Analiza zmogljivosti oblačnih in strežniških storitev

Uredil prof. dr. Miha Mraz

Maj 2017

Kazalo

Predgovor **iii**

1 Analiza zmogljivosti oblačne storitve DigitalOcean **1**
1.1 Opis problema 1
1.2 Izbira tehnologij in algoritmov 1
1.2.1 Tehnologija na strani strežnika 1
1.2.2 Tehnologija za avtomatizacijo odjemalcev 2
1.2.3 Generiranje vhodnih podatkov 2
1.2.4 Izbrani algoritmi za sortiranje 3
1.3 Izbira ponudnikov 4
1.4 Rezultati meritev 5
1.4.1 Stevilo odjemalcev - Eksperiment 1 5
1.4.2 Velikost datotek - Eksperiment 2 6
1.4.3 Ozko grlo na strežniku - Eksperiment 3 8
1.4.4 Lokalnost - Eksperiment 4 9
1.4.5 Različne računske moči - Eksperiment 5 10

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 2016/2017 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 2017

Poglavje 1

Analiza zmogljivosti oblačne storitve DigitalOcean

Žiga Kokelj, Tadej Hiti,
Miha Bizjak, Matej Kristan

1.1 Opis problema

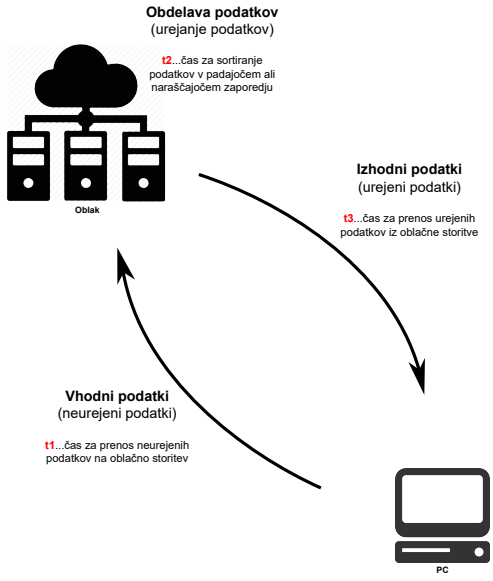
V zadnjem desetletju se na vseh področjih računalništva vse bolj uveljavljajo oblačne storitve in koncept odjemalec - strežnik (angl. client - server). Z večanjem hitrosti internetnih povezav končne delovne točke(predvsem osebni računalniki) izgubljajo del svojih primarnih funkcij. Hranjenje in obdelavo podatkov vse bolj prepuščajo oblačnim storitvam. V naši analizi smo se osredotočili na obdelavo podatkov na strani strežnika. Za breme smo izbrali datoteke različnih velikosti, ki vsebujejo naključna števila. Te datoteke odlemalci pošlejo na strežnik, ki jih mora urediti v naraščajocem vrstnem redu in poslati nazaj. Na sliki 1.1 je grafičen prikaz opisanega problema. Naše testiranje bo obsegalo merjenje čase čakanja klientov na urejeno datoteko ter merjenje obremenjenosti strežnika.

1.2 Izbira tehnologij in algoritmov

V tem razdelku so opisane tehnologije in algoritmi, ki smo jih uporabili za našo analizo.

1.2.1 Tehnologija na strani strežnika

Na oblačni stortivi smo implementirali strežnik, ki je napisan v jeziku JavaScript [3] z uporabo knjižnice Node.js. Strežnik caka na HTTP zahteve. Ko strežnik



Slika 1.1: Shema delovanja sistema.

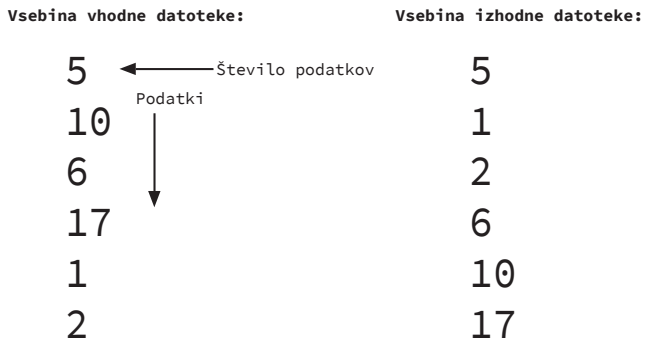
sprejme datoteko se začne izvajati program, ki števila v datoteki uredi po po vrsti. Ker je za ta program bistveno, da se izvede čim hitreje smo za ta namen uporabili programski jezik C [4], ki spada med najhitrejše programske jezike. Po končanem sortiranju strežnik urejeno datoteko pošlje nazaj odjemalcu.

