VILNIAUS UNIVERSITETAS
INFORMATIKOS INSTITUTAS
PROGRAMŲ SISTEMŲ KATEDRA

Rodiklių duomenų apdorojimo sistemų generavimas iš duomenų modelių

Generation of Indicator Data Processing Systems from Data Models

Bakalauro baigiamasis darbas

Atliko: Vytautas Žilinas (parašas)

Darbo vadovas: lekt. Andrius Adamonis (parašas)

Recenzentas: assoc. prof., dr. Karolis Petrauskas (parašas)

Vilnius – 2019

Santrauka

Šį darbą sudaro teorinė ir eksperimentinė dalis. Teorinėje dalyje apibrėžiamas rodiklis, jo struktūra ir struktūros pokyčiai. Apibrėžiami kokie yra įmanomi pokyčiai ir kaip eksperimentinėje dalyje kuriamas sprendimas prie pokyčių prisitaikys. Specifikuojama duomenų struktūra ir duomenų struktūrų apjungimo ir skirtumo operacijas. Apibrėžiamas srautinis apdorojimas ir pasirenkama sistema, kuri bus naudojama eksperimentinėje dalyje. Remiantis gautais rezultatais nustatoma, kad šiam uždaviniui spręsti pasirenkama ”Heron” srautinio apdorojimo sistema. Remiantis pasirinkta srautinio apdorojimo sistema ir apibrėžta rodiklių duomenų struktūra, eksperimentui nusprendžiama generuoti srautinio apdorojimo modulius parašytus ”Python” programavimo kalba. Eksperimentinėje dalyje remiantis pasiūlyta architektūra realizuojama bandomoji sistemos versija. Atliekant skirtingų kiekių rodiklių duomenų ir rodiklių duomenų pokyčių simuliaciją stebėjimais analizuojamas šios sistemos tinkamumas apibrėžtam uždaviniui spręsti. Gauti tyrimų rezultatai lyginami, pateikiamos išvados. Taip įrodoma, kad toks sprendimas gali būti įgyvendinamas ir kad kodo generavimas ir srautinio apdorojimo sistema ”Heron” yra tinkamas sprendimas kintančių rodiklių uždaviniui spręsti.

Raktiniai žodžiai: srautinis apdorojimas, kodo generavimas, rodikli, rodiklio duomens pokyčiai

Summary

This work consists of a theoretical and an experimental part. Theoretical part defines the indicator, its structure and changes in indicator structure. It define what changes are possible and how the developed solution in the experimental part will adapt to the changes, specifies the data structure and data structure merging and difference operations, defines stream processing and selects the system to be used in the experimental part. Based on the results it is determined that ”Heron” stream processing system is chosen to solve this task. Based on the selected stream processing system and the defined data structure of indicator, it is decided to generate streaming modules written in ”Python” programing language. In the experimental part, a pilot version of the system is implemented based on the proposed architecture. By doing the simulation using varying amounts of indicators and indicator changed, the suitability of this system for a defined task is tested. The results of this research are compared and conclusions are given. This demonstrates that such a solution can be implemented and that the code generation and streaming system ”Heron” is the right solution to deal with the challenge of changing indicators.

Keywords: stream processing, code generation, indicator, indactor structure change

TURINYS

Įvadas

Šiame darbe yra nagrinėjamas rodiklių duomenų apdorojimas ir kuriamas sprendimas galintis prisitaikyti prie kintančių rodiklių duomenų struktūros. Rodiklių duomenimis vadiname duomenis, aprašančius kažkokių objektų savybes arba veiklos procesų rezultatus. Šiuos duomenis galima transformuoti, analizuoti ir grupuoti pagal pasirinktus rodiklius, pavyzdžiui: bazinė mėnesinė alga, mirusiųjų skaičius pagal mirties priežastis, krituliai per metus. Taip pat rodiklių struktūra gali keistis laikui bėgant: objektų atributų taksonomija (pvz. mirties priežasčių sąrašas, finansinių sąskaitų sąrašas) arba įrašo atributų sąrašai. Surenkamu rodiklių duomenų kiekis visada didėja, taip pat ir duomenų kiekis, kuriuos reikia apdoroti pagal rodiklius auga, todėl standartiniai sprendimai, pavyzdžiui reliacinės duomenų bazės netinka dėl ilgos apdorojimo trukmės. Rodiklių duomenų bazės pasižymi tuo, kad duomenys į jas patenka iš daug skirtingų tiekėjų ir patekimo laikas tarp tiekėjų nėra sinchronizuojamas, o suagreguotą informaciją vartotojai gali užklausti bet kurio metu. Todėl šiame darbe bus nagrinėjamas srautinis duomenų apdorojimas, kuris patenkančius duomenis apdoros realiu laiku, ir saugos jau apdorotus.

Realaus laiko duomenų apdorojimas (angl. Real-time data processing) yra jau senai nagrinėjamas, kaip būdas apdoroti didelių kiekių duomenis (angl. Big data). Vienas iš realaus laiko apdorojimo sprendimų yra srautinis duomenų apdorojimas [KAE+13; LVP06]. Srautinis duomenų apdorojimas (angl. stream processing) – programavimo paradigma, kuri yra ekvivalenti duomenų srauto programavimo (angl. dataflow programming) paradigmai [Bea15]. Duomenų tėkmės programavimo paradigmos idėja yra, kad programa susidaro iš skirtingu modulių, kurie nepriklauso vienas nuo kito, ir tai leidžia sukonstruoti paraleliai skaičiuojančias programas. Vienas iš pirmųjų duomenų tėkmės programavimo kompiliatorių yra BLODI - blokų diagramų kompiliatorius (angl. BLOck DIagram compiler), su kuriuo buvo kompiliuojamos BLODI programavimo kalba parašytos programos [KLV61]. Šia kalba parašytos programos atitinka inžinerinę elektros grandinės schemą, kur duomenys keliauja per komponentus kaip ir elektros grandinėje. Pasinaudojant šiai programavimo paradigmai sukurtai sistemai bus sukurtas sprendimas, kuris generuos modelius, kurie galės apdoroti rodiklių duomenis ir talpins jau apdorotus duomenis kitoje talpykloje [SÇZ05a].

Kadangi rodiklių yra daug skirtingų ir jie laikui bėgant gali kisti reikia, kad sprendimas kuris juos apdoros galės prisitaikyti prie poreikiu. Yra keli būdai kaip tai galima išspręsti:

- Rankinio atnaujinimo sprendimas. Sukuriamas sprendimas pagal esamus reikalavimus ir išskiriamas žmogus, kuris pagal naujus poreikius gali sukurti naujas arba pakeisti esamas

apdorojimo sistemas.

- Universalus sprendimas. Sukuriamas srautinio duomenų apdorojimo sprendimas, kuris apdoroja visus duomenis pagal visus įmanomus rodiklius.
- Kodo generavimo sprendimas. Sukuriamas sprendimas, kuris generuoja srautinio duomenų apdorojimo sistemas pagal iš anksto aprašytą struktūrą.

