

TARTU ÜLIKOOL

SOTSIAALTEADUSTE VALDKOND

NARVA KOLLEDŽ

ÕPPEKAVA „INFOTEHNOLOOGILISTE SÜSTEEMIDE ARENDUS“

Aleksei Votintsev

**TARTU ÜLIKOOLI NARVA KOLLEDŽI VEEBISERVERI
TÖÖ OPTIMEERIMINE KOORMUSTESTIDE ABIL**

Bakalaureuse töö

Juhendaja: Pavel Kodotšigov, MSc

NARVA 2025

Olen koostanud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, põhimõttelised seisukohad, kasutatud kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

Aleksei Votintsev

18.05.2025

Mina, Aleksei Votintsev,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose: Tartu Ülikooli Narva Kolledži veebiserveri töö optimeerimine koormustestide abil, mille juhendaja on Pavel Kodotšigov, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commons'i litsentsiga CC BY NC ND 4.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Aleksei Votintsev

18.05.2025

ABSTRACT

Optimization of the Web Server of the University of Tartu Narva College Using Load Testing

This thesis focuses on analyzing and optimizing the performance of the web server of the University of Tartu Narva College. The core objective is to improve the efficiency and reliability of the server under load, and to evaluate methods of protection against potentially harmful traffic.

The study was conducted using a test environment with three virtual machines: a replica of the actual server with real data, a clean system with identical components but without data, and a separate machine for initiating load tests. The primary tool for testing was *k6*, which enabled the simulation of HTTP requests under increasing and burst-style loads.

The results showed that the default configuration of the college's web server (based on Nginx) was not fully utilizing the available system resources. After applying server-side optimizations to PHP-FPM, Nginx, and the MySQL database, a significant improvement in latency and resource efficiency was observed. An alternative solution using Apache was also tested, demonstrating lower CPU usage but slightly reduced throughput under load.

Protective measures, including rate limiting and automated IP banning (e.g. with Fail2Ban), were evaluated and integrated to strengthen the server's resilience. As a result, an optimized server configuration has been proposed, suitable for implementation in the college's production environment.

SISUKORD

SISUKORD	5
SISSEJUHATUS	7
1. TEOREETILINE ALUS VEEBISERVERI JÕUDLUSE HINDAMISEKS JA KAITSEKS	9
1.1 Veebiserverite arhitektuur ja jõudlus	9
1.2 Koormustestimine veebiserveri jõudluse hindamiseks.....	10
1.2.1 Koormustestimise roll veebiserveri optimeerimisel	10
1.2.2 Koormustestimise liigid.....	10
1.2.3 Koormustestimise mõõdikud	11
1.2.4 Koormustestimise tööriistade roll.....	12
1.3 Kaitsemeetmed ja turvalisus veebiserveri töökindluse tagamisel	12
2. KATSEKESKKOND JA TESTIMETOODIKA.....	14
2.1 Testkeskkonna kirjeldus	14
2.1.1 Testkeskkonna füüsiline infrastruktuur.....	14
2.1.2 Virtuaalsete serverite ülesehitus	14
2.1.3 Virtuaalsete masinate võrguarhitektuur ja suhtlus	16
2.1.4 Virtuaalkeskkonna piirangud ja märkused	17
2.2 Testide stsenaariumid ja meetodika.....	18
2.2.1 Testide eesmärk ja ülesehitus.....	18
2.2.2 Kasutatud tööriistad	19
2.2.3 Teststsenaariumid	19
2.2.4 Kasutatud meetodid ja mõõdikud	22
3. TESTITULEMUSED JA ANALÜÜS	23
3.1 Testitulemused algses konfiguratsioonis	23
3.1.1 Kasvava koormusega teststsenaarium	23
3.1.2 DDoS-laadne test.....	24
3.2 Probleemide analüüs ja kitsaskohad	25
3.2.1 CPU ülekoormus ja ressursside jaotus	25
3.2.2 PHP-FPM protsesside piiratus.....	25
3.2.3 Andmebaasi jõudluse kitsaskoht.....	26

3.2.4 Ühenduste ja päringukanalite haldus	26
3.2.5 Kokkuvõtte probleemidest ja kitsaskohtadest	26
3.3 Optimeerimismeetmete valik ja põhjendus	27
3.3.1 PHP-FPM optimeerimine	27
3.3.2 Nginx optimeerimine	28
3.3.3 MySQL andmebaasi optimeerimine	29
3.3.4 Turvameetmed ja DDoS-vastane kaitse.....	29
3.4 Tulemused pärast optimeerimist.....	30
3.4.1 Kasvava koormusega teststsenaarium: optimeeritud Nginx ja Apache.....	31
3.4.2. DDos-laadne test ja süsteemi reaktsioon	32
3.5 Tulemuste võrdlemine enne ja pärast optimeerimist	33
3.5.1 Päringute kiirus.....	33
3.5.2 Ressursside kasutus	35
3.5.3 Kaitsemeetmete mõju testides	36
3.5.4 Tulemuste koondvõrdlus	37
KOKKUVÕTE	39
KASUTATUD KIRJANDUS	40

SISSEJUHATUS

Tänapäeva digitaalses maailmas on veebiserverid keskseks komponendiks igasuguste avalike ja sisemiste infosüsteemide, rakenduste ja teenuste toimimisel. Nende roll on eriti oluline haridusasutustes, kus veebipõhised teenused moodustavad suure osa nii tudengite kui ka personali igapäevasest suhtlusest institutsiooniga. Tartu Ülikooli Narva kolledži veebiserver on oluline lüli asutuse akadeemilises ja haldustegevuses, pakkudes ligipääsu mitmetele veebilehtedele ja rakendustele, mille kaudu korraldatakse erinevaid protsesse, sealhulgas õppeprotsessi toetavaid tegevusi, ressursside haldamist ja broneerimissüsteeme.

Veebiserverite töövõime, reageerimiskiirus ja töökindlus on otseselt seotud lõppkasutajate rahulolu ja teenuste kvaliteediga. Kasutajate kasvav arv, intensiivne ligipääs veebirakendustele ning ootamatud või planeeritud koormusetipud võivad serveri tööd märkimisväärselt mõjutada, põhjustades aeglust, katkestusi või täielikku teenuse kättesaamatust. Sellised probleemid ei pruugi olla tingitud ainult suurest kasutajate hulgast, vaid sageli ka serveri ebapiisavast konfiguratsioonist, ebaoptimaalsest ressursihaldusest ning puudulikest kaitsemehhanismidest, mis muudab serverid haavatavaks nii planeeritud kui ka pahatahtlike ülekoormusrünnakute suhtes.

Lisaks kasvavatele kasutajanõudmistele peab server suutma toime tulla ka turvaohutudega, sealhulgas ründevektoritega, mis võivad sihtida autentimisprotsesse, andmebaasipäringuid või muid serveri ressursse. Koormusrünnakud (DoS, DDoS) on muutunud üha sagedasemaks ohuks, mille tulemusel võivad isegi ajutised rünnakud häirida kogu asutuse toimimist. Seetõttu veebiserverite töö optimeerimine ja kaitsemeetmete rakendamine on saanud lahutamatuks osaks tänapäevastes heas haldustavas ja süsteemiadministreerimises.

Probleem

Tartu Ülikooli Narva kolledži veebiserveri puhul seisneb probleem selles, et suuremahuliste või intensiivsete päringute tingimustes võib server töötada ebaefektiivselt, põhjustades teenuse kättesaadavuse vähenemist või jõudluse langust. Lisaks võib olemasolev konfiguratsioon olla ebapiisav, et kaitsta serverit sihitud rünnakute ja pahatahtlike päringute eest, mis võivad tekitada liigset koormust ja ressursikasutust.

Eesmärk

Käesoleva lõputöö eesmärgiks on Tartu Ülikooli Narva kolledži veebiserveri töö optimeerimine, et parandada selle jõudlust suure koormuse tingimustes ning suurendada süsteemi töökindlust ja turvalisust rakendades sobivaid kaitsemeetmeid. Töö käigus analüüsitakse serveri praegust töövõimet ja ressursikasutust ning hinnatakse võimalusi efektiivsuse ja vastupidavuse parandamiseks koormuse tingimustes.

1. TEOREETILINE ALUS VEEBISERVERI JÕUDLUSE HINDAMISEKS JA KAITSEKS

1.1 Veebiserverite arhitektuur ja jõudlus

Veebiserverite jõudlus sõltub mitmest tegurist, sealhulgas serveri arhitektuurist, kasutatavast töötlusmudelist, rakenduse taseme komponentidest ning süsteemi üldisest konfiguratsioonist. Rääkides serveri arhitektuurist ja päringute töötlemismudelist, on kõige levinumateks lahendusteks tänapäeval Apache HTTP Server ja Nginx, mis erinevad teineteisest nii sisemise ülesehituse kui ka tööloogika poolest.

Apache HTTP Server kasutab protsessipõhist ja lõimepõhist töötlusmudelit, mis võimaldab mitme ühenduse paralleelset töötlemist eraldiseisvate protsesside või lõimede kaudu. Selline lähenemine võimaldab paindlikku teenindamist erinevate rakenduste puhul, kuid samas tekitab piiranguid, kuna iga uus ühendus suurendab protsesside arvu ja vastavalt ka ressursitarbimist (Laurie & Laurie, 2022). See võib koormuse suurenedes viia süsteemi jõudluse languseni.

Nginx seevastu kasutab sündmustepõhist ja asünkroonset töötlemismudelit, mis võimaldab üheainsa protsessi või väikese arvu protsesside kaudu hallata tuhandeid ühendusi korraga. Tänu mitteblokeerivale sisend-/väljundtehnoloogiale ja sündmuspõhisele arhitektuurile on Nginx eriti tõhus suurte koormuste ja staatiliste ressursside teenindamisel, pakkudes madalat latentsust ja väikest ressursitarbimist (Sysoev, 2015).

Rakenduse taseme komponentide tase on üks olulisi aspekte veebiserveri jõudluse kujundamisel ning koosneb järgmistest põhielementidest:

- PHP skriptide töötlemine php-fpm kaudu, kus PHP protsessid töötlevad veebiserverilt saadud päringud ja loovad vastuse. PHP-fpm võimaldab protsesside eraldamist veebiserverist, kuid nõuab korrektset optimeerimist, et vältida ressursipuudust või protsesside ummistumist (PHP, 2024).
- Andmebaasisüsteemid, näiteks MySQL, mille jõudlus sõltub päringute keerukusest, andmete hulgast, andmebaasi indeksite kasutamisest ning optimeeritusest (Owens, 2020).
- Võrgu tingimused, sh latentsus ja ribalaius, mis võivad mõjutada andmete edastamise kiirust.

Efektiivne veebiserveri toimimine ei sõltu üksnes selle arhitektuurist, vaid ka sellest, kuidas rakenduse taseme komponendid on seadistatud ja omavahel tasakaalus. Nõrgalt optimeeritud andmebaasid, halvasti hallatud protsessid või ebasoodsad võrgutingimused võivad oluliselt piirata ka tehniliselt võimeka serveri jõudlust (Botros & James, 2021).

