VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

PROGRAMŲ SISTEMŲ KATEDRA

Releacinių ir nereleacinių DBVS palyginimo būdai

Relational and non relational DBMS comparison methods

Bakalauro darbas

Atliko: 4 kurso studentas

Paulius Petkus

Darbo vadovas:

Karolis Uosis

Recenzentas:

Viktoras Golubevas

Vilnius

2016

Santrauka

Praeityje releacinės sistemos buvo naudojamos daugumoje aplikacijų dėl duomenų traukimo paprastumo, užklausų galimybių ir manipuliavimo tranzakcijomis. Vis dėlto didėjant informacijos kiekiams ir plečiantis duomenų bazėms buvo ieškota naujų efektyvesnių būdų saugoti ir manipuliuoti duomenims. Pastaraisiais metais išpopuliarėjusios NoSQL sistemos iškėlė daug klausimų ir ginčų, dėl to, kurios sistemos yra geresnės įvairiais atvejais. Šiame darbe yra lyginamos SQL ir NoSQL sistemų įvairios savybės: duomenų modeliai, nuoseklumo ir saugojimo mechanizmus, duomenų ilgaamžiškumo garantijos. Taip pat darbe yra lyginamas Microsoft SQL server, MySQL, MongoDB ir Cassandra DBVS efektyvumas atliekant įvairias užklausų operacijas, su skirtingais kiekiais duomenų, mažos aplikacijos mąstu. Tyrimo rezultatai parodė, jog kintant duomenų kiekiui ir užklausų sudėtingumui ir atliekant skirtingas operacijas nei viena iš sistemų nėra stabiliai geriausia visuose tyrimo atvejuose.

Summary

In the past, relational databases were used in a large scope of applications as they have rich set of features, query capabilities and transaction management. However due to increasing amounts of information and expanded databases, there was a need to look for a more effective ways to save and manipulate the data. In recent years, NoSQL systems have become increasingly popular and raised many questions and debates on which system is better in different cases. In this work we compare SQL and NoSQL systems at various properties: data models, concurrency and data manipulating mechanizms ,durability waranty. Microsoft SQL server, MySQL, MongoDB and Cassandra DBMS are compared in various quering operations with different amount of data, in small application. The results showed that changing the amount of the data and query complexity and the different operations, none of the systems had best results in all the test cases.

Turinys

Įvadas....6
1. NoSQL duomenų bazių valdymo sistemų apžvalga....8
2. Duomenų modelių terminologija....9
3. Duomenų saugyklų kategorijos....9
3.1. Raktinių reikšmių saugyklos....10
3.1.1. Project Voldemort....10
3.1.2. Riak....10
3.1.3. Redis....12
3.1.4. Scalaris....12
3.1.5. Raktinių reikšmių duomenų bazių saugyklų apibrendrinimas....13
3.1.6. Raktinių reikšmių saugyklų naudojimo pavyzdys....13
3.2. Dokumentų saugyklos....14
3.2.1. SimpleDB....14
3.2.2. CouchDB....14
3.2.3. MongoDB....15
3.2.4. Terrastore....16
3.2.5. Dokumentų tipo saugyklų apibendrinimas....17
3.2.6. Dokumentų saugyklų naudojimo pavyzdys....17
3.3. Išpėsto įrašo saugyklos....18
3.3.1. HBase....18
3.3.2. HyperTable....19
3.3.3. Cassandra....19
3.3.4. Išplėsto įrašo tipo duomenų bazių apibendrinimas....20
3.3.5. Išplėsto įrašo saugyklų naudojimo pavyzdys....20
3.4. Skaliuojamos releacinės sistemos....20
3.4.1. MYSQL Cluster....21
3.4.2. VoltDB....21
3.4.3. Clustrix....22
3.4.4. ScaleDB....22
3.4.5. Skaliuojamų releacinių sistemų apibendrinimas....23
3.4.6. Skaliuojamų RDBVS pavyzdys....23
4. SQL ir NoSQL DBVS palyginimas....24
4.1. DBVS palyginimo rezultatai....27

Rezultatai ir Išvados 35
Šaltiniai 37
Sąvokų apibrėžimai: 38

Įvadas

Duomenų bazių valdymo sistemos (DBVS) gali būti apibūdintos, kaip kolekcija mechanizmų leidžianti saugoti, keisti ir ištraukti duomenis. Pastaraisiais metais DBVS tapo duomenų bazių sinonimu. Duomenų bazių įvairovė svyruoja nuo paprastų, kurios saugo failą su duomenimis iki sudėtingų kurios gali saugoti milijonus įrašų. Evoliucionuojant Informacijos ir komunikacijų technologijoms, duomenų tipai, funkcionalumas ir sąveika su duomenų bazėmis tobulėjo. Duomenų bazės tapo resursu, naudojamu milijonų žmonių, kiekvieną dieną, nesuskaičiuojamame kiekyje aplikacijų. Toks masiškas vartojimas sukūrė poreikį turėti duomenis struktūrizuotus ir paruoštus, taip kad jie būtų ištraukti taip greit ir lengvai, kaip tik įmanoma. Didėjant duomenų kiekiui automatiškai didėja ir duomenų bazės. Su eksponentiniu duomenų bazių dydžio augimu, duomenų pasiekimas privalo būti maksimaliai efektyvus Tai veda prie gerai žinomos efektyvumo traukiant duomenis problemos.

Edgar Frank Codd 1970 metais pristatė releacinį modelį. Kaip tyrimo rezultatą, Codd pateikė išeitį įveikiančia duomenų saugyklų naudojimo sunkumus, remiantis principais, paremtais duomenų sąryšiais. Buvo pristatyta 12 taisyklių, skirtų valdyti duomenis releaciniame modelyje, žinomų kaip „E. F. Codd'o 12 taisyklių". Buvo sukurta eksperimentinė duomenų bazės Sistema, demonstruojanti releacinio modelio naudojimo privalumus. Su šia Sistema buvo sukurta nauja kalba „Structured Query Language", žinoma kaip SQL. Nuo tada SQL tapo duomenų manipuliavimo ir iteravimo standartu. Releacinės duomenų bazės saugo duomenis lentelėse, su skirtinga informacija. Visi duomenys yra susiję, todėl galima pasiekti informaciją iš skirtingų lentelių, tuo pačiu metu. Releacinis modelis yra paremtas „sąryšiais". Originaliam releaciniame modelyje, sąryšiu buvo laikoma lentelė su stulpeliais ir eilutėmis. Kiekviena lentelė buvo suformuota iš „tuple" rinkinio, su vienodais atributais. Atributai turi informaciją apie objektus. Sudėtigos duomenų bazės turi daug lentelių su milijonais įrašų. Lenteltės yra sujungtos raktais. Raktai būna dviejų tipų: pirminiai raktai ir išoriniai raktai. Pirminiai raktai yra unikalūs ir identifikuoja konkrečią lentelės eilutę, o išoriniai raktai naudojami susieti (cross-reference) lenteles. Išorinis raktas vienoje eilutėje atspindi pirminį raktą kitoje eilutėje.

Eksponentiškai didėjant duomenų apimčiai atsiranda įvairių problemų. Viena iš jų yra duomenų bazių efektyvumas pasiekiant duomenis su paprastu struktūriniu releaciniu modeliu. SQL leidžia lengvai gauti duomenis, tačiau kai informacijos kiekis pasidaro didelis, užklausų vykdymo laikas pasidaro lėtas. Tam, kad išspręsti šią efektyvumo problem buvo ieškoma kitokių duomenų bazių modelių.

