VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Baigiamasis bakalauro darbas

NoSQL duomenų bazių analizė

(NoSQL Database Analysis)

Atliko: 4 kurso, 1 grupės studentas
Artur Grudinskij (parašas)
Darbo vadovas:
asist. Liutauras Ričkus (parašas)
Recenzentas:
asist. Giedrius Graževičius (parašas)

Vilnius
2016

Turinys

Įvadas

Visos šiuolaikinio žmogaus gyvenimo sritys yra priklausomos nuo technologijų, o didžioji dalis tų technologijų naudoja duomenų bazes savo funkcionalumui realizuoti, todėl galima teigti, jog duomenų bazės yra neatskiriama žmogaus kasdieninio gyvenimo dalis [Jef11]. Duomenų bazės yra skirstomos į reliacinio (SQL) ir nereliacinio (NoSQL) modelio. Šis suskirstymas egzistuoja dėl to, kad kiekvienas iš šių modelių prioretizuoja skirtingus duomenų bazių aspektus, dėl ko buvo, yra ir bus daugybė situacijų kuomet vienos rūšies produktai pasirodys žymiai geriau negu kitos; ir kadangi labiausiai paplitęs technologinis sprendimas daugeliui verslų yra internetinės programos, NoSQL duomenų bazės, anot daugelio duomenų bazių projektuotojų, turi pranašumą prieš savo oponentą šioje srityje. Tačiau tokia nuomonė jau yra pasenusi, nes reliacinio modelio duomenų bazės niekados nenustojo vystytis, ir ganėtinai gerai prisitaikę prie kintančių technologijų.

Norint padaryti teisingą pasirinkimą, kai yra renkamasi kokios rūšies ir atitinkamai tos rūšies tipo duomenų bazę panaudoti savo projekte, derėtų neatsižvelgti į nusistovėjusius stereotipus ir savarankiškai išbandyti turimas pasirinktis taikymo sąlygose.

Šio darbo tikslas yra palyginti tarpusavyje kiekvieno NoSQL tipo ir reliacinio modelio duomenų bazės.

Tikslui pasiekti keliami uždaviniai yra:

- Nustatyti duomenų bazių aspektus ir kriterijus pagal kuriuos jas galima palyginti;
- Pasirinkti konkrečias duomenų bazes ir išnagrinėti pagal pasirinktus kriterijus;
- Palyginti duomenų bazes tarpusavyje pagal nustatytus kriterijus.

Atlikus uždavinius turėtų tapti aišku kokius privalumus ir trūkumus turi NoSQL duomenų bazės, koks yra skirtumas tarp jų tipų bei reliacinių duomenų bazių ir kuomet derėtų rinktis NoSQL realizuojant savo technologinį sprendimą, o kada būtų sumanu rinktis reliacinio modelio duomenų bazes.

1. Duomenų bazių analizė

Kadangi egzistuoja nė viena rūšis NoSQL duomenų bazių, visos iš kurių drastiškai skiriasi nuo reliacinių duomenų bazių, iškyla iššūkis kaip galima būtų išanalizuoti ir po to palyginti visas jas tarpusavyje. Nors duomenų bazių rūšys iš pirmo žvilgsnio yra skirtingos, jų unikalumą galima paaiškinti išnagrinėjus jų ypatybes ir funkcinius prioritetus. Šiame darbe buvo analizuojamos ypatybės būdingos visoms duomenų bazėms.

1.1. Duomenų bazių tipai

Tarp duomenų bazių projektuotojų nėra visuotinai sutarta į kokius tipus yra skirstomos NoSQL duomenų bazės, todėl yra poreikis apibrėžti kokio suskirstymo yra laikomasi šiame darbe ir kaip yra suprantamas kiekvienas iš jam priklausančių duomenų bazių tipų, įskaitant ir reliacines duomenų bazes, kadangi apie jas šiame darbe yra nemažai kalbama.

1.1.1. NoSQL duomenų bazės

NoSQL buvo ir yra kuriamos remiantis daugeliu skirtingų požiūrių ir idealų dėl nereliacinių duomenų bazių, todėl egzistuoja daugelis klasifikacijų [SF12].

Kiekviena klasifikacija buvo kuriama atsižvelgiant į ją sudarančio žmogaus tikslą, kurio esmė yra pristatyti NoSQL akcentuojant tam tikra pasirinktą aspektą, pavyzdžiui:

- Stephen Yen pristatė devynias NoSQL klases, sakydamas, jog duomenų bazių rūšys su didesniu pasirinkimu triumfuos [Yen09];
- Rick Cattell suskirstė nereliacines duomenų bazes pagal duomenų modelį [Cat11];
- Ken North suskirstė nereliacines duomenų bazes, pagristas debesų kompiuterija, pagal duomenų modelį [Nor09];
- Ben Scofield pristatė bendrai NoSQL ir kokias sritis jos apima [Sco09].

Iš paminėtų klasifikacijų šiam darbui geriausiai tinka Ben‘o Scofield‘o, nes joje yra bendrai pristatytos NoSQL klasės, pagal kurias galima suklasifikuoti dauguma prie NoSQL priskiriamų duomenų bazių:

- žodyno formato duomenų bazės (angl. key-value stores);
- dokumentų talpyklos (angl. document stores);
- grafo formato duomenų bazės (angl. graph databases);
- stulpelių formato duomenų bazės (angl. column stores).

1.1.1.1. Žodyno formato duomenų bazės

Žodyno formato duomenų bazės, dažniausiai vadinamos rakto ir reikšmės duomenų bazėmis, pagal savo pagrindą yra panašios į maišos lentelės (angl. Hash Table) ir asociatyvius

masyvus. Talpinami duomenys turi poros struktūrą: raktas ir reikšmė. Kadangi žodynai naudoja tą patį principą, jos ir yra vadinamos žodyno formato duomenų bazėmis ([MK14], [Nor09], [Tiw11]).

Žiūrint iš funkcionalumo pusės, tai yra paprasta struktūra, nes ji turi tik tris pagrindines operacijas:

- Pridėti reikšmę pagal raktą;
- Išgauti reikšmę pagal raktą;
- Pašalinti reikšmę ir raktą.

Reikia įsidėmėti, kad tai yra tik pagrindinės operacijos, kurių sąrašas yra praplėstas atitinkamuose produktuose, pavyzdžiui, daugumą žodyno formato duomenų bazių suteikia galimybė saugiai keisti poros reikšmė ir/arba rakto reikšmė. Nėra poreikio įgyvendinti jokios struktūrizuotos užklausų kalbos navigacijai tarp duomenų bazės elementų, nes užtenka ir siūlomų operacijų ir kadangi duomenų bazė „supranta“ kokio duomenų tipo yra rakto atstovaujama reikšmė, kai kurie šios rūšies produktai siūlo papildomą funkcionalumą paremta šitomis žiniomis ([Cel14], [Vai13]).

Rakto reikšmė yra unikali simbolių seka, kurios dydis yra ribojamas atitinkamu kiekiu baitų, priklausomai nuo pačios duomenų bazės. Ką simbolizuoja raktas nusako jo atstovaujama reikšmė, bei pati duomenų bazė. Pavyzdžiui [MK14]:

- rakto reikšmė yra žodis, o jo atstovaujama reikšmė yra to žodžio reikšmė;
- rakto reikšmė yra pilnas rinkmenos pavadinimas, o jo atstovaujama reikšmė yra pati rinkmena;
- rakto reikšmė yra kelias į direktoriją kur yra laikoma rinkmena, o jo atstovaujama reikšmė yra pati rinkmena;
- rakto reikšmė yra nuoroda į internetinį puslapį, o jo atstovaujama reikšmė yra to puslapio žymėjimo kalbos, kuria jis parašytas, kodas.

Nesunku pastebėti, kad rakto atstovaujamos reikšmės tipas priklauso nuo pačios duomenų bazės ir joje norimų saugoti duomenų, tačiau kadangi reikšmė nėra analizuojama saugojimo metu, dažniausiai nėra taikoma jokių griežtų apribojimų reikšmei iš pačios duomenų bazės pusės.

Kadangi raktas ir jo atstovaujama reikšmė turi tiesioginį ryšį, reikšmės išgavimas yra greitas, neatsižvelgiant į duomenų bazėje esamų įrašų kiekį [MK14]. Tačiau šita kaip ir kitos operacijos yra galimos tik žinant raktą, todėl vieninteliai bendri apribojimai yra kad būtų raktas ir kad jis atitiktų nustatytus jo reikšmei reikalavimus.

1.1.1.2. Dokumentų talpyklos

Kaip galima spręsti iš pavadinimo, pagrindinė dokumentų talpyklos idėja yra operacijos su dokumentais – pusiau struktūrizuotais duomenimis [Vai13]. Dokumentų talpyklos saugo ir išgauna dokumentus, bei valdo su jais susijusią informaciją.

Dokumentų talpyklos yra laikomos kitu loginiu žingsniu po žodyno formato duomenų bazių, nes abi duomenų bazių klasės saugo duomenis pagal rakto ir reikšmės principą. Tačiau dokumentų talpyklose laikmeną galima išgauti ne tik pateikus jos raktą, tačiau galima į ja žiūrėti kaip į dokumentą, arba duomenų rinkinį, ir pagal jo turinį vykdyti užklausas. Šita savybė leidžia atrinkti dokumentų rinkinius pagal atitinkamą turinį arba jų metaduomenis, kas reiškia, kad dokumentų turinys yra indeksuojamas, todėl dokumentų talpyklos turi atitinkamus reikalavimus vidinių duomenų struktūrai priklausomai nuo indeksavimo realizacijos [Vai13].

Kadangi dokumentų talpyklų klasė naudoja panašų principą kaip ir žodyno formato klasė, galimybė išgauti duomenis pagal kūrimo metu pateikta raktą išliko, bet galima ir nepateikti rakto (jis bus automatiškai sugeneruotas). Šita savybė ir yra esminė priežastis, kodėl dokumentų talpyklos yra vadinamos žodyno formato klasės tęsėju, kadangi jas galima naudoti kaip ir pastarąsias, bet ne atvirkščiai [MK14].

Dar vienas pranašumas prieš žodyno formato klasės produktus būtų tame, kad pridėjus naują reikšmę prie jau egzistuojančio rakto, daugumoje šitos klasės duomenų bazėse šito rakto reikšmė saugotų reikšmių rinkinį, į kurį būtų patalpintos ir sena ir naujai pridėta reikšmės, o ne tik paskutinė pridėta. Tai nėra automatinis numatytas funkcionalumas skirtas tokio pobūdžio situacijoms, o naudotojui yra pranešama apie reikšmių konfliktą per duomenų bazės sąsają ir jis pasirenka kurias iš jų palikti ([MK14], [Tiw11]).

1.1.1.3. Grafo formato duomenų bazės

Grafo formato duomenų bazės yra ypatinga NoSQL rūšis, kurioje dėmesys yra akcentuojamas į ryšius tarp saugomų duomenų, kitais žodžiais santykius. Šita duomenų bazių rūšis saugo duomenis grafo pavidalu, kur mazgai yra duomenys, orientuotos briaunos – santykiai tarp jų, ir briaunų ypatybės – santykių pobūdžiai [Vai13].

Pagrindinė šitos duomenų klasės paskirtis yra analizuoti arba ieškoti ryšius tarp duomenų atsižvelgiant į vieno ar daugiau struktūros elementų ypatybes. Kadangi šita duomenų bazių klasė remiasi grafų teorija, per užklausas galima spręsti tokias problemas kaip trumpiausias kelias tarp dviejų mazgų, mazgų paieška pagal atitinkamas elementų ypatybes ir taip toliau.

Užklausų rezultatai yra mazgų rinkiniai ir ryšiai tarp jų, kas yra vaizduojama grafiniu pavidalu [MK14].

1.1.1.4. Stulpelio formato duomenų bazės

Stulpelio formato duomenų bazės yra savotiška reliacinio modelio priešybė. Šitos duomenų bazių rūšies modelis yra dvimatis asociatyvus masyvas, kas iš pirmo žvilgsnio gali priminti lentelę ([SF12], [Tiw11]). Tačiau šiuo atveju, duomenys yra saugomi ir valdomi stulpeliais, o ne eilutėmis. Visa struktūra yra vienas stulpelis, sudarytas iš atitinkamo kiekio asociatyvių masyvų. Raktinės reikšmės yra tų masyvų pradžioje, o antriniai duomenys yra saugomi po jų. Šios struktūros esminis privalumas atsiskleidžia išnagrinėjus atvejį, kai antriniai duomenys arba jų dalys yra nežinomos. Reliaciniu modeliu besiremiančioje duomenų bazėje tų duomenų vietoje būtų tušti langeliai, o stulpelio formato duomenų bazėse, kadangi įrašas yra asociatyvaus masyvo struktūros, toje vietoje nieko nebus, net „langelio“, kas reiškia, kad šitos duomenų bazės rūšies produktų naudotojai moka lygiai už tiek vietos duomenų bazėje, kiek išnaudoja [Tiw11].

