TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Alvar Sokk 142798IAPB

ERINEVATE VEEBITEENUSTE JÕUDLUSE VÕRDLUS KASUTADES TÖÖKOORMUSE ÜHTLUSTAMIST

Bakalaureusetöö

Juhendaja: Tarmo Veskioja
Tehnikateaduste doktor

Tallinn 2017

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Alvar Sokk

22.05.2017

Annotatsioon

Käesoleva töö põhieesmärgiks on uurida serverite jõudlust kasutades selleks erinevaid modernseid ja populaarseid veebitehnoloogiaid ning võrrelda nende jõudlust piiratud ressurssidega ühel versus mitmel veebiserveril.

Töös teostati erinevate veebirakenduste loomine ning nende testimine kasutades erinevaid tööriistu. Testimise keskkond seati üles kasutades Dockerit ja konteinerite tehnoloogiat. Töökoormuse ühtlustamine sooritati tarkvara tasandil kasutades Nginx ja HAProxy töökoormuse ühtlustajat.

Töö testitulemuste põhjal on analüüsitud vastavate tulemuste põhjuseid ja tehtud järeldused.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 61 leheküljel, 10 peatükki, 23 joonist, 12 tabelit.

Abstract

Performance comparison of different web services using load balancing

The objective of this thesis is to examine server performance using different modern and popular web technologies. To compare single server performance versus multiple using load balancing software such as Nginx and HAProxy in Docker environment. To gather test data and determine the best solutions for load balancing different web services.

As a result of the work multiple web services were created using different programming languages, applications and web servers. Analysis of each web service and load balancing solution was carried out. From the results, it can be seen that with limited resources some applications are effective to load balance and other are not. Applications with demanding processor usage – using frameworks and databases should have more powerful hardware and applications with minimal resource requirements are more beneficial to use load balancing.

As a conclusion, it can be understood what is the best scaling solution for a specific web application. The results could be useful for web developers that are creating a web application that sees a lot of traffic. Also, the method used in this thesis can be used to determine the best scaling option for the web developer himself/herself.

The thesis is in Estonian and contains 61 pages of text, 10 chapters, 23 figures, 12 tables.

Lühendite ja mõistete sõnastik

GET	HTTP päringu variant, kus kõik andmed on päises ja URL-is
POST	HTTP päringu variant, millega saadetakse andmeid serverisse
SSD	*Solid state drive* ehk Pooljuhtketas [1]
Töökoormuse ühtlustaja	Ingl. *load balancer*

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Bakalaureusetöö teemaks on võrrelda erinevaid veebiteenuste töökoormuse ühtlustamise võimalusi mitmel serveril versus ühel mitte töökoormuse ühtlustusega serverit, neid testida erinevate veebiteenustega ja saadud tulemusi võrrelda, et teada saada milline veebiteenus on kasulik jagada mitme serveri vahele ja kasutada töökoormuse ühtlustajat.

1.1 Taust ja probleem

Tänapäeval on interneti kasutajaid [2] ja seadmeid [3] väga palju. Samuti on palju veebilehti ja veebiteenuseid [4], mida inimesed külastavad. Populaarsetel veebilehtedel võib olla väga palju külastajaid [5]. On oht, et kui on väga palju külastajaid, siis veebiserverid ei suuda vastu pidada suurele hulgale päringutele, mis neile saadetakse. Suur liiklus võib olla ettearvamatu [6]. Paljud veebilehtede arendajad ei ole arvestanud järsu suure liikluse kasvuga ning ei ole suutelised enda teenust kõigile kasutajatele pakkuma [7].

1.2 Ülesande püstitus

Töö eesmärk on uurida serverite jõudlust, kasutades selleks erinevaid modernseid ja populaarseid veebitehnoloogiaid ning võrrelda nende jõudlust piiratud ressurssidega ühel versus mitmel veebiserveril. Saadud tulemustest järeldada, millist teenust on kasulikum laiendada vertikaalselt – rohkem servereid või horisontaalselt – võimsam serveri riistvara [8].

1.3 Metoodika

Autor uurib esmalt kuidas ja millega testida ning simuleerida suurt veebiliiklust. Luua näidis veebirakendused, testida veebirakendusi testliiklusega ning luua veebiserverite ette töökoormuse ühtlustaja, mis jagab liikluse veebiserverite vahel laiali. Seejärel antud tulemusi võrrelda ühe serveri ja N serveri vahel, mille vahel on koormus jagatud töökoormuse ühtlustajaga.

1.4 Ülevaade tööst

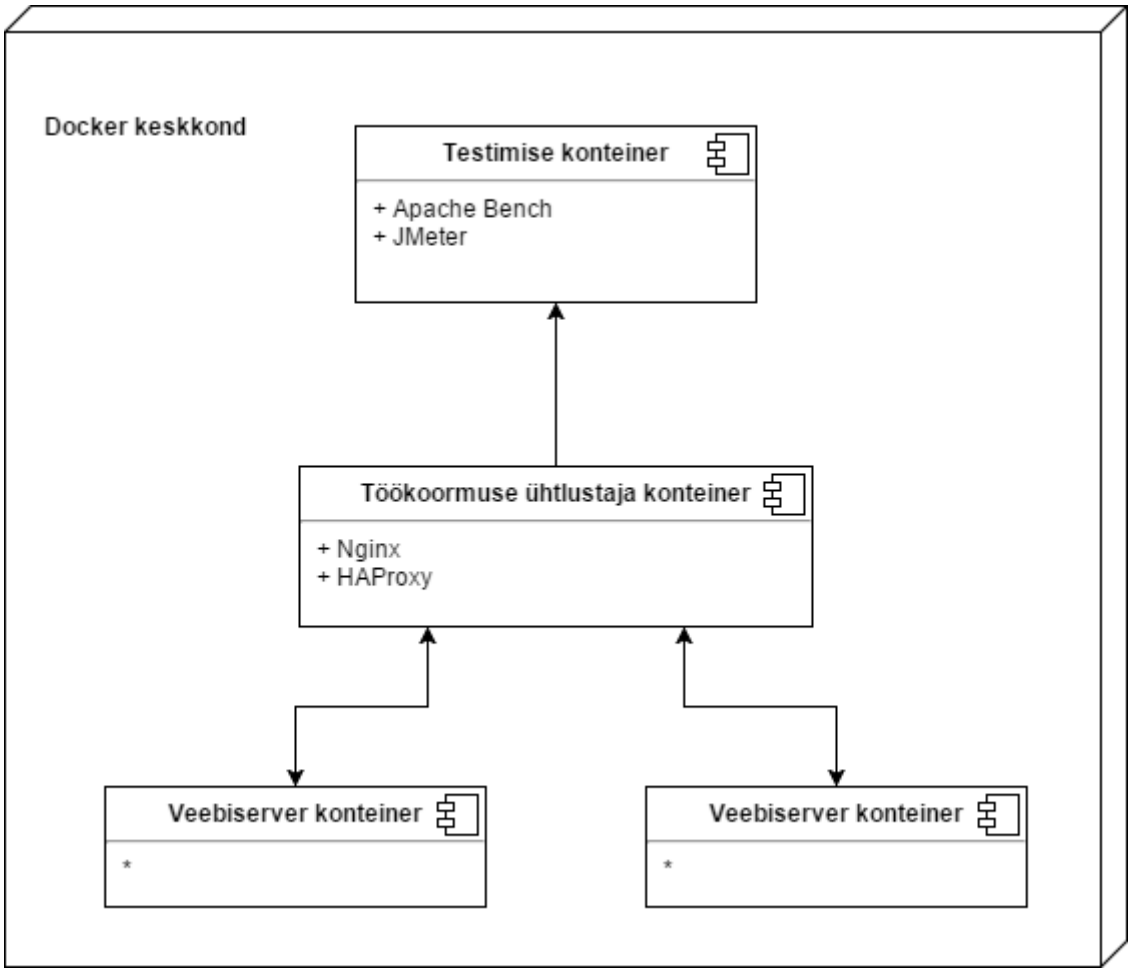
Töö koosneb testimise keskkonnast, veebiteenustest ja veebiteenuste töökoormuse ühtlustajatest, mis on kasutusel ühel arvutil, kasutades konteinerite tehnoloogiat [9].

2 Testimise keskkond

Testimine on üles seatud Linux Ubuntu [10] arvutil, kasutades Dockerit [11]. Docker on avatud lähtekoodiga töövahend, millega saab luua rakendusi ning neid paigaldada ja käivitada, kasutades konteinereid. Konteinerid sisaldavad endas kõike vajalikku, milles rakendust jooksutada, selleks on failisüsteem, kood ja süsteemi tööriistad. See garanteerib, et kõik töötab täpselt samamoodi nagu oli mõeldud ja eemaldab võimaluse, et mõnes keskkonnas töötab ja mõnes mitte. Lisas 1 on toodud Dockerfile näidis ja Lisas 2 on toodud välja docker-compose file, mis sisaldavad vastava rakenduse seadistusi. Peatükis 10.12 on viide testimise keskkonna konfiguratsiooni failidele.

2.1 Jõudlus

Kuna testimise keskkond ja testitavad veebirakendused on sama arvuti peal, jagavad nad ka samu ressursse. Arvutis, kus testimist sooritati, on Intel i7-4700HQ protsessor (4 tuuma, 8 lõime) [12], 1600Mhz kiirusega 16GB muutmälu ja SSD. Vältimaks, et ressursid ei ole võrdselt jagatud või, et mõni ressurss kasutab kogu jõudluse, jättes teistele vähem, kasutati Dockeri võimalusi ressursside võrdseks jagamiseks [13]. Kõigepealt sooritati mitmeid teste, et teada saada, kui palju ressursse erinevad konteinerid erinevate rakendustega kasutavad. Arvestades, et korraga töötavad kolme sorti konteinerid: testimiseks minimaalne Linux operatsioonisüsteem Debian, millel on installeeritud peale operatsioonisüsteemi ainult vajalikud tööriistad (Apache Bench [14] ja JMeter [15]), tööjõudluse ühtlustamise konteiner (Nginx [16] või HAProxy [17]) ja üks kuni mitu veebirakenduse konteinerit (Joonis 1. Docker keskkond).



Joonis 1. Docker keskkond

2.1.1 Muutmälu

Testides ja võrreldes jõudlusega oli selge, et konteinerid kasutavad vähe muutmälu ning muutmälu maht ei saa olema testides ühegi veebiteenuse jõudluse piiravaks faktoriks.

2.1.2 Andmesäilitusseade pooljuhtketas

Dockeri keskkond on ühe pooljuhtketta peal. Vältimaks võimalikku kettale kirjutamise ja lugemise kiiruse suurt hulka, minimeeriti see eemaldades konteineritest logide kirjutamine. Logid on vajalikud ja kasulikud päris süsteemides, aga antud testiskoobist välja jäävad. Testides ja võrreldes jõudlusega oli selge, pooljuhtketta maht ega kiirus ei saa olema testides ühegi veebiteenuse jõudluse piiravaks faktoriks.

2.1.3 Võrguliides

Dockeri keskkonnas on igale konteineritele loodud enda sild (*bridge*) kohtvõrk [18], läbi mille toimub testkonteineri, töökoormuse ühtlustaja ja veebiserverite vaheline liiklus. See minimaliseerib võrguühenduse võimalikku aeglust ning konteinerite vaheline kiirus on maksimaalne antud keskkonnas.

2.1.4 Protsessor

Töö eesmärgiks on eelkõige protsessori jõudluse testimine ja vähem muutmälu, sisend/väljund andmekandjate ja võrgu tööjõu testimine. Kasutati dockeri funktsiooni *cpu_quota* [19], piiramaks konteineri protsessori kasutust. Dockeri muutuja *cpu_quota* annab võimaluse piirata dockeri konteineri ressursse protsendilise täpsusega. Kuna testkeskkonna arvutil on 8 lõime, siis Ubuntu *System monitor* tööriista [20] arvestuse ja *cpu_quota* põhjal, on igal lõimel 100% kasutuse võimalus ning kokku liites on protsessori ressursse 800%. Samuti on töö eesmärgiks testida veebirakenduse jõudlust ühel kuni mitmel veebiserveril ning vähem töökoormuse ühtlustaja ja mitte testimise konteineri jõudlust. Selleks seati veebiserveritele varieeruvad piirangud 2-50% protsessori kasutust. Seetõttu saab arvestada näiteks 5 veebiserveri 5% protsessori kasutust kokku umbes 25% peale. Olles kindel, et töökoormuse ühtlustaja konteineril ei oleks piiravaks faktoriks protsessor jäeti sellel piirang lisamata. Lisaks on testimise konteineril piisavalt suur hulk vabu protsessori resursse, et järgnevates testides ei jõuta kordagi protsessori maksimaalse utiliseerimiseni.

3 Veebiteenused

Veebiteenusteks sai valitud populaarsed veebiserverid ja programmeerimiskeeled. Antud veebiteenused võib jagada kahte gruppi: staatilised ja dünaamilised. Staatilisteks veebiteenusteks valiti staatilise HTML, PHP [21], JSON lehe näitamine. Dünaamilisteks veebiteenusteks valiti PHP koos raamistikuga Laravel [22], kasutades MySQL [23] andmebaasi, Node.js [24] koos MongoDB [25] andmebaasiga ja Python [26] koos raamistikuga Django [27], kasutades PostgreSQL [28] andmebaasi.

HTML ja PHP HTTP serveriks valiti Nginx. Teisena valiti HTTP serveriks Node.js, mis kasutab Javascript programmeerimiskeelt. Python ja Django veebiserveriks valiti Gunicorn [29] HTTP server.

3.1 Nginx veebiserver

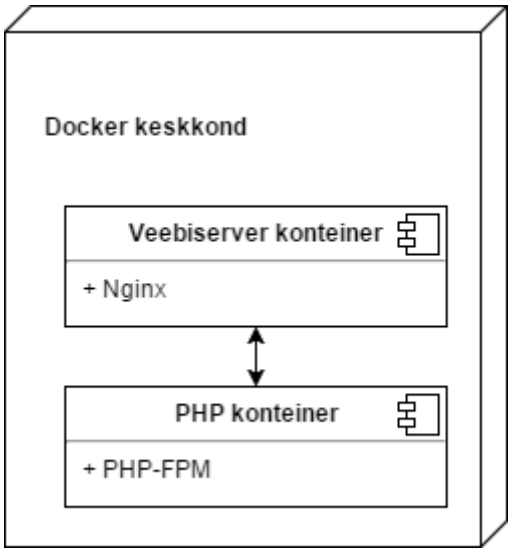
Nginx veebiserver (Joonis 2. Nginx veebiserver) sai üles seatud näitama staatilisel kujul HTML lehte, ilma serveripoolse skripti keeleta. HTML leht on üles seatud näitama staatilist tervitusteksti. Antud veebilehe eesmärk on olla võimalikult kiiresti laaditav, et puuduks protsessori kasutus faili genereerimiseks. Peatükis 10.4 on viide failidele.

