Analiza zmogljivosti oblačnih in strežniških storitev

Uredil prof. dr. Miha Mraz

Maj 2017

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 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

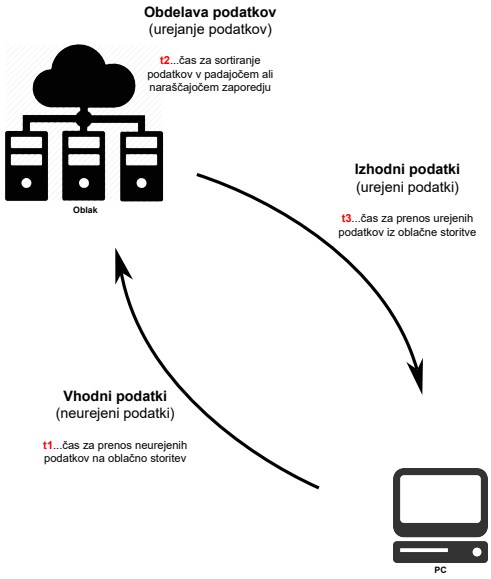
Žiga Kokelj, Tadej Hiti,
Miha Bizjak, Matej Kristan

1.1 Opis problema

Današnje dni se vse bolj uveljavljajo oblačne storitve [1], saj so s stališča uporabnika najenostavnejše za uporabo. Poleg tega imamo na voljo vse hitrejše internetne povezave. Posledično računska moč klientov pada in se to delo prenaša na oddaljene strežnike. Naša naloga je implementirati prenos datoteke na oblačno storitev, nad to datoteko na strežniku izvesti neke operacije ter obdelano datoteko poslati nazaj klientu. Naša operacija nad datoteko bo sortiranje števil. Na sliki 1.1 je grafičen prikaz opisanega problema. Naše testiranje bo obsegalo merjenje različnih izvajalnih časov na podlagi katerih bi prišli do podatkov o zmogljivosti sistema. Breme oblačnega sistema bodo datoteke različnih velikosti, ki bodo vsebovale naključno generirana števila in odjemalci, ki bodo pošiljali datoteke. Oblačna storitev bo vsebine datotek uredila po izbranem algoritmu za urejanje števil. Namen naše naloge je ugotoviti zmogljivost zastonjske oblačne storitve z vidika različnih metrik.

1.2 Izbira ponudnikov

Zaradi brezplačne ponudbe preko Github Education Pack paketa za študente smo se odločili za oblačno storitev DigitalOcean. DigtalOcean je ameriško podjetje, ki ponuja istoimenske oblačne stortive. Na strani strežnika nam ponuja



Slika 1.1: Shema delovanja sistema.

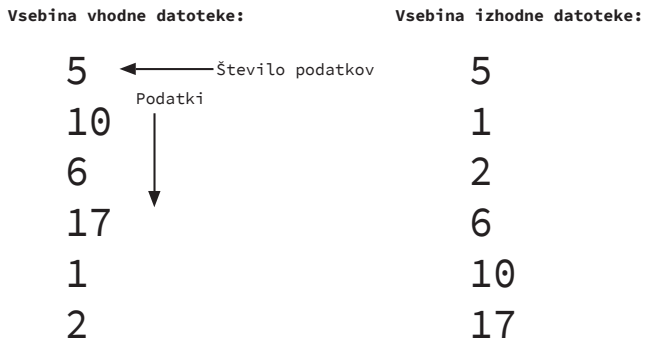
izbiro različnih distribucij operacijskega sistema Linux (Ubuntu, FreeBSD, Fedora, Debian, CentOS, ...). V naši brezplači verziji imamo na voljo 512MB RAM-a, 1 procesor, 20GB prostora na SSD disku ter 1TB prenosa podatkov na mesec. V plačljivi različici pa je na voljo vse do 64GB RAM-a, 20 procesorjev, 640GB prostora na SSD disku ter kar 9TB prenosa podatkov. Poleg velike prilagodljivosti glede zmogljivosti naše rešitve pa imamo veliko možnosti tudi pri izbiri fizične lokacije strežnikov. Izbiralo lahko med Ney Yorkom, San Franciscom, Amsterdamom, Singapurjem, Londonom, Frankfurtom, Torontom in Bangalorejem.