Šita duomenų bazių rūšis yra vadinama stulpelio formato, nes kai naudotojui reikia nagrinėti kelių stulpelių reikšmes daugelyje eilučių, stulpelio formato duomenų bazėje visą tokio pobūdžio informacija yra pavienis informacijos saugojimo vienetas, kadangi viskas yra saugoma viename stulpelyje [SF12]. Iš kitos pusės, kai naudotojas nori pridėti naują stulpelį, jam nereikia nerimauti, kad gali būti sugadinta duomenų bazės struktūra, bei nebūtina užpildyti šį stulpelį tik dėl to, kad jau egzistuoja kiti stulpeliai užpildyti reikšmėmis [Vai13].

1.1.2. Reliacinės duomenų bazės

Reliacinėse duomenų bazėse duomenys yra saugomi sugrupuotais įrašais, kurie sudaro lentelės. Duomenų bazė, kurioje yra saugomi duomenys lentelėmis, yra vadinama reliacinė. Reliacinės duomenų bazės duomenų valdymui naudoja SQL užklausų kalbą. Užklausos, kurios keičia duomenų bazėje saugomus duomenis, yra vykdomos transakcijomis, kurių veikimą nusako ACID (detaliai aprašyta skyriuje 1.2.4) pastovumo modelis. ([Dat15], [Cra16])

1.2. Analizės kriterijai

Kadangi yra nagrinėjamos skirtingų klasių ir rūšių duomenų bazės, vertinimo kriterijai buvo parinkti tokie, kad galima būtų pagal juos išgauti palyginus objektyvius rezultatus.

Analizė susideda iš sekančių dalių:

1) Dabas su duomenų baze;

2) Duomenų bazės struktūros lankstumas;
3) Duomenų bazių jungumas;
4) Pastovumo modelis;
5) Operacijų su duomenis vykdymo greitis.

Kiekvienas kriterijus, išskyrus 4) ir 5), yra vertinamas pagal sekančią skalę ([Geo15], [lentelė 1.2]):

Paaiškinimas	Vertinimas
Atitinka lūkesčius	1
Viršija lūkesčius	2
Išskirtinis	3

Lentelė 1.2. *Analizėje panaudotų subjektyvių rodiklių apibrėžimas.*

Ši skalė neapima „pastovumo modelio“ kriterijaus, nes analizės eigoje jį nagrinėjant yra nustatoma kurį iš dviejų pastovumo modelių naudoja duomenų bazė („ACID“ ar „BASE“ (detaliai aprašyta skyriuje 1.2.4)) ir kaip. Tuo tarpu „operacijų su duomenimis vykdymo greičio“ vertinimo reikšmės atitinka gautus konkrečius skaitinius vykdymo greičius, kuriuos galima tiesiogiai tarpusavyje palyginti.

1.2.1. Darbas su duomenų baze

Darbas su duomenų baze susideda iš dviejų etapų – diegimas ir naudojimasis, kiekvienas iš kurių yra ganėtinai svarbus vartotojui.

1.2.1.1. Duomenų bazės diegimo efektyvumas

Pažintis su kiekviena programine įranga prasideda nuo diegimo proceso. Jo paskirtis yra supažindinti naudotoją su jo teisėmis, bei pačia programine įranga ir atitinkamai ją įdiegti atsižvelgiant į naudotojo poreikius. Naudotojui svarbiausia yra kiek pastangų visas procesas reikalauja ir kaip efektyviai bus įdegta programinė įranga, kitais žodžiais, kuo mažiau reikės padaryti papildomų veiksmų, kad paleisti programą po diegimo, tuo efektyvesnis yra diegimas.

Ypatybės ir jas atitinkančios vertinimo skalės reikšmės:

- Atitinka lūkesčius (1) – Reikalingos darbui rinkmenos yra nukopijuojamos arba sugeneruojamos į nurodytą direktoriją, diegimo įrašas yra patalpinamas į operacinės sistemos registrą ir pati duomenų bazė yra paruošiama darbui.
- Viršija lūkesčius (2) – viskas kas buvo paminėta punkte „Atitinka lūkesčius (1)“ ir papildomai yra koks nors prasmingas diegimo aspektas vartotojo patogumui,

pavyzdžiui, pasirinkimas tarp diegėjo su grafine sąsaja ir diegimo per komandinę eilutę.

- Išskirtinis (3) – viskas kas buvo paminėta punkte „Atitinka lūkesčius (1)“ yra padaroma kokiu nors išskirtiniu būdu arba yra koks nors išskirtinis funkcionalumas palengvinantis diegimo procesą arba ateities naudojimąsi pačia programine įranga, pavyzdžiui, kuomet yra numatomas ilgas diegimas į diegėją galima įterpti tekstinį arba interaktyvų naudojimosi vadovą, kad galima būtų laukiant prasmingai praleisti laiką.

1.2.1.2. Naudojimosi duomenų baze sudėtingumas

Sekantis žingsnis po sėkmingo programinės įrangos diegimo yra jos naudojimas. Duomenų bazių sąsajos būna komandinės eilutės ir grafinės sąsajos formatų. Akivaizdžiai, grafinė sąsaja turi pranašumą prieš komandinę eilutę, kadangi naudotojui mažiau reikia įsiminti komandų, vietoje ko jam užtenka sąveikauti su sąsajos elementais, kad palengvinti sau darbą. Tačiau pilnai išvengti komandų įvedimo ne kaip nesigaus, nes visas duomenų bazių funkcionalumas yra įgyvendinamas per tos duomenų bazės užklausų kalbą, kuri lyginant su kitomis užklausų kalbomis skirsis pagal taikymo sudėtingumą. Taigi, naudotojui dirbant su duomenų bazę yra svarbiausia, kad būtų naudinga sąsaja ir užklausų kalba, per kurią jis efektyviai galėtų įgyvendinti norimus programinius sprendimus.

Ypatybės ir jas atitinkančios vertinimo skalės reikšmės:

- Atitinka lūkesčius (1) – užklausų kalba nėra sudėtinga arba yra pateikiami naudojimosi vadovai/dokumentacija, su kurių pagalba galima išmokti užklausų kalbą norimu lygmeniu.
- Viršija lūkesčius (2) - užklausų kalba nėra sudėtinga ir yra pateikiami naudojimosi vadovai/dokumentacija, su kurių pagalba galima išmokti užklausų kalbą norimu lygmeniu.
- Išskirtinis (3) – yra patenkinami kriterijai bent punkto „Atitinka lūkesčius (1)“ ir yra koks nors išskirtinis naudojimosi aspektas, kuris palieka labai įsimintiną įspūdį.

1.2.2. Duomenų bazės struktūros lankstumas

Kiekvienais metais technologijų taikymo sritis sparčiai didėja ir apima vis daugiau žmogaus gyvenimo aspektų, dėl ko generuojamų duomenų pobūdis, dydis ir struktūra nėra panašūs ir kartais yra ganėtinai nepastovūs. Tokius duomenis yra sutarta vadinti didžiaisiais duomenimis (angl. Big Data). Duomenų bazės skirtos darbui su tokias duomenimis privalo užtikrinti, kad juos galima būtų efektyviai valdyti. Kadangi didžiųjų duomenų struktūra

kartais kinta, turi būti numatyta ir galimybė reaguoti į tuos pokyčius nekenkiant jau patalpintiems duomenų įrašams. Šioje analizės dalyje bus nagrinėjama kaip duomenų bazės sprendžia struktūros lankstumo problemą iškilus poreikiui. ([Buc15], [Tiw11], [Vai13])

Ypatybės ir jas atitinkančios vertinimo skalės reikšmės:

- Atitinka lūkesčius (1) – galima pridėti/panaikinti duomenų įrašo elementą.
- Viršija lūkesčius (2) – viskas kas buvo paminėta punkte „Atitinka lūkesčius (1)“ ir papildomai galima pervadinti duomenų įrašo elemento pavadinimą.
- Išskirtinis (3) – viskas kas buvo paminėta punkte „Atitinka lūkesčius (1)“ ir galima pakeisti duomenų įrašo elemento duomenų tipą.

1.2.3. Duomenų bazių jungimas

Geroji praktika sako, kad vienoje duomenų bazėje, arba lentelėje (priklausomai nuo duomenų bazių tipo), turėtų būti tik to pačio tipo duomenų įrašai, o ne skirtingi, nes taip žymiai paprasčiau juos valdyti [Buc15]. Tačiau kartais atsiranda poreikis dirbti su duomenų įrašais iš kelių skirtingų duomenų bazių. Tokiam poreikiui patenkinti yra naudojami JOIN tipo įrankiai. JOIN - tai bendras pavadinimas duomenų bazės įrankių grupės, įgalinančios naudotoją vienoje užklausoje valdyti kelių duomenų bazių duomenis. Reliacinėse duomenų bazėse šis įrankis taip ir yra vadinamas, tuo tarpu kiekviena NoSQL duomenų bazė turi savo JOIN tipo operacijas, kurių realizacija priklauso nuo pačios duomenų bazės struktūros, dėl ko ir pavadinimas skiriasi – pas kiekvieną savo.

Šioje analizės dalyje bus analizuojamas JOIN tipo operacijų (jei tokių yra) suteikiamas funkcionalumas, iš ko turėtų paaiškėti duomenų bazės struktūros ypatybės ir duomenų valdymo būdai atsižvelgiant į jas.

Ypatybės ir jas atitinkančios vertinimo skalės reikšmės:

- Atitinka lūkesčius (1) – duomenų bazė neturi JOIN tipo operacijų, nes tokio funkcionalumo nereikia/negalima pritaikyti.
- Viršija lūkesčius (2) – duomenų bazė neturi JOIN tipo operacijų, tačiau yra palaikomas kelių užklausų rezultatu apjungimas.
- Išskirtinis (3) – duomenų bazė turi JOIN tipo operacijų.

1.2.4. Pastovumo modelis

Pastovumo modelis (angl. Consistency Model) – tai taisyklių rinkinis, kuris nusako duomenų bazės savybes [Mic08].

Dauguma šiame darbe nagrinėtų duomenų bazių naudoja vieną iš dviejų modelių - ACID arba BASE. Abudu pavadinimai yra akronimai sudaryti iš nusakytų taisyklių pavadinimų pirmųjų raidžių.

- ACID modelis – tai taisyklių rinkinis, kurio laikymasis užtikrina duomenų bazės patikimumą.
    - Atomiškumas (angl. **A**tomicity) – transakcijos vykdymas baigiasi sėkme arba nesėkme. Sėkmės atveju yra pilnai įvykdoma transakcija, o nesėkmės – visi pokyčiai padaryti transakcijos vykdymo eigoje yra atšaukiami, kitais žodžiais – arba viskas arba nieko. Šiame kontekste transakcija tai operacijų seka [Mic08].
    - Pastovumas (angl. **C**onsistency) – transakcijų vykdymas nesutrikdys duomenų bazės veikimo. Taip pat yra užtikrinama, kad transakcijos, prieštaraujančios duomenų bazėje nusakytoms taisyklėms ir/arba apribojimams nebus įvykdytos [Cel14].
    - Izoliacija (angl. **I**solation) – yra užtikrinama, kad kelių transakcijų, tuo pačiu laiku, padaryti pokyčiai vienas kitam neprieštarautų – priklausomai nuo realizacijos, yra užskaitomi tik pokyčiai atlikti pirmosios transakcijos, arba paskutinės [Mic08].
    - Ilgalaikiškumas (angl. **D**urability) – sėkmingai atlikus transakciją, jos padaryti pokyčiai yra išsaugomi [Mic08].
- BASE modelis, dar vadinamas galutinio pastovumo modeliu, kurio taisyklės sudaro paskirstytų duomenų bazių veikimo pagrindą.
    - Bazinis prieinamumas – yra prioretizuojamas transakcijų vykdymas vietoje duomenų bazės pastovumo [MK14].
    - Negriežta būsena – duomenų bazės būsena gali kisti, net nevykdant operacijų [MK14].
    - Galutinis pastovumas – duomenų bazės būsena galiausiai (praėjus nustatytam laikotarpiui) tampa pastovi ([Cel14], [MK14]).

Šioje analizėje yra nagrinėjama koks yra naudojamas modelis ir kaip yra atsižvelgiama į jo taisykles.

1.2.5. Operacijų su duomenimis vykdymo greitis

Duomenų bazės yra dažnai lyginamos pagal duomenų įrašymo, nuskaitymo ir pakeitimo vykdymo greičius, ką galima apibendrinti kaip komandų vykdymo greičio rodiklį (toliau -

greičio rodiklis) [EP15]. Šiame darbe bus naudojamas rodiklis nusakantis per kiek sekundžių buvo įvykdytas nurodytas kiekis operacijų. ([Buc15], [Vai13])

Tam, kad objektyviai ištirti duomenų bazę yra naudojama visuma sudėtinių greičio rodiklių, kurie yra matuojami taikant komandų kombinacijas, besilaikant atitinkamų duomenų bazės apkrovų. Tas yra daroma su tikslu ne tik tam, kad nustatyti rodyklių pokyčių tendencijas, besikeičiant komandų kiekiui, bet ir kad galima būtų įžvelgti kaip atitinkamos komandos įtakoja bendrą vykdymo laiką, kai jos yra naudojamos lygiagrečiai su kitomis komandomis.

Šiame darbe bus tiriami sekančių komandų rodikliai:

- Įrašymo komanda – duomenų patalpinimas į duomenų bazę;
- Nuskaitymo komanda – duomenų išgavimas iš duomenų bazės.