Samuti tuleb arvesse võtta süsteemi üldist infrastruktuuri, sealhulgas seda, kas veebiserver töötab füüsilisel või virtuaalsel keskkonnal. Virtuaalkeskkondades võivad ilmnedä täiendavad piirangud, näiteks ressursside jagamine teiste masinatega, mis võib mõjutada serveri jõudlust koormuse all (Smith & Kim, 2019).

Veebiserveri töökindluse ja jõudluse tagamiseks on oluline terviklik ja süsteemne lähenemine, mis hõlmab serveri ja rakenduste konfiguratsiooni optimeerimist, ressursikasutuse monitooringut ning jõudluskatsetuste läbiviimist erinevates koormuse stsenaariumides.

1.2 Koormustestimine veebiserveri jõudluse hindamiseks

1.2.1 Koormustestimise roll veebiserveri optimeerimisel

Koormustestimine on veebiserveri optimeerimise protsessis oluline, kuna see võimaldab hinnata süsteemi jõudlust erinevates koormustingimustes ja teha kindlaks, kas serveri konfiguratsioon, arhitektuur või rakendus vajavad optimeerimist. Lisaks võimaldab koormustestimine kontrollida, milline mõju on erinevatel kaitsemehhanismidel ja turvalahendustel süsteemi töövõimele (PerfTestPlus Team, 2021).

Korrektelt läbi viidud koormustestid aitavad:

- tuvastada kitsaskohad serveri, rakenduse või infrastruktuuri tasandil;
- hinnata konfiguratsioonimuudatuste mõju jõudlusele;
- planeerida infrastruktuuri skaleerimist ja süsteemi ressursiplaane;
- tõsta süsteemi töökindlust ja vähendada riske koormustippudes või rünnakute ajal.

Koormustestimine on seega süsteemi optimeerimise, töökindluse tõstmise ja turvalisuse tagamise oluline osa, mille regulaarne läbiviimine aitab organisatsioonidel pakkuda stabiilset ja kiiret veebiteenust ka keerulistes töötingimustes (PerfTestPlus Team, 2021).

1.2.2 Koormustestimise liigid

Koormustestimine jaguneb vastavalt eesmärgile, koormusmuustrile ja testitavale olukorrale mitmeks peamiseks liigiks:

- Tavaline koormustest – süsteemi töövõime hindamine tavapärase ja järk-järgult kasvava koormuse tingimustes. Testi eesmärk on määrata süsteemi suutlikkus töödelda eeldatavat maksimaalset päringute või kasutajate arvu, jälgides süsteemi vastamiskiirust ja stabiilsust.
- Stressitest – süsteemi töövõime hindamine äärmusliku ja ülekoormuse tingimustes, et tuvastada süsteemi piire, tõrgete ilminguid ja taastumisvõimet olukordades, kus ressursid on täielikult kasutatud või server saavutab oma kriitilise piiri.
- Järsk koormustest – süsteemi reageerimise ja stabiilsuse hindamine olukordades, kus koormus kasvab ootamatult ja järsult, näiteks massilise kasutajate sissevoolu või pahatahtliku rünnaku ajal.
- Pikaajaline koormustest – süsteemi töökindluse hindamine pikaajalise pideva või muutuva koormuse tingimustes, et avastada võimalikke mälu lekke, ressursikasutuse akumulereerumist või jõudluse halvenemist aja jooksul.
- Konkurentsitest – keskendub süsteemi suutlikkusele hallata suurt arvu samaaegseid kasutajaid, sessioone või ühendusi, tuues esile võimalikud probleemid seoses ühenduste halduse ja samaaegse töötlemisega.
- Erakorraline koormustest – süsteemi käitumise ja taastumisvõime hindamine ootamatute rikete, teenuse katkestuste või infrastruktuuriliste vigade korral, simuleerides süsteemisisesi tõrkeid, katkestatud võrguühendusi või serveri restarti.

Mitmekesiste testide kasutamine annab süsteemiadministraatoritele ja arendajatele terviklikku ülevaate süsteemi töövõimest ning aitab ennetada kriitilisi probleeme, mis võivad ilmneda reaalsetes töötingimustes.

1.2.3 Koormustestimise mõõdikud

Koormustestimise tulemusel saadakse mitmeid võtmemõõdikuid, mis võimaldavad hinnata süsteemi käitumist ja jõudlust:

- Latentsus – aeg, mis kulub serveril päringule vastuse andmiseks.
- Läbilaskevõime – päringute arv, mida server suudab töödelda kindla ajaühiku jooksul.
- Veamäär – päringute protsent, mis ei õnnestu või mille puhul server annab vea.
- Ressursikasutus – serveri CPU (protsessor), RAM (muutmälu), kettakasutuse ja võrguliikluse monitooring koormuse tingimustes.

1.2.4 Koormustestimise tööriistade roll

Koormustestimiseks kasutatakse mitmeid erinevaid tööriistu ja platvorme, mille ühine eesmärk on võimaldada süsteemihalduritel ja arendajatel luua erinevaid koormustsenaariume, jälgida süsteemi käitumist ja analüüsida tulemusi. Tööriistad võimaldavad testida nii lihtsaid HTTP-päringuid kui ka keerukamaid kasutajastsenaariume, sealhulgas autentimist, andmeedastust ja andmebaasipäringuid. Tööriistade valik sõltub testitava süsteemi iseloomust, nõutavatest mõõdikutest ja testide keerukusest.

1.3 Kaitsemeetmed ja turvalisus veebiserveri töökindluse tagamisel

Veebiserverite töökindluse ja kättesaadavuse tagamisel on lisaks jõudluse optimeerimisele oluline roll ka turvalisusel ja kaitsemehhanismidel. Veebiserverid on pidevalt avatud internetile, mistõttu puutuvad nad kokku erinevate turvaohutude ja pahatahtlike rünnetega. Üks levinumaid ohte veebiserverite puhul on koormusrünnakud (DoS – Denial of Service, DDoS – Distributed Denial of Service), mille eesmärgiks on serveri ülekoormamine ja selle töö häirimine (Kaspersky, 2023).

Lisaks koormusrünnetele võivad veebiserverid olla sihtmärgiks ka muudele rünnakutele, nagu sisendipõhised rünnakud, autentimisprotsesside väärkasutamine, andmebaasirünnakud, pahatahtlike skriptide laadimine ja muud turvariskid, mis võivad põhjustada süsteemi kompromiteerimise või andmelekked (OWASP, 2023).

Veebiserveri turvalisuse ja töökindluse tagamiseks kasutatakse erinevaid kaitsemeetmeid, mida võib jagada järgmisteks tasanditeks:

- Rakenduse tase – turvameetmed, mis on seotud veebirakenduse enda koodi ja funktsioonidega, näiteks sisendi valideerimine, autentimis- ja autorisatsioonimehhanismid, andmekaitse ja krüptimine.
- Serveri tase – meetmed, mis hõlmavad serveri operatsioonisüsteemi, veebiserveri tarkvara ja konfiguratsiooni turvalisust, sealhulgas õiguste haldamist, ühenduste piiramist, logimist ja jälgimist.
- Võrgutasand – kaitsemeetmed, mis töötavad võrgu tasemel, näiteks tulemüürid, ühenduste filtreerimine, liikluse piiramine ja koormusbalanseerimine.

Olulisel kohal on töökindluse ja kaitsemehhanismide tasakaalustamine, kuna ülemäärane agressiivsed piirangud ja filtrid võivad mõjutada negatiivselt ka seaduslikke kasutajaid, põhjustades päringute blokeerimist või teenuste aeglustumist. Samas liiga leebed reeglid

võivad muuta süsteemi haavatavaks ja jätta ukse avatuks pahatahtlike rünnete jaoks (Kaspersky, 2023).

Seetõttu on oluline rakendada kihelist kaitset, kus erinevad meetmed töötavad koos, pakkudes nii ennetavat kui ka reaktsioonilist kaitset. Samuti on soovitatav kasutada automatiseeritud monitooringu- ja logianalüüsi tööriistad, mis võimaldavad tuvastada ebatavalist liiklust või võimalikke ründeid reaalajas ning vajadusel võtta automaatselt vastumeetmeid.

Lisaks traditsioonilistele kaitsemehhanismidele on järjest olulisemaks muutunud koormusrünnete ennetus ja mõju leevendamine (DDoS mitigation), kasutades spetsialiseeritud lahendusi ja praktikaid, mis aitavad tagada süsteemi kättesaadavust ka massiivsete koormusrünnete korral.

Kokkuvõttes on veebiserverite kaitsemeetmete rakendamine vältimatu osa serverite haldusest, mille eesmärgiks on:

- kaitsta süsteemi pahatahtlike rünnete ja vigade eest;
- tagada serveri töökindlust ja kättesaadavust igas olukorras;
- vähendada turvariske ja minimeerida võimalikke intsidentide mõju.

Kõik teoreetilises osas käsitletud põhimõtted – serverite arhitektuurilised erinevused, jõudlust mõjutavad tegurid ja kaitsemeetmed – rakendatakse ja kontrollitakse empiirilisel Tartu Ülikooli Narva kolledži veebiserveri näitel järgmises peatükis.

2. KATSEKESKKOND JA TESTIMETOODIKA

Käesolevas töös hinnatakse Tartu Ülikooli Narva kolledži veebiserveri töövõimet ja optimeerimisvõimalusi. Kolledži praegune veebiserver kasutab Nginx tarkvara, mille efektiivsust ja optimeerimisvõimalusi uuritakse käesolevas töös empiiriliselt. Selleks luuakse spetsiaalne virtuaalkeskkond, kus Hyper-V virtualiseerimistehnoloogia abil seadistatakse kolm virtuaalmasinat erinevate konfiguratsioonide testimiseks. Uuringu keskseks rakenduseks on MRBS (Meeting Room Booking System), mis võimaldab kolledži töötajatel hallata auditooriumide ja õpperuumide broneeringuid. MRBS rakendus kasutab PHP skripte, mida töötleb veebiserver PHP-FPM protsessihalduri kaudu, ning kõik andmed salvestatakse ja hallatakse MySQL-andmebaasis.

2.1 Testkeskkonna kirjeldus

Käesolevas töös kirjeldatud katsed viidi läbi spetsiaalselt loodud virtuaalkeskkonnas. Selleks kasutati Tartu Ülikooli Narva kolledžis asuvat füüsilist serverit, millel Hyper-V virtualiseerimistehnoloogia abil loodi kolm virtuaalmasinat. Virtuaalmasinad võimaldasid isoleeritult ja kontrollitud tingimustes testida veebiserverite erinevaid konfiguratsioone ning võrrelda tulemusi. Virtuaalkeskkonna kasutamine aitas tagada katsete korratavuse ning välistas väliste tegurite mõju testide tulemustele.

2.1.1 Testkeskkonna füüsiline infrastruktuur

Testkeskkonna virtuaalmasinad töötasid Tartu Ülikooli Narva kolledži serveriarvutil, millel olid järgmised spetsifikatsioonid:

- Protsessor: Intel® Xeon® Silver 4416+ (20 tuuma, 40 lõime)
- Operatiivmälu: 256 GB DDR5
- Kõvaketas: 850 GB NVMe SSD (NTFS)
- Operatsioonisüsteem: Windows Server 2022 Datacenter Edition (64-bit)

Selline konfiguratsioon võimaldas samal ajal hallata mitu virtuaalmasinat ning läbi viia koormusteste piiranguteta ressursikasutuse osas.

