VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Rodiklių duomenų kaupimas, transformavimas ir analizė, naudojant srautinį duomenų apdorojimą

Indicator Data Collection, Transformation and Analysis Using Stream Processing

Kursinis darbas

Atliko: 3 kurso 5 grupės studentas
Vytautas Žilinas (parašas)

Darbo vadovas: lekt. Andrius Adamonis (parašas)

Vilnius – 2018

TURINYS

Įvadas

Šiame darbe rodiklių duomenys yra apibrėžti kaip didelių duomenų tipas. Šiuos duomenis galima transformuoti, analizuoti ir grupuoti pagal pasirinktus rodiklius, pavyzdžiui: bazinė mėnesio alga, mirusiųjų skaičius pagal mirties priežastis, krituliai per metus. Rodiklių duomenų bazės pasižymi tuo, kad duomenys į jas patenka iš daug skirtingų tiekėjų ir patekimo laikas tarp tiekėjų nėra sinchronizuojamas. Tuo tarpų suagreguotą informaciją vartotojai gali užklausti irgi bet kurio metu.

Lietuvoje rodiklių duomenų bazės pavyzdys yra „Lietuvos statistikos departamento“ duomenų bazė, kurioje užklausas galima vykdyti `https://osp.stat.gov.lt/statistiniu-rodikliu-analize#/` puslapyje, kuris leidžia ieškoti duomenis apdorotus pagal vieną arba kelis rodiklius. Kitas pavyzdys yra „DataBank“ `http://databank.worldbank.org` - pasaulinio lygio rodiklių duomenų bazių rinkinys, turintis 69 skirtingas duomenų bazes, pavyzdžiui - „World development indicators“, „Gander statistics“ [Ban18].

Rodiklių duomenims galima pritaikyti dalį didelių duomenų charakteristikų ir apsibrėžti, kurios iš jų sukelia daugiausiai iššūkių. Didelių duomenų iššūkiai yra apibrėžti Gartner's Doug Laney pristatytu 3V modeliu [Lan01], kuris buvo papildytas Bernard Marr iki 5V modelio [Mar14]:

- Tūris (angl. Volume). Apibrėžia generuojamų duomenų kiekius. Didelių duomenų atveju yra šnekama apie duomenų kiekius, kuriuos yra sudėtinga arba neįmanoma saugoti ir analizuoti centralizuotomis duomenų bazių valdymo sistemomis. Rodiklių duomenų kiekiai nesudaro problemos saugojant, tačiau problema yra rodiklių duomenų analizė, kadangi tuos pačius duomenis reikia apdoroti pagal neribotą kiekį skirtingų rodiklių.
- Greitis (angl. Velocity). Apibrėžia greitį, kuriuo nauji duomenys yra generuojami. Rodiklių duomenų atveju, tai yra svarbu, kadangi nauji duomenys, kurie gali tikti skirtingiems rodikliams, yra generuojami pastoviai.
- Įvairovė (angl. Variety). Apibrėžia duomenų tipus. Duomenys gali būti: struktūrizuoti, nestruktūrizuoti arba dalinai struktūrizuoti [ZE+11]. Rodiklių duomenys yra struktūrizuoti, todėl tai nėra aktualus iššūkis.
- Tikrumas (angl. Veracity). Apibrėžia duomenų teisingumą ir kokybę. Pavyzdžiui, jeigu būtų analizuojamas „Twitter“ socialinio tinklo žinučių turinys, būtų gauta daug gramatikos klaidų, naujadarų, slengo. Statistikos departamento atveju duomenys visada, kiek įmanoma, bus tvarkingi, kadangi tai yra duomenys surinkti iš dokumentų ir apklausų, o ne laisvai įvedami.
- Vertė (angl. Value). Apibrėžia duomenų ekonominę vertę. Rodiklių duomenys yra vertingi įstaigoms, nes tos įstaigos užsiima tik rodiklių duomenų kaupimu ir analize. Iš techninės pusės ši charakteristika yra svarbi, nes duomenų apdorojimo ir kaupimo sprendimai daro įtaką įstaigų veiklai, kurios kaupia ir apdoroja rodiklių duomenis. Todėl šie apdoroti duomenys turi būti pasiekiami be prastovos laiko.