Vienas iš jų yra žinomas kaip „NotOnlySQL”. Carlo Strozzi 1980 metais pristatė šią atviro kodo duomenų bazę, kuri nenaudojo SQL. Pats Carlo Strozzi NoSQL sistemą siūlė vadinti „noseequel” arba „NoRel”, tačiau tai iš esmės skiriasi su technologijom kurios egzistuoja šiomis dienomis. Dabartinių NoSQL sistemų pradininku galima laikyti Google kompanijos duomenų bazę „BigTable”, kurioje yra saugomi įvairūs google projektai, tokie kaip: „Google Erath”. Vėliau kompanija Amazon sukūrė savo sistemą pavadinimu „Dynamo”. Abi šios sistemos stipriai prisidėjo prie NoSQL kūrimo ir evoliucijos. Vis dėlto, NoSQL nebuvo populiarus ir žinomas iki 2009 metais vykusios konferencijos San Francisko mieste. Nuo tada NoSQL ėmė sparčiai populiarėti

Šiomis dienomis, kuriant naujas aplikacijas dažnai abi alternatyvos: SQL ir NoSQL sistemos yra vertinamos lygiareikšmiškai. Iškyla klausimai: kurios sistemos yra geresnės, kuo jos geresnės, ką pasirinkti. Šiais klausimais kyla nemažai ginčų ir diskusijų, bet mokslinių straipsnių nėra pakankamai daug

Darbo tikslas: Palyginti SQL ir NoSQL duomenų bazių valdymo sistemas, rasti sistemų stipriąsias ir silpnąsias vietas, bei sudaryti sąrašus atvejų, kuriais NoSQL ir SQL sistemos užduotis vykdo greičiau.

Darbo uždaviniai:

1) išanalizuoti dabartines populiarias SQL ir NoSQL duomenų bazių valdymo sistemas ir palyginti šių sistemų duomenų modelius, nuoseklumo ir saugojimo mechanizmus,
2) palyginti populiarių SQL sistemų ir NoSQL sistemų užklausų vykdymo greičius kintant sąlygoms
3) Sudaryti sąrašus atvejų, kuriais NoSQL ir SQL sistemos užduotis vykdo greičiau

1. NoSQL duomenų bazių valdymo sistemų apžvalga

Pastaraisiais metais išėjo nemažai naujų sistemų pasižyminčių geru horizontalaus krūvio augimo apdorojimu atliekant paprastas skaitymo ir rašymo operacijas. Dauguma šių sistemų yra laikomos „NoSQL" duomenų bazėmis. NoSQL apibrėžimas: „ne tik SQL" arba „ne releacinė" nėra visiškai teisingas. NoSQL sistemos turi šešis pagrindinius - raktinius bruožus:

- Galimybė apdoroti horizontalaus krūvio, didelį kiekį „paprastų operacijų", per daugybę serverių,
- Galimybė atkartoti ir paskirstyti duomenis per daugybę serverių,
- Paprasta kvietimo lygmens sąsaja (priešingai negu SQL)
- Efektyvus indeksų ir RAM'ų išnaudojimas saugojant duomenis,
- Galimybė dinamiškai pridėti atributus į esamus įrašus

Skirtingos duomenų bazės išsidėsto pagal funkcionalumą nuo paprasčiausių duomenų apdorojimo schemų, naudojant „memcached" atviro kodo podelį[1], iki pat sudėtingų kintančio dydžio dalinių lentelių, palaikomų Google „BigTable". BigTable, Memcached ir Amazon'o Dynamo suteikė pagrindus kuriais rėmėsi daug kitų duomenų bazių:

- Memcached įrodė, kad įrašančios atminties (in-index) indeksai gali būti stipriai kintančio dydžio, paskirstant ir atkartojant objektus skirtingose gijose.
- Dynamo pastatė pagrindus idėjai, kad galimas nuoseklumas(eventual consistency) yra geras būdas pasiekti gerą pasiekiamumą ir skaliuojamumą[2]: gauti duomenys nėra garantuotai pakankamai nauji (up to date), bet galiausiai atnaujinimai garantuotai ateis ir bus įgyvendinti visose gijose
- BigTable įrodė, kad nuolatinė įrašų talpykla gali būti skaliuojama tūkstančiams gijų, tai yra dalykas kurio dauguma sistemų siekia.

„Pagrindinis NoSQL funkcionalumas yra „shared nothing" horizantulus skaliuojamumas - duomenų dalijimas ir atkartojimas per daugybę serverių. Tai leidžia jiems palaikyti didelį kiekį paprastų skaitymo / rašymo operacijų per sekundę." [DPM14] Ši paprasta operacija paprastai yra vadinama OLTP (online transaction processing).

2. Duomenų modelių terminologija

Skirtingai nuo releacinių (SQL) duomenų bazių valdymo sistemų, NoSQL terminologija dažnai yra nenuosekli. Visos aptariamos sistemos suteikia būdą saugoti skaliuojamas reikšmes, tokias kaip skaičiai, string'ai ir BLOB. Kaikurios jų suteikia būdą išsaugoti sudėtingesnius tipus arba nuorodų reikšmes. Visos sistemos saugo atributų reikšmių poras, bet naudoja skirtingas struktūras:

- „Tuple" yra realeacinės lentelės eilutė, kur atributų vardai yra preagreguoti schemoje ir reikšmės privalo būti skaliuojamos. Reikšmės yra nukreipiamos pagal atributo pavadinimą ir palyginamos su masyvu arba sąrašu, kur jos yra sudėliojamos pagal eilės tvarką.
- „Document" leidžia reikšmėms būti įdėtoms dokumento arba sąrašo tipo kaip skaliuojamos reikšmės ir atributų vardai yra apibrėžiami dinamiškai kiekvienam dokumentui vykdymo metu. Dokumentai skiriasi nuo „tuple" tuo, kad atributai nėra apibrėžti schemoje ir leidžiamas platesnis rėžis reikšmių.
- „Extensible record" yra hibridas tarp „tuple" ir „document", kur atributų šeimos yra nurodytos schemoje, bet nauji atributai gali būti pridėti per įrašų (per-record) pagrindus. Atributų reikšmės gal ibūti sąrašai.
- „Object" yra objekto analogas programavimo kalbose, bet be procedūrinių metodų. Reikšmės gali būti nuorodos arba idėtos reikšmės.

3. Duomenų saugyklų kategorijos

Šiame dokumente duomenų saugyklos yra sugrupuotos pagal jų duomenų modelį:

1) Raktinių reikšmių saugyklos: šios sistemos saugo reikšmes ir jų indeksus, tam kad atrasti reikšmes pagal juos.
2) Dokumentų saugyklos: šios sistemos saugo dokumentų duomenis. Dokumentai yra indeksuojami ir apdorojami paprastų užklausų mechanizmu.
3) išplėsto įrašo saugyklos[3]: šios sistemos saugo išplėsto tipo įrašus kurie gali būti paskirstyti ir vertikaliai ir horizontaliai per gijas. Kaikuriuose šaltiniuose naujojama sąvoka: plataus stulpelio saugykla
4) Releacinės duomenų bazės: šios sistemos saugo „tuple"

3.1. Raktinių reikšmių saugyklos

Paprasčiausios duomenų saugyklos naudoja panašų į memcached duomenų modelį: duomenys paskirstyti „in-memmory" podelyje, su vieną raktine reikšme visiems duomenims. Skirtingai nuo memcached, šios sistemos palaiko pastovų mechanizmą ir papildomą funkcionalumą: repkilacija, versijavimas, užrakinimas (locking), tranzakcijos, rikiavimas, ir panašios funkcijos. Kliento sąsajoje galimi įterpimai, ištrynimai, indeksų ieškojimai. Kaip kad ir memcached, šios sistemos neturi antrinių indeksų ar raktų.

