VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

NoSQL, NewSQL ir SQL duomenų bazių vertinimas ir palyginimas ACID atžvilgiu

NoSQL, NewSQL and SQL databases ACID compliancy assessment and comparison

Baigiamasis bakalauro darbas

Atliko:	4 kurso 2 grupės studentas Artūras Jankus	(parašas)
Darbo vadovas:	lektorius Andrius Adamonis	(parašas)
Recenzentas:	lektorius Donatas Čiukšys	(parašas)

Vilnius, 2016

TURINYS

SANTRAUKA

Šiame darbe nagrinėjamos ir lyginamos tradicinės reliacinės, NoSQL bei NewSQL duomenų bazės ACID savybių rinkinio (atomiškumo, vientisumo, atskirties, patvarumo) atžvilgiu. Tokio pobūdžio lyginimas yra aktualus, nes rinkoje atsiranda vis daugiau produktų skirtų duomenims saugoti, kurie informacinėms sistemoms svarbų savybių rinkinį ACID išpildo skirtingai. Autorius darbe nagrinėja ACID savybių rinkinio reikšmę informacinėms sistemoms, palygina NoSQL ir NewSQL tipo duomenų bazes su tradicinėmis reliacinėmis duomenų bazėmis. Darbe palyginimui pasirinkti CP pagal CAP teoremą duomenų bazių produktai: PostgreSQL, VoltDB ir HBase duomenų bazės. Visos trys pasirinktos duomenų bazės palygintos pagal kiekvieną savybę iš ACID savybių rinkinio. Palyginti būdai, kuriais kiekviena iš duomenų bazių užtikrina ACID rinkinio savybę, įvertinta ar savybė užtikrinama pilnai bei pasiūlyti sprendimų variantai, jei duomenų bazė savybės neužtikrina. Darbo išvadose apibendrinami lygintų duomenų bazių pagrindiniai skirtumai ir panašumai, taip pat, pažymėta, į kokius techninius duomenų bazių kūrėjų sprendimus reikia atkreipti dėmesį lyginant tradicinių reliacinių, NoSQL bei NewSQL atstovus ACID atžvilgiu.

SUMMARY

In this thesis traditional relational, NoSQL and NewSQL databases have been analyzed and compared with regard to ACID set of properties (atomicity, consistency, isolation, durability). This kind of comparison is relevant as new database products appear in the market that offers different fulfillment of essential set of properties to information systems – ACID. The author investigated significance of ACID set of properties to information systems, compared NewSQL and NoSQL databases to traditional relational ones. CP from CAP theorem databases have been selected for comparison in this paper: PostgreSQL, VoltDB and HBase. All three of them have been compared according to each of ACID properties. Methods that databases use to ensure ACID properties have been compared, the ability to ensure properties have been assessed and solutions offered if database does not ensure some properties. In the conclusion chapter, differences and similarities between databases have been highlighted and also the attention has been drawn to the databases engineers' technical decisions that should be taken into account while comparing traditional relational, NoSQL and NewSQL databases in the aspect of ACID.

ĮVADAS

Tyrimo objektas

Šio baigiamojo bakalaurinio darbo tyrimas yra palyginimas. Palyginimas - veiksmas, kurio metu yra tyrinėjami du ar daugiau objektų, siekiant pastebėti skirtumus ar panašumus. Objektus lyginti galima įvairiais aspektais, priklausomai nuo lyginamų objektų savybių, norimo gauti rezultato ar naudos. Šiame darbe, kaip pagrindinis palyginimo aspektas, pasirinktas ACID savybių rinkinys. ACID – akronimas, susidarantis iš atomiškumo (angl. *atomicity*), vientisumo (angl. *consistency*), atskirties (angl. *isolation*) bei patvarumo (angl. *durability*). Lyginamieji objektai - skirtingų tipų duomenų bazės: NoSQL, NewSQL bei tradicinės reliacinės.

Problema

Projektuojant bei įgyvendinant informacines sistemas neišvengiamai tenka rinktis įrankius, kuriais bus kuriama sistema bei kuriuos pati sistema naudos. Duomenų bazė – dažniausiai kertinis sistemos elementas, nes didžiosios dalies informacinių sistemų pagrindinis darbas: būsenų ar informacijos išsaugojimas ir pateikimas. Pagrindinė problema, kurią šio darbo autorius bandys išspręsti, yra tinkamas duomenų bazės pasirinkimas pagal tai, kokias ACID savybes duomenų bazė išpildo ir kaip ji tai atlieka, t.y. bus nagrinėjama, kaip reikia lyginti duomenų bazes ACID atžvilgiu. Taip pat, bus siūlomi sprendimai, kaip naudojantis konkrečia duomenų baze kompensuoti jos trūkumus, kurie pasireiškia, kai ACID savybės nėra išpildomos ar išpildomos tik dalinai.

Aktualumas ir naujumas

Išaugus interneto naudotojų kiekiui ir jų poreikiams, skaitmenizuojant analoginius duomenis, standartiniai ir įprasti duomenų apdorojimo, saugojimo būdai ne visuomet gali išspręsti kylančias problemas ar patenkinti augančius poreikius, dėl skaitmeninių duomenų kiekio didėjimo. Šiuo metu pasaulyje yra daugiau nei 3 milijardai interneto vartotojų, tai yra 3 kartus daugiau, nei buvo prieš 10 metų – internetu jau gali naudotis beveik pusė pasaulio gyventojų [ILS15]. Toks didelis kiekis naudotojų generuoja labai daug turinio (pvz. 9 milijonai žinučių per valandą soc. tinkle „Facebook“ ar 50 milijonų žinučių per dieną soc. tinkle „Twitter“ [ILS15]). Šį turinį tenka saugoti, o vėliau leisti naudotojams jį pasiekti, jame ieškoti. Taip pat, beveik visos įmonės savo veiklos procesuose naudoja informacines sistemas, kurios savo ruožtu naudojasi duomenų bazių paslaugomis. Įmonių

efektyvumas, o tuo pačiu ir kuriama pridėtinė vertė, priklauso nuo to ar efektyviai laiko bei kaštų prasme yra tvarkomasi su duomenimis [OES+08].

Neseniai visoms problemoms spręsti buvo pasitelkiama vieno tipo duomenų bazė – tradicinė reliacinė. Nors ir yra skirtumų tarp kiekvienos iš tradicinių reliacinių duomenų bazių realizacijų, tačiau jų galimybės ir panaudojimo būdai yra daugiau mažiau panašūs. Išaugus esamiems ir atsiradus naujiems poreikiams, imta ieškoti kitokių sprendimų. 2009 metais pradėti kurti NoSQL atstovai, tokie kaip MongoDB, CouchDB, Redis ir kiti. Kai kurie iš jų, pvz. MongoDB šiuo metu pagal populiarumą jau lenkia ilgus metus naudotus tradicinių reliacinių duomenų bazių produktus, tokius kaip PostgreSQL [DBE16]. NewSQL terminas – dar naujesnis. Pirmą kartą šis duomenų bazių tipo pavadinimas buvo paaiškintas 2011 m., organizacijos „451 Group“ dienoraštyje [Asl11]. Kadangi šio tipo duomenų bazės yra dar labai naujos, todėl nespėjusios išpopuliarėti [DBE16], tačiau keliančios diskusijas ir susidomėjimą programų sistemų inžinieriams.

Darbo tikslas ir uždaviniai

Pagrindinis bakalauro darbo tikslas – nustatyti esminius konstrukcinius SQL, NoSQL ir NewSQL duomenų bazių skirtumus ACID savybių išpildymo atžvilgiu. Tam, kad nuosekliai pasiekti šį tikslą, reikia atlikti tokius uždavinius:

- Nustatyti esminius NoSQL ir NewSQL pranašumus bei trūkumus, lyginant jas su tradicinėmis reliacinėmis duomenų bazėmis, tuo pačiu sudarant kriterijų rinkinį palyginimui.
- Pasirinkti duomenų bazių tipų atstovus lyginimui.
- Palyginti pasirinktas duomenų bazes pagal kiekvieną iš ACID savybių.
- Pasiūlyti sprendimus, kuomet ACID savybės neužtikrinamos arba užtikrinamos nepilnai.
- Padaryti palyginimo išvadas.

1. DUOMENŲ BAZIŲ APŽVALGINIS PALYGINIMAS

1.1. SQL

Akronimas SQL žymi programavimo kalbą, kuri yra naudojama duomenų valdymui reliacinėse duomenų bazėse. Jos pirmtakė SEQUEL (angl. *Structured English Query Language*) buvo sukurta 1974m. amerikiečių informatikų Donald Chamberlin bei Raymond Boyce ir pritaikyta dirbti su IBM duomenų baze „System R“ [CB74]. SQL tipo duomenų bazes reikėtų sieti su reliacine duomenų bazės struktūra (modeliu) - griežtai apibrėžta schema, kurios pagrindinis elementas yra dvimatė lentelė su iš anksto žinomu stulpelių (atributų) kiekiu. Reliacinė duomenų bazė – tai tokių lentelių rinkinys. Kiekvienas lentelės stulpelis turi savo pavadinimą ir jame saugomų duomenų tipą. Lentelės ir eilutės susikirtimas duoda tam tikro tipo reikšmę. Reliacinių bazių savybės užtikrina griežtą ir patikimą duomenų struktūrą [Bar05].

1.2. NoSQL

Kartais klaidingai manoma, kad NoSQL principai kaip ir pats duomenų bazių tipas yra nauji, tačiau jie imti taikyti gerokai anksčiau, nei buvo sukurtas reliacinis modelis. NoSQL pradžia siekia 1966 metus, kuomet buvo pradėta naudoti MUMPS (angl. *Massachusetts General – Hospital Utility Multi-Programming System*) programavimo kalba su integruota duomenų baze, kuri yra laikoma NoSQL pradininke [LaF13]. NoSQL pavadinimą 1998m sugalvojo italas Carlo Strozzi, kuris taip pavadino savo reliacinio modelio duomenų bazę, kuri turėjo kitokią nei SQL sąsają [Str98]. Šiuo metu visos NoSQL duomenų bazės šio modelio yra atsisakę. Dabar NoSQL duomenų bazės pagal duomenų struktūrą yra skirstomos į šiuos tipus [MH13]:

- Stulpelių: kiekvienas įrašas yra stulpelis sudarytas iš reikšmių – eilučių.
- Dokumentų: kiekvienas įrašas yra, pavyzdžiui, JSON formato objektas.
- Rakto - reikšmės: kiekvienas įrašas yra primityvaus tipo reikšmė, pasiekiama pagal raktą.
- Grafų: duomenims reprezentuoti yra naudojami grafų elementai – viršūnės ir briaunos.
- Įvairių modelių mišinys: naudojami įvairūs, aukščiau išvardinti duomenų modelio tipai, pagal poreikį.

