VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Rodiklių duomenų kaupimas, transformavimas ir analizė, naudojant NoSQL duomenų bazę

(Storage, transformation and analysis of indicator data with the help of NoSQL database)

Kursinis darbas

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

Darbo vadovas: lekt. Andrius Adamonis (parašas)

Vilnius – 2018

TURINYS

Įvadas

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

Užduotys:

1. Analizės budu palyginti srautinį ir reliacinį duomenų apdorojimo sprendimą.
2. Sukurti testiniu duomenų generatorių.
3. Išmatuoti reliacinės duomenų bazės sprendimo pralaidumą.
4. Atlikti skirtingų srautinio duomenų apdorojimo sprendimo architektūrų analizę, ir pasirinkti vieną iš jų tyrimui.
5. Išmatuoti pasirinktos srautinio duomenų apdorojimo architektūros sprendimo pralaidumą.

1. Rodiklių duomenys

1.1. Apibrėžimas

Rodiklių duomenys - tai didelių duomenų tipas, kurį galima transformuoti ir analizuoti ir kuris yra sugrupuotas pagal rodiklius, pavyzdžiui: bazinė menėsio alga, mirusiųjų skaičius pagal mirties priežastis, krituliai per metus. Šie duomenys dažniausiai yra saugomi reliacinėse duomenų bazėse, kur užklausus vartotojui skaičiuojami apibendrinti rodikliai - sumos, vidurkiai ir kita statistika. Lietuvoje pagrindinis rodiklių duomenų bazės pavyzdys yra „Lietuvos statistikos departamento" duomenų bazė, kurios duomenis galima pasiekti `https://osp.stat.gov.lt/statistiniu-rodikliu-analize#/` puslapyje, kuris leidžia ieškoti duomenis 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", „Gander statistics" ir kitus[Ban18].

1.2. Charakteristikos

Apibrėžime minėjau, kad rodiklių duomenis yra didelių duomenų tipas, todėl galime jiems pritaikyti didelių duomenų charakteristikas ir apsibrėžti, kurios iš jų mums sudaro daugiausiai problemų. Šie iššūkiai apibrėžiami Gartner's Doug Laney pristatytu 3V modeliu[Lan01], kuris veliau buvo papildytas Bernard Marr iki 5V modelio[Mar14]:

- Tūris (angl. Volume). Apibrežia generuojamų duomenų kiekius. Didelių duomenų atveju yra šnekama apie duomenų kiekius, kuriuos yra sudetinga arba neįmanoma saugoti ir analizuoti tradicinėmis duomenų bazių technologijomis. Rodiklių duomenų kiekiai dažniausiai nesudaro problemos saugojant, tačiau didelė problema yra rodiklių duomenų analizė, kadangi tuos pačius duomenis reikia apdoroti pagal neapribotą skaičių skirtingų rodiklių.
- Greitis (angl. Velocity). Apibrežia greitį, kuriuo nauji duomenis yra generuojami. Rodiklių duomenų atveju, tai yra labai svarbu, kadnagi nauji duomenis, kurie gali tikti skirtingiems rodikliams yra generuojami visais laikais.
- Įvairovė (angl. Variety). Apibrežia duomenų tipus. Duomenys gali būti: strukturizuoti, nestrukturizuoti arba dalinai strukturizuoti[ZE+11]. Rodiklių duomenis dažinau yra strukturizuoti, todėl tai nėra aktualus iššūkis.
- Tikrumas (angl. Veracity). Apibrežia duomenų tesingumą ir kokybę. Pavyzdžiui, jeigu analizuotume „Twitter" socialinio tinklo žinučių turinį gautume daug gramatikos klaidų, naujadarų, slengo. Statistinio departamento atveju duomenys visada bus tvarkingi, kadangi tai dažniausiai yra duomenys surinkti iš dokumentų ir apklausų, o ne laisvo įvedimo.
- Vertė (angl. Value). Apibrežia duomenų ekonominę vertę. Rodiklių duomenys yra labai vertingi įstaigoms, nes dažniausiai tos įstaigos užsiema tik rodiklių duomenų kaupimų ir analizė, iš techninės pusės ši charakteristika yra svarbi iš tos pusės, kad duomenų apdorojimo ir kaupimo sprendimai labai stipriai daro įtaką įstaigos, kaupiančios rodiklių duomenis, ekonomikai. Taip pat šių duomenys ir jų analizė turi būti pasiekiama be prastovos laiko.

