

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

IVADAS	3
1. DUOMENŲ APDOROJIMO TIPAI	5
1.1. Srautinis duomenų apdorojimas	5
1.2. Paketinis duomenų apdorojimas	6
2. SRAUTINIŲ DUOMENŲ ARCHITEKTŪRŲ SPRENDIMŲ YPATUMAI	8
2.1. Pristatymo semantika	8
2.2. Uždelstumas	8
2.3. Pralaidumas	9
2.4. Abstrakcijos lygis	9
2.5. Apibendrinimas	9
3. EKSPERIMENTINĖ SISTEMA	11
3.1. Eksperimento realizacija	11
3.1.1. Testinių duomenų generavimo sprendimo realizacija	11
3.1.2. Srautinio duomenų apdorojimo sprendimo realizacija	12
3.1.3. Reliacinės duomenų bazės sprendimo realizacija	12
3.2. Eksperimento vykdymas	13
3.2.1. Reliacinės duomenų bazės eksperimento vykdymas	13
3.2.2. Srautinio duomenų apdorojimo sprendimo eksperimento vykdymas	13
3.3. Rezultatai	13
REZULTATAI IR IŠVADOS	15
LITERATŪRA	16

Įvadas

Šiame darbe rodiklių duomenis laikysime, kaip didelių duomenų tipą, kurį galima transformuoti ir analizuoti ir kuris yra sugrupuotas pagal rodiklius, pavyzdžiui: bazinė mėnesio alga, mirusiųjų skaičius pagal mirties priežastis, krituliai per metus. Šie duomenys yra saugomi reliacinėse duomenų bazėse, kur užklausus vartotojui skaičiuojami apibendrinti rodikliai - sumos, vidurkiai ir kita statistika.

Lietuvoje geriausiai žinomas 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. Didesnis pavyzdys yra „DataBank“ <http://databank.worldbank.org> - pasaulinio lygio rodiklių duomenų bazių rinkinys, turintis 69 skirtingas duomenų bazes, pavyzdžiui - „World development indicators“, „Gender statistics“ ir kitus[Ban18].

Šiuo metu pasaulyje yra aktuali didelių duomenų (angl. Big data) tema, kuri yra susijusi su tokiais iššūkiais kaip didelių duomenų kiekių saugojimas, transformacija, analizė. Mes galime apibrėžti rodiklių duomenis kaip didelių duomenų tipą, todėl galime jiems pritaikyti didelių duomenų charakteristikas ir apibrėžti, kurios iš jų mums sudaro daugiausiai problemų. Šie iššūkiai apibūdinami Gartner's Doug Laney pristatytu 3V modeliu[Lan01], kuris vėliau 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 tradicinėmis duomenų bazių technologijomis. Rodiklių duomenų kiekiai nesudaro problemos saugojant, tačiau problema yra rodiklių duomenų analizė, kadangi tuos pačius duomenis reikia apdoroti pagal neapribotą skaičių skirtingų rodiklių.
- Greitis (angl. Velocity). Apibrėžia greitį, kuriuo nauji duomenis yra generuojami. Rodiklių duomenų atveju, tai yra svarbu, kadangi nauji duomenis, kurie gali tikt skirtingiems rodikliams yra generuojami pastoviai.
- Įvairovė (angl. Variety). Apibrėžia duomenų tipus. Duomenys gali būti: struktūrizuoti, nestruktūrizuoti arba dalinai struktūrizuoti[ZE⁺11]. Rodiklių duomenis yra struktūrizuoti, todėl tai nėra aktualus iššūkis.
- Tikrumas (angl. Veracity). Apibrėžia duomenų teisingumą ir kokybę. Pavyzdžiui, jeigu analizuotume „Twitter“ socialinio tinklo žinučių turinį gautume daug gramatikos klaidų, naujadarų, slengo. Statistinio departamento atveju duomenys visada, kiek įmanoma, bus tvarkingi, kadangi tai yra duomenys surinkti iš dokumentų ir apklausų, o ne laisvo įvedimo.
- Vertė (angl. Value). Apibrėžia duomenų ekonominę vertę. Rodiklių duomenys yra vertinami įstaigoms, nes tos įstaigos užsiima tik rodiklių duomenų kaupimu ir analizė, iš techninės pusės ši charakteristika yra svarbi iš tos pusės, kad duomenų apdorojimo ir kaupimo sprendimai tipiai daro įtaką įstaigos, kaupiančios rodiklių duomenis, ekonomikai. Taip pat šių duomenys ir jų analizė turi būti pasiekiami be prastovos laiko.