Šie sprendimai turi būti pritaikyti pagal sprendžiama problemą. Jei nėra numatomas kitimas pagal ką turi būti apdorojami duomenis, tai galima pasirinkti ir rankinio apdorojimo sprendimą, kadangi nėra didelės tikimybės, kad teks keisti sprendimą. Toks sprendimas tiktų apdorojant išmaniųjų skaitiklių duomenis [Nev17]. Universalus sprendimas taip pat gali būti tinkamas jei įeinantis duomenis yra specifiški ir yra poreikis juos visus apdoroti. Toks sprendimas gali būti aktualus apdorojant duomenis iš sensorių, kurie matuoja namų būsena (temperatūra, drėgmę ir t.t.) ir bet koks naujas sensorius taip pat turi būti prijungtas ir apdorotas [Yan17]. Šiame darbe buvo pasirinkta naudoti kodo generavimą siekiant sukurti sprendimą tinkanti bendram atvejui, kai duomenis yra nekonkretus ir ne visi reikalingi ir rodikliai kinta dažnai siekiant išgauti kuo daugiau informacijos iš įeinančių duomenų.

Kodo generavimas tai rašymas programinės įrangos, kuri rašys reikiamą programinę įrangą problemai spręsti. Tai daroma tokiais atvejais, kai sprendžiama problema reikalauja daug rankinio darbo, kurį įmanoma automatizuoti. Kuo didesnio sprendimo reikalauja uždavinys tuo patraukliau tampa naudoti kodo generavimą sprendimo kūrimui. Kodo generavimas suteikia tokius privalumus:

- Architektūrinį nuoseklumą:
  - Verčia programuotojus labiau mąstyti apie architektūrą.
  - Jei sunku ”priversti” generatorių generuoti reikiamą kodą, problema gali būti architektūroje.
  - Geros dokumentacijos buvimas sumažina problemą, kai nariai palieką projektą.
- Abstrakciją:
  - Programuotojai galės kurti naujus šablonus, kurie leis esamą funkcionalumą pritaikyti kitomis kalbomis, sprendimais daug paprasčiau negu rankomis parašytą kodą.
  - Verslo analitikai gali apžvelgti ir patvirtinti sprendimo abstrakciją.
  - Abstrakcija padės paprasčiau paruošti dokumentacija, testavimo atvejus, produkto palaikymo medžiagą ir t.t.
- Aukštą komandos moralę - rašomas kodas bus nuoseklus ir kokybiškas todėl kels komandos pasitikėjimą.
- Tinkamas sprendimas Judriajam programavimui, kadangi kodo generavimo kuriami spren-

dimai yra lankstesni, tai leidžia ateityje juos lengviau keisti ir atnaujinti.
Kodo generavimas tampa tikrai naudingas tada, kai jis naudojamas didelių kiekių rankiniam kodavimui pakeisti [Her03].

Darbe bus nagrinėjamas sprendimas generuojantis srautinio apdorojimo modulius pagal ateinančius rodiklių duomenis. Kadangi rodiklių duomenis gali būti skirtingi ir gali keistis laikui bėgant reikalingas sprendimas galintis apdoroti didelį kiekį kintančių duomenų.

Tikslas: Sukurti rodiklių duomenų srautinio apdorojimo sistemos, pritaikomos prie duomenų struktūrų naudojant kodo generavimą, architektūrą.

Uždaviniai:

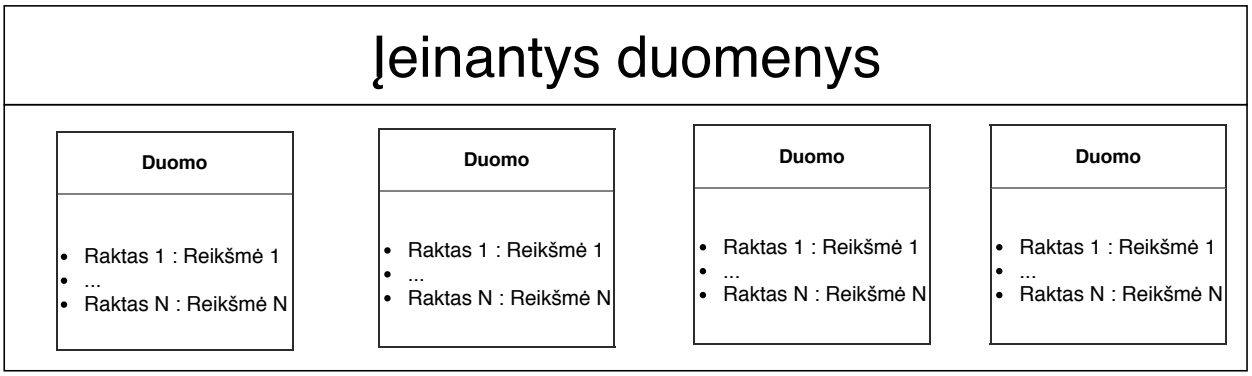
1. Apibrėžti rodiklių duomenų modelį ir galimus rodiklių duomenų struktūros pokyčius.
2. Sukurti architektūra, kuri pasižymėtų reikalingomis savybėmis spręsti rodiklių apdorojimo uždavinį.
3. Atlikus šaltinių analizę pasirinkti srautinio duomenų apdorojimo sprendimą, joje sukurti sudarytos architektūros sprendimą ir atlikti bandymus.

1. Rodiklių duomenys

Rodiklių duomenys - tai realaus pasaulio duomenys, kurie rodo tam tikrų objektų savybes, kurias galime grupuoti pagal kitas tų objektų savybes. Rodyklių duomenys pasižymi tuo, jog nėra aktualus pradiniai neapdoroti duomenis. Taip pat rodikliai yra nepastovus, jų struktūra gali keistis - atsirasti naujų savybių, kurias reikia surinkti arba yra pašalinamos jau esamos savybės. Rodiklių duomenys yra renkami ir analizuojami labai skirtingiems poreikiams - visur kur reikalinga statistika ir apdorojami didelį kiekiai duomenų yra naudojami rodikliai: tai gali būti įmonės vedamą "Microsoft Excel" skaičiuoklė, kurioje užrašyta kiek buvo mokėta už elektrą kiekvienais metais per paskutinius 10 metų arba juodosios skylės nuotrauka padaryta koreliuojant didelio duomenų kiekį iš 8 skirtingų teleskopu ir gauti rodikliai rodo taškus ant paveikslėlio, kurio dydis yra 10^{12} kartų mažesnis nei pradiniai duomenys [AAA+19].

Tam, kad sukurtume sprendimą, kuris tiktų rodiklių apdorojimui reikia bendro rodiklių duomenų modelio, kurio pagalba galėtume apibrėžti visus įmanomus rodiklius. Pagal šį aprašytą modelį bus generuojamos srautinio apdorojimo sistemos, kurios apdoros įeinančius duomenis.

