Analiza zmogljivosti oblačnih in strežniških storitev

Uredil prof. dr. Miha Mraz

Maj 2017

Kazalo

Predgovor

Priďž˝ujoďž˝e delo je razdeljeno v deset poglavij, ki predstavljajo analize zmogljivosti nekaterih tipiďž˝nih streďž˝niďž˝kih in obladž˝nih izvedenk raďž˝unalniďž˝kih sistemov in njihovih storitev. Avtorji posameznih poglavij so sluďž˝atelji predmeta *Zanesljivost in zmogljivost raďž˝unalniďž˝kih sistemov*, ki se je v ďž˝tud.letu 2016/2017 predaval na 1. stopnji univerzitetnega ďž˝tudija raďž˝unalniďž˝tva in informatike na Fakulteti za raďž˝unalniďž˝tvo in informatiko Univerze v Ljubljani. Vsem ďž˝tudentom se zahvaljujem za izkazani trud, ki so ga vloďž˝ili 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

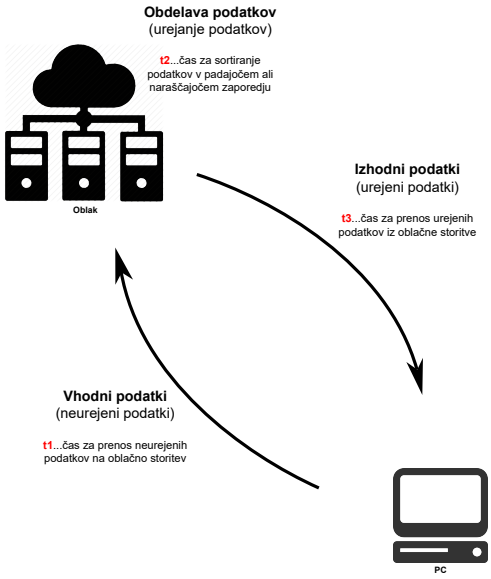
V zadnjem desetletju se na vseh področjih računalništva vse bolj uveljavljajo oblačne storitve in koncept odjemalec - strežnik (angl. *client - server*). Z večanjem hitrosti internetnih povezav končne delovne točke (predvsem osebni računalniki) izgubljajo del svojih primarnih funkcij. Hranjenje in obdelavo 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 odjemalci pošljejo na strežnik, ta pa jih uredi v naraščajočem vrstnem redu in pošlje nazaj. Na sliki 1.1 je grafičen prikaz opisanega problema. Naše testiranje bo obsegalo merjenje časov čakanja odjemalcev 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 storitvi smo implementirali strežnik, ki je realiziran v jeziku JavaScript [1] z uporabo knjižnice Node.js [2]. Strežnik čaka na HTTP zahteve s



Slika 1.1: Shema delovanja sistema.

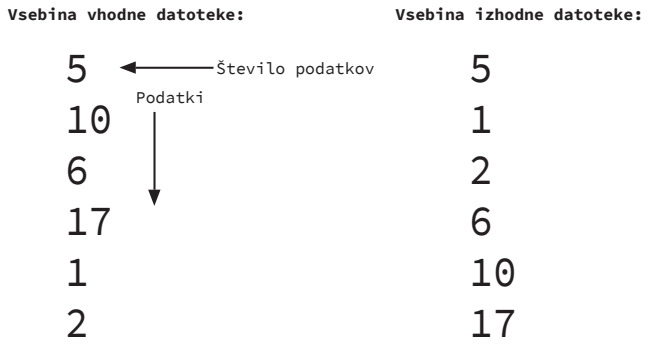
strani klienta, kateri pošlje datoteko števil za urejanje. Ob sprejemu datoteke s strani strežnika se začne izvajati program, ki števila v datoteki uredi v naraščajočem vrstnem redu. Ker je za namene testiranja oblačne infrastrukture bistveno, da se program čim hitreje izvrši smo za ta namen uporabili programski jezik C [3], ki spada med najhitrejše programske jezike, zaradi njegove nizko-nivojske lege na procesorski arhitekturi. 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 [4], ki omogoča avtomatsko pošiljanje datotek na strežnik, ter čaka na odgovor (urejeno datoteko s števili). Ob tem zabeležimo še čas pred pošiljanjem zahteve in čas po prejetju urejene datoteke, da dobimo celotni čas potreben za pošiljanje in sortiranje na strežniku. 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 (ang. *bubble sort*), ki ima časovno zahevnost $O(n^2)$. Da bi testirali vpliv izbire algoritma smo tudi implemenirali hitrejši algoritem urejanja s porazdelitvami (ang. *quicksort*), kateri ima povprečno časovno zahtevnost $O(n * log(n))$.