Pagal apibrėžtas charakteristikas matome, kad pagrindiniai rodiklių duomenų iššūkiai yra tūris, greitis ir vertė. Todėl mūsų bandomas sprendimas turi galėti greitai susidoriti su dideliu kiekių skirtingo pobūdžio duomenų ir turi sugebėti greitai atvaizduoti pokyčius atsiradus naujiems duomenims, taip pat turi būti įmanoma šį sprendimą paleisti į realią aplinką nepertraukiant įstaigos veiklą.

2. Rodiklių duomenų apdorojimo tipai

2.1. Reliacinės duomenų bazės duomenų apdorojimas

Aprašoma kaip generacija vyktu stored proceduros budu.

2.2. Srautinis duomenų apdorojimas

Srautinis duomenų apdorojimas (angl. Stream processing) - yra programavimo paradigma ekvivalenti seniai aprašytai duomenų tekmės programavimo (angl. dataflow programming) paradigmai[Bea15]. Duomenų tekmės programavimo paradigmos idėja, kad visa programa susidaro iš skirtingu modulių, kurie nepriklauso vienas nuo kito ir būtent tai leidžia sukonstruoti praleliai skaičiuojančias programas. Viena iš pirmųjų duomenų tekmės 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 išmokti žmonės, kurie nebuvo programavimo ekspertai[KLV61]. Michael Stonebraker 2005 metais apibrežė 8 taisyklės realaus-laiko srautinio duomenų apdorojimo architektūroms[SÇZ05]:

1 taisyklė: Duomenys turi judėti. Kad būtų užtikrinta žema latencija 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 suvaldyti srautinius duomenų netobulumus. Architektūra turi palaikyti galimybę nutraukti individualius skaičiavimus, tam kad neatsirastų blokuojančių operacijų. Taip pat ši architektūra turi sugebėti susidoroti su veluojančiomis žinutėmis, pratesiant 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 testi 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 sugebėjima 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.

Nors ir yra tokie aprašyti reikalavimai, tačiauyra labai nedaug srautinio apdorojimo architektūrų atitinkančių visas šias taisykles ir jei atitinka, tai kyla sudėtingumas. Išsamiau konkrekčios srautinio apdorojimo architektūros bus apžvelgtos 5 skyriuje.

2.3. Kiti duomenų apdorojimo tipai

2.3.1. Paketinis duomenų apdorojimas

Paketinis duomenų apdorojimas (angl. Batch processing) - didelių duomenų apdorojimo tipas, kai surinktas didelis duomenų kiekis nėra apdorojamas iš karto, o sukaupiamas ir vėliau apdorojamas visas iš karto. Mūsų reliacinės duomenų bazės apdorojimas naudoją primityvią paketinio duomenų apdorojimo versiją. Daug geriau yra žinomas kita paketinio duomenų apdorojimo architektūra - „Apache Hadoop". Ši architektūra naudoja „Google" sukurtą „Map-Reduce" konceptą[DG08], kuris dideli duomenų kiekį suskaido į mažus rinkinius ir daro apdorojima paraleliai ant visų rinkinių.[RHK+15] Ši architektūra mums netinka, nes mes norime rezultatą gauti tik įdėję naujų duomenų.

2.3.2. Lambda architektūra

Lambda architektūra - didelių duomenų apdorojimo architektūra naudojanti srautinio ir paketinio apdorojimo architektūrą. Su šią architektūra bandoma subalansuoti latencija, pralaidumą ir klaidų tolerancija, naudojant paketini apdorojimą tiksliems ir išsamiems duomenų pakentams ir tuo pačiu metu naudoti srautinį apdorojima greitai analizei[HKV14]. Tačiau ši architektūra turi vieną trukūma - norint naudoti šią architektūrą reikia palaikyti dvi skirtingas architektūras, kad kodas būtų atnaujinamas abiejose architektūrose vienu metu[Kre14]. Vienas pavyzdys tokios architektūros būtų - Kafka žinučių sistema siunčianti duomenis į „Apache Storm", kur duomenis apdorojami srautiškai, ir „Apache Hadoop", kur duomenis apdorojami paketiškai, ir rezultatus saugant skirtingose duomenų bazės lentelėse.

