VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ STUDIJŲ PROGRAMA

Srautinio apdorojimo sistemų balansavimas taikant mašininį mokymąsi

Balancing stream processing systems using machine learning

Mokslo tiriamasis darbas II

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

Vilnius – 2020

1. Įvadas

Šio darbo tikslas - apžvelgti susijusius literatūros šaltinius, pateikti jų santraukas ir panaudoti šią informacija suformuluoti problemą, kuri bus nagrinėjama sekančiame darbe.

Realaus laiko duomenų apdorojimas (angl. real-time data processing) yra jau senai nagrinėjamas kaip vienas iš būdų apdoroti didelių kiekių duomenis (angl. Big data). Vienas iš realaus laiko apdorojimo sprendimų yra srautinis duomenų 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].

Yra keli pagrindiniai srautinio apdorojimo varikliai: „Apache Storm“, „Apache Spark“, „Heron“ ir kiti. „Apache Storm“ ir „Heron“ apdoroja duomenis ne duomenų srautais, o „Apache Spark“ mikro-paketais [KKW+15]. „Heron“ srautinio apdorojimo variklis, buvo išleistas „Twitter“ įmonės 2016 metais kaip greitesnė alternatyva „Apache Storm“ srautinio apdorojimo varikliui [Ram16].Šiame darbe bus naudojamas „Heron“, kadangi tai yra naujesnis ir greitesnis srautinio apdorojimo variklis nei „Apache Storm“ [KBF+15].

Srautinio apdorojimo sistemų balansavimas (angl. auto-tuning) - tai sistemos konfigūracijos valdymas siekiant užtikrinti geriausią resursų išnaudojimą - duomenų apdorojimas neprarandant greičio, bet ir naudojant tik reikiamą kiekį resursų. Kadangi srautinio apdorojimo sistemų komponentai yra kuriami kaip lygiagretus skaičiavimo elementai, todėl 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. Darbe naudojamas „Heron“ variklis leidžia optimizuoti sistemas naudojant 56 konfigūruojamus svertus (angl. levers) [Her19].

Yra skirtingi būdai kaip gali būti parenkama tinkama konfigūracija. Kadangi dar nėra naudojimui paruoštų sprendimų, kurie galėtų balansuoti srautinio apdorojimo sistemas savarankiškai, dažniausiai tai daro duomenų inžinieriai, kurie dirba su šiomis sistemomis. Kadangi srautinio apdorojimo sistemų apkrovos gali būti skirtingų pobūdžiu (duomenų kiekis, skaičiavimų sudėtingumas, nereguliari apkrova), o inžinieriai konfigūruodami išbando tik keletą derinių ir pasirenka labiausiai tinkanti [FA17], lieka labai daug skirtingų neišbandytų konfigūracijos variacijų. Optimalios konfigūracijos suradimas yra NP sudėtingumo problema [SSP04], nes 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 ap-

rašo savo sukurtą sprendimą „Dhalion“, kuris konfigūruoja „Heron“ srautinio apdorojimo sistemas pagal esama apkrova ir resursus, t.y. jei apdorojimo elementų išnaudojimas išauga >100%, „Dhalion“ padidina lygiagrečiai dirbančių apdorojimo elementų kiekį [FAG+17]. Tačiau šis sprendimas leidžia reguliuoti tik elementų lygiagretumą ir tai daro tik reaktyviai. Vienas iš naujausių būdų balansuoti srautinio apdorojimo sistemas - mašininis mokymasis. Vienas iš tokių bandymų buvo aprašytas 2018 metų straipsnyje „Auto-tuning Distributed Stream Processing Systems using Reinforcement Learning“, kuriame buvo atliktas tyrimas - „Apache Spark“ sistemos balansavimui buvo naudojamas skatinamojo mokymo REINFORCE algoritmas, kuris pagal dabartinę konfigūraciją ir gaunamas metrikas keitė srautinio apdorojimo sistemos konfigūracijas. Šiame tyrime buvo nustatyta, jog sprendimas, naudojantis mašininį mokymąsi, suranda konfigūraciją per trumpesnį laiką nei žmones ir taip pat sukurtą konfigūraciją naudojančios srautinio apdorojimo sistemos laukimo trukme (angl. latency) yra 60-70% mažesnė nei tyrimo metu žmonių sukurtos konfigūracijos [VC18]. Šiame darbe naudojamas „Heron“ variklis leidžia prie savęs prijungti sukurtą išorinę metrikų surinkimo programą, kuri gali rinkti tokias sistemų metrikas kaip: naudojama RAM atmintis, CPU apkrova, komponentų paralelizmas ir kitas, kurios bus naudojamos sistemos balansavimui.

