Analiza zmogljivosti oblačnih in strežniških storitev

Uredil prof. dr. Miha Mraz

Marec 2016

Kazalo

Poglavje 1

Analiza zmogljivosti oblačne storitve Cloud9 IDE

Ivan Antešić

1.1 Opis Problema

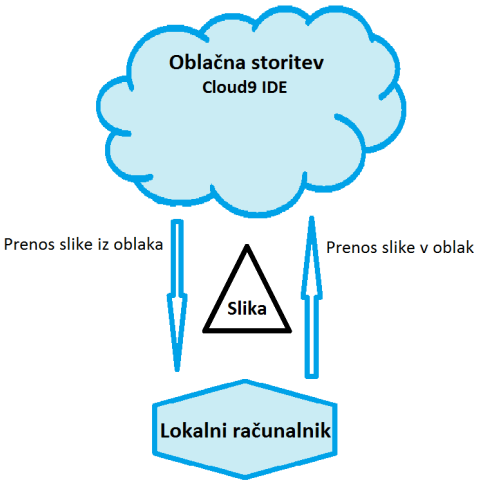
Želel sem preveriti zmogljivost oblačne storitve Cloud9 IDE. Pri testiranju sem se osredotočil na prenašanje datotek med strežnikom na oblaku in lokalnim računalnikom.

Za potrebe testiranja zmogljivosti oblaka sem si zamislil naslednji scenarij. Uporabnik želi imeti shranjene različne datoteke na oblaku, npr. slike, programe itd. Za ta namen lahko uporabi različne že obstoječe storitve (Dropbox, Google Drive, ...). Hitro se zgodi, da porabi prostor, ki mu je namenjen v brezplačni različici. Če ne želi plačati in ima potrebno osnovno znanje spletnega programiranja, si lahko sam postavi takšno aplikacijo na enem od ponudnikov brezplačnih oblačnih storitev.

Zato sem naredil takšno testno aplikacijo in z njo analiziral zmogljivost Cloud9. Aplikacija sprejema poljubne datoteke in jih shranjuje na oblak. Po želji lahko datoteko izbrišemo ali pa jo prenesemo nazaj na računalnik. Na sliki 1.1 je prikazan diagram sistema.

1.2 Oblačna storitev

Cloud9 [1] je spletno integrirano razvijalno okolje, ki gostuje na Google Compute Engine - Googlovi oblačni različici IaaS (*infrastructure as a service* - infrastruk-



Slika 1.1: Diagram sistema.

tura kot storitev). Cloud9 je odprtokodni projekt in podpira večino programskih jezikov z prednastavljenimi delovnimi okolji. Omogoča poganjanje aplikacije na oblaku brez posebnih priprav in drugih programov, zato sem za analizo zmogljivosti izbral to okolje. V zastonjski različici nudi 1 CPU, 1GB RAMa in 5GB HDD, kar zadostuje potrebam moje aplikacije. Z **tracert** ukazom v Microsoftovi ukazni lupini sem izsledil lokacijo strežnika na kateri teče aplikacija. Nahaja se v San Franciscu, v ZDA.

1.3 Tehnologije

1.3.1 HTML

HTML [2] ali *Hyper Text Markup Language* je standardni označevalni jezik, ki se uporablja pri izdelavi spletnih strani. HTML semantično opisuje strukturo strani in omogoča vključevanje drugih datotek, potrebnih za druge vidike spletnih strani.

1.3.2 CSS

CSS [3] ali *Cascading Style Sheets* je standardni jezik za opisovanje izgleda spletnih strani. V HTML vključena CSS datoteka določa barve in obliko elementov spletne strani. Z ločevanjem strukture in izgleda strani omogočimo večjo preglednost kode.

1.3.3 JavaScript

Javascript [4] je dinamičen, objektni programski jezik. Sintaksa jezika ohlapno sledi programskemu jeziku C. Veliko se uporablja za izdelavo spletnih strani skupaj z HTML in CSS, kjer skrbi za interaktivnost in dinamiko strani.

1.3.4 Node.js

Node.js [5] je odprtokodno okolje (jezik) namenjeno razvijanju strežniških spletnih aplikacij. Zgrajen je na Googlovem V8 Javascript engine-u. Uporablja neblokirajoč vzhodno/izhodni model (ki je hiter in raztegljiv) in uporablja sistem odprtokodnih knjižnic npm.

1.3.5 Firefox Network Monitor

Firefox Network Monitor [6] je brezplačno, odprtokodno orodje vgrajeno v novejše različice brskalnika Firefox namenjeno nadzorovanju delovanja spletne strani in njenih komponent (CSS, HTML, Javascript). Z orodjem lahko vidimo katere zahteve se pošiljajo na stran, kaj vsebujejo in koliko časa traja obdelovanje teh zahtev. Čas obdelave zahtev se nato deli na več podčasov, ki jih lahko spremljamo.

1.4 Implementacija aplikacije

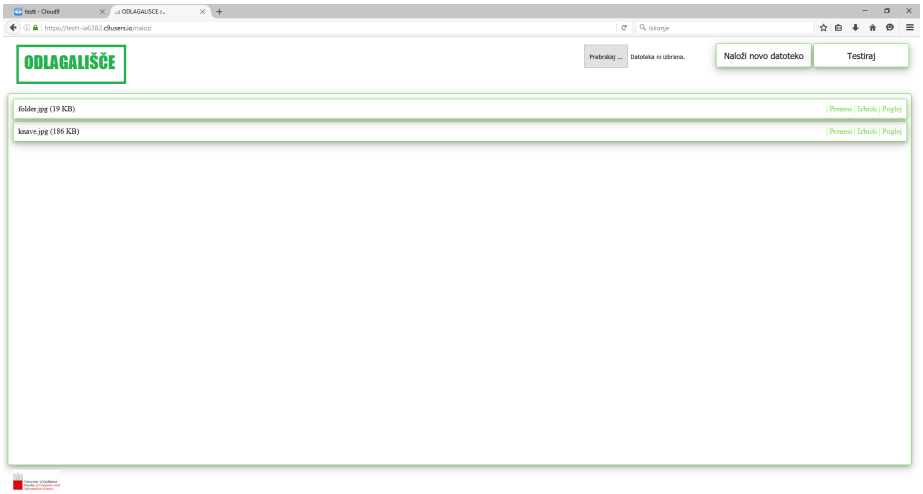
Aplikacija je sestavljena sestavljena iz 2 glavnih delov in sicer strežnika in odjemalca.