Joonis 2. Nginx veebiserver

3.2 Nginx veebiserver koos PHPga

Nginx veebiserver (Joonis 3. Nginx veebiserver koos PHP-ga) koos PHP serveripoolse skriptikeelega, on üles seatud kasutades *FPM (FastCGI Process Manager)* [30]. FPM on kiire ja turvaline viis, kuidas kasutada PHP-d. Selle veebiserveri eesmärk on saada baasjoon, kui võimekas on Nginx veebiserver koos PHP-ga. Veebileht on üles seatud

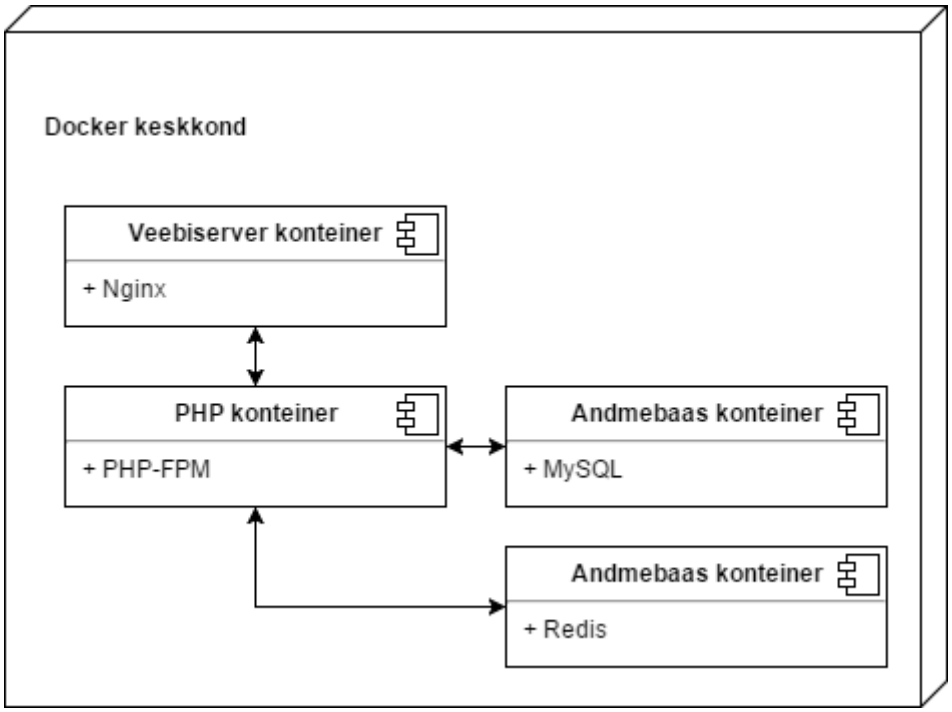
kuvama PHP keeles genereeritud staatilist tervitusteksti. Antud veebiserveris jookseb Nginx server pordil 80 ja PHP FPM pordil 9000, mis suhtleb Nginx serveriga otse enne välisühendust. Peatükis 10.5 on viide failidele.



Joonis 3. Nginx veebiserver koos PHP-ga

3.3 Nginx veebiserver koos Laraveliga

Nginx veebiserver koos väga populaarse PHP serveripoolse skriptikeelega [31] (Joonis 4. Nginx veebiserver koos Laraveliga) ja viimastel aastatel väga populaarne Laravel raamistik [32] [33] [34]. Laravel annab ülevaate kuidas võib tänapäevase suure modernse veebilehe kiirus välja näha. Lisaks PHP Laravel raamistikule, on kasutusel Redis andmebaas [35] vahemälu andmebaasina ja MySQL pikaajalise andmete hoiustamiseks. Antud rakendus on avaliku Docker ja Laravel näidise põhjal [36]. Näidisrakenduse lõi töö autor ise kus saab vormi kirjutada teate ja samal lehel näidatakse 50 viimast teadet. Antud funktsionaalsus on lisaks GET päringuga lehe laadimise kiiruse testimiseks, ka POST päringu kiiruse testimiseks, mis saadab sisestatud andmed andmebaasi ja järgmisel GET päringul vastavaid andmeid kuvab. Antud rakendus vajab Nginx veebirakenduse funktsionaalsust, seetõttu saab seda kasutada ainult Nginx töökoormuse ühtlustajaga. Dockeri konfiguratsioon on Lisas 2. Peatükis 10.6 on viide failidele.



Joonis 4. Nginx veebiserver koos Laraveliga

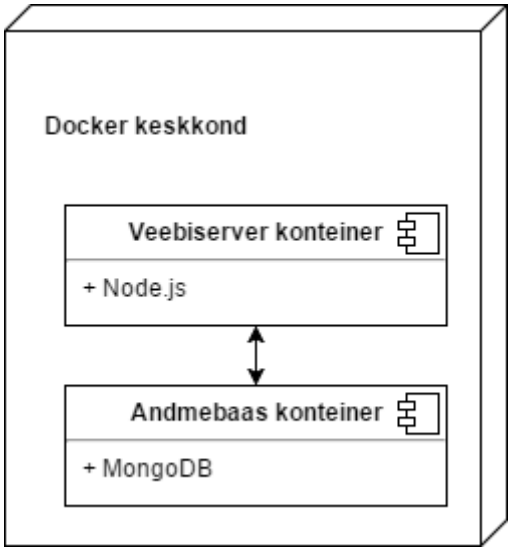
3.4 Node.js veebiserver

Viimastel aastatel on Node.js saanud väga populaarseks [37], millest ka autori valik kasutada Javascript keelel põhinevat Node.js serveritarkvara [24], mis võimaldab kergelt lisada erinevaid võimalusi ja tehnoloogiad kasutades NPM (Node Package Manager) [38] – mis on Node.js lisandpakkide käsitlemise juhtsüsteem. Üks kõige populaarsem minimaalne veebi raamistik Node.js-l on „Express“ [39], mida ka kasutati staatilise veebilehe näitamiseks. Selle testi eesmärk on saada baasjoon, kui kiire ja kui võimekas on Node.js staatilise JSON vastuse näitamisega. Antud veebiserveris (Joonis 5. Node.js veebiserver) jookseb HTTP pordil 8080 ja kuvab ette kirjutatud JSON vastust. Tegemist on välja mõeldud foorumi API päringu vastusega, mis on Lisas 4. Testi tulemusega saab teada, kui kiirelt kuvab minimaalse jõudluse vajadusega Node.js HTTP vastuse. Peatükis 10.7 on viide failidele.

Joonis 5. Node.js veebiserver

3.5 Node.js veebiserver koos MongoDB andmebaasiga

Node.js veebiserver (Joonis 6. Node.js veebiserver MongoDB andmebaasiga), mille külastamisel kõigepealt sisestatakse genereeritud andmed andmebaasi ja küsitakse MongoDB andmebaasist viimased 50 sisestust. Enne igat koormustesti, sisestatakse andmebaasi üle 50 dokumendi, et alati oleks rohkem kui 50 dokumenti MongoDB andmebaasis. Peatükis 10.8 on viide failidele.

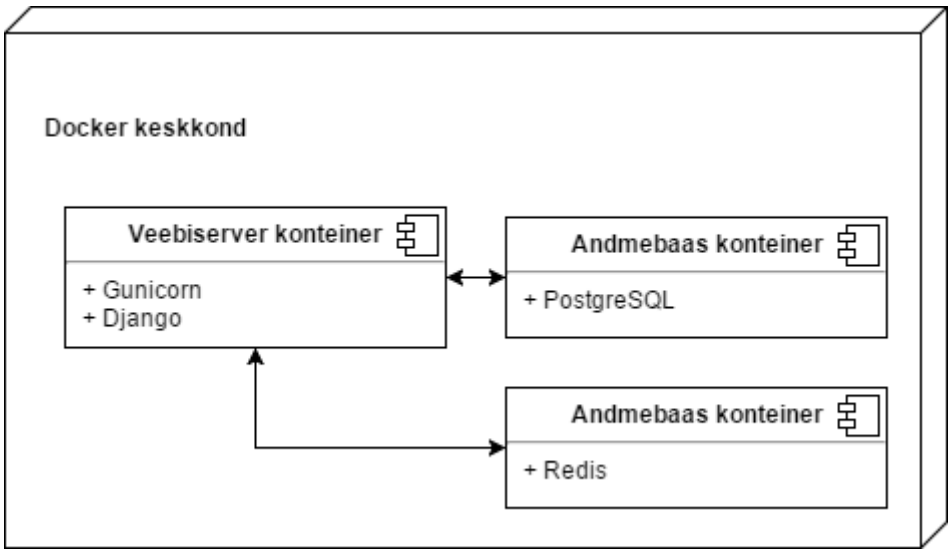


Joonis 6. Node.js veebiserver MongoDB andmebaasiga

3.6 Python Django koos PostgreSQL andmebaasiga

Python keel ja Django (Joonis 7. Python Django koos PostqreSQL andmebaasiga) raamistik on populaarne kooslus veebirakenduste loomiseks [40]. Kuna antud töö eesmärk on testida erinevaid rakendusi, siis valis autor valmis Python ja Django

näidisveebirakenduse [41]. Antud rakendus sisaldab sarnaselt Laravel rakendusele vormi, kuhu on võimalik kirjutada ning samal lehel näidatakse 50 viimast teadet. POST päringuga saadetud vormi andmed salvestatakse PostgreSQL andmebaasi. Samuti on kasutusel Redis andmebaas, mis loeb lehe külastusi [42] ja kuvab neid lehe keskel. Peatükis 10.9 on viide failidele.



Joonis 7. Python Django koos PostqreSQL andmebaasiga

4 Testimise vahendid

4.1 Apache Bench

Apache Bench ehk ab [14] on jõudlustestimise tööriist, millega saab testida HTTP serverit erinevate päringutega. Põhiliselt on selle tööriista eesmärk aidata teada saada, palju veebirakendus päringuid sekundis suudab täita. Tööriist on käivitatav käsurealt. Lisaks testitava serveri aadressile, on võimalik lisaks anda näiteks -n parameeter, mis ütleb mitu päringut serverile saata ning -c parameeter, mis ütleb mitu päringut korraga saata. Näiteks käsk „ab -n 100 -c 10 http://127.0.0.1/“ tähendab 100 GET päringu saatmist 10 korraga, aadressile 127.0.0.1, mis antud juhul on sama server.

4.2 JMeter

JMeter on kasutusel testimaks veebirakendusi koos GET ja POST päringutega, sisestades veebilehel vormi kaudu andmed andmebaasi. JMeter on kasutusel Laravel ja Django veebirakenduses.

5 Töökoormuse ühtlustamise vahendid

Töökoormuse ühtlustamiseks valiti Nginx [16] HTTP server, kasutades selle töökoormuse ühtlustamise võimalusi ja HAProxy [17] töökoormuse ühtlustamise server. Mõlemal töökoormuse ühtlustajal on kasutusel *round-robin* päringute jagamise meetod, mis jagab liikluse alamserverite vahel ringsüsteemis. Töökoormuse ühtlustamise kasutamisel on mitmeid eeliseid. Kasutades mitut veebiserverit töökoormuse ühtlustaja taga, eemaldab rakendusest ühe serveri tõrkepunkti, st. kui üks veebiserver peaks seiskuma, on veel teised selle asemel, mis võimaldavad teenust pakkuda [43].

Samuti võimaldab töökoormuse ühtlustaja veebiteenusele suuremat külastajate arvu eeldades, et üks veebiserver suudab vähem päringutele vastata, kui töökoormuse ühtlustaja jagada. Eelkõige on töökoormuse ühtlustamise eesmärk pakkuda stabiilsemat ja suuremale hulgale külastajatele teenust [44]. Lisaks on erinevaid tasemeid, kus saab töökoormuse ühtlustamist teha [45]. Antud töös on kasutusel seitsmenda taseme töökoormuse ühtlustamine [46]. Eesmärk on leida lisaks erinevate veebiteenuste töökoormuse ühtlustamise efektiivsusele, ka Nginx ja HAProxy töökoormuse ühtlustaja erinevused ja eelised teineteise ees.

5.1 Nginx

Nginx on põhiliselt tuntud ja rohkem kasutusel kui HTTP server, aga võimaldab ka seadeid muutes olla kasutusel töökoormuse ühtlustajana. Kasutusmugavus suurem, tõenäoliselt arendajale tuttav, sest palju kasutusel ka veebiserverina [47]. Nginx tasuta versioonis on piiratud võimalustega monitoorimine [48]. Lisas 5 on näidis Nginx töökoormuse ühtlustaja konfiguratsiooni failist. Peatükis 10.10 on viide failidele.

5.2 HAProxy

HAProxy on alternatiiv Nginx serverile, mis on põhiliselt töökoormuse ühtlustaja erinevalt Nginx serverile, mis võimaldab olla ka HTTP server. HAProxy-s on võimalusterohked monitoorimise võimalused [49]. Lisas 6 on näidis HAProxy töökoormuse ühtlustaja konfiguratsiooni failist. Peatükis 10.11 on viide failidele.

6 Apache Bench jõudlustestid

Apache Bench jõudlustestid toimuvad läbi testserveri, mis asub Dockeris samas *bridge* võrgus koos veebiserverite ja töökoormuse ühtlustajatega. Selleks, et testide kogus oleks vastavuses serverite jõudlusega, on Apache Bench päringute arv (-n parameeter) ja korraga päringute arv (-c parameeter), vastavalt seatud veebiserverite jõudlusega, et ühe testi sooritus võtaks aega umbes 10 kuni 20 sekundit. Näiteks iga veebiserveri kohta töökoormuse ühtlustaja taga 1000 päringut, millest 10 päringut korraga. Sel juhul 5 veebiserveriga näeks testimise käsk välja: ab -n 5000 -c 50 http://172.18.0.2/. Testi tulemusena arvestatakse, mitu päringut sekundis suudab veebiserver täita. Igat testi sooritatakse 10 korda. Väga suurest veebirakenduste päringute täitmise kiiruse erinevusest, on vastavalt töökoormuse jõudlusele, ka veebiserverite protsessori kasutust piiratud erinevalt. Eesmärk on saada vähemalt 10 veebiserverit töökoormuse ühtlustaja taha ilma, et tekiks jõudluse piirang. Samuti on monitooritud Docker käsuga *docker stats [konteiner]* [50], et iga veebiserver kasutaks määratud ressursid täielikult.

Järgnevates peatükkides on lisaks testitulemuste tabelile välja toodud mitu päringut ja samaaegset päringut iga veebiserveri kohta tehakse. Graafikul vertikaalsed jooned punktidel näitavad minimaalse ja maksimaalse päringut sekundis väärtust antud testitulemusel. X-telg iseloomustab serverite arvu ja Y-telg päringute arvu sekundis. Graafiku all on ka minimaalne, maksimaalne ja keskmine tulemus välja toodud, mis saadud kümnel testi tulemusel. Lisaks on igale veebiserveri testitulemustele arvutatud regressioonimudeli tabel. Tabel iseloomustab esimest kuni viimase serverini kus ei toimu jõudluse kasvu kahanemist, kasutades trendiarvutust. Tabelis on välja toodud oodatav keskmine tulemus koos tegeliku tulemusega ning nende erinevus standardhälbega. Oodatav keskmine tulemus on arvutatud valemiga (ptk. 10.1, 10.2):

*sirge vabaliige + serverite arv * sirge kordaja*

Tegeliku keskmise kaugus oodatavast on arvutatud valemiga:

(tegelik keskmine - oodatav keskmine) / ((oodatav 95% piir - oodatav 5% piir) / 2)

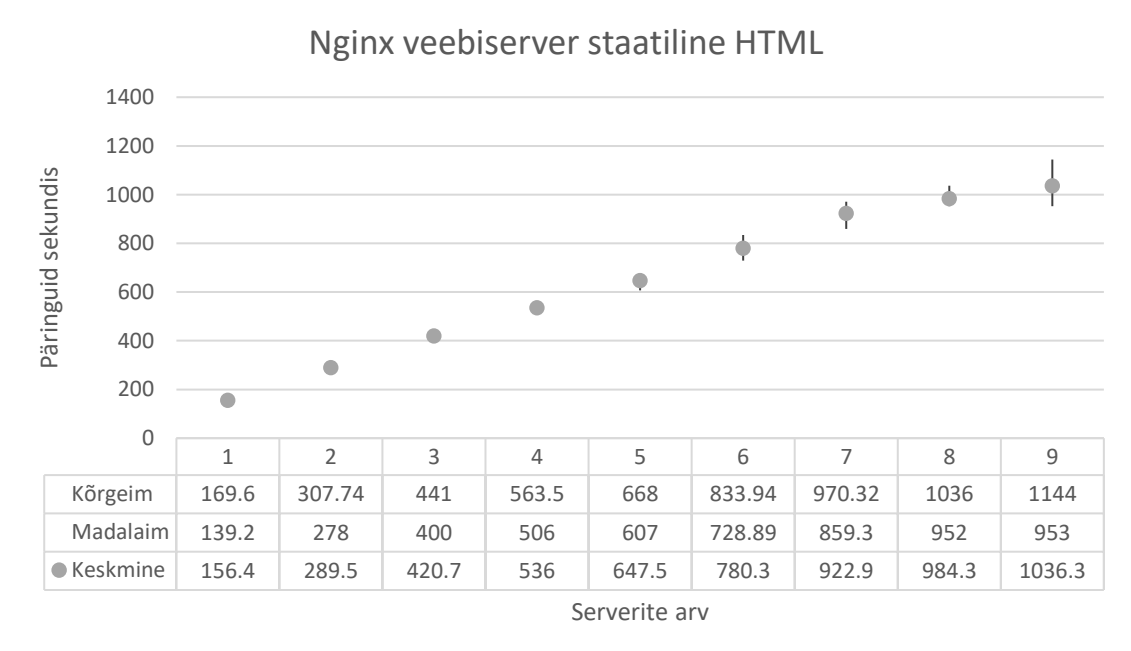
saades tulemuseks standardhälbe kordaja. Täpsemalt on esitatud iga veebiteenuse kohta analüüs koos tulemustega järgnevates peatükkides.

