VILNIAUS UNIVERSITETAS
ŠIAULIŲ AKADEMIJA

INFORMACINIŲ TECHNOLOGIJŲ VALDYMO MAGISTRO STUDIJŲ PROGRAMA

TOMA VISOCKYTĖ

Magistro studijų baigiamasis darbas

DEBESŲ PASLAUGŲ PROGRAMAVIMO SPRENDIMŲ TYRIMAS NAUDOJANT GOOGLE APP ENGINE APLINKĄ

Darbo vadovas: dr. L. Kaklauskas

Šiauliai, 2024

Studijuojančiojo, teikiančio baigiamąjį darbą, GARANTIJA

WARRANTY of Final Thesis

Vardas, pavardė *Name, Surname*	**Toma Visockytė**
Padalinys *Faculty*	**Šiaulių akademija** ***Šiauliai Academy***
Studijų programa *Study Programme*	**Informacinių technologijų valdymas** ***Information Technology Management***
Darbo pavadinimas *Thesis topic*	**Debesų paslaugų programavimo sprendimų tyrimas naudojant Google App Engine aplinką** ***Research on programming solutions for cloud services using the Google App Engine environment***
Darbo tipas *Thesis type*	**Baigiamasis darbas** ***Final Thesis***

Garantuoju, kad mano baigiamasis darbas yra parengtas sąžiningai ir savarankiškai, kitų asmenų indėlio į parengtą darbą nėra. Jokių neteisėtų mokėjimų už šį darbą niekam nesu mokėjęs.

Šiame darbe tiesiogiai ar netiesiogiai panaudotos kitų šaltinių citatos yra pažymėtos literatūros nuorodose.

I guarantee that my thesis is prepared in good faith and independently, there is no contribution to this work from other individuals. I have not made any illegal payments related to this work.

Quotes from other sources directly or indirectly used in this thesis, are indicated in literature references.

Aš, Toma Visockytė, pateikdamas (-a) šį darbą, patvirtinu (*pažymėti*) ☒

Embargo laikotarpis
Embargo Period

Prašau nustatyti šiam baigiamajam darbui toliau nurodytos trukmės embargo laikotarpį:
I am requesting an embargo of this thesis for the period indicated below:

☐ _____ mėnesių / *months*
(embargo laikotarpis negali viršyti 60 mėn. / *an embargo period shall not exceed 60 months*).

☒ Embargo laikotarpis nereikalingas / *no embargo requested.*

Embargo laikotarpio nustatymo priežastis / *Reason for embargo period*:

TURINYS

PAVEIKSLŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

SANTRUMPŲ SĄRAŠAS

GAE - *Google App Engine*;

IT *(angl. Information Technology)* – Informacinės technologijos;

PaaS *(angl. Platform as a Service)* – platforma kaip paslauga;

IaaS *(angl. Infrastructure as a Service)* – infrastruktūra kaip paslauga;

SaaS *(angl. Software as a Service)* – programinė įranga kaip paslauga;

CRM *(angl. Customer Relationship Management)* – klientų santykių valdymo sistema;

ERP *(angl. Enterprise Resource Planning)* – įmonių išteklių planavimo sistema;

API *(angl. Application Programming Interface)* – aplikacijų programavimo sąsaja;

DoS *(angl. Denial of Service)* – paslaugos atsisakymo ataka;

GCP *(angl. Google Cloud Platform)* - *Google* debesų kompiuterijos platforma;

SQL *(angl. Structured Query Language)* – struktūrizuota užklausų kalba;

NoSQL *(angl. Not Only SQL)* – ne tik SQL;

CPN *(angl. Coloured Petri Nets)* – spalvotieji *Petri* tinklai;

JVM *(angl. Java Virtual Machine)* – Java virtualioji mašina;

JIT *(angl. Just-In-Time)* – dinaminis kompiliavimas vykdymo metu;

UPPAAL *(angl. Universal Parallel Processing Architecture and Automated Language)* – universalus lygiagrečių procesų architektūros ir automatizacijos įrankis;

IoT *(angl. Internet of Things)* – daiktų internetas;

XML *(angl. eXtensible Markup Language)* – plečiamosios žymos kalba;

CPU *(angl. Central Processing Unit)* – centrinis procesorius;

HTTP *(angl. Hypertext Transfer Protocol)* – hiperteksto perdavimo protokolas;

MB *(angl. Megabyte)* – megabaitas;

AWS *(angl. Amazon Web Services)* – *Amazon* debesijos paslaugos;

SSH *(angl. Secure Shell)* – saugusis ryšio protokolas;

gRPC *(angl. Google Remote Procedure Call)* – nuotolinės procedūros iškvietimas;

GB *(angl. Gigabyte)* – gigabaitas;

RAM *(angl. Random Access Memory)* – operatyvioji atmintis;

TCP/IP *(angl. Transmission Control Protocol/Internet Protocol)* – perdavimo valdymo protokolas / interneto protokolas;

TLS *(angl. Transport Layer Security)* – transporto lygmens saugumo protokolas;

ALTS *(angl. Application Layer Transport Security)* – „Google“ sukurtas saugumo protokolas, skirtas užtikrinti duomenų šifravimą ir autentifikaciją tarp „Google Cloud“ paslaugų;

IAM *(angl. Identity and Access Management)* – tapatybės ir prieigos valdymas;

KMS *(angl. Key Management Service)* – raktų valdymo paslauga;

CDN *(angl. Content Delivery Network)* – turinio perdavimo tinklas;

DDoS *(angl. Distributed Denial of Service)* – paskirstytas paslaugų trikdymas;

LRU *(angl. Least Recently Used)* – mažiausiai naudotas;

GWT *(angl. Google Web Toolkit)* – *Google* žiniatinklio įrankių rinkinys;

UML *(angl. Unified Modeling Language)* – vieninga modeliavimo kalba;

CLI *(angl. Command-Line Interface)* – komandinės eilutės sąsaja;

SDK *(angl. Software Development Kit)* – programinės įrangos kūrimo rinkinys;

vCPU *(angl. Virtual Central Processing Unit)* – virtualus centrinis procesorius;

RSS *(angl. Really Simple Syndication)* – paprasta informacijos prenumerata;

HTML *(angl. Hypertext Markup Language)* – hiperteksto žymėjimo kalba;

CSS *(angl. Cascading Style Sheets)* – pakopinių stilių aprašo kalba naudojama apibūdinti HTML dokumentų stilių;

JSON *(angl. JavaScript Object Notation)* – duomenų mainų formatas, pagrįstas paprasta teksto struktūra;

REST API *(angl. Representational State Transfer Application Programming Interface)* – programavimo sąsaja, paremta REST architektūros principais;

HMAC *(angl. Hash-based Message Authentication Code)* – maišos pagrindu sukurtas žinutės autentifikavimo kodas;

SHA-256 *(angl. Secure Hash Algorithm 256-bit)* – maišos algoritmas 256 bitų;

AES *(angl. Advanced Encryption Standard)* – pažangus šifravimo standartas;

CBC *(angl. Cipher Block Chaining)* – šifro blokų grandinės kriptografinis režimas;

ID *(angl. Identifier)* – identifikatorius;

KiB *(angl. Kibibyte)* – kibibaitas;

CSV *(angl. Comma-Separated Values)* – kableliu atskirtų reikšmių failas;

VPC *(angl. Virtual Private Cloud)* – virtualus privatus tinklas;

DARPA *(angl. Defense Advanced Research Projects Agency)* – gynybos pažangiųjų tyrimų projektų agentūra;

MIT *(angl. Massachusetts Institute of Technology)* – Masačusetso technologijos institutas;

ARPANET *(angl. Advanced Research Projects Agency Network)* – pirmasis paketinio duomenų perdavimo tinklas;

NIST (angl. *National Institute of Standards and Technology*) – JAV Nacionalinis standartų ir technologijų institutas;

SANTRAUKA

Debesų paslaugų programavimo sprendimų tyrimas naudojant Google App Engine aplinką

Šiame darbe nagrinėjama, kaip *Google App Engine* aplinkoje veikiantys papildomi saugos sprendimai, duomenų šifravimas (*Fernet*) ir autentifikacija (*Firebase Authentication*), daro įtaką *Memcached* naudojančių aplikacijų našumui. Teorinėje dalyje analizuojama debesų kompiuterijos raida, PaaS modelio ypatumai bei GAE architektūra, išryškinamos pagrindinės talpyklos (*Memcached*) ir saugumo komponentų integravimo galimybės. Empiriniuose tyrimuose vertinamas šifravimo ir autentifikacijos poveikis atsako laikui, sistemos stabilumui bei išteklių (CPU, atminties) naudojimui, pasitelkiant apkrovos testavimą (*Locust*) skirtingomis sąlygomis. Tyrimo rezultatai parodė, kad *Fernet* šifravimas, nors ir nežymiai pailgina atsako laiką bei padidina atminties poreikį, *Google App Engine* automatinio mastelio keitimo mechanizmai užtikrina, jog aplikacijų stabilumas išliktų priimtinas net intensyvios apkrovos metu. *Firebase Authentication* taip pat turi menką neigiamą įtaką našumui, tačiau didesnėmis apkrovomis *Google App Engine* geba efektyviai paskirstyti resursus ir kompensuoti papildomą autentifikacijos apkrovą. Abejais atvejais talpyklos „pataikymų“ santykis išliko artimas 100 %, o klaidų dažnis – labai žemas. Šios išvados rodo, kad *Google App Engine* standartinėje aplinkoje įmanoma sėkmingai suderinti duomenų saugumą ir pakankamai aukštą sistemų našumo lygį, tinkamai konfigūruojant resursus bei atsižvelgiant į infrastruktūros automatinio mastelio keitimo savybes.

SUMMARY

Research on programming solutions for cloud services using the Google App Engine environment

This work examines how additional security measures, data encryption (*Fernet*) and authentication (*Firebase Authentication*), affect the performance of applications that utilize Memcached within the *Google App Engine* environment. The theoretical part explores the evolution of cloud computing, the characteristics of the *Platform as a Service* model, and the *Google App Engine* architecture, highlighting the key role of *Memcached* and the possibilities for integrating security components. The empirical research evaluates the impact of encryption and authentication on response time, system stability, and resource consumption (CPU and memory) by employing load testing *(Locust*) under varying conditions. The study's findings reveal that although *Fernet* encryption slightly increases response times and memory usage, GAE's automatic scaling mechanisms ensure that application stability remains acceptable even under heavy loads. *Firebase Authentication* similarly has a minor negative effect on performance, but at higher loads, GAE effectively allocates resources to mitigate the additional authentication overhead. In both cases, the *Memcached* hit ratio remained close to 100%, and the error rate was very low. These conclusions indicate that, with proper resource configuration and leveraging GAE's automated scaling features, it is possible to balance data security and maintain high levels of system performance in GAE's standard environment.

ĮVADAS

Temos aktualumas. Debesų kompiuterijos technologijos yra vienas svarbiausių skaitmeninės transformacijos veiksnių, suteikiančių naujų galimybių IT infrastruktūros ir programinės įrangos valdymui. Platforma kaip paslauga (PaaS) modelis leidžia kūrėjams sutelkti dėmesį į programų kūrimą, išvengiant infrastruktūros administravimo užduočių. *Google App Engine* (GAE) yra vienas pažangiausių PaaS sprendimų, automatizuoja išteklių valdymą, užtikrina sistemos elastingumą ir palengvina sudėtingų aplikacijų kūrimą. Šis darbas yra aktualus dėl debesų kompiuterijos sprendimų populiarumo augimo, o ypač dėl būtinybės geriau suprasti GAE programavimo sprendimus ir jų poveikį sistemų našumui bei saugumui.

Tiriamoji problema: kaip *Google App Engine* aplinkoje taikomi programavimo sprendimai veikia sistemų našumą ir saugumą bei kaip galima optimizuoti jų taikymą?

Darbo objektas: *Google App Engine* aplinkoje taikomi programavimo sprendimai ir jų poveikis sistemų našumui bei saugumui.

Darbo tikslas: įvertinti *Google App Engine* aplinkoje taikomų programavimo sprendimų poveikį sistemų našumui ir saugumui, siekiant nustatyti optimalius jų pritaikymo būdus.

Darbo uždaviniai.

1. Išanalizuoti mokslinę literatūrą apie debesų kompiuterijos technologijas ir PaaS modelio taikymo galimybes;
2. Išanalizuoti mokslinę literatūrą apie *Google App Engine* aplinką, jos architektūrą, komponentus ir funkcionalumą;
3. Ištirti, kaip papildomi saugos sprendimai veikia sistemų našumą ir duomenų saugumą;
4. Atlikti eksperimentinį tyrimą, kuriame bus lyginami kontrolinė ir eksperimentinė aplikacijų grupės, naudojant *Google App Engine* aplinką;
5. Apibendrinti tyrimo rezultatus, įvertinti *Google App Engine* programavimo sprendimų efektyvumą bei pateikti rekomendacijas dėl jų praktinio pritaikymo.

Tyrimo metodai: mokslinės literatūros analizė ir apibendrinimas, projektavimas ir realizavimas, kiekybinis (kontroliuojamas) eksperimentas.

Darbe taikyti programinės realizacijos kūrimo metodai, naudotos priemonės ir įrankiai. Darbe buvo taikyti įvairūs programinės realizacijos kūrimo metodai, priemonės ir įrankiai, užtikrinantys sistemų kūrimo efektyvumą ir tyrimo tikslų pasiekimą. Reikalavimų specifikacijai sudaryti naudota *Microsoft Word* programa, o sistemų projektavimui – UML kalba ir *MagicDraw*

įrankis, leidžiantys tiksliai apibrėžti funkcionalumą per panaudos atvejų, veiklos ir diegimo diagramas. Realizacijos metu sukurtos keturios *Python 3.9* pagrįstos aplikacijų versijos, pasitelkiant *Flask* karkasą, *Fernet* šifravimo biblioteką ir *Firebase Authentication* autentifikacijos sprendimą. Aplikacijos buvo diegiamos GAE standartinėje aplinkoje, o *Memorystore Memcached* talpykla užtikrino efektyvią duomenų saugojimo architektūrą. Testavimo procesas apėmė *Locust* įrankio naudojimą apkrovos scenarijų kūrimui ir vykdymui, automatizavimą per *PowerShell* scenarijus bei duomenų analizę su *Excel* ir *Knime*. Sukurta infrastruktūra ir automatizuoti sprendimai leido atlikti detalią sistemų našumo analizę, įvertinti šifravimo bei autentifikacijos poveikį atsako laikui, resursų naudojimui ir sistemos stabilumui, tuo pačiu užtikrinant tyrimo rezultatų tikslumą ir patikimumą.

Glausta gautų rezultatų apžvalga, reikšmingumas. Tyrimas parodė, kad papildomi saugos sprendimai – *Fernet* šifravimas ir *Firebas*e autentifikacija – turi nedidelį poveikį aplikacijų našumui *Google App Engine* standartinėje aplinkoje. Vidutinis atsako laikas šiek tiek išaugo, tačiau sistema išliko stabili net esant didelėms apkrovoms. Automatinio mastelio keitimo mechanizmai užtikrino efektyvų resursų valdymą, o talpyklos efektyvumas išliko aukštas. Rezultatai patvirtina, kad šie saugos mechanizmai gali būti sėkmingai pritaikomi, derinant našumo ir saugumo poreikius.

Darbo apribojimai ir sunkumai. Darbo eigoje buvo susidurta su keliais apribojimais ir sunkumais, kurie reikalavo papildomo dėmesio ir pastangų. Didžiausias iššūkis buvo tyrimo dalis, kuriai įgyvendinti reikėjo daug laiko ir kruopštumo. Ypač sudėtingas buvo duomenų suvienodinimo procesas – užtikrinant, kad visi gauti rezultatai būtų nuoseklūs ir tinkami kokybiškai analizei. Šis etapas reikalavo ne tik techninių įgūdžių, bet ir atidumo detalėms, siekiant pašalinti neatitikimus ir užtikrinti duomenų patikimumą. Papildomų iššūkių kėlė ir duomenų interpretavimas, kadangi gautos metrikos buvo gausios ir įvairios, o jų analizė reikalavo aiškumo, tikslumo ir tinkamo konteksto. Suprasti duomenų ryšius bei pagrįsti išvadas, pagrindžiant jas tyrimo metodologija, taip pat pareikalavo daug laiko ir pastangų.

Darbo loginės struktūros paaiškinimas: darbe struktūra suskirstyta į keturias dalis, kurios nuosekliai atspindi tyrimo eigą nuo teorinės analizės iki praktinio įgyvendinimo ir rezultatų interpretacijos. Analitinėje dalyje nagrinėjamos *Google App Engine* aplinkos pritaikymo galimybės, akcentuojant architektūrinius sprendimus, programavimo ypatumus ir integruotų saugumo mechanizmų poveikį sistemų našumui bei duomenų apsaugai. Aptariami GAE architektūros ypatumai, talpyklos sprendimų efektyvumas, hibridinių debesijos modelių integracija ir papildomų saugos priemonių, tokių kaip šifravimas ir autentifikacija, įtaka veikimui. Šis skyrius suteikia teorinį pagrindą tolesniems tyrimams. Projektinėje dalyje detaliai pristatomas tyrimo planas, įskaitant internetinių svetainių, skirtų eksperimentiniams tyrimams, specifikacijų kūrimą bei jų realizacijos procesą, iliustruojamą UML diagramomis. Ši dalis užtikrina tyrimo metodo aiškumą ir padeda

efektyviai suplanuoti tolimesnius etapus. Realizavimo skyriuje aptariamos techninės ir programinės įrangos pasirinkimo priežastys bei jų pritaikymas tyrimo įgyvendinimui. Analizuojamas aplikacijų diegimas *Google App Engine* aplinkoje, apkrovos testavimo procesai su *Locust* įrankiu ir testavimo automatizavimas naudojant *PowerShell* scenarijus. Taip pat aprašomi saugumo sprendimai, tokie kaip šifravimo bibliotekos *Fernet* ir *Firebase Authentication* naudojimas. Ši dalis akcentuoja praktinių sprendimų efektyvumą eksperimente. Tyrimo eigos aprašymo dalis skirta įvertinti, kaip šifravimas ir autentifikacija veikia *Memcached* naudojančių aplikacijų našumą GAE standartinėje aplinkoje.

Svarbiausia naudota literatūra. Šiame darbe buvo remtasi šaltiniais, apimančiais mokslinę literatūrą, recenzuotus straipsnius, techninę dokumentaciją bei internetinius informacijos šaltinius.

Duomenys apie sukurtos sistemos įdiegimą. Tyrimui įgyvendinti sukurtos aplikacijų versijos buvo diegiamos *Google App Engine* standartinėje aplinkoje. Jos yra prieinamos šiais adresais: https://tyrimas.uc.r.appspot.com, https://encrypt-app-dot-tyrimas.uc.r.appspot.com, https://control-app-v2-dot-tyrimas.uc.r.appspot.com ir https://firebase-auth-app-dot-tyrimas.uc.r.appspot.com.

Darbo struktūra ir apimtis: Darbo apimtis – 67 puslapiai, 25 paveikslėliai, 2 lentelės, 21 priedas.

1. ANALITINĖ DALIS

Šiame skyriuje bus nagrinėjama *Google App Engine* (GAE) aplinkos pritaikymo galimybės, vertinant jos architektūros sprendimus, programavimo sprendimus bei integruotų saugumo priemonių poveikį sistemų našumui ir duomenų apsaugai. Atsižvelgiant į teorinę analizę ir empirinius tyrimus, bus aptarti tokie aspektai kaip GAE aplinkos architektūriniai ypatumai, talpyklos sprendimų efektyvumas, hibridinių debesijos sprendimų integracija bei papildomų saugumo mechanizmų poveikis našumui.

1.1. Debesų kompiuterijos raida

Debesų kompiuterija simbolizuoja ne tik ilgalaikį technologinį progresą, bet ir IT infrastruktūrų evoliuciją. Ši technologija remiasi tinklų vystymusi, virtualizacijos pažanga ir didelio masto duomenų centrų plėtra. Virtualizacija, leidžianti vienu metu keliems virtualiems serveriams veikti viename fiziniame įrenginyje, tapo pagrindiniu debesų kompiuterijos aspektu. Kompiuterių tinklų modernizacija, įskaitant spartesnius interneto ryšius ir padidėjusį prieinamumą, leido šiai technologijai sparčiai vystytis. Būtina paminėti be serverines (angl. *serverless*) technologijas, kurios paskatino debesų kompiuterijos naudojimą optimizuojant resursus bei mažinant infrastruktūros valdymo kaštus. [1] Be serverinė technologija yra debesijos kompiuterijos modelis, leidžiantis kūrėjams vystyti ir vykdyti programas, nesirūpinant serverių administravimu. Debesijos paslaugų teikėjas atsako už serverių paruošimą, priežiūrą ir resursų dydžio keitimą. Programos be serverinėje aplinkoje automatiškai pritaiko savo veiklos apimtis pagal poreikį, didindamos ar mažindamos resursų naudojimą. Šios paslaugos, teikiamos viešųjų debesų tiekėjų, dažniausiai apmokestinamos pagal poreikį, remiantis įvykių valdomo vykdymo modeliu. Tai reiškia, kad nevykdomos be serverinės funkcijos išlaidų nesukelia. [2] Tokia technologinė paradigma atveria naujas galimybes įmonėms, leisdama joms būti lankstesnėms ir greitesnėms atsakyti į kintančią rinkos aplinką, kartu sumažinant operacinių išlaidų naštą. Be serverinė technologija ne tik efektyvina resursų panaudojimą, bet ir skatina inovacijas, suteikdama kūrėjams galimybę sutelkti dėmesį į produktų kūrimą ir tobulinimą, o ne infrastruktūros valdymą. Taigi, debesijos kompiuterija, ypač jos be serverinės formos, yra svarbi skaitmeninės transformacijos ir technologinio progreso varomoji jėga. [3]

Apibendrinant debesų kompiuterijos evoliucija nuo ankstyvųjų eksperimentų iki plataus masto komercinių sprendimų parodė, kad ši technologija ne tik palengvina prieigą prie skaičiavimo išteklių, bet ir atveria naujas verslo galimybes. Tiek istorinis debesų kompiuterijos vystymasis, tiek šiuolaikinės paslaugos pabrėžia šios technologijos svarbą verslui. Svarbu, kad organizacijos tinkamai

įvertintų savo poreikius ir pasirinkę tinkamą paslaugų modelį efektyviai išnaudotų debesų kompiuterijos teikiamas galimybes. Plačiau apie debesų kompiuterijos istoriją, vystymosi etapus ir įtaką technologijų bei verslo aplinkai skaitykite **1 PRIEDASE**.

1.2. Debesų kompiuterija

Debesų kompiuterija (angl. *Cloud Computing*) yra pažangi informacinių technologijų paradigma, suteikianti galimybę vartotojams nuotoliniu būdu naudotis skaičiavimo ištekliais, duomenų saugyklomis bei programine įranga, nereikalaujant vietinių resursų naudojimo. Ši technologija remiasi kompiuterinių išteklių virtualizacija ir jų pateikimu kaip paslauga internetu. Debesų kompiuterija leidžia vartotojams lanksčiai keisti naudojamų išteklių apimtis pagal poreikius. Ji taip pat sumažina trečiųjų šalių techninės priežiūros ir valdymo poreikį. [4]

Apibendrinant **2 PRIEDASE** analizuojamą debesų kompiuterijos plėtrą, jos technologinius ir rinkos aspektus, bei pateikiamą aiškią sistemą apie paslaugų ir infrastruktūros modelius, galime teigti, jog debesų kompiuterijos plėtra atspindi esminius šiuolaikinių technologijų pokyčius ir jų poveikį verslui bei kasdieniam naudojimui. NIST apibrėžimai ir rinkos analizės duomenys rodo, kad debesų paslaugos tampa vis labiau integruotinos ir įvairesnės, suteikdamos organizacijoms būtiną lankstumą ir elastingumą šiuolaikinėje rinkoje. Dėl šių savybių, tokių kaip savitarna pagal poreikį, platus tinklo prieinamumas ir resursų telkimas, debesų kompiuterija tampa puikia priemone, leidžiančia efektyviai spręsti kintančias rinkos sąlygas ir technologinius iššūkius. Augantis rinkos dydis ir prognozės rodo, kad debesų kompiuterijos svarba tik augs, o tai lems naujų verslo strategijų ir operacinių modelių kūrimą. Taigi, debesų kompiuterija yra ne tik technologinė naujovė, bet ir svarbi šiuolaikinio verslo bei technologijų aplinkos dalis, kurią suprantant ir taikant galima pasiekti didesnį veiklos efektyvumą bei konkurencinį pranašumą.

1.3. PaaS (*Platform as a Service*) modelis

Platforma kaip paslauga (PaaS) yra vienas iš debesų kompiuterijos modelių, kuris teikia kūrimo aplinką su virtualiomis platformomis ir integruotais įrankiais. Skirtingai nuo kitų debesų paslaugų modelių, tokių kaip infrastruktūra kaip paslauga (IaaS), kuri teikia tik pagrindinę infrastruktūrą, ir programinė įranga kaip paslauga (SaaS), kuri siūlo galutines programinės įrangos paslaugas vartotojams, PaaS suteikia kūrėjams visą ekosistemą, skirtą programų kūrimui, testavimui,

diegimui ir valdymui. PaaS modelis apima infrastruktūrą, tačiau taip pat teikia platformos valdymo įrankius, duomenų bazes, serverius, saugumo sprendimus ir kitas priemones, kurios supaprastina ir automatizuoja programų kūrimo procesą, todėl kūrėjai gali susitelkti tik į programų kūrimą ir valdymą. [5] [6] [7] Pavyzdžiui, naudojant *Google App Engine*, kūrėjams suteikiama prieiga prie aukšto lygio kūrimo aplinkų ir įrankių, kuriuos naudojant nereikia rūpintis infrastruktūros valdymu ar programų priežiūra [8].

Vienas didžiausių PaaS modelio privalumų yra infrastruktūros valdymo ir konfigūravimo paprastumas**.** Vartotojai gali koncentruotis į savo programas, o visa infrastruktūros priežiūra (serveriai, tinklai, saugumo mechanizmai) yra patikėta paslaugų teikėjui. Tai leidžia sumažinti laiko sąnaudas, nes kūrėjai išvengia papildomų darbų, tokių kaip serverių nustatymų valdymas ar tinklo priežiūra. [5] [8]. Pavyzdžiui, *Google App Engine* suteikia galimybę diegti programas be tiesioginės sąveikos su serveriais. [6] Greitesnis programų kūrimo procesas yra dar vienas svarbus privalumas. PaaS platformos siūlo integruotus įrankius ir šablonus, kurie padeda programuotojams kurti ir testuoti prototipus greičiau nei naudojant tradicinius metodus. Tai leidžia įmonėms greičiau pateikti rinkai naujas programas ir paslaugas, o *Google App Engine* automatinis diegimo ir testavimo funkcionalumas turi dar labiau supaprastintą šį procesą. [7] [9] Be to, PaaS modeliai pasižymi automatiniu išteklių valdymu ir didinimu. Šios platformos automatiškai paskirsto išteklius priklausomai nuo programos apkrovos, todėl, kai srautas auga, platforma automatiškai priskiria daugiau resursų. Tai leidžia optimizuoti darbą be papildomo vartotojo įsikišimo. [8] [9]. Ši funkcija ypač naudinga naudojant *Google App Engine*, nes automatinis išteklių valdymas užtikrina didelį programų našumą net esant intensyviam vartotojų srautui. Kitas esminis privalumas yra ekonomiškumas ir efektyvumas**.** Kadangi įmonėms nereikia investuoti į savo serverius ar IT personalą jų valdymui, PaaS tampa patraukliu sprendimu, kuris sumažina infrastruktūros išlaidas. [5] [7] Papildomą vertę teikia papildomų paslaugų integracija su kitais debesų kompiuterijos sprendimais. PaaS platformos dažnai turi integracijas su įvairiomis trečiųjų šalių paslaugomis, duomenų bazėmis ir API, kurios leidžia greitai ir efektyviai sąveikauti su išorinėmis sistemomis. Pavyzdžiui, *Google App Engine* turi integraciją su kitais savo ekosistemoje esančiais sprendimais ir gali sąveikauti su *Google Cloud SQL* ar *Cloud Storage* paslaugomis, kurios suteikia dar daugiau funkcionalumo. [5] [8] Galiausiai, PaaS suteikia prieigą prie naujausių technologijų ir inovacijų, nes platformos yra nuolat atnaujinamos. Tai leidžia programuotojams dirbti su moderniausiomis technologijomis, įrankiais ir saugumo standartais, išvengiant infrastruktūros senėjimo ar saugumo problemų. [7] [9]

Nepaisant įvairių privalumų, PaaS modelis turi ir trūkumų. Vienas iš pagrindinių – priklausomybė nuo paslaugų teikėjo**.** Kadangi infrastruktūra valdoma iš išorės, įmonės tampa

priklausomos nuo paslaugų teikėjo techninių sprendimų, kainodaros politikos bei patikimumo. Jei teikėjas patiria trikdžių ar keičia kainas, vartotojai gali susidurti su papildomais iššūkiais. [5] [7] Taip pat PaaS teikėjai dažnai riboja galimybes individualiai pritaikyti infrastruktūrą, o tai gali sukelti iššūkių tam tikrų specifinių programų ar paslaugų diegimui. Pavyzdžiui, *Google App Engine* vartotojai susiduria su ribotomis serverių nustatymo galimybėmis. [9] [7] Labai svarbus aspektas yra saugumas bei privatumas. Vartotojai turi pasikliauti paslaugų teikėjo saugumo politika. Kadangi infrastruktūra ir duomenys yra valdomi išorinės įmonės, tai gali sukelti papildomų rūpesčių dėl jautrių duomenų apsaugos. Todėl kiekvieną kartą renkantis paslaugos teikėją, vartotojas turi labai atidžiai įsivertinti visas galimas rizikas. [9] [6] Ne ką mažesnės problemos yra susijusios su migracijomis. Problema dažniausiai iškylą tuomet, kai vartotojas nori pereiti iš vienos PaaS platformos į kitą. Migracija gali tapti komplikuota dėl priklausomybės nuo specifinių API, karkasų ar programavimo kalbų, naudojamų paslaugos teikėjų platformose. [7] [8]

Kalbant apie PaaS pritaikymo sritis, tai viena jų gali būti **v**erslo aplikacijų kūrimas ir SaaS sprendimai**.** Įmonės gali lengvai kurti ir diegti tokias programas kaip CRM, finansų valdymo sistemas ar ERP, naudojant PaaS platformas. *Google App Engine* suteikia puikią infrastruktūrą tokiems sprendimams diegti. [5] [7] Taip pat PaaS yra ypač naudingas prototipų kūrimui ir testavimui**.** Kadangi kūrėjai gali greitai sukurti ir diegti programų prototipus be ilgo infrastruktūros konfigūravimo, tai puikus pasirinkimas vartotojams, norintiems greitai testuoti naujas idėjas. [9] [8] Be to, PaaS plačiai naudojamas internetinių ir mobiliųjų aplikacijų kūrimui. *Google App Engine* platforma suteikia kūrimo karkasus, skirtus programoms, veikiančioms žiniatinklyje ar mobiliuose įrenginiuose, su integruotu mastelio keitimu ir aukšto našumo užtikrinimu. [8] [9] Didžiųjų duomenų analizė ir IoT sprendimai taip pat dažnai įgyvendinami naudojant PaaS. Pavyzdžiui, *Google Cloud* platformos sprendimai, tokie kaip *Cloud Datastore* ar *BigQuery*, integruojasi su PaaS ir leidžia efektyviai valdyti didelius duomenų kiekius bei juos analizuoti. [5] [7]

Vieni populiariausių PaaS tiekėjų yra *Heroku, Google App Engine, DigitalOcean App Platform, AWS Elastic Beanstalk, Microsoft Azure App Service*. [10] Detali šių PaaS sprendimų apžvalga, įskaitant jų savybes, privalumus, trūkumus bei populiarumo analizę Lietuvos rinkoje, pateikta **3 PRIEDASE**.

PaaS modelis suteikia reikšmingų privalumų, tokių kaip supaprastintas infrastruktūros valdymas, greitesnis programų kūrimo procesas ir ekonomiškumas. Tačiau vartotojai turi atidžiai įvertinti priklausomybę nuo paslaugų teikėjo, saugumo ir privatumo rizikas, bei galimą migracijos sudėtingumą. Lietuvoje Google debesijos paslaugos išsiskiria kaip dominuojančios, užimančios didžiausią rinkos dalį tarp PaaS tiekėjų. Tuo tarpu kiti tiekėjai, tokie kaip *Heroku, DigitalOcean, AWS*

ir *Azure*, turi mažesnę rinkos dalį, tačiau vis tiek gali būti tinkami pasirinkimai priklausomai nuo specifinių poreikių ir projektų reikalavimų.

1.4. Vertinimo kriterijai

Vertinant debesijos paslaugų platformas, svarbu remtis aiškiai apibrėžtais kriterijais, leidžiančiais įvertinti jų tinkamumą ir efektyvumą skirtingose situacijose. Šie kriterijai yra atrinkti remiantis moksline literatūra ir debesijos kompiuterijos srities tyrimais. Jie padeda nuosekliai įvertinti PaaS sprendimus, tokius kaip GAE, atsižvelgiant į įvairius technologinius ir veikimo aspektus. Tarp atrinktų vertinimo kriterijų yra architektūra, palaikomos aplinkos, našumas, saugos sprendimai ir kombinuoti sprendimai.

Architektūra yra esminis vertinimo kriterijus dėl platformos sudėtingumo ir paslėptų išteklių riboto matomumo. PaaS platformų architektūros sudėtingumas kyla dėl daugybės skaičiavimo mazgų, apkrovos balansavimo ir duomenų skaidymo mechanizmų, kurie veikia kaip bendra infrastruktūra. Dėl šios sudėtingos paslaugų architektūros tampa sunku tiksliai numatyti programų veikimo kokybę, naudojant esamus modeliavimo ir našumo vertinimo įrankius. Architektūra turi būti vertinama ankstyvose projektavimo stadijose, kad būtų galima išvengti galimų kokybės pažeidimų, tokių kaip prastesnis pralaidumas, ilgesnis atsako laikas ir ribotos mastelio keitimo galimybės. Tinkamas architektūros įvertinimas padeda užtikrinti efektyvų sistemos veikimą ir prisideda prie sėkmingo debesijos sprendimų pritaikymo. [11]

Palaikomos aplinkos taip pat yra kritiškas vertinimo kriterijus, kadangi jų pasirinkimas daro tiesioginę įtaką programų efektyvumui, našumui ir išteklių valdymui. Vertinant GAE platformą, svarbu atsižvelgti į galimybę naudoti tiek nemokamus, tiek mokamus išteklius, o tai yra reikšmingas aspektas daugeliui projektų, ypač tiems, kurie siekia išnaudoti nemokamas debesijos galimybes optimaliam programų vykdymui. Skirtingos GAE aplinkos – standartinė ir lanksčioji – suteikia kūrėjams įvairias konfigūravimo galimybes bei skirtingus išteklių paskirstymo modelius, o tai leidžia optimizuoti išlaidas ir pasiekti geriausius veikimo rezultatus. Todėl aplinkos pasirinkimas tampa lemiamas veiksnys siekiant efektyvaus resursų valdymo ir stabilaus veikimo, ypač augant projektų sudėtingumui. [12]

Našumas yra vienas svarbiausių kriterijų, vertinant bet kurią platformą ar sistemą, nes jis tiesiogiai veikia aplikacijų efektyvumą, vartotojo patirtį ir bendrą sistemos funkcionalumą. Aukštas našumas reiškia, kad platforma gali apdoroti daugiau užklausų per trumpesnį laiką, sumažinti užlaikymus ir efektyviau išnaudoti išteklius, užtikrindama optimalų programų veikimą net esant

dideliam apkrovimui. Tai ypač svarbu tais atvejais, kai paslaugų kokybė priklauso nuo greito duomenų apdorojimo arba greitos reakcijos į vartotojo veiksmus. Be to, našumas leidžia sistemoms efektyviai prisitaikyti prie kintančių apkrovos lygių, užtikrinant sklandų veikimą ne tik kasdieninėmis sąlygomis, bet ir piko metu. Vertinant našumą, galima įvertinti, kaip sistema tvarkosi su ištekliais, įskaitant mokamus ir nemokamus resursus, ir kiek ji geba išlaikyti optimalią veikimo kokybę plečiantis projektams ar augant duomenų kiekiui. [12]

Saugos sprendimai yra būtinas kriterijus vertinant debesų kompiuterijos platformas, nes debesijos infrastruktūra ir jos architektūra kelia specifinius saugumo iššūkius, kuriems tradiciniai sprendimai nebūtinai yra tinkami. Tyrimai apie GAE atsparumą DoS atakoms rodo, kad debesijos platformos gali būti pažeidžiamos tam tikrų kibernetinių grėsmių, galinčių sutrikdyti paslaugų teikimą. Saugos sprendimų vertinimas yra ypač svarbus norint užtikrinti sistemos patikimumą, paslaugų tęstinumą ir apsaugą nuo galimų grėsmių, o tai yra itin aktualu komercinėms ir didelio masto sistemoms, kur net trumpalaikiai sutrikimai gali turėti reikšmingų finansinių pasekmių. [13]

Kombinuoti sprendimai debesų kompiuterijoje apima kelių skirtingų technologijų ir paslaugų integravimą, siekiant užtikrinti sistemų patikimumą, mastelio keitimą ir efektyvumą. Pavyzdžiui, *Google App Engine* kartu su *Google Datastore* ir *BigTable* sudaro tvirtą infrastruktūrą paskirstytoms sistemoms, kurios suteikia galimybę efektyviai saugoti ir apdoroti duomenis debesyje. Toks technologijų derinimas leidžia ne tik valdyti didelius duomenų kiekius, bet ir užtikrina jų prieinamumą bei patikimumą net esant didelėms vartotojų apkrovoms ar sistemų trikdžiams. Integruojant tokias paslaugas kaip virtualizacija ir debesijos išteklių paskirstymas, kombinuoti sprendimai leidžia sukurti dinamiškas ir saugias aplikacijas, kurios gali lengvai prisitaikyti prie kintančių poreikių. Tai yra ypač svarbu sparčiai besikeičiančiose IT infrastruktūrose, kur patikimumas ir sistemos atsparumas trikdžiams yra prioritetiniai. [14]

Šių kriterijų pasirinkimas leidžia išsamiai įvertinti GAE galimybes, atsižvelgiant į jos architektūros sudėtingumą, aplinkų pasiūlą, našumo charakteristikas, saugumo užtikrinimo priemones ir integracijos galimybes su kitomis paslaugomis. Tai suteikia pagrindą nuodugniai analizuoti, kaip GAE atitinka šiuos kriterijus ir kokie yra jos privalumai bei trūkumai, palyginti su kitomis debesijos platformomis.

1.5. GAE architektūra ir komponentai

GAE yra *Google Cloud Platform* dalis, skirta kurti ir talpinti žiniatinklio bei mobiliąsias programas, naudojant debesų kompiuterijos išteklius. Ši platformos kaip paslaugos (PaaS)

technologija suteikia galimybę kurti plečiamas aplikacijas, kurios automatiškai prisitaiko prie vartotojų užklausų srautų. [15] [16] Tai ypač svarbu aplikacijoms, aptarnaujančioms daugybę vartotojų vienu metu, nes GAE platforma automatiškai reguliuoja serverio pajėgumus pagal apkrovą. [16] Ši architektūra naudojama daugelyje darbų dėl savo patikimumo, paplitimo rinkoje. GAE yra visiškai valdoma, be serverinė platforma, kuri automatizuoja serverių valdymą ir resursų priskyrimą pagal poreikius, taip suteikdama galimybę kūrėjams susitelkti į programinės įrangos kūrimą, o ne į serverių ir infrastruktūros priežiūrą. [16] [15]

Vienas iš pagrindinių GAE architektūros komponentų – *Google Load Balancer*. Tai yra apkrovos balansavimo mechanizmas, skirtas valdyti aplikacijų apkrovą ir užtikrinti tolygų vartotojų užklausų paskirstymą tarp aplikacijos instancijų, taip padidinant paslaugų patikimumą ir našumą. [17] *Google Cloud Load Balancer* palaiko tiek globalų, tiek regioninį balansavimą. Globalus balansavimas paskirsto apkrovas tarp įvairių geografinių regionų, užtikrinant didesnį paslaugų prieinamumą ir greitą atsakymą į užklausas. [18] Naudojant vieną *anycast* IP adresą, srautas paskirstomas tarp skirtingų regionų, o esant gedimui, jis automatiškai nukreipiamas į sveikus atsarginius serverius, taip užtikrinant aukštą paslaugų prieinamumą. [17] Šis balansavimo mechanizmas taip pat leidžia skirstyti apkrovas pagal serverių pajėgumus, optimizuojant sistemos našumą. [18] Daugiau apie *Google Load Balancer* veikimo principus ir technologijas pateikta **4 PRIEDASE**.

Dar vienas svarbus komponentas – *Memcache*. Tai yra spartus talpyklos mechanizmas, skirtas laikinai saugoti dažnai naudojamus duomenis. [19] Šis komponentas padeda sumažinti užklausų apdorojimo laiką, nes dažniausiai reikalingi duomenys gali būti pateikiami tiesiai iš atminties, o ne atliekant papildomas duomenų bazės užklausas. [20] *Memcache* veikia kaip buferis, mažinantis pagrindinės duomenų saugyklos (angl. *datastore*) apkrovą, todėl sistema veikia efektyviau. Ši bendroji talpykla yra pasidalijama tarp skirtingų GAE instancijų, taip padidinant bendrą sistemos našumą ir pagreitinant atsakymus į užklausas. GAE palaiko du *Memcache* paslaugų lygius. Pirmasis – *Shared Memcache* – yra nemokamas ir suteikia talpyklos resursus pagal galimybes, atsižvelgiant į bendrą visų GAE aplikacijų, naudojančių šią paslaugą, apkrovą. Antrasis – *Dedicated Memcache* – yra mokamas ir užtikrina iš anksto nustatytą talpyklos vietą, skirtą tik konkrečiai aplikacijai. Šis sprendimas padeda pasiekti nuoseklesnį veikimą, sumažinant duomenų užklausų iš pagrindinės saugyklos poreikį. Tačiau *Memcache* turi ir tam tikrų apribojimų. Kadangi tai yra nepastovi atmintis, duomenys gali būti pašalinti (angl. *evicted*), jei talpykla persipildo arba sistema yra perkrauta. Tai reiškia, kad tam tikri duomenys gali būti prarasti ir turės būti užklausiami duomenų bazės dar kartą. [19] Kiekviena instancija gali pasinaudoti bendra *Memcache* erdve, o tai leidžia dažnai naudojamiems

duomenims būti greitai pasiekiamiems nepriklausomai nuo konkrečios instancijos, kuri apdoroja užklausą. [21]

Kadangi *Memcache* yra ribotas tik *Google App Engine* platformoje, svarbu paminėti, kad *Google Cloud Platform* siūlo dar vieną alternatyvų talpyklos sprendimą – *Memorystore for Memcached*. Ši paslauga yra visiškai valdoma, mastelį keičianti *Memcached* versija, skirta naudoti su įvairiomis *Google Cloud* platformos paslaugomis. Ji gali būti lengvai integruojama į esamas sistemas, dažniausiai nereikalaujant reikšmingų kodo pakeitimų. Ši paslauga suteikia galimybę kurti visiškai valdomus *Memcached* klasterius, kuriuos galima naudoti duomenų talpyklai įvairioms paskirtims, tokioms kaip dažnai naudojamų duomenų talpinimas, užklausų rezultatų talpinimas, sesijų saugojimas. [22] Detalesnė informacija apie *Memorystore for Memcached* pateikta **4 PRIEDASE**.

Į GAE architektūros komponentų sudėtį įeina *Cloud SQL* ir *Cloud Datastore*. *Cloud SQL* yra visiškai valdoma reliacinė duomenų bazės paslauga, palaikanti *MySQL, PostgreSQL* ir *SQL Server* sistemas. Ji suteikia lankstų duomenų bazės administravimą, leidžiantį užtikrinti patikimą našumą ir prieinamumą, ypač sudėtingesnėms darbo apkrovoms. Paslauga automatizuoja daugelį su duomenų baze susijusių procesų, įskaitant atsarginių kopijų kūrimą, replikavimą, šifravimą ir automatinį talpos padidinimą. [23] *Cloud Datastore* yra *NoSQL* dokumentų duomenų bazė, skirta automatiškam mastelio keitimui, aukštam našumui ir lengvam programų kūrimui. Ji palaiko atomines transakcijas, kurios užtikrina, kad visos operacijos būtų sėkmingai įvykdytos, arba, jei bent viena nepavyksta, nevyksta nė viena. Taip pat užtikrinamas stiprus nuoseklumas visoms užklausoms, o duomenys automatiškai šifruojami prieš juos įrašant į diską, o tai suteikia papildomo saugumo. [24]

GAE architektūra, kaip PaaS sprendimas, pasižymi gebėjimu efektyviai valdyti programų resursus ir užtikrinti aukštą paslaugų prieinamumą. Automatinis mastelio keitimas ir dinamiškas resursų paskirstymas leidžia GAE prisitaikyti prie besikeičiančių užklausų srautų, taip optimizuojant našumą ir mažinant infrastruktūros valdymo sudėtingumą. Detalesnė GAE architektūros komponentų apžvalga pateikta **4 PRIEDASE**. Ši architektūra suteikia galimybę kurti aplikacijas, kurios gali augti ir prisitaikyti prie didelio vartotojų skaičiaus be reikalo rūpintis serverių valdymu ar išankstiniu pasiruošimu. GAE išlaiko savo patikimumą ir efektyvumą net ir esant staigiems apkrovos pokyčiams, užtikrindama greitą reakciją ir mažą atsako laiko vėlinimą. Taigi, ši sistema leidžia kūrėjams susitelkti į aplikacijų kūrimą, o ne į infrastruktūros valdymą, o tai ypač svarbu moderniose, dinamiškose debesų kompiuterijos aplinkose.

1.6. *Google App Engine* aplinkos

GAE suteikia teise naudotis dvejomis aplinkomis – standartine ir lanksčiąja. [25]

GAE standartinė aplinka suteikia galimybę kūrėjams paleisti savo programinę įrangą *Google* valdomoje infrastruktūroje naudojant iš anksto paruoštus konteinerius su tam tikromis programavimo kalbų vykdymo aplinkomis (angl. *runtimes*), pvz., *Go, Java, Node.js, PHP, Python, Ruby*. Ši aplinka orientuota į paprastą diegimą, didelį patikimumą ir automatinį mastelio keitimą, kuris prisitaiko prie kintančių srautų. [26] Programos vykdomos saugioje smėlio dėžės (angl. *sandbox*) aplinkoje, leidžiančioje paskirstyti užklausas tarp kelių serverių, taip padidinant serverių skaičių pagal poreikį, kad būtų patenkinti padidėjęs srautas. standartinėje aplinkoje „Google“ prižiūri visą infrastruktūrą, todėl kūrėjai gali susitelkti tik į kodo rašymą, nereikia valdyti serverių ar virtualių mašinų [26]. Tačiau ši aplinka yra ribota pagal turimų resursų kiekį (pvz., CPU, atmintis), ir kūrėjai turi rinktis fiksuotas instancijų klases, kurios apibrėžia šių išteklių dydį. [26] [27] [28] Ši aplinka taip pat suteikia didelį saugumą ir ribotą prieigą prie failų sistemos bei serverio resursų, užtikrinant stabilią ir saugią veiklą. [26] [29] [30]

GAE lanksčioji aplinka, paremta *Google Compute Engine* virtualiosiomis mašinomis, suteikia kūrėjams daugiau lankstumo ir galimybių pritaikyti infrastruktūrą pagal specifinius poreikius. [31] Šioje aplinkoje galima naudoti pasirinktinius *Docke*r atvaizdus arba sukurti *Dockerfile*, leidžiantį kūrėjams sukurti savo vykdymo aplinką, kuri gali palaikyti bet kurią kalbą ar platformą, galinčią aptarnauti HTTP užklausas. [32] Skirtingai nuo standartinės aplinkos, lanksčioji aplinka suteikia galimybę pritaikyti virtualių mašinų išteklius, tokius kaip CPU ir atmintis, kiekvienai programos instancijai. Taip pat galima gauti prieigą prie visos virtualiosios mašinos per SSH ir įdiegti pasirinktines bibliotekas ar programinės įrangos paketus, kurie reikalingi specializuotoms aplikacijoms. Lanksčioji aplinka taip pat teikia automatizuotą mastelio keitimą pagal sistemos apkrovą, tačiau kūrėjams suteikiama daugiau kontrolės atliekant infrastruktūros atnaujinimus, pavyzdžiui, operacinės sistemos atnaujinimus. [31]

Apibendrinant, GAE aplinkų analizė, kurios detalės pateiktos **5 PRIEDASE**, atskleidžia, kad pasirinkimas tarp standartinės ir lanksčiosios aplinkos priklauso nuo projekto dydžio, sudėtingumo ir specifinių poreikių. Standartinė aplinka yra ypač tinkama mažiems projektams, kuriems reikalingas paprastas diegimas ir minimalus infrastruktūros valdymas. Ji leidžia kūrėjams sutelkti dėmesį į aplikacijos kūrimą, nes infrastruktūros priežiūra ir automatinis mastelio keitimas yra visiškai valdomi Google. Tuo tarpu lanksčioji aplinka siūlo daugiau lankstumo, todėl ji yra labiau tinkama sudėtingesniems ir didesniems projektams, kuriems reikalinga didesnė kontrolė, pritaikymas ir

galimybė naudoti nestandartines kalbas ar pritaikytas *Docker* technologijas. Šios aplinkos privalumai apima pilną virtualios mašinos valdymą, SSH prieigą ir galimybę naudoti įvairias programavimo kalbas, todėl ji suteikia daugiau galimybių ir kontrolės. Renkantis tinkamą aplinką, būtina atsižvelgti į projekto dydį, lankstumo poreikius ir ilgalaikės plėtros perspektyvas.

1.7. Našumas

GAE yra viena iš dominuojančių PaaS debesijos kompiuterijos platformų, pasižyminti našumu.

Naudojant spalvotųjų *Petri* tinklų (CPN) metodologiją buvo tiriamas GAE našumas. Tyrimo tikslas buvo išanalizuoti GAE našumą, diegiant duomenų bazių sandorių sistemas, ir pasiūlyti simuliacijų sistemą, kuri padėtų prognozuoti programų našumą. Tyrimo procesas buvo atliekamas keliais etapais, leidžiančiais sistemingai įvertinti GAE našumą: Pirmame žingsnyje buvo sukurtas simuliacijų modelis, pagrįstas spalvotų *Petri* tinklų (CPN) metodologija. Šis modelis leido analizuoti GAE API elgseną ir tarpusavio sąveiką, padedant suprasti, kaip programų funkcionalumas priklauso nuo API sąveikos. Vėliau buvo atlikti empirinių duomenų rinkimo procesai, apimantys vartotojų sukurtų programų naudojimą. Naudojant šias programas, buvo matuojami vidutiniai ir dispersiniai laikai, susiję su GAE API vykdymu. Šis etapas buvo vienas svarbiausių, nes gauti duomenys leido sukurti patikimą statistinį pagrindą, reikalingą našumo analizei. Surinkti duomenys buvo integruoti į CPN modelį, leidžiantį simuliuoti GAE programų veikimą ir analizuoti jų atsakymo laikus bei pralaidumą. Modeliavimas buvo orientuotas į API sąveiką, laikant GAE struktūrą juoda dėžute, todėl vidinė sistema nebuvo detalizuojama, o dėmesys buvo sutelktas į išorinius rezultatus. Įvykdytos simuliacijos leido nustatyti, kaip įvairūs API vykdymo laikai ir jų tarpusavio sąveika veikia bendrą sistemą. Tai suteikė galimybę stebėti, kaip skirtingi veiksniai, pavyzdžiui, užklausų tipai ir sistemos apkrovimas, paveikia programų našumą. Po simuliacijų buvo analizuojami gauti rezultatai, siekiant apskaičiuoti vidutinius atsakymo laikus, pralaidumą ir kitus našumo rodiklius. Ši analizė leido išskirti svarbius aspektus, turinčius didžiausią įtaką sistemai. Paskutiniame etape, remiantis analizuotais duomenimis, buvo pateiktos išvados apie GAE našumo prognozavimą ir rekomendacijos dėl nuolatinio matavimo bei modelio atnaujinimo. Tai padėjo nustatyti geriausius būdus valdyti kritinių sistemų perkėlimą į debesų aplinką. [33]

Remiantis [33] šaltinio analize, galima teigti, kad GAE našumo prognozavimas, pasitelkiant spalvotųjų *Petri* tinklų (CPN) metodologiją, yra efektyvi priemonė debesų paslaugų programavimo sprendimams vertinti. Tyrimo rezultatai rodo, kad GAE API sąveikos modeliavimas ir surinkti

empiriniai duomenys leidžia tiksliai prognozuoti programų atsakymo laikus bei sistemos pralaidumą, o tai yra ypač svarbu diegiant kritines informacines sistemas debesų kompiuterijos aplinkoje [33].

Remiantis šaltiniu [33], siūloma metodika gali būti pritaikoma realiose debesijos paslaugų kūrimo ir plėtros situacijose, ypač siekiant optimizuoti našumą bei užtikrinti sklandų sistemų perkėlimą į GAE aplinką. Naudojant simuliacijų modelius, pagrįstus literatūros šaltiniuose aprašytais metodais, galima ne tik numatyti galimus našumo trūkumus, bet ir nuolat atnaujinti modelius realiu laiku, siekiant padidinti programų efektyvumą [33]. Tai suteikia vertingų įžvalgų įmonėms, siekiančioms maksimaliai išnaudoti debesijos platformų lankstumą bei mastelio didinimo galimybes, tuo pačiu užtikrinant aukštą paslaugų kokybę vartotojams [33].

Be to, [33] šaltinyje pateikiama tvirta metodologinė bazė tolimesniems tyrimams apie GAE veikimo principus ir galimą optimizaciją. Ateities tyrimuose galėtų būti gilinamasi į specifinių API sąveikų optimizavimo strategijas ir modelių atnaujinimo dažnį, siekiant dar labiau pagerinti našumo valdymą. Tai leistų geriau valdyti rizikas, susijusias su kritinių duomenų bazių sandorių sistemų perkėlimu į debesų aplinką ir padidinti bendrą sistemų veikimo efektyvumą [33].

Tyrime, kuriame lyginami GAE ir *Amazon EC2* dėl aukšto našumo paralelinių skaičiavimų, buvo aptartos GAE stipriosios ir silpnosios pusės. GAE pasirodė esanti efektyvi platforma, pasižyminti mažomis resursų paruošimo sąnaudomis ir žemu vykdymo vėlavimu, ypač trumpalaikiams uždaviniams, trunkantiems mažiau nei vieną valandą. Palyginus su *Amazon EC2*, GAE išsiskiria galimybe sparčiai priskirti ir valdyti resursus esant poreikiui, o *Amazon EC2* reikalauja daugiau laiko ir pastangų resursų paruošimui. Šis pranašumas ypač matomas naudojant trumpalaikes užduotis, kur GAE tampa ekonomiškesnė dėl lankstaus ir dinamiško resursų naudojimo. Tyrimo metu buvo pasitelktas valdovo-pakaliko (angl. *master-slave*) modelis, kuriame valdovas yra atsakingas už užduočių generavimą ir jų paskirstymą pakalikams. Pakalikai, veikiantys kaip atskiros GAE taikomosios programos, vykdo skaičiavimus. Šis modelis leidžia GAE platformai efektyviai panaudoti debesijos išteklius, kadangi GAE automatiškai paskirsto apkrovą ir sukuria papildomas programos instancijas pagal esamą apkrovą. Tai suteikia platformai lankstumą, nes esant didesniam srautui, sistemoje gali būti sukurta daugiau instancijų, o apkrovai sumažėjus – sumažinamas išteklių kiekis. Tyrime buvo išnagrinėti Java virtualios mašinos (JVM) ir JIT (angl. *Just-In-Time*) kompiliavimo efektai, kurie parodė, kad po pradinio vykdymo programos našumas žymiai pagerėjo. Eksperimentai, atlikti naudojant *Monte Carlo* metodais grindžiamus skaičiavimus, parodė teigiamus rezultatus, kurie rodė gerą GAE aplinkos mastelį. Šie eksperimentai parodė, kad nors pirmieji skaičiavimai užtruko ilgiau dėl išankstinio instancijų paruošimo ir kompiliacijos, vėlesni vykdymai buvo gerokai greitesni. Tai iliustruoja JIT kompiliavimo privalumus ir patvirtina, kad GAE gali būti tinkama platforma paraleliniams uždaviniams. Nepaisant teigiamų aspektų, tyrimas taip pat išryškino

dvi pagrindines GAE silpnybes. Pirma, tarpinės programinės įrangos (angl. *middleware*) sukuriamas papildomas apkrovimas mažina bendrą našumą, nes ji įveda papildomus sluoksnius tarp programos ir žemutinės infrastruktūros. Antra, resursų kvotos riboja ilgalaikius ir sudėtingesnius skaičiavimus, nes tam tikrose situacijose sistema gali pasiekti nustatytas ribas ir nebevykdyti naujų užduočių, o tai gali būti problema didesnio masto uždaviniams. Šios kvotos buvo vienas iš pagrindinių veiksnių, ribojusių paralelinių užduočių mastelį GAE aplinkoje. [34]

Remiantis šaltinio [34] tyrimo rezultatais, galima teigti, kad GAE yra ypač efektyvi platforma trumpalaikiams ir mažos apimties debesų paslaugų programavimo sprendimams. Tyrimas parodė, kad GAE pasižymi mažomis resursų paruošimo sąnaudomis ir automatinio resursų valdymo galimybėmis, todėl ši platforma yra tinkama organizacijoms, kurioms svarbu efektyviai ir greitai keisti resursų mastelį pagal poreikį [34]. Ši savybė leidžia įmonėms optimizuoti veikimą bei mažinti infrastruktūros sąnaudas, ypač kai darbo krūviai kinta.

Tyrime taip pat buvo pabrėžta, kad naudojant valdovo-pakaliko (angl. *master-slave***)** modelį, galima skaidriai paskirstyti užduotis tarp skaičiavimo mazgų, o tai yra itin vertinga debesų aplinkoje. Šio modelio pritaikymas GAE platformoje leidžia automatiškai balansuoti apkrovas ir plečiant instancijas pagal apkrovos augimą. Tokiu būdu galima teigti, kad GAE yra pritaikoma realiuose naudojimo scenarijuose, kuriuose reikia dinamiškai valdyti resursus [34]. Be to, šaltinio [34] tyrimo rezultatai atskleidė, jog JIT kompiliacija žymiai pagerina programų vykdymo greitį, ypač po pradinio vykdymo, o tai daro GAE tinkama aukšto našumo skaičiavimams. Eksperimentai, atlikti naudojant *Monte Carlo* metodus, patvirtino, kad GAE gali pasiūlyti gerą mastelio didinimą, tačiau tyrimas taip pat parodė tam tikrus platformos apribojimus, susijusius su tarpinės programinės įrangos apkrova ir resursų kvotomis, kurie riboja sudėtingesnių ir ilgalaikių uždavinių vykdymą.

Taigi, remiantis šaltiniu [34], galima daryti išvadą, kad GAE yra efektyvi debesų paslaugų platforma tam tikrų specifinių uždavinių sprendimui, tačiau siekiant maksimalaus našumo, būtina optimizuoti kvotų valdymą ir taikomųjų programų architektūrą. Šios išvados patvirtina, kad tinkamai valdomi GAE apribojimai gali padidinti platformos efektyvumą ir pritaikomumą realiuose skaičiavimo uždaviniuose.

Galima atlikti našumo prognozavimą, naudojant GAE kaip PaaS tipo debesijos aplinką. Tyrimo metu buvo sukurtas modeliavimo ir simuliacijos metodas, pasitelkiant UPPAAL modelio tikrintuvą, kuris leidžia modelius išreikšti laiko automatais. Ši metodologija buvo naudinga analizuojant našumo svyravimus, kylančius dėl API naudojimo ir jų integracijos su GAE platforma. Modeliavimas atskleidė, kaip įvairios API užklausos daro įtaką bendrai sistemos veiklai, įskaitant uždelstą API atsako laiką ir poveikį sistemos efektyvumui. Tyrimo rezultatai taip pat parodė

galimybes optimizuoti sistemą, pavyzdžiui, keičiant API kvietimo dažnumą ar paskirstant resursus taip, kad sumažėtų uždelsimai ir pagerėtų aplikacijos atsako laikas. Naudota metodologija ir modeliavimo principai gali būti taikomi ne tik GAE, bet ir kitoms PaaS debesijos aplinkoms, pritaikant specifines API ir jų parametrus. Tai suteikia universalų įrankį debesijos paslaugų našumo optimizavimui. Nors GAE platforma susiduria su tam tikrais našumo iššūkiais dėl virtualizuotos struktūros, tyrimas parodė, kad platforma gali būti efektyviai modeliuojama ir analizuojama, siekiant pagerinti bendrą našumą atsižvelgiant į API veikimą ir jų sąveiką su debesijos paslaugomis. Šis tyrimas prisideda prie geresnio debesijos paslaugų, ypač GAE, našumo supratimo ir optimizavimo. [35]

GAE platforma, nepaisant virtualizuotos architektūros ir susijusių našumo svyravimų, gali būti efektyviai naudojama įvairiems debesijos sprendimams kurti. Remiantis [35] šaltinio tyrimu**,** našumo prognozė ir API sąveikų optimizavimas yra pagrindiniai veiksniai, darantys įtaką bendrai sistemos veiklai. Našumo modeliavimas, naudojant tokius įrankius kaip UPPAAL modelio tikrintuvas, leidžia numatyti svarbiausius našumo aspektus, įskaitant API kvietimų dažnumą, jų vykdymo laiką ir resursų paskirstymo įtaką. Be to, GAE platformos universalumas, kaip nurodyta [35], leidžia šią metodiką pritaikyti ir kitose PaaS tipo debesijos aplinkose, keičiant tik specifinius API parametrus. Tai suteikia galimybę efektyviai optimizuoti našumą įvairiose debesijos sistemose, kur pagrindinis dėmesys skiriamas sistemos atsako laikui ir resursų efektyvumui.

Analizuojant [35] šaltinį, galima teigti, kad nors GAE susiduria su tam tikrais našumo iššūkiais, ji gali būti optimizuojama, siekiant pagerinti veikimą ir sumažinti API atsako laikus. Tai svarbus žingsnis link patikimų ir našiai veikiančių debesijos paslaugų sprendimų kūrimo, atitinkančių šiuolaikinius programavimo ir sistemų valdymo poreikius.

Kitas nagrinėjamas tyrimas buvo orientuotas į GAE platformos našumo įvertinimą, susijusį su socialinių tinklų aplikacijų kūrimu ir mastelio pritaikymu naudojant mažai kodo. Projektas naudojo *Apache JMeter* įrankį, kuris yra skirtas apkrovos testavimui ir našumo matavimui. *JMeter* leido išanalizuoti aplikacijų, veikiančių GAE, našumą, įgyvendinant skirtingus apkrovos scenarijus. Testavimas buvo atliktas įvairiais apkrovos lygiais, pradedant nuo žemų iki labai aukštų, siekiant išmatuoti aplikacijos mastelio pritaikymo galimybes ir atsako laikus. Testavimo metu buvo stebima, kaip keičiasi atsakymo laikai, naudojant skirtingą virtualių vartotojų skaičių, nuo minimalaus iki maksimalaus apkrovos scenarijaus. Rezultatai atskleidė, kad GAE sėkmingai tvarko skirtingas apkrovas, užtikrindama aplikacijų stabilumą ir greitį net ir esant intensyviai naudojimui. Aplikacijų našumas buvo vertinamas pagal atsakymo laikus, kurie, padidėjus vartotojų skaičiui, išliko priimtini, rodydami GAE gebėjimą efektyviai mastelio resursus pagal poreikį. Taip pat buvo nustatyta, kad

GAE automatiškai pritaiko resursus, atsižvelgiant į aplikacijos apkrovą, leidžiant vartotojams mokėti tik už realiai panaudotus išteklius. [36]

Remiantis [36] šaltinio tyrimo rezultatais, galima teigti, kad GAE platforma yra efektyvi naudojant ją socialinių tinklų aplikacijoms. *Apache JMeter* įrankiu atlikti testai parodė, kad GAE sėkmingai valdo skirtingas apkrovas ir automatiškai paskirsto resursus pagal poreikį, o tai užtikrina stabilų paslaugų veikimą net ir intensyvaus naudojimo metu.

Nors šaltinyje [36] daugiausia dėmesio skiriama socialinių tinklų aplikacijų kūrimui ir GAE našumo testams, gauti rezultatai leidžia manyti, kad šie sprendimai galėtų būti pritaikomi ir platesnėje verslo aplinkoje, ypač įmonėms, siekiančioms sumažinti IT infrastruktūros kaštus ir optimizuoti išteklių panaudojimą. GAE gebėjimas automatiškai paskirstyti resursus leidžia lanksčiai reaguoti į kintančius verslo poreikius, užtikrinant aukštą paslaugų prieinamumą ir našumą. Be to, šaltinyje nustatyta, kad GAE platforma suteikia galimybę mokėti tik už realiai panaudotus išteklius, o tai dar labiau sumažina veiklos sąnaudas. Tyrimo rezultatai patvirtina, kad GAE platforma yra sėkmingai pritaikoma socialinių tinklų aplikacijose, užtikrinant efektyvų resursų naudojimą ir paslaugų kokybę. Remiantis šiais rezultatais, galima manyti, kad platforma taip pat galėtų būti naudinga ir kitose pramonės šakose, kur reikalingas lankstumas ir mastelio pritaikymas.

Džiugina ir nauji atnaujinimai. Naujas *HttpConnector* režimas GAE *Java runtimes*, įvedęs galimybę apeiti gRPC sluoksnį, ženkliai pagerino atminties ir CPU naudojimą, suteikdamas tiesioginę prieigą prie HTTP užklausų. Šis pokytis ne tik efektyvina užklausų apdorojimą, bet ir rodo GAE tendenciją atsisakyti senųjų technologijų naudai naujų, efektyvesnių sprendimų. Ateityje šis režimas bus numatytasis visoms aplikacijoms, palaipsniui atsisakant senųjų *Java 8 runtimes*. Naujojo režimo privalumai yra tokia, kad anksčiau vykdomos užklausos, kurios reikalavo didelių atminties išteklių, dabar reikalauja žymiai mažiau atminties. Pavyzdžiui, su *HttpConnector*, 30MB užklausoms reikalinga atmintis sumažėjo nuo 78.8 MB iki 15.2 MB, o tai sudaro maždaug 81% sumažėjimą. Naujo režimo naudojimas sumažino CPU laiko poreikį apie 15% visos valandos testavimo metu, mažinant apkrovą ir leidžiant efektyviau naudoti sistemos resursus. Tyrimas atskleidė, kad naujojo režimo metu generuojamų šiukšlių kiekis žymiai sumažėjo. Pavyzdžiui, užklausoms su 1MB duomenų, šiukšlių kiekis sumažėjo nuo 2.3 MB iki 0.1 MB per užklausą. [37] [38]

Šie skaičiai aiškiai rodo, kaip *HttpConnector* režimas pagerina GAE našumą, suteikiant programuotojams galimybę efektyviau valdyti resursus ir mažinti sistemos apkrovą. Ateityje šis režimas bus numatytasis visoms aplikacijoms, palaipsniui atsisakant senųjų *Java 8 runtimes*. XML kodas, reikalingas šiam režimui aktyvuoti, yra pateiktas žemiau *1 pav.*. [37] [38]

```
<system-properties>
    <property name="appengine.use.httpconnector" value="true"/>
 </system-properties>
```

Šaltinis: [38]

1 pav. Java HTTP jungties įjungimas atminčiai ir CPU naudojimui pagerinti

Tyrimo rezultatai atskleidė, kad naudojant naujausius atnaujinimus, tokius kaip *HttpConnector* režimas, galima žymiai pagerinti sistemų efektyvumą, sumažinant atminties ir procesoriaus resursų naudojimą. Šie technologiniai patobulinimai turi tiesioginį pritaikymą realiose aplikacijose, kur svarbu optimizuoti resursų naudojimą ir sumažinti veiklos kaštus.

Atsižvelgiant į atliktus tyrimus, GAE platforma gali būti vertinama kaip veiksminga priemonė debesijos sprendimams dėl savo gebėjimo užtikrinti sistemos našumą ir pralaidumą. Simuliacijų, tokių kaip spalvotųjų *Petri* tinklų (CPN) metodologija, naudojimas suteikė galimybę tiksliai įvertinti API sąveikos įtaką sistemos veikimui. Šis požiūris parodė, kad GAE gali patikimai prognozuoti atsakymo laikus ir pralaidumą, o tai yra būtina kritinių sistemų debesijos aplinkose. Be to, GAE pasižymi lankstumu dinamiškai priskiriant resursus, o tai leidžia efektyviai reaguoti į apkrovos pokyčius. Pavyzdžiai, tokie kaip socialinių tinklų aplikacijų apkrovos testai, parodė, kad GAE gali užtikrinti stabilų veikimą net esant intensyviam vartotojų srautui, o tai rodo platformos tinkamumą didelio masto debesijos paslaugoms. Nauji technologiniai patobulinimai, pavyzdžiui, *HttpConnector* režimas, dar labiau pagerino GAE našumą, ženkliai sumažindami atminties ir CPU resursų naudojimą. Tai rodo, kad GAE nuolat tobulinama siekiant optimizuoti resursų valdymą ir mažinti veiklos sąnaudas, o tai yra ypač svarbu efektyviam debesijos sprendimų kūrimui. Galiausiai, tyrimai apie GAE pritaikymą įvairiose srityse, pavyzdžiui, socialinių tinklų aplikacijose, patvirtina platformos gebėjimą efektyviai tvarkyti apkrovas ir prisitaikyti prie kintančių verslo poreikių. Tai leidžia daryti išvadą, kad GAE yra tinkama platforma, skirta organizacijoms, siekiančioms užtikrinti efektyvų resursų naudojimą, greitą mastelio didinimą ir aukštą paslaugų kokybę.

