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 III

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

Vilnius – 2021

**TURINYS**

ĮVADAS ..... 3

1. SRAUTINĖS ARCHITEKTŪROS SISTEMOS, VALDOMOS GRĮŽTAMUOJU RYŠIU, MODELIS ..... 6
   1.1. Modelis ..... 6
   1.2. Keičiami konfiguracijos parametrai ..... 6
   1.3. Naudojamos metrikos ..... 7
   1.4. Balansavimo tikslo funkcija ..... 8

2. BALANSAVIMO ALGORITMAS ..... 9
   2.1. Srautinės architektūros sistemos konfiguracijos valdymo algoritmas ..... 9
   2.2. Srautinio duomenų apdorojimo aplinkos apibrėžimas balansavimui ..... 10

3. EKSPERIMENTO TYRIMO PLANAS ..... 12
   3.1. Tyrimo tikslas ..... 12
   3.2. Eksperimentinė sistema ..... 12
   3.3. Planuojamų eksperimentų apimtis ..... 12

IŠVADOS ..... 14

LITERATŪRA ..... 15

# 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 apdororojimo 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].

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 kaip patobulinta 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. Pavyzdžiui, darbe naudojamas „Heron“ variklis leidžia optimizuoti 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 taikomasias sistemas išbando tik kelis derinius ir pasirenka labiausiai tinkanti [FA17], lieka daug skirtingų neišbandytų konfigūracijos variacijų. 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 apkrova ir turimus resursus, tai yra jei apdorojimo elementų išnaudojimas išauga virš 100%, „Dhalion“ padidina lygiagrečiai dirbančių apdorojimo elementų kiekį [FAG+17]. Tačiau toks 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 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]. 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 gali būti naudojamos 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. 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 balansą tarp tyrinėjimo ir išnaudojimo tam, kad nebūtų patiriami nuostoliai [FA17].

Yra sukurta daug skatinamojo mokymosi algoritmų (Monte Carl, Q–learning, Deep Q Network ir kiti), šiame darbe jie apžvelgti, pasirinkti ir pritaikytas išsikeltam uždaviniui.

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

Magistro darbo uždaviniai:

1. Sudaryti srautinio apdorojimo sistemų balansavimo modelį ir nustatyti valdymo metrikas ir jų siekiamas reikšmes, kurios bus naudojamos eksperimentinėje sistemoje.
2. Parinkti skatinamojo mokymosi algoritmą eksperimentui, atsirenkant iš algoritmų, aprašomų literatūroje.
3. Sukurti eksperimentinį sprendimą su pasirinktu algoritmu ir atlikti eksperimentus.
4. Palyginti eksperimento rezultatus su alternatyvomis – „Heron“ su standartine konfigūracija bei „Heron“ balansavimas pritaikius REINFORCE algoritmą.

Antroje magistro darbo dalyje atlikta literatūros analizė, sudarytas balansavimo modelis ir parinkti

skatinamojo mokymosi algoritmai, o šiame mokslo tyriamajame darbe – trečio magistro darbo dalyje – siekta tikslo : apibrėžti darbo metodą ir sukurti bei pagrįsti algoritmą, kuris bus naudojamas eksperimentui. Ir įgyvendinti šie uždaviniai:

1. Apibrėžti algoritmą srautinio apdorojimo sistemų balansavimui naudojanti Deep Q Network ir Soft Actor Critic skatinamojo mokymosi algoritmus.
2. Apibrėžti eksperimento eigą ir altenatyvius sprendimus, kurių rezultatai bus naudojami algortimo įvertinimui.

# 1. Srautinės architektūros sistemos, valdomos grįžtamuoju ryšiu, modelis

## 1.1. Modelis

## 1.2. Keičiami konfiguracijos parametrai

Norint koreguoti konfiguracijos parametrus reikia siųsti atnaujinimo komandą į Heron komandinės eilutė įrankį (toliau Heron CLI). Peteikus konfiguracijos parametrus Heron platformą perkrauna srautinio apdorojimo sistema su naujais parametrais. Vienas iš pagrindinių srautinės apdorojimo sistemos konfiguracijos parametrų yra skaičiavimo komponentų lygiagretumas nurodantis kiek paleidžiama tam tikro komponento instancijų. Taip pat tai yra vienintelis keičiamas konfiguracijos elementas, kuris priklauso nuo srautinio apdorojimo sistemos.

Visi kiti konfiguravimo parametrai, kurie taip pat bus keičiami balansavimo metu pateikti 3 lentelėje:

1 lentelė. Keičiami konfiguracijos 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 sugrupuottu vienoje žinutėje |
| heron.instance.ack.batch.time.ms | Didžiausias laikas Spout tipo komponentui atlikti ACK |

| | |
|---|---|
| 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 pajegumo 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 |

## 1.3. 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, sukurti savo metrikų skaitymo priedą ir pateikti jį į Heron platformą prieš ją paleidžiant. Visos metrikos yra saugomos srautinio apdorojimo sistemos kiekvienam skaičiavimo komponentui.