6.1 Nginx töökoormuse ühtlustaja

Järgnevad graafikud ja tabelid põhinevad failidest peatükis 10.1

6.1.1 Nginx veebiserver staatiline HTML

Antud testis on iga ühe veebiserveri kohta 1000 päringut, millest 10 on samaaegsed. Samuti on veebiserveri jõudlus piiratud 2% peale. Tulemustest (Joonis 8. Nginx veebiserver staatiline HTML) on näha, et päringuid sekundis kasvab iga järgneva veebiserveriga lineaarselt, kuni viimase kahe veebiserverini, kus on näha jõudluse kasvu tõusu kahanemist. Vaadates 1 kuni 7 serveri trendi (Tabel 1. Nginx veebiserver staatiline HTML trend 1 kuni 7 serveriga), on kaheksanda serveri oodatav tulemus 1037.34 päringut sekundis, tegelik aga 984.3. Oodatav üheksanda serveri tulemus 1162.63 päringut sekundis, tegelik on aga 1036.3. Antud võrdlusest on näha, et viimasel kahel serveril toimub lineaarse regressiooni trendide arvutustest suurenev erinevus. Tegeliku keskmise kaugus oodatavast erineb vastavalt 1.53 standardhälvet ja 3.38 standardhälvet. On näha, et jõudluse piirang algab kaheksanda serveriga ning tõus kahaneb järgnevaga.



	1	2	3	4	5	6	7	8	9
Kõrgeim	169.6	307.74	441	563.5	668	833.94	970.32	1036	1144
Madalaim	139.2	278	400	506	607	728.89	859.3	952	953
Keskmine	156.4	289.5	420.7	536	647.5	780.3	922.9	984.3	1036.3

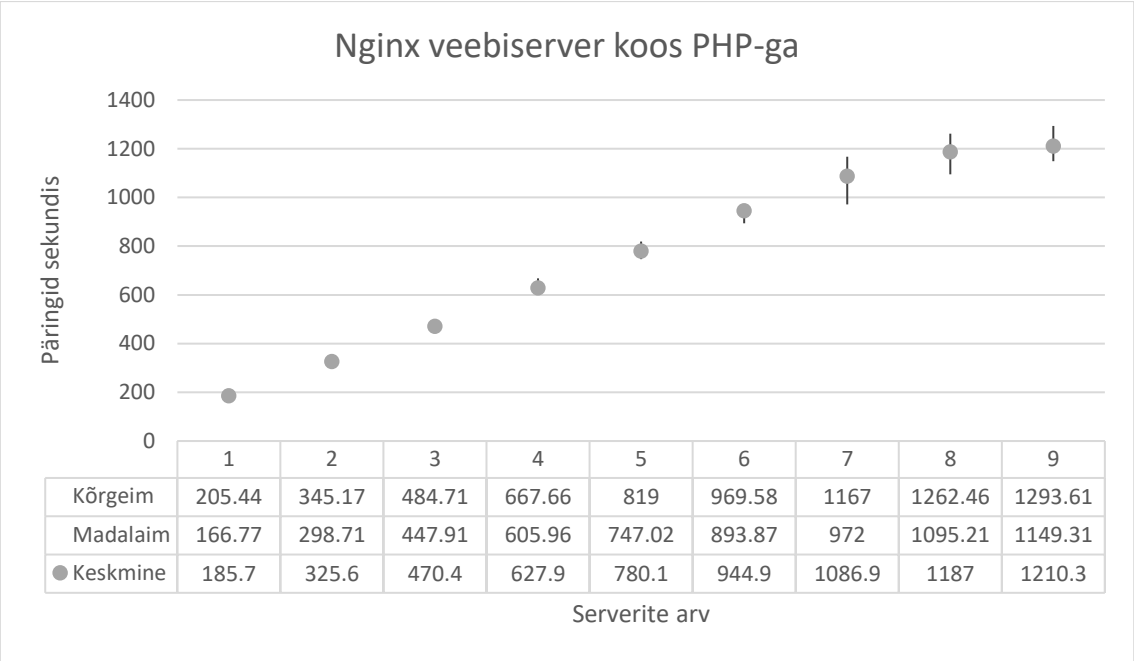
Joonis 8. Nginx veebiserver staatiline HTML

Tabel 1. Nginx veebiserver staatiline HTML trend 1 kuni 7 serveriga

	Koefitsient	*Tegelik keskmine*	*Standard viga*	*t-väärtus*	*P-väärtus*	*Alumine 95%*	*Ülemine 95%*	*Tegeliku keskmise kaugus oodatavast*
Lõikepunkt	35.01		6.22	5.63	0	22.6	47.42	
Üks server	125.29		1.39	90.07	0	122.52	128.07	
Oodatav 8. serveri tulemus	1037.34	984.3				1002.72	1071.96	1.53 standardhälvet
Oodatav 9. serveri tulemus	1162.63	1036.3				1125.23	1200.02	3.38 standardhälvet

6.1.2 Nginx veebiserver koos PHP-ga

Antud testis on iga veebiserveri kohta 1000 päringut, millest 10 on samaaegsed. Samuti on veebiserveri jõudlus piiratud 5% peale. Tulemustest (Joonis 9. Nginx veebiserver koos PHP-ga) on näha, et päringuid sekundis kasvab iga järgneva veebiserveriga lineaarselt, kuni viimase kahe veebiserverini, kus on näha jõudluse kasvu tõusu kahanemist. Vaadates 1 kuni 7 serveri trendi (Tabel 2. Nginx veebiserver koos PHP-ga trend 1 kuni 7 serveriga), on kaheksanda serveri oodatav tulemus 1239.03 päringut sekundis, tegelik aga 1187. Oodatav üheksanda serveri tulemus 1390.88 päringut sekundis, tegelik aga 1210.3. Antud võrdlusest on näha, et viimasel kahel serveril toimub lineaarse regressiooni trendide arvutustest suurenev erinevus. Tegeliku keskmise kaugus oodatavast erineb vastavalt 1.31 standardhälvet ja 4.2 standardhälvet. On näha, et jõudluse piirang algab kaheksanda serveriga ning tõus kahaneb järgnevaga.



	1	2	3	4	5	6	7	8	9
Kõrgeim	205.44	345.17	484.71	667.66	819	969.58	1167	1262.46	1293.61
Madalaim	166.77	298.71	447.91	605.96	747.02	893.87	972	1095.21	1149.31
Keskmine	185.7	325.6	470.4	627.9	780.1	944.9	1086.9	1187	1210.3

Joonis 9. Nginx veebiserver koos PHP-ga

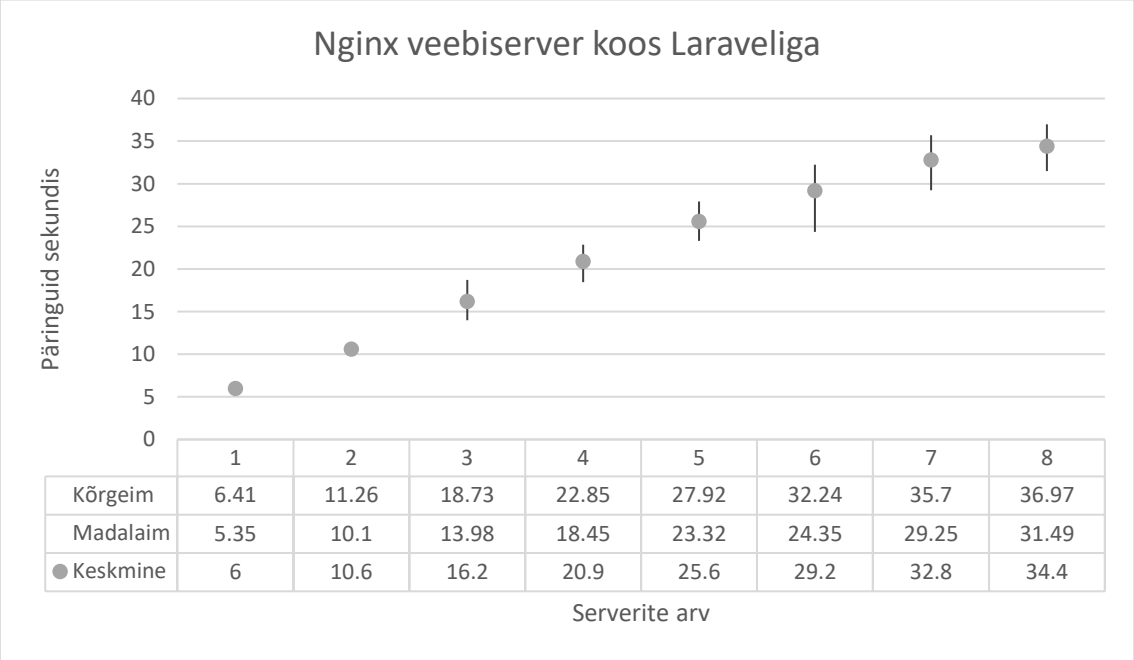
Tabel 2. Nginx veebiserver koos PHP-ga trend 1 kuni 7 serveriga

	Koefitsient	*Tegelik keskmine*	*Standard viga*	*t-väärtus*	*P-väärtus*	*Alumine 95%*	*Ülemine 95%*	Tegeliku keskmise kaugus oodatavast
Lõikepunkt	24.25		7.16	3.39	0	9.97	38.53	
Üks server	151.85		1.6	94.89	0	148.65	155.04	
Oodatav 8. serveri tulemus	1239.03	1187				1199.21	1278.86	1.31 standard hälvet
Oodatav 9. serveri tulemus	1390.88	1210.3				1347.86	1433.9	4.2 standardhälvet

6.1.3 Nginx Laravel koos MySQL andmebaasiga

Antud testis on iga veebiserveri kohta 100 päringut, millest 10 on samaaegsed. Samuti on veebiserveri jõudlus piiratud 50% peale. Tulemustest (Joonis 10. Nginx veebiserver koos Laraveliga) on näha, et päringuid sekundis kasvab iga järgneva veebiserveriga lineaarselt, kuni viimase kahe veebiserverini, kus on näha jõudluse kasvu tõusu kahanemist. Vaadates 1 kuni 6 serveri trendi (Tabel 3. Nginx veebiserver koos Laraveliga

trend 1 kuni 6 serveriga), on seitsmenda serveri oodatav tulemus 34.68 päringut sekundis, tegelik aga 32.8. Oodatav kaheksanda serveri tulemus 39.42 päringut sekundis, tegelik on aga 34.4. Antud võrdlusest on näha, et viimasel kahel serveril toimub lineaarse regressiooni trendide arvutustest suurenev erinevus. Tegeliku keskmise kaugus oodatavast erineb vastavalt 0.8 standardhälvet ja 1.95 standardhälvet. On näha, et jõudluse piirang algab kaheksanda serveriga ning tõus kahaneb järgnevaga.



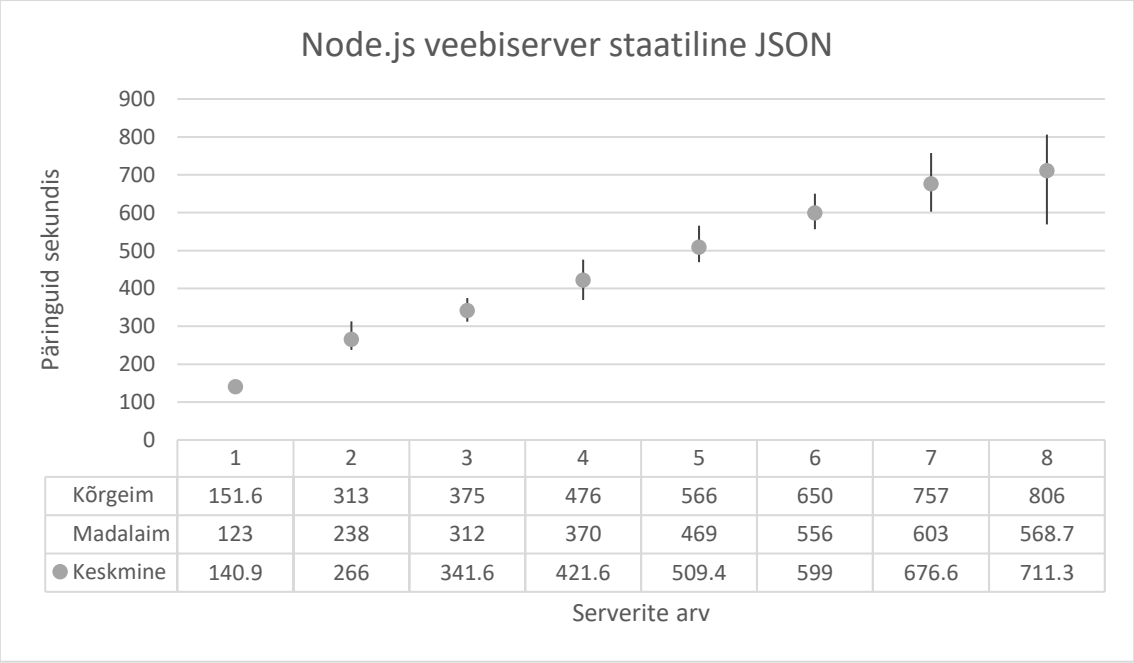
Joonis 10. Nginx veebiserver koos Laraveliga

Tabel 3. Nginx veebiserver koos Laraveliga trend 1 kuni 6 serveriga

	Koefitsient	*Tegelik keskmine*	*Standardviga*	*t-väärtus*	*P-väärtus*	*Alumine 95%*	*Ülemine 95%*	Tegeliku keskmise kaugus oodatavast
Lõikepunkt	1.5		0.42	3.57	0	0.66	2.35	
Üks server	4.74		0.11	43.82	0	4.52	4.96	
Oodatav 7. serveri tulemus	34.68	32.8				32.32	37.04	0.8 standardhälvet
Oodatav 8. serveri tulemus	39.42	34.4				36.85	42	1.95 standardhälvet

6.1.4 Node.js veebiserver staatiline JSON

Antud testis on iga veebiserveri kohta 1000 päringut, millest 10 on samaaegsed. Samuti on veebiserveri jõudlus piiratud 5% peale. Tulemustest (Joonis 11. Node.js veebiserver staatiline JSON) on näha, et päringuid sekundis kasvab iga järgneva veebiserveriga lineaarselt, kuni viimase kahe veebiserverini, kus on näha jõudluse kasvu tõusu kahanemist. Vaadates 1 kuni 6 serveri trendi (Tabel 4. Node.js veebiserver staatiline JSON trend 1 kuni 6 serveriga), on seitsmenda serveri oodatav tulemus 689.83 päringut sekundis, tegelik aga 676.6. Oodatav kaheksanda serveri tulemus 778.42 päringut sekundis, tegelik on aga 711.3. Antud võrdlusest on näha, et viimasel kahel serveril toimub lineaarse regressiooni trendide arvutustest suurenev erinevus. Tegeliku keskmise kaugus oodatavast erineb vastavalt 0.28 standardhälvet ja 1.32 standardhälvet. Graafikult on ka näha, et seitsmendal ja kaheksandal serveril on suurem madalaima ja kõrgeima päringute arv sekundis erinevus võrreldes eelnevatega, mis viitab jõudluse ebastabiilsusele. On näha, et jõudluse piirang algab seitsmenda serveriga ning tõus kahaneb järgnevaga.