2.1.2 Virtuaalsete serverite ülesehitus

Kõik testkeskkonna virtuaalmasinad töötasid Hyper-V virtualiseerimiskeskkonnas. Katsete võrreldavuse tagamiseks määrati kõikidele virtuaalmasinatele identsed ressursid.

Apache HTTP Server valiti alternatiivseks lahenduseks võrdleva analüüsi eesmärgil, et

hinnata Nginx veebiserveri kasutegurit kolledži praeguse konfiguratsiooniga võrreldes traditsioonilise protsessipõhise serveriga. Apache veebiserveri valik oli tingitud selle laialdasest kasutusest ja põhimõttelistest arhitektuurilistest erinevustest Nginxiga, mis võimaldas täpsemalt mõista erinevate töötlusmodelite mõju serveri jõudlusele. Nii Nginx kui ka Apache testid viidi läbi samal virtuaalmasinal, kuid erinevatel ajahetkedel — testimise ajal oli aktiivne ainult üks veebiserver korraga. Virtuaalkeskonna struktuuri ning masinatevahelist seost on visuaalselt kujutatud (joonis 1).

Virtuaalne testserver (puhtalt seadistatud keskkond)

- Operatsioonisüsteem: Ubuntu Server 22.04 LTS
- vCPU (virtuaalset protsessorituumu): 8
- RAM: 16 GB
- Kettaruum: 100 GB
- Paigaldatud tarkvara: Nginx, php-fpm, MySQL (tühi andmebaas), MRBS rakendus

Virtuaalne tootmisserver (reaalse keskkonna koopia)

- Operatsioonisüsteem: Ubuntu Server 22.04 LTS
- vCPU: 8
- RAM: 16 GB
- Kettaruum: 100 GB
- Paigaldatud tarkvara: Nginx, Apache, php-fpm, MySQL (koopia kolledži tootmisandmebaasist), MRBS rakendus

Virtuaalne testimismasin (koormustestide läbiviimiseks)

- Operatsioonisüsteem: Windows Server 2022 Datacenter Edition
- vCPU: 8
- RAM: 16 GB
- Kettaruum: 100 GB
- Paigaldatud tarkvara: koormustestimise tarkvara (täpsustatakse metoodika peatükis)

Kuna kõik virtuaalmasinad töötasid samal füüsilisel serveril, jagasid nad omavahel protsessoriressursse, muutmälu ja kettapöördeid. Kuigi igale masinale oli eraldatud kindel hulk ressursse ning koormus jagati tasakaalustatult, tuleb arvestada, et selline ülesehitus võib tekitada konkurentsiolukordi, mis omakorda mõjutavad serveri reageerimiskiirust. Lisaks võib virtualiseerimisplatvormi (Hyper-V) kasutamine lisada väikest, kuid mõõdetavat latentsust võrreldes füüsiliste seadmetega. Samuti võib mitme virtuaalmasina samaaegne töö mõjutada ketta I/O (Input/Output) jõudlust.

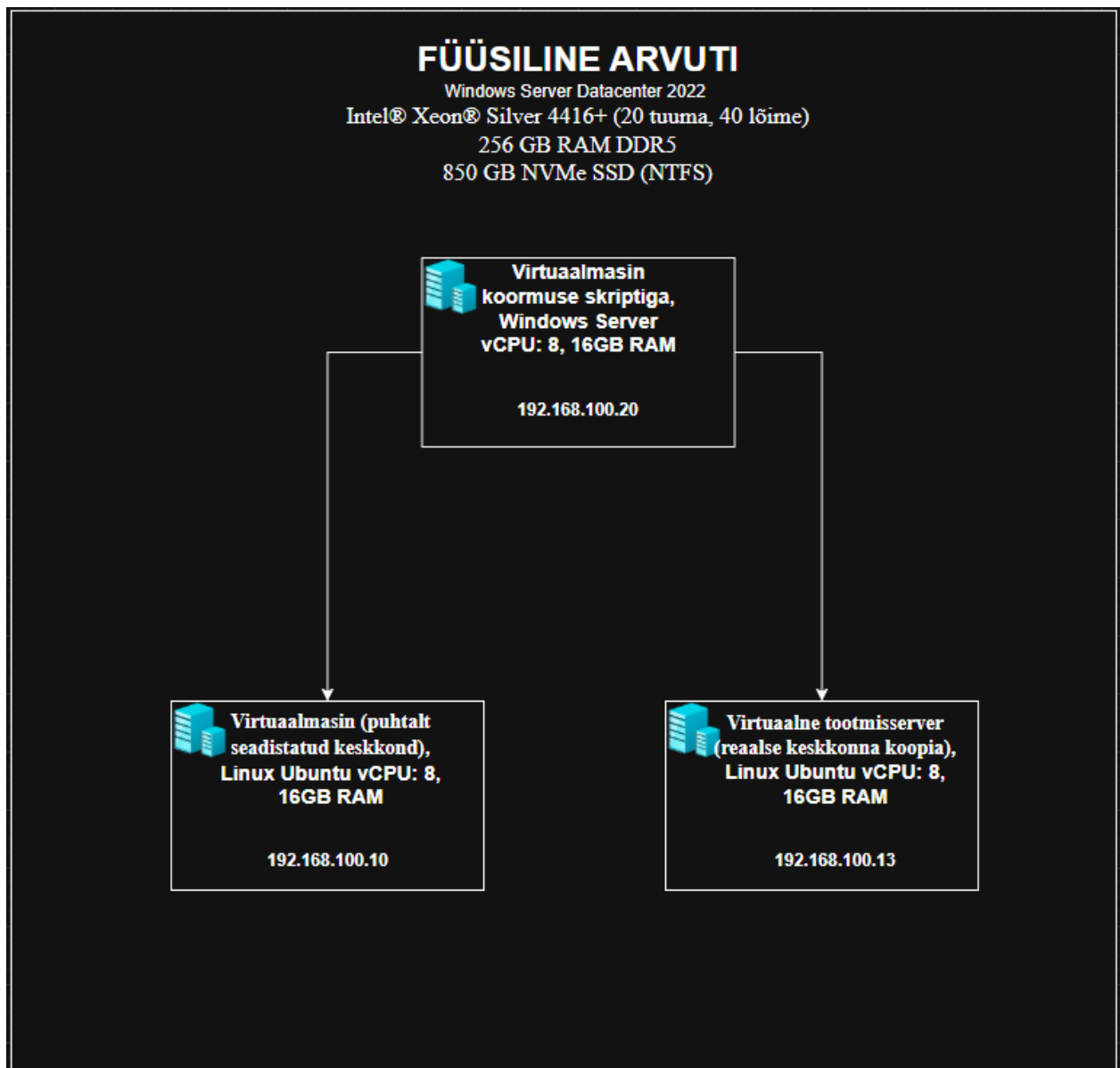
2.1.3 Virtuaalsete masinate võrguarhitektuur ja suhtlus

Et testimasinad saaksid omavahel suhelda ning koormustestide tulemused oleksid usaldusväärsed ja korratavad, seadistati kõik virtuaalmasinad samasse NAT-võrku. Selline võrgukonfiguratsioon võimaldas masinatel omavahel otse suhelda lokaalselt, ilma välise võrgu latentsuse või tulemüüride mõjuta. Iga masinale määrati kindel staatiline IP-aadress, et tagada üheselt määratletav suhtluskatsete käigus.

Virtuaalmasinate IP-aadressid ja rollid:

- 192.168.100.20 – Virtuaalne testimismasin (koormustestide läbiviimine)
 - Kasutati koormustestide ja skriptide käivitamiseks.
 - Saatis päringuid mõlema serveri poole (192.168.100.13 ja 192.168.100.10).
- 192.168.100.13 – Virtuaalne tootmisserver (reaalse keskkonna koopia)
 - Kasutati kolledži reaalsete konfiguratsioonidega veebiserverina.
 - Virtuaalmasinale oli seadistatud Nginx kolledži veebiserveri tegeliku konfiguratsiooni põhjal; samas keskkonnas viidi läbi ka Apache võrdlustestid.
 - Võttis vastu koormustestide päringuid testimismasinast.
- 192.168.100.10 – Virtuaalne testserver (puhtalt seadistatud keskkond)
 - Kasutati puhta keskkonnana, kus MySQL andmebaas oli tühi.
 - Võttis vastu koormustestide päringuid testimismasinast.

Kõik virtuaalmasinad olid konfigureeritud nii, et kogu liiklus toimus lokaalses võrgus, vältides internetiühendust. See võimaldas välistada võrgu latentsuse ja ribalaiuse mõjusid testi tulemustele ning keskenduda ainult serverite töövõime ja konfiguratsiooni hindamisele.



Joonis 1. Virtuaalmasinate ülesehitus ja testikeskkonna võrgustruktuur.

Allikas: autori koostatud.

2.1.4 Virtuaalkeskkonna piirangud ja märkused

Kuigi kõik testid viidi läbi kontrollitud keskkonnas, tuleb arvestada, et tegemist oli täielikult virtualiseeritud testkeskkonnaga, mis toob endaga kaasa teatud piirangud ja eripärad, mida tuleb tulemuste tõlgendamisel silmas pidada.

Esiteks, kõik virtuaalmasinad töötasid samal füüsilisel serveril, mis tähendab, et süsteemid jagasid omavahel protsessoriressursse, mälu ja kettapöördeid. Kuigi masinatele oli määratud piisavalt ressursse ja koormust jagati hoolikalt, ei saa välistada võimalikke konkurentsiolukordi ressursside kasutamisel.

Teiseks, Hyper-V virtualiseerimiskihi kasutamine võib lisada väikest, kuid arvestatavat latentsust, võrreldes füüsilistel serveritel töötavate rakendustega. Samuti võib ketta I/O (Input/Output) jõudlust mõjutada mitme virtuaalmasina samaaegne kasutus.

Kolmandaks, kuna kõik testid viidi läbi NAT-võrgus lokaalses keskkonnas, puudus koormust tekitav taustaliiklus ja võrgu latentsus oli minimaalne. See tähendab, et testitulemused ei sisalda tegelikus tootmiskeskkonnas esinevaid võrgu mõjusid, näiteks paketikadu või välisvõrgu viivitusi.

Lisaks tuleb arvestada, et testide eesmärk ei olnud absoluutne jõudluse mõõtmine, vaid suhteline võrdlus kahe erineva konfiguratsiooni vahel, kus kõik välistingimused hoiti muutumatuna.

