VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ STUDIJŲ PROGRAMA

Srautinio apdorojimo sistemų balansavimas taikant skatinamąjį mokymąsi

Balancing Stream Processing Systems Using Reinforcement Learning

Magistro baigiamasis darbas

Atliko: Vytautas Žilinas (parašas)
Darbo vadovas: Andrius Adamonis (parašas)
Recenzentas: Prof. dr. Aistis Raudys (parašas)

Vilnius – 2021

Santrauka

Šį darbą sudaro literatūros analizė ir tyrimas. Literatūros dalyje nagrinėjamas srautinio apdorojimo sistemų veikimas, jų greitaveikos matavimas bei derinimas. Taip pat analizuojami kitų autorių tyrimai nagrinėjantys sistemų derinimą naudojant skatinamąjį mokymąsi. Apžvelgiami kituose tyrimuose naudojami skatinamojo mokymosi algoritmai ir pasirenkami algoritmai naudojami atlikti eksperimentus. Tiriamojoje dalyje apibrėžiamas srautinės apdorojimo sistemos, valdomos skatinamuoju mokymusi, modelis ir apsirašoma balansavimo tikslo funkcija. Taip pat apibrėžiamas balansavimo algoritmas, kuris naudojamas atlikti eksperimentus. Atlikus eksperimentus su REINFORCE, DQN ir ACER algoritmais buvo įrodyta, jog „Heron“ srautinio apdorojimo sistemas galima balansuoti naudojant skatinamąjį mokymąsi ir, kad geriausią rezultatą iš visų tirtų algoritmų pasiekia ACER – sumažindamas vėlinimą 50 procentų.

Raktiniai žodžiai: srautinis apdorojimas, skatinamasis mokymasis, balansavimas, „Heron“, REINFORCE, DQN, ACER

Summary

This work consists of literature analysis and research. The literature part examines the workings of stream processing systems, way to measure their speed and the ability to tune the performance. Studies by other authors examining the auto–tuning of stream processing systems using reinforcement learning are also analyzed. The reinforcement learning algorithms used in other studies are reviewed and selected for use in the experimental part of this research. The research part defines the model of stream processing systems controlled using reinforcement learning and describes the goal function. The balancing algorithm used to perform the experiments is also defined. Experiments with REINFORCE, DQN and ACER algorithms prove that ”Heron” stream processing systems can be balanced using reinforcement learning and that ACER achieves the best result of all analyzed algorithms, lowering the latency by 50 percent.

Keywords: stream processing, reinforcement learning, balancing, ”Heron”, REINFORCE, DQN, ACER

TURINYS

Įvadas

Realaus laiko duomenų apdorojimas (angl. real–time data processing) yra jau senai nagrinėjamas kaip vienas iš būdų apdoroti didelių kiekių duomenis (angl. Big data). Viena iš didelių duomenų apdorojimo tipinių architektūrų yra srautinis apdorojimas. Srautinis duomenų apdorojimas (angl. stream processing) – lygiagrečių programų kūrimo modelis, pasireiškiantis sintaksiškai sujungiant nuoseklius skaičiavimo komponentus srautais, kad kiekvienas komponentas galėtų skaičiuoti savarankiškai [Bea15].

Srautinio apdorojimo sistemoms valdyti yra naudojami srautinio apdorojimo varikliai. Yra keli pagrindiniai srautinio apdorojimo varikliai: „Apache Storm“, „Apache Spark“, „Heron“ ir kiti. „Apache Storm“ ir „Heron“ apdoroja duomenis duomenų srautais, o „Apache Spark“ mikro–paketais [KKW+15]. „Heron“ srautinio apdorojimo variklis, buvo išleistas „Twitter“ įmonės 2016 metais, kuris buvo sukurtas siekiant patobulinti įmonėje plačiai naudojama „Apache Storm“ srautinio apdorojimo variklį [Ram16]. Šiame darbe bus naudojamas „Heron“, kadangi jis yra naujesnis ir greitesnis srautinio apdorojimo variklis nei „Apache Storm“ [KBF+15].

Srautinio apdorojimo sistemų balansavimas (angl. auto–tuning) – tai sistemos konfigūracijos ir komponentų valdymas siekiant užtikrinti sistemos greitaveiką neprarandant stabilumo ir optimalaus resursų išnaudojimo. Kadangi srautinio apdorojimo sistemų komponentai yra kuriami kaip lygiagretus skaičiavimo elementai, jie gali būti plečiami horizontaliai ir vertikaliai [Bea15] keičiant sistemų konfigūraciją. Tačiau lygiagrečių elementų kiekio keitimas nėra vienintelis būdas optimizuoti resursų išnaudojimą. Kiekvienas variklis turi savo rinkinį konfigūruojamų elementų. Pavyzdžiui, šiame darbe eksperimentams naudojamas „Heron“ variklis leidžia valdyti savo posistemes ir pateiktas srautinio apdorojimo sistemas naudojant 56 konfigūruojamus parametrus [Her19].

Yra skirtingi būdai kaip gali būti parenkama tinkama konfigūracija. Kadangi srautinio apdorojimo sistemų apkrovos gali būti skirtingų pobūdžių (duomenų kiekis, skaičiavimų sudėtingumas, nereguliari apkrova), o inžinieriai kurdami ir konfigūruodami taikomąsias sistemas išbando tik kelis derinius ir pasirenka labiausiai tinkantį [FA17], lieka daug skirtingų nepatikrintų konfigūracijos variantų. Optimalios konfigūracijos suradimas yra NP sudėtingumo problema [SSP04], kadangi žmonėms yra sunku suvokti didelį kiekį konfigūracijos variacijų. Vienas iš būdų automatiškai valdyti konfigūraciją buvo pasiūlytas 2017 metų straipsnyje „Dhalion: self–regulating stream processing in Heron“, kuriame autoriai aprašo savo sukurtą sprendimą „Dhalion“, kuris konfigūruoja „Heron“ srautinio apdorojimo sistemas pagal esamą apkrovą ir turimus resursus. Tai yra, jei apdorojimo elementų išnaudojimas išauga virš 100%, „Dhalion“ padidina lygiagrečiai dirban-

čių apdorojimo elementų kiekį [FAG+17]. Tačiau toks sprendimas leidžia reguliuoti tik elementų lygiagretumą ir tai daro tik reaktyviai.

Vienas iš naujausių tyrimo krypčių nagrinėjančių srautinio apdorojimo sistemų balansavimą yra mašininis mokymasis. Vienas iš tokių bandymų aprašytas 2018 metų straipsnyje „Auto–tuning Distributed Stream Processing Systems using Reinforcement Learning“[VC18] kuriame atliktas tyrimas – „Apache Spark“ sistemos balansavimui naudojamas skatinamojo mokymo REINFORCE algoritmas, kuris, pagal dabartinę konfigūraciją ir renkamas metrikas, keitė srautinio apdorojimo sistemos konfigūracijos parametrus. Šiame tyrime pasiūlytas sprendimas, naudojantis mašininį mokymąsi, suranda efektyvesne konfigūraciją per trumpesnį laiką nei žmonės, o tokiu būdu išskaičiuotą konfigūraciją naudojanti sistema pasiekia 60–70% mažesnį vėlinimą, nei naudojanti ekspertų rankiniu būdu nustatytą konfigūraciją. [VC18].

Skatinamasis mokymasis yra vienas iš mašininio mokymosi tipų. Šis mokymasis skiriasi nuo kitų mašininio mokymosi algoritmų tipų, nes tokio tipo tinklai mokosi darydami bandymus, klysdami ir pagal tai mokydamiesi. Vienas iš pagrindinių privalumų naudojant skatinamąjį mokymąsi balansavimui – nereikia turėti išankstinių duomenų apmokymui, kas leidžia jį paprasčiau pritaikyti skirtingoms srautinio apdorojimo sistemų apkrovoms. Tačiau tokio tipo mašininis mokymasis turi ir problemų: sudėtinga aprašyti tinkamos konfigūracijos atlygio (angl. reward) funkciją ir balansą tarp tyrinėjimo ir išnaudojimo tam, kad nebūtų patiriami nuostoliai [FA17]. Šiame darbe nagrinėjamas skatinamojo mokymosi gebėjimas balansuoti srautinio apdorojimo sistemas.

Yra sukurta daug skatinamojo mokymosi algoritmų (Monte Carlo, Q–learning, Deep Q Network ir kiti), šiame darbe yra apžvelgti algoritmai, kuriuos naudojo kiti autoriai savo tyrimuose susijusiuose su srautinio apdorojimo sistemų veikimo gerinimu ir pasirinkti keli iš jų siekiant patikrinti ar skatinamasis mokymasis yra tinkamas srautinių sistemų balansavimui ir kuris skatinamojo mokymosi algoritmas pasiekia geriausius rezultatus su tam tikru kiekiu apmokymo žingsnių. Magistro darbo tyrimui naudojami REINFORCE, Deep Q network bei Actor–Critic with Experience Replay skatinamojo mokymosi algoritmai ir tiriama, kaip kiekvienas iš jų pasirodo su vienodu kiekiu apmokymo žingsnių ir tokia pačia srautinio apdorojimo sistema siekiant nustatyti kiekvieno iš šių algoritmų gebėjimą balansuoti srautinio apdorojimo sistemas mažinant vėlinimą.

Darbo tikslas – Ištirti skatinamojo mokymosi tinkamumą srautinio apdorojimo sistemų balansavimui.

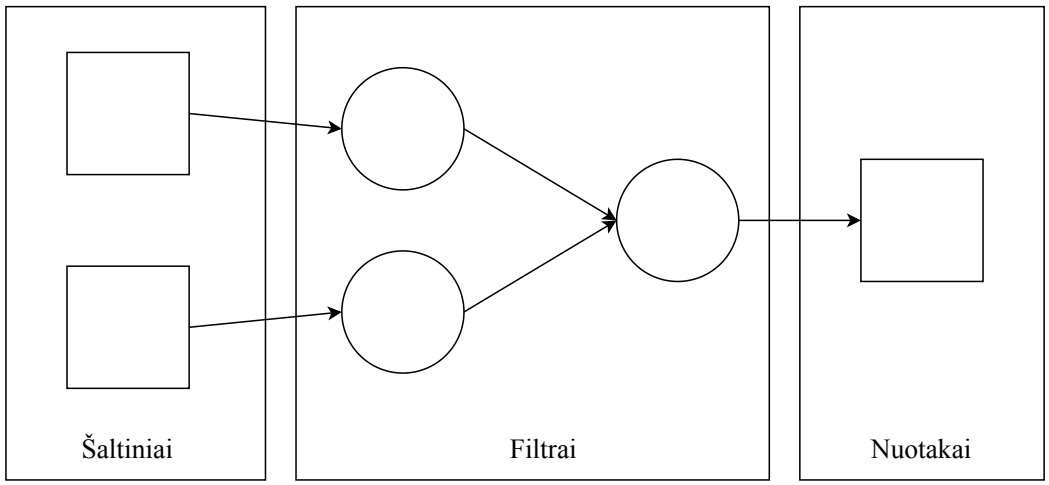
Tikslui pasiekti išsikelti šie uždaviniai:

1. Apibrėžti srautinių duomenų apdorojimo sistemų balansavimo dalykinę sritį – sudarant balansavimo modelį ir identifikuojant valdymo metrikas bei siekiamas metrikų reikšmes.
2. Sudaryti srautinio apdorojimo sistemų balansavimui pritaikytą skatinamojo mokymosi modelį ir parinkti skatinamojo mokymo algoritmus bei srautinio apdorojimo sistemą eksperimentui.
3. Atliekant eksperimentus su pasirinktais skatinamojo mokymosi algoritmais, įvertinti siūlomo modelio validumą srautinio apdorojimo sistemų balansavimui.

1. Srautinio apdorojimo sistemų matavimas ir derinimas

1.1. Srautinio apdorojimo sistemos

Srautinis duomenų apdorojimas (angl. stream processing) – terminas naudojamas apibrėžti sistemas sudarytas iš skaičiavimo elementų (angl. modules) galinčių skaičiuoti lygiagrečiai ir kurios bendrauja kanalais. Tokių sistemų elementai dažniausiai skirstomi į tris klases: šaltinius (angl. sources), kurie paduoda duomenis į sistemą, filtrus (angl. filters), kurie atlieka tam tikrus vienetinius (angl. atomic) skaičiavimus ir nuotakus (angl. sink), kurie perduoda duomenis iš sistemų [Ste97].



1 pav. Srautinio apdorojimo sistemos pavyzdys

Srautinio apdorojimo sistemos literatūroje yra vaizduojamos orientuotais grafikais (1 pav.). Srautinio apdorojimo sistemos skiriasi nuo reliacinio modelio šiais aspektais [BBD+02]:

- Duomenys į sistemą patenka tinklu, o ne iš fizinių talpyklų.
- Duomenų patekimo tvarka negali būti kontroliuojama.
- Duomenų kiekis yra neapibrėžtas.
- Duomenys apdoroti srautinio apdorojimo sistema yra pašalinami arba archyvuojami, t.y. juos pasiekti po apdorojimo yra sunku.