1.3. NewSQL

Apibrėžimas: „NewSQL yra sutrumpinimas, skirtas įvairioms naujoms, pasižyminčioms plečiamumu ir didele sparta SQL duomenų bazėms apibūdinti“[1] [Asl11]. Prieš tai jos dar buvo vadinamos ScalableSQL, tam, kad atskirti jas nuo įprastų SQL sąsają turinčių duomenų bazių. Pagrindinės NewSQL savybės [Glu15]:

- Būtinai turi turėti SQL sąsają.
- ACID transakcijų palaikymas.
- Išplečiamumas, palaikoma *shared nothing* architektūra.

NewSQL duomenų bazės yra skirstomos į kategorijas pagal sprendimus, kurie yra naudojami tam, kad duomenų bazės pasižymėtų NewSQL savybėmis [Glu15]:

- Naujos architektūros. Tai yra duomenų bazės, kurios yra sukurtos ant naujų platformų ir dažniausiai naudoja *shared nothing* architektūrą. Pavyzdžiai: „Google Spanner“, „VoltDB“ ir kiti.
- Patobulinti duomenų bazių valdymo sistemų varikliai. Naudoja tą pačią SQL sąsają, tačiau gali būti išskirstyti. Pavyzdžiai: „InnoDB“, „TokuDB“.
- Sprendimai leidžiantys paskirstyti duomenų bazes nekeičiant pačių duomenų bazių ir juose esančių duomenų ar jų struktūrų.

[1] „"NewSQL" is our shorthand for the various new scalable/high performance SQL database vendors.“ [Asl11]

2. ACID SAVYBIŲ RINKINIO REIKŠMĖ INFORMACINĖMS SISTEMOMS

2.1. Atomiškumas

Atomiškumo reikšmė transakcijai yra paprasta: ji arba įvyksta, arba ne. Naudotojui sudėjus tam tikras operacijas su duomenų bazėje esančiais duomenimis į transakciją, jos negali būti tik dalinai įvykdytos. Jei iškyla klaida vykdant kažkurią iš transakcijoje esančių operacijų, visa transakcija turi būti atšaukta ir duomenų bazės būsena turi išlikti tokia pati, lyg transakcija nebūtų buvusi vykdoma. Ir atvirkščiai – jei klaidų neiškilo, visi transakcijoje atlikti pakeitimai turi atsirasti iškart vienu metu [HR83].

Neretai informacinėms sistemoms tai yra viena iš svarbiausių savybių: dalykinės srities procesai reikalauja atlikti tam tikrą veiksmų seką. Trivialus ir supaprastintas bankinės operacijos pavyzdys - pervedimas iš vienos sąskaitos į kitą sąskaitą. Jei įvyksta klaida bandant paimti pinigų sumą iš siuntėjo sąskaitos, gavėjo sąskaita negali pasipildyti, kaip ir negalint papildyti gavėjo sąskaitos, siuntėjo sąskaita turi išlikti nepaliesta. Visa operacija gali įvykti tik kaip nedalomas vienetas – tuomet verslo procesas būna įvykdytas teisingai. Kitu atveju, įvyksta brangios klaidos, tokios kaip pvz. virtualios valiutos BitCoin vagystės pasinaudojant neatomiškomis transakcijomis [Sir14].

2.2. Vientisumas

Transakcijos vientisumas reiškia, jog duomenų bazės duomenys yra vientisi tiek prieš transakciją, tiek po jos, tiek jai neįvykus. Šiuo atveju vientisi duomenys – duomenys, kurie nepažeidžia taisyklių. Reliacinių duomenų bazių atveju į taisykles gali įeiti pirminiai raktai, išoriniai raktai, trigeriai ir kt. [HR83].

Duomenų bazių taisyklėmis yra užtikrinama tinkama bei norima informacinių sistemų veikla. Informacinės sistemos neretai pasitiki duomenų bazių teikiamomis galimybėmis užtikrinti, kad duomenys bazėje visą laiką atitiktų sistemos apibrėžtas taisykles, pvz. pagal pirminį raktą nesidubliuotų lentelėje esantys duomenys, būtų užtikrinta, kad privalomas lentelės eilutės elementas būtinai egzistuotų ar būtų įvykdyta tam tikra operacija, tam tikru nustatytu metu (trigerių veikimas). Jei tokį funkcionalumą užtikrina duomenų bazė, to nebereikia daryti informacinei sistemai – užtenka tinkamai naudotis duomenų bazės teikiama sąsaja.

2.3. Atskirtis

Atskirtis transakcijose – tai savybė, kuri nurodo, kiek kitiems naudotojams yra matomi ne jų transakcijoje esantys pakeitimai. Atskirtis gali būti skirtingų lygių, kaip ir to pačio lygio atskirties įgyvendinimas gali būti kitoks, lyginant skirtingas duomenų bazes [Kle14]. Galima išskirti keturis pagrindinius lygius (jų yra ir daugiau) pradedant nuo silpniausios atskirties ir baigiant stipriausia [Pos16]:

- Nepatvirtintų transakcijų skaitymo lygmuo (angl. *read uncommitted*) – suteikiama galimybė skaityti duomenis, kurie yra jau pakeisti, bet dar nepatvirtinti (angl. *uncommitted*).
- Patvirtintų transakcijų skaitymo lygmuo (angl. *read committed*) – leidžiama skaityti tik tuos duomenis, kurie jau yra patvirtinti (angl. *committed*).
- Pakartotinio skaitymo lygmuo (angl. *repeatable read*) – leidžiama skaityti tik tuos duomenis, kurie patvirtinti ir draudžiama keisti duomenis, kol nepasibaigė transakcija, kuri tuos duomenis nuskaitė.
- Serijinis operacijų lygmuo (angl. *serializable*) – operacijos atliekamos taip, tarsi būtų vykdomos viena po kitos paeiliui, be jokio prasikeitimo.

ACID reikalauja aukšto atskirties lygmens – viena vykdoma transakcija negali žinoti nieko apie kitą vykdomą transakciją ir jos turi būti vykdomos paeiliui arba bent rezultatas turi būti toks, lyg jos būtų atliktos paeiliui [HR83].

Nepakankamas atskirties užtikrinimas gali sukelti įvairius nepageidaujamus reiškinius, tokius kaip [Pos16]:

- Nešvarus duomenų nuskaitymas (angl. *dirty read*). Tai situacija, kuomet nuskaitomi jau transakcijoje pakeisti, bet dar nepatvirtinti duomenys. Transakcija gali būti ir atšaukta, todėl tikėtina, jog naudotojas nuskaitė neteisingus (pvz. nepilnai pakeistus) duomenis.
- Skirtingų duomenų nuskaitymas transakcijoje (angl. *non - repeatable read*). Situacija, kuomet transakcijoje du ar daugiau kartų nuskaitomi tie patys duomenys (pvz. ta pati eilutė, ją identifikuojant pagal vidinį raktą), o grąžinamos reikšmės – nesutampa, nes kita transakcija spėjo juos pakeisti tarpe tarp nuskaitymų.
- Skirtingų duomenų rinkinių nuskaitymas transakcijoje (angl. *phantom read*). Situacija, kuomet transakcijoje du ar daugiau kartų įvykdoma ta pati užklausa, bet ji grąžina skirtingus duomenų rinkinius (pvz. skirtingą skaičių eilučių).

Informacinės sistemos įprastai pačios atskirties nevaldo, o tiesiog pasitiki duomenų baze: informacinė sistema, sulaukusi jos naudotojo užklausos, tiesiogiai kreipiasi į duomenų bazę, inicijuoja naują transakciją ir ten atlieka norimus veiksmus, o po to ją patvirtina. Įprastai tikimąsi, jog net esant lygiagrečioms užklausoms į tuos pačius informacinės sistemos duomenis, viskas bus atlikta tinkamai – neatsiras jokie, aukščiau paminėti, nepageidaujami reiškiniai arba bent jau jie bus suvaldyti, kad duomenys nebūtų sugadinti. Jei duomenų bazė to neatlieka, užklausų į informacinę sistemą konkurentiškumą tektų valdyti informacinių sistemų kūrėjams.

2.4. Patvarumas

Transakcijų patvarumas – tai užtikrinimas, jog įvykdžius transakciją, jos atlikti pakeitimai niekuomet neišnyks [HR83]. Sistemos gedimai, elektros dingimas ir kt. negali turėti įtakos jau įvykdytoms transakcijoms. Ši savybė yra užtikrinama ne tik programine įranga, bet ir techninė įranga, tačiau šiame darbe aktualus patvarumo užtikrinimas tik programinės įrangos atžvilgiu.

Didžiajai daliai informacinių sistemų yra privaloma, jog naudotojui pranešus apie sėkmingą veiksmą, to veiksmo pasekmės išliktų visam laikui. Kitu atveju gali išaugti naudotojo nepasitenkinimas, tiek naudotojas, tiek sistemos valdytojai gali patirti finansinių ir kitokių nuostolių.