Tyrimui buvo atrinktos šios komandos, nes tokie funkcionalumo primityvai yra įtraukti į didžiąja dalį duomenų bazių. Pasitaiko senesnių duomenų bazių, kuriuose nėra atskiros pakeitimo komandos, tačiau tą patį funkcionalumą galima gauti pakartotinai įrašius duomenis naudojant tą patį raktą koks ir buvo pirmo įrašymo metu, kas iš esmės užtrunka apytiksliai tiek pat laiko, todėl šiame darbe bus daroma prielaida, kad duomenų įrašymo ir pakeitimo komandos užtrunka tiek pat laiko ir jos bus vadinamos tiesiog įrašymo komandomis.

Tyrimas bus atliekamas naudojant sekančius rodiklius ([Coo10], [EP15]):

Pavadinimas	**Duomenų bazės apkrovos paskirstymas**	
	Įrašymo komandos	**Nuskaitymo komandos**
DN	5%	95%
TN	0%	100%
ĮN	50%	50%
MOA	40%	60%

Lentelė 1.2.5.1. *Analizėje panaudoti rodikliai*

Rodiklių apibrėžimai:

- DN – **D**augiausiai **N**uskaitymo. DN apkrova yra naudojama programų sistemose (toliau – sistemos), kuriose yra prioretizuojami nuskaitymas ir smulkūs turinio pakeitimai. Tokios apkrovos naudojimo pavyzdys būtų socialinių tinklų paveikslėlių valdymo moduliai. Paveikslėliai yra nuskaitomi, kad juos galima būtų vizualiai matyti. Turinio pakeitimai šiuo atveju yra gairių (angl. Tags) pridėjimas/pakeitimas.

- TN – **T**ik **N**uskaitymas. TN apkrova yra naudojama sistemose, kuriose yra duodama galimybė tik nuskaitinėti saugomus duomenis, pavyzdžiui, naudotojų profilių duomenų nuskaitymas, kuomet jie yra kuriami išoriniu būdu.
- ĮN – **Į**rašymas ir **N**uskaitymas. ĮN apkrova yra naudojama sistemose, kuriose yra dažniausiai atnaujinamas/papildomas jau turimas turinys, pavyzdžiui, sistemos, kuriose yra saugomi paskutiniai naudotojų atlikti pakeitimai.
- MOA – **M**išrus **O**peracinis ir **A**nalitinis. MOA apkrova yra naudojama sistemose, kuriose yra dažnai nuskaitomi duomenys, kartais atnaujinami bei įrašomi nauji įrašai. Tokios sistemos pavyzdys būtų svetainės, kuriose yra platinami įvairaus pobūdžio straipsniai. Jei buvo padaryta klaidų, palikta neaiškumų arba kilo poreikis papildyti naujais duomenimis, straipsnis yra atnaujinamas. Jei skaitytojams kilo noras pasisakyti apie perskaitytą turinį, arba padiskutuoti su kitais skaitytojais, jie palieka komentarų.

Rodikliai rodo per kiek sekundžių bus atliekamas atitinkamas kiekis komandų. Analizės rezultatai yra tokių rodyklių lentelė ir išvados apie atitinkamą duomenų bazė [Lentelė 1.2.5.1.].

Tam, kad atlikti analizę objektyviai bus įterpiami atsitiktinai sugeneruoti duomenys ir stebimas įterpimo laikas. Kadangi kalba eina apie duomenų bazes, įterpiami duomenys gali būti aprašyti kaip lentelė:

Studento ID	**Vardas**	**Pavardė**	**Adresas**	**Fakultetas**
...	...	...	...	...

Lentelė 1.2.5.2. *Generuojamų duomenų įrašų struktūra.*

Kaip galima spręsti iš lentelės 1.2.5.2, analizės metu bus valdomi fiktyvių studentų duomenys.

- Studento ID - raidė ‚s‘ + atsitiktinai sugeneruotas 7 skaitmenų skaičius;
- Vardas ir pavardė - atsitiktinai sugeneruotos raidžių sekos, kurių ilgis yra tarp 2 ir 20 raidžių;
- Adresas - atsitiktinai sugeneruota 30 raidžių ir skaičių seka;
- Fakultetas - atsitiktinai išrinktas iš Vilniaus Universiteto fakultetų aibės: {chemijos, ekonomikos, filologijos, filosofijos, fizikos, gamtos mokslų, istorijos, Kauno humanitarinis, komunikacijos, matematikos ir informatikos, medicinos, teisės}.

Duomenų įterpimas buvo vykdomas naudojant įterpimo komandas skirtas per vieną užklausą įterpti daugiau nei vieną įrašą.

Duomenų nuskaitymas buvo vykdomas dvejopai:

R1 – Nuskaitomas nurodytas kiekis duomenų įrašų iš duomenų bazės.

R2 – Nuskaitomi visi įrašai kuriuose nurodytas fakultetas sutampa su atsitiktinai parinktu fakultetu. Nuskaitytas duomenų įrašų kiekis negali būti nuspėtas. Šio nuskaitymas buvo naudojamas kaip pagalbinė priemonė nustatyti lygiagrečiai vykdomų operacijų įtaką viena kitai.

Šios analizės rezultatai yra lentelės formato su sekančia antraštė:

OP/ ADĮK	**D**augiausiai **N**uskaitymo			**T**ik **N**uskaitymas			**Į**rašymas ir **N**uskaitymas			**M**išrus **O**peracinis ir **A**nalitinis		
	W	**R1**	**R2**	**W**	**R1**	**R2**	**W**	**R1**	**R2**	**W**	**R1**	**R2**

Lentelė 1.2.5.3. *Analizės rezultatų lentelės antraštė.*

Lentelėje panaudotos sąvokos [lentelė 1.2.5.3]:

- OP - Operacijų kiekis. OP reikšmės buvo parinktos tokios, kad galima būtų stebėti koks yra duomenų bazės produktyvumas vykdant nuo mažo iki didėlio operacijų kiekio:
  5,000; 20,000; 40,000; 60,000; 80,000; 100,000.
- ADĮK – Apdorojamų duomenų įrašų kiekis. Kiek duomenų kokia operacija apdoroja buvo skaičiuojama rankiniu būdu padauginus atitinkamą OP reikšmę iš lentelėje 1.2.5.1. nusakyto procentinio įverčio.
- W – Įrašymas į duomenų bazę [1.2.5. skyrius].
- R1, R2 – Nuskaitymas iš duomenų bazės [1.2.5. skyrius].
- W, R1, R2 rodikliai rodo per kiek sekundžių vidutiniškai buvo įvykdyta nuskaitymo/įrašymo operacija.

Į lentelė patalpintos W, R1, R2 reikšmės buvo gautos paskaičiavus:

$\frac{\sum_{i=1}^{k} t_i}{k}$, kur t_i – atitinkamos operacijos vykdymo laikas, o k = 100, nes siekiant gauti pakankamai patikimą rezultatą operacijos buvo vykdomos 100 kartų.

Analizė buvo atliekama ant kompiuterio su sekančiomis specifikacijomis:

- Microsoft Windows 10 Pro (64 bitų) operacinė sistema;
- Intel Core i7-3537U CPU procesorius su 2.00GHz–2.50GHz taktiniu dažniu (2 branduoliai);

- 6GB vidinės atminties (RAM);
- HDD kietasis diskas.

2. Duomenų bazių analizė

Analizei buvo parinktos šiuo metu populiariausios skirtingų rūšių NoSQL duomenų bazės pagal Ben‘o Scofield‘o klasifikaciją [skyrius 1.1.1] ir papildomai populiariausia reliacinė duomenų bazė [Dbe16]:

- Apache Cassandra – atviro kodo paskirstyta stulpelio formato duomenų bazė [Cas16].
- MongoDB – BSON formato dokumentų talpykla [RW12];
- Neo4j – grafo formato duomenų bazė, kurioje yra akcentuojamas dėmėsis į ryšius tarp saugomų duomenų [RW12];
- Redis - žodyno formato duomenų bazė [Red16];
- MySQL – reliacinių duomenų bazių valdymo sistema [Ora16c].

2.1. Apache Cassandra

Apache Cassandra – tai atviro kodo paskirstyta stulpelio formato duomenų bazė.

Paskirstyta reiškia, kad galima duomenų įrašymą ir nuskaitymą nukreipti į kelis serverius, siekiant balansuoti duomenų bazių apkrovimą ir užtikrinti saugomų duomenų saugumą.

2.1.1. Darbas su duomenų baze

2.1.1.1. Duomenų diegimo efektyvumas

Vartotojų patogumui Apache Cassandra galima atsisiųsti iš platintojo pagrindinės svetainės arba NoSQL produktų platintojo DataStax svetainės. Pagrindinėje svetainėje diegimo procesas yra aprašytas tekstu, o DataStax svetainėje yra pateiktos diegimo būsenų nuotraukos. Iš viso yra 7 diegimo būsenos, kurių eigoje naudotojas yra supažindinamas su savo teisėmis, yra pasiūloma standartinė diegimo direktorija su galimybė ją pakeisti ir leidžiama pasirinkti, kad duomenų bazės programa būtų automatiškai paleista pasibaigus diegimui. Likusios būsenos yra tarpinės, tokios kaip pradėti/pabaigti diegimą. Paprastam naudotojui užtektų ir diegėjo, tačiau tekstinė/vizualinė pagalba panaikina galimybę kur nors suklysti ([Dat16a], [PC16]).

Vertinimas: Viršija lūkesčius – 2/3.

2.1.1.2. Naudojimosi duomenų baze sudėtingumas

Dirbant su duomenų baze tiesiogiai yra naudojamas „Cassandra CQL Shell“ įrankis, kas yra iš esmės komandinė eilutė pritaikyta Apache Cassandra komandoms ir CQL užklausų kalbai. Naudojimasis vyksta per SQL primenančias komandas, kurios smarkiai išsiskiria

kuomet yra naudojami specifiškai CQL būdingi dalykai, pavyzdžiui, prieš kuriant lentelės, reikia sukurti „namespace“, kuriame bus grupuojamos po to jame sukurtos lentelės.

Apache Cassandra komandų įvesties pavyzdys:

- „namespace“ yra sukūriamas per komandą **CREATE KEYSPACE** po ko seka pavadinimas.
- Lentelė yra sukuriama per komandą **CREATE TABLE** po ko seka lentelės pavadinimas ir laužtiniuose skliaustuose visi įrašo elementų pavadinimai su jų duomenų tipais. Pavyzdžiu, šios analizės eigoje aš naudojau sekančią komandą:

  CREATE TABLE studentai (
  StudentoID text **PRIMARY KEY**,
  Vardas text,
  Pavarde text,
  Adresas text,
  Fakultetas text
  };

Naudojimasis yra nesudėtingas žinant reikalingas komandas. Visų komandų dokumentacija yra pateikiama CQL Shell sąsajoje parašius komandą „help“ ir dokumentacijoje [Dat16a].

Vertinimas: Viršija lūkesčius – 2/3.

2.1.2. Duomenų bazės struktūros lankstumas

Kadangi Apache Cassandra yra stulpelio formato duomenų bazė, įgyvendinti operacijas įgalinančias naudotoja atlikti struktūrinius lentelės pakeitimus nėra iššūkis, todėl ir čia yra palaikomas toks funkcionalumas.

Struktūriniams pakeitimams atlikti skirta operacija vadinasi „ALTER“ ir ji įgalina atlikti 4 veiksmus:

- Pridėti nurodyto tipo stulpelį su pageidaujamu pavadinimu;
- Pašalinti stulpelį su atitinkamu pavadinimu;
- Pervadinti atitinkamą stulpelį;
- Pakeisti atitinkamo stulpelio duomenų tipą.

Be to galima nurodyti loginę sąlygą, kad būtų atrinkti tik vartotoją dominantys duomenų įrašai ir atitinkamai tik jiems bus atlikti pokyčiai [Dat16b].

Vertinimas: Išskirtinė – 3/3.

2.1.3. Duomenų bazių jungimas

Apache Cassandra nepalaiko JOIN tipo operacijų, nes jų buvimas prieštarautų modelio esmei – visi tarpusavyje susiję duomenys privalo būti denormalizuoti iki vienos lentelės. Rezultate bus turima viena lentelė su daugybe stulpelių. Kad pridėti pilną duomenų įrašą į didelę lentelę, reikia parašyti ilgą užklausą. Tačiau, kad nuskaityti bet kuriuos duomenų įrašus iš tos lentelės prireiks tik vienos paprastos užklausos, o ne sudėtinės ko reikalautų normalizuotas modelis. Gali susidaryti įspūdis, kad toks sprendimas nėra racionalus atsižvelgiant į galimus ateities struktūrinius pokyčius, bet kadangi struktūra yra ganėtinai lanksti, bet kada galima atnaujinti lentelės struktūrą, kad ji atitiktų atsinaujinusius poreikius. ([Kje14], [Sch14])

Vertinimas: Atitinka lūkesčius – 1/3.