Darbo tikslas: Eksperimento būdu išbandyti rodiklių duomenų kaupimo, transformavimo ir analizės uždavinių sprendinius, palyginant sprendimą, naudojantį reliacinę duomenų bazę, su sprendimu, naudojančiu srautinį duomenų apdorojimą.

Uždaviniai:

1. Atlikti skirtingų srautinio duomenų apdorojimo sprendimo architektūrų analizę ir pasirinkti vieną iš jų tyrimui.
2. Sukurti testinių duomenų generatorių.
3. Realizuoti reliacinės duomenų bazės rodiklių duomenų apdorojimo sprendimą.
4. Realizuoti srautinio duomenų apdorojimo architektūros rodiklių duomenų kaupimo sprendimą.
5. Išmatuoti srautinio duomenų apdorojimo ir reliacinės duomenų bazės sprendimų pralaidumus ir palyginti testavimo rezultatus.

Šiame darbe nagrinėjamas rodiklių duomenų kaupimas ir agregavimas fokusuojantis į greiti. Siekiama palyginti tracidinį sprendimą su duomenų baze su sprendimu, išpildytų srautinio apdorojimo architektūra. Buvo palygintos greitaveikos pagal pralaidumą, kuris parodo koks kiekis duomenų yra apdorojamas nuo įeigos pradžios iki suagregavimo pagal rodiklius. Pralaidumas yra įprastinė tokių sistemų matavimo metrika greitaveikai išmatuoti, kuri yra naudojama ir kitų šaltinių: [CDE+16; KRK+18; LLD16] Eksperimentui buvo naudojamas labai paprastas duomenų modelis, kuriame duomenys buvo apdorojami tik pagal vieną rodiklį.

1. Duomenų apdorojimo tipai

1.1. Srautinis duomenų apdorojimas

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 visa programa susidaro iš skirtingu modulių, kurie nepriklauso vienas nuo kito, ir būtent 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. BLODI kalbos autoriai teigia, kad vienas iš šios programavimo kalbos privalumų yra tas, kad ją galėjo išmokti žmonės, kurie nebuvo programavimo ekspertai .

Siekiant apžvelgti modernias srautinio duomenų apdorojimo architektūras reikia apsibrėžti srautinio apdorojimo sistemų galimybes. 2005 metais Michael Stonebraker apibrėžė 8 taisykles realaus laiko (angl. real-time) srautinio duomenų apdorojimo architektūrai [SÇZ05]:

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 abstrakciojos lygmens srautinio apdorojimo sistemos reikalauja ilgesnio programavimo laiko ir brangesnio palaikymo. Tuo tarpu aukšto abstrakcijo 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 [SÇZ05]. Todėl norint sužinoti tam tikro srautinio duomenų apdorojimo sprendimo tinkamumą uždaviniui reikia išanalizuoti to sprendimo galimybes.

1.2. Paketinis duomenų apdorojimas

Paketinis duomenų apdorojimas (angl. batch processing) - yra duomenų apdorojimo paradigma, kai per vieną kartą apdorojami tam tikrą laiką renkami duomenys [Vas17]. Viena iš tokio apdorojimo sistemų pavyzdžių būtent ir yra reliacinė duomenų bazė. Reliacinių duomenų bazių valdymo sistemos (angl. Relational database management systems) - tai duomenų valdymo sistema paremta reliaciniu modeliu, kurį pirmą kartą aprašė 1969 metais [Cod69]. Remiantis `https://db-engines.com/en/ranking` 2018 metų birželio mėnesio rodiklius šiuo metu pagal populiarumą tarp reliacinių ir NoSQL duomenų bazių sistemų pirmos 5 vietos (iš 343) yra:

1. Oracle (Reliacinė DBVS) - 1311.25
2. MySQL (Reliacinė DBVS) - 1233.69
3. Micorsoft SQL Server (Reliacinė DBVS) - 1087.73
4. PostgreSQL (Reliacinė DBVS) - 410.67
5. MongoDB (NoSQL DBVS paremta dokumentų saugyklos modeliu) - 343.79