3. NOSQL IR NEWSQL PRANAŠUMAI PRIEŠ TRADICINES RELIACINES DUOMENŲ BAZES

3.1. NoSQL

Priklausomai nuo duomenų tipo, kuris bus saugomas duomenų bazėse, reikėtų rinktis ir pačią technologiją. Ne visoms informacinėms sistemoms reikalinga standartizuota duomenų struktūra, nes ne visuomet yra poreikis veiksmams su atskirais įrašo laukais. Taip pat, duomenys gali turėti gilią hierarchinę struktūrą. Tokio tipo duomenis patalpinti į reliacinę duomenų bazę – iššūkis, kurį įveikiant gali atsirasti klaidų, tokius duomenis tenka „surinkti“ atliekant užklausas (pvz. JOIN operacijas). Šiuo atveju labiau tinka NoSQL duomenų bazės, kurios būna bent keturių tipų (jie išvardinti 1.2 skyriuje, o kiekvienas iš jų tinka vis kitokio tipo duomenims talpinti), tarp kurių yra ir dokumentų duomenų bazės. Dokumentas – JSON formato objektas, kuriuo galima atvaizduoti pačią sudėtingiausią hierarchinę struktūrą ir ją vienu įrašymo veiksmu patalpinti į NoSQL duomenų bazę. Taip pat, spėjant, jog informacinėje sistemoje duomenų struktūrų aprašai gali keistis, derėtų rinktis NoSQL, nes reliacinėje duomenų bazėje norint patalpinti nors ir šiek tiek kitokios struktūros duomenis, tenka keisti visą schemą, kuomet NoSQL tai yra nebūtina – struktūra yra dinamiška [Das13].

Dar viena silpnoji reliacinių duomenų bazių vieta – sparta ir pasiekiamumas (angl. *availability*). Didėjant duomenų ar naudotojų kiekiui, reliacinių duomenų bazių atsako laikas neišvengiamai ilgėja. Todėl informacinių sistemų kūrėjams tenka spręsti paskirstymo problemas. Tradicinės reliacinės duomenų bazės įprastai gali būti plečiamos tik vertikaliai: tobulinant esamą serverį, kuriame veikia esama duomenų bazė. Tai yra brangus plėtimo būdas. NoSQL suteikia galimybę plėsti duomenų bazę horizontaliai – pridedant naujus serverius, kuriuose duomenų bazės duomenys yra replikuojami arba skaidomi į dalis (angl. *sharding*). Taip pat, galima pasiekti linijinį plėtimą (angl. *linear scalability*), kuomet užtenka pridėti naują serverį nekeičiant aplikacijos kodo ir kt. [Das13].

Apibendrinant, NoSQL duomenų bazės įgyja pranašumą, kuomet nėra poreikio duomenis griežtai struktūrizuoti ar kai pvz. duomenų struktūra yra hierarchinė. Taip pat, siekiant didesnės spartos ir pasiekiamumo, galima duomenų bazę paskirstyti horizontaliai ir taip laimėti ne tik spartos bei pasiekiamumo, bet tuo pačiu ir saugumo (duomenis replikuojant), ir sutaupyti lėšų.

3.2. NewSQL

NewSQL pagrindinis privalumas prieš tradicines reliacines duomenų bazes, kaip ir NoSQL atveju – galimybė paskirstyti duomenų bazę, tačiau tai galima padaryti neatsisakant SQL programavimo sąsajos.

Paskirstant NewSQL duomenų bazes išlaikoma ne tik SQL sąsaja, tačiau ir transakcijos. NewSQL duomenų bazių kūrėjai įprastai renkasi vieną ar daugiau iš keleto konkurentiškumo kontrolės metodų [LJ14]:

- MVCC (angl *Multi Version Concurrency Control*).
- OCC (angl. *Optimistic concurrency control*).
- BTCC (angl. *Basic Timestamp Concurrency Control*).
- Konkurentiškumo atsisakymas, jei duomenų bazė veikia darbinėje atmintyje.

MVCC veikimo principas grįstas ankstesnių objektų versijų saugojimu. Vietoj to, kad būtų perrašomi egzistuojantys objektai, yra sukuriamos visiškai naujos tų objektų versijos. Tai užtikrina galimybę saugoti duomenis nuosekliai ir kompaktiškai pagrindinėje serverio atmintyje bei leidžia tuo pačiu metu skaityti objektus, kurie tuo metu yra rašomi į duomenų bazę. BTCC veikia lygindamas dabartinį duomenų nuskaitymo/įrašymo laiką su paskutiniuoju nuskaitymo/įrašymo laiku. Skaitymo ar rašymo operacija yra atmetama, jei dabartinio skaitymo ar rašymo operacijos laikas yra senesnis, nei paskutinio įrašymo laikas. Taip pat, rašymo operacija atmetama jei jos laikas yra senesnis, nei paskutinio nuskaitymo laikas. Toks metodas leidžia nuspręsti, kiek dabartinė apdorojama transakcija yra „sena“ lyginant su duomenimis, kuriuos ta pati transakcija nori keisti bei leidžia surikiuoti transakcijas pagal jų sukūrimo laiką. OCC metodas veikia sekdamas rašymo/skaitymo operacijas ir laikinai išsisaugodamas visas rašymo operacijas. Kuomet transakcija yra bandoma įvykdyti, sistema patikrina, ar transakcijos skaitymo ir rašymo operacijos nesikerta. Jei ne, tuomet transakcija įvykdoma, jei taip, tuomet atšaukiama ir pradedama iš naujo.

NewSQL duomenų bazės leidžia paskirstyti duomenis (tokiu būdu laimint spartos, pasiekiamumo) ir replikuoti duomenis (taip laimint spartos, pasiekiamumo bei saugumo) nedarant didelių kompromisų – neatsisakant transakcijų ir įprastos SQL sąsajos.

4. TRADICINIŲ RELIACINIŲ DUOMENŲ BAZIŲ PRANAŠUMAI PRIEŠ NOSQL IR NEWSQL

4.1. NoSQL

Didžiausi reliacinių duomenų bazių privalumai lyginant su NoSQL plaukia iš to, jog pastarojo tipo duomenų bazės neturi pilno ACID išpildymo, kai tuo tarpu praktiškai visos SQL duomenų bazės tai siūlo. Tai reiškia, jog informacinių sistemų kūrėjai sutaupo išteklių, nes nereikia reikiamų savybių bandyti užtikrinti aplikacijos lygmenyje.

SQL duomenų bazės turi daug daugiau vartotojų, o tai užtikrina daug geresnį produkto palaikymą ir trumpina kilusių problemų su pasirinktu produktu, pvz. klaidų, taisymo trukmę. Norint pagrįsti šį teiginį, atliekamas nesudėtingas patikrinimas. Įvedus užklausą „MySQL“ (tai viena dažniausiai naudojamų SQL duomenų bazių [DBE16]) į populiariausią internetinį paieškos variklį rezultatų skaičius viršija 85 milijonus, kai tuo tarpu įvedus „MongoDB“ (tai populiariausia NoSQL duomenų bazė [DBE16]) rezultatų skaičius kur kas kuklesnis – vos daugiau nei 11 milijonų. Iš to galima daryti išvadą, kad kurti informacines sistemas pasitelkiant SQL duomenų bazes galima sparčiau, nes rasti sprendimą ar gauti pagalbos per trumpą laiką yra lengviau.

Įprastas SQL veiksmas JOIN nėra palaikomas NoSQL duomenų bazėse. Todėl yra įprasta laikyti duomenis, kuriuos SQL atveju išskaidome per lenteles, vienoje vietoje, taip didinant netvarką ir tuo pačiu neefektyvumą, nes bandant pasiimti duomenis, reikia juos imti visus, kai tuo pačiu metu SQL atveju, galima pasiimti tik objektą su nuorodomis į kitus objektus, kitose lentelėse. NoSQL inžinieriai sprendžia JOIN problemas ir bando įgyvendinti šį veiksmą kiekvienoje informacinėje sistemoje atskirai. Tačiau daugelio metų įdirbis į SQL duomenų bazes leidžia atlikti JOIN operaciją daug efektyviau [Way12].

Griežta įvedamų duomenų struktūra taip pat didelis SQL privalumas. Turint omenyje, jog į NoSQL duomenų bazes galima dėti beveik bet kokius objektus (pvz. JSON dokumentus) ir jų struktūra nėra tikrinama, tai reiškia, kad visi informacinę sistemą kuriantys komandos nariai turi apsibrėžti tam tikras taisykles, tam tikrą protokolą, pagal kurį bus sprendžiami objektų laukų pavadinimai, struktūra ir pan. [Way12]. Tai be abejo, yra nemažas trukdis ir neefektyvus išteklių naudojimas.

Taip pat SQL turi nemažai patogių funkcijų, tokių kaip COUNT, SUM ir pan., kurios iškart grąžina norimą rezultatą, kurį informacinėje sistemoje galima pateikti vartotojui. NoSQL duomenų bazės tokių funkcijų įprastai neturi, todėl visuomet būtina pasiimti reikiamą dalį informacijos ir tik tuomet, pačioje informacinėje sistemoje ją apdorojus, pateikti norimą rezultatą vartotojui. Tai ne tik žmogiškųjų resursų švaistymas, tačiau ir serverių, kuriuose veikia IS betikslis resursų naudojimas [Way12].

Tradicinės reliacinės duomenų bazės turi nemažai pranašumų prieš NoSQL: ACID palaikymas, gerokai daugiau informacijos (bei didesnė galimybė sulaukti pagalbos) internete, JOIN operacijų palaikymas, griežta duomenų struktūra – taip pat gali būti privalumas.

4.2. NewSQL

Nemaža dalis NewSQL produktų (MemSQL, VoltDB, GemFire XD ir kt.) darbui naudoja pagrindinę serverių/kompiuterių atmintį. Nors ir atminties kainos krenta [PCP15], tačiau ne taip žymiai, jog tai nesudarytų gerokai didesnių finansinių išlaidų, lyginant su tradicinėmis reliacinėmis duomenų bazėmis, kurios dažniausiai dirba kietajame diske (nors yra ir turinčių galimybę veikti ir darbinėje atmintyje, pvz. Oracle duomenų bazė nuo 12c versijos).

Taip pat ankstesniame, 3.2 skyriuje, buvo minėta, kad yra duomenų bazių, kurios nenaudoja jokio konkurentiškumo valdymo mechanizmo atskirties užtikrinimui. Nors tokios duomenų bazės gali užtikrinti atskirtį serijinių operacijų lygiu, tačiau toks lygis pasiekiamas tiesiog neleidžiant jokių konkurentinių užklausų. Paskirstytos duomenų bazės serveris (angl. *node*) gali apdoroti tik vieną transakciją vienu metu – nelieka jokio lygiagretumo iki tol, kol naudojamas tik vienas serveris. Tradicinės reliacinės duomenų bazės įprastai naudoja įvairius atskirties užtikrinimo mechanizmus, todėl vienu metu gali apdoroti ne vieną transakciją, jei tų transakcijų veikla nesikerta [Pag10].