Skatinamasis mokymasis yra vienas iš mašininio mokymosi tipų. Šis mokymasis skiriasi nuo kitų, nes nereikia turėti duomenų apmokymui, o programos mokosi darydamos bandymus ir klysdamos. Pagrindinis uždavinys naudojant skatinamąjį mokymąsi yra surasti balansą tarp naujų sprendimų tyrinėjimo (angl. exploration) ir turimos informacijos išnaudojimo (angl. exploitation) [JOA10]. 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 apdovanojimo (angl. reward) funkciją ir turi būti teisingai aprašytas balansas tarp tyrinėjimo ir išnaudojimo tam, kad nebūtų patirti nuostoliai [FA17].

Yra sukurta daug skatinamojo mokymosi algoritmų (Monte Carl, Q-learning, Deep Q Network ir kiti), tad šiame darbe jie bus apžvelgti ir vienas iš jų bus pasirinktas ir pritaikytas išsikeltam uždaviniui. Algoritmas bus pasirinktas pagal tai, kuris yra tinkamiausias srautinių apdorojimų sistemų balansavimui.

Numatomas magistro darbo tikslas: Ištirti mašininio mokymosi tinkamumą srautinio apdorojimo sistemų balansavimui.

Numatomi magistro darbo uždaviniai:

1. Apibrėžti duomenų pobūdį ir apkrovas, kurios bus naudojamos eksperimente bei pasirinkti srautinio apdorojimo sistemų metrikas, kurios bus naudojamos eksperimento rezultatų paly-

ginimui.

2. Atlikti literatūros analizė apie esamus skatinamojo mokymosi algoritmus ir pasirinkti vieną iš jų eksperimentui.
3. Sukurti eksperimentinį sprendimą, kuris pritaiko pasirinktą mašininio mokymosi algoritmą srautinio apdorojimo sistemų balansavimui.
4. Atlikti eksperimentą ir palyginti gautus rezultatus su alternatyvomis - „Heron“ su standartine konfigūracija, „Heron“ su „Dhalion“ priedu bei „Heron“ balansavimas pritaikius REINFORCE algoritmą.

1.1. Šio darbo tikslas ir uždaviniai

Tikslas: parengti literatūros apžvalgą, kurioje būtų suformuluota darbe numatoma spręsti problema.

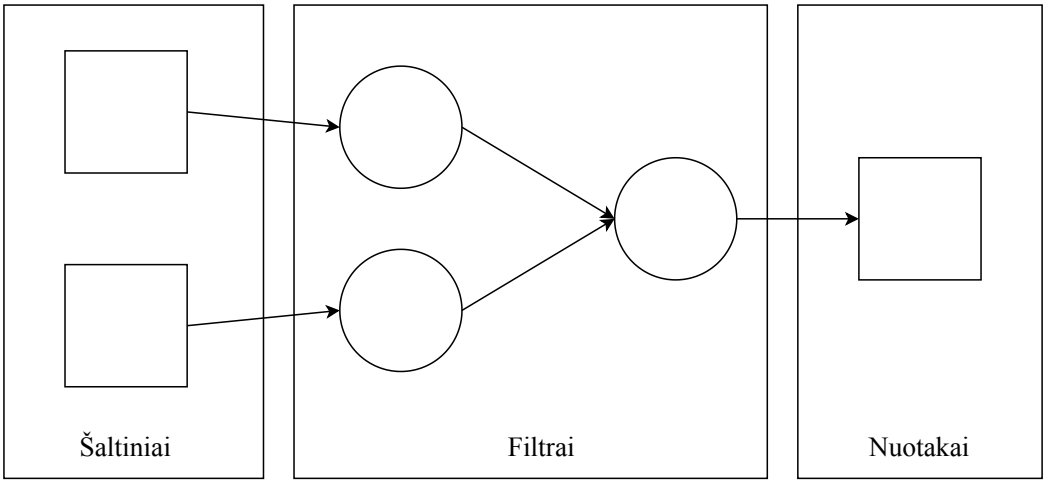
Uždaviniai:

1. Išnagrinėti kaip mokslinėje literatūroje matuojamos srautinio apdorojimo sistemos.
2. Kokius skatinamojo mokymosi algoritmus naudoja kiti autoriai ir kokius rezultatus gauna.