1.3 Izbira tehnologij

V tem razdelku so na kratko opisane vse izbrane tehnologije, ki smo jih realizirali za našo analizo.

1.3.1 Tehnologija na strani strežnika

Na oblačni stortivi DigialOcean smo implementirali strežnik, ki je napisan v jeziku javascript z uporabo knjižnice Node.js [2]. Strežnik čaka na HTTP zahteve. Na 'POST /' zahtevo odgovori z html datoteko, preko katere lahko naložimo datoteko, in jo s klikom na gumb pošljemo s 'POST /nalozi' zahtevo na strežnik.



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

Ko strežnik sprejme datoteko zažene program napisan v programskem jeziku C, ki uredi podatke po vrsti v novo datoteko z algoritmom prikazanim na sliki 1.3, z $O(n^2)$ časovno zahtevnostjo. Program prejeto datoteko bere po vrsticah, kjer prva vrstica vsebuje število podatkov, naslednje vrstice pa podatke. Urejeno datoteko strežnik pošlje, kot odgovor odjemalcu.

1.3.2 Tehnologija za avtomatizacijo odjemalcev

Zaradi avtomatskega testiranja smo napisali skripto v programskem jeziku python [3], ki omogoča avtomatsko pošiljanje datoteke preko URL 'POST /nalozi' zahteve na strežnik, ter počaka na datoteko z urejenimi podatki, kot odgovor. Seveda ob tem zabeležimo še čas pred pošiljanjem zahteve in čas po prejetju urejene datoteke, da dobimo izvajalni čas celotne procedure. 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.

1.3.3 Generiranje vhodnih podatkov

V programskem jeziku C smo napisali generator datotek, ki ustvari datoteko Ľľeljene velikosti z naključnimi števili. Za Test1 smo generirali datoteke velikosti: 10000, 20000, 30000, 40000 in 50000 števil tipa integer.