1.4.1 Strežnik

Strežnik je napisan v Node.js. Kreira http strežnik in omogoča posredovanje različnih statičnih vsebin ter storitev odjemalcu. Z odjemalcem komunicira preko XMLHttp zahtev in odgovorov. Uporabnik tako lahko naloži različne datoteke na strežnik, jih zbriše, pogleda v novem zavihku, ali prenese na svoj lokalni računalnik. Strežnik shranjuje datoteke v prostor na oblaku, ki mu ga dodeli Cloud9.

1.4.2 Odjemalec

Odjemalec je sestavljen iz treh glavnih datotek: iz HTML, CSS in JavaScript datoteke. HTML opiše spletno stran in njene gradnike, CSS jim spremeni izgled, JavaScript datoteka pa omogoča funkcionalnost in komunicira s strežnikom. Od njega zahteva seznam obstoječih datotek na strežniku, njihovo velikost in ime ter implementira funkcionalnost gumbov in drugih elementov. Na sliki 1.2 je prikazana slika izgleda aplikacije.



Slika 1.2: Izgled aplikacije.

1.4.3 Avtomatizacija

Namesto ločene lokalne skripte sem za potrebe testiranja, dodal funkcionalnost avtomatizacije kar direktno v aplikacijo. Na strani odjemalca sem dodal v html datoteki dodaten gumb **Testiraj**. Ob pritisku gumba se na strežnik pošlje zahteva po testiranju, ki sproži avtomatsko nalaganje izbrane slike. Slike se naložijo večkrat v ciklu kjer se izvede nalaganje, brisanje, in čakanje določeno število milisekund pred naslednjo ponovitvijo. V samem programu na strežniku lahko poljubno določimo število ponovitev nalaganj ter čas, ki določa zakasnitev med cikli (in s tem frekvenco nalaganja bremena). Na odjemalcu sem napisal še dodatno funkcijo, ki se pokliče, ko želimo izmeriti čas potovanja zahtevka na strežnik.

V nadaljevanju je prikazan del izvorne kode, ki skrbi za avtomatizacijo (Listing 1.1, 1.2, 1.3 in 1.4).

```
<form action="/nalozi" method="post" enctype="multipart/form-data">
     <input type="file" id="datoteka" style="height: 50px;
         margin-right: 20px;" name="upload">
     <button id="nalozi" class="rob senca" type="submit">Naloži novo
         datoteko</button>
     <button id="test" class="rob senca" type="submit"
         formaction="/testiraj">Testiraj</button>
</form>
```

Listing 1.1: Del Html datoteke, ki pošlje zahtevo /testiraj na strežnik.

```
var streznik = http.createServer(function(zahteva, odgovor) {
```

```
  if (zahteva.url == '/') {
     posredujOsnovnoStran(odgovor);
  } else if (zahteva.url == "/testiraj") {
     console.log("----TESTIRAM---------");
     console.log("----delay: "+postanek+" ms");
     console.log("----stevilo datotek: "+N);
     testirajDatoteko(zahteva, odgovor);
  }
});
```

Listing 1.2: Tukaj strežnik sprejme zahtevo /testiraj in požene funkcijo testirajDatoteko.

```
function testirajDatoteko(zahteva, odgovor){
   zacetniPrenos(zahteva, odgovor, function(){
      brisiDat(odgovor, dataDir+ime[1]+koncnica[1], function(){
         console.log("*");
         for(var i = 0;i < N;i ++){
            cikel(i, zahteva, odgovor);
         }
      });
   });
}

function cikel(i, zahteva, odgovor) {
  setTimeout(function() {
     var t = process.hrtime();
     naloziDat(i, zahteva, odgovor, zacasnaPot, function(indeks){
       t = process.hrtime(t);
       console.log(t[0]+((t[1] / 1000000).toFixed(3))+"; ");
       brisiDat(odgovor, dataDir+ime[1]+indeks+koncnica[1], function(){
       });
   });
  },i*postanek);
}

var naloziDat = function(indeks, zahteva, odgovor, zacasnaPot, callback)
    {
   fs.copy(zacasnaPot, dataDir + ime[1] + indeks + koncnica[1],
       function(napaka) { //kopiraj dat iz zacasnaPot v dataDir+datoteka
      if (napaka) {
         console.log("error nalaganje");
      } else {
         callback(indeks);
      }
   });
}

var brisiDat = function(odgovor, absolutnaPotDoDatoteke, callback){
```

```
    fs.unlink(absolutnaPotDoDatoteke, function(err) {
        if (err) {
            return console.error(err);
        } else {
            callback();
        }
    });
}

var zacetniPrenos = function (zahteva, odgovor, callback){
    var t = process.hrtime();
    var form = new formidable.IncomingForm();

    form.parse(zahteva, function(napaka, polja, datoteke) {
        util.inspect({fields: polja, files: datoteke});
    });

    form.on('end', function(fields, files) {

        var imeReg = /(.+?(?=\..*))/;
        var koncnicaReg = /.*(\..*)/;
        zacasnaPot = form.openedFiles[0].path;
        var datoteka = form.openedFiles[0].name;

        t = process.hrtime(t);
        console.log("cas priprave= "+t[0]+((t[1] / 1000000).toFixed(3)));

        ime = datoteka.match(imeReg);
        koncnica = datoteka.match(koncnicaReg);

        var velikost = form.openedFiles[0].size;
        console.log("----velikost: "+ velikost + " b ");
        console.log("zacetna datoteka: "+ zacasnaPot+ " "+ datoteka);
        fs.copy(zacasnaPot, dataDir + datoteka, function(napaka) {
            if (napaka) {
                posredujNapako500(odgovor);
            } else {
                posredujOsnovnoStran(odgovor);
                callback();
            }
        });
    });
}
```