1.2.2 Tehnologija za avtomatizacijo odjemalcev

Zaradi avtomatskega testiranja smo napisali skripto v programskem jeziku Python, ki omogoča avtomatsko pošiljanje datotek na strežnik, ter počaka na odgovor (datoteka z urejenimi števili). Seveda ob tem zabeležimo še čas pred pošiljanjem zahteve in čas po prejetju urejene datoteke, da dobimo celotni čas potreben za to, da dobimo urejeno datoteko. Ker je odjemalcev lahko večje število, smo ta problem rešili z nitmi, kjer vsaka nit predstavlja enega odjemalca in pošilja zahteve na strežnik preko istega IP naslova.

1.2.3 Generiranje vhodnih podatkov

V programskem jeziku C smo napisali generator datotek, ki ustvari datoteko željene velikosti z naključnimi števili. Za teste smo generirali datoteke velikosti 10000, 20000, 30000, 40000 in 50000 števil tipa integer.



Slika 1.2: Slika prikazuje primer datoteke s katero operirata strežnik in odjemalec.

1.2.4 Izbrani algoritmi za sortiranje

Za namen testiranja smo implementirali dva algoritma. Najprej smo testirali z algoritmom za mehurčno urejanje(bubble sort), ki ima časovno zahevnost $O(n^2)$. Da bi testirali vpliv izbire algoritma smo implemenirali tudi algoritem quicksort z povprečno časovno zahtevnostjo $O(n * log(n))$

```
function sort ( int[] data )
    for ( int i = 0; i < data.length; i++ )
        for ( int j = 0; j < data.length; j++ )
            if ( data[i] < data[j] )
                int tmp = data[i];
                data[i] = data[j];
                data[j] = tmp;
            end if;
        end for;
    end for;
end;
```

Slika 1.3: Slika prikazuje algoritem sortiranja z $O(n^2)$ časovno zahtevnostjo.

```
function quicksort ( int[] data , int lo, int hi )
    if ( lo < hi )
        int p = partition(data, lo, hi);
        quicksort(data, lo, p - 1);
        quicksort(data, p + 1, hi);
    end if;
end;

function partition(int[] data, int lo, int hi)
    int pivot = data[hi];
    int i = lo - 1;
    for( int j = lo; j < hi; j++ )
        if ( data[j] <= pivot )
            i = i + 1;
            if ( i != j )
                swap(data[i], data[j]);
            end if;
        end if;
    end for;
    swap(data[i+1], data[j]);
    return i + 1 ;
end;
```

Slika 1.4: Slika prikazuje algoritem sortiranja z $O(n * log(n))$ časovno zahtevnostjo.

1.3 Izbira ponudnikov

Najprej smo izbrali ponudnika oblačnih storitev Cloud9, ki pa ni najbolj primeren za našo nalogo, saj je v prvi vrsti razvojno okolje na spletu in ne toliko ponudnik strežniskih storitev. Pri računsko zahtevnejših testih smo naleteli na težave, saj je strežnik po 120 sekundah povezavo zaprl. To je bil eden glavnih razlogov, da smo se odločili zamenjati ponudnika, saj smo le tako lahko izvedli vse željene teste. Izbrali smo ameriško podjetje DigitalOcean, ki je ponudnik istoimenske oblačne infrastrukture. Na našo srečo sodelujejo v projektu Github Education Pack in tako študentom nudijo 50$ kredita za uporabo njihovih storitev. Svoje strežnike ima postavljene na osmih lokacijah po vsem svetu.

- New York (ZDA)
- San Francisco (ZDA)
- Amsterdam (Nizozemska)
- Singapur (Singapur)
- London (Združeno kraljestvo)
- Fraknfurt (Nemčija)
- Bangalore (Indija)

V okviru študentskih kreditv so nam na voljo tri različne konfiguracije, ki se med seboj razlikujejo po količini dodeljenih resursov.