2.1.4. Pastovumo modelis

Apache Cassandra atitinka BASE modelio taisykles. Duomenų prieinamumas yra aukštas, o ne bazinis, nes jis yra prioretizuojamas šiame produkte. Toks lygis buvo pasiektas naudojant duomenų replikavimą – duomenų įrašai yra saugomi keliuose blokiniuose (angl. Cluster). Prieš kiekvieną nuskaitymą, dalyje blokinių yra paliginamos nuskaitymui parinktos reikšmės ir jei yra surandamas koks nors neatitikimas, nesutampančios mažumos įrašai yra pakeičiami į daugumos sutampančius, po ko klientui yra gražinama reikšmė. Dėl tokio patikrinimo nuskaitymo metu, bei kada duomenų įrašai yra įrašomi/atnaujinami, duomenų bazė pereina į negriežta būseną - tampa nepastovi. Pabaigus vykdyti nurodytą operaciją, duomenų bazė galiausiai tampa pastovi.

2.1.5. Operacijų su duomenimis vykdymo greitis

Tam, kad atlikti analizę, reikėjo įvedinėti tūkstančius komandų, ką daryti per komandinę eilutę būtų neracionalu, todėl tas bus daroma per tarpininką, C# programavimo kalba parašyta programą Visual Studio aplinkoje, kuriai būtina atsiųsti „CassandraCSharp“ bibliotekų paketą per aplinkos nustatymus [Til16]. Naudojant klases iš atsiųsto paketo, kode buvo įgyvendintas prisijungimas prie duomenų bazės, duomenų įterpimas, peržiūrėjimas ir šalinimas, išvedant vidutinį vykdymo laiką ekrane, įvykdžius užklausą 100 kartų. Apache Cassandra skiria 5 sekundes vienos užklausos vykdymui, tam kad naudotojas pats to nežinodamas nesutrikdytų duomenų bazės darbo. Todėl realizuojant daugybę užklausų vykdančius metodus, reikia į tą atsižvelgti. Programos realizacijoje naudojamoje šiame darbe, kuomet yra norima įvykdyti

daugiau negu 10,000 užklausų per vieną metodo iškvietimą, vykdymas yra padalinamas į iteracijas po ne daugiau negu 10,000 užklausų.

Atlikus analizę buvo gauti rezultatai [Lentelė 2.1.5]:

OP/ ADĮK	**D**augiausiai **N**uskaitymo			**T**ik **N**uskaitymas			**Į**rašymas ir **N**uskaitymas			**M**išrus **O**peracinis ir **A**nalitinis		
	W	**R1**	**R2**	**W**	**R1**	**R2**	**W**	**R1**	**R2**	**W**	**R1**	**R2**
5,000	0.17	0.41	0.87	–	0.08	0.18	0.22	0.05	0.24	0.17	0.07	0.25
20,000	0.19	0.20	0.39	–	0.08	0.19	0.98	0.10	0.28	0.55	0.10	0.27
40,000	0.16	0.10	0.27	–	0.09	0.19	1.85	0.11	0.26	1.49	0.12	0.31
60,000	0.25	0.12	0.37	–	0.09	0.19	3.18	0.12	0.30	2.09	0.10	0.27
80,000	0.40	0.21	0.37	–	0.09	0.18	3.82	0.11	0.29	3.23	0.12	0.29
100,000	0.38	0.09	0.25	–	0.08	0.18	5.15	0.12	0.37	3.95	0.11	0.31

Lentelė 2.1.5. *Apache Cassandra operacijų vykdymo greitis.*

Tarpusavyje palyginus įrašymo ir nuskaitymo greičius iš apkrovų kur buvo įrašomi duomenys, galima pastebėti tendenciją, kad gauti skaičiai svyruoja aukštyn ir žemyn. Šitą reiškinį galima paaiškinti tuo, kad vykdant kiekvieną iš šių operacijų, duomenų bazė pereina į negriežtą būseną ir kadangi šituo atveju abi operacijos yra vykdomos tuo pačiu metu, tą negriežta būsena yra ypač nestabili, iš ko galima padaryti išvadą, kad nuskaitymas ir įrašymas vienas kito veikimą įtakoja. Tuo galima įsitikinti apžvelgus „tik nuskaitymo“ apkrovos vykdymo rezultatus – negriežta būsena yra labiau stabili negu kitų apkrovų atvejais.

Remiantis padarytais pastebėjimais, galima padaryti išvadą, kad, kaip ir teigė jos projektuotojai, Apache Cassandra operacijų vykdymo metu duomenų bazė nėra pastovi. Atitinkamai, kuo mažiau yra vykdoma operacijų (ypač nuskaitymo), tuo stabilesnis yra jau patalpintų duomenų pasiekiamumas [Cas16].

2.2. MongoDB

MongoDB – tai dokumentų talpykla, kurioje dokumentai yra saugomi BSON dokumentų formatu. BSON - tai JSON dokumentų modelio praplėtimas pridedantis papildomų duomenų tipų palaikymą [MI16a].

2.2.1. Darbas su duomenų baze

2.2.1.1. Duomenų diegimo efektyvumas

Iš MongoDB tinklalapio kartu su diegimo failu galima atsisiųsti vadovą su diegimo instrukcijomis visoms 4 palaikomoms operacinėms sistemoms [MI16e]. Pats diegimas, priklausomai nuo pasirinkčių, susideda iš 5-6 žingsnių, kurių eigoje naudotojas gali peržiūrėti savo teises, bei nustatyti, kuriuos MongoDB komponentus jis nori diegti ir kur. Papildomai naudotojams, kurie yra labiau įpratę dirbti su komandine eilute, yra duodama galimybę įdiegti MongoDB per ją. Nepriklausomai nuo pasirinkto būdo diegimo procesas yra paprastas, greitas ir nereikalauja papildomų žinių. Tačiau, kad tuojau pat pradėti dirbti su MongoDB, jį dar reikia ir nustatyti, bet ne daug, nes tereikia nurodyti kur bus saugomi duomenys: arba numatytoje direktorijoje, arba kitur, naudojant standartines komandinės eilutės komandas.

Kadangi darbas su MongoDB yra vykdomas tiesiogiai per komandinę eilutę arba per tarpininką, yra numatoma, kad naudotojas supranta kaip bendrai ja naudotis, todėl diegimas ir direktorijos, kur bus saugomi duomenys, nustatymas yra palyginus paprasti. Tačiau, kilus neaiškumams, juos išsklaido diegimo vadovas. Iš esmės, turint ir diegimo failą ir dokumentaciją, MongoDB gali įdiegti bet koks naudotojas, mokantis naudotis kompiuterių ir suprantantis kas yra komandinė eilutė. ([MDB15a], [MDB15b])

Vertinimas: Viršija lūkesčius – 2/3.

2.2.1.2. Naudojimosi duomenų baze sudėtingumas

Tam kad, įjungti MongoDB, reikia paleisti failą mongod.exe iš diegimo direktorijos, po ko yra įjungiamas serverio langas per komandinę eilutę naudojant numatytą duomenų saugojimo direktoriją. Sekantis žingsnis yra prisijungti prie serverio kaip klientas įjungus mongo.exe, po ko galima pradėti dirbti.

Visų pirmą reikia sukurti/pasirinkti duomenų bazę, kas yra padaroma naudojant komandą **USE** po ko seka duomenų bazės pavadinimas.

Kadangi MongoDB yra dokumentų talpyklą, duomenys yra talpinami kaip BSON dokumentai. Pavyzdžiui, šio darbo kontekste įterpimo komanda atrodytu taip:

```
db.studentai.insert ( {
    studento_id : 's9999999',
    vardas : 'Vardenis',
    pavarde : 'Pavardenis',
    adresas : 'Didlaukio g. 47',
    fakultetas : 'matematikos ir informatikos'
```

})

Kaip ir Apache Cassandra, naudojimasis yra nesudėtingas žinant reikalingas komandas. Komandų sintaksę galima sužinoti per „help“ komanda ir naudojantis dokumentaciją.

Vertinimas: Viršija lūkesčius – 2/3.

2.2.2. Duomenų bazės struktūros lankstumas

Kadangi MongoDB yra dokumentų talpykla, saugomi įrašai neprivalo turėti vieningos struktūros, kitais žodžiais – kiekvienas įrašas nepriklauso nuo jokio kito. Tokiu atveju, kad užtikrinti struktūros lankstumą, reikia modifikuoti įrašus atskirai. Tačiau, MongoDB saugo duomenis kaip BSON formato įrašus, todėl jų modifikavimas iš esmės susiveda prie atitinkamų laukų keitimo. Bendriausia iš tokį funkcionalumą įgalinančių operacijų yra „update“. Jai yra paduodami du parametrai [MI16b]:

- BSON lauko pavadinimas ir reikšmė pagal kurią bus atrenkami įrašai atnaujinimui;
- Kokius laukus reikia keisti ir kokias reikšmes jiems priskirti. Taip pat yra galimybė pervadinti lauko pavadinimą, bei pakeisti jo duomenų tipą.

Vertinimas: Išskirtinė – 3/3.

2.2.3. Duomenų bazių jungimas

MongoDB lentelių jungimas yra realizuojamas per „aggregate“ operacijos „$lookup“ vidinę operaciją. Operacija yra vykdoma pasirinkus vieną iš jungimui skirtų duomenų bazių (toliau bus vadinama vidine duomenų baze) ir yra pateikiami sekantys parametrai:

- Su kokia išorine duomenų baze bus jungiama;
- Vidinės duomenų bazės jungimo laukas;
- Išorinės duomenų bazės jungimo laukas;
- Kaip vadinsis prijungtus duomenis saugantis laukas.

Rezultate yra gaunamas BSON dokumentas kuriame yra pateikti vidinės duomenų bazės laukai ir papildomas laukas su išorinės duomenų bazės laukais (išskyrus jungime panaudota lauką) [MI16c].

Vertinimas: Išskirtinė – 3/3.

2.2.4. Pastovumo modelis

MongoDB atitinka BASE modelio taisyklės. Aukštas duomenų prieinamumas yra užtikrinamas naudojant duomenų replikavimą. Tam, kad duomenų bazė būtų kiek įmanoma

pastovi, kitais žodžiais, kad būtų kiek įmanoma griežtesnė jos būsena, yra naudojamas apkrovos paskirstymas kuomet duomenų bazė naudoja daugiau negu vieną serverį. Akivaizdu, kad esminis privalumas nuo to yra duomenų nuskaitymui, nes, pavyzdžiui, jeigu yra nustatyta, kad būtų daromas pilnas duomenų replikavimas į atitinkamus serverius jų įrašymo metu, rezultate gausis ilgas įrašymas ir greitas nuskaitymas. Todėl, kad gauti optimalų duomenų bazės pastovumą operacijų vykdymo metu, reikia deramai atsižvelgti į techninės įrangos galimybės ir technologinio sprendimo prioritetus. Be to derėtų paminėti, kad nėra vykdoma jokių naudotojo operacijų, yra užtikrinamas galutinis pastovumas nepriklausomai nuo pačios duomenų bazės nustatymų. ([Cho13], [Nya14])

2.2.5. Operacijų su duomenimis vykdymo greitis

Analizei atlikti buvo naudojama MongoDB C# tvarkyklė (angl. driver) [MI16d]. Programiškai buvo įgyvendintas prisijungimas prie duomenų bazės, bei duomenų įrašymas ir nuskaitymas.

Atlikus analizę buvo gauti rezultatai [2.2.5]:

OP/ ADĮK	**D**augiausiai **N**uskaitymo			**T**ik **N**uskaitymas			**Į**rašymas ir **N**uskaitymas			**M**išrus **O**peracinis ir **A**nalitinis		
	W	**R1**	**R2**	**W**	**R1**	**R2**	**W**	**R1**	**R2**	**W**	**R1**	**R2**
5,000	0.04	0.08	1.52	–	0.07	1.76	0.08	0.06	4.66	0.08	0.07	6.97
20,000	0.06	0.27	1.66	–	0.24	1.76	0.32	0.18	5.03	0.21	0.18	7.19
40,000	0.08	0.50	1.71	–	0.52	1.86	0.57	0.30	5.39	0.36	0.36	7.60
60,000	0.12	0.78	1.88	–	0.77	1.96	0.77	0.46	6.11	0.65	0.56	8.29
80,000	0.14	1.02	1.96	–	0.99	1.93	0.98	0.61	6.69	0.80	0.73	8.73
100,000	0.15	1.28	2.10	–	1.31	2.08	1.39	0.76	7.72	0.98	0.92	9.38

Lentelė 2.2.5. *MongoDB operacijų vykdymo greitis.*

Apžvelgus visą lentelę galima pastebėti, kad įrašymo ir nuskaitymo operacijų vykdymas tuo pačiu metu viena kitą įtakoja. Tačiau ne įrašymo operacija negatyviai veikia nuskaitymo kaip galima buvo tikėtis, o atvirkščiai. Toks prieštaringas reiškinys yra vadinamas negriežta duomenų bazės būsena. Šiuo atveju jos buvimas yra toks akivaizdus, nes atliekant analizę buvo naudojamas tik vienas serveris. Iš čia galima padaryti išvada, kad MongoDB kaip duomenų bazė yra ganėtinai nestabili. Kad pagerinti stabilumą, viena iš galimų išeičių pritaikyti bent kelis serverius, nekalbant apie tinkamą jų konfigūravimą.

2.3. Neo4j

Neo4j – tai grafo formato duomenų bazė, kurioje duomenų įrašai yra saugomi kaip viršūnės ir mazgai. Viršūnės saugo įrašų metaduomenis, o mazgai išskiria ryšių pobūdį tarp jų (jei tokių yra) [NT16b]. Analizei buvo naudojama nekomercinė Neo4j versija.