Listing 1.3: Funkcija testirajDatoteko N-krat požene cikel nalaganj in brisanj ter izpiše rezultate meritev. V funkciji se kličejo še pomožne funkcije za nalaganje in brisanje datoteke.

```
  var posljiZahtevek = function(event) {
```

```
    var start = new Date().getTime();
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
       if (xhttp.readyState == 2 && xhttp.status == 200) {
          var end = new Date().getTime();
          console.log(end-start);
       }
    };
    var t1 = new Date().getTime();
    xhttp.open("GET", "/zahtevek/"+t1, true);
    xhttp.send();
}

document.querySelector("#t1").addEventListener('click',
     posljiZahtevek);
```

Listing 1.4: Del kode na odjemalcu, kjer pošljemo zahtevek na strežnik in merimo čas potovanja zahtevka.

Ta funkionalnost mi omogoča pridobivanje velikega števila meritev za poljubna bremena. Rezultati se izpišejo v obliki primerni za uvoz v Microsoft Excel, kjer lahko hitro prikažem podatke v obliki grafov in iz njih izvlečem zaključke o delovanju aplikacije. Na sliki 1.3 je primer izpisa.

1.5 Breme

Zmogljivost aplikacije sem testiral s prenašanjem datotek na strežnik *(upload)*. Za osnovno testno datoteko sem izbral manjšo sliko veliko 1,48 MB. Manjša velikost slike skrajša merjenje časov prenosa in omogoča pošiljanje večjega števila slik, saj je prostor na Cloud9 v brezplačni različici omejen na 1 GB. Zaporedno sem prenesel večjo količino slik z različnimi frekvencami prenosa in tako opazoval vpliv na zmogljivost aplikacije.

1.6 Metrike

Za lažje razumevanje časov T_1, T_2 in T_3, ki jih merim pri testiranju, je na sliki 1.4 prikazan časovni diagram nalaganja (*upload*) slike na strežnik. T_1 je čas prenašanja zahteve z odjemalca na strežnik.

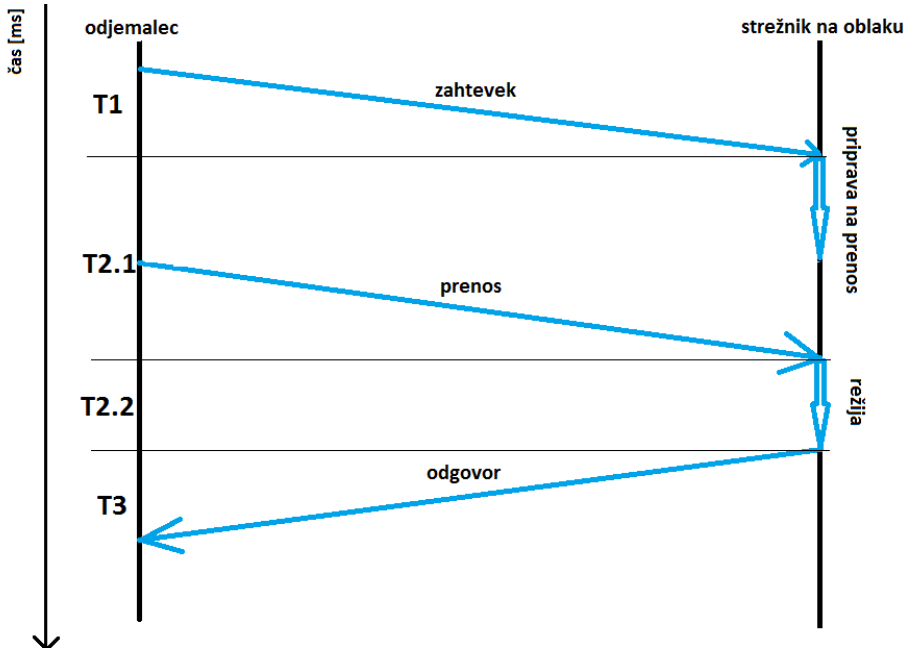
T_2 je čas obdelave zahteve na strežniku. Pri meni se deli na dva dela; $T_{2.1}$ in $T_{2.2}$. $T_{2.1}$ predstavlja pripravo datoteke na prenos (iz polja za izbiro datoteke prebere ime in lokacijo na lokalnem računalniku) in dejanski prenos datoteke iz računalnika na strežnik. $T_{2.2}$ je čas režije prenešene datoteke na strežniku (prikaz na spletni strani, določitev velikosti, po potrebi tudi preimenovanje ali brisanje). T_3 je čas potreben za prenos odgovora s strežnika k odjemalcu. Ta odgovor je lahko potrditev uspešnega prenosa ali koda napake.

T_2 sem lahko pridobil z avtomatskim testiranjem opisanim v prejšnjem poglavju. Ker se obdelava zahteve v celoti dogaja na strežniku, sem lahko na željenih mestih preprosto postavil časovne točke in izmeril potreben čas. T_1 sem tudi pridobil programsko saj v Javascriptu obstaja dogodek (*event*), ki javi odjemalcu, kdaj je njegova zahteva prispela na strežnik.

