le končno mnogo členov. Pri tem se vsota večjega števila členov izraža v bolj natančnem izračunu, vendar pa s tem narašča časovna zahtevnost. Želeli smo, da oba algoritma izračunata število π s podobno natančnostjo, to pa smo dosegli s seštevanjem prvih 24.473.399 členov Leibnizove formule in s seštevanjem le prvih dveh členov Bellardove formule. V obeh primerih naš izračun odstopa od dejanske vrednosti števila π za približno 10^{-7}.

Oba algoritma smo implementirali v Python-u in C-ju, aplikacijo strežnika pa smo namestili pri treh različnih ponudnikih gostovanja (ti so podrobneje predstavljeni v razdelku 1.4). V okviru analize zmogljivosti smo sistem obremenili tako, da smo z večkratnim pošiljanjem zahtev simulirali večje število odjemalcev

(natančnejši opis bremena je v razdelku 1.5). To prikazuje tudi shema na sliki 1.1, kjer T1 označuje čas prenosa zahtevka od odjemalca do strežnika, T2 označuje čas potreben za izračun na strani strežnika, T3 pa čas prenosa rezultatov od strežnika do odjemalca.



Slika 1.1: Shema pošiljanja zahtevkov na strežnik.

1.2 Pregled sorodnih virov

Računanje števila π je numeričen problem, ki ga rešujemo s številnimi zaporednimi aritmetičnimi operacijami nad števili v plavajoči vejici (v dvojni natančnosti). Zato smo že pred samim računanjem želeli preveriti, kako hitre so takšne operacije v primeru oblačnega računanja. V članku "A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing" [3] je analizirana zmogljivost računanja pri ponudniku gostovanja Amazon Web Services. Med drugim so v članku predstavljeni tudi rezultati meritev, kjer je razvidno, kakšno število operacij (seštevanj/množenj v plavajoči vejici, v dvojni natančnosti) lahko strežnik izvede na sekundo. Analizirani so sicer le plačljivi paketi, za te pa je pričakovana nekolika višja zmogljivost v primerjavi z brezplačnim paketom t2.micro, ki smo ga uporabili sami. V povprečju so v analizi namerili približno 0.6 GOPS. Glede na način izvajanja meritev, bomo lahko za računanje števila π po Leibnizovi formuli, implementirano v C-ju na koncu še sami izračunali povprečno število operacij na sekundo in ga primerjali z rezultatom iz prej omenjene analize zmogljivosti.

1.3 Izbira tehnologij

Pri izbiri programskih jezikov smo izbrali Python in C, ki sta različna po načinu izvajanja programov. Python je interpreterski jezik, ki hitreje analizira izvorno kodo, vendar počasneje izvaja program v primerjavi z eksekucijskim jezikom C. Strežnik in odjemalec sta implementirana v Python-u, algoritma za računanje π pa v C-ju in Python-u.

1.4 Ponudniki gostovanja

Aplikacijo strežnika za oddaljeno računanje števila π smo namestili pri treh različnih ponudnikih gostovanja - Amazon Web Services [4], Microsoft Azure [5] in Cloud9 [6]. Za večino testov smo uporabili Amazon Web Services, v enem od testov pa so ponudniki neposredno primerjani med sabo.

Pri ponudniku Microsoft Azure smo izbrali paket A1 standard, pri AWS pa paket t2.micro. To so paketi, ki jih lahko izberemo v brezplačni poskusni dobi uporabe storitve. Specifikacija sistemov, ki jih ponujajo zgornji trije ponudniki, je navedena v tabeli 1.1.

Ponudnik	CPE	Št. jeder	RAM	HDD	Lokacija
AWS	Intel Xeon E5-2676 v3	1	1GB	8GB	Zahodna Evropa
Azure	Intel Xeon E5-2630 v3	1	1.5GB	30GB	Južna Azija
Cloud9	Intel Xeon @2.50GHz	8	1GB	2GB	ZDA

Tabela 1.1: Tabela ponudnikov in njihovih specifikacij.

1.5 Definicija bremena storitve

Obremenjenost našega strežnika, ki ponuja oddaljeno računanje števila π, je odvisna od izbrane metode računanja, od implementacije te metode in od intenzivnosti prejemanja zahtevkov za računanje. Zato smo strežnik obremenili tako, da smo generirali večje število zaporednih zahtevkov za enega od načinov računanja, pri tem pa določili interval čakanja med dvema zahtevkoma.

Interval je v grobem lahko deterministične (točno določen), ali pa nedeterministične (psevdo-naključno generiran in porazdeljen eksponentno). Prva možnost omogoča lažjo analizo rezultatov zaradi konstantnega intervala, druga pa boljše opiše realno breme.

Pomembna je tudi delitev glede na to, kakšen je interval čakanja v primerjavi s časom računanja. Velika razlika v obremenitvi sistema se namreč pojavi, če interval med dvema zahtevkoma zmanjšamo do te mere, da ta postane manjši od povprečnega časa računanja in zahtevki za računanje prihajajo pogosteje, kot jih lahko strežnik obdela.

1.6 Definicija metrik in orodij za meritve

Za posamezno računanje na oddaljenem strežniku dobimo kot rezultat poleg števila π tudi čas, ki ga je strežnik potreboval za računanje (T2), obenem pa merimo še celoten čas od pošiljanja zahtevka za računanje do prejetja rezultata (T1 + T2 + T3). Pri eksperimentih nato opazujemo čas računanja na strežniku (T2) in čas, potreben za komunikacijo s strežnikom (T1 + T3).

1.7 Rezultati meritev

1.7.1 Primerjava različnih načinov računanja

Opis eksperimenta

Prve meritve smo opravili le pri ponudniku gostovanja AWS. Za vsako od štirih različic izračuna smo strežniku 1 uro, iz enega klienta, na vsake 3 sekunde (deterministično) pošiljali zahtevke in ob tem merili čase. S tem eksperimentom smo želeli preveriti delovanje sistema in primerjati štiri različne načine računanja med sabo.

Hipoteza

Pričakujemo lahko, da bodo časi računanja (T2) za vse 4 načine računanja skozi celoten čas testiranja približno konstantni. Strežnik bo namreč vedno obdeloval le eno zahtevo na enkrat, ker nobeno računanje naj ne bi trajalo več kot 3 sekunde.

Računanje implementirano v Python-u bo najverjetneje trajalo dlje od tistega v C-ju (za isto metodo računanja), za vsako od implementacij pa bo Leibnizova metoda verjetno zahtevala več časa od Bellardove. Kljub temu pa težko ocenimo, ali se bo iz časovnega vidika boljše odrezala Leibnizova metoda implementirana v C-ju ali Bellardova metoda implementirana v Python-u.