Šie rezultatai yra apskaičiuojami pagal „DB-engines“ algoritmą, kuris atsižvelgia į sistemų paminėjimus svetainėse, paieškos dažnį paieškos varikliuose, techninių diskusijų kiekį populiariose su informacinėmis technologijomis susijusiose svetainėse, profesinių tinklų („Linkedin“) profiliuose, populiarumą socialiniuose tinkluose [DBE18]. Kadngi reliacinės duomenų bazių valdymo sistemos stipriai lenkia, bet kokias kitas saugyklas. Būtent toks populiarumas ir lemia, kad jos yra naudojamos ir duomenų apdorojimui. Kadangi reliacinė duomenų bazės jau yra įdiegtos daugeliuose įmonių, reiškia įmonei nereikia leisti papildomų lėšų: išanalizuoti kitokios sistemos tinkamumą užduočiai, sukurti sprendimą, palaikyti naują sistemą, pasisamdyti naują žmogų mokanti dirbti su šia sistema arba apmokyti esamą.

Reliacinių duomenų bazių duomenų transformacijos ir agregavimo būdas yra procedūros (angl. stored procedures), kurios apdoroja duomenis tiesiai iš duomenų bazės naudojant reliacinę matematiką. Tačiau jeigu paanalizuoti procedūrą, kaip duomenų apdorojimo architektūrą pagal 1.1 skyriuje apibrėžtas 8 taisykles, galima mtegti, jog procedūra neatitinka 1-os taisyklės, kuri teigia, kad duomenys turi judėti. Procedūros yra leidžiamos tik vartotojui užklausus, todėl šis rei-

kalavimas yra neišpildytas, ir 7-os taisyklės, kuri teigia, jog architektūra turi užtikrinti sugebėjimą paskirstyti sistemos darbus automatiškai, taičiau dalis reliacinių duomenų bazių nepalaiko horizontalų plečiamumą [Cat11; Ram16]. Kadangi kuo daugiau duomenų yra reliacinėje duomenų bazėje tuo ilgiau užtrūks tų duomenų apdorojimas, todėl reliacinių duomenų bazių sprendimas pažeidžia 8-tą duomenų apdorojimo taisyklę ir yra mažiau tinkamas rodiklių duomenų apdorojimui.

2. Srautinio duomenų apdorojimo sprendimų ypatumai

Šiame skyriuje lyginamos trys atviro kodo srautinio apdorojimo sprendimai „Apache Storm“, „Apache Spark“ ir „Apache Flink“ 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 net ir atsiradus klaidoms.
- 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.

2.1. 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, nes reikia užtikrinti, kad operacija bus vykdoma būtent vieną kartą kiekviename srautinio apdorojimo žingsnyje: duomenų gavime, kuris stipriai priklauso nuo duomenų šaltinio, duomenų transformacijos, kuria turi užtikrinti pats 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]. Jeigu sprendimas reikalauja tiksliai vieno karto apdorojimo, tada turi būti pasirinkti „Apache Spark“, „Apache Flink“ sprendimai arba „Apache Storm Trident“ - „Apache Storm“ sprendimu paremtas aukšto abstrakcijos lygmens sprendimas užtikrinantis tiksliai vieno karto apdorojimą. Tačiau 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].

2.2. Uždelstumas

Uždelstumas srautinio apdorojimo sprendimams yra matuojamas laiku, kruis 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“.

2.3. Pralaidumas

Pralaidumas apibrėžia kokį kiekį procesų sistema 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 atveju ir paskutinis liko „Apache Storm“, kuris turi žemą uždelstumą, todėl nukenčia pralaidumas.

2.4. Abstrakcijos lygmuo

„Apache Storm“ parašytos programos 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 programas, kadangi programuotojas turi pilną kontrolę.

„Apache Spark“ parašytos programos yra aukšto abstrakcijos lygmes. Programa aprašoma funkciškai, todėl kodo rašymas trunka daug trumpiau ir jį yra daug patogiau 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 pati 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 sprendimai.

2.5. Apibendrinimas

Iš šių trijų sprendimų reikėjo pasirinkti vieną, kuri labiausiai tiks rodiklių duomenų apdorojimui. Šis sprendimas turi sugebėti greitai apdoroti duomenis, prioretizuojant greitį virš tikslumo, kadangi vienas iš pagrindinių naudotojų yra statistiką renkančios įstaigos, kurios gali sau leisti tam tikrą paklaidą, ir programuotojas turi galėti aprašyti daug skirtingų sprendimų skirtingiems rodikliams.

