VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ BAKALAURO STUDIJŲ PROGRAMA

Debesų kompiuterijos sprendimais paremtas modernių saityno programų peržiūros robotas

Cloud-based Crawler of Modern Web Applications

Bakalauro baigiamasis darbas

Atliko:	Kasparas Taminskas	(parašas)
Darbo vadovas:	lekt. Aurimas Šimkus	(parašas)
Darbo recenzentas:	dr. Vytautas Valaitis	(parašas)

Vilnius – 2020

Santrauka

Šiame darbe nagrinėjami saityno peržiūros robotai – programų sistemos, dažniausiai naudojamos archyvuojant žiniatinklio resursus, vykdant didžiųjų duomenų gavybą, taip pat paieškos sistemose – vykdoma kintančio saityno turinio paieška ir indeksavimas. Tokie robotai atlieka automatinę rekursyvią žiniatinklio resursų peržiūrą keliaujant URL adresų erdvės grafu. Siekiama įvertinti, ar tokių sistemų įgyvendinimas remiantis esamomis debesų kompiuterijos infrastruktūros paslaugomis yra efektyvus ir optimalus kaštų prasme. Taip pat sprendžiama dinamiškai HTML turinį generuojančių svetainių peržiūros problema.

Darbe nagrinėjama peržiūros robotų probleminė sritis: apibrėžiamos tokių sistemų veikimo ribos, atliekama literatūroje egzistuojančių peržiūros robotų architektūrinių sprendimų analizė, identifikuojami skirtingi žiniatinklio svetainių atvaizdavimo modeliai ir jų įtaka roboto peržiūros procesui. Praktinėje darbo dalyje remiantis išanalizuota teorine medžiaga realizuojamas debesų kompiuterijos sprendimais paremtas peržiūros roboto prototipas ir įvertinamos jo peržiūros efektyvumo galimybės.

Nustatyta, kad taikyti išplėstines, į turinio aprėptį orientuotas peržiūras, naudojantis debesų kompiuterijos paslaugomis paremta architektūra nėra įmanoma, tačiau mažo-vidutinio lygio peržiūroms, kurios orientuotos į atrandamo turinio kokybę, tokios architektūros gali būi efektyvios. Nors darbo metu įsitikinta, kad neegizstuoja universalus būdas identifikuoti dinamiškai HTML turinį generuojančias svetaines, pristatytas efektyvus tikimybinis modelis, leidžiantis nustatyti tokią identifikaciją ir selektyviai atlikus Javascript varikliu paremtą peržiūrą, išgauti reikšmingą naujų, neiššvalgytų URL adresų kiekį.

Raktiniai žodžiai: saityno peržiūra, saityno inforamcijos rinkimas, saityno peržiūros robotas, išskirstyta sistema, debesų kompiuterija, klientinės pusės atvaizdavimas, dinaminis HTML turinio atvaizdavimas

Summary

In this paper, Web Crawling systems are investigated. Such systems are mostly used in Web archiving, gathering massive amounts of input data used in Machine Learning, also in search engines – discovery of changing Web content is performed and detected resources are indexed. Described systems perform automatic recursive crawling of Web resources while traveling through the graph of interconnected URL addresses. This paper tries to evaluate the possibility to implement such crawling systems that would be performance-useful and cost-effective, while fully relying on cloud computing infrastructure and its services. Also, the issue of crawling of modern, client-side rendered Web Applications is covered.

The process of this paper work includes investigation of the domain of Web Crawling Systems: boundaries of such systems are identified, architectual comparison of several widely known, publicly described Web Crawlers is made, different rendering models of websites and their impact on Web Crawling process are presented. Practical paper work involves the implementation of cloud-based web crawler prototype and its cralwing capabilities evaluation during detailed cloud experiment.

It was determined, that Broad Web Crawling, which focuses on massive content coverage is not possible with such cloud based implementations, however, small-to-medium focused crawls, which features rich content discovery, might be performed effectively with implemented architecture. Even though during the process of this paper work it was ascertained that any universal, bullet-proof way to identify website, which uses client-side rendering, is currently unknown, the effective probability model was presented, which allows to make such identification and perform selective, Javascript engine-based crawling – significant amount of new, previously undiscovered URL addresses is gathered.

Keywords: Web Crawling, Web Scraping, Web Crawling Robot, Distributed System, Cloud Computing, Client-Side Rendering, JavaScript Rendering, Dynamic HTML Content Crawling

TURINYS

Įvadas

Saityno peržiūros sistemos (angl. – *Web Crawling Systems*) atsirado kartu su pirmaisiais interneto paieškos varikliais ir išlieka esminis jų veikimo pagrindas [Naj09]. Tokios programos leidžia turinio agregavimo platformoms efektyviai identifikuoti naujus ar kintančius saityno resursus ir indeksuoti jų turinį, kurį vėliau gali pasiekti paieškos sistemų naudotojai, formuodami paieškos terminais grįstas užklausas. Saityno peržiūros robotų pritaikymas neapsiriboja tik paieškos sistemomis, jie taip pat dažnai naudojami archyvuojant žiniatinklio resursus [Gor04], identifikuojant autorių teises pažeidžiančias svetaines ar kenksmingus puslapius, taip pat atliekant didžiųjų duomenų gavybą [Naj09].

Žiniatinklio peržiūros problema nėra nauja ir literatūroje analizuojama jau kelis dešimtmečius [Cas04], [HN99a], nuo pat pirmųjų interneto naršyklių atsiradimo. Apie jos aktualumą verčia kalbėti eksponentinio interneto resursų augimo tempai: 2020 duomenimis žiniatinklyje yra virš 1,7 mlrd. svetainių (iš kurių tik 12% veikiančių ir pasiekiamų) [Sta20]. Šis skaičius lyginant su 2010 statistika padidėjo net 8 kartus. Toks spartus žiniatinklio resursų kiekio augimas reikalauja lengvai išplečiamos sistemos architektūros, taip pat daug skaičiavimo ir talpinimo resursų: procesoriaus branduolių skaičiavimo galios, operatyviosios atminties, daug disko vietos ir didelio tinklo pralaidumo.

Literatūroje absoliuti dauguma aprašytų viešų saityno peržiūros sistemų remiasi sudėtingos infrastruktūros nuoma ar pirkimu, jos paruošimu (aplinkų konfigūravimas) ir saityno peržiūros roboto sistemos paketo diegimu naudojantis komandinės eilutės instrukcijomis [HN99b], [Vla01], [PAO18]. Tai dažnai lemia didelius finansinius kaštus išlaikant resursus, taip pat specifinių infrastruktūros valdymo žinių poreikį. Mokslinėje literatūroje plačiai aprašyti vieši tokių sistemų sprendimai remiasi dešimtmečių senumo IT rinkos skaičiavimo galios, resursų talpinimo pajėgumų kontekstu, kuriame debesų kompiuterijos galimybės minimos minimaliai. Dauguma tokių literatūroje aprašytų sprendimų remiasi centralizuotais komponentais, kurie plečiant sistemą lemia „butelio kaklelio“ efektą ir neleidžia pasiekti tiesinio pajėgumų išaugimo efekto. Komercinių saityno peržiūros sistemų (tokių kaip *„Googlebot“*) architektūros yra slepiamos, nes lemia strateginę šių kompanijų sėkmę. Literatūroje taip pat stinga informacijos apie modernių saityno programų peržiūros procesą, nors jis tik pasunkina padėtį, nes nuorodos tokiose svetainėse generuojamos dinamiškai, kai puslapis užkraunamas naršyklės, o tradiciniai peržiūros robotai nenaudoja Javascript atvaizdavimo variklio, todėl negali aptikti tokių resursų. Aprašytos problemos dėl apribotų resursų dažnai neleidžia mažesnėms įmonėms, startuoliams atlikti platesnio Interneto svetainių žvalgymo proceso ir užkerta kelią neišnaudotoms rinkos galimybėms.

Šiame darbe nagrinėjama galimybė saityno peržiūros sistemą realizuoti pasinaudojant viešo debesų kompiuterijos tiekėjo siūlomomis paslaugomis ir resursais. Tai leistų dinamiškai koreguoti tokių sistemų veikimo pajėgumus, mokėti už skaičiavimo laiką tik atliekant peržiūros procesą (pay-as-you-go modelis). Būtų išvengiama sudėtingų infrastruktūros, duomenų centrų priežiūros procesų. Rašto darbe plačiai remiamasi Kalifornijos universiteto Mersedo miesto mokslininkų Mehdi Bahrami, Mukesh Singhal, Zixuan Zhuang atlika studija [MZ14] apie išskirstytą debesų kompiuterijos sprendimu grįstą saityno peržiūros sistemą ir naudojamasi pasiūlyta aukšto lygio

tokio tipo sistemos architektūra. Pabrėžtina, jog nurodomoje studijoje nėra pasiekiama prototipinė realizacija, todėl darbe įgyvendinamas panašios, pakoreguotos architektūros saityno peržiūros roboto prototipas ir tyrimo metu įvertinamos jo veikimo galimybės.

Darbo tikslas – įvertinti debesų kompiuterijos sprendimais paremto modernių žiniatinklio programų peržiūros roboto panaudojimo galimybes.

Tikslui pasiekti keliami uždaviniai:

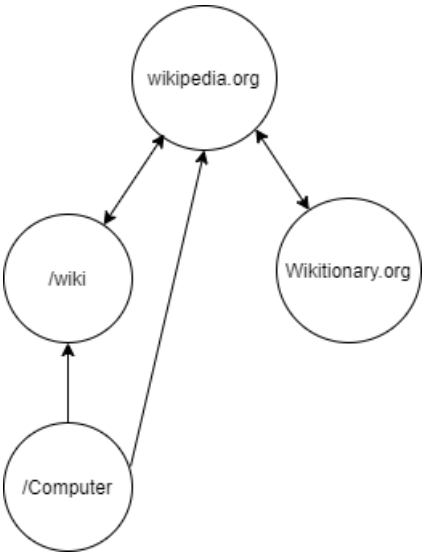
1. Apžvelgti saityno peržiūros robotų probleminę sritį, apibrėžti pagrindinius saityno peržiūros robotų struktūrinius komponentus.
2. Palyginti egzistuojančias peržiūros robotų architektūras.
3. Identifikuoti modernių žiniatinklio puslapių peržiūros problemą ir rinkoje taikomus sprendimus.
4. Suformuluoti funkcinius ir nefunkcinius reikalavimus debesų kompiuterijos technologijomis pagrįstam prototipui.
5. Aprašyti naudojamas debesų kompiuterijos tiekėjo technologijas.
6. Įgyvendinti peržiūros roboto prototipą: apibrėžti sistemos funkcionalumo ribas, pateikti veikimo algoritmus, nubraižyti roboto struktūrinę schemą, atlikti programavimo darbus.
7. Įvertinti šiuos sukurto prototipo aspektus: saityno peržiūros našumą, roboto komponentų apkrovas ir uždelsimus išplečiant peržiūrą.

1. Peržiūros roboto apžvalga

Šiame skyriuje analizuojama teorinė saityno peržiūros roboto sistemų medžiaga, esminės funkcinės charakteristikos, nagrinėjamos skirtingos tokių sistemų kategorijos. Skyriaus siekis – sudaryti skaitytojui aiškesnį probleminės srities kontekstą.

1.1. Keliavimas žiniatinklio nuorodomis

Saitynas ir jį sudarantys puslapių ryšiai dažnai apibūdinami pasitelkiant grafų teorijos terminologiją. Galima teigti, jog kiekvienas puslapis yra viršūnė, o nuorodos – briaunos, kurioms priskiriama kryptis, todėl galima teigti, kad keliavimas žiniatinkliu pasitelkiant žvalgymo robotą yra keliavimas orientuotu grafu [UUD14]. Nuorodos yra paprasčiausi HTML žymėjimo kalbos saitai (angl. – *hyperlinks*). Keliavimo strategija tokiose sistemose yra didelė problema, analizuojama ne viename moksliniame straipsnyje. Kaip pastebima iš 1 paveikslėlio, nuorodos gali būti tiek išorinės (vedančios į kitas svetaines, priklausančias kitiems saityno serveriams), tiek vidinės, sudarančios svetainės hierarchinę struktūrą.



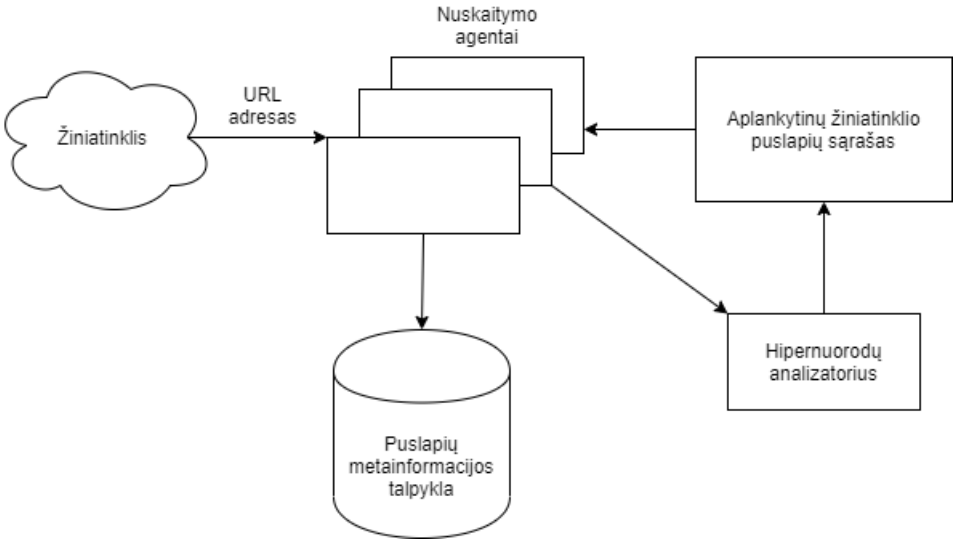
1 pav. Žiniatinklio grafo struktūros vaizdas

1.2. Bazinis veikimo algoritmas

Saityno peržiūros robotų sistemų funkcinis algoritmas nėra sudėtingas – esminė tokių sistemų užduotis yra turint pradinį URL adresų sąrašą parsiųsti šių puslapių turinį naudojantis HTTP protokolu, iš parsiųstų HTML dokumentų išgauti hipernuorodas ir jas suabsoliutinus (nustačius ir resurso serverio vardą) pridėti į lankytinų nuorodų sąrašą tolesiam žvalgymui [ON10]. Parsiųsti dokumentai saugomi didelėse talpyklose – tinkle išskirstytose failinėse sistemose ar duomenų bazėse ir gali būti naudojami tolesniam papildomui apdorojimui (pvz.: svetainės indeksavimui ir semantiniam temos nustatymui, didžiųjų duomenų gavybai ir jų struktūrizavimui ar kt.) [ON10].

Supaprastinta abstrakti tokių sistemų veikimo schema pateikiama 2 paveikslėlyje. Schemoje pastebimas ciklinis veikimo principas – nuorodos analizuojamos nuolat iki kol ištuštėja lankytinų

puslapių sąrašas arba inicijuojamas sistemos darbo nutraukimas iš vartotojo pusės. Tokios sistemos privalo pasižymėti aukštu lygiagretaus agentų veikimo lygiu ir efektyviai išnaudoti sistemos tinklo, procesoriaus ir operatyviosios atminties resursus. Taip pat aktualus ir didelio kiekio duomenų talpinimo klausimas – saugomi šimtai milijonų dokumentų ir nuorodų, o tai reikalauja didelių talpinimo resursų ir optimalios prieigos prie jų.



2 pav. Aukšto lygio saityno pežiūros roboto architektūra [ON10]

1.3. Palyginimas su saityno duomenų rinkimo sistemomis

Terminai **saityno peržiūra** (angl. *Web Crawling*) ir **saityno duomenų rinkimas** (angl. *Web Scraping*) dažnai vartojami kaip sinonimai, nors jų reikšmės skiriasi. Siekiant įvesti aiškią skirtį ir apibrėžti darbe nagrinėjamo tipo sistemų funkcionalumo ribas, šiame skyriuje atliekamas terminų palyginimas.

1.3.1. Skirtumai

Resursų peržiūra dažniausiai apima dideles saityno erdves – vyksta tęstinis procesas, kurio metu siekiama identifikuoti puslapyje pasiekiamas hipernuorodas ir atlikti tų nuorodų tolimesnį žvalgymą [Fat19]. Tuo metu duomenų surinkimo sistema dažniausiai turi aiškų objektą ir jo struktūrą – konkrečią svetainę ar duomenų bazę ir siekia išgauti tam tikrus dominančius duomenis [Fat19]. Galima teigti, jog saityno peržiūros sistemos labiausiai suinteresuotos ryšių tarp puslapių nustatymu tam, kad būtų galima keliauti žiniatinkliu, o žiniatinklio duomenų surinkimo sistemų epicentre – informacijos gavyba [Fat19]. 1 lentelėje pateikta keletas esminių lyginamųjų charakteristikų tarp šių vartojamų terminų.

Nagrinėjami terminai skiriasi, tačiau yra itin susiję, nes saityno peržiūra yra dažniausiai pirmasis informacijos gavybos etapas – reikalingo turinio surinkimas. Žinoma, duomenų surinkimą galima atlikti be peržiūros sistemų pagalbos, tačiau peržiūros sistemos visada kartu naudoja duomenų surinkimo posistemes tam, kad atskirtų vertingą turinį nuo prastos reputacijos svetainės. [Fat19]

1 lentelė. Saityno peržiūros ir duomenų surinkimo sistemų palyginimas [Jha12]

Aspektas	**Peržiūra**	**Duomenų surinkimas**
Veikimo mastelis	Milžiniškas	Koncentruotas
Veikimo erdvė	Žiniatinklis	Žiniatinklis, duomenų bazė, serveris
Dublikatų aptikimas	Esminis veiksmas	Nebūtinas veiksmas
Esminiai funkciniai komponentai	Peržiūros agentai	Duomenų analizatoriai
Rankinio atlikimo galimybė	Negalima	Galima
Tikslas	URL adresų aptikimas	Duomenų išgavimas

Šiame rašto darbe nagrinėjamos saityno peržiūros robotų, dar vadinamų vorais (angl. *Web Spider Bot*), sistemos, kurios keliauja saitynu naudojantis išgaunamomis HTML nuorodomis (angl. – *anchor tags*). Rašto darbe nėra aptariamas svetainės turinio duomenų apdorojimas, struktūrizuotos informacijos gavyba, nes tai nėra tokių sistemų atsakomybės sritys.

1.4. Pagrindinės sistemos veikimo politikos

Saityno peržvalgos roboto sistemų veikimo specifiką sudaro 4 pagrindinių naudojamų strategijų kombinacija [Cas04]. Kiekviena šių politikų yra atskira išsamiai aprašoma mokslinė saityno peržvalgos problema [Cas04], reikalaujanti nuodugnios analizės, todėl šiuose punktuose šios politikos apžvelgiamos bendrais bruožais.

1.4.1. Pasirinkimo politika

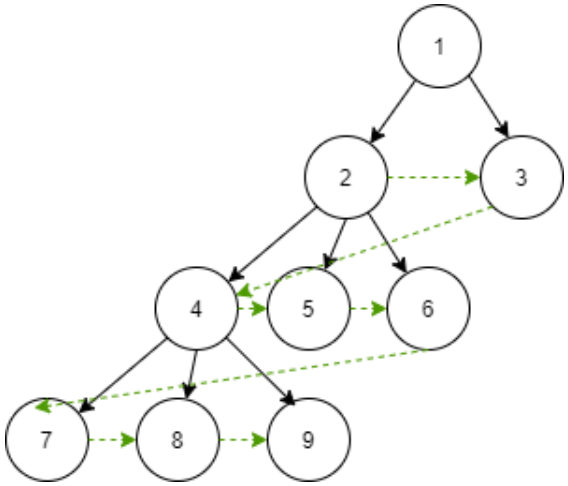
Ši politika apibrėžia žvalgytinų puslapių prioritizavimo tvarką. Vienas didžiausių žiniatinklio žvalgymo iššūkių – milžiniškas Interneto svetainių skaičius, kurio eksponentinis augimas išlieka iki šių dienų. Net patys efektyviausi saityno žvalgymo robotai neturi pakankamai resursų, jog galėtų padengti visas svetaines, todėl atsiranda efektyvaus prioritizavimo klausimas [Cas04]. Reikalinga efektyvi funkcija, kuri galėtų įvertinti žvalgytino puslapio kokybę dar prieš jį atsisiunčiant. Puslapio kokybę galima suprasti kaip kiekvieną iš šių faktorių:

- Nuorodų, rodančių į puslapį, skaičius
- Puslapio semantinės temos atitikimas paieškos užklausai
- Puslapio semantikos atitikimas nuorodos tekstui

1.4.1.1. Paieška į plotį

Viena iš paprasčiausių žvalgymo politikų – paieškos į plotį algoritmas, kurio principas pavaizduotas 3 paveikslėlyje. Atlikti eksperimentiniai žvalgymo tyrimai (aplankyta virš 300 mln. svetainių) parodė, jog paieškos į plotį strategija puikiai veikia surenkant aukštos kokybės puslapius

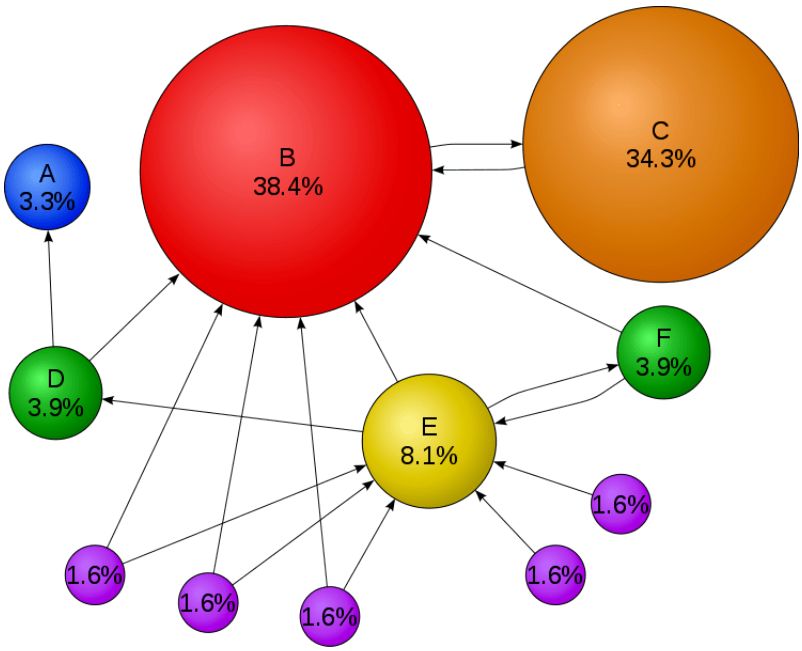
žvalgymo pirmuose etapuose [Cas04]. Ši išvada grįsta nuomone, kad aukštesnės kokybės puslapiai turi daugiau nuorodų, vedančių į juos, todėl atrandami anksčiau [Cas04].