Čas komunikacije (T1 + T3) ni odvisen od načina računanja, zato sklepamo, da bo ta približno enak za vse štiri. Ta čas bo najverjetneje približno konstanten tudi skozi celoten čas testiranja, saj ne pričakujemo večjih sprememb v zasedenosti omrežja med testiranjem.

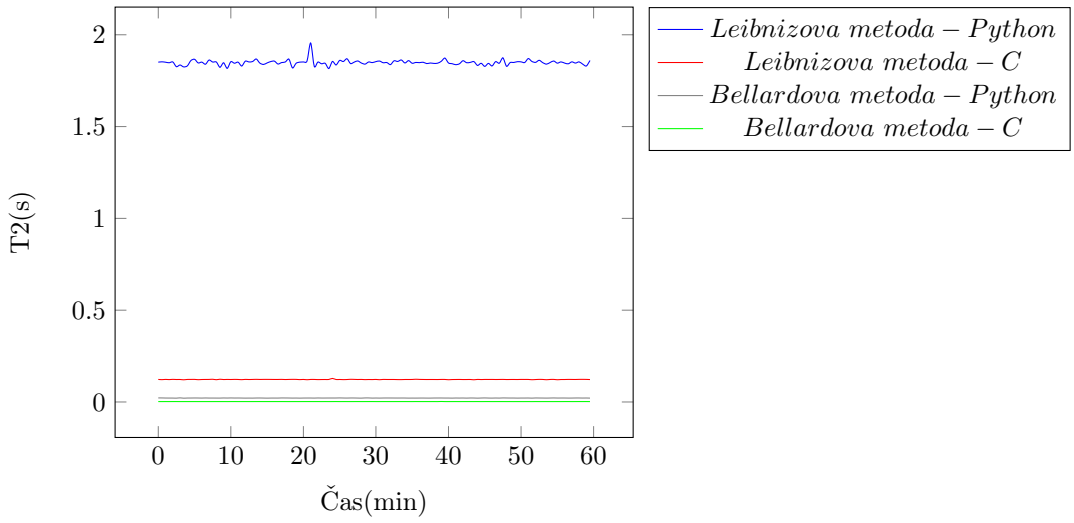
Okoliščine

Meritve so bile opravljene v Ljubljani, s povezavo 100 Mbps / 10 Mbps. Bile so opravljene med 18. in 19. uro v sredo 18. 4. 2018. Vsaka meritev se je izvajala na svojem strežniku, vsi strežniki pa imajo identično konfiguracijo in lokacijo.

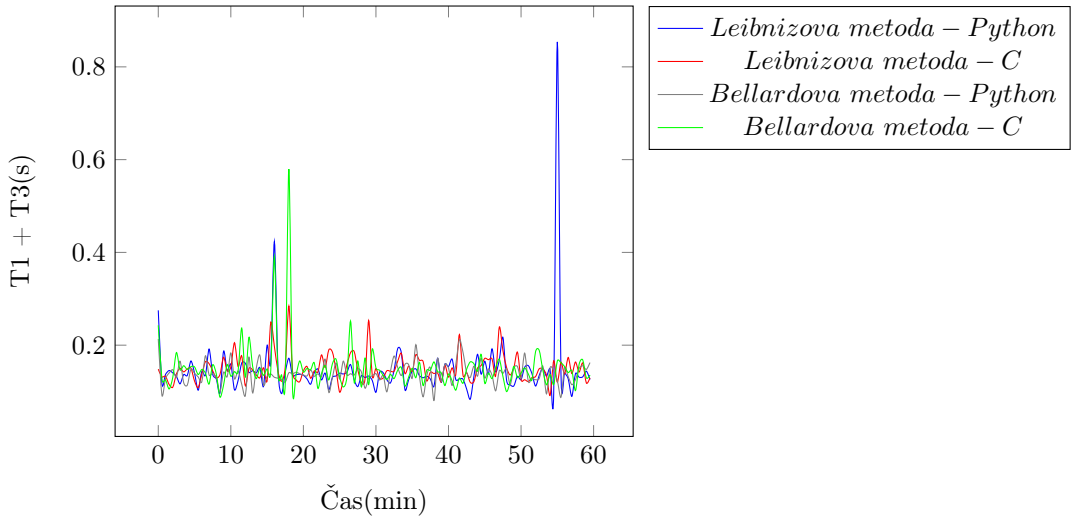
Rezultati

Čas računanja na strežniku (T2) v odvisnosti od časa testiranja je prikazan na sliki 1.2, celoten čas komunikacije (T1 + T3) spet v odvisnosti od časa testiranja pa je prikazan na sliki 1.3.

Hipoteza je bila očitno smiselna, iz rezultatov pa je vidno, da računanje po Bellardovi metodi zahteva manj časa v primerjavi z Leibnizovo metodo, ne glede na tehnologijo implementacije.



Slika 1.2: Čas računanja na strežniku v odvisnosti od časa testiranja.



Slika 1.3: Celoten čas komunikacije v odvisnosti od časa testiranja.

1.7.2 Primerjava ponudnikov gostovanja

Opis eksperimenta

Za ta eksperiment smo aplikacijo strežnika namestili pri treh različnih ponudnikih gostovanja (ti so predstavljeni v razdelku 1.4). Zahtevke smo spet pošiljali 1 uro, na vsake 3 sekunde (deterministično), vendar tokrat do treh različnih strežnikov. Na vsakem od strežnikov smo testirali le računanje z Leibnizovo metodo, implementirano v Python-u.

Hipoteza

Glede na specifikacije sistemov ponudnikov (tabela 1.1), pričakujemo podobne čase računanja (T2) za AWS in Azure, nekoliko krajše čase pa za Cloud9.

Časi komunikacije (T1 + T3) se bodo najverjetneje razlikovali zaradi različnih lokacij strežnikov. Najkrajši čas pričakujemo za AWS, saj je strežnik lociran v Evropi, najdaljši čas pa bo verjetno potreben za dostop do strežnika v Južni Aziji, ki ga ponuja Azure.