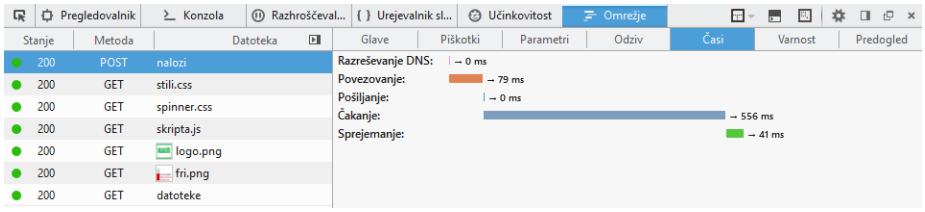
Težave sem imel pri pridobivanju T_3. Pri merjenju časa so bile ure na strežniku in odjemalcu neusklajene in so pokvarile natančne meritve. Zato sem uporabil Firefox Network Monitor, ki prikazuje različne čase pošiljanja in obdelave zahtev. Zame je bil pomemben predvsem čas **Sprejemanje**, ki je definiran kot čas sprejemanja odgovora. Točno to pri meni predstavlja čas T_3. Ker nisem mogel pridobiti T_3 z avtomatskim testiranjem sem „ročno“ za vsako zahtevo pogledal čas v Network Monitorju (kot je prikazano na sliki 1.5 in ocenil potreben čas.

```
----TESTIRAM---------
----delay: 200 ms
----stevilo datotek: 500
cas priprave= 6165.380
----velikost: 1555690 b
zacetna datoteka: /tmp/upload_b5eec14ee854ef414fbb52fbb761d0f1 moon.png
*
datoteka 0. Cas: 09.063ms
datoteka 1. Cas: 010.431ms
datoteka 2. Cas: 06.380ms
datoteka 3. Cas: 09.529ms
datoteka 4. Cas: 030.218ms
datoteka 5. Cas: 015.512ms
datoteka 6. Cas: 09.428ms
datoteka 7. Cas: 06.934ms
datoteka 8. Cas: 07.457ms
datoteka 9. Cas: 06.855ms
datoteka 10. Cas: 06.502ms
datoteka 11. Cas: 06.989ms
datoteka 12. Cas: 06.843ms
datoteka 13. Cas: 07.966ms
datoteka 14. Cas: 09.333ms
datoteka 15. Cas: 08.591ms
datoteka 16. Cas: 012.070ms
datoteka 17. Cas: 010.701ms
datoteka 18. Cas: 011.616ms
datoteka 19. Cas: 013.862ms
datoteka 20. Cas: 08.299ms
datoteka 21. Cas: 09.105ms
datoteka 22. Cas: 06.352ms
datoteka 23. Cas: 07.428ms
datoteka 24. Cas: 08.290ms
datoteka 25. Cas: 07.406ms
datoteka 26. Cas: 08.143ms
datoteka 27. Cas: 08.786ms
datoteka 28. Cas: 012.637ms
datoteka 29. Cas: 07.424ms
datoteka 30. Cas: 07.805ms
datoteka 31. Cas: 011.512ms
datoteka 32. Cas: 011.852ms
datoteka 33. Cas: 020.451ms
datoteka 34. Cas: 05.932ms
datoteka 35. Cas: 07.020ms
datoteka 36. Cas: 09.451ms
datoteka 37. Cas: 07.169ms
datoteka 38. Cas: 06.257ms
datoteka 39. Cas: 07.292ms
```

Slika 1.3: Del izpisa meritev z avtomatskim nalaganjem datotek.



Slika 1.4: Časovni diagram poteka nalaganja datoteke na strežnik.



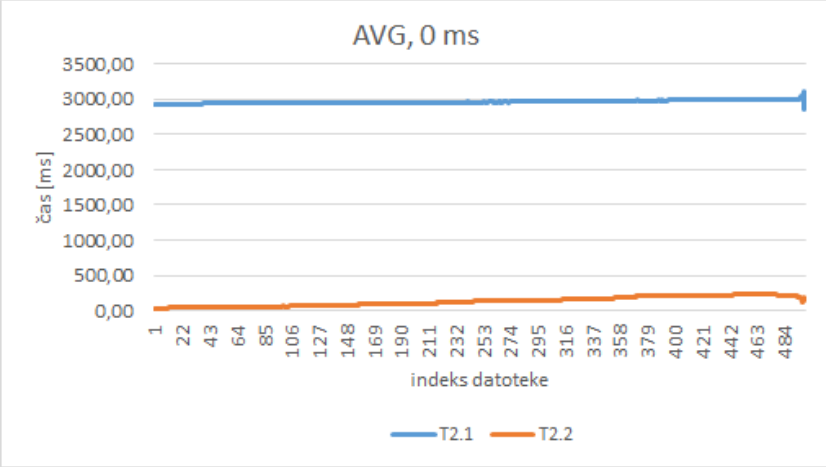
Slika 1.5: Prikaz različnih časov za zahtevo /naloži v Firefox Network Monitor.

1.7 Rezultati meritev

Odločil sem se, da zaporedno pošiljam 500 datotek velikosti 1,48 MB s štirimi različnimi frekvencami (različno velike zakasnitve med zaporednimi prenosi) in analiziram podatke teh meritev. Meritve so bile izvedene v petek 6.5., soboto 7.5. in v ponedeljek 9.5. Ob teh dnevih sem izvedel meritve trikrat na dan ob 11:00, 16:00 in 21:00. S tem sem želel zmanjšati zunanji vpliv ure in dneva na rezultate.

Čas T_2 predstavlja glavni in najzanimivejši del tega testiranja, saj je najbolj odvisen od velikosti datoteke in frekvence pošiljanja datotek na strežnik. Na tem času bom lahko tudi opazil delovanje predpomnilnika na oblaku (če je sploh prisoten).

Spodaj so prikazani grafi za čas T_2 ob različnih frekvencah. Za vsako frekvenco sem naredil maksimalni in minimalni graf, ter graf, ki prikazuje povprečje vseh meritev za to frekvenco. Ko sem izvajal meritve sem opazil, da je čas $T_{2.2}$ zanemarljiv v primerjavi z $T_{2.1}$ (kot lahko vidimo na sliki 1.6), zato sem ga izpustil iz prikaza meritev.