1.3.4 Uporabljeni algoritmi za sortiranje

1.4 Testiranje I.

Testiranje je bilo izvedeno med 18:00 in 22:00 v četrtek 18. Maja 2017 iz študentskega naselja Rožna Dolina. Za dostop je bila uporabljana internetna povezava s hitrostjo prenosa 100Mbps. Strežnik teče na operacijskem sistemu Ubuntu

```
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.

16.04 in ima na voljo 512MB RAM-a, 1 procesor, 20GB prostora na SSD disku in je lociran v Frankfurtu(Nemčija). Na strežniku je bil izbran algoritem bubble sort 1.3.

1.4.1 Nacin testiranja

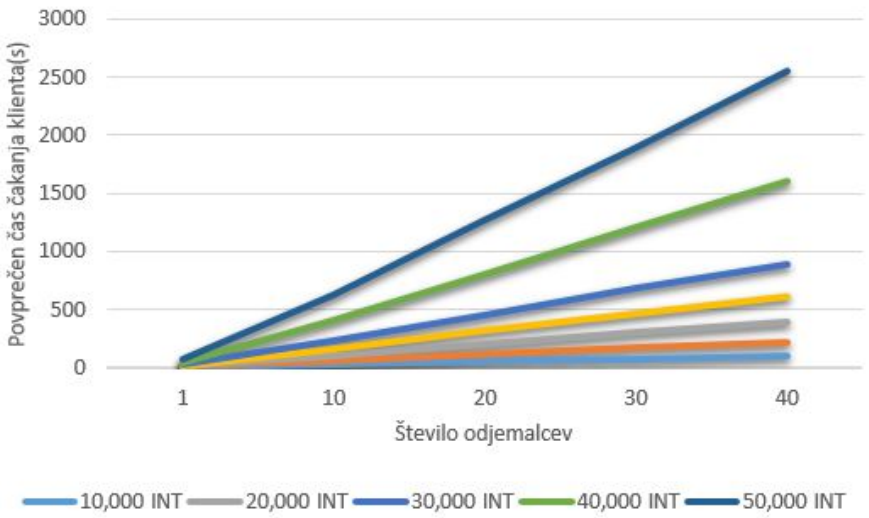
Testiranje smo izvedli s pomočjo generiranih datotek. Namenski program v Python-u je praktično hkrati poslal datoteko preko 1, 10, 20, 30 in 40 odjemalcev. Da bi zagotovili večjo točnost podatkov smo test ponovili s po petimi datotekami v vsakem velikostnem razredu. Izmerili smo vse povprečne čase ter iz njih izračunali povprečje ter standardno deviacijo, kar lahko vidite na spodnjem grafu **??** .

1.4.2 Rezultati meritev

Poleg ponujanja strežniških stortiev nam DigitalOcean nudi tudi grafični prikaz porabe posameznih resursov(CPE, RAM, DISK, ...). Grafe porabe resursov lahko vidite na spodnjem grafu.

1.4.3 Razlaga meritev

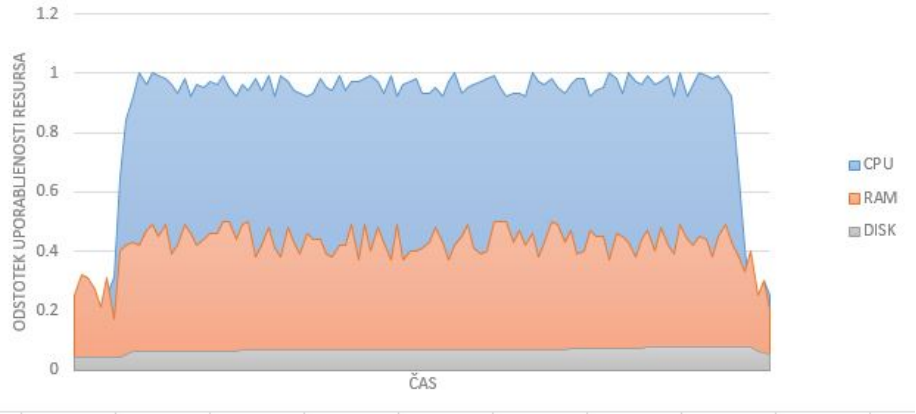
Ob ogladu meritev opazimo, da čas čakanja posameznega odjemalca na datoteke v povprečju narašča približni linearno glede na število odjemalcev, ki hkrati pošljejo datoteko. Naklon grafa je močno odvisen od velikosti sortiranih datotek in pri večjih datotekah na več odjemalcih precej naraste. Iz grafa zasedenosti resursov lahko razberemo, da je šibka tocka našega sistema procesor, saj je ta skoraj ves čas izvajanja testov 100% obremenjen. RAM nikoli med testiranjem