3 pav. Paieškos į plotį grafas

1.4.1.2. PageRank algoritmo metrika

Kitas būdas atlikti efektyvesnį žvalgymą – kiekvienam lanktytinam puslapiui apskaičiuoti kokybės metriką ir lankyti aukštos metrikos vertės puslapius anksčiau. Viena iš tokių metrikų – „Google“ patentuotas ir šios kompanijos pirmasis svetainių kokybės vertinimo algoritmas „PageRank“.



4 pav. „PageRank“ algoritmo principas [Exp13]

Kaip galima pastebėti iš 4 grafo, „PageRank“ algoritmas vertina nuorodų skaičių, kurios rodo į tam tikrą puslapį ir tų nuorodų puslapių kokybę. Matoma, jog B viršūnė turi aukštą autoritetą, nes į ją rodo daug mažų viršūnių, o C viršūnė – nes į ją rodo viena aukšto autoriteto viršūnė.

1.4.2. Etiško žvalgymo politika

Saityno žvalgymo robotai veikia naudodami daug lygiagrečių žvalgymo procesų, kurie sugeneruoja didelius kiekius tinklo I/O srauto. Žvalgant konkretų žiniatinklio serverį, svarbu užtikrinti, jog priskirtas žvalgymo procesas nesutrikdytų sklandaus šio serverio darbo, t.y. nesukeltų netyčinės DoS[1] atakos [Cas04].

1.4.2.1. REP protokolas

Kaip dalinė etiško saityno žvalgymo užtikrinimo priemonė, buvo sukurtas *Robots Exclusion Protocol* taisyklių rinkinys, kuriuo svetainių administratoriai gali nurodyti, kurias svetainės dalis jie leidžia robotams pasiekti, taip pat, koks galimas minimalus žvalgymo užklausų laiko intervalas. Pavyzdinis robots.txt failo turinys matomas 5 paveikslėlyje.

```
# $Revision: 1.25 $ $Date: 2020-01-31 08:44:56 $

User-agent: *
Disallow: /~
Disallow: /poll/
Disallow: /demo/
Disallow: /scripts/
Disallow: /spausdinti/
Disallow: /projektai/reklama/
Disallow: /*print.php
Disallow: /*note.php
Disallow: /*pictureID=
Disallow: /*s2f=
Disallow: /*s2f.php
Disallow: /*com=
Disallow: /fotogalerijos/*p=
Disallow: /apps/banners/
Disallow: /*77676279
Disallow: /*va.php

Sitemap: http://www.delfi.lt/sitemap.xml
Sitemap: http://www.delfi.lt/sitemap_news.xml
```

5 pav. delfi.lt robots.txt pavyzdys

User-agent direktyva apibrėžia robotų pavadinimus, kuriems taikomas taisyklių rinkinys. Kiekvienas etiškas žvalgymo robotas kreipdamasis į žiniatinklio serverį identifikuoja save naudodamas šią HTTP antraštės reikšmę. Pavyzdžiui, „Googlebot“ sistema naudoja tokią šios HTTP antraštės reikšmę:

```
Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Gecko; compatible; Googlebot/2.1;
+http://www.google.com/bot.html) Chrome/W.X.Y.Z‡ Safari/537.36
```

Disallow direktyva nurodo, koks dokumentas ar byla negali būti pasiekiama tam tikram robotui. Taip pat gali būti naudojamas „*“ simbolis (*angl. – wildcard symbol*), nurodantis, jog taisyklių rinkinys turi būti taikomas bet kokiai reikšmei [Kos94].

Į oficialų standartą neįtraukta, bet dažnai sutinkama *Crawl-delay* direktyva, kurią kiekvienas žvalgymo robotas gali interpretuoti skirtingai, dažniausiai tai būna laiko tarpas tarp skirtingų HTTP užklausų į serverį.

[1] DoS - Denial of Service Attack

Akcentuotina, kad šis protokolas negarantuoja, jog robotas nežvalgys uždraustų direktorijų ar resursų, nes žiniatinklyje yra daugybė neetiškų žvalgymo sistemų, siekiančių blogų tikslų.

1.4.2.2. Svetainės žvalgymo intervalai

Ši roboto sistemos charakteristika yra svarbiausia etiško žvalgymo politikoje. Turi būti išlaikytas balansas tarp žvalgymo mandagumo (neapkraunamas žiniatinklio serveris) ir peržvalgos efektyvumo, nes didelės svetainės, turinčios šimtus tūkstančių puslapių, turi būti žvalgomos itin greitai [Cas04].

1.4.3. Pakartotinio apsilankymo politika

Inkrementinio tipo žvalgymo robotai turi užtikrinti, jog sistemos puslapių indekso repozitorijoje turimos lokalios puslapių kopijos yra kiek galima naujesnės ir atitinkančios originalų resursą esamu laiku. [Cas04]

Šiam tikslui yra apibrėžtos dvi pagrindinės funkcijos, sutinkamos literatūroje, kurios kartu nusako puslapio pokyčio neaptikimo kainą.

1.4.3.1. Šviežumo funkcija

Ši funkcija nusako, ar lokali kopija atitinka originalų resursą duotuoju laiko momentu ir gali būti apibrėžiama taip:

$$F\,p(t) = \begin{cases} 1, & p \quad puslapis \quad atitinka \quad laiko \quad momentu \quad t \\ 0 & kitu \quad atveju \end{cases}$$

1.4.3.2. Amžiaus funkcija

Ši funkcija nusako, kokio senumo turima lokali puslapio p kopija:

$$A\,p(t) = \begin{cases} 0, & p \quad puslapis \quad nepakeistas \quad laiko \quad momentu \quad t \\ t & kitu \quad atveju \end{cases}$$

t reikšmė šiuo atveju nusako puslapio modifikavimo laiką.

1.4.3.3. Pakartotinio apsilankymo strategijos

Pagal [Cas04] dažniausiai sutinkamos 2 pakartotinio žvalgymo strategijos:

1. Vieningoji strategija – visi puslapiai pakartotiniai žvalgomi kas tą patį laiko intervalą
2. Proporcinė strategija – dažniau kintantys puslapiai žvalgomi dažniau
3. Svarbos strategija – svarbesni puslapiai (taikomos metrikos įvertinti, pvz.: „PageRank“) žvalgomi dažniau

Ištirta, jog vieningoji strategija plačiajame žvalgyme efektyvesnė už proporcinę [Cas04].

1.5. Peržiūros robotų kategorijos

Saityno peržiūros robotų sistemos skirstomas į skirtingas kategorijas pagal taikomas žvalgymo strategijas. Jos bendrai apžvelgiamos šiame poskyryje.

1.5.1. Plačiosios peržiūros tipo robotai

Šis tipas (*angl. – Broad-Web Crawlers*) nusako robotus, kurių atliekama saityno peržiūra reikalauja itin didelių tinklo pralaidumo resursų, aplankomų vertingų svetainių kiekis per laiko vienetą yra tokios pat svarbos metrika, kaip ir pavienio puslapio peržiūros visiško padengiamumo metrika [Gor04]. Plačiosios peržiūros tipo robotai siekia padengti kuo didesnius saityno resursų kiekius atsižvelgiant į limituotą tinklo srautą, disko vietą ir skaičiavimo resursus [Gor04]. [PAO18] plačiosios peržiūros roboto architektūroje nurodomi apie 4000+ aplankomų puslapių per sekundę peržiūros greičio mastai.

1.5.2. Teminiai peržiūros robotai

Šio tipo (angl. – *Focused Web Crawlers*) saityno žvalgymo sistemos parsiunčia tik puslapius, atitinkačius specifinę semantinę temą [UUD14]. Puslapiai būna glaudžiai semantiškai susiję vienas su kitu. Tokios sistemos turi sugebėti įvertinti svetainės atitikimą apibrėžtai žvalgymo temai ir nuspręsti, kokiu būdu judėti nuorodų grafu pirmyn. Viena iš pasiūlytų idėjų kaip atpažinti resurso aitikimo žvalgymo roboto temai laipsnį – HTML žymių (angl. – *anchor tag*) elemento teksto indeksas ir jo semantinė analizė [Pin94]. Šio tipo robotai atlieka mažo-vidutinio dydžio peržiūras, nereikalaujančias ypač daug tinklo ir talpos resursų, tipiškai tokios peržiūros siekia iki 10 mln. dokumentų (vidutiniškai apie 100 aplankomų dokumentų per sekundę) [Gor04].

1.5.3. Inkrementiniai peržiūros robotai

Tradicinės literatūroje aprašytos saityno žvalgymo sistemos periodiškai atnaujina savo turimą žiniatinklio dokumentų indeksą pakeisdamos senus dokumentus naujai parsiųstais [UUD14]. Inkrementiniai robotai savo turimą indeksą keičia analizuodami kiekvieno resurso pokyčio laipsnį (įvairūs semantiniai algorimtai, NPL[2] algoritmai). Tokie robotai taip pat pakeičia mažiau svarbius puslapius į svarbesnius, todėl išsprendžiama indeksuotų resursų naujumo ir aktualumo problema, taip pat taupoma žvalgymo roboto sistemos disko vieta, interneto srautas.

1.5.4. Išskirstyti peržiūros robotai

Ši kategorija apibrėžia žvalgymo roboto sistemos topologinę struktūrą – sistema susideda iš centrinio serverio, kuris valdo daug išskirstytų agentų: jiems priskiria URL adresus ir liepia atlikti HTTP žvalgybą [UUD14]. Tokios kategorijos sistemos skirtos išgauti plačiam žiniatinklio resursų padengiamumui, šiomis dienomis efektyviam žvalgymui tokia struktūrizacija yra tiesiog privaloma. [UUD14].

[2]NLP - Natural Language Processing

2. Peržiūros roboto architektūra

Šiame skyriuje analizuojamos mokslinėje literatūroje aprašytos saityno peržiūros robotų realizacijos – pagrindiniai komponentai, jų funkcinės atsakomybės, charakteristikos, atliekama palyginamoji analizė tarp nagrinėjamų peržiūros robotų.

2.1. Komponentai

M.Najorc ir C. Olston mokslinė saityno peržiūros sistemų robotų apžvalga ([ON10]) formalizuoja anksčiau literatūroje aprašytas tokių sistemų dizaino specifikas. Joje nusakoma išskirstyto peržiūros roboto architektūra – skirtingose mašinose egzistuojantys žvalgymo procesai, kiekvienas jų turintis keletą lygiagrečiai veikiančių agentų gijų, kurios atlieka kartotinius žvalgymo ciklo žingsius, kuriuose dalyvauja išskiriami pagrindiniai 8 struktūriniai sistemos komponentai.

2.1.1. „Pasienis“

„Pasienio“ duomenų struktūra (angl. – *URL Frontier*) saugo URL[3] adresų sąrašą, kurie bus aplankyti, iš šio sąrašo paduodamas adresas žvalgymo agento gijai pagal atitinkamas žvalgymo mandagumo (angl. – *Politeness*) ir prioritizavimo (angl. – *Priority*) politikas ([ON10]. Tai viena iš pagrindinių žvalgymo roboto būsenos duomenų struktūrų. Jai keliami šie pagrindiniai funkcionalumo reikalavimai:

- Pridėti URL adresą į sąrašą
- Nuskaityti URL adresą iš sąrašo

2.1.2. HTTP parsiuntimo modulis

Žvalgymo agentui gavus URL adresą iškviečiamas HTTP modulis, kuris pirmiausia kreipiasi į *DNS adreso išaiškinimo* komponentą tam, jog būtų nustatytas URL resurso serverio vardo IP protokolo adresas [ON10]. Šis veiksmas reikalingas tam, kad būtų minimizuotas HTTP užklausos atsakymo laikas (išvengiama DNS išaiškinimo užklausų į išorinius serverius).

2.1.3. Saityno nuorodų ištraukiklis

Šis komponentas (angl. – *Link Extractor*) nuskaito parsiųsto HTML dokumento turinį ir išgauna visas HTML nuorodas tiek į išorinius (angl. – *Offsite Links*), tiek į vidinius (*In-site Links*) žiniatinklio serverio puslapius [ON10].

2.1.4. Adresų skirstiklis

Šis modulis (angl. – *URL Distributor*) atsakingas už išgautų nuorodų priskyrimą atitinkamiems žvalgymo procesams [ON10].

[3]URL - Uniform Resource Locator

2.1.5. Adresų filtras

Komponentas, kuris filtruoja priskirtus URL adresus ir gali išmesti taisyklių neatitinkančias nuorodas (pvz.: puslapiai, įtraukti į juodąjį sąrašą) [ON10]. Taisyklės gali būti specializuotos kiekvienam žvalgymui atskirai.

2.1.6. Dublikatų šalintojas

Roboto dalis, kuri atlieka testą, ar URL nuoroda dar nebuvo aplankyta peržiūros metu, pagrindiniai keliami funkcionamulo reikalavimai [ON10]:

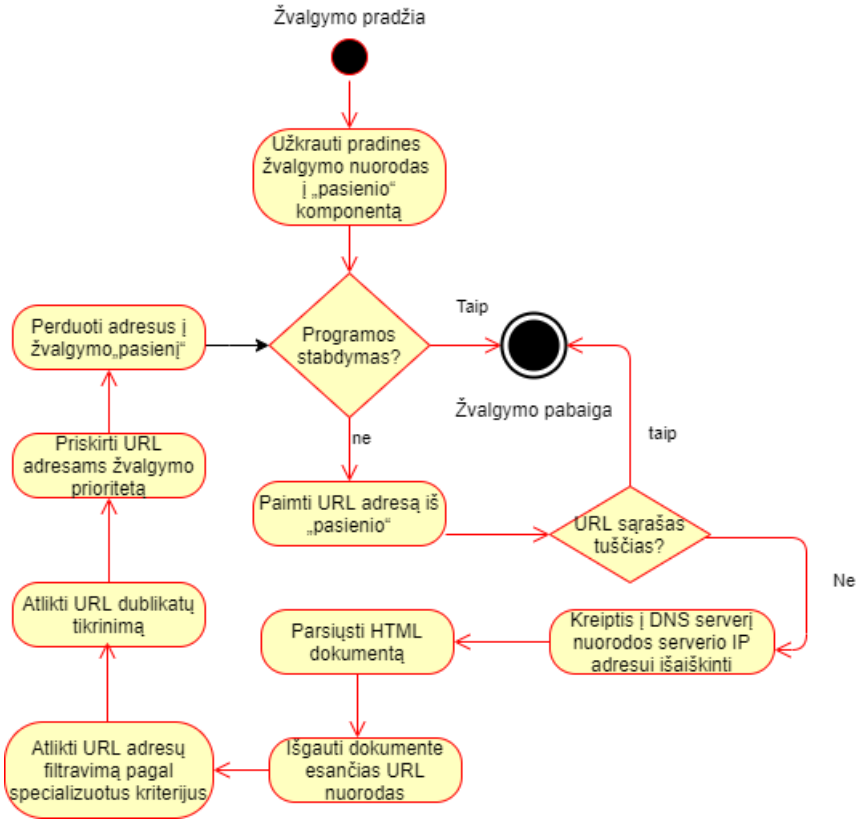
- Pridėti URL adreso aplankymo indikatorių į sąrašą
- Atlikti URL priklausymo sąrašui testą

2.1.7. Adresų prioritizuotojas

Komponentas (angl. – *URL Prioritizer*), kuris kiekvienam URL adresui priskiria tam tikrą prioritetą pagal specializuotus saityno peržiūros roboto sistemos pasirinkimo politikos faktorius, tokius kaip nustatomas puslapio svarbos laipsnis ar puslapio keitimosi greičio faktorius [ON10].

2.2. Peržiūros vykdymo ciklas

Atsivelgus į 3.1 poskyrio struktūrinius komponentus pagal [ON10] pasiūlytą schematinį dizainą, galima sudaryti veiklos diagramą, parodančią sistemos ciklinį funkcionavimą.



6 pav. Saityno peržiūros ciklo UML veiklos diagrama [UUD14]

2.3. Literatūroje aprašytų robotų palyginamoji analizė

Šiame skyriuje lyginama keletas žinomiausių akademiniuose šaltiniuose aprašytų saityno peržiūros robotų architektūrų siekiant suvokti, kaip kiekviena jų koordinuoja peržiūros proceso agentų darbą.

2.3.1. Peržiūros robotų išplečiamumo iššūkis

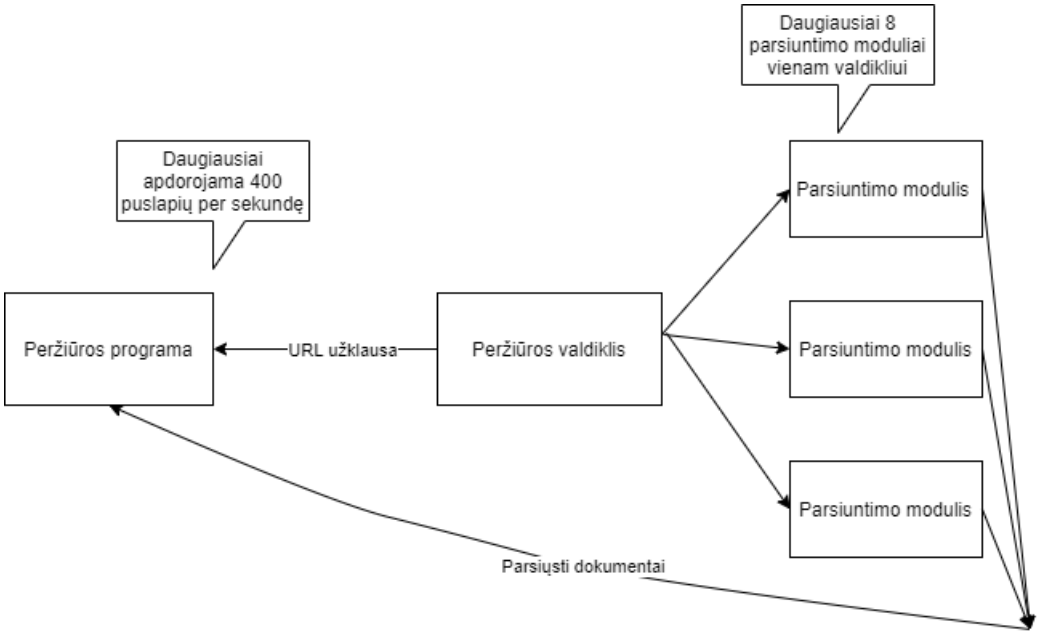
Nors koncepcinis peržiūros sistemų algoritmas, aprašytas 1 skyriuje, yra labai paprastas, tokių sistemų problemos kyla sprendžiant išplėtimo iššūkius – siekiant peržiūrėti milijardus svetainių per pagrįstai trumpą laiką ir išlaikyti peržiūrėtų svetainių naujausią galimą kopiją [Naj09]. Pagal [Naj09] apžvalgą, galima būtų iškelti šį pagrindinį peržiūros robotų architektūrų iššūkį:

- Sugebėti vykdyti peržiūros procesą išskirstytai ir lygiagrečiai, tačiau vykdyti tai etiškai neapkraunant žiniatinklio serverio (etiško žvalgymo politika)

2.3.2. „PolyBot“ sistema

V. Shkapenyuk ir T. Suel aprašytas tinkle skirtinguose mazguose išskirstytas saityno peržiūros robotas, parašytas C++ ir Python programavimo kalbomis [Vla01].

Aprašyta architektūra (7 schema) pasižymi peržiūros valdiklio komponentu, kuris gauna URL adresų užklausas ir paskirsto jas parsiuntimo komponentams [Vla01]. Kiekvienas valdiklis gali komunikuoti daugiausiai su 8 parsiuntimo moduliais [Vla01]. Peržiūros valdikliai tokioje architektūroje sukelia „butelio kaklelio“ efektą. Kitas apribojantis šios architektūros komponentas – peržiūros programa (angl. – *Crawling Application*), kuri atsakinga už HTML dokumentų apdorojimą, peržiūros prioritizavimą ir žvalgytinų puslapių perdavimą valdikliui [Vla01]. Kaip teigiama [Vla01] šaltinyje, šis komponentas gali daugiausiai apdoroti 400 puslapių per sekundę.