Pagal atlikta analizę 1 lentelėje ir apsibrėžtų reikalavimų šiam uždaviniui tinkamiausias srautinio apdorojimo sprendimas yra „Apache Storm“. Nors jos pralaidumas yra žemas, rodiklių duo-

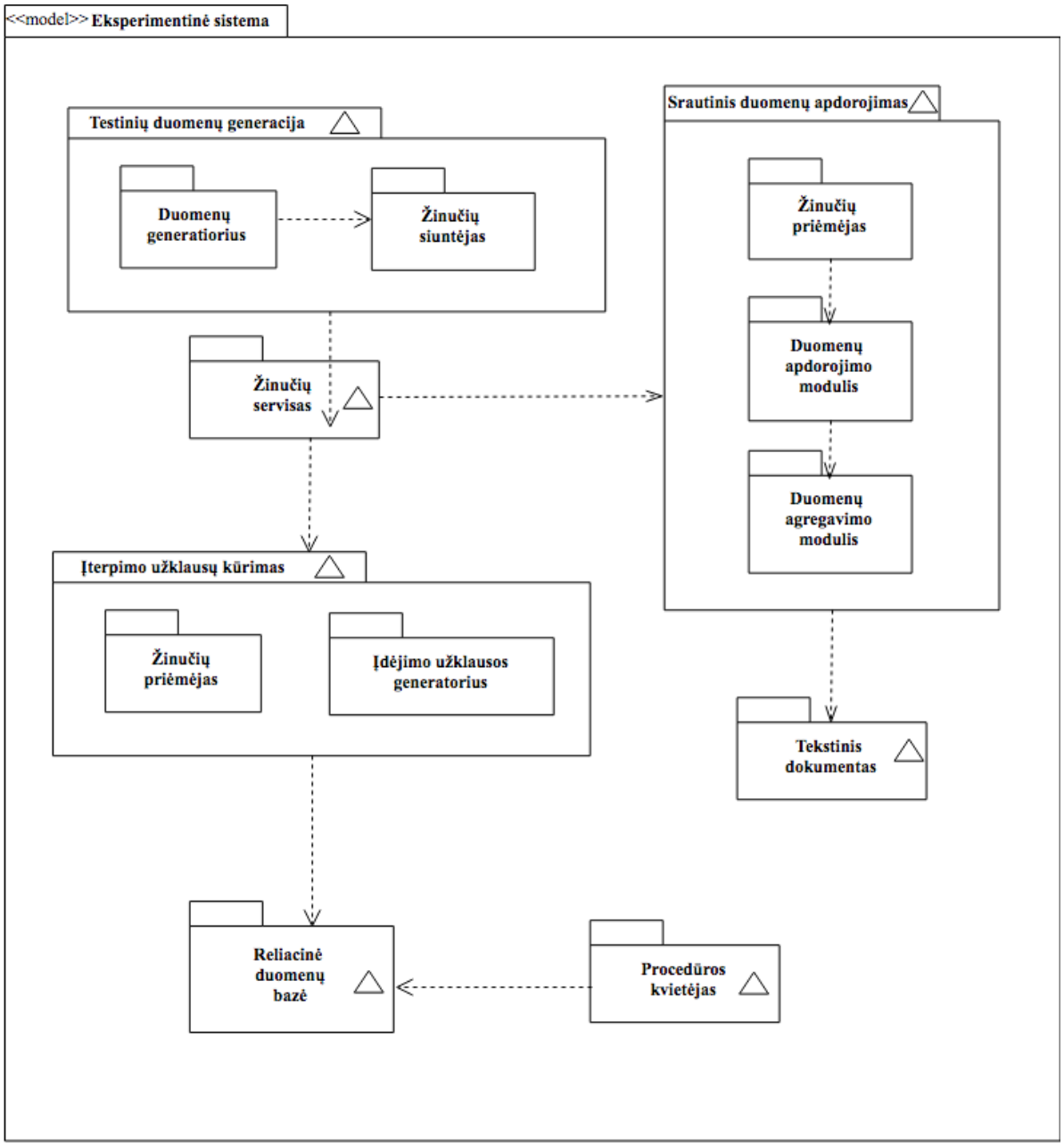
1 lentelė. Srautinių duomenų apdorojimo sprendimų palyginimas

Charakteristika	„Apache Storm“	„Apache Spark“	„Apache Flink“
Pristatymo semantika	Bent vieną kartą	Tiksliai vieną kartą	Tiksliai vieną kartą
Uždelstumas	Žemas	Aukštas	Vidutinis
Pralaidumas	Žemas	Aukštas	Vidutinis
Abstrakcijos lygis	Žemas	Aukštas	Aukštas

menų apdorojimui daug aktualiau yra greitis. Taip pat žemas abstrakcijos lygis leis optimizuoti kūriama sprendimą.