3.1.1. Project Voldemort

„Voldemort" yra pažangi raktinių reikšmių saugykla parašyta Java programavimo kalba. Voldemort atnaujinimams teikia daugiaversijinę lygiagrečią kontrolę (MVCC). Ji atnaujina kopijas asinchroniškai, todėl negarantuoja nuoseklių duomenų. Voldemort palaiko optimistinį užranktą nuosekliems kelių įrašų atnaujinimams: jeigu atnaujinant įvyksta konfliktas su bet kuriuo procesu, nei vienas atnaujinimas neįvyksta. Vektorinis laikrodis kaip ir Dynamo leidžia rikiuoti versijas. Galima pasirinkti kurias versijas norime atnaujinti įdėjimo ir trynimo operacijose. Voldemort palaiko automatinį duomenų šardinimą[4] (sharding). „Consistent hashing" yra naudojamas paskirstyti duomenis aplink mazgų žiedus: duomenys paskirstyti į mazgą K yra atkartoti mazge K+1 … K+n, kur n yra norimų kopijų skaičius. „Naudojant gerą šardinimo techniką „virtualių" gijųų bus daugiau negu fiziškai egzistuojančių serverių." [SV13] Kai tik duomenų pertvarkymas yra sudarytas, jų operacijos yra aišios – naudojomas „SimpleAPI". Mazgai gali būti pridėti arba ištrintos iš duomenų bazės klasterio ir sistema prisitaikys automatiškai, yra mechanizmas kuris automatiškai aptinka ir sutvarko netvarkingas gijas. Voldemort kombinuoja duomenų saugojimą atmintyje ir diske, todėl nėra reikalingas atskiras spartinimo mechanizmas. Voldemort palaiko sąrašus ir įrašus be paprastų skaliarinių reikšmių.

3.1.2. Riak

Riak yra sukurtas Erlang programavimo kalba. Basho padarė šią sistemą atviro kodo 2009 metų viduryje. Basho apbūdina Riak kaip raktinių reikšmių saugyklą arba dokumentų saugyklą. Šiame dokumente Riak taip pat priskirtas prie raktinių reikšmių saugyklos tipo, nes kaip ir Voldemort turi daugiau funkcionalumo šioje srityje negu kitos saugyklos:

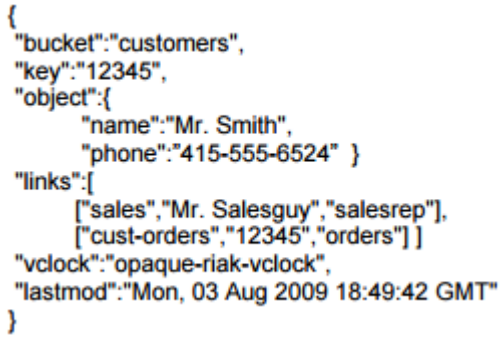
- Riak objektai gali būti įkelti ir saugojami JSON formatu ir gali turėti keletą laukų (kaip ir dokumentai), objektai gali būti sugrupuoti į bucketus, kaip kolekcijos dokumentų saugyklose su leistinais / privalomais (allowed/required) laukais apibrėžtais per-bucket baziškai.
- „Riak nepalaiko indeksų jokiems laukams išskyrus pirminius raktus.” [BCM+14] Vieninteis dalykas kurį galima daryti su nepirminiaislaukais, tai paduoti ir išsaugoti juos kaip dalį JSON objekto. Riak trūksta užklausų mechanizmų dokumentų saugykloms.

Riak palaiko objektų kopijas ir shardinimą skirstant per pirminį raktą. Kopijų reikšmės gali būti laikinai nenuoseklios. Nuoseklumas yra derinamas apibrėžiant kiek kopijų (skirtingų mazgų) privalo atsakyti sėkmingu įrašymu. Skirtingos aplikacijos dalys, gali pasirikti skirtingus kompromisus „per-read” ir „per-write” funkcijoms. Kaip ir Voldemort Riak naudoja MVCC darinį, kur vektorinis laikrodis prisiskiria, kai reikšmės yra atnaujintos. Vektorinis laikrodis gali nustatyti kada objektai yra tiesioginiai palikuoniai, o kada bendriniai tėvai. Taigi Riak dažnai atranda nesusinchornizuotus ir pats pataiso duomenis. Nėra išskirto mazgo sekti sistemos statusui: mazgai naudoja „gossip” protokolą, kad sekti kurie mazgais yra aktyvūs ir kas kokius duomenis turi, betkuris mazgas vykdo vartotojo užklausas. Riak taip pat turi „map-reduce” mechanizmą, tam, kad išskaidyti darbą per mazgus klasteryje.

„Vartotojo sąsaja Riak sistemoje yra pagrįsta „RESTfull” HTTP užklausomis. REST[5] naudoja vienodas, bebūsines, podėliuojamas vartotojo užklausas į serverį.” [BCM+14]

Saugojimo dalis Riak yra „užmaunama”: raktinių reikšmių poros gali būti atmintyje, „ETC” lentelėse, „DETS” lentelėse arba „Osmos” lentelėse. ETS, DETS ir Osmos lentelės yra visos implementuotos su Erlang programavimo kalba su skirtingu pritaikymu ir nustatymais. Viena unikali Riak savybė, tai kad gali saugoti „nuorodas” tarp objektų (dokumentų). Pavyzdžiui: nuoroda tarp autorių objektų į knygas kurias jie parašė. Nuorodos sumažina indeksų poreikį, bet nesuteikia galimybės rašyti užklausų su intervalais (rėžiais).

Čia yra Riak objekto pavyzdys JSON formatu:



1 pav. Riak objekto pavyzdys

Pirminis raktas yra išskirtas, kol tuo tarpu kiti laukai yra „objekto" viduje. Taip pat pastebėtinas dalykas tai, kad vektorinis laikrodis ir koreagavimo data nurodyta kaip dalis objekto, kur yra nuoroda į kitus objektus.

3.1.3. Redis

Redis, raktinių reikšmių duomenų bazė, prasidėjo kaip vieno žmogaus projektas, bet dabar yra keletą bendraautorių ir yra BSD licenzijuotas atviro kodo projektas. Parašytas su C programavimo kalba. Redis serveris yra pasiekiamas „vielos" (wire) protokolu, yra implementuotas daugybėje bibliotekų (kurios turi būti atnaujintos, kai pasikeičia protokolas). Klientinė dalis daro paskirstytą hashinimą per serverius. Serveriai saugo duomenis operatyvioje atmintyje, bet duomenys gali būti nukopijuoti į diską atsarginiam duomenų grąžinimui(backup[6]) arba sistemos išjungimo atveju. Sistemos išjungimas gali būti reikalingas norint pridėti naujų gijų. Kaip ir kitos raktinių reikšmių saugyklos, Redis turi įterpimo, ištrynimo ir ieškojimo operacijas. Kaip Voldemort, Redis leidžia sąrašams ir rinkiniams būti susietai su raktu, ne tik su blob ar string. Taip pat turi sąrašų ir rinkinių operacijas. Redis daro atominius atnaujinimus užrakindamas ir daro asinchroniską kopijavimą. Gali palaikyti apie 100K gavimų / priskyrimų per sekundę su 8 branduolių serveriu.