Listing 1.1: Bubblesort

```
function bubblesort(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;
         }
      }
   }
}
```

Listing 1.2: Quicksort

```
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);
   }
}

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){
            int tmp = data[i];
            data[i] = data[j];
            data[j] = tmp;
         }
      }
   }
   int tmp = data[i+1];
   data[i+1] = data[j];
   data[j] = tmp;
   return i+1;
}
```

1.3 Izbira ponudnikov

Najprej smo izbrali ponudnika oblačnih storitev Cloud9, ki pa ni najbolj primeren za naš problem, saj je v prvi vrsti razvojno okolje na spletu in se ne osredotoča na ponujanje strežniške infrastrukture. 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 za zamenjavo ponudnika oblačnih storitev, saj smo le tako lahko izvedli vse željene teste. Izbrali smo si 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 imajo postavljene na lokacijah naštetih v nadaljevanju:

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

V okviru študentskih kreditov so nam na voljo tri različne konfiguracije, ki se med seboj razlikujejo po količini dodeljenih resursov. Podatki o konfiguracijah so na voljo v tabeli 1.1.

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$

Tabela 1.1: Konfiguracije strežnikov, ki so na voljo za študentske kredite.

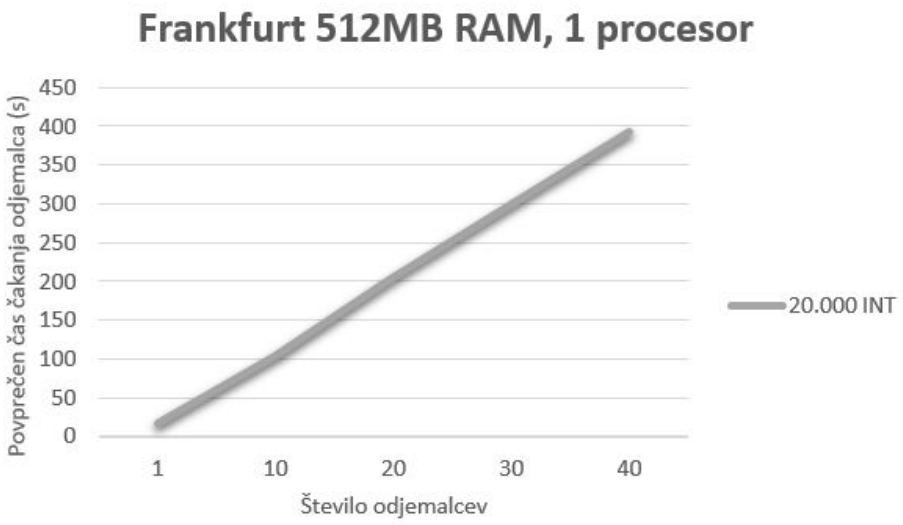
1.4 Rezultati meritev

1.4.1 Število odjemalcev - Eksperiment 1

- **Hipoteza:**
  Naša hipoteza je, da se povprečen čas čakanja odjemalca približno linearno povečuje z večanjem števila 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 (ang. *bubble sort*). Skripto za simulacijo odjemalcev smo pognali iz študentskega naselja Rožna dolina (Ljubljana). Za dostop do interneta je bila uporabljena internetna povezana s hitrostjo 100/100 Mbps.
  Na strežnik so klienti (niti) pošiljali datoteke, ki so vsebovale 20.000 števil tipa integer in merili čase pri 1, 10, 20, 30 in 40 odjemalcih.

- **Rezultati meritve:**
  Rezultati meritve so vidni na sliki 1.3.



Slika 1.3: Graf povprečnega časa čakanja odjemalcev v odvisnosti od števila odjemalcev.

Število odjemalcev	Povprečni cas obdelave [s]	Standardna deviacija
1	16,06	0
10	103,79	1,63
20	205,44	4,88
30	300,89	12,83
40	393,28	14,60

Tabela 1.2: Tabela časov obdelav datoteke z 20 tisoč celimi števili.