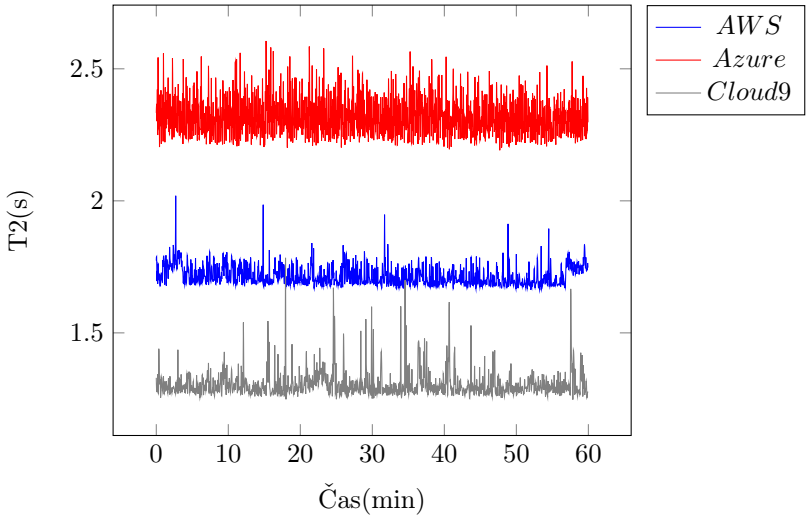
Okoliščine

Meritve so bile opravljene v Ilirski Bistrici, s povezavo 20 Mbps / 5 Mbps. Bile so opravljene med 14. in 15. uro v sredo 2. 5. 2018.

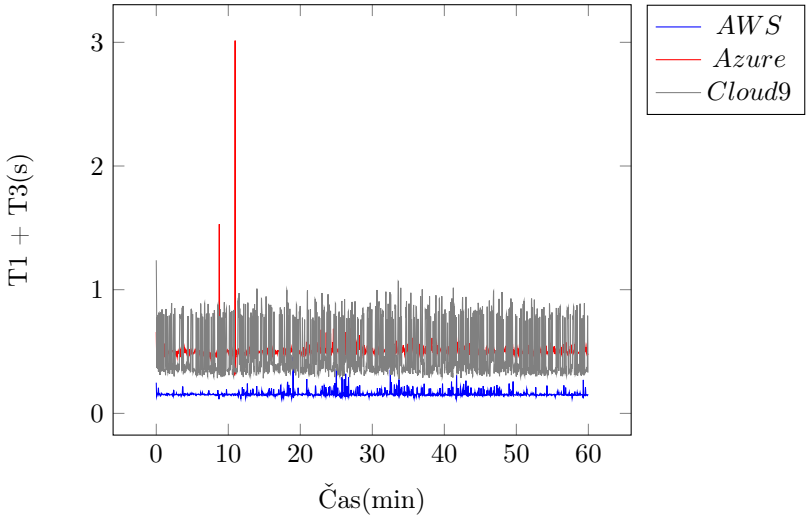
Rezultati

Čas računanja na strežniku (T2) v odvisnosti od časa testiranja je prikazan na sliki 1.4, celoten čas komunikacije (T1 + T3) spet v odvisnosti od časa testiranja pa je prikazan na sliki 1.5.

Rezultati se do neke mere ujemajo z dano hipotezo. Nekoliko so nas presenetili časi računanja na strežniku, ki je bil nameščen pri ponudniku gostovanja Azure, saj so ti očitno precej višji od časov računanja na strežniku, nameščenem pri AWS. Poleg tega, smo slabo predvideli tudi čase komunikacije s strežnikom, nameščenim pri Cloud9. Ti so, za razliko od časov pri ostalih dveh ponudnikih, razpršeni na bistveno širšem intervalu in celo presegajo čase komunikacije s strežnikom, lociranim v Južni Aziji (Azure).



Slika 1.4: Čas računanja na strežniku v odvisnosti od časa testiranja.



Slika 1.5: Celoten čas komunikacije v odvisnosti od časa testiranja.

1.7.3 Nedeterministično pošiljanje zahtevkov

Opis eksperimenta

V prvih dveh eksperimentih smo zahtevke pošiljali deterministično, točno na vsake 3 sekunde. Ker strežniki običajno niso obremenjeni na tak način, smo tokrat generirali zahtevke po Poissonovi porazdelitvi [7], saj tak model boljše

ponazori običajen promet. Časovni interval med dvema zahtevkoma je v takem primeru porazdeljen eksponentno s parametrom λ, pričakovana vrednost intervala pa je enaka $\frac{1}{\lambda}$. V našem primeru smo želeli v povprečju ohraniti 3 sekundni interval, zato smo uporabili parameter porazdelive ali intenzivnosti porajanja zahtev $\lambda = \frac{1}{3}$ zahteve na sekundo. Eksperiment je tudi tokrat trajal 1 uro, meritve pa smo opravili le pri ponudniku gostovanja AWS in sicer za računanje po Leibnizovi metodi, implementirano v Python-u.

Hipoteza

Čas računanja (T2) bo v najboljšem primeru enak tistemu iz eksperimenta z determinističnim intervalom pošiljanja zahtevkov, lahko pa pričakujemo, da se bodo nekateri zahtevki obdelovali nekoliko dlje. Zaradi naključnosti modela, se bo namreč zelo verjetno zgodilo to, da bo nek zahtevek poslan še preden bo prejšnji obdelan, kar pa bo prineslo daljši čas računanja.

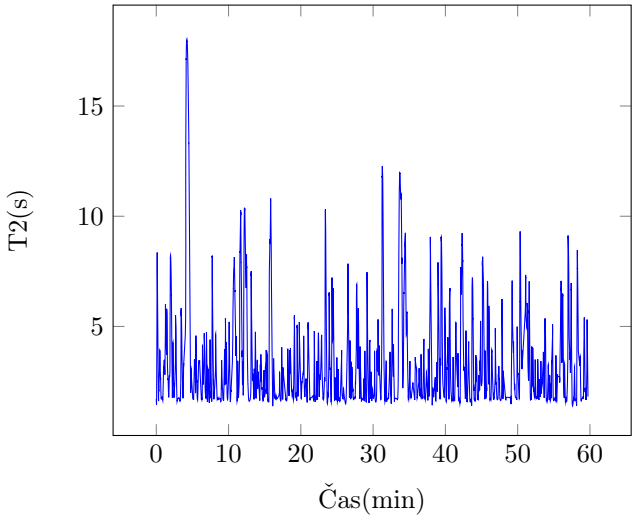
Sprememba modela za generiranje zahtevkov najverjetneje ne bo vplivala na čas komunikacije (T1 + T3), zato predvidevamo, da bo ta čas ostal nespremenjen.