3.1.4. Scalaris

Scalaris yra funkciškai panašus į Redis. Jis buvo parašytas su Erlang, Zuse institute, Berlyne ir yra atviro kodo. „Paskirstydamas duomenis per gijas, Salaris leidžia raktinius rėžius priskirti prie mazgų, vietoje to, kad tiesiai hashint'ų į gijas. Tai reiškia, kad užklausa su rėžinėm reikšmėm neturės kreiptis į kiekvieną giją, taip pat galės išvystyti geresnį užkrovimo balancą priklausomai nuo raktų pasiskirstymo." [KCM+14] Kaip ir kitos raktinių reikšmių saugyklos, leidžia įterpti, ištrinti ir ieškoti. Kopijuoja sinchroniškai (kopijos turi būti atnaujintos, prieš užbaigiant operaciją), todėl yra garantuojama, kad duomenys yra nuoseklūs. Scalaris taip pat palaiko tranzakcijas su ACID

nustatymai keliuose objektuose. Duomenys yra saugojami atmintyje, bet kopijavimas ir atstatymas iš mazgo nesėkmių (failures) suteikia atnaujinimams ilgaamžiškumą. Vis dėlto „kelių mazgų klaidos gali sukelti didelius duomenų praradimus" [KCM+14] ir virtualios atminties limitas yra visos duomenų bazės dydis. Scalaris naudoja gijų žiedą, neįprastą paskirstymo ir kopijavimo strategiją, kuri reikalauga log(N) šuolių, kad perskaitytų / įrašytų raktinių reikšmių porą.

3.1.5. Raktinių reikšmių duomenų bazių saugyklų apibrendrinimas

Visos raktinės reikšmės saugyklos palaiko įvedimo, ištrynimo ir peržiūros operacijas. Visos šios sistemos teikia skaliuojamumą per raktinius paskirstymus gijose. Voldemort, Riak ir sustiprintos memcached sistemos gali saugoti duomenis operatyvioje atmintyje arba diske su saugojimo papildiniais. Kitos saugo duomenys operatyvioje atmintyje ir naudoja diską atsarginiam duomenų grąžinimui (backup), arba pasikliauja kopijavimu ir atstatymu (tuomet atsarginis duomenų grąžinimas yra nereikalingas).

Salaris ir sustiprintos memcached sitemos naudoja snchorininį atkartojimą, kitos sistemos asinchronišką. Scalaris naudoja transakcijas kol tuo tarpu kitos nenaudoja. Voldemort ir Riak naudoja daugiaversijinę lygiagrečia kontrolę (MVCC), kitos sistemos naudoja užrakinimus. Membrain ir Membaseyra pastatytos ant populiarios memcached sistemos pridedant atkartojimo ir kitas funkcijas. Suderinamumassu memcached duoda šiems produktams privalumą.

3.1.6. Raktinių reikšmių saugyklų naudojimo pavyzdys

Raktinių reikšmių saugyklos yra geras sprendimas jeigu yra aplikacija, kuriai reikia tik vieno tipo objektų, ir atitinkamai reikia ieškoti objektų, pagal atributą. Paprastas raktinių reikšmių saugyklų funkcionalumas, padaro jas paprasčiausiai naudojamas, įpatingai jeigu vartotojas turi patirties dirbant su memcached.

Pavyzdys: vartotojas turi internetinę aplikaciją, kuri daro daug RDBVS užklausų ir kuria pritaikytą puslapį kai vartotojas prisijungia. Tarkime toks prisijungimas reikalauja kelių sekundžių ir vartotojo duomenys retai keičiasi, o jeigu pasikeičia, tai per tą pačią sąsają. Tuomet mes kogero norėsime laikyti vartotojui pritaikytą puslapį kaip vieną objektą, raktinių reikšmių saugykloje, išsaugotą taip, kad būtų efektyvu siųsti atsakymą naršyklės užklausoms. Šie objektai bus suindeksuoti pagal vartotojo ID. Jeigu šie objektai bus saugomi tvarkingai, galima išvengti daugybės RDBVS užklausų, perkonstruojant objektą, kuomet vartotojo informacija atsinaujina.

Galima turėti raktinių reikšmių saugyklas su paieškomis pagrįstomis keliais atributais, susikuriant raktinių reikšmių indeksus, kurie tai palaikys, tačiau tokiu atveju patartiniau pereiti į dokumentų saugyklos tipą.

3.2. Dokumentų saugyklos

Dokumentų saugyklos saugo sudėtingesnius duomenis negu raktinių reikšmių saugyklos. Skirtingai nuo raktinių reikšmių saugyklų šios sistemos palaiko antrinius indeksus ir keletą dokumentų(objektų) tipų duombazėje. Kaip kitos NoSQL sistemos dokumentų saugyklos nepalaiko ACID tranzakcinių nustatymų.

3.2.1. SimpleDB

SimpleDB yra dalis Amazon nuosavybės, kurią debesų kompiuterija siūlo savo „Simple Storage Service", jo pagrindu yra sukurtas SimpleDB. SimpleDB turi Select, Delete, GetAttributes ir PutAttributes operacijas dokumentuose. SimpleDB yra paprastesnis negu kitos dokumentų saugyklos, nes neleidžia nested dokumentų. Kaip ir dauguma kitų sistemų, SimpleDB turi asinchorniška kopijavimą. Skirtingai nuo raktinių reikšmių duomenų saugyklų, SimpleDB palaiko keletą grupavimų vienoje duomenų bazėje: dokumentai nėra sukrauti į domenus, kurie palaiko keletą indeksų. Galima išvardinti domenus ir jų metaduomenis. „Select" operacijos yra vieno domeno ir nurodomos konjunkcijos arba suvaržomos atributais. Skirtingi domenai saugo skirtinguose Amazon mazguose. Domeno indeksai yra automatiškai atnaujinai, kai kuris nors dokumento atributas yra modifikuotas. SimpleDb automatiškai ne dalija duomenų per serverius. „Dalis horizontalaus skaliuojamumo gali būti pasiekta skaitant kopijas, jeigu nėra svarbu, turėti naujausius duomenis." [WZS+15] Įrašymai nesiskaliuoja, nes jie privalo eiti asinchroniškai nuo domeno kopijų. Jeigu vartotojas nori geresnio skaliuojamumo, jis pats turi tai padaryti rankiniu būdu shardinant.

Amazon apribojo SimpleDB įstatydama suvaržymus: 10GB maksimalus domeno dydis, 100 domenų maksimalus limitas ir 5 sekundžių limitas užklausoms. Amazon nelicenzijuoja SimpleDB variklio ar binarinio kodo vartotojų serveriuose.