1.1.1. Duomenų vykdymas

Srautinio apdorojimo sistemų veikimui reikalingas srautinio apdorojimo variklis (angl. stream processing engine). Šie varikliai yra skirti srautinio apdorojimo sistemų vykdymui, dislokavimui, plečiamumo (angl. scaling) užtikrinimui ir gedimų tolerancijai (angl. fault–tolerance)

[ZGQ+17]. Populiariųjų srautinio apdorojimo variklių pavyzdžiai: „Apache Storm“, „Apache Heron“, „Apache Spark“, „Apache Samza“ ir t.t [RM19]. Duomenų vykdymas gali būti išskaidytas į tris elementus [ZGQ+17]:

- Planavimas (angl. scheduling) – duomenų apdorojimo užduočių planavimas, kuris daro įtaką bendram srautinio apdorojimo sistemos veikimui [FY11]. Pavyzdžiui, „Apache Samza“ naudoja „Apache YARN“ resursų valdymo sistemos planavimo posistemę, kuri skirsto resursus [NPP+17]
- Plečiamumas (angl. scalability) – daug apdorojimo branduolių turinčios sistemos turi gebėti apdoroti didėjanti kiekį užduočių ir galimybę išskirti resursus sistemai, kad ji galėtų susidoroti su didėjančiu kiekiu duomenų [Bon00]. Srautinio apdorojimo varikliai turi užtikrinti srautinio apdorojimo sistemų plečiamumą [SÇZ05].
- Išskirstytas skaičiavimas (angl. distributed computation) – tarpusavyje nesusiję skaičiavimo elementai turi naudotojui atrodyti kaip viena darni sistemą [TV07]. Srautinio apdorojimo varikliai turi užtikrinti darbų paskirstymą ir skaičiavimo įrenginių koordinaciją, kad kuo daugiau duomenų būtų apdorojami vienu metu [ZGQ+17].

Srautinio apdorojimo sistemos turi viena pagrindinį elementą – srauto procesorių (angl. stream processor), kuris apibrėžia sistemos elementus, aprašo kaip šie sistemos elementai sujungti ir pateikia nustatymus elementams [ZGQ+17]. Pavyzdžiui, „Apache Storm“ ir „Heron“ šis elementas vadinimas „topology“, kuris yra užrašomas Java kalba, naudojant „Apache Storm“ pateiktą biblioteką [IS15].

1.1.2. Duomenų priėmimas

Į srautinio apdorojimo sistemą duomenys patenka per šaltinius, kurie šiuos duomenis perduoda tolimesniems elementams. Dažniausiai duomenis perduodami į sistemą naudojant žinučių eiles (angl. message queues), nes jos turi buferi, kuris leidžia mažinti greičių skirtumus tarp duomenų gavimo ir duomenų apdorojimo ir žinučių eilių brokeriai gali išfiltruoti duomenis ir nukreipti juos į tinkamus šaltinius [KF16]. Tačiau šaltiniai turi turėti galimybę rinkti išsaugotus duomenis ir priimti ateinančius naujus duomenis [SÇZ05], todėl, nors ir šaltiniai dažniausiai skirti priimti srautinius duomenis, jie turi taip pat gebėti naudoti duomenis iš talpyklų [ZGQ+17].

1.2. Srautinio apdorojimo sistemų matavimas

Svarbiausias srautinio apdorojimo sistemų reikalavimas – duomenų apdorojimas ir rezultatų grąžinimas negali turėti atsilikimo – didelių apimčių srautiniai duomenys turi būti apdorojami taip pat greitai kaip jie patenka į sistemą [SÇZ05]. Tam, kad galima būtų užtikrinti šį reikalavimą, turi būti renkamos srautinio apdorojimo sistemų metrikos ir matuojama bendra greitaveiką.

1.2.1. Srautinio apdorojimo sistemų metrikos

Pagrindinės kitų autorių naudojamos metrikos srautinio apdorojimo sistemų matavimui:

- Pralaidumas (angl. Throughput) – per tam tikrą laiko tarpą apdorojamų įvykių kiekis.
- Vėlinimas (angl. Latency) – laiko intervalas nuo apdorojimo arba įvykio pradžios iki apdorojimo pabaigos.

Vėlinimas ir pralaidumas dažniausiai nepriklauso vienas nuo kito – sistemos, apdorojančios srautus mikropaketais, turi didesnį pralaidumą, tačiau atsiranda papildomas vėlinimas, kol laukiama duomenų paketo apdorojimo pradžios [KRK+18].

[SÇZ05] straipsnyje minima, jog srautinio apdorojimo sistemos naudotojas turi išbandyti savo sistemą su tiksliniu darbo krūviu ir išmatuoti jos pralaidumą ir vėlinimą prieš naudodamas ją realiomis sąlygomis. [KRK+18] lygina srautinio apdorojimo variklius ir matavimui naudoja vėlinimą, kurį išskaido į įvykio vėlinimą (angl. event–time latency) – laiko intervalas nuo įvykio laiko iki rezultato gavimo iš srautinio apdorojimo sistemos ir apdorojimo vėlinimą (angl. processing–time latency) – laiko intervalas nuo duomens patekimo į srautinio apdorojimo sistemą iki rezultato grąžinimo. Autoriai atlieka šį skaidymą, nes sistemų vertinime dažnai ignoruojamas įvykio laikas ir rezultatuose gaunamas daug mažesnis vėlinimas nei tikras. Taip pat autoriai išskiria darnų pralaidumą (angl. sustainable throughput) – didžiausią apkrovą įvykių, kurią sistema gali apdoroti be pastoviai augančio įvykių vėlinimo, todėl savo eksperimentuose autoriai užtikrina, kad duomenų generavimo greitis atitiktų sistemos darnų pralaidumą. Kad sužinoti darnų pralaidumą, sistemos autoriai pradžioje leidžia labai didelį srautą duomenų ir mažina jį kol sistemos apdorojimas susivienodina su generavimo greičiais. Skirtingų srautinio apdorojimo variklių vėlinimo rezultatus autoriai pateikia maksimalaus pralaidumo apdorojimo ir 90% pralaidumo apdorojimo vidurkiais, minimumais, maksimumais ir kvantiliais (90, 95, 99).

[HSS+14] autoriai nagrinėja srautiniam apdorojimui galimas optimizacijas ir matavimui naudoja normalizuotą pralaidumą (naudojamas vienetas kaip vidurkis), kadangi tai leidžia lengviau palyginti santykinę greitaveiką. Taip pat, [HSS+14] pastebi, nors ir yra daug metrikų, kuriomis

galima matuoti optimizacijos efektus: pralaidumas, vėlinimas, paslaugos kokybė (angl. quality of service), energijos ir tinklo panaudojimas, tačiau dažniausiai pagerinus pralaidumą pagerėja ir visos kitos metrikos. [QWH+16] srautinių apdorojimo sistemų matavimui naudoja pralaidumą (skaičiuojama baitais per sekundę) ir vėlinimą, kaip vidurkį nuo duomens patekimo į sistemą iki apdorojimo pabaigos. Taip pat, kadangi autoriai lygina srautinio apdorojimo variklius, jie įveda metriką gedimų toleravimo (angl. fault tolerance) matavimui – išjungiamas tam tikras kiekis elementų ir matuojamas pralaidumas ir vėlinimas. [ZYL+20] palyginimui naudoja sistemos įvykdymo vėlinimą (angl. system completion latency), kuris rodo vidutinį laiko tarpą, per kurį duomuo nukeliauja nuo šaltinio iki sistemos galutinio taško. Šio straipsnio autoriai skaičiavo vidutinį laiką 5 sekundžių intervalais. Taip pat [ZYL+20] matavimui naudoja kiekvienos instancijos, kas yra srautinio apdorojimo variklio sudėtas srautinio apdorojimo skaičiavimo komponentų rinkinys, kuris yra izoliuotas ir naudoja bendrą valdymo ir matavimo posistemę, CPU apkrovą, kiekvieno darbinio mazgo (angl. worker node) CPU apkrovą ir apkrovą tarp instancijų/mazgų, kadangi [ZYL+20] užduotis – patobulinti esamą planavimo posistemę. [FAG+17] matavimui naudoja pralaidumą per minutę. [VC18] tyria labai panašią problemą – srautinių apdorojimo sistemų balansavimą taikant skatinamąjį mokymą ir matavimui naudoja vėlinimo 99 percentilę. [CDE+16] srautinio apdorojimo variklių vertinimo tyrimui naudoja vėlinimą.

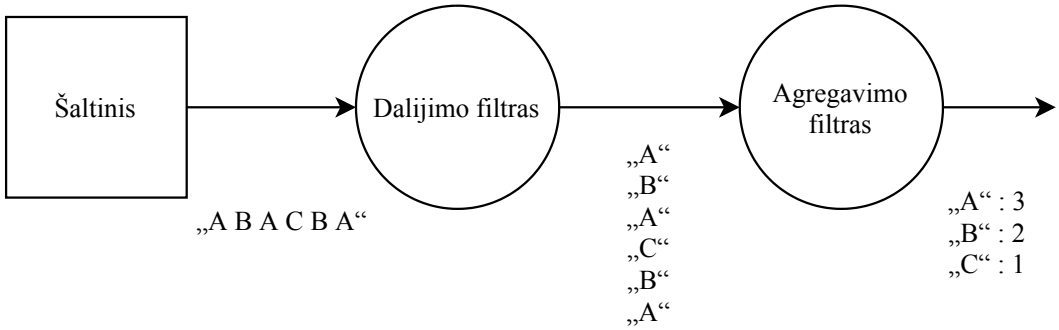
1 lentelė. Metrikos naudojamos tiriant srautinio apdorojimo sistemų greitaveiką

Šaltinis	Vėlinimas	Pralaidumas
[SÇZ05]	Taip	Taip
[KRK+18]	Taip	Taip
[HSS+14]	Ne	Taip
[QWH+16]	Taip	Taip
[ZYL+20]	Taip	Ne
[FAG+17]	Ne	Taip
[VC18]	Taip	Ne
[CDE+16]	Taip	Ne

Pagal literatūros analizę (1 len.) matoma, kad dauguma autorių renkasi vertinti tik pagal vieną metriką ir dažniau matavimui naudojamas vėlinimas. Taip pat [VC18], naudojantis skatinamąjį mokymą, matavimui naudoja vėlinimą ir [CDE+16] straipsnis, kuris siūlo srautinio apdorojimo sistemų vertinimo sprendimą, naudoja vėlinimą.

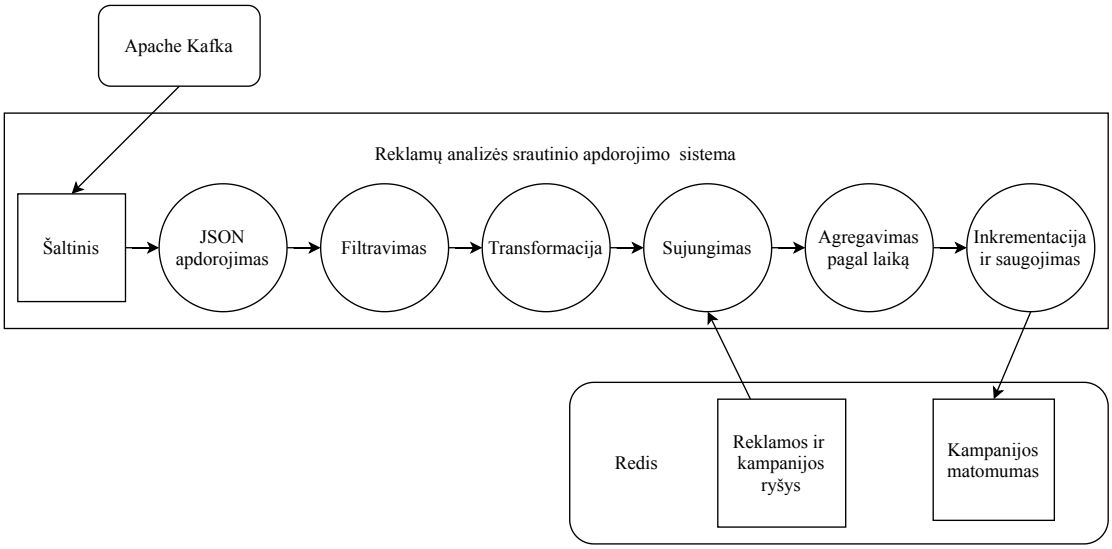
1.2.2. Srautinio apdorojimo sistemos pobūdis

Srautinio apdorojimo sistemos gali turėti skirtingą elementų išsidėstymą ir nuo to priklausys jų greitaveiką. [KRK+18] matavimui naudoja du filtrus – agregavimo, kuris skaičiuoja visus pirkimus ir jungimo (angl. join), kuris skaičiuoja duomenis pagal tam tikrą bendrą rodiklį iš abiejų duomenų srautų. [QWH+16] srautinio apdorojimo variklių palyginimui naudoja septynis skirtingus uždavinius. Vienas iš jų yra WordCount uždavinys, kuris yra plačiai priimtas kaip didelių duomenų apdorojimo sistemos matavimo standartas [HHD+10]. Šis uždavinys susidaro iš dviejų filtrų: pirmas išskaido teksto eilutę į žodžius, o antras agreguoja kiekvieno žodžio bendrą skaitiklį ir atnaujina bendrą žodžių panaudojimo dažnio rezultatą, kuriame raktas – žodis, o reikšmė skaičius, kuris rodo kiek kartojosi šis žodis (2 pav.).



2 pav. WordCount sistemos pavyzdys

[ZYL+20] autoriai naudoja SentenceWordCount sistemą, kuri yra identiška 2 paveikslėlyje pavaizduotai sistemai. Bei autoriai sukūrė FileWordCount sistemą, kuri atlieka tą patį, kaip ir SentenceWordCount, tačiau šaltinis negeneruoja žodžių, o skaito iš tekstinio dokumento, ir taip pat naudoja egzistuojančia Yahoo srautinio apdorojimo vertinimą (angl. benchmarking) [CDE+16]. [FAG+17] autoriai naudoja WordCount eksperimentui. [VC18] eksperimentams naudoja Yahoo srautinio apdorojimo variklių vertinimą [CDE+16] ir taip pat atlieka bandymus su realiais daiktų interneto (angl. internet of things) įmonės duomenimis. [CDE+16] pateikia srautinio apdorojimo sistemą skirtingų srautinio apdorojimo variklių vertinimui. Ši sistema analizuoja reklamas pagal kampaniją ir matomumą ir rezultatus deda į Redis duomenų bazę. Ši sistema sukurta taip, kad aprėptų visas srautinio apdorojimo sistemos savybes (3 pav.).