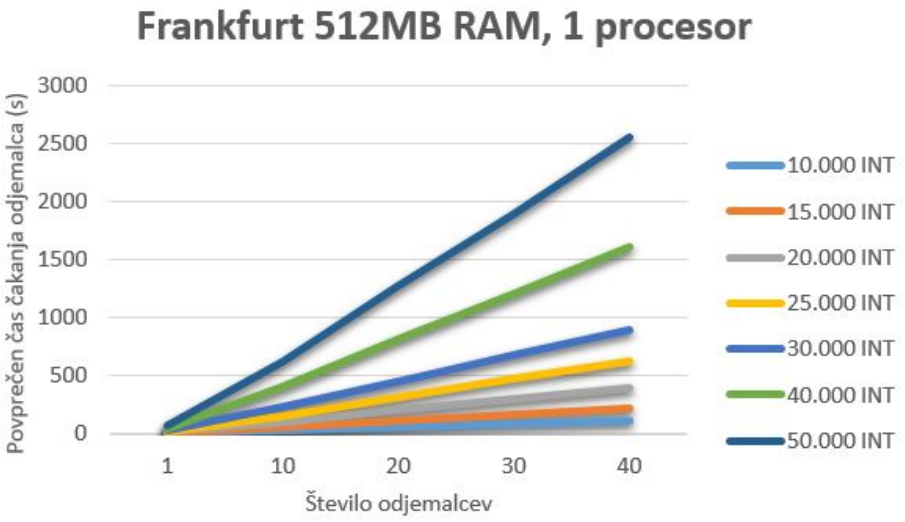
- **Komentar meritve:**
  Iz slike 1.3 in tabele 1.2 lahko zelo razločno vidimo, da se z večanjem števila odjemalcev linearno povečuje tudi povprečen čas čakanja posameznega odjemalca. Opažen rezultat pričakovano sovpada z našo hipotezo, saj ima strežnik omejeno število resursov in morajo zato ob večjem številu zahtevkov odjemalci čakati dlje. Prav tako lahko v tabeli opazimo, da se z večanjem števila odjemalcev povečuje tudi standardna deviacija, kar pa pomeni, da ob višjih obremenitvah (več odjemalcev) vedno težje predvidimo čas obdelave datotek določenega odjemalca, saj prihaja do vedno večjih odstopanj.

1.4.2 Velikost datotek - Eksperiment 2

- **Hipoteza:**
  S sledečim eksperimentom smo želeli preveriti odvisnost povprečnega časa čakanja klientov glede na velikost datotek. Da bi izničili vpliv števila odjemalcev na ta poizkus smo teste ponovili na več različnih številih odjemalcev. V tem primeru predpostavljamo 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 (ang. *bubble sort*). Skripto za simulacijo odjemalcev smo pognali iz študentskega naselja Rožna dolina (Ljubljana). Za dostop do interneta je bila uporabljena internetna povezana s hitrostjo 100/100 Mbps.
  Na strežnik so klienti (niti) pošiljali datoteke, ki so vsebovale velikosti 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 1.4: Graf povprečnega časa čakanja odjemalcev v odvisnosti od velikosti datotek.

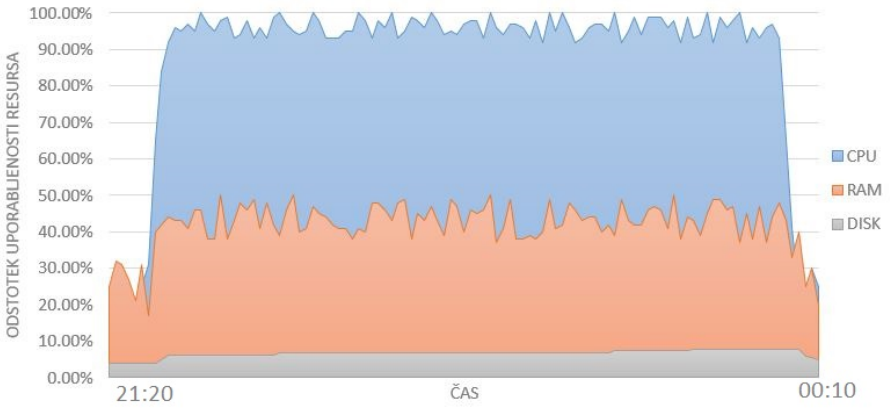
- **Komentar meritve:**
  Iz grafa na sliki 1.4 zelo razločno vidimo, da se povprečen čas čakanja posameznega odjemalca z večanjem velikosti datotek zelo hitro povečuje. Pri 40 odjemalcih in datoteki velikosti 10.000 števil tipa integer je povprečni čas čakanja približno 170 sekund. Pri datoteki veliosti 50.000 števil pa ta čas znaša že več kot 2550 sekund. To je bilo pričakovano, saj je bil za urejanje števil 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 s 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 maksimalni obremenitvi sistema vedno obstaja neko ozko grlo sistema, ki ne more delovati hitreje in s tem znižuje hitrosti delovanja sistema celotnega sistema. V našem primeru predpostavljamo, da je najbolj obremenjena komponenta procesor, saj je sortiranje operacija, ki zahteva relativno veliko računskih operacij.

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

- **Rezultati meritve:**



Slika 1.5: Graf zasedenosti resursov med izvajanem testovs.