Okoliščine

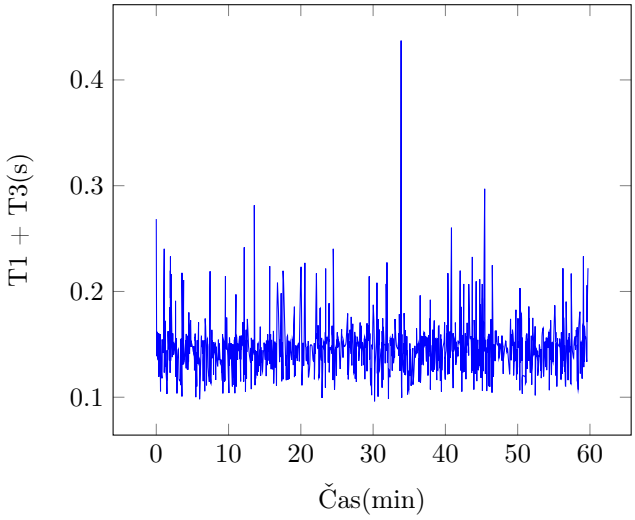
Meritve so bile opravljene v Ilirski Bistrici, s povezavo 20 Mbps / 5 Mbps. Bile so opravljene med 11. in 12. uro v ponedeljek 30. 4. 2018.

Rezultati

Čas računanja na strežniku (T2) v odvisnosti od časa testiranja je prikazan na sliki 1.6, celoten čas komunikacije (T1 + T3) spet v odvisnosti od časa testiranja pa je prikazan na sliki 1.7. Ti rezultati popolnoma potrjujejo dano hipotezo.



Slika 1.6: Čas računanja na strežniku v odvisnosti od časa testiranja.



Slika 1.7: Celoten čas komunikacije v odvisnosti od časa testiranja.

1.7.4 Ogrevanje sistema (postopno večanje obremenitve)

Opis eksperimenta

V predhodnih eksperimentih nismo nikoli pošiljali zahtevkov bolj pogosto, kot jih je strežnik lahko obdelal (v povprečju), zato smo pripravili eksperiment, kjer postopoma višamo frekvenco pošiljanja zahtevkov. Najprej smo 5 minut

pošiljali zahtevke na vsake 3 sekunde, naslednjih 5 minut na 2 sekundi, nato 5 minut vsako sekundo, zadnjih 5 minut pa vsake 0,5 sekunde (vsi intervali čakanja so bili generirani deterministično). Računali smo z Leibnizovo metodo, implementirano v Pythonu, strežnik pa smo namestili pri ponudniku gostovanja AWS.

Hipoteza

Prvih 10 minut lahko pričakujemo konstanten čas računanja (T2), saj še vedno pošiljamo zahtevke dovolj počasi. Interval med dvema zahtevkoma je namreč v tem delu testiranja najprej enak 3 in nato 2 sekundi, kar je več od povprečnega časa trajanja enega izračuna. Naslednjih 5 minut bo čas računanja najverjetneje začel linearno naraščati, saj se program izvaja več kot 1 sekundo, kar pomeni, da bodo novi zahtevki prihajali hitreje kot se lahko obdelajo in bo hkrati v obdelavi vedno več zahtevkov. Enako velja za naslednjih 5 minut, vendar pričakujemo še hitrejše naraščanje časa računanja.

Čas komunikacije (T1 + T3) bo najverjetneje konstanten skozi vse 4 faze, saj ta eksperiment predstavlja večjo obremenitev predvsem za strežnik, ne pa tudi za omrežje. Zahtevke namreč pošiljamo kvečjemu 2-krat na sekundo, paketi pa so ranga velikosti nekaj KB.

Okoliščine

Meritve so bile opravljene v Ilirski Bistrici, s povezavo 20 Mbps / 5 Mbps. Bile so opravljene med 14. in 15. uro v ponedeljek 30. 4. 2018.

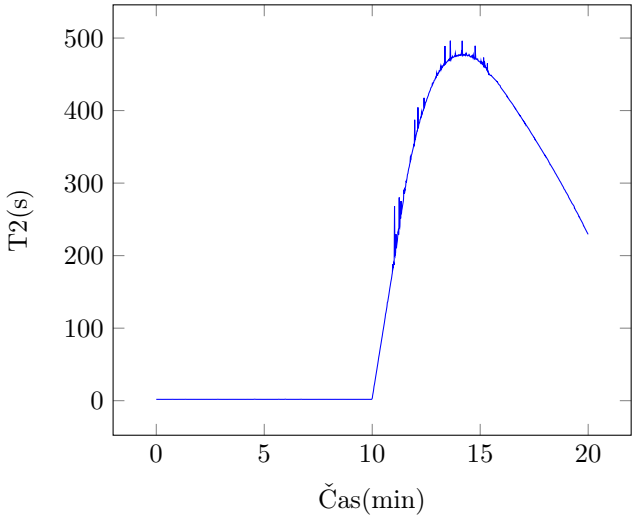
Rezultati

Čas računanja na strežniku (T2) v odvisnosti od časa testiranja je prikazan na sliki 1.8, celoten čas komunikacije (T1 + T3) spet v odvisnosti od časa testiranja pa je prikazan na sliki 1.9.

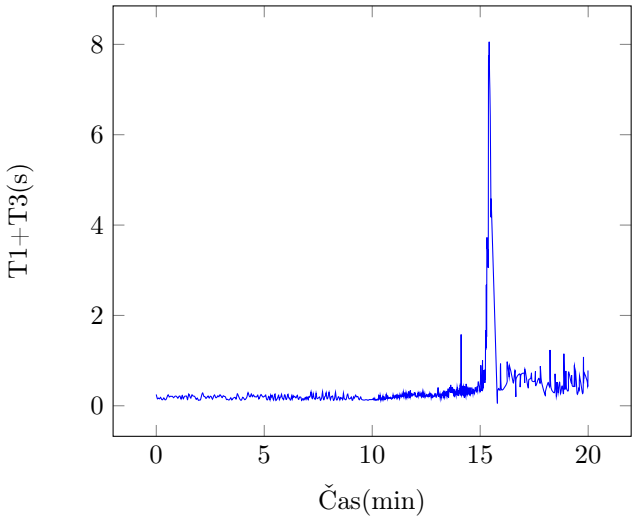
Pri postavljanju hipoteze smo tokrat naredili napako. Skozi prvi dve fazi so časi računanja res konstantni in nato nekaj časa naraščajo, vendar se malo pred koncem tretje faze naraščanje upočasni, kot bi se časi približevali neki zgornji meji, nato pa ti časi celo padajo. Približevanje zgornji meji je smiselno, saj strežnik ne more hkrati obdelovati poljubnega števila zahtevkov (zaradi končnega pomnilnika) in zato ta v neki točki začne spuščati zahtevke (od strežnika smo namreč prejeli le 692 odgovorov za 1150 poslanih zahtevkov). Padanje časov je najverjetneje povezano s koncem eksperimenta. Tisti zadnji zahtevki se (za razliko od predhodnih) obdelujejo še po koncu ekpserimenta, takrat pa novi zahtevki ne prihajajo več na strežnik. Nekaj časa na koncu se torej manjša število zahtevkov, ki so hkrati v obdelavi, to pa pomeni krajše čase računanja.