2.3.1. Darbas su duomenų baze

2.3.1.1. Duomenų diegimo efektyvumas

Neo4j diegimas yra vykdomas per paprastą, niekuo neišsiskiriantį, 6 žingsnių diegėją, primenantį numatytąjį Windows MSI diegėją. Pats diegėjas nesuteikia jokios informacijos apie tai, kas yra diegiama, tačiau tas yra kompensuojama galimybe atsisiųsti dokumentaciją iš diegėjo tinklalapio [NT16a]. Kad pradėti dirbti su pačia duomenų baze, nereikia atlikti jokių papildomų nustatymų.

Vertinimas: Atitinka lūkesčius – 1/3.

2.3.1.2. Naudojimosi duomenų baze sudėtingumas

Kad dirbti su Neo4j yra naudojama „Cypher“ užklausų grafuose vykdymo kalba. Duomenų pridėjimo formatas primena JSON duomenų saugojimo formatą, kaip ir MongoDB, tačiau kadangi Neo4j yra grafo formato duomenų bazė ir joje yra saugomi ryšiai tarp duomenų, jos užklausos nėra panašios į MongoDB užklausas, nes jos turi žymiai daugiau sudėtinių dalių. Dirbant su turimais duomenimis, užklausos yra kažkiek panašios į SQL, tačiau yra kitokio pavidalo. Vizualiai SQL užklausos primena filtravimo užklausą, o „Cypher“ užklausos yra iš esmės funkcijų apibrėžimai, kurie susideda iš naudojamos komandos pavadinimo, filtravimo užklausos ir nurodymo kokie duomenys bus gražinami. Nors SQL irgi turi panašiais sudėtines dalis, jų eiliškumas ne kaip neprimena funkcijos apibrėžimo.

Su Neo4j galima dirbti per naršyklės sąsają ir komandinę eilutę.

Apibendrinus, nors „Cypher“ ir primena SQL, ji yra fundamentaliai kita kalba, todėl gali prireikti laiko prie jos priprasti, tačiau ji nėra sudėtinga ir leidžia filtruoti duomenis ne tik pagal jų savybes, bet ir pagal santykius tarp jų arba jų nebuvimą.

Vertinimas: Išskirtinė – 3/3.

2.3.2. Duomenų bazės struktūros lankstumas

Kadangi duomenys yra saugomi JSON primenančiu formatu, kaip ir MongoDB atveju, jų keitimas yra ganėtinai paprastas. Norimi pakeitimai yra atliekami per „MATCH“ komandą, pridėjus raktažodį SET, po kurio yra nurodomi naujų/pakeistų laukų pavadinimai ir reikšmės.

Tokiu būdu galima modifikuoti atitinkamus mazgus bei ryšius tarp jų. Be to galima ištrinti norimus laukus su komanda „DELETE“. [NT16c]

Vertinimas: Viršija lūkesčius – 2/3.

2.3.3. Duomenų bazių jungimas

Skirtingai nuo MongoDB, skirtingų tipų duomenų įrašai yra saugomi savo lentelėse, todėl Neo4j JOIN tipo operacijos primena SQL‘o JOIN operacijas. Lentelių jungimą galima padaryti dvejais būdais [NT16c]:

- Kelių užklausų rezultatų jungimas tarp jų pridedant raktažodį UNION;
- Išskirti du objektus ir/arba jų laukus su raktažodžiais „MATCH“ ir „OPTIONAL MATCH“ ir nurodyti gražinti norimus laukus.

Vertinimas: Išskirtinė – 3/3.

2.3.4. Pastovumo modelis

Priešingai negu daugelis NoSQL duomenų bazių, Neo4j yra pagrista ACID modelio taisyklėmis.

Operacijų atomiškumą garantuoja transakcijų vykdamas besilaikant tos pačios veiksmų sekos kaip ir reliaciniuose duomenų bazėse:

1) uždedamas prieigos apribojimas apdorojamiems mazgams ir visiems su jais susijusiems ryšiams;
2) atliekami pokyčiai;
3) išsaugomi pokyčiai;
4) anuliuojami 1) žingsnyje uždėti apribojimai.

Pasibaigus bet kuriai transakcijai duomenų bazė lieka pastovi. Prieigos apribojimai užtikrina operacijų izoliaciją. Pokyčių išsaugojimas užtikrina duomenų ilgalaikiškumą.

2.3.5. Operacijų su duomenimis vykdymo greitis

Analizei atlikti buvo naudojama Neo4j C# tvarkyklė [MI16d]. Per programinį kodą buvo įgyvendintas prisijungimas prie duomenų bazės, CSV failo generavimas ir importavimas į duomenų bazė, bei duomenų nuskaitymas iš jos.

Atlikus analizę buvo gauti rezultatai [Lentelė 2.3.5]:

OP/ ADĮK	**D**augiausiai **N**uskaitymo			**T**ik **N**uskaitymas			**Į**rašymas ir **N**uskaitymas			**M**išrus **O**peracinis ir **A**nalitinis		
	W	**R1**	**R2**	**W**	**R1**	**R2**	**W**	**R1**	**R2**	**W**	**R1**	**R2**
5,000	0.34	1.91	1.01	–	2.38	4.23	0.97	0.83	0.25	0.94	0.96	0.22
20,000	0.78	3.12	2.22	–	6.26	4.17	3.77	2.41	0.58	3.08	2.62	0.46
40,000	1.38	5.90	2.68	–	11.59	4.76	8.70	4.10	0.60	7.21	5.28	0.97
60,000	2.01	8.96	3.31	–	16.17	4.40	15.33	6.30	0.89	17.46	7.47	1.17
80,000	2.65	11.54	4.19	–	20.52	4.26	40.58	8.33	1.07	29.04	9.37	1.53
100,000	3.46	14.67	5.18	–	25.50	4.30	61.10	9.78	1.29	32.15	12.25	1.53

Lentelė 2.3.5. *Neo4j operacijų vykdymo greitis.*

Bendrai nagrinėjant visus apkrovų paskirstymus, galima pastebėti dėsningumą, kad augant apdorojamų duomenų kiekiui sparčiai auga operacijų vykdymo laikas. Taip yra todėl, kad Neo4j duomenų bazėje įterpiant ir nuskaitant duomenų įrašus yra atsižvelgiama į tai, kad pati duomenų bazė yra tikėtinai nejungus grafas, o paieškos grafe sudėtingumas blogiausiu atveju yra $O(|E|)$, kur E – briaunų skaičius, iš ko seka, kad visi mazgai ir ryšiai tarp jų gali būti aplankyti. Kad toks funkcionalumas būtų galimas yra aukojami resursai ir vykdymo laikas.

Analizės vykdymo metu buvo pastebėta, kad kuomet nuskaitomų duomenų įrašų kiekis yra didesnis už 500,000 vienetų (virš 0.5GB informacijos), ant dviejų branduolių kompiuterio, kuriame buvo įdegta duomenų bazė, duomenų bazės serveris užsikerta ir po kelių minučių atšaukia nuskaitymo užklausą. Jei bandant vykdyti tokią užklausą, yra tuo pačių metu vykdomas bent 60,000 įrašų įterpimas į duomenų bazę, serveris blokuojasi ir blogiausiu atveju liks tokioje būsenoje 100 sekundžių, po ko bus atšaukta įrašų įterpimo užklausa. Dėl šitos priežasties analizės metu teko neviršyti minėtojo limito.

Apibendrinus, Neo4j funkcionalumas pateisina lūkesčius dėl vykdymo laiko, ką galima pagerinti, pavyzdžiui, prijungus dar bent kelis serverius, kas žymiai pagerins produktyvumą. Šios analizės atvejų, duomenų bazė užsikirsdavo dėl silpno testavimui parinkto kompiuterio ir dėl to, kad naudojama duomenų bazės versija buvo skirta nekomerciniam testavimui.

2.4. Redis

Redis – tai žodyno formato duomenų bazė su ganėtinai didelių komandų rinkiniu [Red16].

2.4.1. Darbas su duomenų baze

2.4.1.1. Duomenų diegimo efektyvumas

Redis nėra oficialiai palaikoma Windows operacinės sistemos, tačiau yra nė viena versija neoficialiai importuota. Analizei buvo naudojama populiariausia iš jų – priklausanti programuotojui su slapyvardžiu „MSOpenTech“ [Git16]. Diegėjas atlieka diegimą per 10 žingsnių, kurių eigoje galima nustatyti sunaudojamos vietos duomenų talpykloje kiekį ir koks serverio prievadas (angl. Port) bus naudojamas. Nors ši versija buvo importuota iš Unix ir OS X operacinių sistemų, kur diegimas ir nustatymas yra vykdomi per komandinę eilutę, šiuo atveju jokių papildomų veiksmų po diegimo nereikia atlikti - norint pradėti dirbti su Redis, užtenka tik įjungti serverį ir klientą.

Vertinimas: Viršija lūkesčius – 2/3.

2.4.1.2. Naudojimosi duomenų baze sudėtingumas

Darbui su Redis yra naudojama komandinė eilutė, kurioje yra įjungtas klientas, kuris savo ruožtu yra prijungtas prie serverio. Serveris ir klientas, akivaizdžiai, yra paleisti skirtinguose komandinės eilutės languose. Darbas su duomenų baze yra vykdomas per komandas ir jų parametrus, kurių pavidalas yra mišinys tarp komandinės eilutės komandų ir SQL sintaksių. Duomenys yra saugomi ir apdorojami JSON formatų, kur kiekviena įrašo dalis gali būti skirtingo duomenų tipo, kuriuos klientų programa atpažįsta ir leidžia atlikti numatytus veiksmus.

Pačios komandos yra palyginus paprastos ir nesunkiai suvokiamos. Pavyzdžiui:

- Įrašų pridėjimas:

  set studentai:s9999999 {

  “studento_id”:9999999,

  “vardas”:Vardenis,

  “pavarde”:Pavardenis,

  “adresas”:”mifas”,

  “fakultetas”:mif

  }

  Po raktažodžio **set** yra įvedamas įrašo tipas po ko dvitaškis ir rakto reikšmė, o po to laužtiniuose skliaustuose pats duomenų įrašas JSON formatu.

- Įrašų nuskaitymas:
    - Norint nuskaityti vieną įrašą - **get** studentai:s9999999.
    - Norint nuskaityti kelis to pačio tipo įrašus – **keys** studentai:*.

Komandų sintaksę galima sužinoti per sąsają ir per dokumentaciją oficialioje svetainėje [Red16].

Vertinimas: Viršija lūkesčius – 2/3.

2.4.2. Duomenų bazės struktūros lankstumas

Redis struktūros lankstumo klausimą išsprendžia tradiciniu būdu - „SET“ tipo komandas panaudojus su jau egzistuojančiais duomenų įrašų raktais vietoj senų reikšmių įrašius naujas. Vienos „SET“ tipo komandos yra skirtos įrašyti vienetinėms reikšmėms, kitos – sudėtinėms (sąrašams, aibėms, maišos lentelėms). Be to galima keisti norimas vidinės reikšmes [Red16].

Vertinimas: Viršija lūkesčius – 2/3.

2.4.3. Duomenų bazių jungimas

Redis nepalaiko JOIN tipo užklausų, nes toks funkcionalumas šiai duomenų bazei yra nereikalingas. Yra operacija „SUNION“, kuri apjungia kelių aibių reikšmės į vieną rezultatų sąrašą ir išveda į sąsają. Taip pat galima tą rezultatą išsaugoti kaip naują įrašą naudojant „SUNIONSTORE“. Kadangi šitos operacijos yra pritaikomos tik aibėms ir iš jų gautus rezultatus galima tik išvesti arba išsaugoti, jos nesiskaito kaip JOIN tipo užklausos [Red16].

Vertinimas: Viršija lūkesčius – 2/3.

2.4.4. Pastovumo modelis

Redis negalima priskirti prie ACID ar BASE pastovumo modeliu. Pati duomenų bazė yra ganėtinai primityvi, nes pagal savo esmę yra tik žodynas ir ne daugiau [Red16].

2.4.5. Operacijų su duomenimis vykdymo greitis

Redis pagrindinio puslapio klientų skiltyje yra pateikti programuotojams, norintiems per programinį kodą dirbti su Redis, klientai įvairiomis programavimo kalbomis. Dėl kalbos patogumo buvo nagrinėjami C# skirti klientai ir buvo išrinktas „StackExchange.Redis“ klientas dėl savo populiarumo ir gerų atsilepimų [SE16].

Tyrimui buvo realizuoti prisijungimas prie duomenų bazės, duomenų įterpimas ir išgavimas. Prieš kiekvieno apkrovimo tyrimą, buvo pridėta 100,000 duomenų įrašų, kad nuskaitymo užklausos būtų vykdomos norimu mastu, o pasibaigus kiekvienam tyrimui įrašai buvo anuliuoti.