1.8. Saugos sprendimai

Anksčiau šiame darbe trumpai buvo pristatyti tokie saugos sprendimai kaip *Google Cloud Armor, Cloud SQL* šifravimas ir atsarginių kopijų kūrimas, *Memcache*.

Taip pat GAE siūlo automatinį duomenų šifravimą tiek esant ramybės būsenoje (angl. *at rest*), tiek perduodant duomenis (angl. *in transit*). Duomenys yra šifruojami pagal nutylėjimą, naudojant stiprius šifravimo algoritmus, kad būtų užtikrinta jų sauga tiek debesyje, tiek tinkluose. Šis mechanizmas padeda apsaugoti duomenis nuo perėmimo, kai jie keliauja tarp vartotojo ir GAE platformos, taip pat kai jie yra saugomi vidinėje GAE infrastruktūroje. [39] [40]

Google Cloud automatiškai šifruoja visus savo infrastruktūroje saugomus duomenis, įskaitant GAE duomenis, naudodama pažangųjį šifravimo standartą (AES), dažniausiai su 256 bitų šifravimo raktu (AES-256). Šifravimas įgyvendinamas keliuose sluoksniuose, nuo saugojimo sistemos iki fizinės aparatūros, užtikrinant papildomą apsaugą. Kiekviena duomenų dalis yra šifruojama su unikaliais duomenų šifravimo raktais (DEK), kurie, savo ruožtu, yra šifruojami naudojant šifravimo raktų valdymo raktus (KEK). Jei duomenys atnaujinami, jie šifruojami nauju raktu, taip apsaugant nuo galimo raktų kompromitavimo. Be to, šifruoti duomenys yra replikuojami atsarginėms kopijoms ir nelaimių atvejams. Vartotojai gali valdyti savo šifravimo raktus naudodamiesi *Google Cloud* raktų valdymo paslauga (KMS), kuri suteikia galimybę sukurti, sukti ir sekti raktus. [39]

Visi duomenys, perduodami tarp naudotojo ir GAE platformos, yra šifruojami pagal nutylėjimą naudojant transportinio lygmens saugumą (TLS). Tai apsaugo duomenis nuo perėmimo ir užtikrina jų konfidencialumą bei vientisumą, kai jie perduodami per nesaugius tinklus. Be to, vidinėms „Google“ paslaugų komunikacijoms naudojamas *Application Layer Transport Security* (ALTS) protokolas, kuris padeda apsaugoti vidinius tinklo ryšius ir duomenis perduodant tarp skirtingų GAE komponentų ir kitų debesų paslaugų. Naudotojams, turintiems papildomų saugumo reikalavimų, yra prieinamos tokios galimybės kaip IPSec tuneliai, valdomi SSL sertifikatai ir kiti apsaugos mechanizmai. [40]

Šiuolaikinėje debesų kompiuterijos aplinkoje saugumo iššūkiai tampa vis sudėtingesni, ypač kai organizacijos vis dažniau pasikliauja debesų paslaugų tiekėjais savo duomenų valdymui. GAE saugumo funkcijos, tokios kaip duomenų šifravimas ramybės būsenoje (angl. *at rest*) ir perduodant duomenis (angl. *in transit*), yra kritinės užtikrinant duomenų apsaugą ir pasitikėjimą debesų paslaugomis. Pasak *Google Cloud Security Whitepapers*, GAE naudojamas daugiasluoksnis šifravimo modelis užtikrina, kad duomenys yra šifruojami nuo saugyklos sistemos iki fizinės aparatūros lygmens, todėl net jei fizinis prieigą prie įrenginių būtų pažeistas, duomenys liktų nepasiekiami be šifravimo raktų. Be to, tokios technologijos kaip transportinio lygmens saugumas (TLS) ir *Application Layer Transport Security* (ALTS) padeda apsaugoti tiek išorinius ryšius su klientais, tiek vidinius *Google* infrastruktūros tinklus. [41]

Tokie sprendimai atspindi dabartines tendencijas, kai debesų kompiuterijos infrastruktūros turi būti ne tik saugios, bet ir lanksčios, kad galėtų apsaugoti nuo augančių grėsmių, kaip aptarnavimo perkrovos (DoS) atakos ar neleistinos prieigos bandymai. *Google* investicijos į saugumą apima ne tik technologines priemones, bet ir saugumo ekspertų komandą, kuri rūpinasi šių priemonių nuolatiniu tobulinimu ir jų auditu. [41]

Kitas saugumo sprendimas, kurį verta paminėti yra *Google Cloud Identity and Access Management* (IAM). Jis suteikia galimybę smulkiai valdyti prieigą prie *Google Cloud* resursų, įskaitant GAE. Naudojant IAM, galite priskirti vartotojams ir paslaugų paskyroms vaidmenis, kurie apibrėžia, kokius veiksmus jie gali atlikti su konkrečiais resursais, įgyvendinant mažiausių privilegijų principą. Vaidmenys apima leidimus, kurie nėra suteikiami tiesiogiai vartotojui, o grupuojami į vaidmenis (angl. *role-based access control*). IAM politika leidžia tiksliai nustatyti, o tai gali pasiekti ar modifikuoti resursus, taip užtikrinant saugumą ir efektyvią prieigos kontrolę visoje organizacijos infrastruktūroje. [42]

GAE siūlo įvairius vartotojų autentifikavimo metodus, kurie užtikrina aplikacijų saugumą ir vartotojų prieigą. Rekomenduojama naudoti *Identity Platform*, kuri palaiko autentifikaciją naudojant slaptažodžius, telefono numerius bei įvairius tapatybės tiekėjus, tokius kaip *Google*, *Facebook*, *Twitter*, taip pat protokolus, tokius kaip SAML ir *OpenID Connect*. *Firebase Authentication* yra lengvas ir greitas sprendimas vartotojų autentifikacijai, palaikantis *Google*, *Facebook* ir *Twitter* tapatybes bei užtikrinantis mažiausią kodo kiekį. *Google Identity Services for Web* leidžia autentifikuoti vartotojus naudojant *Google* paskyras, o *OAuth 2.0* ir *OpenID Connect* suteikia galimybę naudoti „federuotą“ autentifikavimą su pasirinktu tapatybės tiekėju, įskaitant *Google*. Be to, *Identity-Aware Proxy* (IAP) prideda papildomą apsaugos sluoksnį, tikrindamas vartotojų prieigos teises prieš leidžiant jiems pasiekti *App Engine* resursus. *Users API* taip pat suteikia galimybę lengvai valdyti vartotojų autentifikavimą *Google* paskyromis bei kontroliuoti prieigą prie administracinių funkcijų. [43]

GAE suteikia platų saugumo priemonių rinkinį, kuris padeda užtikrinti saugų aplikacijų veikimą bei apsaugoti jas nuo galimų grėsmių. Pirmiausia, GAE palaiko HTTPS užklausas, kurios leidžia užtikrinti saugų ryšį tarp naudotojų ir aplikacijos. Naudojant HTTPS protokolą, vartotojų duomenys yra šifruojami, o tai sumažina galimybes neteisėtai perimti jautrią informaciją. Be to, GAE teikia galimybę naudoti automatinius SSL sertifikatus tinkintiems domenams, o tai padeda apsaugoti svetaines su nuosavais domenais. Kitas svarbus saugos aspektas yra prieigos kontrolė, kuri leidžia nustatyti, o tai ir kokias teises turi GAE projekte. Prieigos kontrolė leidžia riboti prieigą tik tam tikriems vartotojams ar paslaugoms, naudojant vaidmenimis paremtą teisių valdymo sistemą. Be to, GAE naudoja ugniasienės (angl. *firewall*) taisykles, kurios suteikia galimybę blokuoti arba leisti

prieigą iš tam tikrų IP adresų ar tinklų, užtikrinant, kad tik įgaliotos užklausos pasiektų aplikaciją. Taisykles galima pritaikyti tiek testavimo, tiek paleidimo aplinkose, leidžiant apriboti prieigą pagal skirtingus kriterijus. Taip pat galima blokuoti kenkėjišką veiklą, tokią DoS atakos. Kita priemonė – saugumo skeneris, kuris leidžia aptikti pažeidžiamumus aplikacijoje, nuskaitydamas ją ir testuodamas įvairias naudotojų sąsajas bei įvykių tvarkytuvus. Galiausiai, naudojant išėjimo kontrolės (angl. *egress controls*) ir įėjimo kontrolės (angl. *ingress controls*) mechanizmus, galima tiksliai reguliuoti srautą tarp GAE ir kitų tinklų ar paslaugų, taip padidinant aplikacijų saugumą. Šios priemonės kartu padeda užtikrinti, kad GAE aplikacijos veiktų saugiai ir būtų apsaugotos nuo įvairių saugumo grėsmių. [44]

Atlikta šaltinių analizė parodė, jog GAE aplinką galima pritaikyti spręsti tokioms saugumo problemoms kaip *data stomping*. Tai situacija, kai kelios užklausos vienu metu bando modifikuoti tą patį duomenų objektą, dėl ko vieno proceso atlikti pakeitimai gali būti netyčia perrašyti kito proceso pakeitimais. Detalesnė informacija apie *data stomping* sprendimo būdus ir *Khan Academy* sukurtus modulius *db_hooks.py* ir *txn_safety.py* pateikta **6 PRIEDASE**. [45]

Apibendrinant, GAE saugos sprendimai užtikrina ne tik efektyvią duomenų apsaugą, bet ir patikimą aplikacijų veikimą debesų kompiuterijos aplinkoje. Automatinis šifravimas tiek ramybės būsenoje, tiek duomenų perdavimo metu, prieigos kontrolė su IAM, bei lankstūs autentifikavimo sprendimai, tokie kaip *Firebase Authentication* ir *Identity-Aware Proxy*, sudaro stiprią daugiasluoksnę apsaugos architektūrą. Šios priemonės leidžia GAE ne tik apsaugoti duomenis nuo įvairių grėsmių, bet ir užtikrinti, kad naudotojai galėtų valdyti savo resursus pagal individualius saugumo poreikius.

Be to, *Khan Academy* sukurti moduliai *db_hooks.py* ir *txn_safety.py*, pritaikyti GAE aplinkai, reikšmingai prisideda prie sistemos patikimumo didinimo (detaliau žr. **6 PRIEDASĄ**). Jie leidžia realiuoju laiku identifikuoti nesuderinamus veiksmus ir automatiškai generuoti klaidų pranešimus su išsamia atsekamumo informacija, mažinant duomenų praradimo riziką bei optimizuojant klaidų šalinimo procesą. Tokie sprendimai užtikrina aukštą saugumo ir patikimumo lygį sudėtingose aplinkose. [45]

Ši saugos infrastruktūra ne tik atitinka šiuolaikinius debesų kompiuterijos standartus, bet ir suteikia vartotojams pasitikėjimą, kad jų aplikacijos veiks saugiai ir patikimai.

1.9. Kombinuoti GAE sprendimai

Hibridinės debesijos architektūros tampa vienu iš sprendimų, kai siekiama užtikrinti duomenų privatumą ir efektyvų sistemų veikimą. Pavyzdžiui, *ScaLib* bibliotekos sistemos kūrimas parodė, kaip galima sėkmingai sujungti vietinę infrastruktūrą su debesijos paslaugomis (daugiau techninių detalių apie *ScaLib* sprendimą pateikta **7 PRIEDAS Kombinuotas GAE programavimo sprendimas *ScaLib* sistemoje**). Ši sistema buvo kuriama siekiant sukurti plėtrą palaikančią bibliotekos katalogo valdymo platformą, kurioje vartotojai gali ieškoti ir skolintis objektus, o bibliotekininkai gali modifikuoti objektus ir jų informaciją. Vienas pagrindinių iššūkių buvo užtikrinti duomenų privatumą. Kadangi ši sistema naudoja hibridinę debesijos infrastruktūrą, jautri vartotojų informacija buvo laikoma vietinėje infrastruktūroje, o likusi dalis buvo laikoma GAE debesijos platformoje. Kūrėjams teko išspręsti iššūkį, kaip sujungti šias duomenų dalis taip, kad vartotojų asmeniniai duomenys niekada neperduotų per debesijos infrastruktūrą, bet tuo pačiu užtikrintų bibliotekininkams galimybę patogiai naudoti ir vietinius, ir debesyje saugomus duomenis. [46]

Dėl saugumo sumetimų, *ScaLib* sistemoje buvo nuspręsta vartotojų duomenis saugoti vietiniame serveryje, o jų identifikatorius naudoti debesijoje, taip išvengiant jautrios informacijos perdavimo per viešus tinklus. Sistemoje buvo naudojamos kliento pusėje vykdomos užklausos (angl. *client-side joins*), kurios leido užtikrinti, kad duomenų susiejimas vyktų vietinėje aplinkoje, taip užtikrinant, kad vartotojų vardai, slaptažodžiai ar kiti privatūs duomenys nebūtų perduodami per GAE infrastruktūrą. Vartotojai, naršantys bibliotekos sistemoje, gaudavo nuorodas į vartotojų identifikatorius, o sistema automatiškai išsiųsdavo užklausas vietinei aplinkai, kad išgautų atitinkamus duomenis. Tai sumažino galimybę, jog jautri vartotojų informacija gali būti perimta per išorinius tinklus, taip padidinant sistemos saugumą. [46]

Sistemos našumo vertinimas parodė, kad ji sugebėjo efektyviai veikti esant didelėms apkrovoms. Apkrovos testai, atlikti naudojant *Google Kubernetes Engine* ir *Locust*, parodė, kad didinant vartotojų apkrovą nuo 10 iki 300 naudotojų, sistema sugebėjo stabilizuotis per 15 sekundžių po kiekvieno apkrovos padidinimo. Buvo atliktas ir testas su 1000 vartotojų, kuris sukūrė apie 400 užklausų per sekundę. Nors sistema pradinį apkrovos piką tvarkė su 1,3 sekundės užlaikymu, ilgainiui *Google Kubernetes Engine* aptiko DDoS atakos ženklus ir automatiškai suaktyvino apsaugos mechanizmus. Tokie rezultatai parodė, kad *ScaLib* gali efektyviai tvarkytis su didelėmis apkrovomis, nors itin didelės apkrovos atveju reikalingos papildomos apsaugos priemonės. [46]

Tvarkaraščio valdymo sistema, sukurta naudojant GAE ir *Go* programavimo kalbą, demonstruoja debesijos platformų potencialą kuriant efektyvias verslo aplikacijas. Sistemos kūrimas prasidėjo nuo paprastos architektūros, susidedančios iš vartotojų autentifikacijos ir tvarkaraščių valdymo modulių. Autentifikacijos modulis buvo sukurtas naudojant *net/http* biblioteką, kartu su *Gorilla Web Toolkit* autentifikacijos sprendimui įgyvendinti. Vartotojų duomenys ir tvarkaraščiai buvo saugomi naudojant GAE *Datastore, NoSQL* duomenų bazę, kuri yra optimizuota debesijos aplinkai. Vienas iš esminių šios sistemos privalumų buvo sumažėjusios kūrimo ir eksploatacinės išlaidos. Naudojant GAE, kūrėjai neturėjo rūpintis serverių ar duomenų bazės valdymu, nes platforma automatiškai užtikrino resursų didinimą ir infrastruktūros valdymą pagal poreikį. Be to, *Go* kalbos privalumai, tokie kaip *garbage collection* ir palaikymas konkurencingiems procesams („concurrency“), leido pasiekti didelį efektyvumą. Tačiau, kartu su privalumais, kilo ir tam tikrų iššūkių. *Go* kalba, nors ir perspektyvi, vis dar yra mažai paplitusi, lyginant su *Python* ar *Java*. Dėl šios priežasties kūrėjams trūko patirties, o esami karkasai buvo riboti. Be to, pritaikyti naujas technologijas, tokias kaip *Gorilla Web Toolkit* ir GAE *Datastore*, reikėjo papildomų mokymosi išteklių. [47]

Sistema suteikė galimybę naudotojams valdyti tvarkaraščius, prisijungti prie sistemos ir keisti savo vartotojo informaciją. Kuriant šią sistemą, buvo įdiegtos pagrindinės funkcijos: prisijungimas, vartotojų valdymas, bei tvarkaraščių tvarkymas. Administratoriaus puslapis leido valdyti visus naudotojus ir jų teises, o viešieji naudotojai galėjo tik peržiūrėti tvarkaraščius. Visi duomenys buvo saugomi GAE *Datastore* naudodami paprastą *Key-Value* saugyklą, o tai užtikrino greitą prieigą ir resursų didinimą. Šios sistemos rezultatai parodė, kad ją galima efektyviai valdyti net ir esant ribotiems finansiniams ištekliams, o naudotojų kiekis gali būti lengvai didinamas naudojant GAE platformą. [47]

Realaus laiko komunikacijos aplikacija *Cloudtarun* buvo sukurta *Android* platformai, naudojant GAE kaip pagrindinę debesijos paslaugą. Aplikacijos kūrimui buvo pasirinktos šios technologijos: GAE, *Eclipse*, *Google Web Toolkit* (GWT) ir *Android SDK*. Ši technologijų kombinacija leido integruoti realaus laiko komunikaciją su debesijos paslaugomis, užtikrinant efektyvų duomenų valdymą ir stabilumą. GAE platforma suteikė aplikacijai lankstumą ir galimybę apdoroti dideles duomenų apimtis naudojant *MapReduce* programavimo modelį. Aplikacijos kūrimas buvo palyginti greitas, nes naudojant GAE nereikėjo investuoti į serverių infrastruktūrą, o GWT leido lengvai integruoti interneto aplikacijų funkcionalumą į mobilias platformas. Atlikti programos stabilumo ir veikimo efektyvumo testai parodė, kad GAE gali efektyviai valdyti duomenų transakcijas net ir esant dideliems apkrovos šuoliams. Simuliacijos metu buvo stebimi duomenų perdavimo greičiai ir sistemos atsparumas didelėms vartotojų apkrovoms. Rezultatai parodė, kad *Cloudtarun*

išlaiko stabilumą esant didesnėms apkrovoms, o tai konstatuoja, jog GAE gali būti patikima platforma realaus laiko aplikacijoms kurti. [48]

Didelių duomenų apdorojimas tapo dar viena svarbia debesijos platformų taikymo sritimi, todėl verta paminėti GAE galimybes sprendžiant didžiųjų duomenų apdorojimo iššūkius. Granuliuotasis skaičiavimas (GrC) – tai konceptuali informacijos apdorojimo teorija, kuri imituoja žmogaus mąstymą, siekiant supaprastinti sudėtingas problemas. Šios teorijos pagrindas yra informacijos granuliavimas – procesas, kuriame problema yra skaidoma į mažesnius, lengviau valdomus informacines granules. Šis metodas ypač aktualus didžiųjų duomenų analizei, kadangi leidžia tvarkyti didelius duomenų kiekius ir spręsti kompleksinius klausimus, tokius kaip neapibrėžtų ir triukšmingų duomenų apdorojimas. GrC modeliai, tokie kaip grubaus rinkinio teorija, dalinės informacijos analizė ir klasifikavimas, suteikia būdus analizuoti duomenis pagal skirtingus jų aspektus, taikant „skaldyk ir valdyk“ (angl. *divide and conquer*) principą. Ši teorija yra plačiai taikoma didžiųjų duomenų gavyboje ir sudėtingų problemų sprendime, ypatingai tais atvejais, kai reikia apdoroti didelius duomenų rinkinius įvairiose srityse, tokiose kaip moksliniai tyrimai, duomenų gavyba ir dirbtinis intelektas. [49]

GAE yra viena iš pirmaujančių debesijos platformų, kuri remiasi PaaS modeliu, leidžiančiu vartotojams kurti ir diegti didžiųjų duomenų apdorojimo sprendimus. Vienas svarbiausių GAE komponentų yra *MapReduce* programavimo modelis, skirtas didžiųjų duomenų apdorojimui paskirstytoje aplinkoje. *MapReduce* naudoja du pagrindinius veiksmus – *Map* ir *Reduce* – kurie leidžia efektyviai padalinti duomenų rinkinius į mažesnes dalis ir paskirstyti juos per kelias mazgų grupes. Kiekviena *Map* užduotis tvarko savo duomenų granules, o po to rezultatų suvestinė atliekama *Reduce* funkcijomis, kurios sujungia tarpinius duomenis į galutinius rezultatus. Ši „skaldyk ir valdyk“ strategija itin gerai dera su GrC teorija, nes leidžia skaidyti duomenų apdorojimo užduotis į mažesnius blokus, kurie gali būti sprendžiami lygiagrečiai. [49]

Pavyzdys, pateiktas remiantis GrC ir *MapReduce* modeliais, parodo, kaip GAE platformoje duomenys gali būti efektyviai struktūrizuojami ir apdorojami. Šiame pavyzdyje informacinės granulės suformuojamos naudojant GrC principus ir *MapReduce* modelį. Kiekviena duomenų eilutė yra paverčiama granulėmis ir lygiagrečiai apdorojama Map užduotyse. Po to, kiekviena granulė yra sujungiama pagal unikalius raktus, naudojant *Reduce* funkcijas, kurios apjungia ir suformuoja galutinį rezultatą. Šis pavyzdys parodo, kaip GrC teorija, taikoma *MapReduce* modelyje, gali efektyviai suskaidyti didelius duomenų kiekius ir paspartinti apdorojimo procesus. Grubaus rinkinio teorijos pagrindu galima apytiksliai apskaičiuoti apatinę ir viršutinę rinkinių ribas, taikant *MapReduce* modelio lygiagrečią apdorojimo galią. [49]

GAE integracija su GrC ir MapReduce modeliais suteikia galimybę efektyviai spręsti didžiųjų duomenų analizės iššūkius. Paskirstytas ir lygiagretus apdorojimas leidžia ne tik efektyviai valdyti didelius duomenų kiekius, bet ir užtikrinti tikslų sprendimų priėmimą esant netiksliems ar neapibrėžtiems duomenims. „skaldyk ir valdyk“ principas, kurį GrC teorija ir *MapReduc*e modelis remiasi, padeda pasiekti aukštą efektyvumą ir mastelio pritaikomumą. Tai yra svarbu analizuojant ir tvarkant šiuolaikinius duomenų srautus debesų kompiuterijos platformose. [49]

Taigi hibridinės debesijos sprendimai siūlo lankstų požiūrį į duomenų valdymą, ypač jautrių duomenų apsaugos kontekste. Tokių architektūrų naudojimas, kaip matoma *ScaLib* pavyzdyje, leidžia derinti vietinę infrastruktūrą su debesijos paslaugomis, tačiau kartu atskleidžia iššūkius, susijusius su duomenų integracija ir saugumu. Tai rodo, kad nors hibridinė debesija yra veiksmingas sprendimas, ji reikalauja gilių techninių žinių ir atidžiai apgalvotų apsaugos mechanizmų. Taip pat GAE platforma išsiskiria savo pritaikomumu ir efektyvumu, ypač kuriant ir plečiant verslo sprendimus, kurie reikalauja dinamiško resursų valdymo. Automatinis mastelio didinimas, integruotos *NoSQL* duomenų bazės (pvz., *Datastore*), bei galimybė naudoti įvairias programavimo kalbas leidžia šiai platformai būti patikimu pasirinkimu debesijos sprendimams. Tačiau svarbu atkreipti dėmesį, kad mažiau populiarių technologijų, tokių kaip *Go* kalba ar specializuotos bibliotekos, naudojimas gali tapti kliūtimi plačiam pritaikymui dėl ribotų kūrėjų išteklių ir jų patirties stygiaus. Didžiųjų duomenų apdorojimo galimybės GAE platformoje, remiantis *MapReduce* ir GrC modeliais, parodo debesijos paslaugų galingumą sprendžiant kompleksinius duomenų analizės uždavinius. Nors šie metodai suteikia efektyvų lygiagretų ir paskirstytą duomenų apdorojimą, jų įgyvendinimas gali būti sudėtingas ir reikalauti specifinių technologinių žinių. Tai reiškia, kad tokie sprendimai yra labiau tinkami pažangioms technologijomis aprūpintoms komandoms ar organizacijoms. GAE platformos universalumas ir automatizuoti sprendimai mažina kūrimo bei eksploatacines sąnaudas, tačiau tuo pačiu metu reikalauja specifinio technologijų supratimo ir patirties. Todėl, nors GAE gali būti puikus pasirinkimas didesniems projektams ar patyrusioms komandoms, mažesnėms įmonėms gali kilti sunkumų dėl technologinės kompetencijos reikalavimų. Galime teigti, kad GAE platforma siūlo išskirtines galimybes debesijos paslaugų programavimui, tačiau šios galimybės yra glaudžiai susijusios su organizacijų technologiniu pasirengimu. Norint pilnai išnaudoti platformos potencialą, būtina užtikrinti išteklius ir kompetencijas, kurie leidžia spręsti sudėtingus techninius iššūkius, tokie kaip duomenų integracija, didelių apkrovų valdymas bei pažangių duomenų apdorojimo modelių taikymas.

1.10. GAE architektūros panaudojimas

Kaip šiame darbe buvo kalbama anksčiau, GAE siūlo architektūrinius sprendimus, leidžiančius automatizuoti infrastruktūros valdymą, optimizuoti resursų paskirstymą bei užtikrinti sistemos mastelio keitimą ir patikimumą. Vienas iš svarbiausių architektūrinių iššūkių, su kuriais susiduria šiuolaikinės didelės apimties aplikacijos, yra pajėgumas valdyti didelius vartotojų srautus, nuolatos kintančius užklausų kiekius ir būtinybė greitai reaguoti į infrastruktūros poreikius [50]. *Khan Academy* sėkmingai pritaikė GAE kaip pagrindinę savo *serverless* infrastruktūrą, siekdama efektyviai išspręsti augančios vartotojų apkrovos ir greito plėtimosi iššūkius. Jų architektūrinis sprendimas, grindžiamas automatizuotu mastelio keitimu ir minimaliu infrastruktūros valdymu, leido ne tik palaikyti milijonų naudotojų srautą, bet ir užtikrinti aukštą našumą bei prieinamumą pasauliniu mastu. [51]

Vienas iš pagrindinių *Khan Academy* architektūros sprendimų buvo perėjimas prie GAE, siekiant išvengti rankinio serverių valdymo ir optimizuoti plėtros procesus. Jie nusprendė atsisakyti tradicinės serverių infrastruktūros ir naudoti GAE, nes ši platforma leidžia automatiškai plėsti išteklius pagal poreikį. Tai tapo itin svarbu pandemijos metu, kai svetainės naudojimas išaugo 2,5 karto per dvi savaites. GAE automatinio mastelio keitimo mechanizmas leido be trikdžių susidoroti su šiuo netikėtu srauto padidėjimu. *Khan Academy* taip pat panaudojo *Memcache* kaip kritinį talpyklos sprendimą. *Memcache* padėjo optimizuoti duomenų gavimo laiką, talpinant dažnai naudojamus duomenis atmintyje ir sumažinant apkrovą duomenų bazėms. Tai buvo ypač naudinga sprendžiant užklausas, susijusias su naudotojų sesijomis, preferencijomis bei dažniausiai užklausiamais duomenimis. Dėl šios optimizacijos buvo užtikrintas greitesnis atsakas ir sumažinta apkrova pagrindinėms duomenų saugykloms. Be *Memcache*, *Fastly CDN* taip pat buvo integruota į architektūrą, siekiant sumažinti serverio apkrovą ir užtikrinti greitesnį statinių duomenų, tokių kaip vaizdo įrašai ir nuotraukos, pristatymą naudotojams. CDN sprendimas leido nukreipti dalį užklausų nuo GAE instancijų, padidindamas bendrą sistemos efektyvumą ir sumažindamas serverio išteklių naudojimą. *Khan Academy* naudojo *Google Datastore*, kad valdyti didelius duomenų kiekius ir automatiškai skalauti duomenų saugojimo pajėgumus. *Datastore* yra *NoSQL* duomenų bazė, kuri leidžia efektyviai apdoroti duomenis ir prisitaikyti prie didėjančio užklausų srauto. Ši duomenų bazė automatiškai didina talpą ir prieigos pajėgumus, užtikrindama, kad naudotojų duomenys būtų saugomi ir apdorojami greitai bei efektyviai. [51] Naudodama GAE platformą, *Khan Academy* užtikrino ne tik mastelio keitimą, bet ir lengvą diegimo procesą bei veikimo priežiūrą. Kadangi GAE yra pilnai valdomas, kūrėjų komanda galėjo kasdien diegti atnaujinimus, net iki devynių kartų per dieną, nesirūpindama serverių administravimu. Tai leido komandos nariams sutelkti dėmesį į produkto tobulinimą, o ne infrastruktūros priežiūrą. [52]

Khan Academy pavyzdys aiškiai parodo, kaip efektyvus GAE architektūros panaudojimas gali padėti įmonėms lengvai prisitaikyti prie kintančių naudotojų poreikių. GAE automatinio mastelio keitimo, *Google Datastore*, *Memcache* sprendimai, kartu su *Fastly CDN* naudojimu, užtikrino, kad net ir didelių srautų metu sistema išliktų greita, patikima ir lengvai valdoma. Toks architektūrinis sprendimas padėjo *Khan Academy* efektyviai aptarnauti milijonus naudotojų visame pasaulyje, nereikalaujant didelės IT infrastruktūros komandos. [51] [52]

Apibendrinant, *Khan Academy* architektūrinis sprendimas, panaudojant GAE, atskleidžia, kaip *serverless* infrastruktūra, automatizuotas mastelio keitimas ir integruoti *Memcache, Google Datastore* bei CDN sprendimai gali padėti organizacijoms susidoroti su sparčiai augančiu srautu bei užtikrinti optimizuotą veikimą.

1.11. GAE programavimo sprendimai *Flexible* aplinkoje

Kaip šiame darbe buvo kalbama anksčiau, GAE *Flexible* aplinka suteikia galimybių įgyvendinti įvairius programavimo sprendimus, leidžiančius efektyviai valdyti ir optimizuoti programų veikimą debesų kompiuterijos platformose. Vienas iš tokių sprendimų yra HTTP srautinio perdavimo (angl. *streaming*) įgyvendinimas naudojant *Remix* karkasą. Šis programavimo sprendimas leidžia serveriui siųsti duomenis klientui dalimis, o tai itin naudinga apdorojant didelius duomenų kiekius ar užtikrinant greitą jų pateikimą vartotojui. [53]

Vienas iš pagrindinių šio sprendimo privalumų yra galimybė panaudoti srautą tam, kad naršyklė gautų duomenis ne laukiant viso atsakymo apdorojimo, o nuosekliai. Tai sumažina užklausų "krioklio" (angl. *request waterfall*) problemą, kuri būdinga vieno puslapio aplikacijoms (angl. *Single Page Apps*). *Remix* karkasas naudojamas tam, kad duomenų įkėlimas būtų sinchronizuojamas su komponentų atvaizdavimu, taip užtikrinant, kad klientas matytų pirmąją informaciją, kai tik ji tampa prieinama, net jei likę duomenys dar nėra visiškai apdoroti. Šis sprendimas realizuojamas naudojant keletą specifinių funkcijų *Remix* karkase. Pavyzdžiui, funkcija *defer* naudojama norint išsiųsti dalinę informaciją į naršyklę dar nebaigus viso užklausos apdorojimo. Kartu su *Suspense* komponentu, vartotojas gali matyti pirminius duomenis realiuoju laiku, o likusi informacija bus atnaujinama, kai tik ji bus paruošta. [53]

Esminis srautinio perdavimo realizavimo pavyzdys, remiantis [53] šaltiniu pateiktas *2 pav.* Šis kodo fragmentas demonstruoja asinchroninį duomenų pateikimą, naudojant *defer* ir *Suspense*, kad duomenys būtų perduodami ir atvaizduojami naršyklėje, kai tik jie tampa prieinami. Tai leidžia optimizuoti vartotojo patirtį ir sumažinti laukimo laiką. [53]

```
import { defer } from "@remix-run/node";
import { Await, useLoaderData } from "@remix-run/react";
import { Suspense } from "react";

export async function loader() {
    const pokemonJson = await import("../pokemon.json");
    return defer({ pokemonJson });}

export default function Pokemon() {
    const { pokemonJson } = useLoaderData<typeof loader>();

    return (
        <Suspense fallback={<p>loading</p>}>
            <Await resolve={pokemonJson}>
                {(data) => (
                    <ul>
                        {data.results.map((pokemon, index) => (
                            <li key={index}>{pokemon.name}</li>
                        ))}
                    </ul>
                )}
            </Await>
        </Suspense>
    );
}
```

2 pav. HTTP srautinio perdavimo realizacijos pavyzdys naudojant *Remix* karkasą.

Kaip buvo aptarta anksčiau, GAE *Flexible* aplinka suteikia didesnį lankstumą, palyginti su tradicine GAE *Standard* aplinka, nes ji leidžia naudoti platesnį programavimo kalbų ir įrankių spektrą bei suteikia galimybę lanksčiai konfigūruoti aplikacijų aplinką. Nors *Docker* konteinerių naudojimas nėra privalomas GAE *Flexible* aplinkoje, paslaugų konfigūravimas gali būti atliekamas tiesiogiai naudojant *app.yaml* failą. Pavyzdžiui, *Remix* karkasas kartu su *Node.js* gali būti sėkmingai diegiamas šioje aplinkoje, siekiant realizuoti HTTP srautinį perdavimą, nenaudojant *Docker* konteinerių, o tai leidžia efektyviau valdyti aplikacijos veikimą ir diegimą. [53]

Vienas iš iššūkių, susijusių su srautiniu perdavimu GAE *Flexible* aplinkoje, yra numatytasis atsakymų „buferiavimas“ (angl. *buffering*). Tai reiškia, kad platforma dažnai laiko atsakymus ir siunčia juos kaip vieną didelį duomenų bloką. Tačiau šį „buferiavimą“ galima išjungti, pridedant HTTP antraštę *X-Accel-Buffering: no*. Ši modifikacija užtikrina, kad duomenys bus siunčiami klientui nuosekliai, kai tik jie tampa prieinami, o tai yra būtina norint įgyvendinti tikrąjį srautinį perdavimą. [53]

Šis programavimo sprendimas, įgyvendinant HTTP srautinį perdavimą naudojant *Remix* karkasą GAE *Flexible* aplinkoje, parodo didelį lankstumą ir efektyvumą duomenų pateikimo

procesuose. Jis leidžia efektyviai valdyti didelius duomenų srautus ir sumažina vartotojų laukimo laiką, nes duomenys siunčiami dalimis realiuoju laiku. Be to, galimybė pritaikyti sprendimą be *Docker* konteinerių naudojimo supaprastina diegimo procesą ir suteikia daugiau pasirinkimo laisvės.

1.12. Programavimo sprendimas naudojant GAE *Memcached*

GAE suteikia galimybę automatizuoti resursų valdymą ir optimizuoti aplikacijų našumą. Našumo optimizacija yra ypač svarbi sistemose, kuriose valdomi dideli duomenų kiekiai ir aptarnaujami dideli srautai. *Khan Academy*, siekdama optimizuoti savo sistemos našumą ir duomenų užklausų greitį, įgyvendino sprendimus, susijusius su *Google App Engine Memcached* naudojimu. Pagrindinis šio sprendimo tikslas buvo pagerinti sistemos gebėjimą efektyviai valdyti didelius duomenų kiekius ir sumažinti užklausų vėlavimą, kuris galėtų neigiamai paveikti vartotojų patirtį. [54]

```python
@app.route('...')
@developer_required
def profile_memcached():
    num_bytes = int(request.args.get(num_bytes))
    data = os.urandom(num_bytes)
    key = 'profile_memcache_%s' % base64.b64encode(os.urandom(16))
    success = memcache.set(key, data)
    if not success:
        raise RuntimeError('Memcached set failed!')

    get_start = time.time()
    data_again = memcache.get(key)
    get_end = time.time()

    memcache.delete(key)
    return flask.jsonify(
        time=get_end - get_start,
        correct=data == data_again,
        key=key)
```

Šaltinis: [54].

3 pav. *Memcached* atsako vėlinimo matavimas GAE

Vienas iš iššūkių buvo optimizuoti prieigą prie turinio, kuris laikomas atmintyje (angl. *in-memory*). Kadangi *Khan Academy* palaiko įvairių kalbų turinio struktūras, reikėjo užtikrinti dinaminį turinio užkrovimą pagal poreikį. Naudoti *App Engine Memcached* buvo logiškas sprendimas, tačiau kilo klausimas, kaip tai paveiks našumą, ypač kai vienos užklausos metu reikia gauti keliasdešimt ar net šimtus turinio elementų. Norėdama suprasti, koks yra *Memcached* atsako laikas, *Khan Academy* sukūrė specialią API, leidžiančią matuoti *Memcached* užklausų greitį. Pateikta API leido įvertinti, kiek laiko trunka duomenų gavimas iš *Memcached*. [54] Įgyvendinta *Memcached* atsako laiko matavimo funkcija pateikiama *3 pav.*

Atlikus daugiau nei 10 000 API užklausų, rezultatai parodė, kad užklausų vėlinimas daugeliui užklausų buvo tarp 1-4 ms, o didesniems duomenims (iki 1 MB) vėlinimas siekė nuo 11 iki 21 ms. Ši informacija buvo itin svarbi tolimesnei optimizacijai, nes padėjo suprasti, kurie duomenų kiekiai sukelia didžiausią vėlinimą. [54]

Remdamasi gautais duomenimis, *Khan Academy* atliko simuliacijas, siekdama išbandyti įvairias talpyklos strategijas. Viena iš strategijų buvo naudoti *multi-get* operacijas, leidžiančias vienu metu užklausyti kelis duomenų elementus. Simuliacijos parodė, kad *multi-get* našumas buvo panašus į pavienių užklausų našumą, todėl komanda nusprendė optimizuoti kodą taip, kad susijusios užklausos būtų apdorojamos kartu, mažinant bendrą vėlinimą. Be to, buvo eksperimentuojama su įvairiomis talpyklos strategijomis, įskaitant LRU (angl. *Least Recently Used*) *cache* [1] ir fiksuotos talpyklos strategijas, kai tam tikri turinio elementai buvo iš anksto laikomi talpykloje. Simuliacijos parodė, kad fiksuota talpyklos strategija veikė geriau nei LRU, nes ji sumažino užklausų vėlinimą. Eksperimentai taip pat parodė, kad daugiau nei 10 % užklausų sulėtėjo bent 50 %, kai kiekvienas turinio elementas buvo paimtas iš *Memcached* pirmą kartą atliekant užklausą. Galiausiai, *Khan Academy* optimizavo savo sistemą, naudodama *multi-get* operacijas, kai buvo užklausiamas visas turinio puslapis. Ši strategija leido vienu metu užkrauti visus puslapyje reikalingus turinio elementus, taip sumažinant atskirų elementų užklausų skaičių ir pagerinant duomenų valdymo efektyvumą. [54]

Khan Academy atliktos *Memcached* našumo analizės ir simuliacijos parodė, kad tinkamai naudojamos *multi-get* operacijos ir talpyklos strategijos gali reikšmingai sumažinti sistemos vėlinimą ir pagerinti vartotojų patirtį. „Profilavimo“ ir simuliavimo įrankiai leido komandoms išbandyti įvairias strategijas prieš jas diegiant gamybos aplinkoje, taip taupant laiką ir išteklius. Šis programavimo sprendimas padėjo optimizuoti sistemos našumą ir sumažinti užklausų vykdymo laiką, ypač apdorojant daugybines užklausas.

1.13. Saugumo priemonių poveikis debesijos sistemų našumui

Debesijos kompiuterija, užtikrinanti lankstumą ir įvairiapusį funkcionalumą, tapo esminiu šiuolaikinių informacinių sistemų komponentu, leidžiančiu efektyviai valdyti išteklius ir saugoti duomenis debesyje. Tačiau debesijos aplinkoje kylantys duomenų saugumo ir sistemos našumo iššūkiai išlieka svarbiausi klausimai. Šios dvi savybės yra glaudžiai susijusios, nes papildomi

[1] LRU cache yra laikina duomenų saugykla, naudojama stebėti neseniai pasiektus elementus ir, kai talpykla pasiekia savo talpos ribą, pašalinti mažiausiai neseniai naudotus elementus, kad atlaisvintų vietos naujiems duomenims. [112]

saugumo mechanizmai, tokie kaip šifravimas ir autentifikacija, dažnai reikalauja papildomų resursų, o tai gali turėti įtakos sistemos atsako laikui bei resursų naudojimui. [55]

Našumo vertinimas debesijos aplinkoje apima kokybės rodiklių, tokių kaip atsako laikas ir sistemos stabilumas, analizę. Papildomos saugumo priemonės gali įtakoti šiuos rodiklius, nes jos didina skaičiavimo resursų poreikį. Tokia sąveika tarp našumo ir saugumo leidžia įvertinti, kaip įvairios priemonės, pavyzdžiui, šifravimas ir autentifikacija, paveikia bendrą sistemos veikimą. [56]

Saugumo priemonių integravimas į debesijos aplinką yra esminis užtikrinant duomenų apsaugą. Šifravimo algoritmai, tokie kaip *Fernet Encryption,* padeda apsaugoti duomenis nuo nesankcionuotos prieigos, o autentifikacijos sprendimai, tokie kaip *Firebase Authentication*, užtikrina vartotojų tapatybės patvirtinimą. Tačiau šių sprendimų įgyvendinimas kelia papildomų našumo iššūkių, nes jie gali padidinti atsako laiką ir sistemos apkrovą. [55] [57]

Apibendrinant, galima teigti, kad papildomų saugumo priemonių poveikis sistemų našumui yra kompleksinis, reikalaujantis detalesnės analizės. Šiame darbe bus atliekamas tyrimas, kuriame vertinamas *Fernet Encryption* ir *Firebase Authentication* integracijos poveikis sistemų atsako laikui, resursų panaudojimui ir stabilumui esant skirtingoms apkrovos sąlygoms. Tyrimo rezultatai, pateikti 4 skyriuje, leis geriau suprasti šių priemonių įgyvendinimo poveikį praktikoje ir jų svarbą kuriant efektyvias bei saugias debesijos sprendimų architektūras.

1.14. Analizės dalies išvada

Atliekant analitinę dalį, buvo išsamiai įvertintos GAE platformos pritaikymo galimybės, apimant jos architektūrinius sprendimus, programavimo metodus bei integruotas saugumo priemones ir jų poveikį sistemos našumui bei duomenų apsaugai. Debesų kompiuterijos raidos analizė atskleidė, kaip GAE įsitvirtino šiuolaikinėje IT infrastruktūroje, ypatingą dėmesį skirdama *serverless* modelio privalumams ir automatinio mastelio keitimo galimybėms. Detali architektūros ir komponentų analizė parodė, kad GAE efektyviai valdo išteklius ir užtikrina aukštą paslaugų prieinamumą, integruojant tokias paslaugas kaip *Memcached* ir *Cloud Datastore*, kurios pagerina sistemos našumą ir duomenų prieinamumą.

Vertinimo kriterijų taikymas leido identifikuoti GAE stiprybes ir silpnybes, ypač susijusias su našumu ir saugumu, bei nustatyti optimizavimo galimybes praktiniuose sprendimuose. Empiriniai tyrimai ir praktiniai atvejai, tokie kaip *Khan Academy* integracija su GAE, patvirtino, kad platforma geba efektyviai valdyti dideles apkrovas ir užtikrinti aukštą duomenų saugumo lygį, nepaisant

papildomų saugumo priemonių įtakos sistemos našumui. Be to, nustatyta, kad pažangūs šifravimo ir autentifikacijos sprendimai leidžia pasiekti aukštą duomenų apsaugos lygį, tuo pačiu išlaikant patenkinamą našumą, pasinaudojant GAE automatinio mastelio keitimo galimybėmis.

Galutiniai tyrimo rezultatai rodo, kad GAE yra patikimas ir efektyvus PaaS sprendimas, tinkamas įvairioms verslo ir techninėms reikmėms, teikiantis aukštą našumą, lankstumą ir duomenų apsaugą. Tai patvirtina, kad GAE gali būti vertinga debesijos kompiuterijos platforma, atitinkanti šiuolaikinių informacinių sistemų reikalavimus dėl našumo, patikimumo ir saugumo.

2. PROJEKTINĖ DALIS

Šiame skyriuje pristatomas tyrimo planas, skirtas baigiamojo darbo įgyvendinimui. Taip pat pateikiamos kuriamų internetinių svetainių, kurios bus naudojamos tyrimui atlikti, specifikacijos ir jų realizacijos procesą iliustruojančios UML diagramos.

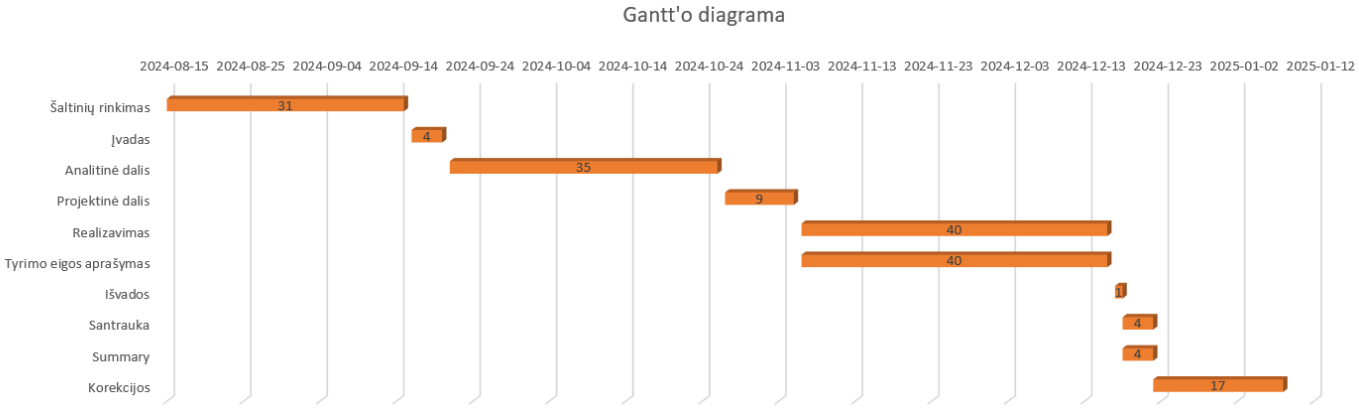
2.1. Projekto planas

Prieš pradedant kurti internetines svetaines, buvo būtina detaliai išnagrinėti GAE aplinkos savybes, saugos mechanizmus ir kitus galimus programavimo sprendimus. Buvo sudarytas išsamus darbo planas, pagal kurį vykdyta magistrinio darbo analitinė dalis ir pats tyrimas.

Darbo pradžia: 2024-08-15;

Darbo pabaiga: 2025-01-08;

Darbo trukmė: 185 d;



Šaltinis: sudaryta autoriaus.

4 pav. Darbo planas pavaizduotas *Gantt'o* diagramoje

Darbo eigai koordinuoti buvo pasirinkta *Gantt'o* diagrama (*4 pav.*), kuri padėjo vizualizuoti užduočių seką ir laiko paskirstymą. Parengtas planas sudarė galimybes kryptingai vykdyti užduotis, aiškiai apibrėžiant laiką, skirtą kiekvienai darbo daliai bei tyrimo etapams.

2.2. Sistemų projektavimas

Baigiamojo darbo tema nenumatė specifinių reikalavimų kuriamų sistemų tipui, todėl nuspręsta realizuoti paprastesnio pobūdžio internetines svetaines. Šis sprendimas buvo priimtas siekiant susitelkti į tyrimo realizaciją, o ne į pačių sistemų sudėtingumą ar inovatyvumą.

Projektavimo etape buvo parengtos svetainių specifikacijos, kurios pateikiamos prieduose (žr. **8 PRIEDAS**). Šių specifikacijų parengimas leido tiksliai apibrėžti reikalavimus ir užtikrinti, kad būtų laikomasi iš anksto numatytos idėjos bei planų.

Prieš pradedant realizuoti internetines svetaines, suprojektuoti pagrindiniai svetainių langų šablonai, kuriuose pateikiama pagrindinė funkcija, naujienų atvaizdavimas. Taip pat suprojektuoti eksperimentinių svetainių, kurios turės autentifikacijos simuliuotą arba realų mechanizmą, registracijos, prisijungimo langai, bei langas, kuris bus matomas po sėkmingo prisijungimo. Suprojektuoti šablonai pateikiami **9 PRIEDASE.**

UML diagramos buvo naudojamos detaliam sistemų funkcionalumo aprašymui. Projektavimo metu buvo sukurtos vartotojo panaudos atvejų diagramos, kuriose apibrėžiami galimi vartotojo veiksmai svetainėje, taip pat veiksmai, reikalaujantys papildomų sąlygų. Šios diagramos uždavinys – padėti išlaikyti fokusą į reikalavimų specifikacijos išpildymą. Taip pat buvo parengtos autentifikacijos funkcionalumo vartotojo veiklos diagramos, kurios vizualizuoja veiksmų seką ir jų sąveiką. Galiausiai, diegimo diagramose pavaizduoti naudojami mazgai ir komponentai, taip pat jų tarpusavio ryšiai bei protokolai. Visos projektavimo metu sukurtos UML diagramos yra pateikiamos **10 PRIEDASE.**

Apibendrinant, parengtas tyrimo planas, internetinių svetainių specifikacijos ir UML diagramos sudarė aiškią tyrimo įgyvendinimo struktūrą. Šis planavimas leido efektyviai valdyti laiką ir išlaikyti kūrimo proceso kryptingumą, neprarandant pagrindinės vizijos.

2.3. Technologinių sprendimų pasirinkimo pagrindimas

Eksperimentiniame tyrime buvo pasirinkta naudoti GAE standartinę aplinką, siekiant sumažinti tyrimo išlaidas. GAE suteikia nemokamą 90 dienų bandomąjį laikotarpį su 270 € kreditu, kuris leido efektyviai paskirstyti biudžetą. Papildomai, norint praturtinti tyrimo infrastruktūrą, vietoj nemokamo integruoto *Memcache* buvo pasirinktas valdomas *Memcached* sprendimas, kurio kaina,

pasirinkus mažiausius parametrus, siekia 37,71 € per mėnesį. Paskaičiuota, kad gautų lėšų pakaks tyrimo įgyvendinimui.

GAE standartinė aplinka, kaip aptarta šio darbo analitinėje dalyje, palaiko *Go, Java, Node.js, PHP, Python* ir *Ruby* programavimo kalbas. Šiame projekte pasirinkta *Python* kalba dėl jos dominavimo 2024 m. programavimo kalbų reitinguose.

Remiantis *IEEE Spectrum 2024* ataskaita, *Python* išlaiko lyderio pozicijas dėl savo populiarumo, palaikomų bibliotekų ir plačių pritaikymo galimybių. Tai ypač lemia *Python* naudojimas tokiose aktualiose srityse kaip dirbtinis intelektas bei jos išskirtinė pedagoginė reikšmė – dauguma pradedančiųjų programuotojų pirmiausia išmoksta būtent šią kalbą. *Python* taip pat pasižymi paklausa darbo rinkoje, kurioje ji užima svarbią vietą kaip universali kalba, dažnai naudojama kartu su *SQL* debesų ir tinklų architektūrose. [58]

Projektui įgyvendinti GAE standartinėje aplinkoje pasirinktas *Flask* karkasas dėl jo lengvos ir minimalistinės struktūros, kuri leidžia efektyviai realizuoti pagrindinį funkcionalumą. *Flask* išsiskiria savo paprastumu ir lankstumu, todėl jis yra itin tinkamas pasirinkimas mažiems ir vidutinio dydžio projektams, kuriuose būtinas greitas vystymo procesas. *Flask* suteikia galimybę lengvai integruotis su įvairiais *front-end* karkasais, užtikrinant dinamišką naudotojo patirtį. Jo lengvumas ir mažesnis funkcijų kiekis, lyginant su sudėtingesniais karkasais, tokiais kaip *Django*, leidžia išvengti perteklinių konfigūracijų ir resursų naudojimo, taip optimizuojant kūrimo procesą. [59]

Fernet šifravimas buvo pasirinktas dėl savo paprastumo, efektyvumo ir saugumo, kurie yra būtini šio projekto reikalavimai. *Fernet* įgyvendina simetrinį rakto šifravimą, užtikrindamas, kad pranešimai negali būti perskaityti ar pakeisti be atitinkamo rakto. Jis naudoja *Advanced Encryption Standard* su *Cipher Block Chaining* režimu bei HMAC su SHA-256 algoritmu, užtikrindamas duomenų konfidencialumą, vientisumą ir autentiškumą. Šis metodas yra lengvai įgyvendinamas ir tinkamas įvairaus dydžio projektams, reikalaujantiems saugios bei patikimos šifravimo sistemos. *Fernet* taip pat palaiko rakto rotaciją, o tai užtikrina papildomą saugumo lygį ilgalaikiam raktų naudojimui. Paprasta sąsaja ir mažas įgyvendinimo sudėtingumas leidžia efektyviai integruoti šį šifravimo sprendimą į projektą. [60] [61]

Autentifikacijos mechanizmui įgyvendinti buvo pasirinktas *Firebase Authentication,* nes tai yra patogiausias ir efektyviausias būdas nustatyti vartotojų autentifikaciją GAE aplikacijoms, kaip nurodoma oficialioje *Google Cloud* dokumentacijoje. *Firebase Authentication* leidžia greitai realizuoti prisijungimo funkcionalumą naudojant minimalų kodo kiekį, kartu užtikrinant sklandų sistemos veikimą. *Firebase* palaiko įvairius prisijungimo būdus: vartotojo vardą ir slaptažodį; „federuotą“ prisijungimą su *Google, Facebook, Twitter* ir kitais tiekėjais. *Firebase Authentication*

suteikia galimybę lengvai valdyti didelius vartotojų kiekius, tuo pačiu užtikrinant sistemos patikimumą ir našumą. *Firebase Authentication* yra pripažintas kaip lengviausias būdas įgyvendinti autentifikaciją GAE aplikacijoms, nes abu sprendimai priklauso *Google Cloud* ekosistemai. [43]

Automatiniam apkrovos testavimui pasirinktas *Locust* įrankis dėl jo paprastumo, lankstumo ir galimybės kurti scenarijus naudojant *Python* programavimo kalbą. *Locust* išsiskiria savo įvykių pagrindu veikiančia architektūra, kuri, palyginus su gijų pagrindu veikiančiais įrankiais, sunaudoja apie 70% mažiau resursų. Šis pranašumas leidžia efektyviau naudoti sistemos resursus atliekant apkrovos testus. *Locust* taip pat palaiko paskirstytą testavimą, kuris suteikia galimybę imituoti tūkstančius virtualių naudotojų. Testavimo procesą galima stebėti realiu laiku naudojant *web* sąsają, kurioje pateikiami sistemos atsako laikai ir kiti svarbūs rodikliai. Šis funkcionalumas užtikrina aiškų ir patogų duomenų stebėjimą bei analizę. Be to, *Locust* aktyviai palaiko atvirojo kodo bendruomenė, o jo populiarumą patvirtina 23,4 tūkst. žvaigždučių *GitHub* platformoje. Šie rodikliai rodo įrankio patikimumą ir plačią naudotojų paramą. Dėl šių savybių *Locust* yra tinkamas pasirinkimas atliekant apkrovos testavimą, užtikrinant efektyvų sistemos našumo vertinimą naudojant mažiau resursų. [62]

Tokiu būdu pasirinkti technologiniai sprendimai atitinka projekto reikalavimus, užtikrindami efektyvų kūrimo procesą, saugų duomenų valdymą bei patikimą autentifikacijos funkcionalumą.

2.4. Projektinės dalies išvados

Projektinėje dalyje buvo sukurta nuosekli tyrimo įgyvendinimo struktūra, apimanti išsamų darbo planą, sistemų projektavimą ir tinkamų technologinių sprendimų pasirinkimą. Naudojant *Gantt'o* diagramą, buvo efektyviai koordinuojamos užduotys ir laikotarpiai, užtikrinant struktūrizuotą darbo eigą per visą tyrimo laikotarpį. Sistemų projektavimo etape buvo parengtos tikslios internetinių svetainių specifikacijos ir UML diagramos, kurios suteikė aiškų funkcionalumo ir architektūros supratimą, leidžiantį efektyviai įgyvendinti kūrimo procesą. Technologinių sprendimų pasirinkimo pagrindimas leido pasirinkti optimalias priemones – GAE standartinę aplinką, *Python* programavimo kalbą, *Flask* karkasą, *Fernet* šifravimo metodą, *Firebase Authentication* autentifikacijos sprendimą bei *Locust* apkrovos testavimo įrankį. Pasirinkti technologiniai sprendimai užtikrino pasirengimą tolimesnei techninei realizacijai ir tyrimo įgyvendinimui.

3. REALIZAVIMAS

Šiame skyriuje aptariama, kokia buvo pasirinkta techninė ir programinė įranga tyrimo realizacijai bei tai, kaip buvo realizuojamas apkrovos testavimas, generuojant virtualius vartotojus ir užklausas.

3.1. Programinė, techninė įranga ir testinė aplinka

Projekto įgyvendinimui buvo naudotas asmeninis *Lenovo Y700* nešiojamasis kompiuteris, turintis *Intel Core i7* procesorių ir 16 GB operatyviosios atminties (RAM). Nors tokio našumo kompiuterio internetinės svetainės kūrimui nereikia, šis pasirinkimas buvo padarytas dėl jo prieinamumo ir galimybės dirbti be našumo apribojimų, ypač atliekant užduotis, kurioms gali prireikti didesnių resursų. Kompiuteryje veikia 64 bitų *Windows 10* operacinė sistema, kuri užtikrina sklandų programinės įrangos palaikymą ir sistemos stabilumą.

Pradinis aplikacijų kūrimo ir funkcionalumo testavimo etapas buvo vykdomas lokalioje aplinkoje. Testinei aplinkai paruošti buvo naudojamos šios priemonės:

- *Python 3.9* – programavimo kalba;
- *Google Cloud SDK* – skirtas GAE aplikacijų kūrimui, diegimui ir valdymui;
- *Virtualenv* – izoliuotoms *Python* aplinkoms sukurti;
- *Firebase CLI* – *Firebase* funkcijų ir emuliatorių valdymui lokalioje aplinkoje.

Saugumo sprendimų realizacijai buvo naudojamos šios programinės bibliotekos:

- *Cryptography* – duomenų šifravimui ir dešifravimui naudojant *Fernet* algoritmą, užtikrinantį duomenų konfidencialumą, vientisumą ir saugumą.
- *Firebase-admin* – vartotojų autentifikacijos procesams valdyti, leidžianti integruoti *Firebase Authentication* sprendimus.

Naudojant minėtą programinę įrangą ir bibliotekas, buvo sukurtos keturios aplikacijų versijos, atitinkančios tyrimo tikslus ir eksperimentinio dizaino reikalavimus:

- *Control-app* – kontrolinė versija, naudojanti *Memcached* be papildomų saugumo mechanizmų.

- *Encrypt-app* – eksperimentinė versija, pritaikanti duomenų šifravimą su *Fernet* prieš juos talpinant *Memcached*.
- *Control-app-v2* – kontrolinė versija, naudojanti *Memcached*, įgyvendinanti simuliuotą vartotojų autentifikacijos procesą be realaus autentifikacijos mechanizmo.
- *Firebase-auth-app* – eksperimentinė versija, kurioje integruota *Firebase Authentication* vartotojų autentifikacija kartu su *Memcached*.

Control-app ir *encrypt-app* versijų sukurtą failų struktūrą sudaro *app.yaml* (diegimo konfigūracijos failas, nustatantis GAE aplinką, resursus bei paslaugas), *main.py* (pagrindinis aplikacijos kodo failas), *requirements.txt* (bibliotekų, reikalingų aplikacijos veikimui, sąrašas) ir du katalogai: *templates* (talpinami vartotojo sąsajai skirti failai), kuriame patalpintas *index.html* failas, ir *static* (talpinami stiliaus failai), kuriame patalpintas *styles.css*. *Control-app-v2* ir *firebase-auth-app* failų struktūra atrodo identiškai, tik *templates* katalogas talpina daugiau papildomų *html* failų vartotojų autentifikacijai įgyvendinti: *index.html, login.html, register.html*.

Testavimo etape buvo tikrinamas aplikacijų funkcionalumas lokalioje aplinkoje be iš anksto suformuluotų testavimo scenarijų. Buvo patikrintas visas pagrindinių funkcijų veikimas: naujienų gavimas iš RSS srauto, duomenų talpinimas *Memcached* ir šifravimo funkcionalumas. Sukurtos ir sukonfigūruotos aplikacijos buvo paruoštos tolimesniam diegimui GAE standartinėje aplinkoje.

3.2. Produkcijos aplinkos paruošimas ir aplikacijų perkėlimas į GAE

Prieš perkeliant aplikacijas į GAE, reikėjo atlikti paruošiamuosius darbus. Vienas iš jų – GAE standartinės aplinkos projekto sukūrimas (regionas: *us-central1*). Sukonfigūruotas *Google Cloud Memorystore* sprendimas *Memcached* pagrindu (*regionas: us-central1*, mazgų (angl. *nodes*) skaičius: 1, atmintis per mazgą: 1 GB, procesoriaus branduoliai per mazgą: 1 vCPU). Pasirinktas vienodas regionas tiek standartinei aplinkai, tiek *Memorystore* sprendimui talpinti, kad būtų užtikrintos kiek įmanoma vienodesnės sąlygos. Įjungta ir sukonfigūruota papildoma *Google Cloud* paslauga *Firebase Autentification*. Įgalinta *Secret Manager* paslauga, kurioje patalpinti sukurti ir sukonfigūruoti slapti duomenys (*Fernet* šifravimo raktai, *Firebase API* raktas, *Flask* raktas, *Memcached* serverio IP adresas). Visiems slaptiems duomenims saugoti taip pat pasirinkta *us-central1* lokacija. Taip pat nustatyta *VPC Access Connector* jungtis, leidžianti aplikacijoms saugiai bendrauti su *Memcached* serveriu. GAE projektui suteiktos teisės prie papildomai sukonfigūruotų paslaugų naudojant *IAM & ADMIN*. Paruošus produkcinę GAE aplinką, visos aplikacijų versijos perkeltos į GAE naudojant Google Cloud SDK komandą *gcloud app deploy*.

3.3. Apkrovos testų realizacija

Apkrovos testai buvo vykdomi toje pačioje lokalioje aplinkoje, kurioje buvo kuriamos ir testuojamos aplikacijos. *Locust* įrankis įdiegtas naudojant *Python* paketų tvarkyklę *pip*, o testiniai scenarijai realizuoti *Python* kalba, pasitelkiant *Locust* programavimo sąsają (*HttpUser*, *TaskSet*, *@task* dekoratorius, laukimo laiko (*wait_time*) parametrus, ir t. t.). Testų paleidimas ir rezultatų rinkimas buvo automatizuotas naudojant *PowerShelL* scenarijus. Pastarieji leido vienu komandos paleidimu vykdyti kelis testavimo etapus su skirtingais parametrų rinkiniais, taip užtikrinant pakartojamumą ir mažinant rankinių veiksmų kiekį.

Viena esminių eksperimento dalių buvo realaus autentifikacijos mechanizmo testavimas (*Firebase Authentication*). Siekiant atlikti apkrovos testus sąlygomis, panašiomis į realias, reikėjo sukurti gausų kiekį testinių vartotojų. Tai leido ne tik tiksliai imituoti realių vartotojų elgseną, bet ir patikrinti, kaip sistema reaguoja į intensyvų autentifikacijos bei registracijos užklausų srautą.

Tam buvo sukurtas atskiras *Python* skriptas, kuris:

3.3.1. Sugeneruoja unikalius vartotojų duomenis:

3.3.1.1. Kiekvienam testiniam vartotojui buvo sugeneruoti unikalūs el. pašto adresai, sudaryti iš atsitiktinių raidžių ir skaičių kombinacijų, pridedant *@test.com* priesagą. Slaptažodžiai parinkti vienodi (pvz., TestPassword123!), kad būtų užtikrintas paprastas prisijungimo proceso modeliavimas.

3.3.2. Integracija su *Firebase Authentication REST API*:

3.3.2.1. Naudojant *Firebase Authentication REST API* ir pateikiant projekto API raktą, sukurtas skriptas automatiškai registravo naujus vartotojus *Firebase* platformoje. Kiekvienam sugeneruotam vartotojui atlikta *signUp* užklausa, grąžinanti sėkmingai užregistruoto vartotojo duomenis.

3.3.3. Klaidų valdymas ir atkartojamumas:

3.3.3.1. Skriptas buvo suprogramuotas taip, kad bet kokių klaidų (pvz., *TOO_MANY_ATTEMPTS_TRY_LATER*) atveju nutrauktų registracijos procesą ir išsaugotų jau sukurtų vartotojų sąrašą, užtikrinant testinės aplinkos stabilumą bei atkartojamumą.

3.3.4. Duomenų išsaugojimas *JSON* faile:

3.3.4.1. Visi sėkmingai sukurti vartotojai buvo išsaugoti *JSON* faile *test_users.json*. Toks formatas leidžia paprastai pakrauti vartotojų duomenis „Locust“ testų metu ir juos naudoti prisijungimo, registracijos bei kitose autentifikacijos scenarijų dalyse.

Tokiu būdu sukurta gausi testinių vartotojų bazė (pvz., 5000 vartotojų), išlaikant patogų bei paprastą duomenų pasiekiamumą. Tai užtikrino, kad apkrovos testų metu ne tik būtų generuojamos anoniminės užklausos, bet ir imituojami realūs vartotojų autentifikacijos veiksmai. *JSON* failus *Locust* testų failai įkeldavo prieš pradedant apkrovos generavimą *Control-app-v2* ir F*irebase-app* aplikacijoms.

Kiekvienam iš keturių aplikacijos variantų (*Control-app, Encrypt-app, Control-app-v2, Firebase-auth-app*) buvo sukurti atskiri *Locust* scenarijai. Scenarijuose kiekvienas virtualus vartotojas yra aprašomas kaip *HttpUser* klasės egzempliorius, kuriame nurodomos testinės užklausos ir galimos vartotojo elgsenos savybės. Funkcijos, pažymėtos *@task* dekoratoriais, aprašė specifinius vartotojo veiksmus: puslapio peržiūrą, registraciją, prisijungimą, atsijungimą. Kiekviena iš šių funkcijų imituoja realias HTTP užklausas į testuojamą aplikaciją. Naudojant *wait_time = between(x, y)* nustatyta, kad vartotojai darytų atsitiktines pertraukas tarp veiksmų. Tai leido priartėti prie realių sistemos naudojimo sąlygų, kai vartotojai neskuba nuolat siųsti užklausų, bet skaito turinį, peržiūri puslapį ar laukia kitų veiksmų.

Kad apkrovos testai geriau atspindėtų realias naudojimosi sistema sąlygas *Control-app-v2* ir F*irebase-app* aplikacijose, buvo pasirinktos veiksmų proporcijos: 15,38% (registracija), 30,77% (prisijungimas), 23,08% (pagrindinio puslapio peržiūra neprisijungus), 23,08% (pagrindinio puslapio peržiūra prisijungus), 7,69% (atsijungimas). Toks modelis leidžia imituoti realią situaciją, kurioje didžiausią krūvį dažniausiai sudaro prisijungimai ir autentifikacijos būsenos palaikymas.

Įvertinant skirtingus sistemos apkrovimo scenarijus, buvo taikyti du testavimo metodai: testavimas su pertraukomis ir testavimas be pertraukų. Testavimas su pertraukomis vykdomas padalijant apkrovos procesą į kelis 10 minučių etapus, o tarp kiekvieno etapo daroma 3 minučių pertrauka, leidžianti sistemai „atsikvėpti“. Kiekviename etape didinamas virtualių vartotojų skaičius, sekant tvarkai: 50, 100, 200, 400, 800, 1000. Ši strategija leidžia stebėti, kaip sistema reaguoja į staigius apkrovos šuolius ir kaip jos būklė kinta, kai apkrova staiga sumažinama. Testavimas be pertraukų trunka 60 minučių be jokių pauzių. Pradinis vartotojų skaičius yra 4 per sekundę ir per minutę padidinimas vartotojų skaičius 150-čia. Vartotojų kiekis 150-čia yra didinamas kas 10 minučių, kol pasiekiamas 900 vartotojų kiekis. Šis metodas leidžia įvertinti ilgalaikį sistemos stabilumą nuolat augant apkrovai, atskleisti galimus našumo butelio kaklelius ir nustatyti, ar

papildomi saugumo mechanizmai, tokie kaip šifravimas ar autentifikacija, nelemia esminio ilgalaikio sistemos degradavimo. Visų testų paleidimas buvo automatizuotas naudojant *PowerShell* scenarijus, kuriuose iš anksto sukonfigūruoti parametrai (apkrovos lygis, testavimo trukmė, pauzių trukmės) ir nurodyti konkretūs *Locust* failai, atitinkantys skirtingus testavimo scenarijus. Po kiekvieno testo vykdymo rezultatai saugomi CSV formatu, taip pat generuojamos HTML ataskaitos. Surinkti duomenys apima pagrindinius metrikos rodiklius: vidutinį atsako laiką, atsakymų sėkmingumo procentą, klaidų dažnį bei kitus rodiklius. Šių duomenų analizė bus pateikta 4-ajame skyriuje, kuriame detaliai aptariama tyrimo eiga ir rezultatai.