3.2.2. CouchDB

CouchDB buvo parašytas Erlang programavimo kalba. CouchDB dokumentų „kolekcija" yra panaši į SimpleDB domeną, bet CouchDB duomenų modelis yra turtingesnis. Kolekcijos turi vienintelę schemą ir antriniai indeksai turi būti aiškiai sukurti laukuose, kolekcijose. Dokumentas turi laukų

reikšmes ir gali būti skaliuojamas (tekstas, skaičius arba loginė reiškmė) arba jungtinis. Užklausos yra vykdomos su CouchDb kvietimais, kurie yra apibrėžti Javascript, tam kad nurodyti laukų apribojimus. Indeksai yra nuoseklūs, vykdant užklausą ieškomi B-medžio(B-tree) algoritmu, todėl užklausos rezultatai, gali būti surikiuoti rėžiais. „Užklausos gali būti paskirstytos paraleliai per kelis mazgus naudojant „map-reduce" mechanizmą." [WZS+15]. „CouchDb vaizdavimo mechanizmas duoda daugiau naštos programuotojui negu konstruojamai užklausos kalbai." [SSY+11]

Kaip ir SimpleDB, CouchDB pasiekia skaliuojamumą per asinchorininį kopijavimą, ne per shardinimą. Skaitymai gali eiti į betkurį serverį, jeigu mums nėra svarbu turėti naujausius duomens, ir atnaujinimai turi būti paskleisti į visus serverius. Vis dėlto naujas projektas pavadinimu CouchDB Lounge buvo sukurtas, kad leistų shardinimą CouchDB. Kaip ir SImpleDB, CouchDB negarantuoja nuoseklumo. Skirtingai nuo SimpleDB, kiekvienas klientas mato mato savo atveju nuoseklų (self-consistent) vaizdą duomenų bazėje, su pakartotinais skaitymais: CouchDB turi keliaversijinę lygiagrečią individualių dokumentų kontrolę, su užklausos ID, kuris yra automatiškai sukurtas kiekvienai dokumento versijai. CouchDB išves pranešimą, jeigu kažkas atnaujins dokumentą, nuo tada kai jis buvo įtrauktas į sistemą. Aplikacija galės kombinuoti atnaujimus: atnaujinti arba perrašyti.

CouchDB taip pat suteikia galimybę sistemai sugriūti Visi atnaujinimai (dokumentai ir indeksai) yra perkelti į diską, rašant iki failo pabaigos. Pagal nutylėjimą, įrašymas į diską vyksta po kiekvieno dokumento atnaujinimo. Kartu su MVCC mechanizmu, CouchDB teikia dokumento lygio ACID semantikas.

Klientai kviečia CouchDB per RESTful protokolą. Daugybė programavimo kalbų turi bibliotekas, kurios konvertuoja įprastus API pakvietimus į Restfull pakvietimus. CouchDB turi bazinį duomenų bazių adminstravimo funcionalumą.

3.2.3. MongoDB

MongoDB yra GPL atviro kodo dokumentų saugykla, parašyta C++ programavimo kalba. Ji turi panašumų su CouchDB: kolekcijose turi indeksus, neturi užraktų ir teikia dokumentų užklausų mechanizmą. Vis dėlto turi esminių skirtumų:

1) MongoDB palaiko atomatinį shardinimą, paskirsto dokumentus per serverius

2) Kopijavimas MongoDB yra daugiausiai naudojamas apsisaugojimui nuo klaidų, o ne skaliuojamumui, kaip kad CouchDB. MongoDB neteikia globalaus nuoseklumo tradiciniam DBVS, bet galima gauti lokalų nuosekluma naujausiai pirminei dokumento kopijai

3) MongoDB palaiko dinamines užklausas su automatiniu indeksų naudojimu, kaip realiacinė duomenų bazių valdymo sistema.

4) CouchDB suteikia dokumentams MVCC, kol tuo tarpu MongoDb suteikia atominių operacijų laukus.Atominių operacijų laukų funkcionalumas:

4.1) „Atnaujinimo komanda palaiko modifikatorius kurie palengvina atominius pakeitimus į reikšmes: $set - priskiria reikšmę; $inc - praplėčia reikšmę, $push - prideda reikšmę į masyvą, $pushAll - įdeda kelias reikšmes į masyva, $pullAll - išima kelias reikšmes iš masyvo. Kuomet šie atnaujinimai yra įvyksta jie išvengia nereikalingo kreipimosi į server" [SSY+11]

4.2) Yra „atnaujinti, jeigu dabartinis" konvencija, keičianti dokumentą tik jeigu lauko reikšmė sutampa su duota prieš tai reikšme

4.3) „MongoDB palaiko „findAndModify" komandą, atlikti atominiam atnaujinimui ir tuoj pat grąžinti atnaujintą dokumentą." [SSY+11] Tai yra naudinga, formuojant užklausas kur duomenų struktūra reikalauja atomiškumo.

MongoDB indeksai yra aiškiai apibrėžti naudojant „ensureIndex" kvietimus ir visi egzistuojantys indeksai yra automatiškai panaudoti užklausos vykdymui. Jeigu indeksai yra apibrėžti kaip užklausiami laukai, MongoDB automatiškai juos naudos. „MongoDB taip pat palaiko „map-decuce", kuris leidžia sudėtingas agregacijas per dokumentus." [PPV13]

„MongoDB saugo duomenis binariniu JSON tipo formatu vadinamu BSON" [SSY+11] . BSON palaiko loginę reikšmę, sveiką skaičių, dešimtainį skaičių, datą, tekstą ir binarinius tipus. Kliento tvarkyklė iškoduoja lokalios kalbos dokumento duomenų struktūrą. Į BSON ir išsiunčia į MongoDB serverį per „socket" susijungimą. MongoDB taip pat palaiko GridFS specifikaciją dideliems binariniams objektams, paveikslėliams ir vaizdo įrašams. Jie yra saugomi gabalais, kad būtų galima efektyviai srautu juos teikti vartotojui.

MongoDB palaiko automatinį atsistatymą po sistemos sugriūvimo. Kopijavimas(ir atstatymas) yra atliekamas shardinimo lygmenyje. Kolekcijos yra automatiškai sushardinamos su vartotojo nurodytu shadinimo raktu. „Kopijavimas yra asinchroniškas greitesniam veikimui, todėl atnaujinimas gali būti prarastas sistemos klaidos atveju." [CS10]

3.2.4. Terrastore

Terrastore yra sukurta su Terracotta (Java VM produktas). Kaip ir dauguma kitų NoSQL sistemų, kliento prieiga į Terrastore yra pastatyta su HTTP operacijomis, kad paduoti ir išsaugoti duomenis.

Java ir Python API taip pat buvo implementuota. Terrastore automatiškai paskirsto duomenis serverio gijose ir gali automatiškai perskirstyti duomenis, kuomet serveriai yra pridedami arba panaikinami. Kaip ir MongoDB, gali atlikti užklausas su predikatais, įtraukiant užklausas su rėžiais. Kaip ir CouchDB, įeina „map-reduce" mechanizmas sudėtingesnėms paieškoms ir agreguotiems duomenims. Taip pat kaip ir kitos dokumentų duomenų bazės, Terrastore neturi schemos, neteikia ACID tranzakcijų. Kaip ir MongoDB teikia nuoseklumą dokumentų apdorojime: skaitymas visada ims vėliausią dokumento versiją. Terrastore palaiko kopijavima ir sistemos klaidų šalinimą

3.2.5. Dokumentų tipo saugyklų apibendrinimas

Dokumentų saugyklos yra be schemės, išskyrus atributus (kurie yra tik pavadinimai ir nėra išanksto specifikuojami), kolekcijas (kurios paprasčiausiai grupuoja dokumentus) ir indeksus nurodytus kolekcijose. Dokumentų saugyklos yra labai panašios bet naudoja skirtingą terminologiją. Pavyzdžiui SimpleDB domenas = CouchDB duomenų bazė = MongoDB Kolekcija = Terrastore Bucket. SimpleDB dokumentai vadinami „items", atributai yra laukai CouchDB duomenų bazėje, arba ranktas MongoDB ir Terrastore duomenų bazėse.