2 lentelėje aprašytos metrikos yra standartinės visiems Heron srautinio apdorojimo sistemos komponentams ir gražinamos iš Heron per Heron API. Šios metrikos ir bus teikiamos į mašininio

mokymosi algortimą apibūdinti aplinkai.

2 lentelė. Naudojamų metrikos

| Metrikos pavadinimas | Paaiškinimas |
|---|---|
| __emit–count | Išsiųstų kortežų kiekis |
| __execute–count | Apdorotų kortežų kiekis Bolt komponentuose |
| __complete–latency | Vidutinė trukmė Spout komponentui nuo kortežo išsiųntimo iki FAIL arba ACK komandos |
| __execute–latency | Vidutinė trukmė Bolt komponentui apdoroti kortežą |
| __process–latency | Vidutinė trukmė Bolt komponentui nuo kortežo apdorojimo pradžios iki kito komponento ACK arba FAIL komandos |
| __jvm–process–cpu–load | JVM apkrova procesoriui |
| __jvm–memory–used–mb | JVM išnaudota operatyvi atmintis |
| __jvm–memory–mb–total | JVM turima operatyvi atmintis |
| __jvm–gc–collection–time–ms | JWM šiušklių surinkimo trukmė |
| __time_spent_back_pressure_by_compid | Kiek laiko komponentui buvo įjungtas priešslėgio režimas |

## 1.4. Balansavimo tikslo funkcija

# 2. Balansavimo algoritmas

## 2.1. Srautinės architektūros sistemos konfiguracijos valdymo algoritmas

Algoritmo tikslas – pagal esamą būseną ir esamą konfiguraciją apskaičiuoti naują konfiguraciją, kuri pagerintų srautinės apdorojimo sistemos greitaveiką. Algortimo pagrindas yra pasirinktas skatinamojo mašinio mokymosi algortimas.

Kad algortimas galėtų apskaičiuoti naują konfiguraciją, jam yra paduodami šie duomenys:

- Srautinio apdorojimo sistemos metrikos (2 lentelė)
- Aplinkos naudojami resursai – procesoriaus apkrova procentais, operatyviosios atminties apkrova procentais.
- Srautinio apdorojimo sistemos pradinė konfiguracija (3 letntelė).
- Konfiguruojamų parametrų sąrašas ir jų galimos reikšmės, aprašytos intervalu (pavyzdžiui $[512,4096]$) arba tiksliais variantais (pavyzdžiui $[512, 1024, 2048, 4096]$).

Duomenis, kurie paduodami balansavimo algoritmui yra renkami pastoviai, neatsižvelgiant į pačio algoritmo viekimą.

Balansavimo algortimas veikia ciklais, visą srautinės apdorojimo sistemos veikimo laiką ir tarpas tarp ciklų turi būti pakankamai ilgas srautinio apdorojimo sistemai atsinaujinti su naujais konfiguracijos parametrais bei, kad surinktų duomenų kiekis būtų pakankamai svarūs pateikti mašininio mokymosi algoritmui. Balansavimo algoritmas, atlikęs skaičiavimus, gražina naują konfiguracijos parametrų rinkinį ir laukia naujo ciklo pradžios.

Balansavimui naudojami skatinamojo mašininio mokymosi algoritmai Deep Q network ir Soft Actor Critic turi bendrus veikimo bruožus:

- Jie yra model–free tipo skatinamojo mokymosi algoritmai – tai yra jie daro sprendimus pagal tai ką išmoko veikimo metu ir negeneruoja bendro modelio. Tai aktualu mūsų atveju kadangi balansavimo algortimas turi galėti prisitaikyti prie kintačių duomenų kiekio ir greičio bei kintačios aplinkos.
- Jie yra off–policy tipo – jie mokosi atliekant skirtingus veiksmus nebūtinai tuos kurie buvo atlikti su dabartine strategija.

Tačiau šie algoritmai buvo pasirinkti nes turi esmini skirtumą – Deep Q Network galimų veiksmų aibę priima kaip rinkinį, o Soft Actor Critic kaip intervalą, todėl naudojant šiuos tinklus konfiguruojamų parametrų sąrašas yra skirtingas.

## 2.2. Srautinio duomenų apdorojimo aplinkos apibrėžimas balansavimui

Balansavimui atlikti reikalingas tikslus apibrėžimas srautinio apdorojimo aplinkos, šis apibrėžimas susidaro iš: dabartinės srautinio apdorojimo sistemos būsenos, galimų veiksmų aibės ir žingsnio funkcijos, kuri atlieka veiksmą ir apskaičiuoja atpilda.