3 pav. Reklamų analizės sistema [CDE+16]

Straipsniai ([FAG+17; HHD+10; QWH+16]) naudoja WordCount (2 pav.), o [CDE+16; VC18] naudoja Reklamų analizės srautinę apdorojimo sistemą (3 pav.). Reklamų analizės srautinio apdorojimo sistemos privalumas palyginus su WordCount srautinio apdorojimo sistema – galimybė valdyti į sistemą patenkančių duomenų srautą, kas leidžia atlikus tyrimą teigti, jog duomenų apdorojimo pastovumas nedarė įtakos rezultatams.

1.2.3. Srautinio apdorojimo sistemų matavimui naudojami duomenys

Vertinant sistemų greitaveiką reikia atsižvelgti ir į testavimui naudojamus duomenis. [KRK+18] naudoja žaidimų kūrimo įmonės Rovio duomenis ir naudoja du duomenų srautus – pirkimo srautas, kuriame siunčiami kortežai (angl. tuples) – sudaryti iš nupirktos valiutos kiekio, laiko ir naudotojo, kuris ją nupirko ir reklamų srautas, kuris siunčia valiutos reklamas tam tikru laiku. Šiame sprendime duomenys generuojami naudojant raktinių laukų reikšmes pagal normalinį paskirstymą. [QWH+16] naudoja tekstinius duomenis iš AOL paieškos variklio ir apdoroja juos pagal pasirinktus uždavinius. [ZYL+20] matavimui naudoja šaltinių generuojamą tekstą, kadangi lyginamas tas pats srautinio apdorojimo variklis tik su patobulinta planavimo posisteme ir naudoja iš anksto sugeneruotą tekstą patalpintą į tekstinį dokumentą. [CDE+16] aprašo sistemą, kuri daro skirtingų srautinio apdorojimo variklių vertinimą. Šiam vertinimui naudojami duomenys, simuliuojantys reklamas ir reklamų kampanijas, ir todėl autoriai naudoja savo sukurtą duomenų generatorių.

1.3. Srautinio apdorojimo sistemų derinimas

Sistemų greitaveika yra tiesiogiai susijusi su konfigūravimo parametrais, kurie valdo tokius aspektus kaip: atminties valdymas, gijų skaičius, planavimas, resursų valdymas [LCH+19]. Taip pat, neteisingi nustatymai turi nuostolingus efektus sistemos greitaveikai ir stabilumui [HLL+11].

[HCL20] išskiria 3 pagrindinius automatinio derinimo iššūkius:

1. Didelė ir sudėtinga parametrų erdvė – „Apache Spark“ ir „Apache Storm“ turi virš 150 konfigūruojamų parametrų [BC17; PGT16]. Taip pat, nustatymų reikšmės, kurios tinka vienam uždaviniui, gali turėti neigiamos įtakos kitam [HLL+11; JC16].
2. Sistemų mastas ir sudėtingumas – Sistemų administratoriai turi gebėti konfigūruoti didelius kiekius skaičiavimo mazgų, kurie gali turėti skirtingus CPU, atminties, tinklo tipus [HCL20].
3. Pradinių duomenų statistikos trūkumas – įvedimo duomenys srautinėse apdorojimo sistemose yra realus srautai, kurie stipriai varijuoja savo apimtimi [DP18].

[TLW17] nagrinėjantis tinkamos konfigūracijos radimą naudojant genetinius algoritmus „Apache Storm“ srautinio apdorojimo sistemoms nustatė, jog lygiagretumo laipsnis labiausiai daro įtaką srautinio apdorojimo sistemų greitaveikai. „Apache Heron“ srautinio apdorojimo variklis, kuris yra „Apache Storm“ su patobulinimais [KBF+15], pateikia naują būdą kontroliuoti srautą – priešslėgis (angl. backpressure), kuris leidžia filtrui sulėtinti prieš jį einantį elementą, kas leidžia sumažinti vėlinimą ir taip pat gali būti naudojamas kaip greitaveikos praradimo indikatorius [BCB+18]. Taip pat [BCB+18] nagrinėja „Apache Heron“ automatinį konfigūravimą naudojant iš anksto aprašytas taisykles.

2. Skatinamasis mokymasis

Skatinamasis mokymasis yra mašininio mokymosi rūšis, išsiskirianti iš kitų tuo, kad turi spręsti visą problemą atlikdamas veiksmus su nežinoma aplinka [SB18].

2.1. Mašininis mokymasis srautinio apdorojimo sistemų derinimui

[HCL20] aprašo skirtingus sprendimus automatiniam konfigūravimui ir išskiria šiuos mašininio mokymosi privalumus:

- Nebūtina suprasti sistemos, užduočių ir duomenų, kadangi naudojamas juodos dėžės (angl. black–box) principas.
- Mašininio mokymosi modelis pats save tobulina, ir yra vis tikslesnis kuo daugiau gauna duomenų.

[HCL20] straipsnio autoriai išskiria mašininio mokymosi iššūkius:

- Parametrų parinkimas – kadangi konfigūruojamų parametrų kiekis yra didelis [BC17; PGT16], ne visi iš jų vienodai daro įtaką greitaveikai, todėl pirma verta išsirinkti aktualiausius parametrus resursų valdymo, užduočių planavimo ir duomenų valdymo užduotims. Tam dažnai naudojama eksperto pagalba [WXH16], gidai arba eksperimentavimas. Tačiau galima naudoti mašininio mokymosi algoritmą koreliacijos nustatymui tarp parametrų ir greitaveikos [YLL+12; VC18]
- Mašininio mokymosi modelio pasirinkimas – kadangi yra nemažai skirtingų mašininio mokymosi metodų kurie tinka derinimo uždaviniui.

Taip pat autoriai pateikia paketinio ir srautinio apdorojimo derinimo naudojant mašininį mokymąsi straipsnius (2 lentelėje pateikiami tik išrinkti srautinio apdorojimo pavyzdžiai).

2 lentelė. Srautinių sistemų derinimo naudojant mašininį mokymąsi pavyzdžiai [HCL20]

Šaltinis	Įvesties savybės	Mašininio mokymosi metodai
Zacheilas et al. [ZKZ+15]	Rinkinys kelių konfigūracijos parametrų	Gaussian Processes
Li et al. [LTX16]	Atminties dydžiai ir branduolių ir gijų kiekis skirtingose stadijose	Support Vector Regression

Trotter et al. [TLW17]	Darbinių procesų kiekis, vykdytojų kiekis	Genetic Algorithm, Bayesian Optimization
Trotter et al. [TWH19]	Vykdytojai, šaltinių ir filtrų lygiagretumas, acker lygiagretumas	Genetic Algorithm, Support Vector Machines
OrientStream [WMG+17]	Įvairios duomenų, plano, filtrų ir klasterio lygio savybės	Ensemble/ Incremental ML
Vaquero et al. [VC18]	Parametrai ir metrikos parinkti faktorinės analizės (angl. factor analysis) pagalba	Reinforcement Learning

2.2. Skatinamasis mašininis mokymasis

Standartiniame skatinamojo mašininio mokymosi algoritme agentas (angl. agent) yra prijungtas prie aplinkos (angl. environment) per stebėjimus (angl. perception) ir veiksmus (angl. action). Kiekviename žingsnyje agentas atlieka veiksmą ir to veiksmo vertė yra perduodama agentui per atlygį. Agentas renkasi veiksmus, kurie per laiko tarpą didina veiksmo atlygį. Tai pasiekiama per tam tikrą laiko tarpą atliekant bandymus ir klystant, su papildoma algoritmų pagalba siekiant padidinti spėjimo efektyvumą ir sprendimų stabilumą [KLM96].

2.2.1. Skatinamasis mokymasis srautinio apdorojimo sistemų veikimo gerinimui

[VC18] nagrinėja srautinių sistemų derinimą naudojant skatinamąjį mokymąsi. Autoriai pradžioje naudoja Lasso kelių analizės algoritmą, kurio pagalba atlieka parametrų atranką, kad išrinktų svarbiausius parametrus pasirinktos metrikos valdymui. Konfigūracijos valdymui pasirinktas modifikuotas REINFORCE algoritmas naudojantis tinklą su 20 paslėptų sluoksnių ir RMSprop optimizacijos funkciją su 0.001 mokymosi žingsniu. Atliktas eksperimentas su „Apache Spark“ ir pasiektas 60–70% sumažintas vėlinimas palyginus su nekeista konfigūracija.

[NLY+19] nagrinėja resursų valdymo problemą – kaip efektyviai paskirstyti sistemos turimus resursus, kad įrangos specifikacija kuo mažiau ribotų greitaveika, srautinio apdorojimo sistemose ir siūlo sprendimą naudojanti skatinamąjį mokymą, kuris daro optimizacijas pagal srautinės apdorojimo sistemos grafus. Eksperimentas atliekamas naudojant REINFORCE [Wil92] algoritmą su Adam optimizacijos funkcija [KB14] ir atliekamas eksperimentas naudojant tyrimui parašytą srautinio apdorojimo variklį ir sistemas.

[LXT+18] nagrinėja planavimo problemos (apkrovos paskirstymas darbiniams elementams) sprendimą naudojant skatinamąjį mokymąsi. Autoriai siūlo naudoti Actor–Critic [LHP+15] metodą naudojant Deep Q Learning [MKS+15] tinklą kaip aktorių ir bet kokį gilųjį neuroninį tinklą kaip kritiką. Autorių rezultatai rodo 45% greitaveikos padidėjimą lyginant su integruota „Apache Storm“ planavimo posisteme.

[RCP19] nagrinėja srautinių sistemų dislokavimą naudojant skatinamąjį mokymą. Sprendžiama dislokavimo valdymo problema su skirtingais skaičiavimo mazgų tipais. Naudojamas Q Learning algoritmas su įvairiomis modifikacijomis (kombinuojama su tiksliais modeliais). Autoriai matuoja dislokavimo tikslumą ir konvergavimo greitį.

3 lentelė. Skatinamojo mokymosi naudojimas

Šaltinis	Skatinamojo mokymosi algoritmas
[VC18]	Adaptuotas REINFORCE [Wil92]
[NLY+19]	REINFORCE [Wil92] su Adam optimizacijos funkcija [KB14]
[LXT+18]	Deep Q Learning [MKS+15] ir Actor–Critic [LHP+15]
[RCP19]	Q–learning [MKS+15] su papildomomis funkcijomis

2.2.2. Balansavimas naudojant REINFORCE algoritmą

Straipsnis [VC18] nagrinėja automatinį balansavimą srautinio apdorojimo sistemų Apache Spark platformoje. Straipsnyje nagrinėjamas sprendimas susidaro iš trijų sistemų:

1. Sistema, kurioje iš anksto sugeneruoti konfigūracijų deriniai leidžiami srautinio apdorojimo sistemose ir surenkamos metrikos bei konfigūracijos įverčiai. Gauti duomenis analizuojami naudojant Factor Analysis + k–means ir gaunamas sąrašas pagrindinių metrikų bei konfigūracijos elementai darantys daugiausiai įtakos greitaveikai. Ši sistema naudojama vieną kartą prieš leidžiant sekančią sistemą.
2. Sistema, kurioje surinktos metrikos ir konfigūracijos elementai yra leidžiami iš naujo ir naudojant Lasso path analizę konfigūracijos elementų sąrašas surūšiuojamas pagal įtaką greitaveikai. Tai daroma siekiant statistiškai užtikrinti, jog pasirinkti konfigūracijos elementai yra įtakingiausi. Ši sistema naudojama vieną kartą prieš leidžiant skatinamąjį mokymąsi.
3. Pagrindinė mašininio mokymosi sistema, naudojanti REINFORCE algoritmą, kuri naudodamas surikiuotų konfigūracijos elementų sąrašą ir pagrindinių metrikų sąrašą periodiškai

atnaujina konfigūraciją. Ši sistema paleidžiama tuo pačiu metu kaip ir srautinio apdorojimo sistema ir veikia visą laiką iki tyrimo pabaigos.

Eksperimentinis sprendimas buvo sukonfigūruotas kas 5 minutes atnaujinti vieną konfigūracijos elementą. Autoriams pavyko pasiekti 70% sumažinta vėlinimą po 50 minučių, o veiksmų vertingumas pilnai konvergavo po 11 valandų.

2.2.3. Deep Q Network algoritmas

Deep Q Network yra Q Learning įgyvendinimas naudojant giliuosius neuroninius tinklus. Q Learning – skatinamojo mokymosi algoritmas, kuris bet kokiam baigtiniam Markovo pasirinkimo procesui randa optimalius sprendimus maksimizuojančius galutinio rezultato gavimą per bet kokį kiekį žingsnių pradedant nuo esamos būsenos [Mel01]. Q Learning naudoja Q funkciją, kurios įeigą yra būsenos ir veiksmo kombinacija, o rezultatas yra atlygio aproksimacija. Pradžioje visos Q reikšmės yra 0 ir algoritmas atlikdamas veiksmus pildo lentelę atnaujintomis reikšmėmis. Tačiau, kai veiksmų ir būsenų pasidaro per daug Q Learning algoritmo nebeužtenka ir tenka naudoti giliuosius neuroninius tinklus.

Deep Q Network skiriasi nuo Q Learning tuo, kad vietoj būsenos ir veiksmų į jį paduodama būsena ir jis gražina Q reikšmę visų įmanomų veiksmų. Taip pat Deep Q Network naudoja patirties pakartojimą (angl. experience replay) – vietoj to, kad kurti sprendimą pagal paskutinį veiksmą yra paduodamas rinkinys atsitiktinių veiksmų pagal kurį algoritmas gali efektyviau mokytis.