	1	2	3	4	5	6	7	8
Kõrgeim	151.6	313	375	476	566	650	757	806
Madalaim	123	238	312	370	469	556	603	568.7
Keskmine	140.9	266	341.6	421.6	509.4	599	676.6	711.3

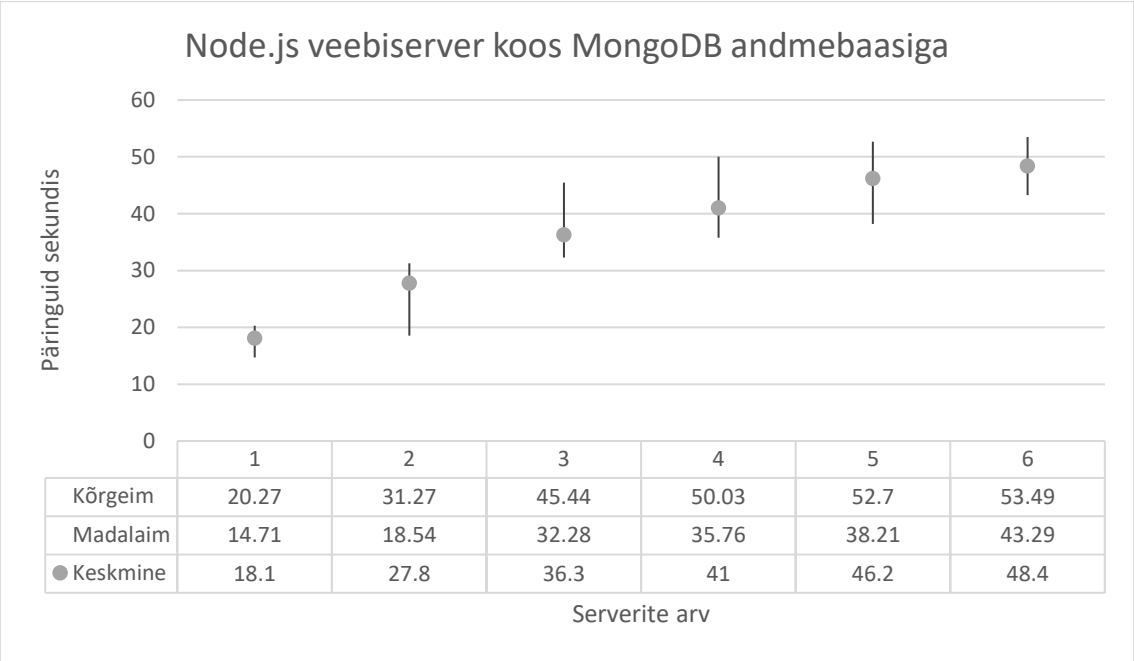
Joonis 11. Node.js veebiserver staatiline JSON

Tabel 4. Node.js veebiserver staatiline JSON trend 1 kuni 6 serveriga

	Koefitsient	*Tegelik keskmine*	*Standard viga*	*t-väärtus*	*P-väärtus*	*Alumine 95%*	*Ülemine 95%*	Tegeliku keskmise kaugus oodatavast
Lõikepunkt	69.66		8.3	8.4	0	53.05	86.26	
Üks server	88.6		2.13	41.59	0	84.33	92.86	
Oodatav 7. serveri tulemus	689.83	676.6				643.38	736.28	0.28 standard hälvet
Oodatav 8. serveri tulemus	778.42	711.3				727.71	829.14	1.32 standard hälvet

6.1.5 Node.js veebiserver koos MongoDB andmebaasiga

Antud testis on iga veebiserveri kohta 100 päringut, millest 10 on samaaegsed. Samuti on veebiserveri jõudlus piiratud 5% peale. Tulemustest (Joonis 12. Node.js veebiserver koos MongoDB andmebaasiga) on näha, et päringuid sekundis kasvab iga järgneva veebiserveriga lineaarselt, kuni viimase kolme veebiserverini, kus on näha jõudluse kasvu tõusu kahanemist. Vaadates 1 kuni 4 serveri trendi (Tabel 5. Node.js veebiserver koos MongoDB andmebaasiga trend 1 kuni 4 serveriga), on viienda serveri oodatav tulemus 50.1 päringut sekundis, tegelik aga 41. Oodatav kuuenda serveri tulemus 57.82 päringut sekundis, tegelik aga 46.2. Oodatav kuuenda serveri tulemus 65.54 päringut sekundis, tegelik on aga 48.4. Antud võrdlusest on näha, et viimasel kolmel serveril toimub lineaarse regressiooni trendide arvutustest suurenev erinevus. Tegeliku keskmise kaugus oodatavast erineb vastavalt 1.14 standardhälvet, 1.29 standardhälvet ja 1.71. On näha, et jõudluse piirang algab viienda serveriga ning tõus kahaneb kahe järgnevaga.



Joonis 12. Node.js veebiserver koos MongoDB andmebaasiga

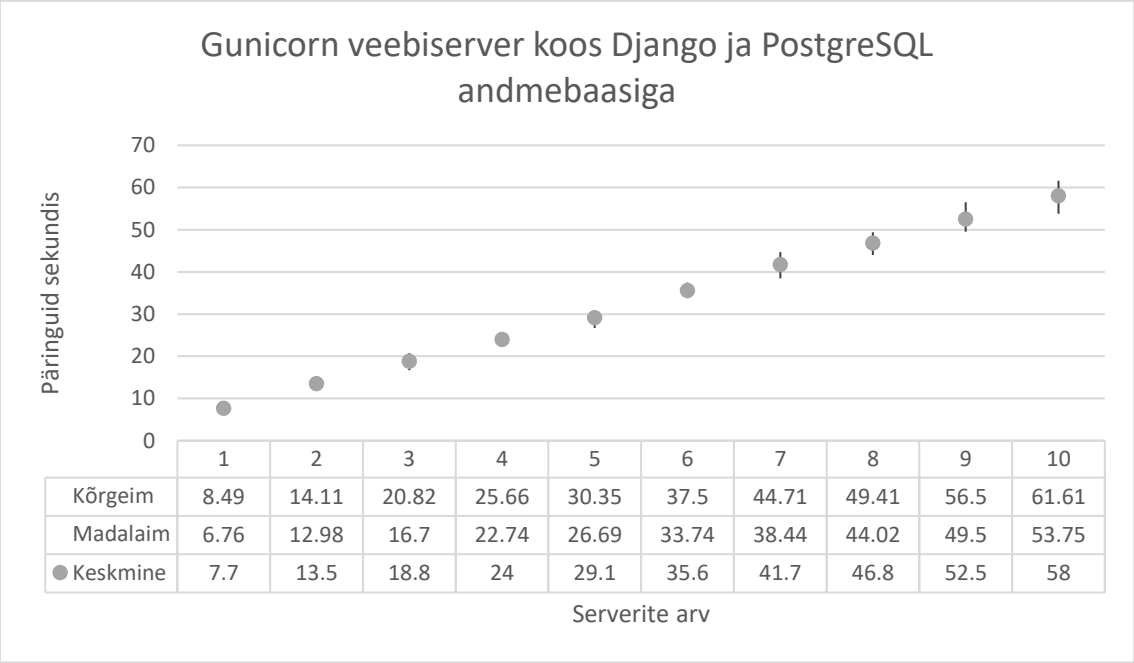
Tabel 5. Node.js veebiserver koos MongoDB andmebaasiga trend 1 kuni 4 serveriga

	Koefitsient	*Tegelik keskmine*	*Standard viga*	*t-väärtus*	*P-väärtus*	*Alumine 95%*	*Ülemine 95%*	Tegeliku keskmise kaugus oodatavast
Lõikepunkt	11.51		1.39	8.28	0	8.7	14.33	
Üks server	7.72		0.51	15.2	0	6.69	8.75	
Oodata v 5. serveri tulemus	50.1	41				42.15	58.06	1.14 standard hälvet
Oodata v 6. serveri tulemus	57.82	46.2				48.84	66.8	1.29 standard hälvet
Oodata v 7. serveri tulemus	65.54	48.4				55.53	75.55	1.71 standard hälvet

6.1.6 Gunicorn veebiserver koos Django ja PostgreSQL andmebaasiga

Antud testis on iga veebiserveri kohta 100 päringut, millest 10 on samaaegsed. Samuti on veebiserveri jõudlus piiratud 10% peale. Tulemustest (Joonis 13. Gunicorn veebiserver

koos Django ja PostgreSQL andmebaasiga) on näha, et päringuid sekundis kasvab iga järgneva veebiserveriga lineaarselt. Vaadates 1 kuni 9 serveri trendi (Tabel 6. Gunicorn veebiserver koos Django ja PostgreSQL andmebaasiga trend 1 kuni 9 serveriga), on kümenda serveri oodatav tulemus 58.03 päringut sekundis, tegelik on aga 58. Antud võrdlusest on näha, et viimasel serveril ei toimu lineaarse regressiooni trendide arvutustest suurenev erinevus. Tegeliku keskmise kaugus oodatavast erineb 0.02 standardhälvet. On näha, et jõudluse piirang puudub 10 serveriga antud veebirakendusel kasutades töökoormuse ühtlustamist.



	1	2	3	4	5	6	7	8	9	10
Kõrgeim	8.49	14.11	20.82	25.66	30.35	37.5	44.71	49.41	56.5	61.61
Madalaim	6.76	12.98	16.7	22.74	26.69	33.74	38.44	44.02	49.5	53.75
Keskmine	7.7	13.5	18.8	24	29.1	35.6	41.7	46.8	52.5	58

Joonis 13. Gunicorn veebiserver koos Django ja PostgreSQL andmebaasiga

Tabel 6. Gunicorn veebiserver koos Django ja PostgreSQL andmebaasiga trend 1 kuni 9 serveriga

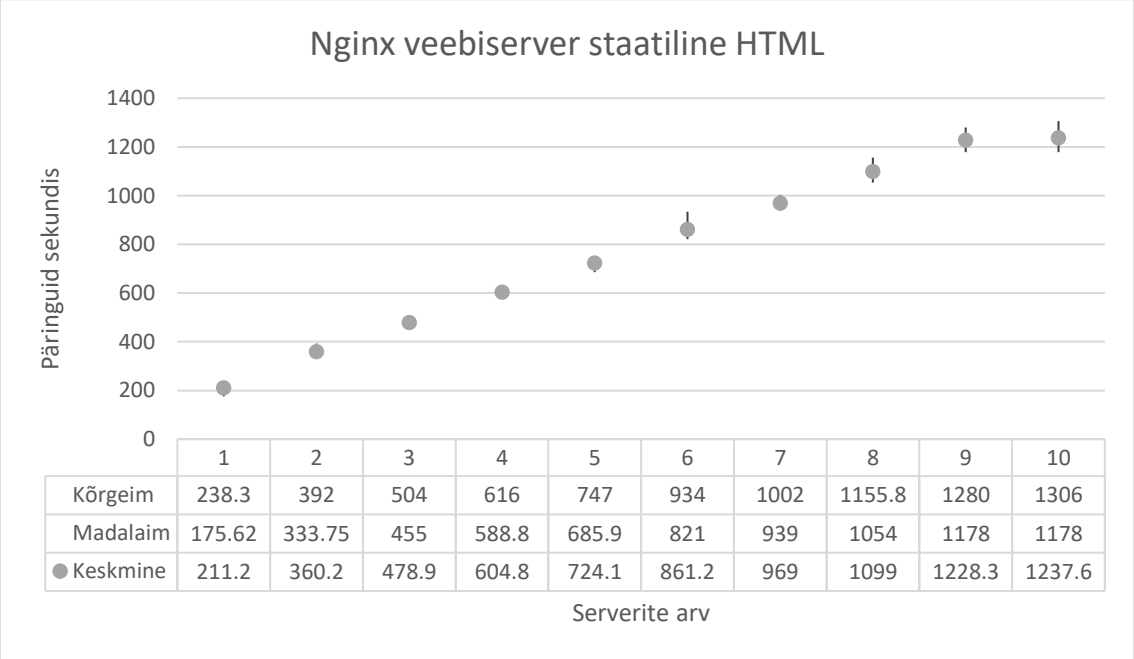
	Koefitsient	*Tegelik keskmine*	*Standard viga*	*t-väärtus*	*P-väärtus*	*Alumine 95%*	*Ülemine 95%*	Tegeliku keskmise kaugus oodatavast
Lõikepunkt	1.93		0.3	6.42	0	1.33	2.53	
Üks server	5.61		0.05	104.88	0	5.5	5.72	
Oodatav 10. serveri tulemus	58.03	58				56.37	59.69	0.02 standard hälvet

6.2 HAProxy töökoormuse ühtlustaja

Järgnevad graafikud ja tabelid põhinevad failidest peatükis 10.2

6.2.1 Nginx veebiserver staatiline HTML

Antud testis on iga veebiserveri kohta 1000 päringut, millest 10 no samaaegsed. Samuti on veebiserveri jõudlus piiratud 2% peale. Tulemustest (Joonis 14. Nginx veebiserver staatiline HTML) on näha, et päringuid sekundis kasvab iga järgneva veebiserveriga lineaarselt, kuni viimase veebiserverini, kus on näha jõudluse kasvu tõusu järsku kahanemist. Vaadates 1 kuni 9 serveri trendi (Tabel 7. Nginx veebiserver staatiline HTML trend 1 kuni 8 serveriga), on kümnenda serveri oodatav tulemus 1353.07 päringut sekundis, tegelik on aga 1237.6 päringut sekundis. Antud võrdlusest on näha, et viimasel serveril toimub lineaarse regressiooni trendide arvutustest erinevus. Tegeliku keskmise kaugus oodatavast erineb vastavalt 0.03 standardhälvet ja 3.46 standardhälvet vastavalt. On näha, et jõudluse piirang algab kümnenda serveriga.



	1	2	3	4	5	6	7	8	9	10
Kõrgeim	238.3	392	504	616	747	934	1002	1155.8	1280	1306
Madalaim	175.62	333.75	455	588.8	685.9	821	939	1054	1178	1178
Keskmine	211.2	360.2	478.9	604.8	724.1	861.2	969	1099	1228.3	1237.6

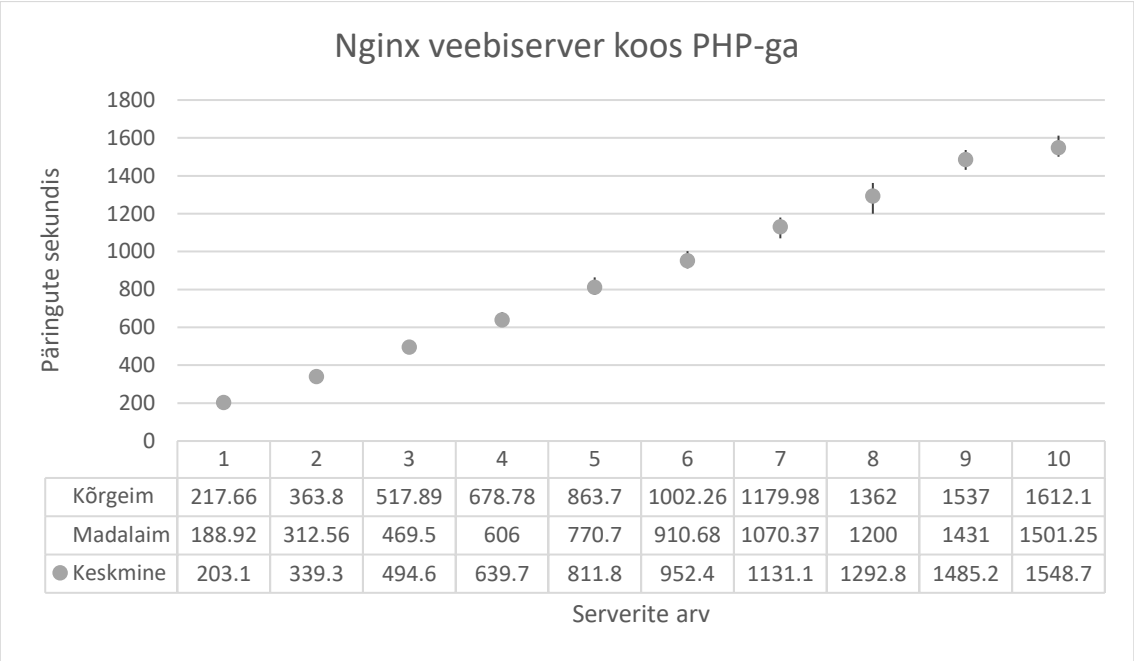
Joonis 14. Nginx veebiserver staatiline HTML

Tabel 7. Nginx veebiserver staatiline HTML trend 1 kuni 8 serveriga

	Koefitsient	*Tegelik keskmine*	*Standard viga*	*t-väärtus*	*P-väärtus*	*Alumine 95%*	*Ülemine 95%*	Tegeliku keskmise kaugus oodatavast
Lõikepunkt	99.75		5.6	17.81	0	88.6	110.9	
Üks server	125.29		1.11	112.98	0	123.08	127.5	
Oodatav 9. serveri tulemus	1227.36	1228.3				1196.35	1258.38	0.03 standard hälvet
Oodatav 10. serveri tulemus	1352.66	1237.6				1319.43	1385.88	3.46 standard hälvet

6.2.2 Nginx veebiserver koos PHP-ga

Antud testis on iga veebiserveri kohta 1000 päringut, millest 10 on samaaegsed. Samuti on veebiserveri jõudlus piiratud 2% peale. Tulemustest (Joonis 15. Nginx veebiserver koos PHP-ga) on näha, et päringuid sekundis kasvab iga järgneva veebiserveriga lineaarselt, kuni viimase kahe veebiserverini, kus on näha jõudluse kasvu tõusu kahanemist. Vaadates 1 kuni 8 serveri trendi (Tabel 8. Nginx veebiserver koos PHP-ga trend 1 kuni 8 serveriga), on üheksanda serveri oodatav tulemus 1436.59 päringut sekundis, tegelik aga 1485.2. Oodatav kümnenda serveri tulemus 1592.92 päringut sekundis, tegelik on aga 1548.7. Antud võrdlusest on näha, et viimasel kahel serveril toimub lineaarse regressiooni trendide arvutustest suurenev erinevus. Tegeliku keskmise kaugus oodatavast erineb vastavalt 1.1 standardhälvet ja 0.93 standardhälvet. On näha, et jõudluse piirang algab kaheksanda serveriga ning tõus kahaneb järgnevaga.