2. Srautinio apdorojimo sistemų matavimas ir derinimas

2.1. 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. Tokios 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.). Srautinių apdorojimo sistemos skiriasi nuo reliacinių modelio [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 yra sunku.

2.1.1. Duomenų vykdymas

Srautinio apdorojimo sistemų veikimui reikalingas srautinio apdorojimo variklis (angl. stream processing engine). Šie varikliai yra skirti srautinio apdorojimo sistemų vykdymui, dislokavimui ir užtikrinti plečiamumą (angl. scaling) ir toleranciją gedimams (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 komponentus [ZGQ+17]:

- Planavimas (angl. scheduling) - duomenų apdorojimo užduočių planavimas daro įtaką bendram srautinio apdorojimo veikimui [FY11]. Pavyzdžiui ”Apache Samza” naudoja ”Apache YARN” resursų valdymo sistemą, kuri turi planavimo posistemę, kuri skirsto resursus [NPP+17]
- Plečiamumas (angl. scalability) - apibrėžia daug apdorojimo branduolių turinčios sistemos gebėjimą apdoroti didėjanti kiekį užduočių ir galimybę didinti pačią sistemą kad susidoroti su didėjančių kiekiu duomenų [Bon00]. Srautinio apdorojimo varikliai turi užtikrinti srautinio apdorojimo sistemų plečiamumą [SÇZ05].
- Išskirstytas skaičiavimas (angl. Distributed computation) - tarpusavyje nesusiję skaičiavimo elementai turi naudotojam atrodyti kaip viena darni sistemą [TV07]. Srautinio apdorojimo varikliai turi užtikrinti darbų paskirstymą ir skaičiavimo įrenginių koordinacija, kad daug 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” šis elementas vadinimas ”topology”, kuris yra užrašomas Java kalba naudojant ”Apache Storm” pateiktą biblioteką [IS15].

2.1.2. Duomenų priėmimas

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

2.2. Srautinio apdorojimo sistemų matavimas

Svarbiausias srautinio apdorojimo sistemų reikalavimas - duomenų apdorojimas ir rezultatų grąžinimas turi būti be atsilikimo - didelių apimčių srautiniai duomenys turi būti apdorojami taip

pat greitai kaip ateiną [SÇZ05].

2.2.1. Srautinio apdorojimo sistemų metrikos

Kitų autorių naudojamos metrikos:

- Pralaidumas (angl. Throughput) - per tam tikrą laiko tarpą apdorojamų įvykių kiekis. Kuo didesnis pralaidumas tuo greitaveiką didesnė.
- Gaišaties laikas (angl. Latency) - laiko intervalas nuo apdorojimo arba įvykio pradžios iki apdorojimo pabaigos. Kuo jis mažesnis tuo greitaveiką didesnė.

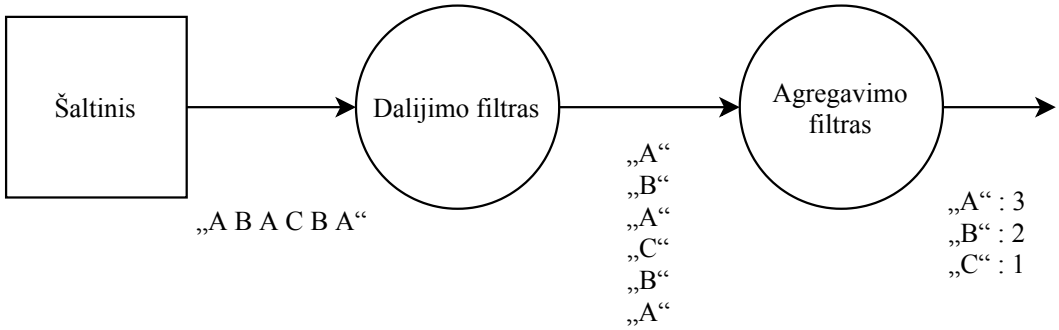
Gaišaties laikas ir pralaidumas dažniausiai nekoreliuoja - sistemos kurios apdoroja srautus paketais turi didesnį pralaidumą, tačiau laikas per kurį renkamas paketas duomenų paveikia laiką kada duomenis patenka į apdorojimo sistemą [KRK+18].