Deep Q Network algoritmo veikimas[Cho19]:

1. Perduoti aplinkos būseną į Deep Q Network, kuris gražins visus įmanomus veiksmus būsenai.
2. Pasirinkti veiksmą naudojant ϵ–greedy strategiją, kuriuo metu pasirenkamas atsitiktinis veiksmas arba pasirenkamas veiksmas turintis didžiausia Q reikšmę.
3. Įvykdomas veiksmas ir pereinama į naują būseną. Šis perėjimas išsaugomas kaip patirties pakartojimo kortežas – susidarantis iš būsenos, veiksmo, atlygio ir naujos būsenos.
4. Pasirenkamas atsitiktinis rinkinys perėjimų iš patirties pakartojamo kortežų rinkinio ir apskaičiuojamas nuostolis (angl. loss).

$$Loss = (r + \gamma max_{a'} Q(s',a';\theta') – Q(s,a;\theta))^2$$

5. Atliktas gradiento nusileidimą su tikrais tinklo parametrais siekiant sumažinti nuostolį.
6. Po kiekvieno žingsnio, perkelti tikrojo tinklo svorius į pasirinkto tinklo svorius.

7. Visi žingsniai kartojami iki nustatytos pabaigos.

2.2.4. Actor–Critic with Experience Replay

Actor–Critic algoritmai išsiskiria tuo, kad naudojami du tinklai, aktorius – pasirenka veiksmą, o kritikas – praneša aktoriui, kiek geras jo pasirinktas veiksmas ir kaip jis turi būti gerinamas. Šio tipo skatinamojo mokymosi algoritmai reikalauja mažiau skaičiavimo pajėgumų bei gali išmokti optimalią strategiją [SB18]. Šio algoritmo veikimas yra paremtas:

- Q–reikšmės įvertinimo atsiminimu (angl. Retrace Q–value estimation) – Q–reikšmės įvertinimo algoritmas užtikrinanti strategijos ir veiksmų konvergavimą ir gerą duomenų išnaudojimą [MSH+16].
- Svarbos svorių sutrumpinimu (angl. Importance weights truncation) – atliekamas svorių koregavimas siekiant sumažinti dispersiją ir pasiekti nešališką vertinimą.
- Efektyvus patikimo regiono strategijos optimizavimu (angl. Efficient trust region policy optimization) – naudojamas patobulintas [SLA+15] pateiktas algoritmas siekiant sumažinti per didelį parametrų atnaujinimą apribojant strategijos atnaujinimo dydį.

2.2.5. Apibendrinimas

Siekiant atsakyti į klausimą, ar skatinamasis mokymasis tinka srautinio apdorojimo sistemų balansavimui, naudojami keli algoritmai siekiant užtikrinti, jog rezultatai atitiktų skatinamojo mokymosi gebėjimą balansuoti, o ne pavienio algoritmo ir skirtingi kiekiai apmokymo žingsnių, siekiant patikrinti algoritmų apmokymo greičius.

Pasirinkti algoritmai:

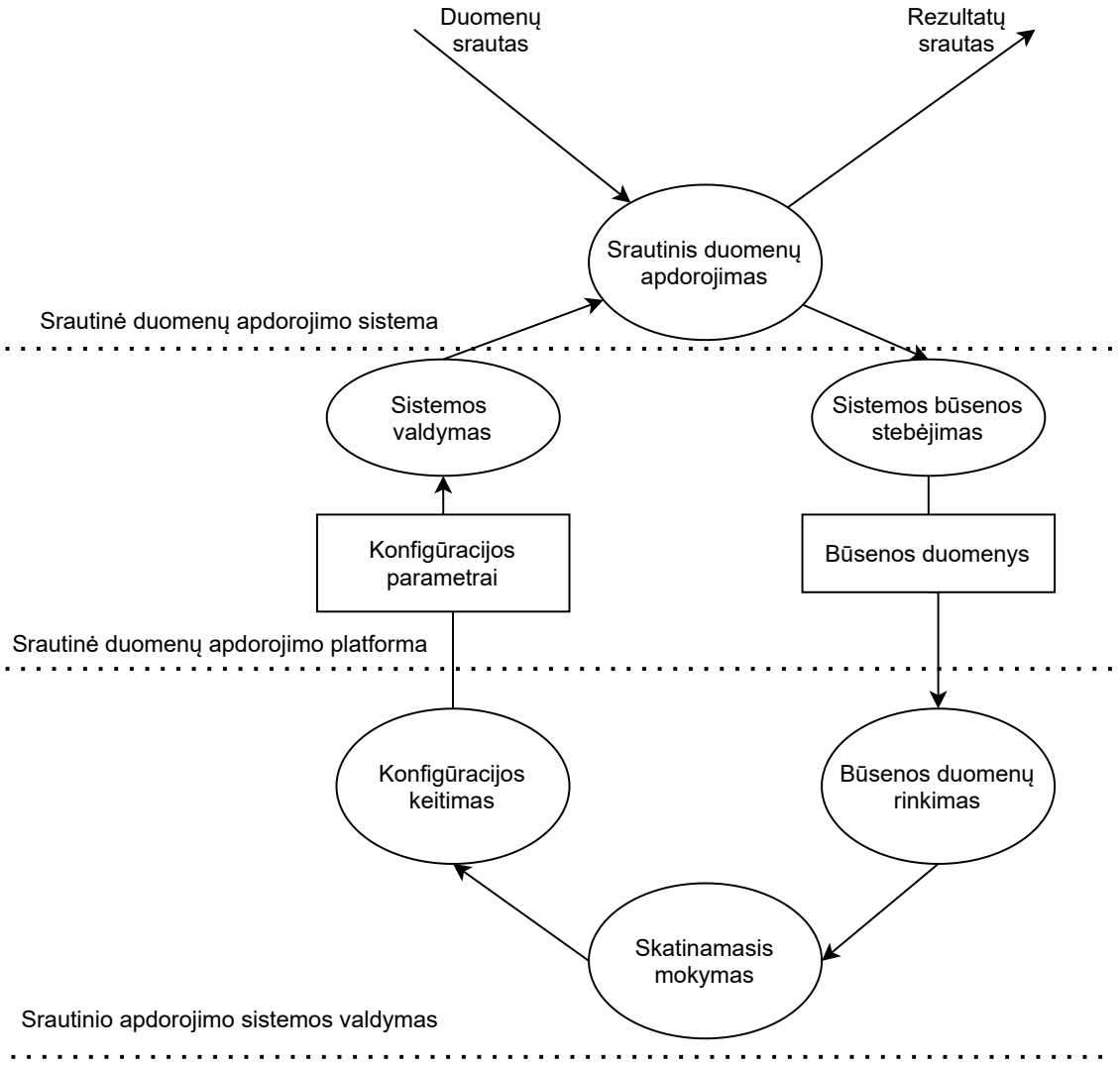
- REINFORCE [Wil92] sukonfigūruotas pagal [VC18] pateiktą konfigūracija.
- Deep Q Network remiantis [MKS+15].
- Actor–Critic with Experience Replay remiantis [WBH+16], kadangi jam reikia mažiau duomenų, kad jis būtų efektyvesnis nei kiti algoritmai.

O tyrime pasirinktiems algoritmams apmokymui naudotos keturios apmokymo trukmės: 50, 100, 250 ir 1000 žingsnių.

3. Tiriamos sistemos modelis

Tyrime nagrinėjamas srautinės sistemos, balansuojamos skatinamuoju mokymų, modelis (4 pav.) susidaro iš trijų pagrindinių elementų: srautinio duomenų apdorojimo sistemos, srautinio duomenų apdorojimo platformos ir valdymo sistemos.

3.1. Modelis



4 pav. Srautinės apdorojimo sistemos modelis

Srautinės duomenų apdorojimo sistemą sudaro šie elementai:

- Duomenų srautas – duomenys patenkantys į srautinio apdorojimo sistemą nepertraukiamu srautu, iš anksto nežinomu greičiu ir nekontroliuojamu kiekiu.
- Srautinis duomenų apdorojimas – srautinio duomenų apdorojimo sistema, atliekanti skaičiavimus su duomenimis ateinančiais iš duomenų srauto. Tyrime naudojamos Heron srauti-

nio apdorojimo sistemos pasižymi individualiais skaičiavimo komponentais, kurie skaičiuoja kiekvieną patenkantį duomenį ir yra sukurti užtikrinant lygiagretumą komponento lygyje.

- Rezultatų srautas – duomenys apdoroti srautinio duomenų apdorojimo sistemos ir perduoti iš paskutinio skaičiavimo komponento į kitas sistemas.

Srautinio apdorojimo sistemų platforma turi šiuos elementus:

- Srautinį duomenų apdorojimą.
- Sistemos būsenos stebėjimą – srautinio apdorojimo platformos sistema renkanti srautinio apdorojimo sistemų veikimo metrikas ir šias metrikas atskleidžianti į išorę. Tyrime naudojama Heron platforma metrikas pateikia kiekvienam skaičiavimo komponentui individualiai per HTTP protokolą arba į naudotojo pateiktą metrikų surinkimo sistemą.
- Būsenos duomenys – tai metrikos vaizduojančios kiekvienos srautinio apdorojimo sistemos skaičiavimo komponentų veikimo rodiklius, tokius kaip vėlinimas, pralaidumas, apkrovos ir t.t.
- Konfigūracijos parametrai – tai konfigūracijos rinkinys kurį apibrėžia srautinio apdorojimo platforma. Šie konfigūracijos parametrai nurodo srautinės apdorojimo sistemos veikimą ir taip pat gali apibrėžti parametrus skirtus individualiems skaičiavimo komponentams. Tyrime naudojama Heron platforma apibrėžia ir valdo visus srautinės apdorojimo sistemos konfigūracijos parametrus bei skaičiavimo komponentų lygiagretumo konfigūraciją.
- Sistemos valdymas – konfigūracijos parametrų pateikimas į srautinio apdorojimo platformą. Šie konfigūracijos parametrai nurodo srautinės apdorojimo sistemos ir jos skaičiavimo komponentų veikimą. Tyrime naudojama Heron platformą leidžia pateikti konfigūracijos failą per komandinę eilutę. Pateikus konfigūraciją platforma sustabdo srautinę apdorojimo sistemą, atnaujina jos konfigūraciją ir paleidžia sistemą iš naujo. Kai sistema sustabdoma, taip pat nustojama ir skaityti duomenis iš duomenų srauto, o paleidus sistemą duomenys skaitomi toliau.

Srautinio apdorojimo sistemos valdymo elementai susidaro iš:

- Būsenos duomenų rinkimo – tai sistema renkanti duomenis apie srautinės apdorojimo sistemos būseną iš srautinės apdorojimo platformos. Ši sistema atsakinga už aktualių metrikų surinkimą, kurios naudojamos suformuoti vaizdą apie srautinio duomenų apdorojimo sistemos būseną.
- Mašininio mokymosi – sistema, kurį gauna srautinės apdorojimo sistemos būsenos duomenis ir pagal tai apskaičiuoja naujas konfigūracijos parametrų reikšmes. Tyrime naudojamas ska-

tinamasis mokymasis, kuris bando gerinti sistemos būseną apskaičiuojant geriausią veiksmą konfigūracijos keitimui ir mokosi iš prieš tai atliktų veiksmų.

- Konfigūracijos keitimo – sistema priimanti atnaujintus konfigūracijos parametrus ir pateikianti juos į srautinio apdorojimo platformą.

Visumoje 4 paveikslėlis apibrėžia duomenų judėjimą srautinio duomenų apdorojimo sistemos valdymo modelyje. Srautinio apdorojimo sistema yra pagrindinis elementas atsakingas už patį duomenų apdorojimą ir veikia nepriklausomai nuo kitų elementų modelyje. Srautinio apdorojimo platforma talpina ir palaiko srautinio apdorojimo sistemas, taip pat suteikia prieiga gauti informaciją apie srautinio apdorojimo sistemų būseną bei suteikia galimybę valdyti srautinio apdorojimo sistemas. Srautinio apdorojimo sistemų valdymo elementai atsakingi už srautinės apdorojimo sistemos konfigūracijos keitimą pagal surinktus būsenos duomenis.

3.2. Keičiami konfigūracijos parametrai

Norint koreguoti konfigūracijos parametrus reikia siųsti atnaujinimo komandą su atnaujinto konfigūracijos failo vieta į Heron komandinės eilutės įrankį (toliau Heron CLI). Pateikus konfigūracijos parametrus Heron platformą perkrauna srautinio apdorojimo sistemą su naujais parametrais. Vienas iš pagrindinių srautinės apdorojimo sistemos konfigūracijos parametrų – skaičiavimo komponentų lygiagretumas, kuris nurodo kiek paleidžiama tam tikro komponento instancijų. Taip pat tai yra vienintelis keičiamas konfigūracijos elementas, kuris priklauso nuo valdomos srautinio apdorojimo sistemos apibrėžimo.

Visi konfigūravimo parametrai [Her19], kurie keičiami balansavimo metu, pateikti „Priedas nr. 1“ skyriuje 6 lentelėje.

3.3. Naudojamos metrikos

Metrikos iš Heron srautinio apdorojimo sistemų gali būti pasiektos keliais skirtingais būdais: darant užklausą į Heron API, skaitant iš tekstinio failo, kurį pildo Heron platformą arba naudojant savo sukurtą metrikų skaitymo priedą, kuris pateikiamas į Heron platformą prieš ją paleidžiant. Visos metrikos yra saugomos srautinio apdorojimo sistemos kiekvienam skaičiavimo komponentui.

4 lentelėje aprašytos metrikos yra standartinės visiems Heron srautinio apdorojimo sistemos komponentams ir grąžinamos iš Heron per Heron API [Her20]. Šios metrikos perduodamos į mašininio mokymosi algoritmą.

4 lentelė. Naudojamos metrikos

Metrika	Paaiškinimas
__emit–count (toliau išsiųstas kiekis)	Išsiųstų kortežų kiekis
__execute–count (toliau apdorotas kiekis)	Apdorotų kortežų kiekis Bolt komponentuose
__execute–latency (toliau vidutinis apdorojimo vėlinimas)	Vidutinė trukmė Bolt komponentui apdoroti kortežą

3.4. Tikslo funkcija

Srautinio apdorojimo sistemos valdymo tikslas – keičiant konfigūraciją, pasiekti didžiausią greitaveiką. Šiame tyrime greitaveika matuojama vėlinimu, todėl pasirinkta tikslo funkcija – tiesiogiai apskaičiuojamą pagal vėlinimą (__execute–latency) ir duodamas teigiamas atlygis, jeigu pasiektas mažiausias vėlinimas, priešingu atveju neigiamas atlygis.