Joonis 15. Nginx veebiserver koos PHP-ga

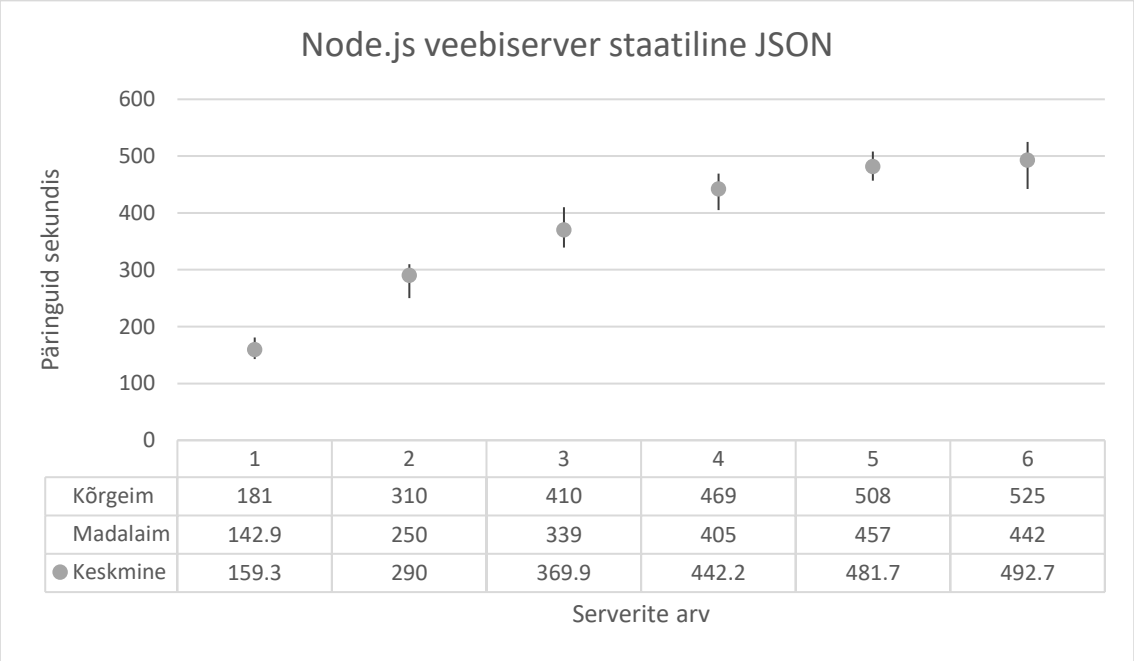
Tabel 8. Nginx veebiserver koos PHP-ga trend 1 kuni 8 serveriga

	Koefitsient	*Tegelik keskmine*	*Standard viga*	*t-väärtus*	*P-väärtus*	*Alumine 95%*	*Ülemine 95%*	Tegeliku keskmise kaugus oodatavast
Lõikepunkt	29.59		7.99	3.7	0	13.67	45.51	
Üks server	156.33		1.58	98.74	0	153.18	159.49	
Oodatav 9. serveri tulemus	1436.59	1485.2				1392.3	1480.87	1.1 standardhälvet
Oodatav 10. serveri tulemus	1592.92	1548.7				1545.48	1640.36	0.93 standard hälvet

6.2.3 Node.js veebiserver staatiline JSON

Antud testis on iga veebiserveri kohta 1000 päringut, millest 10 on samaaegsed. Samuti on veebiserveri jõudlus piiratud 5% peale. Tulemustest (Joonis 16. Node.js veebiserver staatiline JSON) on näha, et päringuid sekundis kasvab iga järgneva veebiserveriga lineaarselt, kuni viimase kahe veebiserverini, kus on näha jõudluse kasvu tõusu kahanemist. Vaadates 1 kuni 4 serveri trendi (Tabel 9. Node.js veebiserver staatiline

JSON trend 1 kuni 4 serveriga), on viienda serveri oodatav tulemus 547.48 päringut sekundis, tegelik aga 481.7. Oodatav kuuenda serveri tulemus 640.33 päringut sekundis, tegelik on aga 492.7. Antud võrdlusest on näha, et viimasel kahel serveril toimub lineaarse regressiooni trendide arvutustest suurenev erinevus. Tegeliku keskmise kaugus oodatavast erineb vastavalt 1.18 standardhälvet ja 2.34 standardhälvet. On näha, et piirang algab viienda serveriga ning tõus kahaneb järgnevaga.



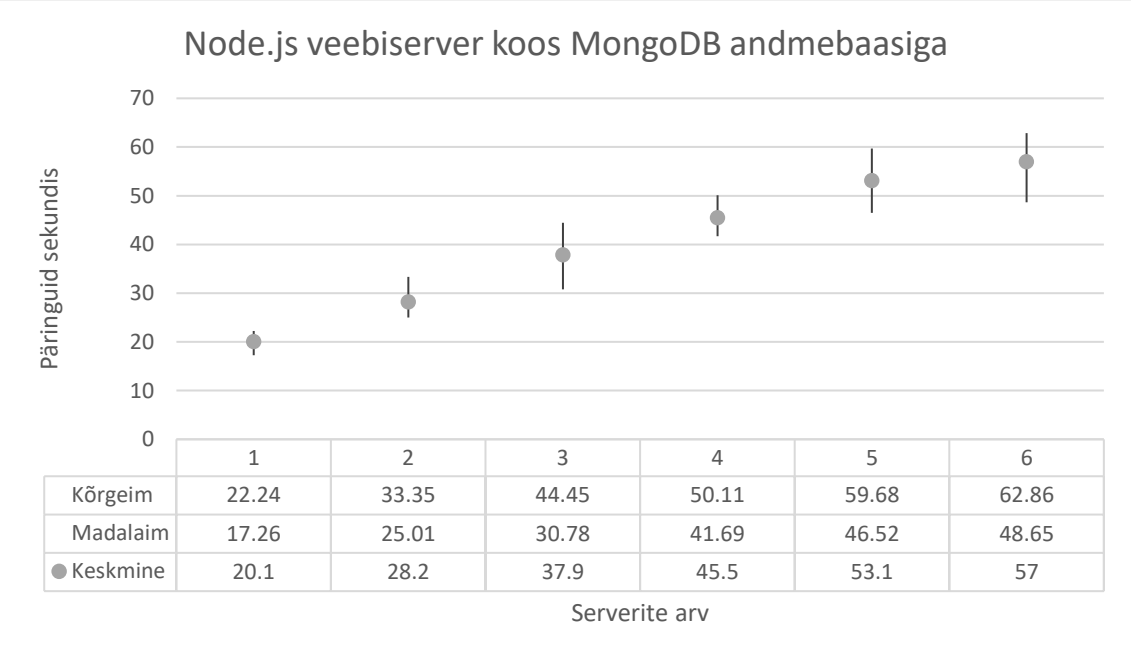
Joonis 16. Node.js veebiserver staatiline JSON

Tabel 9. Node.js veebiserver staatiline JSON trend 1 kuni 4 serveriga

	Koefitsient	*Tegelik keskmine*	*Standard viga*	*t-väärtus*	*P-väärtus*	*Alumine 95%*	*Ülemine 95%*	Tegeliku keskmise kaugus oodatavast
Lõikepunkt	83.23		9.78	8.51	0	63.43	103.03	
Üks server	92.85		3.57	25.99	0	85.62	100.08	
Oodatav 5. serveri tulemus	547.48	481.7				491.52	603.44	1.18 standard hälvet
Oodatav 6. serveri tulemus	640.33	492.7				577.13	703.53	2.34 standard hälvet

6.2.4 Node.js veebiserver koos MongoDB andmebaasiga

Antud testis on iga veebiserveri kohta 100 päringut, millest 10 on samaaegsed. Samuti on veebiserveri jõudlus piiratud 5% peale. Tulemustest (Joonis 17. Node.js veebiserver koos MongoDB andmebaasiga) on näha, et päringuid sekundis kasvab iga järgneva veebiserveriga lineaarselt, kuni viimase veebiserverini, kus on näha jõudluse kasvu tõusu kahanemist. Vaadates 1 kuni 5 serveri trendi (Tabel 10. Node.js veebiserver koos MongoDB andmebaasiga trend 1 kuni 5 serveriga), on kuuenda serveri oodatav tulemus 61.95 päringut sekundis, tegelik on aga 57. Antud võrdlusest on näha, et viimasel serveril toimub lineaarse regressiooni trendide arvutustest suurenev erinevus. Tegeliku keskmise kaugus oodatavast erineb vastavalt 0.86 standardhälvet. On näha, et piirang algab viienda serveriga.



	1	2	3	4	5	6
Kõrgeim	22.24	33.35	44.45	50.11	59.68	62.86
Madalaim	17.26	25.01	30.78	41.69	46.52	48.65
Keskmine	20.1	28.2	37.9	45.5	53.1	57

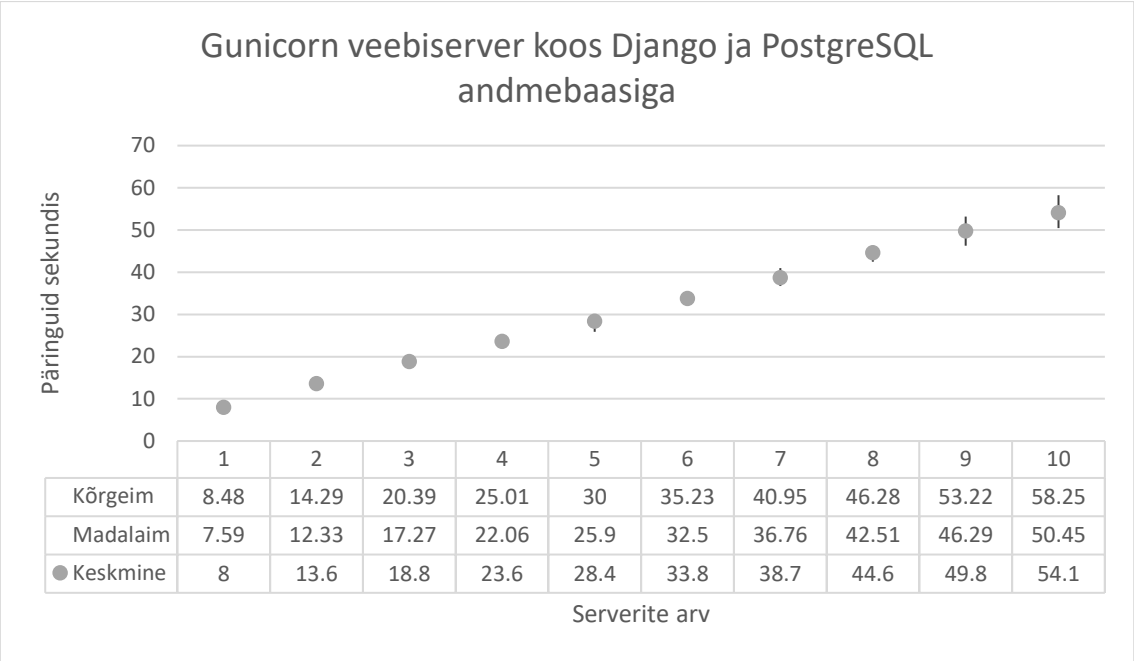
Joonis 17. Node.js veebiserver koos MongoDB andmebaasiga

Tabel 10. Node.js veebiserver koos MongoDB andmebaasiga trend 1 kuni 5 serveriga

	Koefitsient	*Tegelik keskmine*	*Standard viga*	*t-väärtus*	*P-väärtus*	*Alumine 95%*	*Ülemine 95%*	Tegeliku keskmise kaugus oodatavast
Lõikepunkt	11.97		1.02	11.77	0	9.93	14.02	
Üks server	8.33		0.31	27.17	0	7.71	8.95	
Oodatav 6. serveri tulemus	61.95	57				56.21	67.7	0.86 standard hälvet

6.2.5 Gunicorn veebiserver koos Django ja PostgreSQL andmebaasiga

Antud testis on iga veebiserveri kohta 100 päringut, millest 10 on samaaegsed. Samuti on veebiserveri jõudlus piiratud 10% peale. Tulemustest (Joonis 18. Gunicorn veebiserver koos Django ja PostgreSQL andmebaasiga) on näha, et päringuid sekundis kasvab iga järgneva veebiserveriga lineaarselt. Vaadates 1 kuni 9 serveri trendi (Tabel 11. Gunicorn veebiserver koos Django ja PostgreSQL andmebaasiga trend 1 kuni 9 serveriga), on kümenda serveri oodatav tulemus 54.67 päringut sekundis, tegelik aga 54.1. Antud võrdlusest on näha, et viimasel serveril ei toimu lineaarse regressiooni trendide arvutustest suurenev erinevus. Tegeliku keskmise kaugus oodatavast erineb 0.38 standardhälvet. On näha, et jõudluse piirang puudub 10 serveriga antud veebirakendusel, kasutades töökoormuse ühtlustamist.