Susidūrus su problema kuriant IS ir naudojant NewSQL duomenų bazę, sudėtingiau gauti greitos pagalbos. Tam įrodyti pakanka įvesti kelių populiariausių NewSQL atstovų pavadinimus (VoltDB, NuoDB, Clustrix) į populiariausią paieškos sistemą. Rezultatų grąžinama labai nedaug – iki 200 tūkst. kiekvienai duomenų bazei.

Rinkoje yra nemažai tradicinių reliacinių duomenų bazių, kurios kuriamos jau 30 ir daugiau metų (pvz. Oracle, kuriai jau 37 metai [Ora16]), todėl tai labai stabilūs produktai. To negalima pasakyti apie NewSQL atstovus, kurių ilgiausias galimas kūrimo laikotarpis – 5 metai, nes tik 2011

metais NewSQL terminas pirmą kartą buvo paaiškintas. Tai reiškia, jog kuriant IS su NewSQL atstovais galimi dažni susidūrimai su klaidomis.

Apibendrinant, akivaizdu, kad informacijos apie NewSQL duomenų bazes – labai negausu, o ir klaidų tikimybė didesnė. Tai sukeltų problemų ir pareikalautų daugiau laiko šias duomenų bazes panaudojant IS kūrime. Taip pat, didesnė tikimybė, jog naudojant NewSQL serveriuose teks turėti daugiau darbinės atminties, kuri didina IS kūrimo projekto išlaidas. Kitas trūkumas, priklausomai nuo to, kaip valdomas konkurentiškumas ir koks konkretus NewSQL produktas pasirenkamas, gali būti ir lėtesnis transakcijų apdorojimas, jei priimamas sprendimas NewSQL duomenų bazės, kuri užtikrina atskirtį neleisdama konkurentiškų užklausų, neskaidyti.

5. DUOMENŲ BAZIŲ LYGINIMAS ACID ATŽVILGIU

5.1. Duomenų bazių lyginimui parinkimas

Iki 2013 metų vienvaldės populiarumo reitingų lyderės buvo tradicinės reliacinės duomenų bazės [DBE16] – vadinasi, didžioji dalis IS remdavosi būtent jomis. Tokių duomenų bazių atstovai siūlydavo ir vis dar siūlo universalų produktą, tinkantį tiek vykdyti transakcijoms realiuoju laiku (angl. *OLTP*), tiek duomenų analizei realiuoju laiku (angl. *OLAP*). Taip buvo dėl keleto priežasčių: kaštų taupymo, lengvesnio suderinamumo, lengvesnio produktų marketingo ir pardavimų. Tačiau šiuo metu kyla vis daugiau problemų, kurių negalima išspręsti arba jos sprendžiasi neefektyviai naudojant produktus „tinkančius viskam“ [Sto05]. Todėl tenka rinktis produktą, kuris tenkina konkrečios informacinės sistemos poreikius.

Iš to plaukia, jog lyginant duomenų bazes, jas reikia pasirinkti tokias, kurios gebėtų atlikti tokias pat funkcijas: daugiau ar mažiau turėtų tą pačią paskirtį, tam tikru aspektu. Kitu atveju lyginimas bus neprasmingas, o jo rezultatai nepanaudojami praktikoje, kuriant informacines sistemas. Šio darbo autorius duomenų bazės paskirties įvertinimui pagal duomenų bazės elgesį išskaidytoje sistemoje pasirinko CAP teoremą, suformuluotą Eric Brewer, 1998 metais. Ši teorema pasirinkta duomenų bazės paskirčiai nustatyti todėl, kad renkantis duomenų bazę konkrečiai informacinei sistemai, reikia nuspręsti, kiek svarbu užklausos atsako teisingumas, o kiek – garantija, jog tas atsakas visuomet bus. CAP – akronimas, susidedantis iš vientisumo, kurio prasmė yra kitokia, nei ACID rinkinyje, pasiekiamumo (angl. *availability*) bei galimybės veikti duomenų bazių particijų atveju (angl. *partition tolerance*) – kuomet išskaidytoje sistemoje nutrūksta ryšiai tarp mazgų. Vientisumas CAP teoremoje reiškia, kad išskaidytoje sistemoje duomenys turi tik vieną naujausią versiją [Bre12]. Ilgą laiką buvo manyta, jog autorius kurdamas šią teoremą griežtai išreiškė, jog duomenų bazė gali turėti tik kažkurias dvi savybes iš CAP rinkinio. Paradoksalu, tačiau tai pasitarnavo programų sistemų inžinieriams projektuojant sistemas – buvo nagrinėjama, kokių duomenų bazių savybių informacinės sistemos gali atsisakyti, o kokių ne, iki tol tai buvo daroma vangiai. Tačiau po dvylikos metų, CAP teoremos autorius patikslino savo išreikštas mintis ir jas pritaikė šiandienos situacijai – viskas yra šiek tiek lanksčiau, nei „dvi iš trijų“ taisyklė [Bre12], tačiau esminis dalykas, skirstymas pagal tai, kaip duomenų bazė elgiasi, kuomet trūksta ryšiai tarp mazgų išliko.

Darbo autorius lyginimui pasirinko duomenų bazes, kurios atsisako pasiekiamumo ir išsaugo vientisumą atsiradus particijoms, t.y. CP duomenų bazės, remiantis CAP teorema. Toks sprendimas

priimtas todėl, kad particijų atsiradimo tikimybė yra sąlyginai nedidelė, palyginus ją su kitomis problemomis, kurios gali kilti dėl programuotojų kaltės ar pan., todėl atsisakyti vientisumo – neverta [Sto10]. Taip pat, tai bus duomenų bazės, kurios skirtos ar labiau orientuotos į transakcijų vykdymą realiu laiku t.y. OLTP tipo. Be abejo, norint pademonstruoti palyginimo kriterijus, reikia pasirinkti SQL, NewSQL, NoSQL duomenų bazes atstovaujančius produktus. Reikia nepamiršti, kad tiek SQL, tiek NewSQL ir NoSQL tipų duomenų bazės realizuojamos skirtingai, todėl palyginus konkrečias tokių tipų duomenų bazes negalima daryti išvadų apie visas rinkoje egzistuojančias tokio tipo duomenų bazes.

Darbo autorius NewSQL atstovu pasirinko VoltDB duomenų bazę. Tai yra viena populiariausių naujos architektūros NewSQL duomenų bazių [DBE16], pirmą kartą pristatyta 2010 metais, veikianti operatyvinėje atmintyje. Ši duomenų bazė buvo pasirinkta dėl savo populiarumo ir dėl to, kad tai yra visiškai naujas, naujos architektūros, NewSQL produktas. Tai CP tipo pagal CAP duomenų bazė pritaikyta OLTP sistemoms [Vol16a]. Šios duomenų bazės kūrėjai teigia, jog ji pilnai palaiko ACID. Šis faktas bus patikrintas atliekant palyginimą.

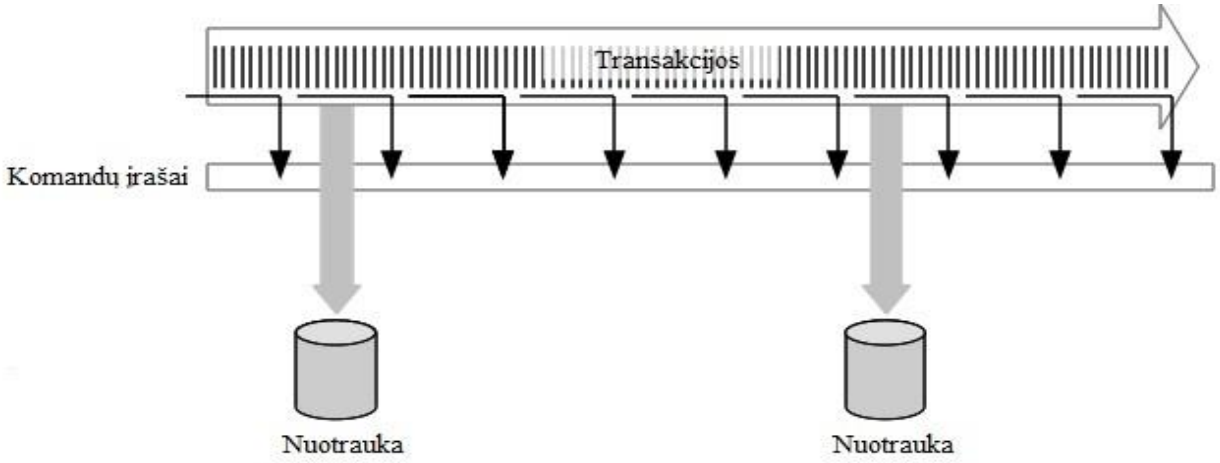
Atstovauti NoSQL duomenų bazes buvo pasirinkta duomenų bazė HBase. Tai yra atviro kodo Apache produktas, kurio pirmoji stabili versija pasirodė 2015 metas. Tai yra taip pat CP tipo duomenų bazė [Sah15], kuri dažniausiai naudojama OLTP sistemose [Nol14]. Viena iš pagrindinių priežasčių, dėl kurios buvo pasirinkta būtent ši duomenų bazė yra tai, jog ji viena iš nedaugelio CP tipo NoSQL duomenų bazių – kitos populiarios duomenų bazės, tokios kaip MongoDB, Cassandra yra AP tipo. Ši duomenų bazė priskiriama rakto – reikšmės tipo NoSQL duomenų bazėms, o duomenis saugo stulpeliais (naudotojui atvaizduojama eilutėmis). Duomenų bazės kūrėjai teigia, jog tai nėra ACID palaikanti duomenų bazė, tačiau dalinai užtikrinanti kai kurias ACID savybes. Atliekant palyginimą tai bus išanalizuota.