7 pav. „Polybot“ peržiūros roboto architektūra [Vla01]

2.3.3. „Mercator“ sistema

1999 m. išsamiai literatūroje autorių A.Heydon ir M.Najork aprašyta ir eksperimentiškai įvertinta paskirstyta saityno peržvalgos roboto architektūra. Ši sistema parašyta naudojantis Java programavimo kalba ir jos vykdomąja aplinka (angl. – *runtime environment*). Aprašytas robotas didžiąją dalį savo aprašomų duomenų struktūrų talpina disko atmintyje, taip pat nedideli buferiai saugomi operatyvioje atmintyje ir skirti pasiekti greitesnei duomenų prieigai (kešavimo mechanizmas) [HN99a].

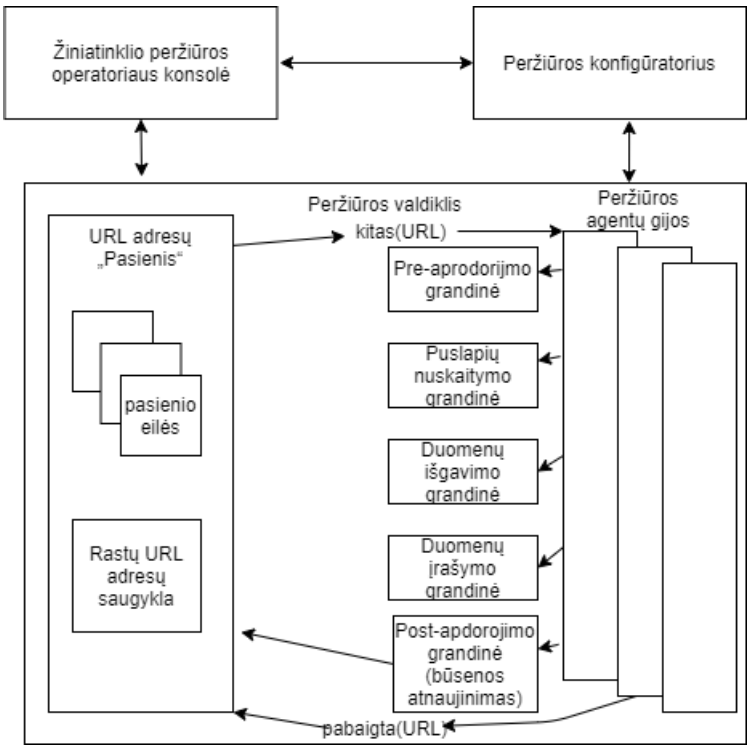
2.3.3.1. Peržiūros išskirstymas ir lygiagretumas

Sistemoje saityno peržiūra vyksta naudojant žvalgymo procesų gijas (angl. – *Worker Threads*, kurios veikia lygiagrečiai ir įprastai sistemoje skaičiuojamos šimtais vienetų vienu metu [HN99a]. Kiekviena gija atsakinga už puslapio atsiuntimą ir apdorojimą (nuorodų suradimą, suabsoliutinimą) Nors fizinės mašinos lygmenyje veikia daug gijų, tačiau sprendimas neparemtas topologiniu peržiūros išskaidymu [HN99a]. Šioje architektūroje aprašomas peržiūros robotas yra centralizuotas – egzistuoja peržiūros koordinatoriaus komponentas, kuris atsitiktiniu I/O būdu komunikuoja su lygiagrečiose gijose veikiančiais peržiūros agentais ir renka jų informaciją [MZ14].

2.3.4. „Heritrix“ sistema

Nepelno siekiančios organizacijos „Internet Archive“ suprojektuotas peržiūros robotas, kurio pagrindinis tikslas rinkti svetainių kopijas ir saugoti jų registrą ateities kartoms. [Gor04].

8 schemoje pavaizduota šio roboto architektūra, pasižyminti 3 esminiais komponentais – operatoriaus konsole (asministratoriaus valdymo sąsaja), peržiūros konfigūratoriumi (peržiūros parametrų nustatymas) ir peržiūros valdikliu (peržiūros proceso koordinavimas) [Gor04]. Sistema nepalaiko išskirstyto tinkle peržiūros koordinavimo [Gor04], žvalgymas vyksta vienoje fizinėje mašinoje, programa turi daug peržiūros agentų gijų, dirbančių lygiagrečiai [Gor04]. Šioms gijoms duodami žvalgytini URL adresai kitas(URL) operacijos pagalba. Šį koordinavimo procesą atlieka peržiūros valdiklis [Gor04].



8 pav. „Heritrix“ peržiūros roboto architektūra [Gor04]

Iš 2 lentelės apžvelgtų literatūroje pateikiamų klasikinių peržiūros robotų palyginimo matoma, jog aprašyti sprendimai plačiai taiko centralizuotą peržiūros modelį (koordinatorius), kuris didinant lygiagrečiai veikiančių agentų skaičių lemia ribotą greitaveiką, nes yra bendrinis visiems agentams. Taip pat pastebima, jog didžioji dauguma sprendimų nepalaiko išskirstytos tinkle peržiūros koordinacijos (nebent atskiri peržiūros procesai), išskyrus „PolyBot“, kuris teoriškai tą palaiko, tačiau pagal [Vla01] pateikiamą literatūrinę medžiagą, tokia konfigūracija niekada nebuvo išbandyta praktiškai.

2.3.5. Palyginimo lentelė

2 lentelė. Nagrinėtų architektūrų palyginimas

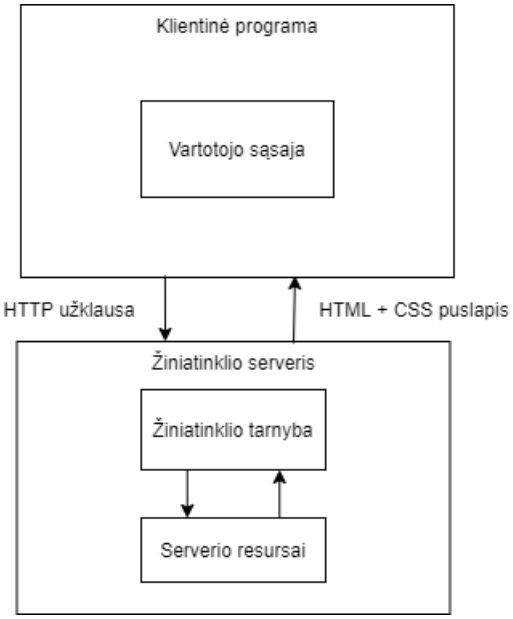
Robotas	**Kalba**	**Išskirstytos tinkle peržiūros koordinavimas**	**Išskirstyto žvalgymo modelis**	**Koordinatoriaus egzistavimas**
„PolyBot“	C++, Python	Palaikoma	Lygiagrečios gijos	Taip
„Mercator“	Java	Nepalaikoma	Lygiagrečios gijos	Taip
„Heritrix“	Java	Nepalaikoma	Lygiagrečios gijos	Taip

3. Dinaminių puslapių peržiūros problema

Saityno peržvalgos robotų sistemų tradicinis modelis remiasi HTTP užklausomis į konkretų žiniatinklio serverį, kurį identifikuoja URL adresas. Taigi, tradicinės tokių sistemų implementacijos darydavo prielaidą, jog URL adresas unikaliai identifikuoja statinį turinį žiniatinklyje. Šis požiūris tapo nebetinkamas, kai W3C konsorciumas standartizavo AJAX[4] formatą asinchroninei žiniatinklio kliento komunikacijai su serveriu [KKU18]. Išplito dinaminiai puslapiai, kurie savo turinio dalis keičia asinchroniškai bendraudami su serveriu ir apsikeisdami informacija. Ši problema tapo itin aktuali su Vieno puslapio aplikacijų [5] ir jų kūrimo karkasų, tokių kaip React, Angular ar VueJS išpopuliarėjimu. Šiame skyriuje palyginamas tradicinis žiniatinklio resursų modelis su nauju, AJAX technologija grįstu modeliu, apžvelgiamos rinkoje taikomos praktikos naujo tipo žiniatinklio programų žvalgymui.

3.1. Tradicinis žiniatinklio programos modelis

Tradicinis žiniatinklio programos modelis remiasi sinchronine kliento-serverio komunikacija, kurioje klientas neturi jokios verslo logikos, visa puslapio žymėjimo ir stiliaus kodo sukonstravimo logika vyksta serverio pusėje. Tokiame modelyje saityno serveriai apkraunami tinklo užklausomis, nes norint pakeisti ar atnaujinti bet kokią puslapio dalį, reikalinga perkrauti visą puslapį iš naujo – nusiųsti pakartotinę HTTP užklausą serveriui [KKU18]. Supaprastinta tokio modelio schema pateikiama 9 paveikslėlyje.



9 pav. Tradicinis žiniatinklio svetainės komunikacijos modelis [KKU18]

Pagrindiniai tokio modelio trūkumai:

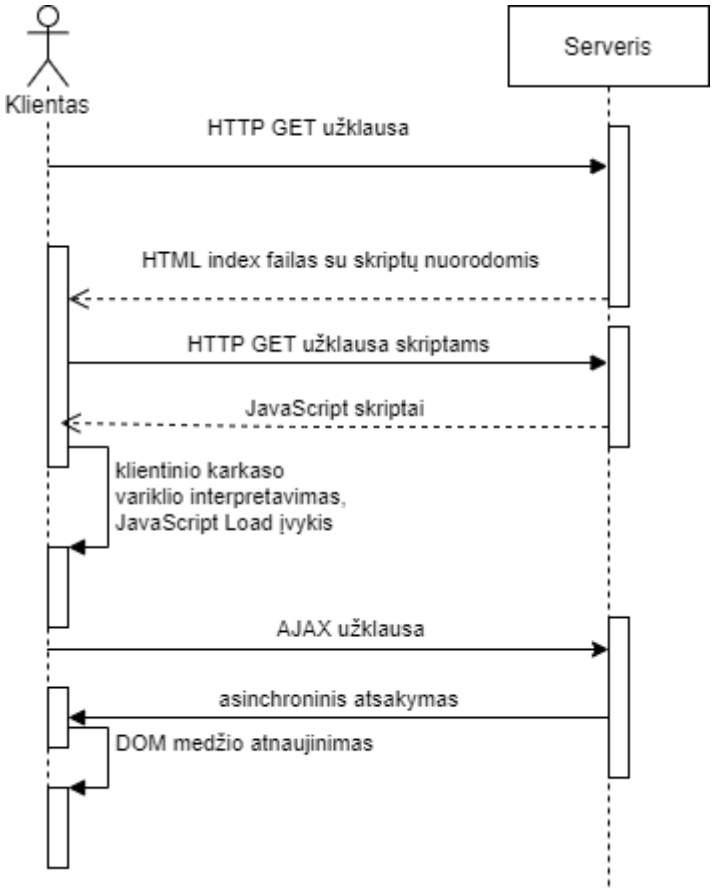
- Visa kodo vykdymo logika atliekama serveryje

[4] AJAX - Asynchronous JavaScript and XML
[5] SPA - Single Page Applications

- Pakartotinis dublikuotų duomenų perdavimas
- Ganėtinai lėti serverio atsako laikai dėl pilno resurso užklausimo

3.2. AJAX žiniatinklio programos modelis

AJAX modelis taikomas naujo tipo vieno puslapio žiniatinklio svetainėse, kurių pagrindinis bruožas – būsena kliento pusėje ir jos atnaujinimas dinamiškai reaguojant į kliento įvykius (angl. – *DOM Events*) ir siunčiant asinchronines AJAX užklausas į serverį. Detalesnė tokio modelio schema pateikiama 10 paveikslėlyje.



10 pav. Kliento pusės atvaizdavimas [Sit18]

3.2.1. AJAX modelio žvalgymo problema

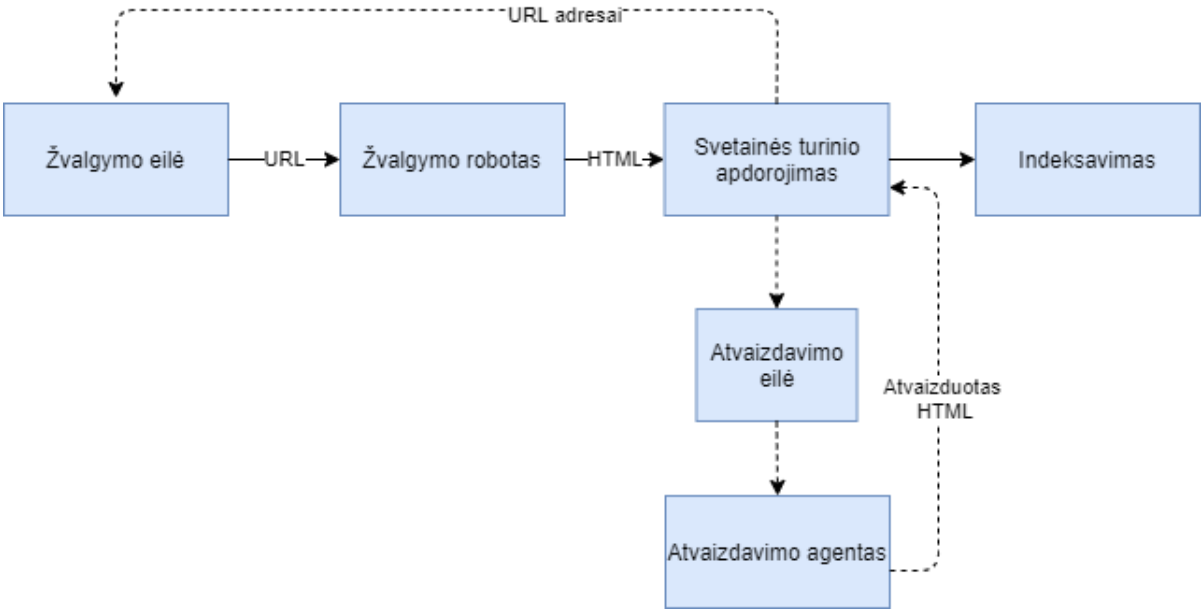
Tradiciniai žvalgymo robotai keliauja žiniatinklio grafu vykdydami HTTP GET užklausas, išgaudami nuorodas tiesiai iš serverio grąžinamo HTML kodo. Naujojo AJAX modelio svetainės reikalauja, jog svetainė būtų užkraunama pilnai, t.y. atsiunčiamas ir įvykdomas JavaScript kodas. Kitaip tariant, paieškos žvalgymo robotas turi elgtis panašiai kaip paprastas naršykle besinaudojantis žmogus [Sit18].

3.2.1.1. „Googlebot“ taikoma praktika žvalgant AJAX svetaines

„Googlebot“ žvalgymo robotas paiešką atlieka 3 etapais:

1. Žvalgymas (angl. – *Crawling*)
2. Atvaizdavimas (angl. – *Rendering*)
3. Indeksavimas

Atvaizdavimo etapas atliekamas būtent AJAX tipo dinaminėms svetainėms, kaip skelbiama „Google“, šiam procesui naudojama atvaizdavimo eilė, iš kurios agentai žinutes paima tuomet, kai atsilaisvina „Google“ žvalgymo infrastruktūros resursai. Tipiškai tai gali užtrukti nuo kelių minučių iki kelių savaičių. Šio modelio pagrindinė problema – poreikis turėti efektyvią indikaciją, jog puslapis reikalauja Javascript atvaizdavimo ir jį reikia siųsti į atvaizdavmo eilę. Išsamesnė „Googlebot“ žvalgymo ir atvaizdavimo (angl. – *Rendering*) schema pavaizduota 11 paveikslėlyje.



11 pav. „Googlebot“ žvalgymo procesas [Goo20]

Pavaizduotoje schemoje atvaizdavimo procese yra naudojamas „Chromium“ naršyklės variklis be vartotojo sąsajos, kuris sugeba įvykdyti svetainės JavaScript kodą. Tiesa, „Google“ neatskleidžia, koks roboto svetainės užkrovimo maksimalus laukimo laikotarpis. Kaip teigiama, kol kas „Chromium“ variklis yra leidžiamas atskirai nuo bendrų „Google Chrome“ naršyklės išleidimų, todėl jo funkcionalumai ir palaikomos ECMAScript naujovės atsilieka, todėl atvaizdavimo metu „Googlebot“ gali nesuprasti naujausių svetainės sintaksės elementų. [Goo20].

Šiame skyriuje pristatytos dinaminių svetainių atvaizdavimo schemos realizacija (technologijos ir priemonės) nagrinėjama praktinėje rašto darbo dalyje, 4 ir 5 skyriuose, kuriuose taip pat nurodoma, kokiu algoritmu įgyvendinamame prototipe priskiriama indikacija, jog svetainė reikalauja dinaminio Javascript atvaizdavimo.

4. Pasirinktos debesų kompiuterijos technologijos

Šis skyrius žymi praktinės rašto darbo dalies pradžią, jame apžvelgiamos pasirinktos „Microsoft Azure“ debesų kompiuterijos tiekėjo paslaugos ir resursai argumentuojant pasirinkimo motyvus. „Azure“ platforma pasirinkta dėl autoriaus asmeninės patirties, tačiau rašto darbo autorius neabejoja dėl galimybės panašias paslaugas pasirinkti kitų tiekėjų, tokių kaip „Amazon AWS“ [6] ar „Google Cloud Platform“ platformose. [MZ14] moksliniame darbe, kuriuo šiame rašto darbe plačiai remiamasi kuriant peržiūros robotą, aprašyta sistema buvo kurta pagal senąjį „Azure“ infrastruktūros modelį, kuris šiuo metu aktyviai nebepalaikomas, todėl įgyvendinant šį prototipą buvo pasitelkti naujesni „Azure“ debesų kompiuterijos tiekėjo įrankiai, kurie leistų užtikrinti analogišką aprašytos architektūros įgyvendinimą.

4.1. Sistemos orchestravimas

Įgyvendinant detalųjį architektūrinį saityno peržvalgos roboto sistemos dizainą ir renkantis iš technologijų, buvo pasirinkta „Microsoft Azure Service Fabric“ mikroservisų orchestravimo platforma, kuri apibrėžia ne tik sistemų klasterių valdymą ir išplėtimą, tačiau taip pat turi savo vykdomąją aplinką (angl. – *Service Fabric Runtime Environment*), kuri palaiko „Service Fabric“ programavimo modelius [Cor18]. Alternatyvi galimybė buvo rinktis „Azure Kubernetes Service“ orchestratoriaus paslaugą, tačiau „Service Fabric“ buvo pasirinktas dėl didelių vykdomosios aplinkos ir programavimo modelių galimybių, kurios suteikia daug abstrakcijos programuotojams – leidžia orientuotis į verslo kodo rašymą paliekant infrastruktūrines surišimo problemas pačiai platformai.

4.1.1. „Service Fabric“ infrastruktūra

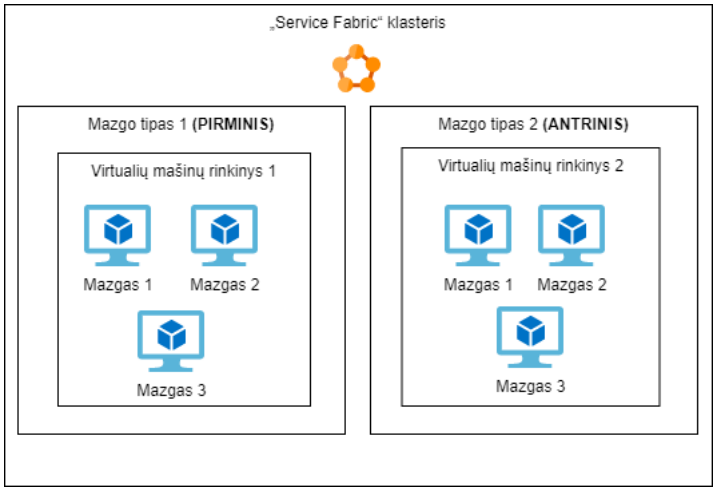
Šios platformos infrastruktūros konceptą apibrėžia 4 pagrindiniai komponentai [Cor18]:

- Klasteris (angl. – *Cluster*) – tinklu sujungtas išplečiamas rinkinys virtualių ar fizinių mašinų, kuriose diegiami „Service Fabric“ programų paketai, gali turėti kelis priskirtus mazgo tipus;
- Mazgo tipas (angl. – *Node Type*) – komponentas, kuris sujungia „Service Fabric“ klasterį su „Azure“ virtualių mašinų rinkiniu. Gali turėti ne daugiau nei vieną priskirtą virtualių mašinų rinkinį;
- Virtualių mašinų išplėtimo rinkinys (angl. – *Virtual Machine Scale Set*) – virtualių mašinų kolekcija, kurioje galima pridėti/trinti mašinas;
- Mazgas (angl. – *Node*) – virtuali arba fizinė mašina, kuri yra sudedamoji klasterio dalis, turi priskiriamą vardą;

12 paveikslėlyje matomas infrastruktūros komponentų išsidėstymas. Vienas mazgo tipas visada būna pirminis, o kiti – antriniai. Nerekomenduojama keisti pirminio mazgo tipo virtualių mašinų korekcijos virtualias mašinas jau paleidus klasterį, nes pakeitimas gali turėti neigiamos įtakos sklandžiam klasterio veikimui. Rašto darbe prototipas sudiegtas į vieno mazgo tipo klasterį,

[6] AWS – Amazon Web Services

tai leido sutaupyti kaštų – ši paslauga nerekomenduojama produkcinėms aplinoms, tačiau skirta testavimo tikslams.