Taigi, šiame poskyryje aprašyti apkrovos testų realizavimo principai, pasiruošimo darbai, testavimo scenarijų logika bei skirtingi testavimo metodai. Toks kompleksinis testavimo būdas, apimantis realių vartotojų elgsenos modeliavimą, autentifikacijos procedūras bei apkrovos didinimą tiek su pertraukomis, tiek ilgalaikėje perspektyvoje be pauzių, sudaro prielaidas visapusiškam sistemos našumo, stabilumo ir atsparumo vertinimui.

3.4. Techninės realizacijos išvada

Techninės realizacijos skyriuje buvo sėkmingai įgyvendintas tyrimo procesas, apimantis tinkamos techninės ir programinės įrangos parinkimą bei konfigūravimą. Pasitelkus asmeninį kompiuterį ir *Python* programavimo kalbą, sukurtos keturios aplikacijos versijos su integruotais saugumo ir autentifikacijos mechanizmais. Lokalioje testinėje aplinkoje buvo išbandytas aplikacijų funkcionalumas, paruošus jas diegimui GAE platformoje. Produkcinė aplinka buvo kruopščiai sukonfigūruota, įskaitant duomenų saugojimo sprendimus ir autentifikacijos integraciją. Apkrovos testavimo procesas buvo vykdomas naudojant *Locust* įrankį ir automatizuotus scenarijus, leidžiančius generuoti realistišką vartotojų srautą bei autentifikacijos užklausas. Sukurta testinių vartotojų bazė ir išplėtotas testavimo scenarijų rinkinys užtikrino išsamų sistemos našumo, stabilumo ir atsparumo vertinimą įvairiose apkrovos sąlygose. Automatizuotas testavimo procesas, pagrįstas *PowerShell* scenarijais, leido efektyviai vykdyti testus ir rinkti patikimus duomenis, kurie bus toliau analizuojami tyrimui skirtame skyriuje. Šio skyriaus metu sukonfigūruota infrastruktūra ir paruoštos aplikacijos našumo bei stabilumo vertinimui sudaro tvirtą pagrindą tolimesniems tyrimo etapams. Tai užtikrina, kad sistema atitinka nustatytus techninius reikalavimus ir yra pasirengusi veikti įvairiose operacinėse aplinkose, prisidedant prie visapusiško tyrimo tikslų pasiekimo.

4. TYRIMO EIGOS APRAŠYMAS

4.1. Tyrimo aprašymas

Tyrimo pavadinimas: papildomų saugos sprendimų poveikio, *Memcached* naudojančių aplikacijų, našumui tyrimas GAE standartinėje aplinkoje

Problema: augant debesijos paslaugų naudojimui, didėja reikalavimai duomenų saugumui ir sistemų našumui. *Memcached* plačiai taikomas siekiant spartesnės duomenų prieigos debesijos aplikacijose, tačiau nėra aišku, kaip papildomi saugos mechanizmai (pvz., šifravimas, autentifikacija) paveikia šias programas GAE standartinėje aplinkoje.

Tyrimo objektas: programų, naudojančių *Memcached* GAE standartinėje aplinkoje, našumo pokyčiai, integravus šifravimo ir autentifikacijos mechanizmus.

Hipotezė: papildomų saugos mechanizmų (šifravimo ir autentifikacijos) naudojimas sumažina programų našumą.

Tyrimo tikslas: įvertinti, kaip šifravimas ir autentifikacija veikia *Memcached* pagrindu veikiančių programų našumą GAE standartinėje aplinkoje, palyginant kontrolines ir eksperimentines grupes.

Tyrimo uždaviniai:

1. Sukurti keturias programų versijas, atspindinčias skirtingus saugos mechanizmų naudojimo scenarijus:
   - *Control-app* (bazinė versija be šifravimo ir autentifikacijos);
   - *Encrypt-app* (su šifravimu);
   - *Control-app-v2* (be šifravimo, bet su simuliuota autentifikacija);
   - *Firebase-auth-app* (su realia autentifikacija).
2. Išmatuoti visų programų versijų našumo rodiklius;
3. Palyginti rezultatus tarp kontrolinių ir eksperimentinių grupių.

Tyrimo metodai: kiekybinis (kontroliuojamas) eksperimentas. Našumo rodiklių stebėsena taikant nustatytas metrikas ir *Google Cloud Console Monitoring* priemones.

Tyrimo vertinimo kriterijai:

1. Šifravimo ir autentifikacijos poveikis užklausų vykdymo procesui tarp GAE ir *Memcached*.

2. Šifravimo ir autentifikacijos mechanizmų įgyvendinimo galimybės GAE standartinėje aplinkoje.
3. Šifravimo ir autentifikacijos įtaka atsako laikui ir užklausų apdorojimo greičiui.
4. Palyginimas tarp kontrolinių ir eksperimentinių grupių.

4.2. Eksperimento aprašymas

Eksperimentinis dizainas: keturios tiriamosios grupės – dvi kontrolinės (be papildomų saugos mechanizmų / su simuliuota autentifikacija) ir dvi eksperimentinės (su šifravimu / su realia autentifikacija). Kintamųjų kontrolė užtikrina vienodas sąlygas, išskyrus saugos mechanizmus. Tikslas – nustatyti, kaip nepriklausomi kintamieji (šifravimas, autentifikacija) veikia priklausomus kintamuosius (atsako laiką, CPU / atminties naudojimą).

Eksperimente dalyvaujančios grupės:

1. Grupė 1:
   1.1. *Control-app* (be šifravimo, be autentifikacijos);
   1.2. *Encrypt-app* (su šifravimu).
2. Grupė 2:
   2.1. *Control-app-v2* (be šifravimo, su simuliuota autentifikacija);
   2.2. *Firebase-auth-app* (su realia autentifikacija).

Kintamieji:

- Nepriklausomi: šifravimo ir autentifikacijos mechanizmų naudojimas.
- Priklausomi: užklausų apdorojimo greitis (atsako laikas), sistemos resursų (CPU, atminties) naudojimas.

Kintamųjų kontrolė:

- Visose aplikacijose naudojamas tas pats GAE resursų nustatymas;
- Vienas bendras *Memcached* egzempliorius.
- Vienodas užklausų pobūdis ir dažnis, valdant juos *Locust* įrankiu.

4.3. Įgyvendinimo ir testavimo eiga

Eksperimentui buvo sukurti keturi aplikacijų variantai, naudojant *Python 3.9* programavimo kalbą ir įdiegiant aplikacijas GAE standartinėje aplinkoje. Aplikacijos pasižymėjo identiška pagrindine logika, o duomenų talpyklai buvo naudojamas *Memorystore Memcached*, konfigūruotas su 1 GB atminties ir 1 vCPU. Visos aplikacijos buvo sukurtos naudojant *Flask* žiniatinklio karkasą, papildomai pasitelkiant *Secret Manager* (slaptų duomenų saugojimui) ir *Logs Explorer* (žurnalų stebėjimui ir analizei). Eksperimentinėse versijose integruoti papildomi saugos mechanizmai, įgyvendinti naudojant *Fernet* šifravimo biblioteką ir *Firebase Authentication*. Išsami technologijų ir saugos sprendimų apžvalga pateikiama prieduose (žr. **11 PRIEDASĄ**).

Sukurtų programų sąrašas pateikiamas *1 Lentelė 1*.

Lentelė 1

Sukurtų programų sąrašas

Aplikacjos versija	Aplikacijos tipas	Saugos mechanizmai	Funkcionalumas	URL
Control-app	Kontrolinė	Nėra	RSS duomenų talpinimas į „Memcached“	https://tyrimas.uc.r.appspot.com
Encrypt-app	Eksperimentinė (identiška control-app)	Šifravimas („Fernet“)	Šifruotas RSS duomenų talpinimas į „Memcached“	https://encrypt-app-dot-tyrimas.uc.r.appspot.com
Control-app-v2	Kontrolinė	Nėra	RSS duomenų talpinimas į „Memcached“; Simuliuotas autentifikavimas;	https://control-app-v2-dot-tyrimas.uc.r.appspot.com
Firebase-auth-app	Eksperimentinė (identiška control-app-v2)	Autentifikacija („Firebase“)	RSS duomenų talpinimas į „Memcached“; Tikra autentifikacija;	https://firebase-auth-app-dot-tyrimas.uc.r.appspot.com

Šaltinis: sudaryta autoriaus.

Testavimui ir analizei pasirinktos priemonės:

- *Locust* - apkrovos testavimo įrankis, generuojantis virtualius vartotojus ir užklausas.
- *PowerShell* skriptai – testavimų automatizavimui.
- *Google Cloud Console Monitoring* – sisteminių metrikų stebėjimui.
- *Knim*e ir *Excel* – duomenų analizei.

Testavimas atliktas dviem būdais:

1. Testavimas su pertraukomis:
   1.1. Kiekvienas apkrovos etapas trunka 10 minučių, tarp etapų – 3 minučių pertrauka.
   1.2. Vartotojų skaičius: 50, 100, 200, 400, 800, 1000.
   1.3. Vertinamas sistemos elgesys apkrovos šuolių metu ir tarp jų.
2. Testavimas be pertraukų:
   2.1. 60 min. trukmės testas, be pertraukų.
   2.2. Vartotojų skaičius didinamas palaipsniui (nuo 4 iki 900). Vartotojų didinimas vyksta kas 10 min. ir didinama po 150 vartotojų.
   2.3. Vertinamas ilgalaikės, nepertraukiamos apkrovos poveikis našumui.

Eksperimento metu buvo modeliuojama vartotojų elgsena ir apkrovos paskirstymas, siekiant imituoti realias sistemos naudojimo sąlygas. Elgsena buvo modeliuojama aplikacijai be realaus autentifikacijos mechanizmo ir aplikacijai su realiu autentifikacijos sprendimu, siekiant įvertinti jų poveikį sistemos našumui. Apkrovos paskirstymas buvo suplanuotas taip, kad atspindėtų dažniausiai vartotojų atliekamus veiksmus (žr. *Lentelė 2*).

Lentelė 2

Apkrovos paskirstymas tesatvimo metu *control-app-v2* ir *firebase-auth-app* aplikacijoms

Veiksmas	Procentinė dalis (%)	Papildoma informacija
Prisijungimas	30,77%	Pagrindinis autentifikacijos proceso elementas; dažniausiai atliekamas veiksmas.
Pagrindinio puslapio peržiūra	46,16%	Vienodai padalinta tarp prisijungusių (23,08%) ir neprisijungusių (23,08%) vartotojų.
Registracija	15,38%	Atliekama tik 15,38% vartotojų; dauguma jau turi paskyrą arba naršo neprisijungę.
Atsijungimas	7,69%	Retai atliekamas veiksmas; dauguma vartotojų uždaro naršyklę ar palieka svetainę.

Šaltinis: sudaryta autoriaus.

Išsamesnė informacija apie eksperimento apkrovų modeliavimą ir naudojamus failus pateikta **12 PRIEDASE**.

Tyrime naudojamos metrikos apima: vidutinį atsako laiką, atsako laiko medianą, didžiausią atsako laiką, užklausų per sekundę skaičių, klaidų dažnį, procentilinius atsako laikus (90%, 95%, 99%), atminties naudojimą, CPU naudojimą, talpyklos pataikymo santykį, tinklo srauto apimtis, atsako uždelsimą ir kitas našumo bei efektyvumo vertinimo priemones. Visas naudotų metrikų sąrašas ir jų apibrėžtys pateikiamos **13 PRIEDASE**.

Tyrimas apsiribojo GAE standartine aplinka ir *Memcached* technologija, vertinant pagrindines našumo metrikas. Kiti aplinkos tipai ar talpyklų sprendimai nebuvo įtraukti.

4.4. Tyrimo rezultatų analizė testavimo su pertraukomis metu

Eksperimentinio tyrimo metu buvo siekiama nustatyti papildomų saugos mechanizmų (duomenų šifravimo ir autentifikacijos) poveikį *Memcached* naudojančių aplikacijų veikimui GAE standartinėje aplinkoje. Našumo duomenys buvo surinkti naudojant *Locust* generuotus testavimo failus (*stats.csv*) ir analizuoti naudojant *Excel* įrankį. Papildomi sisteminiai rodikliai gauti iš *Google Cloud Console Monitoring*.

Šifravimo poveikio analizė (*control-app* ir *encrypt-app*)

Apibendrinant gautus *Locust stats.csv* failų suvestinės šifravimo tyrimui rezultatus (detali suvestinės rezultatų analizė pateikiama **14 PRIEDASE**) galime teigti, kad eksperimentinė aplikacija su šifravimu daugeliu atvejų demonstruoja ilgesnį vidutinį atsako laiką, didesnę atsako laiko medianą ir sumažintą apdorojamų užklausų kiekį per sekundę. Nors kai kuriose situacijose (pvz., esant 400 vartotojų apkrovai) eksperimentinė aplikacija pasižymėjo mažesniais didžiausiais ar procentiliniais atsako laikais, bendras našumo sumažėjimas buvo akivaizdus. Analizuojant procentilinių atsako laikų (90%, 95%, 99%) reikšmes, pastebėta, kad kontrolinė aplikacija dažniausiai pasižymėjo mažesnėmis reikšmėmis, o tai parodo, jog be šifravimo sistema efektyviau apdoroja daugumą užklausų. Tačiau kai kuriose apkrovos situacijose (pvz., esant 200, 400 ir 800 vartotojų apkrovai) eksperimentinė aplikacija pasižymėjo mažesniais 95% ir 99% atsako laikais, rodydama, kad šifravimas tam tikrais atvejais gali teigiamai paveikti užklausų pasiskirstymą. Vis dėlto abejose aplikacijose klaidų dažnis išliko 0 %, o tai rodo, kad sistema išlaikė stabilumą net ir taikant šifravimą. Šie rezultatai leidžia teigti, kad šifravimo mechanizmas, nors ir sukelia našumo mažėjimą, tuo pačiu užtikrina sistemos patikimumą ir tam tikrose situacijose padeda sumažinti „ekstremalių“ užklausų pikus.

Atlikus išsamią *Google Cloud Console Monitoring* grafikų analizę (grafikų analizė pateikiama **15 PRIEDASE**), galima nustatyti keletą svarbių dėsningumų, susijusių su aplikacijų, kurios naudoja arba nenaudoja šifravimo, (eksperimentinės ir kontrolinės) našumo rodikliais GAE bei *Memorystore Memcached* aplinkose.

GAE aplinkoje pastebima, kad eksperimentinės aplikacijos (taikančios „Fernet“ šifravimą) CPU apkrova išlieka stabili ir kai kuriais atvejais net mažesnė, palyginti su kontroline versija. Tai rodo, jog kriptografinės operacijos nėra reikšmingai susietos su ilgalaikiu procesoriaus ciklų skaičiaus didėjimu, o daugiausia pasireiškia per momentinius trumpalaikius šuolius, kuriuos galima sieti su rakto generavimu ar duomenų blokų šifravimu / iššifravimu. Tuo tarpu atminties sunaudojimas eksperimentinėje aplinkoje yra akivaizdžiai didesnis, ypač esant didesnei apkrovai (400, 1000 vartotojų). Šis rezultatas suponuoja, kad šifravimas reikalauja papildomų laikinosios saugyklos (buferių) resursų, tačiau bendrą sistemos stabilumą tai veikia nežymiai.

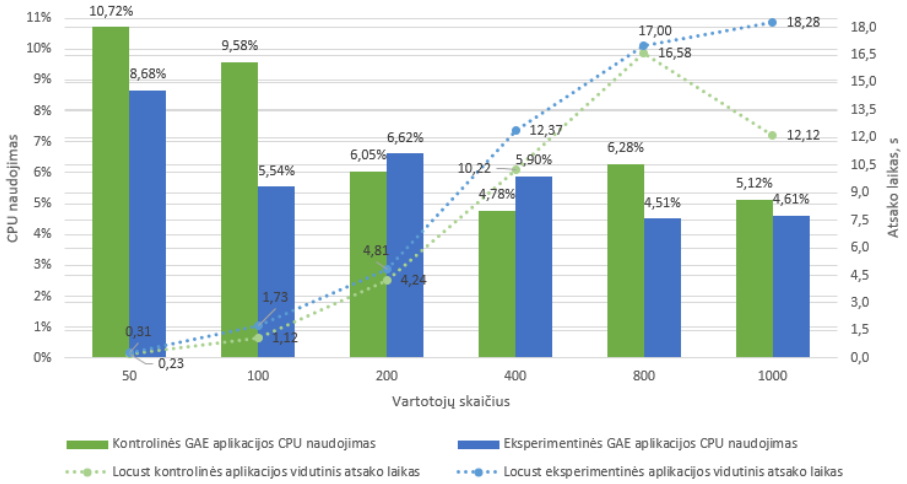
Vertinant GAE atsako laiką, matyti nedidelis vidutinio uždelsimo padidėjimas eksperimentinėje versijoje. Vis dėlto 95-asis procentilis rodo, jog didžioji dalis užklausų aptarnaujamos panašiu ar priimtinu greičiu kaip ir kontrolinėje sistemoje, o reti šuoliai (iki ~1,28 s) nėra dažni. Svarbu paminėti, kad eksperimentinėje versijoje vidutiniškai sumažėja išsiunčiamų duomenų srautas, o tai gali būti siejama su galimai suspausta ar kitokios struktūros šifruotų duomenų reprezentacija. Dėl to per laiko vienetą apdorojamų užklausų skaičius šiek tiek mažėja, tačiau tai neturi reikšmingo poveikio sistemos bendrai kokybei.

Memorystore Memcached aplinkoje neaptikta jokių neigiamų šifravimo integracijos poveikių. Talpyklos atminties naudojimas išlieka stabilus (~461 MiB), abiejose versijose nėra užfiksuota iškeldinimų (angl. *evictions*), o talpyklos pataikymų santykis (angl. *hit ratio*) išlieka artimas 100 %. Šios metrikos leidžia daryti išvadą, kad šifravimas nesutrikdo duomenų talpinimo, paieškos mechanizmų ir nesukelia papildomo *Memcached* resursų poreikio. Be to, CPU išteklių naudojimas *Memcache*d lygiu išlieka labai žemas (abiem aplinkoms artimas ~0,04–0,05 %/s intervale), tad kriptografinė logika, įgyvendinta aplikacijos lygmeniu, nesukelia pastebimos apkrovos pačiai talpyklos paslaugai.

Apibendrinant, detali metrikų analizė rodo, kad *Fernet* šifravimo integravimas neturi esminio neigiamo poveikio nei GAE, nei *Memorystore Memcached* aplinkų našumui. Nors pastebimas nedidelis atminties sąnaudų padidėjimas ir retkarčiais šiek tiek išaugęs uždelsimas, bendras CPU resursų vartojimas, duomenų srautų apimtys, talpyklos efektyvumas ir atsako laiko kokybės rodikliai išlieka stabilūs ir priimtini. Šios įžvalgos patvirtina, kad šifravimo technologiją galima integruoti į programinę įrangą, neaukojant pagrindinių sistemų charakteristikų ir išlaikant patenkinamą vartotojų

patirtį. Toks rezultatas yra itin svarbus, vertinant debesų kompiuterijos sprendimų saugumą magistriniuose tyrimuose, nes įgalina didesnį duomenų konfidencialumą, išlaikant pakankamai aukštą sistemos našumo lygį.

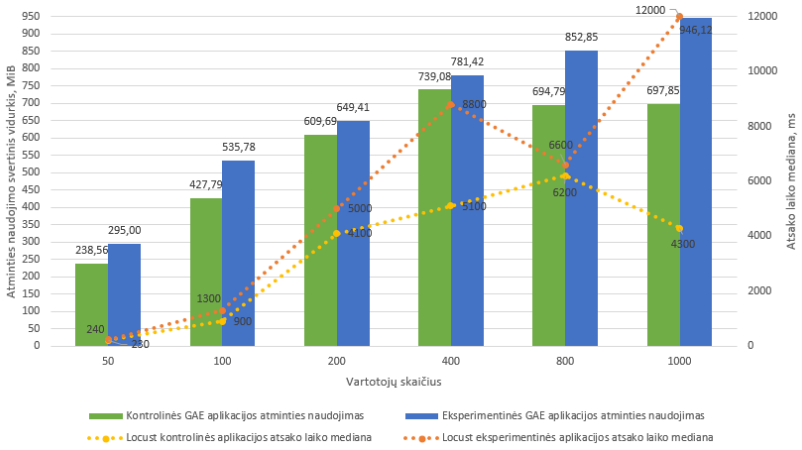
Galutinio naudotojo atsako laiko matavimai (*Locust*) koreliuoja su vidiniais sisteminiais rodikliais (CPU, atminties, talpyklos naudojimu) ir leidžia suprasti šifravimo įtaką sudėtingose, didelės apkrovos situacijose. Atlikta *Google Cloud Console Monitoring* grafikų analizė rodo (grafikų analizė pateikiama **15 PRIEDASE**), kad GAE aplinkoje ir *Memcached* lygmeniu šifravimo (*Fernet*) integracija nedarė reikšmingos neigiamos įtakos CPU apkrovai bei talpyklos veikimui. *Memcached node* lygmeniu CPU išteklių naudojimas išliko labai žemas ir stabilus (žr. *5 pav.*), o talpyklos atminties naudojimas nekito. Nebuvo aptikta ir talpyklos iškeldinimų, o *hit ratio* išliko beveik 100 %, tai rodo efektyvią talpyklos sistemą. Galima daryti išvadą, kad šifravimas nesujaukia bazinių resursų naudojimo principų: nei reikšmingai išauga CPU apkrova, nei sumažėja talpyklos efektyvumas.



Šaltinis: sudaryta autoriaus. Grafikas parodo CPU stabilumą kontrolinėje ir eksperimentinėje aplinkoje. Tuo pačiu iliustruoja, kad vidutinis atsako laikas kyla augant vartotojų skaičiui, tačiau CPU apkrova išlieka minimali.

5 pav. Koreliacijos diagrama tarp vidutinio atsako laiko (*Locust*) ir CPU naudojimo GAE aplinkoje.

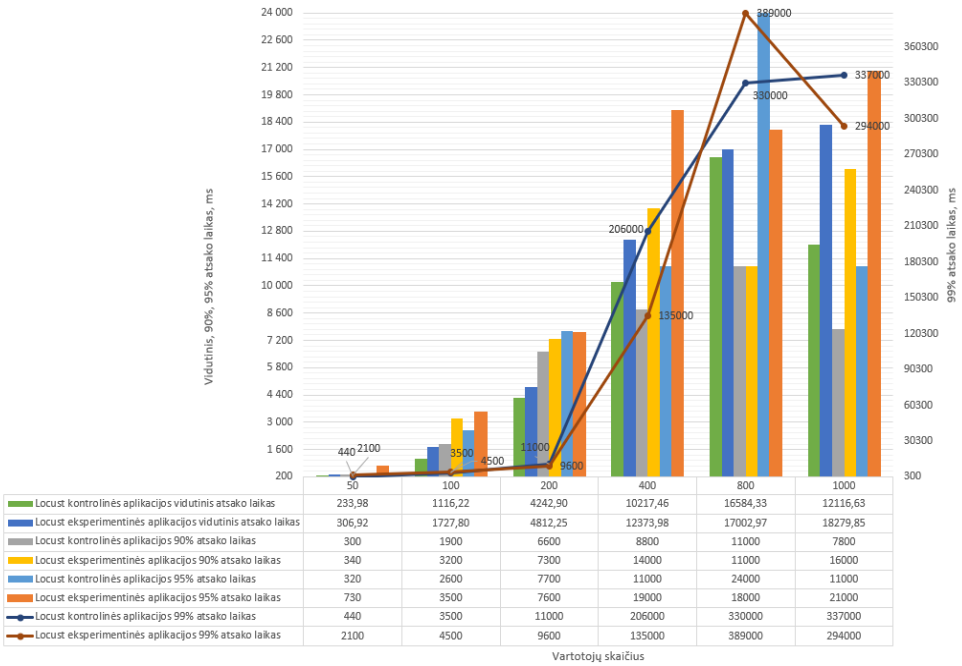
GAE aplinkoje CPU naudojimas aplikacijoje, naudojančioje šifravimą, ilgalaikėje perspektyvoje net šiek tiek mažėjo (žr. *5 pav.*), o atminties sunaudojimas padidėjo, bet ne iki kritinio lygio (žr. *6 pav.*) . Taigi, iš vidinių metrikų perspektyvos, sistema su šifravimu veikia stabiliai, o jos resursų naudojimas nėra drastiškai paveiktas.



Šaltinis: sudaryta autoriaus. Grafikas iliustruoja, kad atminties sunaudojimas padidėja (bet ne iki kritinio lygio).

6 pav. GAE aplikacijos atminties sunaudojimas ir „Locust“ atsako laiko medianos pokytis

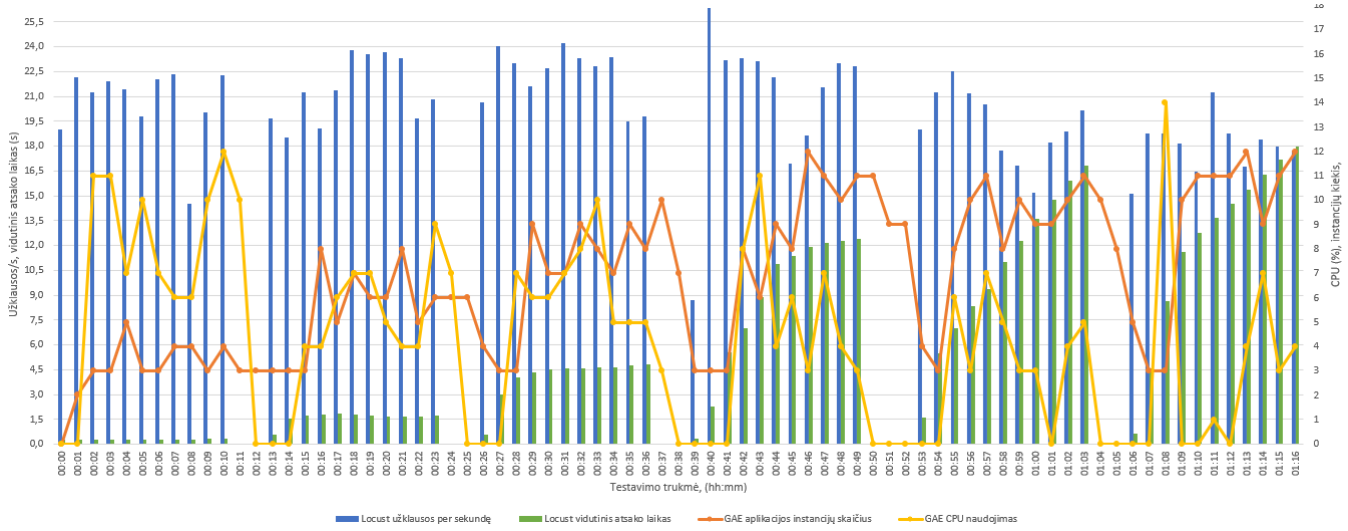
Locust testai pateikia išorinį, galutinio vartotojo patiriamą atsako laiką. Čia matome žymiai didesnius atsako laiko rodiklių pokyčius didėjant apkrovai, ypač vidutinio, medianos ir didžiausio atsako laiko vertėse. Pvz., kontrolinėje aplikacijoje vidutinis atsako laikas nuo 233,98 ms (50 vartotojų) išauga iki 16584,33 ms (800 vartotojų), o eksperimentinėje – nuo 306,92 ms iki 18279,85 ms prie 1000 vartotojų (žr *7 pav.*). Taigi, nors sistemos vidiniai resursai (CPU, atmintis, talpykla) išlieka gana stabilūs, išoriniai matavimai rodo didelę atsako laikų degradaciją, pasiekus tam tikrą vartotojų skaičių.



Šaltinis: sudaryta autoriaus. Grafikas iliustruoja, kad atminties sunaudojimas padidėja (bet ne iki kritinio lygio). Grafikas parodo, kaip auga vidutiniai ir 90%, 95%, 99% procentiliniai atsako laikai didėjant vartotojų skaičiui. Tai aiškiai iliustruoja, kad vidutinis atsako laikas eksperimentinėje aplinkoje yra didesnis nei kontrolinėje.

7 pav. Vidutinis atsako laikas ir procentiliniai rodikliai

Nors CPU, atmintis ir talpyklos rodikliai tiesiogiai nerodo problemų, kriptografinės operacijos gali sukurti papildomą (nors ir nedidelę) apkrovą užklausos apdorojimo grandinėje. Esant žemam vartotojų skaičiui, toks poveikis minimalus ir beveik nepastebimas. Tačiau didėjant apkrovai, net ir nedidelės papildomos delsos (pvz., susijusios su šifravimu, rakto valdymu, šifruoto turinio skaitymui/rašymui) pradeda kauptis. Tai sukelia užklausų eilių augimą, padidina jų aptarnavimo laiką bei išplečia atsako laikų pasiskirstymą. Ši situacija grindžiama eilių teorija ir *Little's Law* principais, teigiančiais, kad net menkas apdorojimo laiko padidėjimas gali neproporcingai padidinti laukimo laikus, kai sistema veikia artėdama prie savo pralaidumo ribų (žr. *8 pav.*). [63]



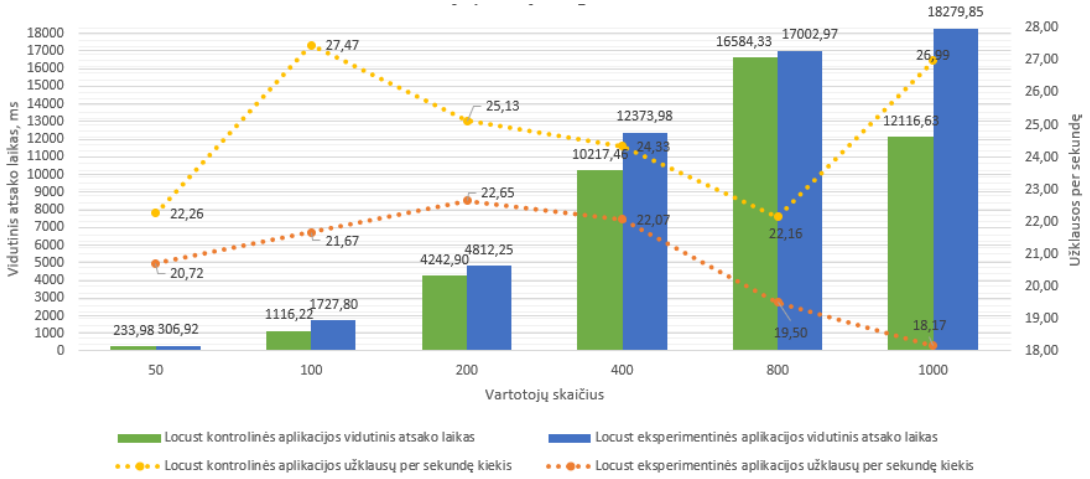
Šaltinis: sudaryta autoriaus. Grafikas parodo CPU apkrovą, atsako laiką ir instancijų skaičių eksperimentinėje aplinkoje, leidžiant suprasti, kaip šifravimas daro poveikį ilgalaikei sistemų dinamikai.

8 pav. Eksperimentinės aplikacijos sistemos veikimo metrikos per laiką

Vidinių metrikų stabilumas (CPU, *cache hit ratio*) gali sudaryti klaidingą įspūdį, jog sistema veikia optimaliai. Tačiau *Locust* rodo, kad augant vartotojų skaičiui, tarp užklausų atsiranda didelis dispersijos padidėjimas (tai matyti iš 90%, 95%, 99% atsako laikų (žr *7 pav.*)). Eksperimentinėje aplikacijoje šifravimas gali sukelti momentinių procesų, kurie nėra matomi kaip tiesioginiai CPU ar atminties šuoliai, bet lemia lėtesnį kai kurių užklausų apdorojimą. Didelio masto sistemose, vienetinis šifravimo ciklas gali būti greitas ir beveik nepastebimas CPU apkrovos grafikuose, tačiau kai tūkstančiai ar dešimtys tūkstančių užklausų vienu metu reikalauja papildomo apdorojimo, atsiranda „uodeginės“ užklausos (angl. *tail latencies*), kurių apdorojimas užtrunka ženkliai ilgiau. [50]

Vidinių metrikų analizėje pastebėta, kad eksperimentinėje versijoje išsiunčiamas duomenų srautas šiek tiek sumažėja, o užklausų dažnis GAE lygmenyje yra nežymiai mažesnis. *Locust* rezultatai atspindi šiuos dinamikos pokyčius: esant didesnėms apkrovoms, užklausų per sekundę skaičius eksperimentinėje aplinkoje mažesnis nei kontrolinėje. Tai derinasi su vidinėmis metrikomis:

šifravimas neprideda papildomos CPU apkrovos, bet padidina atminties ir nedideles apdorojimo laiko sąnaudas vienai užklausai. Dėl to bendra per laiko vienetą aptarnaujamų užklausų apimtis gali lengvai sumažėti, kai sistema veikia artėjant prie savo pralaidumo ribų (žr. *9 pav.*). [63]



Šaltinis: sudaryta autoriaus. Grafikas parodo užklausų apdorojimo greičio ir atsako laiko pokyčius kontrolinėje ir eksperimentinėje aplinkoje. Aiškiai matoma šifravimo įtaka atsako laikui ir apdorotų užklausų kiekiui.

9 pav. „Locust“ užklausų apdorojimo greitis ir atsako laikas

Apibendrinant, vidinių resursų (CPU, atminties, talpyklos) analizė ir *Locust* pateikiama vartotojo patirties (atsako laiko, procentilių, maksimalių verčių) analizė atskleidžia kompleksinį ryšį tarp bazinių sistemų resursų stabilumo ir galutinio vartotojo patirties. Nors vidiniai matavimai rodo, kad *Fernet* šifravimas nesukelia papildomos CPU ar atminties naštos, *Locust* duomenys parodo, kad net nedideli apdorojimo laiko padidėjimai, augant vartotojų skaičiui, gali lemti atsako laikų augimą. Ši situacija patvirtina eilių teorijos ir didelio masto sistemų architektūros principus: net ir stabilūs vidinių resursų rodikliai negali garantuoti stabilios vartotojo patirties, kai papildomi procesai, kaip šifravimas, sukelia vėlinimus. [50]

Autentifikacijos poveikio analizė (*control-app-v2* ir *firebase-auth-app*)

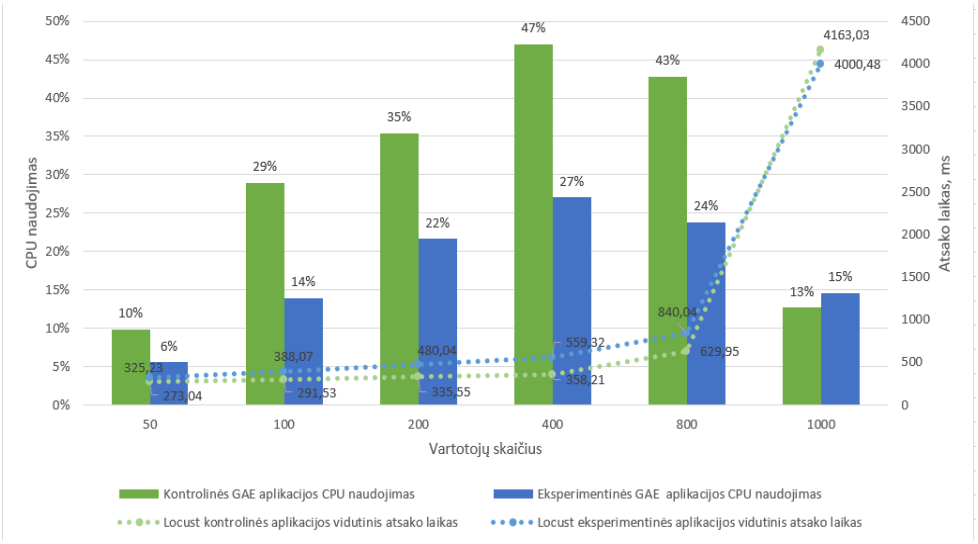
Apibendrinant gautus *Locust stats.csv* failų suvestinės šifravimo tyrimui rezultatus (detali suvestinės rezultatų analizė pateikiama **16 PRIEDASE**) galime teigti, jog *Firebase Authentication* padidina vidutinį atsako laiką dėl papildomų skaičiavimo resursų poreikio. Tačiau prie maksimalios apkrovos (1000 vartotojų) eksperimentinė aplikacija parodė mažesnį atsako laiką nei kontrolinė, o tai galime sieti su infrastruktūros optimizacija. Autentifikacija reikšmingai veikia atsako laiko medianą, tačiau prie didelių apkrovų (1000 vartotojų) infrastruktūros optimizacijos sumažina šį poveikį. Eksperimentinėje aplikacijoje didžiausias atsako laikas buvo didesnis nei kontrolinėje prie mažesnių

apkrovų, tačiau skirtumai sumažėjo prie maksimalios apkrovos, rodančios infrastruktūros gebėjimą efektyviai valdyti srautus. Eksperimentinė aplikacija išlaikė beveik tokį pat užklausų per sekundę skaičių kaip kontrolinė, rodančią minimalų autentifikacijos poveikį užklausų apdorojimui. Abi aplikacijos buvo stabilios, su beveik nuliu klaidų dažniu. Prie 1000 vartotojų eksperimentinėje aplikacijoje pastebėtas minimalus klaidų dažnis (0,0057%), kuris yra nereikšmingas. Eksperimentinėje aplikacijoje procentiliniai atsako laikai dažniausiai buvo didesni, tačiau prie 1000 vartotojų jie buvo mažesni, o tai siejama su infrastruktūros optimizacijomis.

Atlikus išsamią *Google Cloud Console Monitoring* grafikų analizę (grafikų analizė pateikiama **17 PRIEDASE**), galima nustatyti keletą svarbių dėsningumų, susijusių su šifruotos ir nešifruotos aplikacijų versijų (eksperimentinės ir kontrolinės) našumo rodikliais GAE bei *Memorystore Memcached* aplinkose. Nors eksperimentinėje aplikacijoje pastebimas didesnis atminties naudojimas (GAE lygmenyje), CPU apkrova išlieka stabilesnė ir net šiek tiek mažesnė. Tai reiškia, kad autentifikacijos procesai labiau remiasi atminties išteklių naudojimu (pvz., laikinų duomenų struktūrų palaikymu) nei CPU intensyviu skaičiavimu. Nors autentifikacija nepadidina CPU ar atminties sunaudojimo iki kritinio lygio ir nedidina tinklo srauto, ji sukelia didesnius atsakymų vėlavimus. Tai sufleruoja, kad autentifikacijos operacijos, kurios gali būti susijusios su trečiųjų šalių paslaugų (*Firebase*) užklausomis, yra labiau linkusios į momentinius vėlinimus. Šie vėlinimai gali kilti dėl išorinių tinklo sąlygų ar asinchroninių procesų. Tai rodo, kad saugumo funkcionalumo integravimas gali paveikti ne sisteminius resursus, o jos reakcijos laikų tolygumą. Gauti/išsiųsti duomenys, atsakymų ir *Memcached* operacijų skaičius turi labai panašią augimo, piko ir mažėjimo dinamiką. Kadangi autentifikacija nekeičia šių dėsningumų, galima daryti išvadą, kad pagrindiniai užklausų apdorojimo šablonai išlieka tie patys. Autentifikacija tiesiog „įterpiama“ į esamą sistemos eigą, nepakeisdama jos esminės veikimo logikos. Itin aukštas „pataikymų“ santykis parodo, kad talpyklos naudojimas išlieka veiksmingas ir efektyvus nepriklausomai nuo autentifikacijos.

Apibendrinant, *Firebase* autentifikacija nedaro reikšmingo neigiamo poveikio baziniams sistemos resursų rodikliams, tokiems kaip CPU, atminties naudojimas ar tinklo srautas, tačiau gali sukelti atsako laiko nestabilumą. Ši variacija nėra tiesiogiai susijusi su didesne resursų apkrova, o kyla dėl kartais pasitaikančių vėlinimo šuolių, kurie priklauso nuo išorinių faktorių. Šie rezultatai atitinka eilių teorijos ir kompleksinių sistemų principus, kurie teigia, kad ne visi funkcionalumo išplėtimai būtinai padidina bazinių resursų naudojimą, tačiau gali sukelti didesnį atsako laiko neapibrėžtumą [50]. Todėl, vertinant sistemų našumą, būtina ne tik analizuoti resursų naudojimo

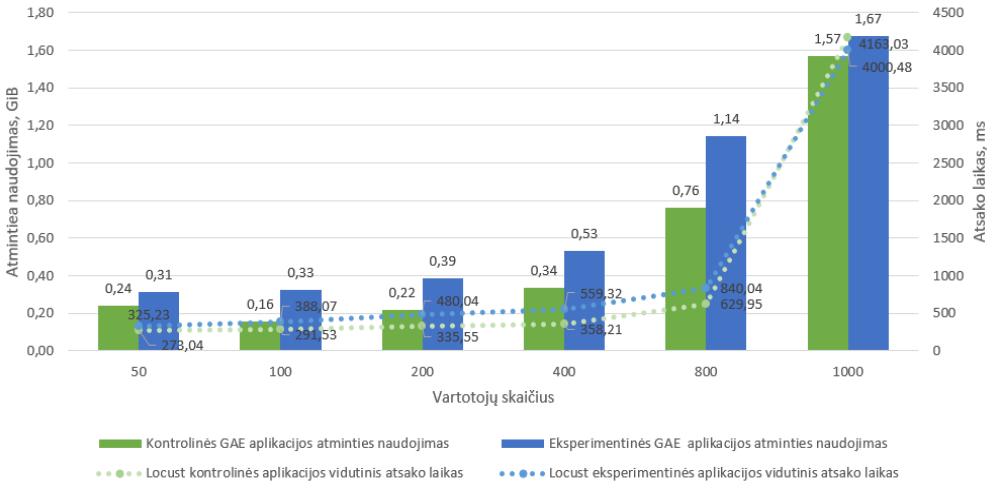
rodiklius, bet ir įvertinti atsako laiko stabilumą bei jo pasiskirstymo variaciją, nes tai atskleidžia svarbius aspektus, kurie daro įtaką bendram sistemos efektyvumui.



Šaltinis: sudaryta autoriaus. Grafikas parodo, kad CPU apkrova nei kontrolinėje, nei eksperimentinėje aplinkoje nėra reikšmingai išaugusi, nors *Locust* atsako laikai (punktyrinės linijos) auga didėjant vartotojų skaičiui.

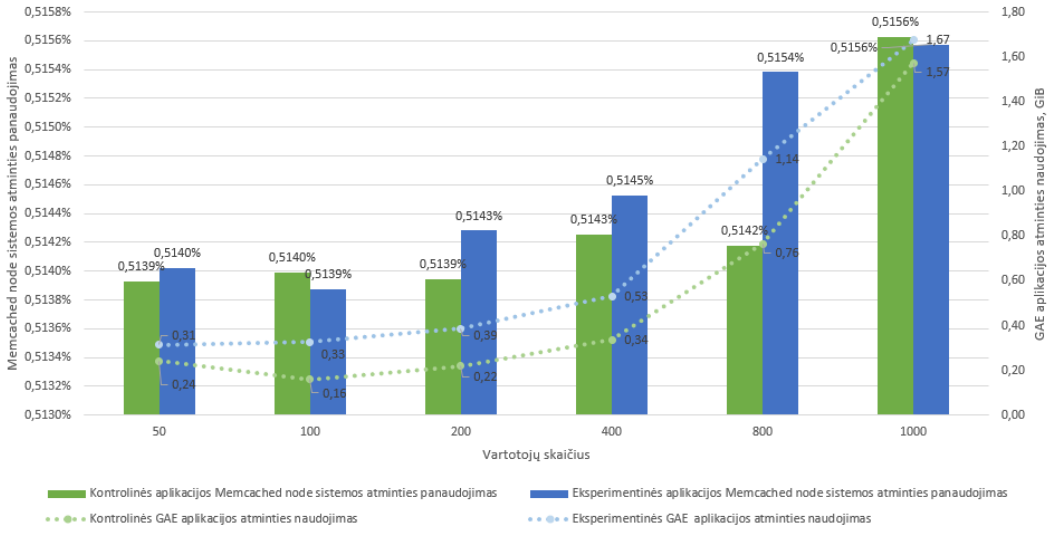
10 pav. GAE aplikacijos CPU naudojimas ir atsako laikas („Locust“)

Iš GAE ir *Memcached* metrikų matyti, kad *Firebase* autentifikacija iš esmės nepablogina bazinių išteklių (CPU, atminties, talpyklos) panaudojimo. CPU apkrova *Memcached* lygiu išlieka minimali ir net nežymiai mažėja eksperimentinėje aplinkoje, o sisteminis atminties naudojimas talpyklos lygiu nepatiria reikšmingų pokyčių. (žr. *10 pav.*, *11 pav.*, *12 pav.*)



Šaltinis: sudaryta autoriaus. Grafikas parodo, kad bendra atminties sąnauda išlieka gana stabili abiejose aplinkose, nors atsako laikas ilgėja.

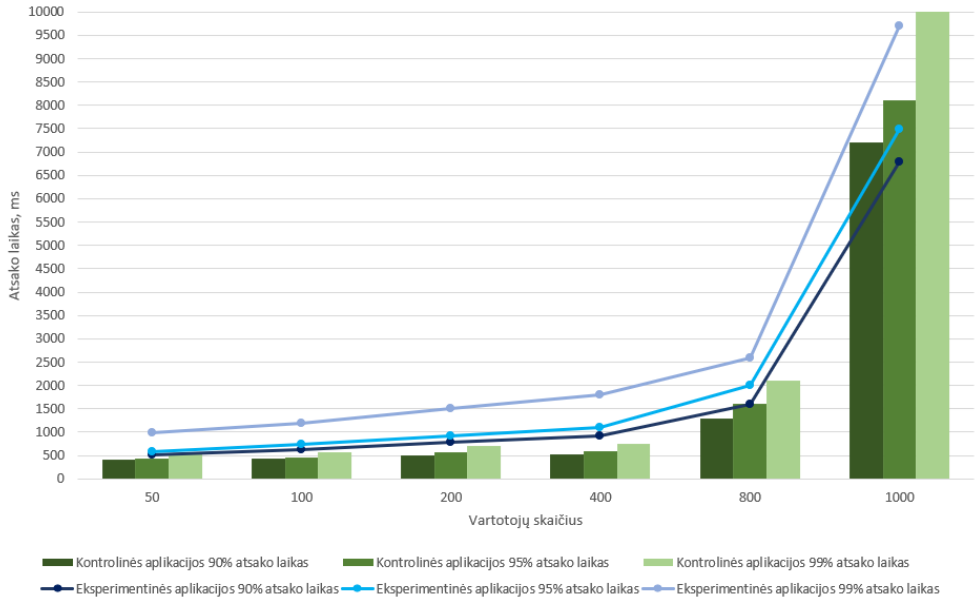
11 pav. GAE aplikacijos atminties panaudojimas ir atsako laikas (*Locust*)



Šaltinis: sudaryta autoriaus. Grafikas parodo, kad net ir įjungus autentifikaciją, *Memcached Node* bei GAE atminties rodikliai kinta nedaug. Parodo, kodėl talpyklos *hit ratio* išlieka labai aukštas, o bendras naudojimas – stabilus.

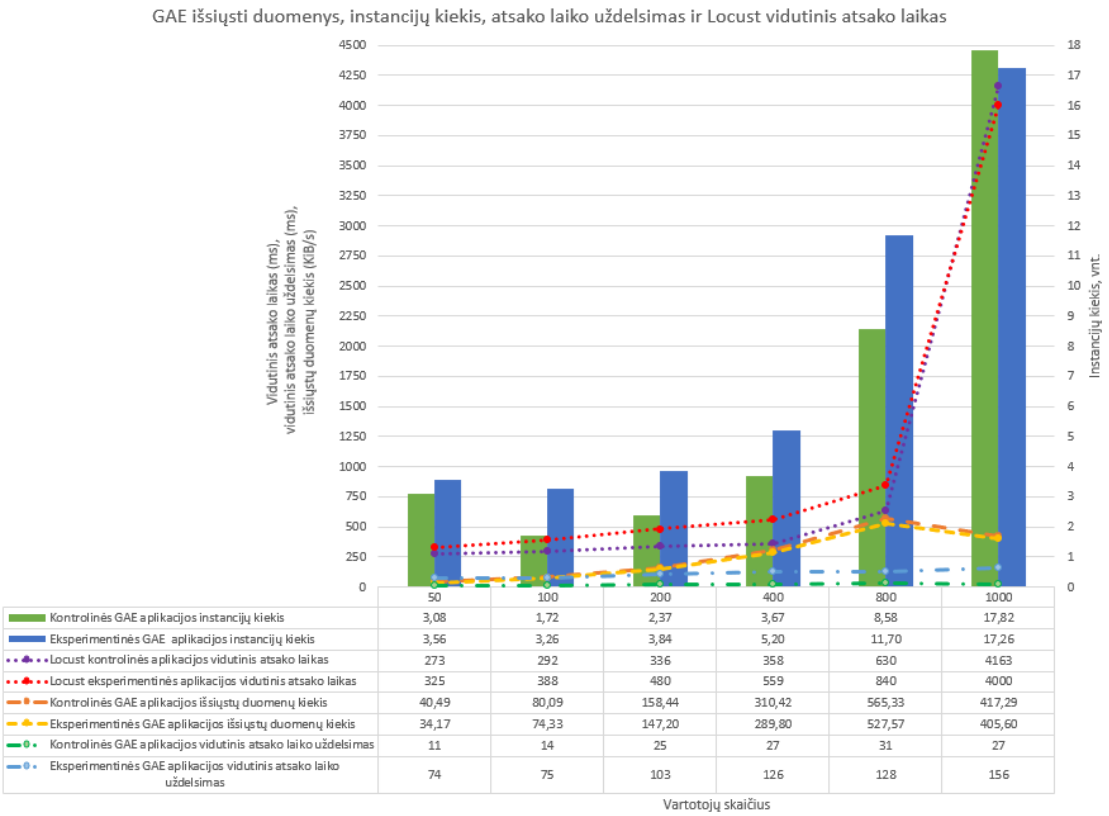
12 pav. Atminties naudojimas GAE ir atminties naudojimas *Memcached Node*

Tačiau *Locust* matavimai parodė, jog vidutinis ir procentilinis atsako laikas (tiek medianos, tiek 90–99% intervaluose) auga tiek kontrolinėje, tiek eksperimentinėje aplinkoje didėjant vartotojų skaičiui. Eksperimentinėje (su autentifikacija) aplikacijoje šis augimas kai kuriose apkrovos situacijose yra spartesnis, pvz., medianos augimas prie 800 vartotojų. Šis neatitikimas tarp stabilių vidinių resursų rodiklių ir augančių atsako laikų rodo, kad atsako laiko padidėjimas nėra tiesiog nulemtas sisteminių resursų trūkumo. (žr. *13 pav.*)



Šaltinis: sudaryta autoriaus. Grafikas rodo, kad didėjant apkrovai, atsako laikas kyla tiek kontrolinėje, tiek eksperimentinėje aplinkoje (ir eksperimentinė neretai vėluoja labiau), nors CPU/atmintis to „drastiškai“ nerodo (žr. *10 pav.*, *11 pav.*, *12 pav.*).

13 pav. Procentiliniai atsako laikai (*Locust*)

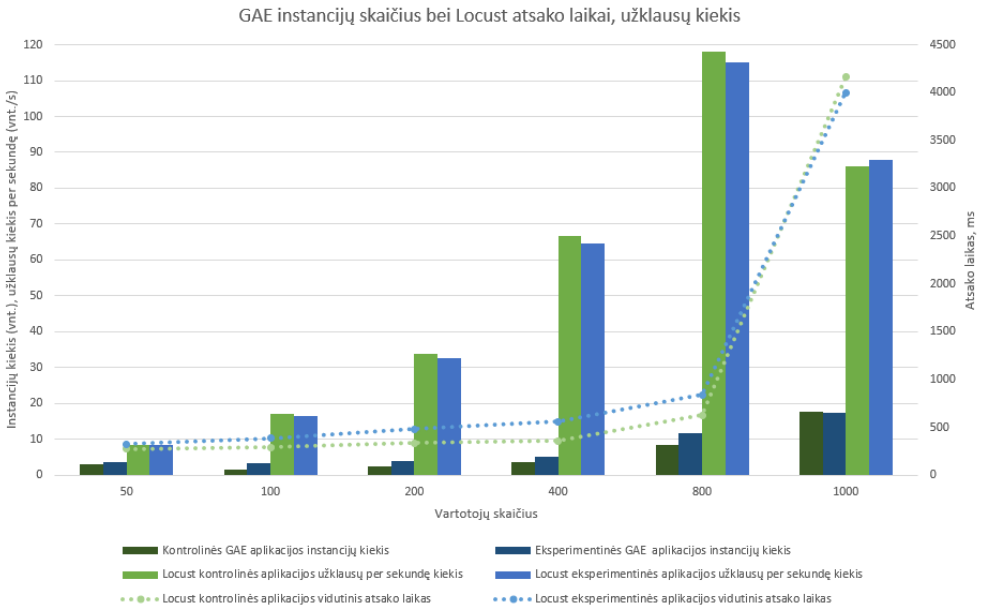


	50	100	200	400	800	1000
Kontrolinės GAE aplikacijos instancijų kiekis	3,08	1,72	2,37	3,67	8,58	17,82
Eksperimentinės GAE aplikacijos instancijų kiekis	3,56	3,26	3,84	5,20	11,70	17,26
Locust kontrolinės aplikacijos vidutinis atsako laikas	273	292	336	358	630	4163
Locust eksperimentinės aplikacijos vidutinis atsako laikas	325	388	480	559	840	4000
Kontrolinės GAE aplikacijos išsiųstų duomenų kiekis	40,49	80,09	158,44	310,42	565,33	417,29
Eksperimentinės GAE aplikacijos išsiųstų duomenų kiekis	34,17	74,33	147,20	289,80	527,57	405,60
Kontrolinės GAE aplikacijos vidutinis atsako laiko uždelsimas	11	14	25	27	31	27
Eksperimentinės GAE aplikacijos vidutinis atsako laiko uždelsimas	74	75	103	126	128	156

Šaltinis: sudaryta autoriaus.

14 pav. GAE išsiųsti duomenys, instancijų kiekis, atsako laiko uždelsimas ir *Locust* vidutinis atsako laikas

Vietoje to, jį gali lemti užklausų apdorojimo logika, užklausų eilės augimas, papildomos (nors mažos apimties) operacijos, susijusios su autentifikacija, kurios padidina apdorojimo grandinės ilgį. GAE ir *Memcached* analizė rodė, kad talpyklos *hit ratio* išlieka labai aukštas, CPU naudojimas GAE aplinkoje stabilus arba net kiek tolygesnis su autentifikacija, o tinklo srautas labai nesiskiria (žr. *14 pav.*). Tai reiškia, kad autentifikacijos patikrinimas nėra didelės apimties operacija išteklių atžvilgiu. Tačiau *Locust* rezultatai atskleidė, kad eksperimentinėje aplikacijoje tam tikrų apkrovų metu (pvz., 50–800 vartotojų) atsako laikai (vidutinis, mediana ir didžiausias) dažnai didesni nei kontrolinėje. Šis faktas gali būti siejamas su momentiniais vėlinimais, kurių neparodo bendras CPU ar atminties naudojimo rodiklis. Autentifikacijos metu vykstantys žetonų validavimai, ryšiai su išorine *Firebase* paslauga ar asinchroninės operacijos, nepastebimos iš vidinės metrikos pusės, gali sukurti nereguliarų uždelsimą, ypatingai kai apkrova didėja. Šiuos procesus galima sieti su eilės teorija: net mažas papildomas vėlinimo faktorius, veikiantis išorines užklausas, dideliame vartotojų sraute didina bendrą atsako laiką. *Locust* rezultatai rodo, kad esant 1000 vartotojų, eksperimentinėje aplikacijoje vidutinis, atsako laiko mediana ir procentilių atsako laikas šiek tiek mažesnis nei kontrolinėje. Iš GAE metrikų žinome, kad didinant apkrovą, atsakymų skaičius, duomenų srautas ir *Memcached* operacijos pasiekia piką ties 800 vartotojų ir tada sumažėja.



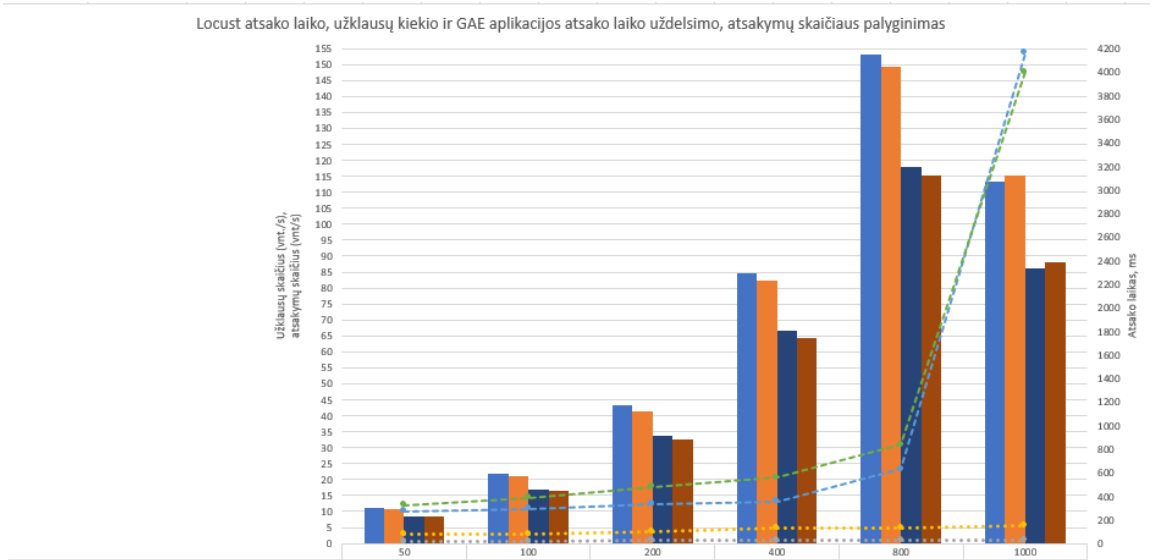
Šaltinis: sudaryta autoriaus. Grafike matyti, kaip didėja GAE instancijų kiekis didėjant vartotojų skaičiui, taip pat pateikiamas *Locust* atsako laikas bei užklausų kiekis. Ši diagrama atspindi automatinio mastelio keitimo mechanizmo poveikį - ties 800–1000 vartotojų instancijų gerokai padaugėja.

15 pav. GAE instancijų skaičius bei *Locust* atsako laikai, užklausų kiekis

Esant 1000 vartotojų apkrovai, pastebimas sistemos našumo pagerėjimas gali būti siejamas su infrastruktūros perėjimu į efektyvesnį resursų valdymo režimą. Tai gali būti susiję su GAE automatinio mastelio keitimo (angl. *autoscaling*) mechanizmais, kurie aktyvuojasi didesnės apkrovos atveju (žr. *15 pav.*). Tokie mechanizmai leidžia inicijuoti papildomus egzempliorius, optimizuoti užklausų paskirstymą tarp jų, užtikrinant efektyvesnį resursų panaudojimą. Šioje situacijoje autentifikacijos logika, vidutinėse apkrovose veikusi kaip nedidelis našumo stabdis, tampa mažiau pastebima dėl infrastruktūros gebėjimo prisitaikyti ir kompensuoti autentifikacijos pridėtinę kainą. Toks elgesys atitinka debesų kompiuterijos automatinio mastelio keitimo principus, pagal kuriuos pasiekus tam tikrą apkrovos ribą, paskirstymo ir valdymo procesai optimizuojami, o tai gali lemti net ir atsako laiko pagerėjimą. Ši reakcija į apkrovos didėjimą yra būdinga GAE standartinei aplinkai, kuri dinamiškai paskirsto resursus pagal sistemos poreikius. [64] Eksperimentinės aplikacijos testavimo metu GAE automatinio mastelio keitimo mechanizmas suaktyvino daugiausiai 17,26 instancijų (svertinis vidurkis) prie 1000 vartotojų, kad galėtų apdoroti užklausų srautą, o kontrolinės aplikacijos testavimo metu – 17,82.

Locust duomenys rodo, kad eksperimentinėje aplinkoje prie 1000 vartotojų pastebimas minimalus klaidų dažnis (0,0057%), o kontrolinėje – 0%. Iš vidinės metrikos (GAE ir *Memcached*) matyti, jog išteklių nepriteklius nepasireiškia, tad klaidų priežastys greičiausiai ne susijusios su resursų trūkumu. Greičiau tai momentiniai autentifikacijos paslaugos atsako sutrikimai ar tinklo

paklaidos. Nors šis klaidų dažnis labai mažas, jis parodo, kad egzistuoja papildomas komponentas (autentifikacija), didinantis sistemos kompleksiškumą. (Žr. *16 pav.*)



	50	100	200	400	800	1000
Kontrolinės GAE aplikacijos atsakymų skaičius	10,98	21,72	43,11	84,60	153,27	113,41
Eksperimentinės GAE aplikacijos atsakymų skaičius	10,71	21,10	41,55	82,19	149,51	115,11
Locust kontrolinės aplikacijos užklausų per sekundę kiekis	8,60	17,01	33,82	66,66	117,97	86,17
Locust eksperimentinės aplikacijos užklausų per sekundę kiekis	8,39	16,48	32,52	64,53	115,14	87,93
Kontrolinės GAE aplikacijos vidutinis atsako laiko uždelsimas	11	14	25	27	31	27
Eksperimentinės GAE aplikacijos vidutinis atsako laiko uždelsimas	74	75	103	126	128	156
Locust kontrolinės aplikacijos vidutinis atsako laikas	273	292	336	358	630	4163
Locust eksperimentinės aplikacijos vidutinis atsako laikas	325	388	480	559	840	4000

Šaltinis: sudaryta autoriaus. Diagrama leidžia pamatyti, kiek realiai užklausų sėkmingai apdorojama (atsakymų kiekis), koks atsako laikas, kiek apytikriai yra užklausų per sekundę. Nors tekste minimas klaidų dažnis nurodytas nedideliu skaičiumi, tačiau matyti, kad eksperimentinės aplinkos atsako laikas ir atsakymų kiekis šiek tiek svyruoja dėl papildomo autentifikacijos komponento.

16 pav. *Locust* atsako laiko, užklausų kiekio ir GAE aplikacijos atsako laiko uždelsimo, atsakymų skaičiaus palyginimas

Remiantis GAE bei *Memcached* metrika, užklausų apimtis ir operacijų intensyvumas koreliuoja su apkrova, tačiau nesikeičia esminiai jų dinamikos principai tarp kontrolinės ir eksperimentinės aplinkos. *Locust* duomenys tai patvirtina: užklausų per sekundę skaičius abiejose aplinkose panašus, o tai reiškia, kad autentifikacija neiškreipia bazinio užklausų generavimo ir aptarnavimo modelio. Kitaip tariant, CPU, atminties ar tinklo srauto rodikliai nerodo, jog eksperimentinė aplinka būtų ribojama labiau nei kontrolinė, todėl tiek kontrolinė, tiek eksperimentinė aplikacija generuoja panašų užklausų per sekundę lygį.

Locust procentilių atsako laikų analizė (pvz., 90%, 95%, 99%) rodo, kad esant didesnei apkrovai, ypač ties 1000 vartotojų, eksperimentinė aplikacija netgi aplenkia kontrolinę (žr. *13 pav.*). Iš ankstesnių GAE analizės išvadų žinoma, kad GAE elgsena gali kisti su didėjančiu vartotojų skaičiumi, pvz., papildomai inicijuojant daugiau instancijų. Toks automatinis mastelio keitimo mechanizmas gali geriau išlyginti apkrovos nelygumus eksperimentinėje aplinkoje (buvo aktyvuota daugiau instancijų nei kontrolinės aplikacijos metu), sudarant sąlygas autentifikuotoms užklausoms

būti tvarkomoms efektyviau. Nors autentiškumo patikrinimas sukuria tam tikrą vėlinimą, infrastruktūros prisitaikymas prie didelės apkrovos sugeba šį neigiamą poveikį sumažinti (žr. *16 pav.*). Ši teorinė prielaida dera su debesų kompiuterijos principais, kur automatinio mastelio keitimo mechanizmas ne tik didina resursus, bet ir gali optimizuoti užklausų maršrutus, spartindamas atsako laikus. Analizė parodė, kad našumo vertinime nepakanka žiūrėti tik į bazinius resursų rodiklius. Autentifikacija, nors ir neapkrauna CPU ar atminties itin stipriai, prideda papildomą asimetrinį vėlinimą, kuris tampa reikšmingas tuomet, kai vartotojų skaičius auga, o užklausų srautai tampa jautrūs net nedidelėms papildomoms operacijoms. Kita vertus, esant kraštutinei apkrovai, debesijos infrastruktūra (GAE) gali efektyviai kompensuoti šiuos vėlinimus, netgi pagerindama procentilinius atsako laikus.

Tokiu būdu, susiejant *Locust* atsako laiko, procentilių ir klaidų rodiklius su GAE bei *Memcached* sisteminiais matavimais, susidaro holistinis vaizdas: *Firebase* autentifikacija nėra didelis resursų vartotojas, bet ji keičia sistemos elgseną atskirais momentais (momentiniais vėlinimais, aukštesniais procentiliniais atsako laikais), o tai pasireiškia vartotojo patirtimi, matoma *Locust* testų rezultatuose. Galiausiai, didžiausios apkrovos scenarijuje infrastruktūros prisitaikymas gali kompensuoti šiuos neigiamus padarinius, rodydamas šiuolaikinės debesų infrastruktūros lankstumą ir efektyvumą valdant sudėtingas, kriptografiją ar autentifikaciją naudojančias paslaugas.

4.5. Tyrimo rezultatų analizė testavimo be pertraukų metu

Eksperimentinio tyrimo metu buvo siekiama nustatyti papildomų saugos mechanizmų (duomenų šifravimo ir autentifikacijos) poveikį *Memcached* naudojančių aplikacijų veikimui GAE standartinėje aplinkoje. Našumo duomenys buvo surinkti naudojant *Locust* generuotus testavimo failus (*stats.csv*, *stats_history.csv*) ir analizuoti naudojant *Excel* bei *Knime* įrankius. Papildomi sisteminiai rodikliai buvo gauti iš *Google Cloud Console Monitoring*.

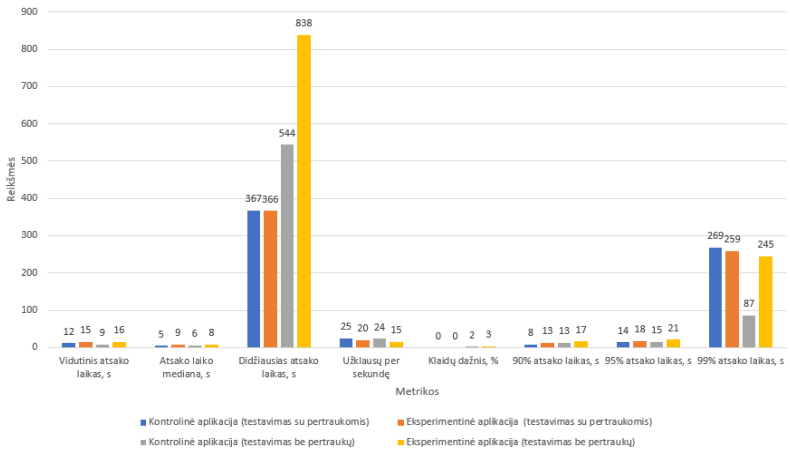
Šifravimo poveikio analizė (*control-app* ir *encrypt-app*)

Atlikus eksperimentinį tyrimą be pertraukų, buvo analizuotas autentifikacijos poveikis aplikacijų *control-app* ir *encrypt-app* veikimui. Naudojant iš *Locust* sugeneruotus *stats.csv* ir *stats_history.csv* failų duomenis, sudaryta rezultatų suvestinė ir vidutinio atsako laiko palyginimo grafikas (žr. **18 PRIEDASĄ**). **18 PRIEDASE** pateikta kontrolinės ir eksperimentinės aplikacijų našumo analizė parodė, kad šifravimas sumažina sistemos našumą. Vidutinis eksperimentinės

aplikacijos atsako laikas siekė 15568,98 ms, o didžiausias – 837909,46 ms, palyginti su 8751,09 ms vidutiniu ir 544259,39 ms didžiausiu kontrolinės aplikacijos atsako laiku. Užklausų apdorojimo greitis eksperimentinėje aplikacijoje sumažėjo iki 15 užklausų/s, palyginti su 23,52 užklausų/s kontrolinėje aplikacijoje. Šie rezultatai rodo, kad šifravimo procesai padidina sistemos resursų poreikį. Eksperimentinės aplikacijos klaidų dažnis siekė 3%, palyginti su 2% kontrolinėje aplikacijoje, o tai leidžia daryti išvadą, jog šifravimas sukuria papildomą sistemos apkrovą. Be to, šifravimas padidino atsako laiko svyravimus, ypač esant didesnei apkrovai, kaip rodo 99% atsako laikas (245000 ms eksperimentinėje ir 87000 ms kontrolinėje aplikacijoje). Atsako laiko nevienalytiškumas gali neigiamai paveikti sistemų, turinčių aukštus našumo reikalavimus, veikimą.