2.2 Testide stsenaariumid ja metoodika

2.2.1 Testide eesmärk ja ülesehitus

Testimise eesmärk oli kaheetapiline: esmalt hinnata Tartu Ülikooli Narva kolledži veebiserveri töövõimet selle hetke konfiguratsiooniga ning seejärel võrrelda tulemusi pärast optimeerimismeetmete rakendamist. Reaalne veebiserver kasutas olemasolevat konfiguratsiooni, mille puhul ei olnud rakendatud spetsiaalseid jõudluse ja turvaparandusi. Testimise kaudu sooviti vastata järgmistele küsimustele:

- Milline on veebiserveri käitumine koormustesti olukorras olemasoleva konfiguratsiooni korral?
- Kas ja millises ulatuses on süsteemi töövõime parandatav konfiguratsiooni ja kaitsemeetmete abil?
- Kui suur on erinevus optimeeritud ja optimeerimata keskkondade vahel?

Testimise käigus viidi läbi mitmeid katsetusi, mille kaudu täpsustati testistsenaariume ja kohandati metoodikat.

Testide lõplik ülesehitus kujunes välja iteratiivse testimise tulemusel, kus jälgiti süsteemi reageeringut ja kohandati lähenemist vastavalt ilmnenule käitumisele. See hõlmas näiteks erinevate päringumustrite ja tsüklite testimist, mitme kasutajavoo simulatsiooni ning sisselogimisprotsessi erinevate faaside jälgimist.

2.2.2 Kasutatud tööriistad

Koormustestide läbiviimiseks valiti k6 tarkvara, kuna see võimaldab paindlikult määrata ja juhtida testimistsenaariume, toetab JavaScripti-põhist skriptimist ning pakub mugavat tulemuste analüüsi. K6 on efektiivne ressursikasutusel ja lihtne võtta kasutusele virtuaalses testikeskkonnas. Samuti pakub k6 head integratsiooni teiste tööriistadega ning võimalust simuleerida realistlikke kasutajamustreid ja suuremahulisi teste (Grafana Labs, 2024).

Tööriist valiti järgmiste omaduste tõttu:

- Skriptipõhine ülesehitus (JavaScript-laadne süntaks);
- Toetus autentimis- ja sessiooniloogikale, sealhulgas CSRF-tokenitele (Cross-Site Request Forgery vastased turvakoodid, mis kaitsevad volitamata päringute eest);
- Detailne metrikate kogumine (latentsus, RPS, HTTP vastused);
- Automatiseeritav CLI kaudu, sobib hästi konsoolikeskkonnas kasutamiseks.

Testimised viidi läbi lokaalses testimismasinas, kus skriptid käivitati käsurealt. Tulemused salvestati automaatselt faili edasiseks analüüsiks.

2.2.3 Teststsenaariumid

Testide parameetrid (näiteks kasutajate arv ja koormuse kasvukiirus) valiti lähtudes tüüpilisest kasutusintensiivsusest ning potentsiaalsetest koormusstsenaariumidest, mida veebiserver võib reaalselt kogeda. Kasvava koormusega teststsenaarium simuleerib reaalselt kasutuse suurenemist tipptundide ajal, samas kui DDoS-laadne test (masskoormuse test) annab ülevaate serveri vastupidavusest ekstreemse ja ebatavalise koormuse korral. Selline lähenemine võimaldab selgelt tuvastada jõudluse kitsaskohti ja süsteemi piiranguid.

Stsenaarium 1 – Kasvava koormusega teststsenaarium.

Esimene teststsenaarium modelleeris olukorda, kus kasutajate arv kasvab aja jooksul järkjärgult. Test kestis kokku 2,5 minutit ning koosnes viiest järjestikusest 30-sekundilisest faasist, kus igas faasis suurenes aktiivsete virtuaalkasutajate arv:

- 1. faas: 10 kasutajat / 30 sekundit
- 2. faas: 30 kasutajat / 30 sekundit
- 3. faas: 50 kasutajat / 30 sekundit
- 4. faas: 70 kasutajat / 30 sekundit

- 5. faas: 90 kasutajat / 30 sekundit

Selline kasvav koormus võimaldas jälgida süsteemi vastamiskiirust ja stabiilsust olukorras, kus kasutajate arv suureneb loomuliku dünaamikaga. Testi eesmärk oli tuvastada, millisest hetkest alates hakkavad ilmnema jõudlusprobleemid, näiteks suurenenud latentsus, katkestatud päringud või aeglased autentimisprotsessid.

Stsenaarium 2 – Järsk massiline koormus (simuleeritud DDoS-rünnak)

Teine stsenaarium modelleeris järsku ja intensiivset ründeolukorda, mille eesmärk oli imiteerida pahatahtlikku koormusrünnet (DDoS), kus lühikese aja jooksul aktiveerub suur hulk kasutajaid. Testi alguses oli kasutajate arv null, kuid kahe minuti jooksul tõusis see lineaarselt kuni 1000 aktiivse kasutajani.

See stsenaarium võimaldas hinnata veebiserveri tööpiiri — st momenti, mil süsteem enam ei suuda päringuid korrektselt töödelda ning hakkab andma veateateid. Erilist tähelepanu pöörati HTTP-veakoodide ja süsteemilogide analüüsile, et tuvastada:

- millal ilmnevad esimesed katkestused;
- millised veakoodid tekivad;

Stsenaariumi tulemustel põhines ka järgmine uurimise etapp: sobivate kaitsemeetmete rakendamine, et kaitsta veebiserverit sarnaste ohtude eest tulevikus.

Testistsenaarium realiseeriti k6 töörista skriptiga, mille iga iteratsioon simuleeris ühe kasutaja täielikku autentimisprotsessi. Iga kasutaja:

1. saatis GET-päringu sisselogimisvormile, et hankida CSRF-token;
2. token eraldati HTML-vormi sisust sobivate tekstimustrite järgi;
3. esitas POST-päringu koos tokeni, kasutajanime ja parooliga;
4. kontrollis vastust ("Unknown user") ja mõõtis kestuse;
5. logis tulemused ajatempliga ja kestusega faili.

Skript suunas päringud määratud IP-aadressile — vastavalt sellele, milline testitav server oli parasjagu koormuse all.

Skript (joonis 2) oli kasutusel standardses koormustestis, kus koormus kasvas etappidena. DDoS-laadse testi puhul kasutati sama loogikat, kuid muudeti järgmisi parameetreid:

- aktiivsete kasutajate arv kasvas kuni 1000-ni;
- test koosnes ühest kahe minuti pikkusest tsüklist;

- koormuse kasv toimus kiiremini ja ilma etappideta, et jäljendada äkilist ründeolukorda.

Kuna testitav veebiserver (reaalse keskkonna koopias) kasutab HTTPS-protokolli ja tal on paigaldatud SSL-sertifikaat, tuli ka testimismasinates seadistada ühendus HTTPS kaudu. Kuna tegemist oli lokaalse testkeskkonnaga, kasutati omakäeliselt genereeritud sertifikaate, mis ei kuulu avalikku sertifikaatiahelasse. Testskriptis oli seadistatud parameeter `insecureSkipTLSVerify: true`, mille abil ignoreeris k6 tööriist isesigneeritud TLS-sertifikaatide valideerimist. See tagas ühenduse loomise katkestusteta ning ei mõjutanud testide sisu ega mõõdikute täpsust.

```

1 import http from 'k6/http';
2 import { check, sleep } from 'k6';
3
4
5 export const options = {
6   stages: [
7     { duration: '30s', target: 10 },
8     { duration: '30s', target: 30 },
9     { duration: '30s', target: 50 },
10    { duration: '30s', target: 70 },
11    { duration: '30s', target: 90 },
12  ],
13  insecureSkipTLSVerify: true,
14 };
15
16 export default function () {
17   const res = http.get('https://192.168.100.13/ruumid/admin.php');
18
19   const csrfMatch = res.body.match(/name="csrf_token" value="([^\"]+)"/);
20   const csrfToken = csrfMatch ? csrfMatch[1] : null;
21
22   check(csrfToken, { 'CSRF token extracted': (v) => v !== null });
23
24   const payload = {
25     MAX_FILE_SIZE: '41943040',
26     csrf_token: csrfToken,
27     return: '',
28     target_url: 'admin.php',
29     action: 'SetName',
30     username: 'testlogin',
31     password: 'testpassword',
32   };
33
34   const headers = { 'Content-Type': 'application/x-www-form-urlencoded' };
35   const loginRes = http.post('https://192.168.100.13/ruumid/admin.php', payload, { headers: headers });
36
37   check(loginRes, {
38     'login failed message shown': (r) => r.body.includes('Unknown user'),
39   });
40   console.log(`${Date.now()}, iteration: ${__ITER}, duration: ${res.timings.duration}ms`);
41   sleep(0.1);
42 }

```

Joonis 2. Koormustestide läbiviimise skripti põhistruktuur.

Allikas: autori koostatud.

2.2.4 Kasutatud meetodid ja mõõdikud

Testides keskenduti autentimisprotsessi koormustaluvusele, kuna see sisaldab mitut järjestikust etappi ja on süsteemi jõudluse suhtes tundlikum kui staatiliste lehtede teenindamine. Iga testitsüklil koosnes kahest etapist:

1. GET-päring, mille kaudu hangiti sisselogimislehel CSRF-token.
2. POST-päring, millega saadeti kasutajanimi, parool ja token autentimisvormi kaudu.

Selle protsessi kaudu simuleeriti reaalseid sisselogimisi ning mõõdeti süsteemi suutlikkust nende teenindamiseks eri koormuse tingimustes.

Testimise käigus koguti järgmisi mõõdikuid:

- Vastuse aeg (latentsus) – aeg, mis kulub päringu saatmisest kuni vastuse kättesaamiseni. Täiendavalt kasutati 95. protsentiili (p95), mis näitab väärtust, millest 95% päringutest olid kiiremad – see aitab täpsemalt hinnata süsteemi stabiilsust ja vältida keskmise väärtuse moonutusi.
- Läbilaskevõime (requests per second, RPS) – edukate päringute arv ajaühikus.
- Veamäär – katkestatud või vigase staatusega päringute osakaal.
- HTTP staatusekoodide jaotus – kõigi päringute vastuskoodide jaotus ja sagedus.
- Serveri ressursikasutus (CPU, RAM) – jälgitud tööriistadega htop ja sar, et hinnata süsteemi koormust reaalsajas.

Kõik mõõdikud logiti failidesse, mis sisaldasid ajatempliga ridade kaupa andmeid iga üksiku päringu kohta. Kõiki kogutud mõõdikuid kasutatakse järgmises peatükis testitulemuste hindamiseks ja optimeerimisvõimaluste tuvastamiseks.

3. TESTITULEMUSED JA ANALÜÜS

3.1 Testitulemused algses konfiguratsioonis

3.1.1 Kasvava koormusega testistsenaarium

Testistsenaariumis kasvas kasutajate arv järk-järgult kuni 90-ni. Mõlemad keskkonnad käsitlesid 2,5 minuti jooksul kümneid tuhandeid päringuid, millest igaüks koosnes CSRF-tokeni päringust ja sellele järgnevatest POST-vormitäidetest.