Tudi časi komunikacije se nekoliko razlikujejo od pričakovanih, vendar to odstopanje povezujemo z napako pri merjenju časa računanja (T2) v primeru, ko strežnik obdeluje veliko število zahtevkov hkrati. Lahko se zgodi, da zaradi zelo velikega števila niti, ki obdelujejo posamezne zahtevke, glavna nit, ki sprejema

zahtevke in meri čas računanja, nekaj časa ne pride na vrsto in posledično zamudi pri beleženju časa, ko je zahtevek prišel do strežnika. Ta napaka se odraža kot nekaj sekundni pribitek času komunikacije in enako dolg odbitek času računanja, žal pa je odvisna od operacijskega sistema, ki razvršča niti in je ne moremo odpraviti.



Slika 1.8: Čas računanja na strežniku v odvisnosti od časa testiranja.



Slika 1.9: Celoten čas komunikacije v odvisnosti od časa testiranja.

1.7.5 Konstantna visoka obremenitev

Opis eksperimenta

Pri prejšnjem eksperimentu so nas rezultati rahlo presenetili, zato smo želeli našo interpretacijo rezultatov preveriti še z dodatnim eksperimentom. V ta namen smo tokrat izvedli meritve le za tretjo fazo prejšnjega eksperimenta (deterministično pošiljanje zahtevkov na vsako sekundo), vendar smo eksperiment izvajali več časa - 1 uro.

Hipoteza

Pričakujemo, da bodo časi računanja (T2) najprej naraščali dokler ne dosežejo zgornje meje, nato bodo večji del testiranja ti časi približno konstantni, pred koncem pa se bodo časi spet zmanjšali.

Čas komunikacije (T1 + T3) bo najverjetneje spet višji kot bi bilo smiselno, vendar bo to zaradi napake merjenja časov, omenjene pri prejšnjem eksperimentu. Če so višji izmerjeni časi komunikacije res posledica napake, potem bodo ti časi tudi nekoliko korelirani s časi računanja. Časi računanja namreč dobro kažejo na obremenjenost sistema z velikim številom hkrati aktivnih niti, velikost napake pa je odvisna ravno od števila niti, ki si z glavno nitjo delijo računske vire.

Okoliščine

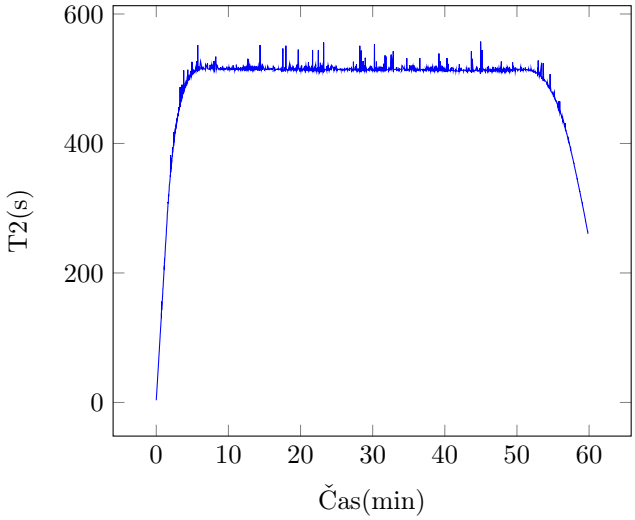
Meritve so bile opravljene v Ilirski Bistrici, s povezavo 20 Mbps / 5 Mbps. Bile so opravljene med 15. in 16. uro v ponedeljek 30. 4. 2018.

Rezultati

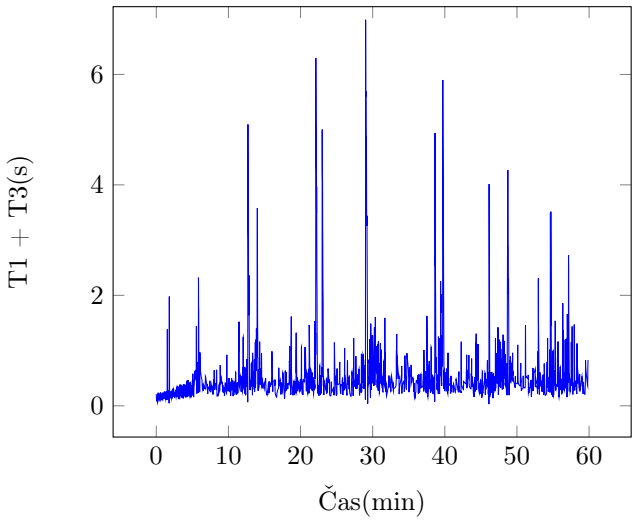
Čas računanja na strežniku (T2) v odvisnosti od časa testiranja je prikazan na sliki 1.10, celoten čas komunikacije (T1 + T3) spet v odvisnosti od časa testiranja pa je prikazan na sliki 1.11.

Rezultati se ujemajo z dano hipotezo, to pa potrjuje tudi celotno interpretacijo rezultatov iz prejšnjega eksperimenta.

Zanimivo je še pogledati število zahtevkov, ki so bili na strežniku spuščeni. Od 3600 poslanih zahtevkov, smo odgovor dobili le za 2049 zahtevkov. V eni uri je strežnik torej 2049-krat uspešno izračunal število π, kar pomeni, da je za posamezen izračun efektivno potreboval le 1,76 s ($\frac{3600s}{2049}$), ta čas je pa zelo podoben času, ki ga tak strežnik potrebuje za en izračun, ko sploh ni preobremenjen. Iz tega sledi, da strežnik v primeru preobremenitve spusti praktično najmanjše možno število zahtevkov. Če bi želeli spuščanje zahtevkov popolnoma preprečiti, bi te lahko generirali kvečjemu na vsake 1,76 s (interval je torej potrebno prilagoditi času, ki ga strežnik potrebuje za en izračun).



Slika 1.10: Čas računanja na strežniku v odvisnosti od časa testiranja.



Slika 1.11: Celoten čas komunikacije v odvisnosti od časa testiranja.