Tradicinėms reliacinėms duomenų bazėms atstovauti parinkta populiari, atviro kodo PostgreSQL duomenų bazė. Pasirinkimo priežastis – puiki šios duomenų bazės dokumentacija bei populiarumas. Ši duomenų bazė – standartiškai nėra skaidoma, įprastai veikia viena jos kopija, kurioje patalpinti visi reikalingi duomenys. Todėl jos priskirti kažkuriai iš CAP kategorijų – neįmanoma. Tačiau kaip ir visos duomenų bazės, kurios saugo duomenis eilutėmis, o ne stulpeliais, tai yra OLTP pritaikyta duomenų bazė [Mak15].

5.2. Palyginimas atomiškumo atžvilgiu

Tradicinė reliacinė duomenų bazė PostgreSQL yra žinoma kaip viena iš griežčiausiai ACID palaikančių duomenų bazių, tad pirmoji ACID savybė – atomiškumas, nėra išimtis. Ši savybė PostgreSQL duomenų bazėje užtikrinama įrašant būsenų pakitimus į įrašų registrą (kietajame diske) prieš juos įrašant į pagrindinį duomenų bazės failą (angl. *write-ahead logging*) [Pos16]. Ši technika leidžia užtikrinti, jog net netikėtai nutrūkus duomenų bazės veikimui ir vėl paleidus ją iš naujo, pagal išsaugotus įrašus registre, ji galės arba atmesti, arba pabaigti vykdyti buvusių transakcijų veiksmus, taip išlaikant transakcijos atomiškumą. Taigi, tinkamai veikiant techninei įrangai, atomiškumas šioje duomenų bazėje užtikrinamas visiškai.

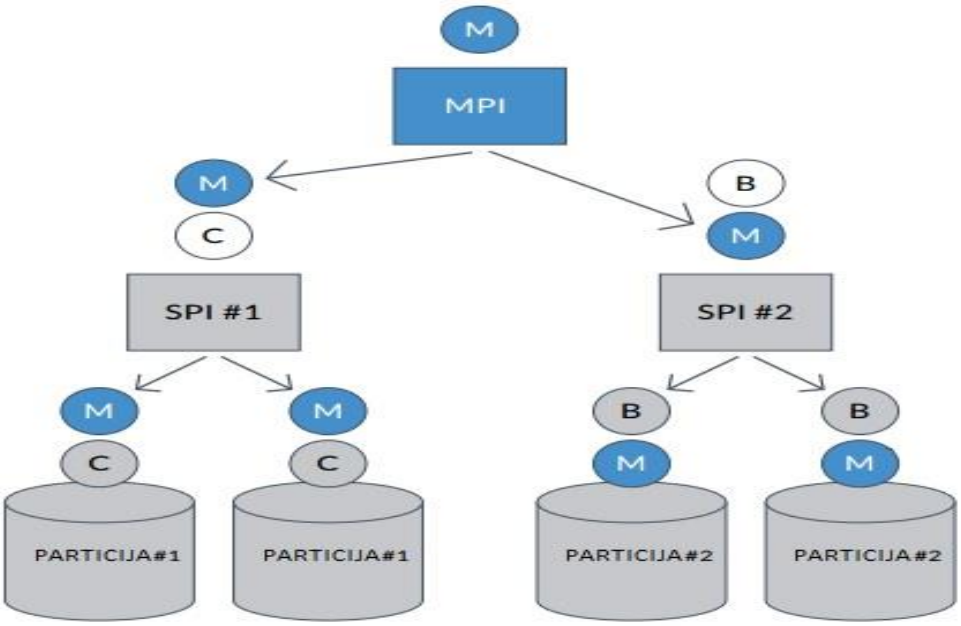
Darbo autoriaus pasirinktos NewSQL atstovės VoltDB kūrėjai taip pat teigia, jog visiškai palaiko atomiškumą. Tačiau technikos naudojamos šiai savybei užtikrinti – kitokios nei PostgreSQL duomenų bazėje. Užtikrinant atomiškumą VoltDB naudojamos dvi technikos: momentinės duomenų nuotraukos saugomos kietajame diske bei komandų įrašų registras [Vol16b]. Momentinė duomenų nuotrauka - pilna visų duomenų bei pačios aplikacijos (duomenų bazėje išsaugotų procedūrų ir t.t.) nuotrauka tam tikru laiku. Akivaizdu, kad darant tokias nuotraukas po kiekvienos transakcijos, kentėtų sparta, todėl reikia papildomos, „lengvesnės“ apsaugos tarpe tarp nuotraukų – komandų įrašo registro. Tai pagal paskirtį panašus registras į pakitimų (arba kitaip, duomenų) įrašo registrą naudojamą PostgreSQL duomenų bazėje, tačiau šiuo atveju saugomos ne būsenos (duomenys), o vykdytos



Paveikslėlis 1 VoltDB komandų įrašų registravimo schema

komandos. Tai yra, norint atkurti duomenis, reikia pasiimti paskutinę nuotrauką ir registre išsaugotas komandas vykdyti, kad gautumėme rezultatą, kai tuo tarpu būsenų įrašų registre – būtent rezultatas ir yra saugomas. Privalumai saugant tik komandas yra akivaizdūs. Kadangi šį veiksmą reikia atlikti prieš įvykdant transakciją, nereikia laukti, kol užklausos rezultatas bus sugeneruotas ir išsaugotas į kietąjį

diską, kaip kad pakitimų įrašų registre. Gavus užklausą ją galima tiesiog išsaugoti į komandų įrašų registrą. Taip pat, pačio įrašo dydis – gerokai mažesnis, nes jame saugojama tik pati komanda, o ne pakeisti duomenys. Kita vertus, norint atkurti kažkurio konkretaus laiko taško duomenis – šis veiksmas užtruks ilgiau, nei PostgreSQL, nes bus privaloma imti naujausią nuotrauką ir pritaikyti visas komandas iki tam tikro laiko taško.



Paveikslėlis 2 Komandų vykdymas skaidytoje VoltDB sistemoje. M, C, B - komandos. M – multiparticinė komanda. MPI - multiparticinis iniciatorius (angl. multi-partition initiator). SPI - vienos particijos iniciatorius (angl. single-partition initiator). Particijos replikuotos: dvi #1 ir dvi #2.

Kadangi VoltDB duomenų bazę galima tiek replikuoti, tiek paskirstyti, ši taip pat turi užtikrinti atomiškumą net ir toms operacijoms, kurios paliečia ne vieną mazgą, o kelis (žr. į paveikslėlį nr. 2). Ši problema sprendžiama pasitelkus dviejų fazių transakcijos patvirtinimą. Pirmoje fazėje multiparticinis iniciatorius paskirsto užduotis kiekvienos particijos iniciatoriams. Jei kiekvienas mazgas gali įvykdyti (nepažeidžiant taisyklių ir t.t.) duotas komandas, apie tai praneša užduotį skyrusiam iniciatoriui ir laukia nurodymo įvykdyti transakciją. Laukimo metu mazgai nieko nebevykdo, tik laukia. Multiparticinis iniciatorius iš visų vaikinių iniciatorių gavęs sėkmės pranešimą, visiems mazgams liepia galutinai įvykdyti transakciją, arba gavęs nesėkmės pranešimą nors iš vieno mazgo, liepia visiems atšaukti transakciją [Vol16b]. Akivaizdu, jog vienam mazgui nėra esminio skirtumo, ar jis atlieka dalį visos operacijos dalindamasis darbą su kitais mazgais, ar pats vienas visą operaciją – tarp particinės operacijos atomiškumui tai įtakos neturi.

Trečioji duomenų bazė, HBase, užtikrina atomiškumą tik eilutės lygmenyje [Hba16]. Eilutė HBase duomenų bazėje – stulpelių šeimų rinkinys. Stulpelių šeima – rinkinys iš rakto ir reikšmės. Pvz. viena eilutė gali talpinti visą reikalingą informaciją apie asmenį, jo duomenis skirstant į stulpelių šeimas, tokias kaip susisiekimo informaciją, darbinė patirtis ar pan. Susisiekimo informacija gali turėti raktus „el.paštas“, „gyvenamoji vieta“ ir t.t., tuo pačiu, be abejo, prie kiekvieno rakto saugant reikšmę. Taigi, saugant tokią eilutę HBase garantuoja, jog ji arba išsisaugos pilnai, arba nepakis visai. Šį funkcionalumą HBase garantuoja naudodama būsenų pakitimų registrą (angl. *write-ahead logging*). Duomenų saugojimas šioje duomenų bazėje vyksta taip: pirmiausia įrašas įrašomas į būsenų pakitimų registrą, tuomet į serverio atmintį (angl. *RAM*) ir tik po to į kietąjį diską – duomenų bazės failą [Xia12]. Jei šios duomenų bazės serveris užstringa ar išsijungia įrašinėjant eilutę, paleidus duomenų bazę iš naujo ši pajėgs įrašyti visą eilutę, arba ją visą atmesti.

Aktualu, kaip užtikrinamas atomiškumas, kuomet yra daug duomenų bazės mazgų. HBase duomenų bazė išvengė šios problemos sprendimo tiesiog atsisakant transakcijų, kurios paliestų ne vieną mazgą [BH14]. Transakcijos galimos tik vienos eilutės lygmenyje.

Apžvelgus technikas, kuriomis naudojasi kiekviena iš duomenų bazių tam, kad užtikrintų atomiškumą, galima padaryti išvadas. PostgreSQL ir HBase duomenų bazės abi naudoja būsenų pakitimų registrą. Ši technika yra labai populiari užtikrinant atomiškumą. VoltDB pasinaudojo kitokiu sprendimu, dalinai lengvesniu, kas leidžia užtikrinti didesnę spartą, jei duomenų bazė veikia sklandžiai – didžioji dalis įrašymo operacijų pavyksta iš pirmo karto, duomenų bazės veikimas netrūkinėja ir t.t. Jei dažna situacija, jog reikia grįžti į tam tikrą laiko tašką duomenų atžvilgiu – VoltDB čia praranda efektyvumą, nes turi pasiimti paskutinę duomenų nuotrauką ir įgyvendinti komandas iš naujo, kuomet nei HBase, nei PostgreSQL duomenų bazėse to atlikti nereikėtų, nes turimi visi taškai kiekvienu laiko momentu su visa informacija.