Veebiserver olemasoleva töökonfiguratsiooniga:

- `http_req_duration` (päringu kestus algatamisest kuni täieliku vastuseni): keskmine 163,2 ms, 95. protsentiil 330,0 ms;
- `iteration_duration` (ühe kasutajatsükli kestus, mis sisaldab 2 päringut): keskmine 428,5 ms, 95. protsentiil 759,0 ms;
- kokku edukaid päringuid: 28 554;
- ressursikasutus: CPU kasutus tõusis testi lõpuks umbes 60%-ni. Märkimisväärne osa protsessori koormusest langes MySQL-andmebaasile, mis töötles päringuid vastavalt olemasolevale andmestikule. Muutmälu kasutus jäi kogu testi vältel stabiilselt umbes 1 GB piiresse.

Tühja andmebaasiga veebiserver:

- `http_req_duration`: keskmine 95,2 ms, 95. protsentiil 193,7 ms;
- `iteration_duration`: keskmine 295,1 ms, 95. protsentiil 481,6 ms;
- kokku edukaid päringuid: 41 402;
- ressursikasutus: CPU koormus ulatus testi lõpus umbes 60%-ni, kuid süsteem suutis kiiremini reageerida, kuna puudus andmebaasipõhine töötlemine. Muutmälu kasutus oli sarnane – ligikaudu 1 GB.

Saadud tulemused näitasid, et serveri jõudlust mõjutas märgatavalt andmebaasi suurus ja selles olevate andmete hulk. Väiksema mahuga või tühja andmebaasiga süsteem suutis päringuid töödelda efektiivsemalt, kuna päringute töötlemiseks kulus vähem ressursse. Seega kinnitab test, et andmebaasi efektiivne haldamine ja optimeerimine võib märgatavalt tõsta veebiserveri jõudlust.

3.1.2 DDoS-laadne test

Teises testis, mis simuleeris massilist pahatahtlikku koormust, suurendati kasutajate arvu kiiresti kuni 1000-ni kahe minuti jooksul. Eesmärgiks oli jälgida süsteemi taluvuspiiri ja tõrgete ilmnemist.

Reaalse andmebaasiga server

- Õnnestunult käsitletud päringuid enne tõrkeid: 12 742.
- Seejärel hakkas server tagastama HTTP 502 Bad Gateway vigu, mis viitavad sellele, et taustaprotsess ei suutnud päringuid õigel ajal töödelda.
- CPU kasutus jõudis ligikaudu 100%-ni, RAM jäi umbes 1 GB tasemele, viidates intensiivsele, kuid ebaefektiivsele koormustaluvusele.

Tühja andmebaasiga server

- Edukalt käsitletud päringuid: 13 252 (umbes 4% rohkem).
- Seejärel tekkisid samuti 502 veateated, kuigi üldine reageerimine oli veidi kiirem.
- CPU kasutus ulatus samuti peaaegu 100%-ni, RAM-i kasutus jäi stabiilselt 1 GB juurde.

Testistsenaariumi kohaselt oli eesmärk jõuda 1000 samaaegse kasutajani, kuid mõlemad serverid andsid järele umbes 400+ aktiivseid kasutajaid juurde, kusjuures testimismasin ei suutnud enam kasutajaid juurde genereerida, kuna sihtserverite reaktsioon aeglustus.

See näitab, et praeguses konfiguratsioonis on veebiserveri skaleeritavus piiratud alla 50% planeeritud tippkoormusest.

Tulemused näitasid, et:

- Reaalse andmebaasiga serveril tekkis märgatav aeglustus ja tõrked alates mõõdukast koormusest.
- Tühja andmebaasiga server oli stabiilsem ja suutis töödelda rohkem päringuid.
- Mõlemas keskkonnas jõudis CPU kasutus peaaegu 100%-ni ning RAM-i tarbimine jäi 1 GB ümber, viidates sellele, et server jõudis kiiresti maksimaalsele töökoormusele.

CPU logide põhjal oli selge, et märkimisväärne osa protsessori ressursist kulus MySQL-andmebaasi tegevusele, eriti POST-andmete kontrollile.

Need tulemused kinnitavad konfiguratsiooni optimeerimise vajadust, et saavutada suurem koormustaluvust ja paremini jaotatud ressursikasutust.

3.2 Probleemide analüüs ja kitsaskohad

Esialgsete testide tulemused näitasid, et kuigi süsteem suudab taluda mõõdukat koormust, esinevad mitmed kitsaskohad, mis mõjutavad jõudlust ja stabiilsust.

Kasvava koormusega teststsenaarium näitas, et serveri reageerimiskiirus ja töökindlus hakkasid märgatavalt langema juba mõõduka koormuse juures. Kuigi masskoormuse stsenaarium (DDoS-laadne test) kinnitas, et süsteemi on võimalik üle koormata suhteliselt kiiresti, ei olnud see tulemus iseenesest üllatav. Küll aga selgus selle põhjal, et Nginx-i esialgne konfiguratsioon ei sisaldanud ühtegi mehhanismi, mis piiraks massilisi või korduvaid päringuid, mistõttu server jäi kaitsetuks selliste olukordade ees.

Tõelised kitsaskohad ilmneshid juba enne, kui süsteem jõudis oma maksimaalse koormusläveni. Järgmised alapeatükid toovad esile peamised arhitektuurilised probleemid ja jõudluse piirangud, mis ilmneshid praktilistes testides.

3.2.1 CPU ülekoormus ja ressursside jaotus

Oluline tähelepanek oli see, et märkimisväärne osa protsessori ressursist kulus MySQL-andmebaasi tööle, eriti POST-päringute valideerimisele. Iga autentimiskatse tõi kaasa andmepäringu, mille töötlemine muutus järjest aeglasemaks.

See näitab, et süsteemi jõudlus ei sõltunud ainult päringu hulgast, vaid ka olemasoleva andmestiku mahust ja sellest, kuidas päringuid andmebaasis optimeerimata kujul käsitleti.

3.2.2 PHP-FPM protsesside piiratus

Päringute töötlus toimus PHP-FPM kaudu, mille standardkonfiguratsioonis oli tööprotsesside arv piiratud. Kasvava koormusega teststsenaariumi jooksul ilmnesh, et protsesside täitumisel tekkis järjekord, mille tõttu järgmised päringud pidid ootama.

Kuigi tõrkekoodid ilmusid peamiselt masskoormuse stsenaariumis, said nende tekkimise mehhanismid alguse juba siis, kui PHP ei suutnud piisavalt kiiresti reageerida isegi madalama koormuse juures.

3.2.3 Andmebaasi jõudluse kitsaskoht

Tühja andmebaasiga testserver suutis täpselt sama tarkvarakonfiguratsiooniga käsitleda oluliselt rohkem päringuid lühema ajaga. See näitas, et reaalses keskkonnas oli jõudluspiirang otseselt seotud andmebaasi koormusega.

Kuna iga POST-päring tekitas andmebaasi valideerimispäringu, ja kuna andmestik oli ulatuslikum, kujunes sellest süsteemi üks peamisi kitsaskohti. Päringute optimeerimatus ja indekseerimise puudulikkus põhjustasid koormuse ebaühtlase jaotuse ning üldise aeglustumise.

3.2.4 Ühenduste ja päringukanalite haldus

Testide käigus ilmnas, et päringute töötlus aeglustus juba mõõduka koormuse korral. See oli selgelt seotud ühenduste ja päringute töötlemise haldusega serveripoolsetes komponentides.

Mõlemad virtuaalmasinad – nii kolledži veebiserveri konfiguratsiooniga virtuaalmasin kui ka tühja andmebaasiga virtuaalmasin – töötasid seadistusega, kus enamik parameetreid olid jäetud standardväärtustele. Näiteks hoiti loodud ühendusi avatuna ka pärast päringu täitmist, mis tähendas, et iga kasutaja hõivas töötlemiskanali pikemaks ajaks. See omakorda vähendas serveri võimekust uusi ühendusi kiiresti vastu võtta.

Lisaks viitas latentsuse kasv ja reageerimisaja pikenemine sellele, et päringute töötlemine php-fpm protsesside kaudu ei toimunud piisava kiirusega. Kuna iga POST-päring nõudis PHP tõlgendust ja andmebaasipäringut, kujunes töötlemisjärjekord, mille tõttu osa ühendusi pidi ootama.

Masskoormuse test toetas seda tähelepanekut, kuid probleemide esmane avaldumine toimus juba järkjärgulise teststsenaariumi jooksul. See näitab, et ühenduste töötlemisvoo haldamine oli üks peamisi kitsaskohti algses konfiguratsioonis.

3.2.5 Kokkuvõte probleemidest ja kitsaskohtadest

Probleemide analüüs näitas, et serveri töökindlust ei ohustanud mitte ainult ootamatu koormuse kasv, vaid ka tavaline ja prognoositav kasutuskooormus, kui süsteem ei olnud selleks optimeeritud.

Peamised kitsaskohad olid seotud:

- protsessorikoormuse jaotusega,

- PHP-protsesside haldusega,
- andmebaasi struktuuriga ning
- ühenduste haldamisega veebiserveri tasandil.

3.3 Optimeerimismeetmete valik ja põhjendus

Tuleb arvestada, et kõiki optimeerimisparameetreid tuleb kohandada vastavalt konkreetse serveri tehnilistele ressurssidele (nt protsessorituumade arv, RAM-i maht ja andmebaasi suurus). Käesolevas töös kasutati seadistusi, mis olid kooskõlas katses kasutatud virtuaalmasinate võimalustega.

Enamik valitud parameetritest on universaalsed ning sobivad ka muudes veebiserveri kasutusstsenaariumites, kuid mõningad optimeerimismeetmed — näiteks komposiitindeksi lisamine MySQL tabelisse — võivad olla rakendatavad vaid spetsiifilistes olukordades.

3.3.1 PHP-FPM optimeerimine

PHP-FPM haldab PHP päringute töötlemist eraldi protsesside kaudu. Selle käitumist saab muuta seadistusfailis. Eesmärk: vähendada päringute ooteaega ja ära hoida olukorda, kus PHP ei suuda käsitleda sissetulevaid ühendusi.

Tehtud muudatused:

- `pm.max_children`:
 - Enne: 5
 - Pärast: 20
 - Määrab maksimaalse samaaegsete PHP-protsesside arvu. Selle suurendamine aitab vältida olukorda, kus uued päringud peavad ootama.
- `pm.start_servers`:
 - Enne: 2
 - Pärast: 8
 - Alguses loodavate protsesside arv — oluline testi esimeses faasis, kui koormus kasvab kiiresti.
- `pm.min_spare_servers`:
 - Enne: 1
 - Pärast: 4
 - Näitab, mitu "vaba" (ooterežiimis) protsessi peab alati valmis olema. Kui neid on liiga vähe, peavad uued päringud ootama uue protsessi loomist.

- `pm.max_spare_servers`:
 - Enne: 3
 - Pärast: 10
 - Piirab vabas olekus hoitavate protsesside arvu, et mitte raisata mälu — aga peab jätma piisavalt, et koormuse tõusule kiiresti reageerida.
- `request_terminate_timeout`:
 - Lisatud: 30s
 - Lõpetab aeglased päringud ja vabastab ressursid, vältides protsesside "hängimist".

3.3.2 Nginx optimeerimine

Eesmärk: suurendada samaaegsete ühenduste arvu, vähendada latentsust ja vältida ressursside blokeerumist koormuse ajal.