Skirtingai nei raktinių reikšmių saugyklos, dokumentų saugyklos teikia mechanizmą, kelekcijų užklausoms su keliomis atributų reikšmėmis (apribojimais). Vis dėlto CouchDB nepalaiko ne procedūrinių užklausų kalbos: programuotojui reikia daugiau darbo panaudoti indeksus. Dokumentų saugyklos neturi užraktų, turi silpnesnį lygiagretumą ir atomiškumą negu tradicinės ACID atitikmens duomenų bazės. Dokumentai gali būti paskirstyti per gijas visose sistemose, bet skaliuojamumas skirsis. Visos sistemos gali pasiekti gerus skaliuojamumo rezultatus, leidžiant skaityti ne pačias naujausias kopijas. MongoDB ir Terrastore gali pasiekti skaliuojamumą, be minėto kompromiso, jos gali skaliuoti įrašymus per automatinį shardinimą ir atomines operacijas dokumentuose. CouchDB gali pasiekti įrašymo skaliuojamumą patilekiant CouchDB Lounge.

3.2.6. Dokumentų saugyklų naudojimo pavyzdys

Geras aplikacijos pavyzdys skirtas dokumentų saugyklai būtų keletas skirtingų tipų objektų, kur reikėtų daryti objektų paieškas pagal keletą laukų iškarto. Pavyzdžiui automatizuotų transporto priemonių aplikacija (ATP), kuri turi vairuotojus ir transporto priemones ir atitinkamai gali reikėti atlikti paiešką pagal vairuotojo vardą, vairuotojo pažymėjimo numerį, turimą transporto priemonę, gimimo datą. Svarbus faktorius apsvarstymui yra kokio lygmens lygiagretumo mums reikia. Jeigu mus tenkina „galiausiai nuoseklus" modelis su ribotu automatiškumu ir izoliacija - dokumentų

saugykla tinka labai gerai. Tai gali būti atvejis ATP aplikacijoje: vargiai tikėtina, kad du ATP ofisai atnauins to paties vairuotojo informaciją tuo pačiu metu. Tačiau jeigu reikia,kad informacija būtų atnaujinta automatiškai nuosekliai, tuomet reikia ieškoti kitų alternatyvų arba naudoti „quorum-read" arba panašų mechanizmą.

3.3. Išpėsto įrašo saugyklos

Išplėsto įrašo saugyklos buvo įkvėptos google sėkme su BigTable. Duomenų modelis yra eilutės ir stulpeliai, skaliuojamumo modelis skaido eilutes ir stulpelius į keletą gijų:

- „Eilutės yra skaidomos į gijas shardinant, naudojant pirminį raktą. Dažniausiai yra skaidomos pagal rėžį, o ne pagal maišymo funkciją. Tai reiškia, kad užklausos patenkančios į rėžius neturi eiti į skirtingas gijas" [AJ15]
- Lentelės stulpeliai yra paskirstyti per keletą gijų naudojant stulpelių grupavimą.

Šie du paskirstymai (horizontalus ir vertikalus) gali būti panaudoti vienu metu, toje pačioje lentelėje. Stulpelių grupės turi būti išanksto priskirtos su išplėsto įrašo saugyklomis. Vis dėlto tai nėra didelis apribojimas, nes nauji atrbutai gali būti pridedami vykdymo metu. Eilutės yra analogiškas dalykas dokumentams: jos gali turėti kintančius kiekius atributų (laukų), atributų pavadinimai privalo būti unikalūs, eilutės grupuojamos į kolekcijas (lenteles) ir individualios eilutės atributai gali būti betkokio tipo. (CouchDB ir MongoDB palaiko nested objektus, kol tuo tarpu išplėsto įrašo saugyklos palaiko tik skaliuojamus tipus).

3.3.1. HBase

HBase yra Apache projektas parašytas Java programavimo kalba. Jo karkasas yra panašus į BigTable:

- HBase naudoja Hadoop išplatintą failų sistemą vietoje Google failų sistemos. Ji krauna atnaujinimus į atmintį ir periodiškai įrašo juos į failus diske
- Atnaujinimai eina į duomenų failo pabaigą, tam, kad išvengti ieškojimų. Failai yra periodiškai sutankinami. Atnaujinimai taip pat eina į įrašų priekį, atstatymui serverio nulūžimoatveju
- Eilutės operacijos yra atominės, su eilutės lygmens užrakinimu ir tranzakcijomis. „Yra naudojama optimistinio nuoseklumo kontrolė, nutraukianti veiksmą, jeigu yra konfliktų su kitais atnaujinimais" [MT13]
- „Skirstymas ir platinimas yra aiškūs: klientinėje dalyje nėra hashinimo ar fiksuotų raktinių tarpų kaip kaikuriose NoSql sistemose. „MapReduce" leidžia operacijoms būti paskirstytoms efektyviai." [ZNH+15]

- HBase B-Trees leidžia greitas rėžines užklausas ir rikiavimą

3.3.2. HyperTable

HyperTable yra parašytas su C++ programavimo kalba. Tai yra Zvents atviro kodo produktas. HyperTable yra labai panašus į HBase ir BigTable, jis naudoja stulpelių šeimas, kurios gali turėti betkokį kiekį stulpelių „qualifiers”. Duomenims su MVCC yra naudojamos laiko žymos. „HyperTable reikalauja parindinės platinamų failų sistemos tokios kaip Hadoop ir platinamo užrakto valdiklio.“ [JPY+13] Lentelės yra kopijuojamos dalijant per serverius, pagal rėžius. Atnaujinimai yra daromi atmintyje, o vėliau surašomi į diską. Hypertable palaiko daugybės pragramavimo kalbų kliento sąsajas. Naudojama užklausų kalba vadinama HQL

3.3.3. Cassandra

Cassandra yra panaši į kitas praplėsto įrašo saugyklas savo duomenų modeliu ir baziniu funkcionalumu. Ji turi stulpelių grupavimą, atnaujinimai yra laiko atmintyje ir tuomet įrašomi į diską, diske esanti informacija periodiškai suspaudžiama. Dalijimas ir duomenų kopijavimas yra galimi. Klaidų aptikimas ir atstatymas yra pilnai automatiškas. Tačiau „Cassandra turi silpnesnį lygiagretumo modelį negu dalis kitų sistemų: nėra rakinimo mechanizmo ir kopijos yra atnaujinamos asinchroniškai.” [AB13]

Kaip ir HBase, Cassandra yra parašyta Java programavimo kalba ir naudojama pagal Apache licenziją. Kliento sąsaja yra sukurta naudojant Facebook Thrift karkasą. Cassandra automatiškai suteikia naujų galimų mazgų į klasterį, naudoja „phi accrual” algoritmą aptikti klaidingas gijas. Cassandra prideda „super stulpelio” sąvoką, kuri leidžia kito lygmens grupavimą su stulpelių grupėmis. Duomenų bazės (keyspaces) turi stulpelių grupes, stulpelių grupės turi „super stulpelius” arba stulpelius (bet ne abu iškart), super stulpeliai turi stulpelius. Kaip ir kitose sistemose, betkuri eilutė, gali turėti betkurią kombinaciją stulpelio reikšmių. (eilutės yra kintamo ilgio ir nėra apribotos lentelės schema).