Atlikus analizę buvo gauti rezultatai [Lentelė 2.4.5]:

OP/ ADĮK	**D**augiausiai **N**uskaitymo			**T**ik **N**uskaitymas			**Į**rašymas ir **N**uskaitymas			**M**išrus **O**peracinis ir **A**nalitinis		
	W	**R1**	**R2**	**W**	**R1**	**R2**	**W**	**R1**	**R2**	**W**	**R1**	**R2**
5,000	0.0006	0.04	1.06	–	0.04	1.11	0.001	0.02	1.32	0.001	0.03	1.29
20,000	0.0010	0.19	1.22	–	0.20	1.12	0.006	0.09	2.38	0.003	0.12	2.09
40,000	0.0014	0.35	1.68	–	0.41	1.08	0.008	0.21	4.34	0.007	0.24	3.71
60,000	0.0018	0.57	2.07	–	0.61	1.13	0.011	0.30	7.28	0.010	0.39	6.02
80,000	0.0020	0.79	2.50	–	0.78	1.16	0.017	0.40	10.68	0.012	0.53	9.14
100,000	0.0032	1.02	3.09	–	1.01	1.17	0.019	0.54	15.57	0.022	0.64	12.80

Lentelė 2.4.5. *Redis operacijų vykdymo greitis.*

Redis duomenų bazė buvo suprojektuota kaip paprasta, tačiau labai greita žodyno formato duomenų bazė, ir tam įrodymas yra gauti duomenų įrašymo greičio įverčiai. Palyginus apkrovų paskirstymus su ir be duomenų įrašymo, galima pastebėti, kad operacijų vykdymas vienas kito neįtakoja.

Taigi, Redis, kaip ir giriasi jos projektuotojai, labai greitai įrašo ir nuskaito duomenis, bei yra stabili tai darant [Red16]. Taip pat ji turi platu komandų rinkinį ir detalią dokumentaciją. Vienintelis pastebėtas rimtas trukumas yra tame, kad įrašant palyginus didelį kiekį duomenų kaip paketą, apytiksliai puse procento įrašų nėra įrašomi.

2.5. MySQL

MySQL – tai labiausiai naudojama reliacinė duomenų bazių valdymo sistema su plačiu funkcionalumu ir taikymo spektru [Ora16a].

2.5.1. Darbas su duomenų baze

2.5.1.1. Duomenų diegimo efektyvumas

MySQL turi kelias skirtingas versijas ir daug įrankių. Šiai analizei buvo atsiusti visi galimi įrankiai per „MySQL Installer“ tam, kad analizės metu nieko nepritruktų. Pats diegimas yra beveik automatinis: reikia pasirinkti kokius įrankius atsiusti, po to kai kurių iš jų nustatymus galima pakeisti, pavyzdžiui serverio prievadą [Ora16a].

Vertinimas: Išskirtinė – 3/3.

2.5.1.2. Naudojimosi duomenų baze sudėtingumas

Atlikus pilną diegimą galima rinktis ar dirbti per MySQL komandinės eilutės klientą, arba per MySQL Workbench grafine sąsaja.

Dirbant su komandine eilute, reikia žinoti reikalingas komandas ir jų sintaksę. Užklausų rezultatai yra pateikiami tekstinių formatų ir esant žemai monitoriaus rezoliucijai gali netilpti į ekraną.

Dirbant su grafine sąsaja, rašant komandas arba užklausas, „intellisense“ įrankis parodo sintaksės klaidas arba kaip galima užbaigti rašomą žodį realiu laiku. Užklausų rezultatai yra pateikiami kaip interaktyvi lentelė, kurioje turinį galima rikiuoti pagal stulpelius. Taip pat yra daugybė įrankių palengvinančių darbą, pavyzdžiui, galima sugeneruoti duomenų bazės struktūros diagramą.

Kokią sąsają rinktis sprendžia naudotojas pagal savo pageidavimus.

Vertinimas: Viršija lūkesčius – 2/3.

2.5.2. Duomenų bazės struktūros lankstumas

MySQL duomenų bazė užtikrina struktūros lankstumą per „ALTER TABLE“ užklausą. Per šią užklausą galima [Ora16a]:

- Pridėti/panaikinti lentelės stulpelį;
- Pridėti/panaikinti indeksą;
- Pakeisti stulpelio duomenų tipą;
- Pervadinti stulpelį.

Vertinimas: Išskirtinė – 3/3.

2.5.3. Duomenų bazių jungimas

MySQL duomenų bazė lentelių jungimui turi „JOIN“ užklausą, kuri savo ruožtu turi daugybę variacijų, kiekviena iš kurių kitaip negu kitos jungia lenteles, tačiau rezultatai nebūtinai gali skirtis. Sintaksiškai jos tarpusavyje skiriasi pavadinimu ir tuo, kad ne visos iš jų reikalauja jungimo sąlygų. [Ora6]

Vertinimas: Išskirtinė – 3/3.

2.5.4. Pastovumo modelis

MySQL, kaip reliacinio modelio duomenų bazė, yra pagrista ACID taisyklėmis. Vienas iš pagrindinių įrankių įgyvendinančių duomenų bazės veikimo logika yra InnoDB duomenų saugojimo variklis, kuris iš esmės ir užtikrina ACID taisyklių laikymąsi.

Atomiškumas, izoliacija ir duomenų ilgalaikiškumas yra užtikrinami per transakcijų mechanizmą. Transakcijos metu apdorojami duomenys yra izoliuoti. Sėkmingai atlikus transakciją pokyčiai yra išsaugomi; nesėkmingai – anuliuojami. Pastovumą užtikrina InnoDB apsauga nuo duomenų bazės veiklos sutrikimų [Ora16a].

2.5.5. Operacijų su duomenimis vykdymo greitis

Analizei buvo naudojama „Connector/Net“ .NET tvarkyklė [Ora16b]. Programoje buvo įgyvendinti prisijungimas prie duomenų bazės, sudėtinis įrašų pridėjimas ir 2 „SELECT“ sakinių vykdymai.

Atlikus analizę buvo gauti rezultatai [Lentelė 2.5.5]:

OP/ ADĮK	**D**augiausiai **N**uskaitymo			**T**ik **N**uskaitymas			**Į**rašymas ir **N**uskaitymas			**M**išrus **O**peracinis ir **A**nalitinis		
	W	**R1**	**R2**	**W**	**R1**	**R2**	**W**	**R1**	**R2**	**W**	**R1**	**R2**
5,000	0.10	0.03	0.55	–	0.01	0.84	0.26	0.01	0.45	0.24	0.01	0.42
20,000	0.25	0.05	0.64	–	0.06	0.86	0.55	0.02	0.48	0.56	0.03	0.63
40,000	0.44	0.13	0.87	–	0.07	0.77	0.94	0.04	0.86	0.67	0.06	0.68
60,000	0.52	0.74	1.06	–	0.47	1.08	1.25	0.07	1.54	1.25	0.30	1.92
80,000	0.73	1.03	1.39	–	0.63	1.15	2.01	0.43	3.25	2.14	0.66	3.55
100,000	0.75	1.36	1.80	–	0.89	1.32	2.79	0.61	4.45	2.55	0.84	4.53

Lentelė 2.5.5. *MySQL operacijų vykdymo greitis.*

Operacijų vykdymo laikas auga netolygiai, ką galima paaiškinti tuo, kad lygiagrečiai vykdant keliais operacijas, jos viena kitą įtakoja, ko pasekmėje blogėja produktyvumas. Sprendžiant pagal ĮN ir MOA apkrovas [1.2.5. skyrius], nuskaitymo operacija labiau įtakoja įrašymo operaciją, negu atvirkščiai. Remiantis šiuo faktu, galima spekuliuoti, kad MySQL geriausiai save parodo esant labiau į nuskaitymą orientuotiems apkrovų paskirstymams. Papildomai, šitą spekuliaciją paremia tas faktas, kad kuo daugiau yra įrašoma duomenų įrašų, tuo labiau greitėja įrašymo procesas.

Taigi, MySQL, kaip duomenų bazių valdymo sistema, patenkina visus vartotojo lūkesčius (bent jau taip buvo šio tyrimo metu). Ji yra greita ir su ja patogu dirbti tiesiogiai per siūlomas sąsajas ir netiesiogiai per įvairioms programavimo kalboms skirtus klientus.

3. Duomenų bazių palyginimas

Duomenų bazių palyginimas susidėjo iš dviejų dalių:

- Ypatybių palyginimas;
- Operacijų su duomenimis vykdymo laiko palyginimas.

3.1. Duomenų bazių ypatybių palyginimas

Duomenų bazės yra nesunkų palyginti kuomet yra turimi skaitiniai vertinimai pagal sutartą skalę. Tačiau kuomet vertinimai sutampa, reikia plačiau ištirti pačias duomenų bazės ir kodėl joms buvo priskirti atitinkami vertinimai. Šitam duomenų bazių palyginimui skaitiniai vertinimai buvo gauti atlikus duomenų bazių analizę [lentelė 3.1], o platesniam tyrimui atlikti yra naudojami analizės metu padaryti pastebėjimai ir išvados.

Duomenų bazė / Kriterijus	Apache Cassandra	MongoDB	Neo4j	Redis	MySQL
Duomenų bazės diegimo efektyvumas	2	2	1	2	3
Naudojimosi duomenų baze sudėtingumas	2	2	3	2	2
Duomenų bazės struktūros lankstumas	3	3	2	2	3
Duomenų bazių jungimas	1	3	3	2	3
Pastovumo modelis	BASE	BASE	ACID	–	ACID

Lentelė 3.1. *Duomenų bazių analizės atkliktos pagal lentelėje 1 apibrėžtą skalę rezultatai [2 skyrius].*

3.1.1. Duomenų bazės diegimo sudėtingumas

Analizės metu buvo pastebėta, kad analizuotų duomenų bazių diegėjus galima suskirstyti į du tipus: tie, kurie tiesiog įdiegia duomenų bazę, ir tie, kurie turi papildomą funkcionalumą.

Neo4j turėjo labai paprastą diegėją, todėl jis buvo įvertintas 1/3. Likusių duomenų bazių diegėjai, išskyrus MySQL, buvo įvertinti 2/3, kiekvienas iš kurių turi savo individualią

pagirtiną savybę. Labiausiai išsiskyrė MySQL diegėjas, gavęs įvertinimą 3/3 – galima buvo pasirinkti kuriuos MySQL produktus automatiškai atsisiųsti ir per tą pačią sąsają juos įdiegti bei sutvarkyti nustatymus. Kitos duomenų bazės mažai ko išsiskyrė: Apache Cassandra, MongoDB ir Redis galima įdiegti dvejais būdais.

3.1.2. Naudojimosi duomenų baze sudėtingumas

Darbo su duomenų bazėmis sudėtingumą daugiausiai lemia darbui skirta sąsaja. Nagrinėtos duomenų bazės turi komandinės eilutės pavidalo, grafinę arba abiejų tipu sąsajas, su visomis iš kurių yra ganėtinai nesudėtinga dirbti.

Iš visų patogiausia ir labiausiai palengvinanti darbą buvo Neo4j grafinė sąsaja, kuri iš esmės yra komandinės eilutės ir grafinės sąsajos mišinys. Komandų įvedimui yra naudojama įvedimo eilutė, o rezultatas gali būti gražintas arba tekstiniu formatu, arba grafiniu.

Iš žemesnį vertinimą gavusių duomenų bazių. patogiausią grafinę sąsają turėjo MySQL, dėl bendro patogumo ir papildomo funkcionalumo. Su likusiomis duomenų bazėmis yra dirbama per komandinės eilutės sąsaja, kas tarpusavyje lyginant beveik nesiskiria.

3.1.3. Duomenų bazės struktūros lankstumas

Visos iš nagrinėtų duomenų bazių parodė ganėtinai gerą struktūros lankstumą. Iš jų išsiskyrė Apache Cassandra, MongoDB ir MySQL, nes juose yra įgyvendintas funkcionalumas, leidžiantis pakeisti duomenų įrašo elemento duomenų tipą į kitą prie jo sutaikomą tipą. Kitose duomenų bazėse, kad gauti tokį rezultatą, reikia ištrinti ir iš naujo pridėti atitinkamą elementą, kas yra ne tik nepatogu, bet ir proceso eigoje padarius klaidą galima netekti tarpinių duomenų.

Pažiūrėjus į rezultatus iš duomenų bazių modelių pusės, galima padaryti išvadą, kad reliacinės duomenų bazės privalo turėti struktūros keitimo funkcionalumą, o NoSQL – priklauso nuo duomenų bazės klasės.

3.1.4. Duomenų bazių jungimas

Duomenų bazių jungimo funkcionalumo buvimą, kaip ir bet kokio kito funkcionalumo, buvimą lemia jo reikalingumas. Šituo atžvilgiu tyrime nagrinėtos duomenų bazės išsiskirstė į dvi grupės – tos, kurioms toks funkcionalumas yra reikalingas, ir tos, kurioms jo nereikia.

MySQL, kaip reliacinė duomenų bazė, reikalauja tokio funkcionalumo ir tam įrodymas yra tai, kad besilaikant lentelių normalizavimo, duomenų saugomo sumetimais arba dėl kokios nors kitos priežasties, tik jo dėka galima vienu metu dirbti su keliomis lentelėmis, ko

reikalauja didžioji dalis technologinių sprendimų, kuriuose yra taikomos reliacinės duomenų bazės.

Nagrinėtos NoSQL duomenų bazės dėl jungimo funkcionalumo išsiskirstė. Apache Cassandra ir Redis duomenų bazėms toks funkcionalumas nėra reikalingas. Apache Cassandra atveju dėl to, kad priešingai negu MySQL, duomenų įrašai yra saugomi vienoje lentelėje, o Redis tokį funkcionalumą nebūtų kaip pritaikyti. Likusioms duomenų bazėms – MongoDB ir Neo4j toks funkcionalumas yra reikalingas.