1.7.6 Primerjava različno intenzivnih preobremenitev

Opis eksperimenta

Pri prejšnjem eksperimentu smo zahtevke pošiljali pogosteje kot jih lahko strežnik obdela in opazili, da se v neki točki časi računanja ustalijo. To se zgodi, ko je dosežena zgornja meja za število zahtevkov, ki se lahko hkrati obdelujejo

(tisti zahtevki, ki bi to mejo presegli pa so izpuščeni in nanje ne dobimo odgovora). S tem eksperimentom bi radi preverili, do kakšne spremembe pride pri časih računanja, če zahtevke pošiljamo nekoliko pogosteje ali pa nekoliko redkeje (vendar še vedno bolj pogosto, kot jih lahko strežnik obdela). Zato smo tokrat 20 minut deterministično pošiljali zahtevke, na vsake 0,8 s, na vsako sekundo ter na vsake 1,2 s do strežnika, nameščenega pri ponudniku AWS. Eksperimenti so potekali istočasno, vsak na svoji instanci. Vse instance so identične.

Hipoteza

Čas računanja je v našem primeru odvisen od števila zahtevkov, ki se hkrati obdelujejo, zato pričakujemo, da bodo časi računanja naraščali hitreje v primeru, kjer pogosteje pošiljamo zahtevke in obratno. Poleg tega, lahko pričakujemo, da se bodo v vseh treh primerih časi ustalili pri približno enaki vrednosti, saj bo meja, pri kateri strežnik spušča zahtevke enaka v vseh treh primerih, ne glede na to, kako pogosto pošiljamo zahtevke.

Pri časih komunikacije ne pričakujemo večjih razlik, saj noben od treh primerov pošiljanja zahtevkov ne predstavlja resnejše obremenitve za omrežje. Po drugi strani pa v vseh treh primerih vseeno pričakujemo prisotnost že omenjene napake pri merjenju časov, saj bo sistem obremenjen s precej visokim številom niti.

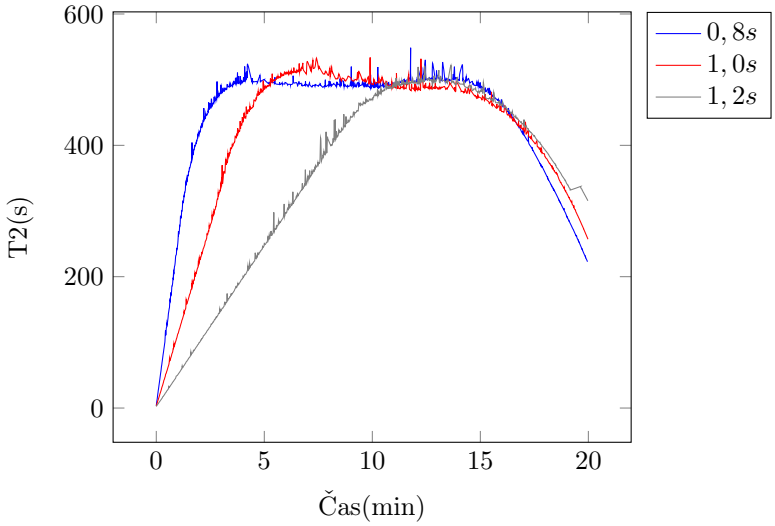
Okoliščine

Meritve so bile opravljene v Sežani, s povezavo 10 Mbps / 10 Mbps. Bile so opravljene med 7. in 8. uro v soboto 19. 5. 2018.

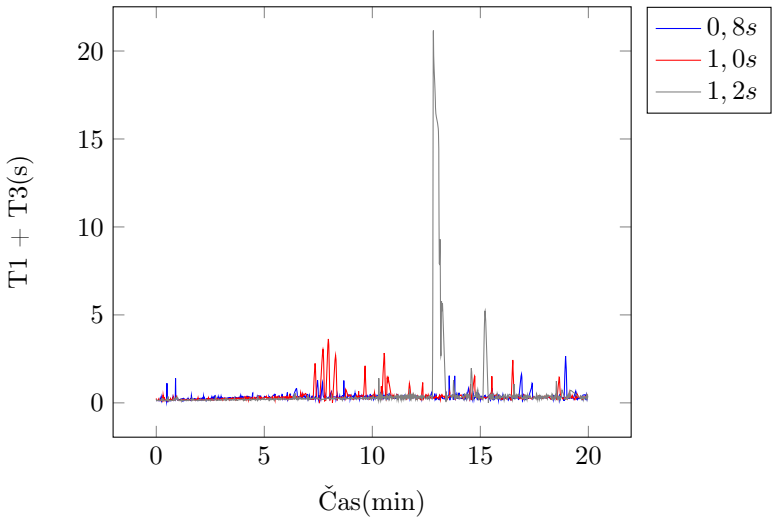
Rezultati

Čas računanja na strežniku (T2) v odvisnosti od časa testiranja je prikazan na sliki 1.12, celoten čas komunikacije (T1 + T3) spet v odvisnosti od časa testiranja pa je prikazan na sliki 1.13.

Hipoteza se je izkazala za pravilno, saj časi računanja najprej naraščajo intervalu primerno, nato pa se v vseh treh primerih ustalijo na približno 500 s. Tudi časi komunikacije ustrezajo hipotezi, vključno z omenjeno napako, ki je še posebej vidna v primeru 1,2 s intervala.



Slika 1.12: Čas računanja na strežniku v odvisnosti od časa testiranja.



Slika 1.13: Celoten čas komunikacije v odvisnosti od časa testiranja.

1.7.7 Eksperimentalno določanje meje preobremenitve

Opis eksperimenta

V enem od prejšnjih eksperimentov (razdelek 1.7.5) je že bilo govora o največji možni obremenitvi strežnika, pri kateri se zahtevki pravočasno obdelujejo in se novi zahtevki ne kopičijo v vrsti. Predvideli smo, da je interval pošiljanja

potrebno prilagoditi povprečnemu času računanja posameznega zahtevka, s tem eksperimentom pa bi to želeli še dodatno preveriti. V našem primeru je za strežnik, nameščen pri ponudniku gostovanja AWS ta čas enak približno 1,76 s. Zato smo tokrat za gostovanje spet uporabili ponudnika AWS, zahtevke pa smo 20 minut pošiljali nedeterministično z intervali 1,70 s, 1,75 s ter 1,80 s. Eksperimenti so potekali istočasno, vsak na svoji instanci. Vse instance so identične.

Hipoteza

Najverjetneje bodo časi računanja v obeh primerih, kjer je interval generiranja zahtevkov nižji od povprečnega časa računanja nekoliko naraščali skozi čas, v primeru višjega intervala pa pričakujemo, da bo čas računanja približno konstanten. Seveda ne smemo pozabiti, da se tokrat zahtevki generirajo nedeterministično, kar pomeni, da bo pri vseh treh primerih prisotno tudi nekakšno nihanje v časih računanja - zgornje trditve se torej nanašajo bolj na trend, kot pa na dejansko gibanje vrednosti skozi čas.