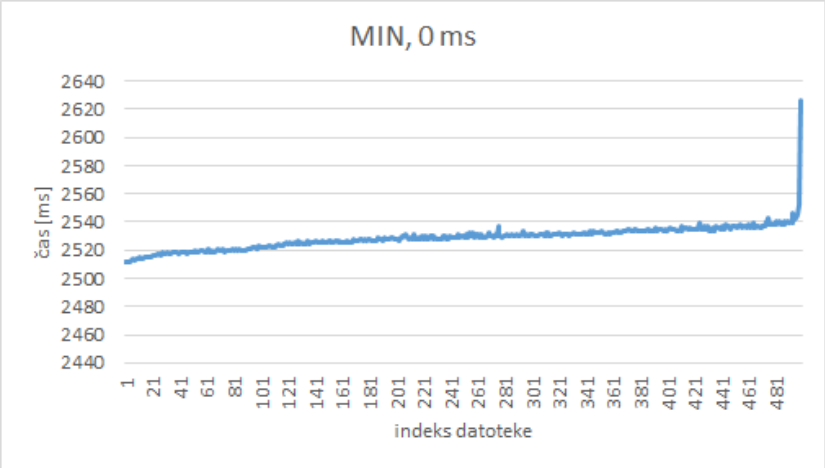


Slika 1.6: Povprečni graf za zakasnitev 0 ms za $T_{2.1}$ in $T_{2.2}$.

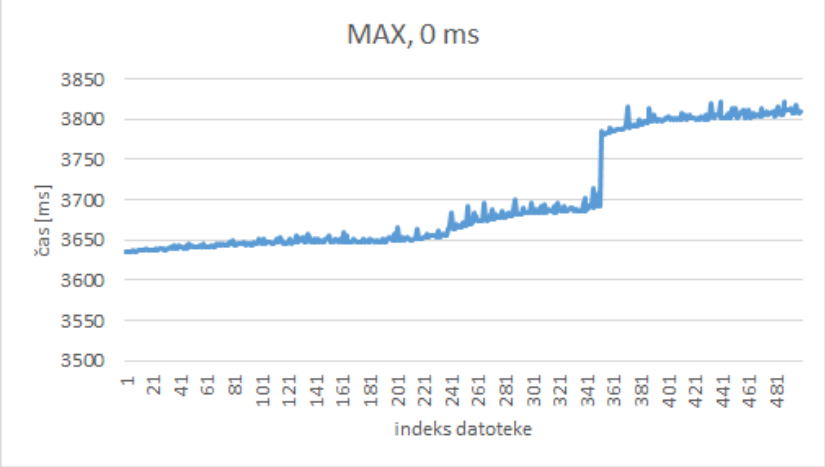
1.7.1 Zakasnitev 0 ms

Najprej sem pognal testiranje brez zakasnitve, torej vseh 500 datotek se bo začelo nalagati praktično naenkrat. Verjetno je rezultat takšen, ker se v zanki datoteke zaporedno prenašajo skoraj naenkrat, vendar še vedno z majhnim zamikom. Prva datoteka se začne prenašati, nato nekaj nanosekund kasneje druga itd. Tako tudi končajo prenos: prejšnja malo prej kot naslednja, kar privede do linearnega naraščanja. Čas prenosa je večji kot, če bi se prenašala vsaka posebej saj si med sabo podaljšujejo prenos.

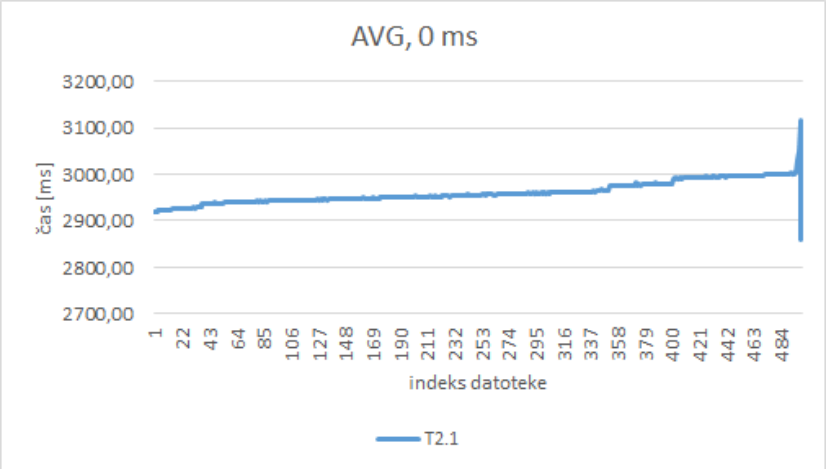
Minimalni graf (slika 1.7) je bil izmerjen v soboto ob 21:00, maksimalni (slika 1.8) v petek ob 16:00. Slika 1.9 prikazuje graf povprečja vseh meritev za to frekvenco.



Slika 1.7: Minimalni graf za zakasnitev 0 ms. Povprečni čas je 2528,92 ms.



Slika 1.8: Maksimalni graf za zakasnitev 0 ms. Povprečni čas je 3701,08ms.

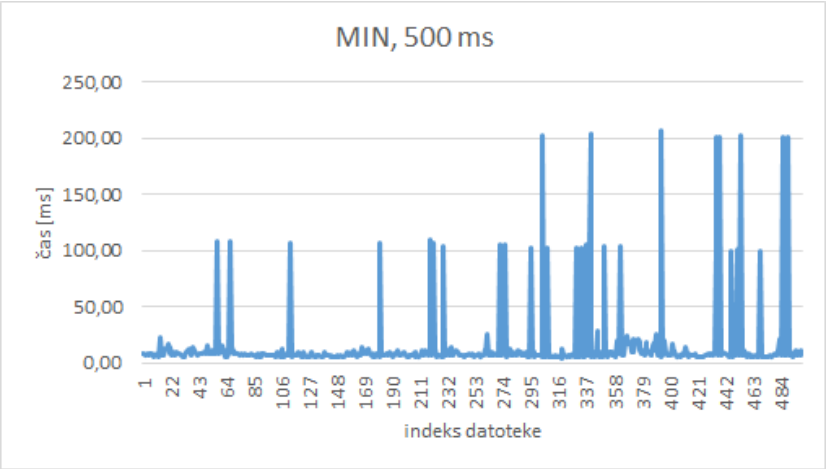


Slika 1.9: Povprečni graf za zakasnitev 0 ms. Povprečni čas je 2960,7 ms.