HBase duomenų bazė siūlo atomiškumą tik vienos eilutės lygmenyje, o tai gali būti problema kuriant informacines sistemas – jas įgyvendinant gali reikėti atnaujinti kelias eilutes iš karto ir užtikrinti, jog jos visos kartu pasikeitė arba nepasikeitė. Pvz. norint pervesti pinigus ar kitus resursus iš vieno naudotojo, išsaugoto kaip eilutė duomenų bazėje, kitam. HBase duomenų bazė suteikdama tokias savybes kaip: skirtingų versijų duomenų saugojimą toje pačioje eilutėje, versijas atskiriant pagal laiko žymas, konkurentinius naujų versijų rašymus bei skaitymus, greitą paiešką pagal stulpelių reikšmes, leidžia sukurti metodus, kurie užtikrintų atomiškumą išskaidytoje duomenų bazėje, daugiau nei vienoje eilutėje [ZS14]. Vienas iš galimų sprendimų variantų, tai OCC kartu su dviejų fazių

transakcijos patvirtinimu. Optimistinė konkurentiškumo kontrolė neblokuoja transakcijų, o tiesiog prieš patvirtinant transakciją įsitikina, ar su vėliau įvykdytomis transakcijomis, nei prasidėjo šioji, neiškilo konfliktų. Jei iškilo – transakciją atmeta (arba pasirenka kitą būdą, kaip su tuo tvarkytis). Egzistuoja jau paruoštų HBase duomenų bazei pritaikytų bibliotekų, kurios realizuoja tokį būdą. Viena iš jų – Haeinsa [VCN13]. Deja, tačiau toks transakcijų įgyvendinimas neišvengiamai lėtina duomenų bazės veikimą – apie du kartus [ZS14].

5.3. Palyginimas vientisumo atžvilgiu

Pirmiausia reikėtų apibrėžti, kas yra vientisumas kiekvienai iš pasirinktų duomenų bazių t.y. kokios taisyklės gali būti pritaikytos duomenims, dėl kurių, jei jos būtų pažeistos, transakcija turėtų būti atmesta, nes kitu atveju, duomenų bazės būsena taptų klaidinga (nevientisa).

PostgreSQL duomenų bazė palaiko 6 tipų taisykles, kurios leidžia užtikrinti duomenų vientisumą [Pos16]:

- Netuščios reikšmės taisyklė, apriboja, kad reikšmė nebūtų *null*.
- Unikalios reikšmės taisyklė, apriboja, kad reikšmė lentelėje būtų unikali.
- Vidinių raktų taisyklė, skirta identifikuoti lentelės eilutę kitose lentelėse.
- Išorinių raktų taisyklė, skirta nurodyti kitos lentelės eilutę ir trinant objektus suteikti galimybę ištrinti vaikinius objektus.
- Reikšmės patikrinimo taisyklė, leidžia pačiam apsibrėžti taisyklę (pvz. x < 5, kai x – bandoma įterpti reikšmė) ir pagal ją tikrinti.
- Išimčių taisyklė, skirta užtikrinti, jog jei dvi eilutės yra lyginamos pagal tam tikrą stulpelį, bent vienas iš palyginimų įgis reikšmę *false*.

VoltDB duomenų bazė palaiko šias taisykles [Vol16b]:

- Netuščios reikšmės taisyklė.
- Unikalios reikšmės taisyklė (angl. *UNIQUE*), kuri užtikrinama jei lentelė yra neparticionuota arba jei stulpelis ant kurio dedama unikalios reikšmės taisyklė sutampa su lentelės particionavimo stulpeliais (t.y. tais stulpeliais, pagal kuriuos duomenys skirstomi į partitijas).
- Galimai unikalios reikšmės taisyklė (angl. ASSUMEUNIQUE), kuri užtikrina stulpelio unikalumą partitijoje, tačiau duomenų bazės naudotojas turi užtikrinti pats, jog reikšmė

unikali visoje sistemoje. Kitu atveju gali kilti problemų, jei duomenys keistų savo buvimo vietą – persikeltų iš vienos particijos į kitą ir susikirstų du įrašai su vienodomis reikšmėmis.

- Vidinių raktų taisyklė, kuriai galioja tos pačios sąlygos, kaip ir unikalios reikšmės taisyklei.
- Įrašų vienoje particijos lentelėje taisyklė, kuri apriboja įrašų skaičių.

HBase duomenų bazė neturi jokių taisyklių duomenims, kurias galėtų užtikrinti.

Akivaizdu, jog didžiausią įvairovę funkcijų, skirtų užtikrinti duomenų vientisumui turi PostgreSQL duomenų bazė, ji, beje, geriausiai jas ir užtikrina – be jokių išlygų. Šių funkcijų didžiajai daliai informacinių sistemų pilnai pakanka tam, kad būtų užtikrintas verslo procesų teisingumas – nebūtų galima įvesti įrašų su trūkstamais laukais, sukurti keletą objektų, kurie turėjo būti unikalūs ar įvesti reikšmę, kuri neatitinka apibrėžtų informacinės sistemos reikalavimų ar pan.

Su likusiomis duomenų bazėmis tenka ieškoti kompromisų. VoltDB stengiasi užtikrinti tas pačias taisykles, kaip kad siūlo tradicinės reliacinės duomenų bazės: netuščios reikšmės taisyklė, unikalių reikšmių taisyklė, vidinių raktų taisyklė. Tačiau siekiant efektyviai užtikrinti šias ir kitas taisykles, kurias užtikrina PostgreSQL, VoltDB duomenų bazei kyla sunkumų. Pvz. užtikrinant reikšmės unikalumą kyla problema siekiant patikrinti, ar tokia reikšmė dar nėra įvesta, nes lentelės gali būti padalintos į particijas, todėl patikrinimas – brangi operacija, nes tektų iteruoti visas eilutes esančias visose lentelės particijose. Reikšmės unikalumo užtikrinimas ir vidiniai raktai įgyvendinami labai panašiai – abi reikšmės turi būti unikalios, todėl kyla ta pati problema. Dėl panašios priežasties, VoltDB duomenų bazėje kol kas nėra ir išorinių raktų: prieš įterpiant elementą į duomenų bazę reikėtų tikrinti ar nuoroda į kitos lentelės elementą egzistuoja. Ta lentelė taip pat gali būti particionuota, kas reikštų, jog vėl reikia iteruoti visas lentelės particijas. Vis dėlto, šios problemos nesudėtingai išsprendžiamos, tik tenka paaukoti greitaveiką: pasinaudojant duomenų bazės procedūromis (angl. *stored procedures*), kurios VoltDB duomenų bazėje yra transakcijoje. Tai yra,. galima sukurti procedūrą pvz. konkrečios lentelės įrašo įterpimui, kuri prieš įterpdama įrašą patikrintų išorinį raktą – ar įrašas, į kurį bandoma rodyti, iš tikrųjų egzistuoja. Taip pat, įmanoma įgyvendinti ir duomenų teisingumo užtikrinimo problemos sprendimą: procedūroje patikrinant atėjusius parametrus ir tik tuomet įterpiant įrašą (esant netinkamiems parametrams – transakcija atmetama). Tačiau tokie veiksmai kainuoja brangiai. Ypatingai, jei jie reikalauja kreipimosi į lenteles išskaidytas po particijas (kurios gali būti net skirtinguose fiziniuose serveriuose sujungtuose tinklu [Vol16c]) – duomenų surinkimas užtrunka ilgesnį laiko tarpą, taip pat padidinamas visų particijų apkrovimas nuskaitymo operacijomis.

Būtina apžvelgti, kaip HBase duomenų bazėje galima spręsti vientisumo užtikrinimo problemas. Viena iš tokių galimybių – naudoti kartu su HBase ateinančiu funkcionalumu – papildomu procesoriumi (angl. *coprocessor*). Tai yra karkasas, leidžiantis vykdyti HBase duomenų apdorojimą išskaidytai ir paraleliai nesirūpinant, kurioje particijoje randasi norimi apdoroti duomenys. Šis karkasas suteikia dvi funkcijas pavadinimais „stebėtojas“ (angl. *observer*), kuris daugiau mažiau atstoja trigerius tradicinėse reliacinėse duomenų bazėse bei „pabaigos taško“ (angl. *endpoint*) papildomas procesorius, kuris panašus į procedūras (angl. *stored procedures*) tradicinėse reliacinėse duomenų bazėse [Apa12]. Turint šias alternatyvas, galima spręsti netuščios reikšmės ir apskritai – įrašomos vertės teisingumo užtikrinimą. Prieš (yra galimybė kviesti ir po) įvykdant duomenų įdėjimo metodą, galima kviesti „stebėtoją“, kuris gali patikrinti norimas įrašyti vertes ir atmesti duomenų įkėlimą arba jį pratęsti. Atrodytų, jog turint trigerius ir procedūras įmanoma užtikrinti visas duomenų vientisumo taisykles, tačiau taip nėra. Reikia prisiminti, kad atomiškumas HBase duomenų bazėje užtikrinamas tik vienos eilutės pakeitimo lygmenyje. Todėl, tarkime, bandant užtikrinti unikalios reikšmės taisyklę naudojant „stebėtoją“, kai prieš įvykdant įdėjimo operaciją bus įvykdytas patikrinimas, ar tokios reikšmės dar nėra ir po to bus vykdoma įdėjimo operacija – per tą laiką, gali atsirasti toks pat įrašas duomenų bazėje ir tai nebus pastebėta. Šiai problemai išspręsti galima panaudoti MVCC, ką atlieka kai kurie įrankiai veikiantys kaip papildiniai HBase duomenų bazei. Vienas iš jų – Tephra. Naudodamas MVCC ir papildomus HBase procesorius, jie užtikrina transakcijas net tarp lentelių [Tep16].