4. Balansavimo algoritmas

4.1. Srautinės architektūros sistemos konfigūracijos valdymo algoritmas

Algoritmo tikslas – pagal esamą būseną ir esamą konfigūraciją apskaičiuoti naują konfigūraciją, kuri pagerintų srautinės apdorojimo sistemos greitaveiką. Algoritmo pagrindui yra pasirinkti skatinamojo mašininio mokymosi algoritmai REINFORCE, Deep Q Network ir Actor–Critic with Experience Replay.

Kad algoritmas galėtų apskaičiuoti naują konfigūraciją, jam reikalingi šie duomenys:

- Dabartinė srautinio apdorojimo sistemos būsena – užklausos metų esantis komponentų lygiagretumas bei pasirinkta konfigūracija.
- Srautinio apdorojimo sistemos metrikos (4 lentelė) – atlygio apskaičiavimui.
- Srautinio apdorojimo sistemos pradinė konfigūracija (7 lentelė).
- Veiksmų aibė kuriuos gali pasirinkti algoritmas.

Balansavimo algoritmas veikia ciklais visą srautinės apdorojimo sistemos veikimo laiką ir tarpas tarp ciklų turi būti pakankamai ilgas srautinio apdorojimo sistemai atsinaujinti su naujais konfigūracijos parametrais bei, kad surinktų duomenų kiekis būtų pakankamai svarus pateikti mašininio mokymosi algoritmui. Balansavimo algoritmas, atlikęs pasirenką veiksmą ir atlikęs jį laukia sekančio žingsnio.

Balansavimui naudojami skatinamojo mašininio mokymosi algoritmai REINFORCE, Deep Q network ir Actor–Critic with Experience Replay aprašyti skyriuje „Skatinamasis mašininis mokymasis“ turi šiuos bendrus veikimo bruožus:

- Jie yra bemodeliniai (angl. model–free) tipo skatinamojo mokymosi algoritmai. Tai aktualu mūsų atveju, kadangi balansavimo algoritmas turi galėti prisitaikyti prie kintančių duomenų kiekio ir greičio bei kintančios aplinkos.
- Jie yra be strategijos (angl. off–policy) tipo, tai reiškia, kad jie mokosi atliekant skirtingus veiksmus, nebūtinai tuos, kurie buvo pasirinkti pagal dabartinę tinklo strategiją.

4.2. Srautinio duomenų apdorojimo aplinkos apibrėžimas balansavimui

Balansavimui atlikti reikalingas tikslus apibrėžimas srautinio apdorojimo aplinkos, šis apibrėžimas susidaro iš: galimų srautinio apdorojimo sistemos būsenų aibės, galimų veiksmų aibės ir funkcijų:

- Inicializavimo funkcija – nustato pradinius sistemos parametrus bei apibrėžia galimų veiksmų aibę (5 len.) ir aplinkos stebėjimo aibę.
- Žingsnio funkcija – kaip įeitį priima veiksmą, jį atlieka ir grąžina žingsnio rezultatus – būseną, atlygi ir papildomus parametrus.
- Perrinkimo funkcija – grąžina sistemą į pradinę būseną ir grąžina ją skatinamojo mokymosi algoritmui.

Kadangi pasirinkti algoritmai naudoja diskrečią veiksmų aibę, pasirinkta veiksmų aibė sudaryta iš srautinio apdorojimo komponentų lygiagretumo ir konfigūracijos rinkinio skaičiaus. Konfigūracijos parametrų rinkiniai sudaryti pagal „Priedas nr. 2“ skyriaus 7 lentelės pateiktas reikšmių aibes, kurios pasirinktos pagal standartinė konfigūracija, kaip centrinę reikšmę, o ribos paskirstytos aplinkui šią reikšmę.

Sudaryti 10 konfigūracijos rinkiniai išdalinus konfigūracijos elementus į lygias dalis pagal reikšmių aibę, siekiant, kad srautinės apdorojimo sistemos metrikos, atlikus vieną konfigūracijos pakeitimą, pakeistų vėlinimą, per vienodą arba didesnį kiekį nei yra paklaida, kas leistų geriau pastebėti bei priduotų vertę atlikti konfigūracijos keitimą.

Sujungus konfigūracijos atnaujinimą ir komponentų lygiagretumo valdymo sudarytas galimų veiksmų rinkinys, kurio dydis yra lygus $3 + (2 * \text{komponentų kiekis})$ ir kiekvieno veiksmo reikšmė pateikta 5 lentelėje.

5 lentelė. Galimų aplinkos veiksmų aibė

Veiksmo numeris	Veiksmas	Riba
0	Nedaryti nieko	–
1	Pasirinkti konfigūracijos rinkinį, kurio numeris vienetų žemesnis	0
2	Pasirinkti konfigūracijos rinkinį, kurio numeris vienetų aukštesnis	10
1 + 2n	Atnaujinti n–tojo komponento konfigūracija, sumažinant jo lygiagretumą	1
2 + 2n	Atnaujinti n–tojo komponento konfigūracija, padidinant jo lygiagretumą	9

Srautinio apdorojimo sistemos būsenos aibė – apibrėžiama kaip masyvas, kuris susidaro iš

srautinio apdorojimo komponentų lygiagretumo ir pasirinkto konfigūracijos rinkinio numerio. Pradinė būsena nustatyta, kad kiekvieno komponento lygiagretumo skaičius būtų 2, o pradinis konfigūracijos rinkinio numeris 5.

Skatinamojo mokymosi algoritmo veikimui reikalingas atlygis aplinkos apskaičiuojamas pagal formulę:

$$Atlygis(n) = \begin{cases} \text{min(Vėlinimas)} - \text{Vėlinimas}_n & \text{, kai Vėlinimas}_n \geq \text{min(Vėlinimas)} \\ \text{C} - \text{Vėlinimas}_n & \text{, kai Vėlinimas}_n < \text{min(Vėlinimas)} \end{cases}$$

C – konstanta apskaičiuojama per pirmą paleidimą su pradine konfigūracija, suapvalinus gautą vėlinimą.

Srautinio apdorojimo sistemos balansavimo žingsnio funkcija susidaro iš:

- Įeities – atliekamo veiksmo skaičiaus, kuris nurodo aplinkai kokį veiksmą ji turi įvykdyti.
- Išeities – atlikto veiksmo rezultatas, kuris susidaro iš:
  - Būsenos – srautinio apdorojimo sistemos dabartinis komponentų lygiagretumas ir sistemos konfigūracija.
  - Atlygio – apskaičiuoto naudojant metrikas gautas iš srautinio apdorojimo sistemos.
  - Epizodo baigtumo rodiklio – nurodančio, ar aplinka baigė epizodą ir turi būti paleista iš naujo.
  - Papildoma informacijos – naudojamos derinimo pagalbai.

4.3. Balansavimui naudojamo algoritmo tikslas

Kadangi nagrinėjamos srautinės apdorojimo sistemos turi paklaidą atsirandančia dėl duomenų skirtumų ir dėl net ir nedidelių pokyčių aplinkoje, siekiant nagrinėti rezultatus, darbe apsibrėžta 5 sekundžių paklaida, kurį nepriklauso nuo balansavimo, o nuo pačios srautinio apdorojimo sistemos paklaidos.

Siekiant nustatyti algoritmo tinkamumą balansavimui, nustatoma tinkamumo sąlyga – algoritmo balansavimo metu nuo pradinio testavimo žingsnio iki testavimo pabaigos yra pasiektas vėlinimo sumažinimas ir vėlinimo pokytis yra didesnis nei apibrėžta 5 sekundžių paklaida.

Tuo tarpu norint teigti, jog algoritmo atliekami sprendimai yra stabilus apibrėžiama sąlyga, kad algoritmo balansavimo metu srautinio apdorojimo vėlinimo svyravimai turi būti ne didesni nei apibrėžta 5 sekundžių paklaida.

4.4. Balansavimo algoritmo apmokymas

Balansavimo algoritmo apmokymas yra atliekamas epizodais, kur vienas epizodas susidaro iš 25 žingsnių. Kiekvieno epizodo pradžioje srautinio apdorojimo sistemą perkraunama ir nustatoma į pradinę būseną. Tinklas su pasirinktų algoritmų yra apmokoma nustatyta kiekį kartų ir po to naudojant apmokytą tinklą atliekami veiksmai ant srautinio apdorojimo sistemos su pradinę konfigūracija. Pagrindinis srautinio apdorojimo sistemos balansavimo iššūkis – ilgas laiko tarpas tarp balansavimo žingsnių, nes srautinio apdorojimo sistema turi būti perkraunama pakeitus konfigūraciją, kad pokyčiai įsigaliotų. Balansavimo algoritmas tarp pakeitimų laukia 3 minutes, kas yra pakankamas laikas sistemai pasileisti ir pasiekti stabilią būseną – surenkamų metrikų nekeičiant konfigūracijos paklaida nėra didesnė nei 5 sekundės. Dėl šios trukmės vienas skatinamojo mokymosi epizodas trunka apie 1,5 valandos (epizodo trukmės paklaida susidaro, dėl skirtingos trukmės atnaujinimo veiksmo).

Šiame darbe pasirinkti nagrinėti kelis apmokymo trukmės variantus su skirtingais algoritmais siekiant palyginti algoritmų mokymosi efektyvumą balansavimo problemai spręsti. Pasirinktas 1000 žingsnių kiekis pagal [VC18] nagrinėjama „Balansavimas naudojant REINFORCE algoritmą“ skyriuje. Taip pat pasirinkti 50, 100 ir 250 žingsnių apmokymo kiekiai, norint nustatyti kokio apmokymo žingsnių reikia, kad sistemos balansavimas būtų atliktas sėkmingai testavimo metu.

Algoritmo apmokymo testavimui, pasirinkta atlikti 25 žingsnius balansavimo, kuriuo metu algoritmas kas žingsnį pakeičia konfigūraciją ir po 25 žingsnių pasiekia konfigūracijos rinkinį, kuris, sėkmingo balansavimo atveju pasiekia mažesnį vėlinimą nei su pradinę konfigūracija.

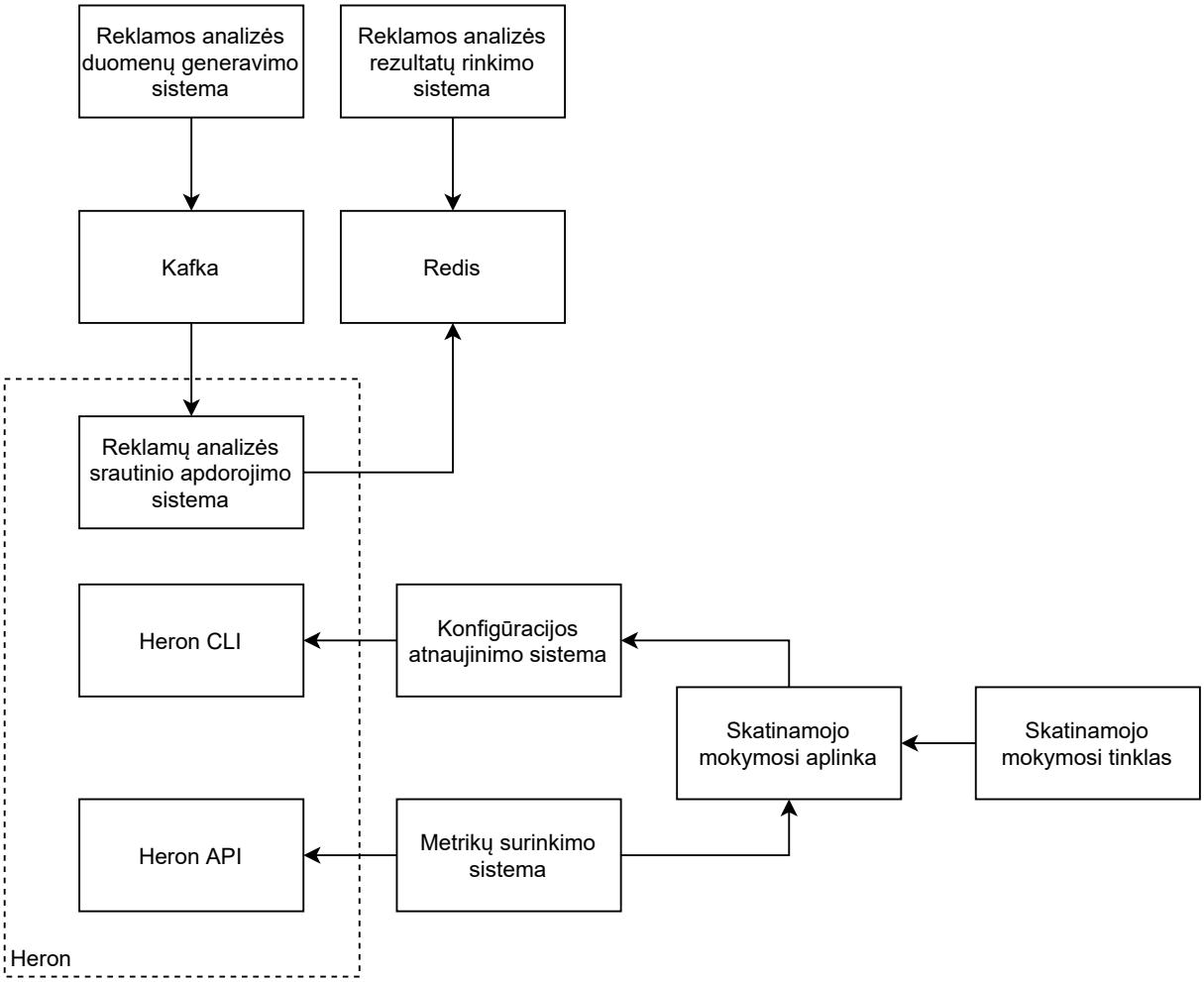
5. Eksperimentinis tyrimas

5.1. Tyrimo tikslas

Šio tyrimo tikslas – įvertinti siūlomo balansavimo modelio ir pasirinkto optimizavimo algoritmo validumą. Tam atlikti bandymai su eksperimentine sistema naudojančia REINFORCE, Deep Q Network ir Actor–Critic with Experience Replay algoritmus.