Slika 1.4: Graf časa obdelave v odvisnosti od števila odjemalcev in velikosti datotek.

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 1.5: Tabela časov obdelav datoteke z 20 tisoč integer števili.



Slika 1.6: Graf zasedenosti resursov med izvajanjem testov.

ne preseĹľe 50% zasedenosti. Najmanj obremenjena komponenta je trdi disk, saj vsako datoteko po obdelavi sproti izbriše.

1.5 Testiranje II.

Testiranje je bilo izvedeno med 10:00 in 14:00 (po lokalnem ÄŤasu 1:00 do 5:00) v ponedeljek 22. Maja 2017 iz študentskega naselja Rožna Dolina. Za dostop je bila uporabljana internetna povezava s hitrostjo prenosa 100Mbps. Strežnik teče na operacijskem sistemu Ubuntu 16.04 in ima na voljo 512MB RAM-a, 1 procesor, 20GB prostora na SSD disku in je lociran v San Franciscu(Vzhodna obala ZDA). Na strežniku je bil izbran algoritem bubble sort 1.3.

1.5.1 Nacin testiranja

Testiranje smo izvedli s pomočjo generiranih datotek. Namenski program v Python-u je praktično hkrati poslal datoteko na 1, 10, 20, 30 in 40 odjemalcev. Da bi zagotovili večjo točnost podatkov smo test ponovili s po petimi datotekami v vsakem velikostnem razredu. Izmerili smo vse povprečne case ter iz njih izračunali povprečje ter standardno deviacijo, kar lahko vidite na spodnjem grafu **??**.

1.5.2 Rezultati meritev

Poleg dobljenih časov meritev smo pridobili tudi podatke o zasedenosti resursov na streni strežnika. Zasedenost resursov je zelo podobna zasedenosti pri prvem testiranju.

1.5.3 Razlaga meritev

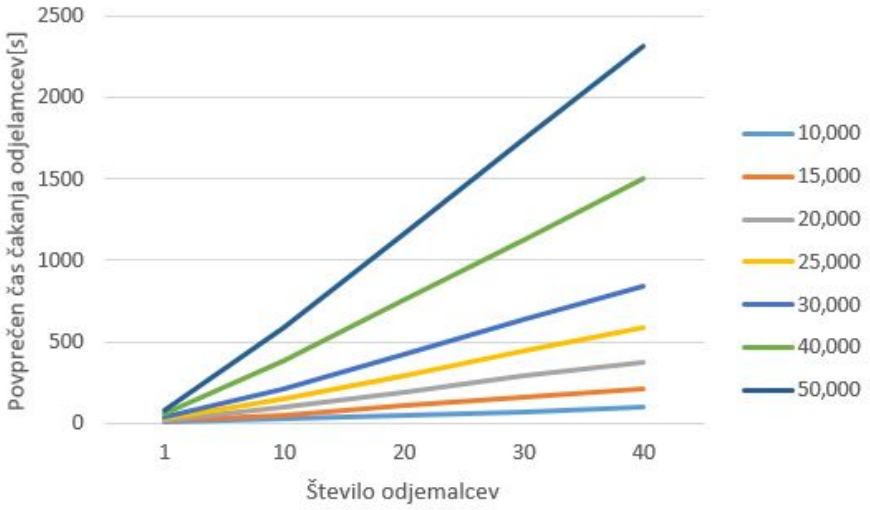
Rezultati meritev so zelo podobni rezultatom meritev na strežniko v Frankfurtu.. TODO: Dopiši razlago!

1.6 Testiranje III.