RAM	Stevilo procesorjev	Prostor na disku (SSD)	Cena na mesec
512 MB	1	20 GB	5$
1 GB	1	30 GB	10$
2 GB	2	40 GB	20$

Slika 1.5: Konfiguracije strežnikov, ki so na voljo za študentske kredite.

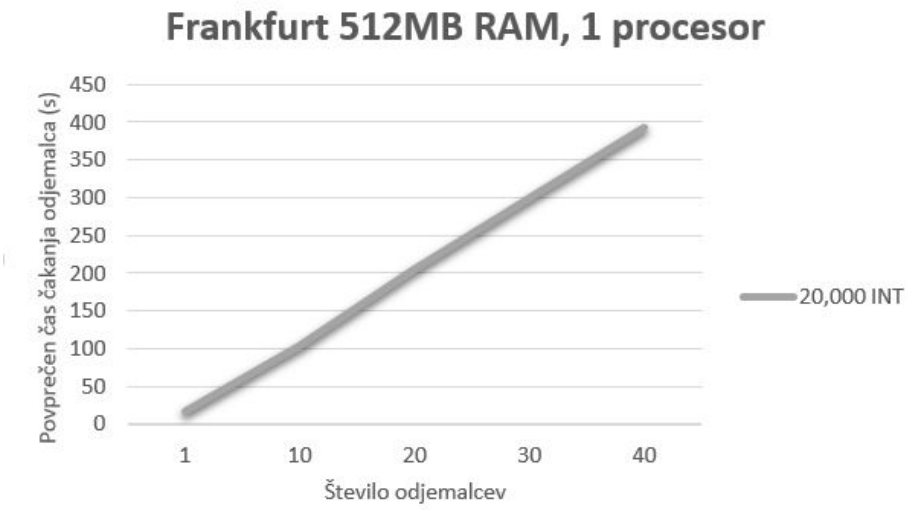
1.4 Rezultati meritev

1.4.1 Stevilo odjemalcev - Eksperiment 1

- **Hipoteza:**
  Naša hipoteza je, da se povprečen čas čakanja odjemalca približno linearno povečuje z večanjem stevila odjemalcev, ki hkrati pošiljajo datoteke na strežnik.

- **Okoliščine meritve:**
  Testiranje smo izvedli v četrtek 18.5.2017 med 21:00 in 21:30 na strežniku v Frankfurtu. Na strežniku je tekel operacijski sistem Linux Ubuntu 16.04. Na voljo smo imeli 512 MB RAM-a, 1 procesor ter 20 GB prostora na SSD trdem disku. Na strežniku je bil izbran algoritem za mehurčno urejanje (bubble sort). Skripto za simulacijo odjemalcev smo pognali iz študentskega naselja Rožna dolina (Ljubljana). Da dostop je bila uporabljna internetna povezana s hitrostjo 100 Mbps. Na strežnik smo pošiljali datoteke, ki so vsebovale 20,000 števil tipa integer in merili čase pri 1, 10, 20, 30 in 40 odjemalcih.

- **Rezultati meritve:**



Slika 1.6: Graf povprečnega časa cakanja odjemalcev v odvisnosti od števila odjemalcev.

Število odjemalcev	Povprečni cas obdelave [s]	Standardna deviacija
1	16.059	0
10	103.792	1.625
20	205.444	4.881
30	300.890	12.827
40	393.277	14.598

Slika 1.7: Tabela časov obdelav datoteke z 20 tisoč celimi števili.