3. Eksperimentinė sistema

Eksperimentui buvo pasirinkta supaprastinta uždavinio versija norint palyginti reliacinės duomenų bazės sprendimo ir srautinio duomenų apdorojimo sprendimo greitaveikas . Uždaviniui pasirinkta sukurti supaprastintą elektroninės parduotuvės duomenų bazės schemą (2 pav.), kurios rodiklis - uždirbti pinigai pagal parduotuvę. Reliacinės duomenų bazės apdorojimui parašyta procedūra, o srautinio duomenų apdorojimo sprendimui sukurta programa, kurių rezultatas yra vienodas. Eksperimento tikslas - išmatuoti ir palyginti abiejų sistemų greitaveiką, matuojant jų pralaidumą ir vieno naujo duomens apdorojimo laiką. Šiam eksperimentui sukurta architektūra pavaizduota 1 paveikslėlyje.



1 pav. Eksperimentinės sistemos architektūra

2 pav. Reliacinės duomenų bazės schema testavimui

3.1. Eksperimento realizacija

3.1.1. Testinių duomenų generavimo sprendimo realizacija

Testinių duomenų generavimo sistema yra sudaryta iš dviejų dalių: „Python“ kalba parašytas duomenų generatorius ir siuntėjas į žinučių sistemą bei „Apache Kafka“ žinučių sistema, kuri yra atsakinga už duomenų perdavimą iš generatoriaus į kitus modulius.

Su „Python“ kalba parašytas duomenų generatorius, kuris sukuria nustatytą kiekį duomenų, naudojant „random“ biblioteką. Kai apibrėžtas kiekis duomenų sugeneruotas, „Python“ programa juos po vieną siunčia į „Apache Kafka“ žinučių sistemą, naudojant „kafka-python“ biblioteką. Generuojami duomenys yra pritaikyti skirtingiems sprendimams. Į srautinio apdorojimo sistemą siunčiami tik duomenys, kuriuos reikia apdoroti, ir raktas, pagal kurį apdoroti duomenys agreguojami, o į reliacinės duomenų bazės sprendimą siunčiami tik vienos iš lentelių pirminis raktas, kuris parenkamas atsitiktiniai iš visų įmanomų raktų, ir naujas duomuo. Duomenų siuntimas į žinučių sistemą vykdomas nepertraukiamai.

„Apache Kafka“ - tai servisas gaunantis, saugojantis ir siunčiantis duomenis, kurie yra vadinami žinutėmis (angl. messages). Žinutės yra skirstomos pagal temas (angl. topics). Yra žinučių kūrėjai (angl. producers), kurie siunčia žinutes į tam tikras temas, ir vartotojai (angl. consumers), kurie prenumeruoja (angl. subscribe) prie temų, kad gautu su jomis susijusias žinutes [The14]. Taip pat visos žinutės šia sistema yra perduodamos tekstiniu formatu tam, kad bendrauti su „Apache Kafka“ būtų įmanoma su bet kokia programavimo kalba. Bet dėl to nukenčia greitaveika, kadangi prisideda papildomas darbas - konvertavimas iš teksto į reikiamą objektą.

Šis testinių duomenų generavimo sprendimas tinka srautinio apdorojimo sprendimams, nes „Apache Kafka“ siunčia žinutes, kai tik gauna duomenis ir todėl duomenų apdorojimas gali vykti iš karto, o ne tada, kai sistema užklausia. Taip pat „Pyhton“ programos sugeneruotus duomenis galime be sunkumų apdoroti su srautinio apdorojimo sprendimu, kuris yra parašytas bet kokia kita programavimo kalba. Autoriaus sukurtas reliacinių duomenų bazės sprendimas taip pat naudoja „Apache Kafka“. Nors ir reliacinės duomenų bazės sprendimo atžvilgiu, pasirinkta architektūra yra perteklinė, tačiau net pridėjus šį papildomą sluoksnį duomenų įrašymas sulėtėja nežymiai.