Joonis 18. Gunicorn veebiserver koos Django ja PostgreSQL andmebaasiga

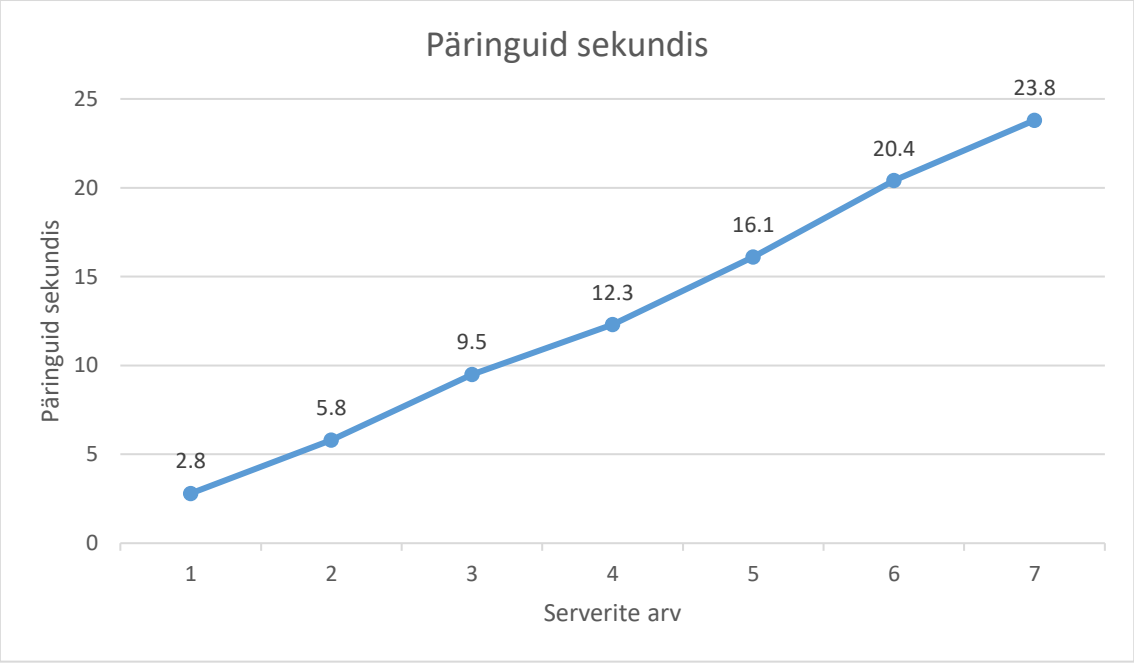
Tabel 11. Gunicorn veebiserver koos Django ja PostgreSQL andmebaasiga trend 1 kuni 9 serveriga

	Koefitsient	*Tegelik keskmine*	*Standard viga*	*t-väärtus*	*P-väärtus*	*Alumine 95%*	*Ülemine 95%*	Tegeliku keskmise kaugus oodatavast
Lõikepunkt	2.98		0.27	11.05	0	2.44	3.51	
Üks server	5.17		0.05	108.02	0	5.07	5.26	
Oodatav 10. serveri tulemus	54.67	54.1				53.18	56.16	0.38 standard hälvet

7 JMeter

7.1 Nginx veebiserver koos Laravel raamistikuga

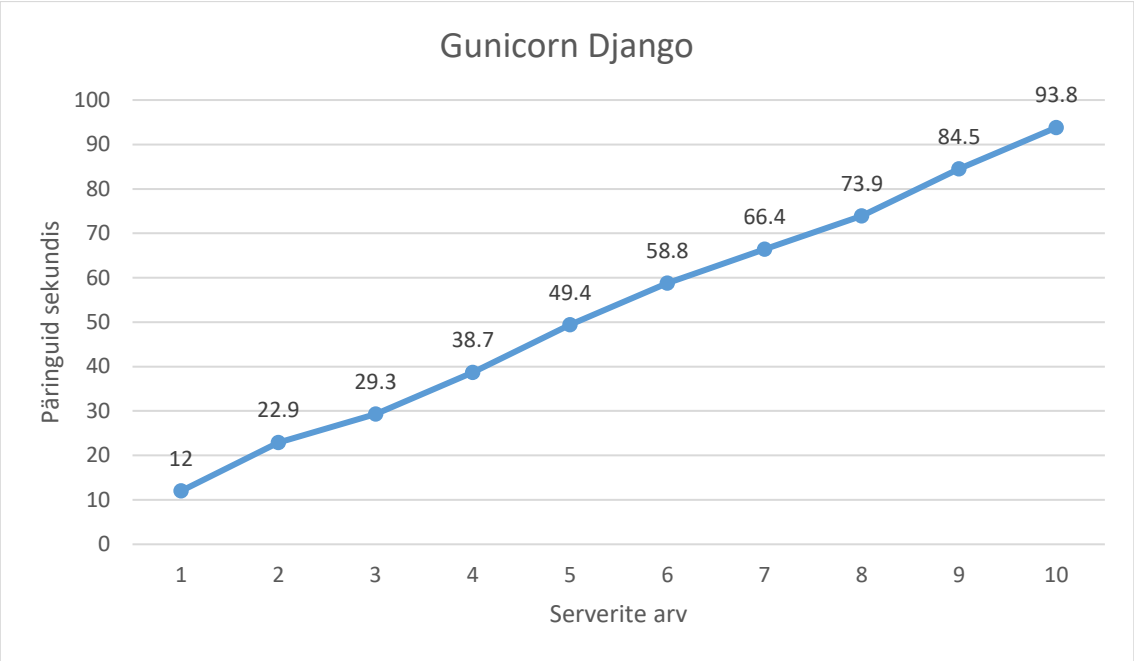
Antud testis on iga test 200 päringut, millest 10 samaaegsed. Samuti on veebiserveri jõudlus piiratud 50% peale. Graafikult (Joonis 19. JMeter GET, POST päringute jõudlustest Nginx veebiserver koos Laravel raamistikuga), on näha lineaarset päringute arvu sekundis kasvu.



Joonis 19. JMeter GET, POST päringute jõudlustest Nginx veebiserver koos Laravel raamistikuga

7.2 Gunicorn veebiserver koos Django raamistikuga

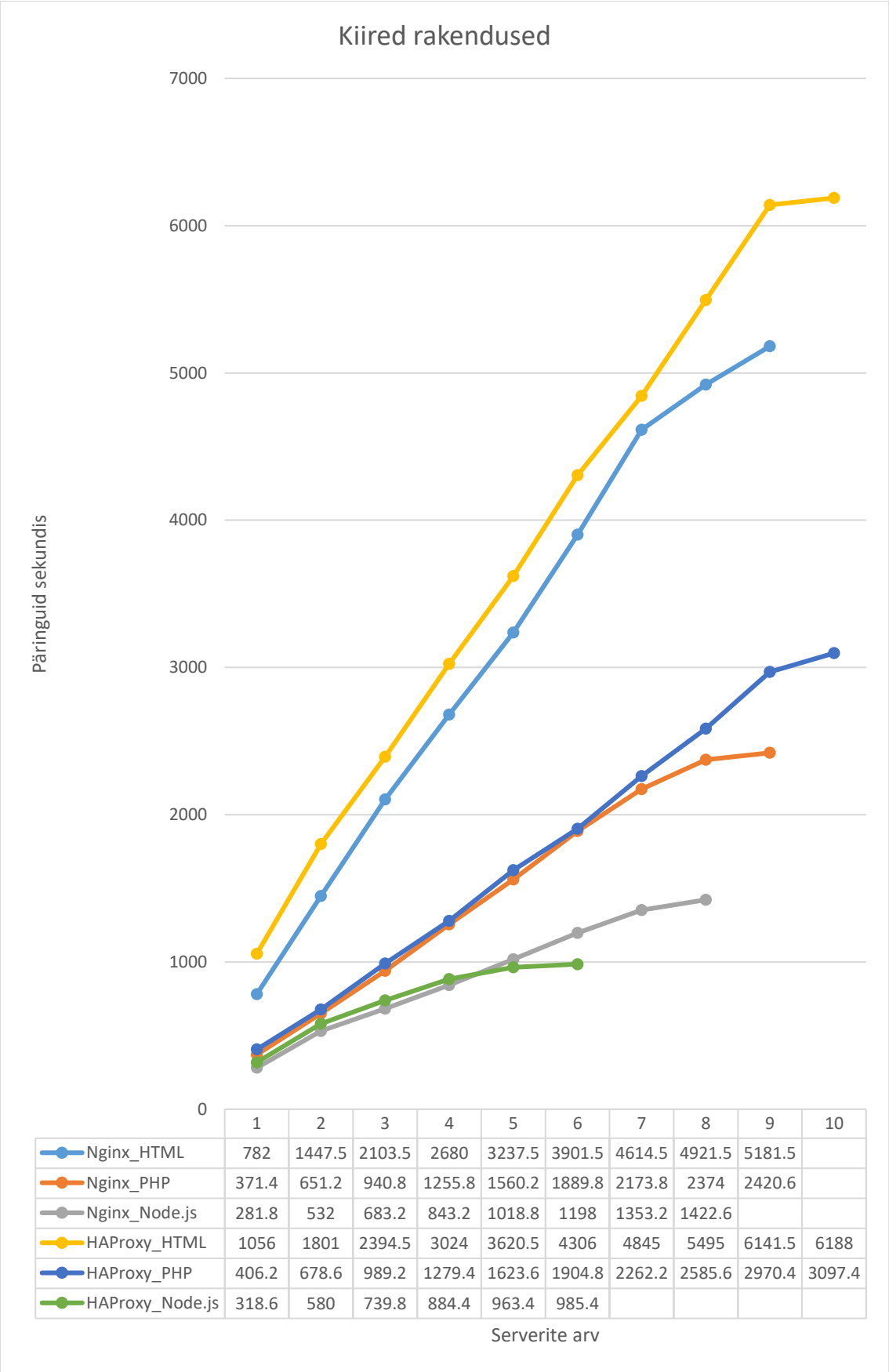
Antud testis on iga test 200 päringut, millest 10 on samaaegsed. Samuti on veebiserveri jõudlus piiratud 10% peale. Graafikult (Joonis 20. JMeter GET, POST päringute jõudlustest Gunicorn veebiserver koos Django raamistikuga) on näha lineaarset päringute arvu sekundis kasvu.



Joonis 20. JMeter GET, POST päringute jõudlustest Gunicorn veebiserver koos Django raamistikuga

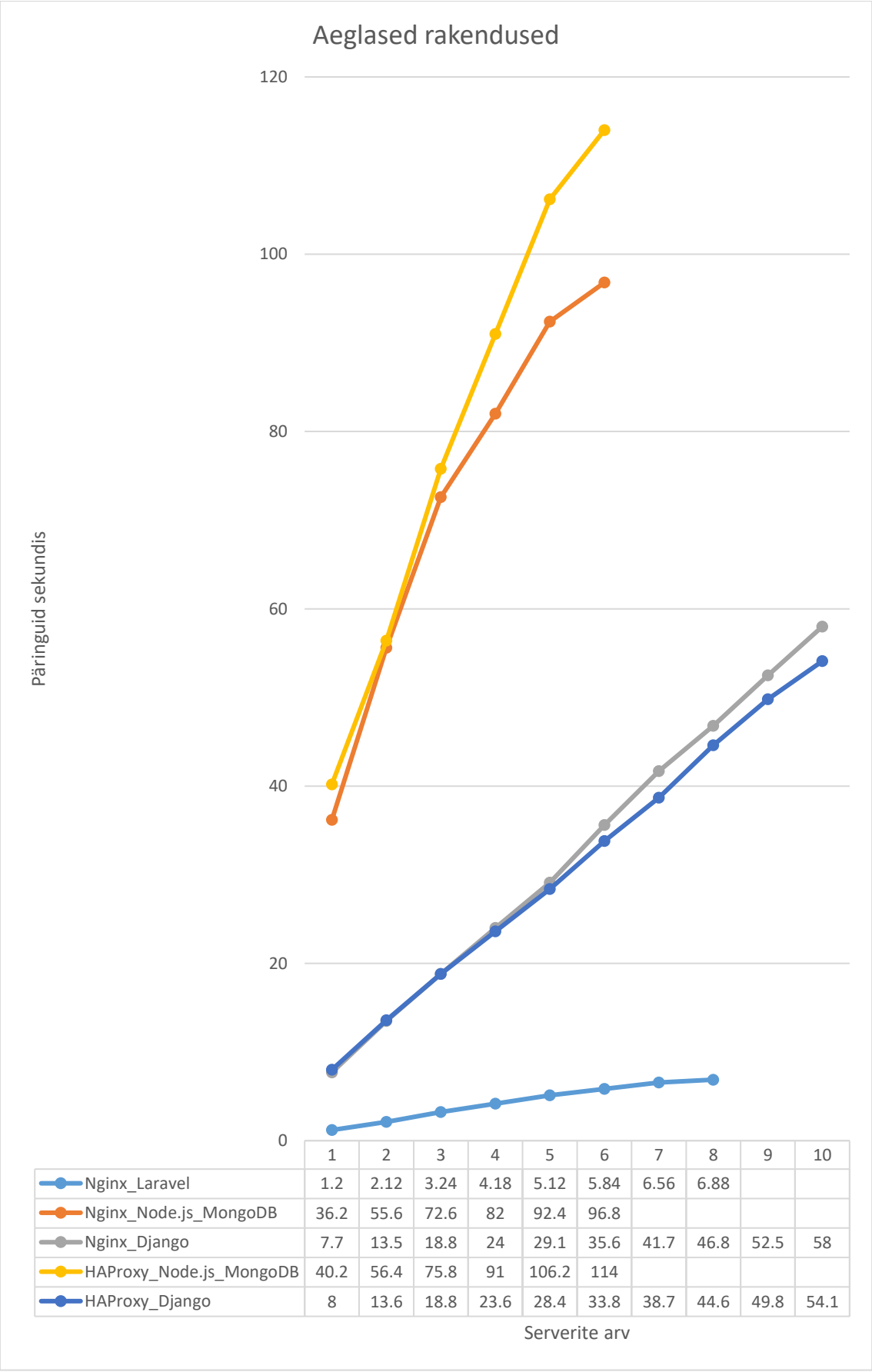
8 Testitulemuste analüüs

Tulemused on arvutatud näitama 10% protsendilist jõudlust. Antud veebirakendused on jaotatud kahte gruppi: aeglased ja kiired rakendused. Aeglaste veebirakenduste alla kuuluvad rakendused, mille kõige kõrgem päringute arv sekundis piirdub 100-ga ning kiirete veebirakenduste alla kuuluvad rakendused, mille aeglaseim kiirus on üle 100 päringu sekundis. Vastavalt on tulemused kokku pandud illustreerival eesmärgil kahes graafikus (Joonis 22. Aeglased veebirakendused, Joonis 21. Kiired veebirakendused). Lisaks on kõik tulemused koos graafikus (Joonis 23. Kõik veebirakendused). Kuna tulemustel on suured erinevused, kasutati graafikus kümnendlogaritmimist. Samuti on kõik arvulised väärtused toodud välja tabelis (Tabel 12. Kõikide veebirakenduste kiirus). Graafikul ja tabelis, kus kõik andmed on koos, on tulemused tähistatud vastavate värvidega. Järgnevad graafikud ja tabelid põhinevad failidest peatükis 10.3.