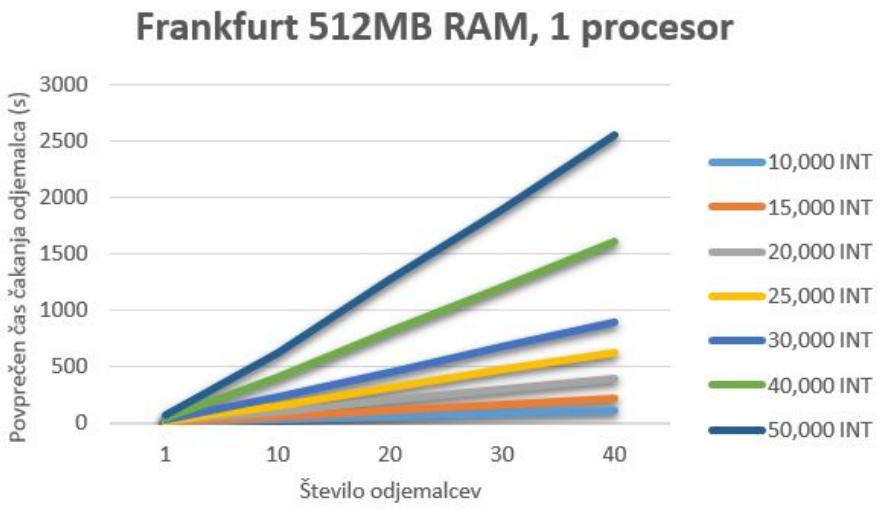
- **Komentar meritve:**
  Iz grafa lahko zelo razločno vidimo, da se z večanjem števila odjemalcev linearno povečuje tudi povprečen čas čakanja posameznega odjemalca. To je rezultat, ki smo ga napovedali že v hipotezi, saj ima strežnik omejeno število resursov in morajo zato ob večjem stevilu zahtevkov odjemalci dlje čakati. Prav tako lahko v tabeli opazimo, da se z večanjem stevila odjemalcev povečuje tudi standardna deviacija. To pomeni, da ob višjih obremenitvah (več odjemalcev) vedno težje predvidimo čas obdelave datotek določenega klenta, saj prihaja do vedno večjih odstopanj.

1.4.2 Velikost datotek - Eksperiment 2

- **Hipoteza:**
  Z tem eksperimentom smo želeli preveriti odvisnost povprečnega časa čakanja klientov glede na velikost datotek. Da bi se izničili vpliv števila

odjemalcev na ta poizkus smo teste ponovili na več različnih stevilih odjemalcev. V tem primeru postavljamo hipotezo, da se z večanjem velikosti datoteke veča tudi povprečni čas čakanja klientov.

- **Okoliščine meritve:**
  Testiranje smo izvedli v četrtek 18.5.2017 med 21:30 in 24:00 na strežniku v Frankfurtu. Na strežniku je tekel operacijski sistem Linux Ubuntu 16.04. Na voljo smo imeli 512MB RAM-a, 1 procesor ter 20GB prostora na SSD trdem disku. Na strežniku je bil izbran algoritem za mehurčno urejanje (bubble sort). Skripto za simulacijo odjemalcev smo pognali iz študentskega naselja Rožna dolina (Ljubljana). Da dostop je bila uporablejna internetna povezana s hitrostjo 100Mbps. Na strežnik smo pošiljali datoteke, ki so vsebovale 10,000, 15,000, 20,000, 25,000, 30,000, 40,000 in 50,000 števil tipa integer. Meritve smo ponovili na 1, 10, 20, 30 in 40 odjemalcih.
- **Rezultati meritve:**



Slika 1.8: Graf povprečnega časa cakanja odjemalcev v odvisnosti od velikosti datotek.

- **Komentar meritve:**
  Iz grafa zelo razločno vidimo, da se povprečen čas čakanja posameznega odjemalca z večanjem velikosti datotek zelo hitro povečuje. Pri 40 odjemalih in datoteki velikosti 10,000 stevil tipa integer je povprečni čas čakanja približno 170 s. Pri datoteki veliosti 50,000 stevil pa ta čats znaša že več kot 2550 s. To je bilo pričakovano, saj je uporabljen algoritem za mehurčno

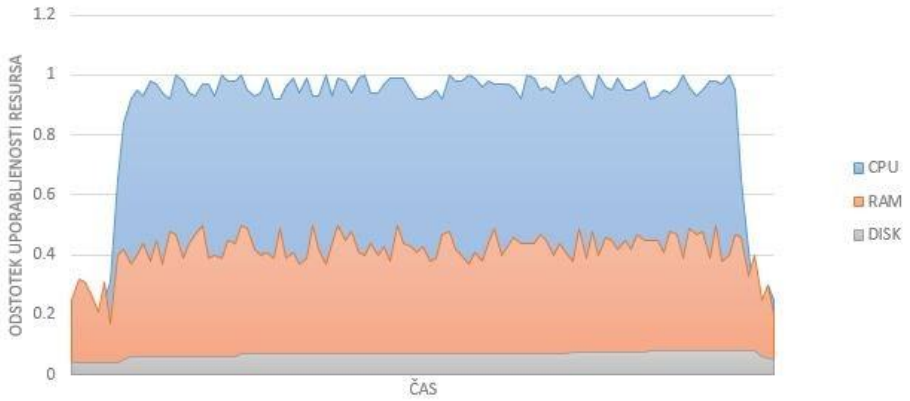
urejanje (bubble sort), ki ima časovno zahtevnost $O(n^2)$. Rezultati tega eksperimenta potrdijo, da se tudi na oblaku potreben čas za obdelavo datotek z algoritmom bubble sort povečuje približno z kvadratom velikosti datotek.