Testiranje je bilo izvedeno med 00:30 in 2:30 v torek 23. Maja 2017 iz študentskega naselja Rožna Dolina. Za dostop je bila uporabljana internetna povezava s hitrostjo prenosa 100Mbps in nalaganja 100Mbps. Strežnik teče na operacijskem sistemu Ubuntu 16.04 in ima na voljo 2GB RAM-a, 2 procesorja, 20GB prostora na SSD disku in je lociran v Frankfurtu(Nemčija). Na strežniku je bil izbran algoritem bubble sort 1.3

1.6.1 Nacin testiranja

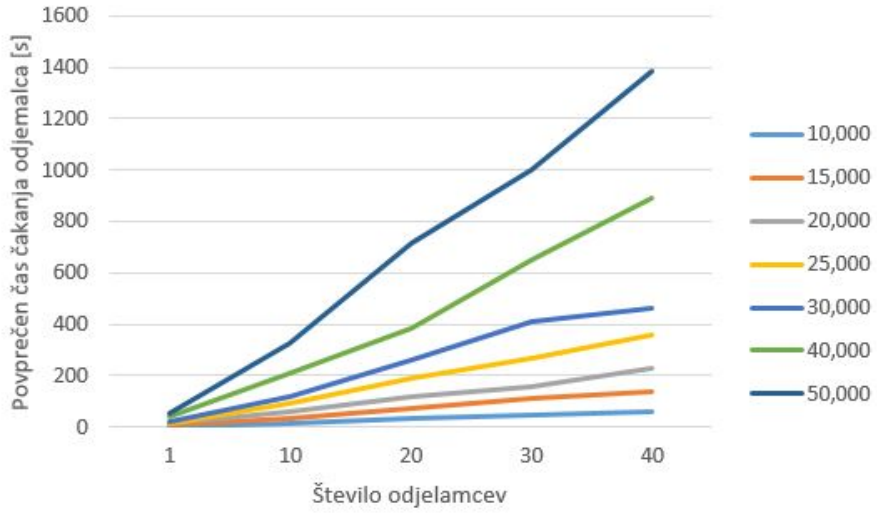
Testiranje smo izvedli s pomočjo generiranih datotek. Namenski program v Python-u je praktično hkrati poslal datoteko na 1, 10, 20, 30 in 40 odjemalcev.



Slika 1.7: Graf časa obdelave v odvisnosti od števila odjemalcev in velikosti datotek.

Število odjemalcev	Povprečni cas obdelave[s]	Standardna deviacija
1	22.659	0
10	100.001	6.552
20	190.552	10.279
30	288.468	8.757
40	377.072	19.697

Slika 1.8: Tabela časov obdelav datoteke z 20 tisoč integer števili.



Slika 1.9: Graf časa obdelave v odvisnosti od števila odjemalcev in velikosti datotek.

Da bi zagotovili večjo točnost podatkov smo test ponovili s po petimi datotekami v vsakem velikostnem razredu. Izmerili smo vse povprečne case ter iz njih izračunali povprečje ter standardno deviacijo, kar lahko vidite na spodnjem grafu ?? .

1.6.2 Rezultati meritev

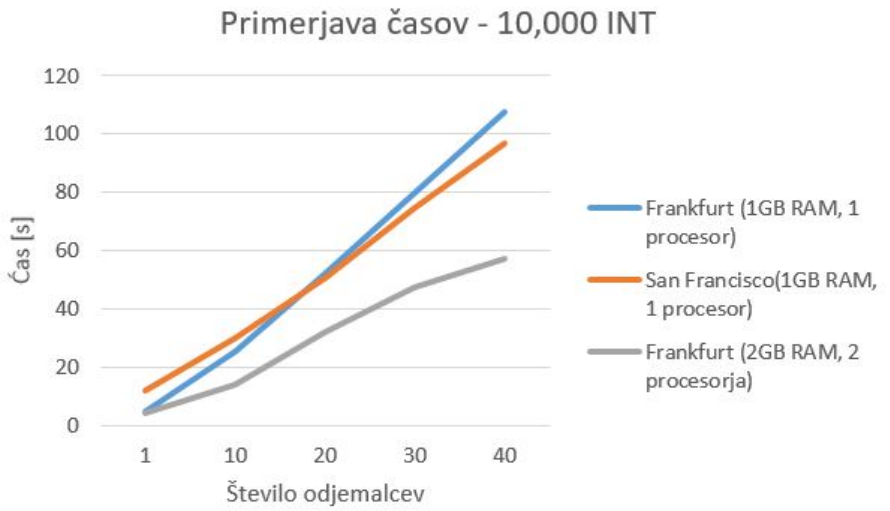
Poleg dobljenih časov meritev smo pridobili tudi podatke o zasedenosti resursov na streni strežnika. TODO: zasedenost resursov

Število odjemalcev	Povprečni cas obdelave[s]	Standardna deviacija
1	22.659	0
10	100.001	6.552
20	190.552	10.279
30	288.468	8.757
40	377.072	19.697

Slika 1.10: Tabela časov obdelav datoteke z 20 tisoč integer števili.

1.6.3 Razlaga meritev

Rezultati meritev so zelo podobni rezultatom meritev na strežniko v Frankfurtu.. TODO: Dopiši razlago!



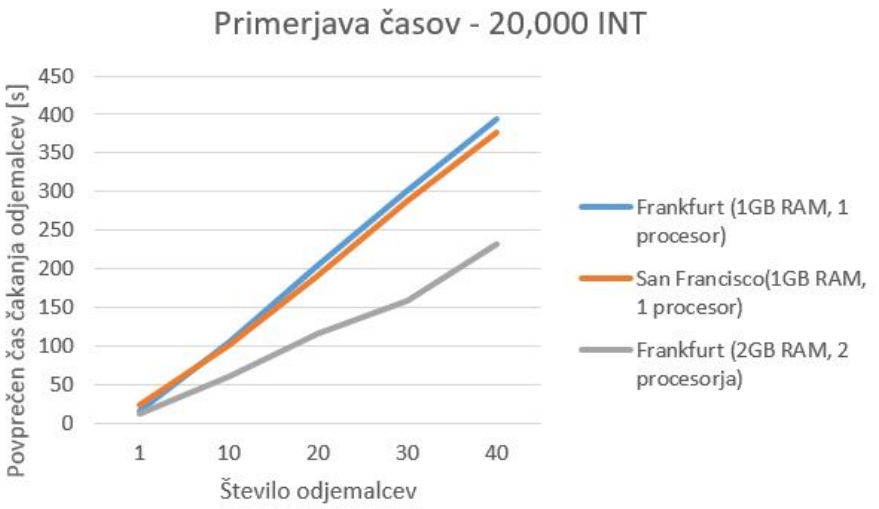
Slika 1.11: Primerjava povprečnih casov na treh različnih strežnikih.