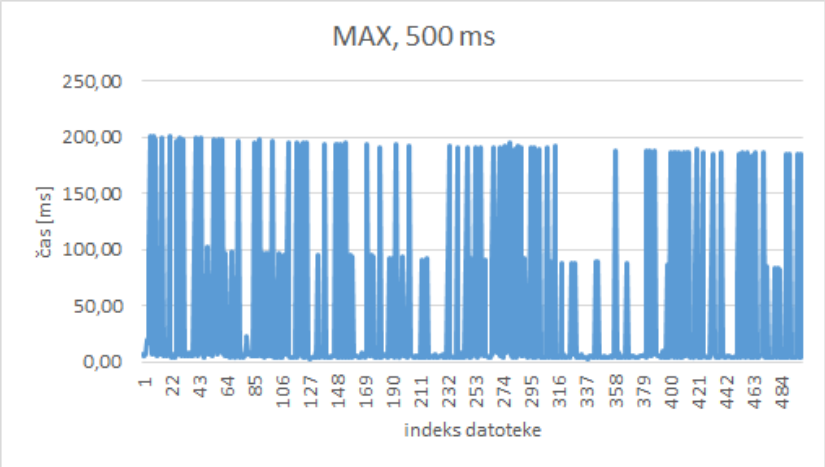
1.7.2 Zakasnitev 500 ms

Nato sem uporabil zakasnitev veliko 500 ms, ki je dovolj velika, da se bo vsaka datoteka prenesla posebej, preden se bo začel prenos naslednje datoteke.

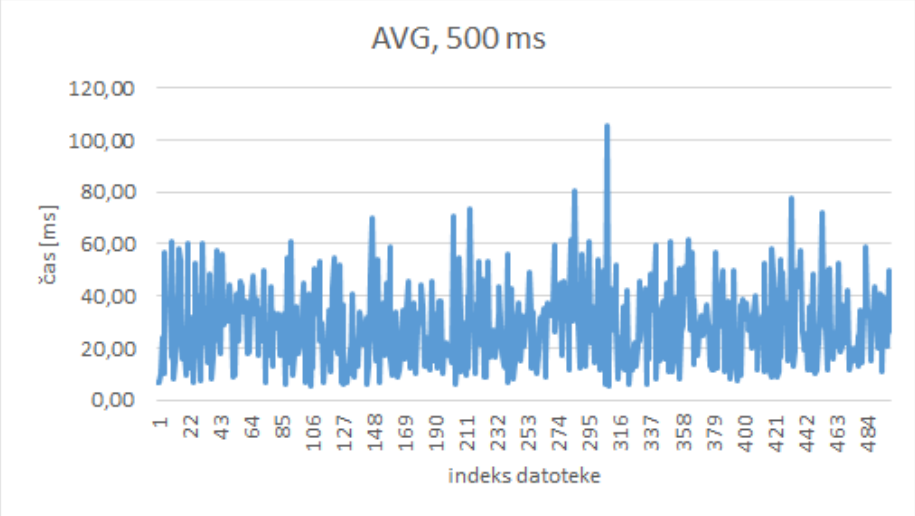
Pričakoval sem enakomerno krivuljo grafa, saj se prenos posameznih datotek ne bo prekrival, in to tudi dobil. Minimalni graf (slika 1.10) je bil izmerjen v petek ob 21:00, maksimalni (slika 1.11) v ponedeljek ob 21:00. Slika 1.12 prikazuje graf povprečja vseh meritev za to frekvenco.



Slika 1.10: Minimalni graf za zakasnitev 500 ms. Povprečni čas je 15,47 ms.



Slika 1.11: Maksimalni graf za zakasnitev 500 ms. Povprečni čas je 41,08 ms.



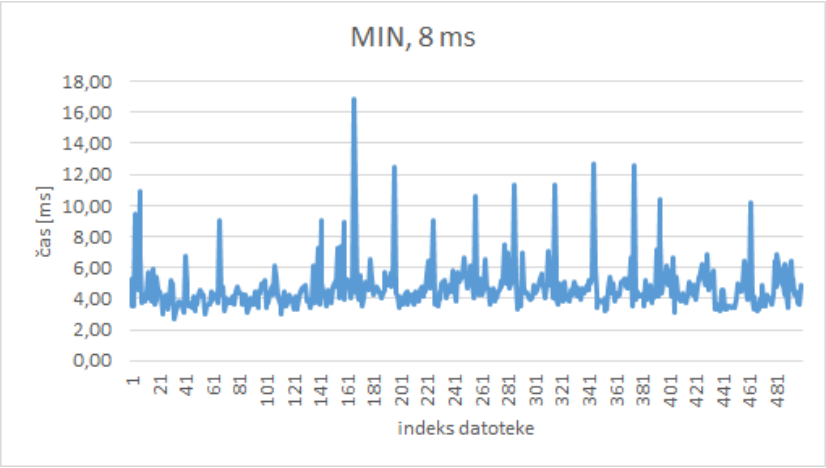
Slika 1.12: Povprečni graf za zakasnitev 500 ms. Povprečni čas je 28,48 ms.