12 pav. „Service Fabric“ infrastruktūra [Cor18]

„Azure Service Fabric“ infrastruktūra palaiko tiek „Windows“, tiek „Linux“ operacines sistemas, rašto darbe naudojamas „Windows“ klasteris, kuris reikalauja daugiau minimalių sistemos resursų. Minimalūs rekomenduojami reikalavimai „Windows“ klasteriui yra šie:

- 3,5 GB RAM operatyviosios atminties;
- 50 GB SSD laikinos disko vietos;
- 1 vienetas 2.4 GHz Intel Xeon vCPU procesorius;

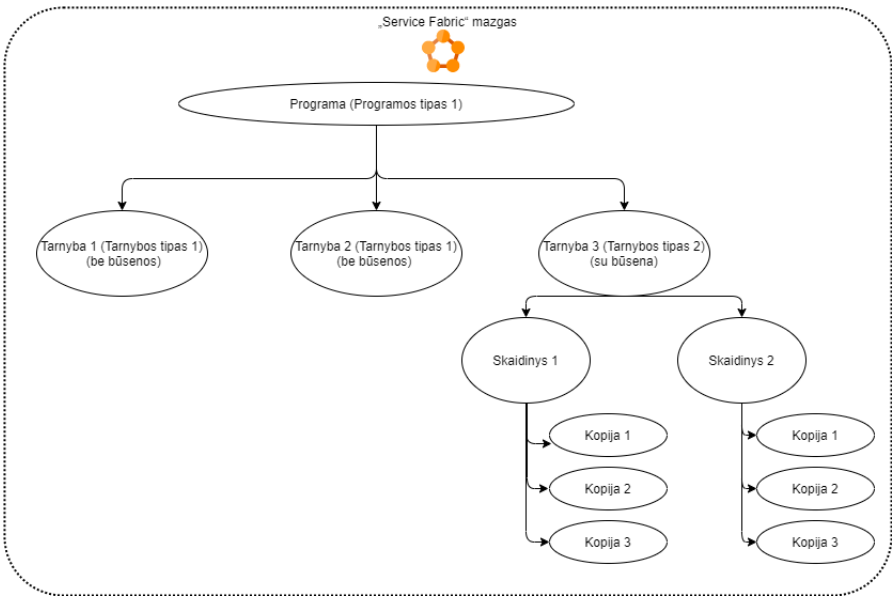
4.1.2. „Service Fabric“ programavimo modelis

„Service Fabric“ programavimo modelį sudaro šie pagrindiniai komponentai [Cor18]:

- Programa (angl. – *Application*) – rinkinys savarankiškų tarnybų (angl. – *Services*), kurios atlieka tam tikrą funkciją sistemoje;
- Tarnyba (angl. – *Service*) – tam tikros funkcijos atlikimo vienetas, mikroservisų architektūros vienetas;
- Skaidinys (angl. – *Partition*) – tarnybos duomenų skaidymo loginis vienetas, šiame rašto darbo prototipe nenaudojamas, todėl plačiau neaptariamas;
- Kopija (angl. – *Replica*) – duomenų kopijos, naudojamos tarnybose su būsena (angl. – *stateful services*), šio rašto darbo prototipo įgyvendinime nenaudojamos;

„Azure Service Fabric“ programavimo modelis apibrėžia 4 tipų tarnybas:

1. Tarnybos be būsenos (angl. – *Stateless services*) – tarnybos, kurios operatyvioje atmintyje nesaugo ir nenaudoja duomenų, visi duomenys išoriniai – gali būti SQL duombazė ar NoSQL talpykla [Cor18];
2. Tarnybos su būsena (angl. – *Stateful services*) – tarnybos, kurios turi savo priskirtus duomenis ir būseną, kuri yra replikuojama tarp skirtingų mazgų klasteryje [Cor18];



13 pav. „Service Fabric“ programavimo modelis [Cor18]

3. Išorinio kodo tarnybas (angl. – *Guest executables*) – klateryje galima paleisti tarnybas, kurios vykdo išorinį vykdomąjį failą (.exe failą), kuris sukurtas visiškai kitomis programavimo kalbomis, nei vykdomoji „Service Fabric“ aplinka [Cor18];
4. Konteinerių tarnybas (angl. – *Container services*) – tarnybos paketą galima talpinti virtualiame konteineryje (pvz.: „Docker“), kuriame taip pat izoliuota „Service Fabric“ vykdomoji aplinka ir visos reikalingos priklausomos bibliotekos [Cor18] ;

Šiame rašto darbe peržiūros agentai įgyvendinti kaip tarnybos be būsenos, nes visa jiems reikalinga informacija saugoma arba eilėse, arba SQL/NoSQL duombazėse, todėl vidinės būsenos mechanizmai, kuriuos siūlo „Service Fabric“ modelis, nereikalingi.

4.2. Virtualių mašinų rinkiniai

„Service Fabric“ klasteris turi priskiriamus vieną arba daugiau virtualių mašinų rinkinius (angl. – *„Virtual Machine Scale Sets“*), kurie leidžia logiškai grupuoti daug virtualių mašinų, kurias galima valdyti centralizuotai – diegti atnaujinimus, keisti konfigūracija. [Cor19e]. Visos virtualios mašinos rinkinyje yra sukuriamos iš bendro operacinės sistemos atvaizdžio (angl. – *OS image*). [Cor19e].

Kuriant virtualių mašinų išplėtimo rinkinį, automatiškai sukuriamas ir apkrovos išlyginimo (angl. – *load balancing*) resursas, kuris leidžia reaguoti į virtualių mašinų apkrovų duomenis ir tolygiai paskirstyti srautą. [Cor19e]. Taip pat sukuriamas virtualus tinklas, kuris leidžia rinkinyje esančioms virtualioms mašinoms bendrauti vienai su kita.

Įgyvendinant šio rašto darbo prototipą naudojamas klasteris su vienu virtualių mašinų išplėtimo rinkiniu, kuriame sudiegta viena virtuali mašina.

4.3. Eilių mechanizmas

Įgyvendinant išskaidytą peržvalgos robotą, asinchroninę komunikaciją tarp atskirų sistemos komponentų užtikrina eilių mechanizmas, veikiantis FIFO[7] principu. „Azure“ paslaugų modelis siūlo 2 skirtingus eilių sprendimus:

- „Service Bus Queues“ – „Azure Storage“ paslaugų infrastruktūros dalis, paprasta REST programinė sąsaja [Cor19b];
- „Storage Queues“ – „Azure messaging“ infrastruktūros dalis, specializuotai skirta sistemų integracijos šablonų įgyvendinimui (pvz.: – „publish-subscribe“) modelis [Cor19b];

Platesnis šių sprendimų palyginimas matomas lentelėje.

3 lentelė. „Storage Queues“ ir „Service Bus Queues“ palyginimas [Cor19b]

Aspektas	**„Azure Storage Queues“**	**„Azure Service Bus Queues“**
Atsiradimo laikas	Atsirado anksčiau	Pristatyta vėliau, 2014 metais
Eilės talpa	1 TB ir daugiau	<= 80GB
FIFO užtikrinimas	Negarantuotas	Garantuotas
„Long-polling“ palaikymas	Nepalaikoma	Palaikoma
„AMQP 1.0“ standartas	Nepalaikomas	Palaikomas
Automatinis „dead-letter“ mechanizmas	Nepalaikomas	Palaikomas
Eilių skaičius	Neribojamas	10000

Įgyvendinant prototipą rašto darbo autoriaus sprendimu pasirinkta „Azure Service Bus Queues“ technologija dėl šių esminių priežasčių:

1. Palaikoma „long-polling“ technologija – kuriant „Service Fabric“ tarnybas ir jų klausytojus (angl. – *service listeners*) nenorėta cikle naudoti HTTP „polling“ metodo, kuris lemtų sudėtingesnės kodo infrastruktūros poreikį;
2. FIFO palaikymas – norima, kad pirmiau aptikti puslapiai būtų atitinkamai žvalgomi greičiau;
3. Rasta biblioteka, kuri užtikrino „Service Fabric“ ir „Service Bus“ paslaugų integraciją be papildomo kodo rašymo[8];

4.3.1. Pasirinktos technologijos rizikos

[MZ14] siūlytoje architektūroje buvo naudojama „Azure Storage Queues“ paslauga, pagrindinės rizikos, kurios gali kilti naudojant rašto darbo autoriaus pasirinktą technologiją yra susijusios su apribotu plečiamumu, nes „Service Bus“ nepalaiko daugiau nei 10000 eilių, o šiame roboto prototipe kiekvienas domenas turi dinamiškai sukuriamą agento eilę. Taip pat apribotas eilės dydis (80GB), nes platus žvalgymas apima dešimtis milijonų puslapių.

[7] First-In-First-Out

[8] Service Fabric Service Bus – https://github.com/loekd/ServiceFabric.ServiceBus

4.4. Reliacinė duombazė

Vienas iš prototipo komponentų – agentų registras – realizuotas panaudojant griežtą schemą palaikančios „Microsoft SQL Server“ reliacinės duomenų bazės atitikmenį „Azure“ platformoje, kuris veikia PaaS[9] principu – „Azure SQL“ duomenų bazės serverį. Ši debesų kompiuterijos paslauga leidžia kontroliuoti ugniasienės nustatymus, kad tik tam tikrų IP adresų klientai galėtų komunikuoti su duomenų baze, todėl įgyvendinant prototipą, virtualių mašinų kolekcijos virtualus tinklas buvo pridėtas kaip SQL duomenų bazės ugniasienės taisyklė. Platforma taip pat palaiko automatinį resursų išplečiamumą, kuris gali būti vykdomas reaguojant į naudojimo apkrovų statistiką.

4.4.1. Duombazės pajėgumo metrika

Kuriant „Azure SQL“ serverį svarbu atsižvelgti, kokio galingumo duombazės serverio sistemai reikia. Kadangi ši paslauga veikia PaaS principu, duombazės serverio pajėgumas nėra nustatomas tradicine procesoriaus branduolių, operatyviosios atminties ir disko talpos kombinacija. Vietoje to naudojama abstraktesnė metrika, pavadinimu „Data Trancation Unit“ (toliau – DTU) [Mal17].

DTU apibrėžimas pagal „Microsoft“ dokumentaciją – maišyta procesoriaus, operatyviosios atminties, duomenų ir tranzakcijų įrašų I/O skaičiaus metrika. [Mal17] Dvigubinant DTU skaičių atitikamai dvigubinamas duomenų bazei priskiriamų resursų pajėgumas.

4.5. NoSQL saugykla

Pastovus aplankytų URL adresų sąrašas, taip pat svetainių Javascript atvaizdavimo analizavimo kontrolinės sumos saugomos „Azure Table Storage“ NoSQL duomenų bazėje. Ši paslauga yra „Key-value“ tipo NoSQL duombazės tipo.

Tokio tipo duomenų bazės yra tam tikra prasme maišos lentelės (angl. – *hash tables*) ir pasižymi [Cor19a]:

- Itin greitos užklausos pagal rakto reikšmę (ieškoti pagal reikšmę nėra efektyvu, lyginant su reliacinėmis duomenų bazėmis ir jų WHERE sakinių konstruktais);
- Nėra duomenų bazės schemos, todėl lengva prisitaikyti prie evoliucionuojančių sistemų ir duomenų;
- Lengvą išplėsti tokios duomenų bazės talpą pagal poreikius, nes duomenys paskirstomi skirtinguose fiziniuose mazguose, sujungtuose tinklu;
- Nepalaiko sudėtingų paieškos operacijų, reikalaujančių duomenų apjungimo, išorinių raktų ar duombazės procedūrų;

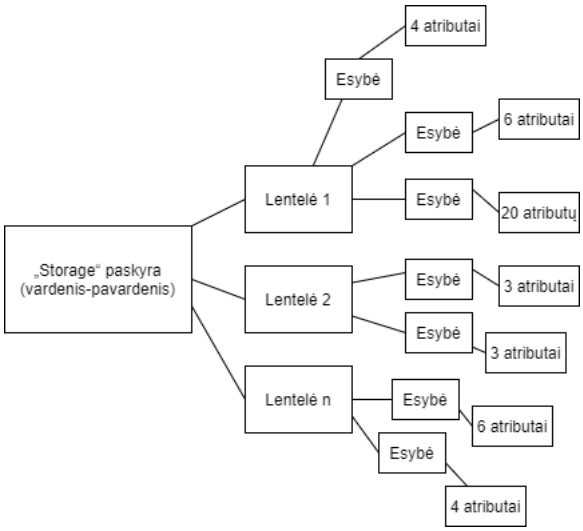
[9] Platform-as-a-service

4.5.1. „Azure Storage Table“ modelis

Šios paslaugos esmę apibrėžia 4 pagrindiniai elementai [Cor19d]:

- *„Storage“ paskyra* – unikalus resurso savininko indentifikavimo ir autorizavimo mechanizmas;
- *Lentelė* – paskyroje esantis obektų rinkinys, neturintis griežtos atributų schemos;
- *Esybė* – atributų rinkinys, panašus į reliacinių duomenų bazių eilutę;
- *Atributas* – (angl. – *properties*) rakto-reikšmės pora, nusakanti esybės konkretų požymį;

Šių esybių sąryšis pavaizduotas 14 paveikslėlyje:



14 pav. „Storage Table“ modelis

Aprašytos paslaugos modelyje esybė palaiko daugiausiai iki 255 atributų, iš kurių 3 yra sisteminiai ir privalomi: [Cor19d]

- *PartitionKey* – pirmoji sudėtinė pirminio rakto dalis, skaidinys, kuris leidžia skaidyti saugyklos duomenis topologiškai skirtinguose mazguose ir lemia didelį tokių duomenų bazių išplečiamumą;
- *RowKey* – antroji sudėtinė pirminio rakto dalis, turi būti unikali kiekviename skaidinyje;
- *Timestamp* – paskutinė esybės koregavimo data ir laikas;

PartitionKey	RowKey	Timestamp	puslapis	paveikslėlio tipas
abc.lt	12485afec158	2020-04-10T10:15:14	/puslapis.html	
def.lt	feac1c54ef4ef	2020-04-11T23:50:59	/img1.jpg	jpeg

15 pav. „Storage Table“ lentelės struktūra [Cor19d]

Kaip matome iš 15 lentelės, esybių atributų skaičius lentelėje gali skirtis, nes nėra apibrėžtos schemos.

5. Prototipo realizacija

Šiame skyriuje detaliai aprašomi pasirinkti architektūriniai sprendimai įgyvendinant debesų kompiuterijos technologijomis paremtą saityno peržiūros robotą.

5.1. Reikalavimai prototipinei sistemai

Šiame poskyryje pagal išnagrinėtą teorinę medžiagą apibrėžiami reikalavimai prototipinei sistemai.

5.1.1. Funkciniai reikalavimai

Žemiau, 4 lentelėje pateikiami sudaryti prototipo funkcionalumo reikalavimai, atsižvelgiant į pirmame ir antrame skyriuose analizuotą peržiūros robotų literatūros teorinę medžiagą.

4 lentelė. Funkciniai prototipo reikalavimai

Numeris	**Reikalavimo aprašymas**
FR1	Paduoti pradinį peržiūros adresų sąrašą
FR2	HTTP protokolo pagalba nuskaityti svetainės turinio kodą ir surasti visus saitus (angl. „hyperlinks“)
FR3	Atlikti rekursyvų puslapių peržiūrėjimą „į plotį“ paremtu algoritmu
FR4	Sukelti rastus naujus saitus į žvalgytinų adresų sąrašo eilę
FR5	Filtruoti žvalgytinus adresus pagal dominančius nuorodų kriterijus (plečiamumas)
FR6	Atsižvelgti į „Robots Exclusion“ protokolą (robots.txt) žiniatinklio serveriuose
FR7	Identifikuoti sistemą naudojantis User-Agent HTTP antraštės lauku
FR8	Saugoti peržiūros rezulatus svetainių indekso repozitorijoje
FR9	Įgyvendinti mandagios peržiūros politiką: žiniatinklio serverio žvalgymo greitį nuspręsti iš HTTP atsako laiko, svetainės populiarumo metrikos
FR10	Dinamiškai koreguoti roboto peržiūros pajėgumus
FR11	Išgauti saitų nuorodas iš dinaminių Javascript atvaizdavimą naudojančių žiniatinklio svetainių
FR12	Identifikuoti jau aplankytas nuorodas

5.1.2. Nefunkciniai reikalavimai

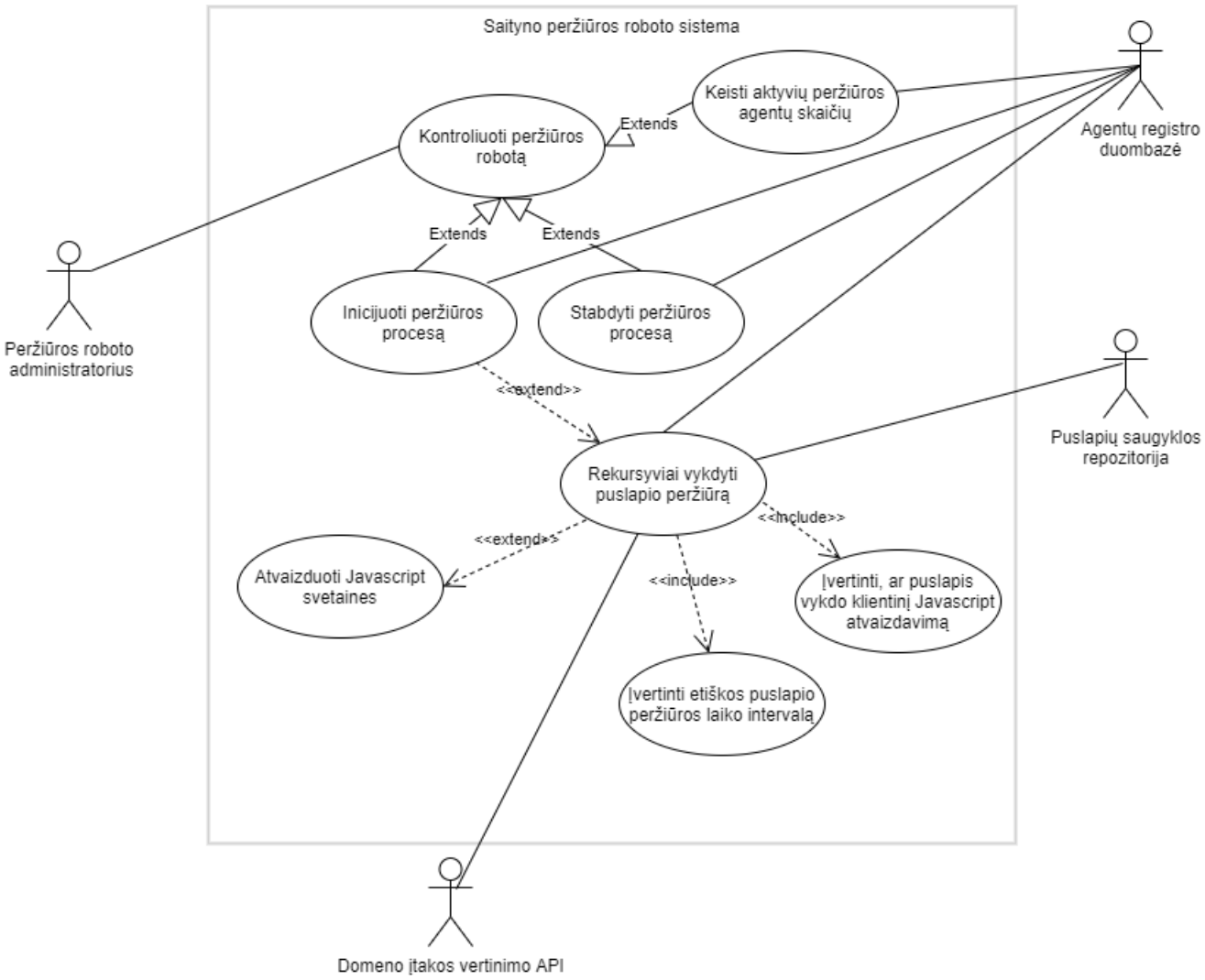
Šis skyrius aprašo praktinę rašto darbo pusę – debesų kompiuterijos technologijomis grįstos saityno peržiūros roboto realizacijos sprendimus. Architektūra ir naudojami įrankiai parinkti pagal 4 skyriuje apibrėžtus sistemos reikalavimus.

5 lentelė. Nefunkciniai prototipo reikalavimai

Numeris	**Reikalavimo aprašymas**
NFR1	Naudoti tik debesų kompiuterijos viešų tiekėjų PaaS, SaaS, IaaS sprendimus
NFR2	Žvalgyti bent 100 puslapių per sekundę greičiu (250 milijonų puslapių per mėnesį)
NFR3	Debesų kompiuterijos infrastruktūros paslaugų kaštų visuma neturi viršyti 0,000001 EUR už puslapį

5.2. Sistemos panaudojamumas

Įgyvendinant prototipo architektūrą pagal išsikeltus reikalavimus, apibrėžtos pagrindinės veiklos, kurios sudaro sistemos funkcionalumą, jos matomos 16 paveikslėlyje pavaizduotoje UML panaudojamumo diagramoje.



16 pav. UML peržiūros roboto prototipo panaudojamumo diagrama

Svarbu paminėti keletą aiškinamųjų aspektų apie 16 diagramoje pavaizduotas veiklas ir aktorius.

Peržiūros roboto administratorius – klientinė programa, kontroliuojanti žvalgymo procesą. Klientas gali būti įvairių formų: komandinės eilutės programa, žiniatinklio, darbalaukio programa. Prototipo realizacijoje klientinė programa imituojama naudojantis *„Service Bus Explorer“*[10] įrankiu ir rankiniu būdu siunčiant žinutes į sistemos žvalgymo kontroliavimo eilę.

Domeno įtakos vertinimo API – išorinis tarnybos serveris, kuris sugeba skalėje nuo 1 iki 100 įvertinti konkretaus domeno autoritetą, kuris reikalingas žvalgymo prioritizavimui. Prototipo įgyvendinimo metu naudota *openrank.io*[11] API paslauga, suteikianti 10000 nemokamų užklausų per 24 valandas. Akcentuotina, kad ši išorinė sistema nėra privaloma – darbe pasirinkta etiško žvalgymo politika atsižvelgia į šią metriką siekiant nustatyti populiaresnes svetaines, kurias galima žvalgyti greičiau.