5.2. Eksperimentinė sistema

Eksperimentui atlikti naudojama lokali Heron aplinka ir papildomos posistemės skirtos duomenų generavimui, rezultatų rinkimui ir t.t. Visa eksperimento sistemos architektūros diagrama pateikta 5 paveikslėlyje.



5 pav. Eksperimento sistemos architektūra

Sistemos, kurios naudojamos tyrime:

1. Skatinamojo mašininio mokymosi sistema. Kadangi atliekami bandymai su skirtingais skatinamojo mokymosi algoritmais, kiekvienam jų sukurta atskira sistema:
    - Skatinamojo mokymosi sistema, naudojanti REINFORCE algoritmą, įgyvendinta naudojant tensorflow biblioteką ir vadovaujantis tensorflow pateiktą dokumentaciją šio algoritmo įgyvendinimui. Šio algoritmo hiperparametrai sudėti pagal aprašytus „Balansavimas naudojant REINFORCE algoritmą“ poskyryje.
    - Skatinamojo mokymosi sistema, naudojanti Deep Q Network algoritmą, įgyvendinta naudojant tensorflow ir stable–baselines bibliotekas ir naudojant rekomenduojamus hiperparametrus.
    - Skatinamojo mokymosi sistema naudojanti Actor–Critic with Experience Replay algoritmą, įgyvendinta naudojant tensorflow ir stable–baselines bibliotekas ir naudojant rekomenduojamus hiperparametrus.
2. Skatinamojo mokymosi aplinka – aprašanti srautinio apdorojimo sistemos būseną, veiksmų aibę ir žingsnio funkciją, kuro atnaujina srautinio apdorojimo sistemą, surenka būsenos duomenis ir apskaičiuoja atlygį.
3. Aplinkai reikalingos papildomos posistemės atlikti atnaujinimus ir surinkti duomenis:
    - Metrikų surinkimo sistema parašyta Python kalba, naudojanti HTTP protokolą, būsenos duomenų surinkimui iš Heron API.
    - Konfigūracijos atnaujinimo sistema parašyta Python, skirta konfigūracijos atnaujinimui ir pateikimui į Heron sistemą per Heron CLI.
4. Reklamų srautinio apdorojimo sistema parašyta JAVA, gaunanti duomenis iš Kafka žinučių eilės ir sauganti rezultatus Redis duomenų bazėje.
5. Rezultatų apdorojimo sistema parašyta su Python, kuri surenka duomenis iš rezultatų failų, apdoroja juos ir grąžina skirtingų sprendimų diagramas.
6. Reklamos analizės rezultatų rinkimo sistema parašyta su Clojure ir pateikta kartu su Reklamos analizės greitaveikos testu.
7. Reklamos analizės duomenų generavimo sistema parašyta su Clojure ir pateikta kartu su Reklamos analizės greitaveikos testu.

Kompiuterinė įranga, kuria atliekamas eksperimentas parametrai:

- Procesorius: Intel Core i7–5930k (6 branduoliai/12 gijų)
- Operatyvi atmintis: 64 GB (2666 MHz)

- Vaizdo plokštė: Nvidia GTX 1080Ti
- Operacinė sistema: Windows 10 Education

Kadangi Heron nepritaikyta Windows operacinei sistemai, visi tyrimai ir visos reikiamos posistemės leidžiamos naudojant Windows Subsystem Linux (toliau WSL) su Ubuntu 18.04. Pagrindinės tyrime naudojamos programinės įrangos versijos:

- Apache Heron: 0.20.3
- Kafka: 2.13
- Redis: 4.0.9
- Java: 8
- Python: 3.6.9
  - Tensorflow 1.14 su stable–baselines 2.10 – DQN ir ACER algoritmams
  - Tensorflow 2.4 ir tf–agents 0.7.1 – REINFORCE algoritmui

5.3. Eksperimentų apimtis ir naudojama duomenų imtis

Tyrimas atliktas balansuojant srautinio apdorojimo sistemą naudojant tinklą apmokytą pasirinktais algoritmais:

- REINFORCE algoritmą skatinamojo mokymosi posistemėje.
- Deep Q Network algoritmą skatinamojo mokymosi posistemėje.
- Actor–Critic with Experience Replay algoritmą skatinamojo mokymosi posistemėje.

Kiekvienas skatinamojo mokymosi sprendimas yra testuojamas su Reklamų analizės srautinio apdorojimo sistema aprašyta „Srautinio apdorojimo sistemų matavimas ir derinimas“ skyriuje.

Reklamų analizės sistema rezultatus talpina į tekstinį failą, kur kiekvienas įrašas yra vėlinimas milisekundžių tikslumu nuo paskutinio išsiųsto įrašo į žinučių eilę tam tikram kampanijos langui iki kol jis yra įrašomas į Redis duomenų bazę. Pasinaudojant šiais duomenimis, po kiekvienos žingsnio apskaičiuojamas paskutinių 30 sekundžių vėlinimo vidurkis.

Eksperimentai atlikti su visais algoritmais apmokius juos 50 žingsnių, 100 žingsnių, 250 žingsnių ir 1000 žingsnių tuo siekiant palyginti rezultatus tarp algoritmų ir mokymosi trukmės. Algoritmo efektyvumui patikrinti apmokytas modelis atlieka konfigūracijos keitimą 25 žingsnius ir matuojamas vėlinamas po kiekvieno pakeitimo praėjus 3 minutėms – laikas, kurio reikia srautinio apdorojimo sistemai pasileisti ir pasiekti stabilią apkrovos stadiją. Kadangi srautinio apdorojimo sistemos nėra pastovios, rezultatų įvertinimui apsibrėžta paklaida – 5 sekundės vėlinimo.

5.4. Eksperimentai

Eksperimentų tikslas – išmatuoti srautinės apdorojimo sistemos vėlinimą, balansuojant ją su REINFORCE, Deep Q Network ir Actor–Critic with Experience Replay algoritmais ir palyginti pasiektą vėlinimą su skirtingu apmokymo žingsnių kiekiu.

Eksperimento metu tinklas yra apmokomas su kiekvienu nagrinėjamu algoritmu 4 atskirus kartus su 50, 100, 250 ir 1000 žingsnių. Po apmokymo su skirtingu žingsnių kiekiu sistema paleidžiama iš naujo ir įvykdomi papildomi 25 testavimo žingsniai, kurių metu tinklas nėra mokomas, o renkami rezultatai. Surinkti rezultatai pateikti grafikuose, kurie rodo srautinės apdorojimo sistemos vėlinimą, po kiekvieno žingsnio keičiant konfigūraciją naudojant atitinkamą algoritmą.

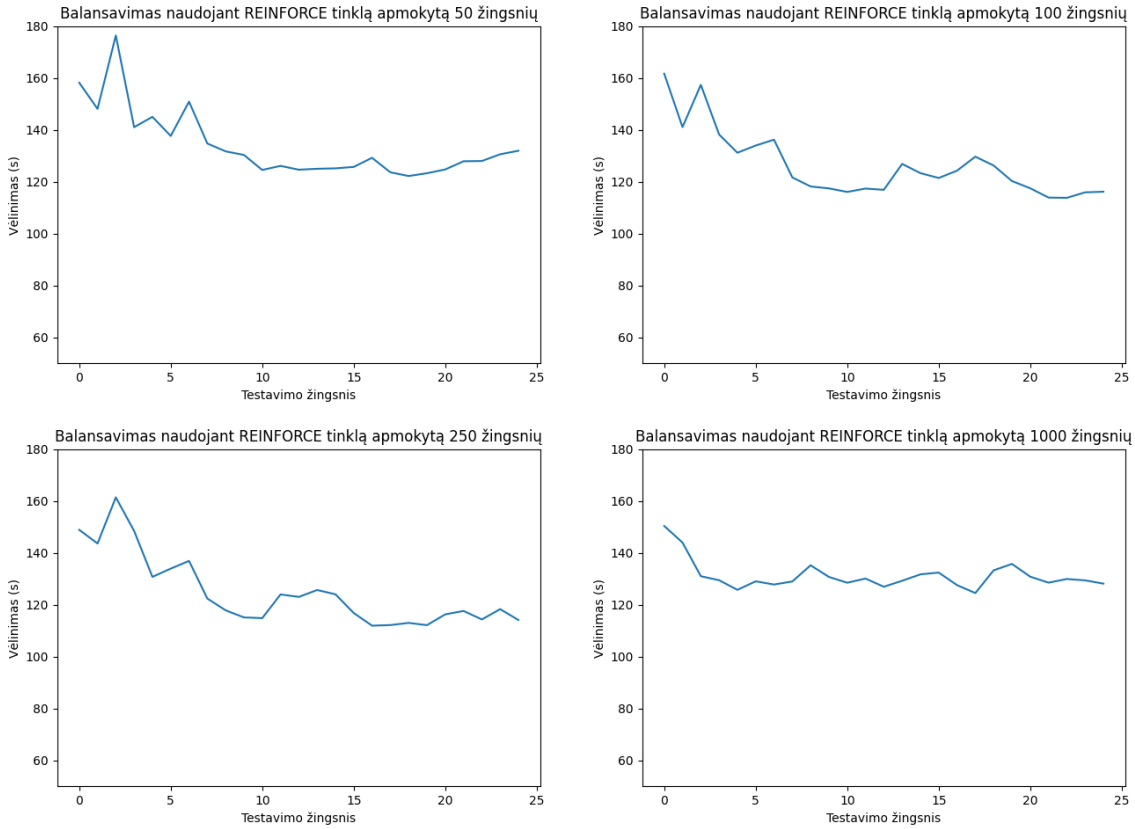
5.4.1. Eksperimentas nr. 1 – balansavimas naudojant REINFORCE tinklą

Po skirtingo apmokymo žingsnių kiekio pasiekti rezultatai pateikti 6 paveikslėlyje, kurie parodo, kad po:

- 50 žingsnių – testavimo metu vėlinimas sumažėja nuo 158 sekundžių iki 131 sekundžių.
- 100 žingsnių – testavimo metu vėlinimas sumažėja nuo 161 sekundžių iki 116 sekundžių.
- 250 žingsnių – testavimo metu vėlinimas sumažėja nuo 148 sekundžių iki 114 sekundžių.
- 1000 žingsnių – testavimo metu vėlinimas sumažėja nuo 150 sekundžių iki 128 sekundžių.

REINFORCE algoritmas su visais išbandytais apmokymo kiekiais testavimo metu pasiekia mažesnį vėlinimą. Visų bandymu metu sumažėjimas yra labai panašus – 15–20 procentų. Apmokius algoritmą 50, 100, 250 žingsnių kiekiu testavimo pradžioje vėlinimas svyruoja, o apmokius 1000 žingsnių – pradinis svyravimas dingsta. Visų bandymų metu REINFORCE tinklo vėlinimo rezultatai po 5–to testavimo žingsnio keičiasi nežymiai (apsibrėžtos paklaidos ribose).

Kadangi algoritmas su visais apmokymo žingsnių kiekiais testavimo pabaigoje pasiekia vienodą vėlinimo rezultatą, galima teigti, kad REINFORCE tinklas balansuoja srautinio apdorojimo sistemą darydamas veiksmus generuojančius panašų atlygį.Tai reiškia, kad tokio tipo algoritmą galima naudoti vėlinimo mažinimui ir stabilios sistemos užtikrinimui. Taip pat, kadangi apmokius tinklą su skirtinais žingsnių kiekiais gaunami panašūs rezultatai, todėl galima teigti, jog šis algoritmas nepasiekia geresnio rezultato su didesniu kiekiu apmokymo žingsnių.



6 pav. Balansavimo su REINFORCE algoritmu testavimo rezultatai

5.4.2. Eksperimentas nr. 2 – balansavimas naudojant Deep Q Network tinklą

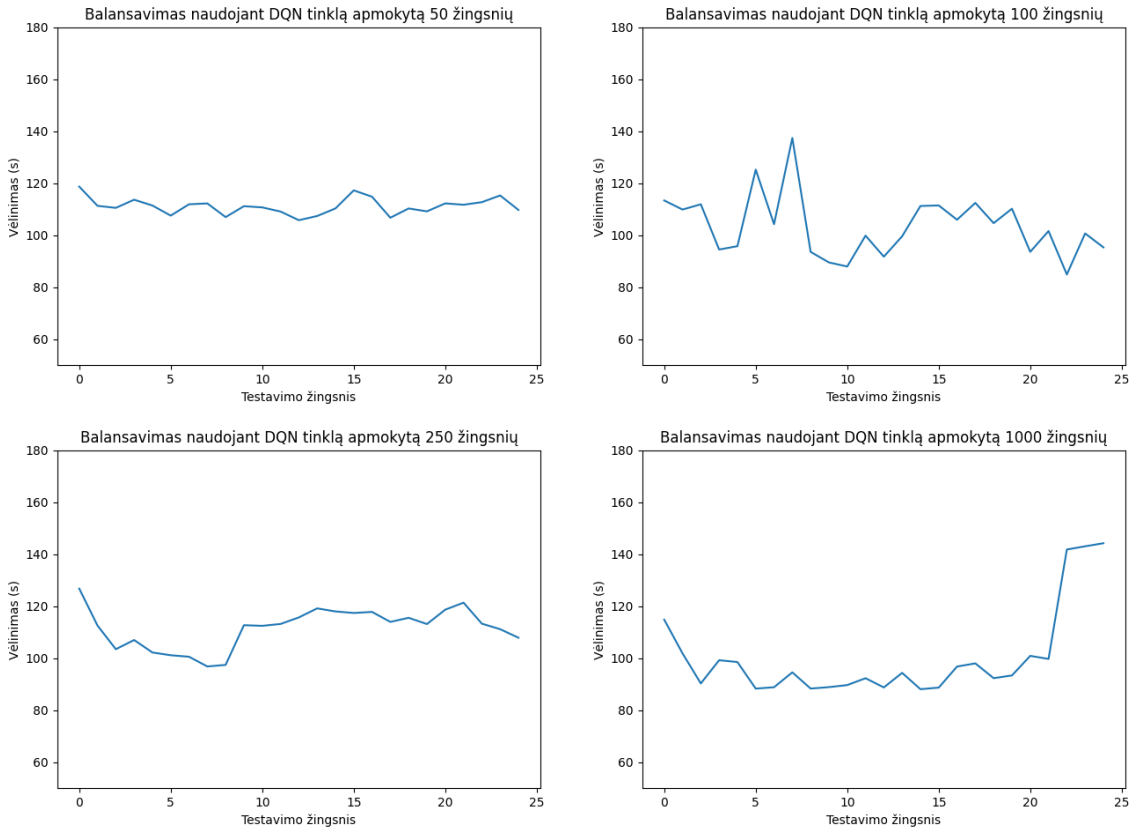
Pasiekti rezultatai pateikti 7 paveikslėlyje po skirtingo apmokymo žingsnių kiekio parodo, kad po:

- 50 žingsnių – testavimo metu vėlinimas sumažėja nuo 118 sekundžių iki 109 sekundžių.
- 100 žingsnių – testavimo metu vėlinimas sumažėja nuo 113 sekundžių iki 95 sekundžių.
- 250 žingsnių – testavimo metu vėlinimas sumažėja nuo 126 sekundžių iki 107 sekundžių.
- 1000 žingsnių – testavimo metu vėlinimas padidėja nuo 115 sekundžių iki 144 sekundžių.

