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. Apsibrėžti skirtingus duomenų apdorojimo būdus ir jų tinkamumą rodiklių duomenų apdorojimui.
2. Sukurti testinių 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.
6. Palyginti testavimo rezultatus ir gauti išvadas.

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. Duomenų apdorojimo tipai

2.1. 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].

Kad apžvelgti modernias srautinio duomenų apdorojimo architektūras reikia apsibrėžti srautinio apdorojimo sistemų galimybes. 2005 metais Michael Stonebraker apibrežė 8 taisyklės realauslaiko srautinio duomenų apdorojimo architektūroms[SÇZ05]:

1 taisyklė: Duomenys turi judėti. Kad būtų užtikrinta žemas uždelstumas 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 daugu-ma 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 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.

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 tesingas 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, konkrekčios srautinio apdorojimo architektūros yra apžvelgiamos ir lyginamos 3 skyriuje.

2.2. Reliacinės duomenų bazės duomenų apdorojimas

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 paskirstytos atitinkamai:

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 varikliuse, 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 dažnai naudojamos ir duomenų apdorojimui. Kadangi reliacinė duomenų bazė jau egzistuoja, reiškia įmonei nereikia leisti papildomų lėšu: išanalizuoti kitokios sistemos tinkamumą užduočiai, sukurti sprendimą, palaikyti naują sistemą, pasisiamdyti naują žmogų mokanti dirbti su šia sistema arba apmokyti esamą.

Reliacinių duomenų bazių duomenų apdorojimo būdas yra išsaugotos procedūros(angl. stored procedures), kurios aprašomos SQL kalba ir gali apdoroti duomenis tiesiai iš duomenų bazės naudojant reliacinę matematiką. Tačiau jeigu pažiuresimę į išsaugota procedurą, kaip duomenų apdorojimo architektūrą pagal 2.1 skyriuje apibrėžtas 8 taisykles, pastebėsime, kad ji neatitinką: 1-os taisyklės: Duomenys turi judėti. Išsaugota procedūra yra leidžiama tik vartotojui užklausus, todėl šis reikalavimas yra neišpildytas, ir 7-os taisyklės: Architektūra turi užtikrinti sugebėjima paskirstyti sistemos darbus automatiškai. Didžioji dalis reliacinių duomenų bazių nepalaiko horizontalų

plečiamumą[Cat11; Ram16] ir todėl vieną sitemą gali apdoroti tik pas ją esančius duomenis. Svarbiausiai, išsaugota procedūra negali apdoroti greitai naujų duomenų jeigu 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 taisyklė, ir ko pasekoje reliacinis sprendimas analytiškai yra mažiau tinkamas rodiklių duomenų problemai.

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 uždelstumą, 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, todėl nebeatsiranda anksčiau minetį iššūkiai su skirtingų sistemų palaikymu[Kre14; Pat14].

3. Srautinio apdorojimo architektūros

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

- Pristatymo semantika (angl. delivery semantics) - apibrežia 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 duplikatų.
  - 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) - apibrežia kiek laiko užtruks įvykdyti kažkokį veiksmą arba gauti rezultatą.
- Pralaidumas (angl. Throughput) - apibrežia kiek pavyks įvykdyti operacijų per tam tikrą laiko tarpą.
- Abstrakcijos lygis (angl. Abstraction) - apibrežia kokio lygio programavimo sąsają pateikia architektūra.