[SÇZ05] straipsnyje minima, jog srautinio apdorojimo sistemos naudotojas turi išbandyti savo sistemą su tiksliniu darbo krūviu ir išmatuoti pralaidumą ir gaišaties laiką. [KRK+18] lyginą srautinio apdorojimo variklius ir matavimui naudoja gaišaties laiką, kurį išskaido į įvykio gaišaties laiką (angl. event-time latency) - laiko intervalas nuo įvykio laiko iki rezultato gavimo iš srautinio apdorojimo sistemos ir apdorojimo gaišaties laiką (angl. processing-time latency) - laiko intervalas nuo duomens patekimo į srautinio apdorojimo sistemą iki rezultato grąžinimo. Šį skaidymą autoriai daro, nes vertinant sistemas dažnai ignoruojamas įvykio laikas ir gaunamas daug mažesnis gaišaties laikas negu jis iš tikro yra. Taip pat autoriai išskiria darnų pralaidumą (angl. sustainable throughput) - didžiausia apkrova įvykių, kurią sistema gali apdoroti be pastoviai augančio įvykio gaišaties laiko, todėl savo eksperimentuose autoriai užtikrina kad duomenų generavimo greitis atitiktų sistemos darnų pralaidumą. Tam kad sužinoti darnų pralaidumą sistemos autoriai pradeda leisti labai didelį srautą duomenų ir mažina iki kol sistemos apdorojimas palaiko generavimo greičius. Visus gaišaties laiko rezultatus autoriai pateikia maksimalaus pralaidumo apdorojimo ir 90% pralaidumo apdorojimo vidurkiu, minimumu, maksimumu, kvantiliais (90, 95, 99). [HSS+14] autoriai nagrinėja srautiniam apdorojimui galimas optimizacijas ir matavimui naudoja normalizuotą pralaidumą (naudojamas vienetas kaip vidurkis), nes tai leidžia lengviau palyginti santykinę greitaveiką. Taip pat [HSS+14] pastebi, kad nors ir yra daug metrikų, kuriomis galima matuoti optimizaciją: pralaidumas, gaišaties laikas, paslaugos kokybė (angl. quality of service), energijos ir tinklo panaudojimas, tačiau pagerinus pralaidumą pagerėja ir visos kitos metrikos. [QWH+16] srautinių apdorojimo sistemų matavimui naudoja pralaidumą (skaičiuojamą baitais per sekundę) ir gaišaties laiką, kaip vidurkį nuo duomens patekimo į sistemą iki apdorojimo pabaigos. Taip pat

kadangi autoriai lyginą srautinio apdorojimo variklius jie įveda metriką gedimo toleravimo (angl. fault tolerance) vertinimui, kuriam autoriai išjungia tam tikrą kiekį elementų ir matuoja pralaidumą ir gaišaties laiką. [ZYL+20] palyginimui naudoja sistemos įvykdymo gaišaties laiką (angl. system completion latency), kuris parodo vidutinį laiko tarpą per kurį kortežas nukeliauja nuo šaltinio iki sistemos galutinio taško. Autoriai skaičiavo vidutinį laiką 5 sekundes intervalais. Taip pat autoriai naudojo matavimui CPU apkrovą kiekvienos instancijos (angl. instance), CPU apkrovą kiekvieno darbinio mazgo (angl. worker node) ir apkrovą tarp instancijų/tarp mazgų, kadangi jų užduotis buvo patobulinti esamą planavimo posistemę. [FAG+17] matuoja pralaidumą per minutę. [VC18] tyria labai panašią problemą - srautinių apdorojimo sistemų balansavimą taikant skatinamąjį mokymą ir matavimui naudoja 99 kvantilį gaišaties laiko. [CDE+16] naudoja gaišaties laiko srautinio apdorojimo variklių vertinimui.