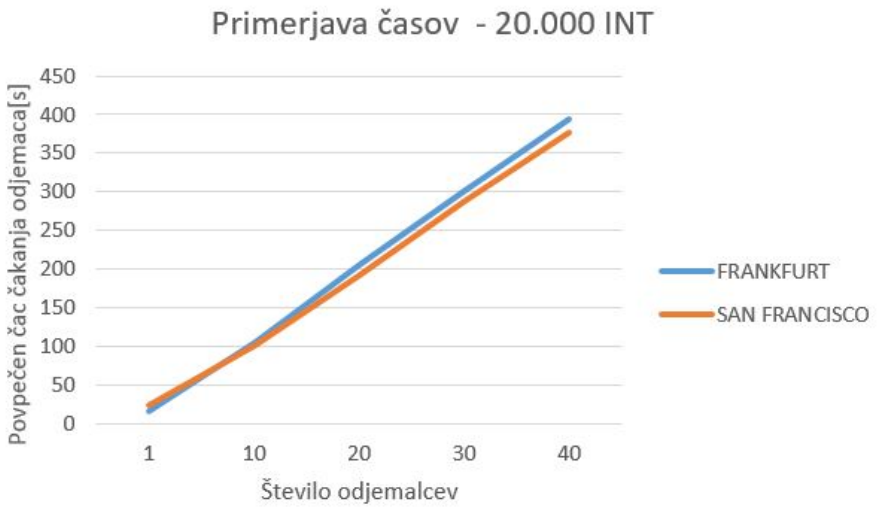
- **Komentar meritve:**
  Iz slike 1.5 lahko razločno razberemo, da je ozko grlo našega sistema procesor, saj je praktično ves čas izvajanja testov 100% obremenjen. Poraba RAM-a in kapacitete trdega diska niso nikoli ekstremno obremenjene, saj med izvajanjem nikoli ne uporabljamo več kot 50% RAM-a. Razlog za majhno porabo RAM-a je v tem, da odjemalci čakajo na odgovor strežnika pred pošiljanjem novih datotek. Tudi v primeru, ko odjemalci ne čakajo, pa nam ni uspelo doseči dovolj visoke zasedenosti, da bi zrušili sistem. Razloga za to sta omejeno število niti na odjemalcu ter omejeno število hkrati odprtih datotek na odjemalcu. Zasedenost diska pa je praktično zanemarljiva, saj so datoteke relativno majhne glede na ponujeno velikost trdega diska, datoteke pa se sproti brišejo in tako preprečimo težave s prezasedenostjo trdega diska.

1.4.4 Lokalnost - Eksperiment 4

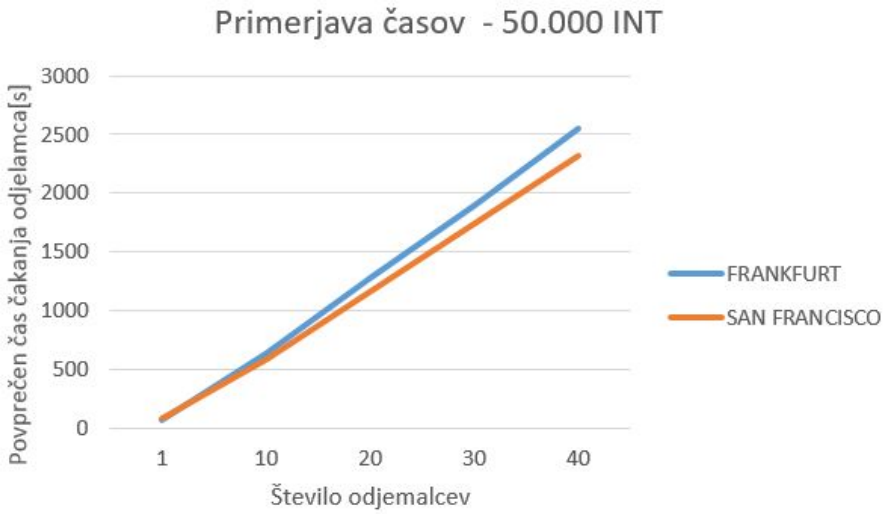
- **Hipoteza:**
  Zaradi možnosti izbire lokacije strežnika smo želeli preveriti vpliv lokacije na povprečni čas čakanja odjemalcev. Poleg strežnika v Frankfurtu smo vzpostavili tudi strežnik v San Franciscu. Predvidevamo, da bomo hitrejše odzive dobivali od strežnika v Frankfurtu, saj nam je fizično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

- **Okoliščine meritve:**
  Strežnika imata dodeljeno enako količino resursov in nameščeno identično programsko opremo. Ta je enaka kot v testu 2 (poglavje 1.4.2) Slednje je bilo potrebno zagotoviti, da izključimo vpliv programske opreme na hitrost odziva 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 1.6: Graf povprečnega časa čakanja odjemalcev v odvisnosti od lokacije na primeru datoteke z 20.000 števili.



Slika 1.7: Graf povprečnega časa cakanja odjemalcev v odvisnosti od lokacije na primeru datoteke s 50.000 števili.

- **Komentar meritve:**
  Rezultati meritev iz slik 1.6 in 1.7 niso potrdili naše hipoteze, saj se 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 infrastrukture v San Franciscu. Drug možen razlog bi lahko bili zmogljivejši procesorji v strežnikih lociranih v San Franciscu, vendar interne informacije o zgradbi strežniških centrov žal niso javne.