Srautinio apdorojimo sistemos dabartinė būsena – srautinio apdorojimo sistemos metrikos patalpintos masyvę.

Srautinio apdorojimo sistemos balansavimo galimų veiksmų aibė – kadangi Deep Q Network reikalauja diskrečių reikšmių, o Soft Actor Critic testinių reikšmių aibės, jos bus skirtingai paduodamos į balansavimo algoritmą, tačiau reikšmių ribos vienodos.

3 lentelė. Galimų veiksmų aibė

| Parametras | Naturaliųjų reikšmių aibė |
|---|---|
| component–parallelism=[skaičiavimo komponento pavadinimas] | $[1; 20] * \text{Komponentų kiekis}$ |
| 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.ack.batch.time.ms | $[64; 512]$ |
| 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 | $[0; 20]$ |

Srautinio apdorojimo sistemos balansavimo žingsnio funkcija:

- Atliekamas veiksmas - naujos konfiguracijos įkelimas ir nustatyto laiko laukimas, kad srautinio apdorojimo sistema pasiektų apkrovą.
- Atpildas - skaitinė reikšmė apskaičiuotą naudojant normalizuotas metrikas su koeficientais ir 1 formulė.

$$\begin{aligned} \text{Atpildas} = &\text{„emit-count“} + \text{„execute-count“} - \text{„complete-latency“} - \\ &\text{„execute-latency“} - \text{„process-latency“} - (100 - \text{„jvm-process-cpu-load“}) - \\ &(|\,\text{„jvm-memory-mb-total“} - \text{„jvm-memory-used-mb“}\,|) - \\ &\text{„jvm-gc-collection-time-ms“} - \text{„time\_spent\_back\_pressure\_by\_compid“} \end{aligned} \tag{1}$$

# 3. Eksperimento tyrimo planas

## 3.1. Tyrimo tikslas

Šio tyrimo tikslas – įvertinti siūlomo balansavimo modelio ir pasirinkto optimizavimo algortimo validumą. Tam reikia atlikti bandymus su eksperimentine sistema naudojančią aprašytą optimizavimo algoritmą, su eksperimentine sistema naudojančia REINFORCE algoritmą ir su aplinka naudojančią standartinę konfiguracija be jokių pakeitimų. Gautus bandymo duomenis palyginti ir nustatyti, ar pasiūlytas sprendimas tinką srautinio apdorojimo sistemų balansavimui.

## 3.2. Eksperimentinė sistema

Pridėti čia tikslia diagrama su komponentais

1. Čia bus posistemių aprašymas

## 3.3. Planuojamų eksperimentų apimtis

Magistro darba eksperimentai bus atliekami su keturiais skirtigais sprendimais:

- Srautinio apdorojimo sistemos veikimas su standartine konfiguracija.
- Srautinio apdorojimos sistemos veikimas balansuojant ją naudojant sukurtą eksperimentinį sprendimą pagal apibrėžtą balansavimo algoritma naudojant:
  - REINFORCE algortimą skatinamojo mokymosi posistemėje.
  - Deep Q Network algoritmą skatinamojo mokymosi posistemėje.
  - Soft Actor Critic algortimą skatinamojo mokymosi posistemėje.

Kiekvienas sprendimas bus testuojamas su dviem srautinio apdorojimo sistemų implementacijomis:

- Reklamų analizės srautinio apdorojimo sistema.
- WordCount srautinio apdorojimo sistema.

Reklamų analizės sistema rezultatus talpina tekstini failą, kur kiekvienas įrašas yra vėlinimas milisekundėmis nuo paskutinio išsiųsto įrašo į žinučių eilę tam specifiniam kampanijos langui iki kol jis yra įrašomas į Redis duombazę. WordCount sistemos rezultatai bus skaitomi tiesiai iš Heron platformos naudojant Heron API ir saugomi į failą tokiu pačiu formatu kaip ir reklamų analizės sistema, naudojant absoliutų vėlinimą gauta iš Heron API.

Eksperimentai su visomis sistemomis bus vykdomi po 10 valandų, kadangi [VC18] straipsnio autoriai nustatė, kad jų balanasavimo algoritmas konverguoja po maždaug 11 valandų. Gauti rezul-

tatai iš failo bus skaitomi pasirašyta Python programa, kuri apdoroja duomenis į dvi grupes: visų sprendimų vėlinimą ir visų sprendimų vėlinimo 99–tą percentilę ir sugeneruoja linijinas diagramas, kiekvienai vėlinimo grupei pagal sprendimą.

Taigi iš viso magistro darbe planuojama atlikti 8 eksperimentus, kurių bendra trukmė – 80 valandų.

# Išvados

# Literatūra


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

[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.

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

[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.

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

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

[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.

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