Pri časih komunikacije v tem eksperimentu ne pričakujemo nobenih večjih posebnosti.

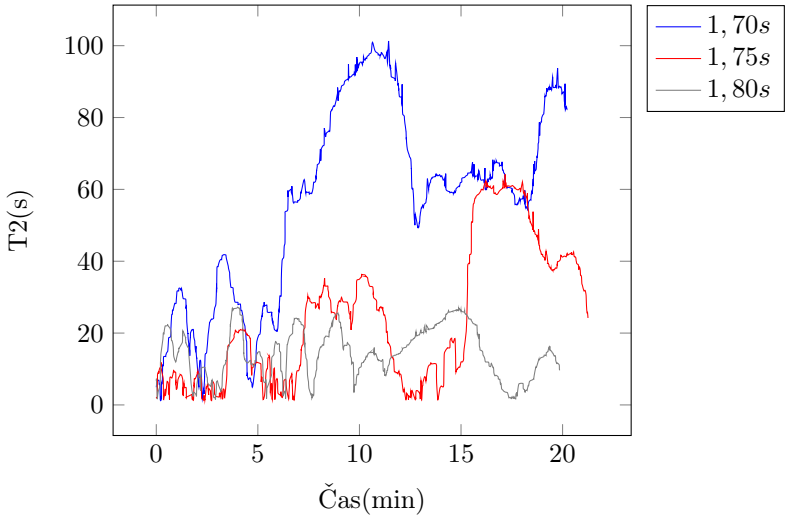
Okoliščine

Meritve so bile opravljene v Sežani, s povezavo 10 Mbps / 10 Mbps. Bile so opravljene med 13. in 14. uro v soboto 19. 5. 2018.

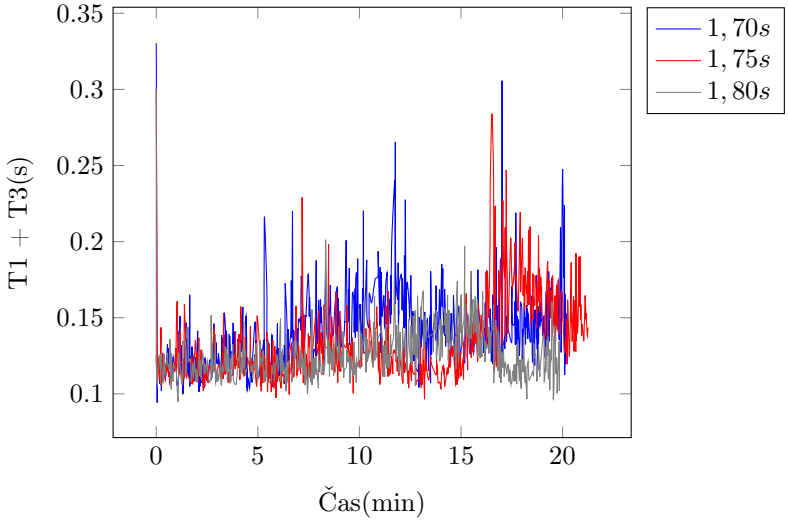
Rezultati

Čas računanja na strežniku (T2) v odvisnosti od časa testiranja je prikazan na sliki 1.14, celoten čas komunikacije (T1 + T3) spet v odvisnosti od časa testiranja pa je prikazan na sliki 1.15.

Na prvi pogled izgledajo časi računanja v vseh treh primerih precej podobno, vendar lahko opazimo, da se v primeru 1,80 s intervala časi računanja po naraščanju vedno vrnejo do običajnega nivoja, v ostalih dveh primerih pa časi postopoma naraščajo intervalu primerno in s tem kažejo na predpostavljeno preobremenitev. S časi komunikacije v tem eksperimentu res ni bilo posebnosti. Izkaže se, da ti sploh niso korelirani z intervalom pošiljanja zahtevkov, kar je tudi smiselno za tako majhne razlike v obremenitvi omrežja.



Slika 1.14: Čas računanja na strežniku v odvisnosti od časa testiranja.



Slika 1.15: Celoten čas komunikacije v odvisnosti od časa testiranja.

1.7.8 24-urni eksperiment

Opis eksperimenta

S prejšnjim eksperimentom smo potrdili, da mejni interval generiranja zahtevkov, preden sistem preobremenimo, leži nekje med 1,75 s ter 1,80 s (za ponudnika gostovanja AWS). Kot zadnji eksperiment pa smo želeli pripraviti še testiranje,

kjer sistem obremenimo na približno dve tretjini mejne obremenitve, vendar takšno obremenitev vzdržujemo celih 24 ur. Zahtevke smo torej pošiljali 24 ur do strežnika, nameščenega pri ponudniku gostovanja AWS, nedeterministično z intervalom 2,65 s.

Hipoteza

Ta eksperiment simulira promet, pri katerem uporabniki občiajno že opazijo, da sistem deluje rahlo počasneje kot sicer. Pričakujemo torej nekoliko višje čase računanja (v primerjavi s časom, ki je potreben za obdelavo enega samega zahtevka).

Pri časih komunikacije tudi tokrat ne pričakujemo večjih posebnosti. Pozorni bomo lahko na razlike skozi različne ure dneva, vendar nismo prepričani, če bodo te razlike dovolj očitne.

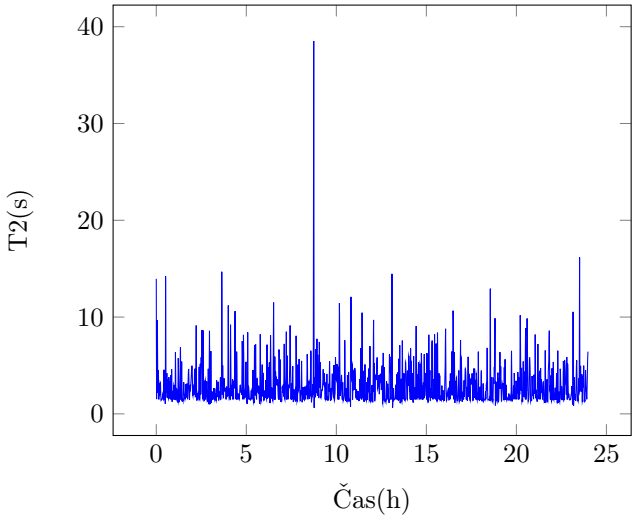
Okoliščine

Meritve so bile opravljene v Sežani, s povezavo 10 Mbps / 10 Mbps. Bile so opravljene med 14. uro v soboto 19. 5. 2018 in 14. uro v nedeljo 20. 5. 2018.