1.1. Įeinančių duomenų struktūra



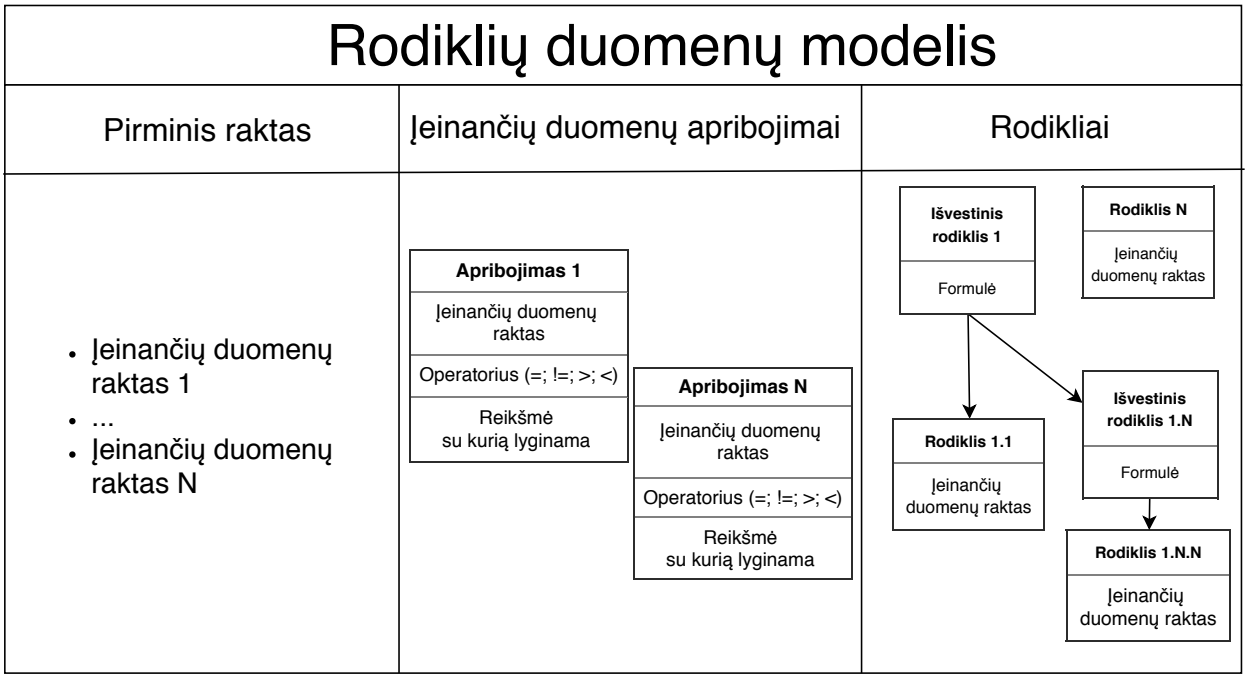
1 pav. Įeinantys duomenys

Rodiklių duomenis mes gauname iš neapdorotų duomenų. Kadangi rodiklių gali būti daug skirtingų, todėl ir įeinantis duomenis neturi būti apriboti. Visi duomenys, kurie bus apdorojami bus sudaryti iš raktų ir reikšmių žodyno (1 pav.), kur raktas gali būti tik tekstinio tipo, o reikšmė - tekstas arba skaičius. Duomenų raktai ir raktų kiekis gali nesutapti, vieni žodynai gali turėti daugiau raktų-reikšmių elementų, kitų raktai gali būti visiškai nesusiję su apdorojamais rodikliais. Kai yra neaktualus duomenys jie turi būti ignoruojami.

Kadangi neapdoroti duomenys gali būti gaunami iš skirtingų šaltinių, skirtingais laikais, todėl

kuriamas sprendimas turi galėti priimti duomenis asinchroniškai ir iš skirtingų šaltinių, o sprendimas tikru laiku turi sugebėti apdoroti reikiamus duomenis, o nereikalingus atmesti. Realus pavyzdys būtų protingų namų sistema, kur skirtingi sensoriai siunčia duomenis skirtingais intervalais ir ne visada yra reikalingi visų sensorių parodymai.

1.2. Rodiklių duomenų modelis



2 pav. Rodiklių duomenų modelis

Norit apdoroti daug skirtingų rodiklių turime apsibrėžti bendrą rodiklių duomenų modelį (2 pav.), pagal kurį bus generuojamos srautinio duomenų apdorojimo sistemos. Siūlomas apibrėžimas susidaro iš trijų dalių:

- Rodiklių duomenų pirminis raktas sudarytas iš įeinančių duomenų raktų rinkinio.
- Įeinančių duomenų reikšmių srities apribojimai.
- Įeinančių duomenų raktų rinkinys, kuris aprašo renkamus rodiklius ir išvestinių rodiklių kombinavimo apibrėžimai.

1.2.1. Pirminis raktas

Pirma rodiklio apdorojimo apibrėžimo dalis nusako raktų rinkinį reikšmių pagal kurias grupuojami apdoroti duomenis. Pavyzdžiui:

- Statistikos srityje: norint surinkti duomenis aprašančius **skirtingų savivaldybių** nekilnojamo

turto kainas pagal **metus**, raktų rinkinys atrodys taip:

[*"Savivaldybė"; "Metai"*]

- Sensoriniai rodikliai: norint surinkti vienos dienos duomenis aprašančius **skirtingų sensorių** parodymus **kiekvieną valandą**, **kiekvienam ofiso kabinetui**, raktu rinkinys atrodys taip:

[*"Sensoriaus tipas"; "Valanda"; Kabineto Numeris*]

- Buhalterijoje: norint surinkti duomenis parodančių kiek ir **kokie skyriai** patyrė išlaidų kiekvieną šių **metų mėnesį**, raktų rinkinys atrodys taip:

[*"Skyriaus pavadinimas"; "Mėnesis"*]

Raktų kiekis neturi būti ribojamas, kadangi realiame pasaulyje yra neribotas kiekis duomenų pagal kuriuos galima grupuoti. Visų raktų reikšmių kiekių sandauga - apdorotų rodiklių duomenų pirminių raktų aibė. Nuo šios aibės dydžio priklausys ir apdorotų duomenų kiekis ir naudojamos atminties kiekis, nes visos apdoroti duomenis turi būti saugomi atmintyje.

1.2.2. Įeinančių duomenų apribojimai

Antra rodiklio duomenų modelio dalis susidaro iš masyvo apribojimų, kurie aprašo kokie duomenys turi būti apdorojami, o kokie ne. Apribojimas susidaro iš:

- Įeinančių duomenų rakto.
- Operatorius, kuris gali būti (==, !=, >, <)
- Reikšmė su kuria lyginama - riba.