„Cassandra naudoja rikiuotus hash indeksus, kurie duoda didžiają dalį privalumų iš hash ir B-tree indeksų: galima žinoti kurios gijos gali turėti tam tikrą ilgį vietoje to, kad ieškoti visose gijose.” [SSY+11]

„Tačiau rikiavimas išlieka lėtesnis negu B-trees.” [SSY+11]

3.3.4. Išplėsto įrašo tipo duomenų bazių apibendrinimas

Išplečiamų įrašų saugyklos daugiausiai yra Bigtable pagrindo. Jos visos panašios, bet skiriasi lygiagretumo mechanizmais. Cassandra fokusuojasi į „silpną" lygiagretumą (su MVCC), o HBase ir HyperTable į „stiprų" lygiagretumą (užraktai ir loginimas).

3.3.5. Išplėsto įrašo saugyklų naudojimo pavyzdys

Naudojimo atvejai išplėsto įrašo saugykloms yra panašūs kaip ir dokumentų saugykloms: kelių tipų objektai su ieškojimu betkuriam laukui. Vis dėlto išplėsto įrašo saugyklos projektai įprastai yra fokusuoti į aukštesnį pralaidumą ir teikia geresnį lygiagretumo garantą už nedidelį sudėtingumo prieaugį palyginus su dokumentų saugyklomis.

Tarkime mes ketiname saugoti informaciją apie vartotojus, eBay stiliaus aplikacijoje ir ketiname padalinti savo duomenis horizontaliai ir vertikaliai:

1) Vartotojus sugrupuoti pagal šalį, kad galima būtų efektyviai ieškoti vartotojus vienoje šalyje.
2) Retai keičiamą informaciją, pavyzdžiui vartotojo adresą saugoti vienoje vietoje, o dažnai atnaujinamą informaciją, pavyzdžiui informaciją apie statymus, saugoti kitoje vietoje, tam kad pagerinti našumą.

Tokį horizontalų / vertikąlų skaidymą galima padaryti ir dokumentų saugykloje sukuriant kelias kolekcijas, kelioms dimensijoms, tačiau lengviausiai skaidymas padaromas naudojant pratęsto įrašo saugyklas kaip HBase arba HyperTable.

3.4.Skaliuojamos releacinės sistemos

Skirtingai nuo kitų duomenų saugyklų, releacinės duomenų bazių valdymo sistemos turi išbaigtą išankstinę schemą, Sql sasają ir ACID tranzakcijas. Tradiciškai RDBVS nepasiekia tokio skaliuojamumo kaip Nosql duomenų saugyklos. Tačiau pastaraisiai metais stipriai patobulėjo, bei išėjo naujų produktų (VoltDB, Clustrix), kurie turi gerą skaliuojamumą ir darbo paskirstymą tarp gijų. Releacinės sistemos lyginant su NoSQL teikia panašų skaliuojamumą, bet turi du skirtumus:

1) Naudoja mažos apimties operacias: operacijos kurios naudoja daug mazgų, susijungia per daug lentelių, gerai nesiskaliuos shardinant
2) Naudoja mažos apimties tranzakcijas: tranzakcijos kurios naudoja daug mazgų yra labai neefektyvios.

NoSQL sistemos vengia šių dviejų problemų apribodami, kad būtų negalima arba labai sudėtinga naudoti didelės apimties operacijas ir tranzakcijas. Skaliuojamos RDBVS neturi uždrausti didelės

apimties operacijų - jos paprasčiausiai „baudžia" vartotoją už naudojimą. Skaliuojamos RDBVS turi privalumą prieš NoSQL duomenų saugyklas, nes jos turi aukštesnio lygmens SQL kalbą ir ACID nustatymus, bet reikia „sumokėti kainą" kai jos naudoja mazgus

3.4.1. MYSQL Cluster

MYSQL Cluster yra dalis MySQL. „MySQL Cluster veikia pakeisdamas InnoDB variklį NDB API." [CS10] Jis yra prieinamas iš MYSQL (dabar Oracle) ir nėra atviro kodo. MYSQL Cluster shardina duomenis per keletą serverių („sharding nothing" architektūra). „Kiekvienas shardas yra nukopijuotas, atstatymo galimybei. Abipusis kopijavimas taip pat yra palaikomas." [CS10]

MYSQL klasteris palaiko duomenis ir atmintyje ir diske. Saugojimas atmintyje leidžia atsakyti realiu laiku. Nors MYSQL klasteris leidžia skaliuoti per daugiau mazgų negu kitos RDBVS, realiai susiplnėja pokelių tuzinų mazgų.

3.4.2. VoltDB

VoltDB yra ganėtinai nauja atviro kodo releacinė duomenų bazių valdymo sistema turinti gerą skaliuojamumą. Skaliuojamumas ir galimybės yra konkurencingos MySQL Custer'iui ir NoSQL sistemos:

- Lentelės suskirstytos per serverius ir klientai gali kviesti betkurį serverį. „Paskirstymas yra nematomas sql vartotojams, bet vartotojas gali pasirinkti shardinimo atributus." [LSR14]
- Pasirinktos gijos gali buti kopijajamos serveriuose (greitam pasiekimui, pvayzdžiui skaitymo operacijai)
- „Betkuriuo atveju shardai yra kopijuojami, duomenys gali būti atkurti gijos nulūžimo atveju." [LSR14] Duomenų bazės esama būnesa taip pat bus palaikoma, tęs darbą arba bus įtraukta į darbų planą.

Lyginant su kitomis duomenų bazių valdymo sistemomis VoltDB turi gerą vieno mazgo produktyvumą. VoltDB pašalina visus „laukimus" SQL vykdyme, taip leisdamas labai efektyvią implementaciją:

- Sistema yra sukurta duomenų bazei kuri telpa į (išplatintą) operatyvią atmintį serveriuose, todėl sistema niekada neturi laukti kietojo disko. Indeksai ir įrašų struktūros yra pritaikyti operatyviajai atminčiai, o ne diskui ir podėlis / buferis yra nenaudojami taip pat. Našumas bus labai prastas jeigu virtuali atmintis viršys operatyvią, bet esant geram operatyviosios atminties santykiui efektyvumas bus labai geras.
- „SQL vykdymas yra vieno mazgo kiekvienam shardui, naudojant „sharded-nothing" architektūrą, todėl nėra išlaidų dėl daugiamazgio blokavimo." [MKM12]
- „Visi SQL kvietimai yra padaryti per išsaugotas procedūras, su kiekviena procedūra yra viena tranzakcija." [LSR14] Tai reiškia, kad jeigu duomenys yra shardinti, taip kad leistų vieną tranzakciją viename mazge, tuomet nereikalingi rakinimai, to pasekoje nėra laukimų rakinant. Tranzakcijų kordinavimas yra taip pat nenaudojamas.
- „Išsaugotos procedūros yra kompiliuojamos tam kad pateikti kodą palyginamą su pasiekimo lygio kvietimais NoSQL sistemoje. Jie gali buti paleisti iš to paties mazgo arba mazgo kopijų." [MKM12]

VoltDB kūrėjų teigimu, šios optimizacijos ženkliai sumažina mazgų kiekį reikalingą palaikyti duotai aplikacijai, su minimaliais apribojimais duomenų bazių projaktavime. „Tam kad pasiekti maksimalius rezultatus, veikianti sistema turi tilpti į operatyvios atminties ribas." [MKM12]

3.4.3. Clustrix

Clustrix siūlo produktą panašų į VoltDB ir MySQL klasterį, bet Clustrix mazgai yra realizuoti, kaip „imontuoti" įtaisai. Jie teikia skaliuojamumą šimtams mazgų su automatiniu shardinimu ir kopijavimu. Klaidų aptikimo mechanizmas yra automatinis ir mazgai kuriuose įvyko klaidos atsistato automatiškai. Jie taip pat naudoja kietuosius diskus papildomam efektyvumui pasiekti. Kaip ir kitos releacinės sistemos, Clustrix palaiko SQL su pilnai ACID tranzakcijomis. Duomenų paskirstymas ir apkrova yra nematoma aplikacijos programuotojui. Taip pat, sistema yra sukurta taip, kad yra sklandžiai suderinama su MySQL ir gali palaikyti egzistuojančias MySQL aplikacijas ir front-end sąsajas.