1.4.3 Ozko grlo na strežniku - Eksperiment 3

- **Hipoteza:**
  Z našimi meritvami smo strežnike obremenili do najvišjih dovoljenih meja. V praksi ob maksimani obremenitvi sistema vedno obstaja neko ozko grlo sistema, ki ne more delovati hitreje in z tem postavlja neko mejo hitrosti delovanja sistema. V našem primeru predpostavljamo, da je najbolj obremenjena komponenta procesor, saj je sortiranje operacija, ki zahteva relativno veliko računske moči.

- **Okoliščine meritve:**
  Obremenitve strežnika smo merili med Eksperimentom 2 (poglavje 1.4.2).

- **Rezultati meritve:**



Slika 1.9: Graf zasedenosti resursov med izvajanem testovs.

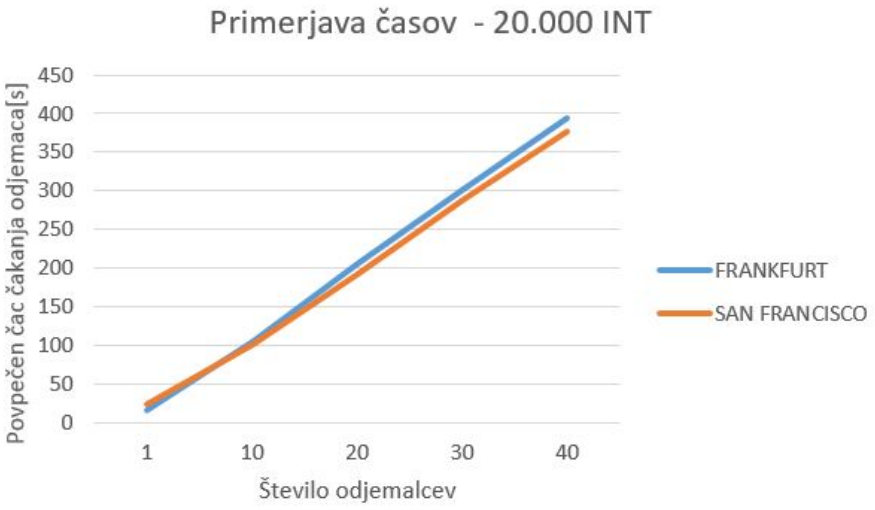
- **Komentar meritve:**
  Iz grafa 1.9 lahko razločno razberemo, da je ozko grlo našega sistema procesor, saj je praktično ves čas izvajanja testov 100% obremenjen. Poraba RAM-a in kapacitete trdega diska niso nikoli ekstremo obremenjene, saj med izvajanem nikoli ne uporabljamo več kot 50% RAM-a. Zasedenost diska pa je praktično zanemarljiva, saj so datoteke relativno majhne glede na ponujeno velikost trgeda diska, datoteke pa se sproti brišejo in tako preprečimo težave z zasedenostjo trdega diska.