2.2.2. Srautinio apdorojimo sistemos pobūdis

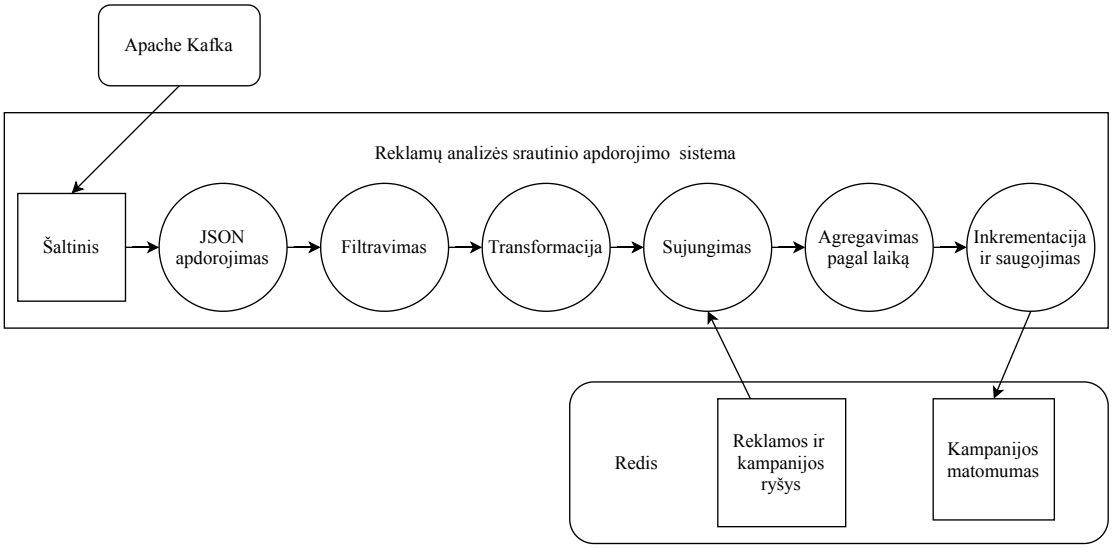
Srautinio apdorojimo sistemos gali turėti labai skirtingą elementų išsidėstymą ir nuo to priklauso jų greitaveiką. [KRK+18] matavimui naudojo du filtrus - agregavimo, kuris skaičiavo visus pirkimus ir jungimo (angl. join) kuris skaičiavo duomenis pagal tam tikrą bendrą rodiklį iš abiejų duomenų srautų. [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 elementų: pirmas padalina teksto eilutę į žodžius, antras agreguoja kiekvieno žodžio bendra skaitiklį ir atnaujina bendrą rezultatą žodžių dažnio, kur raktas tai žodis, o reikšmė tai kiek kartojosi šis žodis (2 pav.).



2 pav. WordCount sistemos pavyzdys

[ZYL+20] matavimui naudoja WordCount sistemą, kuri yra paprastesnė, nei 2 pav., nes šaltinis generuoja ir siunčia po vieną žodį ir yra tik agregavimo filtras, ir naudoja SentenceWordCount

sistemą, kuri yra identiška 2 pav. pavaizduotai sistemai. Taip pat autoriai sukūrė FileWordCount sistemą, kuri atlieką tą patį kaip ir SentenceWordCount, tačiau šaltinis ne pats generuoja žodžius, o skaito iš tekstinio dokumento ir taip pat naudojo egzistuojančią Yahoo srautinio apdorojimo vertinimą (angl. benchmarking) [CDE+16]. [FAG+17] naudojo WordCount eksperimentui. [VC18] eksperimentams naudojo Yahoo srautinio apdorojimo palyginimą [CDE+16] ir taip pat darė bandymus su realiais daiktų interneto (angl. internet of things) įmonės duomenimis. [CDE+16] apibrėžia srautinio apdorojimo sistemą vertinimui skirtingu srautinio apdorojimo variklių. Ši sistemą analizuoja reklamas pagal tipą ir matomumą ir rezultatus deda į Redis duomenų bazę. Sistema sukurta taip, kad aprėptu visas srautinio apdorojimo sistemos savybes (3 pav.).



3 pav. Reklamų analizės sistema [CDE+16]

2.2.3. Srautinio apdorojimo sistemų matavimo 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 kur siunčiami kortežai (angl. tuples) susidarantys iš nupirktos valiutos kiekio, laiko ir naudotojo kuris nupirko ir reklamų srautas kuris siunčia nagrinėja valiutos pirkimo siūlymus tam tikru laiku. Generuojami duomenis naudojant normalizuotą paskirstymą ant raktinio lauko. [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 pakeista planavimo posisteme ir naudoja iš anksto sugeneruotą tekstą patalpintą į tekstinį dokumentą. [CDE+16] aprašo sistemą kuri daro skirtingų srautinio apdorojimo variklių palyginimus. Šiam vertinimui naudojami duomenys simuliuojantys reklamas.

Autoriai naudoja savo duomenų generatorių.

2.3. Srautinio apdorojimo sistemų derinimas

[ZYL+20] - čia derinimas L-HERON

3. Skatinamojo mokymosi algoritmai

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.

[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.

[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.

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

[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.

[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.

[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.

[JOA10] Thomas Jaksch, Ronald Ortner ir Peter Auer. Near-optimal regret bounds for reinforcement learning. *J. Mach. Learn. Res.*, 11:1563–1600, 2010-08. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1756006.1859902.

[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.

[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.

[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.

[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.

[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.

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

[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.

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

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

[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.