Pavyzdžiui:

- Statistikos srityje: norint surinkti žmonių, **vyresnių nei 60 metų** atlyginimus pagal kalendorinius metus, apribojimų sąrašas atrodys taip:

[{*"Amžius" ; > ; 59*}]

- Sensoriniai rodikliai: norint surinkti **tik savaitgalio** sensorių rodiklius, apribojimų sąrašas

atrodys taip:

$$[\{\textit{"Savaitės Diena"} ; > ; 5\}; \{\textit{"Savaitės Diena"} ; < ; 8\}]$$

- Buhalterijoje: norint apdoroti tam tikrus duomenis **tik "Marketingas" skyriaus** apribojimų sąrašas atrodys taip:

$$[\{\textit{"Skyriaus pavadinimas"} ; == ; \textit{"Marketingas"}\}]$$

Apribojimų kiekis kaip ir raktų, taip pat neturi būti ribojamas. Šie apribojimai leis naudoti tuos pačius duomenis skirtingiems rodikliams ir rodiklių rinkiniams apdoroti. Įeinantis duomenys bus ribojami pagal reikšmes, kurios turi priklausyti apribojimų aibių sankirtai. Jei pateikiame du apribojimus, tai įeinančio duomens ribojamos reikšmės turi patekti į abiejų apribojimų aibes.

1.2.3. Rodikliai

Rodiklių duomenų modelio dalis, kuri aprašo, kokie rodikliai renkami iš pradinių duomenų ir išvestinių rodiklių formules. Ši dalis susidaro iš masyvo medžių. duomenis medžio šakomis eina iš apačios į viršų. Kiekviena medžio viršūnė saugo apdorotą rodiklį ir talpiną jį duomenų bazėje. Medžio viršūnės gali aprašyti du skirtingus dalykus: rodiklį ir išvestinį rodiklį.

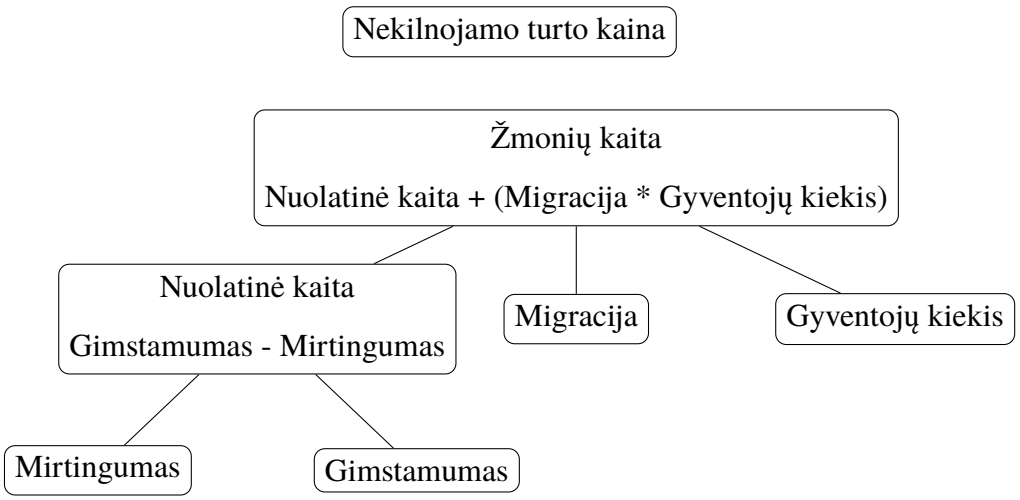
Rodiklis gauna ir apdoroja įeinančius duomenis. Rodiklio apibrėžimas susidaro iš įeinančių duomenų elemento rakto, reikšmės kuri bus naudojama apdorojimui. Rodikliuose ši reikšmė visada yra skaitinė, ji gali nurodyti skaičiuojamas savybes arba tai gali būti dvejetainė reikšmė.

Išvestinis rodiklis gauna duomenis iš prieš tai ėjusio rodiklio. Išvestinio rodiklio gaunami duomenys - tai prieš tai ėjusios viršūnės bet kokio rodiklio paskutinė reikšmė. Kaip duomenis bus apdoroti išvestinio rodiklio viršūnėje aprašo formulė, kurioje pateikiama veiksmai atliekami su prieš tai ėjusių viršūnių paskutinėmis reikšmėmis ir atliekami susiejant duomenis unikaliu identifikatoriumi, kuris sukuriamas kiekvienam įeinančiam duomeniui patenkant į apdorojimo sprendimą. Norint rodiklį gauti ne tik naudojant pagrindinius matematinius veiksmus kaip sudėtis arba atimtis, turi būti įmanoma naudoti, bet kokį iš anksto aprašytą metodą duomenims apdoroti.

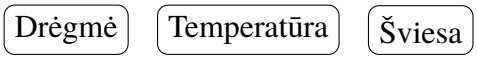
Pavyzdžiui:

- Statistikos srityje: norint surinkti rodiklių duomenis nekilnojamo turto kainų ir žmonių kaitos šalyje ($\textit{Žmonių kaita} = (\textit{Gimstamumas} - \textit{Mirtingumas}) + (\textit{Migracija} * \textit{Gyventojų kiekis})$,

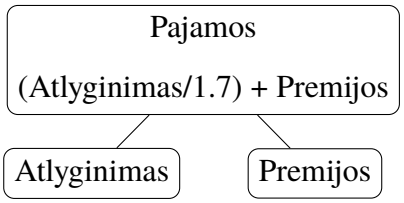
renkamų rodiklių medžiai apsirašytų taip:



- Sensoriniai rodikliai: norint surinkti drėgmės, temperatūros ir šviesos sensorių rodiklius, rodiklių medžiai atrodys taip:



- Buhalterijoje: norint suskaičiuoti kiek įmonės darbuotojai gauna pajamų (*Pajamos* $=$ $(\mathit{Atlyginimas}/1.7) + \mathit{Premijos}$) rodiklių medis bus toks:



1.3. Rezultatų struktūra

Rezultatai			
Pirminis raktas	Rodiklis 1	...	Rodiklis N
[Duomo[Raktas1], ..., Duomo[RaktasN]]	Apdoroti duomenys 1	...	Apdoroti duomenys N
...	...	⋱	...
[Duomo[Raktas1], ..., Duomo[RaktasN]]	Apdoroti duomenys N	...	Apdoroti duomenys N

3 pav. Rezultatų struktūra

Pagal pateikta rodiklio modelį sukurtas sprendimas gražina apdorotų duomenų rinkinį, kurį galima pateikti lentelės pavidalu (3 pav.). Stulpelių kiekis susidaro iš vieno pirminio rakto ir rodiklių kiekio, o eilutės kiekis iš įeinančių duomenų raktų reikšmių sandaugos. Tokiu būdu saugomus duomenis galima patogiai analizuoti ir esant poreikiui toliau apdoroti.