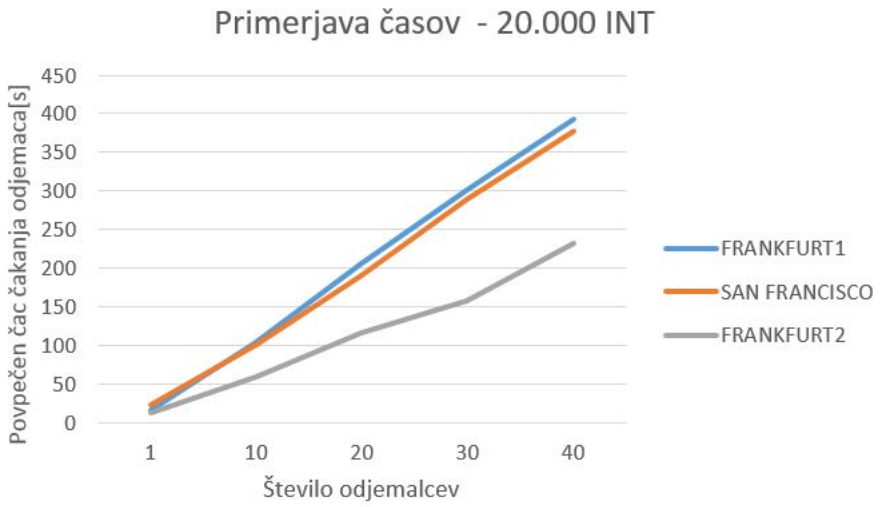
1.4.5 Različne računske moči - Eksperiment 5

- **Hipoteza:**
  Naša hipoteza, ki jo bomo preverili v tem poizkusu pravi, da se povprečni čas čakanja odjemalcev bistveno zmanjša, če je strežnik bolj zmogljiv (več procesorjev in RAM-a).

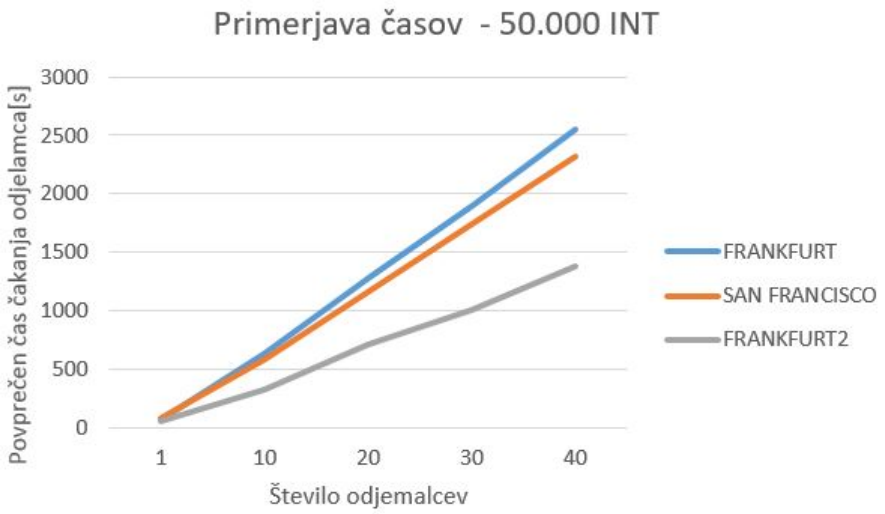
- **Okoliščine meritve:**
  Testiranje smo izvedli v torek 23.5.2017 med 00:30 in 02:30 na strežniku v Frankfurtu. Na strežniku je tekel operacijski sistem Linux Ubuntu 16.04. Na voljo smo imeli 2 GB RAM-a ter 2 procesorja. Na strežniku je bil izbran algoritem za mehurčno urejanje (ang. *bubble sort*).
  Skripto za simulacijo odjemalcev smo pognali iz študentskega naselja Rožna dolina (Ljubljana). Za dostop do interneta je bila uporabljena internetna povezana s hitrostjo 100/100Mbps.
  Na strežnik so klienti (niti) pošiljali datoteke velikosti 20.000 in 50.000 števil tipa integer in merili čase pri 1, 10, 20, 30 in 40 odjemalcih.

- **Rezultati meritve:**

  V grafih na slikah 1.8 in 1.10 so vizualizani povprečni časi odziva treh različnih streznikov. Strežnik Frankfurt1 in San Francisco imata identično konfiguracijo, kot je opisana v eksperimentu 4 (razdelek 1.4.4). Strežnik Frankfurt2 pa ima na voljo 2 procesorja ter 2 GB RAM-a.



Slika 1.8: Graf povprečnega časa čakanja odjemalcev treh različnih strežnikov na primeru datotek z 20.000 števili.



Slika 1.9: Graf povprečnega časa čakanja odjemalcev treh različnih strežnikov na primeru datotek s 50.000 števili.