Deep Q Network tinklas po 50, 100 ir 250 apmokymo žingsnių vėlinimą sumažina, tačiau nežymiai – gaunamas vėlinimo skirtumas yra arti apsibrėžtos paklaidos. Tinklas po 100 apmokymo žingsnių atliekant testavimą turi daug vėlinimo svyravimų, o likę bandymai panašių svyravimų neturi. Po 1000 žingsnių apmokytas tinklas pasiekia mažesnį vėlinimą iki 20 žingsnio (nuo 115 iki 92 sekundžių), tačiau po šio žingsnio atliekamas veiksmas, dėl kurio vėlinimas smarkiai padidėja – nuo 100 sekundžių iki 140 sekundžių.

Su visais apmokymo žingsnių kiekiais tinklas testavimo metu atlikdamas balansavimą pasiekia nedidelį vėlinimo pokytį, iš ko galima daryt prielaidą, kad algoritmas mokymosi metu nepakankamai ištyrinėjo veiksmų aibę ir dėl to nepasiekė gerų rezultatų. Net po apmokymo su 1000

žingsnių testavimo pabaigoje tinklas atliko veiksmą, dėl kurio pakyla vėlinimas.



7 pav. Balansavimo su Deep Q Network algoritmu testavimo rezultatai

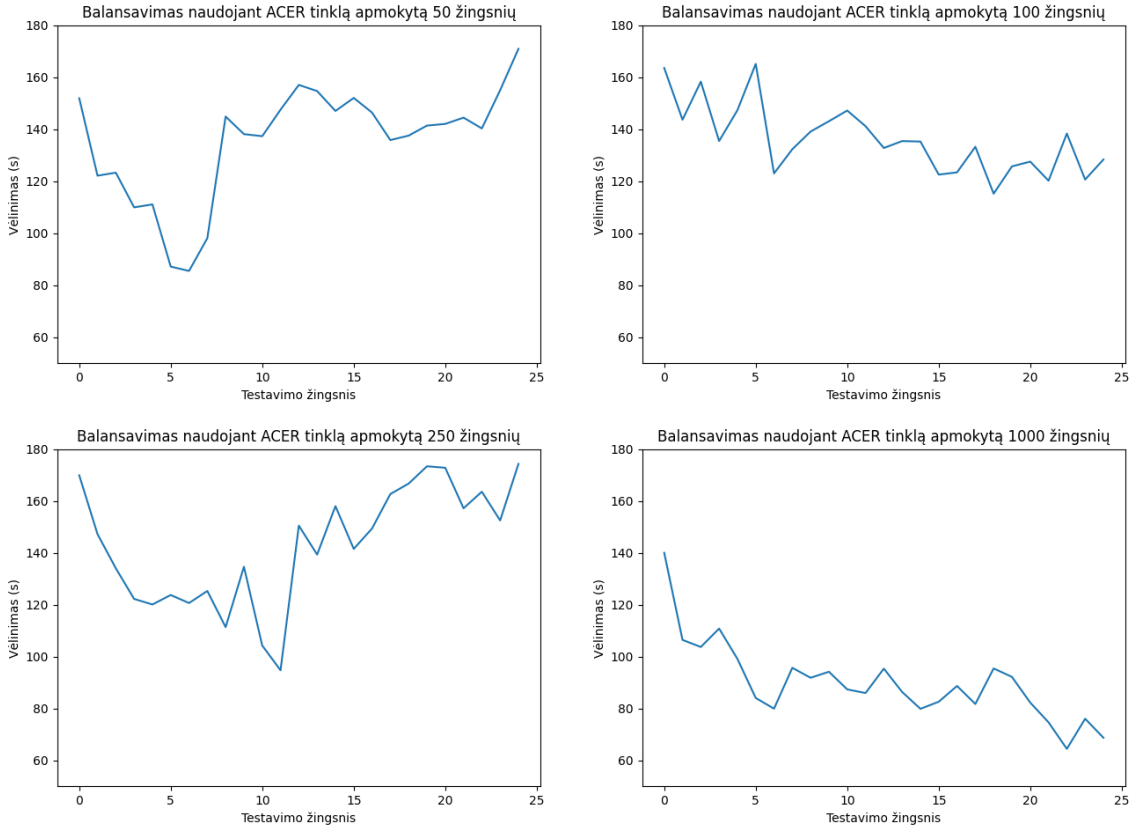
5.4.3. Eksperimentas nr. 3 – balansavimas naudojant Actor–Critic with Experience Replay tinklą

Po skirtingo apmokymo žingsnių kiekio pasiekti rezultatai pateikti 8 paveikslėlyje, kurie parodo, kad po:

- 50 žingsnių – testavimo metu vėlinimas padidėja nuo 151 sekundžių iki 171 sekundžių.
- 100 žingsnių – testavimo metu vėlinimas sumažėja nuo 163 sekundžių iki 128 sekundžių.
- 250 žingsnių – testavimo metu vėlinimas išauga nuo 169 sekundžių iki 174 sekundžių.
- 1000 žingsnių – testavimo metu vėlinimas padidėja nuo 140 sekundžių iki 68 sekundžių.

Actor–Critic with Experience Replay tinklas apmokytas 50 žingsnių po 5–to žingsnio sumažina vėlinimą nuo 151 iki 85 sekundžių, tačiau po to vėlinimas pradeda kilti ir pasiekia didesnę reikšmę negu pradiniame žingsnyje. Panašų sumažinimą turi ir 250 žingsnių apmokytas tinklas, kuris taip pat testavimą baigia su didesniu vėlinimu negu pradiniame žingsnyje. Po 100 apmokymo žingsnių tinklas testavimo metu turi vėlinimo mažinimo tendenciją ir pasiekia mažesnį vėlinimą negu pradiniame žingsnyje, tačiau viso testavimo metu vėlinimo reikšmė svyruoja daugiau

nei paklaida. Tinklas apmokytas su 1000 žingsnių iki 7–to testavimo žingsnio smarkiai sumažina vėlinimą, o tolimesniuose testavimo žingsniuose išlaiko panašų arba mažesnį vėlinimą ir kelių paskutinių žingsnių metu dar labiau sumažina vėlinimą. Ir taip sumažina vėlinimą 50 procentų po viso testavimo.



8 pav. Balansavimo su Actor–Critic with Experience Replay algoritmu testavimo rezultatai

Actor–Critic with Experience Replay algoritmas po apmokymo su 1000 žingsnių pasirodo žymiai geriau negu po apmokymo su 250 žingsnių, kas parodo, jog optimalus apmokymo žingsnių kiekis yra didesni negu 250 žingsnių, tačiau tikslus apmokymo žingsnių kiekis, kai tinklo balansavimo rezultatai konverguoja, nebuvo nustatytas bandymų metu. Tačiau su mažiau nei 1000 apmokymo žingsnių rezultatai yra labai stipriai svyruojantys – po 50 ir 250 apmokymo žingsnių testavimo rezultatai turi staigų sumažėjimą po kurio seka smarkus kylimas ir nusistovėjimas, o po 100 apmokymo žingsnių testavimo rezultatai neturi staigaus mažėjimo, tačiau viso testavimo metu turi mažesnius svyravimus. Todėl galima teigti, jog šis algoritmas su neoptimaliu kiekiu apmokymo žingsnių yra jautrus stebėjimo aibės pokyčiams ir gali pasirinkti netinkamus veiksmus, nes nėra pakankamai ištyrinėjęs veiksmų aibės.

5.5. Eksperimentų rezultatai

Eksperimentų rezultatus nagrinėjant pagal tinkamumo ir stabilumo sąlygas apibrėžtas „Balansavimo algoritmo“ skyriuje galima matyti, kad:

- Tinklas naudojantis Actor–Critic with Experience Replay algoritmą ir apmokytas 1000 žingsnių testavimo metu pasiekia didžiausią vėlinimo pokytį palyginus su pradiniu žingsniu – 50 procentų sumažinimą ir yra tinkamas balansavimo uždaviniui spręsti. Tuo tarpu REINFORCE pasiektas didžiausias pokytis yra 20 procentų sumažinimas ir irgi yra tinkamas balansavimui, o Deep Q Network pokyčio dydis yra arti apsibrėžtos 5 sekundžių paklaidos.
- Tinklas naudojantis REINFORCE algoritmą apmokytas su išbandytais žingsnių kiekiais visada sumažina vėlinimą (15–20 procentų) palyginus su pradiniu žingsniu ir išlaiko sistemą stabilią palyginus su kitais algoritmais. Tuo tarpu, nors ir Deep Q Network neturi didelių vėlinimo svyravimų, tačiau nepasiekia tokio mažo pokyčio.

Rezultatai ir išvados

Rezultatai

1. Apibrėžti srautinių duomenų apdorojimo sistemų balansavimo dalykinę sritį ir identifikuotos valdymo metrikos bei siekiamos metrikų reikšmės.
2. Pasirinkta reklamų analizės srautinio apdorojimo sistema ir REINFORCE, Deep Q Network ir Actor–Critic with Experience Replay skatinamojo mokymosi algoritmai.
3. Sudarytas modelis, ir atlikti eksperimentai su REINFORCE, Deep Q Network ir Actor–Critic with Experience Replay skatinamojo mokymosi algoritmais bei įvertintas siūlomo modelio validumas srautinio apdorojimo sistemų balansavimui.

Išvados

1. Apibrėžus skatinamojo mokymosi aplinką, kuri balansuoja srautinio apdorojimo sistemas, nustatyta, jog vieno žingsnio atlikimo trukmė turi būti ne mažesnė nei 3 minutės siekiant užtikrinti, kad sistema pasiektų vientisą duomenų apdorojimą bei surinktos sistemos metrikos nusistovėtų. Dėl šios priežasties skatinamojo mokymosi tinklo apmokymas reikalauja daug laiko.
2. REINFORCE algoritmas yra tinkamas balansuoti srautines apdorojimo sistemas ir išlaiko sistemą stabilią balansavimo metu. Kadangi šio algoritmo apmokytas tinklas balansuodamas srautinę sistemą vėlinimą sumažina 15–20 procentų ir tinklą apmokius su skirtingais žingsnių kiekiais balansavimas nepasiekia vėlinimo svyravimų didesnių nei 5 sekundės.
3. Atlikus bandymus su Deep Q Network algoritmu ir keičiant apmokymo žingsnių kiekius buvo nustatyta, kad balansuojamos sistemos gautas vėlinimo pokytis buvo panašus į darbo tyrimo dalyje apsibrėžtą paklaidą (5 sekundės) net kai apmokymas buvo daromas su 1000 žingsnių. Tad su darbe apibrėžtu didžiausiu mokymo žingsnių kiekiu tinklui su Deep Q Network algoritmu nepavyko pasiekti reikšmingo vėlinimo pokyčio sumažinimo – didesnio nei apibrėžta 5 sekundžių paklaida.
4. Tyrimo metu buvo nustatyta, kad tinklas su Actor–Critic with Experience Replay algoritmu yra tinkamas balansuoti srautinio apdorojimo sistemas, kurios reikalauja žemo vėlinimo. Kadangi apmokius tinklą naudojantį šį algoritmą su 1000 žingsniu buvo pasiektas 50 procentų vėlinimo sumažinimas. Iš visų atliktų bandymų Actor–Critic with Experience Replay algoritmo pasiektas rezultatas yra didžiausias vėlinimo sumažinimas iš visų analizuotų algo-

ritmų.

5. Remiantis šiomis išvadomis galima teigti, jog srautinio apdorojimo sistemas galima balansuoti naudojant skatinamąjį mokymąsi.

Literatūra

[BBD+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani ir Jennifer Widom. Models and issues in data stream systems. *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, p. 1–16, 2002.

[BC17] Muhammad Bilal ir Marco Canini. Towards automatic parameter tuning of stream processing systems. P. 189–200, 2017-09. DOI: 10.1145/3127479.3127492.

[BCB+18] Manu Bansal, Eyal Cidon, Arjun Balasingam, Aditya Gudipati, Christos Kozyrakis ir Sachin Katti. Trevor: automatic configuration and scaling of stream processing pipelines, 2018. arXiv: 1812.09442 [cs.DC].