Atvaizduoti Javascript svetaines – šis panaudojamumo atvejis yra vykdomas kaip atskira veikla po peržvalgos proceso, jei konkretus puslapis pasižymi dinaminės svetainės, kuri turinį generuoja klientinėje pusėje, savybėmis.

[10] https://github.com/paolosalvatori/ServiceBusExplorer
[11] https://openrank.io/

5.2.1. Panaudojamumo atvejų paaiškinimai

6 lentelėje pateikti 16 diagramoje pavaizduotų panaudojamumo atvejų identifikatoriai ir detalesnis paaiškinimas:

6 lentelė. Panaudojamumo atvejų paaiškinimai

Id	**Pavadinimas**	**Aprašymas**
U1.1	Inicijuoti peržiūros procesą	Perkelti pradinį peržiūros URL adresų sąrašą į peržiūros eilę prieš tai patikrinti, ar URL adresai dar neperžiūrėti, ir sukurti pirmąjį peržiūros agentą, jam priskirti domeno vardo zoną
U1.2	Stabdyti peržiūros procesą	Panaikinti visus aktyvius peržiūros robotus ir jų peržiūros eiles
U1.3	Keisti aktyvių peržiūros agentų skaičių	Padidinti arba pamažinti maksimalų leistiną peržiūros ir Javascript atvaizdavimo agentų kiekį sistemoje
U2	Rekursyviai vykdyti puslapio peržiūrą	Aplankyti URL adresu nurodytą svetainę, išnagrinėti jos HTML turinį ir išgauti visas nuorodas, patikrinti, ar jos dar neperžiūrėtos, perduoti jas atitinkam peržiūros robotui
U2.1	Įvertinti, ar puslapis vykdo klientinį Javascript atvaizdavimą	Nustatyti, ar HTML dokumentas naudoja Javascript bibliotekas, kurios naudojamos klientinio atvaizdavimo vieno puslapio žiniatinklio programose, jei taip – nusiųsti tokį URL adresą į Javascript atvaizdavimo eilę
U2.2	Įvertinti etiškos puslapio peržiūros laiko intervalą	Atsižvelgti į HTTP užklausos serverio atsakymo laiką, domeno populiarumą, robots.txt nurodytą leistiną peržiūros intervalą
U2.3	Atvaizduoti Javascript svetaines	Naudojant Javascript variklį atvaizduoti pilną HTML turinį svetainėms, kurios naudoja klientinio atvaizdavimo bibliotekas

5.2.2. Atsekamumo matrica

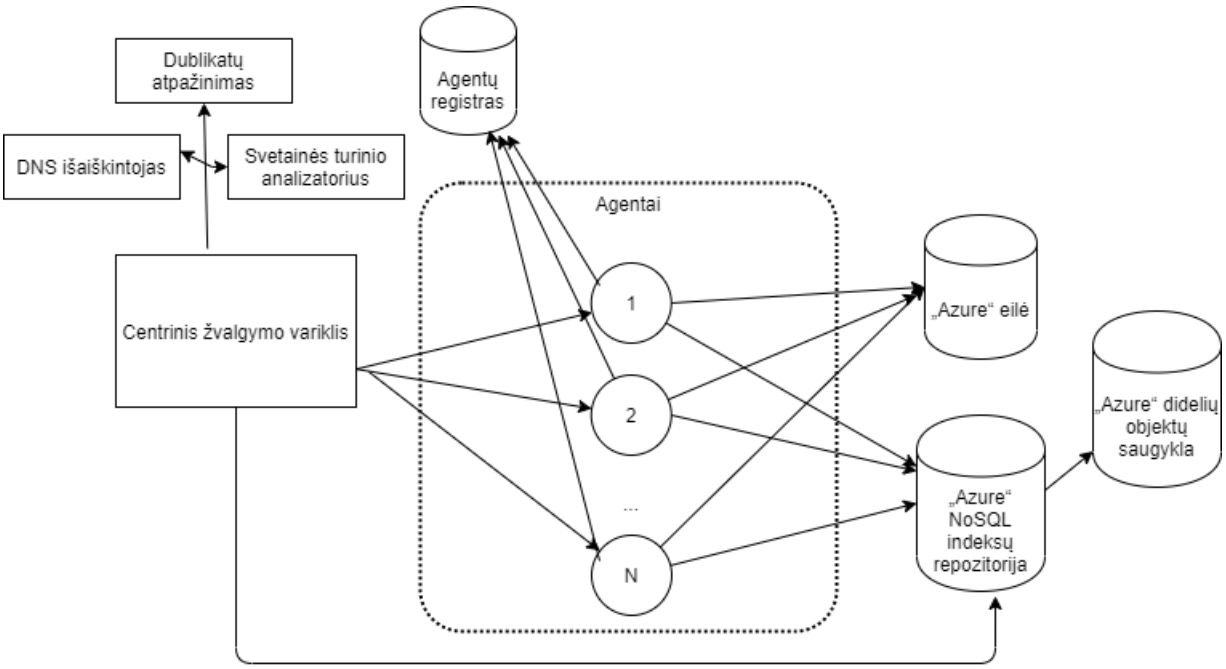
7 lentelėje pateikta funkcinių, nefunkcinių reikalavimų ir nurodytų panaudojamumo atvejų atsekamumo matrica siekiant nurodyti, jog visi apibrėžti reikalavimai yra padengiami įgyvendinant peržiūros roboto prototipą.

7 lentelė. Reikalavimų ir panaudojamumo atvejų atsekamumo matrica

	U1.1	**U1.2**	**U1.3**	**U2**	**U2.1**	**U2.2**	**U2.3**
FR1	**X**						
FR2				**X**			
FR3				**X**			
FR4	**X**			**X**			
FR5				**X**			
FR6				**X**		**X**	
FR7				**X**			**X**
FR8				**X**			**X**
FR9						**X**	
FR10			**X**				
FR11					**X**		**X**
FR12	**X**			**X**			**X**
NFR1	**X**	**X**	**X**	**X**	**X**	**X**	**X**
NFR2			**X**				
NFR3							

5.3. Naudojama architektūra

Prototipas rengiamas plačiai remiantis Kalifornijos Mersedo universiteto mokslininkų literatūrine medžiaga apie debesų kompiuterijos išskirstyto saityno žvalgymo roboto sistemą ir jos architektūrą, dizaino principus [MZ14]. Atsižvelgiant į darbo rašymo metus ir patobulėjusias debesų kompiuterijos tiekėjų siūlomas technologijas, pasiūlyta architektūra realizuojama pritaikius specifinius pokyčius pakeičiant realizacijos įrankius ir platformas. Šiame poskyryje aprašomi pasiūlytos architektūros pagrindiniai komponentai, pavaizduoti 17 diagramoje ir jų funkcinės atsakomybės.



17 pav. Mersedo universiteto mokslininkų pasiūlyta saityno peržiūros roboto architektūra [MZ14]

5.3.1. Centrinis žvalgymo variklis

Tai pagrindinis sistemos struktūrinis komponentas, kuris inicijuoja visą žvalgymo procesą ir kuria/naikina žvalgymo agentus. Pagrindiniai šio komponento veikimo etapai:

1. Inicializacija
   (a) Paimami pradiniai URL adresai iš žvalgymo adresų sąrašo
   (b) Adresai sudedami į „Azure“ eilę prieš tai patikrinant, ar kiekvienas URL adresas dar nebuvo žvalgytas (naudojamas *„Azure NoSQL Table Storage“*)
   (c) Sukuriamas pirmasis žvalgymo agentas, jam priskiriama jo žvalgymo zona (svetainės serverio vardas)
2. Ciklinis žvalgymo procesas
   (a) Iš žvalgymo eilės paimamas URL adresas
   (b) Patikrinama, ar esamam URL adresui egzistuoja aktyvus agentas (SQL Agentų registro duombazė)
   (c) Jei agentas egzistuoja, URL adresas ignoruojamas, nes priskirtas agentas jį apdoros
   (d) Jei aktyvaus agento nėra, paskaičiavus aktyvių agentų skaičių $A_{aktyvūs}$ tikrinamas maksimalus leistinas agentų skaičius $A_{max} > A_{aktyvūs}$, jei sąlyga patenkinta, bandoma sukurti naują agentą ir jam priskirti URL adreso vardo zoną. Kitu atveju laukiama, kol sąlyga bus patenkinta (agentai baigs savo darbą).
   (e) Kartojama nuo pirmo žingsnio

5.3.2. Žvalgymo agentas

Šis komponentas iš žvalgymo eilės ima URL adresą, jei adreso vardo sritis sutampa su agentui priskirtu aptarnavimo vardo adresu, jis inicijuoja HTTP užklausą į nurodytą žiniatinklio serverį, parsiunčia HTML turinį, išgauna visas jame esančias nuorodas, kiekvienai jų patikrina, ar nuoroda dar neregistruota indeksų repozitorijoje ir atitinkamai perduoda aktyviam žvalgymo agentui, kurio zona sutampa su URL adreso vardo zona [MZ14]. Agentai užtikrina išskirstytos sistemos veikimo principą, juos galima dinamiškai pridėti, šalinti. Apibrėžtoje architektūroje nėra centrinio komunikacijos komponento, kuris deleguotų žvalgymo darbus agentams.

5.3.3. Agentų registras

Tai SQL duombazė, kurioje registruojami visi aktyvūs agentai, taip pat jie periodiškai išregistruojami aptikus, jog agentas kurį laiką nebuvo aktyvus. Šis komponentas yra bendras, todėl rašto darbo paskutiniame skyriuje, atliekant tyrimą, bus analizuojama, ar jis negali lemti „butelio kaklelio“ efekto.

5.3.4. „Azure“ žvalgymo eilė

Tai komponentas, kuris saityno žvalgymo sistemų literatūroje minimas kaip „žvalgymo pasienio“ (angl. – *Crawling Frontier*) komponentas, kuriame laikinai kaupiami žvalgytinų puslapių URL adresai [Cas04]. Žvalgymo agentai ir Centrinis žvalgymo variklis klausosi šios eilės žinučių ir jas apdoroja. Tai dar vienas bendrinis sistemos komponentas, todėl eksperimento metu taip pat bus stebimas jo veikimo pralaidumas. [MZ14] moksliniame darbe jau buvo atliktas šio komponento tyrimas ir įsitikinta, kad didinant žinučių kiekio apkrovą, komponento pralaidumas išlieka gana patovus. Naudota „Azure Storage Queues“ eilių paslauga. Šio rašto darbo tyrime naudojama „Azure Service Bus“ eilių infrastruktūra, todėl taip pat pakartotinai bus stebimas komponento veikimo pralaidumas.

5.3.5. „Azure“ NoSQL indeksų repozitorija

Tai NoSQL rakto-reikšmės (angl. – *Key-Value*) tipo saugykla, kurioje agentai registruoja identifikuotus svetainių URL adresus, taip pat keičia tų adresų žvalgymo statusą, aptiktų URL nuorodų paminėjimų skaičių (angl. – *hit count*) [MZ14].

5.3.6. „Azure“ didelių objektų saugykla

Šis komponentas aprašytoje architektūroje atsakingas už didelių multimedijos failų saugojimą – nuotraukų, vaizdo įrašų, PDF dokumentų ir kitų žiniatinklio resursų, kurie nėra HTML dokumentai [MZ14]. Pasižymi tuo, jog gali talpinti didelius kiekius duomenų.

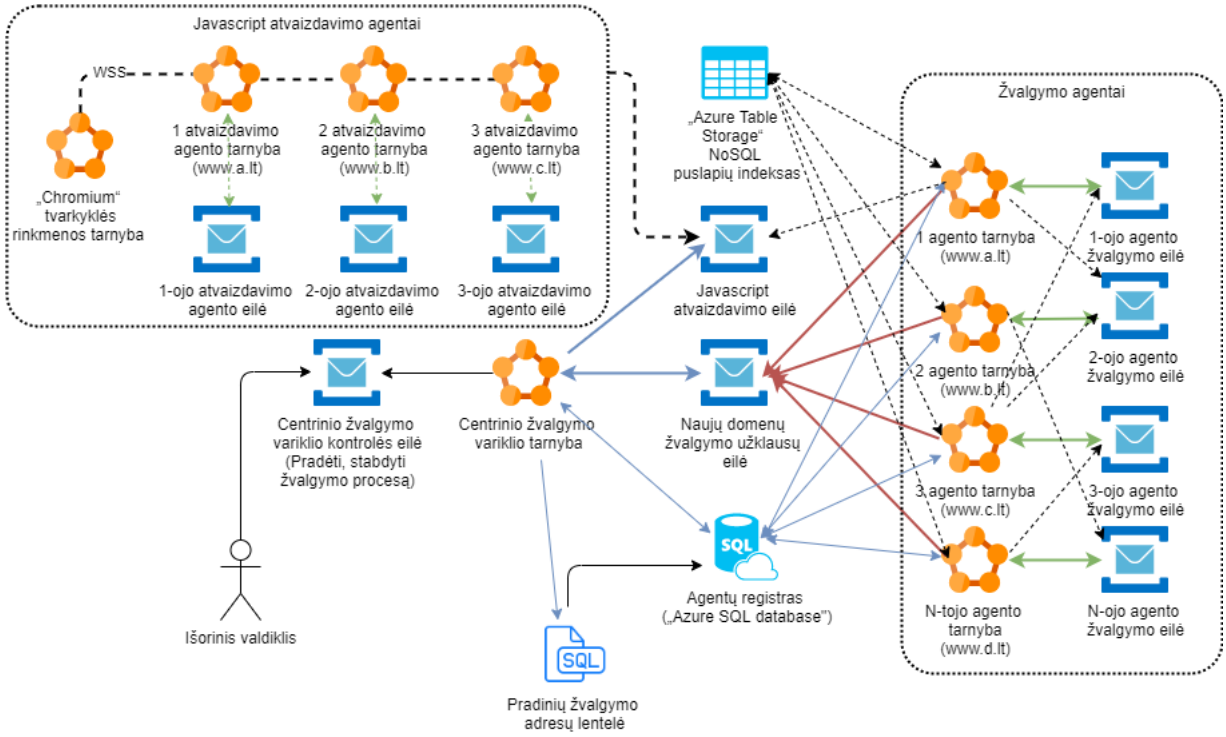
5.3.7. Svetainės turinio analizatorius ir dublikatų aptikimas

Vienas iš minimų sistemos komponentų – svetainės turinio analizatorius – semantiškai apdoroja nuskaitytą HTML dokumento turinį siekdamas nustatyti puslapio tematiką, pagrindinius raktinius žodžius, taip pat identifikuoti pasikartojančio, dublikuoto turinio svetaines. Šio prototipo įgyvendinimo ribose nebuvo numatytas semantinis svetainės turinio analizavimas, todėl plačiau šie komponentai aptariami nebus. Prototipo įgyvendinimo ribose nuamtytas tik URL adresų dublikatų aptikimas siekiant išvengti bereikalingos pakartotinės resurso žvalgymo naštos.

5.4. Struktūrinė prototipo schema

Šiame poskyryje aprašoma koreguota [MZ14] mokslininkų pasiūlyta išskirstyto saityno peržiūros roboto struktūrinė schema.

Pateiktoje 18 schemoje detalizuojami esminiai sistemos struktūriniai komponentai ir jų „Azure“ platformos infrastruktūros paslaugų rūšys.



18 pav. Struktūrinė prototipo sistemos schema

5.4.1. Duomenų saugojimo talpyklos

Prototipas naudoja dvi talpyklas agentų informacijai ir puslapių peržiūros būsenai saugoti.

5.4.1.1. Agentų registras

„Azure SQL“ duomenų bazė, kurioje apibrėžta:

- Peržiūros agentų registracijos lentelė

- Javascript atvaizdavimo agentų registracijos lentelė
- Pradinis peržiūros URL adresų sąrašas
- Javascript klientinio atvaizdavimo karkasų raktažodžių sąrašo lentelė, pagal kuriuos atliekama Javascript atvaizdavimo poreikio indentifikacija

8 lentelė. Agentų registro lentelės schema [MZ14]

Id	**Agento_Vardas**	**Agento_Aptarnavimo_Sritis**	**Paskutinis_Aktyvumas**	**Ištrintas**
1	A1	abc.lt	2020-05-02 15:30:15	false
2	A2	def.lt	2020-05-02 13:30:15	true

8 lentelėje pavaizduotame agentų registre raudonai pažymėtas paskutinio aktyvumo laiko stulpelis originaliai [MZ14] literatūroje neminėtas, tai – rašto darbo autoriaus prototipo planavimo metu įgyvendinta korekcija siekiant atlaisvinti nebenaudojamus agentus.

Įgyvendintas algoritmas, atlaisvinantis nebenaudojamus agentus, apibrėžiamas šiais žingsniais:

1. Gavus naujo domeno žvalgymo užklausą, centrinis žvalgymo variklis tikrina visų aktyvių agentų paskutinio aktyvumo stulpelio reikšmes
2. Išrenkami visi agentai, kurių stulpelio „Paskutinis_Aktyvumas“ reikšmė yra „NULL“ arba < DABARTINIS_LAIKAS - NUSTATYMUOSE_APIBRĖŽTAS_NEAKTYVUMO_PERIODAS
3. Kiekvienam išrinktam agentui tikrinama, ar jo peržiūros eilė yra tuščia, jei taip – agentas naikinamas ir ištrinama jo peržiūros eilė

5.4.1.2. Puslapių indeksas

Tai – „Azure NoSQL Table Storage“ duomenų bazė, kurioje saugomi visi surasti URL adresai ir jų peržiūros būsena. 9 lentelėje nurodyta peržiūros URL adresų saugojimo lentelė. *PartitionKey* apibrėžia domeno vardo sritį, o *RowKey* – MD5 kontroline suma užkoduotas pilnas URL adresas, kuris unikaliai identifikuoja puslapį jo domeno zonoje.

9 lentelė. Peržiūros URL adresų lentelė

PartitionKey	RowKey	PageUrl	Timestamp	Visited	StatusCode	Rendered
abc.lt	0575118B9D...	abc.lt/naujienos/	2020-05-01T...	TRUE	200	TRUE
def.lt	52F8F6432...	def.lt/orai	2020-05-10T...	FALSE		FALSE

10 lentelėje pavaizduota skriptų saugojimo informacijos talpykla, kurioje saugomos HTML identifikuotų skriptų kontrolinės sumos ir Javascript atvaizdavimo sprendimas.

- *PartitionKey* – domeno vardo sritis
- *RowKey* – skripto URL adreso MD5 kontrolinė suma
- *Checksum* – skripto turinio MD5 kontrolinė suma
- *ScriptFile* – skripto pavadinimas
- *RenderDecision* – indikacija, ar rasta klientinio atvaizdavimo požymių

10 lentelė. Skriptų informacijos saugojimo lentelė

PartitionKey	RowKey	Checksum	ScriptFile	RenderDecision
abc.lt	0575118B9D...	3d572837...	bundle.js	TRUE
def.lt	52F8F64324...	999101d8...	jquery.js	FALSE

5.4.2. Žvalgymo agentai

Žvalgymo agentai įgyvendinti kaip „Service Fabric“ tarnybos be būsenos (angl. – *Stateless services*). Kiekvienas iš jų turi savo atitinkamą eilę, iš kurios ima žvalgytinų puslapių URL adresus. Priskirtoje „Service Bus“ eilėje talpinami tik tie puslapių URL adresai, kurie priklauso agentui priskirtai domeno vardo sričiai.

5.4.3. Centrinis žvalgymo variklis

Šis komponentas taip pat įgyvendintas, kaip „Service Fabric“ tarnyba be būsenos, jis sukuriamas iš karto paleidus klasterį ir yra atsakingas už naujų agentų ir jų žvalgymo eilių kūrimą/naikinimą. Šis komponentas klausosi 3 skirtingų „Service Bus“ eilių ir apdoroją jų žinutes:

- **Centrinio žvalgymo variklio kontrolės eilė** – į šią eilę roboto klientinė sąsaja talpina peržiūros komandas: pradėti, stabdyti peržiūros procesą, keisti aktyvių agentų skaičių
- **Naujų domenų žvalgymo užklausų eilė** – tai pagrindinė „žvalgymo pasienio“ eilė, kurioje kaupiami visi URL adresai neturintys aktyvaus peržiūros agento. Jeigu agentas visgi egzistuoja, žvalgymo variklis atvykusį URL adresą peradresuoja jo atitinkamai žvalgymo eilei. Kitu atveju, žvalgymo variklis bando URL adresui kurti naują agentą, jei neviršytas maksimalus aktyvių agentų limitas
- **Javascript atvaizdavimo eilė** – šioje eilėje talpinami URL adresai tų puslapių, kuriems buvo nustatyta indikacija, jog jie naudoja klientinį Javascript atvaizdavimo karkasą. Centrinis žvalgymo variklis gavęs žinutę iš šios eilės, patikrina, ar maksimalus aktyvių Javascript atvaizdavimo agentų skaičius neviršytas ir bando kurti atvaizdavimo agentą URL adreso domeno vardo sričiai.