1.7.3 Zakasnitev 8 ms

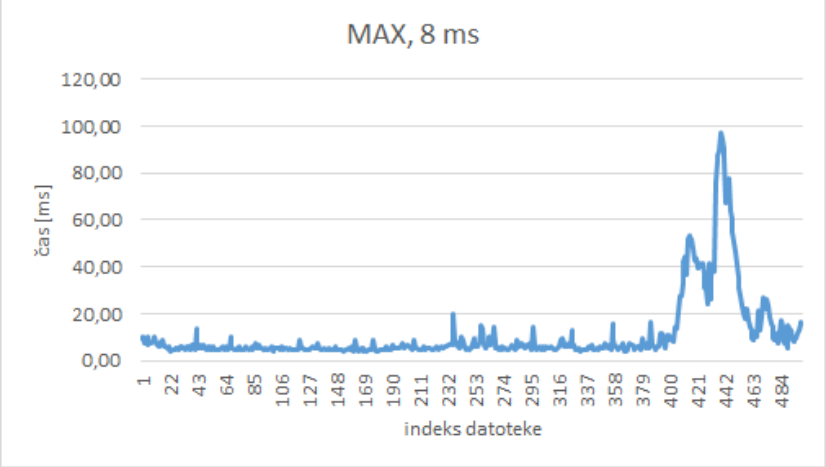
Zakasnitev sem postopoma zmanjševal, da bi našel mejo pri kateri je graf še enakomeren (kot pri zamiku 500 ms) preden se spremeni v naraščajočega (kot brez zamika). Pri tem sem opazil, da se z zmanjšujem zakasnitve rahlo zmanjšuje tudi čas prenosa datoteke (čeprav zakasnitev ni všteta v meritve). Verjetno se to dogaja zaradi predpomnilnika na oblaku. Pri manjšem presledku med prenosi predpomnilnik še pomni, pri večji zakasnitvi pa se ponastavi. Po nadaljnjem

testiranju sem odkril, da se to zgodi pri približni zakasnitvi 200 ms.

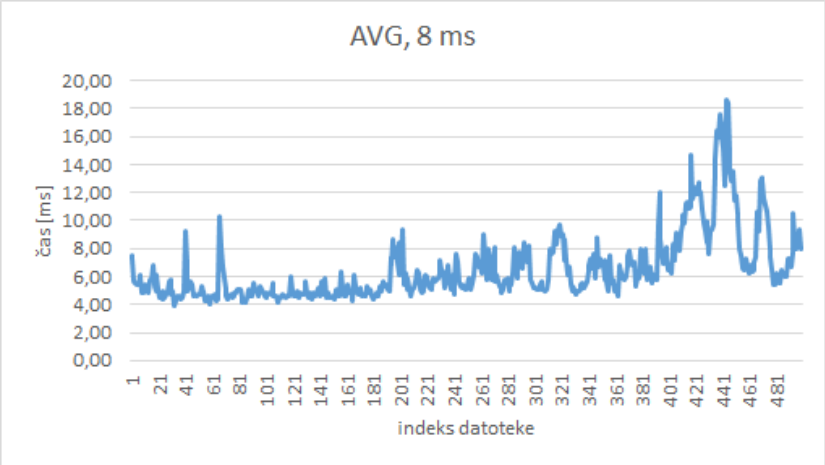
Minimalni graf (slika 1.13) je bil izmerjen v ponedeljek ob 16:00, maksimalni (slika 1.14) v petek ob 16:00. Slika 1.15 prikazuje graf povprečja vseh meritev za to frekvenco. Pri njih lahko vidimo, da se je čas prenosa res zmanjšal v primerjavi z 500 ms.



Slika 1.13: Minimalni graf za zakasnitev 8 ms. Povprečni čas je 4,7 ms.



Slika 1.14: Maksimalni graf za zakasnitev 8 ms. Povprečni čas je 11,22 ms.

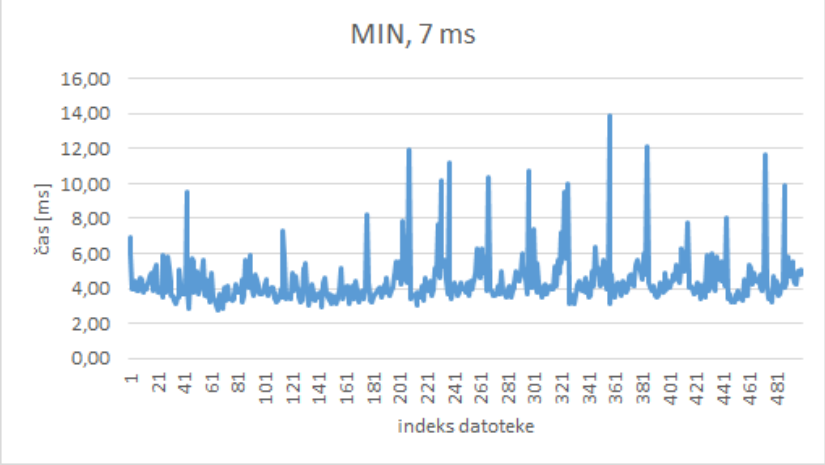


Slika 1.15: Povprečni graf za zakasnitev 8 ms. Povprečni čas je 6,53 ms.

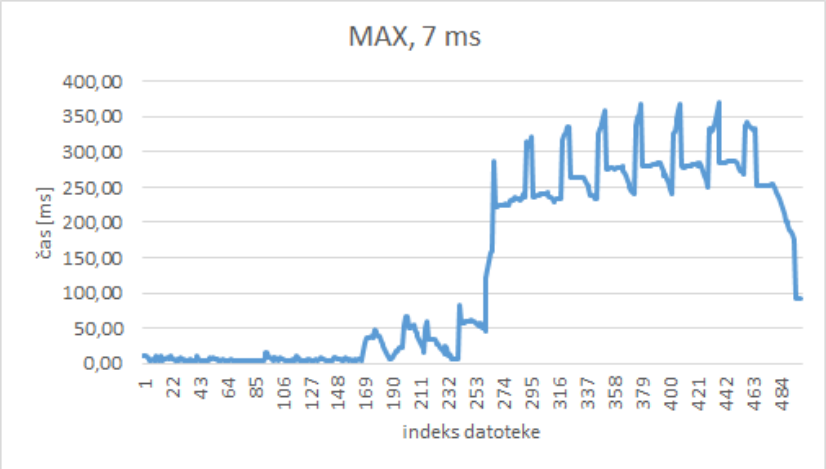
1.7.4 Zakasnitev 7 ms

Seveda pri določeni zakasnitvi pride do naraščanja časa prenosa. Z testiranjem sem to mejo določil pri 8 ms. Pri tej zakasnitvi je graf enakomeren, nato pa naraščujoč. To se dogaja, ker se pri tej zakasnitvi prenos prejšnje datoteke nadaljuje v prenos trenutne, kar povzroči čedalje večje čase.