[Bea15] Jonathan Beard. A short intro to stream processing. http://www.jonathanbeard.io/blog/2015/09/19/streaming-and-dataflow.html, 2015-09.

[Bon00] André B Bondi. Characteristics of scalability and their impact on performance. *Proceedings of the 2nd international workshop on Software and performance*, p. 195–203, 2000.

[CDE+16] S. Chintapalli, D. Dagit, B. Evans, R. Farivar ir k.t. Benchmarking streaming computation engines: storm, flink and spark streaming. *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, p. 1789–1792, 2016.

[Cho19] Ankit Choudhary. A hands-on introduction to deep q-learning using openai gym in python, 2019.

[DP18] Miyuru Dayarathna ir Srinath Perera. Recent advancements in event processing. *ACM Comput. Surv.*, 51(2), 2018-02. ISSN: 0360-0300. DOI: 10.1145/3170432.

[FA17] Avrilia Floratou ir Ashvin Agrawal. Self-regulating streaming systems: challenges and opportunities. *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics*, BIRTE '17, 1:1–1:5, Munich, Germany. ACM, 2017. ISBN: 978-1-4503-5425-7. DOI: 10.1145/3129292.3129295. URL: http://doi.acm.org/10.1145/3129292.3129295.

[FAG+17] Avrilia Floratou, Ashvin Agrawal, Bill Graham, Sriram Rao ir Karthik Ramasamy. Dhalion: self-regulating stream processing in heron. *Proceedings of the VLDB Endowment*, 10:1825–1836, 2017-08. DOI: 10.14778/3137765.3137786.

[FY11] Zbynek Falt ir Jakub Yaghob. Task scheduling in data stream processing. *DATESO*, p. 85–96, 2011.

[HCL20] Herodotos Herodotou, Yuxing Chen ir Jiaheng Lu. A survey on automatic parameter tuning for big data processing systems. *ACM Computing Surveys (CSUR)*, 53(2):1–37, 2020.

[Her19] Apache Heron. Heron documentation on cluster configuration. `http://heron.incubator.apache.org/docs/cluster-config-overview/`, 2019.

[Her20] Apache Heron. Heron tracker rest api. `https://heron.incubator.apache.org/docs/user-manuals-tracker-rest`, 2020.

[HHD+10] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie ir Bo Huang. The hibench benchmark suite: characterization of the mapreduce-based data analysis. *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*, p. 41–51. IEEE, 2010.

[HLL+11] Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin ir Shivnath Babu. Starfish: a self-tuning system for big data analytics. *Cidr*, tom. 11 numeris 2011, p. 261–272, 2011.

[HSS+14] Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik ir Robert Grimm. A catalog of stream processing optimizations. *ACM Computing Surveys (CSUR)*, 46(4):1–34, 2014.

[YLL+12] Hailong Yang, Zhongzhi Luan, Wenjun Li, Depei Qian ir Gang Guan. Statistics-based workload modeling for mapreduce. *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, p. 2043–2051. IEEE, 2012.

[IS15] Muhammad Hussain Iqbal ir Tariq Rahim Soomro. Big data analysis: apache storm perspective. *International journal of computer trends and technology*, 19(1):9–14, 2015.

[JC16] Pooyan Jamshidi ir Giuliano Casale. An uncertainty-aware approach to optimal configuration of stream processing systems. *CoRR*, abs/1606.06543, 2016. eprint: `1606.06543`.

[KB14] Diederik P. Kingma ir Jimmy Ba. Adam: a method for stochastic optimization, 2014. arXiv: `1412.6980 [cs.LG]`.

[KBF+15] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy ir Siddarth Taneja. Twitter heron: stream processing at scale. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, p. 239–250, Melbourne, Victoria, Australia. ACM, 2015. ISBN: 978-1-4503-2758-9. DOI: 10.1145/2723372.2742788. URL: http://doi.acm.org/10.1145/2723372.2742788.

[KF16] Supun Kamburugamuve ir Geoffrey Fox. Survey of distributed stream processing. *Bloomington: Indiana University*, 2016.

[KKW+15] Holden Karau, Andy Konwinski, Patrick Wendell ir Matei Zaharia. *Learning spark: lightning-fast big data analysis*. " O'Reilly Media, Inc.", 2015.

[KLM96] Leslie Pack Kaelbling, Michael L. Littman ir Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996. URL: https://arxiv.org/abs/cs/9605103.

[KRK+18] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen ir Volker Markl. Benchmarking distributed stream processing engines. *ArXiv*, abs/1802.08496, 2018.

[LCH+19] Jiaheng Lu, Yuxing Chen, Herodotos Herodotou ir Shivnath Babu. Speedup your analytics: automatic parameter tuning for databases and big data systems. *Proceedings of the VLDB Endowment*, 12(12):1970–1973, 2019.

[LHP+15] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver ir Daan Wierstra. Continuous control with deep reinforcement learning, 2015. arXiv: 1509.02971 [cs.LG].

[LTX16] Teng Li, Jian Tang ir Jielong Xu. Performance modeling and predictive scheduling for distributed stream data processing. *IEEE Transactions on Big Data*, 2(4):353–364, 2016.

[LXT+18] Teng Li, Zhiyuan Xu, Jian Tang ir Yanzhi Wang. Model-free control for distributed stream data processing using deep reinforcement learning. *Proc. VLDB Endow.*, 11(6):705–718, 2018-02. ISSN: 2150-8097. DOI: 10.14778/3199517.3199521. URL: https://doi.org/10.14778/3199517.3199521.

[Mel01] Francisco S Melo. Convergence of q-learning: a simple proof. *Institute Of Systems and Robotics, Tech. Rep*:1–4, 2001.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu ir k.t. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[MSH+16] Rémi Munos, Tom Stepleton, Anna Harutyunyan ir Marc G Bellemare. Safe and efficient off-policy reinforcement learning. *arXiv preprint arXiv:1606.02647*, 2016.

[NLY+19] Xiang Ni, Jing Li, Mo Yu, Wang Zhou ir Kun-Lung Wu. Generalizable resource allocation in stream processing via deep reinforcement learning, 2019. arXiv: 1911.08517 [cs.LG].

[NPP+17] Shadi A Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta ir Roy H Campbell. Samza: stateful scalable stream processing at linkedin. *Proceedings of the VLDB Endowment*, 10(12):1634–1645, 2017.

[PGT16] Panagiotis Petridis, Anastasios Gounaris ir Jordi Torres. Spark parameter tuning via trial-and-error, 2016. arXiv: 1607.07348 [cs.DC].

[QWH+16] S. Qian, G. Wu, J. Huang ir T. Das. Benchmarking modern distributed streaming platforms. *2016 IEEE International Conference on Industrial Technology (ICIT)*, p. 592–598, 2016.

[Ram16] Karthik Ramasamy. Open sourcing twitter heron, 2016.

[RCP19] Gabriele Russo Russo, Valeria Cardellini ir Francesco Lo Presti. Reinforcement learning based policies for elastic stream processing on heterogeneous resources. *Proceedings of the 13th ACM International Conference on Distributed and Event-Based Systems*, DEBS '19, p. 31–42, Darmstadt, Germany. Association for Computing Machinery, 2019. ISBN: 9781450367943. DOI: 10.1145/3328905.3329506. URL: https://doi.org/10.1145/3328905.3329506.

[RM19] Henriette Röger ir Ruben Mayer. A comprehensive survey on parallelization and elasticity in stream processing. *ACM Computing Surveys (CSUR)*, 52(2):1–37, 2019.

[SB18] Richard S Sutton ir Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[SÇZ05] Michael Stonebraker, Uğur Çetintemel ir Stan Zdonik. The 8 requirements of real-time stream processing. *ACM Sigmod Record*, 34(4):42–47, 2005.

[SLA+15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan ir Philipp Moritz. Trust region policy optimization. *International conference on machine learning*, p. 1889–1897. PMLR, 2015.

[SSP04] David G. Sullivan, Margo I. Seltzer ir Avi Pfeffer. Using probabilistic reasoning to automate software tuning. *SIGMETRICS Perform. Eval. Rev.*, 32(1):404–405, 2004-06. ISSN: 0163-5999. DOI: 10.1145/1012888.1005739. URL: http://doi.acm.org/10.1145/1012888.1005739.

[Ste97] Robert Stephens. A survey of stream processing. *Acta Informatica*, 34(7):491–541, 1997.

[TLW17] M. Trotter, G. Liu ir T. Wood. Into the storm: descrying optimal configurations using genetic algorithms and bayesian optimization. *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, p. 175–180, 2017.

[TV07] Andrew S Tanenbaum ir Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[TWH19] Michael Trotter, Timothy Wood ir Jinho Hwang. Forecasting a storm: divining optimal configurations using genetic algorithms and supervised learning. *2019 IEEE International Conference on Autonomic Computing (ICAC)*, p. 136–146. IEEE, 2019.

[VC18] Luis M. Vaquero ir Felix Cuadrado. Auto-tuning distributed stream processing systems using reinforcement learning, 2018. arXiv: 1809.05495 [cs.DC].

[WBH+16] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu ir Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

[Wil92] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[WMG+17] Chunkai Wang, Xiaofeng Meng, Qi Guo, Zujian Weng ir Chen Yang. Automating characterization deployment in distributed data stream management systems. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2669–2681, 2017.

[WXH16] Guolu Wang, Jungang Xu ir Ben He. A novel method for tuning configuration parameters of spark based on machine learning. *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, p. 586–593. IEEE, 2016.

[ZGQ+17] Xinwei Zhao, Saurabh Garg, Carlos Queiroz ir Rajkumar Buyya. A taxonomy and survey of stream processing systems. *Software Architecture for Big Data and the Cloud*, p. 183–206. Elsevier, 2017.

[ZYL+20] Yitian Zhang, Jiong Yu, Liang Lu, Ziyang Li ir Zhao Meng. L-heron: an open-source load-aware online scheduler for apache heron. *Journal of Systems Architecture*, 106:101727, 2020.

[ZKZ+15] Nikos Zacheilas, Vana Kalogeraki, Nikolas Zygouras, Nikolaos Panagiotou ir Dimitrios Gunopulos. Elastic complex event processing exploiting prediction. *2015 IEEE International Conference on Big Data (Big Data)*, p. 213–222. IEEE, 2015.

Priedas nr. 1

Visų keičiamų konfigūracijos parametrų lentelė

6 lentelė. Keičiami konfigūracijos parametrai

Parametras	Paaiškinimas
component–parallelism=[skaičiavimo komponento pavadinimas]	Tam tikro skaičiavimo komponento lygiagretumas
heron.instance.tuning.expected.bolt.read.queue.size	Numatomas skaitomos eilės dydis Bolt tipo komponentuose
heron.instance.tuning.expected.bolt.write.queue.size	Numatomas rašomos eilės dydis Bolt tipo komponentuose
heron.instance.tuning.expected.spout.read.queue.size	Numatomas skaitomos eilės dydis Spout tipo komponentuose
heron.instance.tuning.expected.spout.write.queue.size	Numatomas rašomos eilės dydis Spout tipo komponentuose
heron.instance.set.data.tuple.capacity	Didžiausias kiekis kortežų sugrupuotu vienoje žinutėje
heron.instance.emit.batch.time.ms	Didžiausias laikas Spout tipo komponentui išsiųsti gautą kortežą
heron.instance.emit.batch.size.bytes	Didžiausias partijos dydis Spout tipo komponentui išsiųsti gautą kortežą
heron.instance.execute.batch.time.ms	Didžiausias laikas Bolt tipo komponentui apdoroti gautą kortežą
heron.instance.execute.batch.size.bytes	Didžiausias partijos dydis Bolt tipo komponentui apdoroti gautą kortežą
heron.instance.internal.bolt.read.queue.capacity	Skaitomos eilės dydis Bolt komponentams
heron.instance.internal.bolt.write.queue.capacity	Rašomos eilės dydis Bolt komponentams
heron.instance.internal.spout.read.queue.capacity	Skaitomos eilės dydis Spout komponentams
heron.instance.internal.spout.write.queue.capacity	Rašomos eilės dydis Spout komponentams
heron.api.config.topology_container_max_ram_hint	Daugiausiai operatyvios atminties kiekio konteineriui išskyrimo užuomina
heron.api.config.topology_container_max_cpu_hint	Daugiausiai procesoriaus pajėgumo konteineriui išskyrimo užuomina
heron.api.config.topology_container_max_disk_hint	Daugiausiai kietojo disko atminties kiekio konteineriui išskyrimo užuomina
heron.api.config.topology_container_padding_percentage	Užuominų galimą paklaidą

Priedas nr. 2

Keičiamų konfigūracijos elementų aibės lentelė

7 lentelė. Keičiamų konfigūracijos elementų aibė

Parametras	Natūraliųjų reikšmių aibė
heron.instance.tuning.expected.bolt.read.queue.size	$[2; 20]$
heron.instance.tuning.expected.bolt.write.queue.size	$[2; 20]$
heron.instance.tuning.expected.spout.read.queue.size	$[128; 512]$
heron.instance.tuning.expected.spout.write.queue.size	$[2; 20]$
heron.instance.set.data.tuple.capacity	$[64; 512]$
heron.instance.emit.batch.time.ms	$[8; 128]$
heron.instance.emit.batch.size.bytes	$[8192; 65536]$
heron.instance.execute.batch.time.ms	$[8; 128]$
heron.instance.execute.batch.size.bytes	$[8192; 65536]$
heron.instance.internal.bolt.read.queue.capacity	$[64; 512]$
heron.instance.internal.bolt.write.queue.capacity	$[64; 512]$
heron.instance.internal.spout.read.queue.capacity	$[512; 4096]$
heron.instance.internal.spout.write.queue.capacity	$[64; 512]$
heron.api.config.topology_container_max_ram_hint	$[256; 2048]$
heron.api.config.topology_container_max_cpu_hint	$[20; 80]$
heron.api.config.topology_container_max_disk_hint	$[8192; 65536]$
heron.api.config.topology_container_padding_percentage	$[0; 20]$