5.4.4. Javascript atvaizdavimo agentai

Šių agentų veikimo principas beveik identiškas žvalgymo agentams – jie turi savo domeno srities atvaizdavimo eiles. Skirtumas, jog pastarieji nenaudoja paprastų HTTP užklausų į URL

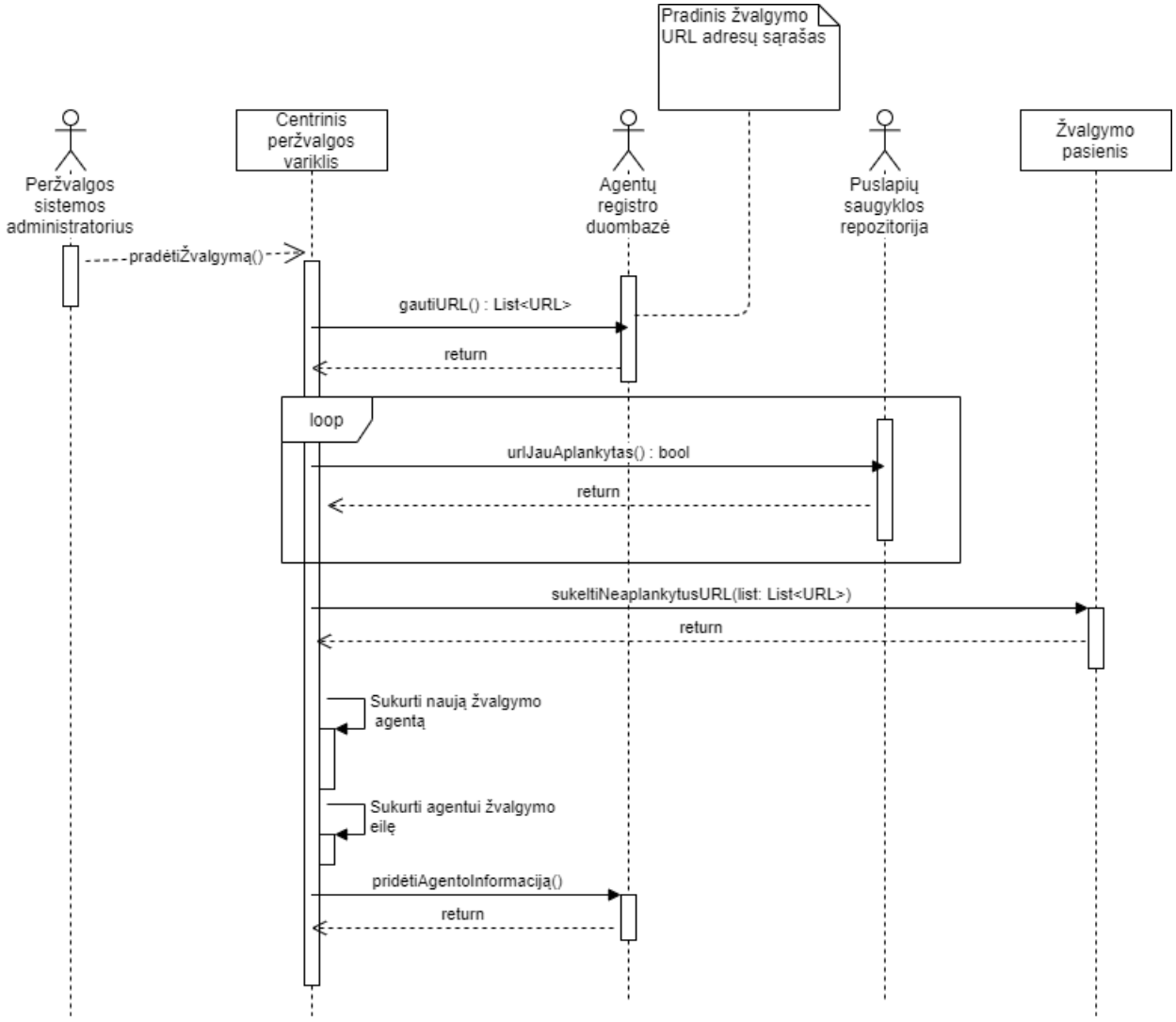
adresu pasiekiamą žiniatinklio serverį, o pasitelkia kitoje „Service Fabric“ tarnyboje esančią „Chromium“ tvarkyklę, kuri turi integruotą Javascript atvaizdavimo variklį. Javascript atvaizdavimo agentai su šia tarnyba bendrauja per „Websocket“ protokolo jungtį, kuri leidžia tinklu kontroliuoti nutolusią naršyklę, tereikia paleidžiant tvarkyklę nurodyti parametrą, apibrėžiantį prisijungimo jungtį: *chrome.exe –remote-debugging-port=9222*.

5.5. Panaudojamumo atvejų dinamika

Šiame poskyryje detaliau grafiškai pristatomos apibrėžtų panaudojamumo atvejų algoritminės sekos, kurios perteikiamos UML sekų diagramomis.

5.5.1. U1.1 Inicijuoti peržvalgos procesą

Žemiau pateiktoje 19 UML sekos diagramoje pavaizduotas sistemos administratoriaus inicijuojamas žvalgymo pradėjimo procesas, kurio metu pradinis URL adresų sąrašas perkeliamas į žvalgymo pasienio komponentą ir sukuriamas pradinis žvalgymo agentas.

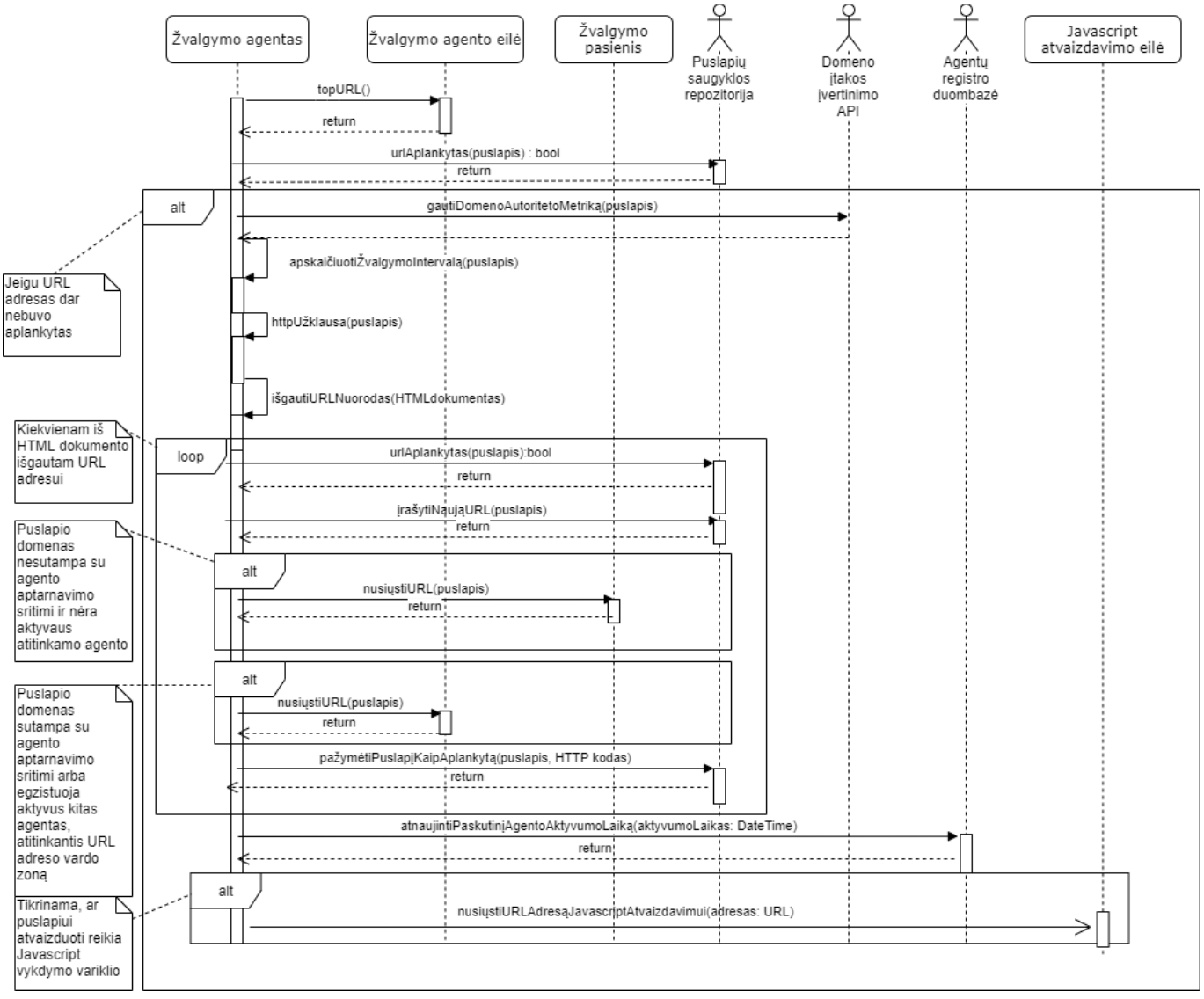


19 pav. Žvalgymo inicijavimo proceso UML sekos diagrama

Peržvalgos sistemos administratorius yra klientinė programa, valdanti robotą. Galima realizuoti tiek žiniatinklio, tiek darbalaukinės programos klientą. Prototipo realizacijos metu klientas simuliuojamas naudojant „Service Bus Explorer“ eilių programą ir rankomis siunčiant žinutes į valdymo eilę.

5.5.2. U2 Vykdyti puslapio peržvalgą

20 UML sekos diagramoje pavaizduota žvalgymo agento darbo vykdymo schema. Akcentuotina, kad šis žvalgymo procesas kartojamas iki to laiko, kai žvalgymo agento eilė tampa tuščia ir po sistemos konfigūracijoje nustatyto agento neaktyvumo intervalo, Centrinis žvalgymo variklis tiesiog panaikina nebeaktyvų žvalgymo agentą.

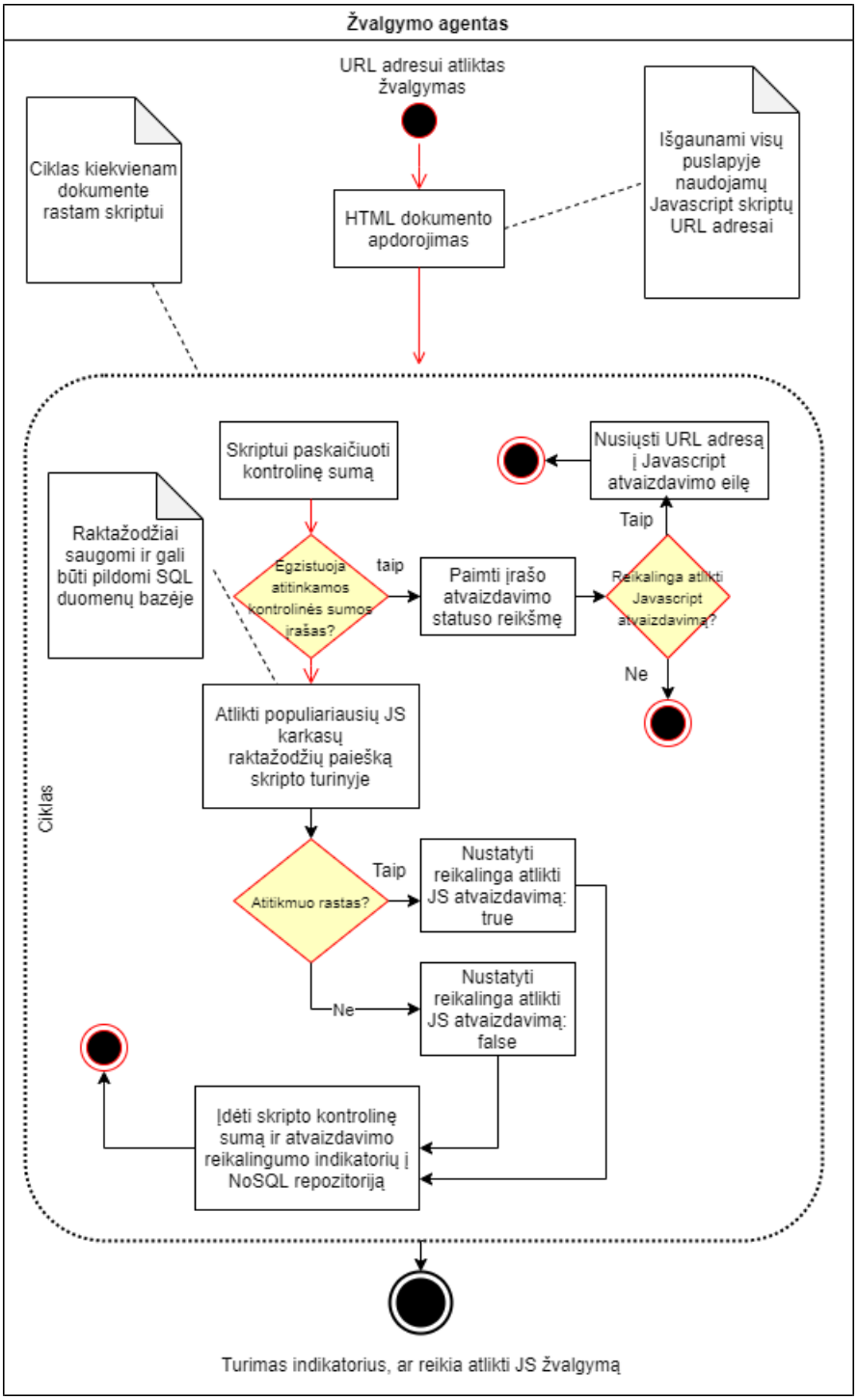


20 pav. Žvalgymo vykdymo proceso UML sekos diagrama

5.5.3. U2.1 Įvertinti, ar puslapis vykdo klientinį Javascript atvaizdavimą

20 diagramoje paminėta, jog URL adresas po žvalgymo į Javascript atvaizdavimo eilę siunčiamas tik nustačius, jog puslapis reikalauja Javascript atvaizdavimo variklio, kad galėtų dinamiš-

kai generuoti DOM medį. Šiame punkte pateikiama UML veiklos diagrama 21, kurioje nurodoma, kaip prototipinė sistema atlieka šį tikrinimą.



21 pav. Poreikio atlikti Javascript atvaizdavimą indikatoriaus nustatymo UML veiklos diagrama

5.6. Kiti prototipo realizacijoje panaudoti įrankiai

Šiame poskyryje aprašomos įgyvendinant prototipą panaudotos bibliotekos, vykdomosios aplinkos, programavimo kalbos.

5.6.1. Programavimo aplinka

„Service Fabric“ klasterio kodas buvo rašytas C# programavimo kalba panaudojant .NET Core 3.1 karkaso, kuris leidžia vienai kodo bazei veikti skirtingose operacinėse sistemose, versiją. Daug panaudotų „NuGet“ paketų valdymo sistemos bibliotekų (pvz.: PuppeteerSharp, AngleSharp) palaiko .NET Standard 2.0 versiją, kuri skirta tiek tradiciniam .NET karkasui, tiek ir .NET Core aplinkai.

5.6.2. Bibliotekos

- *AbotCrawler*[12] – patogi ir aktyviai palaikoma .NET biblioteka, skirta specialiai puslapių žvalgymui. Įgyvendina daug žemo lygio funkcinių žvalgymo detalių (DNS išaiškinimas, žvalgymo intervalai, etiško žvalgymo politiką), todėl programuotojas gali papildomai nebesirūpinti jų įgyvendinimu
- *Dapper* – minimali ORM biblioteka, skirta komunikacijai su duomenų baze. Prototipui nereikėjo ORM bibliotekų, tokių kaip Entity Framework, funkcionalumo, todėl įgyvendinant achitektūrinį repozitorijos sluoksnį panaudotas šis paketas, kuris reikalauja rašyti užklausas rankomis [Dap18]
- *AngleSharp* – HTML dokumento apdorojimo .NET biblioteka, leidžianti serverio pusėje sukurti DOM medį, jį analizuoti, išgauti informaciją, keisti [Ang18]. Panaudota Javascript skriptų išgavimui.
- *PuppeteerSharp* – komunikacijai su „Chromium“ tvarkykle naudojama .NET biblioteka, kuri originaliai sukurta Node.JS platformai [Pup19]. Ši biblioteka naudojama Javascript atvaizdavimo agentų prisijungimui „Websocket“ jungtimi prie „Chromium“ tvarkyklės agento

Skyriuje detaliai išanalizuota prototipo architektūra, kurios realizacijos variantas praktiškai išbandytas tyrimo, aprašomo kitame skyriuje, metu vykdant saityno peržiūrą su skirtingu lygiagrečiai veikiančių agentų skaičiumi.

[12] Abot – https://github.com/sjdirect/abot

6. Prototipo vertinimas

Šiame skyriuje pristatomas tyrimas, kurio metu vertinama įgyvendinto roboto URL adresų peržiūros ir Javascript atvaizdavimo greitaveika, taip pat atskirų sistemos komponentų apkrovos.

6.1. Tyrimo aprašymas

Tiriant saityno peržiūros roboto greitaveiką, mokslinėje literatūroje dažniausiai vertinamas peržiūrėtų URL adresų skaičius per laiko vienetą, taip pat iš viso surastų URL nuorodų skaičius [Gor04]. Dėl to, šio tyrimo metu stebėti pasirinktos šios funkcinės prototipo greitaveikos charakteristikos:

1. **Peržiūrėtų URL adresų skaičius**
2. **Vidutiškai peržiūrėtų URL adresų / s**
3. **Visų surastų URL adresų skaičius**
4. **JS variklio pagalba atvaizduotų URL adresų skaičius**
5. **JS variklio pagalba surastų URL adresų skaičius**
6. **JS variklio pagalba surastų URL adresų dalis (proc.)**

4,5,6 punktai pasirinkti dėl rašto darbo temos išskirtinumo tirti modernių saityno programų žvalgymo greitaveiką ir stebėti, ar reikšmingą surandamų URL adresų dalį sudaro adresai, rasti pasitelkiant JS atvaizdavimo variklio pagalbą.

6.1.1. Vykdytos iteracijos

Vykdytos 7 skirtingos iteracijos (kiekviena iteracija trunka lygiai 1 val.), kurių metu didintas lygiagrečiai veikiančių aktyvių peržiūros ir Javascript atvaizdavimo agentų skaičius tokia tvarka:

1. **1 peržiūros agentas + 1 Javascript atvaizdavimo agentas**
2. **2 peržiūros agentai + 1 Javascript atvaizdavimo agentas**
3. **4 peržiūros agentai + 2 Javascript atvaizdavimo agentas**
4. **8 peržiūros agentai + 4 Javascript atvaizdavimo agentas**
5. **16 peržiūros agentų + 8 Javascript atvaizdavimo agentas**
6. **32 peržiūros agentai + 16 Javascript atvaizdavimo agentas**

Tokia seka tyrimui pasirinkta, nes ji buvo taikyta [MZ14] šaltinyje aprašyto debesų kompiuterijos technologijomis paremto saityno peržiūros roboto, kurio architektūros principais remiasi šio rašto darbo prototipas, tyrime. Dvigubai mažesnis Javascript atvaizdavimo agentų skaičius pasirinktas tikintis, jog Javascript atvaizdavimo poreikis sudarys tik gana nedidelę dalį visų peržiūrėtų URL adresų.

6.1.2. Roboto komponentų apkrovos

[MZ14] moksliniame straipsnyje aprašytame peržiūros roboto tyrime stebėtos architektūroje naudojamų bendrų komponentų apkrovos (komponentų, į kuriuos kreipiasi kiekvienas peržiūros

agentas). Stebėti „Azure Queue“ ir „Azure Table Storage“ komponentų vidutiniai uždelsimų laikai, priklausantys nuo tranzakcijų skaičiaus. Įgyvendinant šio rašto darbo prototipą, vietoje „Azure Queue“ pasirinkta „Service Bus Queues“ eilių infrastruktūros paslauga, kuri neturi vidutinio uždelsimo laiko vertinimo metrikos, dėl šios priežasties eilių mechanizmui stebėti pasirinkta ateinančių užklausų skaičiaus metriką per sekundę (angl. – *Incoming Requests per second*), taip pat atmestų užklausų kiekis (angl. – *Throttled Requests*). „Azure Service Bus Standard Tier“ planas, kuris naudojamas šio prototipo tyrime, apibrėžia maksimalų skiriamą kreditų kiekį per laiko vienetą, t.y. – 1000 kreditų per sekundę [Cor19c]. Viršijus šį limitą, visos perviršiaus užklausos yra atmetamos, kol kreditai atstatomi. Ateinančių užklausų skaičius bus stebimas siekiant nustatyti, kada būtų artėjama prie šios ribos. Tyrimo metu stebimi komponentai ir jų metrikos:

- **„Azure Service Bus Queue“ eilių infrastruktūra (agentų peržiūros eilės)** – ateinančių užklausų skaičius per sekundę ir atmestų užklausų skaičius
- **„Azure Table Storage“ (URL adresų repozitorija)** – tranzakcijų skaičius ir vidutinis uždelsimo laikas (taip pat, kaip stebėta [MZ14] aprašytame tyrime)
- **SQL Agentų registras: DTU[13] išnaudojimas, proc.** – [MZ14] tyrime nenagrinėta, tačiau pasirinkta, nes šio prototipo realizacija stipriai priklauso nuo SQL duomenų bazės (automatinis peržiūros agentų išregistravimas remiantis paskutiniu aktyvumo laiku, saugomu agentų registro lentoje)
- **Išeinantis tinklo srautas, GB (angl. – *egress traffic*)** – [MZ14] tyrime nenagrinėta, tačiau pasirinkta, nes svarbu įvertinti realius piniginius kaštus vykdant saityno peržiūrą, į kuriuos įtraukiamas ir tinklo srautas, paliekantis debesų kompiuterijos tiekėjo duomenų centrus.

6.2. Tyrimo sąlygos

Sistema sudiegta į „Azure Service Fabric“ klasterį. Visi sistemos komponentai sudiegti Šiaurės Europos regiono „Microsoft“ duomenų centruose. Eksperimento metu puslapių peržvalgos ir Javascript atvaizdavimo kiekiai, tap pat sistemos našumo rodikliai (angl. – *Performance Counters*) stebimi ir registruojami naudojantis „Azure Monitor“ infrastruktūra ir jos siūloma „Application Insights“ metrikų stebėjimo paslauga.