Apibendrinti *Locust* duomenų grafiko rezultatai (žr. **18 PRIEDASĄ**) rodo, kad šifravimo procesai daro įtaką jau ankstyvoje apkrovos stadijoje. Vidutinio atsako laiko didėjimas eksperimentinėje aplikacijoje pastebėtas 6 minutėmis anksčiau nei kontrolinėje aplikacijoje, kai apkrova pasiekė 250 vartotojų. Nepaisant našumo sumažėjimo, eksperimentinė aplikacija išlaikė stabilų atsako laiko augimą, o tai rodo tinkamai integruotus šifravimo procesus. Kontrolinė aplikacija užtikrino efektyvų resursų paskirstymą ir optimalią veikimo eigą, tačiau nesiūlė papildomo saugumo lygio. Rezultatai rodo, kad debesų kompiuterijos aplikacijose būtina subalansuoti našumo ir saugumo poreikius. Nors kontrolinė aplikacija pasiekė geresnius našumo rodiklius, eksperimentinė aplikacija užtikrino aukštesnį saugumo lygį, kuris yra būtinas jautrių duomenų apdorojimui.

Tyrimas parodė, kad papildomi šifravimo procesai gali sukelti našumo sumažėjimą ir atsako laiko padidėjimą esant didesnei apkrovai. Siekiant sumažinti šifravimo poveikį našumui, rekomenduojama naudoti efektyvesnius algoritmus arba hibridinius metodus, taip pat pritaikyti sistemas, kurios užtikrintų geresnį resursų valdymą esant didelėms apkrovoms.



Šaltinis: sudaryta autoriaus.

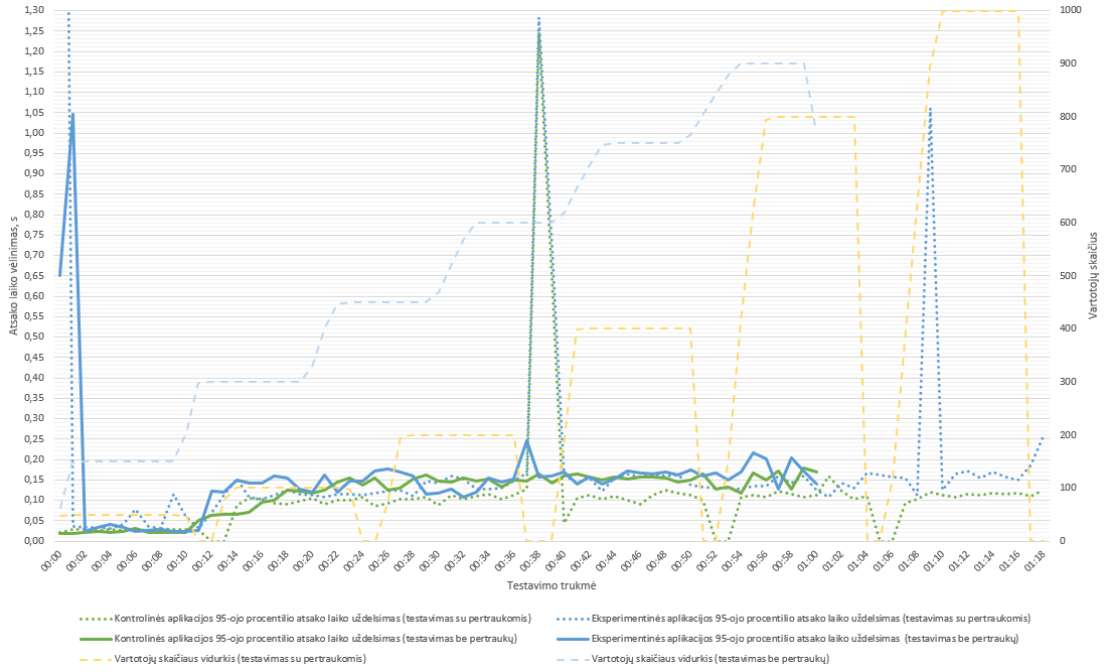
17 pav. *Locust* metrikų palyginimas testų metu (šifravimo poveikis)

Grafike (žr. *17 pav.* Locust *metrikų palyginimas testų metu (šifravimo poveikis)17 pav.*) pateikiami *Locust* įrankio surinkti duomenys aiškiai iliustruoja šifravimo poveikį aplikacijų našumui skirtingomis testavimo sąlygomis – su pertraukomis ir be jų. Pateikti rezultatai demonstruoja, kaip testavimo metodologija ir šifravimo procesai įtakoja aplikacijų atsako laiką, užklausų apdorojimo spartos dinamiką bei stabilumą. Testavimo rezultatai atskleidžia kontrolinės ir eksperimentinės aplikacijų našumo skirtumus, priklausančius nuo testavimo pobūdžio – su pertraukomis ar be jų. Kontrolinė aplikacija pasižymi trumpesniu vidutiniu atsako laiku ir mediana, lyginant su eksperimentine, nepriklausomai nuo testavimo būdo. Testuojant su pertraukomis, kontrolinės aplikacijos vidutinis atsako laikas yra 12 sekundžių, o eksperimentinės – 15 sekundžių, mediana atitinkamai yra 5 ir 9 sekundės. Testuojant be pertraukų, kontrolinė aplikacija pasiekia dar geresnius rezultatus – vidutinis atsako laikas yra 9 sekundės, o eksperimentinės aplikacijos – 16 sekundžių. Šie duomenys patvirtina, kad eksperimentinė aplikacija, naudojanti papildomus saugumo mechanizmus, reikalauja daugiau resursų ir pasižymi ilgesniu atsako laiku. Didžiausio atsako laiko analizė rodo reikšmingus skirtumus tarp aplikacijų, ypač testuojant be pertraukų. Su pertraukomis didžiausias atsako laikas yra panašus abiem aplikacijoms (367 ir 366 sekundės), tačiau be pertraukų testuojant eksperimentinė aplikacija pasiekia 838 sekundes, palyginti su 544 sekundėmis kontrolinėje. Tai leidžia daryti prielaidą, kad papildomi saugumo mechanizmai eksperimentinėje aplikacijoje gali sukelti našumo kritimą esant intensyviam krūviui. Užklausų apdorojimo greitis taip pat yra didesnis kontrolinėje aplikacijoje: su pertraukomis ji apdoroja 25 užklausas per sekundę, eksperimentinė – 20. Be pertraukų testuojant, šie rodikliai siekia atitinkamai 24 ir 15 užklausų per sekundę. Šis skirtumas rodo, kad saugumo mechanizmai lėtina eksperimentinę aplikaciją. Klaidų dažnio analizė rodo, kad su pertraukomis klaidų nefiksuojama nei vienoje aplikacijoje, tačiau be pertraukų klaidų dažnis eksperimentinėje aplikacijoje siekia 3 %, o kontrolinėje – 2 %. Tai rodo, kad eksperimentinė aplikacija yra jautresnė nuolatiniam darbo krūviui. Procentilių atsako laikų analizė išryškina kontrolinės aplikacijos pranašumą, išskyrus vieną atvejį. Testuojant su pertraukomis, kontrolinės aplikacijos 99 procentilio atsako laikas yra 10 sekundžių ilgesnis nei eksperimentinės (269 sekundės palyginti su 259 sekundėmis). Tai gali reikšti didesnį kontrolinės aplikacijos netolygumą tam tikromis sąlygomis. Vis dėlto, testuojant be pertraukų, eksperimentinė aplikacija aiškiai nusileidžia kontrolinei – 99 procentilio atsako laikas siekia 245 sekundes, palyginti su 87 sekundėmis kontrolinėje aplikacijoje.

Apibendrinant, testavimo pobūdis daro reikšmingą įtaką abiejų aplikacijų veikimui. Testavimas su pertraukomis mažina skirtumus tarp aplikacijų, tuo tarpu testavimas be pertraukų akivaizdžiai parodo eksperimentinės aplikacijos našumo trūkumus: ilgesnį atsako laiką, didesnį klaidų dažnį ir mažesnį apdorojamų užklausų skaičių. Papildomi saugumo mechanizmai eksperimentinėje aplikacijoje neigiamai veikia našumą, ypač esant intensyvioms apkrovoms, todėl

būtina kruopščiai įvertinti ir optimizuoti šių mechanizmų poveikį siekiant išlaikyti balansą tarp saugumo ir efektyvumo.

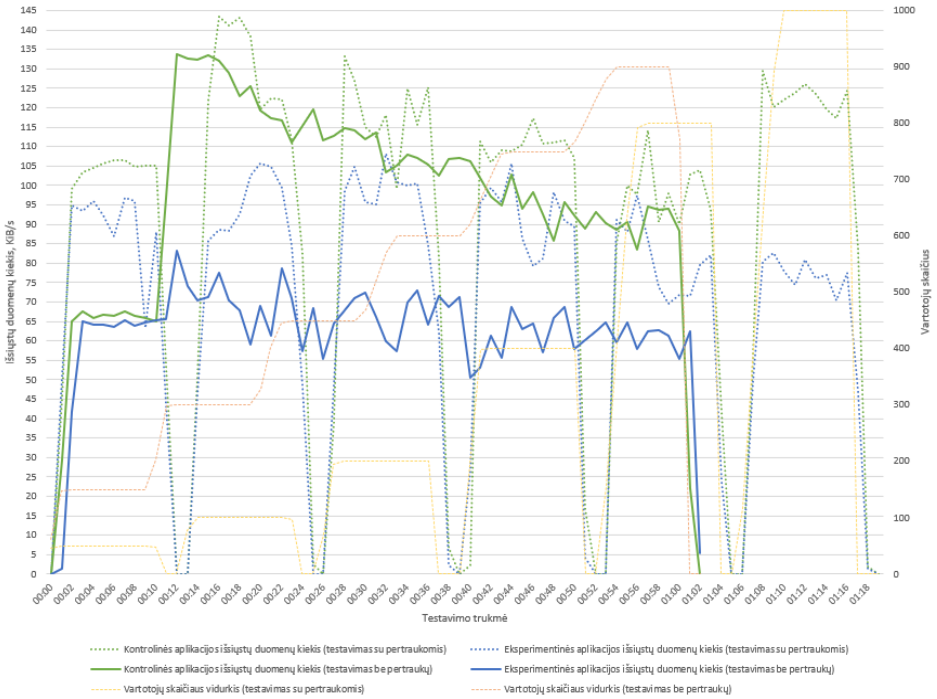
Lyginant gautus GCP *Monitoring* (žr. **15 PRIEDASĄ** ir **19 PRIEDASĄ**) rezultatus testavimo be pertraukų metu ir testavimo su pertraukomis, galima pastebėti, kad šifravimo įtraukimas turi skirtingą poveikį priklausomai nuo apkrovos dinamikos. Testavimo be pertraukų metu atsako laikas pasižymi didesne variacija (žr. *18 pav.*), ypač eksperimentinėje aplikacijoje, kurioje 95-ojo procentilio atsako laiko vėlinimas dažniausiai svyruoja 110–180 ms ribose. Tuo tarpu testuojant su pertraukomis, šifravimo poveikis vidutiniams atsako laikams išlieka mažesnis, tačiau retkarčiais pasitaiko reikšmingi vėlavimo šuoliai (iki 1,28 s), kurie gali būti susiję su sistemos „įšilimo“ procesu. Šie skirtumai rodo, kad nuolatinė apkrova labiau išryškina šifravimo operacijų poveikį sistemos našumui. Abejuose testavimo scenarijuose *hit ratio* išlieka artimas 100%, o tai leidžia teigti, kad šifravimas neturi neigiamos įtakos talpyklos efektyvumui ar duomenų paieškos mechanizmams.



Šaltinis: sudaryta autoriaus.

18 pav. GAE aplikacija - 95-ojo procentilio atsako laiko uždelsimas testavimų metu (šifravimo poveikis)

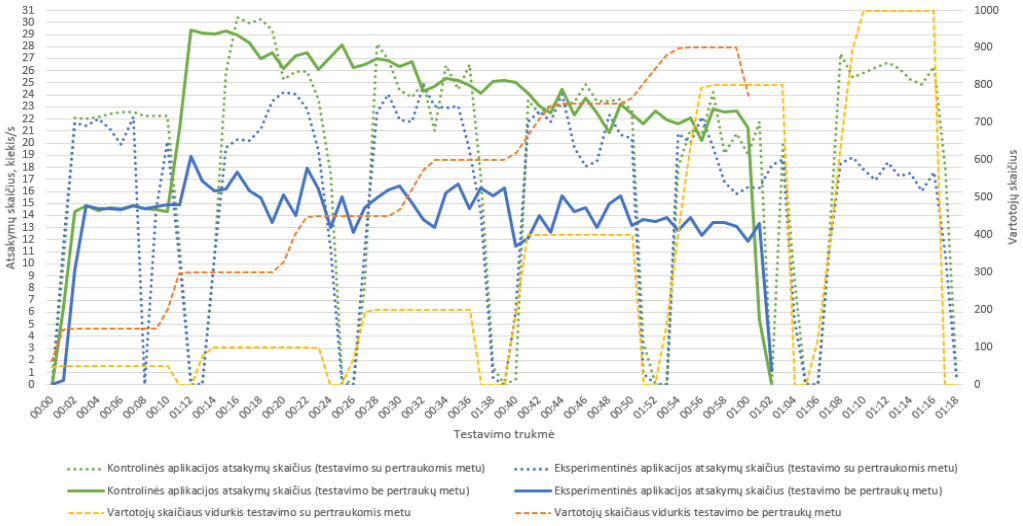
Tačiau duomenų srauto analizė parodė, kad eksperimentinėje aplikacijoje šifravimas sukelia didesnius srauto svyravimus. Testavimo be pertraukų metu vidutinis išsiųstų duomenų kiekis sumažėja iki ~65 KiB/s (vidurkis), palyginti su kontrolinės aplikacijos ~98 KiB/s (vidurkis). Testuojant su pertraukomis, srautas stabilizuojasi, tačiau vis dar išlieka mažesnis (~68,81 KiB/s) nei kontrolinėje aplikacijoje (žr. *19 pav.*).



Šaltinis: sudaryta autoriaus.

19 pav. GAE aplikacija - išsiųstų duomenų kiekio palyginimas testavimų metu (šifravimo poveikis)

Atsakymų skaičius taip pat rodo šifravimo įtaką. Testuojant be pertraukų, eksperimentinėje aplikacijoje atsakymų skaičius svyruoja nuo 11 iki 19 ats./s, o kontrolinėje – nuo 20 iki 30 ats./s (rezultatai nelabai panašūs ir testavimo su pertraukomis metu). Pertraukiamo testavimo metu eksperimentinės aplikacijos atsakymų skaičius stabilizuojasi vidutiniškai ties ~18 ats./s, tačiau vis dar išlieka mažesnis nei kontrolinėje aplikacijoje (žr. *20 pav.*).



Šaltinis: sudaryta autoriaus.

20 pav. GAE aplikacija - atsakymų skaičiaus palyginimas testavimų metu (šifravimo poveikis)

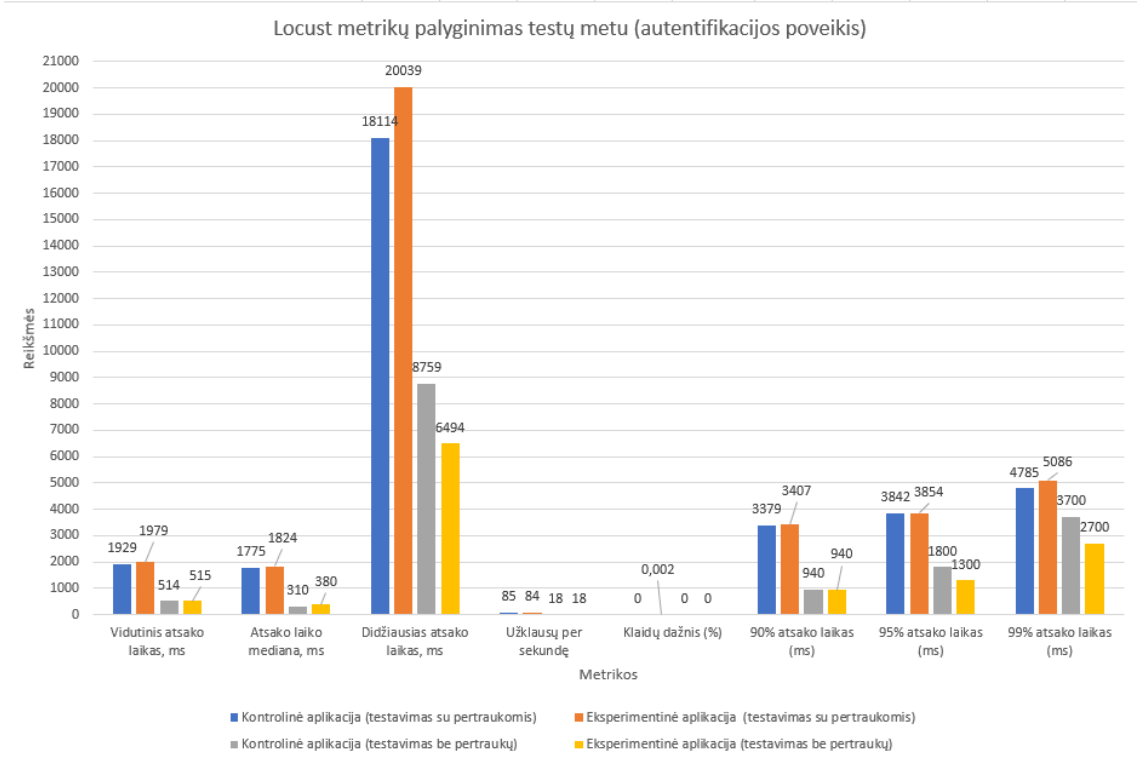
Nuolatinės apkrovos sąlygomis šifravimas sukelia didesnę atsako laikų variaciją ir tinklo srauto svyravimus dėl nuolatinio skaičiavimo krūvio. Pertraukiamo testavimo metu sistema demonstruoja didesnį stabilumą, tačiau retkarčiais pasireiškia staigūs vėlavimo šuoliai. Rezultatai rodo, kad būtina optimizuoti šifravimo operacijas, ypač nuolatinės apkrovos scenarijuose, siekiant sumažinti našumo praradimą. Nors šifravimas šiek tiek mažina našumą, jo teikiama saugumo nauda yra reikšminga, todėl toks kompromisas gali būti priimtinas sistemose, kuriose prioritetas teikiamas duomenų apsaugai. Apibendrinant, šie rezultatai leidžia daryti išvadą, kad šifravimo įtraukimas be pertraukų atliktame testavime labiau pabrėžia sistemos našumo ir pralaidumo pokyčius, o pertraukiamo testavimo metu šifravimo poveikis yra mažesnis, tačiau pastebimi netolygumai. Tai rodo, kad šifravimo operacijos poveikis labai priklauso nuo naudojimo scenarijaus ir sistemos apkrovos dinamikos.

Autentifikacijos poveikio analizė (*control-app-v2* ir *firebase-auth-app*)

Atlikus eksperimentinį tyrimą be pertraukų, analizuotas šifravimo poveikis aplikacijų *control-app* ir *firebase-auth-app* veikimui. Naudojant iš *Locust* sugeneruotus *stats.csv* ir *stats_history.csv* failų duomenis, sudaryta rezultatų suvestinė ir vidutinio atsako laiko palyginimo grafikas (žr. **20 PRIEDASĄ**). Gauti rezultatai rodo, kad autentifikacijos mechanizmų integracija turi minimalų poveikį sistemos našumui, užtikrinant stabilų veikimą tiek kontrolinėje, tiek eksperimentinėje aplikacijose. Vidutinio atsako laiko skirtumas tarp aplikacijų buvo nežymus (1,03 ms), o tai leidžia daryti išvadą, kad *Firebase Authentication* mechanizmai nesukėlė reikšmingos papildomos apkrovos sistemai. Taip pat analizuojant atsako laiko medianą, eksperimentinė aplikacija parodė nedidelį laiko padidėjimą, palyginti su kontroline aplikacija, o tai rodo nedidelį papildomą autentifikacijos poveikį užklausų apdorojimo laikui. Didžiausio atsako laiko rodikliai parodė reikšmingą skirtumą tarp aplikacijų, kur eksperimentinė aplikacija su autentifikacija užtikrino trumpesnį maksimalų atsako laiką. Tai rodo, kad papildomi autentifikacijos mechanizmai gali sumažinti ekstremaliomis situacijomis pasireiškiančias resursų apkrovas. Tuo tarpu užklausų per sekundę skaičiaus analizė ir klaidų dažnio rezultatai parodė, kad abi aplikacijos efektyviai tvarkėsi su augančia apkrova ir užtikrino visišką stabilumą, nepriklausomai nuo taikomų saugos sprendimų.

Grafiko analizė (žr. **20 PRIEDASĄ**) patvirtino, kad pradinėje fazėje abi aplikacijos susidūrė su „įšilimo“ etapu, kuris lėmė nestabilius atsako laikus. Stabilizacijos fazėje eksperimentinė aplikacija demonstravo šiek tiek didesnį atsako laiką, tačiau skirtumas išliko nereikšmingas (50 – 100 ms). Testavimo pabaigoje, esant maksimalioms apkrovoms, abi aplikacijos užtikrino stabilų veikimą, o vidutinis atsako laikas buvo panašus.

Apibendrinant galima teigti, kad papildomų autentifikacijos mechanizmų integracija į eksperimentinę aplikaciją neturėjo reikšmingo neigiamo poveikio sistemos našumui. Be to, tokie sprendimai padėjo sumažinti ekstremaliais atvejais pasireiškiančius našumo šuolius, užtikrinant stabilumą ir prognozuojamumą. Šie rezultatai rodo, kad autentifikacijos mechanizmai gali būti sėkmingai taikomi panašiuose sprendimuose, minimaliai paveikiant jų veikimą.



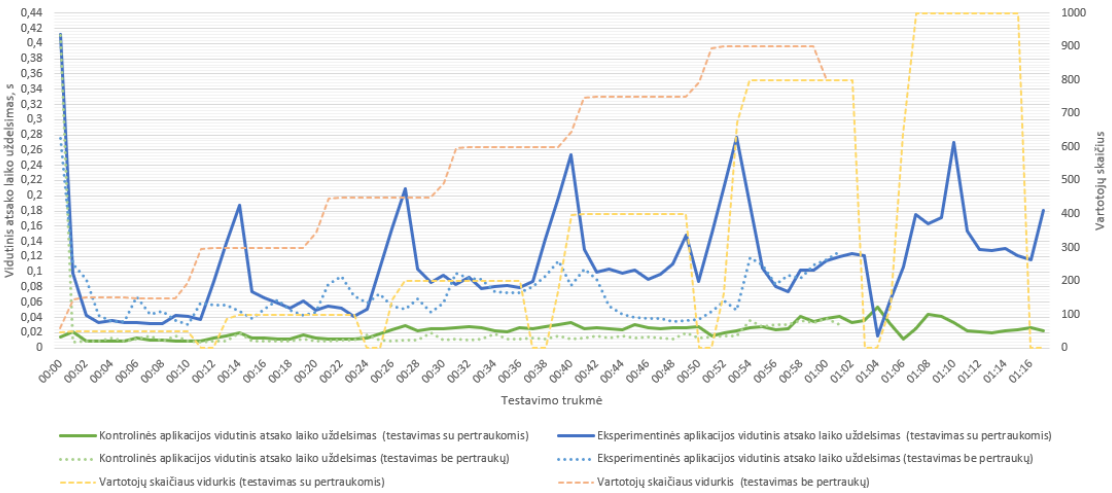
Šaltinis: sudaryta autoriaus.

21 pav. *Locust* metrikų palyginimas testų metu (autentifikacijos poveikis)

Grafikas (žr. *21 pav.*) atskleidžia įvairius *Locust* metrikų pokyčius testuojant kontrolinę ir eksperimentinę aplikacijas su skirtingomis sąlygomis – su pertraukomis ir be jų. Analizė parodė, kad testavimo režimas (su pertraukomis ar be jų) daro reikšmingą poveikį tam tikriems našumo rodikliams, pavyzdžiui, didžiausiam atsako laikui, 95% ir 99% atsako laikui, tuo tarpu vidutiniai atsako laikai išlieka santykinai stabilūs abiem testavimo sąlygomis. Kontrolinės ir eksperimentinės aplikacijų vidutinio atsako laiko rodikliai tarpusavyje buvo panašūs tiek testuojant su pertraukomis, tiek be jų, o tai rodo, kad sistemų veikimo efektyvumas šiuo aspektu nėra stipriai paveikiamas testavimo režimo. Šis pastovumas leidžia manyti, kad abiejų aplikacijų našumas yra optimizuotas daugumai užklausų, nepriklausomai nuo testavimo sąlygų. Tačiau didžiausias atsako laikas, 95% ir 99% atsako laikas testavimo be pertraukų metu buvo mažesni nei testavimo su pertraukomis metu. Tai rodo, kad testavimas su pertraukomis gali sukelti papildomų iššūkių sistemoms, galimai dėl pradinio „įšilimo“ efekto ar pertraukų metu pasikeitusios infrastruktūros apkrovos. Rezultatai leidžia

teigti, jog autentifikacijos mechanizmai gali sumažinti ekstremalių situacijų poveikį sistemos našumui nuoseklaus darbo metu. Užklausų per sekundę skaičius išliko stabilus visais testavimo atvejais, o klaidų dažnis buvo praktiškai lygus nuliui. Tai rodo, kad abi aplikacijos efektyviai susidoroja su skirtingomis apkrovos sąlygomis, nepriklausomai nuo pertraukų buvimo.

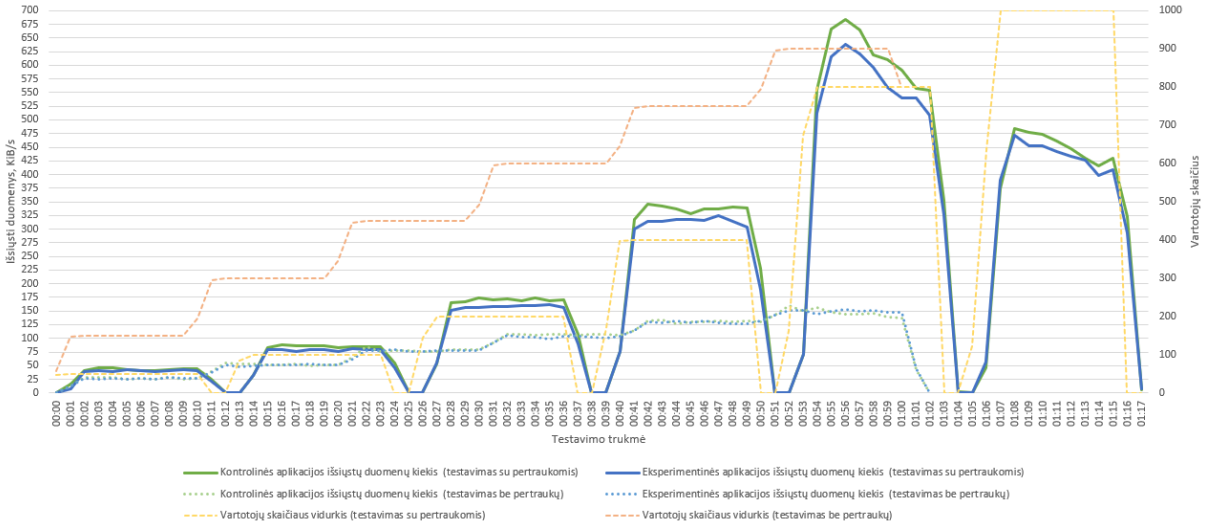
Apibendrinant galima teigti, kad testavimas su pertraukomis daro didesnį poveikį ekstremaliems atsako laikams, tuo tarpu vidutiniai rodikliai išlieka santykinai stabilūs. Eksperimentinė aplikacija, turinti papildomus autentifikacijos mechanizmus, pasirodė efektyvesnė valdant ekstremalių atsako laikų riziką, ypač testuojant be pertraukų. Šie rezultatai pabrėžia testavimo sąlygų svarbą analizuojant aplikacijų našumą ir autentifikacijos mechanizmų integravimo įtaką



Šaltinis: sudaryta autoriaus.

22 pav. GAE aplikacija - vidutins atsako laiko uždelsimo palyginimas testavimų metu (autentifikacijos poveikis)

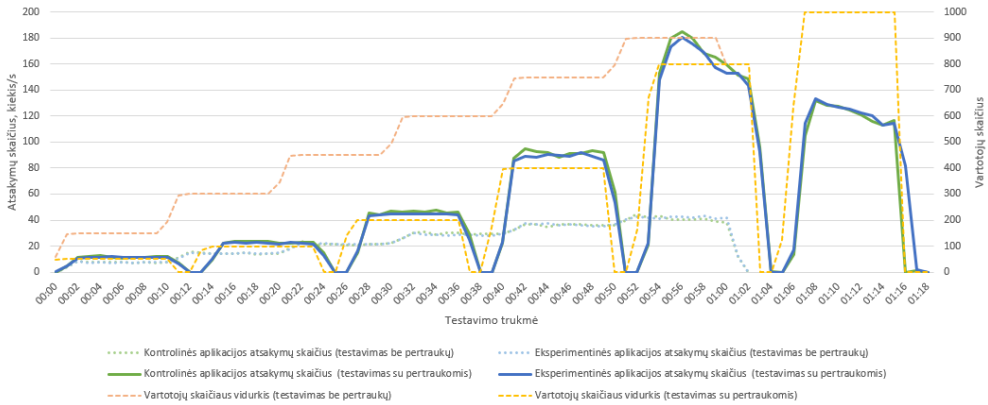
Lyginant gautus GCP *Monitoring* (žr. **17 PRIEDASĄ** ir **21 PRIEDASĄ**) rezultatus testavimo be pertraukų ir su pertraukomis metu, išryškėjo reikšmingi skirtumai atsako laiko, užklausų apdorojimo stabilumo ir resursų valdymo srityse tarp kontrolinės ir eksperimentinės aplikacijų. Testuojant su pertraukomis, atsako laiko vėlinimas abiejose aplikacijose buvo pastebimai didesnis nei testuojant be pertraukų (žr. *22 pav. GAE aplikacija - vidutins atsako laiko uždelsimo palyginimas testavimų metu (autentifikacijos poveikis)22 pav.*). Šį skirtumą galima paaiškinti *Firebase Authentication* ID žetonų patikros proceso dinamika, kuri, esant nuolatinei apkrovai, tampa labiau nuspėjama, o pertraukiamo testavimo metu kyla daugiau tinklo svyravimų dėl greičiau kintančio vartotojų srauto.



Šaltinis: sudaryta autoriaus.

23 pav. GAE aplikacija - išsiųstų duomenų palyginimas testavimų metu (autentifikacijos poveikis)

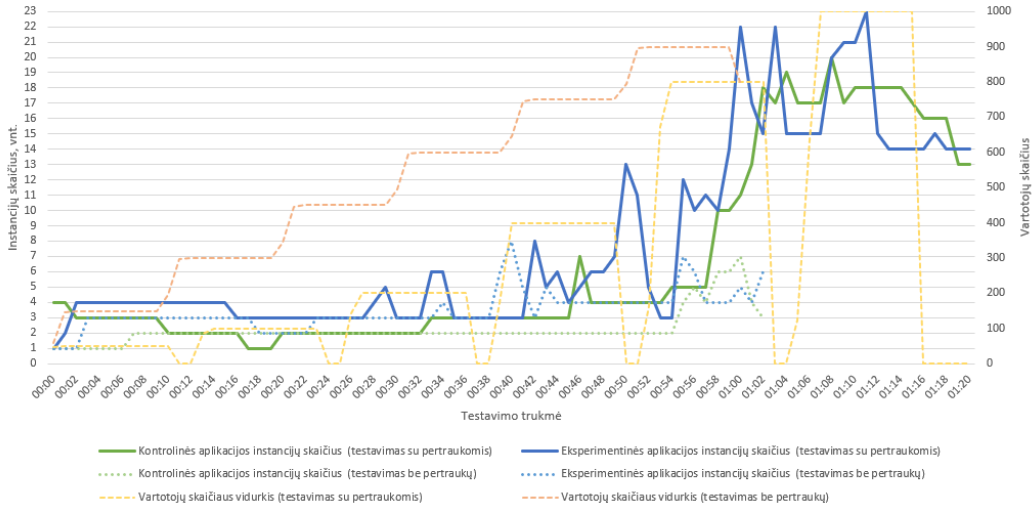
Tinklo srauto analizė parodė (žr. *23 pav.*), kad testavimo be pertraukų metu tiek kontrolinė, tiek eksperimentinė aplikacijos demonstravo ženkliai sumažėjusį išsiųstų duomenų kiekį, o skirtumai tarp jų buvo minimalūs. Tai leidžia daryti išvadą, kad autentifikacija neturi reikšmingos įtakos tinklo srauto našumui, o pastebėtas sumažėjimas susijęs su sistemos optimizavimo ar resursų taupymo mechanizmais.



Šaltinis: sudaryta autoriaus.

24 pav. GAE aplikacija - atsakymų skaičiaus palyginimas testavimų metu (autentifikacijos poveikis)

Panaši situacija matoma analizuojant atsakymų skaičių (žr. *24 pav.*) – jis išlieka artimas abiejose aplikacijose, nepriklausomai nuo testavimo metodikos, o tai reiškia, kad autentifikacija neturi reikšmingo poveikio sistemos atsakų kiekiui.



Šaltinis: sudaryta autoriaus.

25 pav. GAE aplikacija - instancijų skaičiaus palyginimas testavimų metu (autentifikacijos poveikis)

Resursų valdymo skirtumai ypač ryškiai pasireiškė mastelio keitimo procese (žr. *25 pav.*). Testavimo su pertraukomis metu abiem aplikacijoms prireikė žymiai daugiau papildomų instancijų. Eksperimentinėje aplikacijoje jų skaičius pasiekė 23, o kontrolinėje – 21, esant didžiausiam vartotojų kiekiui. Testuojant be pertraukų, kai eksperimentinėje aplikacijoje vartotojų skaičius pasiekė beveik 750, prireikė tik 8 instancijų, o kontrolinė aplikacija, esant 900 vartotojų, pasiekė didžiausią – 7 instancijų skaičių. Tai rodo, kad dinaminės apkrovos sąlygos (su pertraukomis) kelia didesnius iššūkius efektyviam resursų paskirstymui, tuo tarpu nuolatinės apkrovos metu resursų poreikis išlieka mažesnis ir labiau nuspėjamas. Šie rezultatai taip pat leidžia daryti išvadą, kad autentifikacijos procesai, nepaisant tam tikro atsako laiko padidėjimo, yra veiksmingai integruojami į sistemą be reikšmingo neigiamo poveikio tinklo srautui ar atsakymų kiekiui. Tai parodo, kad autentifikacija, esant tinkamai optimizacijai, gali būti diegiama net ir intensyviai naudojamose sistemose. Autentifikacija ir testavimo sąlygų dinamika leidžia atskleisti keletą svarbių tendencijų. Pirma, tinklo srauto ir atsakymų skaičiaus stabilumas rodo, kad autentifikacija neturi neigiamo poveikio esminiams sistemos našumo rodikliams. Antra, testavimo su pertraukomis metu, kai vartotojų srautas yra dinamiškesnis, sistema reikalauja žymiai daugiau resursų, todėl mastelio keitimo procesas tampa sudėtingesnis ir brangesnis. Tai pabrėžia būtinybę optimizuoti resursų valdymą dinaminėmis apkrovos sąlygomis.

Apibendrinant, testavimo rezultatai parodė, kad nuolatinės apkrovos scenarijus leidžia geriau kontroliuoti sistemos stabilumą ir sumažinti resursų poreikį, o pertraukiamas testavimas atskleidė didesnį autentifikacijos procesų jautrumą dinaminiam vartotojų srautui. Vis dėlto abiem testavimo metodais buvo užtikrintas stabilus *hit ratio*, efektyvus tinklo srauto valdymas ir minimalus poveikis

sistemos atsakų kiekiui, o tai rodo, kad autentifikacija yra tinkamas sprendimas net sudėtingose sistemų konfigūracijose. Šie rezultatai pabrėžia autentifikacijos integracijos efektyvumą ir parodo galimybes tobulinti resursų valdymą dinamiškose aplinkose.

4.6. Tyrimo išvados ir praktinės rekomendacijos

Atliktas tyrimas atskleidė, kad šifravimo (*Fernet*) ir autentifikacijos (*Firebase Authentication*) mechanizmai turi pastebimą, tačiau nedidelį neigiamą poveikį aplikacijų našumui. Tyrimo rezultatai parodė, kad šifravimas ir autentifikacija padidina atsako laiką ir, tam tikrais atvejais, atminties naudojimą, tačiau šis poveikis buvo ribotas ir nesukėlė reikšmingų sistemos veikimo sutrikimų. GAE automatinio mastelio keitimo mechanizmai veiksmingai kompensavo šiuos neigiamus padarinius, užtikrindami stabilų ir patikimą sistemų veikimą net ir esant intensyvioms apkrovoms.

Šifravimo mechanizmo poveikio analizė parodė, kad atsako laikas eksperimentinėje aplikacijoje buvo vidutiniškai 25 % ilgesnis nei kontrolinėje aplikacijoje testuojant su pertraukomis ir beveik 78 % ilgesnis testuojant be pertraukų. Didžiausio atsako laiko skirtumas tarp aplikacijų buvo dar ryškesnis, ypač be pertraukų – eksperimentinėje aplikacijoje jis buvo apie 1,5 karto didesnis. Nepaisant to, talpyklos efektyvumas, išreikštas *hit ratio*, išliko artimas 100 %, o tai rodo, kad šifravimo integracija nepakeitė pagrindinių duomenų paieškos procesų. Be to, šifravimas padidino GAE automatinio mastelio keitimo metu naudojamų instancijų skaičių iki 30 %, o tai leido sistemai išlaikyti mažą klaidų dažnį ir stabilų veikimą.

Autentifikacijos mechanizmai, priešingai nei šifravimas, turėjo dar mažesnį poveikį našumui. Vidutinis atsako laikas padidėjo vos 2,6 % testuojant su pertraukomis ir beveik nepakito testuojant be pertraukų. Didžiausio atsako laiko skirtumas buvo pastebėtas tik testuojant su pertraukomis (10,6 % padidėjimas), tačiau be pertraukų autentifikacija netgi sumažino šį rodiklį apie 25,8 %, atskleisdama efektyvesnį resursų paskirstymą nuolatinės apkrovos sąlygomis. Sistemos talpyklos *hit ratio* išliko stabilus, o CPU naudojimas nepatyrė reikšmingų svyravimų, nes didžioji dalis autentifikacijos procesų buvo atliekama už GAE aplinkos ribų, naudojant *Firebase* infrastruktūrą.

Tyrimas taip pat patvirtino, kad naudojama testavimo metodologija daro įtaką rezultatų interpretacijai. Testavimas su pertraukomis padėjo nustatyti momentinius atsako laiko šuolius ir mažesnius našumo skirtumus tarp aplikacijų. Tuo tarpu testavimas be pertraukų išryškino ilgalaikius našumo pokyčius, atskleidžiant didesnį šifravimo poveikį atsako laikui ir resursų naudojimui.

Apibendrinant, atliktas tyrimas patvirtino hipotezę, kad šifravimo ir autentifikacijos mechanizmai gali turėti ribotą neigiamą poveikį aplikacijų našumui. Nepaisant šio poveikio, debesijos infrastruktūros gebėjimas dinamiškai prisitaikyti prie apkrovos užtikrino, kad sistemų stabilumas, talpyklos efektyvumas ir bendras veikimas išliktų priimtini net ir esant intensyviems saugumo sprendimams. Todėl šie saugumo mechanizmai gali būti rekomenduojami projektuose, kuriuose prioritetas teikiamas duomenų apsaugai.

Rekomendacijos:

- Optimizacija: siekiant sumažinti šifravimo ir autentifikacijos poveikį našumui, rekomenduojama naudoti efektyvesnius algoritmus arba hibridinius metodus.
- Resursų valdymas: tinkamai sukonfigūruoti automatinio mastelio keitimo mechanizmo taisykles (pvz., pagal atsako laiko vėlinimą ar CPU naudojimą), kad sistema galėtų greičiau reaguoti į apkrovos šuolius ir užtikrinti stabilų veikimą.
- Tolimesni tyrimai: atlikti panašius testus kitose debesijos aplinkose ar naudojant kitokias talpyklos technologijas (pvz., Redis), norint įvertinti, ar pastebėtas neutralus arba minimalus poveikis išlieka universalus.

Papildomos pastabos:

- Saugumo sprendimų naudojimas: papildomi saugumo komponentai nepadidino klaidų dažnio ir neužkirto kelio stabiliai sistemų veiklai. Eksperimentinėse aplikacijose (su šifravimu ar autentifikacija) klaidų dalis išliko labai maža arba beveik nulinė, o tai rodo patikimą šių komponentų integraciją.
- Infrastruktūros prisitaikymas: esant maksimaliai apkrovai, GAE automatinio mastelio keitimo mechanizmai leido išlaikyti stabilų atsako laiką ir mažą klaidų dažnį, net ir integruojant papildomas saugumo priemones.

IŠVADOS

1. Išanalizavus šaltinius, nustatyta, kad GAE standartinė aplinka pasižymi dinamišku resursų valdymu, efektyviu mastelio keitimu, integruotomis talpyklomis bei bazinėmis saugumo priemonėmis. Šaltinių analizė atskleidė, kaip debesų kompiuterijos raida, įskaitant be serverinį modelį, leido GAE įsitvirtinti kaip patikima šiuolaikinės IT infrastruktūros dalimi.
2. Išanalizavus architektūrinius sprendimus ir komponentus, nustatyta, kad GAE efektyviai valdo išteklius, užtikrina aukštą paslaugų prieinamumą ir integruoja tokius komponentus kaip *Memcached* ir *Cloud Datastore*, kurie pagerina sistemos našumą ir duomenų prieinamumą. Taip pat identifikuotos GAE stiprybės ir silpnybės, susijusios su našumu bei saugumu, ir nustatytos praktinės optimizavimo galimybės.
3. Empirinių tyrimų analizė parodė, kad tokie atvejai kaip *Khan Academy* integracija su GAE patvirtina, jog platforma geba efektyviai valdyti dideles apkrovas ir užtikrinti aukštą duomenų saugumo lygį, nepaisant papildomų saugumo priemonių poveikio sistemos našumui. Nustatyta, kad pažangūs šifravimo ir autentifikacijos sprendimai leidžia išlaikyti patenkinamą našumą, pasinaudojant GAE automatinio mastelio keitimo galimybėmis.
4. Projektinės dalies metu buvo suprojektuoti keturi aplikacijų variantai: kontrolinės aplikacijos be papildomų saugumo priemonių bei eksperimentinės aplikacijos su *Fernet* šifravimo ir *Firebase Authentication* mechanizmais. Parengti svetainių maketai, UML diagramos ir specifikacijos užtikrino kryptingą eksperimento įgyvendinimą, o pasirinktos technologijos leido efektyviai patenkinti eksperimentui keliamus reikalavimus.
5. Techninės realizacijos metu buvo sėkmingai sukurtos keturios, projektinės dalies metu suprojektuotos, aplikacijų versijos. Sukonfigūruota infrastruktūra užtikrino išsamų sistemos testavimą tiek lokalioje, tiek produkcinėje aplinkoje. Apkrovos testavimas, vykdytas naudojant *Locust* ir automatizuotus *PowerShell* scenarijus, leido realistiškai įvertinti sistemų stabilumą ir našumą įvairiose apkrovos sąlygose.
6. Atliktas tyrimas patvirtino, kad papildomų saugumo mechanizmų, *Fernet* šifravimo ir *Firebase Authentication,* integracija GAE standartinėje aplinkoje turi ribotą, tačiau valdomą poveikį aplikacijų našumui.
7. Ištirta, kad šifravimas vidutiniškai padidina atsako laiką, ypač esant intensyvioms apkrovoms, tačiau autentifikacijos poveikis yra minimalus. Nepaisant šių pokyčių,

GAE automatinio mastelio keitimo mechanizmai veiksmingai kompensavo našumo sumažėjimą, užtikrindami stabilų sistemų veikimą ir išlaikydami aukštą talpyklos efektyvumą.

8. Tyrimo rezultatai atskleidė, kad papildomų saugumo mechanizmų integracija nekeičiant esminių sistemos architektūrinių sprendimų yra įmanoma, pasinaudojant debesų platformų teikiamais išteklių valdymo įrankiais. Autentifikacijos procesų delegavimas *Firebase* infrastruktūrai sumažino CPU apkrovą ir užtikrino efektyvų resursų paskirstymą net nuolatinės apkrovos sąlygomis. Be to, šifravimo sprendimai parodė, kad tinkamai konfigūruota GAE aplinka gali išlaikyti sistemos patikimumą ir sumažinti klaidų dažnį.
9. Tyrimo metu įrodyta, kad testavimo metodologija turi reikšmingą poveikį rezultatų interpretacijai. Testavimas su pertraukomis leido nustatyti momentinius atsako laiko svyravimus, o testavimas be pertraukų išryškino ilgalaikius našumo pokyčius ir šifravimo poveikį resursų naudojimui.
10. Nustatyta, kad papildomi saugumo sprendimai gali būti sėkmingai taikomi projektuose, kuriuose prioritetas teikiamas duomenų apsaugai, neprarandant esminio sistemos našumo. GAE lankstumas ir dinamiškas prisitaikymas prie apkrovos rodo, kad ši platforma yra tinkama saugių ir našumui optimizuotų debesų kompiuterijos sprendimų kūrimui.

LITERATŪROS ŠALTINIAI

[1] R. Buyya, C. Vecchiola, T. S. Somasundaram, *Mastering Cloud Computing: Foundations and Applications Programming*, Elsevier, 2013. DOI: 10.1016/C2012-0-06719-1.

[2] Red Hat, „What is serverless?,“ 11 05 2022. [Tinkle]. Available: https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless. [Kreiptasi 15 09 2024].

[3] H. Soller, „Cloud 2.0: Serverless architecture and the next wave of enterprise offerings“, McKinsey Digital, 15 05 2020. [Tinkle]. Available: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/tech-forward/cloud-20-serverless-architecture-and-the-next-wave-of-enterprise-offerings. [Kreiptasi 15 09 2024].

[4] T. Alam, „Cloud Computing and its role in the Information Technology,“ *IAIC Transactions on Sustainable Digital Innovation (ITSDI)*, t. 1, nr. 2, p. 108-115, bal. 2020. DOI: /10.34306/itsdi.v1i2.103.

[5] M. Boniface, B. Nasser, J. Papay, S. C. Phillips, A. Servin, X. Yang, Z. Zlatev, S. V. Gogouvitis, G. Katsaros, K. Konstanteli, G. Kousiouris, A. Menychtas, D. Kyriazis, „Platform-as-a-Service Architecture for Real-Time Quality of Service Management in Clouds,“ įtraukta *2010 Fifth International Conference on Internet and Web Applications and Services*, Barcelona, Spain, 2010, p. 155-160. DOI: 10.1109/ICIW.2010.91.

[6] K. Gai, A. Steenkamp, „A Feasibility Study of Platform-as-a-Service Using Cloud Computing for a Global Service Organization“, *Journal of Information Systems Applied Research*, t.7, nr. 3, p. 28-42, 2014.

[7] S. Kolb, G. Wirtz, „Towards Application Portability in Platform as a Service“ įtraukta *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, Oksfordas, JK, 2014, p. 218–229. DOI: 10.1109/SOSE.2014.26.

[8] C. A. Ardagna, E. Damiani, F. Frati, D. Rebeccani, M. Ughetti, „Scalability Patterns for Platform-as-a-Service,“ įtraukta *2012 IEEE Fifth International Conference on Cloud Computing*, Honolulu, HI, JAV, 2012, p. 718–725. DOI: 10.1109/CLOUD.2012.41.

[9] S, Yangui, P. Ravindran, O. Bibani, R. H. Glitho, N. B. Hadj-Alouane, M. J. Morrow, P. A. Polakos, „A platform as-a-service for hybrid cloud/fog environments“ įtraukta *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, Roma, Italija, 2016, p. 1–7. DOI: 10.1109/LANMAN.2016.7548853.

[10] N. Drake, „Best PaaS provider of 2024“, TechRadar, 01 03 2024. [Tinkle]. Available: https://www.techradar.com/pro/best-paas-provider-of-year. [Kreiptasi 01 09 2024].

[11] D. Gesvindr, O. Gasior, B. Buhnova, „Architecture design evaluation of PaaS cloud applications using generated prototypes: PaaSArch Cloud Prototyper tool“, *Journal of Systems and Software*, t. 169, 110701, 2020. DOI: 10.1016/j.jss.2020.110701.

[12] Z. Li, X. Guo, „Do Google App Engine’s Runtimes Perform Homogeneously? An Empirical Investigation for Bonus Computing“, įtraukta *IEEE Access*, t. 7, p. 4698–4708, 2019. DOI: 10.1109/ACCESS.2018.2889400.

[13] B. Ferriman, T. Hamed, Q. H. Mahmoud, „Storming the cloud: A look at denial of service in the Google App Engine“, įtraukta *2015 International Conference on Computing, Networking and Communications (ICNC)*, Garden Grove, CA, JAV, 2015, p. 363–368. DOI: 10.1109/ICCNC.2015.7069370.

[14] I. Shabani, A. Kovaçi, A. Dika, „Possibilities Offered by Google App Engine for Developing Distributed Applications Using Datastore“, įtraukta *2014 Sixth International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN)*, Tetova, Makedonija, 2014, p. 113–118. DOI: 10.1109/CICSyN.2014.35.

[15] S. Roberts, „Google App Engine: Features, Benefits, and Architecture“, The knowledge academy, 24 08 2023. [Tinkle]. Available:

https://www.theknowledgeacademy.com/blog/google-app-engine-in-cloud-computing/. [Kreiptasi 12 09 2024].

[16] R. Awati, „Google App Engine“, TechTarget. [Tinkle]. Available: https://www.techtarget.com/searchcloudcomputing/definition/Google-App-Engine. [Kreiptasi 12 09 2024].

[17] Google Cloud, „Cloud Load Balancing overview,“ [Tinkle]. Available: https://cloud.google.com/load-balancing/docs/load-balancing-overview. [Kreiptasi 12 09 2024].

[18] Avi Networks, „Google Cloud Load Balancer Definition,“ [Tinkle]. Available: https://avinetworks.com/glossary/google-cloud-load-balancer/. [Kreiptasi 12 09 2024].

[19] Google Cloud, „Memcache API for legacy bundled services,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/legacy/standard/java/memcache. [Kreiptasi 21 09 2024].

[20] Google Cloud, „Using Memcache,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/legacy/standard/java/memcache/using?hl=en. [Kreiptasi 21 09 2024].

[21] Color Orange, „Appengine Memcache Service : Google Cloud Platform,“ 17 06 2016. [Tinkle]. Available: https://medium.com/google-cloud/appengine-memcache-service-google-cloud-platform-8214ee9eada1. [Kreiptasi 21 09 2024].

[22] Google Cloud, „Memorystore for Memcached overview,“ [Tinkle]. Available: https://cloud.google.com/memorystore/docs/memcached/memcached-overview. [Kreiptasi 15 09 2024].

[23] Google Cloud, „Focus on your application, and leave the database to us,“ [Tinkle]. Available: https://cloud.google.com/sql?hl=en. [Kreiptasi 21 09 2024].

[24] Google Cloud, „Datastore Overview,“ [Tinkle]. Available: https://cloud.google.com/datastore/docs/concepts/overview. [Kreiptasi 21 09 2024].

[25] Google Cloud, „Choose an App Engine environment,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/the-appengine-environments. [Kreiptasi 21 09 2024].

[26] Google Cloud, „App Engine standard environment,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/standard. [Kreiptasi 14 09 2024].

[27] Google Cloud, „Go runtime environment,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/standard/go/runtime. [Kreiptasi 14 09 2024].

[28] Google Cloud, „Java runtime environment,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/standard/java-gen2/runtime. [Kreiptasi 14 09 2024].

[29] Google Cloud, „PHP runtime environment,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/standard/php-gen2/runtime. [Kreiptasi 14 09 2024].

[30] Google Cloud, „Python 3 Runtime Environment,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/standard/python3/runtime. [Kreiptasi 14 09 2024].

[31] Google Cloud, „App Engine flexible environment,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/flexible. [Kreiptasi 14 09 2024].

[32] Google Cloud, „About Custom runtimes,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/flexible/custom-runtimes/about-custom-runtimes. [Kreiptasi 14 09 2024].

[33] S. Nishida, Y. Shinkawa, „CPN based GAE performance prediction framework“, įtraukta *2014 9th International Conference on Software Engineering and Applications (ICSOFT-EA)*, Viena, Austrija, 2014, p. 401–406. [Tinkle]. Available: https://ieeexplore.ieee.org/document/7293889. [Kreiptasi 15 09 2024].

[34] R. Prodan, M. Sperk, S. Ostermann, „Evaluating High-Performance Computing on Google App Engine“, įtraukta *IEEE Software*, t. 29, nr. 2, p. 52–58, kov.-bal. 2012. DOI: 10.1109/MS.2011.131.

[35] S. Nishida, Y. Shinkawa, „A Performance Prediction Model for Google App Engine“ įtraukta *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, Krokuva, Lenkija, 2015, p. 134–140. DOI: 10.1109/3PGCIC.2015.9.

[36] U.S. Pandey, A. Jain, „Google App Engine and performance of the Web Application“, *International Journal of Science and Advanced Information Technology*, t. 2, nr. 2, p. 8–16, kov.-bal. 2013. [Tinkle]. Available: https://www.warse.org/IJSAIT/static/pdf/file/ijsait01222013.pdf. [Kreiptasi 15 09 2024].

[37] L. Roberts, „Google App Engine Performance Improvements“, Webtide, 19 07 2024. [Tinkle]. Available: https://webtide.com/google-app-engine-performance-improvements/. [Kreiptasi 15 09 2024].

[38] Google Cloud Platform, „Google App Engine Java new performant HTTP connector“, GitHub Repository, 17 07 2024. [Tinkle]. Available: https://github.com/GoogleCloudPlatform/appengine-java-standard/blob/main/httpconnector.md. [Kreiptasi 07 10 2024].

[39] Google Cloud, „Default encryption at rest,“ [Tinkle]. Available: https://cloud.google.com/docs/security/encryption/default-encryption#key_management. [Kreiptasi 13 10 2024].

[40] Google Cloud, „Encryption in transit,“ [Tinkle]. Available: https://cloud.google.com/docs/security/encryption-in-transit. [Kreiptasi 12 10 2024].

[41] Google Cloud, „Google Cloud Security Whitepapers,“ 03 2018. [Tinkle]. Available: https://services.google.com/fh/files/misc/security_whitepapers_march2018.pdf. [Kreiptasi 13 10 2024].

[42] Google Cloud, „IAM overview,“ [Tinkle]. Available: https://cloud.google.com/iam/docs/overview. [Kreiptasi 13 10 2024].

[43] Google Cloud, „Authenticating users,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/standard/authenticating-users. [Kreiptasi 13 10 2024].

[44] Google Cloud, „Overview of app security,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/standard/application-security. [Kreiptasi 13 10 2024].

[45] C. Silverstein, „Ensuring transaction-safety in Google App Engine“, Khan Academy Blog, 27 06 2016. [Tinkle]. Available: https://blog.khanacademy.org/ensuring-transaction-safety-in-google-app-engine/. [Kreiptasi 20 10 2024].

[46] P. Matesanz; D. Buchner; A. Forderer; A. Koschel; T. Lange; I. Astrova, „ScaLib: Scalable Library System Based on Google App Engine“, įtraukta *2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA)*, Zakintas, Graikija, 2018, p. 1–6. DOI: 10.1109/IISA.2018.8633629.

[47] N. Togashi; V. Klyuev, „A novel approach for web development: A schedule management system using GAE/Go“ įtraukta *2015 IEEE 7th International Conference on Awareness Science and Technology (iCAST)*, Qinhuangdao, Kinija, 2015, p. 55–59. DOI: 10.1109/ICAwST.2015.7314020.

[48] T. Goyal; A. Singh; A. Agrawal, „Cloudtarun Application deployed over GAE and android emulator“, įtraukta *2012 2nd IEEE International Conference on Parallel, Distributed and Grid Computing (PDGC)*, Solanas, Indija, 2012, p. 651–656. DOI: 10.1109/PDGC.2012.6449897.

[49] L. Shi; J. Meng; Y. Zhou; T. Lin, „Construct rough approximation based on GAE“ įtraukta *2013 IEEE International Conference on Granular Computing (GrC)*, Pekinas, Kinija, 2013, p. 259–264. DOI: 10.1109/GrC.2013.6740418.

[50] J. Dean, L. A. Barroso, „The Tail at Scale“, *Communications of the ACM*, vol. 56, nr. 2, p. 74–80, vas., 2013. DOI: 10.1145/2408776.2408794.

[51] M. Kosarchyn, „How Khan Academy Successfully Handled 2.5x Traffic in a Week“, Khan Academy Blog, 09 05 2020. [Tinkle]. Available: https://blog.khanacademy.org/how-khan-academy-successfully-handled-2-5x-traffic-in-a-week/. [Kreiptasi 22 10 2024].

[52] Google Cloud, „Khan Academy: Scaling and simplifying,“ [Tinkle]. Available: https://cloud.google.com/customers/khan-academy. [Kreiptasi 22 10 2024].

[53] Leejjon, „Streaming with Remix on Google Cloud Platform (App Engine Flex & Cloud Run)“, Medium, 10 06 2024. [Tinkle]. Available:

https://leejjon.medium.com/streaming-with-remix-on-google-cloud-platform-app-engine-flex-cloud-run-4e3e16b8c68d. [Kreiptasi 23 10 2024].

[54] B. Kraft, „Profiling App Engine Memcached“, Khan Academy Blog, 01 05 2017. [Tinkle]. Available: https://blog.khanacademy.org/profiling-app-engine-memcached/. [Kreiptasi 20 10 2024].

[55] D. Sharma, F. Chelladurai, „Cloud Storage Security using Firebase and Fernet Encryption,“ *International Journal of Engineering Trends and Technology*, t. 70, p. 371–375, 2022. DOI: 10.14445/22315381/IJETT-V70I9P237.

[56] D. Ardagna, G. Casale, M. Ciavotta, „Quality-of-service in cloud computing: modeling techniques and their applications“, įtraukta *Journal of Internet Services and Applications*, t. 5, nr. 11, 2014. DOI: 10.1186/s13174-014-0011-3.

[57] V. Chang, M. Ramachandran, „Towards Achieving Data Security with the Cloud Computing Adoption Framework,“ įtraukta *IEEE Transactions on Services Computing*, t. 9, nr. 1, p. 138–151, 2016 m. saus.-vas., DOI: 10.1109/TSC.2015.2491281.

[58] S. Cass, „The Top Programming Languages 2024,“ IEEE Spectrum, 22 08 2024. [Tinkle]. Available: https://spectrum.ieee.org/top-programming-languages-2024. [Kreiptasi 02 10 2024].

[59] P. Jain, „Python Frameworks 2024: Breaking Down the Best for Your Development Journey,“ Medium, 22 12 2023. [Tinkle]. Available: https://medium.com/@prashantjain/python-frameworks-2024-breaking-down-the-best-for-your-development-journey-173edc66824e. [Kreiptasi 07 10 2024].

[60] Cryptography, „Fernet (symmetric encryption),“ [Tinkle]. Available: https://cryptography.io/en/latest/fernet/. [Kreiptasi 15 10 2024].

[61] E. R. Collins, „5 Best Ways to Encrypt and Decrypt Data in Python,“ Finxter, 08 03 2024. [Tinkle]. Available: https://blog.finxter.com/5-best-ways-to-encrypt-and-decrypt-data-in-python/. [Kreiptasi 05 10 2024].

[62] Test Guild, „16 Top Load Testing Software Tools for 2024 (Open Source Guide),“ 12 03 2024. [Tinkle]. Available: https://testguild.com/load-testing-tools/#3Locust. [Kreiptasi 15 11 2024].

[63] Iron Contributor, „Little Law of Queuing Theory and How It Impacts Load Testers,“ Microsoft Tech Community, 27 10 2017. [Tinkle]. Available: https://techcommunity.microsoft.com/blog/testingspotblog/little-law-of-queuing-theory-and-how-it-impacts-load-testers/367620. [Kreiptasi 30 11 2024].

[64] Google Cloud, „How instances are managed,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/standard/how-instances-are-managed. [Kreiptasi 29 11 2024].

[65] K. D. Foote, „A Brief History of Cloud Computing“, Dataversity, 17 12 2021. [Tinkle]. Available: https://www.dataversity.net/brief-history-cloud-computing/. [Kreiptasi 31 08 2024].

[66] V. Fryer, „The History of SaaS: From Emerging Technology to Ubiquity“, BigCommerce, [Tinkle]. Available: https://www.bigcommerce.com/blog/history-of-saas/. [Kreiptasi 11 09 2024].

[67] G. Borgoni, „The Evolution of Cloud Computing: From the Early Ideas of John McCarthy to Modern Platforms“, Elemento, 06 07 2023. [Tinkle]. Available: https://www.elemento.cloud/en/post/the-evolution-of-cloud-computing-from-the-early-ideas-of-john-mccarthy-to-modern-platforms. [Kreiptasi 11 09 2024].

[68] G. Aggarwal, „How The Pandemic Has Accelerated Cloud Adoption“, Forbes, 14 04 2022. [Tinkle]. Available: https://www.forbes.com/councils/forbestechcouncil/2021/01/15/how-the-pandemic-has-accelerated-cloud-adoption/. [Kreiptasi 11 09 2024].

[69] Lionel Sujay Vailshery, „Zoom Video Communications daily meeting participants worldwide from 2019 to 2020“, Statista, 20 02 2024. [Tinkle]. Available: https://www.statista.com/statistics/1253972/zoom-daily-meeting-participants-global/. [Kreiptasi 11 09 2024].

[70] Mordor Intelligence, „Cloud Computing Market Size & Share Analysis - Growth Trends & Forecasts (2024 - 2029)“, Mordor Intelligence, 2023. [Tinkle].

Available: https://www.mordorintelligence.com/industry-reports/cloud-computing-market. [Kreiptasi 12 09 2024].

[71] P. Mell, T. Grance, „The NIST Definition of Cloud Computing“, *NIST Special Publication,* Nacionalinis standartų ir technologijų institutas, Gaithersburg, MD, 2011. [Tinkle]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf?lspt_context=gdpr. [Kreiptasi 11 09 2024].

[72] Google Cloud, „What are the different types of cloud computing?“, Heroku, [Tinkle]. Available: https://cloud.google.com/discover/types-of-cloud-computing. [Kreiptasi 31 08 2024].

[73] Heroku, „Deploy and run apps on today's most innovative Platform as a Service“, Heroku, [Tinkle]. Available: https://www.heroku.com/platform. [Kreiptasi 01 09 2024].

[74] Google Cloud, „App Engine,“ [Tinkle]. Available: https://cloud.google.com/appengine?hl=en. [Kreiptasi 01 09 2024].

[75] Google Cloud, „Deploying to App Engine,“ [Tinkle]. Available: https://cloud.google.com/artifact-registry/docs/integrate-app-engine. [Kreiptasi 09 09 2024].

[76] K. K. Ghosh, „Google App Engine - Architecture, Features, Advantages and Limitations“, Infiflex. [Tinkle]. Available: https://www.infiflex.com/google-app-engine--architecture-features-advantages-and-limitations. [Kreiptasi 09 09 2024].

[77] DigitalOcean Documentation, „App Platform Features,“ 01 08 2024. [Tinkle]. Available: https://docs.digitalocean.com/products/app-platform/details/features/. [Kreiptasi 01 09 2024].

[78] Amazon Web Services, „What is AWS Elastic Beanstalk?,“ [Tinkle]. Available: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html. [Kreiptasi 01 09 2024].

[79] Amazon Web Services, „AWS Elastic Beanstalk Deploy and scale web applications,“ [Tinkle]. Available: https://aws.amazon.com/elasticbeanstalk/. [Kreiptasi 01 09 2024].

[80] Amazon Web Services, „AWS Elastic Beanstalk Features,“ [Tinkle]. Available: https://aws.amazon.com/elasticbeanstalk/details/. [Kreiptasi 01 09 2024].

[81] Microsoft Learn, „App Service overview,“ 20 06 2024. [Tinkle]. Available: https://learn.microsoft.com/en-us/azure/app-service/overview. [Kreiptasi 01 09 2024].

[82] Cloudflare, „Cloudflare CDN,“ [Tinkle]. Available: https://www.cloudflare.com/application-services/products/cdn/. [Kreiptasi 20 09 2024].

[83] R. Litterst, „Automattic, the parent to WordPress, is a private company you should know about“, The Hustle, 20 08 2021. [Tinkle]. Available: https://thehustle.co/08202021-automattic. [Kreiptasi 20 09 2024].

[84] Cloudflare, „What is Anycast?,“ [Tinkle]. Available: https://www.cloudflare.com/learning/cdn/glossary/anycast-network/. [Kreiptasi 21 09 2024].

[85] G. Patil, „Exploring the Architecture of Google App Engine“, Medium, 24 01 2023. [Tinkle]. Available: https://medium.com/@gauravpatilsde/exploring-the-architecture-of-google-app-engine-by-gaurav-patil-dab2b9e957d6. [Kreiptasi 13 09 2024].

[86] Google Cloud, „Task Queue Overview,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/legacy/standard/python/taskqueue?hl=en. [Kreiptasi 13 09 2024].

[87] Google Cloud, „Use Cloud Storage,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/standard/using-cloud-storage?tab=python. [Kreiptasi 13 09 2024].

[88] Google Cloud, „Get started with Batch,“ [Tinkle]. Available: https://cloud.google.com/batch/docs/get-started. [Kreiptasi 15 10 2024].

[89] R. Gosavi, „Encryption and Decryption with Fernet in Python,“ Medium, 07 10 2024. [Tinkle]. Available: https://medium.com/@rahulgosavi.94/fernet-encryption-and-decryption-2101f0e3097a. [Kreiptasi 15 11 2024].

[90] F. Toledo, „Top 3 Performance Testing Metrics Explained,“ Abstracta, 20 09 2024. [Tinkle]. Available: https://abstracta.us/blog/performance-testing/performance-testing-metrics/. [Kreiptasi 16 11 2024].

[91] J. Kanjilal, „13 application performance metrics and how to measure them,“ TechTarget, 25 11 2024. [Tinkle]. Available: https://www.techtarget.com/searchapparchitecture/tip/5-application-performance-metrics-all-dev-teams-should-track. [Kreiptasi 20 11 2024].

[92] Coralogix, „Top 10 Application Metrics and Why You Need Them,“ [Tinkle]. Available: https://coralogix.com/guides/observability/application-metrics/. [Kreiptasi 16 11 2024].

[93] Google Cloud, „Memory management best practices,“ [Tinkle]. Available: https://cloud.google.com/memorystore/docs/memcached/memory-management-best-practices. [Kreiptasi 16 01 2024].

[94] Google Cloud, „Supported monitoring metrics for Memorystore for Memcached,“ [Tinkle]. Available: https://cloud.google.com/memorystore/docs/memcached/supported-monitoring-metrics. [Kreiptasi 16 11 2024].

[95] Google Cloud, „Google Cloud metrics,“ [Tinkle]. Available: https://cloud.google.com/monitoring/api/metrics_gcp. [Kreiptasi 16 11 2024].

[96] R. Gershman, „Google Cloud Memorystore - Proven Best Practices,“ Dragonfly, 25 09 2024. [Tinkle]. Available: https://www.dragonflydb.io/guides/gcp-memorystore-best-practices. [Kreiptasi 05 12 2024].

[97] MDN Web Docs, „How the web works,“ [Tinkle]. Available: https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Web_standards/How_the_web_works. [Kreiptasi 08 12 2024].

[98] J. Lake, „What is Fernet? Secure data encryption in Python,“ Comparitech, 05 09 2023. [Tinkle]. Available: https://www.comparitech.com/blog/information-security/what-is-fernet/. [Kreiptasi 07 12 2024].

[99] N. Kaleev, „Time to First Byte (TTFB): What It is and How to Reduce It,“ NitroPack, 14 02 2024. [Tinkle]. Available: https://nitropack.io/blog/post/time-to-first-byte-ttfb. [Kreiptasi 09 12 2024].

[100] Firebase, „Firebase Authentication,“ [Tinkle]. Available: https://firebase.google.com/docs/auth/. [Kreiptasi 05 12 2024].

[101] Firebase, „Manage User Sessions,“ [Tinkle]. Available: https://firebase.google.com/docs/auth/admin/manage-sessions. [Kreiptasi 06 12 2024].

[102] Firebase, „Authentication State Persistence,“ [Tinkle]. Available: https://firebase.google.com/docs/auth/web/auth-state-persistence. [Kreiptasi 06 12 2024].