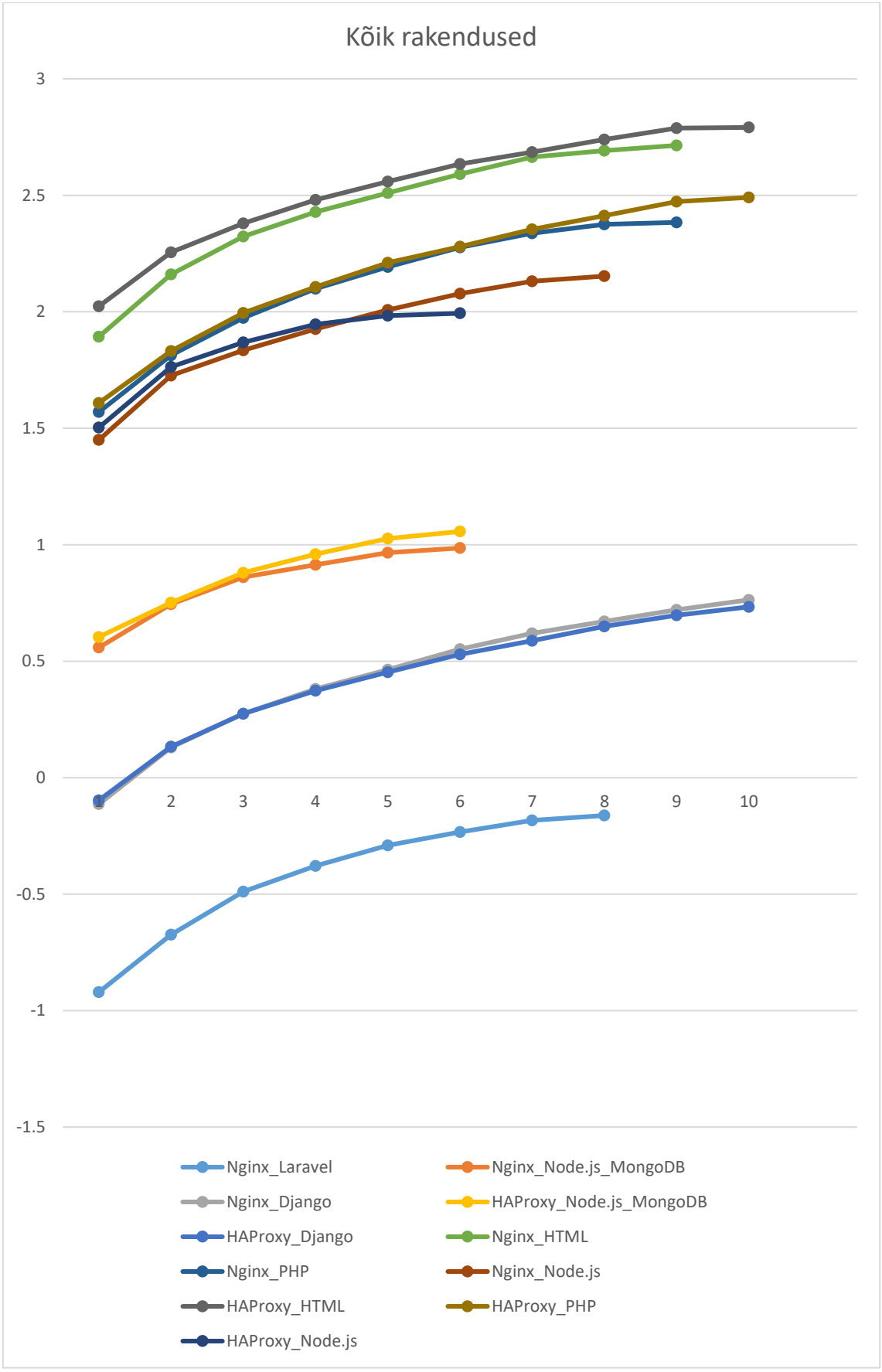
	1	2	3	4	5	6	7	8	9	10
Nginx_HTML	782	1447.5	2103.5	2680	3237.5	3901.5	4614.5	4921.5	5181.5	
Nginx_PHP	371.4	651.2	940.8	1255.8	1560.2	1889.8	2173.8	2374	2420.6	
Nginx_Node.js	281.8	532	683.2	843.2	1018.8	1198	1353.2	1422.6		
HAProxy_HTML	1056	1801	2394.5	3024	3620.5	4306	4845	5495	6141.5	6188
HAProxy_PHP	406.2	678.6	989.2	1279.4	1623.6	1904.8	2262.2	2585.6	2970.4	3097.4
HAProxy_Node.js	318.6	580	739.8	884.4	963.4	985.4				

Joonis 21. Kiired veebirakendused



	1	2	3	4	5	6	7	8	9	10
Nginx_Laravel	1.2	2.12	3.24	4.18	5.12	5.84	6.56	6.88		
Nginx_Node.js_MongoDB	36.2	55.6	72.6	82	92.4	96.8				
Nginx_Django	7.7	13.5	18.8	24	29.1	35.6	41.7	46.8	52.5	58
HAProxy_Node.js_MongoDB	40.2	56.4	75.8	91	106.2	114				
HAProxy_Django	8	13.6	18.8	23.6	28.4	33.8	38.7	44.6	49.8	54.1

Joonis 22. Aeglased veebirakendused



Joonis 23. Kõik veebirakendused

Tabel 12. Kõikide veebirakenduste kiirus

	Nginx_Laravel	Nginx_Node.js_MongoDB	Nginx_Django	HAProxy_Node.js_MongoDB	HAProxy_Django	Nginx_HTML	Nginx_PHP	Nginx_Node.js	HAProxy_HTML	HAProxy_PHP	HAProxy_Node.js
1	1.2	36.2	7.7	40.2	8	782	371.4	281.8	1056	406.2	318.6
2	2.12	55.6	13.5	56.4	13.6	1447.5	651.2	532	1801	678.6	580
3	3.24	72.6	18.8	75.8	18.8	2103.5	940.8	683.2	2394.5	989.2	739.8
4	4.18	82	24	91	23.6	2680	1255.8	843.2	3024	1279.4	884.4
5	5.12	92.4	29.1	106.2	28.4	3237.5	1560.2	1018.8	3620.5	1623.6	963.4
6	5.84	96.8	35.6	114	33.8	3901.5	1889.8	1198	4306	1904.8	985.4
7	6.56	#N/A	41.7	#N/A	38.7	4614.5	2173.8	1353.2	4845	2262.2	#N/A
8	6.88	#N/A	46.8	#N/A	44.6	4921.5	2374	1422.6	5495	2585.6	#N/A
9	#N/A	#N/A	52.5	#N/A	49.8	5181.5	2420.6	#N/A	6141.5	2970.4	#N/A
10	#N/A	#N/A	58	#N/A	54.1	#N/A	#N/A	#N/A	6188	3097.4	#N/A

8.1 Nginx veebiserver staatiline HTML

Joonisel (Joonis 21. Kiired veebirakendused) on näha, kuidas mõlemad (joonisel nimega Nginx_HTML ja HAProxy_HTML) töökoormuse ühtlustatud serverid kasvavad sarnase tõusuga, aga HAProxyga on keskmiselt 300 päringut sekundis rohkem, kui Nginx töökoormuse ühtlustajaga. Tulemused kasvavad kuni seitsmenda serverini, kus on näha, et Nginx töökoormuse ühtlustajaga (ptk. 6.1.1) tekib kasvu kahanemine – toimub jõudluse piirang. Kuna veebiserver ja näidatav sisu (ptk. 3.1) – staatiline HTML on mõlemal töökoormuse ühtlustajal täpselt sama, saab järeldada, et jõudluse piirangu tingib Nginx töökoormuse ühtlustaja. Näha on, et HAProxy töökoormuse ühtlustajal (ptk. 6.2.1) tekib kasvu kahanemine üheksanda serveri juures, millest saab järeldada, et HAProxy töökoormuse ühtlustaja on võimeline kuvama rohkem päringute vastuseid, kui Nginx töökoormuse ühtlustaja. Selle rakenduse suur päringute arv sekundis näitab, et jõudluse piiravaks faktoriks on veebiserverite arv ning mitte protsessori ressursside kasutamine.

Kuna rakendus ei vaja ressursse veebilehe genereerimiseks, on kasulik lisada rohkem servereid töökoormuse ühtlustaja taha.

8.2 Nginx veebiserver koos PHP-ga

Joonisel (Joonis 21. Kiired veebirakendused) on näha kuidas mõlemad (joonisel nimega Nginx_PHP ja HAProxy_PHP) töökoormuse ühtlustatud serverid kasvavad sarnase tõusuga. Tulemused kasvavad kuni seitsmenda serverini, kus on näha, et Nginx töökoormuse ühtlustajaga (ptk. 6.1.2, 6.1.1) tekib kasvu kahanemine – toimub jõudluse piirang. Kuna veebiserver ja näidatav sisu (ptk. 3.2) – PHP keeles genereeritud tervitustekst, on mõlemal töökoormuse ühtlustajal täpselt sama, saab järeldada, et jõudluse piirangu tingib Nginx töökoormuse ühtlustaja. Näha on, et HAProxy töökoormuse ühtlustajal (ptk. 6.2.2) tekib kasvu kahanemine üheksanda serveri juures, millest saab järeldada, et HAProxy töökoormuse ühtlustaja on võimeline kuvama rohkem päringute vastuseid, kui Nginx töökoormuse ühtlustaja. Selle rakenduse suur päringute arv sekundis näitab, et jõudluse piiravaks faktoriks on veebiserverite arv ning mitte protsessori ressursside kasutamine. Kuna rakendus vajab vähe ressursse veebilehe genereerimiseks, on kasulik lisada rohkem servereid töökoormuse ühtlustaja taha.

8.3 Nginx Laravel koos MySQL andmebaasiga

Joonisel (Joonis 22. Aeglased veebirakendused) on näha kuidas (joonisel nimega Nginx_Laravel) töökoormuse ühtlustatud serverite päringute arv sekundites kasvab. Tulemused kasvavad kuni kuuenda serverini, kus on näha, et Nginx töökoormuse ühtlustajaga (ptk. 6.1.3) tekib kasvu kahanemine – toimub jõudluse piirang. Kuna veebiserver ja näidatav sisu (ptk. 3.3, 3.2) raamistiku Laravel ja MySQL andmebaasist andmetega on protsessorile väga koormav (ptk. 6), saab järeldada, et jõudluse piirangu ja päringute arvu sekundis vähesus võrreldes teiste rakendustega (Tabel 12. Kõikide veebirakenduste kiirus), tingib suur protsessori kasutus, mis näitab, et päringutele ei jõua piisavalt kiirelt vastust genereerida (ptk. 6). Antud rakenduse vähese päringute arvu sekundis ja suure protsessori ressursside kasutamine näitab, et veebiserveri võimsust on vaja tõsta ning sama jõudlusega veebiserverite lisamine ei anna suurt kasu.

8.4 Node.js veebiserver staatiline JSON

Joonisel (Joonis 21. Kiired veebirakendused) on näha kuidas mõlemad (joonisel nimega Nginx_Node.js ja HAProxy_Node.js) töökoormuse ühtlustatud serverid kasvavad sarnase tõusuga. Tulemused kasvavad kuni neljanda serverini kus on näha, et HAProxy töökoormuse ühtlustajaga (ptk. 6.2.3, 6.1.1) tekib kasvu kahanemine – toimub jõudluse piirang. Kuna veebiserver ja näidatav sisu (ptk. 3.4, 3.2) staatiline JSON – on mõlemal töökoormuse ühtlustajal täpselt sama, saab järeldada, et jõudluse piirangu tingib Nginx töökoormuse ühtlustaja. Näha on, et Nginx töökoormuse ühtlustajal (ptk. 6.1.4) tekib kasvu kahanemine kuuenda serveri juures, millest saab järeldada, et Nginx töökoormuse ühtlustaja on võimeline kuvama rohkem päringute vastuseid, kui HAProxy töökoormuse ühtlustaja. Antud rakenduste puhul toimub anomaalia. HAProxy päringute arv sekundis kahaneb enne Nginx töökoormuse ühtlustajat, mis on eelnevate tulemustega (ptk. 8.1, 8.2) vastuolus. Tegemist on eelkõige töökoormuse ühtlustajate võrdlusega, seetõttu on täpsem põhjuse uurimine antud töö eesmärgist väljas.

8.5 Node.js veebiserver koos MongoDB andmebaasiga

Joonisel (Joonis 22. Aeglased veebirakendused) on näha kuidas mõlemad (joonisel nimega Nginx_Node.js_MongoDB ja HAProxy_Node.js_MongoDB) töökoormuse ühtlustatud serverid kasvavad sarnase tõusuga. Tulemused kasvavad kuni kolmanda serverini, kus on näha, et Nginx töökoormuse ühtlustajaga (ptk. 6.1.5, 6.1.1) tekib kasvu kahanemine – toimub jõudluse piirang. Kuna veebiserver ja näidatav sisu (ptk. 3.5) – 50 viimast dokumenti MongoDB andmebaasist on mõlemal töökoormuse ühtlustajal täpselt sama, saab järeldada, et jõudluse piirangu tingib suur MongoDB vastuste päring. Kuna iga päring sisestab ühe ja küsib 50 dokumenti, on MongoDB-l väga suur koormus. Näha on, et HAProxy töökoormuse ühtlustajal (ptk. 6.2.4) tekib kasvu kahanemine viienda serveri juures, millest saab järeldada, et HAProxy töökoormuse ühtlustaja on võimeline kuvama rohkem päringute vastuseid, kui Nginx töökoormuse ühtlustaja. Selle rakenduse juures toimub andmebaasi jõudluse piirang ning rakenduse skaleerimise efektiivsuse tagaks andmebaasi võimsuse tõstmine.

8.6 Gunicorn veebiserver koos Django ja PostgreSQL andmebaasiga

Joonisel (Joonis 22. Aeglased veebirakendused) on näha kuidas mõlemad (joonisel nimega Nginx_Django ja HAProxy_Django) töökoormuse ühtlustatud serverid kasvavad sarnase tõusuga. Nginx (ptk. 6.1.6, 6.1.1) ega HAProxy (ptk. 6.2.5) töökoormuse ühtlustajaga ei toimu jõudluse piirangut. Antud tulemustest saab järeldada, et töökoormuse ühtlustajaga 10 serveriga piirangut ei teki. Selle rakenduse juures ei toimu jõudluse piirangut 10 serveriga, seetõttu saab järeldada, et antud rakendus töötab efektiivselt ja parim lahendus rakenduse skaleerimiseks, on veebiserverite lisamine töökoormuse ühtlustajate taha.

8.7 Järeldused

Veebiteenustel ja veebiserveritel, millel on suur arv külastajaid ja väike serverivõimsuse (ptk. 8.1, 8.2) nõudlus, on kasulikum kasutada töökoormuse ühtlustajat, laiendada horisontaalselt – rohkem veebiservereid töökoormuse ühtlustaja taga. Kui veebiteenuse või veebiserveri piiravaks teguriks jääb andmetalletamise kiirus, muutmälu või võrguühenduse läbilaskvus ja kiirus, siis on kindlasti parem kui liiklus on ühtlustatud, mis tagab kindluse, et teenus on kättesaadav. Selle alla kuuluvad staatilised lehed.

Kui tegemist on suure ja kõrget protsessori kiirust nõudva rakendusega (ptk. 8.3, 8.5), siis on kasulikum laiendada pigem vertikaalselt – võimsam server. Siiski kui rakendus on väga arvutuste nõudlik, võib tekkida piir kui ei tasu soetada võimsamat serverit, vaid rohkem nõrgemaid veebiserverid, mille töökoormus on ühtlustatud. Selle alla kuuluvad suuremad rakendused, mis kasutavad raamistikke (ptk. 3.3, 3.6), mitmeid andmebaasi päringuid (ptk. 3.5) ja vastava vastuse genereerimine võtab kauem aega kui staatilisel failil.

Nginx ja HAProxy võrdlusest tuli välja asjuolu, et tihti on HAProxy võimaline rohkem päringuid täitma ja rohkem serverite vahel töökoormust ühtlustama (Joonis 23. Kõik veebirakendused, Tabel 12. Kõikide veebirakenduste kiirus) kui Nginx. Seetõttu on soovitatav kasutada töökoormuse ühtlustamise eesmärgil HAProxy töökoormuse ühtlustajat.

9 Kokkuvõte

Käesolevas töös võrreldi erinevaid töökoormuse ühtlustajaid Nginx ja HAProxy ning veebiserverite jõudlust piiratud ressurssidega kasutades töökoormuse ühtlustamist Docker keskkonnas. Koguti testitulemused erinevatelt veebiserveritelt ning analüüsiti saadud tulemuste põhjal sobivaid lahendusi töökoormuse ühtlustamiseks.

Töö tulemusena loodi ja kasutati erinevaid modernseid veebis kasutatavaid keeli, rakendusi ja veebiservereid. Saadi valmis analüüs iga kasutatud veebiteenuse jõudluse kohta ning võrdlus Nginx ja HAProxy töökoormuse ühtlustamise võimekusest. Antud tulemuste põhjal on näha, et mõnedel piiratud ressurssidega veebirakendusetel on kasulik rakendada töökoormuse ühtlustamist, teistel mitte. Suurt protsessori jõudlust vajavad, suurte raamistike ja andmebaasidega veebirakendused, tuleks eelkõige seada võimsamale riistvarale ning vähest protsessori kasutust nõudvad rakendused, staatilised failid, on kasulikum seadistada kasutama töökoormuse ühtlustajat.

Töö järeldusena saab aimu ja leiti, milline oleks soovituslik lahendus veebirakenduse skaleerimiseks. Antud tulemused ja analüüs on kasulik veebiarendajatele, kes loovad veebirakendust, mis peab vastu pidama suurele veebiliiklusele, olenemata rakenduse tüübist. Samuti on võimalik arendajal enda veebiteenust testida ja leida sobivaim laiendamise viis, kasutades meetodikat sellest tööst.