3.1.5. Pastovumo modelis

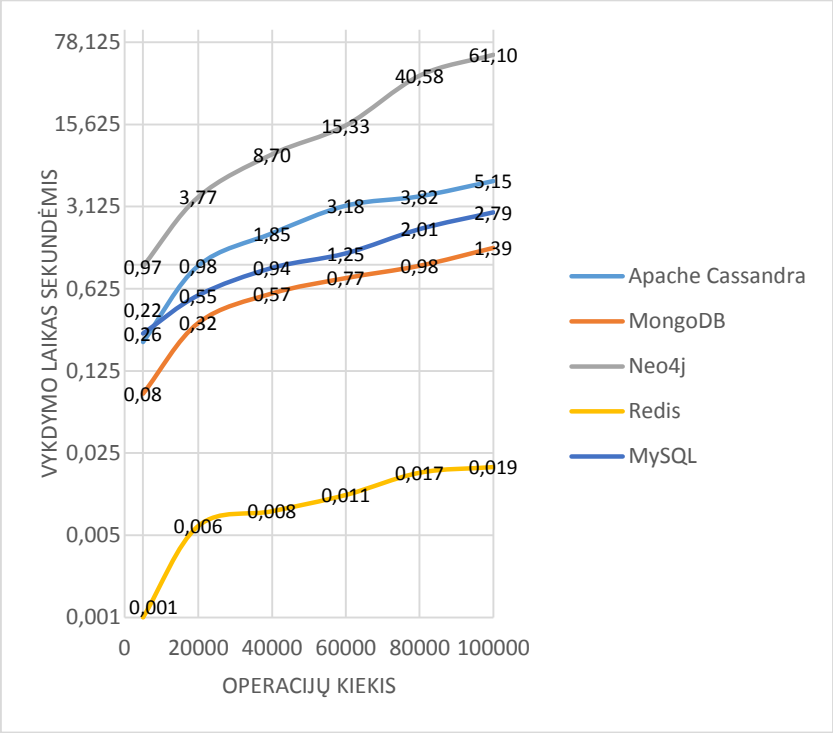
Analizuojant duomenų bazių pastovumo modelius buvo įsitikinta, kad NoSQL duomenų bazės nebūtinai griežtai skiriasi nuo reliacinių duomenų bazių. Tam įrodymas yra Neo4j, kuri kaip ir MySQL laikosi ACID taisyklių. Tačiau ne visoms NoSQL duomenų bazėms tinka ACID modelis. Apache Cassandra ir MongoDB turi kitokius funkcinius prioritetus, kuriuos apima BASE modelis, dėl ko ir yra jo laikomasi. Pasitaiko ir duomenų bazių, kurios nesilaiko jokio pastovumo modelio; šio darbo kontekste tai buvo Redis duomenų bazė.

3.2. Duomenų bazių operacijų su duomenimis vykdymo laiko palyginimas

Duomenų bazių operacijų su duomenimis vykdymo laiko palyginimas bus atliekamas grafikų pagalba. Grafikai buvo sudaryti naudojant atitinkamus greičio rodiklius iš kiekvienos duomenų bazės. Iš viso buvo sudaryti 3 grafikai ir atitinkamai 3 kriterijai palyginimui pagal lentelės 2.1.5, 2.2.5, 2.3.5, 2.4.5, 2.5.5 [Lentelė 1.2.5.1]:

- Duomenų įrašymo greitis lygiagrečiai vykdant nuskaitymą. Grafikas buvo sudarytas pagal analizės metu gautus duomenų įrašų įrašymo greičio rodiklius besilaikant ĮN duomenų bazės apkrovos paskirstymo.
- Duomenų nuskaitymo greitis lygiagrečiai vykdant įrašymą. Grafikas buvo sudarytas pagal analizės metu gautus duomenų įrašų nuskaitymo, lygiagrečiai vykdant naujų įrašų įrašymą greičio rodiklius, besilaikant DN duomenų bazės apkrovos paskirstymo.
- Duomenų nuskaitymo greitis. Grafikas buvo sudarytas pagal analizės metu gautus duomenų įrašų nuskaitymo greičio rodiklius besilaikant TN duomenų bazės apkrovos paskirstymo.

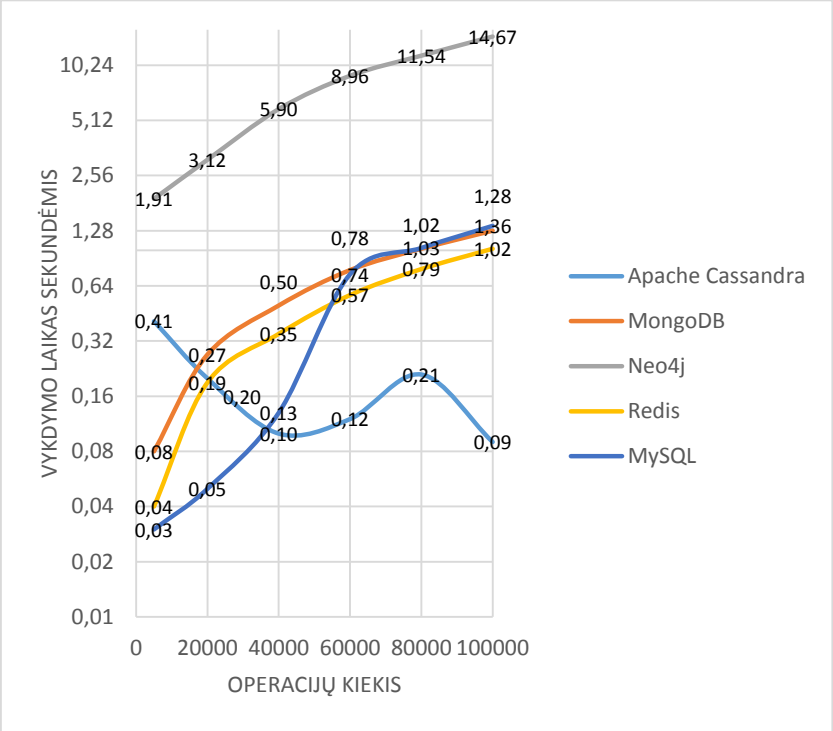
3.2.1. Duomenų įrašymo greitis lygiagrečiai vykdant nuskaitymą



Grafikas 3.2.1. Duomenų įrašymo greitis lygiagrečiai vykdant nuskaitymą.

Pagal duomenų įrašymo greitį lygiagrečiai vykdant jau turimų duomenų nuskaitymą, Redis smarkiai lenkia kitas duomenų bazes. Blogiausiai iš visų pasirodė Neo4j. Iš likusių trijų duomenų bazių blogiausiai pasirodė Apache Cassandra, o kitos dvi, esant OP <= 5000 vienetų, turėjo panašius greičio rodiklius, tačiau esant dideliam OP, MongoDB pasirodė geriau negu MySQL. Taip pat buvo pastebėta, kad Redis beveik tiesiogiai proporcingai auga didinant apdorojamų duomenų kiekiui – esant OP = 5000 buvo vykdoma 0.001 sekundės, o su OP = 100,000 užtruko 0.019 sekundės. [Grafikas 3.2.1]

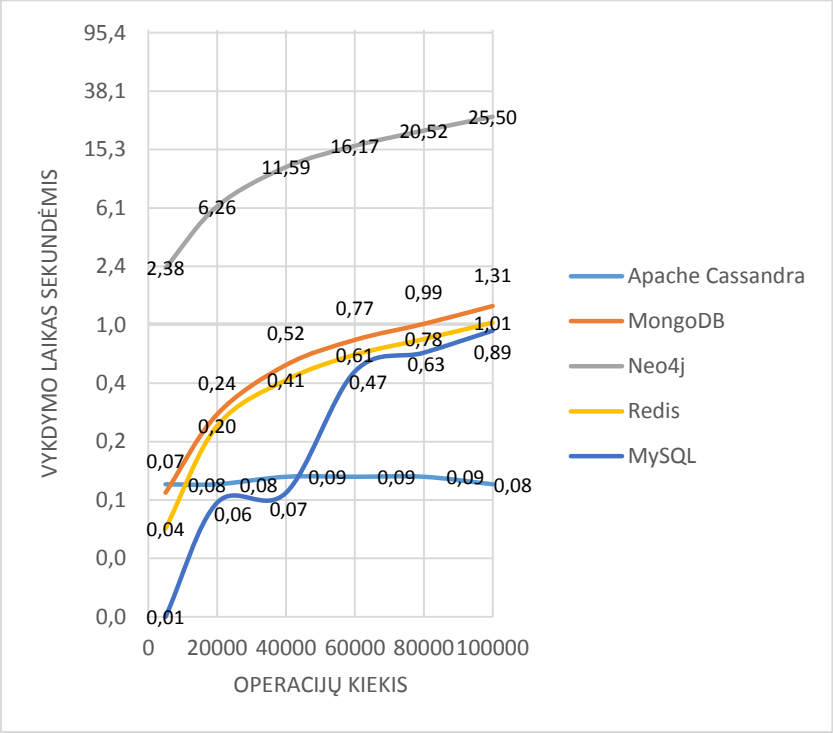
3.2.2. Duomenų nuskaitymo greitis lygiagrečiai vykdant įrašymą



Grafikas 3.2.2. Duomenų nuskaitymo greitis lygiagrečiai vykdant įrašymą.

Pagal duomenų įrašų nuskaitymo greitį, lygiagrečiai vykdant naujų įrašų įrašymą, daugumoje atvejų geriausiai pasirodė Apache Cassandra. Blogiausiai ir vėl pasirodė Neo4j. Kitos duomenų bazės labai panašius rezultatus turėjo. Ir šiuo atveju yra pastebima tendencija, kad Redis vykdymo laikas auga tiesiogiai proporcingai operacijų kiekiui. [Grafikas 3.2.2]

3.2.3. Duomenų nuskaitymo greitis



Grafikas 3.2.3. Duomenų nuskaitymo greitis.

Pagal duomenų įrašų nuskaitymo greitį, lygiagrečiai neįrašinėjant naujų, ir vėl daugumoje atveju geriausiai pasirodė Apache Cassandra. Neo4j ir čia blogiausiai pasirodė. Esant mažam kiekiui apdorojamų duomenų įrašų, geriau negu kiti tvarkosi MySQL, tačiau pradedant nuo 60,000 įrašų nuskaitymo produktyvumas smarkiai krinta, o Apache Cassandra atveju vykdymo laikas išlieka apytiksliai toks pats visais atvejais. Kaip ir praeitais atvejais, Redis ir čia išlaiko savo tiesiogiai proporcinga vykdymo laiko augimą didėjant apdorojamų duomenų kiekiui. [Grafikas 3.2.3]

3.3. Palyginimo rezultatai

Atlikus palyginimus pagal visus kriterijus, visi rezultatai yra suvedami į vieną bendrą lentelę siekiant padaryti išvadas apie pačias duomenų bazės bendrame lygmenyje. Sudarant lentelė bus sujungti 3.2 skyriuje gauti duomenys su lentele 3.1.

Duomenų bazė / Kriterijus	Apache Cassandra	MongoDB	Neo4j	Redis	MySQL
Duomenų bazės diegimo sudėtingumas	2	2	1	2	3
Naudojimosi duomenų baze sudėtingumas	2	2	3	2	2
Duomenų bazės struktūros lankstumas	3	3	2	2	3
Duomenų bazių jungimas	1	3	3	2	3
Pastovumo modelis	BASE	BASE	ACID	–	ACID
Duomenų įrašymo greitis lygiagrečiai vykdant nuskaitymą	#4	#2	#5	#1	#3
Duomenų nuskaitymo greitis lygiagrečiai vykdant įrašymą	#1	#3	#5	#2	#4
Duomenų nuskaitymo greitis	#1	#4	#5	#3	#2

Lentelė 3.3. *Visi duomenų bazių aspektų palyginimo rezultatai, kur skaičiai po simbolio „#“ žymi eilės numerį tarp 1 (geriausiai pasirodžiusi duomenų bazė) ir 5 (blogiausiai pasirodžiusi duomenų bazė) remiantis grafikų duomenimis skyriuose 3.2.1 – 3.2.3.*

Nagrinėtos duomenų bazės pagal diegimo sudėtingumą, naudojimo sudėtingumą beveik nesiskiria, ko negalima pasakyti apie kitus šiame darbe apibrėžtus kriterijus [Lentelė 3.3].

Visos reliacinės duomenų bazės turi duomenų bazių jungimo funkcionalumą, nes be jo nebūtų priežasties normalizuoti duomenų. Tačiau ne visos NoSQL duomenų bazės reikalauja tokio funkcionalumo, nes ne visur būtų prasminga normalizuoti duomenis, o nemažai kur to aplamai neįmanoma padaryti. Jungimo funkcionalumo buvimas yra griežtai susijęs su

pritaikytu pastovumo modeliu. Kadangi kai kurios NoSQL duomenų bazės neturi griežtai apibrėžto pastovumo modelio, poreikis turėti tokį funkcionalumą gali išnykti arba tiesiog būti neįgyvendinamu atitinkamos NoSQL duomenų bazės struktūros kontekste.