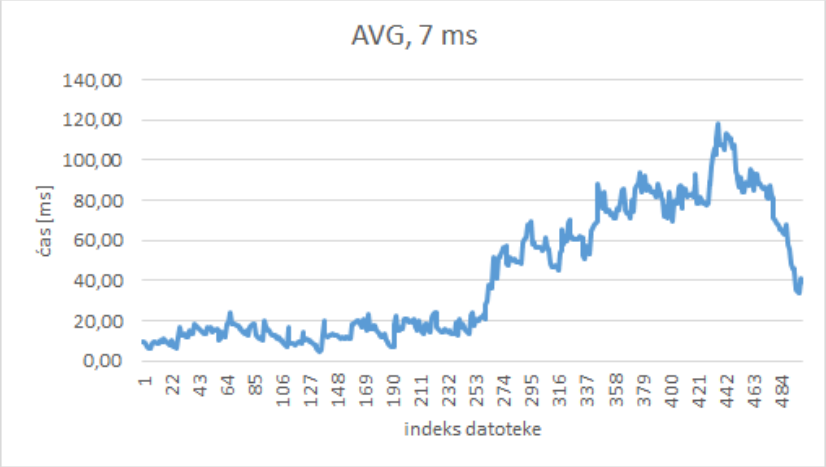
Minimalni graf (slika 1.13) je bil izmerjen v soboto ob 16:00, maksimalni (slika 1.14) v soboto ob 11:00. Slika 1.15 prikazuje graf povprečja vseh meritev za to frekvenco. V soboto ob 16:00 so se datoteke prenašale tako hitro, da je graf postal naraščujoč šele pri 3 ms.



Slika 1.16: Minimalni graf za zakasnitev 7 ms. Povprečni čas je 4,04 ms.



Slika 1.17: Maksimalni graf za zakasnitev 7 ms. Povprečni čas je 135,22 ms.



Slika 1.18: Povprečni graf za zakasnitev 7 ms. Povprečni čas je 42,09 ms.

1.7.5 Razmerje $T_1 : T_2 : T_3$

Za čas T_1 in T_3 nisem naredil grafov saj nisem mogel pridobiti dovolj meritev in ker nista tako pomembna pri tem testiranju (potrebujem ju le za razmerje s časom T_2), zato sem ju samo ocenil. Kljub temu sem meritve za T_1 in T_3 pridobil ob istih dnevih in časih kot za T_2. T_1 sem vsakič pomeril 50-krat, T_3 pa 20-krat.

Minimalni T_1 je znašal 82 ms, maksimalni 171 ms, povprečni pa 103 ms. Za T_3 sem izmeril minimalni čas 12 ms, maksimalni 75 ms in povprečni 39 ms. Za izračun razmerja med T časi sem vzel povprečna časa T_1 in T_3 ter povprečni čas T_2 pri zakasnitvi 500 ms, saj se tako izognem vplivu predpomnilnika in vplivu

frekvence (pri tej zakasnitvi se prenos obnaša kot, da bi pošiljali vsako datoteko posebej, saj ne vplivajo ena na drugo). Čas T_2 je sestavljen iz $T_{2.1}$ (povprečni čas 28,48 ms) in $T_{2.2}$ (povprečni čas 0,55 ms) zato je razmerje $T_1 : T_{2.1}+T_{2.2} : T_3$ približno enako $3.6 : 1 : 1.3$.

1.8 Zaključek

Pri analiziranju oblačne storitve Cloud9 IDE sem se predvsem osredotočil na preverjanje zmogljivosti prenašanja datotek. Skupaj je bilo pri tem testiranju naloženih 36.000 datotek. Po dobljenih rezultatih lahko sklepam, da storitev ima nek predpomnilnik, ki pomni do določenega časa in ima opazen vpliv na zmogljivost.

Iz rezultatov se vidi precej velik vpliv dneva in ure na hitrost prenosa, kar me je presenetilo, saj sem mislil, da v današnjem času, kjer nas je večina konstanto povezanih na internet, dan in čas merjenja ne bo imel takšnega vpliva. Ne morem pa z zagotovostjo reči, da je to posledica zasedenosti omrežja zaradi drugih uporabnikov ali pa samo naključje. Za to bi bilo potrebno narediti veliko več meritev.

Zanimivo je tudi razmerje $T_1 : T_2 : T_3$. Pri tako majhni datoteki, kakršno sem prenašal, se opazi, da je v bistvu čas obdelave zahteve T_2 najmanjši. Pošiljanje zahteve (T_1) traja skoraj štirikrat več, T_3 pa je približno enak T_2. Pri takšni velikosti torej velja: $T_3 > T_2 > T_1$. Verjetno bi bili pri večji datoteki časi drugačni, T_2 bi se povečal, T_1 in T_3 pa bi ostala ista.

Literatura

[1] "Cloud9 IDE." https://c9.io/, Marec 2016.

[2] "HTML." https://en.wikipedia.org/wiki/HTML/, Marec 2016.

[3] "CSS." https://en.wikipedia.org/wiki/Cascading_Style_Sheets, Marec 2016.

[4] "JavaScript." https://www.javascript.com/, Marec 2016.

[5] "Node.js." https://nodejs.org/, Marec 2016.

[6] "Firefox Network Monitor." https://developer.mozilla.org/en-US/docs/Tools/Network_Monitor, Marec 2016.