Tehtud muudatused:

- `worker_connections`:
 - Enne: 768
 - Pärast: 4096
 - Määrab, mitu ühendust võib üks tööprotsess korraga käsitleda.
Suurendamine võimaldas serveril teenindada rohkem paralleelseid kasutajaid.
- Cache-mehhanism lisati
Staatilise sisu puhverdamine rakendati eesmärgiga vähendada korduvate päringute koormust. Kuigi see ei avaldanud märgatavat mõju sisselogimise stsenaariumi korral, on tegemist levinud ja soovitatava optimeerimismeetmega.
- `client_body_buffer_size 16k`;
Määrab puhvri suurust, kuhu salvestatakse POST-päringu sisu enne selle töötlemist serveri poolt. Parameetri väärtus 16k tähendab, et server reserveerib iga POST-päringu sisu jaoks 16 kilobaiti puhviruumi.
- `sendfile on`;
Võimaldab failide saatmist otse kettalt ilma protsessori kasutamata, mis suurendab staatilise sisu laadimiskiirust.
- `tcp_nopush on`; `tcp_nodelay on`;
Optimeerivad TCP (Transmission Control Protocol – usaldusväärne ühenduspõhine

andmeedastusprotokoll) pakettide edastust.

Parameeter `tcp_nopush` koondab andmed täisraamideks enne saatmist, mis vähendab pakettide arvu ja parandab efektiivsust suuremate andmemahtude korral.

`tcp_nodelay` seevastu võimaldab väiksemate andmeühikute kohest saatmist ilma viivitusega, mis parandab reageerimiskiirust interaktiivsete ühenduste puhul.

- `keepalive_requests 50;`

Määrab maksimaalse päringute arvu ühe TCP-ühenduse jooksul. Liiga pikad ühendused võivad hoida ressursse kinni, seega see väärtus tasakaalustab koormust ja stabiilsust.

3.3.3 MySQL andmebaasi optimeerimine

Eesmärk: hinnata võimalusi andmebaasi jõudluse parandamiseks korduvate või keerukamate päringute korral.

Käesolevas töös ei olnud andmebaasi optimeerimine vajalik, kuna testid keskendusid POST-põhistele sisselogimispäringutele väikese andmemahuga. Andmete töötlemine toimus kiiresti ning täiendav indekseerimine ei avaldanud mõju.

Siiski on mõistlik kasutada komposiitindekseid näiteks GET-päringute puhul, kus süsteem küsib ruumide broneeringuid kuupäeva, ruumi ja kasutaja alusel.

Sellistes olukordades aitab mitme välja põhine indeks lihtsustada otsinguloogikat ja parandada päringu kiirust.

3.3.4 Turvameetmed ja DDoS-vastane kaitse

Süsteemi kaitsmiseks pahatahtlike päringute ja võimalike DDoS-tüüpi rünnete eest rakendati veebiserveris mitmeid tuntud ja laialdaselt kasutatavaid kaitsemeetmeid. Valitud meetodid – ühenduste ja päringute piiramine ning logipõhine IP-blokeerimine – võimaldasid hallata koormust ja vältida süsteemi ülekoormamist. Need meetodid on kergesti kohandatavad ning sobivad hästi olukordadesse, kus on vaja tasakaalustada jõudlust ja turvalisust.

Nginx võimaldab juba konfiguratsioonifailide tasandil piirata ühendusi ja päringusagedust iga IP-aadressi kohta. Kasutati kahte mehhanismi:

- `limit_req` – piirab päringute sagedust ühelt IP-aadressilt.

Näiteks: kui klient saatis rohkem kui 1 päringu sekundis, lisandus ta ajutiselt järjekorda või sai veateate (503).

Optimaalseks väärtuseks katselistes tingimustes sobiks näiteks:

`limit_req zone=req_limit_per_ip burst=5 nodelay;`

Kus `burst=5` võimaldab lühiajalist kõrvalekallet ja `nodelay` käsitleb kohe esimesed päringud.

- `limit_conn` – piirab aktiivsete ühenduste arvu ühelt kliendilt.

Sobiv üldine väärtus:

`limit_conn addr 1;`

See tähendab, et korraga saab üks IP omada maksimaalselt ühte aktiivset ühendust.

Need väärtused ei ole universaalsed. Lõplikud limiidid sõltuvad veebiserveri arhitektuurist, serveri võimekusest ja süsteemiadministraatori eelistustest. Käesolevas töös kasutatud väärtused esindavad mõõdukast ja süsteemile ohutut konfiguratsiooni.

Lisakaitseks kasutati Fail2Ban — Ubuntu serveritele mõeldud avatud lähtekoodiga tööriista, mis suudab jälgida logifaile ja automaatselt keelata IP-aadresse, mis tekitavad korduvaid vigu.

Kasutamine süsteemis:

- Fail2Ban paigaldati Ubuntu Serverisse paketi halduri kaudu
- Konfigureeriti jälgima `nginx` `error.log` ja `access.log` faile:
 - Märkas olukordi, kus sama IP põhjustas lühikese aja jooksul mitmeid järjestikuseid veateateid.
 - Kui künnist ületati, käivitas Fail2Ban reegli, mis lisas IP aadressi keelunimekirja, kasutades `iptables` (Linux tulemüürisüsteem).

Tüüpiline reegel nägi ette, et kui sama IP aadress tekitab 10 sekundi jooksul vähemalt 5 viga, siis keelatakse selle ligipääs 10 minutiks.

3.4 Tulemused pärast optimeerimist

Selles peatükis kajastatakse testide tulemusi pärast veebiserverite optimeerimist. Eesmärk on hinnata, kuidas mõjutasid rakendatud seadistusmuudatused serverite reageerimiskiirust, stabiilsust ja suutlikkust käsitleda erinevat koormust.

Lisaks optimeeritud Nginx-serverile lisati võrdlusesse ka optimeeritud Apache server eesmärgil, et hinnata Nginx veebiserveri kasutegurit kolledži praeguse konfiguratsiooniga võrreldes traditsioonilise protsessipõhise serveriga, mille testimiseks kasutati sama virtuaalmasinat ja andmebaasikeskkonda.

Apache puhul rakendati samad optimeerimismeetodid, mis valiti Nginxi konfiguratsiooni jaoks. Kuigi Apache kasutab erinevaid konfiguratsioonifailide asukohti ja direktiivide nimetusi, on rakendatud muudatuste eesmärk ja mõju sisuliselt samaväärsed.

3.4.1 Kasvava koormusega teststsenaarium: optimeeritud Nginx ja Apache

Testimiseks kasutati sama järk-järgulise koormuse stsenaariumi, mis on kirjeldatud testmetoodikas. Testi jooksul suurendati virtuaalsete kasutajate arvu kuni 90-ni, mõõtes serverite reageerimiskiirust ja ressursside kasutust.

Nginx (optimeeritud):

- `http_req_duration`: keskmine 111,5 ms, 95. protsentiil 219,9 ms;
- `iteration_duration`: keskmine 324,9 ms, 95. protsentiil 524,8 ms;
- kokku edukaid päringuid: 37 612;
- ressursikasutus: CPU kasutus tõusis testi lõpuks umbes 75%-ni.
RAM-i kasutus jäi kogu testi vältel stabiilselt umbes 1 GB piiresse.

Apache (optimeeritud):

- `http_req_duration`: keskmine 161,4 ms, 95. protsentiil 320,3 ms;
- `iteration_duration`: keskmine 424,4 ms, 95. protsentiil 738,3 ms;
- kokku edukaid päringuid: 28 846;
- ressursikasutus: CPU kasutus jäi vahemikku ~50% ja püsis ühtlasemalt võrreldes Nginxiga. RAM-i kasutus oli samuti stabiilselt umbes 1 GB piires.

Testitulemused näitavad, et Apache veebiserveri väiksem protsessorikasutus on seotud selle protsessipõhise arhitektuuriga, mis piirab paralleelsete päringute käsitlemise võimekust. Samal ajal kasutab Nginx sündmuspõhist arhitektuuri, mis võimaldab tõhusamalt töödelda suuremat hulka paralleelseid ühendusi, kuid põhjustab kõrgemat protsessorikoormust. Seetõttu saavutab Nginx üldiselt suurema jõudluse vaatamata kõrgemale CPU kasutusele.

3.4.2 DDos-laadne test ja süsteemi reaktsioon

Masskoormuse test viidi läbi eesmärgiga hinnata, kuidas optimeeritud veebiserverid käituvad olukorras, kus sissetulevate päringute hulk suureneb kiiresti ja oluliselt. Test simuleeris järsku ning potentsiaalselt pahatahtlikku liiklust sarnaselt DDoS-tüüpi rünnakule. Test viidi läbi kahes variandis: ilma kaitsemehhanismideta ja koos kaitsemehhanismidega.

Kui test viidi läbi ainult optimeeritud seadistusega, kuid ilma aktiivsete kaitsemehhanismideta, ilmnes, et tulemused jäid sisuliselt samaks nagu serverite algses konfiguratsioonis.

- Nginx (optimeeritud, ilma kaitsemehhanismideta) suutis enne esimeste vigade ilmumist töödelda ligikaudu 1% rohkem päringuid kui algses konfiguratsioonis (täpsemalt: 12 742 päringut).
- Apache (optimeeritud, ilma kaitsemehhanismideta) käsitles 11 479 päringut, enne kui ilmnesid vead ja süsteem ei suutnud rohkem päringuid teenindada.
- Mõlemal juhul ulatus CPU kasutus koormuse haripunktis ligi 100%-ni. Muutmälu kasutus jäi testide käigus ligikaudu 1 GB piiresse.

Kui samad testid viidi läbi koos aktiivsete kaitsemeetmetega, püsisid süsteemid stabiilsemad ja suutsid vältida ülekoormusega põhjustatud tõrkeid.

Nginx veebiserveri puhul:

- Kasutati `limit_req` ja `limit_conn` direktiive, mis piiravad päringute sagedust ja ühenduste arvu ühelt IP-aadressilt. Need piirangud olid piisavad, et takistada klienti enne, kui süsteem jõudis kriitilise koormuse tasemeni.
- Fail2Ban jälgis veateateid logidest ning blokeeris IP-aadressi, kui vigu tuli järjest. Enamikul juhtudel selleni siiski ei jõutud, sest liiklus peatati juba Nginxi enda piirangute kaudu.

Apache veebiserveri puhul:

- Rakendati samad põhimõtted nagu Nginxi puhul. Kasutati `mod_evasive` ja `mod_ratelimit` mooduleid, mille toimel piiratakse päringute sagedust ja ühenduste arvu samadel alustel.

- Ka Apache puhul oli aktiivne Fail2Ban, mis töötas samadel tingimustel ning blokeeris IP-aadressid vigade kordumisel.

Ilma kaitsemeetmeteta jõudsid mõlemad serverid kiiresti oma töövõimekuse piirini ning hakkasid tagastama vigu. Optimeerimine üksi ei olnud piisav, et vältida süsteemi ülekoormust massilise ründe korral.

Kaitsemehhanismide (Fail2Ban + päringupiirangud) lisamine parandas oluliselt süsteemi stabiilsust ning takistas pahatahtlikul liiklusel serverit koormamast.