[103] Google Cloud, „Quotas and limits,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/standard/quotas. [Kreiptasi 05 12 2024].

[104] Firebase, „Verify ID Tokens,“ [Tinkle]. Available: https://firebase.google.com/docs/auth/admin/verify-id-tokens. [Kreiptasi 06 12 2024].

[105] Google Cloud, „The .NET runtime,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/flexible/dotnet/runtime. [Kreiptasi 17 09 2024].

[106] Google Cloud, „The Go runtime,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/flexible/go/runtime. [Kreiptasi 14 09 2024].

[107] Google Cloud, „The Java runtime,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/flexible/java/runtime. [Kreiptasi 17 09 2024].

[108] Google Cloud, „The Node.js runtime,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/flexible/nodejs/runtime. [Kreiptasi 17 09 2024].

[109] Google Cloud, „The PHP runtime,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/flexible/php/runtime. [Kreiptasi 17 09 2024].

[110] Google Cloud, „The Python runtime,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/flexible/python/runtime. [Kreiptasi 17 09 2024].

[111] Google Cloud, „The Ruby runtime,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/flexible/ruby/runtime. [Kreiptasi 17 09 2024].

[112] M. Bayramov, „LRU(Least Recently Used) Cache,“ Medium, 04 09 2023. [Tinkle]. Available: https://medium.com/@mayilb77/lru-least-recently-used-cache-82bb279769e7. [Kreiptasi 24 10 2024].

[113] Google Cloud, „Node.js Runtime Environment,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/standard/nodejs/runtime. [Kreiptasi 17 09 2024].

[114] Google Cloud, „Ruby Runtime Environment,“ [Tinkle]. Available: https://cloud.google.com/appengine/docs/standard/ruby/runtime. [Kreiptasi 17 09 2024].

1 PRIEDAS

Debesų kompiuterijos istoriją, vystymosi etapus ir įtaką technologijų bei verslo aplinkai

Debesų kompiuterijos istorija siekia 1963 metus, kai DARPA (Gynybos pažangiųjų tyrimų projektų agentūra) finansavo MIT projektą MAC. Šis projektas buvo itin reikšmingas, nes pirmą kartą buvo pradėta plėtoti virtualizacijos sąvoka. Projektas siekė užtikrinti efektyvų resursų naudojimą, leidžiant vienu metu keliems vartotojams naudotis vienu kompiuteriu. Ši idėja tapo pagrindiniu debesų kompiuterijos principu. Be technologinio proveržio, projektas buvo svarbus ir dėl požiūrio į kompiuterinių išteklių efektyvumą, kuris vėliau tapo debesų kompiuterijos pagrindu. [65]

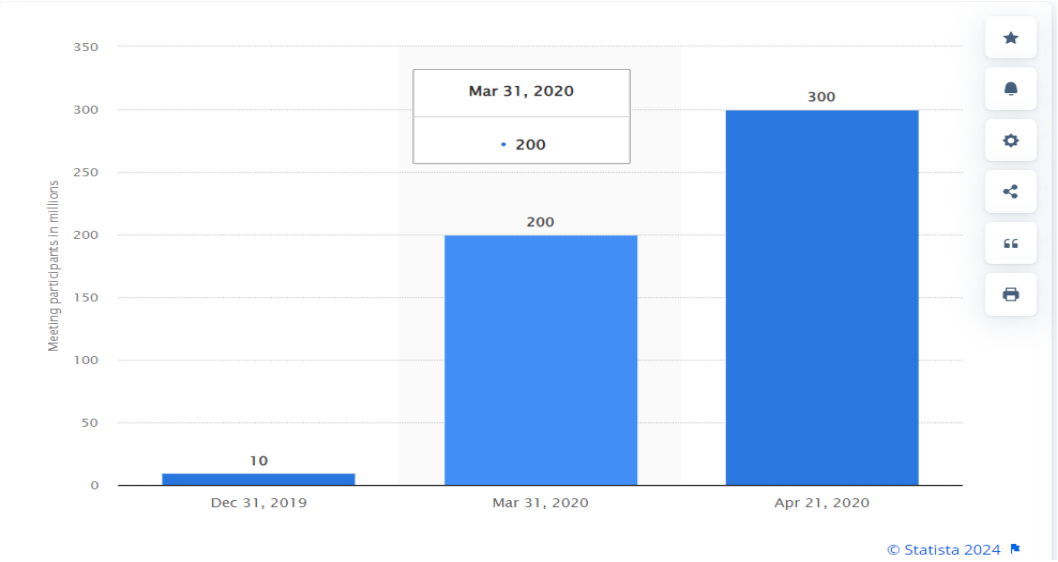
1969 m. J. C. R. Licklider sukūrė ARPANET – pirmąją interneto versiją, kuri padėjo pagrindą debesų kompiuterijos plėtrai. ARPANET leido kompiuteriams bendrauti tarpusavyje ir dalytis informacija per tinklą. Lickliderio vizija apie „Intergalaktišką kompiuterių tinklą“, kur visi pasaulio kompiuteriai būtų sujungti, tapo pagrindu moderniam internetui. ARPANET ir jo plėtra buvo būtini debesų kompiuterijos evoliucijai, nes suteikė infrastruktūrą ir technologinę bazę debesų paslaugoms teikti. Virtualizacija, prasidėjusi nuo MIT MAC projekto ir išvystyta su ARPANET technologijomis, tapo debesų kompiuterijos dalimi. Virtualios mašinos, veikiančios kaip realūs kompiuteriai su visomis funkcijomis, leido plėtoti debesų kompiuterijos infrastruktūrą ir skatino virtualių privačių tinklų bei kompleksinių debesų paslaugų augimą. [65]

1999 m. *Salesforce* tapo pirmąja kompanija, kuri sėkmingai pradėjo naudoti debesų kompiuteriją, siūlydama programinę įrangą internetu. Tai leido įmonėms lengviau ir pigiau naudotis programomis, atsisakant brangios programinės įrangos diegimo. *Salesforce* sukūrė naują verslo modelį – SaaS (Software as a Service), kuris leido įmonėms naudotis internetiniais sprendimais, remiantis prenumeratos modeliu. Tai tapo efektyviu sprendimu net didelėms organizacijoms. [66]

2006 m. *Amazon Web Services* (AWS) pradėjo teikti IaaS (*Infrastructure as a Service*) paslaugas, suteikdama verslams galimybę nuomotis virtualius serverius. Šis sprendimas ženkliai sumažino infrastruktūros kaštus ir atvėrė naujas galimybes startuoliams bei inovatyvioms įmonėms. [65] [67] Tais pačiais metais *Google* pradėjo siūlyti *Google Docs*, suteikdama galimybę saugoti ir redaguoti dokumentus internete. 2010 m. debesų kompiuterija ir toliau sparčiai vystėsi, išpopuliarėjo privatūs ir hibridiniai debesys, o į šią rinką įžengė tokios kompanijos kaip *Oracle* ir *Apple*, siūlančios kompleksines debesų paslaugas (IaaS, PaaS, SaaS). [65]

2019 m. pabaigoje prasidėjusi COVID-19 pandemija tapo paskata spartesniam skaitmeninės transformacijos įgyvendinimui, ypač debesų kompiuterijos srityje. Daugelis organizacijų, siekdamos

užtikrinti veiklos tęstinumą ir prisitaikyti prie nuotolinio darbo bei socialinio atsiribojimo reikalavimų, greitai perkėlė savo veiklas į debesų platformas. Remiantis *Gartner* duomenimis, pasaulinės išlaidos debesų paslaugoms 2021 m. turėjo išaugti 18,4% iki 304,9 mlrd. JAV dolerių. Pandemija ne tik paskatino įmonių atsparumo paieškas, bet ir išryškino poreikį naujiems veiklos modeliams, tokiems kaip nuotolinis darbas, e. prekyba ir skaitmeninės sveikatos priežiūros paslaugos. Tai rodo ilgalaikę tendenciją, kad vis daugiau organizacijų persikelia į skaitmeninę erdvę. [68] Pavyzdžiui, *Zoom* platformos naudojimas išaugo eksponentiškai (žr. 1 pav.). Iki 2020 m. balandžio mėnesio *Zoom Video Communications* patyrė kasdienių susitikimų dalyvių skaičiaus padidėjimą, pasiekdama 300 milijonų pasaulyje. Šis skaičius yra reikšmingai didesnis nei 10 milijonų, kurie buvo užfiksuoti 2019 m. pabaigoje. *COVID-19* pandemija tapo pagrindiniu šio augimo katalizatoriumi, nes ji paskatino įmones visame pasaulyje įdiegti *Zoom* kaip pagrindinę priemonę palaikyti ryšį tarp darbuotojų ir klientų dirbant nuotoliniu būdu. Padidėjęs *Zoom* naudojimas 2020 m. taip pat buvo papildytas asmeniniu platformos naudojimu, siekiant išlaikyti ryšius su draugais ir šeima. [69]



Šaltinis: [69]

1 pav. *Zoom Video Communications* kasdienių susitikimų dalyviai visame pasaulyje 2019–2020 m. (milijonais)

2 PRIEDAS

Debesų kompiuterijos rinkos plėtra, pagrindinės charakteristikos ir paslaugų modeliai

Remiantis naujausiais tyrimais, debesų kompiuterijos rinka auga itin sparčiai. Prognozuojama, kad šios rinkos dydis 2024 metais sieks 0,68 trilijonus JAV dolerių, o iki 2029 metų išaugs iki 1,44 trilijono JAV dolerių, pasiekdama 16,40% metinį augimo tempą (CAGR) (žr. 1 pav.**Klaida! Nerastas nuorodos šaltinis.**). Augimą palaiko sparčiai besivystančios technologijos, tokios kaip dirbtinis intelektas (AI), didieji duomenys ir mašininis mokymasis (ML), kurie padeda kurti verslo sprendimus įvairiose srityse: verslo išteklių planavime (ERP), operacijų valdyme (OLTP) ir tiekimo grandinės valdyme (SCM). [70]

Šaltinis: [70]

1 pav. Debesų kompiuterijos rinka

Remiantis NIST (*National Institute of Standards and Technology*), debesų kompiuterija apibūdinama kaip modelis, leidžiantis universalią ir patogią prieigą prie tinklo resursų, kurie gali būti greitai suteikti ir naudojami pagal poreikį su minimaliu paslaugų tiekėjo įsitraukimu. [71].

Kaip nurodo NIST, pagrindinės debesų kompiuterijos ypatybės apima šiuos aspektus:

1. Savitarna pagal poreikį (angl. o*n-demand self-service*): vartotojai gali savarankiškai užsisakyti skaičiavimo išteklius, tokius kaip serverio laikas ar tinklo saugykla, be būtinybės tiesiogiai bendrauti su paslaugų teikėjais. Tai supaprastina resursų valdymą ir pagreitina prieigą prie reikalingų paslaugų. [71]

2. Platus tinklo prieinamumas (angl. *broad network access*): paslaugos gali būti pasiekiamos per tinklą ir prieinamos per įvairias platformas, įskaitant mobiliuosius telefonus, planšetinius kompiuterius, nešiojamuosius ar stalinius kompiuterius. Tai skatina debesų paslaugų prieinamumą iš bet kurio įrenginio, prisijungusio prie tinklo. [71]
3. Išteklių telkimas (angl. *resource pooling*): paslaugų teikėjų kompiuteriniai ištekliai telkiami, kad būtų galima aptarnauti kelis vartotojus naudojant daugiabučių modelį (angl. *multi-tenant*). Ištekliai dinamiškai priskiriami pagal vartotojų poreikius, tačiau vartotojai neturi tiesioginės įtakos jų fizinei vietai, nors gali valdyti išteklių lokacijos lygmenyje, pvz., pasirinkti šalį ar duomenų centrą. [71]
4. Greitas elastingumas (angl. *rapid elasticity*): ištekliai gali būti greitai pritaikomi didėjantiems ar mažėjantiems vartotojų poreikiams. Vartotojams dažnai atrodo, kad ištekliai yra neriboti. Ši elastingumo galimybė padeda užtikrinti, kad paslaugos būtų teikiamos efektyviai, nepriklausomai nuo paklausos svyravimų. [71]
5. Matuojama paslauga (angl. *measured service*): debesų kompiuterijos sistemos automatiškai optimizuoja resursų naudojimą, remiantis matavimo mechanizmu (angl. *metering capability*), kuris leidžia stebėti, kontroliuoti ir pranešti apie vartotojų naudojamus išteklius. Tai padeda stebėti išteklių suvartojimą bei apskaičiuoti sąskaitas pagal realų naudojimą. [71]

Be šių ypatybių, debesų kompiuterija remiasi virtualios infrastruktūros principais, kuriuose fiziniai ištekliai, tokie kaip procesoriai, atmintis ir tinklo pajėgumai, yra pateikiami kaip virtualūs ištekliai. Taip pat vyksta dinamiškas resursų priskyrimas, užtikrinantis, kad ištekliai būtų pritaikomi realiuoju laiku, atsižvelgiant į vartotojų poreikius, ir išlaikomas aukštas saugumo bei patikimumo lygis. Debesų kompiuterijos paslaugos prieinamos įvairiuose įrenginiuose, įskaitant asmeninius kompiuterius, nešiojamuosius kompiuterius ar mobiliuosius įrenginius, taip pat valdomos ir automatizuojamos, kad vartotojai mokėtų tik už faktiškai naudotus išteklius. [4]

Debesų kompiuterijos infrastruktūra gali būti diegiama naudojant skirtingus modelius:

1. Privatus debesis (angl. *private cloud*): infrastruktūra yra naudojama tik vienai organizacijai ir gali būti valdoma viduje arba trečiosios šalies. [71]
2. Viešasis debesis (angl. *public cloud*): infrastruktūra yra prieinama visiems vartotojams ir valdoma trečiosios šalies. [71]
3. Hibridinis debesis (angl. *hybrid cloud*): infrastruktūra yra sudaryta iš kelių skirtingų debesijos tipų (pvz., privataus ir viešojo debesies), kurie sąveikauja tarpusavyje per standartizuotas technologijas . [71]

Debesų kompiuterija teikia įvairius paslaugų modelius, pritaikytus skirtingiems verslo poreikiams. Svarbiausi debesų paslaugų modeliai yra IaaS (infrastruktūra kaip paslauga), PaaS (platforma kaip paslauga) ir SaaS (programinė įranga kaip paslauga). Kiekvienas iš šių modelių turi specifinius privalumus ir taikymo sritis. [72]

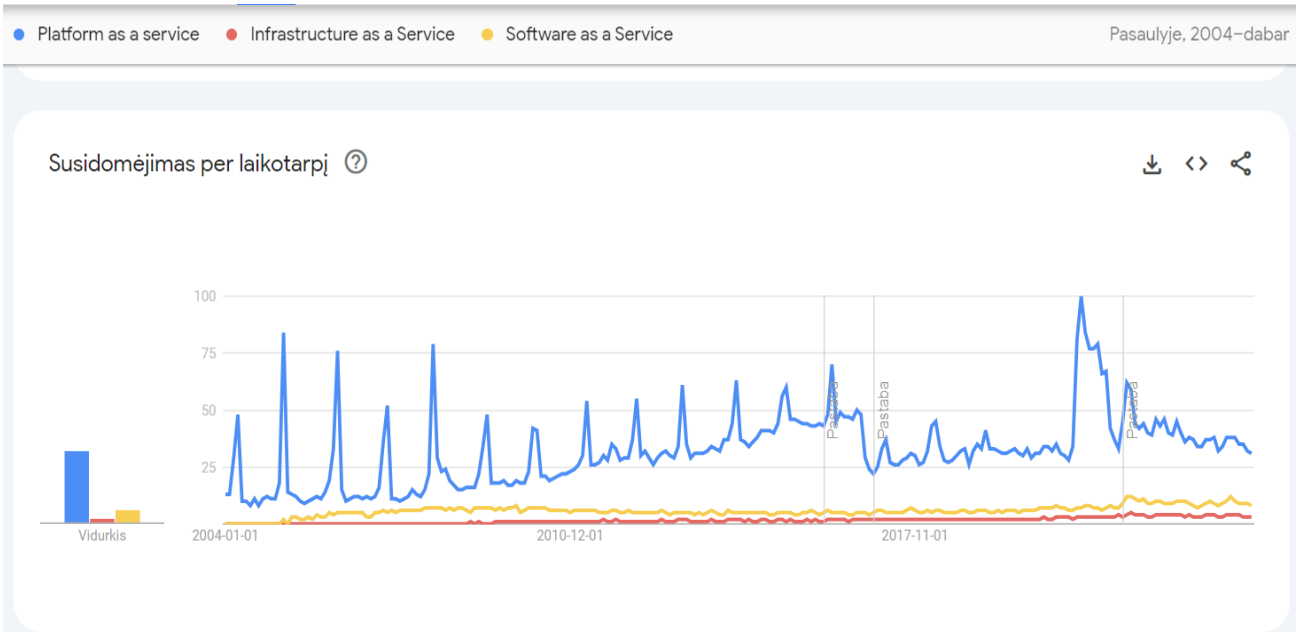
IaaS suteikia prieigą prie svarbių IT infrastruktūros išteklių, tokių kaip skaičiavimo galia, saugyklos ir tinklai. Pasirinkus šį modelį, vartotojai nuomojasi reikiamus išteklius iš debesų paslaugų teikėjo, tačiau yra atsakingi už operacinių sistemų, taikomųjų programų ir duomenų valdymą. IaaS modelis ypač naudingas įmonėms, kurios siekia išlaikyti visišką kontrolę ir lankstumą valdant savo infrastruktūrą be fizinės techninės įrangos priežiūros. [72]

PaaS suteikia integruotą aplinką, leidžiančią kurti, testuoti, diegti ir valdyti taikomąsias programas. Šiame modelyje debesų paslaugų teikėjas atsakingas už infrastruktūros valdymą, todėl kūrėjai gali susitelkti į programinės įrangos kūrimą ir diegimą. PaaS yra tinkamas sprendimas įmonėms, siekiančioms pagreitinti programinės įrangos kūrimo procesą, nes jis eliminuoja poreikį rūpintis infrastruktūros technine priežiūra. [72]

SaaS modelis suteikia visiškai paruoštą naudoti programinę įrangą, prie kurios vartotojai gali prisijungti internetu. Debesų paslaugų teikėjas valdo visą infrastruktūrą, operacines sistemas, duomenų saugyklas ir pačią programinę įrangą, todėl vartotojai gali tiesiog naudotis programomis ir neturi atsakyti už jų priežiūrą ar atnaujinimą. SaaS modelis populiarus tarp įmonių, kurios siekia greitai ir patogiai naudotis įvairiomis programomis be techninių rūpesčių. [72]

Kiekvienas iš šių debesų paslaugų modelių siūlo skirtingas galimybes, leidžiančias įmonėms pasirinkti sprendimus, geriausiai atitinkančius jų specifinius poreikius.

Paveiksle (žr. 2 pav.) matoma, kaip pasiskirstė trijų debesijos paslaugų modelių – PaaS, IaaS ir SaaS – populiarumas pagal *Google Trends* paieškas visame pasaulyje nuo 2004 metų. Iš diagramos matome, kad PaaS modelis sulaukė daugiausia susidomėjimo per visą analizuojamą laikotarpį, ženkliai lenkdamas IaaS ir SaaS, kurios išliko nuosaikesnės (mėlyna spalva žymi PaaS, raudona spalva – IaaS, geltona spalva – SaaS paieškos kiekio rezultatus). Vis dėlto SaaS paslaugų augimas pastaraisiais metais taip pat yra pastebimas, ypač po 2017 metų, kai debesijos paslaugų poreikis rinkoje smarkiai išaugo.



Šaltinis: *Google Trends*. *Susidomėjimas per laikotarpį*. Žiūrėta: 2024 08 31 per internetą: <https://trends.google.com/trends/explore?date=all&q=%2Fm%2F0414krx,Infrastructure%20as%20a%20Service,Software%20as%20a%20Service&hl=lt>

2 pav. PaaS, IaaS, SaaS populiarumas žiniatinklio *Google* paieškoje nuo 2004 m. iki dabar visame pasaulyje remiantis *Google Trends* duomenimis

3 PRIEDAS

Platform as a service sprendimų apžvalga: savybės, privalumai ir populiarumo analizė lietuvos rinkoje

Vieni populiariausių PaaS tiekėjų yra *Heroku, Google App Engine, DigitalOcean App Platform, AWS Elastic Beanstalk, Microsoft Azure App Service*. [10]

„Heroku“ yra viena iš populiariausių PaaS paslaugų, išsiskirianti lengvu naudojimu, leidžiančiu kūrėjams greitai diegti programas per komandų eilutę ar web sąsają. Platforma palaiko įvairias programavimo kalbas, tokias kaip *Ruby, Java, Python, PHP, Scala, Clojure* ir *Go,* bei automatiškai atnaujina sistemines ir programavimo kalbų aplinkas. Tai leidžia kūrėjams susitelkti į programų kūrimą, nesirūpinant technine priežiūra. [10] [73] *Heroku* naudoja *dynos* konteinerius, kurie užtikrina patikimą veikimą, o ištekliai automatiškai paskirstomi pagal programų apkrovą. Papildomai, platforma siūlo integruojamas paslaugas, tokias kaip *Heroku Postgres, Redis* ir *Apache Kafka*, kurios praplečia programų funkcionalumą. [73] Saugumo srityje *Heroku* atitinka PCI, HIPAA, ISO ir SOC standartus, reguliariai vykdo auditus, užtikrindama duomenų saugumą ir patikimumą. Vis dėlto *Heroku* labiau tinkama mažiems ir vidutiniams projektams, nes didėjant srautui ir išteklių naudojimui, auga ir paslaugos kaštai, o tai gali apriboti jos pritaikymą didelio masto aplikacijoms. [10] [73]

GAE yra pažangi PaaS paslauga, išsiskirianti lankstumu ir integracija su *Google Cloud* ekosistema. Platforma palaiko įvairias programavimo kalbas, tokias kaip *Java, Go, Node.js, PHP, Python, Ruby, .NET*, bei leidžia naudoti *Docker* konteinerius, suteikdama galimybę diegti bet kokio tipo programas, nepriklausomai nuo naudojamos kalbos ar technologijos. GAE automatiškai rūpinasi resursų didinimu, apkrovos balansavimu ir infrastruktūros priežiūra, todėl kūrėjai gali susitelkti į programų kūrimą, išvengdami sudėtingų administracinių užduočių. Šis automatizavimas leidžia efektyviai naudoti resursus ir mažina priežiūros kaštus. Platforma integruoja *Google Cloud* įrankius, tokius kaip *Cloud Monitoring, Logging* ir *Cloud Debugger*, kurie padeda stebėti programų veikimą, greitai identifikuoti ir taisyti klaidas. GAE taip pat suderinama su *Cloud SQL, Cloud Storage* ir *BigQuery,* suteikdama kūrėjams platų papildomo funkcionalumo spektrą. Saugumo srityje GAE leidžia apibrėžti prieigos taisykles per ugniasienę ir suteikia nemokamą SSL/TLS sertifikatų integraciją pasirinktiniams domenams, taip užtikrindama duomenų apsaugą. GAE galinga Google infrastruktūra užtikrina efektyvų didelių apkrovų valdymą bei aukštą našumą. Vis dėlto, nors GAE siūlo lankstų kainodaros modelį, intensyvus platformos naudojimas gali smarkiai padidinti kaštus, todėl ji laikoma vienu iš brangesnių pasirinkimų. [10] [74] [75] [76]

DigitalOcean App Platform yra efektyvi PaaS paslauga, kuria paprasta naudoti ir kuri palaiko pagrindines programavimo kalbas, tokias kaip *Python, Node.js, PHP, Ruby* ir *Go*, bei leidžia dirbti su konteineriais, suteikiant lankstumo naudoti ir neoficialiai palaikomas kalbas. Platforma automatiškai tvarko infrastruktūrą, įskaitant SSL sertifikatų atnaujinimą ir apsaugą nuo DDOS atakų, todėl kūrėjai gali susitelkti į programos kūrimą, o ne technines detales. [10] [77] *DigitalOcean App Platform* išsiskiria prieinama kainodara, todėl ji yra patraukli mažesnio biudžeto projektams ir alternatyva didesnėms platformoms, tokioms kaip *AWS, Azure* ar *Google Cloud* [10]. Paprastas diegimo procesas, įskaitant automatinį diegimą iš *GitHub* ir intuityvią valdymo sąsają, leidžia greitai perprasti platformos naudojimo specifiką [10] [77]. Platforma palaiko resursų didinimą tiek horizontaliai, tiek vertikaliai, o dedikuoti CPU suteikia galimybę pasiekti aukštesnį našumą ir užtikrintus resursus. Be to, platforma integruota su *DigitalOcean Databases*, įskaitant *PostgreSQL, MySQL* ir *Redis,* o tai leidžia paprastai valdyti duomenų bazes. [77] Nors *DigitalOcean App Platform* siūlo patogumą ir prieinamumą, jos funkcionalumas yra ribotas, todėl ji gali būti mažiau tinkama projektams, kuriems reikalingos išplėstines galimybės [10].

AWS Elastic Beanstalk yra lanksti PaaS paslauga, kuri palaiko įvairias programavimo kalbas, tokias kaip *Go, Java, .NET, Node.js, PHP, Python, Ruby,* bei *Docker* konteinerius, suteikdama kūrėjams pasirinkimo laisvę ir lankstumą. Platforma leidžia lengvai diegti ir valdyti programas AWS debesyje, naudojant *AWS Management Console* ar AWS CLI, taip supaprastinant infrastruktūros valdymą. *Elastic Beanstalk* automatiškai parengia reikalingus AWS resursus, įskaitant Amazon EC2, RDS ir S3, taip pat užtikrina apkrovos balansavimą ir resursų didinimą, efektyviai paskirstydama apkrovas. [78] [79] [80] *Elastic Beanstalk* siūlo integraciją su *Amazon CloudWatch* ir *AWS X-Ray,* kurie leidžia stebėti programų našumą, sveikatą bei nustatyti perspėjimus dėl pasiektų ribinių rodiklių [79] [80]. Platforma taip pat atitinka tarptautinius saugumo standartus, tokius kaip ISO, PCI, SOC 1, SOC 2, SOC 3 ir HIPAA, todėl yra tinkama sprendimams, kuriems reikalingi šie sertifikatai [80]. Be to, kainodaros modelis priklauso tik nuo naudojamų AWS resursų, o pati *Elastic Beanstalk* platforma nėra papildomai apmokestinama, o tai suteikia lankstumo valdant kaštus [10] [78]. Vis dėlto, nors AWS infrastruktūra yra patikima, *Elastic Beanstalk* programų derinimo ir stebėjimo įrankiai gali būti sudėtingi naudoti, o tai apsunkina programų priežiūrą. Taip pat, didėjant srautui, platformos kaštai gali augti, todėl aukšto srauto programos gali patirti didesnes išlaidas. [10]

Microsoft Azure App Service yra pažangi PaaS platforma, skirta *web* aplikacijų, REST API ir mobiliųjų *back-end* kūrimui bei valdymui. Ši platforma palaiko įvairias programavimo kalbas, tokias kaip *.NET, .NET Core, Java, Ruby, Node.js, PHP* ir *Python*, taip pat leidžia naudoti *Docker* konteinerius, suteikdama kūrėjams lankstumo ir galimybes integruoti skirtingas technologijas. *Azure App Service* siūlo pilnai valdomą aplinką, automatizuojančią infrastruktūrines užduotis, tokias kaip

operacinės sistemos ir programavimo kalbų pagrindų atnaujinimai. Valdymas vykdomas per *Azure Portal*, CLI ar *PowerShell*, o diegimui galima naudoti tokias priemones kaip *Visual Studio*, *GitHub* ir *Bitbucket*. Platforma užtikrina horizontalų ir vertikalų resursų didinimą, leidžiantį efektyviai valdyti apkrovą tiek mažoms, tiek didelėms aplikacijoms. Automatizuotas apkrovos balansavimas ir resursų didinimas užtikrina aplikacijų prieinamumą bei stabilumą. Be to, *Azure App Service* atitinka aukštus tarptautinius saugumo standartus, tokius kaip ISO, SOC ir PCI, todėl yra tinkama naudoti reguliuojamose pramonės šakose. Nors platforma siūlo daug funkcionalumo, sudėtinga kainodaros struktūra gali apsunkinti galutinių hosting'o išlaidų įsivertinimą. [10] [81]

Top In Cloud PaaS Usage Distribution in Lithuania

Technology	Websites	%
Google	33,619	56.1
Cloudflare Hosting	18,363	30.64
Amazon	6,693	11.17
Vercel	478	0.8
Microsoft Azure	476	0.79
Automattic	295	0.49
Heroku	4	0.01
Azion	1	0
Salesforce Hosted	1	0

Šaltinis: Built with (2024 10 19). *Cloud PaaS Usage Distribution in Lithuania*. Žiūrėta: 2024 10 26 per internetą: <https://trends.builtwith.com/hosting/cloud-paas/country/Lithuania>

1 pav. Debesų PaaS naudojimo pasiskirstymas Lietuvoje

Atsižvelgiant į Lietuvos rinką, debesų PaaS naudojimo pasiskirstymas yra gana ženklus. Tai galime pamatyti 1 pav., kuriame aiškiai matyti, kaip skirtingi PaaS tiekėjai yra naudojami internetinėms svetainėms Lietuvoje. Kiekviena platforma nurodyta pagal svetainių, naudojančių jų technologijas, skaičių ir procentinę viso naudojimo dalį. *Google* užima didžiausią rinkos dalį Lietuvoje, aptarnaudama 33,619 svetainių, o tai sudaro 56.1 % visos PaaS rinkos. Tai rodo, kad *Google Cloud* ekosistema, įskaitant tokias paslaugas kaip GAE ir GCP, yra itin populiari Lietuvoje. *Cloudflare Hosting* yra antras pagal populiarumą, naudojamas 18,363 svetainėse, sudarydamas 30.64 % rinkos dalies. *Cloudflare* yra gerai žinoma dėl savo turinio pristatymo tinklo (CDN) ir apsaugos nuo kibernetinių atakų [82], todėl tai patrauklus pasirinkimas daugeliui svetainių. Amazon (*Amazon Web Services* – AWS) aptarnauja 6,693 svetaines, užimdama 11.17 % rinkos. Tai patvirtina, kad AWS yra svarbus žaidėjas debesų kompiuterijos srityje, nors ir ne toks dominuojantis kaip *Google* ar *Cloudflare*. *Vercel* naudojama 478 svetainėse ir sudaro 0.8 % rinkos, o *Microsoft Azure* aptarnauja

476 svetaines, užimdama 0.79 % rinkos dalies. Nors *Azure* yra viena iš didžiųjų pasaulinių debesų tiekėjų, Lietuvoje jos naudojimas yra ganėtinai mažas, lyginant su *Google* ir AWS. *Automattic*, platforma, gerai žinoma kaip *WordPress.com* kūrėja [83], naudojama 295 svetainėse, sudarydama 0.49 % rinkos. Tai rodo, kad *WordPress* platforma vis dar yra populiari tarp svetainių, kurios naudojasi PaaS sprendimais. *Heroku*, *Azion* ir *Salesforce Hosted* naudojamos labai nedaug – turi tik po 1 ar 4 svetaines kiekviena, ir jų procentinė rinkos dalis yra beveik nulinė. Nors *Heroku* ir panašios platformos yra populiarios kitose rinkose dėl gana paprastų PaaS sprendimų, Lietuvoje jų naudojimas yra labai menkas. Visi šie duomenys trends.builtwith.com svetainėje yra fiksuoti 2024 metų spalio 19-ąją, todėl galima teigti, kad informacija vis dar yra aktuali šiai dienai.

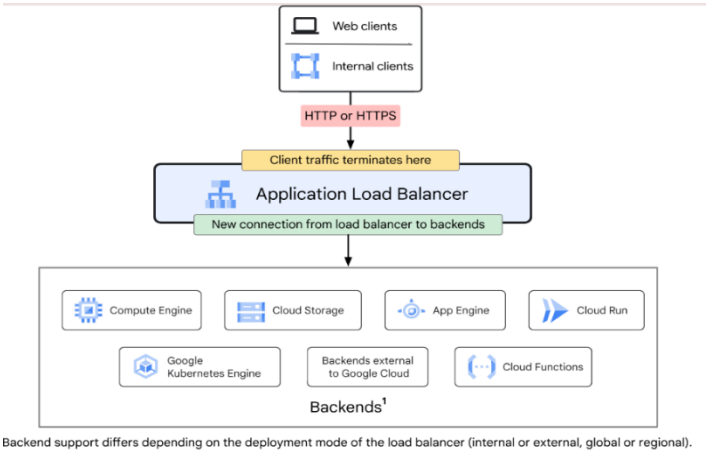
Remiantis PaaS tiekėjų populiarumo statistika Lietuvoje, galime teigti, kad *Google* platforma, užimdama 56.1 % visos rinkos, smarkiai lenkia kitus tiekėjus, tokius kaip *Amazon* ar *Microsoft Azure*. Tai rodo, kad Lietuvos rinkoje *Google* infrastruktūra yra plačiai naudojama, o jos siūlomos paslaugos, įskaitant GAE, yra itin populiarios.

4 PRIEDAS

GAE architektūros komponentų apžvalga

Vienas iš pagrindinių GAE architektūros komponentų yra *Google Load Balancer*. Šis komponentas naudoja *Anycast* tinklo technologiją. *Anycast* yra maršrutizavimo būdas, leidžiantis nukreipti užklausas į įvairius duomenų centrus. Šis maršrutizavimo metodas parenka artimiausią duomenų centrą, galintį apdoroti užklausą, taip sumažindamas vėlavimą (angl. *latency*) ir optimizuodamas tinklo veikimą. *Anycast* tinklai padeda paskirstyti apkrovą tarp skirtingų mazgų, užtikrindami atsparumą dideliam srautui ir galimai tinklo perkrovai ar DDoS atakoms. Naudojant šį metodą, užklausos pasiekia vieną bendrą IP adresą, o tinklas automatiškai parenka tinkamiausią centrą, kuris efektyviausiai apdoros užklausą. [84] Tai leidžia tinklui veikti efektyviai ir patikimai, nepriklausomai nuo užklausų srauto ar jų paskirstymo.

Google Load Balancer taip pat suteikia galimybę dinamiškai keisti infrastruktūros išteklius, kai auga arba mažėja vartotojų srautas, nes sistema palaiko *seamless autoscaling* funkcionalumą, kuris leidžia prisitaikyti prie netikėtų apkrovos šuolių be išankstinio pasiruošimo (angl. *pre-warming*). Tai ypač svarbu didelio srauto metu, pavyzdžiui, esant netikėtiems vartotojų skaičiaus padidėjimams. Be to**,** *Google Load Balancer* palaiko integraciją su saugumo sprendimais, pvz., *Google Cloud Armor***,** kuris apsaugo infrastruktūrą nuo paskirstytų atsisakymo paslaugos (DDoS) atakų ir kitų tikslinių aplikacijų atakų. Tokios funkcijos kaip visada įjungta DDoS apsauga užtikrina papildomą saugumo sluoksnį, ypač naudojant išorinį *Application Load Balancer* arba *Network Load Balancer*. [17] Tai leidžia apsaugoti tiek globaliai, tiek regioniniu lygmeniu esančias aplikacijas nuo potencialių grėsmių.



Šaltinis: *[17]*

1 pav. Programos apkrovos balansavimo

1 pav. pavaizduota *Google Cloud* apkrovos balansavimo architektūra, kuri leidžia efektyviai paskirstyti vartotojų užklausas tarp įvairių infrastruktūrų ir *backend* sistemų, užtikrinant optimalų našumą ir patikimumą. [17] Paveikslėlio viršuje pateikti užklausų šaltiniai, tokie kaip *web* klientai ir vidiniai klientai (angl. *internal clients*), kurie gali būti sujungti per internetą arba vidinį tinklą (1 pav.). Šios užklausos pasiekia *Application Load Balancer*, kuris yra pagrindinis taškas, skirtas vartotojų užklausoms priimti ir apdoroti per HTTP arba HTTPS protokolus. [17] Balansavimo mechanizmas paskirsto užklausas tarp įvairių *backend* sistemų, atsižvelgiant į jų apkrovą ir pajėgumus. Kai užklausos pasiekia balansavimo tašką, jos nukreipiamos į atitinkamus *backend* serverius, suformuojant naujas jungtis tarp apkrovos balansatoriaus ir *backend* serverių. [18] Remiantis 1 pav., *Application Load Balancer* gali nukreipti užklausas į įvairias *Google Cloud* infrastruktūras, tokias kaip *Compute Engine, Cloud Storage, App Engine, Cloud Run, Google Kubernetes Engine* ir *Cloud Functions.* Taip pat yra galimybė nukreipti užklausas į išorinius *backend* serverius, kurie nėra tiesiogiai integruoti į *Google Cloud* infrastruktūrą, bet gali būti naudojami su ja. [17] Svarbu paminėti tai, kad *Application Load Balancer* palaikymas gali skirtis priklausomai nuo naudojamo balansavimo tipo (pvz., globalus, regioninis, vidinis ar išorinis). Šis teiginys yra pateikiamas paveikslėlio 1 pav.**Klaida! Nerastas nuorodos šaltinis.** apačioje.

Kitas GAE architektūros komponentas - *front-end app*. Komponentas yra atsakingas už užklausų nukreipimą į tinkamus paslaugų modulius. Kai vartotojas pateikia užklausą, *front-end* dalis priima ją ir perduoda atitinkamai programos daliai arba servisui. [85]

Dar vienas iš svarbių komponentų yra *Memcache*. GAE palaiko du *Memcache* paslaugų lygius. Pirmasis – *Shared Memcache* – yra nemokamas ir suteikia talpyklos resursus pagal galimybes, atsižvelgiant į bendrą visų *App Engine* aplikacijų, naudojančių šią paslaugą, apkrovą. Antrasis – *Dedicated Memcache* – yra mokamas ir užtikrina iš anksto nustatytą talpyklos vietą, skirtą tik konkrečiai aplikacijai. [19]

Kadangi *Memcache* yra ribotas tik GAE platformoje, svarbu paminėti, kad GCP siūlo dar vieną alternatyvų talpyklos sprendimą – *Memorystore for Memcached*. Ši paslauga yra visiškai valdoma, mastelį keičianti *Memcached* versija, skirta naudoti su įvairiomis *Google Cloud* platformos paslaugomis. Ji gali būti lengvai integruojama į esamas sistemas, dažniausiai nereikalaujant reikšmingų kodo pakeitimų. Ši paslauga suteikia galimybę kurti visiškai valdomus *Memcached* klasterius, kuriuos galima naudoti duomenų talpyklai įvairioms paskirtims, tokioms kaip dažnai naudojamų duomenų talpinimui, užklausų rezultatų talpinimui, sesijų saugojimui. *Memorystore for Memcached* palaiko automatinio mazgų atradimo paslaugą (angl. *auto discovery*), kuri leidžia dinamiškai aptikti klasterio mazgus ir efektyviai paskirstyti duomenis. Klasterio konfigūracija, įskaitant mazgų skaičių, jų atminties dydį ir procesorių skaičių, gali būti pritaikyta pagal darbo

apkrovos pobūdį ir reikalavimus. Didžiausias palaikomas klasterio dydis siekia 5 TB. Be to, *Memorystore for Memcached* suteikia galimybę valdyti prieigą per *VPC Network*, tačiau autentifikacijos instancijų lygiu nesiūlo. Naudojant *Memorystore*, vartotojai gali sutelkti dėmesį į aplikacijų kūrimą, nes visa talpyklos priežiūra ir mastelio keitimas yra tvarkomas automatiškai. [22]

Kitas komponentas - *Google App Engine Task Queues*. Šis mechanizmas yra svarbus komponentas, skirtas efektyviam ilgai trunkančių užduočių apdorojimui ir atskyrimui nuo vartotojų užklausų apdorojimo [15]. Šis mechanizmas leidžia perkelti ilgalaikius procesus į užduočių eiles, todėl pagrindiniai *front-end* serveriai gali išlikti laisvi ir pasiruošę apdoroti naujas vartotojų užklausas. *Task Queues* mechanizmas apima dvi pagrindines užduočių eilės rūšis: *push queues* ir *pull queues*. *Push queues* skirtos užduočių vykdymui per HTTP užklausas, kurias siunčia *App Engine worker* paslaugos. Šios eilės užtikrina patikimą užduočių vykdymą, reguliuojant užduočių siuntimo tempą ir *worker'ių* mastelio valdymą. *Pull queues* suteikia lankstesnę užduočių valdymo galimybę, leidžiančią *worker'iams* savarankiškai „išimti" užduotis iš eilės, suteikiant daugiau kontrolės, bet reikalaujant papildomo proceso valdymo. Visos užduotys vykdomos asinchroniniu būdu – aplikacija, kuri sukuria užduotį, ją perduoda į eilę ir nelaukia užduoties baigties ar sėkmės. Jei *worker'iui* nepavyksta apdoroti užduoties, *Task Queue* suteikia galimybę atlikti pakartojimą, leidžiantį bandyti apdoroti užduotį dar keletą kartų. *Task Queues* naudojamos įvairiose situacijose, pavyzdžiui, socialinių tinklų žinučių sistemose, kur kiekviena žinutė reikalauja laiko apdorojimui, kaip sekėjų atnaujinimui. Be to, *pull queues* gali būti naudojamos užduočių grupavimui pagal žymeklius (angl. *tags*), pavyzdžiui, tvarkant lyderių lenteles žaidimuose, kur *worker* gali apdoroti susijusias užduotis pagal tą patį žymeklį. [86]

Vienas iš GAE komponentų yra *Cloud Storage*. Tai saugojimo sprendimas, skirtas saugoti failams, tokiems kaip vaizdo įrašai, nuotraukos, ar kitiems statiniams turinio elementams. Šis saugojimo sprendimas rekomenduojamas tais atvejais, kai reikia saugoti arba pateikti failus, kuriuos programa naudoja vykdymo metu. Kai sukuriama *App Engine* programa, sistema automatiškai sukuria numatytąją talpyklą (angl. *bucket*), kuri suteikia 5 GB nemokamos saugyklos vietos, taip pat į šią talpą įeina nemokamas I/O operacijų kvotų limitas. Tokiu būdu, programa gali efektyviai naudotis saugojimo paslaugomis tol, kol neviršijami nemokami resursų limitai. [87]

Į GAE architektūros komponentų sudėtį įeina ir *Cloud SQL*. *Cloud SQL* yra *Google Cloud* siūloma visiškai valdoma reliacinė duomenų bazės paslauga, palaikanti *MySQL, PostgreSQL* ir *SQL Server* sistemas. Ši paslauga užtikrina aukštą duomenų nuoseklumą. Be to, *Cloud SQL* suteikia lankstų duomenų bazės administravimą, leidžiantį užtikrinti patikimą našumą ir prieinamumą, ypač sudėtingesnėms darbo apkrovoms. Paslauga automatizuoja daugelį su duomenų baze susijusių procesų, įskaitant atsarginių kopijų kūrimą, replikavimą, šifravimą ir automatinį talpos padidinimą.

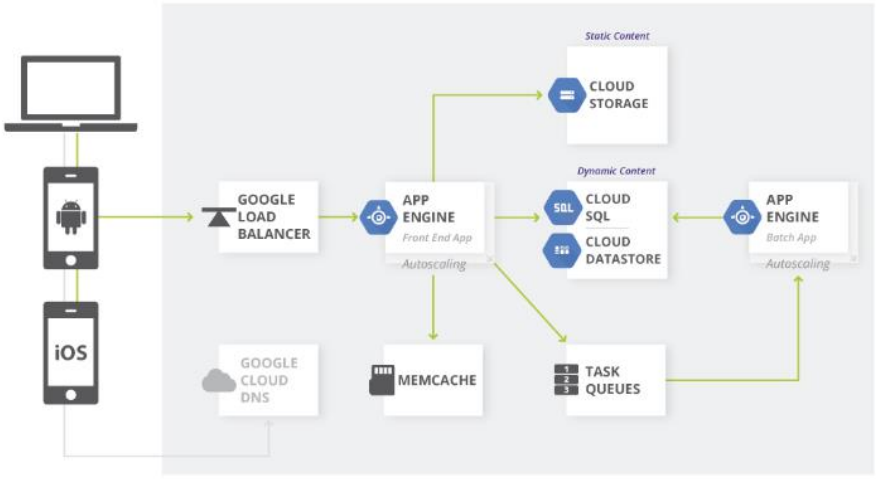
[23] Tai sumažina administravimo sudėtingumą ir leidžia įmonėms susitelkti į pagrindines verslo veiklas, užtikrinant sklandų duomenų bazės veikimą. *Cloud SQL* taip pat suteikia aukštą sistemos patikimumą, nes galima lengvai konfigūruoti aukšto prieinamumo nustatymus, įskaitant automatinį persijungimą tarp zonų, kad būtų apsaugota nuo galimų gedimų [23].

Cloud Datastore yra *NoSQL* dokumentų duomenų bazė, skirta automatiškam mastelio keitimui, aukštam našumui ir lengvam programų kūrimui. Ji palaiko atomines transakcijas, kurios užtikrina, kad visos operacijos būtų sėkmingai įvykdytos, arba, jei bent viena nepavyksta, nevyksta nė viena. Užtikrinamas duomenų skaitymas ir rašymas, nes *Datastore* veikia *Google* duomenų centruose, kuriuose naudojama redundancija, siekiant sumažinti gedimų poveikį. *Datastore* mastelio keitimas yra automatiškai valdomas, todėl užtikrinamas našumas nepriklausomai nuo duomenų bazės dydžio – užklausos pritaikomos pagal rezultatų, o ne visos duomenų bazės dydį. Be to, *Datastore* leidžia lankstų duomenų saugojimą ir užklausų vykdymą, nes ji susiejama su objektinėmis programavimo kalbomis ir palaiko SQL tipo užklausas. Taip pat užtikrinamas stiprus nuoseklumas visoms užklausoms, o duomenys automatiškai šifruojami prieš juos įrašant į diską, o tai suteikia papildomo saugumo. Ši duomenų bazė yra tinkamas sprendimas programoms, kurios reikalauja didelio prieinamumo ir lankstumo duomenų struktūroje, pvz., produktų katalogams, vartotojų profiliams ar transakcijoms, paremtoms ACID savybėmis. [24]

Į GAE architektūros komponentų sudėtį įeina ir *Batch* aplikacijos, kurios naudodamos *Task Queues*, leidžia efektyviai paskirstyti užduotis per kelias instancijas, pasiekiant optimizuotą našumą ir automatinį mastelio keitimą. Automatinis mastelio keitimo režimas dinamiškai pritaiko instancijų skaičių pagal užklausų apimtį, todėl papildomos instancijos sukuriamos tik tada, kai to reikia, o sumažėjus apkrovai, instancijų skaičius sumažinamas, taip optimizuojant resursų naudojimą ir mažinant kaštus. *Task Queues* užtikrina, kad užduotys būtų paskirstytos ir apdorotos efektyviai, o papildomos instancijos kuriamos tik tada, kai užklausų eilė tampa ilga. Šis automatinis mastelio keitimas leidžia užtikrinti žemą atsako vėlinimo laiką ir aukštą našumą, nepriklausomai nuo užduočių apimties. [88]

2 pav. matomas GAE platformos architektūros veikimo principas, iliustruojantis pagrindines sistemos dalis (buvo aptartos anksčiau) ir jų sąveikas. Vartotojų užklausos patenka į sistemą iš įvairių įrenginių (kompiuterių, *Android* ar *iOS* įrenginių), kur jos pirmiausia yra nukreipiamos į *Google Load Balancer* – apkrovos balansavimo mechanizmą. Šis balansavimo sluoksnis paskirsto užklausas tarp skirtingų *App Engine front-end* instancijų, užtikrindamas, kad sistemos apkrova būtų tolygiai paskirstyta .*App Engine front-end* dalis apdoroja vartotojų užklausas, dažnai naudodamasi *Memcache*, kad greitai pateiktų atsakymus iš talpyklos. Jei užklausos reikalauja ilgo apdorojimo laiko, jos perduodamos į *Task Queues* – užduočių eilę, kurioje laukiančios užduotys vėliau vykdomos

App Engine Batch instancijose. Platforma taip pat turi duomenų saugyklos sprendimus, kurie yra suskirstyti į statinę ir dinaminę saugyklas. *Cloud Storage* saugo statinį turinį, pavyzdžiui, failus ar kitus neredaguojamus duomenis. Dinaminiai duomenys, tokie kaip vartotojų informacija, yra saugomi *Cloud SQL* arba *Cloud Datastore* duomenų bazėse. Ši duomenų bazės infrastruktūra leidžia lengvai apdoroti tiek reliacinius, tiek nestruktūruotus duomenis, priklausomai nuo programos poreikių . Šis paveikslas iliustruoja visą GAE veikimo schemą, pradedant nuo užklausos pateikimo, apdorojimo ir baigiant atsakymo pateikimu vartotojui.



Šaltinis: Borges, M., Barros, E., Maia, P. H. (2018). *„Cloud restriction solver: A refactoring-based approach to migrate applications to the cloud*". Information and Software Technology, 95, 346-365. Prieiga per DOI: 10.1016/j.infsof.2017.11.014

2 pav. *Google App Engine* architektūra

5 PRIEDAS

GAE aplinkų palyginimas

Kaip matyti iš lentelės (žr. Lentelė 1), standartinė aplinka yra tinkamesnė mažesniems projektams, kuriuose užtenka iš anksto nustatytų kalbų versijų. Priešingai, lanksčioji aplinka suteikia daugiau galimybių, ypač kai reikia naudoti nestandartines kalbas arba pritaikyti programos aplinką su *Docker* atvaizdais, todėl ji yra labiau tinkama didesniems ir sudėtingesniems projektams.

Lentelė 1

Aplinkų palyginimas

Funkcija	Standartinė aplinka	Lanksčioji aplinka
Palaikomos kalbos	*Go, Java, Node.js, PHP, Python, Ruby*	*Go, Java, Node.js, PHP, Python, Ruby, .NET,* kitos kalbos naudojant *Docker*
Docker palaikymas	Ne	Taip
Kalbų versijos	Fiksuotos versijos	Lankstesnis versijų pasirinkimas

Šaltinis: sudaryta autoriaus remiantis: [26] [27] [28] [113] [29] [30] [114] [31] [106] [107] [108] [109] [110] [111]

Be to, verta atkreipti dėmesį į 2018 metais publikuotą Zheng Li ir Xue Guo tyrimą, kuriame buvo analizuojama, kaip skirtingos GAE vykdymo aplinkos ir jų geografinis išsidėstymas daro įtaką aplikacijų veikimo našumui. Eksperimentai buvo vykdomi naudojant GAE standartinę aplinką, kuri siūlo nemokamas kvotas ir leidžia išsamiai išnagrinėti našumo skirtumus tarp skirtingų vykdymo aplinkų ir regionų. [12]

Tyrime naudota *DoKnowMe* metodika, kuri struktūruotai analizuoja vykdymo aplinkų našumą pradedant reikalavimų identifikavimu ir baigiant išvadų formulavimu. Ši metodologija buvo pasirinkta dėl gebėjimo struktūriškai ir tvirtai įvertinti skirtingų vykdymo aplinkų našumą. Eksperimentų metu buvo naudojamas *Fibonacci* skaičių generavimas, tiek rekursiniu, tiek nerekursiniu būdu, siekiant išmatuoti vykdymo laiką milisekundėmis ir palyginti skirtingų aplinkų našumą. Eksperimentuose buvo analizuojamos šios aplinkos: *Java* (7 versija), *Java* (8 versija), *Python* (2.7 versija), *Go* (1.8 versija), *PHP* (5.5 versija). [12]

Lentelė 2 atspindi vykdymo aplinkų palyginimą atliekant rekursyvius *string* sujungimus. Lentelėje nurodyta, kiek vidutiniškai užtrunka atlikti 35-ą *Fibonacci string* sujungimą ir koks yra kiekvienos aplinkos atminties perpildymo indeksas [12]:

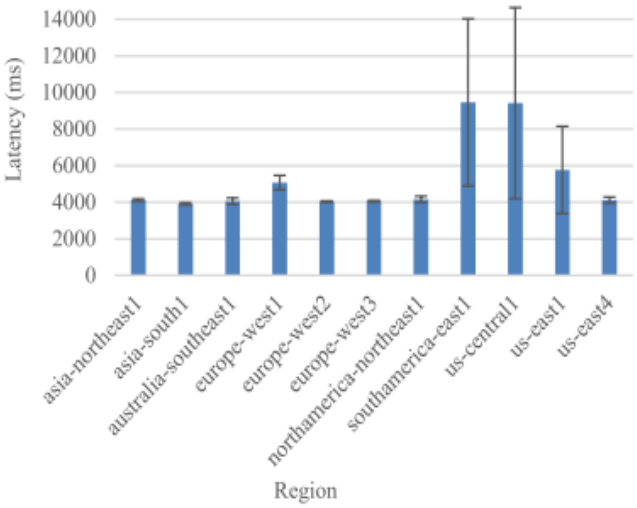
Lentelė 2

Vykdymo aplinkų palyginimas GAE standartinėje aplinkoje (rekursyvus *String* sujungimas)

Vykdymo aplinka	Vidutinis delsos laikas (35-as)	Atminties perpildymo indeksas
Java 7	1479.0 ms	37-as
Java 8	990.4 ms	37-as
Python	8833.7 ms	40-as
Go	805.5 ms	44-as
PHP	10058.1 ms	45-as

Šaltinis: sudaryta autoriaus remiantis [12].

Analizė atskleidė, kad vykdymo aplinkos ne tik skiriasi pagal vykdymo laiką, bet ir pagal atminties efektyvumą. Pavyzdžiui, *Go* aplinka rodė geriausią našumą pagal vykdymo laiką (805,5 ms) ir didžiausią atminties perpildymo indeksą (44). Tai rodo geriausią balansą tarp skaičiavimo greičio ir atminties valdymo. Tuo tarpu PHP aplinka užfiksavo prasčiausius rezultatus, pasiekdama ilgiausią vykdymo laiką (10058,1 ms) ir aukščiausią atminties perpildymo indeksą (45), o tai rodo, kad ji nėra tinkama skaičiavimo intensyvioms užduotims. Šiame tyrime taip pat atskleista, kad skirtingų GAE regionų našumas taip pat yra nevienodas. Pavyzdžiui, pietų Amerikos regionas (*southamerica-east1*) ir JAV centrinis regionas (*us-central1*) pasižymėjo žymiai prastesniu našumu nei kiti regionai (1 pav.). Tai leidžia manyti, kad šiuose regionuose yra naudojama mažiau efektyvi infrastruktūra, dėl kurios šių regionų veikimo našumas yra žymiai prastesni. Dėl to šie regionai gali būti netinkami sistemoms, kurios reikalauja aukšto spartos. [12]



Šaltinis: [12].

1 pav. Našumo lyginimas tarp Java 8 vykdymo aplinkų, skaičiuojant rekursyviai 45-ąjį ilgojo tipo Fibonacci skaičių 11 skirtinguose regionuose (klaidų juostos rodo standartinius nuokrypius)

Remiantis tyrimo rezultatais, galima suformuluoti rekomendacijas dėl vykdymo aplinkos pasirinkimo. *Go* aplinka yra rekomenduojama projektams, kuriuose reikalingas optimalus balansavimas tarp greito vykdymo ir efektyvios atminties alokacijos. *Java 8* taip pat pasirodė kaip tinkamas pasirinkimas ypač resursų reikalaujančioms užduotims, tačiau norint išvengti atminties perpildymo problemų, gali prireikti optimizuoti aplinkos kintamuosius. Tuo tarpu PHP vykdymo aplinka nėra tinkama intensyvioms skaičiavimo užduotims, todėl jos naudojimą vertėtų riboti projektuose, kuriuose pagrindinis dėmesys skiriamas ne skaičiavimo efektyvumui, o naudoti kitoms funkcijoms, pvz., žiniatinklio svetainių kūrimui. Kalbant apie regionų pasirinkimą, Europos ir Azijos regionai pasižymėjo aukštesniu našumu, todėl svarbu apsvarstyti šių regionų naudojimą, ypač kai kalbama apie sistemas, kurioms reikia greito atsako laiko ir efektyvaus išteklių naudojimo. [12]

Toliau pateiktoje lentelėje (žr. Lentelė 3) lyginami infrastruktūros valdymo aspektai GAE standartinėje ir lanksčiojoje aplinkose. Šie skirtumai turi didelę reikšmę projektuojant sistemas, ypač kai reikia pasirinkti tarp didesnės kontrolės arba automatizuoto valdymo.

Lentelė 3

Infrastruktūros skirtumai

Funkcija	Standartinė aplinka	Lanksčioji aplinka
Serverio prieiga	Ne, smėlio dėžės aplinka	Taip, prieiga per SSH prie VM
Išteklių didinimas (angl. *scaling*)	Rankinis, Basic, automatinis	Rankinis, automatinis, su VM rezervacijomis
Konteineriai	Valdomi *Google*	Pritaikomi su *Docker*
Atminties ir CPU dydis	Fiksuotas, pagal instancijų klases	Lankstus, konfigūruojamas

Šaltinis: sudaryta autoriaus remiantis šaltinais: [26] [27] [28] [113] [29] [30] [114] [31] [106] [107] [108] [109] [110] [111] [105] [32].

Lentelėje matome, kad standartinė aplinka daugiausiai dėmesio skiria automatizuotam valdymui ir paprastumui, todėl ši aplinka tinka tiems kūrėjams, kurie nori sumažinti infrastruktūros priežiūros sudėtingumą. Kita vertus, lanksčioji aplinka suteikia daugiau galimybių pritaikyti sistemą pagal specifinius poreikius, pavyzdžiui, leidžia valdyti VM ir naudoti pasirinktinius *Docker* konteinerius. Šie skirtumai ypač aktualūs kūrėjams, kuriems reikia didesnio lankstumo ir kontrolės savo projektuose.

Klaida! Nerastas nuorodos šaltinis. Lentelė 4 palygina saugumo ir priežiūros aspektus standartinėje ir lanksčiojoje aplinkose. Nors abi aplinkos užtikrina aukštą saugumo lygį, jos skiriasi galimybėmis valdyti OS atnaujinimus ir SSH prieigą, o tai gali turėti reikšmės projekto saugumo reikalavimams.

Lentelė 4

Saugumo ir priežiūros aspektai

Funkcija	Standartinė aplinka	Lanksčioji aplinka
OS atnaujinimai	Automatiniai	Galimybė valdyti OS atnaujinimus
SSH prieiga	Ne	Taip, galimybė įjungti/ išjungti SSH
Prieiga prie failų sistemos	Ribota prieiga tik į /tmp aplanką	Visa prieiga prie VM failų sistemos

Šaltinis: sudaryta autoriaus remiantis: [26] [27] [28] [113] [29] [30] [114] [31] [106] [107] [108] [109] [110] [111]

Lentelėje pateikti duomenys parodo, kad lanksčioji aplinka suteikia daugiau kontrolės OS ir SSH prieigai, o tai yra svarbu tiems, kurie nori individualiai valdyti sistemos atnaujinimus ir apsaugą. Tuo tarpu standartinė aplinka yra labiau standartizuota ir automatiškai prižiūrima *Google* su minimaliu kūrėjų įsikišimu. Todėl kūrėjai neturi rūpintis saugumo atnaujinimais, tačiau turi mažiau galimybių prisitaikyti prie specifinių saugumo poreikių.

Toliau pateikta lentelė (Lentelė 5) lygina GAE standartinės ir lanksčiosios aplinkų kainodaros ir kvotų sistemas. Šie skirtumai yra esminiai renkantis tarp dviejų aplinkų, ypač atsižvelgiant į projekto biudžetą ir reikalingų išteklių dydį. Standartinė aplinka siūlo tam tikras nemokamas kvotas ir ribotus fiksuotus išteklius, o lanksčioji aplinka visiškai mokama ir suteikia daugiau lankstumo pritaikant resursus.

Lentelė 5

Kainodara ir kvotos

Funkcija	Standartinė aplinka	Lanksčioji aplinka
Nemokama kvota	Taip – 1 GB duomenų saugojimui ir eismui	Ne, mokama už visus resursus
Kainodaros modelis	Mokama už resursus viršijus kvotas	Mokama už naudojamus CPU, atminties ir kitus resursus be nemokamos kvotos
Resursų valdymo kaštai	Mažesni, dėl riboto resursų naudojimo	Didesni, dėl lankstesnių resursų poreikių

Šaltinis: sudaryta autoriaus remiantis: [26] [27] [28] [113] [29] [30] [114] [31] [106] [107] [108] [109] [110] [111]

Iš lentelės matyti, kad standartinė aplinka siūlo ekonomiškesnį sprendimą mažesniems projektams, ypač tiems, kurie gali pasinaudoti nemokamomis kvotomis. Ši aplinka yra tinkama projektams, kuriuose nereikia didelių išteklių, ir kai svarbu išlaikyti mažus kaštus. Priešingai, lanksčioji aplinka suteikia daugiau galimybių pritaikyti resursų naudojimą pagal projekto poreikius, tačiau ji yra visiškai mokama ir skirta projektams, kurie reikalauja daugiau išteklių ir kontrolės. Tai

reiškia, kad didesni projektai gali patirti didesnes išlaidas, tačiau šios išlaidos yra labiau prognozuojamos dėl lankstesnių resursų valdymo galimybių.

Atlikta GAE aplinkų analizė leidžia padaryti kelias esmines išvadas. Pirma, Standartinė aplinka yra optimali mažesniems projektams, kuriems nereikia didelės lankstumo ar kontrolės infrastruktūrai. Jos privalumas slypi paprastume ir automatizuotoje infrastruktūros priežiūroje, o tai leidžia kūrėjams sutelkti dėmesį tik į programinės įrangos kūrimą, nereikšmingai rūpinantis valdymu. Ši aplinka ypač tinkama mažo masto projektams arba tiems, kurie nori pasinaudoti nemokamomis kvotomis ir išvengti didelių išlaidų.

Antra, lanksčioji aplinka labiau tinka didesniems ir sudėtingesniems projektams, kuriems reikia didesnės kontrolės ir pritaikomumo. Lankstesnės *Docker* technologijos, galimybė naudotis SSH prieiga ir pilna virtualios mašinos kontrolė suteikia galimybę pritaikyti aplinką pagal individualius projekto poreikius. Be to, ši aplinka yra naudinga tais atvejais, kai reikia naudoti įvairias programavimo kalbas ir palaikyti nestandartines kalbų versijas.

Todėl pasirinkimas tarp šių dviejų aplinkų turėtų būti grindžiamas ne tik dabartiniais projekto poreikiais, bet ir ilgalaike plėtros perspektyva. Mažesniems, ekonomiškai efektyviems projektams rekomenduojama standartinė aplinka, tuo tarpu projektams, kuriems reikia lankstumo, didesnės kontrolės ir specialių konfigūracijų, lanksčioji aplinka yra optimalus pasirinkimas.

6 PRIEDAS

Saugos sprendimai naudojant GAE

Atlikta šaltinių analizė parodė, jog GAE aplinką galima pritaikyti spręsti tokioms saugumo problemoms kaip *data stomping*. *Data stomping* – tai situacija, kai kelios užklausos vienu metu bando modifikuoti tą patį duomenų objektą, dėl ko vieno proceso atlikti pakeitimai gali būti netyčia perrašyti kito proceso pakeitimais. Siekdama spręsti šią problemą, *Khan Academy* sukūrė modulius *db_hooks.py* ir *txn_safety.py*, kurie leidžia stebėti ir registruoti *GET* ir *PUT* operacijas, užtikrinant, kad duomenų modifikavimo procesai būtų vykdomi saugiai ir nuosekliai. Šie moduliai padeda aptikti galimas klaidas realiuoju laiku ir apsaugo nuo duomenų praradimo bei saugumo pažeidimų. [45]

Norint, kad šios saugumo priemonės veiktų tinkamai, būtina vadovautis *GET - MODIFY - PUT* modeliu, kuriame duomenys pirmiausia nuskaitomi (*GET*), tada modifikuojami (*MODIFY*) ir tik tuomet išsaugomi (*PUT*). Labai svarbu, kad *GET* būtų atliekamas transakcijos ar užrakto ribose, kitaip gali įvykti *data stomping*. Dažna klaida – *GET* vykdymas už transakcijos ribų, o tai kelia riziką duomenų nesuderinamumui. [45]

Db_hooks.py modulis veikia kaip *hook* sistema, leidžianti registruoti *GET* ir *PUT* operacijas bei užfiksuoti nesuderinamas operacijas, kai jos vykdomos už transakcijos ar užrakto ribų. Ši sistema padeda užkirsti kelią *data stomping* rizikai, kai kelios užklausos vienu metu bando modifikuoti tą patį duomenų objektą. [45]

Txn_safety.py modulis sukurtas aptikti ir įspėti apie saugumo pažeidimus, kai *GET-MODIFY-PUT* operacijos atliekamos neapsaugotai. Šis modulis naudoja specialius dekoratorius, kurie nurodo, kaip konkretūs duomenų modeliai turi būti apsaugoti. Pavyzdžiui, *@written_in_transaction_model()* dekoratorius nurodo, kad *GET-MODIFY-PUT* operacijos turi būti vykdomos tik transakcijos ribose, o *@written_with_user_lock_model()* leidžia užrakinti konkrečią naudotojo sesiją, ribojant prieigą prie duomenų objektų tik vienai užklausai vienu metu. [45]

Šie moduliai ne tik padeda realiuoju laiku aptikti galimas klaidas, bet ir automatiškai generuoja klaidos pranešimus su išsamia atsekamumo informacija. Pavyzdžiui, jei *GET* arba *PUT* operacijos atliekamos neapsaugotai, sistema pateikia klaidos pranešimą, leidžiantį programuotojams tiksliai nustatyti, kur įvyko pažeidimas, ir jį ištaisyti. Tai žymiai palengvina nesuderinamumo šalinimo procesą ir užtikrina, kad duomenų operacijos vyktų pagal nustatytas saugumo taisykles. [45]

Apibendrinant, galima teigti, kad *Khan Academy* sukurti *db_hooks.py* ir *txn_safety.py* moduliai, pritaikyti GAE aplinkai, padidina sistemos patikimumą, nes suteikia galimybę realiuoju

laiku identifikuoti nesuderinamus veiksmus ir automatiškai generuoti klaidų pranešimus su išsamia atsekamumo informacija. Tokie sprendimai ne tik mažina duomenų praradimo riziką, bet ir optimizuoja klaidų šalinimo procesą, stiprindami sistemos patikimumą ir saugumą sudėtingose aplinkose.

7 PRIEDAS

Kombinuotas GAE programavimo sprendimas *ScaLib* sistemoje

Didėjant vartotojų poreikiams ir duomenų srautams, informacinėms sistemoms reikalingi efektyvūs sprendimai, leidžiantys valdyti didelius duomenų kiekius, užtikrinti greitą prieigą bei aukštą duomenų saugumo ir privatumo lygį. Vienas iš tokių sprendimų yra hibridinės debesijos infrastruktūros naudojimas, derinant debesijos paslaugų teikėjų galimybes su vietinėmis sistemomis.