10 Failide loetelu

Tegemist on failide loeteluga, mis on lisatud kaasa zip failis.

10.1 Nginx töökoormuse ühtlustaja – graafikud

/Excel/Nginx_Nginx_HTML.xlsx

/Excel/Nginx_Nginx_PHP.xlsx

/Excel/Nginx_Laravel.xlsx

/Excel/Nginx_Node.js.xlsx

/Excel/Nginx_Node.js_MongoDB.xlsx

/Excel/Nginx_Django.xlsx

10.2 HAProxy töökoormuse ühtlustaja – graafikud

/Excel/HAProxy_Nginx_HTML.xlsx

/Excel/HAProxy_Nginx_PHP.xlsx

/Excel/HAProxy_Node.js.xlsx

/Excel/HAProxy_Node.js_MongoDB.xlsx

/Excel/HAProxy_Django.xlsx

10.3 Tulemused – graafikud

/Excel/AB_tulemused_aeglased.xlsx

/Excel/AB_tulemused_kiired.xlsx

/Excel/AB_tulemused_koos.xlsx

10.4 Nginx veebiserver staatiline HTML

/Docker/nginxhtml/*

10.5 Nginx veebiserver koos PHP-ga

/Docker/php/*

10.6 Nginx Laravel koos MySQL andmebaasiga

/Docker/laravel/*

10.7 Node.js veebiserver staatiline JSON

/Docker/node/*

10.8 Node.js veebiserver koos MongoDB andmebaasiga

/Docker/node_mongo/*

10.9 Gunicorn veebiserver koos Django ja PostgreSQL andmebaasiga

/Docker/django/*

10.10 Nginx töökoormuse ühtlustaja

/Docker/lbnginx/*

10.11 HAProxy töökoormuse ühtlustaja

/Docker/lbha/*

10.12 Testkeskkond

/Docker/testenv/*

Kasutatud kirjandus

[1] Wikipedia, „Pooljuhtketas,“ [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Pooljuhtketas. [Kasutatud 20 mai 2017].

[2] I. L. Stats, „Internet Users,“ [Võrgumaterjal]. Available: http://www.internetlivestats.com/internet-users/. [Kasutatud 20 mai 2017].

[3] Statista, „Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions),“ Statista GmbH, [Võrgumaterjal]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/. [Kasutatud 20 mai 2017].

[4] I. L. Stats, „Total number of Websites,“ [Võrgumaterjal]. Available: http://www.internetlivestats.com/total-number-of-websites/. [Kasutatud 20 mai 2017].

[5] F. Patrick Shuff, „Building a Billion User Load Balancer,“ [Võrgumaterjal]. Available: https://www.usenix.org/conference/lisa16/conference-program/presentation/shuff. [Kasutatud 20 mai 2017].

[6] A. W. S. Jinesh Varia, „The Total Cost of (Non) Ownership of,“ [Võrgumaterjal]. Available: https://media.amazonwebservices.com/AWS_TCO_Web_Applications.pdf. [Kasutatud 20 mai 2017].

[7] D. C. services, „Dealing with website traffic spikes – stop your website crashing,“ [Võrgumaterjal]. Available: https://dmjcomputerservices.com/blog/prepare-for-website-traffic-spikes/. [Kasutatud 20 mai 2017].

[8] Wikipedia, „Horizontal and vertical scaling,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Scalability#Horizontal_and_vertical_scaling. [Kasutatud 20 mai 2017].

[9] I. Docker, „What is a Container,“ Docker, Inc, [Võrgumaterjal]. Available: https://www.docker.com/what-container. [Kasutatud 20 mai 2017].

[10] Ubuntu, „Ubuntu,“ Canonical Ltd, [Võrgumaterjal]. Available: https://www.ubuntu.com/. [Kasutatud 20 mai 2017].

[11] I. Docker, „What is Docker,“ Docker, Inc, [Võrgumaterjal]. Available: https://www.docker.com/what-docker. [Kasutatud 20 mai 2017].

[12] Intel, „Intel® Core™ i7-4700HQ Processor,“ [Võrgumaterjal]. Available: https://ark.intel.com/products/75116/Intel-Core-i7-4700HQ-Processor-6M-Cache-up-to-3_40-GHz. [Kasutatud 20 mai 2017].

[13] Docker, „Limit a container's resources,“ [Võrgumaterjal]. Available: https://docs.docker.com/engine/admin/resource_constraints/#cpu. [Kasutatud 20 mai 2017].

[14] A. S. Foundation, „ab - Apache HTTP server benchmarking tool,“ [Võrgumaterjal]. Available: https://httpd.apache.org/docs/2.4/programs/ab.html. [Kasutatud 20 mai 2017].

[15] A. S. Foundation, „Apache JMeter™,“ [Võrgumaterjal]. Available: http://jmeter.apache.org/. [Kasutatud 20 mai 2017].

[16] Nginx, „Using nginx as HTTP load balancer,“ [Võrgumaterjal]. Available: http://nginx.org/en/docs/http/load_balancing.html. [Kasutatud 20 mai 2017].

[17] HAProxy, „HAProxy,“ [Võrgumaterjal]. Available: http://www.haproxy.org/. [Kasutatud 20 mai 2017].

[18] Docker, „Docker container networking,“ [Võrgumaterjal]. Available: https://docs.docker.com/engine/userguide/networking/. [Kasutatud 20 mai 2017].

[19] T. D. E. -. c. book, „CPU CONTROL,“ [Võrgumaterjal]. Available: https://washraf.gitbooks.io/the-docker-ecosystem/content/Chapter%204/Section%204/cpu_control.html. [Kasutatud 20 mai 2017].

[20] T. G. Project, „System Monitor,“ [Võrgumaterjal]. Available: https://wiki.gnome.org/Apps/SystemMonitor. [Kasutatud 20 mai 2017].

[21] T. P. Group, „PHP is a popular general-purpose scripting language,“ [Võrgumaterjal]. Available: https://secure.php.net/. [Kasutatud 20 mai 2017].

[22] T. OTWELL, „The PHP Framework For Web Artisans,“ [Võrgumaterjal]. Available: https://laravel.com/. [Kasutatud 20 mai 2017].

[23] O. Corporation, „MySQL,“ [Võrgumaterjal]. Available: https://www.mysql.com/. [Kasutatud 20 mai 2017].

[24] N. Foundation, „Node.js,“ [Võrgumaterjal]. Available: https://nodejs.org/en/. [Kasutatud 20 mai 2017].

[25] I. MongoDB, „MongoDB for GIANT Ideas,“ [Võrgumaterjal]. Available: https://www.mongodb.com/. [Kasutatud 20 mai 2017].

[26] P. S. Foundation, „Python,“ [Võrgumaterjal]. Available: https://www.python.org/. [Kasutatud 20 mai 2017].

[27] D. S. Foundation, „Django: The Web framework for perfectionists with deadlines,“ [Võrgumaterjal]. Available: https://www.djangoproject.com/. [Kasutatud 20 mai 2017].

[28] T. P. G. D. Group, „PostgreSQL,“ [Võrgumaterjal]. Available: https://www.postgresql.org/. [Kasutatud 20 mai 2017].

[29] https://github.com/benoitc/gunicorn, „Gunicorn 'Green Unicorn' is a Python WSGI HTTP Server for UNIX,“ [Võrgumaterjal]. Available: http://gunicorn.org/. [Kasutatud 20 mai 2017].

[30] T. P. Group, „FastCGI Process Manager (FPM),“ [Võrgumaterjal]. Available: http://php.net/manual/en/install.fpm.php . [Kasutatud 20 mai 2017].

[31] Q.-S. W3Techs, „Usage statistics and market share of PHP for websites,“ [Võrgumaterjal]. Available: https://w3techs.com/technologies/details/pl-php/all/all. [Kasutatud 20 mai 2017].

[32] A. Monus, „10 PHP Frameworks For Developers – Best of,“ [Võrgumaterjal]. Available: http://www.hongkiat.com/blog/best-php-frameworks/. [Kasutatud 20 mai 2017].

[33] R. fryan, „Language Framework Popularity: A Look at PHP,“ [Võrgumaterjal]. Available: http://redmonk.com/fryan/2016/11/01/language-framework-popularity-a-look-at-php/. [Kasutatud 20 mai 2017].

[34] noeticforce, „PHP Frameworks: The Best 10 for Modern Web Development,“ [Võrgumaterjal]. Available: http://noeticforce.com/best-php-frameworks-for-modern-web-development. [Kasutatud 20 mai 2017].

[35] R. Labs, „Redis database,“ [Võrgumaterjal]. Available: https://redis.io/. [Kasutatud 20 mai 2017].

[36] K. Ferguson, „Laravel Development with Docker,“ [Võrgumaterjal]. Available: https://github.com/kyleferguson/laravel-with-docker-example. [Kasutatud 20 mai 2017].

[37] L. Aguilar, „Why is Node.js so Popular?,“ [Võrgumaterjal]. Available: https://dzone.com/articles/why-nodejs-so-popular. [Kasutatud 20 mai 2017].

[38] I. npm, „npm is the package manager for JavaScript,“ [Võrgumaterjal]. Available: https://www.npmjs.com/. [Kasutatud 20 mai 2017].

[39] https://www.npmjs.com/package/express/access, „Fast, unopinionated, minimalist web framework for node.,“ [Võrgumaterjal]. Available: https://www.npmjs.com/package/express. [Kasutatud 20 mai 2017].

[40] S. Daityari, „Python on the Web,“ [Võrgumaterjal]. Available: https://www.sitepoint.com/python-on-the-web/. [Kasutatud 20 mai 2017].

[41] R. Python, „Django Development With Docker Compose and Machine,“ [Võrgumaterjal]. Available: https://realpython.com/blog/python/django-development-with-docker-compose-and-machine/. [Kasutatud 20 mai 2017].

[42] Redislabs, „INCR key,“ [Võrgumaterjal]. Available: https://redis.io/commands/incr. [Kasutatud 20 mai 2017].

[43] M. Rouse, „single point of failure (SPOF),“ [Võrgumaterjal]. Available: http://searchdatacenter.techtarget.com/definition/Single-point-of-failure-SPOF. [Kasutatud 20 mai 2017].

[44] Nginx, „Load balancing,“ [Võrgumaterjal]. Available: https://www.nginx.com/resources/glossary/load-balancing/. [Kasutatud 20 mai 2017].

[45] R. Natário, „Load Balancing,“ [Võrgumaterjal]. Available: http://networksandservers.blogspot.com.ee/2011/03/balancing-iii.html . [Kasutatud 20 mai 2017].

[46] Nginx, „WHAT IS LAYER 7 LOAD BALANCING?,“ [Võrgumaterjal]. Available: https://www.nginx.com/resources/glossary/layer-7-load-balancing/. [Kasutatud 20 mai 2017].

[47] N. Ltd, „February 2016 Web Server Survey,“ [Võrgumaterjal]. Available: https://news.netcraft.com/archives/2016/02/22/february-2016-web-server-survey.html. [Kasutatud 20 mai 2017].

[48] Nginx, „NGINX LOAD BALANCING – HTTP LOAD BALANCER,“ [Võrgumaterjal]. Available: valikud. https://www.nginx.com/resources/admin-guide/load-balancer/. [Kasutatud 20 mai 2017].

[49] I. Datadog, „Monitoring HAProxy performance metrics,“ [Võrgumaterjal]. Available: https://www.datadoghq.com/blog/monitoring-haproxy-performance-metrics/. [Kasutatud 20 mai 2017].

[50] I. Docker, „Docker Compose,“ [Võrgumaterjal]. Available: https://docs.docker.com/compose/. [Kasutatud 20 mai 2017].

[51] Nginx, „Nginx,“ [Võrgumaterjal]. Available: https://nginx.org/en/. [Kasutatud 20 mai 2017].

Lisa 1 – Näidis Dockerfile fail – testkeskkond

```
FROM debian:jessie

RUN apt-get update && \
  apt-get upgrade -y && \
  apt-get install -y net-tools && \
  apt-get install -y wget && \
  apt-get install -y apache2-utils && \
  apt-get install -y openjdk-7-jre-headless && \
  wget -c https://archive.apache.org/dist/jmeter/binaries/apache-jmeter-
3.0.tgz && \
  tar -xf apache-jmeter-3.0.tgz && \
  apt-get clean -y && \
  apt-get autoclean -y && \
  apt-get autoremove -y && \
  rm -rf /usr/share/locale/* && \
  rm -rf /var/cache/debconf/*-old && \
  rm -rf /var/lib/apt/lists/* && \
  rm -rf /usr/share/doc/*docker
```

Lisa 2 – Näidis docker-compose.yml fail Laravel

```
version: '2'
services:
    web:
        build:
            context: ./
            dockerfile: deploy/web.docker
        volumes:
            - ./nginx_laravel.conf:/etc/nginx/nginx.conf
            - ./:/var/www
        ports:
            - "8090:80"
        links:
            - app
    app:
        build:
            context: ./
            dockerfile: deploy/app.docker
        volumes:
            - ./php/php.ini:/usr/local/etc/php/conf.d/php.ini
            - ./:/var/www
        links:
            - database
            - cache
        environment:
            - "DB_PORT=3306"
            - "DB_HOST=database"
            - "REDIS_PORT=6379"
            - "REDIS_HOST=cache"
        cpu_quota: 50000
    database:
        image: mysql:5.6
        environment:
            - "MYSQL_ROOT_PASSWORD=secret"
            - "MYSQL_DATABASE=dockerApp"
    cache:
        image: redis:3.0
networks:
    default:
        external:
            name: bridgenet
```

Lisa 3 – Näidis Apace Bench testitulemus

Server Software: nginx/1.11.12
Server Hostname: 172.19.0.13
Server Port: 80

Document Path: /
Document Length: 9 bytes

Concurrency Level: 100
Time taken for tests: 1.471 seconds
Complete requests: 1000
Failed requests: 0
Total transferred: 239000 bytes
HTML transferred: 9000 bytes
Requests per second: **679.58** [#/sec] (mean)
Time per request: 147.149 [ms] (mean)
Time per request: 1.471 [ms] (mean, across all concurrent requests)
Transfer rate: 158.61 [Kbytes/sec] received

Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	2 5.0	0	20
Processing:	10	136 43.2	137	237
Waiting:	8	135 43.3	136	237
Total:	25	138 41.4	137	237

Percentage of the requests served within a certain time (ms)
50% 137
66% 152
75% 166
80% 179
90% 195
95% 207
98% 219
99% 227
100% 237 (longest request)

Lisa 4 – Näidis JSON vastus


```
{
    "url": "http://example.com/forum",
    "thread": {
        "id": 528512,
        "createdBy": "username1",
        "title": "This is a made up Title",
        "postsCount": 2,
        "posts": [
            {
                "content": "first post",
                "id": "1",
                "postedBy": "username1"
            },
            {
                "content": "second post",
                "id": "2",
                "postedBy": "username2"
            }
        ]
    }
}
```

Lisa 5 – Nginx töökoormuse ühtlustaja seaded

```
events { }
http {
    upstream lb {
        server 172.19.0.3:80;
        server 172.19.0.4:80;
        server 172.19.0.5:80;
        server 172.19.0.6:80;
        server 172.19.0.7:80;
    }
    server {
        listen 80;

        location / {
            proxy_pass http://lb;
        }
    }
    include /etc/nginx/conf.d/*.conf;
}
```

Lisa 6 – HAProxy töökoormuse ühtlustaja seaded (ebatähtis eemaldatud)

```
global
  log 127.0.0.1 local0
  log 127.0.0.1 local1 notice
  pidfile /etc/haproxy/haproxy.pid
  daemon

defaults
  log global
  mode http
  option httplog
  option dontlognull

  frontend http-frontend
    bind *:5000
    mode http
    default_backend http-backend

  backend http-backend
    mode http
    balance roundrobin
    option forwardfor

    server http-server-1 172.19.0.3:80  check
    server http-server-3 172.19.0.4:80  check
    server http-server-4 172.19.0.5:80  check
    server http-server-5 172.19.0.6:80  check
    server http-server-2 172.19.0.7:80  check

listen default
  bind *:4242
```