3.1.2. Srautinio duomenų apdorojimo sprendimo realizacija

„Apache Storm“ sistemai kuriama programa yra vadinama topologija (angl. topology), kuri susideda iš „Spout“ ir „Bolt“ modulių. „Spout“ tai modulis gaunantis duomenis iš išorinės sistemos ir perduodantis juos į pirmą „Bolt“ modulį. „Bolt“ yra atsakingas už duomenų apdorojimą ir atidavimą atgal į išorę. Moduliai duomenis tarpusavyje perduoda „Apahe Storm“ pateikiamu „Tuple“ tipu, kuris laiko duomenis ir „Spout“ modulio sugeneruotą identifikatorių, kurio pagalba yra užtikrinama, kad duomenys sėkmingai nuėjo iki sekančio žingsnio. Pasirinktam uždaviniui spręsti buvo pasirinkta sukurti:

1. „kafka-spout“ - „Spout“ modulis, kuris gauna duomenis iš „Apache Kafka“ žinučių sistemos ir perduoda juos po jo einančiam „Bolt“ moduliui.
2. „calculate-price-bolt“ - „Bolt“ modulis, kuris gauna tekstinio tipo duomenis iš „kafka-spout“ ir apdoroja po vieną atėjusį „Tuple“.
3. „aggregate-price-bolt“ - „Bolt“ modulis, kuris gauna apdorotą „Tuple“ iš „calculate-price-bolt“ ir įdeda jį į „HashMap“ tipo sąrašą. Visą šį sąrašą kas tam tikra laiko intervalą spausdina į tekstinį dokumentą.

Kadangi „Apache Storm“ yra žemo abstrakcijos lygmens sprendimas, programuotojas turi pats numatyti paralelizmo lygį kiekviename modulyje. Po bandymų buvo pasirinkta modelių konfigūraciją daryti tokią: „kafka-spout“ - 5 paralelus procesai, „calculate-price-bolt“ - 5 paralelus procesai ir „aggregate-price-bolt“ - 1 procesas. Poskutinio modulio negalima leisti paraleliai, nes jame saugomas bendras sąrašas, kurio keitimas iš kelių gijų sugadina duomenis. Srautinio sprendimo greitaveiką matuosime:

1. Kiek duomenų srautinis sprendimas apdoroja per sekundę.
2. Kiek užtrunka vieno duomens apdorojimas nuo duomens gavimo iš „Apache Kafka“ serviso iki gauto duomens agregavimo į bendrą sąrašą pabaigos.

3.1.3. Reliacinės duomenų bazės sprendimo realizacija

Reliacinės duomenų bazės realizavimui buvo pasirinkta „Microsoft SQL Server“ duomenų bazė, o duomenų apdorojimui parašyta procedūra (angl. stored procedure) „GetReports“. Sukurta minimali duomenų bazė, su 3 lentelėmis (2 pav.), kurių pirminiai raktai yra indeksuojami. Sukurta procedūra „GetReports“ sujungia šias tris lenteles į vieną, kurioje yra parodoma kiekvienos parduotuvės uždirbti pinigai pagal nesudėtingą sumavimo formulę.

Kad duomenys iš generatoriaus patektų į duomenų bazę buvo sukurta programa su „Python“ kalba, kuri gauna duomenis iš „Apache Kafka“ žinučių sistemos ir talpina jas duomenų bazėje. Kiekviena gauta žinutė yra paverčiama į „Python“ žodyno (angl. dictonary) tipą ir įterpiama į įvedimo užklausą, kuri iš karto yra įvykdoma duomenų bazėje. Taip užtikrinamas nepertraukiamas duomenų įvedimas į reliacinę duomenų bazę.

Procedūros testavimui buvo naudojama papildoma „Python“ kalba parašyta programa, kuri gavusi žinutę iš „Apache Kafka“ sistemos sukūria įdėjimo užklausą į duomenų bazę. Taip pat „GetReports“ procedūros iškvietimui buvo naudojama „Python“ kalba parašyta programa, kuri periodiškai kviečia šią procedūrą. Kadangi negalima testuoti procedūros taip pat kaip srautinio ap-

dorojimo sprendimą, šis testas buvo daromas kitaip. Vienu metu buvo įvedami nauji duomenys ir tuo pat metu, tam tikru laiko intervalu, kviečiama procedūra ir buvo tikrinama:

1. Kiek duomenų per sekundę galima įdėti į duomenų bazę.
2. Kiek sekundžių sulėtėja procedūra, kai yra pastoviai įvedami nauji duomenys.