Galima pastebėti, jog HBase duomenų bazė yra mažiausiai gebanti užtikrinti vientisumą. Sprendimai norint išgauti kažkiek vientisumo užtikrinimo – sudėtingiausi ir brangiai kainuojantys. Kuriant šios duomenų bazės schemas, projektuojant – tą reikia daryti kitaip, nei PostgreSQL ar VoltDB duomenų bazėse. Jei pastarosiose galima skaidyti duomenis į skirtingas lentas, pvz. turint esybę „Studentas“, o joje lauką su kita esybe „Universitetas“, pastarojo duomenis galima laikyti skirtingoje lentoje, taip išvengiant duomenų dubliavimo. HBase duomenų bazėje toks būdas netinkamas, nes joje nėra išorinių raktų (o kitais būdais jų užtikrinti nepavyktų), todėl patogu laikyti esybės „Universitetas“ duomenis kartu su kiekvienu įrašu esybės „Studentas“, taip sunaudojant gerokai daugiau vietos.

5.4. Palyginimas atskirties atžvilgiu

PostgreSQL duomenų bazė užtikrina tris iš keturių pagrindinių atskirties lygmenų: patvirtintų transakcijų skaitymą, pakartotinį skaitymą, ir serijinį lygmenį. PostgreSQL duomenų bazės naudotojas gali pasirinkti, kokio atskirties lygio jam reikia, nors standartiškai PostgreSQL duomenų bazė yra nustatyta patvirtintų transakcijų skaitymo atskirties režime. Nepatvirtintų transakcijų skaitymo režimas PostgreSQL duomenų bazėje nepalaikomas dėl to, kad ši duomenų bazė naudoja MVCC [Pos16].

VoltDB duomenų bazės atveju pasirinkimo tarp atskirties lygių – nėra. Ši duomenų bazė užtikrina tik serijinį lygmenį [Vol16b]. Aktualu, kaip tai atliekama išskaidytoje duomenų bazėje. Vienoje particijoje viskas labai paprasta – veiksmai tiesiog atliekami vienas po kito. Jei užklausa daugiaparticinė, tuomet problema sprendžiama taip pat, kaip ir užtikrinamas atomiškumas (aprašyta 5.2 skyriuje).

HBase duomenų bazės naudotojas taip pat negali rinktis atskirties lygio. Kadangi ši duomenų bazė užtikrina transakcijas tik vienos eilutės lygmenyje, tai tas pats galioja ir atskirčiai – ji taip pat užtikrinama tik vienai eilutei. Duomenų įrašymo/keitimo atskirčiai užtikrinti HBase naudoja eilutės „užrakinimą“: prieš atliekant veiksmą, eilutė užrakinama, atlikus veiksmą eilutė atrakinama. Tuo metu, kai eilutė užrakinta, kitos transakcijos, bandančios keisti tą eilutę – to padaryti negalės. Duomenų skaitymo atskirčiai užtikrinti HBase naudoja MVCC. Kiekvieną kartą įrašinėjant eilutę prie kiekvienos reikšmės įterpiamas (kaip metaduomuo) įrašymo numeris. Nuskaitant eilutę pradžioje nuskaitymo operacijai priskiriamas didžiausias tos eilutės reikšmių įrašymo numeris. Toliau, skaitant eilutės duomenis, stebima, ar kiekviena reikšmė turi vienodą įrašymo numerį. Jei jis mažesnis, nei pats didžiausias tos eilutės numeris – vadinasi, nuskaitymo operacijos įrašymo numeris sumažinamas iki aptiktojo [Apa13]. Tikslas – grąžinti eilutę su reikšmėmis, kurios turės vienodus, kuo didesnius, įrašymo numerius, kad grąžinti duomenys būtų kiek įmanoma naujesni.

Nei su VoltDB, nei su PostgreSQL duomenų baze nekyla jokių esminių problemų, kas liečia atskirtį – jos abi gali užtikrinti maksimalų atskirties lygį. PostgreSQL duomenų bazė čia turi privalumą, nes leidžia jį rinktis, taip laimint spartos vykdant užklausas. HBase duomenų bazė taip pat užtikrina atskirtį, tačiau kaip visuomet, tik vienos eilutės lygmenyje. Norint praplėsti galimybes šiuo požiūriu, tektų spręsti problemą panašiai, kaip yra sprendžiamos problemos norint užtikrinti atomiškumą ar vientisumą daugiau, nei vienos eilutės lygmenyje – naudojant šalutines bibliotekas, kurios mažintų spartą.

5.5. Palyginimas patvarumo atžvilgiu

PostgreSQL duomenų bazė naudoja ne vieną metodą, kad užtikrintų patvarumą. Visų pirma, serverio operacinė sistema, kuriame veikia PostgreSQL duomenų bazė, gali būti verčiama įrašyti į kietąjį diską visus pakeitimus tuomet, kai to reikia duomenų bazei, o ne paliekant tai spręsti operacinei sistemai. Tokiu būdu, duomenų bazė gali būti tikra, kad kietajame diske yra visa reikalinga informacija ir atsitikus nenumatytiems trukdžiams, tokiems kaip elektros dingimas – duomenis bus galima atkurti. Šį funkcionalumą galima ir išjungti, prarandant patvarumą, tačiau laimint spartos, nes duomenų bazei nebereikia laukti, kol duomenys yra įrašinėjami į kietąjį diską. Taip pat, standartiškai PostgreSQL duomenų bazė laukia, kol registro įrašas apie veiksmą yra įrašomas ir tik tuomet naudotojui grąžina sėkmės pranešimą. Šį funkcionalumą taip pat galima išjungti, sumažinant užklausos laiką, tačiau atsirandant galimybei prarasti tam tikro laiko tarpo transakcijas (įprastai iki 1 sekundės). Dar vienas saugiklis, skirtas duomenų apsaugojimui, tai pradinis visų atminties puslapių nukopijavimas į įvykių įrašų registrą (angl. *full page writes*) [Pos16].

VoltDB duomenų bazė užtikrina patvarumą tais pačiais būdais, kaip ir atomiškumą (5.2 skyrius). Galima rinktis ir silpnesnius patvarumo užtikrinimo būdus [Bet14]:

- Visa informacija laikoma tik darbinėje atmintyje, todėl patvarumas neužtikrinamas visiškai.
- Nesinchroniškas komandų įrašų registro pildymas – galimybė prarasti labai nedidelę dalį įrašų.
- Sinchroniškas komandų įrašų registro pildymas – neprarandami jokie duomenys, nes prieš grąžinant naudotojui užklausos rezultatą, užklausos komanda jau būna išsaugota registre ir garantuotai išsaugota kietajame diske.

Tačiau tas faktas, jog VoltDB duomenų bazę galima išskaidyti, prisideda ir prie patvarumo užtikrinimo. VoltDB duomenų bazė palaiko duomenų replikavimą (angl. *WAN replication*) – duomenų bazių sujungimą, kurios gali būti nors ir skirtingose pasaulio pusėse, tačiau talpinti tuos pačius duomenis. Galimi dviejų tipų santykiai tarp replikuojamų duomenų bazių – pasyvus (angl. *passive*) ir duomenų centro (angl. *cross datacenter*). Pirmuoju atveju duomenys keliauja tik iš tėvinės duomenų bazės į vaikines, antruoju visos duomenų bazės yra lygiavertės ir dalinasi duomenimis tarpusavyje. Replikavimas vyksta su uždelsimu, kuomet transakcija lokaliai įvykdoma, tik po to imama perkelti duomenis į nutolusias duomenų bazes. Taip pat, VoltDB suteikia galimybę duomenis replikuoti ir vienos duomenų bazės lygmenyje. Matas, kuris rodo kiek kartų particijos bus

replikuojamos vadinasi „K-Safety“. Tai panašu į duomenų replikavimą, tačiau replikuojama ne visa duomenų bazė, o tik tam tikros, pasirinktos duomenų bazės particijos. Tai atliekama nenaudojant galutinio vientisumo modelio (angl. *eventual consistency*): užklausos atsakymas negrąžinamas tol, kol transakcija nėra įvykdoma visose particijos ir particijų kopijose [Vol16b].

HBase duomenų bazė taip pat užtikrina patvarumą – visi duomenys nuskaitomi iš duomenų bazės yra patvarūs, savaime nepranykstantys [Hba16]. HBase užtikrina patvarumą taip pat, kaip ir atomiškumą (5.2 skyrius). Yra trys palaikomi patvarumo lygmenys:

- Nerašoma į įvykių registrą – nėra ir patvarumo.
- Rašoma į įvykių registrą nesinchroniškai – galimybė prarasti nedidelę dalį įrašų.
- Sinchroniškas rašymas į įvykių registrą, tačiau neužtikrinant išsaugojimo kietajame diske prieš grąžinant naudotojui atsakymą į užklausą.

HBase duomenų bazę taip pat galima išskaidyti ir gauti papildomą patvarumo užtikrinimą. Tai atliekama replikuojant (pernešant) įvykių registro įrašus, kurie susidaro iš pakeitimų, atsirandančių vykdant naudotojo užklausas. Tarp duomenų bazės mazgų nėra būtinas pastovus ryšys, replikuojama su galimu uždelsimu [Hba15].

Galima padaryti išvadas, jog geriausiai patvarumą užtikrina VoltDB duomenų bazė. Ji suteikia galimybę užtikrinti visišką patvarumą tiek skaidant duomenų bazę, tiek to neatliekant. PostgreSQL siūlo plačią įvairovę patvarumo valdymo nustatymuose, tačiau nesuteikia galimybės skaidyti duomenų bazę ir taip dubliuoti duomenis. HBase duomenų bazė kol kas negeba garantuoti patvarumo vieno mazgo lygyje, tačiau suteikia galimybę replikuoti duomenis, kaip ir VoltDB. Galima teigti, jog visos trys duomenų bazės suteikia pakankamai galimybių prisitaikyti patvarumo užtikrinimą kuriamai informacinei sistemai. Kuo mažesnio patvarumo reikalaujama, tuo daugiau spartos galima laimėti ir atvirkščiai.

REZULTATAI IR IŠVADOS

Šiame darbe buvo įgyvendinti išsikelti tikslai:

- Nustatyti esminiai NoSQL ir NewSQL trūkumai bei pranašumai, šias duomenų bazes lyginant su tradicinėmis reliacinėmis duomenų bazėmis.
- Nustatinėjant trūkumus ir pranašumus sudarytos gairės, į kokius aspektus reikia atkreipti dėmesį lyginant tokių tipų duomenų bazes.
- Lyginimo pademonstravimui pasirinktos duomenų bazės: VoltDB, HBase ir PostgreSQL.
- Pagal kiekvieną iš ACID rinkinio savybių palygintos pasirinktos duomenų bazės.
- Įvertinta, kaip kiekviena duomenų bazė išpildo kiekvieną iš ACID savybių, o jei ji to neatlieka, tuomet pasiūlyti sprendimai, kaip būtų galima išpildyti šias savybes.