Darbo tikslas: Eksperimento būdu išbandyti rodiklių duomenų kaupimo, transformavimo ir analizės uždavinių sprendinius, palyginant sprendimą, naudojanti reliacinę duomenų bazę, su

sprendimu naudojančiu srautinį duomenų apdorojimą.

Užduotys:

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

Eksperimento metu testuojami sprendimai bus vykdomi ne su reliais rodiklių duomenimis, o su supaprastinto uždavinio sugeneruotais duomenimis.

1. Duomenų apdorojimo tipai

1.1. Srautinis duomenų apdorojimas

Srautinis duomenų apdorojimas (angl. Stream processing) - yra programavimo paradigma ekvivalenti anksčiau aprašytai duomenų tėkmės programavimo (angl. dataflow programming) paradigmai[Bea15]. Duomenų tėkmės programavimo paradigmos idėja, kad visa programa susidaro iš skirtingų modulių, kurie nepriklauso vienas nuo kito ir būtent tai leidžia sukonstruoti paraleliai skaičiuojančias programas. Viena iš pirmųjų duomenų tėkmės programavimo kompiliatorių yra BLODI - blokų diagramų kompiliatorius (angl. BLOck DIagram compiler), su kuriuo buvo kompiliuojamos BLODI programavimo kalba parašytos programos. Šia kalba parašytos programos atitinka inžinierinę elektros grandinės schemą, kur duomenis keliauja per komponentus kaip ir elektros grandinėje. Vienas iš šios programavimo kalbos privalumų buvo tai, kad ją galėjo išmokyti žmonės, kurie nebuvo programavimo ekspertai[KL61].

Kad apžvelgti modernias srautinio duomenų apdorojimo architektūras reikia apsibrėžti srautinio apdorojimo sistemų galimybes. 2005 metais Michael Stonebraker apibrėžė 8 taisyklės realaus laiko srautinio duomenų apdorojimo architektūroms[SCZ05]:

- 1 taisyklė: Duomenys turi judėti. Kad būtų užtikrinta žemas uždelšdamas sistema turi apdoroti duomenis nenaudojant duomenų saugojimo operacijas. Taip pat sistema turi ne pati užklausti duomenis, o gauti juos iš kito šaltinio automatiškai.
- 2 taisyklė: Duomenų transformacijos turi būti vykdomas SQL pobūdžio užklausomis. Žemo lygio srautinio apdorojimo sistemos reikalauja ilgesnio programavimo laiko ir brangesnio palaikymo. Tuo tarpu aukšto lygio sistema naudojanti SQL užklausas, kurias žino dauguma programuotojų ir naudojama daug 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. Taip pat ši architektūra turi sugebėti susidoroti su vėluojančiomis žinutėmis, pratęsiant laiko tarpą per kurį tą žinutė turi ateiti.
- 4 taisyklė: Architektūra turi generuoti nuspėjamus rezultatus. 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 atsirado, nes reikėjo galimybės nepastebimai perjungti apdorojimą iš istorinių duomenų į gyvus 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 momentaliai. Anksčiau minėtos taisyklės nėra svarbius, jeigu sistema nesugeba greitai susidoroti su dideliu kiekiu naujų duomenų. Kad tai sistema pasiektu turi būti naudojamas ne tik teisingas ir greitas srautinio apdorojimo sprendimas, bet ir gerai optimizuota sistema.

Šie reikalavimai yra sukurti tik teoriškai ir egzistuoja labai nedaug srautinio apdorojimo architektūrų atitinkančių visas šias taisykles. Tam kad išsirinkti tinkamą architektūrą sprendžiamam uždaviniui, konkrečios srautinio apdorojimo architektūros yra apžvelgiamos ir lyginamos 2 skyriuje.

1.2. Paketinis duomenų apdorojimas

Paketinis duomenų apdorojimas (angl. Batch processing) - yra duomenų apdorojimo paradigma, kai duomenis yra saugomi saugykloje, o vėliau, po užklauso, apdorojami. Vienas iš tokio apdorojimo sistemų pavyzdžių būtent ir yra reliacinė duomenų bazė. Reliacinės duomenų bazių valdymo sistemos (angl. Relational database management systems) - tai duomenų valdymo sistema paremta reliaciniu modeliu pirmą kartą aprašytu 1969 metais[Cod69]. Pagal <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 paskirtos atitinkamai:

1. Oracle (Reliacinė DBVS) - 1311.25
2. MySQL (Reliacinė DBVS) - 1233.69
3. Microsoft 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 žinomose su informacinėmis technologijomis susijusiose svetainėse, profesionalių tinklų profiliuose, populiarumą socialiniuose tinkluose[DBE18]. Aiškiai matome, kad 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 bazė jau egzistuoja, reiškia jūmonei 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 apdorojimo būdas yra Procedūros(angl. stored procedures), kurios aprašomos SQL kalba ir gali apdoroti duomenis tiesiai iš duomenų bazės naudojant reliacinę matematiką. Tačiau jeigu pažiūrėsime į procedūrą, kaip duomenų apdorojimo architektūrą pagal 1.1 skyriuje apibrėžtas 8 taisykles, matysime, kad ji neatitinka: 1-os taisyklės: Duomenys turi

judėti. Procedūros yra leidžiamos tik vartotojui užklausus, todėl šis reikalavimas yra neišpildytas, ir 7-os taisyklės: Architektūra turi užtikrinti sugebėjimą paskirstyti sistemos darbus automatiškai. Didžioji dalis reliacinių duomenų bazių nepalaiko horizontalų plečiamumą[Cat11; Ram16] ir todėl viena sistema gali apdoroti tik pas ją esančius duomenis. Svarbiausiai, procedūra negali apdoroti greitai naujų duomenų įėjimo duomenų bazėje jau yra didelis kiekis duomenų, kuriuos reikia apdoroti, nes procedūra apdoroja ne tik naujus bet visus duomenis, tai pažeisdama 8-tą duomenų apdorojimo taisyklę, ir ko pasekoje reliacinis sprendimas analitiškai yra mažiau tinkamas rodiklių duomenų problemai.

2. Srautinių duomenų architektūrų sprendimų ypatumai

Šiame skyriuje palyginsime tris atviro kodo srautinio apdorojimo architektūras „Apache Storm“, „Apache Spark“ ir „Apache Flink“ pagal:

- Pristatymo semantika (angl. delivery semantics) - apibrėžia pagal kokį modelį bus pristatyti duomenis. Egzistuoja trys semantikos[Kah18]:
 - Bent vieną kartą (angl. At-least-once) užtikrina, kad duomenis bus apdoroti bent kartą, bet gali atsirasti dublikatų.
 - Ne daugiau vieno karto (angl. At-most-once) užtikrina, kad duomenis bus apdoroti daugiausiai tik vieną kartą, bet gali atsirasti praradimų.
 - Tiksliai vieną kartą (angl. Exactly-once) užtikrina, kad duomenis bus apdoroti tik vieną kartą net ir atsiradus klaidoms.
- Uždelstumas (angl. Latency) - apibrėžia kiek laiko užtruks įvykdyti kažkokį veiksmą arba gauti rezultatą.
- Pralaidumas (angl. Throughput) - apibrėžia kiek pavyks įvykdyti operacijų per tam tikrą laiko tarpą.
- Abstrakcijos lygis (angl. Abstraction) - apibrėžia kokio lygio programavimo sąsają pateikia architektūra.

2.1. Pristatymo semantika

„Apache Spark“ ir „Apache Storm“ architektūrų 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 architektūra sunaudoja daug resursų, nes reikia užtikrinti, kad operacija bus vykdoma būtent vieną kartą kiekvieniame srautinio apdorojimo žingsnyje: duomenų gavime, kas stipriai priklauso nuo duomenų šaltinio, duomenų transformacijos, kuri turi užtikrinti pati srautinio apdorojimo architektūra ir duomenų saugojime, kas turi būti užtikrinta architektūros ir naudojamų saugyklos[ZHA17].

„Apache Storm“ pristatymo semantika yra bent viena kartą (angl. at-least-once), kas reiškia, kad per šią architektūrą leidžiami duomenys visada pasieks pabaigą, tačiau kartais gali dubliuotis[DAM16]. Jeigu sprendimas reikalauja tiksliai vieno karto apdorojimo, tada mes turime rinktis arba vieną iš aukščiau minėtų architektūrų arba „Apache Storm Trident“ - ant „Apache storm“ architektūros pastatyta aukšto abstrakcijos lygio architektūra galinti užtikrinti tiksliai vieno karto apdorojimą. Tačiau jei uždavinys to nereikalauja, greičio, ypač uždelstume, atveju daug geriau pasirinkti pigesnę pristatymo semantiką[ZHA17].