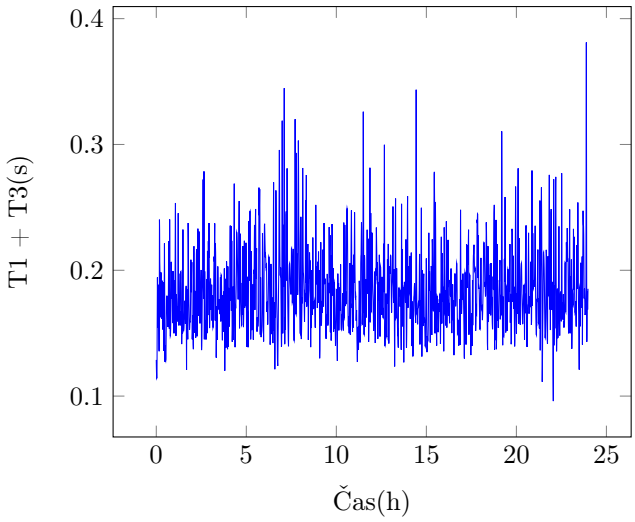
Rezultati

Čas računanja na strežniku (T2) v odvisnosti od časa testiranja je prikazan na sliki 1.16, celoten čas komunikacije (T1 + T3) spet v odvisnosti od časa testiranja pa je prikazan na sliki 1.17.

Rezultati v celoti potrjujejo hipotezo. V najslabšem primeru je računanje trajalo skoraj 40 s (sicer pa za obdelavo enega samega zahtevka v povprečju preteče približno 1,76 s). Iz časov komunikacije žal ne razberemo odvisnosti od ure dneva.



Slika 1.16: Čas računanja na strežniku v odvisnosti od časa testiranja.



Slika 1.17: Celoten čas komunikacije v odvisnosti od časa testiranja.

1.8 Zaključek

Z zgornjimi eksperimenti smo ovrednotili časovno zahtevnost štirih različnih načinov računanja števila π in zmogljivost strežnikov pri treh različnih ponudnikih gostovanja. Iz rezultatov lahko razberemo, da je Bellardova metoda konstantno hitrejša od Leibnizove ne glede na implementacijo. Prav tako lahko opazimo, da

je računanje pri ponudniku Cloud9 najhitrejše, pri Azure najpočasnejše, AWS pa pade nekje vmes. Drugačno razporeditev vidimo pri času komunikacije, kjer je pri ponudniku AWS najkrajši, zaradi nam ugodne lokacije strežnikov. Azure in Cloud9 imata podoben čas komunikacije, vendar ima slednji veliko večjo varianco.

S stopnjevanim deterministčnim pošiljanjem zahtevkov smo prišli do ugotovitve, da strežnik pri preveliki obremenitvi začne spuščati zahtevke. Na podlagi te ugotovitve, smo z dodatnim testom določili zgornjo mejo obremenitve (1,76 s), pri kateri strežnik še obdela vse poslane zahtevke (ne glede na čas testiranja). V eksperimentu 1.6.6 smo opazovali tudi kako različni intervali pošiljanja zahtevkov vplivajo na hitrost polnjenja vrste, pri tem pa smo tudi določili zgornjo mejo časa izračuna posameznega zahtevka (v primeru maksimalnega števila zahtevkov v hkratni obdelavi). Opazimo tudi, da proti koncu testiranja strežnik potrebuje vse manj časa za obdelavo posameznega zahtevka, saj se po zadnjem poslanem zahtevku, hkrati obdeluje vedno manjše število zahtevkov. To potrdimo še z dodatnim eksperimentom, kjer zahtevke eno uro deterministično pošiljamo vsako sekundo.

Ko smo določili zgornjo mejo frekvence deterministčnega pošiljanja zahtevkov, smo strežnik preizkusili tudi na način ki simulira realno okolje. Zahtevke smo večkrat nedeterministično pošiljali z različnimi frekvencami, blizu predvidene meje preobremenitve. Opazimo, da se zahtevki na strežniku počasi kopičijo in včasih nastanejo daljše čakalne vrste, kljub temu pa v primeru, ko pošiljamo z nekoliko manjšo frekvenco od predvidene meje preobremenitve, ne izgubimo nobenega zahtevka. Za konec na strežnik nedeterministično pošiljamo zahtevke pri $\frac{2}{3}$ sposobnosti delovanja strežnika, saj je to meja pri kateri uporabnik občuti zakasnitve. Čas obdelovanja zahtev je v tem primeru podoben kot v prvem eksperimentu, kjer strežnik ni bil tako intenzivno obremenjen, le da občasno pride do nekoliko večjih časov računanja.

V primerjavi z meritvami v članku “A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing” [3], kjer so namereli 0.6 GOPS je naš rezultat z brezplačnim paketom t2.micro približno 0.42 GOPS. To smo izračunali s pomočjo Leibnizove metode implementirane v c-ju. Celoten čas izračuna je približno 0,12 s, za izračun je pa potrebnih približno 25 milijonov seštevanj in deljenj. V enem izračunu se torej zgodi približno 50 milijonov operacij, opravimo pa lahko 8,3 izračunov na sekundo kar pomeni da dosežemo 0,416 GOPS. Teoretična maksimalna zmogljivost procesorja je 4,4 GOPS [3], ponudnik pa nam da na razpolago 10 odstotkov tega [8], torej 0,44 GOPS. To pomeni, da je bilo v času našega testiranja izkoriščenih 94,5% računske moči.

Literatura

[1] Wikipedia contributors, "Leibniz formula for pi — Wikipedia, the free encyclopedia," 2018. [Online; dostopano 2-April-2018].

[2] Wikipedia contributors, "Bellard's formula — Wikipedia, the free encyclopedia," 2018. [Online; dostopano 2-April-2018].

[3] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of ec2 cloud computing services for scientific computing," in *International Conference on Cloud Computing*, pp. 115–131, Springer, 2009.

[4] "Amazon web services." https://aws.amazon.com/. [Online; dostopano 2-April-2018].

[5] "Microsoft azure." https://azure.microsoft.com/en-gb/. [Online; dostopano 2-April-2018].

[6] "Aws cloud9." https://aws.amazon.com/cloud9/?origin=c9io. [Online; dostopano 2-April-2018].

[7] M. Moškon and M. Mraz, *Modeliranje računalniških omrežij*. Založba FE in FRI, 2012.

[8] "Amazon elastic compute cloud." https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/t2-credits-baseline-concepts.html. [Online; dostopano 30-Maj-2018].