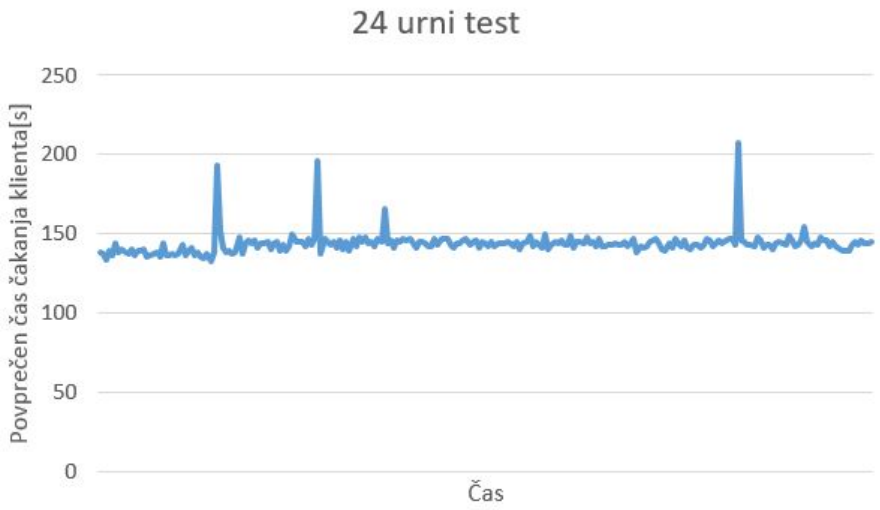
- **Komentar meritve:**
  Meritve potrdijo našo predpostavko, da višja zmogljivost strežnikov pripomore k bistveno nižjim povprečnim časom potrebnim za odziv.

1.4.6 24 urni test - Eksperiment 6

- **Hipoteza:**
  Zaradi ugotavljanja zasedenosti omrežja ter strežnikov smo izvajali še 24 ur trajajoči test. Naša predpostavka je, da bodo časi odzivov višji v dnevnih časovnih razmerah, tj. čez dan kot pa ponoči.

- **Okoliščine meritve:**
  Testiranje smo izvedli v petek 26. in v soboto 27.5.2017. Test smo začeli izvajatvi v petek ob 13:30 in smo ga izvajali 24 ur na strežniku v Frankfurtu. Na strežniku je tekel operacijski sistem Linux Ubuntu 16.04. Na voljo smo imeli 2GB RAM-a, 2 procesorja ter 40 GB prostora na SSD trdem disku. Na strežniku je bil izbran algoritem za mehurčno urejanje (ang. *bubble sort*).
  Skripto za simulacijo odjemalcev smo pognali v Trzinu. Za dostop je bila uporabljena internetna ponudba Telekom Slovenije s hitrostjo 20/10Mbps. Na strežnik so klienti (niti) pošiljali datoteke velikosti 20.000 celih števil za sortiranje v intervalih po 300 sekund. Slednje je počelo 20 odjemalcev hkrati. Merili smo povprečen čas čakanja odjemalcev in kako se ta spreminja skozi celoten dan.

- **Rezultati meritve:**



Slika 1.10: Graf povprečnega časa čakanja odjemalcev v 24 urah.

- **Komentar meritve:**
  Z meritvijo smo ugotovili, da je povprečen čas čakanja odjemalcev preko

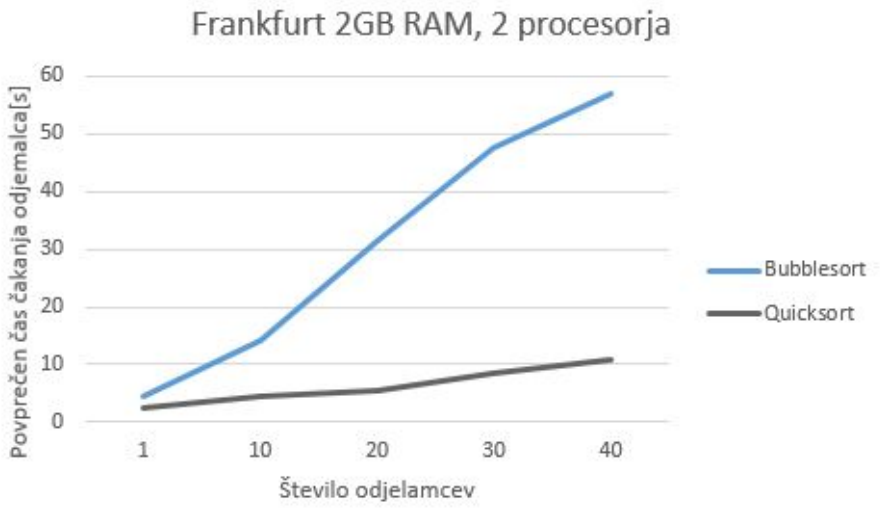
celega dneva praktično enak. To si razlagamo tako, da strežniki nikoli niso tako obremenjeni, da bi to vplivalo na odzivne čase. Čas prenašanja datotek pa zaradi razmeroma hitrih internetnih povezav predstavlja manjši delež skupnega časa. Iz povprečja bolj očitno odstopajo le štiri meritve, ki so trajale občutno dalj časa. Te meritve pa nam ne povedo veliko uporabnega, saj so razporejene dokaj naključno čez celoten dan. Naša hipoteza, ki predpostavlja, da bodo časi obdelave sredi dneva občutno višji od tistih ponoči, je bila napačna.