1.4. Rodiklių duomenų struktūros kitimas

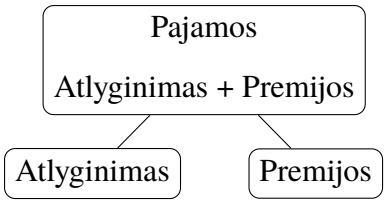
Rodikliniai duomenys dažnai kinta. Buhalterijos srityje tai gali būti dėl naujo įstatymo, kuris pakeičia kažkurių rodiklių skaičiavimo formulę. Tai gali būti naujos rūšies išmaniųjų namų sensorius, kuriuo duomenys norima irgi rinkti, kad galėtume matyti pilną vaizdą. Statistikos srityje gali atsirasti poreikis išskaidyti tam tikrus rodiklius į mažesnes dalis, pavyzdžiui rinkti ne migracijos rodiklį kaip vieną, o atskirai emigraciją ir imigraciją.

Pagal mūsų aprašyta rodiklių duomenų modelį, mes taip pat turime apsibrėžti kas gali keistis:

- Pirminis raktas negali keistis, nes tokiu atveju prarandama prasmė. Tarkim buvo renkami rodikliai pagal metinius duomenis, o pagal reikalavimus reikia pradėti apdoroti kas mėnesį. Kaip naujas rodiklis tai yra tinkamas uždavinys, tačiau praplėtus esamą, būtų prarasta prasmė lyginti duomenis ir todėl neturėtų būti dalis to pačio rodiklio.
- Apribojimai gali keistis, tačiau šie pokyčiai turi turėti prasmę ir naudojami tik tam, kad būtų išlaikytos tos pačios apdorojamu įeinančių duomenų aibės. Tarkime mums reikėjo apdoroti duomenis iš visų skyrių išskyrus administraciją ir tokiam uždaviniui buvo sukurtas toks apribojimas $[\{\textit{"Skyriaus pavadinimas"; != ;"Administracija"}\})$. Tačiau po laiko dalis administracijos skyriaus atsiskyrė ir susikūrė naujas HR skyrius. Norėdami, kad būtų išlaikyta prasmė, nauji apribojimai turi atrodyti taip: $[\{\textit{"Skyriaus pavadinimas";!=;"Administracija"}\},\{\textit{"Skyriaus pavadinimas";!=;"HR"}\})$
- Rodikliai ir išvestiniai rodikliai gali keistis, kai tam atsiranda poreikis ir turėtų nebūti apribojami. Rodiklių medžio viršūnės gali būti pridedamos ir pašalinamos, pridedant naujus rodiklius jų apdorojimas prasidės nuo to momento, kai bus pridėti, tačiau jau esamų rodiklių apdoroti duomenis neturi būti pakeisti ir apdorojimas esamų rodiklių turi būti tęsiamas kur sustojo. Tarkime jog skaičiavome pajamas naudodami formulę:

$$Pajamos = Atlyginimas + Premijos$$

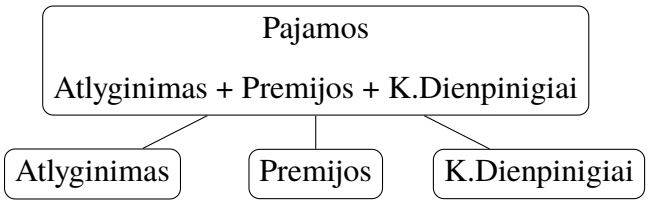
Ir pagal šią formulę buvo sukurtas toks rodiklių medis:



Po tam tikro laiko atsirado poreikis pajamas skaičiuoti pagal formulę:

$$Pajamos = Atlyginimas + Premijos + Komandiruočių\ dienpinigiai$$

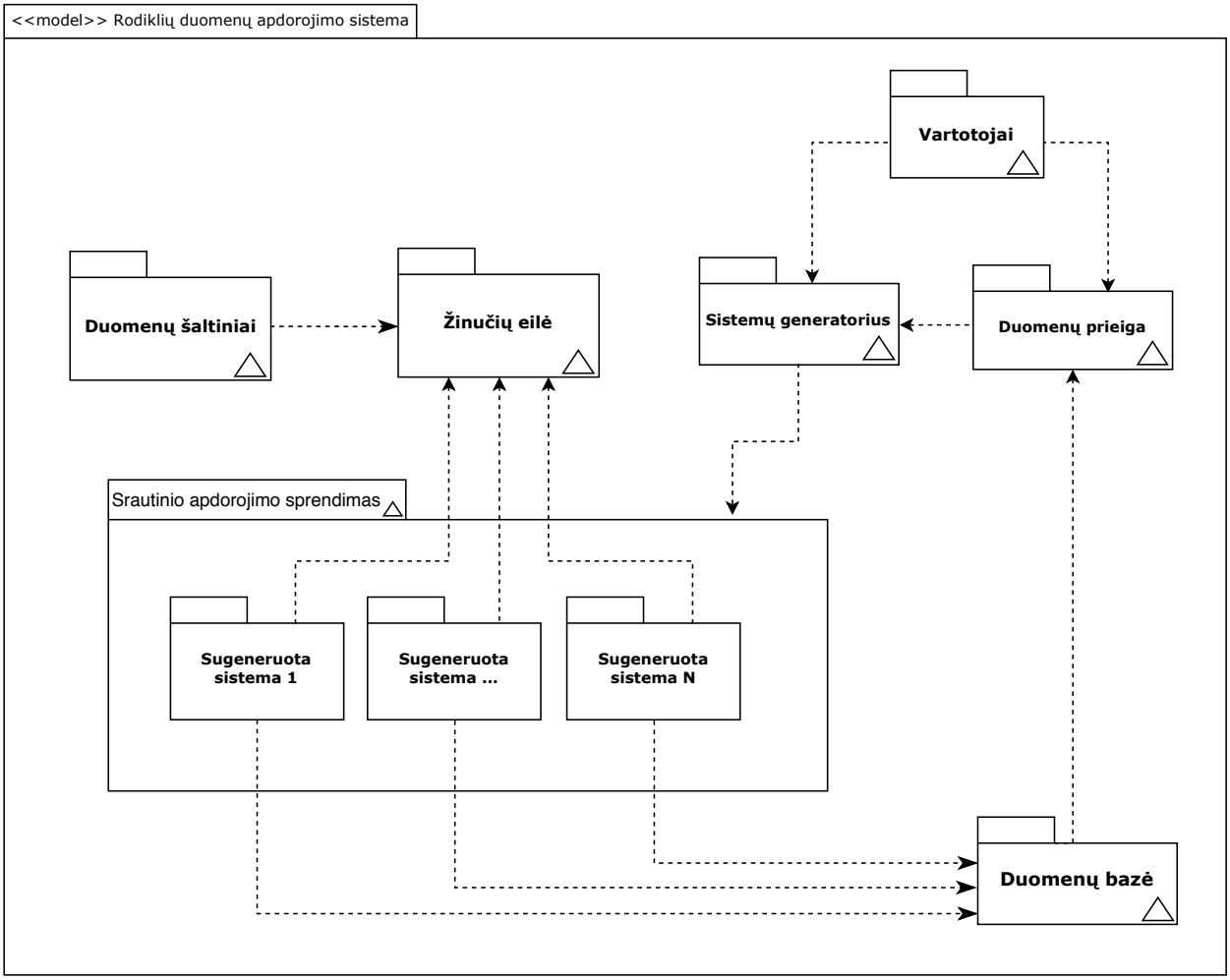
Pagal naują formulę rodiklių medis turės atrodyti taip:



Šie pokyčiai turi prasmę, kai daromi tuo metu kai jų reikia. Jie netinka kai apdorojami duomenys yra leidžiami iš naujo (Komentaras: nežinau ar šitą reikią čia pridėti, kaip ir galima padaryti turėdami rodiklių versijas ir turint duomenų įeinančių timestamp, bet čia dėl laiko nepadaryta)

2. Rodiklių duomenų apdorojimo architektūra

Kuriama architektūra susidarys iš skirtingų komponentų, kuriu visumą turi gebėti generuoti rodiklių duomenų sistemas



4 pav. Architektūra

Kuriama architektūra turi sugebėti prisitaikyti prie tokių rodiklių duomenų struktūros pokyčių ir pakitus rodikliams tęsti jau vykdomus apdorojimus.

3. Srautinio duomenų apdorojimo sprendimų analizė

3.1. Srautinis duomenų apdorojimas

Siekiant apžvelgti modernius srautinio duomenų apdorojimo sprendimus turime apsibrėžti jų savybes. 2005 metais Michael Stonebraker apibrėžė 8 taisykles realaus laiko (angl. real-time) srautinio duomenų apdorojimo architektūrai [SÇZ05b]:

1 taisyklė: Duomenys turi judėti. Žemo uždelstumo užtikrinimui sistema turi apdoroti duomenis nenaudojant duomenų saugojimo operacijų. Taip pat sistema turi ne pati užklausti duomenų, o gauti juos iš kito šaltinio asinchroniškai.

2 taisyklė: Duomenų transformacijos turi būti vykdomas SQL pobūdžio užklausomis. Žemo abstrakcijos lygmens srautinio apdorojimo sistemos reikalauja ilgesnio programavimo laiko ir brangesnio palaikymo. Tuo tarpu aukšto abstrakcijos lygmens sistema naudojanti SQL užklausas, kurias žino dauguma programuotojų ir yra naudojamos daugelyje skirtingų sistemų, leidžia efektyviau kurti srautinio apdorojimo sprendimus.

3 taisyklė: Architektūra turi susidoroti su duomenų netobulumais. Architektūra turi palaikyti galimybę nutraukti individualius skaičiavimus tam, kad neatsirastų blokuojančių operacijų, kurios sustabdo vieno modulio veikimą ir tuo pačiu visos architektūros veikimą. Taip pat ši architektūra turi sugebėti susidoroti su vėluojančiomis žinutėmis, pratęsiant laiko tarpą, per kurį ta žinutė turi ateiti.

4 taisyklė: Architektūra turi būti deterministinė. Kiekvieną kartą apdorojant tuos pačius duomenis rezultatai turi būti gaunami tokie patys.

5 taisyklė: Architektūra turi gebėti apdoroti išsaugotus duomenis ir realiu laiku gaunamus duomenis. Sistema parašyta su tokia architektūra turi galėti apdoroti jau esančius duomenis taip pat kaip ir naujai ateinančius. Toks reikalavimas buvo aprašytas, nes atsirado poreikis nepastebimai perjungti apdorojimą iš istorinių duomenų į realiu laiku ateinančius duomenis automatiškai.

6 taisyklė: Architektūra turi užtikrinti duomenų saugumą ir apdorojimo prieinamumą. Kadangi sistema turi apdoroti didelius kiekius duomenų, architektūra klaidos atveju, turi sugebėti persijungti į atsarginę sistemą ir tęsti darbą toliau. Taip pat tokios klaidos atveju atsarginė sistema turi būti apdorojusi visus duomenis ir sugebėti iš karto priimti naujus duomenis, o ne apdoroti duomenis iš pradžių.

7 taisyklė: Architektūra turi užtikrinti sugebėjimą paskirstyti sistemos darbus automatiškai.

Srautinio apdorojimo sistemos turi palaikyti kelių procesoriaus gijų operacijas. Taip pat sistema turi galėti veikti ant kelių kompiuterių vienu metu ir prireikus paskirstyti resursus pagal galimybes.

8 taisyklė: Architektūra turi apdoroti ir atsakyti akimirksniu. Anksčiau minėtos taisyklės nėra svarbios, jeigu sistema nesugeba greitai susidoroti su dideliu kiekiu naujų duomenų. Todėl turi būti naudojamas ne tik teisingas ir greitas srautinio apdorojimo sprendimas, bet ir gerai optimizuota sistema.

Šie reikalavimai yra sukurti tik teoriškai ir srautinio apdorojimo sprendimai neprivalo jų įgyvendinti [SÇZ05b]. Todėl norint sužinoti tam tikro srautinio duomenų apdorojimo sprendimo tinkamumą uždaviniui reikia išanalizuoti jų savybes.

3.2. Srautinio duomenų apdorojimo sistemos

Šiame skyriuje lyginami keturi atviro kodo srautinio apdorojimo sprendimai ”Apache Storm”, ”Apache Spark” ir ”Apache Flink” ir ”Heron” pagal:

- Pristatymo semantika (angl. delivery semantics) - apibrėžia pagal kokį modelį bus pristatyti duomenys. Egzistuoja trys semantikos [Kah18]:
  - Bent vieną kartą (angl. at-least-once) užtikrina, kad duomenys bus apdoroti bent kartą, bet gali atsirasti dublikatų.
  - Ne daugiau vieno karto (angl. at-most-once) užtikrina, kad duomenys bus apdoroti daugiausiai tik vieną kartą, bet gali atsirasti praradimų.
  - Tiksliai vieną kartą (angl. exactly-once) užtikrina, kad duomenys bus apdoroti tik vieną kartą ir klaidos bus suvaldytos.
- Uždelstumas (angl. latency) - apibrėžia laikų sumą - kiek laiko trūko viena operacija ir kiek laiko ši operacija turėjo laukti eilėje kol bus pabaigtos kitos operacijos [KRK+18].
- Pralaidumas (angl. throughput) - apibrėžia kiek pavyks įvykdyti operacijų per tam tikrą laiko tarpą.
- Abstrakcijos lygmuo (angl. abstraction) - apibrėžia kokio lygmens programavimo sąsają pateikia sprendimas.

3.3. Pristatymo semantika

”Apache Spark” ir ”Apache Flink” sprendimų pristatymo semantika yra tiksliai vieną kartą (angl. exactly-once), tai reiškia, kad visi duomenys bus apdoroti tik vieną kartą. Tačiau tam, kad

užtikrinti šią semantiką sprendimas sunaudoja daug resursų, kadangi reikia užtikrinti, kad operacija bus vykdoma būtent vieną kartą kiekviename srautinio apdorojimo žingsnyje: duomenų gavime, kuris stipriai priklauso nuo duomenų šaltinio, duomenų transformacijos, kurią turi įvykdyti srautinio apdorojimo sprendimas, ir duomenų saugojime, tai turi būti užtikrinta sprendimo ir naudojamos saugyklos [ZHA17].