1.7 Primerjava časov na različnih strežnikih z datotekami velikosti 10,000 INT

1.7.1 Graf

1.7.2 Razlaga

Pri majhnem številu odjemalcev opazimo razliko v temu, da imata oba strežnika v Frankfurtu nižje čase, kar lahko pojastnimo z bližjo fizično razdaljo med nami(klientom) in strežnikom. Pri višjem številu odjemalcev je hitrejši enako zmogljiv strežnik v San Franciscu, kot v Frankfurtu. To je najbrž posledica časa testiranja, saj je bil strežnik v Frankfurtu testiran zveÄŤer, v San Franciscu pa sredi noči, ko je obremenjenost praviloma manjša. Iz grafa lahko tudi lepo vidimo, da je čas cakanja klientov precej manjši pri pri zmogljivejšem strežniku. Razlika se pri večanju stevila odjemalcev še povečuje.



Slika 1.12: Graf časa obdelave v odvisnosti od števila odjemalcev in velikosti datotek.

1.8 Primerjava časov na različnih strežnikih z datotekami velikosti 20,000 INT

1.8.1 Graf

1.8.2 Razlaga

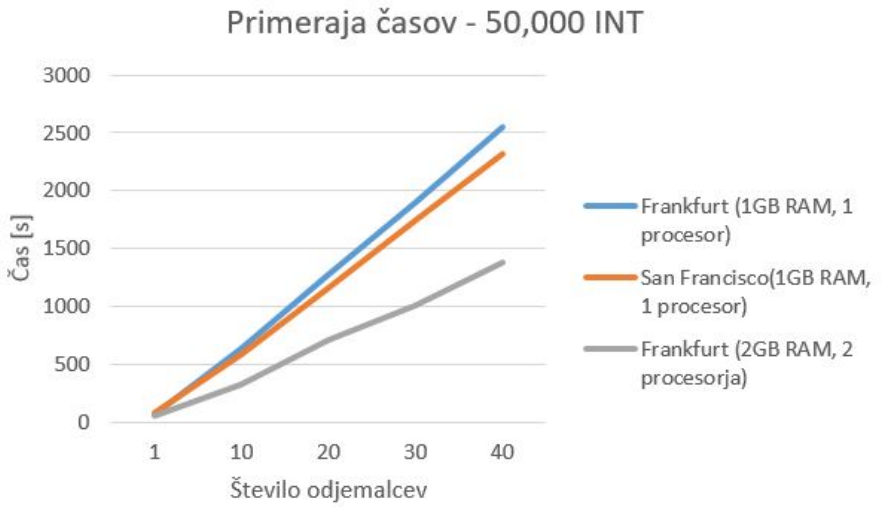
Iz grafa lahko razberemo, da je fizična oddaljenost strežnika velikega pomena le, dokler sistem ni popolnoma obremenjen. To se zgodi pri le enem klientu in tu strežnika v Frankfurtu hitreje opravita svoje delo, kot tisti v San Franciscu. Pri višjih obremenitvah pa bolj do izraza bolj pride obremenjenost strežnika. Še vedno je povprečni čas čakanja odvisen od zmogljivosti strežnika.

1.9 Primerjava časov na različnih strežnikih z datotekami velikosti 50,000 INT

1.9.1 Graf

1.9.2 Razlaga

TODO: razlaga



Slika 1.13: Graf časa obdelave v odvisnosti od števila odjemalcev in velikosti datotek.

Literatura

[1] "Cloud computing." https://en.wikipedia.org/wiki/Cloud_computing/, Marec 2017.

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

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