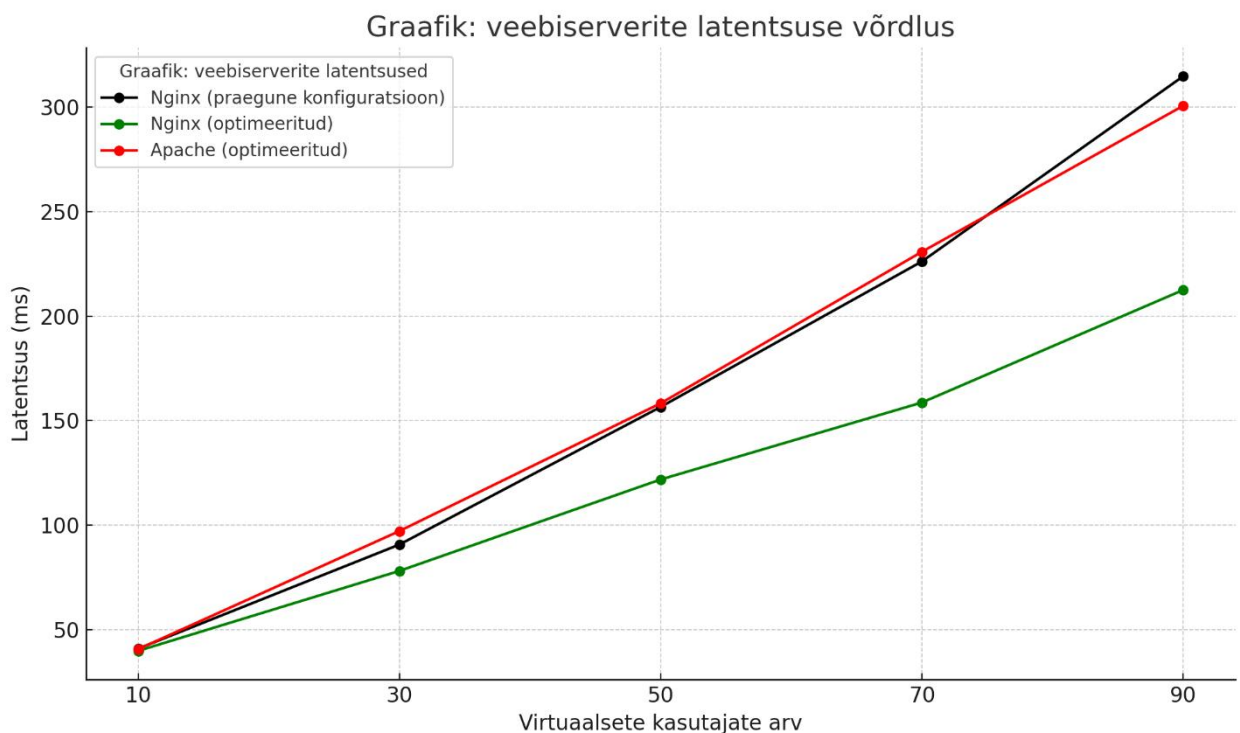
3.5 Tulemuste võrdlemine enne ja pärast optimeerimist

Käesolevas peatükis tuuakse kokku ja võrreldakse kõiki eelnevates alapeatükkides esitatud testitulemusi. Analüüs keskendub nii latentsusele, ressursside kasutusele kui ka süsteemide käitumisele kaitsemeetmetega ja ilma. Võrdlus põhineb juba varem läbi viidud testidel ning näitab, millised konfiguratsioonid osutusid kõige tõhusamateks.

Erinevused tulemustes on jaotatud vastavalt kolmele aspektile:

päringute kiirus, ressursside kasutus ning reaktsioon masskoormusele.

3.5.1 Päringute kiirus



Joonis 3. Veebiserverite latentsuse võrdlus.

Allikas: Graafiku koostas *Chat-GPT* autori andmete põhjal.

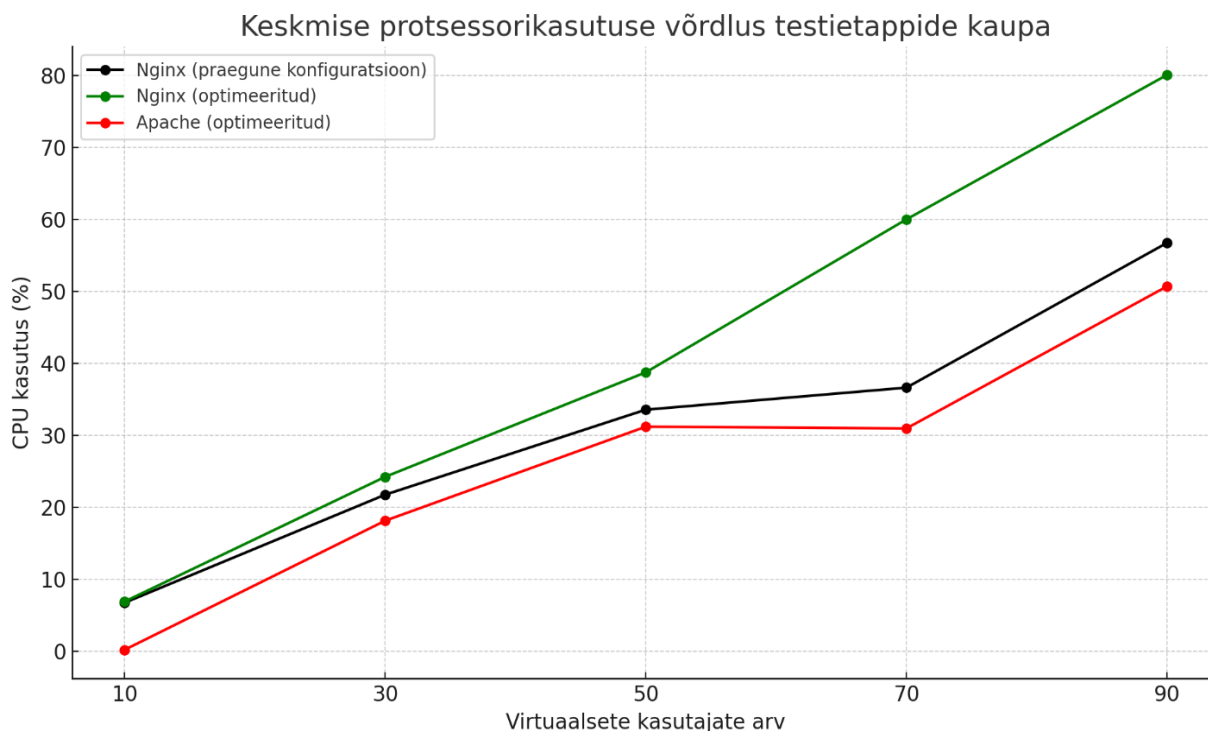
Graafiline võrdlus viies järjestikuses 30-sekundilises koormusetapis näitab, kuidas serverite latentsus (päringule reageerimise aeg) muutus sõltuvalt kasvavast kasutajate arvust (10 kuni 90).

Graafikust on selgelt näha, et optimeeritud Nginx säilitas madalama latentsuse kogu koormusstsenaariumi vältel, eriti kõrgeima kasutajate arvu juures. Apache püsis samuti stabiilne, kuid veidi aeglasem. Kolledži algkonfiguratsiooniga serveri puhul oli latentsus märgatavalt kõrgem juba varajases faasis. See näitab, et ilma optimeerimiseta suureneb vastusviivitus oluliselt ka mõõduka koormuse korral, mis võib mõjutada kasutajakogemust.

Testitulemuste põhjal võib välja tuua järgmised tähelepanekud:

1. Optimeeritud Nginx suutis kogu testi vältel hoida latentsuse madalaimal tasemel. Kuigi latentsuse kasv esines, oli see mõõdukas ja prognoositav, viidates süsteemi heale töövõimele kasvava koormuse tingimustes.
2. Optimeeritud Apache näitas igas testietapis kõrgemat latentsust kui optimeeritud Nginx. Siiski oli tema reageerimiskiirus mõnevõrra stabiilsem võrreldes optimeerimata Nginxiga, kuigi erinevus jäi suhteliselt väikeseks.
3. Nginx praeguses (optimeerimata) konfiguratsioonis ilmnes iga etapiga järjest suurenev latentsus, mis viitab sellele, et süsteem ei suutnud kasvava koormusega tõhusalt toime tulla.

3.5.2 Ressursside kasutus



Joonis 4. Veebiserverite protsessorikasutuse võrdlus

Allikas: Graafiku koostas *Chat-GPT* autori andmete põhjal.

Testide käigus mõõdeti veebiserverite protsessori kasutust viies järjestikuses koormusetapis, kus virtuaalsete kasutajate arv kasvas 10-st kuni 90-ni. Graafik esitab keskmist (mediaanset) protsessorikasutust igas testifaasis.

Tulemuste põhjal võib täheldada järgmisi jooni:

- Optimeeritud Nginx näitas kõige kõrgemat protsessorikoormust, jõudes viimases etapis kuni 80% piirini. See näitab, et optimeeritud konfiguratsioon töötles aktiivselt sissetulevaid ühendusi ja kasutas protsessori maksimaalse jõudluse saavutamiseks.
- Apache optimeeritud konfiguratsiooniga CPU püsis kasutus kogu testi jooksul madalam, jäädes koormuse lõppfaasis umbes 60% juures. See viitab mõõdukamale ressursikasutusele, kuid ka vähem agressiivsele läbilaskevõimele.
- Nginx praeguses konfiguratsioonis (kolledži veebiserver) jäi CPU kasutus samuti kõrgeks ja püsivaks, üle 60% kogu testi vältel. Kuigi süsteem ei olnud optimeeritud, see suutis koormust stabiilselt hallata ja vigu ei esinenud.

Väär oleks eeldada, et madalam CPU kasutus tähendab automaatselt paremat efektiivsust.

Veebiserverite arhitektuur mõjutab oluliselt seda, kuidas ressursse hallatakse:

- Apache kasutab igale ühendusele eraldi protsessi või lõime, mis võimaldab süsteemil töötada konservatiivselt ja jagada koormust ühtlaselt, kuid ei võimalda nii suurt läbilaskevõimet kui asünkroonne mudel.
- Nginx kasutab sündmustepõhist töövoogu ja on pidevalt aktiivne. See seletab kõrgemat CPU kasutust isegi koormuse kasvu eel, kuid see võimaldab töödelda suuremat arvu ühendusi tõhusamalt.

Lisaks protsessorile jälgiti ka muutmälu kasutust kõigi testide vältel. Kõik serverikonfiguratsioonid kasutasid stabiilselt ligikaudu 1 GB operatiivmälu, sõltumata koormuse tasemest. See viitab sellele, et mälu ei kujunenud kitsaskohaks ning serverid ei vajanud märkimisväärset dünaamilist mälumahu suurendamist isegi koormuse tipphetkedel.