1.4.4 Lokalnost - Eksperiment 4

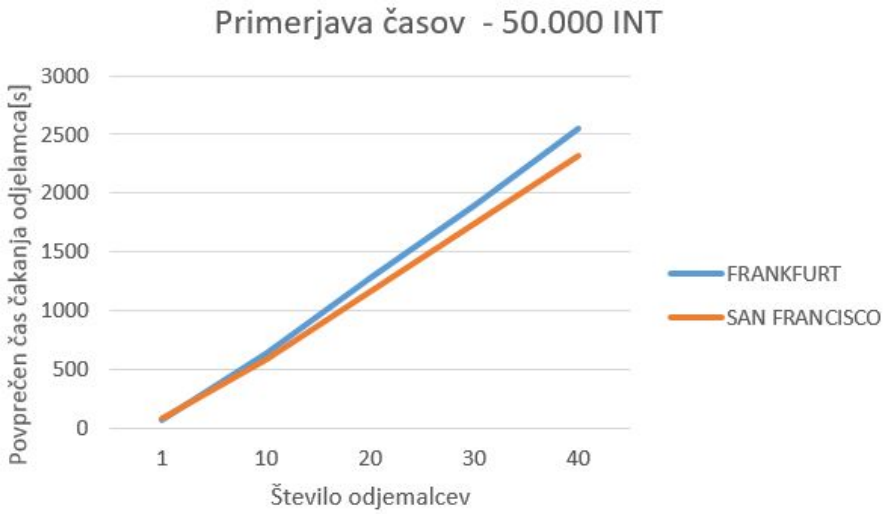
- **Hipoteza:**
  Zaradi možnosti izbire lokacije strežnika smo hoteli preveriti vpliv lokacije na povprečni čas čakanja odjemalcev. Poleg strežnika v Frankfurtu smo vzpostavili tudi strežnik v San Franciscu. Naša hipoteza je, da bomo hitrejše odzive dobivali od strežnika v Frankfurtu, saj nam je fizično precej bližje, kot San Francisco.

- **Okoliščine meritve:**
  Strežnika imata dodeljeno enako količčino resursov in nameščeno identično programsko opremo. Ta je enaka kot v testu 2 (poglavje 1.4.2) To je bilo potrebno zagotoviti, da izključimo vpliv programske opreme na hitrost odzivov strežnika. Testiranje smo izvedli v ponedeljek 22.5.2017 med 10:00 in 14:00 po času UTC+01:00. Zaradi časovnih pasov je bila ura v San Franciscu v času testiranja med 1:00 in 5:00 zjutraj. Teste smo ponovili z datotekami velikosti 20,000 in 50,000 celih stevil.

- **Rezultati meritve:**



Slika 1.10: Graf povprečnega časa cakanja odjemalcev v odvisnosti od lokacije na primeru datoteke z 20,000 števili



Slika 1.11: Graf povprečnega časa cakanja odjemalcev v odvisnosti od lokacije na primeru datoteke z 50,000 števili

- **Komentar meritve:**
  Rezultati meritev niso potrdili naše hipoteze, saj je je strežnik v San Franciscu kljub večji oddaljenosti hitreje odzival in vračal urejene datoteke. Razlike so vse bolj opazne ob večji obremenitvi sistema (večje datoteke in več odjemalcev). Pri datoteki velikost 20,000 števil in enem odjemalcu lahko vidimo, da nam rezultate hitreje vrne strežnik v Frankfurtu. Rezultate si razlagamo na način, da je bil strežnik v Frankfurtu bolj obremenjen v času izvajanja testov. Pri majhnih primerih je kljub svoji obremenjenosti uspel rezultat vrniti pred strežnikom v San Franciscu. Z večanjem računske zahtevnosti problema pa je vse bolj do izraza prišla manjša zasedenost infrastruktire v San Franciscu. Drug možen razlog bi bili lahko zmoglivejši procesorji v strežnikih lociranih v San Franciscu, vendar interne informacije o zgradbi strežniških centrov žal niso javne.

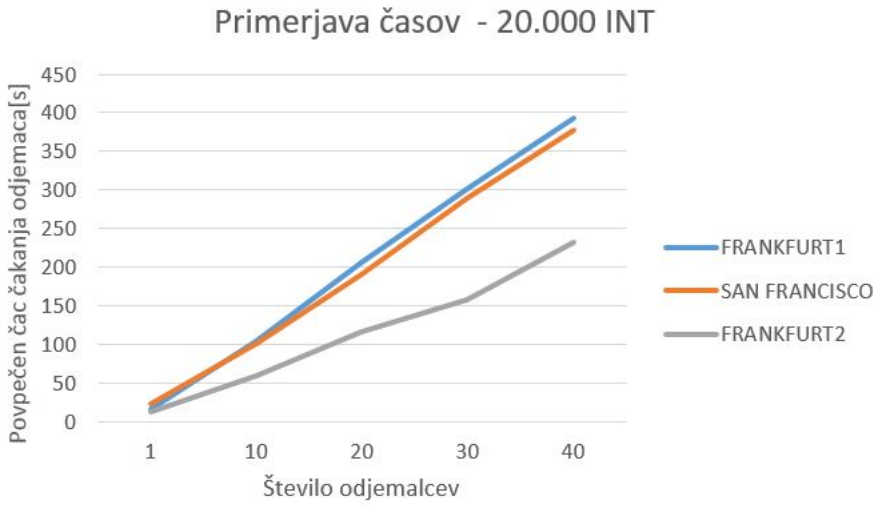
1.4.5 Različne računske moči - Eksperiment 5