2.2. Uždelstumas

Uždelstumas, srautinio apdorojimo architektūroms yra matuojamas milisekundėmis, kas parodo kaip greitai architektūra įvykdo vieną operaciją. Pagal Martin Andreoni Lopez darytus bandymus su šiomis architektūromis galime matyti, kad būtent „Apache Storm“ turi mažiausią uždelstumą, parinkus teisingai paralelizmo parametą ši architektūra su užduotimi susidorojo net iki 15

kartų greičiau. Antroje vietoje liko „Apache Flink”, o po jos „Apache Spark”[LLD16]. Tačiau kai architektūra turi žemą uždelstumą, tai ji turės taip pat ir žemą pralaidumą, kas nėra gerai, kai norima apdoroti daug duomenų iš karto.

2.3. Pralaidumas

Pralaidumas apibrėžia kokį kiekį procesų sistema gali įvykdyti per tam tikrą laiko tarpą. 2016 metais Sanket Chintapalli su kolegomis išmatavo „Apache Storm”, „Apache Spark” ir „Apache Flink” architektūrų pralaidumą ir uždelstumą ir palygino rezultatus. Kaip ir anksčiau manyta, „Apache Spark” turėjo aukščiausią pralaidumą iš visų, kadangi jis vienintelis iš trijų 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[CDE⁺16].

2.4. Abstrakcijos lygis

„Apache Storm” parašytos programos yra žemo lygio abstrakcijos. Kadangi tai yra žemo lygio programa mes turime apsirašyti visus srauto apdorojimo modulius, tai yra: setSpouts(..), kur nustatoma duomenų įeiga ir koks bus paralelizmas lygis, setBolt(..), kur 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]. Šios architektūros programų kūrimo laikas užtruks ilgiau negu kitoms architektūroms su aukštu abstrakcijos lygiu, tačiau žemas abstrakcijos leidžia rašyti daug greičiau veikiančias programas, kadangi programuotojas turi pilną kontrolę.

„Apache Spark” parašytos programos yra aukšto lygio abstrakcijos. Programa aprašoma funkciškai, todėl kodo rašymas trunka daug trumpiau ir tokį kodą daug patogiau skaityti. Tačiau prarandama galimybė optimizuoti ir paralelizmo klausimas paliekamas architektūrai. Kadangi „Apache Spark” yra ne pilnai srautinio, o mikro-paketinė (angl. micro-batching) architektūra, todėl vartotojas turi apsirašyti kokio dydžio paketais bus renkami duomenys [SS15].

„Apache Flink” parašytos programos yra aukšto lygio abstrakcijos. „Apache Flink” architektūra pati užsiima resursų distribucija, todėl programuotojui lieka tik parašyti veikiančią kodą, o sistema pati susitvarkys su paralelizmu[doc18]. Tačiau tai reiškia, kad su šia architektūra parašytos programos nepavyks optimizuoti taip pat gerai kaip žemo lygio abstrakcijos architektūros.

2.5. Apibendrinimas

Iš šių trijų architektūrų reikia pasirinkti vieną, kuri labiausiai tiks rodiklių duomenų apdorojimui. Ši architektūra turi sugebėti greitai apdoroti duomenis, prioretizuojant greitai 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.

1 lentelė. Srautinių architektūrų 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

Kaip matome iš atliktos analizės 1 lentelėje ir apsibrėžtų reikalavimų, mums labiausiai tinkanti srautinio apdorojimo architektūra yra „Apache Storm”. Nors jos pralaidumas yra žemas, mums daug aktualiau yra greitis, taip pat žemas abstrakcijos lygis leis daug geriau optimizuoti sprendimą su šia architektūra. Su ją kursime sprendimą, kurį lyginsime su reliacinės duomenų bazės sprendimu.

3. Eksperimentinė sistema

Eksperimentui buvo sukurta supaprastinta uždavinio versija tam, kad galima būtų palyginti reliacinės duomenų bazės sprendimo ir srautinį duomenų apdorojimo sprendimo greitaveikas. Šis pavyzdys yra aktualus tada, kai neapdorotų duomenų nereikia saugoti, o svarbi tik jų galutinė forma. Srautinio duomenų apdorojimo architektūra išveda tik apdorotus duomenis, o reliacinė duomenų bazė duomenis turi saugoti atmintyje, kadangi šie duomenis vėliau turi būti apdoroti su sukurta procedūra. Eksperimento tikslas - išmatuoti ir palyginti abiejų sistemų greitaveiką, matuojant jų pralaidumą ir vieno naujo duomens apdorojimo laiką.