6.2.1. Skaičiavimo resursai

Eksperimento metu klasteryje naudojami 3 mazgai, kuruos sudaro atskiros virtualios mašinos. Esant poreikiui visada galima klasteryje pridėti daugiau mazgų. Naudojamos 3 **„Azure F4“** tipo optimizuotos procesoriaus skaičiavimo galios virtualios mašinos, kurių parametrai:

- 4 vCPU (3,4 GHz Intel® Xeon® Platinum 8168)
- 8 GB RAM operatyviosios atminties
- 64 GB SSD laikinos disko vietos (3000 I/O operacijų/sec.)
- 1 Gbps tinklo pralaidumas
- Naudojimosi kaina: 0,18€/val.

[13] DTU – Data Transaction Unit: „Azure“ abstrakti metrika, skirta įvertinti duombazės resursų naudojimą

6.2.2. Reliacinė duombazė

Naudojama žemiausio pajėgumo „Azure SQL“ duomenų bazė, pasižyminti šiomis charakteristikomis:

- 5 DTU vienetai (pralaidumo metrika)
- 2 GB maksimali talpa
- 0,0057€/val. kaina

6.3. Tyrimo rezultatai

Poskyryje aprašomos ir analizuojamos prototipo veikimo stebėjimo metu fiksuotos funkcinės charakteristikos ir komponentų apkrovos.

6.3.1. Funkcinės roboto greitaveikos charakteristikos

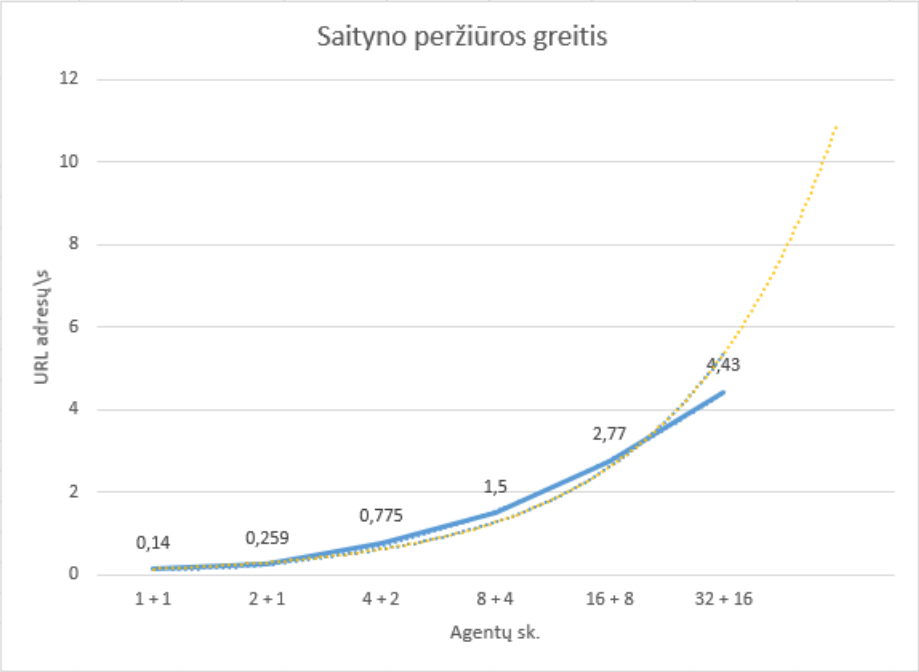
11 lentelėje pateikiami prototipo peržiūros greitaveikos rezultatai. Pažymėtina, jog šioje lentelėje esantys skaičiai parodo paprastos HTTP užklausomis paremtos peržiūros rodiklius, gautus vykdant eksperimentą (be Javascript atvaizdavimo).

Kaip matoma, didinant aktyvių agentų skaičių 2^n laipsniu, saityno peržiūros greitis auga panašiu tempu (didėja 1,5-2 kartais). Tai rodo, jog išplėtimo schema, paremta [MZ14] šaltinyje aprašytoje ir šiame darbe realizuotoje architektūroje, veikia. Taip pat pastebėtina, jog randamų naujų URL resursų adresų skaičius auga dar didesniais tempais. Paskutiniame stulpelyje matomas aplankytų vardų serverių skaičius (angl. – *Web Hosts*) pradinėse iteracijose (1 ir 2) nebūtinai auga didinant agentų skaičių, nes aptikus svetainę su itin dideliu nuorodų kiekiu (pvz. naujienų svetainė), tyrimo metu skirta valanda buvo praleista peržiūrint ją.

11 lentelė. Peržiūros agentų veikimo funkcinės metrikos

Agentai	**Peržiūrėta URL adresų**	**Peržiūros greitis (URL adresų/s)**	**Rasta URL adresų**	**Aplankyta vardų serverių**
1 + 1	504	0,14	2543	3
2 + 1	918	0,259	6584	37
4 + 2	2791	0,775	30169	10
8 + 4	5417	1,5	113530	83
16 + 8	10000	2,77	64990	125
32 + 16	15951	4,43	419115	173

22 diagramoje aiškiau matomas artimas eksponentiniui augimo greitis, nubrėžta trendo linija parodo, kad kitoje iteracijoje numatomas vidutinis peržiūros greitis siektų beveik 12 puslapių per sekundę.



22 pav. Saityno peržiūros greitis didinant agentų skaičių

6.3.1.1. Javascript atvaizdavimo funkcinės metrikos

12 lentelėje matomi eksperimento statistikos duomenys, kurie pasiekti vykdant Javascript atvaizdavimą, t.y. nustačius indikaciją, jog puslapis reikalauja dinaminio, klientinės pusės atvaizdavimo (prototipo realizacijoje identifikavus, jog naudojamas klientinis karkasas).

12 lentelė. Javascript agentų peržiūros funkcinės metrikos

Agentai	Peržiūrėta URL adresų (JS)	Peržiūrėta URL adresų (JS), proc.	Rasta URL adresų (JS)	Rasta URL adresų (JS), proc.
1 + 1	1	0,20%	20	0,79%
2 + 1	1	0,11%	4	0,06%
4 + 2	246	8,81%	2147	7,12%
8 + 4	213	3,93%	870	0,77%
16 + 8	1614	16,14%	10500	16,16%
32 + 12	5458	34,22%	47016	11,22%

Raudonai pažymėtos eilutės demonstruoja iteracijas, kurių metu agentai visą skirtą eksperimentui laiką peržiūrinėjo ribotą kiekį vardų serverių, neturinčių klientinio atvaizdavimo ir turinčių didelį kiekį puslapių, dėl to demonstruoti netendencingi rodikliai (nėra augimo). Trečio ir penkto stulpelių kombinacija demonstruoja realizuoto modernių žiniatinklio programų žvalgymo, panaudojant „Chromium“ tvarkyklę ir jos naudojamą Javacript V8 variklį, naudą:

- Pasitelkiant „Chromium“ tvarkyklę vidutiniškai panaudojant JS variklį pakartotinai peržiūrėta tik 10,53% puslapių – tai leido sutaupyti daug procesoriaus ir operatyviosios atminties, taip pat tinklo resursų (tiek puslapių gavo klientinio atvaizdavimo poreikio indikaciją)

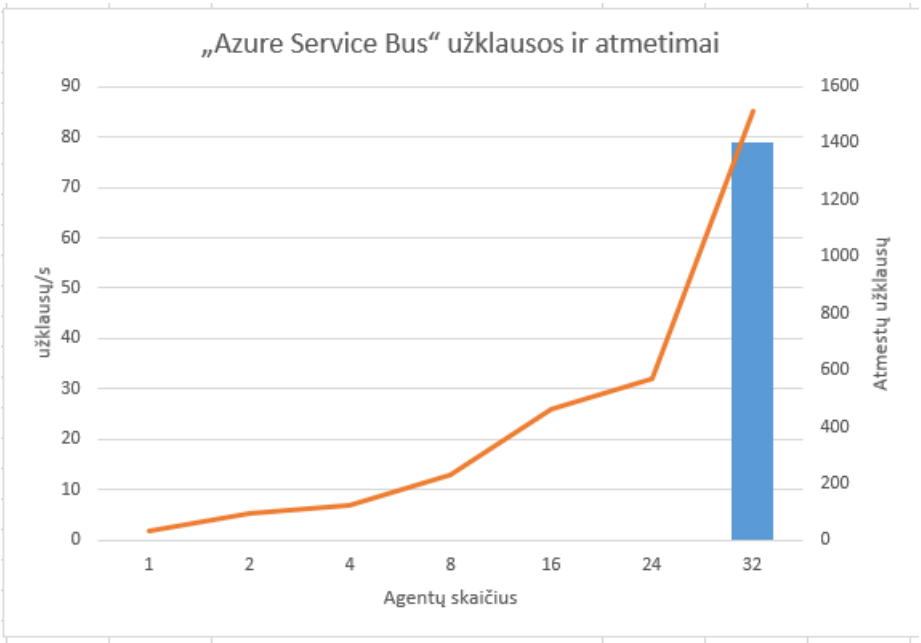
- Javascript atvaizdavimo metu rasta vidutiniškai 6,02% visų žvalgymo iteracijos metu aptiktų URL adresų. Šie adresai nebūtų aptikti naudojant vieną iš tradicinių literatūroje aprašomų peržiūros robotų (pvz. [Gor04] ar [HN99b], t.y. be „Chromium“ tvarkyklės pagalbos, todėl roboto URL duomenų bazė papildyta šiomis URL nuorodomis

6.3.2. Bendrų peržiūros roboto komponentų apkrovos

Šiame punkte aiškinamasi, ar kuris nors bendrasis peržiūros komponentas nelemia „butelio kaklelio“ efekto – neartėjama prie veikimo limito ribos.

6.3.2.1. „Service Bus“ eilių infrastruktūra

23 diagramoje pavaizduoti „Service Bus“ eilių stebėjimo duomenys (agentų peržiūros, kūrimo, atvaizdavimo eilės). Į užklausas įsiskaičiuoja ir eilių valdymo operacijos – eilių agentams kūrimas, naikinimas. Jau minėta, jog „Service Bus“ bazinio paketo paslauga suteukia 1000 kreditų per sekundę, kuriuos išnaudojus, visos viršijančios užklausos atmetamos (mėlynas stulpelis rodo jų skaičių), kiekviena operacija verta 1 arba 10 kreditų, priklausomai nuo to, ar ji apibrėžia siuntimo/gavimo veiksmą, ar eilės valdymo veiksmą [Cor19c].

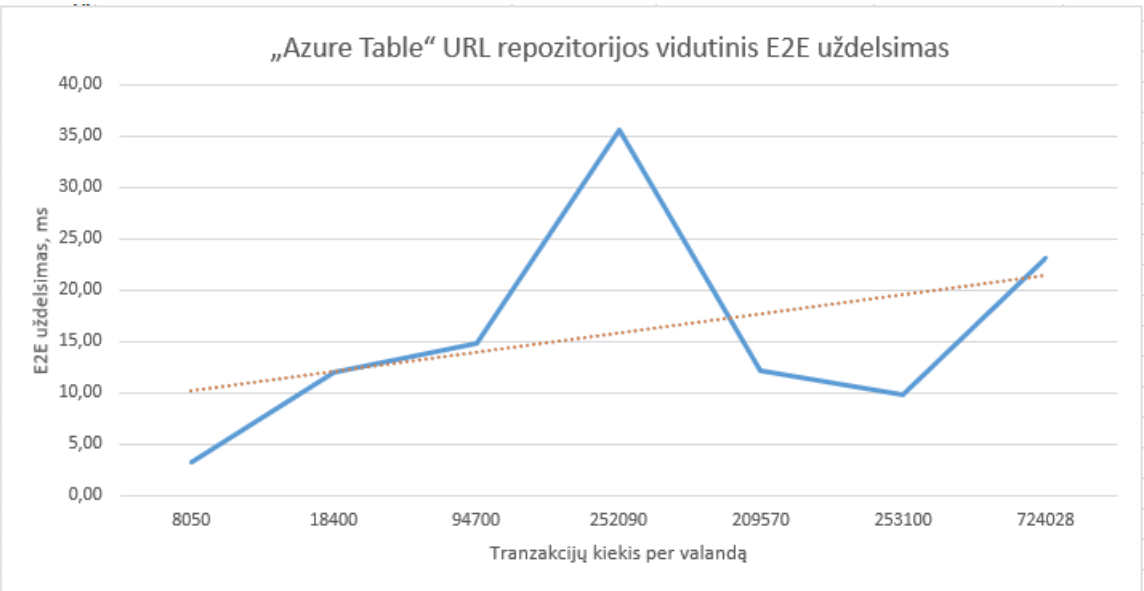


23 pav. Saityno peržiūros greitis didinant agentų skaičių

Kaip pastebima, sistemai dirbant 32 aktyvių agentų režimu, pradeda atsirasti atmetamų užklausų, kas rodo, jog su realizuota architektūra galimai artėjama prie „Basic“ plano siūlomų 1000 operacijų kreditų per sekundę ribos. Norint išplėsti sistemą toliau, tektų naudoti „Premium“ planą, kuris leidžia pasitelkti dedikuotus resursus pagal reikalingą galingumo poreikį.

6.3.2.2. „Azure NoSQL Table Storage“ URL repozitorija

24 diagrama rodo, jog URL repozitorijos komponentas, kuris realizuotas panaudojant „Azure Table Storage“ talpinimo infrastruktūrą, išlaiko gana pastovų uždelsimą net žymiai didinat tranzakcijų skaičių per valandą (kiekviena operacija su lentelės esybe skaitoma kaip tranzakcija). Pavyzdžiui, tranzakcijų skaičiui siekiant 18400, uželsimo rodiklis buvo užfiksuotas 12,04s, o tranzakcijų skaičiui išaugus iki 209570 (daugiau nei dešimt kartų), fiksuotas labai panašus uždelsimas – 12,13 ms. Šie tyrimo rezultatai stipriai koreliuoja su [MZ14] mokslininkų darytu eksperimentu, kurio metu irgi testuotas „Azure Table Storage“ E2E[14] uždelsimas. Pastebėtina, kad pastarieji savo tyrime gavo, jog 238479 tranzakcijos per valandą sugeneravo vidutinišką 8,78ms uždelsimo laiką, o rašto darbo autoriaus tyrimo metu panašiam tranzakcijų skaičiui gauta kiek didesnė nei 10ms reikšmė. Prastesnis rodiklis gali būti sietinas su priimtais skirtingais architektūros realizaciniais sprendimais, tačiau bendras rezultatas rodo, jog šis komponentas neriboja sistemos darbo augant agentų skaičiui.

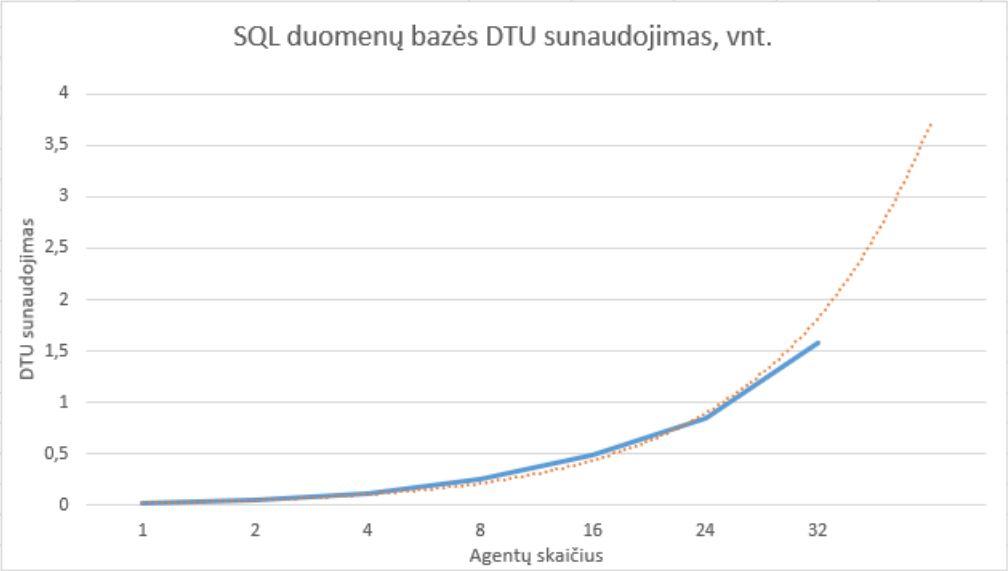


24 pav. „Azure Table Storage“ vidutinis E2E uždelsimas

[14]End-to-End – metrika, rodanti visą laiką, kiek truko užlausos kreipimasis, atsakymo išsiuntimas

6.3.2.3. SQL agentų registro duomenų bazė

Šio tyrimo metu buvo pasirinkta stebėti SQL duomenų bazės apkrovą, nes būtent ja remiasi agentų naikinimo logika. Pagal gautus 25 lentelėje pateiktus tyrimo duomenis matmoma, jog dvigubinant agentų skaičių DTU sąnaudos auga taip pat smarkiai (eksponentiškai). Nepaisant to, 32 agentų veikimo režimas naudojo vidutiniškai tik 1,5 DTU vienetus iš minimalaus „Basic“ plano, suteikiančio 5 DTU. Numatant, jog išliktų panašūs augimo tempai toliau dvigubinant agentų skaičių, galima tikėtis, jog po 7 iteracijų, kai agentų būtų apie 4000, vidutinės DTU sąnaudos siektų apie 200-250 DTU vienetų. Tokio pajėgumo duombazė kainuoja 8 EUR dienai ir leistų pasiekti 75 puslapių per sekundę vidutinį peržiūros greitį (6,5 mln. puslapių per parą).

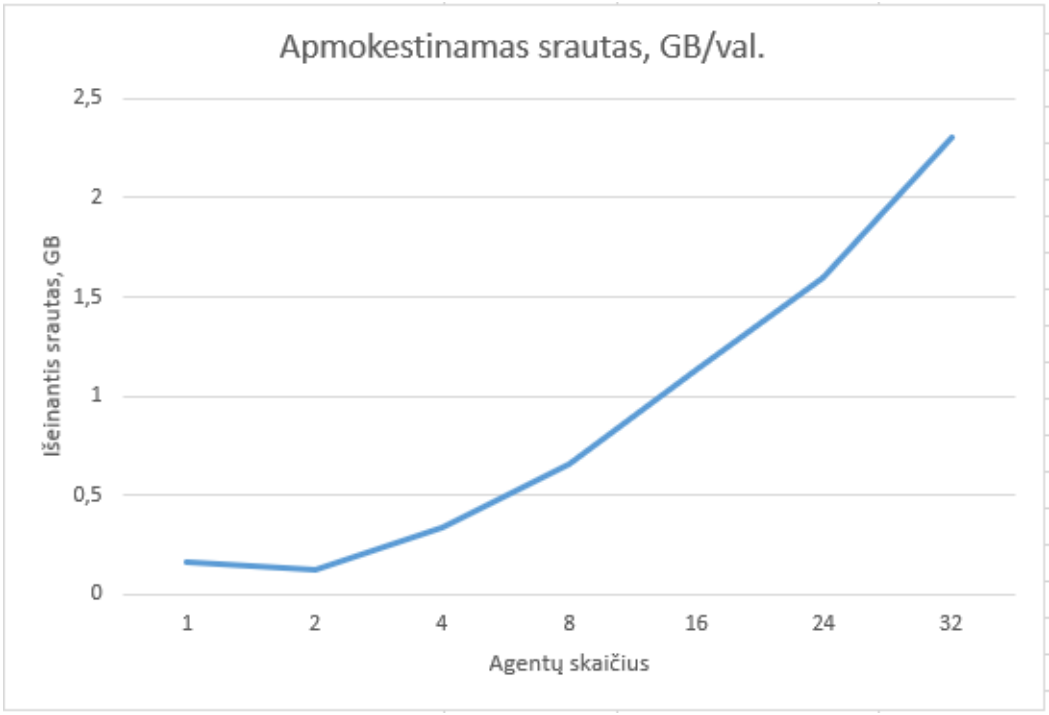


25 pav. Agentų registro SQL duombazės DTU sunaudojimas

Svarbu atsižvelti, jog realizuojant peržiūros robotą agentų registro duomenų bazėje buvo pasitelktas trigeris, kuris duomenų bazės lygiu užtikrina, jog kelios lygiagrečios Centrinio peržiūros variklio gijos (komponentas, kuriantis peržiūros agentus) nesukurs daugiau peržiūros agentų, nei apibrėžia maksimalus nustatymuose esantis limitas. Trigerio kodo fragmentas pavaizduotas nr. 2 priede, kuriame matosi, jog konstruktas iškviečiamas kiekvieną kartą bandant koreguoti agentų registro lentos įrašą ir nustatyti agento statusą į aktyvų. Ši duomenų bazės validacija yra dažna, dėl to gana brangi operacija, lemianti aukštą DTU naudojimą. Siekiant optimalaus duomenų bazės panaudojimo vertėtų analizuoti kitus būdus, kurie leistų užtikrinti, jog lygiagrečios centrinio peržiūros variklio gijos, vienu metu besikreipiančios į duomenų bazę, nesukurtų per daug peržiūros agentų (nei peržiūros administratoriaus nustatytas limitas).