Duomenų bazių vykdymo greitis taip pat glaudžiai yra susijęs su pritaikytu pastovumo modeliu. Kadangi dauguma duomenų bazių besilaikančių ACID pastovumo modelio naudoja transakcijų mechanizmą duomenų įrašymui, bendras įrašymo operacijos vykdymo laikas didėja. Šitą teiginį įrodo analizės metu gauti rodikliai iš Neo4j ir MySQL. Neo4j pasirodė blogiausiai iš visų duomenų bazių, nes maža to, kad operacijos yra vykdomos per transakcijas, pats duomenų valdymas reikalauja ganėtinai daug serverio resursų. MySQL, palyginus su Neo4j, neblogai pasirodė, tačiau ne geriau negu visos kitos NoSQL duomenų bazės.

Pagal duomenų nuskaitymo operacijos vykdymo greitį geriausiai pasirodė Apache Cassandra, o pagal įrašymo – Redis. Tačiau tai nereškia, kad jų turimas funkcionalumas gali patenkinti visus vartotojo technologinio sprendimo realizacijos poreikius. Kaip pavyzdį galima paminėti MongoDB duomenų bazę, kuri gali vykdyti užklausas atsižvelgiant į talpinamų dokumentų turinį, ko negali Apache Cassandra, tačiau toks funkcionalumas reikalauja ir papildomų resursų. Todėl renkantis kokią duomenų bazę pritaikyti savo technologiniame sprendime, būtina atsižvelgti ne tik į operacijų vykdymo greitį, bet ir į duomenų bazės turimą funkcionalumą ir koks bus galutinis vykdymo laikas pritaikius jį, nes gali būti taip, kad labiau apsimoka programiškai įgyvendinti atitinkamos duomenų bazės funkcionalumą, o ne naudoti ją pačią, siekiant pabaigoje gauti geresnį greičio rodiklį.

Išvados

Šio darbo analizei buvo parinktos Apache Cassandra, MongoDB, Neo4j ir Redis duomenų bazės, kurios kartu atitinka visus NoSQL tipus pagal Ben‘o Scofield‘o klasifikaciją, ir MySQL reliacinė duomenų bazė, su tikslu padaryti analizės palyginimo etapą labiau informatyvų.

Nustačius duomenų bazių tipus, buvo apibrėžti analizės kriterijai, kuriuos pritaikius paaiškėjo esminės analizuojamų duomenų bazių savybės:

- Darbas su duomenų baze. Šis kriterijus nagrinėja duomenų bazės diegimo efektyvumą ir darbo su pačia duomenų baze sudėtingumą. Pagal šį kriterijų labiausiai išsiskyrė MySQL reliacinė duomenų bazė pagal diegimo efektyvumą ir Neo4j grafo formato duomenų bazė pagal naudojimosi sudėtingumą.
- Duomenų bazės struktūros lankstumas. Šis kriterijus nagrinėja duomenų bazių struktūros pakeitimams atlikti skirtų operacijų funkcionalumą. Pagal šį kriterijų geriausią vertinimą gavo Apache Cassandra (stulpelio formato duomenų bazė), MongoDB (dokumentų talpykla) ir MySQL duomenų bazės.
- Duomenų bazių jungimas. Šis kriterijus nagrinėja duomenų bazių jungimo funkcionalumo buvimą arba nebuvimą ir to priežastis. Atlikus analizę pagal šį kriterijų paaiškėjo, kad jungimo funkcionalumas nėra duomenų bazių charakteristika pagal kuria jas galima tarpusavyje palyginti, nes tokio funkcionalumo buvimą lemia jo poreikis.
- Pastovumo modelis. Šis kriterijus nagrinėja koks buvo panaudotas pastovumo modelis projektuojant duomenų bazę. Analizės eigoje paaiškėjo, kad parinktas duomenų bazės galima suskirstyti į 3 grupes pagal jų pastovumo modelį:
    - Apache Cassandra ir MongoDB yra realizuotos pagal BASE pastovumo modelio taisyklės;
    - MySQL ir Neo4j yra realizuotos pagal ACID pastovumo modelio taisyklės;
    - Redis (žodyno duomenų bazė) negalima priskirti prie jokio pastovumo modelio.
- Operacijų su duomenimis vykdymo greitis. Šis kriterijus nagrinėja duomenų įrašymo ir nuskaitymo operacijų vykdymo greitis ir kaip vienos operacijos įtakoja kitų vykdymą. Pagal šį kriterijų buvo išanalizuoti 3 greičio aspektai ir atitinkamai buvo gautos pagal juos geriausiai pasirodžiusios duomenų bazės:

- Pagal duomenų įrašymo greitį lygiagrečiai vykdant nuskaitymą geriausiai pasirodė Redis duomenų bazė;
- Pagal duomenų nuskaitymo greitį lygiagrečiai vykdant įrašymą geriausiai pasirodė Apache Cassandra duomenų bazė;
- Pagal duomenų nuskaitymo greitį geriausiai pasirodė Apache Cassandra.

Tam, kad sužinoti operacijų su duomenimis vykdymo greičius, kiekvienai duomenų bazei buvo parašytas programinis klientas įgalinantis vartotoja įrašyti ir nuskaityti pageidaujamą kiekį duomenų įrašų tik įvedus skaičių. Analizuojant duomenų bazes pagal šitą kriterijų paaiškėjo kai kurių duomenų bazių silpnosios pusės:

- Neo4j duomenų bazės stabilumas krinta didėjant apdorojamų duomenų įrašų kiekiui, kas blogiausiu atveju gali pasibaigti duomenų bazės kliento užsikirtimu ir apdorojamų duomenų praradimu.
- Įrašant didėlį kiekį duomenų įrašų per Redis, dalis jų yra neįrašoma ir prarandama.

Apibendrinant visą analizę, pagal didžiąją dalį kriterijų Apache Cassandra pasirodė geriau negu kitos analizėje nagrinėtos duomenų bazės. Tačiau MySQL pasirodė ne blogiau už kai kurias duomenų bazės, todėl galima padaryti išvadą, kad negalima vienareikšmiškai pasakyti, kad NoSQL ir reliacinės duomenų bazės yra viena už kitą geresnės kaip duomenų bazių rūšis.

Abstract

The purpose of this analysis was to identify and compare the characteristics of NoSQL database types.

The analysis was conducted by researching various characteristics of distinct types of NoSQL and, for comparison purposes, relational databases. There were two types of characteristics – abstract and simulation based. The abstract characteristics were rated on a scale of 1 to 3, where each value was based on certain database features or their absence. The simulation characteristics involved conducting experiments to measure data insertion and reading speeds in seconds. According to the results, the column store database has a great amount of features and in 2 out of 3 cases displayed the best operation execution speed.

Having summarized, it was concluded that even though the column store database in most cases showed better results than all other databases, it does not prove that NoSQL databases are superior than relational, because the relational database in some cases produced better results than some NoSQL databases.

Literatūros sąrašas

[Buc15] C. Buckler. SQL vs NoSQL: The Differences, URL: http://www.sitepoint.com/sql-vs-nosql-differences/. 29.95 kB, 2015.

[Cas16] Apache Cassandra. The Apache Cassandra Project, URL: http://cassandra.apache.org/. 4.45 KB, 2016.

[Cat11] R. Cattell. Scalable SQL and NoSQL Data Stores, URL, www.cattell.net/datastores/Datastores.pdf, 378 KB, 2011.

[Cel14] J. Celko. Joe Celko's Complete Guide to NoSQL: What Every SQL Professional Needs to Know about Non-Relational Databases. Elsevier Inc, Waltham, 2014.

[Cho13] K. Chodorow. MongoDB: The Definitive Guide, 2nd Edition. O'Reilly Media, Sebastopol, 2013.

[Coo10] B. F. Cooper. YCSB Core Workloads, URL: https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads. 7.86 kB, 2010.

[Cra16] Cram101 Textbook Reviews. Studyguide for Management Information Systems. Content Technologies, Inc., Moorpark, 2016.

[Dat15] C. J. Date. SQL and Relational Theory: How to Write Accurate SQL Code, Third Edition. O'Reilly Media, Sebastopol, 2015.

[Dat16a] Datastax. Installing DataStax Community on Windows, URL: http://docs.datastax.com/en/cassandra_win/3.x/cassandra/cassandraAbout.html. 5.76 kB, 2016.

[Dat16b] Datastax. CQL for Cassandra, URL: https://docs.datastax.com/en/cql/3.0/cql/cql_reference/cqlCommandsTOC.html. 445 kB, 2016.

[EP15] End Point. Benchmarking Top NoSQL Databases. End Point Corporation, New York, 2015.

[Geo15] Georgemtchua. Performance Appraisals: Forced Ranking, URL:

https://peoplecentre.wordpress.com/2015/05/01/forced-ranking/. 14.25 kB, 2015.

[Git16] GitHub. MSOpenTech/redis Releases, URL: https://github.com/MSOpenTech/redis/releases. 11.93 kB, 2016.

[Yen09] S. Yen. NoSQL is a horseless carriage, URL: https://dl.dropboxusercontent.com/u/2075876/nosql-steve-yen.pdf, 0.36 KB, 2009.

[Jef11] K. G. Jeffery. Database Technology, URL: http://www.ercim.eu/medconf/papers/jeffery.html. 8.92 Kb, 2011.

[Kje14] M. Kjellman. An Advanced Cassandra Data Modeling Guide, URL: https://www.hakkalabs.co/articles/cassandra-data-modeling-guide. 12.51 kB, 2014.

[MI16a] MongoDB, Inc. MongoDB Documentation Release 3.0.7, URL: https://docs.mongodb.org/manual/MongoDB-manual.pdf?_ga=1.120092493.43765488.1447578332. 12.7mB, 2016.

[MI16b] MongoDB, Inc. Modify Documents, URL: https://docs.mongodb.org/manual/tutorial/modify-documents/. 25.02 kB, 2016.

[MI16c] MongoDB, Inc. $lookup (aggregation), URL:

https://docs.mongodb.org/manual/reference/operator/aggregation/lookup/. 23.04 kB, 2016.

[MI16d] MongoDB, Inc. Getting Started with MongoDB (C# Edition), URL: https://docs.mongodb.org/getting-started/csharp/. 5.18 kB, 2016.

[MI16e] MongoDB, Inc. MongoDB Download Center, URL: https://www.mongodb.com/download-center, 31.65 kB, 2016.

[Mic08] Microsoft. Transactions (Database Engine), URL: https://technet.microsoft.com/en-us/library/ms190612%28v=sql.105%29.aspx. 14.86 kB, 2008.

[MK14] D. McCreary, Ann Kelly. Making Sense of NoSQL: A guide for managers and the rest of us. Manning Publications Co, New York, 2014.

[Nya14] A.Nayak. MongoDB Cookbook. Packt Publishing, Birmingham, 2014.

[Nor09] K. North. Databases in the Cloud: Elysian Fields or Briar Patch?. URL: http://www.drdobbs.com/database/databases-in-the-cloud-elysian-fields-or/218900502. 33.01 KB, 2009.

[NT16a] Neo Technology, Inc. Download Neo4j, URL: http://neo4j.com/download/. 8.65 kB, 2016.

[NT16b] Neo Technology, Inc. Get Started, URL: http://neo4j.com/developer/get-started. 12.52 kB, 2016.

[NT16c] Neo Technology, Inc. The Neo4j Manual, URL: http://neo4j.com/docs/stable/. 3.88 kB, 2016.

[Ora16a] Oracle Corporation. MySQL, URL: https://www.mysql.com/. 4.7 kB, 2016.

[Ora16b] Oracle Corporation. Download Connector/Net, URL: https://dev.mysql.com/downloads/connector/net/6.9.html. 4.68 kB, 2016.

[Ora16c] Oracle Corporation. What is MySQL?, URL: http://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html, 9.3 kB, 2016.

[PC16] Planet Cassandra. Download DataStax Community Edition — Apache Cassandra™, URL: http://www.planetcassandra.org/cassandra/?dlink=http://downloads.datastax.com/community/datastax-community-64bit_2.1.9.msi. 18.77 kB, 2016.

[Red16] Redislabs. Redis data structure server, URL: http://redis.io. 2.32 kB, 2016.

[RW12] E. Redmond, J. R. Wilson. Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement. The Pragmatic Bookshelf, New Carolina, 2012.

[Sch14] R. Schumacher. Comparing Oracle RAC and NoSQL, URL: http://www.datastax.com/2014/08/comparing-oracle-rac-and-nosql. 107 kB, 2014.

[Sco09] B. Scofield. NoSQL: Death to Relational Databases(?),,, URL: http://www.slideshare.net/bscofield/nosql-death-to-relational-databases, 58.19 KB, 2009.

[SE16] Stack Exchange Inc. StackExchange.Redis, URL: https://github.com/StackExchange/StackExchange.Redis. 19.76 kB, 2016.

[SF12] P. J. Sadalag, M. Fowler. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Professional, Crawfordsville, 2012.

[Til16] Titan. Titan - Distributed Graph Database. URL: http://thinkaurelius.github.io/titan/. 2.2 KB, 2016.

[Tiw11] S. Tiwari. Professional NoSQL. John Wiley & Sons, Inc., Indianapolis, 2011.

[Vai13] G. Vaish. Getting Started with NoSQL. Packt Publishing, Birmingham, 2013.