ScaLib yra išplečiama bibliotekos sistema, sukurta naudojant hibridinę debesijos infrastruktūrą, kurioje GAE integruojama su vietine (angl. *self-hosted*) vartotojų duomenų valdymo sistema. Šis sprendimas leidžia optimaliai išnaudoti GAE debesijos teikiamas galimybes, kartu užtikrinant aukštą duomenų saugumo ir privatumo lygį, nes jautrūs vartotojų duomenys saugomi vietinėje sistemoje. Pagrindinis iššūkis kuriant *ScaLib* buvo užtikrinti vartotojų duomenų privatumą; dėl šios priežasties jautrūs duomenys laikomi už GAE infrastruktūros ribų, o duomenų sujungimai atliekami kliento pusėje (angl. *client-side joins*). Vienas esminių *ScaLib* sprendimų yra jautrių vartotojų duomenų apsauga naudojant kliento pusės sujungimus. GAE tvarko tik anoniminius vartotojo identifikatorius ir kitus viešai prieinamus duomenis, o tikrieji vartotojų duomenys, tokie kaip vardai ir slaptažodžiai, saugomi lokaliai, už organizacijos užkardos. Norėdamas pasiekti šiuos duomenis, klientas naudoja *JavaScript* ir *AJAX* technologijas, atlikdamas duomenų sujungimus tiesiogiai kliento pusėje. Tai užtikrina, kad jautrūs vartotojų duomenys niekada nebus perduoti per GAE infrastruktūrą, taip sumažinant saugumo pažeidimų riziką. Šis sprendimas leidžia efektyviai tvarkyti duomenis naudojant kliento pusės operacijas. Pavyzdžiui, kai klientas pasiekia tinklalapį, kuriame yra vartotojo ID, JavaScript identifikuoja šiuos ID ir siunčia užklausas vietinei duomenų valdymo sistemai, kuri pateikia reikiamą informaciją. Taip jautrūs duomenys apdorojami tik lokaliai, o ne per GAE infrastruktūrą. [46]

GAE suteikia įvairias funkcijas, leidžiančias kurti išplečiamas ir efektyvias sistemas. *ScaLib* išnaudoja GAE privalumus, tokius kaip apkrovos balansavimas, *Google Cloud Datastore* ir *TaskQueue* paslaugos. Šie funkcionalumai prisideda prie sistemos našumo ir išplečiamumo, tačiau vartotojų duomenų privatumas užtikrinamas per kliento pusės sujungimus ir vietinį duomenų saugojimą. GAE integruotas apkrovos balansavimas yra esminė *ScaLib* sistemos dalis. Apkrovos balansavimas leidžia automatiškai paskirstyti užklausas tarp serverių, užtikrinant efektyvų sistemos veikimą net esant didelėms apkrovoms. Ši paslauga svarbi siekiant užtikrinti, kad vartotojų užklausos būtų apdorojamos greitai ir be trikdžių. Be to, GAE siūlo patikimą apsaugą nuo DDoS atakų, nes užklausos paskirstomos per kelis serverius, taip sumažinant vieno serverio perkrovimo riziką ir didinant sistemos stabilumą. *ScaLib* naudoja GAE *Google Cloud Datastore*, kuris užtikrina duomenų

saugojimą ir efektyvų užklausų vykdymą. *Datastore* siūlo *entity group* konceptą, leidžiantį efektyviai organizuoti duomenis, kurie logiškai priklauso vienai grupei. Kiekvienas *ScaLib* vartotojas turi savo *entity group*, kurioje saugomi jo rezervuoti ir paskolinti knygų egzemplioriai. Nors *entity group* leidžia greitai ir efektyviai atlikti skaitymo ir rašymo operacijas, sistema turi tam tikrų apribojimų. Pavyzdžiui, vienam vartotojui galima atlikti tik vieną rašymo operaciją per sekundę, o duomenų kiekis *entity group* ribojamas maždaug 1 MB dydžiu. Nepaisant šių apribojimų, šis sprendimas padeda užtikrinti sistemos efektyvumą ir išplečiamumą. [46]

GAE taip pat siūlo *Blobstore* – saugojimo paslaugą, skirtą binarinių failų, tokių kaip vaizdai, valdymui. *ScaLib* sistema naudoja *Blobstore* knygų viršelių ir autorių nuotraukų saugojimui. Įkeliant vaizdus į sistemą, jie saugomi *Blobstore*, o jų URL generuojamas naudojant *Google Image Service*. Ši paslauga leidžia keisti vaizdų dydį pagal vartotojo poreikius, pateikiant tiek miniatiūras, tiek pilnos raiškos vaizdus. GAE *TaskQueue* paslauga *ScaLib* sistemoje naudojama laikui jautrių operacijų, tokių kaip knygų rezervacijų galiojimo valdymas, įgyvendinimui. Kiekvieną kartą, kai vartotojas rezervuoja knygą, *TaskQueue* sukuria užduotį, kuri vykdoma rezervacijos galiojimo pabaigoje. Jei rezervacija baigiasi ir knyga nėra atšaukta, užduotis automatiškai pašalina rezervaciją iš sistemos. Šis sprendimas užtikrina, kad *ScaLib* sistema nepriklausytų nuo vartotojų veiksmų ir galėtų automatiškai tvarkyti pasibaigusias rezervacijas. *TaskQueue* taip pat naudoja ETA (angl. *Estimated Time of Arrival*) funkciją, leidžiančią užduotį vykdyti tiksliai numatytu laiku. Tai užtikrina, kad kiekviena užduotis būtų vykdoma laiku ir nesukeltų papildomos sistemos apkrovos. [46]

ScaLib programavimo sprendimuose naudojamas duomenų dublikavimas (angl. *duplicity*) siekiant sumažinti užklausų vykdymo laiką. Užuot kiekvieną kartą atlikus kelias užklausas (angl. *fetch operations*) norint gauti knygų autorių duomenis ir susijusių knygų sąrašus, *ScaLib* dubliuoja reikalingą informaciją autorių įrašuose. Tai leidžia išvengti užklausų vėlavimo ir pagerina sistemos našumą. Nors duomenų dublikavimas paprastai vengiamas tradicinėse SQL duomenų bazėse, *NoSQL* architektūrose, tokiose kaip *Google Cloud Datastore*, tai laikoma efektyviu būdu optimizuoti duomenų prieigą. [46]

Apibendrinant, *ScaLib* sistemos programavimo sprendimas, paremtas GAE, leidžia efektyviai sujungti debesijos teikiamas galimybes su vietiniu duomenų valdymu. GAE užtikrina automatinį apkrovos balansavimą, efektyvų duomenų saugojimą per *Google Cloud Datastore* bei laikui jautrių užduočių valdymą per *TaskQueue*. Kartu su vietinės sistemos sprendimais ir kliento pusės sujungimais, šis sprendimas suteikia aukštą našumą, duomenų saugumą ir sistemos išplečiamumą, atitinkantį šiuolaikinius vartotojų poreikius ir duomenų srautų apimtis.

8 PRIEDAS

Svetainių specifikacija

Projektas: internetinė naujienų agregatoriaus svetainė.

Projektuojamas objektas: internetinė naujienų agregatoriaus svetainė, skirta atlikti eksperimentinį tyrimą, naudojant *Memcached* technologiją ir papildomus saugumo mechanizmus.

Projektuojamo objekto paskirtis: įvertinti duomenų tvarkymo našumą, lyginant kontrolines ir eksperimentines aplikacijų versijas.

1. Bendra svetainių specifikacija:
    1.1. Pagrindinis svetainių funkcionalumas:
        1.1.1. Naujienų atvaizdavimas;
            1.1.1.1. Svetainė atvaizduoja naujienas, gaunamas iš RSS srauto.
                1.1.1.1.1. Naujienų elementai:
                    1.1.1.1.1.1. Naujienos pavadinimą;
                    1.1.1.1.1.2. Naujienos santrauką (trumpą aprašymą);
                    1.1.1.1.1.3. Publikavimo datą;
                    1.1.1.1.1.4. Nuorodą į pilną naujieną;
                    1.1.1.1.1.5. Paveikslėlį, jei jis yra prieinamas RSS sraute.
        1.1.2. Naujienų talpinimas į *Memcached*:
            1.1.2.1. Svetainė talpina naujienas į *Memcached* trumpalaikiam saugojimui.
            1.1.2.2. Talpyklos galiojimo laikas nustatytas 300 sekundžių (5 minutėms).
            1.1.2.3. Jei duomenys yra *Memcached*, jie atvaizduojami iš talpyklos.
            1.1.2.4. Jei duomenų nėra *Memcached*, jie gaunami iš RSS srauto ir iškart išsaugomi talpykloje.
        1.1.3. Slaptų duomenų valdymas:
            1.1.3.1. *Memcached* serverio IP adresas gaunamas iš *Google Cloud Secret Manager*.
2. Specifikacija aplikacijų versijoms
    2.1. *Control-app* (kontrolinė versija, naudojanti *Memcached* be papildomų saugumo mechanizmų. Ši versija neturi vartotojų autentifikacijos funkcionalumo.):
        2.1.1. Funkcionalumas:
            2.1.1.1. Pagrindinis svetainių funkcionalumas (pateikta 1 punkte).

2.1.2. Vartotojo specifikacija::

2.1.2.1. Visi vartotojai gali peržiūrėti naujienas.

2.2. *Control-app-v2* (kontrolinė versija, naudojanti *Memcached*, įgyvendinanti simuliuotą vartotojų autentifikacijos procesą be realaus autentifikacijos mechanizmo):

2.2.1. Funkcionalumas:

2.2.1.1. Pagrindinis svetainių funkcionalumas (pateikta 1 punkte);

2.2.1.2. Simuliuotas registracijos procesas:

2.2.1.2.1. Sistema nurodo pateikti el. pašto adresą, slaptažodį ir prisijungimo vardą.

2.2.1.2.1.1. Sistema tikrina, ar el. paštas turi „@“ simbolį.

2.2.1.2.1.1.1. Jei el. paštas neturi „@“ simbolio, rodomas klaidos pranešimas;

2.2.1.2.1.1.2. Jei el. paštas turi „@“ simbolį, sistema nukreipia vartotoją į prisijungimo langą;

2.2.1.2.1.1.2.1. Rodomas pranešimas apie sėkmingą registraciją;

2.2.1.3. Simuliuotas prisijungimo procesas;

2.2.1.3.1. Sistema nurodo pateikti el. pašto adresą ir slaptažodį.

2.2.1.3.1.1. Sistema nukreipia vartotoją į pagrindinį langą su naujienomis:

2.2.1.3.1.1.1. Prisijungusiam vartotojui rodoma personalizuota žinutė (pvz., „Sveiki, Simuliuotas Vartotojas!“).

2.2.1.4. Simuliuotas atsijungimo procesas.

2.2.1.4.1. Sistema nukreipia vartotoją į pagrindinį puslapį, neberodoma personalizuota žinutė.

2.2.2. Vartotojo specifikacija:

2.2.2.1. Visi vartotojai gali peržiūrėti naujienas.

2.2.2.2. Registracija:

2.2.2.2.1. Vartotojas pateikia el. pašto adresą, slaptažodį ir prisijungimo vardą.

2.2.2.2.1.1. Vartotojas pateikia el. pašto adresą be „@“ simbolio ir mato klaidos pranešimą;

2.2.2.2.1.2. Vartotojas pateikia el. paštą su „@“ simboliu ir yra nukreipiamas į prisijungimo langą.

2.2.2.2.1.2.1. Vartotojas mato pranešimą apie sėkmingą registraciją.

2.2.2.3. Prisijungimas:

2.2.2.3.1. Vartotojas pateikia el. pašto adresą ir slaptažodį.

2.2.2.3.1.1.1. Vartotojas nukreipiamas į pagrindinį langą su naujienomis.

2.2.2.3.1.1.1.1. Prisijungęs vartotojas mato personalizuotą sveikinimo žinutę;

2.2.2.3.1.1.1.2. Prisijungęs vartotojas gali peržiūrėti naujienas.

2.2.2.4. Atsijungimas:

2.2.2.4.1. Prisijungęs vartotojas turi galimybę atsijungti nuo paskyros.

2.3. *Encrypt-app* (eksperimentinė versija, pritaikanti duomenų šifravimą prieš juos talpinant į *Memcached*):

2.3.1. Funkcionalumas:

2.3.1.1. Pagrindinis svetainių funkcionalumas (pateikta 1 punkte).

2.3.1.2. Duomenų šifravimas:

2.3.1.2.1. Prieš talpinant naujienas į *Memcached*, duomenys šifruojami naudojant šifravimo algoritmą.

2.3.1.2.2. Ištraukus duomenis iš *Memcached,* jie dešifruojami.

2.3.2. Vartotojo specifikacija:

2.3.2.1. Visi vartotojai gali peržiūrėti naujienas.

2.3.3. Slaptų duomenų valdymas:

2.3.3.1. Šifravimo raktas saugomas ir gaunamas iš *Google Cloud Secret Manager*.

2.4. *Firebase-auth-app* (eksperimentinė versija, naudojanti *Memcached*, įgyvendinanti realų vartotojų autentifikacijos procesą su autentifikacijos mechanizmu):

2.4.1. Funkcionalumas:

2.4.1.1. Pagrindinis svetainių funkcionalumas (pateikta 1 punkte);

2.4.1.2. Visi vartotojai gali peržiūrėti naujienas.

2.4.1.3. Registracijos procesas:

2.4.1.3.1. Sistema nurodo pateikti el. pašto adresą, slaptažodį ir prisijungimo vardą.

2.4.1.3.1.1. Sistema tikrina, ar el. paštas turi el. paštui būdingą struktūrą:

2.4.1.3.1.1.1. Jei el. paštas pateiktas blogos struktūros, rodomas klaidos pranešimas;

2.4.1.3.1.2. Sistema tikrina, ar slaptažodis yra sudarytas iš 6 simbolių:

2.4.1.3.1.2.1. Jei pateiktas slaptažodis neturi 6 simbolių, rodomas klaidos pranešimas;

2.4.1.3.1.3. Jei sistema neaptinka klaidų pateiktuose duomenyse, autentifikacijos mechanizmas duomenų bazėje tikrina, ar nėra registruoto vartotojo su pateiktu el. pašto adresu.

2.4.1.3.1.3.1. Duomenų bazėje jau yra registruotas vartotojas su pateiktu el. pašto adresu:

2.4.1.3.1.3.1.1. Sistema grąžina klaidos pranešimą.

2.4.1.3.1.3.2. Duomenų bazėje nėra registruoto vartotojo su pateiktu el. pašto adresu.

2.4.1.3.1.3.2.1. Į duomenų bazę įrašomas vartotojo pateiktas el pašto adresas ir šifruotas slaptažodis;

2.4.1.3.1.3.2.2. Sistema nukreipia vartotoją į prisijungimo langą;

2.4.1.3.1.3.2.2.1. Rodomas pranešimas apie sėkmingą registraciją;

2.4.1.4. Prisijungimo procesas;

2.4.1.4.1. Sistema nurodo pateikti pateikia el. pašto adresą ir slaptažodį.

2.4.1.4.1.1. Autentifikacijos mechanizmas tikrina ar pateikti duomenys sutampa su duomenų bazėje patalpintais duomenimis.

2.4.1.4.1.1.1. Pateikti duomenys sutampa su duomenų bazėje patalpintais duomenimis:

2.4.1.4.1.1.1.1. Sistema nukreipia vartotoją į pagrindinį langą su naujienomis:

2.4.1.4.1.1.1.1.1. Sistema vartotojui rodo personalizuotą žinutę (pvz., „Sveiki, [vardas]!“).

2.4.1.4.1.1.2. Pateikti duomenys nesutampa su duomenų bazėje patalpintais duomenimis:

2.4.1.4.1.1.2.1. Sistema rodo klaidos pranešimą.

2.4.1.5. Atsijungimo procesas.

2.4.1.5.1. Sistema nukreipia vartotoją į pagrindinį puslapį, neberodoma personalizuota žinutė.

2.4.2. Vartotojo specifikacija:

2.4.2.1. Visi vartotojai gali peržiūrėti naujienas.

2.4.2.2. Registracija:

2.4.2.2.1. Vartotojas pateikia el. pašto adresą, slaptažodį ir prisijungimo vardą.

2.4.2.2.1.1. Vartotojas pateikia el. pašto adresą be el. paštui būdingos struktūros

2.4.2.2.1.1.1. Vartotojas mato klaidos pranešimą;

2.4.2.2.1.2. Vartotojas pateikia slaptažodį su mažiau nei 6 simboliais;

2.4.2.2.1.2.1. Vartotojas mato klaidos pranešimą;

2.4.2.2.1.3. Vartotojas pateikia unikalų el. pašto adresą su el. paštui būdinga struktūra, slaptažodį su daugiau nei 6 simboliais:

2.4.2.2.1.3.1. Vartotojas nukreipiamas į prisijungimo langą su naujienomis;

2.4.2.2.1.3.1.1. Vartotojas mato pranešimą, jog registracija yra sėkminga.

2.4.2.3. Prisijungimas:

2.4.2.3.1. Vartotojas pateikia duomenų bazėje registruotus el. pašto adresą ir slaptažodį.

2.4.2.3.1.1.1. Vartotojas nukreipiamas į pagrindinį langą su naujienomis.

2.4.2.3.1.1.1.1. Prisijungęs vartotojas mato personalizuotą sveikinimo žinutę (pvz., „Sveiki, [vardas]!“).

2.4.2.3.1.1.1.2. Prisijungęs vartotojas gali peržiūrėti naujienas.

2.4.2.3.2. Vartotojas pateikia duomenų bazėje neregistruotus el. pašto adresą ir slaptažodį.

2.4.2.3.2.1. Vartotojas mato klaidos pranešimą.

2.4.2.4. Atsijungimas:

2.4.2.4.1. Prisijungęs vartotojas turi galimybę atsijungti nuo paskyros.

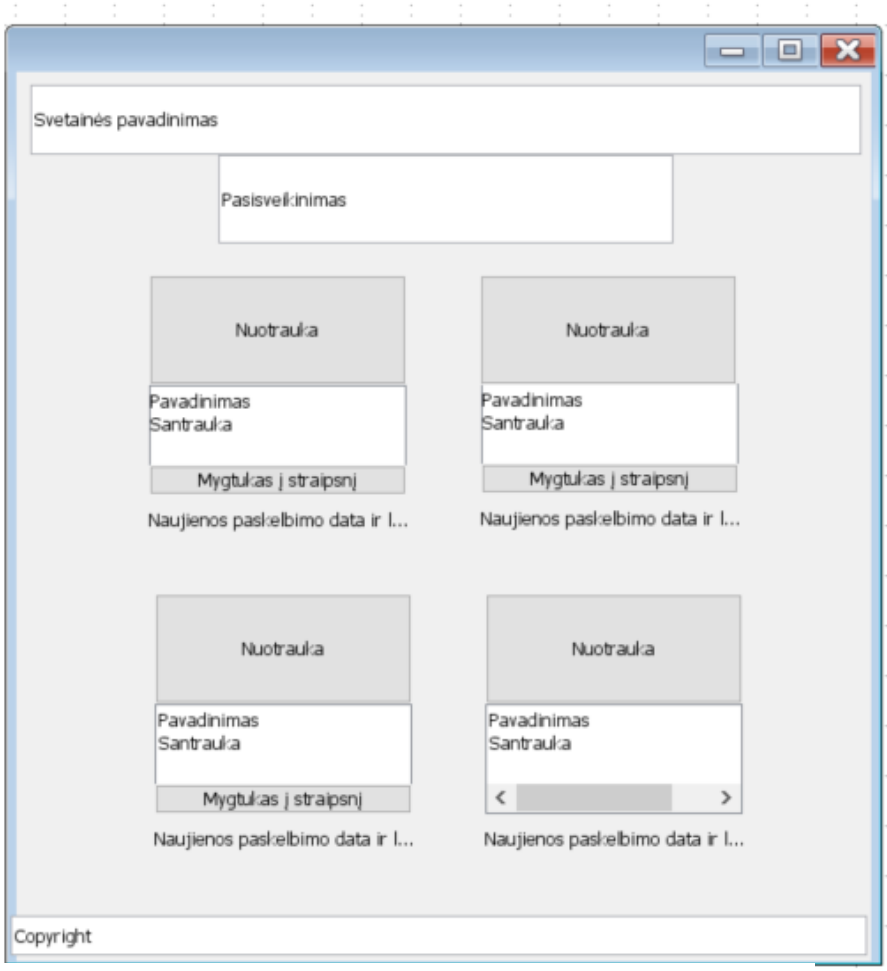
2.4.2.4.1.1. Vartotojas yra nukreipiamas į pagrindinį puslapį, nebemato personalizuotos žinutės.

2.4.3. Slaptų duomenų valdymas:

2.4.3.1. Autentifikacijos raktai gaunami iš *Google Cloud Secret Manager*.

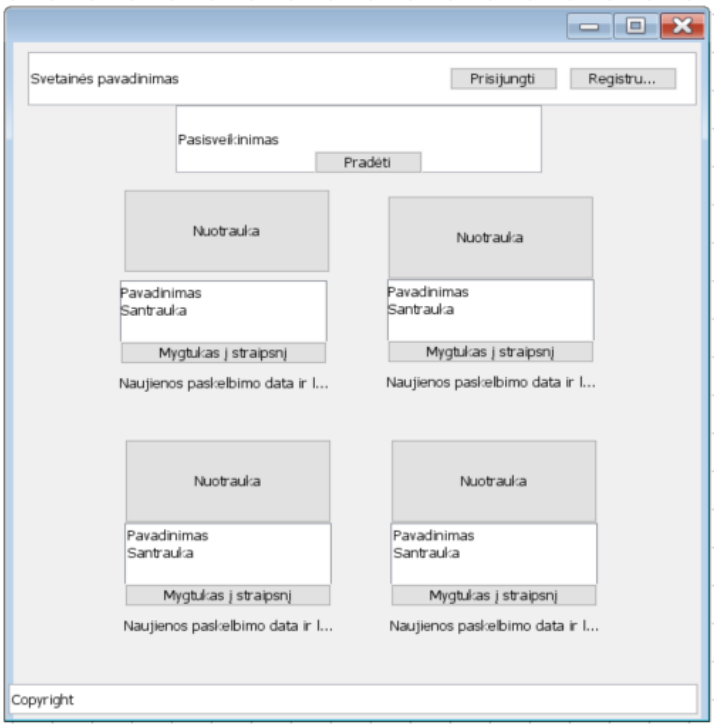
9 PRIEDAS

Svetainių langų šablonai



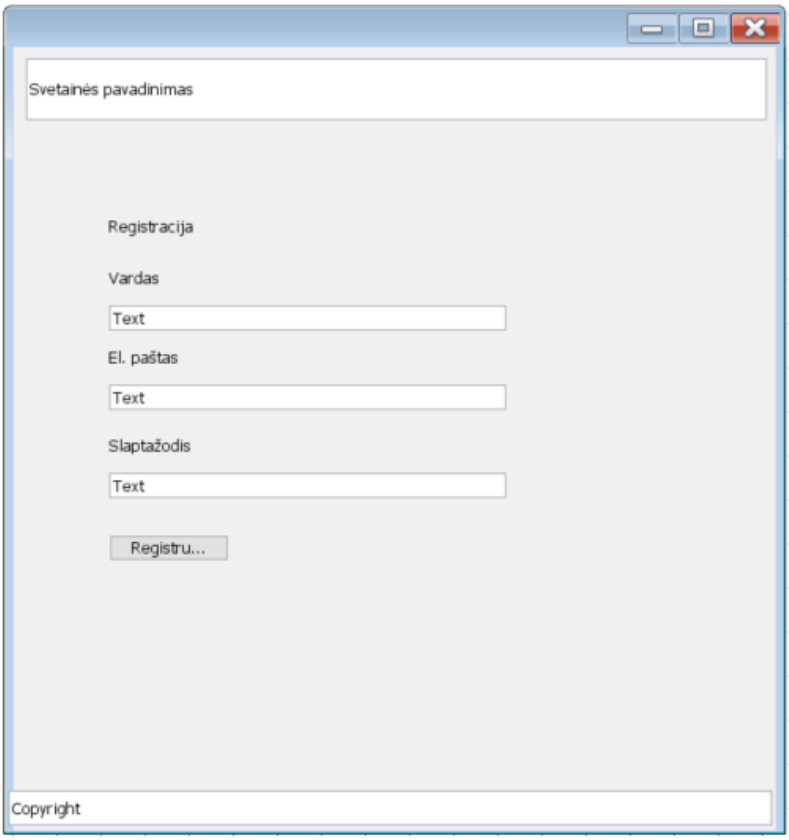
Šaltinis: sudaryta autoriaus.

1 pav. Kontrolinės aplikacijos be šifravimo ir eksperimentinės aplikacijos su šifravimu pagrindinio lango šablonas



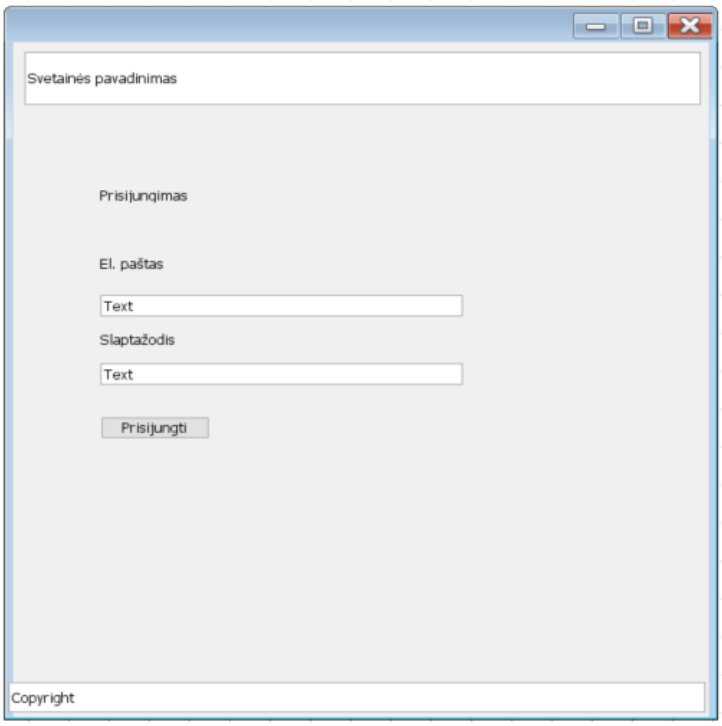
Šaltinis: sudaryta autoriaus.

2 pav. Kontrolinės aplikacijos be autentifikacijos mechanizmo ir eksperimentinės aplikacijos su autentifikacijos mechanizmu pagrindinio lango šablonas



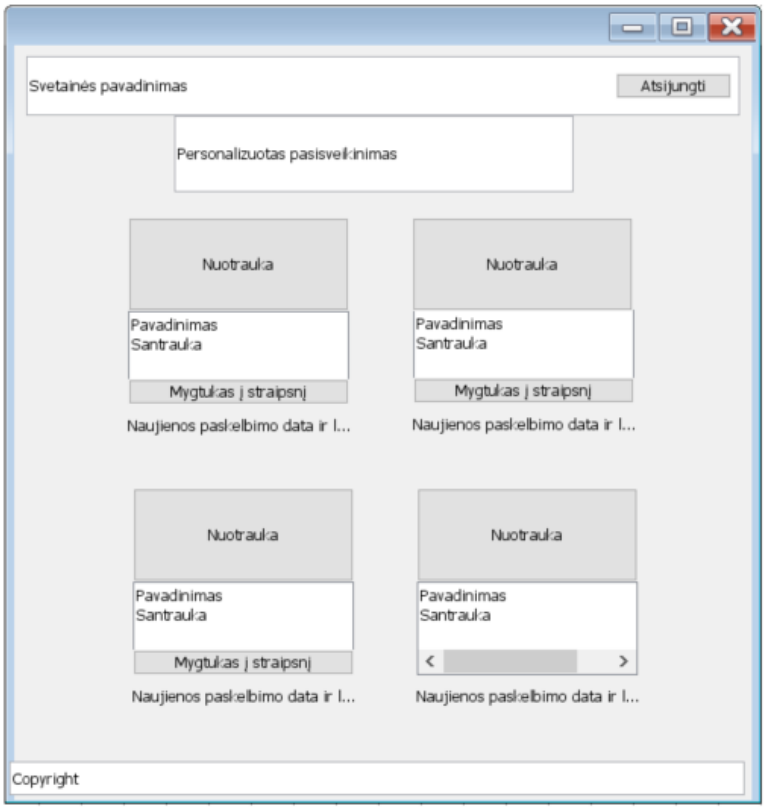
Šaltinis: sudaryta autoriaus.

3 pav. Kontrolinės aplikacijos be autentifikacijos mechanizmo ir eksperimentinės aplikacijos su autentifikacijos mechanizmu registracijos lango šablonas



Šaltinis: sudaryta autoriaus.

4 pav. Kontrolinės aplikacijos be autentifikacijos mechanizmo ir eksperimentinės aplikacijos su autentifikacijos mechanizmu prisijungimo lango šablonas

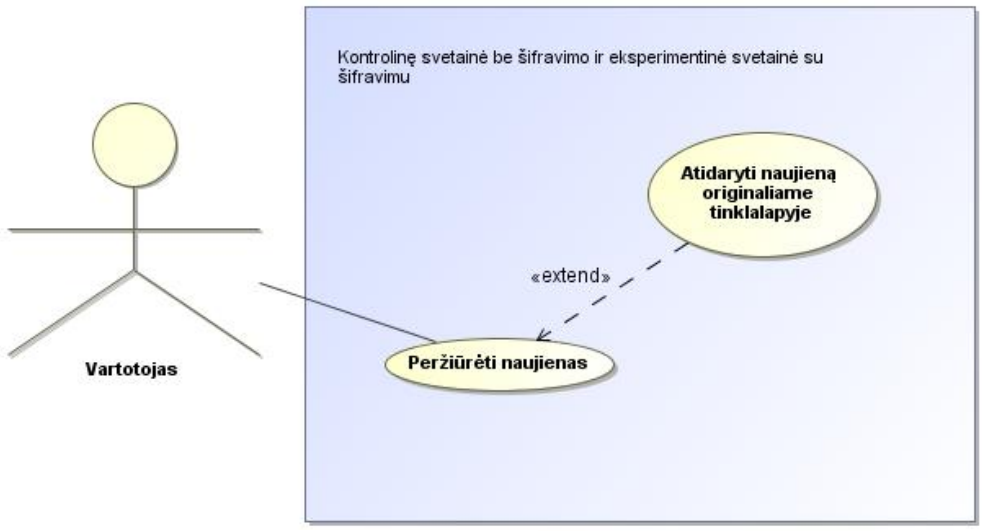


Šaltinis: sudaryta autoriaus.

5 pav. Kontrolinės aplikacijos be autentifikacijos mechanizmo ir eksperimentinės aplikacijos su autentifikacijos mechanizmu pagrindinio lango (po prisijungimo) šablonas

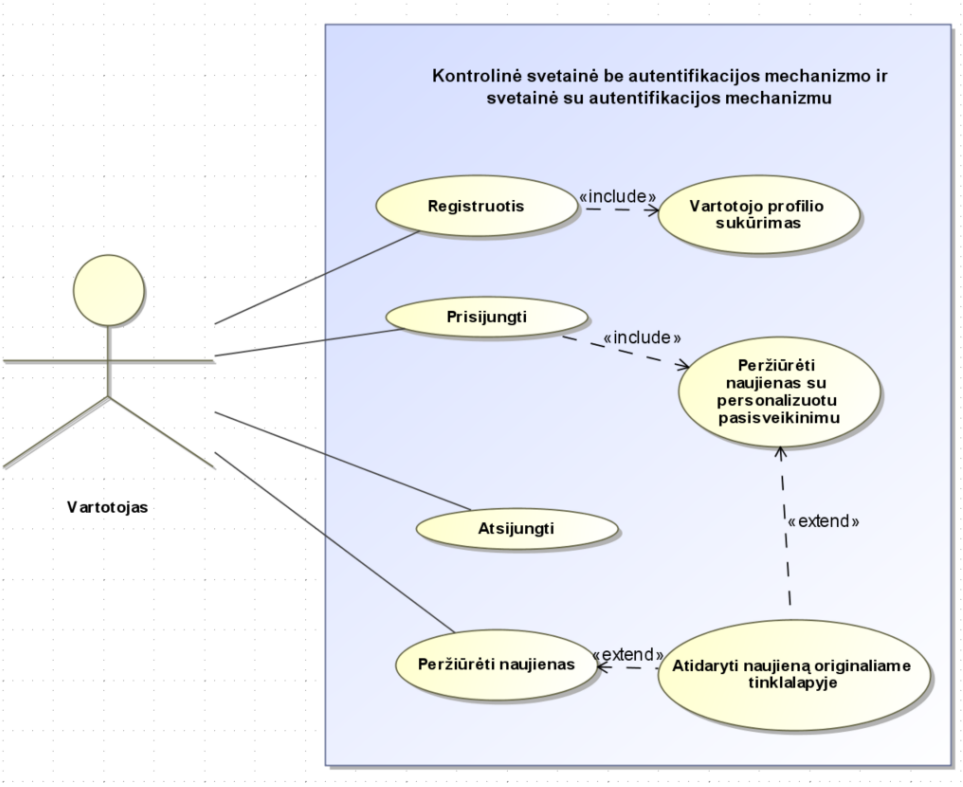
10 PRIEDAS

Svetainių UML diagramos



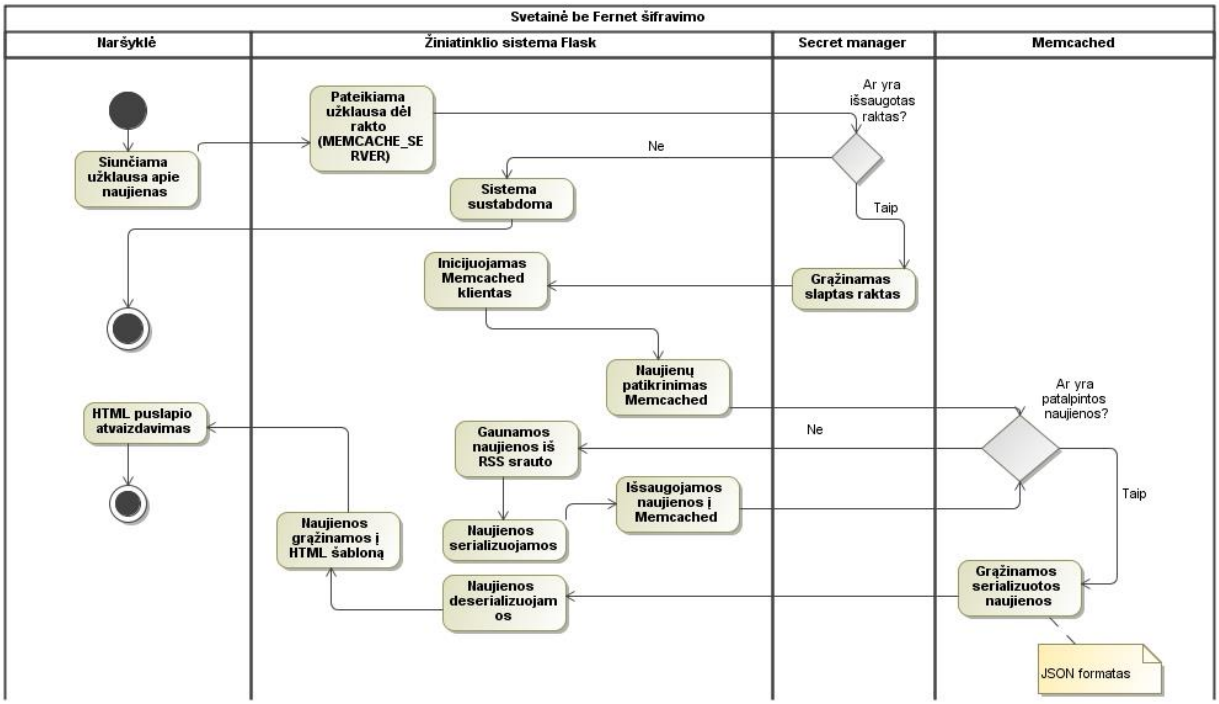
Šaltinis: sudaryta autoriaus.

1 pav. Kontrolinės svetainės be šifravimo ir eksperimentinės svetainės su šifravimu panaudos atvejų diagrama



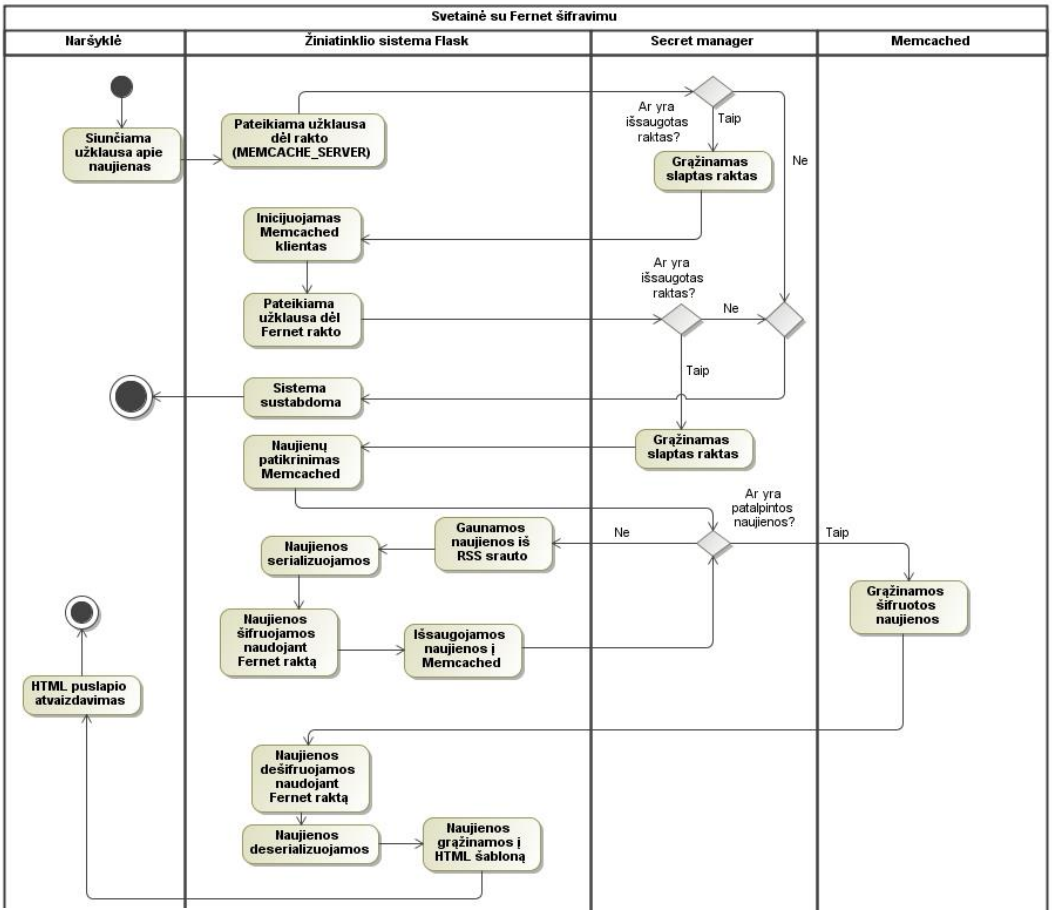
Šaltinis: sudaryta autoriaus.

2 pav. Kontrolinės svetainės be autentifikacijos mechanizmo ir eksperimentinės svetainės su autentifikacijos mechanizmu panaudos atvejų diagrama



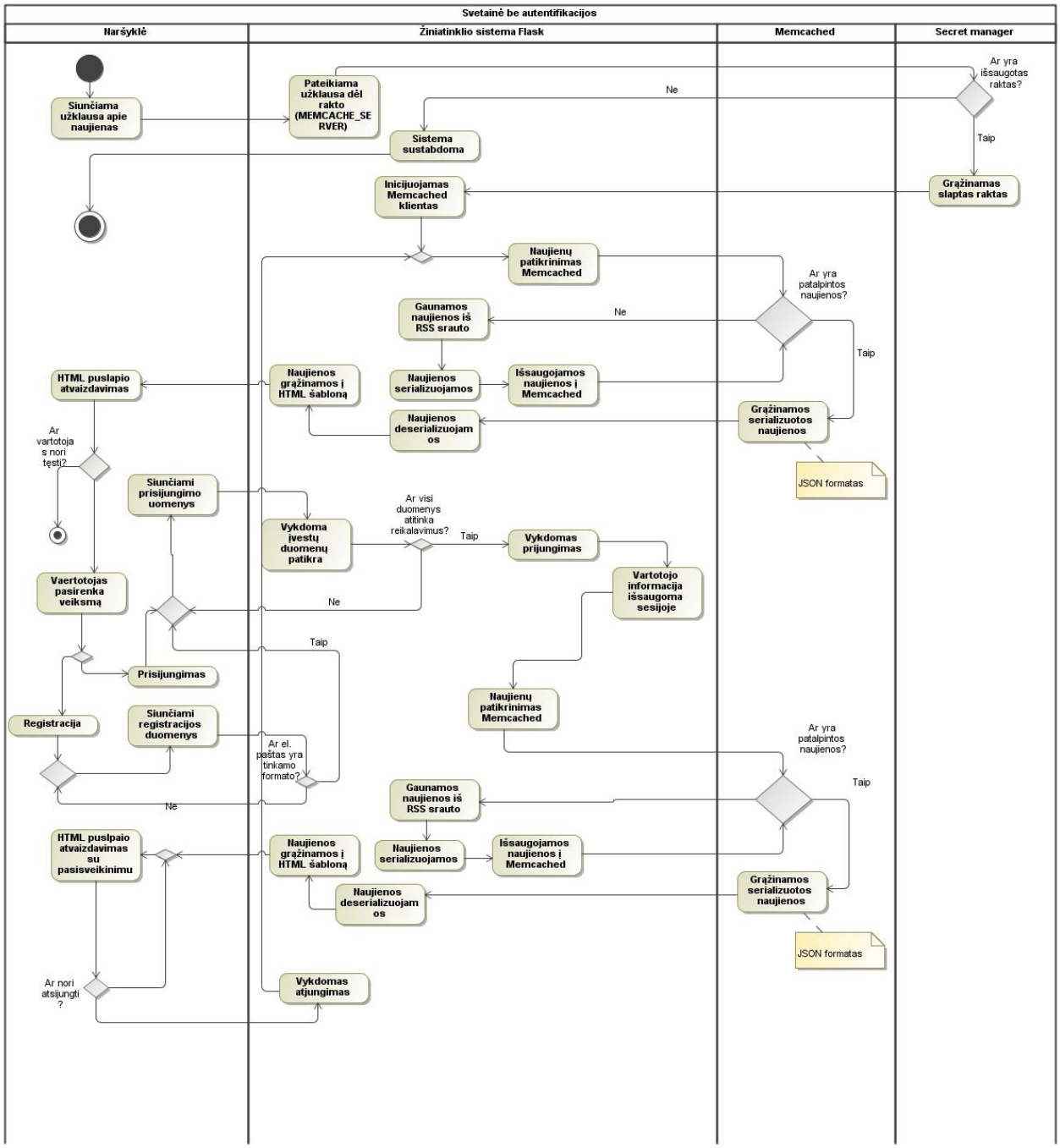
Šaltinis: sudaryta autoriaus.

3 pav. Svetainės be šifravimo veiklos diagrama



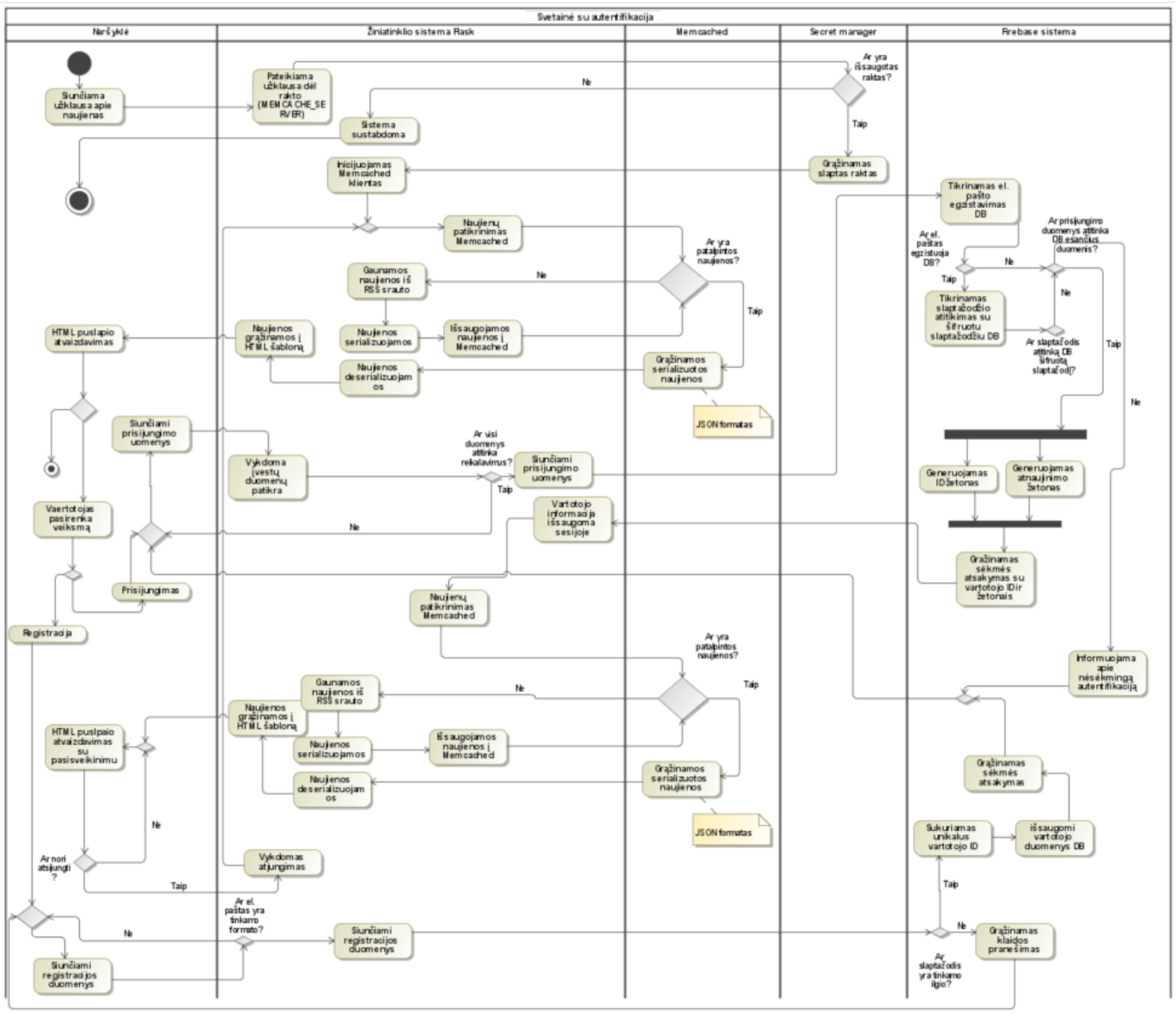
Šaltinis: sudaryta autoriaus.

4 pav. Svetainės su šifravimu veiklos diagrama



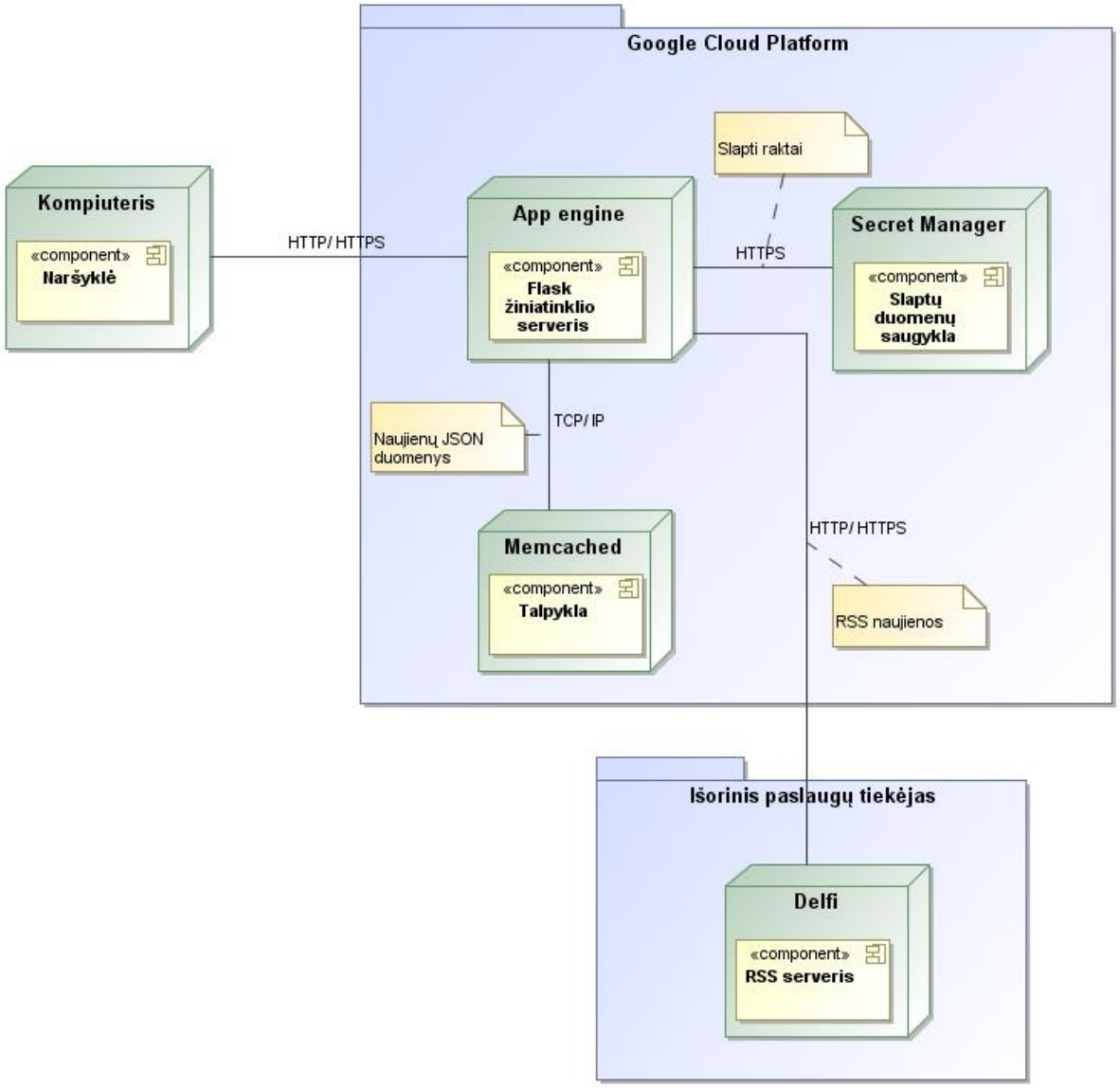
Šaltinis: sudaryta autoriaus.

5 pav. Svetainės be autentifikacijos mechanizmo veiklos diagrama



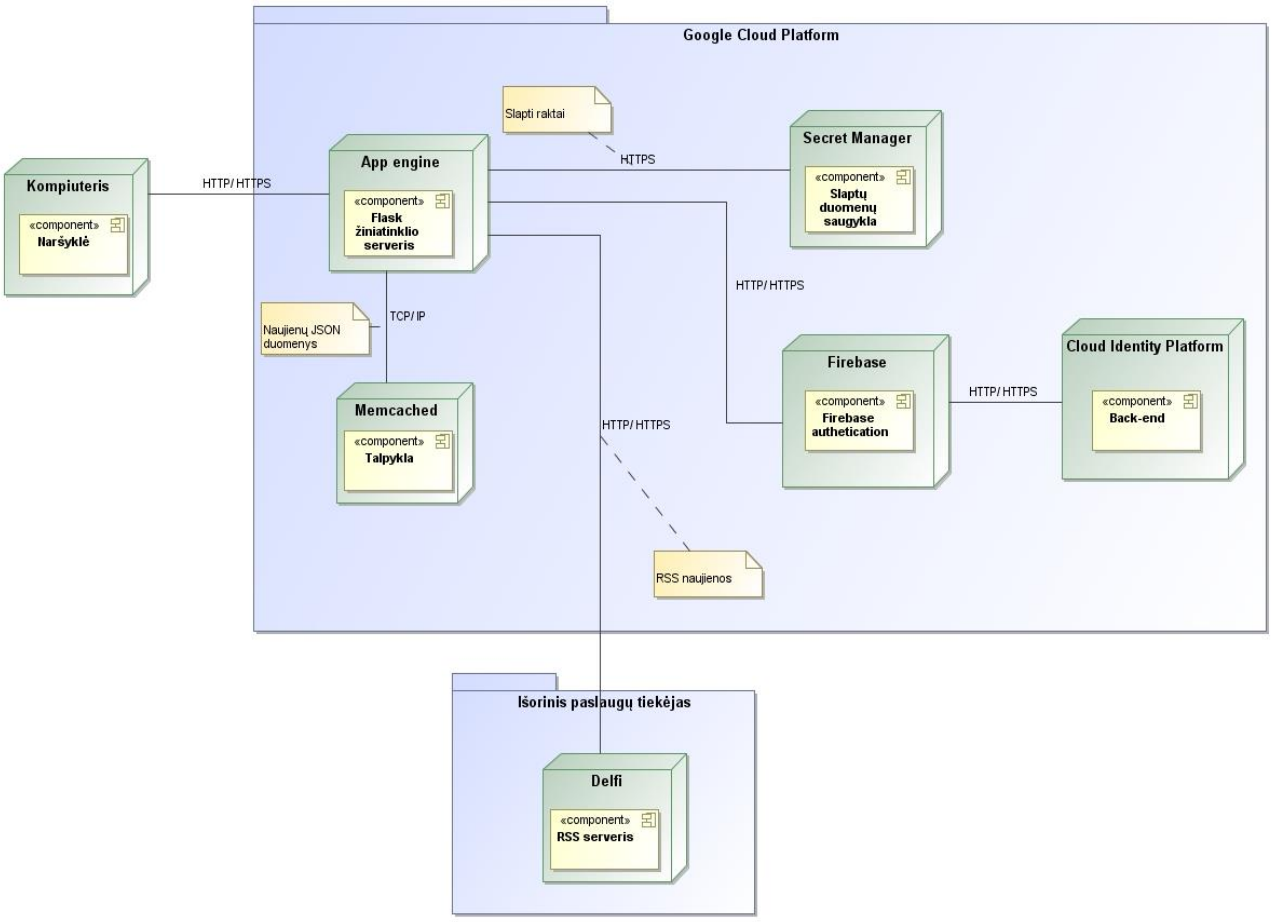
Šaltinis: sudaryta autoriaus.

6 pav. Svetainės su autentifikacijos mechanizmu veiklos diagrama



Šaltinis: sudaryta autoriaus.

7 pav. Svetainių (kontrolinė be šifravimo, eksperimentinė su šifravimu, kontrolinė be autentifikacijos) diegimo diagrama



Šaltinis: sudaryta autoriaus.

8 pav. Svetainės su *Firebase* autentifikacijos mechanizmu diegimo diagrama

11 PRIEDAS

Technologijų ir saugos sprendimų apžvalga

Visos programų versijos naudoja vienodas technologijas:

- Programavimo kalba: *Python 3.9*;
- Debesų kompiuterijos platforma: GCP;
    - *Memorystore* duomenų talpykla;
        - Tipas: *Memcached*;
        - Regionas: *us-central1*;
        - Mazgų (angl. *nodes*) skaičius: 1;
        - Atmintis per mazgą: 1 GB;
        - Procesoriaus branduoliai per mazgą: 1 vCPU;
        - *Memcached* versija: 1.6.15;
    - Kiti GCP komponentai:
        - *Secret Manager* (slaptų duomenų saugojimui);
        - *Logs Explorer* (žurnalų (angl. *logs*) stebėjimui ir analizei);
    - PaaS: GAE;
        - Aplinka: standartinė aplinka;
        - Regionas: *us-central1*;
- Žiniatinklio karkasas: *Flask*;

Eksperimentinių versijų specifinės technologijos:

- Autentifikacijai:
    - *Firebase* (papildoma GCP paslauga);
        - Planas: *Spark* (nemokamas, skirtas nedideliems projektams);
        - Autentifikacija: *Firebase Authentication*;
- Duomenų šifravimui:
    - *Cryptography* biblioteka (*Fernet*);

Tiriamųjų aplikacijų versijos (kontrolinės ir eksperimentinės) pasižymi identiška pagrindine logika, išskyrus saugos komponentų integraciją. Visos aplikacijos diegiamos vienodomis sąlygomis GAE standartinėje aplinkoje, dalijasi tuo pačiu *Memcached* egzemplioriumi.

Eksperimentiniame tyrime buvo pasirinkta naudoti GAE standartinę aplinką, siekiant sumažinti tyrimo išlaidas. GAE suteikia nemokamą 90 dienų bandomąjį laikotarpį su 270 € kreditu, kuris leido efektyviai paskirstyti biudžetą. Papildomai, norint praturtinti tyrimo infrastruktūrą, buvo pasirinktas valdomas *Memcached* sprendimas, kurio kaina, pasirinkus mažiausius parametrus, siekia 37,71 € per mėnesį. Paskaičiuota, kad gautų lėšų pakaks iki tyrimo įgyvendinti.

Fernet buvo pasirinktas dėl jo paprasto įgyvendinimo ir aukšto saugumo lygio. Jis užtikrina konfidencialumą, vientisumą ir autentifikaciją, garantuojant, kad duomenys negali būti modifikuoti ar perskaityti be rakto. Naudodamas 128 bitų raktą ir inicializavimo vektorių (IV), *Fernet* apsaugo duomenis nuo pakartotinio šifravimo identiškų rezultatų. Be to, tai yra *Python Cryptography* bibliotekos dalis, kuri yra gerai dokumentuota ir plačiai pritaikoma. [89] Šios savybės užtikrina sklandžią integraciją į projektą ir efektyvų šifravimą.

Firebase Authentication buvo pasirinktas dėl jo universalumo ir paprastos integracijos su GAE. Šis sprendimas palaiko įvairius autentifikavimo būdus, įskaitant slaptažodžius ir federuotą prisijungimą su *Google, Facebook, Twitter*, o tai leidžia lengvai pritaikyti sistemą augančioms vartotojų bazėms. Be to, *Firebase Authentication* yra lengviausias būdas nustatyti vartotojų autentifikaciją GAE programoms, nes jis sumažina diegimo laiką ir papildomas išlaidas. [43] Jo patikimumas ir gebėjimas apdoroti didelius srautus daro jį idealiu sprendimu šiam tyrimui.

12 PRIEDAS

Eksperimento apkrovų modeliavimas ir vartotojai

Eksperimento metu, testuojant aplikaciją be realaus autentifikacijos mechanizmo ir aplikaciją su realiu autentifikacijos mechanizmu, buvo pasirinktas specifinis apkrovų ir vartotojų elgsenos modeliavimas, siekiant imituoti realias naudojimo sąlygas ir išanalizuoti svarbiausius autentifikacijos sprendimo poveikio aspektus. Apkrovos pasiskirstymas procentais užtikrino, kad testavimo scenarijai apimtų visus reikšmingus vartotojų veiksmus ir atspindėtų realų sistemos naudojimą.

Didžioji dalis apkrovos (76,93%) buvo skirta prisijungimui ir pagrindinio puslapio peržiūrai, nes šie veiksmai sudaro pagrindinę vartotojų elgseną elektroninėse sistemose. Likusi apkrovos dalis buvo paskirstyta registracijai (15,38%) ir atsijungimui (7,69%), kurie atspindi mažiau dažnus, tačiau autentifikacijos mechanizmų testavimui svarbius veiksmus.

Modeliuojama, kad tik 20% vartotojų yra nauji ir gali atlikti registracijos procesą, tačiau registracija sudaro 15,38% visos apkrovos. Tai užtikrina, kad registracija būtų įvertinta kaip svarbi funkcija, tačiau ne dominuojanti, nes dauguma svetainės lankytojų jau turi paskyrą arba naršo neprisijungę. Didžiausia veiksmų dalis buvo skirta prisijungimui (30,77%), nes tai pagrindinis autentifikacijos proceso elementas. Autentifikuoti vartotojai dažnai prisijungia prie savo paskyrų tam, kad pasiektų asmeninius duomenis ar naudotų papildomas funkcijas. Prieigai prie pagrindinio puslapio buvo skirta 46,16%. Ši dalis buvo padalyta po lygiai tarp prisijungusių ir neprisijungusių vartotojų (23,08% kiekvienai grupei). Tai atspindi realias elgesio tendencijas, kai reikšminga dalis vartotojų naršo be autentifikacijos, o prisijungę vartotojai dažnai lanko pagrindinį puslapį ar susijusius skyrius. Modeliuojama, kad tik nedidelė vartotojų dalis (7,69%) aktyviai atlieka atsijungimo veiksmą, nes dauguma vartotojų uždaro naršyklę arba palieka svetainę neatlikę atsijungimo proceso.

Testavimui buvo naudojamas automatiškai sugeneruotas *JSON* failas *test_users.json*, kuriame saugoma 2000 vartotojų prisijungimo informacija. Šis failas buvo sukurtas siekiant užtikrinti autentifikuotų vartotojų veiksmų modeliavimą bei supaprastinti autentifikacijos procesą. Naudojant šį failą, vartotojai buvo automatiškai registruojami *Firebase Authentication* platformoje, o tai palengvino autentifikacijos mechanizmų integraciją ir leido tiksliai įvertinti jų veikimo efektyvumą.

13 PRIEDAS

Tyrimo metrikos

Locust matavimo metrikos:

1. Vidutinis atsako laikas *(*angl. *average response time)* parodo bendrą sistemos našumą, apskaičiuojant visų užklausų atsako laikų vidurkį. Vidutinis atsako laikas gali klaidinti, nes neatspindi užklausų pasiskirstymo ar kraštutinių atvejų, sukuriant neteisingą įspūdį apie sistemos našumą. [90] [91]
2. Atsako laiko mediana *(*angl. *median response time)* nurodo vidurinę atsako laiko reikšmę, kai pusė užklausų turi trumpesnį, o kita pusė – ilgesnį atsako laiką.
3. Didžiausias atsako laikas *(*angl. *max response time)* nurodo ilgiausią užfiksuotą atsako laiką, rodantį blogiausią sistemos reakcijos scenarijų.
4. Užklausos per sekundę *(*angl. *requests/s)* parodo, kiek užklausų sistema gali apdoroti per sekundę. Tai leidžia įvertinti sistemos pajėgumą ir stabilumą esant skirtingoms apkrovoms. [92]
5. Klaidų dažnis *(*angl. *error rate)* parodo procentinę dalį užklausų, kurios baigėsi klaidomis, sistemos patikimumą. Aukštas klaidų dažnis rodo problemas, kurios gali sumažinti vartotojo patirtį arba netgi sukelti paslaugos nutraukimą. [91]
6. 90% atsako laikas *(*angl. *90th percentile response time)*: 90-asis procentilis nurodo, kad 90% imties verčių yra žemiau šios ribos, o likusieji 10 % yra virš jos. Šis rodiklis leidžia įvertinti daugumos vartotojų patirtį. [90]
7. 95% atsako laikas *(*angl. *95th percentile response time):* 95-asis procentilis nurodo, kad 95% užklausų atsako laikas yra mažesnis už šią reikšmę, o likusieji 5% užtrunka ilgiau. [90]
8. 99% atsako laikas *(*angl. *percentile response time):* 99-asis procentilis parodo, kad 99% užklausų atsako laikas yra mažesnis už šią reikšmę, o tik 1% užklausų viršija šią ribą. Šis rodiklis yra ypač naudingas aptinkant išskirtinius atvejus. [90]

Google Cloud Console Monitoring App Engine rodiklių metrikos:

1. *Memorystore Memcached Node* - talpyklos atminties naudojimas *(*angl. *cache memory usage)* leidžia įvertinti resursų efektyvumą ir nustatyti galimus atminties trūkumo ar fragmentacijos atvejus. [93]

2. *Memorystore Memcached Node* - CPU naudojimo procentas *(*angl. *CPU usage percent).* CPU naudojimo procentas rodo procesoriaus apkrovą. Aukštas CPU naudojimas gali reikšti procesų neoptimizavimą, kuris sumažina sistemos atsako greitį. [94]
3. „*Memorystore Memcached Node* - iškeldinimų skaičius *(*angl. *eviction count)* nurodo, kiek elementų pašalinta iš talpyklos dėl atminties apribojimų. Dažnas iškeldinimas gali reikšti, kad talpyklos dydis yra per mažas. [93]
4. GAE aplikacija - atminties naudojimas *(*angl. *memory usage)* rodo bendrą atminties kiekį, naudojamą aktyvių GAE aplikacijos instancijų veikimo metu. [95]
5. GAE aplikacija - atsakymų skaičius *(*angl. *response count)* matuoja HTTP atsakymų, kuriuos sugeneruoja GAE aplikacija, skaičių per tam tikrą laikotarpį. [95]
6. *Memorystore Memcached Node* - pataikymo dažnis *(*angl. *Memorystore Memcachedd Node - Hit Ratio)* rodo, kiek užklausų aptarnaujama iš talpyklos, o tai mažina pagrindinės duomenų bazės apkrovą. [93]
7. GAE aplikacija - atsako uždelsimas *(*angl. *response latency)* matuoja laiką, per kurį GAE aplikacija apdoroja HTTP užklausą ir pateikia atsakymą. [95]
8. GAE aplikacija - gautų ir išsiųstų baitų kiekis *(*angl. *received and sent bytes)* nurodo vidutinį duomenų kiekį, kurį GAE aplikacija gauna ir išsiunčia per tam tikrą laiką. [95]
9. *Memorystore Memcached Node* - sistemos atminties panaudojimas *(*angl. *system memory utilization)* nurodo, kiek procentų visos prieinamos atminties (įskaitant talpykloje saugomus elementus ir pridėtą atminties režiją) yra išnaudojama. Tai svarbi metrika, nes ji leidžia stebėti, kaip yra arti atminties perkrovos būsena. Kai šis rodiklis artėja prie 100%, didėja tikimybė, kad sistema nebegalės vykdyti naujų užklausų. [93]
10. GAE aplikacija - CPU naudojimas *(*angl. *CPU utilization)* nurodo vidutinį procesoriaus panaudojimo lygį visose aktyviose GAE aplikacijos instancijose. [95]
11. *Memorystore Memcached Node* - operacijų skaičius *(*angl. *Memcached operation count)* leidžia įvertinti, kiek operacijų per tam tikrą laiką apdorojama *Memcached* mazge, identifikuojant galimus srauto sutrikimus. [94]

14 PRIEDAS

Locust stats.csv failų suvestinės analizė šifravimo tyrimui testavimo su pertraukomis metu

Rezultatų suvestinėje pateikiami skirtingų vartotojų skaičiais paremtų testų rezultatai, įskaitant vidutinį ir atsako laiko medianą, didžiausią atsako laiką, užklausų per sekundę skaičių, klaidų dažnį bei atsako laikus pagal 90%, 95% ir 99% procentilius.

Vartotojų skaičius	Aplikacijos tipas	Vidutinis atsako laikas, ms	Atsako laiko mediana, ms	Didžiausias atsako laikas, ms	Užklausų per sekundę	Klaidų dažnis (%)	90% atsako laikas (ms)	95% atsako laikas (ms)	99% atsako laikas (ms)
50	Kontrolinė	233,98	230	1761,51	22,26	0	300	320	440
50	Eksperimentinė	306,92	240	3418,01	20,72	0	340	730	2100
100	Kontrolinė	1116,22	900	6272,64	27,47	0	1900	2600	3500
100	Eksperimentinė	1727,80	1300	7002,05	21,67	0	3200	3500	4500
200	Kontrolinė	4242,90	4100	18227,88	25,13	0	6600	7700	11000
200	Eksperimentinė	4812,25	5000	19357,63	22,65	0	7300	7600	9600
400	Kontrolinė	10217,46	5100	224382,72	24,33	0	8800	11000	206000
400	Eksperimentinė	12373,98	8800	156458,72	22,07	0	14000	19000	135000
800	Kontrolinė	16584,33	6200	436774,66	22,16	0	11000	24000	330000
800	Eksperimentinė	17002,97	6600	482936,56	19,50	0	11000	18000	389000
1000	Kontrolinė	12116,63	4300	492522,98	26,99	0	7800	11000	337000
1000	Eksperimentinė	18279,85	12000	480751,47	18,17	0	16000	21000	294000

Šaltinis: sudaryta autoriaus.

Iš lentelėje pateiktų duomenų matyti, kad vidutinis atsako laikas eksperimentinėje aplikacijoje buvo didesnis visuose apkrovos taškuose. Pvz., esant 50 vartotojų apkrovai, kontrolinė aplikacija užfiksavo 233,98 ms, o eksperimentinė – 306,92 ms (padidėjimas 31 %). Didėjant apkrovai iki 1000 vartotojų, vidutinis atsako laikas kontrolinėje aplikacijoje sumažėjo iki 12116,63 ms, tačiau eksperimentinėje jis pasiekė 18279,85 ms (padidėjimas 51 %). Ryškiausias skirtumas pastebimas didelių apkrovų metu, kur eksperimentinės aplikacijos našumas sumažėja dėl šifravimo sukeliamos papildomos resursų apkrovos.

Atsako laiko mediana eksperimentinėje aplikacijoje taip pat buvo reikšmingai didesnė, ypač esant didesnėms apkrovoms. Pvz., esant 400 vartotojų apkrovai kontrolinės aplikacijos mediana buvo 5100 ms, o eksperimentinės – 8800 ms (72,55% padidėjimas). Esant 1000 vartotojų apkrovai kontrolinės aplikacijos mediana sumažėjo iki 4300 ms, tačiau eksperimentinėje ji pasiekė 12000 ms, beveik tris kartus viršydama kontrolinės aplikacijos reikšmę. Šie rezultatai rodo, kad šifravimas ypač veikia atsako laiko stabilumą esant didelėms apkrovoms.

Didžiausias atsako laikas eksperimentinėje aplikacijoje dažniausiai buvo didesnis, išskyrus kai kurias specifines apkrovos situacijas. Pvz., ties 50 vartotojų kontrolinėje aplikacijoje užfiksuota

1761,51 ms, o eksperimentinėje – 3418,01 ms (94 % padidėjimas). Vis dėlto esant 400 vartotojų kontrolinė aplikacija užfiksavo 224382,72 ms, o eksperimentinė – 156458,72 ms (30 % sumažėjimas). Šis neatitikimas gali būti susijęs su vidinių eilių dinamika ir resursų paskirstymo mechanizmais.