3.2. Eksperimento vykdymas

Kompiuterio komponentų ir programinės įrangos versijos, su kuriomis buvo bykdytas eksperimentas specifikacijos pateikto 2 lentelėje.

2 lentelė. Įrangos specifikacija

Procesorius	Intel i7-7700k 4,5 GHz
Operatyvioji atmintis	16 GB - 2666 Mhz
Ilgalaikė atmintis	SSD 512 GB
Operacinė sistema	Windows 10 64bit
Reliacinė duomenų bazė	Windows SQL Server 14
Srautinio apdorojimo sprendimas	Apache Storm 1.2.2
Žinučių sistema	Apache Kafka 1.1.0

3.2.1. Reliacinės duomenų bazės sprendimo eksperimento vykdymas

Į pradinę duomenų bazę (2 pav.) buvo įrašyti: 2 000 parduotuvių , kurios turi 0 arba daugiau prekių, 1 000 000 prekių , kurios turi 0 arba daugiau pirkimų, ir 10 000 000 pirkimų. Prieš pradedant eksperimentą buvo išmatuota, kiek užtrunka „GetReports“ procedūra, neapkraunant duomenų bazės kitais veiksmais, ir kiek laiko užtrunka visų duomenų įdėjimas, nedarant kitų užklausų. Eksperimento eiga su reliacinės duomenų bazės sprendimu:

1. Paleidžiama „Python“ programa gaunanti žinutes iš „Apache Kafka“ žinučių sistemos ir įterpianti gautus duomenis į duomenų bazę.
2. Paleidžiama programa kviečianti procedūrą „GetReports“ ir nustatyta, kad ji procedūrą kviestų kas 5 sekundes norint simuliuoti realią sistemos apkrovą. Po kiekvieno kvietimo ši programa fiksuoja, kiek laiko užtrūko procedūra ir kelintą kartą ji yra kviečiama.
3. Paleidžiamas testinių duomenų generatorius.

3.2.2. Srautinio duomenų apdorojimo sprendimo eksperimento vykdymas

Prieš pradedant testavimą į sistemą buvo sugeneruoti 10 000 000 įrašų, kad būtų sudarytos sąlygos panašios į reliacinės duomenų bazės. Šiai sistemai pagal nutylėjimą buvo išskirti 832 megabaitai operatyvios atminties (angl. RAM). Eksperimento eiga su srautinio duomenų apdorojimo sprendimu:

1. Paleidžiama „Apache Storm“ sistema ir į ją užkraunama sukurta topologija.
2. Paleidžiamas testinių duomenų generatorius.

3.3. Eksperimento rezultatai

3 lentelė. Rezultatų palyginimas

	Reliacinės duomenų bazės sprendimas	Srautinio apdorojimo sprendimas
Pralaidumas (duomenų kiekis per sekundę)	~4,200	~5,475
Naujo duomens apdorojimas ir agregavimas	~12 sekundžių	~75 milisekundės

3.3.1. Reliacinės duomenų bazės sprendimo eksperimento rezultatai

Tiesiog kviečiama procedūra „GetReports“ trunka vidutiniškai 12 sekundžių. Leidžiant įdėjimo užklausas, be jokios kitos apkrovos duomenų bazei, 1 500 000 duomenų buvo įdėti per 358 sekundžių, beveik 6 minutes. Tačiau paleidus tokį pat pastovų įrašų srautą į duomenų bazę ir bandant tuo pačiu metu gauti apdorotus duomenis, procedūra užtruko vidutiniškai 29 sekundes bei apdorotas duomenų sąrašas po generavimo atsiliko virš 120 000 įrašų - tiek duomenų buvo įdėta procedūros veikimo metu. Per visą naujų duomenų įdėjimo laiką procedūra „GetReports“ buvo ivykdyta 10 kartų. Taip pat vienas iš procedūros trūkumų - įdėjus bent vieną naują duomenį ir norint gauti ataskaitą turi iš naujo būti vykdoma procedūra, kurios trukmė, kai nėra paleista jokia kita užklausa, apie 12 sekundžių su testuojama duomenų imtimi.