”Apache Storm” pristatymo semantika yra bent vieną kartą (angl. at-least-once), tai reiškia, kad į šį sprendimą siunčiami duomenys bus visada apdoroti, tačiau kartais gali būti apdoroti kelis kartus [DAM16]. Jei uždavinys nereikalauja tiksliai vieno karto apdorojimo, tai geriau rinktis bent vieną kartą ar ne daugiau vieno karto semantikas, kadangi jos neturi papildomų apsaugų, kurios reikalingos tiksliai vieno karto apdorojimui, ir todėl veikia greičiau [ZHA17].

”Heron” sprendimų pristatymo semantika gali būti keičiama, programuotojas kurdamas srautinio apdorojimo sistemą šiam sprendimui aprašo kokio tipo pristatymo semantikos jo sistema reikalauja [Ram17]. Todėl šis sprendimas yra tinkamiausias jei kuriamos sistemos turi būti skirtingų poreikių ir pobūdžių.

3.4. Uždelstumas

Uždelstumas - laiko trukmė, kuri parodo kaip greitai sprendimas įvykdo vieną operaciją, nuo jos patekimo į eilę iki šios operacijos apdorojimo pabaigos. Pagal [LLD16] aprašytus Martin Andreoni Lopez ”Apache Storm”, ”Apache Spark” ir ”Apache Flink” bandymus galima matyti, kad būtent ”Apache Storm” turi mažiausią uždelstumą. Kadangi parinkus tinkamą paralelizmo parametrą šis sprendimas su užduotimi susidorojo net iki 15 kartų greičiau. Antroje vietoje liko ”Apache Flink”, o po jos ”Apache Spark”.

Tačiau ”Heron” sprendimas yra sukurtas siekiant pagerinti ”Apache Storm” sprendimo greitaveika ir suteikti lengvą būdą pereiti nuo ”Apache Storm” API prie ”Heron”, todėl jo uždelstumas yra dar mažesnis nei ”Apache Storm” [KBF+15].

3.5. Pralaidumas

Pralaidumas apibrėžia kokį kiekį procesų sprendimas gali įvykdyti per tam tikrą laiko tarpą. 2016 metais Sanket Chintapalli [CDE+16] išmatavo ”Apache Storm”, ”Apache Spark” ir ”Apache Flink” sprendimų pralaidumą ir uždelstumą bei palygino rezultatus. Kaip ir anksčiau manyta, ”Apache Spark” turėjo aukščiausia pralaidumą iš visų, kadangi jis vienintelis duomenis apdoroja mikro-paketais. Antroje vietoje liko ”Apache Flink”, kuris yra subalansuotas pralaidumo atžvil-

giu ir paskutinis liko ”Apache Storm”, kuris turi žemą uždelstumą, todėl nukenčia pralaidumas. ”Heron” tuo tarpu turi aukštesnį pralaidumą ir žemesnį uždelstumą nei ”Apache Storm” [Ram15].

3.6. Abstrakcijos lygmuo

”Apache Storm” parašytos sistemos yra žemo abstrakcijos lygmens, tai reiškia, kad turi būti aprašyti visi srautinio apdorojimo moduliai: setSpouts(..), kuriame nustatoma duomenų įeiga ir koks bus paralelizmo lygis, setBolt(..), kuriame nustatomi apdorojimo moduliai, kokius duomenis gaus iš prieš tai buvusio modulio ir paralelizmo lygis. Kiekvieno modulio execute() metodas aprašo, kaip šis modulis turi apdoroti duomenis [tut18]. Šio sprendimo programų kūrimo laikas užtruks ilgiau negu kitiems sprendimams su aukštu abstrakcijos lygmeniu, tačiau žemas abstrakcijos leidžia rašyti daug greičiau veikiančias sistemas, kadangi programuotojas turi pilną kontrolę.

”Apache Spark” parašytos programos yra aukšto abstrakcijos lygmens. Sistemos aprašomos funkciškai, todėl kodo rašymas trunka daug trumpiau ir jį yra daug lengviau skaityti. Tačiau prarandama galimybė optimizuoti ir paralelizmo klausimas paliekamas sprendimui. Kadangi ”Apache Spark” yra ne pilnai srautinis, o mikro-paketinis (angl. micro-batching) sprendimas, todėl vartotojas turi apsirašyti kokio dydžio paketais bus renkami duomenys [SS15].

”Apache Flink” parašytos programos yra aukšto abstrakcijos lygmens. ”Apache Flink” sprendimas pats užsiima resursų distribucija, todėl programuotojui lieka tik parašyti veikianti kodą, o sistema pati susitvarkys su paralelizmu [Doc18]. Tačiau tai reiškia, kad su šiuo sprendimu parašytos programos nepavyks optimizuoti taip pat gerai kaip žemo abstrakcijos lygmens sprendimus.

”Heron” sprendimas turi skirtingus API, kurie naudoja skirtingus abstrakcijos lygius, kuriuos galima rinktis pagal tinkamumą sprendžiamai problemai. Darbo rašymo metu ”Heron” turi 4 skirtingus API:

- ”Heron Streamlet API” - aukšto abstrakcijos lygmens API, rašomas su ”Java” programavimo kalba. Panaši sintaksė į ”Apache Flink” rašomų sprendimų.
- ”Heron ECO API” - eksperimentinis aukšto abstrakcijos lygmens API, rašomas su ”Java” programavimo kalba. Skiriasi nuo ”Heron Streamlet API”, nes modulių apdorojimo eiliškumas apsirašo YAML formatų, kas leidžia keisti sukurtos srautinio duomenų apdorojimo programos struktūrą nekeičiant kodo.
- ”Heron Topology API for Java” - žemo abstrakcijos lygmens API, rašomas su ”Java” programavimo kalba. Rašomas identiškas kodas, kaip ir ”Apache Storm”, kadangi ”Heron” sprendimas buvo sukurtas siekiant pagerinti ”Apache Storm”.

- ”Heron Topology API for Python” - žemo abstrakcijos lygmens API, rašomas su ”Python” programavimo kalba. Su šiuo API kuriami srautinio apdorojimo sprendimai yra panašus į ”Apache Storm”, tik su ”Python” programavimo kalbos privalumais.

3.7. Apibendrinimas

Iš šių keturių sprendimų reikėjo pasirinkti vieną, kuris labiausiai tiks rodiklių duomenų apdorojimui. Pasirinktas sprendimas turi sugebėti greitai ir patikimai apdoroti duomenis.