1.4.7 Izbira algoritma - Eksperiment 7

- **Hipoteza:**
  Na strani strežnika smo implementirali dva različna algoritma za urejanje števil opisana v razdelku 1.2.4. V tem eksperimentu bomo iste naključne datoteke pošiljali na strežnik ter merili čase do prejema datotek pri obeh algoritmih. Naša hipoteza je, da bo strežnik na katerem teče algoritem *quicksort* (Listing 1.1) z nalogo opravil veliko hitreje, kot strežnik z izbranim algoritmom *bubblesort* (Listing 1.2).

- **Okoliščine meritve:**
  Testiranje smo izvedli v sredo 31.5.2017 med 15:00 in 16:00 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. Skripto za simulacijo odjemalcev smo pognali iz študentskega naselja Rožna dolina (Ljubljana). Za dostop do interneta je bila uporabljena internetna povezana s hitrostjo 100/100Mbps.
  Na strežnik so klienti (niti) pošiljali datoteke velikosti 20.000 števil tipa integer in merili čase pri 1, 10, 20, 30 in 40 odjemalcih.

- **Rezultati meritve:**
  Na sliki 1.11 lahko vidimo gafično predstavljene rezultate meritve.



Slika 1.11: Graf povprečnega časa čakanja odjemalcev pri različnih algoritmih.

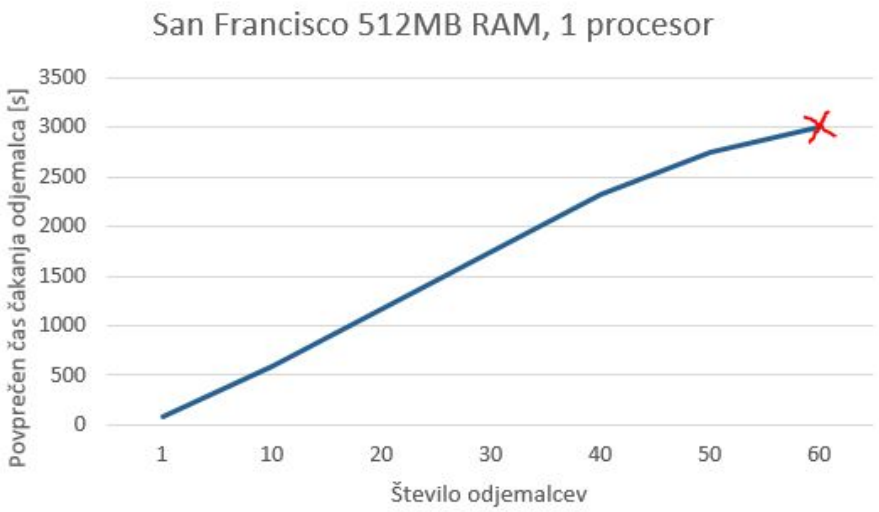
- **Komentar meritve:**
  Rezultati vidni na sliki 1.11 potrjujejo našo hipotezo. Čas sortiranja na strežniku na naključnih datotekah je precej manjši ob uporabi algoritma *quicksort*.

1.4.8 Zlom sistema - Eksperiment 8

- **Hipoteza:**
  Za konec smo želeli tudi zlomit sistem oz. ga prisiliti v to, da nam ne vrača več pravilnih rezultatov. Naša hipoteza je, da bomo prej ali slej prišli do točke, ko bo strežnik zaradi predolgega časa trajanja, potrebnega za odgovor, povezavo zaprl.

- **Okoliščine meritve:**
  Testiranje smo izvedli v četrtek 1.6.2017 med 19:00 in 20:30 na strežniku v San Franciscu. Na strežniku je tekel operacijski sistem Linux Ubuntu 16.04. Na voljo smo imeli 512MB RAM-a ter 1 procesor. Na strežniku je bil izbran algoritem za mehurčno urejanje (ang. *bubble sort*).
  Skripto za simulacijo odjemalcev smo pognali iz študentskega naselja Rožna dolina (Ljubljana). Za dostop do interneta je bila uporabljena internetna povezana s hitrostjo 100/100Mbps.
  Na strežnik so klienti (niti) pošiljali datoteke velikosti 50.000 števil tipa integer in merili čase pri 1, 10, 20, 30, 40, 50, 60 in 80 odjemalcih.