3.3.2. Srautinio duomenų apdorojimo sprendimo eksperimento rezultatai

Paleista testavimo programa siunčianti 1 500 000 duomenų neribojant siuntimo greičio, rezultatai buvo stebimi įrašais tekstiniame dokumente. Visi duomenys buvo apdoroti per 274 sekundes, tai yra maždaug 4,5 minutės. Vidutiniška vienos operacijos trukmė nuo patekimo į „Apache Kafka“ žinučių sistemą iki agregavimo pabaigos - 75 milisekundės. Lėčiausia sistemos grandis buvo „aggregate-price-bolt“ modulis, nes jį negalima leisti paraleliai. Šios sistemos veikimą pagreitinti įmanoma tik perdavus paskutinio modulio agregavimą kitai architektūrai, kadangi visus kitus veiksmus galima leisti paraleliai.

Rezultatai ir išvados

Darbo rezultatai:

1. Sukurtas testinių duomenų generatorius - `https://git.mif.vu.lt/vyzi2849/test-data-generator-python`, sukurtas supaprastinto uždavinio sprendimas su reliacine duomenų baze - `https://git.mif.vu.lt/vyzi2849/database-testing-tools-python` ir su pasirinktu srautinio apdorojimo sprendimu - `https://git.mif.vu.lt/vyzi2849/storm-topology-java`.
2. Uždavinio sprendimai palyginti pagal pralaidumo testų rezultatus.

Išvados:

1. Eksperimento būdu pagrįsta, kad siūloma sistema gali būti įgyvendinama.
2. Pralaidumo testavimo metu įrodyta, kad srautinio duomenų apdorojimo sprendimas duomenis išskaičiuoja agreguotas rodiklio reikšmes greičiau (75 milisekundės) nei reliacinės duomenų bazės sprendimas (12 sekundžių) ir srautinio apdorojimo sprendimas nėra priklausomas nuo kitų tuo pačiu metu vykdomų procesų.
3. Eksperimentui sukurtos sistemos srautinio duomenų apdorojimo sprendimo pralaidumas yra aukštesnis (5,475 duomenų per sekundę) nei reliacinės duomenų bazės (4,200 duomenų per sekundę).
4. Srautinio apdorojimo architektūros sprendimas „Apache Storm“ pasirinktam uždaviniui užtikrina duomenų apdorojimą milisekundžių greičiu, kadangi suagreguotos reikšmės laikomos operatyviojoje atmintyje ir nauji apdoroti duomenys yra pridedami prie jau esamo sąrašo.

Literatūra

[Ban18] The World Bank. List of databank databases. http://databank.worldbank.org/data/databases/. 2018.

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

[Cat11] Rick Cattell. Scalable sql and nosql data stores:12–27, 2011. URL: http://doi.acm.org/10.1145/1978915.1978919.

[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*. IEEE, 2016, p. 1789–1792.

[Cod69] Edgar F Codd. Derivability, redundancy, and consistency of relations stored in large data banks. Tech. atask. IBM Research Report, 1969.

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

[DBE18] DB-Engines. Method of calculating the scores of the db-engines ranking. https://db-engines.com/en/ranking_definition. 2018.

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

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

[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.

[Lan01] Doug Laney. 3d data management: controlling data volume, velocity and variety. *Meta group research note*, 6(70), 2001.

[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.

[Mar14] Bernard Marr. Big data: the 5 vs everyone must know. *Linkedin pulse*, 6, 2014.

[Ram16] Jokūbas Ramanauskas. Išmaniųjų skaitiklių duomenų kaupimas, transformavimas ir analizė, naudojant newsql duomenų bazę. 2016.

[SÇZ05] 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.

[The14] KMM Thein. Apache kafka: next generation distributed messaging system. *International journal of scientific engineering and technology research*, 3(47):9478–9483, 2014.

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

[Vas17] Gowthamy Vaseekaran. Big data battle : batch processing vs stream processing. `https://medium.com/@gowthamy/big-data-battle-batch-processing-vs-stream-processing-5d94600d8103`. 2017.

[ZE+11] Paul Zikopoulos, Chris Eaton ir k.t. *Understanding big data: analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

[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.