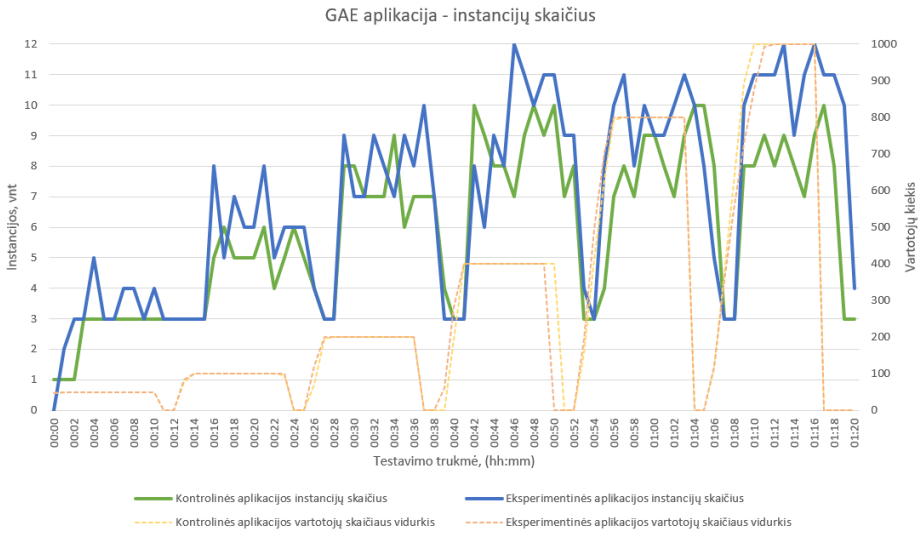
Eksperimentinėje aplikacijoje užklausų apdorojimo sparta buvo nuosekliai mažesnė nei kontrolinėje. Pvz., esant 50 vartotojų apkrovai, kontrolinė aplikacija apdorodavo 22,26 užklausos per sekundę, o eksperimentinė – 20,72 (7 % sumažėjimas). Esant 1000 vartotojų, kontrolinė aplikacija pasiekė 26,99 užklausos per sekundę, o eksperimentinė – 18,17 (33 % sumažėjimas). Šie rezultatai atskleidžia, kad šifravimas sumažina apdorojamų užklausų kiekį didėjant apkrovoms.

Procentiliniai rodikliai rodo bendrą atsako laiko padidėjimą eksperimentinėje aplikacijoje, tačiau kai kuriais atvejais fiksuojamos mažesnės reikšmės. Pvz., esant 400 vartotojų 99% atsako laikas kontrolinėje aplikacijoje buvo 206000 ms, o eksperimentinėje – 135000 ms (34 % sumažėjimas). Tačiau esant 1000 vartotojų apkrovai, 99% atsako laikas eksperimentinėje aplikacijoje buvo 294000 ms, palyginti su 337000 ms kontrolinėje (13 % sumažėjimas).

15 PRIEDAS

Google Cloud Console Monitoring grafikų analizė (šifravimo poveikis testavimo su pertraukomis metu)

Instancijų skaičius GAE aplinkoje (angl. *GAE Application – instance count*):



Šaltinis: sudaryta autoriaus.

GAE aplikacijos instancijų skaičiaus grafike matyti, kaip GAE automatinio mastelio keitimo mechanizmas pritaiko resursus, kai didėja apkrova (vartotojų skaičius). Instancijų skaičius gali svyruoti tarp 1 ir 12, priklausomai nuo apkrovos. Pvz., kai vartotojų skaičius pasiekia 800–1000, instancijų skaičius išauga iki 8–12 eksperimentinėje aplikacijoje ir iki 7–10 kontrolinėje. Tai rodo, kad eksperimentinė aplikacija dažniau paleidžia daugiau instancijų, galimai reaguodama į papildomą apkrovą (pvz., šifravimo procesą).

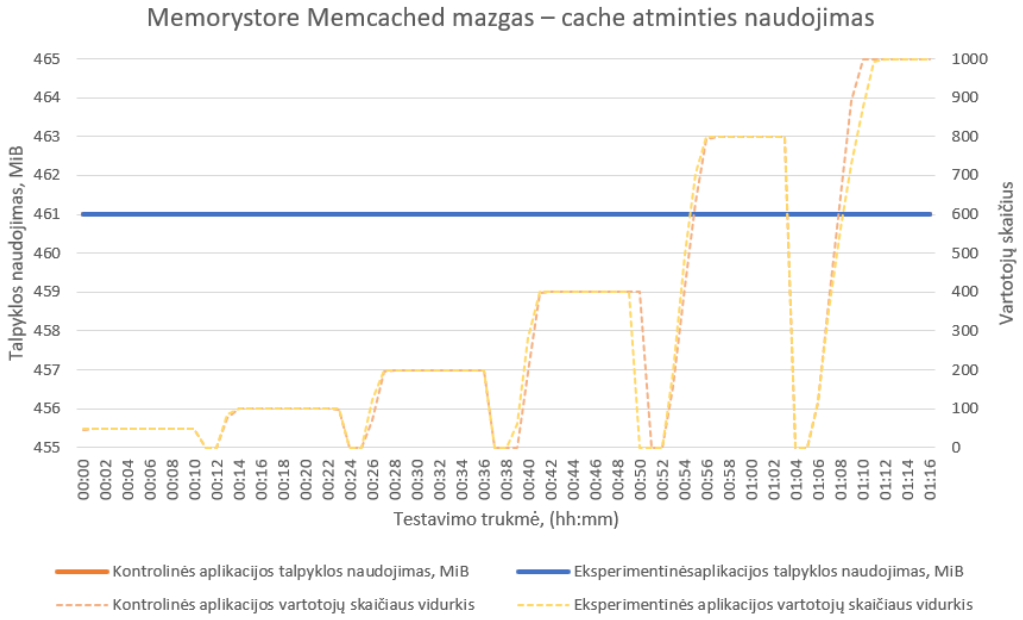
Eksperimentinė aplikacija periodiškai turi daugiau aktyvių instancijų nei kontrolinė. Tai reiškia, kad GAE automatinio mastelio keitimo mechanizmas, užfiksavęs didesnį atsako laiką ar didesnę apkrovą, automatiškai „išplečia“ eksperimentinės aplikacijos resursus. Tokia strategija dažnai kompensuoja šifravimo / autentifikacijos sukeliamus uždelsimus – daugiau instancijų leidžia apdoroti daugiau užklausų lygiagrečiai. Kartais būtent dėl šios priežasties matome, kad kai kuriuose apkrovos etapuose eksperimentinė aplikacija pasiekia labai panašius ar net geresnius procentilinius atsako laikus nei kontrolinė.

Grafike pastebimos atkarpos (pvz., apie 00:39 arba 01:07 val.), kai vartotojų skaičius sumažėja, ir automatinio mastelio keitimo mechanizmas sumažina dalį instancijų. Tai padeda valdyti

sąnaudas (mokama tik už aktyvius resursus) ir rodo, kad GAE realiu laiku reaguoja į apkrovos pokyčius. Svarbu pabrėžti (atsižvelgiant į *Locust* suvestinės duomenis, kurie pateikiami **14 PRIEDAS *Locust stats.csv* failų suvestinės analizė šifravimo tyrimui testavimo su pertraukomis metu**), kad daugiau instancijų ne visada garantuoja mažesnį atsako laiką, ypač jei uždelsimai kyla dėl konkrečių procesų (pvz., šifravimo logikos). Tačiau didesnis instancijų skaičius paprastai padeda išlaikyti mažą klaidų dažnį (0 %) ir sumažinti eilių susidarymą, kai užklausų kiekis pasiekia piką.

Šis instancijų skaičiaus kitimo grafikas aiškiai pabrėžia, kaip GAE automatinio mastelio keitimo mechanizmas realiu laiku prisitaiko prie augančios apkrovos tiek kontrolinėje, tiek eksperimentinėje aplikacijoje. Eksperimentinė aplikacija dažnai reikalauja šiek tiek daugiau resursų (iki 12 instancijų), tikėtina, reaguodama į šifravimo / autentifikacijos papildomą krūvį. Toks elgesys padeda paaiškinti, kodėl esant tam tikroms apkrovos situacijoms eksperimentinėje aplikacijoje aukštųjų procentilių (pvz., 95% ar 99%) atsako laikas gali būti geresnis (arba tik šiek tiek prastesnis) nei kontrolinėje versijoje: išaugintas instancijų skaičius subalansuoja papildomą šifravimo naštą. Šitaip instancijų skaičiaus dinamika tampa vienu iš svarbiausių debesų kompiuterijos pranašumų: nors šifravimas ar autentifikacija turi tam tikrą neigiamą poveikį atsako laikui, automatinis resursų išplėtimas leidžia minimalizuoti bendrą poveikį vartotojų patirčiai bei užkirsti kelią klaidų augimui.

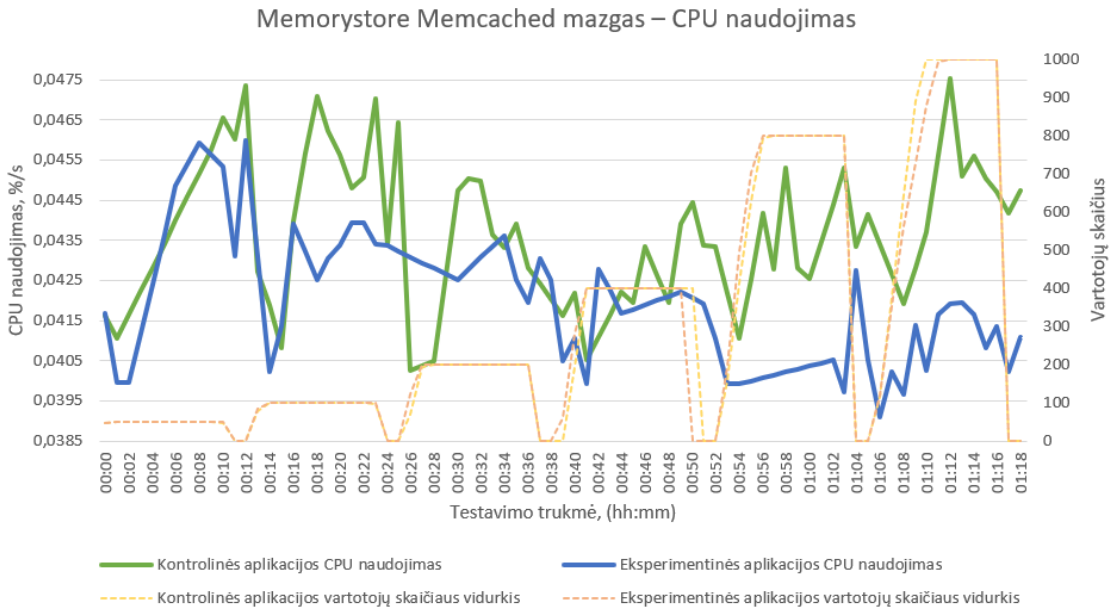
Talpyklos atminties naudojimas (angl. *Memorystore Memcached node – cache memory usage*):



Šaltinis: sudaryta autoriaus.

Atlikta analizė parodė, kad abiejų – kontrolinės ir eksperimentinės – sistemų *cache* talpyklos atminties naudojimas išlieka praktiškai nepakitęs ir stabilus, ties 461 MiB. Tai leidžia daryti išvadą, kad įtraukus šifravimo mechanizmą, papildoma našumo apkrova, susijusi su atminties vartojimu *Memcached* lygmeniu, nėra pastebima. Stabilus atminties naudojimas rodo, kad šifravimas neiškraipo duomenų talpyklos struktūros ar jos valdymo principų, o duomenų talpyklos dydis ir konfigūracijos parametrai yra pakankami ir tinkami esamai apkrovai tiek be šifravimo, tiek ir su juo. [96]

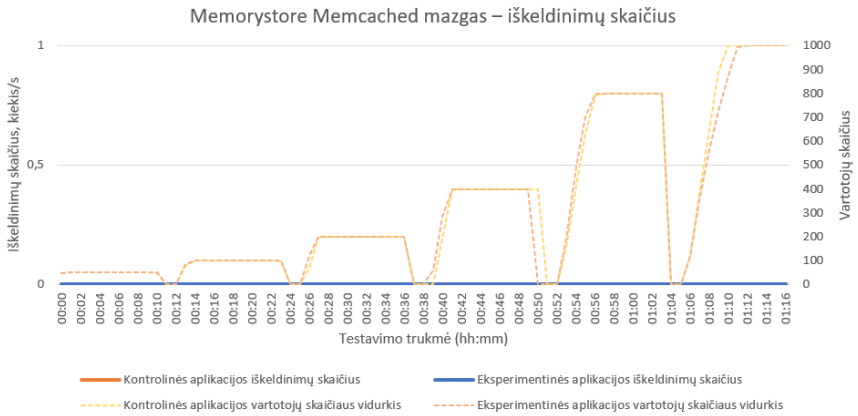
Procesoriaus naudojimas (angl. *Memorystore Memcached Node – CPU usage percent*):



Šaltinis: sudaryta autoriaus.

Analizuojant procesoriaus išteklių panaudojimą *Memcached* lygiu, matyti, kad kontrolinės sistemos CPU apkrova svyravo intervale nuo ~0,041 iki 0,0475 %/s (vidurkis - 0,0439 %/s). Eksperimentinėje sistemoje, kurioje taikomas šifravimas, CPU apkrovos vidurkis siekė ~0,0419 %/s ir svyravo nuo ~0,039 iki 0,046 %/s. Pastebima, kad eksperimentinėje aplinkoje kai kuriais atvejais CPU apkrova buvo netgi mažesnė nei kontrolinėje sistemoje. Šie rezultatai rodo, kad šifravimo mechanizmas, įgyvendintas aplikacijos lygiu, neturi pastebimos neigiamos įtakos *Memcached* paslaugos CPU išteklių naudojimui. Priešingai, CPU resursų vartojimas išlieka efektyvus ir stabilus, o šifravimo sukuriama papildoma apkrova procesoriui yra minimali bei neturi reikšmingo poveikio paslaugos veikimui. Be to, šie rezultatai koreliuoja su GAE instancijų skaičiaus dinamika. Matyti, kad eksperimentinėje sistemoje instancijų skaičiaus pokyčiai atitinka CPU apkrovos tendencijas, o tai rodo efektyvų apkrovos paskirstymą tarp instancijų.

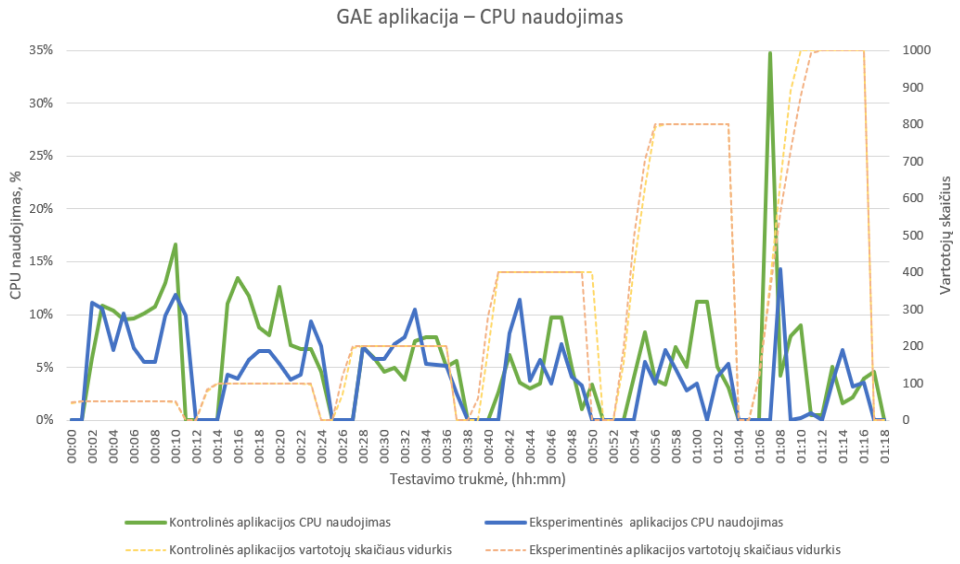
Talpyklos iškeldinimų skaičius (angl. *Memorystore Memcached node – eviction count*):



Šaltinis: sudaryta autoriaus.

Atliekant bandymus ir stebint talpyklos veikimą nė vienu atveju nebuvo užfiksuota *Memcached* iškeldinimų (angl. *evictions*). Šis rodiklis yra svarbus indikatorius, parodantis, ar talpyklos dydis ir konfigūracija atitinka apkrovos poreikius. Iškeldinimų nebuvimas rodo, kad talpykla efektyviai tenkina esamą užklausų dažnį bei duomenų apimtis. Tokia pati situacija matoma ir eksperimentinėje (su šifravimu) sistemoje, todėl galima daryti išvadą, kad šifravimo procesų integracija nesukelia papildomos atminties įtampos ar duomenų perkėlimo poreikio. Optimalūs talpyklos valdymo parametrai išlieka veiksmingi nepriklausomai nuo to, ar naudojamas šifravimas. [93]

CPU naudojimas GAE aplinkoje (angl. *GAE Application – CPU utilization*):

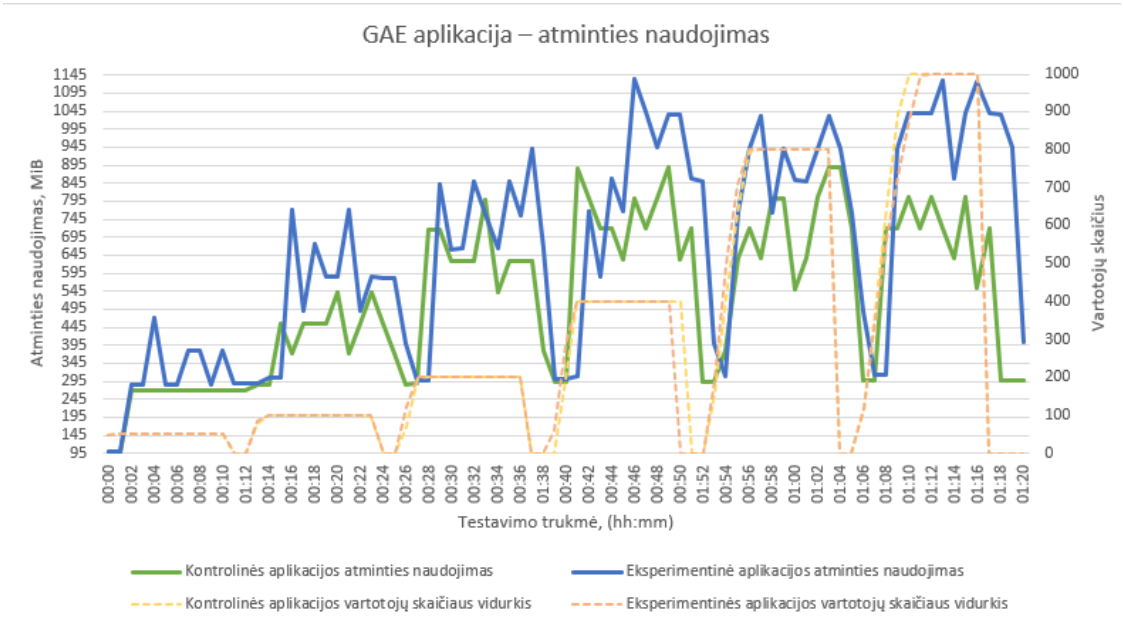


Šaltinis: sudaryta autoriaus.

GAE aplinkoje vykdomos aplikacijos CPU išteklių vartojimo analizė parodė skirtingus apkrovos dydžius. Kontrolinė aplikacija, esant 50–100 vartotojų, naudoja apie 10–15 % CPU, vėliau stabilizuojasi ties 5–10 %. Pasiekus 1000 vartotojų apkrovą, laikinai CPU naudojimas padidėja iki ~35 %. Bendras CPU naudojimo vidurkis testavimo metu yra ~7 %. Šis elgesys rodo, kad sistema geba valdyti didėjančias apkrovas be ilgalaikio neigiamo poveikio našumui.

Eksperimentinėje sistemoje, kurioje integruotas šifravimas, CPU apkrova, esant 50–100 vartotojų, yra kiek mažesnė (6–11 %) nei kontrolinėje, tačiau pastebimi dažni trumpalaikiai šuoliai. Šie nereguliarūs svyravimai gali būti susiję su momentiniais šifravimo operacijų poreikiais, pvz., naujo rakto generavimu, tam tikrų duomenų blokų šifravimu ar iššifravimu [60]. Vis dėlto, ilgalaikės tendencijos rodo, kad CPU vartojimo vidurkis – apie 6 % – nesudaro reikšmingo skirtumo lyginant su kontroline sistema. Tai reiškia, kad šifravimo procesai nepadidina CPU naudojimo taip, kad sukeltų ilgalaikių našumo problemų.

Atminties naudojimas GAE aplinkoje (angl. *GAE Application – memory usage*):



Šaltinis: sudaryta autoriaus.

Analizuojant GAE aplinkos atminties naudojimą, kontrolinėje versijoje pastebimas tolygus atminties resursų poreikio didėjimas, kuris proporcingai atitinka augančias apkrovas. Atminties naudojimas kyla nuo ~270 MiB iki ~600–890 MiB intervalo, vidutiniškai siekdamas apie 529 MiB. Šis sklandus augimas, be ryškių svyravimų, leidžia daryti išvadą, kad sistema efektyviai valdo atminties resursus, o didėjantis jų poreikis natūraliai atspindi apkrovos didėjimą. Kontrolinės versijos atveju instancijų skaičius taip pat auga tolygiai, pasiekdamas iki 9 instancijų esant maksimaliam

vartotojų skaičiui (1000). Tai rodo, kad sistema geba efektyviai paskirstyti apkrovą tarp papildomų instancijų.

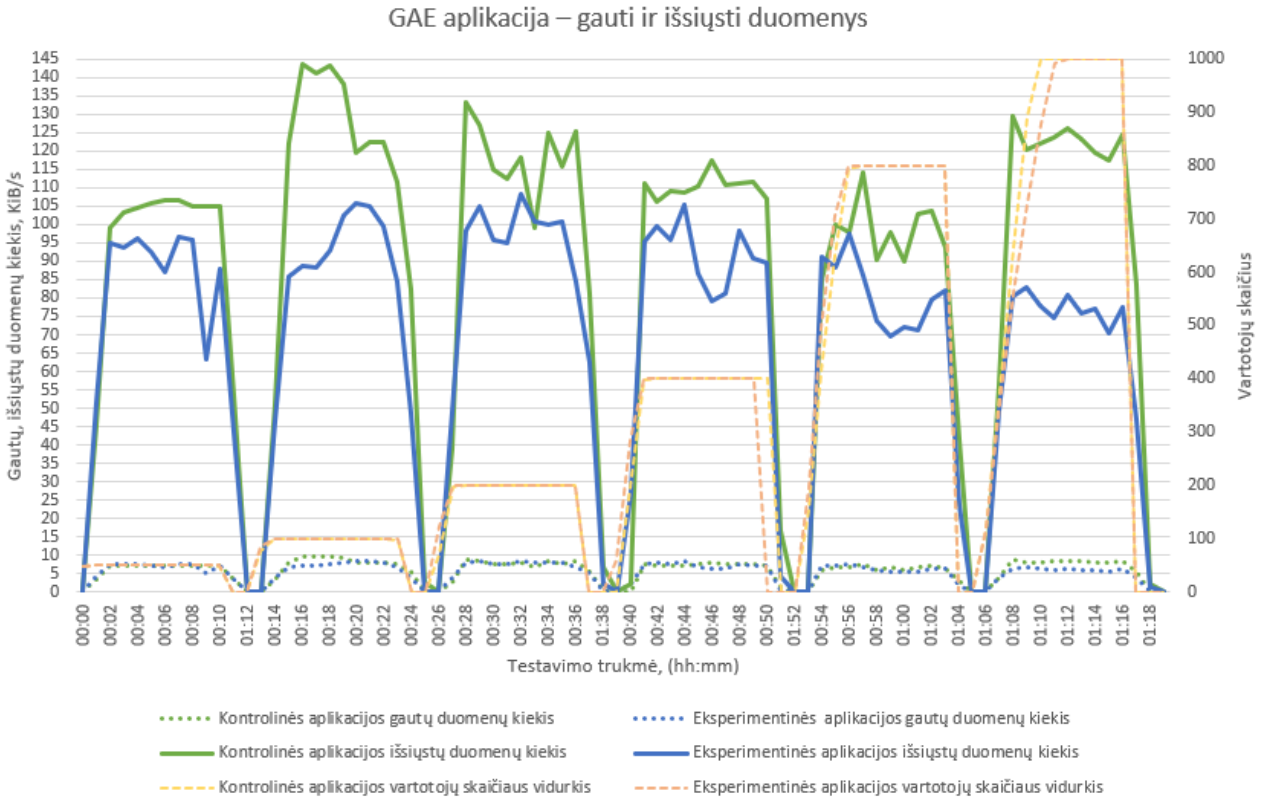
Eksperimentinėje versijoje, naudojančioje šifravimą, atminties poreikis yra didesnis. Esant 400, 1000 vartotojų apkrovai, kartais atminties naudojimas priartėja prie ~1,1 GiB, o vidutinis sunaudojimas sudaro apie 664 MiB. Tai rodo didesnį atminties poreikį, kuris greičiausiai susijęs su šifravimo logika, reikalaujančia papildomų duomenų struktūrų ar buferių, skirtų šifruojamiems ir iššifruojamiems duomenims laikyti [60]. Eksperimentinėje aplikacijoje instancijų skaičius pasiekia iki 12, o tai rodo padidėjusį resursų poreikį šifravimo procesams aptarnauti. Instancijų skaičiaus augimas yra tiesiogiai proporcingas vartotojų skaičiui, tačiau tai taip pat reiškia didesnes sąnaudas bei išaugusį resursų naudojimą. Eksperimentinės aplikacijos bendras atminties naudojimas, pasiekęs ~1,1 GiB, apima kelių instancijų sumą, o ne vienos instancijos poreikį. Vidutiniškai kiekviena instancija sunaudoja apie 91,67 MiB, o tai yra žymiai mažiau nei F1 klasės atminties riba (384 MB). Kadangi atminties naudojimas eksperimentinėje aplikacijoje (vidutiniškai ~664 MiB, pikas ~1,1 GiB) paskirstomas tarp kelių instancijų, kiekvienos instancijos atminties sunaudojimas išlieka gerokai žemiau kritinio lygio. Šie rodikliai rodo, kad šifravimas neturėjo kritinio poveikio instancijų atminties naudojimui, o sistema išlieka stabili net esant didesnėms apkrovoms. [26]

Nors šis atminties ir instancijų skaičiaus skirtumas nėra kritinis ir sistema išlieka stabili, šie rezultatai pabrėžia šifravimo įtaką resursų sąnaudoms. Siekiant išlaikyti stabilų veikimą esant dar didesnėms apkrovoms, gali būti tikslinga peržiūrėti ir optimizuoti atminties valdymo bei instancijų paskirstymo strategijas. Tai galėtų padėti sumažinti šifravimo poveikį resursų naudojimui ir užtikrinti efektyvesnį sistemos našumą.

Išsiųsti ir gauti duomenys (angl. *GAE Application – Received, sent bytes*):

Duomenų srauto analizė parodė skirtingas tendencijas kontrolinėje ir eksperimentinėje aplinkose. Kontrolinėje aplikacijoje išsiunčiamų duomenų srautas stabilus (~100–150 KiB/s), vidutiniškai siekiantis 87,68 KiB/s. Gautų duomenų kiekis yra gerokai mažesnis (~5,86 KiB/s). Ši pusiausvyra rodo tipišką web aplikacijos elgesį, kai serveris dažniausiai siunčia daugiau duomenų klientui, nei jų gauna [97].

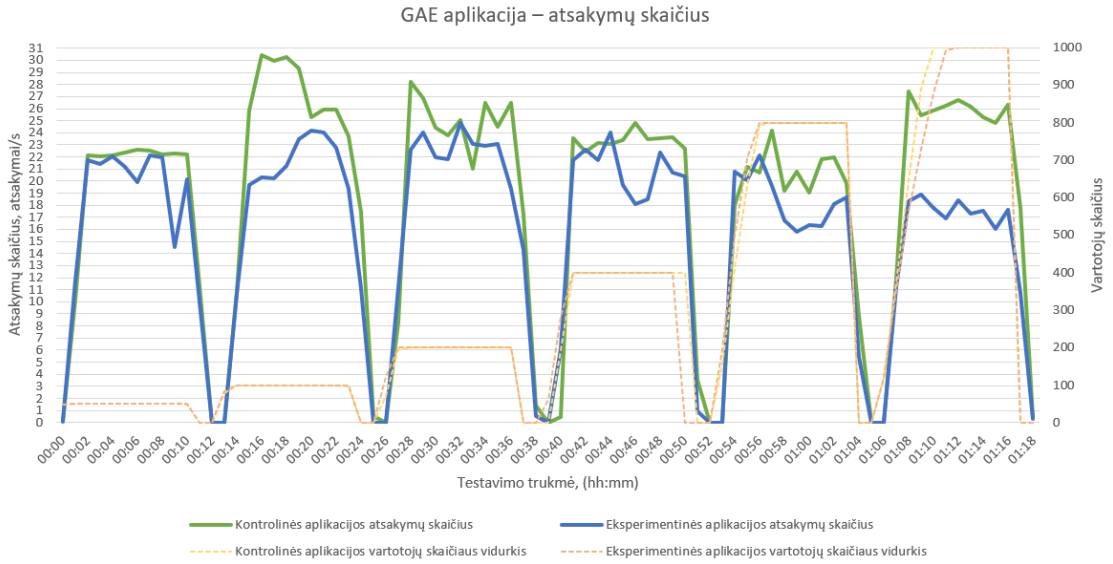
Eksperimentinėje aplikacijoje, nors būtų galima tikėtis, kad šifravimas gali padidinti siunčiamų duomenų kiekį (pvz., dėl papildomų metaduomenų ar šifruotų blokų dydžio [98], matome kitokią tendenciją. Išsiunčiamų duomenų srautas yra kiek mažesnis (80–110 KiB/s, vidutiniškai 68,81 KiB/s), apie 21 % mažesnis nei kontrolinėje. Gautų duomenų kiekis (~5,44 KiB/s) išlieka panašus. Šis rezultatas gali būti interpretuojamas kaip šifravimo procesų įtaka duomenų dydžiams. Vis dėlto ši hipotezė reikalautų papildomos analizės. Svarbu paminėti, kad šifravimo integracija nesukėlė neigiamo apkrovos antplūdžio duomenų srautams, bet veikiau asimetriškai sumažino išsiųstų duomenų apimtį.



Šaltinis: sudaryta autoriaus.

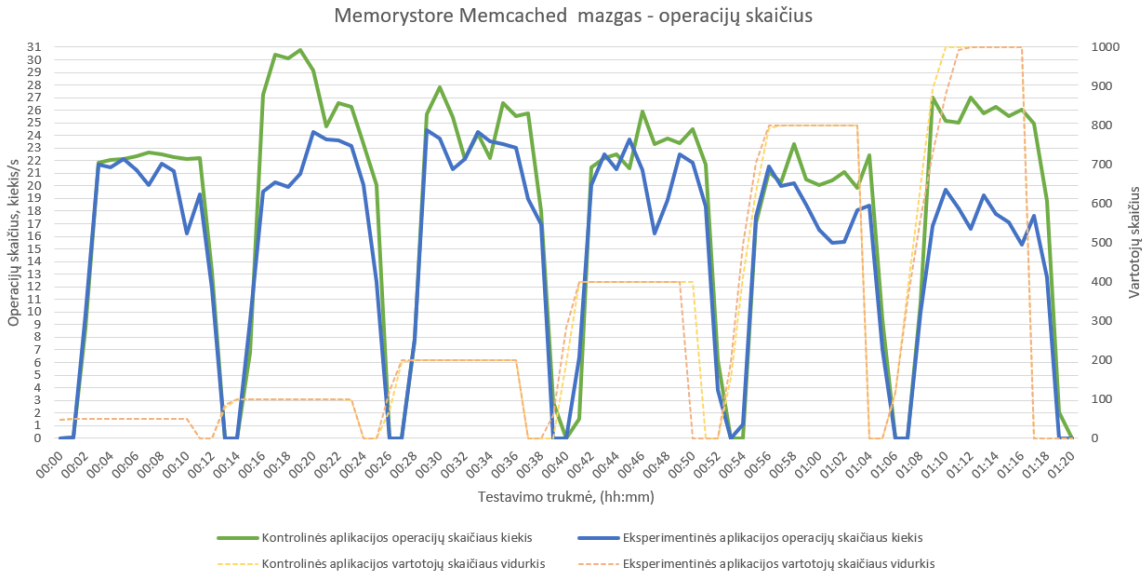
Atsakymų skaičius (angl. *GAE Application – response count*):

Atsakymų skaičius, t. y. per laiko vienetą apdorojamų užklausų dažnis, kontrolinėje versijoje svyruoja apie 20–30 atsakymų/s, vidutiniškai 21 atsakymas/s. Eksperimentinėje versijoje užfiksuoti kiek mažesni rodikliai (17–24 atsakymų/s, vidutiniškai 18 atsakymų/s). Šis nežymus nuokrypis koreliuoja su mažesniu išsiunčiamų duomenų kiekiu ir galimai lėtesne reagavimo dinamika. Vis dėlto skirtumas nėra reikšmingas ir gali būti paaiškinamas momentinėmis šifravimo operacijomis, kurios retkarčiais trumpam laikui lėtina atsako pateikimą.



Šaltinis: sudaryta autoriaus.

Operacijų skaičius (angl. *Memorystore Memcached node – operations count*):

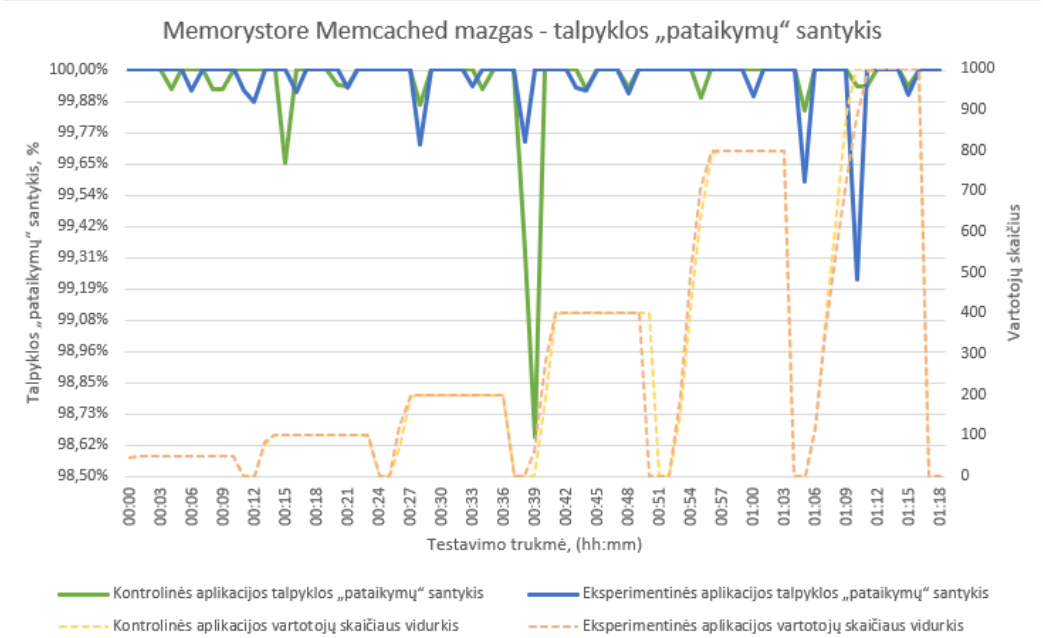


Šaltinis: sudaryta autoriaus.

Kontrolinėje sistemoje *Memcached* operacijų dažnis siekia ~20–30 operacijų/s, vidutiniškai 21 operaciją/s, o eksperimentinėje aplinkoje fiksuojama šiek tiek mažiau – 16–24/s, vidutiniškai 18 operacijų/s. Šis rezultatas rodo tam tikrą koreliaciją su sumažėjusiu atsakymų skaičiumi ir siunčiamų duomenų kiekiu eksperimentinėje aplinkoje. Gali būti, kad šifravimas įveda papildomą logiką, dėl kurios mažėja vienu metu vykstančių operacijų intensyvumas. Operacijų skaičiaus mažėjimas taip

pat gali atspindėti pakeistą užklausų struktūrą ar retesnį tam tikrų duomenų poreikį. Tačiau tai nėra faktai, kuriuos galime patvirtinti, tam reikalinga papildoma analizė.

Talpyklos "pataikymų" santykis (angl. *Memorystore Memcached node – hit ratio*):



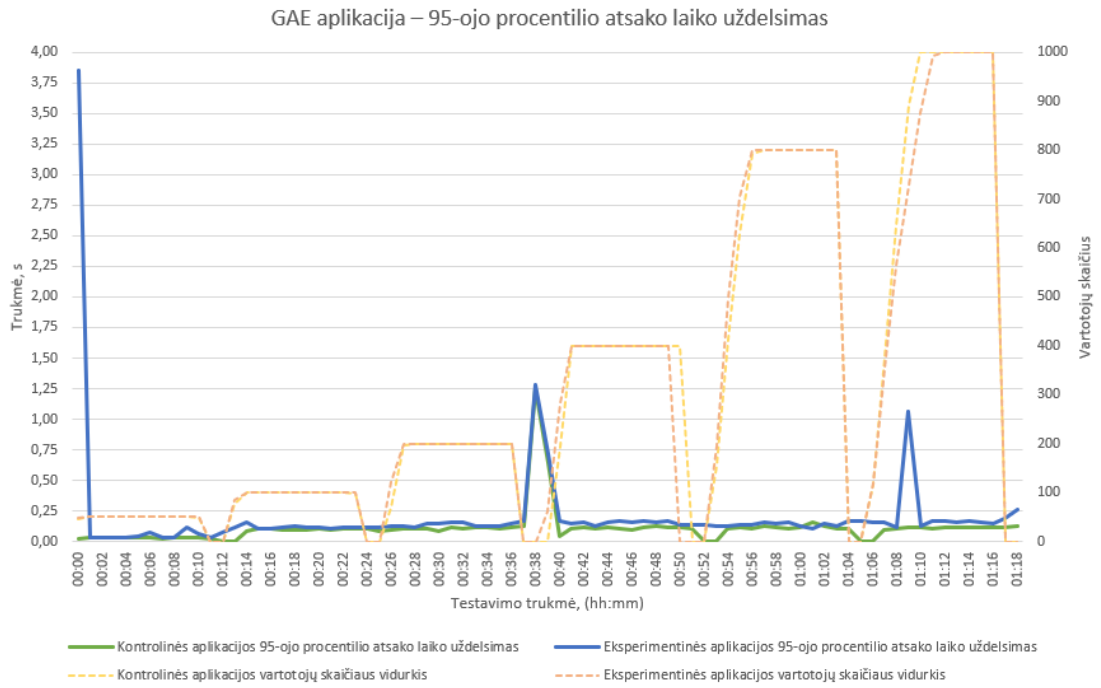
Šaltinis: sudaryta autoriaus.

Abiejose (kontrolinėje ir eksperimentinėje) sistemose fiksuojamas artimas 100 % *hit ratio*. Tai reiškia, kad beveik visos užklausos duomenų gaunamos tiesiai iš talpyklos, nereikalaujant papildomų užklausų į pagrindinę duomenų bazę [93].. Šifravimo įtraukimas, sprendžiant iš šios metrikos, neturi jokių neigiamų pasekmių talpyklos pasiekiamumui ar paieškos efektyvumui. Kitaip tariant, šifravimas nekeičia duomenų talpinimo ir paieškos mechanizmų efektyvumo, užtikrinant stabiliai gerą talpyklos panaudojimą.

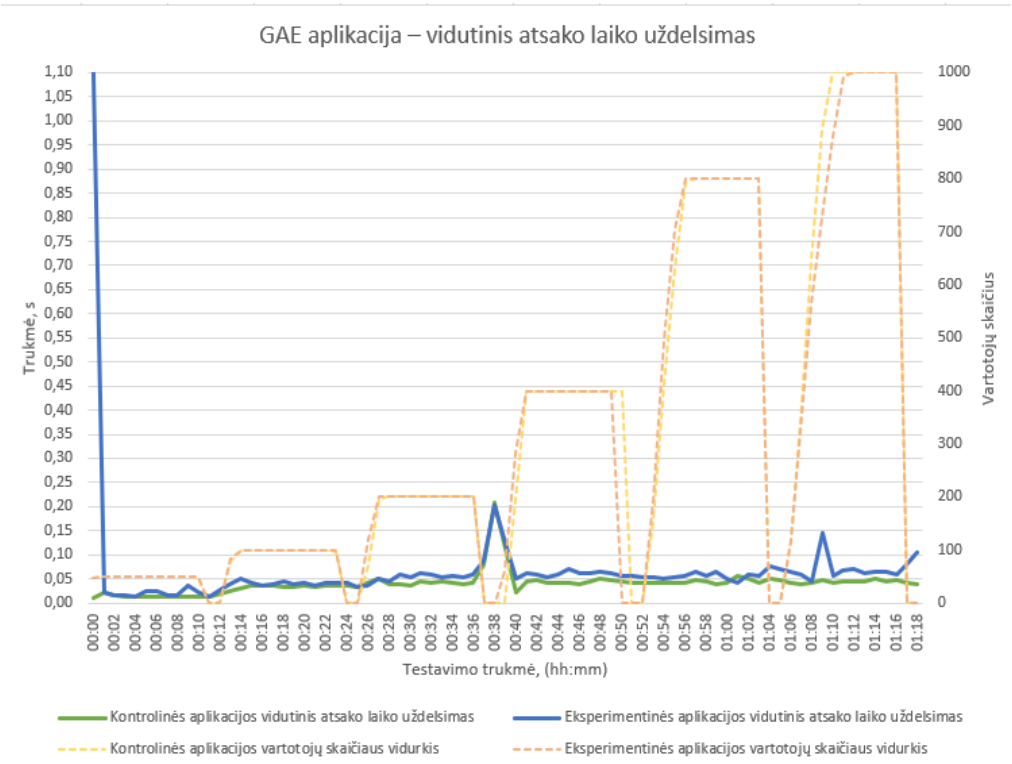
Atsako laiko uždelsimas (angl. *GAE Application – Response latency*):

Atsako laiko uždelsimo rodikliai yra itin svarbūs vertinant galutinę vartotojo patirtį. Kontrolinėje aplikacijoje vidutinis uždelsimas siekia ~0,04 s, o 95-asis procentilis yra 0,025–0,115 s, su vienu staigiu šuoliu iki 1,25 s. Eksperimentinėje aplinkoje vidutinis uždelsimas kiek didesnis (~0,05 s), o 95-asis procentilis – 0,1–0,16 s, su keliais šuoliais iki 1,28 s. Šie duomenys rodo, kad šifravimas gali turėti retkarčiais pasireiškiančią, bet ne itin didelę įtaką reagavimo greičiui. Nors

vidutinis ir procentilinis uždelsimas šiek tiek padidėja, pokyčiai nėra dideli ir neperžengia priimtinų vėlinimo ribų sparčiai veikiančiose *web* sistemose [99] .

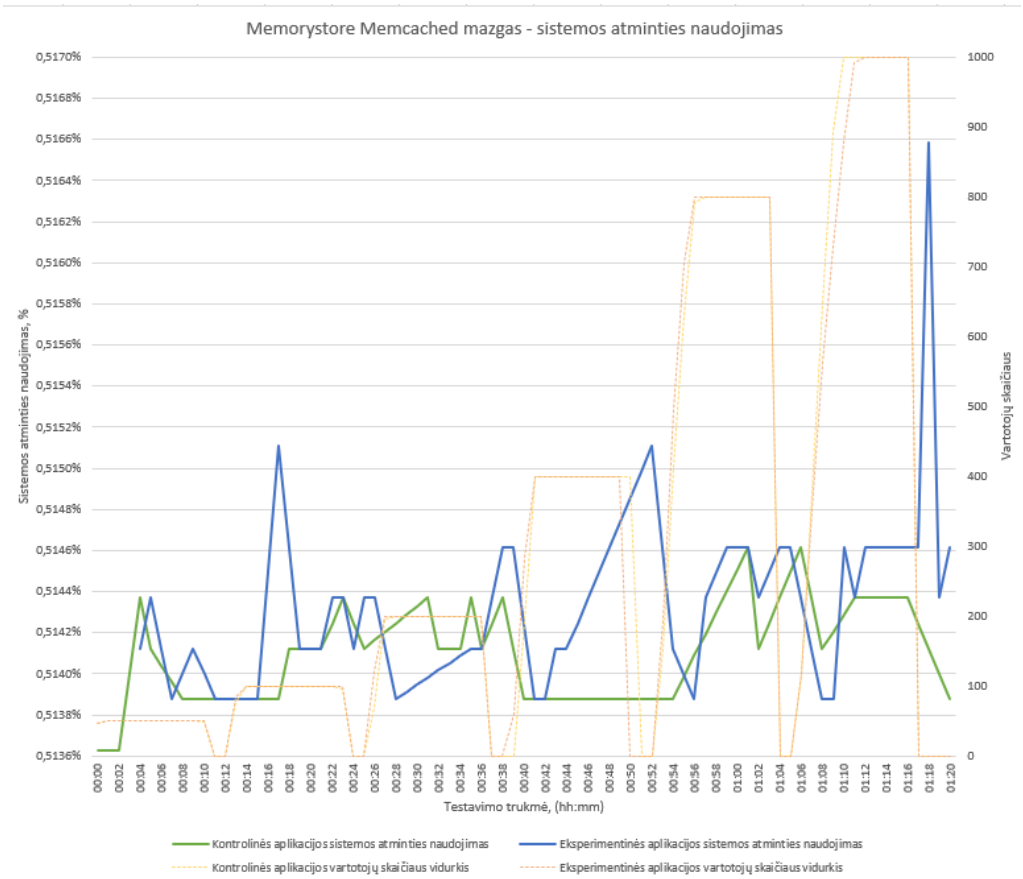


Šaltinis: sudaryta autoriaus.



Šaltinis: sudaryta autoriaus.

Sistemos atminties naudojimas (angl. *Memorystore Memcached node – system memory utilization*):



Šaltinis: sudaryta autoriaus.

Vertinant bendrą sistemos atminties panaudojimą *Memcached* mazge, kontrolinė versija svyruoja apie 0,5141–0,5143 %, o eksperimentinė – apie 0,5139–0,517 %. Šie skirtumai yra itin menki. Kitaip tariant, šifravimas neturi jokios reikšmingos įtakos bendrai *Memcached* mazgo atminties panaudojimo procentinei išraiškai. Stabilus atminties naudojimas *Memcached* kontekste pabrėžia, kad šifravimas nepakeičia bazinių talpyklos veikimo principų ar jos naudingumo.

16 PRIEDAS

Locust stats.csv failų suvestinės analizė autentifikacijos tyrimui testavimo su pertraukomis metu

Rezultatų suvestinėje pateikiami skirtingų vartotojų skaičiais paremtų testų rezultatai, įskaitant vidutinį ir atsako laiko medianą, didžiausią atsako laiką, užklausų per sekundę skaičių, klaidų dažnį bei atsako laikus pagal 90%, 95% ir 99% procentilius.

Vartotojų skaičius	Aplikacijos tipas	Vidutinis atsako laikas, ms	Atsako laiko mediana, ms	Didžiausias atsako laikas, ms	Užklausų per sekundę	Klaidų dažnis (%)	90% atsako laikas (ms)	95% atsako laikas (ms)	99% atsako laikas (ms)
50	Kontrolinė	273,04	260	1567,48	8,60	0	410	440	530
50	Eksperimentinė	325,23	280	2134,88	8,39	0	520	590	980
100	Kontrolinė	291,53	270	2313,07	17,01	0	430	460	560
100	Eksperimentinė	388,07	330	2813,04	16,48	0	630	750	1200
200	Kontrolinė	335,55	310	1857,52	33,82	0	510	570	700
200	Eksperimentinė	480,04	420	4380,99	32,52	0	780	920	1500
400	Kontrolinė	358,21	330	3556,83	66,66	0	530	590	750
400	Eksperimentinė	559,32	480	8463,6	64,53	0	910	1100	1800
800	Kontrolinė	629,95	490	3785,63	117,97	0	1300	1600	2100
800	Eksperimentinė	840,04	660	6584,95	115,14	0	1600	2000	2600
1000	Kontrolinė	4163,03	3900	41058,26	86,17	0	7200	8100	10000
1000	Eksperimentinė	4000,48	3800	41182,49	87,93	0,0057	6800	7500	9700

Šaltinis: sudaryta autoriaus.

Iš lentelėje pateiktų duomenų matyti, kad eksperimentinėje aplikacijoje vidutinis atsako laikas visuose apkrovos taškuose buvo didesnis, išskyrus prie 1000 vartotojų, kur jis buvo mažesnis nei kontrolinėje (4000,48 ms ir 4163,03 ms). Prie mažų apkrovų, pvz., 50 vartotojų, kontrolinė aplikacija užfiksavo 273,04 ms, o eksperimentinė – 325,23 ms (padidėjimas 19%). Augant apkrovai, eksperimentinės aplikacijos atsako laikas padidėjo sparčiau, tikėtina, dėl papildomo *Firebase Authentication* skaičiavimo.

Eksperimentinės aplikacijos atsako laiko mediana buvo reikšmingai didesnė esant mažoms ir vidutinėms apkrovoms. Pvz., esant 800 vartotojų, eksperimentinė aplikacija užfiksavo 660 ms, o kontrolinė – 490 ms (padidėjimas 34,69%). Esant 1000 vartotojų apkrovai, atsako laiko mediana eksperimentinėje aplikacijoje sumažėjo iki 3800 ms, palyginti su 3900 ms kontrolinėje, rodant infrastruktūros prisitaikymą.

Eksperimentinėje aplikacijoje didžiausias atsako laikas buvo didesnis visose apkrovos situacijose, išskyrus prie 1000 vartotojų, kur skirtumas buvo minimalus (41182,49 ms ir 41058,26 ms). Pvz., esant 50 vartotojų, kontrolinė aplikacija užfiksavo 1567,48 ms, o eksperimentinė – 2134,88 ms (36% padidėjimas). Šie rezultatai rodo, kad autentifikacija prisideda prie ekstremalaus uždelsimo.

Eksperimentinėje aplikacijoje užklausų apdorojimo sparta buvo šiek tiek mažesnė nei kontrolinėje, ypač esant didelėms apkrovoms. Pvz., esant 1000 vartotojų, eksperimentinė aplikacija apdorodavo 87,93 užklausos per sekundę, o kontrolinė – 86,17 (2% padidėjimas). Šis skirtumas minimalus ir neleidžia daryti reikšmingų išvadų.

Kontrolinėje aplikacijoje klaidų dažnis buvo 0% visuose apkrovos lygiuose, rodo sistemos stabilumą. Eksperimentinėje aplikacijoje pastebėta minimali klaidų dalis (0,0057%) prie 1000 vartotojų, o tai rodo, kad „Firebase Authentication“ turi nedidelį, bet pastebimą poveikį sistemos patikimumui.

Eksperimentinėje aplikacijoje procentiliniai atsako laikai dažniausiai buvo didesni, išskyrus prie 1000 vartotojų apkrovos, kur rezultatai buvo mažesni nei kontrolinėje. Pvz., ties 1000 vartotojų:

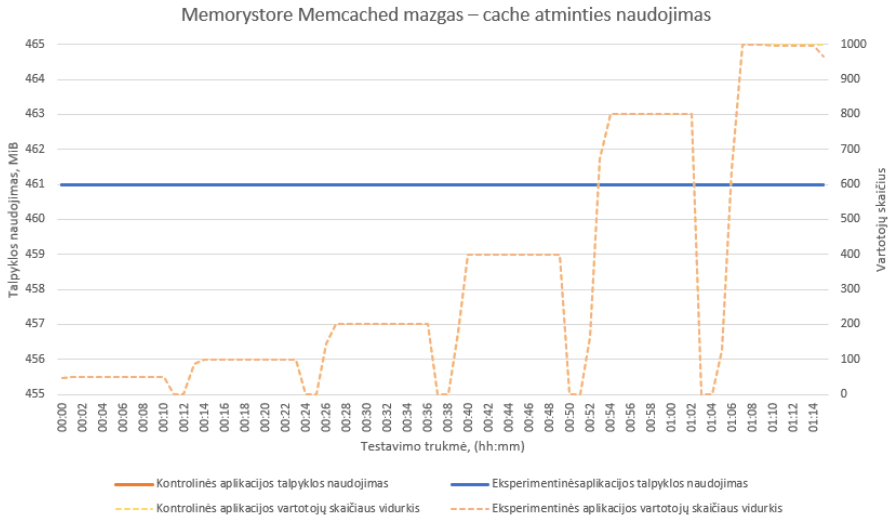
- 90% atsako laikas: 6800 ms (kontrolinėje – 7200 ms, sumažėjimas 5,56%).
- 95% atsako laikas: 7500 ms (kontrolinėje – 8100 ms, sumažėjimas 7,41%).
- 99% atsako laikas: 9700 ms (kontrolinėje – 10000 ms, sumažėjimas 3%).

Šie duomenys rodo, kad autentifikacija gali pagerinti sistemos atsako laikus esant maksimalioms apkrovoms dėl optimalaus resursų paskirstymo.

17 PRIEDAS

Google Cloud Console Monitoring grafikų analizė (autentifikacijos poveikis testavimo su pertraukomis metu)

Talpyklos atminties naudojimas (angl. *Memorystore Memcached node – cache memory usage*):



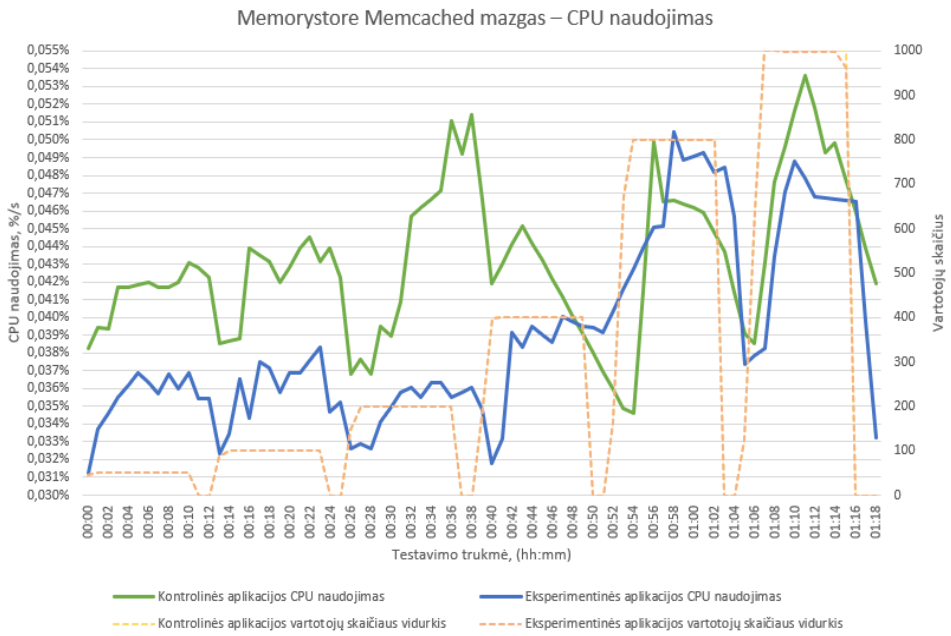
Šaltinis: sudaryta autoriaus.

Kontrolinės ir eksperimentinės aplikacijų talpyklos atminties naudojimo rodikliai išlieka stabilūs, nepaisant didėjančio vartotojų skaičiaus. Net padidėjus užklausų apkrovai, talpyklos panaudojimo lygis nekeičia savo dinamikos. Tai reiškia, kad autentifikacijos integravimas iš esmės nesukuria papildomų duomenų talpinimo reikalavimų *Memcached* aplinkoje. Kadangi talpyklos funkcija yra greitai pateikti jau paruoštą informaciją, stabilus atminties naudojimas rodo, kad autentifikacijos procesas nepakeičia duomenų talpinimo ar jų užklausų struktūros taip, kad prireiktų daugiau talpyklos erdvės. Iš teorinės perspektyvos, autentifikacijos informacija nėra saugoma išoriniuose talpyklų sprendimuose, todėl ir turime nepakitusius talpyklos atminties naudojimo rodiklius [100].

Procesoriaus naudojimas (angl. *Memorystore Memcached Node – CPU usage percent*):

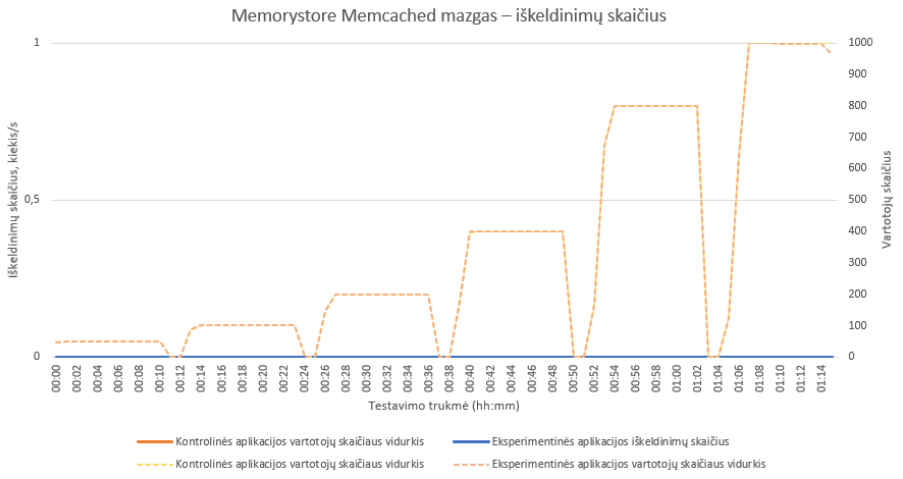
Abiejų (kontrolinės ir eksperimentinės) sistemų atvejais *Memcached* CPU apkrova išlieka itin maža. Pastebėta, kad eksperimentinėje aplikacijoje, kurioje integruotas *Firebase* autentifikavimas, CPU apkrova yra net šiek tiek mažesnė (CPU naudojimo vidurkis kontrolinėje sistemoje – 0,043%/s, eksperimentinėje – 0,038%/s). Šį netikėtą rezultatą galima paaiškinti tuo, kad *Firebase*

autentifikavimo procesai atliekami už aplikacijos ribų, naudojant *Firebase* serverių infrastruktūrą, kaip nurodo oficiali *Firebase* dokumentacija [100]. Dėl to autentifikavimo sukelta apkrova neatsispindi GAE aplinkoje, o *Memcached* gali aptarnauti labiau vienarūšius užklausų tipus, optimizuojant resursų paskirstymą. Taigi, *Firebase* integracija ne tik neturi neigiamo poveikio CPU našumui, bet ir gali prisidėti prie bendro sistemos efektyvumo gerinimo.



Šaltinis: sudaryta autoriaus.

Talpyklos iškeldinimų skaičius (angl. *Memorystore Memcached node – eviction count*):

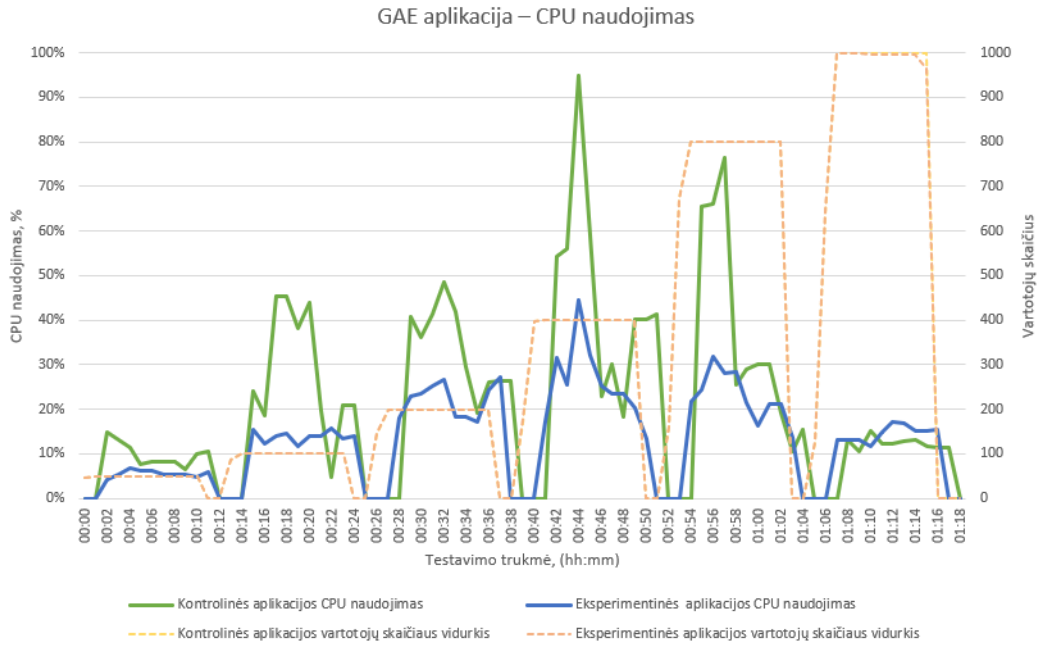


Šaltinis: sudaryta autoriaus.

Abiejose aplinkose neužfiksuota iškeldinimų (angl. *evictions*), o tai reiškia, jog talpyklos dydis ir valdymo politika yra tinkamai sukonfigūruota. Kadangi autentifikacija nepadidina

iškeldinimų skaičiaus, galima teigti, kad šis mechanizmas nepakeičia talpyklos darbo principų ir nenaudoja jos taip intensyviai, kad talpykloje pritrūktų vietos dažnai naudojamiems duomenims. Iškeldinimų nebuvimas rodo optimalią talpyklos pusiausvyrą ir išvengiamą papildomą apkrovą pagrindinei duomenų sistemai, užtikrinant greitą duomenų prieigą bei stabilų talpyklos elgesį.

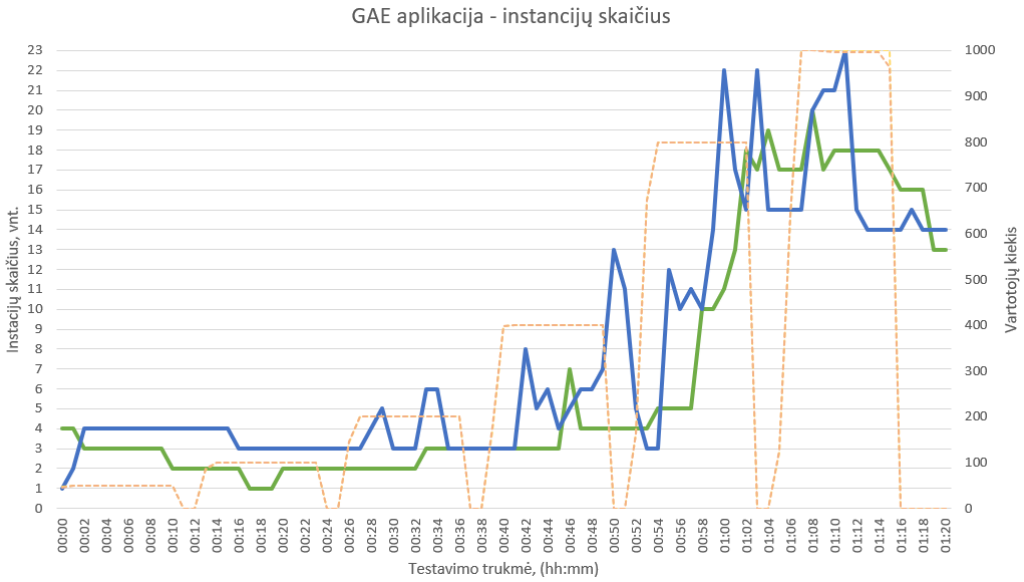
CPU naudojimas GAE aplinkoje (angl. *GAE Application – CPU utilization*):



Šaltinis: sudaryta autoriaus.

Kontrolinės aplikacijos atveju stebimi trumpi, bet ryškūs CPU apkrovos šuoliai (pvz., prie 400 vartotojų apkrovos CPU apkrova pasiekia beveik 100%). Tokie šuoliai gali atsirasti dėl „šaltojo starto“ efekto. Priešingai, eksperimentinėje aplikacijoje CPU apkrova yra tolygesnė ir žemesnė (nuo 15% iki 30%, o 400 vartotojų apkrovos metu pasiekia 45%). *Firebase* autentifikacija, pagal oficialią dokumentaciją, atlieka autentifikavimo procesus *Firebase* serverių infrastruktūroje, o tai gali sumažinti papildomą GAE apkrovą. Be to, reguliarios žetonų tikrinimo operacijos gali stabilizuoti resursų naudojimą, amortizuodamos staigius apkrovos šuolius. Šie procesai galimai yra optimizuoti naudojant *Firebase* mechanizmus, tokius kaip vietinė talpykla ar efektyvūs tinklo užklausų valdymo algoritmai, todėl CPU apkrova išlieka tolygesnė ir mažiau jautri vartotojų skaičiaus pokyčiams. Toks stabilumas yra svarbus užtikrinant aplikacijos patikimumą didelio masto sistemose. [100]

Instancijų skaičius GAE aplinkoje (angl. *GAE Application – instance count*):



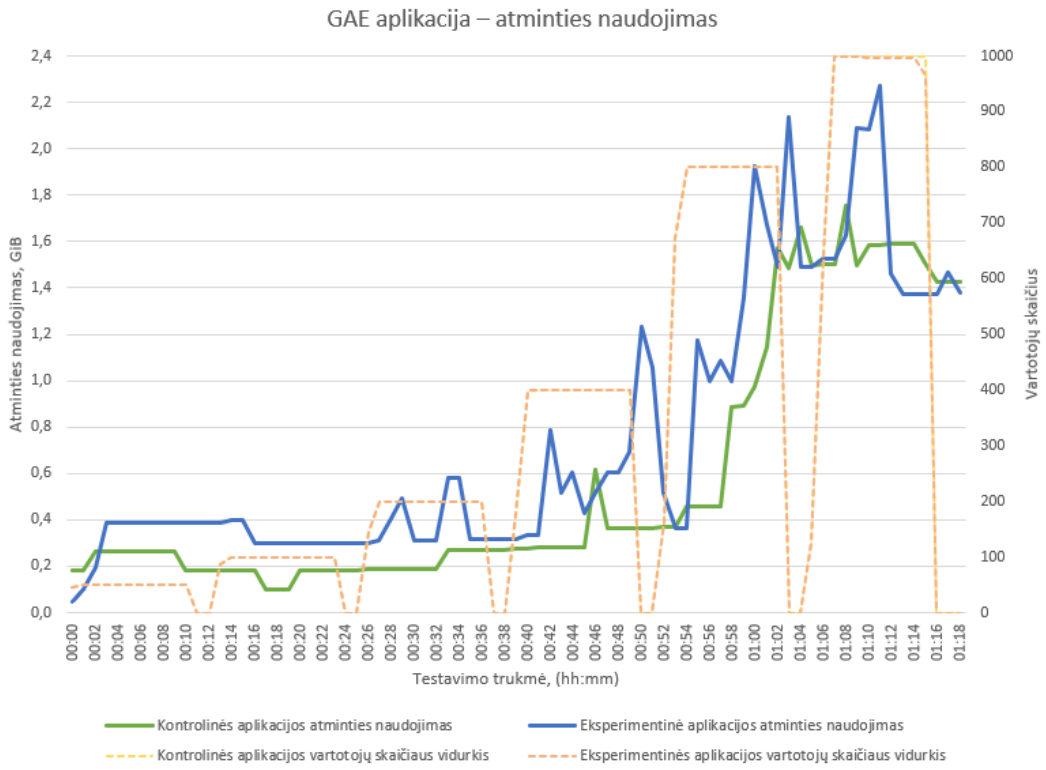
Šaltinis: sudaryta autoriaus.

Kontrolinė aplikacija pasižymi mažesniais instancijų skaičiaus svyravimais per visą testavimo laikotarpį, palyginti su eksperimentine aplikacija. Pradiniame testavimo etape instancijų skaičius abiejose aplikacijose yra stabilus ir mažas, tačiau eksperimentinės aplikacijos instancijų skaičius pradeda augti anksčiau ir sparčiau reaguodamas į didėjančią apkrovą. Eksperimentinėje aplikacijoje stebimas dažnesnis instancijų skaičiaus augimas ir mažėjimas, o tai rodo intensyvesnį resursų panaudojimą. Kontrolinėje aplikacijoje instancijų skaičiaus pokyčiai vyksta rečiau ir yra mažesnio masto, tačiau abi aplikacijos reaguoja į vartotojų skaičiaus pokyčius, ypač didžiausios apkrovos laikotarpiais. Didžiausias instancijų skaičius kontrolinėje aplikacijoje buvo lygus 20, o eksperimentinėje – 23. Testavimo pabaigoje abiejų aplikacijų instancijų skaičius mažėja, sekdamas vartotojų apkrovos mažėjimo tendenciją. Tai rodo, kad abi aplikacijos dinamiškai prisitaiko prie besikeičiančių apkrovos sąlygų, tačiau jų elgesys skirtingas pagal instancijų skaičiaus pokyčių intensyvumą ir amplitudę.

Atminties naudojimas GAE aplinkoje (angl. *GAE Application – memory usage*):

Kontrolinėje aplikacijoje atminties naudojimas didėja proporcingai apkrovai, nėra staigių šuolių. Maksimalus atminties pasiekimas (~1,8 GiB) parodo, kad augant vartotojų skaičiui, tiesiogiai didėja reikalingų duomenų (sesijų, užklausų konteksto) kiekis. Eksperimentinėje aplikacijoje atminties naudojimas ne tik didesnis vidutiniškai (~0,8 GiB lyginant su 0,6 GiB), bet ir pasižymi didesniais svyravimais - maksimalus atminties pasiekimas (~2,27 GiB) Tikėtina, kad papildoma

autentifikacijos logika (pvz., žetonų validavimas, naudotojų sesijų informacijos saugojimas atmintyje) padidina momentinių atminties regionų poreikį [101]. Šie svyravimai gali kilti dėl asinchroninio token‘ų atnaujinimo, laikino rezultatų kešavimo, ar nereguliarių I/O operacijų su trečiųjų šalių tarnybomis, tokiomis kaip *Firebase Authentication backend* [101] [102]. Nors tai nėra kritiškas atminties panaudojimo padidėjimas, jis rodo, kad autentifikacija yra šiek tiek daugiau atminties reikalaujantis komponentas, lyginant su baziniu aplikacijos funkcionalumu.