6.3.2.4. Apmokestinamas srautas

Iš 28 diagramos matoma, jog agentų skaičiui didėjant, išeinantis tinklo srautas (angl. – *egress traffic*) auga itin sparčiai. Pavyzdžiui, dirbant 32 aktyvių agentų režimu, per valandą sunaudoto srauto kiekis siekė 2,5 GB. Dvigubinant agentų skaičių išeinantis tinklo srautas didėja apie 1,4 karto. Vertinant, jog efektyvia saityno peržiūra laikoma apie 100 puslapių per sekundę, pagal 22 diagramoje pavaizduotus peržiūros gautus rezultatus galime daryti prielaidą, jog toks peržiūros greitis būtų pasiektas po 8 iteracijų dvigubinant peržiūros agentus, kol jų bus apie 8000. Išlaikant 1,4 karto srauto augimą, galima prognozuoti, jog išeinantis tinklo srautas siektų apie 35 GB per valandą. Pagal dabartinę „Azure“ virtualių mašinų srauto kainodarą 1GB išeinančio tinklo srauto apmokestinama 0,07 EUR, todėl tokio kiekio srautas kainuotų 2,45 EUR/val.



26 pav. Apmokestinamas peržiūros roboto tinklo srautas

Rezultatai

Darbe pasiekti **rezultatai**:

1. Atlikta saityno peržiūros robotų probleminės srities literatūrinė apžvalga:
    - identifikuoti saityno nuorodų ryšiai remiantis grafų teorijos sąvokomis
    - pristatytas bazinis peržiūros robotų veikimo algoritmas
    - atliktas saityno peržiūros robotų ir saityno duomenų surinkimo robotų sistemų palyginimas: išskirti panašumai ir trūkumai
    - aprašytos 3 saityno peržiūros robotų veikimo politikos: pasirinkimo, etiško žvalgymo ir pakartotinio apsilankymo
    - Identifikuotos 4 peržiūros robotų kategorijos: plačiosios peržiūros, teminės peržiūros, inkrementinės peržiūros, išskirstytos peržiūros
2. Apibrėžti saityno peržiūros robotų komponentai ir jų paskirtis: peržiūros pasienis, HTTP parsiuntimo modulis, saityno nuorodų ištraukiklis, adresų skirstiklis, adresų filtras, dublikatų šalintojas, adresų prioritizuotojas. Remiantis identifikuotais komponentais, UML veiklos diagrama apibrėžtas saityno peržiūros vykdymo ciklas
3. Palygintos 3 saityno peržiūros robotų architektūros analizuojant peržiūros išplečiamumo ir darbo koordinavimo aspektus: „PolyBot“, „Mercator“, „Heritrix“. Nustatyta, jog visos jos remiasi centriniu koordinatoriumi, kuris vykdant saityno peržiūros procesą ir išplečiant sistemą tampa ribojančiu faktoriumi.
4. Palygintas tradicinis ir AJAX žiniatinklio atvaizdavimo modeliai, identifikuotas saityno peržiūros robotų ribotumas žvalgyti AJAX tipo svetaines, pristatytas „Googlebot“ peržiūros roboto taikomas modelis atvaizduojant dinaminį turinį generuojančias svetaines
5. Apibrėžta 12 funkcinių ir 3 nefunkciniai saityno peržiūros roboto prototipo reikalavimai
6. Pristatytos prototipo įgyvendinimui pasirinktos „Azure“ debesų kompiuterijos tiekėjo technologijos: „Service Fabric“ orchestravimo platforma, „Service Bus“ eilių infrastruktūra, „Azure Table Storage NoSQL saugykla“, „Azure SQL“ duomenų bazė. Argumentuoti pasirinkimo motyvai, identifikuotos nepasirinktos alternatyvos
7. Įgyvendintas saityno peržiūros roboto prototipas:
    - Apibrėžti 7 panaudojamumo atvejai, kurie atsekamumo matrica susieti su prototipui iškeltais reikalavimais
    - Pristatyta realizuojant prototipą naudojama architektūra
    - Nubrėžtos prototipo dinamiką vaizduojančios UML sekų ir veiklos diagramos, taip pat struktūrinė schema
    - C# kalba naudojant .NET Core karkasą realizuotas apibrėžtos architektūros prototipas, aprašytos panaudotos bibliotekos

8. Atliktas prototipo vertinimas – įvykdytas roboto veikimo stebėsenos eksperimentas, surinktos ir išanalizuotos funkcinės metrikos, taip pat prototipo komponentų apkrovos, nustatyta:
    - 75-100 aplankomų puslapių per sekundę greitis (194 - 263 mln. puslapių per mėnesį) naudojant darbe apibrėžtą architektūrą yra efektyvių debesų kompiuterijos paslaugų kaštų viršutinė riba;
    - siekiant išplėsti roboto veikimo pajėgumą virš apibrėžtos ribos, susiduriama su eksponentiškai augančiomis paslaugų išlaidomis arba pasiekiamomis maksimaliomis leistinomis paslaugų naudojimosi ribomis;
    - taikant populiariųjų klientinių Javascript karkasų ar bibliotekų programinio kodo identifikaciją tarp visų puslapyje naudojamų skriptų, atlikus tokių puslapių pakartotinę peržiūrą naudojant Javascript atvaizdavimo variklį, vidutiniškai atrandama 6% naujų URL adresų;

Išvados

Apibendrinus rezultatus, pateikiamos šios **išvados**:

1. Atsižvelgiant į tyrimo metu nustatytą peržiūros roboto darbo pagreitėjimą, galima daryti išvadą, kad esamas debesų kompiuterijos tiekėjų siūlomų paslaugų lygis užtikrina galimybę efektyviai vykdyti nedidelio-vidutinio lygio automatinį saityno peržiūros procesą. Dėl šios priežasties debesų kompiuterijos paslaugomis paremti saityno peržiūros robotai tinkami atlikti temines peržiūras, akcentuojančias turinio kokybę, o ne aprėptį. Plačiųjų, inkrementinių saityno peržiūrų vykdymas tokiomis aplinkybėmis nėra įmanomas.
2. Javascript variklio naudojimas saityno peržiūros roboto darbo ciklo metu yra brangi operacija, todėl kiekvieno resurso žvalgymas pasitelkiant tokias naršyklių tvarkykles yra neefektyvus.
3. Šiuo metu nėra universalios metodikos atpažinti faktą, jog svetainė naudoja dinaminį HTML turinio atvaizdavimą.
4. Remiantis tyrimo metu nustatytu vidutiniu atrandamų naujų URL adresų skaičiumi taikant populiariųjų karkasų ir bibliotekų atpažinimo metodiką, galima daryti išvadą, kad metodas užtikrina gana efektyvų **tikimybinį** modelį atlikti žiniatinklio peržiūros procesą įtraukiant ir modernias, dinaminį atvaizdavimą naudojančias svetaines, tačiau:
    - metodas išlieka tikimybiniu – egzistuoja reali galimybė neaplankyti resurso, kuris naudoja dinaminį atvaizdavimą, tačiau tam pasitelkia specializuotas, nepopuliarias bibliotekas ar karkasus;
    - taikant modelį privaloma atlikti svetainės naudojamų skriptų turinio analizę – tam sunaudojamas didelis tinklo srautas, todėl efektyvi kešavimo ir kontrolinės sumos skaičiavimo strategija yra būtina;

Galimos ateities darbų kryptys

1. Efektyvios vertingų URL adresų pasirinkimo strategijos realizacija – įgyvendintas prototipas atlieka į plotį paremtą peržiūrą ir nepriskiria jokių skirtingų svorių URL adresams, dėl to švaistoma daug tinklo resursų, nes žiniatinklyje dominuoja prastos kokybės turinio URL nuorodos, kurių žvalgymas neturi prasmės.
2. Įgyvendinto centrinio peržiūros variklio komponento gijų išlygiagretinimo duomenų pastovumas užtikrinamas SQL duomenų bazės trigerio pagalba (žiūrėti nr. 2 priedą). Trigerio iššaukimas lemia dideles duomenų bazės resursų sąnaudas, todėl vertėtų realizuoti efektyvesnę šio komponento lygiagretaus darbo užtikrinimo strategiją.
3. Efektyvesnio dinaminį HTML turinį generuojančių svetainių identifikacijos algoritmo realizacija .

Literatūra

[Ang18] AngleSharp. Getting started. https://anglesharp.github.io/docs/Basics.html, 2018. tikrinta 2020-05-09.

[Cas04] Carlos Castillo. *Effective Web Crawling*. PhD thesis, University of Chile, 2004.

[Cor18] Microsoft Corporation. Service fabric terminology overview. https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-technical-overview, 2018. tikrinta 2020-04-09.

[Cor19a] Microsoft Corporation. Azure table storage overview. https://docs.microsoft.com/en-us/azure/cosmos-db/table-storage-overview, 2019. tikrinta 2020-04-09.

[Cor19b] Microsoft Corporation. Storage queues and service bus queues - compared and contrasted. https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-azure-and-service-bus-queues-compared-contrasted, 2019. tikrinta 2020-04-09.

[Cor19c] Microsoft Corporation. Throttling operations on azure service bus. https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-throttling, 2019. tikrinta 2020-05-20.

[Cor19d] Microsoft Corporation. Understanding the table service data model. https://docs.microsoft.com/en-us/rest/api/storageservices/Understanding-the-Table-Service-Data-Model, 2019. tikrinta 2020-04-09.

[Cor19e] Microsoft Corporation. What are virtual machine scale sets? https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/overview, 2019. tikrinta 2020-04-09.

[Dap18] Dapper. What's dapper? https://dapper-tutorial.net/dapper, 2018. tikrinta 2020-05-09.

[Exp13] The Montessori Experience. https://themontessoriexperience.wordpress.com/2013/12/13/the-google-pagerank-algorithm-explained-the-montessori-way/, 2013. tikrinta 2020-04-09.

[Fat19] Gabija Fatenaite. Data scraping: the powerful force of the business world. https://oxylabs.io/blog/crawling-vs-scraping, 2019. tikrinta 2020-04-19.

[Goo20] Google. Understand the javascript seo basics. https://developers.google.com/search/docs/guides/javascript-seo-basics, 2020. tikrinta 2020-04-09.

[Gor04] Igor Ranitovic Gordon Mohr Michael Stack. An introduction to heritrix. http://crawler.archive.org/Mohr-et-al-2004.pdf, 2004. tikrinta 2020-05-22.

[HN99a] Allan Heydon ir Marc Najork. Mercator: a scalable, extensible web crawler. https://web.archive.org/web/20140309035834/http://www.bagualu.net/linux/crawler.pdf, 1999. tikrinta 2020-04-09.

[HN99b] Allan Heydon ir Marc Najork. Mercator: a scalable, extensible web crawler. http://www.bagualu.net/linux/crawler.pdf, 1999. tikrinta 2020-03-09. Java kalba realizuotas plečiamas, papildomas saityno paieškos robotas – dar vienas architektūrinis pavyzdys, atskleidžiantis esminius tokių sistemų sudedamuosius komponentus.

[Jha12] Arpan Jha. Web crawling: data scraping vs. data crawling. https://www.promptcloud.com/blog/data-scraping-vs-data-crawling/, 2012. tikrinta 2020-04-19.

[KKU18] Shah Khalid, Shah Khusro ir Irfan Ullah. Crawling ajax-based web applications: evolution and state-of-the-art. http://mjs.um.edu.my/index.php/MJCS/article/view/10558, 2018. tikrinta 2020-04-09.

[Kos94] Martijn Koster. A standard for robot exclusion. http://www.robotstxt.org/orig.html#status, 1994. tikrinta 2020-04-09.

[Mal17] Andy Mallon. No, seriously. what is a dtu? https://sqlperformance.com/2017/03/azure/what-the-heck-is-a-dtu, 2017. tikrinta 2020-04-09.

[MZ14] Mukesh Singhal Mehdi Bahrami ir Zixuan Zhuang. A cloud-based web crawler architecture. http://cloudlab.ucmerced.edu/sites/cloudlab.ucmerced.edu/files/documents/bahrami_et_al._a_cloud-based_web_crawler_architecture_cloud_lab_ucm.pdf, 2014. tikrinta 2020-04-09.

[Naj09] Marc Najorc. Web crawler architecture. https://www.microsoft.com/en-us/research/wp-content/uploads/2009/09/EDS-WebCrawlerArchitecture.pdf, 2009. tikrinta 2020-05-09.

[ON10] C. Olston and Marc Najork. Web crawling. http://infolab.stanford.edu/~olston/publications/crawling_survey.pdf, 2010. tikrinta 2020-03-09.

[PAO18] MASSIMO SANTINI+ PAOLO BOLDI ANDREA MARINO. Bubing: massive crawling for the masses. https://dl.acm.org/doi/pdf/10.1145/3160017, 2018. tikrinta 2020-05-20.

[Pin94] Brian Pinkerton. Finding what people want: experiences with the webcrawler. https://web.archive.org/web/20010904075500/http://archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/pinkerton/WebCrawler.html, 1994. tikrinta 2020-04-09.

[Pup19] PuppeteerSharp. Puppeteer sharp. https://www.puppeteersharp.com/api/index.html, 2019. tikrinta 2020-05-09.

[Sit18] Sitebulb. How to crawl javascript websites. https://sitebulb.com/resources/guides/how-to-crawl-javascript-websites/, 2018. tikrinta 2020-04-09.

[Sta20] Internet Live Stats. Total number of websites. https://www.internetlivestats.com/total-number-of-websites/, 2020. tikrinta 2020-04-20.

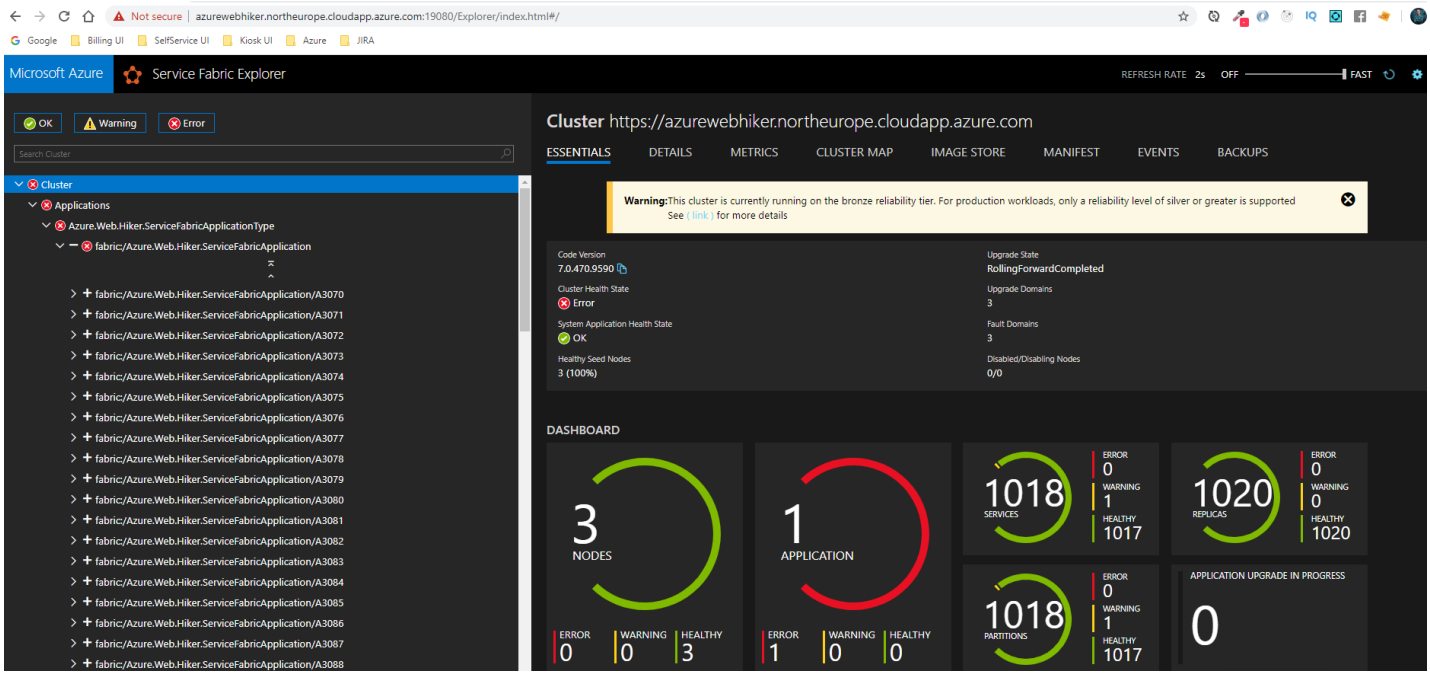
[UUD14] Trupti V. Udapure, Trupti V. Udapure ir Rajesh C. Dharmik. Study of web crawler and its different types. https://pdfs.semanticscholar.org/3260/1ca5ac22427b6ad56938e44f88098035593a.pdf, 2014. tikrinta 2020-04-09.

[Vla01] Torsten Suel Vladislav Shkapenyuk. Design and implementation of a high-performance distributed web crawler. http://cis.poly.edu/tr/tr-cis-2001-03.pdf, 2001. tikrinta 2020-05-09.

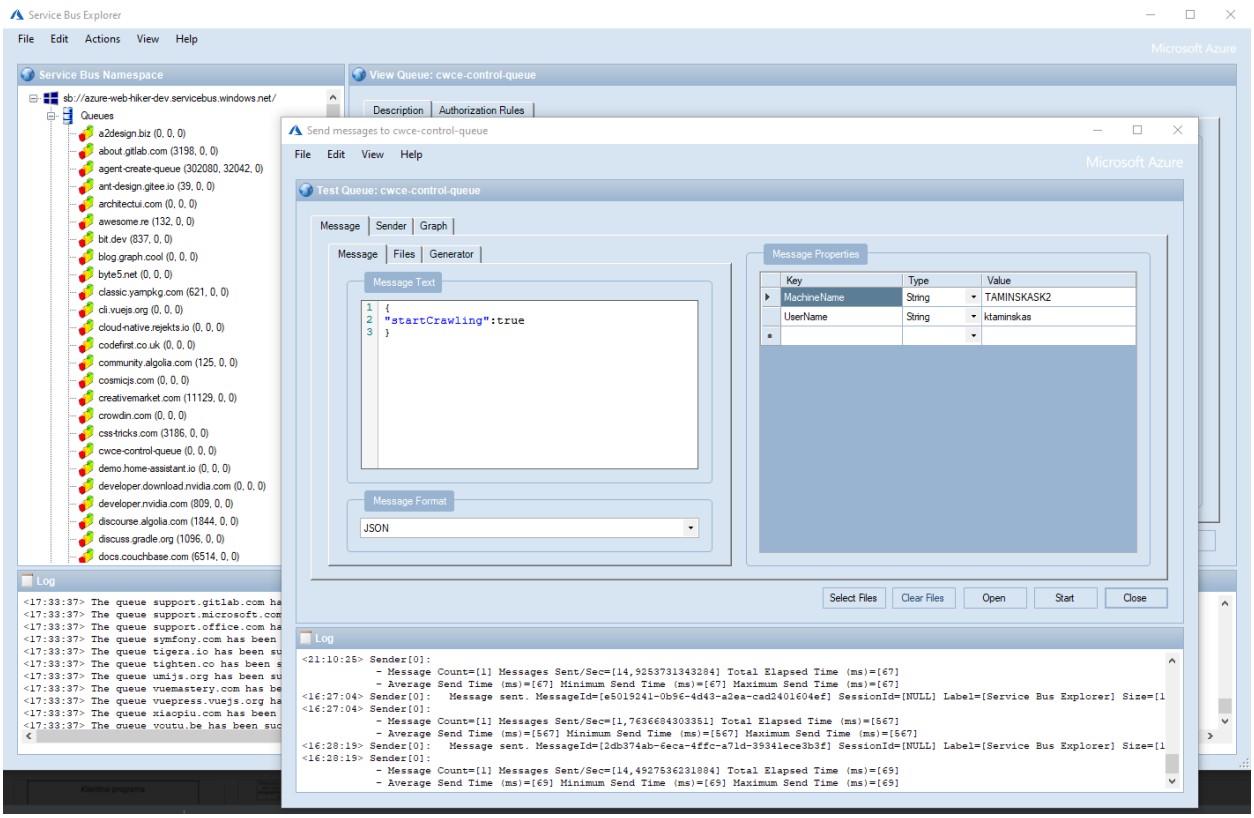
Priedas nr. 1

Peržiūros roboto prototipo programinis kodas

Peržiūros robotas pasiekiamas šiuo adresu: `https://github.com/kasparas12/Azure-Web-Hiker/tree/dev`



27 pav. „Service Fabric“ klasterio valdymo langas saityno peržiūros metu



28 pav. Panaudotas „Service Bus Queue“ klientas valdyti peržiūros robotą

Priedas nr. 2

Agentų registro peržiūros agentų limito trigeris

Pateikiamas T-SQL kodo fragmentas, kurio pagrindu buvo įgyvendintas trigeris, duomenų bazės lygmeniu užtikrinantis, jog aktyvių agentų peržiūros limitas nebūtų viršytas.

```
CREATE TRIGGER [dbo].[InsteadOfUPDATEIsActive] on [dbo].[agent_registrar] FOR UPDATE AS
DECLARE @InsertedIsActive bit;
DECLARE @InsertedId int;
DECLARE @MaxCrawlingLimit int;
DECLARE @CurrentlyActiveAgents int;
     SELECT @InsertedId = INSERTED.id, @InsertedIsActive = INSERTED.is_active
       FROM INSERTED

-- Patikriname, ar bandoma agenta padaryti aktyvu
IF UPDATE(is_active)
BEGIN
    -- Gauname perziuros limita, esanti nustatymu lentoje
  SELECT @MaxCrawlingLimit = CONVERT(INT,setting_value) FROM dbo.settings S
    WHERE S.setting_name = 'crawling_agents_limit'

    -- Patikriname, kiek siuo metu uzregistruota aktyviu perziuros agentu
  SELECT @CurrentlyActiveAgents = COUNT(*) FROM dbo.agent_registrar D WHERE D.
    is_active = 1 AND D.is_deleted = 0


  IF @CurrentlyActiveAgents > @MaxCrawlingLimit
    THROW 51000, 'Agents Limit Exceeded!', 1;
END
```