2.3.3. Kappa architektūra

Kappa architektūra - tai supaprastinta lambda architektūra, kuri vietoj paketinio apdorojimo naudoja papildomą srautinio adorojimo systemą. Pavyzdžiui yra vienas srautinio apdorojimo darbas, kuris veikia visą laiką ir atvaizduoja duomenis gyvai, ir yra kitas darbas kuris paleidžiamas kas kažkiek laiko, kuris susirenka per visą tą laiką susikaupusius duomenis ir praleidžia per save srautu. Kadangi kodai yra vienodi, tai nebeatsiranda anksčiau minetos problemos su skirtingų sistemų palaikymu[Kre14; Pat14].

3. Testinių duomenų generatorius

3.1. Apibūdinimas

Papasakoti čia apie Kafka

3.2. Paskirtis ir panaudojimo būdas

Papasakoti čia apie throtlinimo testavimą

4. Sprendimo naudojančio relaicinę duomenų bazę pralaidumo testas

4.1. Apibūdinimas

Kaip paruošiu sistema ir kodėl python turėtų būti pakankamai greitas sprendimas, kad netrukdytų rezultatams

4.2. Testavimo rezultatai

Oh shit its slow.

5. Srautinio apdorojimo architektūros

Šiame skyriuje palyginsiu skirtingas srautinio apdorojimo architektūrų privalumus ir trūkumus ir pasirinksiu vieną su kuria sukursiu sprendimą rodiklių duomenų apdorojimui.

5.1. „Apache Storm”

5.2. „Apache Spark”

5.3. „Apache Flink”

5.4. Apibendrinimas

Parašau kuri pasirenku ir kodėl

6. Srautinio duomenų apdorojimo sprendimo pralaidumo testas

6.1. Apibūdinimas

Dar čia parašysiu kokius išbandžiau variantus architektūros. Kaip testuosiu, su visais spoutais ir boltais ir t.t.

6.2. Rezultatas

Koks pralaidumas buvo, kaip sunku buvo paruošti sistemą, kiti rodikliai, kur stabdė.

7. Experimento apibendrinimas

7.1. Rezultatai

Palyginsiu srautinės ir reliacinės sprendimų pralaidumus.

7.2. Eksperimento išvados

Ką mes iš tu testu galime pasakyti.

Rezultatai ir išvados

Darbo rezultatai:

- Išnagrinėti rodiklių duomenų analizės būdai pagal jų privalumus ir trūkumus.
- Sukurtas testinių duomenų generatorius, kurio pagalba buvo vykdomas pralaidumo testavimas.
- Atliktas pralaidumo testas Micorsoft SQL Express duomenų bazei su sukurtu testiniu duomenų generatorium ir tarpine Python aplikacija.
- Išanalizuoti skirtingos srautinio apdorojimo architektūros ir pasirinkta tinkamiausia sprendimo kurimui.
- Sukurtas sprendimas su pasirinkta srautinio apdorojimo architektūra.
- Atliktas pralaidumo testas sukurtam srautinio apdorojimo sprendimui.
- Palyginti sprendimų pralaidumo testų rezultatai.

Darbo išvados:

-
-

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.

[DG08] Jeffrey Dean ir Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[HKV14] Zirije Hasani, Margita Kon-Popovska ir Goran Velinov. Lambda architecture for real time big data analytic. *ICT Innovations*, 2014.

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

[Kre14] Jay Kreps. Questioning the lambda architecture. *Online article, July*, 2014.

[Lan01] Doug Laney. 3d data management: controlling data volume, velocity and variety. *META Group Research Note*, 6(70), 2001.

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

[Pat14] Milinda Pathirage. What is kappa architecture. `http://milinda.pathirage.org/kappa-architecture.com/`, 2014.

[RHK+15] Shyam R, Barathi Ganesh Hullathy Balakrishnan, Sachin Kumar S, Prabaharan Poornachandran ir Soman Kp. Apache spark a big data analytics platform for smart grid. 21:171–178, 2015-12.

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

[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.