3.1. Pristatymo semantika

3.2. Uždelstumas

3.3. Pralaidumas

3.4. Abstrakcijos lygis

Pats populiariausias srautinio apdorojimo architektūrų programos pavyzdys yra žodžių skaičiuoklė, kurios tikslas suskaičiuoti kiek kartu pasikartojo tas pats žodis, per visą programos veikimo laiką. Todėl abstrakcijos lygį geriausia palyginti šios programos kodo pavyzdžiais[Zap16]:

```
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("spout", new RandomSentenceSpout(), 5);
builder.setBolt("split", new Split(), 8).shuffleGrouping("spout");
builder.setBolt("count", new WordCount(), 12).fieldsGrouping("split", new Fields("word"));
...
Map<String, Integer> counts = new HashMap<String, Integer>();

public void execute(Tuple tuple, BasicOutputCollector collector) {
  String word = tuple.getString(0);
  Integer count = counts.containsKey(word) ? counts.get(word) + 1 : 1;
  counts.put(word, count);
  collector.emit(new Values(word, count));
}
```

1 pav. „Apache Storm" žodžių skaičiavimo programos kodo pavyzdys

„Apache Storm" parašytos programos yra žemo lygio abstrakcijos. 1 pavyzdyje matome labai sumažintą programos pavyzdį. Kadangi tai yra žemo lygio programa mes turime apsirašyti visas srauto apdorojimo stadijas pažingsniui, tai yra: setSpouts(..), kur nustatoma duomenų įeiga ir koks bus paralelizmas, setBolt(..), kur nustatatomi apdorojimo moduliai, kokius duomenis gaus iš prieš tai buvusio modulio ir paralelizmas. Žemiau, execute() metodas aprašo, kaip gali atrodyti apdorojimo modulis, kuris suskaičiuoja kiek skirtingų žodžių pro jį praėjo. Šios architektūros programų kūriamo laikas užtruks ilgiau negu kitoms architektūroms su aukštu abstrakcijos lygių.

```
SparkSession spark = SparkSession
    .builder()
    .appName("WordCount")
    .getOrCreate();

JavaRDD<String> lines = spark.read().textFile(...).javaRDD();
JavaRDD<String> words = lines.flatMap(s -> Arrays.asList(SPACE.split(s)).iterator());
JavaPairRDD<String, Integer> ones = words.mapToPair(s -> new Tuple2<>(s, 1));
JavaPairRDD<String, Integer> counts = ones.reduceByKey((i1, i2) -> i1 + i2);
List<Tuple2<String, Integer>> output = counts.collect();
for (Tuple2<?,?> tuple : output) {
    System.out.println(tuple._1() + ": " + tuple._2());
}
spark.stop();
```

2 pav. „Apache Spark" žodžių skaičiavimo programos kodo pavyzdys

„Apache Spark" parašytos programos yra aukšto lygio abstrakcijos. 2 pavyzdyje matome realu beveik išbaigtą programos pavyzdį. Programa aprašoma funkciškai, todėl kodo rašymas trunka daug trumpiau ir tokį kodą daug patogiau skaityti.

```
final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

env.getConfig().setGlobalJobParameters(params);
DataStream<String> text = env.fromElements(...);
DataStream<Tuple2<String, Integer>> counts =
    text.flatMap(new Tokenizer())
        .keyBy(0).sum(1);
counts.print();
env.execute("Streaming WordCount");
```

3 pav. „Apache Flink" žodžių skaičiavimo programos kodo pavyzdys

„Apache Flink" parašytos programos yra aukšto lygio abstrakcijos. 3 pavyzdyje matome pilnai veikiančios programos pavyzdį. „Apache Flink" architektūra pati užsiema distribucija, todėl programuotojui lieka tik parašyti veikianti kodą, o sistema pati susitvarkys su paralelizmu.

3.5. Apibendrinimas

Parašau kuri pasirenku ir kodėl

4. Testinių duomenų generatorius

4.1. Apibūdinimas

Papasakoti čia apie Kafka

4.2. Paskirtis ir panaudojimo būdas

Papasakoti čia apie throtlinimo testavimą

5. Sprendimo naudojančio relaicinę duomenų bazę pralaidumo testas

5.1. Apibūdinimas

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

5.2. Testavimo rezultatai

Oh shit its slow.

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.

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

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

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

[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.

[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.

[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.

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

[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.

[Zap16] Petr Zapletal. Comparison of apache stream processing frameworks: part. https://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-1, 2016.

[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.