- **Hipoteza:**
  Naša hipoteze, ki jo bomo prevelili v tem poizkusu je, da se povprečni čas čakanja odjemalcev bistveno zmanjša, če je strežnik bolj zmogljiv.

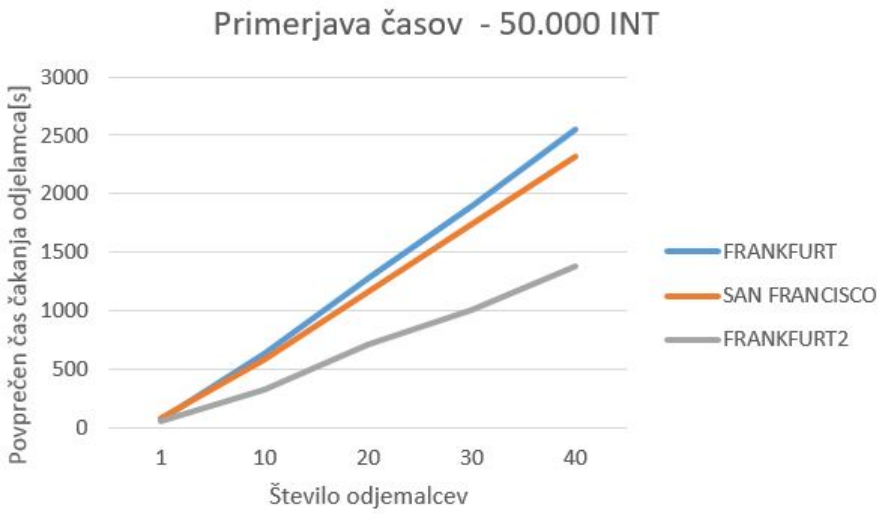
- **Okoliščine meritve:**
  Testiranje smo izvedli v torek 23.5.2017 med 00:30 in 02:30 na strežniku v Frankfurtu. Na strežniku je tekel operacijski sistem Linux Ubuntu 16.04. Na voljo smo imeli 2 GB RAM-a ter 2 procesorja. Na strežniku je bil

izbran algoritem za mehurčno urejanje (bubble sort). Skripto za simulacijo odjemalcev smo pognali iz študentskega naselja Rožna dolina(Ljubljana). Da dostop je bila uporabljena internetna povezana s hitrostjo 100Mbps. Na strežnik smo pošiljali datoteke, ki so vsebovale 20,000 in 50,000 števil tipa integer in merili čase pri 1, 10, 20, 30 in 40 odjemalcih.

- **Rezultati meritve:**
  V spodnjig grafih so vizualizani povprečni casi odziva treh različnih streznikov. Streznik Frankfurt1 in San Francisco imata identično konfiguracijo, kot je opisana v eksperimentu 4 (poglavje 1.4.4). Strežnik Frankfurt2 pa ima na voljo 2 procesorja ter 2 GB RAM-a.



Slika 1.12: Graf povprečnega časa cakanja odjemalcev treh različnih strežnikov na primeru datotek z 20,000 števili.



Slika 1.13: Graf povprečnega časa cakanja odjemalcev treh različnih strežnikov na primeru datotek z 50,000 števili.

- **Komentar meritve:**
  Meritve potrdijo našo predpostavko, da višja zmogljivost strežnikov pripomore k bistveno nižjim povprečnim časom potrebnim za odziv.

Literatura

[1] "Welcome to Python.org." https://www.python.org/, Marec 2017.

[2] "Node.js." `https://nodejs.org/`, Marec 2017.

[3] "JavaScript." `https://en.wikipedia.org/wiki/JavaScript`, Marec 2017.

[4] "C (programming language)." `https://en.wikipedia.org/wiki/C_(programming_language)`, Marec 2017.