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 ir „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 ir, kai apibrėžtas kiekis duomenų sugeneruotas, po vieną siunčia į „Apache Kafka” žinučių sistemą, naudojant „kafka-python” biblioteką. Generuojami duomenis yra pritaikyti skirtingiems sprendimams. Į srautinio apdorojimo sistemą siunčiama tik duomenis, kuriuos reikia apdoroti ir raktas pagal kurį apdoroti duomenis agreguojami, o į reliacinės duomenų bazės sprendimą siunčiami tik vienos iš lentelių pirminis raktas kuris parenkamas atsitiktiniais iš visų įmanomų, ir nauja duomenį. 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 žinutės 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 gali nukentėti šiek tiek greitaveikos kadangi prisideda papildomas konvertavimas iš teksto į tinkamą objektą.

Ši architektūra tinka srautinio apdorojimo sprendimams, kadangi ji išsiunčia žinutes, kai tik gauna duomenis ir todėl duomenų apdorojimas gali vykti iš karto, o ne tada kai sistema užklausia. Taip pat „Python” programos sugeneruotus duomenis galime be sunkumų apdoroti su srautinio apdorojimo sprendimu, kuris yra parašytas bet kokia kita programavimo kalba. Mūsų sukurta reliacinių duomenų bazės sprendimas taip pat naudoja „Apache Kafka”. Nors ir reliacinės duomenų bazės atžvilgiu, ši architektūra yra perteklinė, tačiau net pridėjus šį papildomą sluoksnį duomenų įrašymas sulėtėja nežymiai.

3.1.2. Srautinio duomenų apdorojimo sprendimo realizacija

„Apache Storm” programa yra vadinama topologija, kuri susideda iš „Spout” ir „Bolt” modulių. „Spout” tai modulis gaunantis duomenis iš išorinės sistemos ir perduodantis juos į pirmą „Bolt” modulį. „Bolt” yra atsakingi už duomenų apdorojimą ir atidavimą atgal į išorę. Moduliai tarpusavyje perduoda duomenų „Tuple” tipu, kuris laiko duomenis ir architektūros sugeneruotą identifikatorių, kurio pagalba užtikrina, kad duomenis sėkmingai nuėjo iki sekančio žingsnio. Šiam uždaviniui spręsti buvo pasirinkta suskurti:

1. „kafka-spout” - „Spout” modulis, kuris gauna duomenis iš „Apache Kafka” žinučių sistemos ir perduoda juos pirmam „Bolt” moduliui.
2. „calculate-price-bolt” - „Bolt” modulis, kuris gauna tekstinio tipo duomenis iš „kafka-spout” ir apdoroja vieną atejusį „Tuple”.
3. „aggregate-price-bolt” - „Bolt” modulis, kuris gauta apdorota „Tuple” įdeda į „HashMap” tipo sąrašą ir visą sąrašą perduoda toliau.

Kadangi „Apache Storm” yra žemo lygio architektūra programuotojas turi pats numatyti paralelizmo lygi kiekviename modulyje. Po bandymų buvo nuspręsta modelių konfigūraciją daryti tokią: „kafka-spout” - 5 paralelus procesai, „calculate-price-bolt” - 5 paralelus procesai ir „aggregate-price-bolt” - 1 paralelus procesas. Paskutinį moduli negalima leisti paraleliai, nes jame saugomas bendras sąrašas, kurio pakartotinas keitimas sugadina duomenis. Srautinio sprendimo greitimeika matuosime:

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

3.1.3. Reliacinės duomenų bazės sprendimo realizacija

Reliacinės duomenų bazės realizavimui pasirinkta „Microsoft SQL Server” duomenų bazė, o duomenų apdorojimui parašyta procedūra (angl. stored procedure) „GetReports”. Sukurta minimali duomenų bazė, su 3 lentelėmis (1 pav.). Sukurta procedūra „GetReports” sujungia šias tris



1 pav. Reliacinė duomenų bazė testavimui

lenteles į vieną, kurioje yra parodoma kiekvienos parduotuvės uždirbtus pinigus pagal nesudėtingą sumavimo formulę. Kad duomenis iš generatoriaus patektų į duomenų bazę sukurta programa su „Python” kalba kuri gauna duomenis iš „Apache Kafka” žinučių sistemos ir talpina jas duomenų bazėje, kiekviena gauta JSON užkoduota žinutė yra paverčiama į „Python” žodyno (angl. dicto-

nary) tipą ir įdedama į įdėjimo užklausą, kuri iškarto yra vykdoma duomenų bazėje. Tokiu būdu užtikrinamas nepertraukiamas duomenų įvedimas į duomenų bazę.