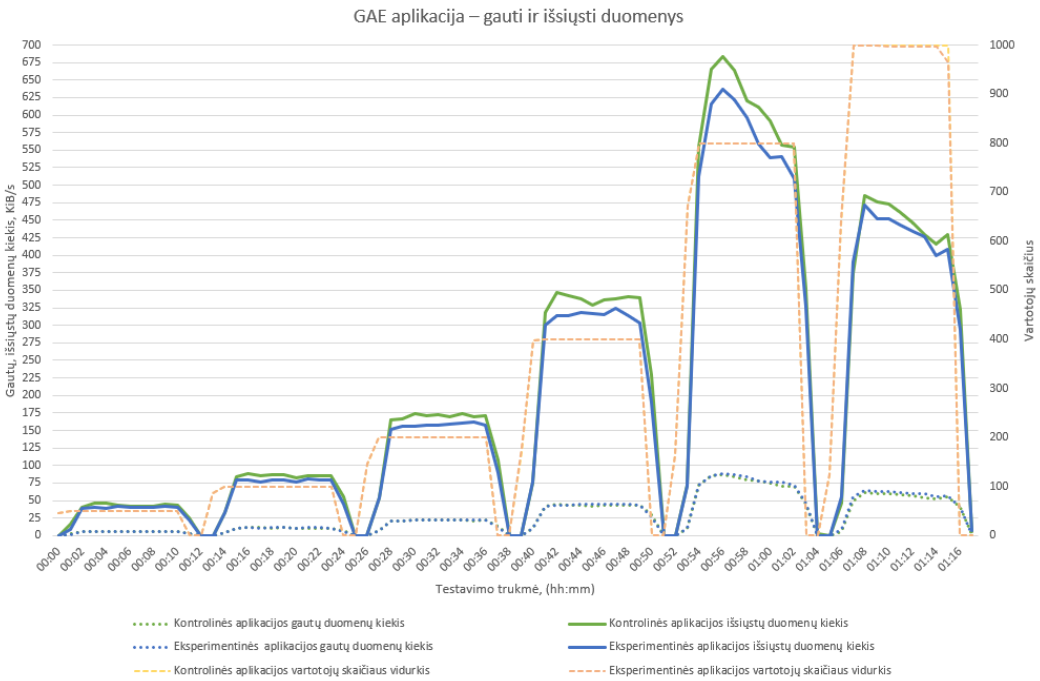


Šaltinis: sudaryta autoriaus.

Pagal GAE standartinės aplinkos dokumentaciją, kiekvienai instancijai priskiriama tam tikra atminties riba, kuri priklauso nuo naudojamos instancijos klasės. Pavyzdžiui, F1 klasės instancijai skiriama 384 MB. Kontrolinėje aplikacijoje, kai buvo pasiektas maksimalus 1,8 GiB atminties panaudojimas, veikė 20 instancijų, todėl vidutinis vienos instancijos atminties naudojimas buvo ~92 MB (1,8 GiB ÷ 20). Eksperimentinėje aplikacijoje, kai buvo pasiektas maksimalus 2,27 GiB atminties panaudojimas, veikė 23 instancijos, o vidutinis vienos instancijos atminties naudojimas siekė ~98,7 MB (2,27 GiB ÷ 23). Šie skaičiavimai rodo, kad abiem atvejais instancijų vidutinis atminties naudojimas išliko žemiau standartinės aplinkos F1 klasės atminties limito (384 MB). Tai reiškia, kad nei kontrolinė, nei eksperimentinė aplikacija neperžengė GAE standartinės aplinkos nustatytų atminties apribojimų. Tačiau eksperimentinė aplikacija naudojo daugiau atminties dėl papildomos autentifikacijos logikos, kuri turėjo įtakos momentiniams atminties poreikių svyravimams. Instancijų skaičiaus padidėjimas tiesiogiai koreliuoja su atminties naudojimo augimu, nes kiekviena papildoma

instancija užtikrina didesnį bendrą atminties prieinamumą, reikalingą augančiai apkrovai palaikyti. [26] [103]

Išsiųsti ir gauti duomenys (angl. *GAE Application – Received, sent bytes*):



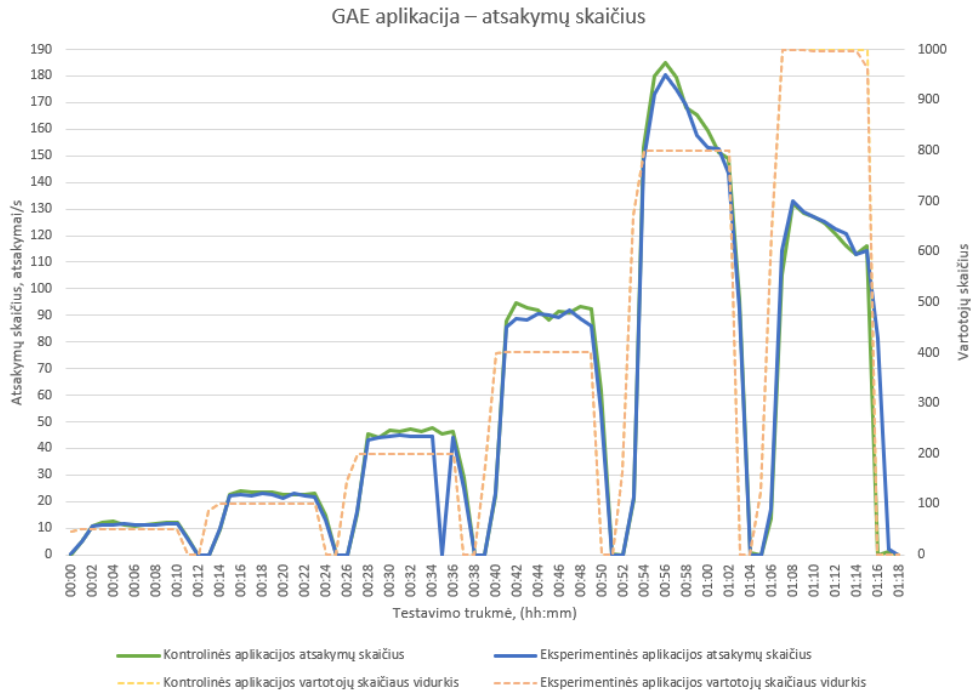
Šaltinis: sudaryta autoriaus.

Tiek be autentifikacijos, tiek su ja, tinklo srauto diagramos labai panašios: didėjant vartotojų skaičiui, išsiunčiamų ir gaunamų duomenų kiekis taip pat didėja, pasiekdamas piką prie 800 vartotojų, o vėliau, esant 1000 vartotojų, sumažėja. Ši tendencija rodo, kad užklausų ir atsakymų apimtis tiesiogiai priklauso nuo apkrovos, tačiau integruota autentifikacija nedidina tinklo srauto.

Atsakymų skaičius (angl. *GAE Application – response count*):

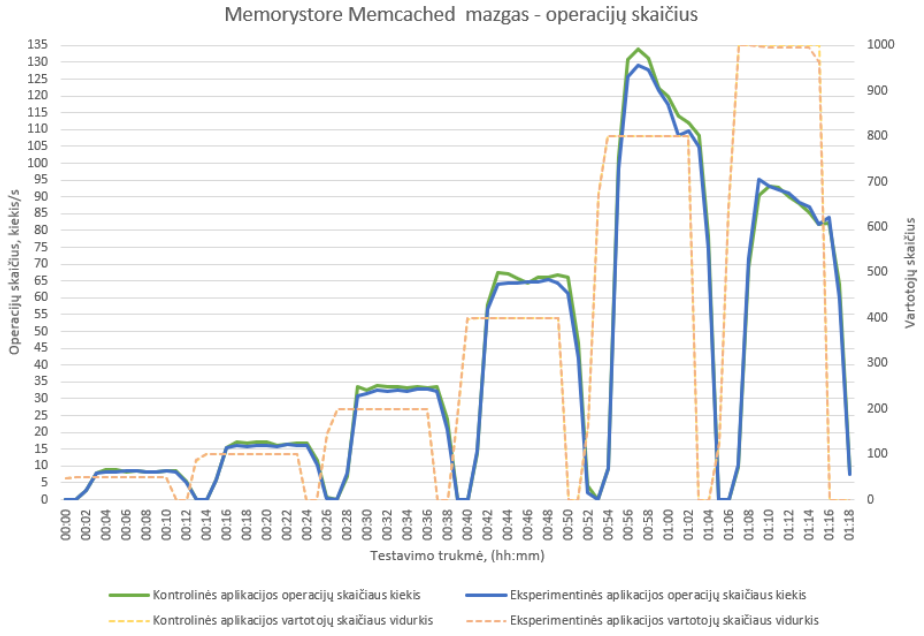
Abiejose sistemose atsakymų skaičius proporcingai auga didėjant vartotojų skaičiui iki 800 vartotojų, o vėliau, pasiekus 1000, sumažėja. Tai rodo, kad egzistuoja tam tikros pralaidumo ribos, kurioms esant sistema nebepajėgia apdoroti daugiau užklausų ir šis rodiklis pradeda mažėti. Svarbiausia, kad autentifikacija neturi esminės įtakos šiai dinamikai: eksperimentinė aplikacija generuoja labai panašią atsakymų kreivę, kaip ir kontrolinė. Tai leidžia daryti išvadą, kad

autentifikacijos procesai nėra esminis našumą ribojantis faktorius, trukdantis sistemai atsakyti į didesnį užklausų srautą.



Šaltinis: sudaryta autoriaus.

Operacijų skaičius (angl. *Memorystore Memcached node – operations count*):



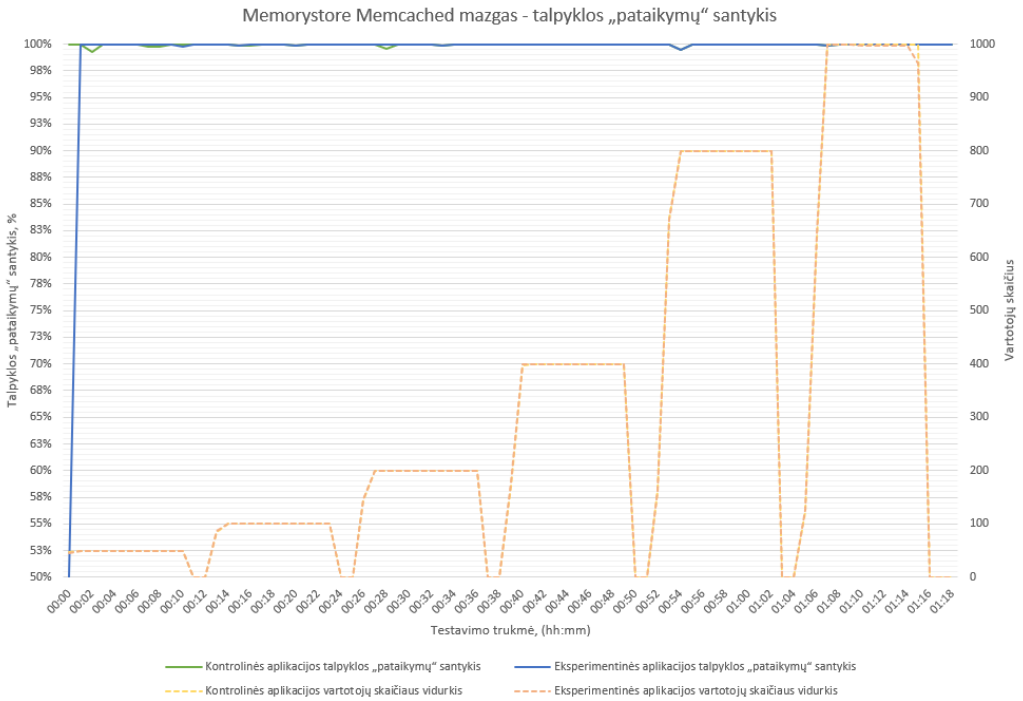
Šaltinis: sudaryta autoriaus.

Memcached operacijų skaičius yra tiesiogiai susijęs su užklausų kiekiu ir intensyvumu. Abu scenarijai (kontrolinis ir eksperimentinis) rodo identišką tendenciją: didėjant apkrovai didėja ir

operacijų skaičius iki 800 vartotojų, o prie 1000 vartotojų, operacijų kiekis sumažėja. Eksperimentinėje aplinkoje operacijų skaičius labai panašus į kontrolinės aplinkos rodiklius. Tai reiškia, kad autentifikacija, kaip papildomas funkcionalumas, nesiūlo naujo ir kitokio duomenų naudojimo scenarijaus talpyklai. Taigi, sistemos elgsena lieka tokia pati – operacijų dažnis didele dalimi priklauso nuo vartotojų srauto, o ne nuo autentifikacijos mechanizmo.

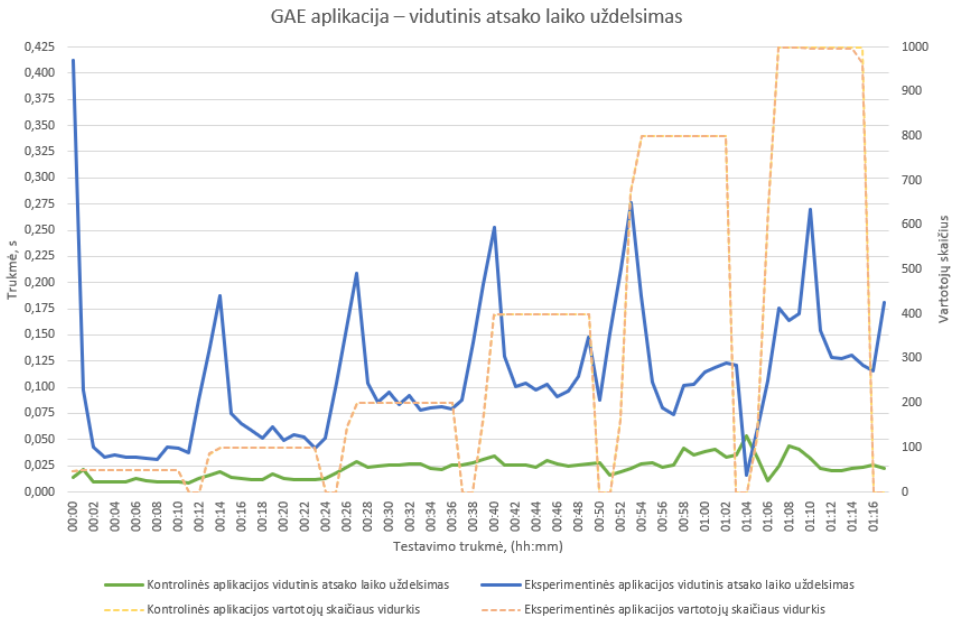
Talpyklos „pataikymų“ santykis (angl. *Memorystore Memcached node – hit ratio*):

Abiejose sistemose „pataikymų“ santykis išlieka itin aukštas ir artimas 100%. Tai svarbus rodiklis, rodantis, kad didžioji dalis užklausų į talpyklą sėkmingai aptarnaujama iš esamų duomenų be poreikio daryti papildomas užklausas į pagrindinę duomenų bazę. Autentifikacijos integracija nesumažino šio rodiklio, todėl galima teigti, kad papildomos autentifikacijos logikos duomenų poreikiai yra patenkinami iš talpyklos su tokiu pačiu efektyvumo lygiu, kaip ir neautentifikuotoje aplinkoje. Tokia išvada rodo gerą talpyklos valdymo strategiją ir stabilų darbo pobūdį, nepaisant funkcionalumo išplėtimo.



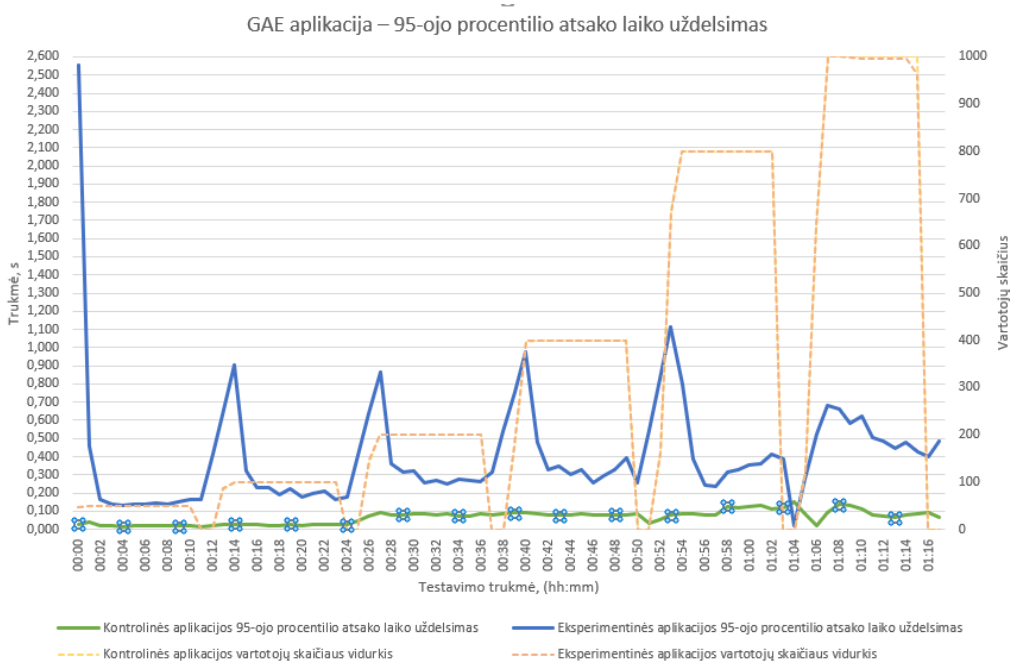
Šaltinis: sudaryta autoriaus.

Atsako laiko uždelsimas (angl. *GAE Application – Response latency*):



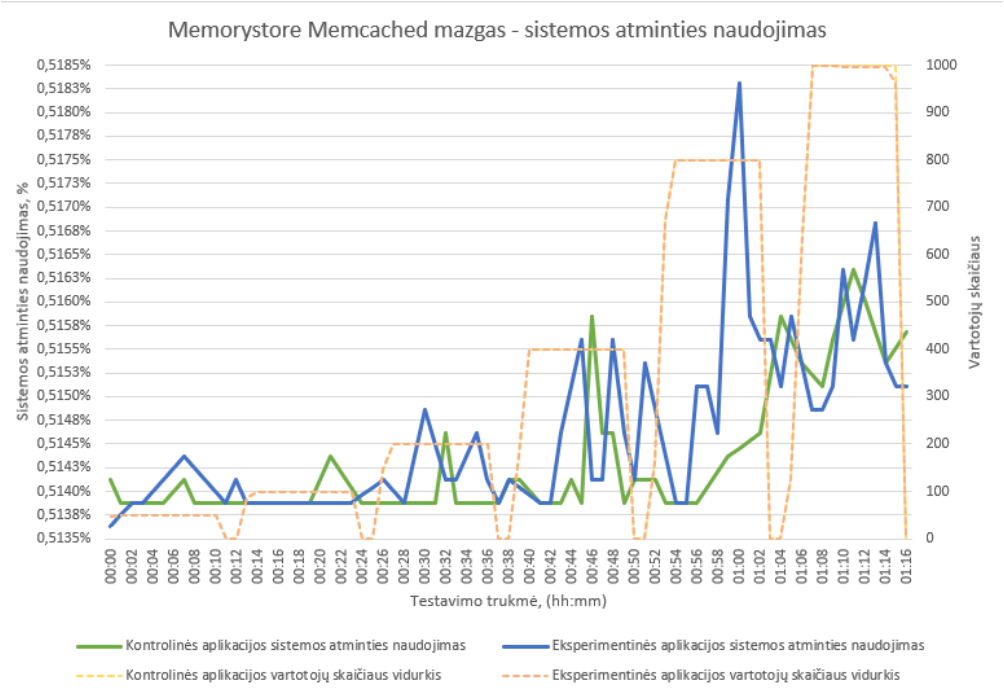
Šaltinis: sudaryta autoriaus.

Atsako laiko uždelsimas (tiek vidutinis, tiek 95-ojo procentilio) yra pagrindinis rodiklis, vertinant galutinę vartotojo patirtį. Kontrolinėje aplikacijoje uždelsimas didėja su apkrova gana nuspėjamai (nuo ~25–150 ms 95% procentiliui), išlaikant sąlyginai stabilų, žemą lygį. Tuo tarpu eksperimentinėje aplinkoje atsako laiko vėlinimas yra didesnis ir nestabilesnis, kartais siekia net 1 s 95-ojo procentilio atveju, nepaisant tiesioginio ryšio su apkrovos lygiu. Kontrolinės aplikacijos 95-ojo procentilio vidurkis yra 0,07s, vidutinio – 0,023s. Eksperimentinės aplikacijos 95-ojo procentilio vidurkis yra 0,357s, vidutinio – 0,101s. Eksperimentinėje aplikacijoje atsako laiko vėlinimo nestabilumas ir ilgesnis jo laikas, gali būti siejamas su „Firebase Authentication“ ID žetonų validavimo procesu, įgyvendinamu per *Firebase REST API*. Pagal *Firebase* dokumentaciją, prisijungus vartotojui, sukuriamas ID žetonas, kuris siunčiamas į serverį per HTTPS užklausą. Serveris naudoja *Firebase Admin SDK*, kad patikrintų žetono galiojimą, autentiškumą ir pasirašymą. Analizuojamoje aplikacijoje šis procesas reikalauja papildomų tinklo užklausų, kurios gali padidinti atsako laiką dėl tinklo ryšio vėlavimų ar išorinių paslaugų prieinamumo. Nors aplikacija tiesiogiai nenaudoja trečiųjų šalių viešųjų raktų, *Firebase* automatiniai procesai naudoja viešuosius raktus žetonų patikrai, o tai taip pat gali prisidėti prie tinklo delsos, jei raktai nėra iš karto prieinami. [104] Tokiu būdu papildomos saugumo funkcijos, tokios kaip autentifikacija, gali padidinti atsako laiką ir sukelti reaktyvumo netolygumą, ypač esant didesnei apkrovai ar nestabiliai tinklo infrastruktūrai.



Šaltinis: sudaryta autoriaus.

Sistemos atminties naudojimas (angl. *Memorystore Memcached node – system memory utilization*):



Šaltinis: sudaryta autoriaus.

Sistemos atminties išteklių procentinis naudojimas išlieka labai panašus abiejose aplinkose. Nors eksperimentinėje aplinkoje matoma daugiau nedidelių svyravimų, bendras vidurkis žymiai nesiskiria. Tai rodo, kad nors eksperimentinė aplinka (su autentifikacija) bendroje GAE aplinkoje

naudoja daugiau atminties, „Memcached node“ lygmenyje situacija beveik nesikeičia. Kitaip tariant, daugiau atminties sunaudojama aplikacijos logikos lygmenyje, o ne talpyklos mazge.

18 PRIEDAS

Locust stats.csv failų suvestinės ir *stats_history.csv* duomenų grafiko analizė šifravimo tyrimui testavimo be pertraukų metu

Vartotojų skaičius	Aplikacijos tipas	Vidutinis atsako laikas, ms	Atsako laiko mediana, ms	Didžiausias atsako laikas, ms	Užklausų per sekundę	Klaidų dažnis (%)	90% atsako laikas (ms)	95% atsako laikas (ms)	99% atsako laikas (ms)
Bendras	Kontrolinė	8751,09	5600	544259,39	23,52	2	13000	15000	87000
Bendras	Eksperimentinė	15568,98	8200	837909,46	15,00	3	17000	21000	245000

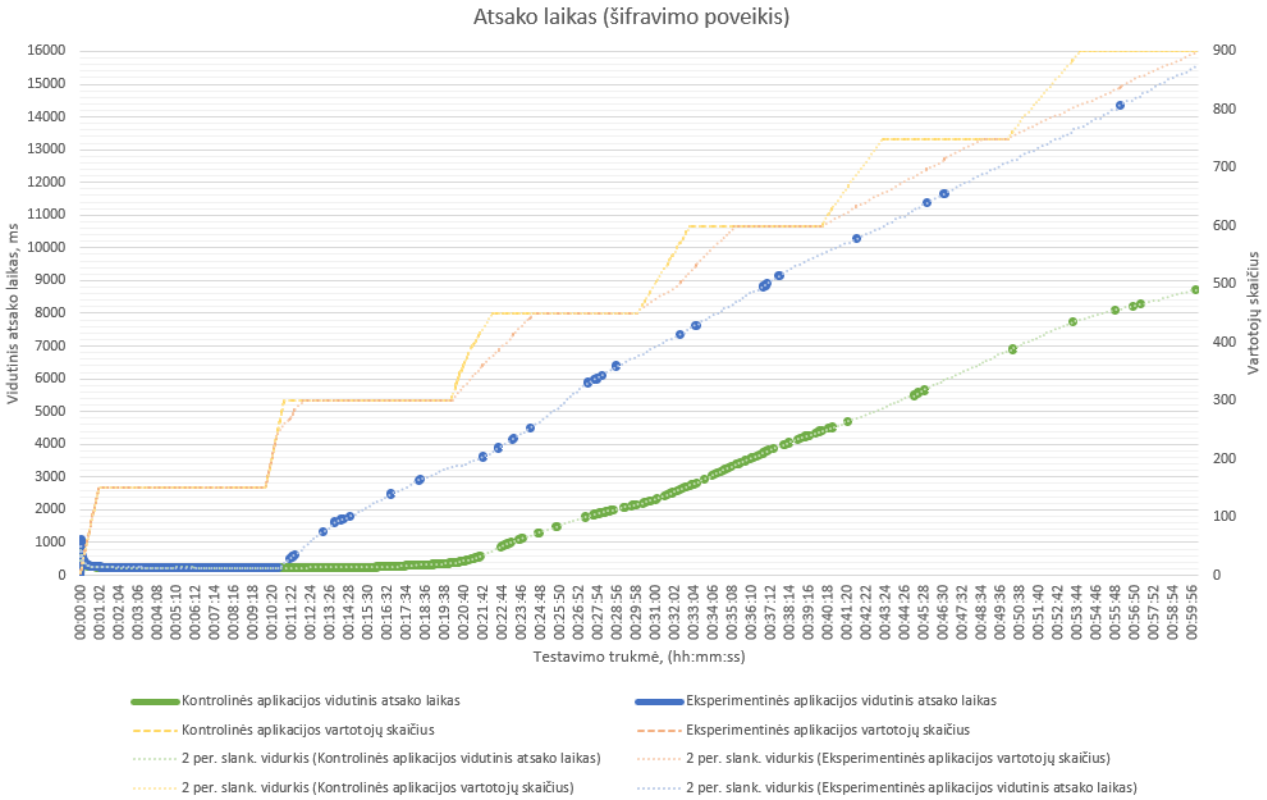
Šaltinis: sudaryta autoriaus.

Remiantis suvestinės duomenimis, galima daryti šias išvadas apie testuotų aplikacijų našumą ir stabilumą. Kontrolinės aplikacijos vidutinis atsako laikas siekia 8751,09 ms, o tai rodo, kad aplikacija geba užtikrinti spartesnį užklausų apdorojimą nei eksperimentinė aplikacija. Eksperimentinės aplikacijos vidutinis atsako laikas, siekiantis 15568,98 ms, beveik dvigubai viršija kontrolinės aplikacijos laiką, o tai akivaizdžiai parodo šifravimo proceso poveikį sistemos našumui.

Atsako laiko mediana kontrolinėje aplikacijoje yra 5600 ms, o eksperimentinėje – 8200 ms. Šie duomenys rodo, kad dauguma užklausų kontrolinėje aplikacijoje buvo apdorotos greičiau, tačiau šifravimo procesas eksperimentinėje aplikacijoje reikšmingai prisidėjo prie atsako laiko padidėjimo. Tuo tarpu didžiausias atsako laikas kontrolinėje aplikacijoje siekia 544259,39 ms, o eksperimentinėje – 837909,46 ms. Tai atskleidžia, jog eksperimentinė aplikacija yra labiau pažeidžiama kritiniais atvejais, kai atsiranda didesnė apkrova.Užklausų per sekundę skaičius kontrolinėje aplikacijoje buvo 23,52, o eksperimentinėje – 15. Šis skirtumas atspindi šifravimo proceso įtaką užklausų apdorojimo spartai. Be to, kontrolinės aplikacijos klaidų dažnis siekė 2%, o eksperimentinės – 3%. Didesnis klaidų dažnis eksperimentinėje aplikacijoje gali būti siejamas su papildomais skaičiavimais, reikalingais šifravimo procesui.

Galiausiai, procentiliniai atsako laikai parodė, kad kontrolinės aplikacijos 90%, 95%, ir 99% atsako laikai buvo atitinkamai 13000 ms, 15000 ms, ir 87000 ms, tuo tarpu eksperimentinėje aplikacijoje šie rodikliai siekė 17000 ms, 21000 ms, ir 245000 ms. Šie rezultatai rodo, kad šifravimas reikšmingai padidino atsako laikus, ypač kraštutiniais atvejais, kai sistemos apkrova buvo didžiausia.

Grafikas iliustruoja vidutinio atsako laiko dinamiką kontrolinėje aplikacijoje (be šifravimo) – mėlyna linija, ir eksperimentinėje aplikacijoje (su šifravimu) – žalia linija, atliekant ilgalaikį testavimą be pertraukų, kai abiejų aplikacijų apkrova palaipsniui didinama. Grafikas yra sudarytas remiantis Locust sugeneruotų stats_history.csv failų duomenimis.



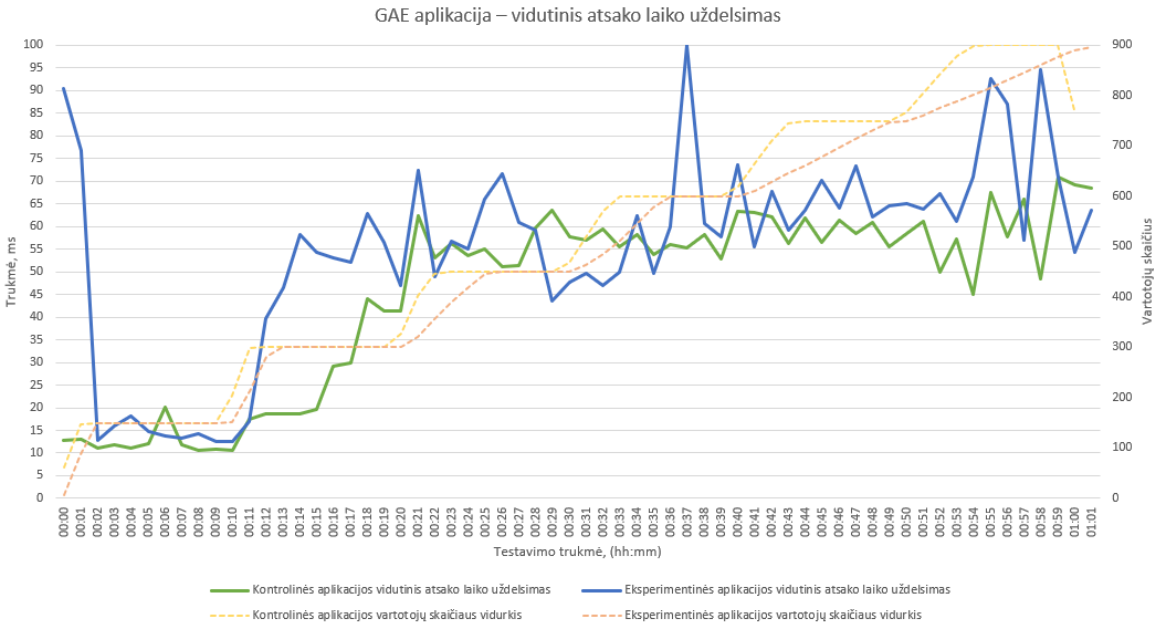
Šaltinis: sudaryta autoriaus.

Testavimo metu abi aplikacijos per tą patį laiką buvo apkraunamos vienodai, tačiau eksperimentinės aplikacijos atsako laikas pradėjo reikšmingai didėti 6 minutėmis anksčiau nei kontrolinės aplikacijos, kai apkrova pasiekė 250 vartotojų. Tai rodo, kad šifravimas jau ankstyvoje stadijoje sukuria papildomą resursų apkrovą. Ilgėjant testavimo laikui ir didėjant apkrovai, šis poveikis tapo dar ryškesnis, o skirtumas tarp aplikacijų vidutinio atsako laiko nuosekliai augo. Kontrolinė aplikacija užtikrino tolygų ir stabilų atsako laiką su nuosekliu augimu, kuris išliko mažesnis nei 9000 ms. Tuo tarpu eksperimentinėje aplikacijoje vidutinis atsako laikas pamažu didėjo ir testavimo pabaigoje pasiekė 15 569 ms. Šifravimo mechanizmai akivaizdžiai padidino resursų apkrovą, tačiau sistema išlaikė veikimo stabilumą, o augimo tempas buvo prognozuojamas. Nors šifravimas reikšmingai padidino atsako laiką, abiejų aplikacijų elgesys išliko stabilus. Kontrolinė aplikacija pademonstravo efektyvų resursų valdymą be papildomos apkrovos, o eksperimentinė aplikacija išlaikė nuspėjamą atsako laiko augimą. Eksperimentinės aplikacijos augantis atsako laikas signalizuoja apie ribą, kurioje sistema galėtų pradėti veikti neefektyviai, jei apkrova toliau didėtų. Tai pabrėžia šifravimo optimizavimo svarbą.

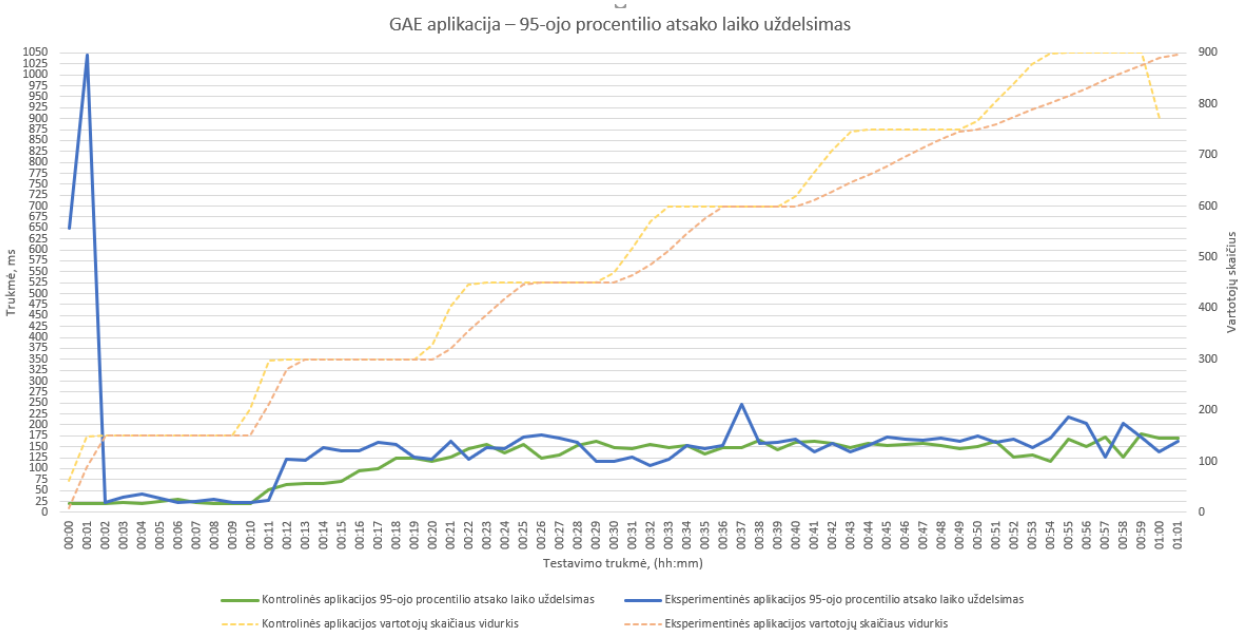
19 PRIEDAS

Google Cloud Console Monitoring grafikų analizė (šifravimo poveikis testavimo be pertraukų metu)

Atsako laiko uždelsimas (angl. *GAE Application – Response latency*):



Šaltinis: sudaryta autoriaus.

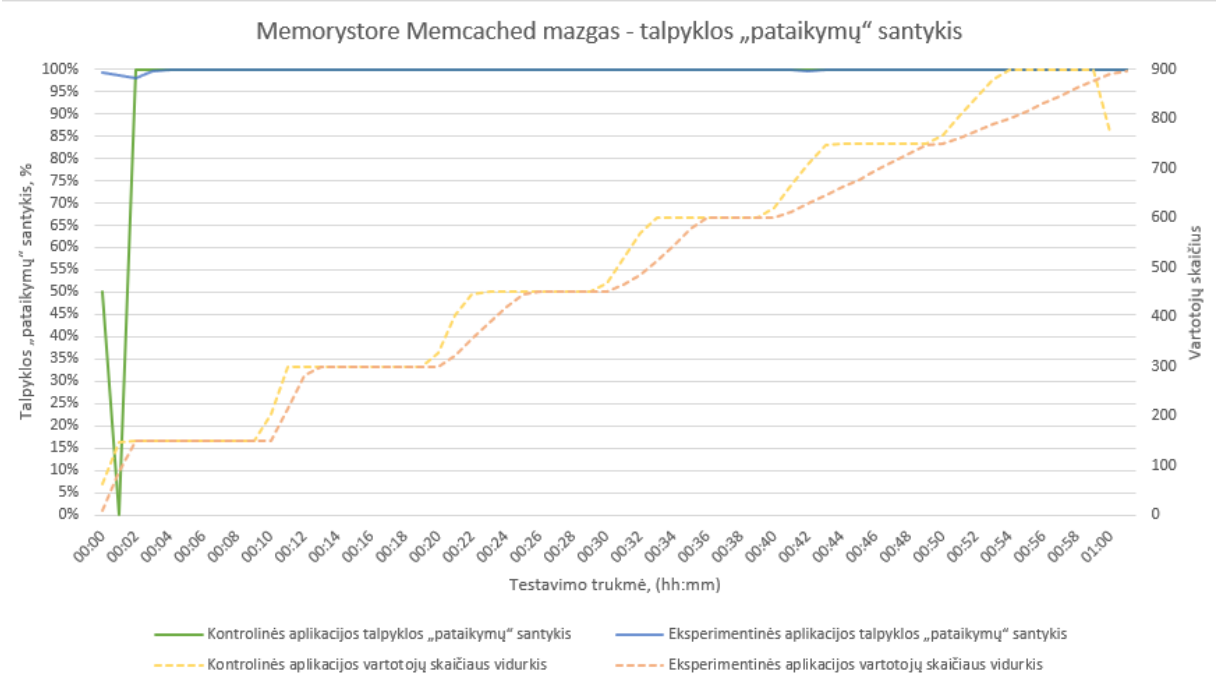


Šaltinis: sudaryta autoriaus.

Kontrolinėje aplikacijoje, esant 200 vartotojų ar didesnei apkrovai, 95-ojo procentilio atsako laikas nusistovi ties ~120–160 ms riba, testavimo metu vidurkis apie 119 ms. Vidutinis atsako vėlinimas yra apie ~46 ms, o tai gana žemas rodiklis dinamiškai didėjančioje apkrovos aplinkoje. Tokio stabilumo priežastis gali būti efektyvus resursų valdymas: aplikacijos logika, be papildomų kriptografinių operacijų, gali sparčiai reaguoti į užklausas, išnaudodama GAE ir talpyklos (*Memcached*) teikiamus pranašumus.

Eksperimentinėje aplikacijoje, tokios pačios apkrovos sąlygomis (200 vartotojų ir daugiau), 95-ojo procentilio atsako vėlinimas svyruoja 120–180 ms intervale, vidutiniškai 134 ms, o vidutinis vėlinimas kinta 40–90 ms diapazone (vidurkis 54 ms). Nors skirtumai nėra labai dideli, svarbus pokytis yra didesnio atsako laiko variacijos lygyje.

Talpyklos „pataikymų“ santykis (angl. *Memorystore Memcached node – hit ratio*):



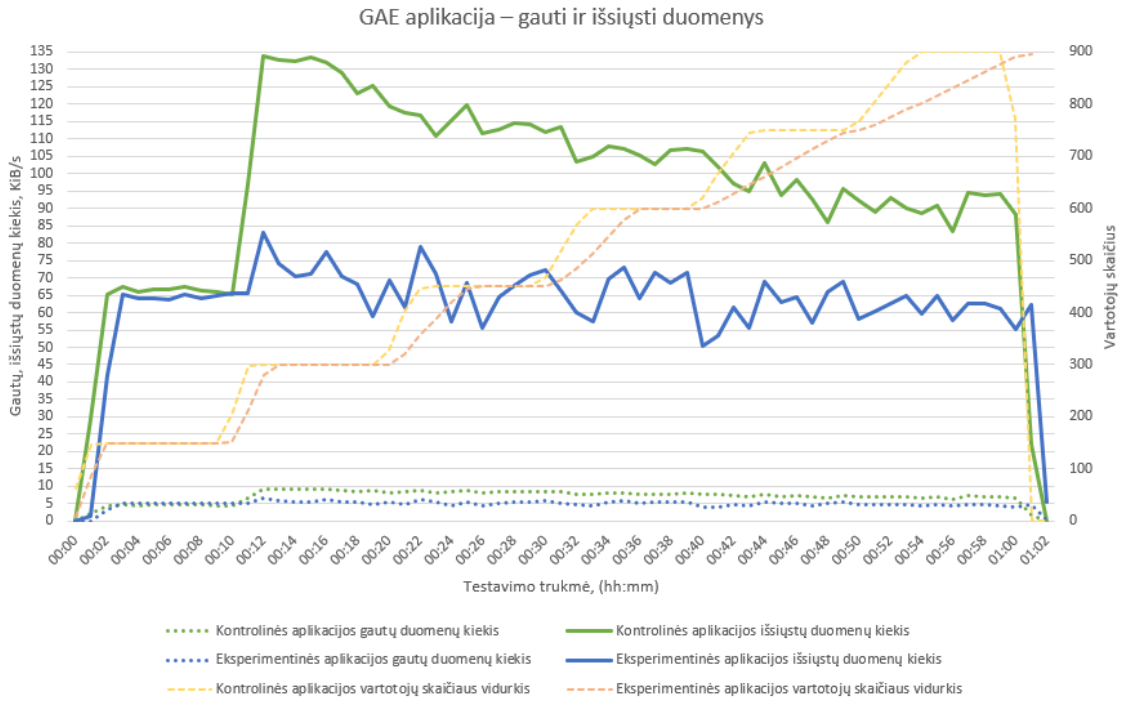
Šaltinis: sudaryta autoriaus.

Kontrolinėje aplikacijoje *hit ratio* išlieka artimas 100%. Tai reiškia, kad talpykloje laikomi duomenys yra beveik visada sėkmingai randami, nereikalingi dažni užklausų nukrypimai į pagrindinę duomenų bazę ar kitus duomenų šaltinius.

Eksperimentinėje aplikacijoje *Hit ratio* taip pat praktiškai 100%, tik retkarčiais pasiekia ~99,8%. Šis nedidelis svyravimas nėra labai reikšmingas. Tai rodo, kad papildoma kriptografija neturi reikšmingos įtakos talpyklos efektyvumui – duomenys talpykloje ir toliau yra gerai

panaudojami. Nesumažėjęs hit ratio rodo, kad atsako laiko vėlavimo augimo priežastis slypi ne duomenų gavimo iš talpyklos mechanizmuose, o greičiausiai pačioje šifravimo logikoje.

Išsiųsti ir gauti duomenys (angl. *GAE Application – Received, sent bytes*):



Šaltinis: sudaryta autoriaus.

Kontrolinėje aplikacijoje duomenų srautas (gauti/išsiųsti baitai) esant didesnei apkrovai po pradinio pakilimo didėjant apkrovai nuosekliai mažėja ir stabilizuojasi. Išsiųstų duomenų vidurkis testavimo metu siekia 98 KiB/s, o gautų – 7,16 KiB/s.

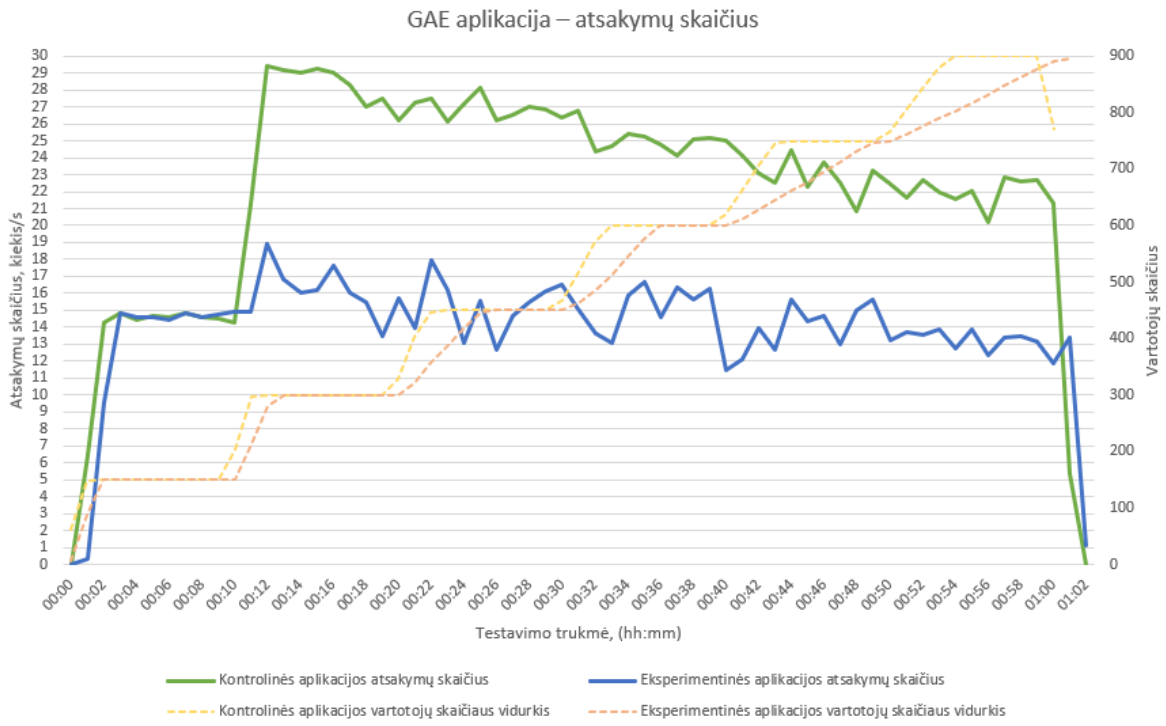
Eksperimentinėje aplikacijoje pastebimi didesni svyravimai tinklo srautuose (60–80 KiB/s intervale), o vidurkiai yra mažesni nei kontrolinės (65 KiB/s išsiųstų, 5,04 KiB/s gautų), tačiau svarbesnis rodiklis – nepastovumas.

Atsakymų skaičius (angl. *GAE Application – response count*):

Kontrolinė aplikacija: Atsakymų kiekis per sekundę yra stabilesnis, iš pradžių siekia beveik 30 ats./s, tačiau didėjant apkrovai palaipsniui mažėja, tačiau išlieka >20 ats./s.

Eksperimentinėje aplikacijoje atsakymų skaičius per sekundę yra gerokai labiau svyruojantis (11–20 ats./s diapazonas). Net ir teoriškai naudojant tą pačią infrastruktūrą, papildomas šifravimo

žingsnis prideda nereguliarumų, kurie stabdo pastovų didelio pralaidumo palaikymą. Kitaip tariant, sistema nebeapsiriboja vien tik duomenų paieška ir atsakymo generavimu. Dabar ji turi atlikti papildomus kriptografinius veiksmus, kurie nors ir trumpi, bet pakankamai varijuoja ir sukelia netolygius atsako laikų svyravimus. Šios variacijos galiausiai lemia mažesnį atsakymų per sekundę skaičiaus vidurkį ir mažesnes galimybes pasiekti stabilų pralaidumo tašką.



Šaltinis: sudaryta autoriaus.

20 PRIEDAS

Locust stats.csv failų suvestinės ir *stats_history.csv* duomenų grafiko analizė autentifikacijos tyrimui testavimo be pertraukų metu

Vartotojų skaičius	Aplikacijos tipas	Vidutinis atsako laikas, ms	Atsako laiko mediana, ms	Didžiausias atsako laikas, ms	Užklausų per sekundę	Klaidų dažnis (%)	90% atsako laikas (ms)	95% atsako laikas (ms)	99% atsako laikas (ms)
Bendras	Kontrolinė	513,94	310	8758,59	18,28	0	940	1800	3700
Bendras	Eksperimentinė	514,97	380	6494,26	18,41	0	940	1300	2700

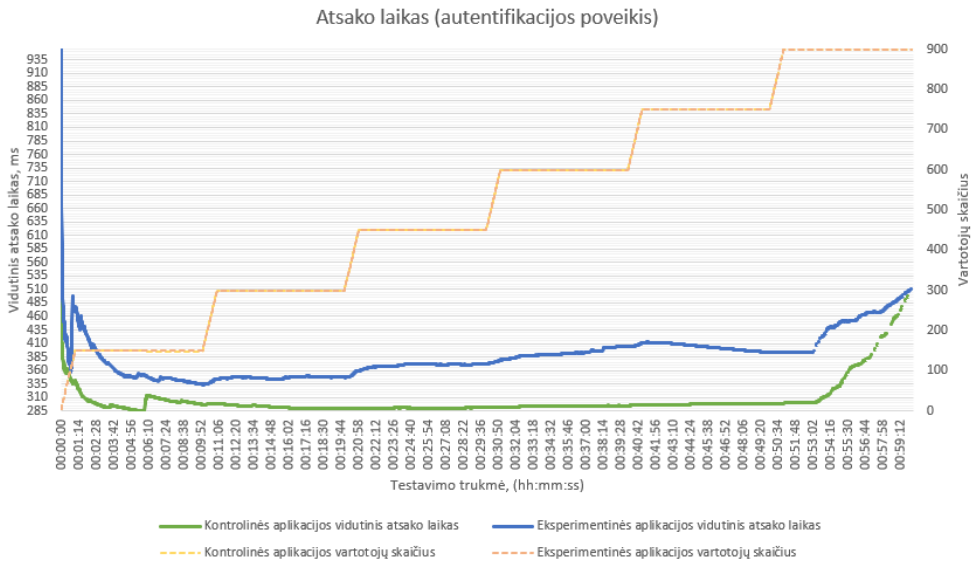
Šaltinis: sudaryta autoriaus.

Lentelėje pateiktų rezultatų analizė parodė, kad vidutinis kontrolinės aplikacijos atsako laikas siekė 513,94 ms, o eksperimentinės – 514,97 ms. Šis skirtumas, sudarantis tik 1,03 ms, leidžia teigti, kad autentifikacija praktiškai neturi įtakos vidutiniam atsako laikui, užtikrinant, kad sistemos veikimo našumas išlieka stabilus.

Analizuojant atsako laiko medianą, kontrolinėje aplikacijoje ji buvo 310 ms, atspindėdama stabilų daugumos užklausų apdorojimą. Tuo tarpu eksperimentinėje aplikacijoje atsako laiko mediana buvo šiek tiek didesnė – 380 ms, o tai rodo nedidelį papildomą autentifikacijos poveikį užklausų apdorojimo laikui. Didžiausio atsako laiko analizė parodė reikšmingus skirtumus tarp aplikacijų. Kontrolinėje aplikacijoje didžiausias atsako laikas siekė 8758,59 ms, tuo tarpu eksperimentinėje – 6494,26 ms. Šis 2264 ms sumažėjimas leidžia manyti, kad autentifikacijos mechanizmai padeda sistemai efektyviau valdyti ekstremalias situacijas, mažindami užklausų, reikalaujančių daugiausiai laiko, apdorojimo trukmę.

Taip pat buvo analizuojamas užklausų per sekundę skaičius. Kontrolinėje aplikacijoje jis siekė 18,28 užklausos per sekundę, o eksperimentinėje – 18,41 užklausos per sekundę. Šie rezultatai patvirtina, kad autentifikacija neturi reikšmingo poveikio užklausų apdorojimo greičiui. Abu testuojami sprendimai pasižymėjo visišku stabilumu, nes klaidų dažnis buvo lygus nuliui. Tai rodo, kad tiek kontrolinė, tiek eksperimentinė aplikacija užtikrina patikimą veikimą net ir esant apkrovai.

Galiausiai buvo analizuojami 90%, 95% ir 99% atsako laikai. Kontrolinėje aplikacijoje 90% atsako laikas buvo 940 ms, o eksperimentinėje – toks pat, tai rodo, kad daugumai užklausų autentifikacija nedaro reikšmingo poveikio. 95% atsako laikas kontrolinėje aplikacijoje buvo 1800 ms, o eksperimentinėje sumažėjo iki 1300 ms, o tai atskleidžia tam tikrą autentifikacijos procesų optimizacinį poveikį. 99% atsako laikas kontrolinėje aplikacijoje siekė 3700 ms, tuo tarpu eksperimentinėje jis buvo ženkliai mažesnis – 2700 ms. Tai rodo, kad eksperimentinė aplikacija pasižymi didesniu stabilumu ekstremaliais atvejais.



Šaltinis: sudaryta autoriaus.

Grafike pavaizduota vidutinio atsako laiko dinamika dviejose aplikacijose: kontrolinėje (be autentifikacijos), pažymėtoje mėlyna linija, ir eksperimentinėje (su autentifikacija), pažymėtoje žalia linija. Testavimo metu buvo taikomas laipsniškas apkrovos didinimas, o rezultatai leidžia įvertinti autentifikacijos poveikį sistemos našumui.

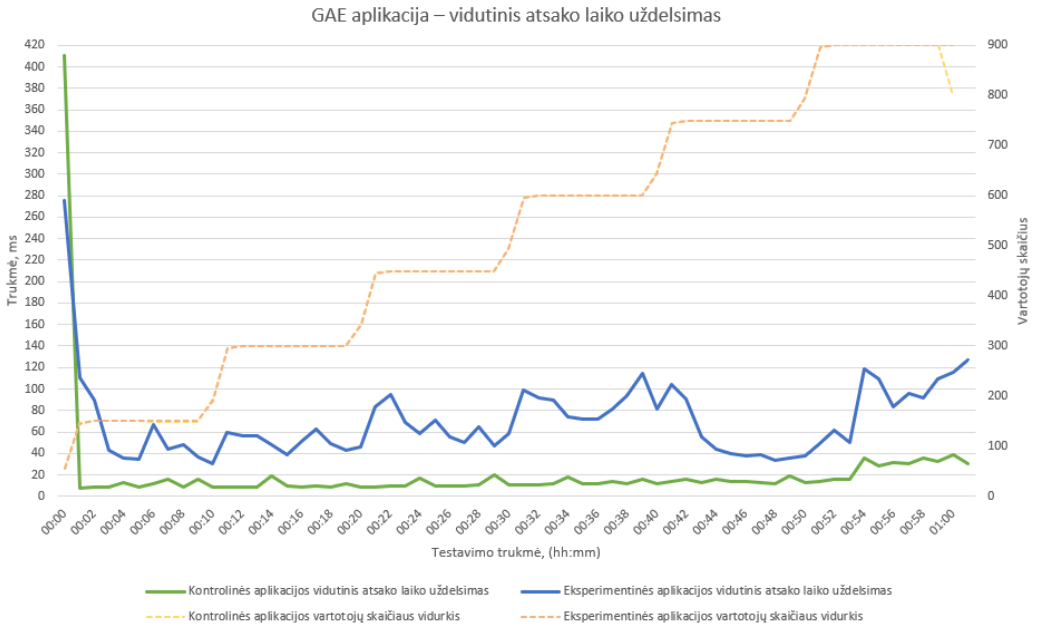
Testavimo pradžioje (0–350 s) abi aplikacijos susidūrė su nestabiliais atsako laikais. Ši fazė atspindi infrastruktūros „įšilimo“ etapą, kai sistema prisitaiko prie apkrovos. Eksperimentinėje aplikacijoje vidutinis atsako laikas buvo ženkliai didesnis (~954 ms), nes papildomi autentifikacijos procesai sukėlė didesnę resursų apkrovą. Tuo tarpu kontrolinės aplikacijos vidutinis atsako laikas buvo mažesnis ir svyravo artimesniame stabilumo lygiui. Stabilizacijos fazėje (~350–3170 s) abi aplikacijos pasiekė stabilų veikimą. Kontrolinė aplikacija demonstravo itin žemą ir stabilų vidutinį atsako laiką (~285–300 ms). Eksperimentinė aplikacija stabilizavosi diapazone nuo 350 iki 410 ms, tačiau skirtumas tarp abiejų aplikacijų išliko nedidelis (~50–100 ms). Šioje fazėje autentifikacijos poveikis buvo matomas, tačiau neesminis, o sistema išlaikė prognozuojamą veikimą. Testavimo pabaigoje (>3170 s), kai apkrova pasiekė maksimumą, abiejų aplikacijų vidutinis atsako laikas pradėjo augti. Tiek kontrolinė, tiek eksperimentinė aplikacijos užfiksavo panašų vidutinį atsako laiką (~510 ms), o tai rodo, jog abi sistemos efektyviai tvarkėsi su apkrovos didėjimu. Nepaisant to, eksperimentinėje aplikacijoje vidutinis atsako laikas buvo kiek didesnis, tačiau autentifikacijos mechanizmai nesukėlė reikšmingų našumo sutrikimų.

Grafike taip pat pastebimas aiškus vartotojų skaičiaus poveikis atsako laikui. Didėjant vartotojų skaičiui, tiek kontrolinė, tiek eksperimentinė aplikacijos demonstravo proporcingą atsako laiko augimą. Tačiau net ir esant didžiausiai apkrovai, abi aplikacijos išlaikė stabilų veikimą be reikšmingų našumo šuolių.

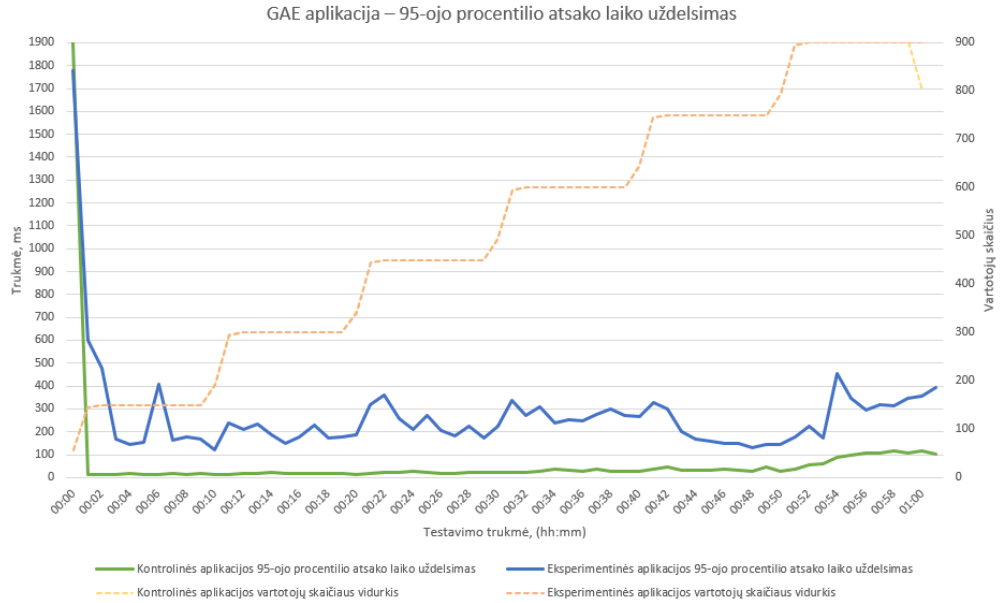
21 PRIEDAS

Google Cloud Console Monitoring grafikų analizė (autentifikacijos poveikis testavimo be pertraukų metu)

Atsako laiko uždelsimas (angl. *GAE Application – Response latency*):



Šaltinis: sudaryta autoriaus.



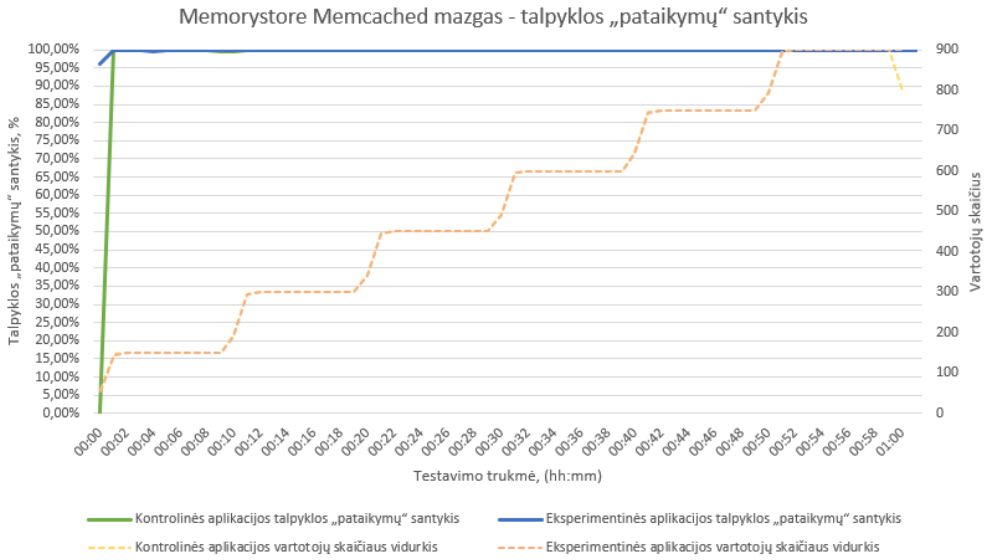
Šaltinis: sudaryta autoriaus.

Esant didėjančiai apkrovai, kontrolinė aplikacija pasiekia puikų stabilumo ir efektyvumo derinį. 95-ojo procentilio atsako laiko vėlinimas išlieka labai žemas (30 – 50 ms), o vidutinis atsako

laikas – vos 9–20 ms. Šie rodikliai rodo, kad kontrolinė aplikacija yra gerai optimizuota, geba efektyviai skirstyti resursus. Vėlinimo rodikliai rodo ir laiko dispersijos stabilumą: nėra žymių „tail latency“ išaugimų, kuriuose nedidelis procentas užklausų smarkiai atsiliktų. Net galutiniame testavimo etape pastebimas šuolis iki 115 ms 95-ojo procentilio vėlinime atrodo kaip izoliuotas įvykis, o ne tvari degraduojanti tendencija. Vidutiniškai 95-ojo procentilio vėlinimo laiko vidurkis yra 36 ms, o vidutinis vėlinimas – 15 ms, o tai atitinka didelio našumo architektūrose siektiną rodiklį.

Esant toms pačioms didėjančios apkrovos sąlygoms, eksperimentinė aplikacija, papildyta autentifikacijos logika, rodo ženkliai didesnį atsako laiko nepastovumą ir vidutinių bei 95-ojo procentilio vėlinimo reikšmių padidėjimą. 95-ojo procentilio atsako laikas reguliariai svyruoja 120–400 ms rėžiuose, o vidutinis atsako laikas pakyla iki 34–110 ms. Vidutinis 95-ojo procentilio vėlinimo laiko vidurkis – 236 ms, o vidutinis atsako vėlinimo laikas siekia 66 ms – tai reikšmingai prastesni rezultatai lyginant su kontroline aplikacija.

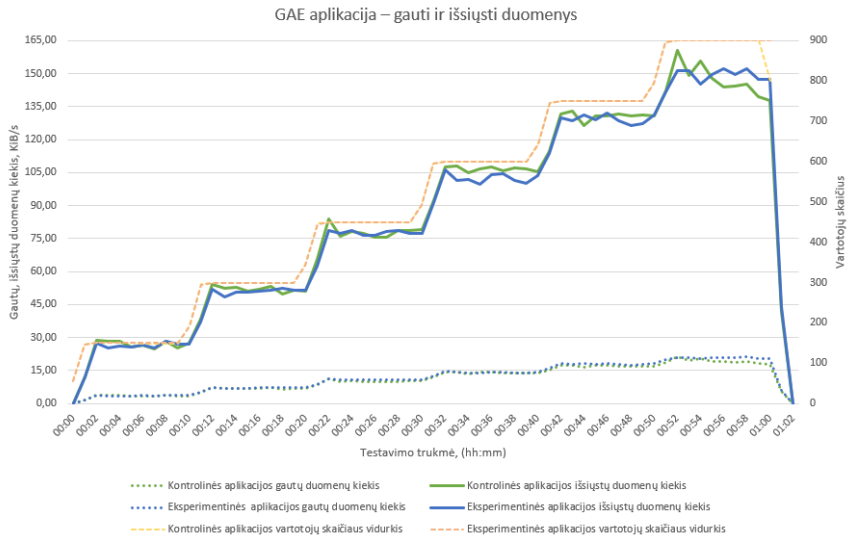
Talpyklos "pataikymų" santykis (angl. *Memorystore Memcached node – hit ratio*):



Šaltinis: sudaryta autoriaus.

Tiek kontrolinė, tiek eksperimentinė aplikacijos palaiko bendrą, labai aukštą (99,97%) hit ratio, su menkais minimaliais nukrypimais, neviršijančiais 1%. Šis aukštas ir stabilus hit ratio rodo, kad talpyklos pasiekiamumas ir duomenų paieškos efektyvumas išlieka nepakitęs nepaisant autentifikacijos integravimo.

Išsiųsti ir gauti duomenys (angl. *GAE Application – Received, sent bytes*):

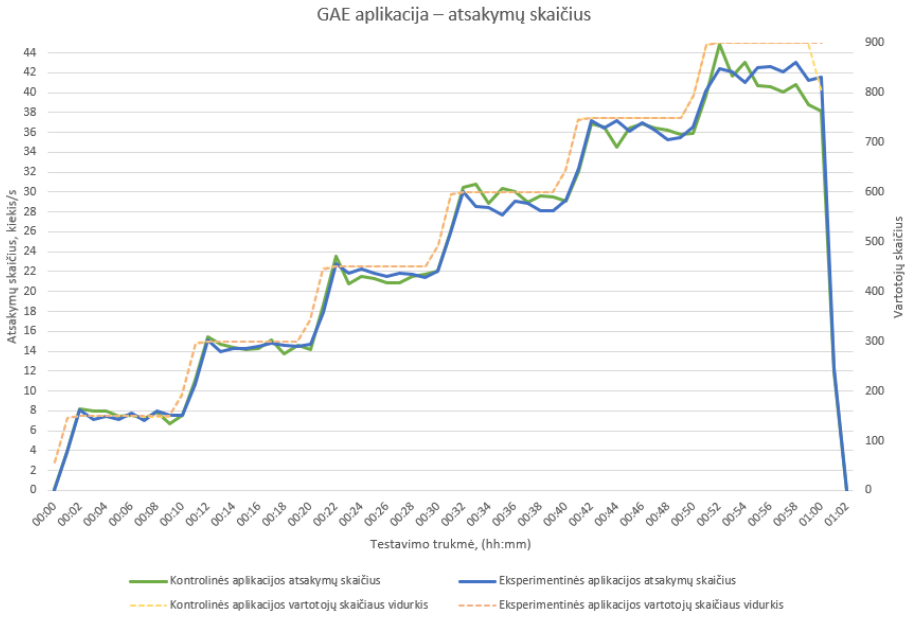


Šaltinis: sudaryta autoriaus.

Abiejose aplikacijose pastebimi „kauburėliai“ gautų ir išsiųstų duomenų grafikuose, kurie tiesiogiai koreliuoja su vartotojų skaičiaus didinimu etapais. Kiekvieną kartą padidinus apkrovą, padidėja ir tinklo srautas (daugiau užklausų – daugiau atsakymų, daugiau duomenų per tinklą), po to srautas stabilizuojasi iki kito augimo etapo. Kadangi tiek kontrolinė, tiek eksperimentinė aplikacija demonstruoja šią proporcingą dinamiką, galima daryti išvadą, kad autentifikacija nekeičią bendros tinklo srauto logikos, nesukelia srauto šuolių ar kritimų, kurie rodytų netinkamą duomenų mainų valdymą.

Atsakymų skaičius (angl. *GAE Application – response count*):

Abu scenarijai (kontrolinis ir su autentifikacija) demonstruoja panašų atsakymų skaičiaus kitimo modelį – „kauburėliai“ atitinka apkrovos didinimo žingsnius. Maksimalios atsakymų reikšmės ir jų dinamika yra labai panašios. Tai leidžia daryti išvadą, kad nepaisant ženklaus vėlinimo augimo eksperimentinėje aplinkoje, bendras atsakymų srautas (užklausų aptarnavimo apimtis per laiko vienetą) išlieka panašus. Kitaip tariant, sistema vis dar sugeba generuoti ir pateikti panašų kiekį atsakymų, net jei kai kuriems iš jų reikia daugiau laiko. Tokia situacija gali reikšti, kad autentifikacija daugiau paveikia atskirų užklausų atsako laiko vėlinimo pasiskirstymą, bet ne riboja bendrą sistemos pajėgumą.



Šaltinis: sudaryta autoriaus.