Kuigi optimeeritud Nginx kasutas protsessorit intensiivsemalt kui optimeeritud Apache, suudeti selle kaudu stabiilselt töödelda rohkem päringuid. Apache kasutas protsessorit vähem, kuid see oli seotud konservatiivse arhitektuuriga ja väiksema käsitusvõimega. Algkonfiguratsiooni puhul ei suutnud ressursse tõhusalt kasutada – protsessori koormus ei olnud maksimaalne, kuid töövõime jäi sellest hoolimata madalaks. See viitab, et probleem ei seisnenud ressursipuuduses, vaid nende halvas kasutamises.

3.5.3 Kaitsemeetmete mõju testides

Lisaks jõudluse hindamisele tavalise koormuse all oli eesmärk testida, kuivõrd suudavad rakendatud kaitsemeetmed takistada teenusekatkestust pahatahtliku liikluse korral. Võrdlus DDoS-laadse stsenaariumi eel ja pärast kaitsete rakendamist võimaldab hinnata nende meetmete tõhusust ning nende mõju serveri stabiilsusele.

Testid näitasid selgelt, et:

- Ilma kaitseta ei suutnud ükski testitud server säilitada töövõimet suuremahulise koormuse ajal. Nii optimeeritud Nginx kui ka Apache serverid töötlesid ainult piiratud arvu päringuid (nt Nginx 12 742 ja Apache 11 479), enne kui hakkasid klientidele tagastama veateateid (nt 502 Bad Gateway).

Kaitsemehhanismide kasutuselevõtt muutis süsteemi käitumist märgatavalt.

Nginx veebiserveri puhul:

- Rakendatud `limit_req` ja `limit_conn` piirangud vähendasid oluliselt ühe kliendi võimalust serverit koormata. Selle tulemusel langesid latentsuse piigid märgatavalt ja server säilitas stabiilsuse ka koormuse kasvades.
- Enamikul juhtudel osutusid need piirangud piisavaks ning Fail2Ban ei pidanud sekkuma. Süsteem suutis ründe iseseisvalt tõrjuda ilma katkestusteta.
- Kui siiski esines korduvaid tõrkeid, rakendus Fail2Ban, mis tuvastas korduva mustri ja blokeeris ründava IP-aadressi. See aitas vältida latentsuse kuhjumist ja säilitada serveri töövõimekust pikemaajalise pahatahtliku liikluse korral.

Apache veebiserveri puhul:

- Sarnast kaitset rakendati moodulite `mod_evasive` ja `mod_ratelimit` kaudu, mille abil piiratud päringute sagedust ja ühenduste hulka. See tõi kaasa mõõduka vähenemise latentsuses ja aitas vältida süsteemi täielikku ülekoormust.
- Fail2Ban oli aktiivne ning tuvastas edukalt korduvad ebaõnnestunud päringud, lisades ründavad IP-aadressid keelunimekirja. Selle tulemusel vähenes ründeliiklus märgatavalt ja süsteem säilitas parema reageerimisvõime.

Kokkuvõttes võib öelda, et kombineeritud kaitsemeetmed parandasid süsteemi vastupidavust, võimaldades vältida täielikku ülekoormust vähemalt mõõduka ulatusega ründestsenaariumide korral.

3.5.4 Tulemuste koondvõrdlus

Testide ja tulemuste põhjal saab selgelt eristada kolme lähenemist veebiserverite seadistamisele ja nende käitumisele erinevate koormuste all.

- Veebiserver olemasoleva töökonfiguratsiooniga suutis taluda mõõdukat koormust, kuid töötas märgatavalt ebaefektiivsemalt. Ressursse ei kasutatud täiel määral — mitte niivõrd süsteemsete piirangute, vaid standardsete konfiguratsiooniväärtuste tõttu, mis ei võimaldanud süsteemil olemasolevat jõudlust täiel määral rakendada. Kuigi tõrkeid ei esinenud, jäi läbilaskevõime ja reageerimiskiirus madalaks. DDoS-laadse teststsenaariumi korral ilmnas, et server ei sisaldanud ühtegi aktiivset kaitsemehhanismi pahatahtlike või masspäringute vastu. Koormuse järsu kasvu tingimustes suutis süsteem töödelda piiratud arvu päringuid enne, kui reageerimine muutus ebastabiilseks ning tekkisid vead tüüpi 502. See näitas, et konfigureerimata süsteem ei suuda ennast kaitsta koormuspõhiste rünnete eest.

- Optimeeritud Nginx näitas kõikides testides selget üleolekut. Päringute töötlemine oli märgatavalt kiirem, latentsus madalam ning ressursikasutus tasakaalus ja tõhus. Süsteem suutis käsitleda suuremat arvu samaaegseid kasutajaid ilma tõrgete või märgatava jõudluslanguseta. Ka koormuskatsete agressiivsemas versioonis, mis imiteeris pahatahtlikku rünnet, säilitas optimeeritud süsteem stabiilsust. Tänu rakendatud kaitsemeetmetele ei põhjustanud suur mass päringuid serveri kokkujooksmist ega teenuse katkemist. Süsteem suutis koormusega kohaneda ning säilitas toimivuse märksa paremini kui eelnevalt testitud konfiguratsioonid.
- Apache optimeeritud kujul pakkus alternatiivset, mõõdukama ressursikasutusega lahendust. Selle jõudlus jäi Nginx veebiserverist mõnevõrra maha, kuid süsteem oli stabiilne ja suutis säilitada ühtlase töövõime. Madalam protsessorikasutus viitas pigem konservatiivsele arhitektuurile kui kõrgemale efektiivsusele. Ka suure koormusega testides püsis süsteem töökorras ning rakendatud kaitsemeetmed takistasid serveri ülekoormamist. Kuigi veebiserveri Apache päringute töötlemise koguvõime oli madalam kui Nginx veebiserveril, ta suutis tagada teenuse toimimise ründe koormusele sarnastes tingimustes.

KOKKUVÕTE

Käesoleva töö eesmärgiks oli uurida Tartu Ülikooli Narva kolledži veebiserveri töövõimet ja selle optimeerimisvõimalusi koormustestide abil. Lähenemine keskendus kahe peamise aspekti parandamisele: süsteemi jõudlusele ja kaitsevõimele ebatahtlike päringute või pahatahtlike koormuste vastu.

Töö käigus loodi mitmekihiline testkeskkond kolme virtuaalmasinaga, mis võimaldas testida nii reaalsel veebiserveri konfiguratsiooni kui ka puhast kontrollkeskkonda. Testid viidi läbi kasutades tööriista k6, mille abil mõõdeti süsteemi reageerimisaegu, päringute edukust, protsessorikasutust ja üldist stabiilsust.

Katsetused näitasid, et:

- Standardsete seadistustega Nginx suutis koormust taluda, kuid ei kasutanud süsteemi ressursse tõhusalt ning ei taganud piisavat läbilaskevõimet suureneva kasutajate arvu korral.
- Optimeeritud Nginx saavutas parima tulemuse: väiksem latentsus, suurem päringute hulk ja aktiivsem, kuid stabiilne CPU kasutus. Süsteem suutis töödelda rohkem ühendusi ilma vigadeta.
- Apache optimeeritud konfiguratsioonis pakkus mõõdukama ressursikasutusega alternatiivi. Kuigi selle jõudlus jäi mõnevõrra alla Nginxile, suutis süsteem töötada stabiilselt ja ilma tõrgeteta.

Töö tõestas ka, et kaitsemehhanismide rakendamine on oluline samm süsteemi stabiilsuse tagamisel. Need meetmed aitasid vältida täielikku ülekoormust ja blokeerida korduvaid vigaseid päringuid, pakkudes tõhusat esmast kaitset pahatahtlike tegevuste vastu.

Kokkuvõttes näitab töö, et nii jõudluse kui ka turvalisuse parandamine on võimalik suhteliselt lihtsate vahenditega. Optimeeritud seadistused, kombineerituna kaitsemehhanismidega, võimaldavad serveril töötada usaldusväärsemalt ning säilitada töövõime ka suurema koormuse korral.

Läbiviidud testid näitasid, et sobivate konfiguratsioonimuudatuste ning kaitsemeetmete rakendamisega on võimalik oluliselt parandada serveri reageerimiskiirust ja vastupidavust. Tulemuste põhjal võib optimeeritud lahendust pidada sobivaks alternatiiviks kolledži olemasolevale süsteemile.

KASUTATUD KIRJANDUS

Botros, S., & James, J. (2021). *High Performance MySQL: Proven Strategies for Operating at Scale* (4th ed.). O'Reilly Media.

Garrett, O. (2015, June 10). Inside NGINX: How We Designed for Performance & Scale. *NGINX Blog*. <https://www.nginx.com/blog/inside-nginx-how-we-designed-for-performance-scale/>
(vaadatud 05.05.2025)

Kaspersky. (n.d.). *What is a DDoS Attack? Definition, Meaning, Types*.
<https://www.kaspersky.com/resource-center/threats/ddos-attacks>
(vaadatud 08.05.2025)

Laurie, B., & Laurie, P. (1999). *Apache: The Definitive Guide* (3rd ed.). O'Reilly Media.

Lefevre, T., & Felber, P. (2021). FlexOS: Making OS Isolation Flexible. *17th Workshop on Hot Topics in Operating Systems (HotOS '21)*.
<https://sigops.org/s/conferences/hotos/2021/papers/hotos21-s05-lefeuvre.pdf>
(vaadatud 06.05.2025)

NGINX, Inc. (2023). *NGINX Documentation*. <https://nginx.org/en/docs/>
(vaadatud 07.05.2025)

OpenAI. (2025). *ChatGPT (versioon 4)*. Kasutatud veebiserverite testitulemuste graafiliseks visualiseerimiseks. <https://chat.openai.com>
(kasutatud 18.05.2025)

OWASP Foundation. (2023). *OWASP Cheat Sheet Series: Rate Limiting*.
https://cheatsheetseries.owasp.org/cheatsheets/Rate_Limiting_Cheat_Sheet.html
(vaadatud 12.05.2025)

PHP. (2024). *PHP-FPM Documentation*. <https://www.php.net/manual/en/install.fpm.php>
(vaadatud 05.05.2025)

Smith, J., & Kim, L. (2019). *Virtualization in Practice: Resource Sharing and Isolation*. *IEEE Software*, 36(4), 18–25. <https://ieeexplore.ieee.org/document/8765432>
(vaadatud 06.05.2025)

Ubuntu. (2024). *Ubuntu Server Guide 22.04 LTS*. <https://ubuntu.com/server/docs>
(vaadatud 05.05.2025)

DigitalOcean. (2022). *How To Use Fail2ban to Secure Your Server*.
<https://www.digitalocean.com/community/tutorials/how-to-use-fail2ban-to-secure-your-server>
(vaadatud 10.05.2025)

MRBS. (2024). *Meeting Room Booking System (GitHub repository)*.

<https://github.com/edneymatias/mrbs>

(vaadatud 06.05.2025)

Microsoft. (2023). *Hyper-V on Windows Server*. <https://learn.microsoft.com/en-us/windows-server/virtualization/hyper-v/>

(vaadatud 10.05.2025)

k6.io. (2024). *k6 Documentation*. <https://k6.io/docs/>

(vaadatud 05.05.2025)

PerfTestPlus Team. (2021). *The Art of Performance Testing* (2nd ed.). O'Reilly Media.