Testavimui bus naudojama papildoma „Python“ kalba parašyta programa, kuri gavusi žinutę iš „Apache Kafka“ sistemos sukurs įdėjimo užklausą į duomenų bazę. Taip pat „GetReports“ procedūros iškvietimui yra naudojama „Python“ kalba parašyta programa, kuri periodiškai kviečia minėtą procedūrą. Kadangi mes negalime testuoti procedūros taip pat kaip srautinę apdorojimo sprendimą, mes darysime šį testą kitaip. Vienų metų bus įvedami nauji duomenys ir tuo pačiu metu tam tikrais laiko intervalais kviečiama procedūra ir bus tikrinama:

1. Kiek duomenų per sekundę galima įdėti į duomenų bazę per sekundę.
2. Kiek sekundžių suletėja procedūra, kai yra pastoviai dedami duomenys.

3.2. Eksperimento vykdymas

Kompiuterio komponentų ir programų versijų su kuriais buvo vykdomas eksperimentas specifikacijos (2 lentelė).

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 architektūra	Apache Storm 1.2.2
Žinučių sistema	Apache Kafka 1.1.0

3.2.1. Reliacinės duomenų bazės eksperimento vykdymas

Pradinėje duomenų bazėje egzistuoja 2,000 parduotuvių, kurios turi 0 arba daugiau prekių, 1,000,000 prekių, kurios turi 0 arba daugiau pirkimų, 10,000,000 pirkimų.

3.2.2. Srautinio duomenų apdorojimo sprendimo eksperimento vykdymas

Prieš pradedant testavimą į sistemą buvo sugeneruota 10 milijonų įrašų, kad būtų sudarytos sąlygos tokios pat kaip ir reliacinės duomenų bazės. Paleista testavimo programa siunčianti 1,5 milijono duomenų neribojant siuntimo greičio, rezultatai buvo stebimi įrašais tekstiname dokumente. Šiai sistemai pagal nutylėjimą buvo išskirti 832 megabaitai operatyvios atminties (angl. RAM).

3.3. Rezultatai

Tiesiog kviečiant procedūrą „GetReports“ trunka vidutiniškai 12 sekundžių. Kviečiant įdėjimo funkciją, be jokios kitos apkrovos ant duomenų bazės, 1,5 milijonų duomenų buvo įdėti per

358 sekundžių, beveik 6 minutes. Tačiau vien paleidus vieną tokį pat pastovų įrašų srautą į duomenų bazę ir bandant tuo pačiu metu gauti apdorotus duomenis, procedūra užtruko vidutiniškai 29 sekundes ir taip pat apdorotas duomenų sąrašas po generavimo atsiliko virš 120,000 įrašų. Taip pat vienas iš procedūros trūkumų - įdėjus bent vieną naują duomenį ir norint gauti ataskaita turi iš naujo būti vykdoma procedūra, kurios trukmė apie 12 sekundžių su testuojama duomenų imtimi.

DB

Procedūra be jokių kitų procesų, su pradinių duomenų kiekiu, vidutiniškai užtrunka 12 sekundžių. T Visi, 1,5 milijonų, duomenų buvo apdoroti per 274 sekundes, kas yra maždaug 4,5 minutės. Vidutiniška vienos operacijos trukmė nuo patekimo į „Apache Kafka“ žinučių sistemą iki galutinio pridėjimo prie bendro sąrašo - 75 milisekundės. Lėčiausia sistemos grandis buvo „aggregate-price-bolt“, nes jo negalima buvo leisti paraleliai. Šios sistemos veikimą pagreitinti įmanoma tik perdavus paskutinio modulių agregavimą į sąrašą kitai architektūrai, kadangi visus kitus veiksmus galima leisti paraleliai.

Rezultatai ir išvados

Darbo rezultatai:

1. Sukurtas testavimo duomenų generatorius, sukurtas supaprastinto uždavinio sprendimas su reliacina duomenų baze ir su srautinio apdorojimo architektūra.
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 apdoroja duomenis greičiau nei reliacinių duomenų bazės sprendimas ir nėra priklausomas nuo kitų tuo pačiu metu vykdomų procesų
3. Srautinio apdorojimo architektūra „Apache Storm“ šiam uždaviniui užtikrina duomenų apdorojimo milisekundžių greičiu.

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*, p. 1789–1792. IEEE, 2016.
- [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://bytecontinuum.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 docs. 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.
- [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.
- [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.