1 lentelė. Srautinių duomenų apdorojimo sprendimų palyginimas

Charakteristika	”Apache Storm”	”Apache Spark”	”Apache Flink”	”Heron”
Pristatymas	Bent vieną kartą	Tiksliai vieną kartą	Tiksliai vieną kartą	Pasirenkamas
Uždelstumas	Žemas	Aukštas	Vidutinis	Žemas
Pralaidumas	Žemas	Aukštas	Vidutinis	Vidutinis
Abstrakcijos lygis	Žemas	Aukštas	Aukštas	Pasirenkamas

Pagal atlikta analizę 1 lentelėje ir apsibrėžtų reikalavimų šiam uždaviniui tinkamiausias srautinio apdorojimo sprendimas yra ”Heron”.

Čia parašyti apie abstrakcija

Čia parašyti apie Python

4. Pagal architektūrą sukurtas sprendimas

4.1. Srautinio apdorojimo sistemų kodo generavimo komponentas

.Net Core biblioteka

4.2. Apdorotų rodiklių duomenų bazė

Redis duomenų bazė

4.3. Papildoma programinė įranga

Kafka, Zookeeper Pants

5. Eksperimentas ir sukurto sprendimo savybės

5.1. Eksperimento tikslas

Nustatyti sukurto sprendimo ir architektūros savybes ir trukumus atliekant bandymus su rodiklių duomenimis:

1. Sukurti sistemą apdorojančia rodiklius iš duomenų su apribojimais, grupuojančia juos pagal pirminį raktą.
2. Esamai sistemai pridėti arba pašalinti apribojimą.
3. Esamai sistemai pridėti rodiklį.
4. Esamai sistemai pašalinti rodiklį.
5. Paduoti nereikalingus arba netinkamus duomenis į sistemą.
6. Paduoti didelį kiekį duomenų vienu metu.
7. Gauti apdorotus rodiklius.
8. Gauti apdorotus rodiklius didelio duomenų kiekio padavimo į sistemą metu.

5.2. Eksperimento vykdymo aplinka

Kompiuterio, kuriame buvo vykdomas eksperimentas specifikacijos:

- Procesorius - Intel Core i7-7700k 4,5GHz
- Sisteminis diskas - 512 GB SSD
- Operatyvioji atmintis - 16 GB
- Operacinė sistema - Ubuntu 18.04

5.3. Eksperimentui naudojama papildoma programinė įranga

Testavimo duomenų generatorius, Postman, naršyklė

5.4. Eksperimento eiga

5.5. Eksperimento išvados

5.6. Sprendimo savybės

Rezultatai

1. Apibrėžta rodiklių duomenų struktūra ir galimi duomenų struktūros pokyčiai.
2. Sukurtas architektūra tinkama spręsti rodiklių apdorojimo uždavinį.
3. Pasirinktam srautinio apdorojimo sprendimui, pagal sukurtą architektūra sukurtas sprendimas
4. Atlikti eksperimentai su sukurtu sprendimu ir apžvelgtos jo savybės.

Išvados

- Išvada 1
- Išvada 2

Literatūra

[AAA+19] Kazunori Akiyama, Antxon Alberdi, Walter Alef, Keiichi Asada ir k.t. First m87 event horizon telescope results. iii. data processing and calibration. *The Astrophysical Journal Letters*, 875(1):L3, 2019.

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

[CDE+16] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar ir k.t. Benchmarking streaming computation engines: storm, flink and spark streaming. *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, p. 1789–1792. IEEE, 2016.

[DAM16] PRITHIVIRAJ DAMODARAN. "exactly-once" with a kafka-storm integration. `http://bytecontinnum.com/2016/06/exactly-kafka-storm-integration/`, 2016.

[Doc18] Flink Documentation. Flink datastream api programming guide. `https://ci.apache.org/projects/flink/flink-docs-release-1.5/dev/datastream_api.html`, 2018.

[Her03] Jack Herrington. *Code generation in action*. Manning Publications Co., 2003.

[Yan17] Shusen Yang. Iot stream processing and analytics in the fog. *IEEE Communications Magazine*, 55(8):21–27, 2017.

[KAE+13] S. Kaisler, F. Armour, J. A. Espinosa ir W. Money. Big data: issues and challenges moving forward. *2013 46th Hawaii International Conference on System Sciences*, p. 995–1004, 2013-01. DOI: 10.1109/HICSS.2013.645.

[Kah18] Ensar Basri Kahveci. Processing guarantees in hazelcast jet. `https://blog.hazelcast.com/processing-guarantees-hazelcast-jet/`, 2018.

[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.

[KLV61] John L Kelly Jr, Carol Lochbaum ir Victor A Vyssotsky. A block diagram compiler. *Bell System Technical Journal*, 40(3):669–676, 1961.

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

[LLD16] Martin Andreoni Lopez, Antonio Gonzalez Pastana Lobato ir Otto Carlos Muniz Bandeira Duarte. A performance comparison of open-source stream processing platforms. *2016 IEEE Global Communications Conference (GLOBECOM)*:1–6, 2016.

[LVP06] Ying Liu, Nithya Vijayakumar ir Beth Plale. Stream processing in data-driven computational science. *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, GRID '06, p. 160–167, Washington, DC, USA. IEEE Computer Society, 2006. ISBN: 1-4244-0343-X. DOI: 10.1109/ICGRID.2006.311011. URL: https://doi.org/10.1109/ICGRID.2006.311011.

[Nev17] Mantas Neviera. Išmaniųjų apskaitų didelių duomenų kiekių apdorojimas modernioje duomenų apdorojimo architektūroje, 2017.

[Ram15] Karthik Ramasamy. Flying faster with twitter heron. https://blog.twitter.com/engineering/en_us/a/2015/flying-faster-with-twitter-heron.html, 2015-06.

[Ram17] Karthik Ramasamy. Why heron? part 2. https://streaml.io/blog/why-heron-part-2, 2017-09.

[SÇZ05a] Michael Stonebraker, Uğur Çetintemel ir Stan Zdonik. The 8 requirements of real-time stream processing. *SIGMOD Rec.*, 34(4):42–47, 2005-12. ISSN: 0163-5808. DOI: 10.1145/1107499.1107504. URL: http://doi.acm.org/10.1145/1107499.1107504.

[SÇZ05b] Michael Stonebraker, Uğur Çetintemel ir Stan Zdonik. The 8 requirements of real-time stream processing. *ACM Sigmod Record*, 34(4):42–47, 2005.

[SS15] Abdul Ghaffar Shoro ir Tariq Rahim Soomro. Big data analysis: apache spark perspective. *Global Journal of Computer Science and Technology*, 2015.

[tut18] tutorialspoint.com. Apache storm - core concepts. https://www.tutorialspoint.com/apache_storm/apache_storm_core_concepts.htm, 2018.

[ZHA17] Ji ZHANG. How to achieve exactly-once semantics in spark streaming. http://shzhangji.com/blog/2017/07/31/how-to-achieve-exactly-once-semantics-in-spark-streaming/, 2017.