- **Rezultati meritve:**



Slika 1.12: Graf povprečnega časa čakanja odjemalcev pri uspešnem poizkusu zloma sistema.

- **Komentar meritve:**
  Z eksperimentom, katerega rezultati so prikazani na sliki 1.12, nam je uspelo uspešno doseči zlom sistema, saj je strežnik po približno 3000 s čakanja na odgovor povezavo prekinil. Podrobnejše sporočilo o napaki je na vojo v listingu 1.3. Ker nismo vedeli na kakšen način in kdaj bo prišlo do zloma strežnika, je bilo potrebno veliko poizkusov, da smo dosegli željen rezultat.

Listing 1.3: Napaka ob zlomu sistema.

```
Traceback (most recent call last): File
    "C:\Anaconda3\lib\site-packages\requests\adapters.py", line
    324, in send timeout=timeout
  File "C:\Anaconda3\lib\site-packages\requests\packages\urllib3\
      connectionpool.py", line 528, in urlopen raise
      MaxRetryError(self, url, e)
requests.packages.urllib3.exceptions.MaxRetryError:
    HTTPConnectionPool(host='139.59.132.145', port=8080): Max
    retries exceeded with url: /nalozi (Caused by <class
    'TimeoutError'>: [WinError 10060] A connection attempt failed
    because the connected party did not properly respond after a
    period of time, or established connection failed because
    connected host has failed to respond)
```

1.5 Zaključek

Naš cilj je bil preveriti zmogljivost oblačnih storitev za izvajanje računsko zahtevnih operacij. Uporabili smo strežniško infrastrukturo podjetja DigitalOcean in na njej izvajali operacijo urejanja števil, saj ta vsebuje veliko operacij. Izvajali smo različne teste, ter tako preverjali vpliv določenih spremenljivk na zmogljivost našega sistema.
S testi smo hoteli ugotoviti vplive števila odjemalcev, velikosti datotek, lokacije strežnika, količine razpoložljivih resursov na strežniku, ure v dnevu ter izbire algoritma na hitrost odziva strežnika. Poleg tega smo izvedli tudi test s katerim smo povzročili zlom sistema. Merili smo predvsem čase potrebne za pošiljanje in sortiranje datotek ter zasedenost resursov na strežniku.
Kot glavno ozko grlo strežnika se je pričakovano izkazal CPU, saj mora pri sortiranju opraviti zelo veliko operacij primerjanja. Maksimalno zasedenost CPU-ja laho hitro dosežemo z večanjem števila odjemalcev in velikosti datotek.
Za metrike smo se odločili na podlagi članka [5]. V veliko pomoč pa so nam bile tudi ugotovitve iz člankov [6], [7] in [8]. Nekatere podobne meritve smo našli v članku [9], kjer so ravno tako kot mi merili zmogljivost programov v oblaku. Na podlagi eksperimentov smo ugotovili, da je izbira algoritma zelo pomembna in bistveno vpliva na čas obdelave. Prav tako je ključna obremenjenost strežnika s številom klientov. Kljub vsemu, pa strežnik s tako nizkimi zmogljivostmi ne pride v poštev za praktičo uporabo reševanja računskih problemov namesto končnih delovnih točk (osebnih računalnikov). Ob tem pa je potrebno poudariti, da na trgu obstajajo tudi precej zmogljivejše rešitve, ki omogočajo bistveno večjo računsko moč. Vedno hitrejši strežniki in povečevanje povprečne hitrosti internetne povezave, pa bo slej ko prej pripeljalo do točke, ko bomo iz končnih točk računsko zahtevne probleme v obdelavo pošiljali na strežnik. Naše naprave (končne točke), pa bodo skrbele le za interakcijo s strežnikom in prikaz rezultatov.

Literatura

[1] "Javascript - homepage." https://www.javascript.com/.

[2] "Node.js - documentation." https://nodejs.org/en/docs/.

[3] "C - programming language." https://en.wikipedia.org/wiki/C_(programming_language).

[4] "Python - documentation." https://www.python.org/doc/.

[5] "Enhanced network timing calculations for web response time metrics." https://www.ibm.com/support/knowledgecenter/SS5MD2_7.4.0.1/com.ibm.itcamt.doc/rt/Install_Guide/wrt_enhanced_network_timing.html.

[6] "Performance challenges in cloud computing." https://www.cmg.org/wp-content/uploads/2014/03/1-Paliwal-Performance-Challenges-in-Cloud-Computing.pdf.

[7] "Performance analysis of cloud computing centers." http://www.scs.ryerson.ca/~jmisic/papers/qshine2010hamzeh.pdf.

[8] "Performance evaluation methodology for cloud computing using data envelopment." https://www.thinkmind.org/download.php?articleid=icn_2015_3_20_30083.

[9] "Performance testing of cloud applications." http://www.remics.eu/system/files/REMICS_D6.6.lowres.pdf.