Atliekant šį darbą pastebėta, jog lyginant duomenų bazes pagal ACID užtikrinimą yra keli esminiai aspektai. Visų pirma, lyginant pagal atomiškumą bei patvarumą, esminis skirtumas tarp duomenų bazių yra tai, kokius metodus duomenų bazės naudoja vidiniams registrų įrašams kaupti. VoltDB atveju tai yra komandų įrašų registras, HBase ir PostgreSQL atveju – visų duomenų (jų pakitimų) įrašų registras. Šis aspektas svarbus tiek skirstomoms duomenų bazėms, tiek monolitinėms. Kitas svarbus aspektas paliečiantis atomiškumą yra metodai, kurie užtikrina atomiškumą paskirstytoje sistemoje. Paskirstytoje VoltDB atomiškumas užtikrinamas dalinai atsisakant konkurentiškumo ir darbus skirstant particijų iniciatoriams, o HBase atomiškumo paskirstytoje sistemoje neužtikrina apskritai. Lyginant duomenų bazes pagal vientisumą, esminis kriterijus yra taisyklių įvairovė ir jų užtikrinimo besąlygiškumas. PostgreSQL duomenų bazė užtikrina labai plačią taisyklių įvairovę be jokių išimčių. VoltDB taisyklių įvairovė kuklesnė, o ir atsiranda išimtys dėl duomenų bazės paskirstymo. HBase taisyklių galinčių užtikrinti norimą duomenų vientisumą neturi. Atskirties atžvilgiu svarbiausias aspektas yra atskirties lygis, kurį duomenų bazė gali užtikrinti ir tai, kiek „plačiai“ ji tai gali padaryti. Pvz. HBase atskirtį užtikrina tik vienai eilutei, kai tuo tarpu į PostgreSQL transakciją gali įeiti daug veiksmų. Patvarumo atžvilgiu lyginant paskirstomas duomenų bazes esminis aspektas yra tai, ar duomenų bazė leidžia duomenų replikavimą ir kaip ji tai atlieka. Šio darbo palyginime paskirstymo lyderis buvo VoltDB. Ši duomenų bazė siūlo net dviejų tipų replikavimą.

Darant bendras išvadas apie konkrečias lygintas duomenų bazes, pastebėta, jog daugiausia pastangų norint užtikrinti ACID savybes reikėtų įdėti naudojant HBase duomenų bazę. Ją geriausia rinktis tuomet, kai ACID savybės nėra kritiškai svarbios. PostgreSQL duomenų bazę būtų galima

laikyti ACID užtikrinimo etalonu, nes ji siūlo ne tik ACID užtikrinimą, bet ir galimybę pasirinkti tinkamą ACID užtikrinimo lygį per parametrus. VoltDB duomenų bazė – geras sprendimas, jei yra būtinybė skaidyti duomenis, bet nenorima visiškai atsisakyti ACID transakcijų privalumų.

ŠALTINIAI

[Apa12] The Apache Software Foundation. *Apache HBase Coprocessor Introduction*, https://blogs.apache.org/hbase/entry/coprocessor_introduction

[Apa13] The Apache Software Foundation. *Apache HBase Locking and Multiversion Concurrency Control*, https://blogs.apache.org/hbase/entry/apache_hbase_internals_locking_and

[Asl11] M. Aslett. *What we talk about when we talk about NewSQL*, https://blogs.the451group.com/information_management/2011/04/06/what-we-talk-about-when-we-talk-about-newsql/

[BH14] A. Baranau, G. Helmling. *Transactions over HBase.* http://www.slideshare.net/alexbaranau/transactions-over-hbase

[Bar05] R. Baronas. *Duomenų bazių valdymo sistemos.* TEV, 2005, ISBN 9955-680-09-1, 184 psl.

[Bet14] R. Betts, *Don't Accept Data Loss when Upgrading to In-memory Database Performance*, https://voltdb.com/blog/don%E2%80%99t-accept-data-loss-when-upgrading-memory-database-performance

[Bre12] E. Brewer. CAP twelve years later: How the "rules" have changed, *Computer*, 45, 2012, p. 23-29.

[CB74] D. Chamberlin, R. Boyce. *SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE*, http://www.almaden.ibm.com/cs/people/chamberlin/sequel-1974.pdf

[DBE16] DB-ENGINES. *DB-Engines Ranking*. Prieiga per internetą: http://db-engines.com/en/ranking

[Das13] J. Dash. *RDBMS vs. NoSQL: How do you pick?*, http://www.zdnet.com/article/rdbms-vs-nosql-how-do-you-pick/

[Glu15] I. Glushkov. *NewSQL overview*, http://www.slideshare.net/IvanGlushkov/newsql-overview

[HR83] T. Haerder, A. Reuter. Principles of transaction-oriented database recovery, *ACM Computing Surveys (CSUR)*,15, 1983, p.287 – 317.

[Hba15] HBase. *Replication*, https://hbase.apache.org/0.94/replication.html

[Hba16] HBase. *Acid semantics*, https://hbase.apache.org/acid-semantics.html

[ILS15] Internet Live Stats. *Internet Users in the world*. Prieiga per internetą: http://www.internetlivestats.com/internet-users/

[Kle14] M. Kleppmann, *Hermitage: Testing the "I" in ACID*, https://martin.kleppmann.com/2014/11/25/hermitage-testing-the-i-in-acid.html

[LJ14] S. Lokegaonkar, A. Joshi. Concurrency control schemes in NewSQL systems. *International Journal of Computer Engineering & Technology (IJCET)*, 5, 2014, p. 97-104. Prieiga per internetą:

http://www.iaeme.com/MasterAdmin/UploadFolder/CONCURRENCY%20CONTROL%20SCHEMES%20IN%20NEWSQL%20SYSTEMS/CONCURRENCY%20CONTROL%20SCHEMES%20IN%20NEWSQL%20SYSTEMS.pdf

[LaF13] W. LaForest. *NoSQL the Modern MUMPS*, http://osehra.org/sites/default/files/NoSQLtheModernMUMPS.pdf

[MH13] A.B.M. Moniruzzaman, S.A. Hossain. NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. *International Journal of Database Theory and Application*, 4, 2013.

[Mak15] V. Makovsky. *Query Throughput on Analytics on OLTP (Postgres)*, http://www.briskat.com/blog/OLTP-Throughput

[Nel15] R. Nelson. *NewSQL takes advantage of main-memory databases like H-Store*, http://www.evaluationengineering.com/2015/02/25/newsql-takes-advantage-of-main-memory-databases-like-h-store/

[Nol14] M. Noland. *The Truth about SQL on Hadoop*, https://phdata.io/the-truth-about-sql-on-hadoop-part-1/

[OES+08] M. Olugbode, I. Elbeltagi, M. Simmons, T. Biss. The Effect of Information Systems on Firm Performance and Profitability Using a Case-Study Approach, *EJISE,* 11, 2008, p. 11-16. Prieiga per internetą: http://www.ejise.com/issue/download.html?idArticle=574

[Ora16] Oracle. *Introduction to Oracle Database*, https://docs.oracle.com/database/121/CNCPT/intro.htm

[PCP15] PCPARTPICKER. *Price trends*, https://pcpartpicker.com/trends/memory/

[Pag10] D. Page. *Comparing VoltDB to Postgres*, http://pgsnake.blogspot.com/2010/05/comparing-voltdb-to-postgres.html

[Pos16] PostgreSQL. *Manual*, http://www.postgresql.org/docs/manuals/

[Sah15] B. K. Sahu. *A Real Comparison Of NoSQL Databases HBase, Cassandra & MongoDB*, https://www.linkedin.com/pulse/real-comparison-nosql-databases-hbase-cassandra-mongodb-sahu

[Sir14] E. G. Sirer. *NoSQL Meets Bitcoin and Brings Down Two Exchanges: The Story of Flexcoin and Poloniex*, http://hackingdistributed.com/2014/04/06/another-one-bites-the-dust-flexcoin/

[Sto05] M. Stonebraker. *One Size Fits All": An Idea Whose Time Has Come and Gone*, https://cs.brown.edu/~ugur/fits_all.pdf

[Sto10] M. Stonebraker. *Errors in Database Systems, Eventual Consistency, and the CAP Theorem*, http://cacm.acm.org/blogs/blog-cacm/83396-errors-in-database-systems-eventual-consistency-and-the-cap-theorem/fulltext

[Str98] Carlo Strozzi. *NoSQL A Relational Database Management System*. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page

[Tep16] Tephra. *Transactions for Apache HBase*, https://github.com/caskdata/tephra

[VCN13] VCNC. *Haeinsa Overview*, https://speakerdeck.com/vcnc/haeinsa-overview-hbase-transaction-library

[Vol16a] VoltDB. TECHNICAL OVERVIEW, http://downloads.voltdb.com/datasheets_collateral/technical_overview.pdf

[Vol16b] VoltDB. *Documentation*, https://docs.voltdb.com/

[Vol16c] VoltDB Blog. *Partition or Replicate? Getting the most out of VoltDB's distributed data management*. https://voltdb.com/blog/partition-or-replicate-getting-most-out-voltdbs-distributed-data-management

[Way12] P. Wayner. *7 hard truths about the NoSQL revolution*, http://www.infoworld.com/article/2617405/nosql/7-hard-truths-about-the-nosql-revolution.html

[Xia12] J. Xiang. *Apache HBase Write Path*, http://blog.cloudera.com/blog/2012/06/hbase-write-path/

[ZS14] C. Zhang, H.Sterck. *Supporting Multi-row Distributed Transactions with Global Snapshot Isolation Using Bare-bones HBase*, https://www.researchgate.net/publication/221547848_Supporting_multi-row_distributed_transactions_with_global_snapshot_isolation_using_bare-bones_HBase