3.4.4. ScaleDB

ScaleDB yra naujas MySQL darinys. Skirtingai nuo MySQL Cluster, jis pakeitė InnoDB variklį ir naudoją klasterinimą per kelis serverius, tam kad pasiekti skaliuojamumą. ScaleDB skiriasi tuo, kad reikalauja diskų pasidalijimo mazgams. Kiekvienas serveris turi turėti prieigą prie kiekvieno disko.

ScaleDB's shardinimas yra automatinis: daugiau serverių gali būti pridėta betkuriuo metu. Serverių klaidų tvarkymas yra taip pat automatinis. ScaleDB perskirsto apkrovą per egzistuojančius serverius. ScaleDB palaiko ACID tranzakcijas ir eilutės lygmens rakinimus. Turi kelių lentelių(multi-table) indeksavimą (jis yra galimas dėl diskų pasidalijimo).

3.4.5. Skaliuojamų releacinių sistemų apibendrinimas

MySQL klasteris naudoja „shared nothing" architektūrą, kaip ir dauguma kitų sistemų savo sprendimus, geram skaliuojamumui. VoltDB turi gerą horizontalų skaliuojamumą ir gerą darbo efektyvumą tarp mazgų. Clustrix taip pat turi gerą charakteristiką šioje vietoje ir palaiko kietuosius diskus, tačiau yra pagrįstas patentuota programine įranga.

Teoriškai releacinės duomenų bazių valdymos sistemos turėtų suteikti gerą skaliuojamumą tol kol aplikacijos leidžia naudoti daugiamazges operacijas. SQL per ACID tranzakcijas paprastumas daugumoje aplikacijų suteikia privalumą prieš NoSQL.

3.4.6. Skaliuojamų RDBVS pavyzdys

Releacinių DBVS privalumai yra šiose sferose:

1) Jeigu aplikacija reikalauja daugybės lentelių su skirtingais duomenų tipais, releacinės schemos apibrėžimas centralizuos ir supaprastins duomenų apibrėžimą ir SQL padarys operacijas, kurios apima lenteles, paprastesnėmis.
2) Dauguma programuotojų yra įgudę dirbti su SQL ir teigia jog SQL naudojimas yra paprastesnis negu žemesnio lygmens komandos, teikiamos NoSQL sistemų.
3) Tranzakcijos padaro lygiagrečių prieigų kodą paprastesniu. ACID semantikos leidžia programuotojui nenaudoti užrakinimų, garantuoja, kad nesusidurs su nenaujausiais duomenimis, atnaujinimo klaidomis ir nereikia vargintis dėl nuoseklumo.
4) Releacinės DBVS turi daugybę galimų papildomų įrankių skirtų ataskaitų generavimui, formoms ir t.t.

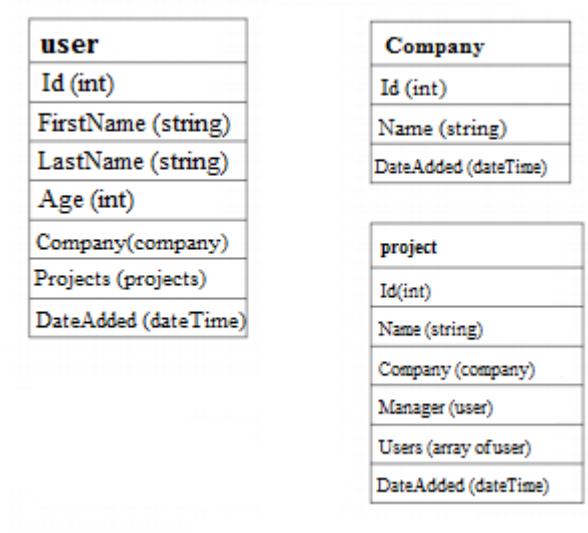
Geras releacinės duomenų bazių valdymo sistemos naudojimo pavyzdys būtų pasunkinta ATP(automatizuotų transport priemonių) aplikacija, kur būtų sąsaja ir galima ieškoti transport priemonę, pagal spalvą, modelį, metus ir t.t. su apribojimais savininkui,tokiais kaip valstybė, plaukų spalva, lytis ar panašiai. ACID tranzakcijos leistų atnaujinimus iš daugybės lokacijų. Šie RDBVS teikiami privalumai tenkina aplikacijos poreikius. Reikia užsitikrinti kad aplikacija nereikalaus atnaujinimų ir sujungimų kurie apims daug mazgų, nes priešingu atveju duomenų migracija būtų labai

brangi. Vis dėlto, NoSQL sistemos išvis negali pasiūlyti tranzakcijų arba užklausų sujungimų per mazgus.

4. SQL ir NoSQL DBVS palyginimas

Savo eksperimentui pasirinkome dvi SQL ir dvi NoSQL sistemas patenkančias į populiariausių dešimtuką. Mes įrašėme MongoDB, Cassandra, Mysql ir Microsoft SQL Server Express į kompiuterį kurio duomenys: Intel i5 (keturių branduolių) 2.4 GHz procesorius, 8 GB DDR3 1600 Mhz. operatyvi atmintis. Visos keturios duomenų bazės buvo įrašytos ir saugojo duomenis SSD(Solid State Drive) diske, kad veiktų taip greit kaip tik galima. Visos keturios testinės aplikacijos parašytos C# programavimo kalba, naudojant MonoDevelop IDE. SQL Server aplikacijai naudojome „entity framework", duomenų įrašymui ir ojektų susiejimui, tačiau duomenų atnaujinimams užklausos buvo rašomos naudojant SQL, kad rezultatai nebūtų iškreipti. MySQL naudojome 5.7.12 versiją, tiek duomenų įrašymai, tiek duomenų traukimai ir atnaujinimai buvo vykdomi standartinėmis SQL užklausomis. MongoDB naudojome 3.2 versiją, visos užklausos buvo parašytos MongoDB užklausomis. Cassandra buvo naudojama 3.4 versija, visos užklausos buvo rašomos Cassandra programavimo kalba (CQL).

Kadangi yra atlinkta daug tyrimų, kur lyginami DBVS pajėgumai su dideliais kiekiais duomenų, mes savo testams pasirinkome tirti rezultatus mažos apimties aplikacijoje. Mūsų testai susideda iš keturių skirtingų testų. 2 pav. parodo objektų schemą naudojamą testams. Kaip parodyta 2 pav., duomenys susideda iš 3 lentelių / kolekcijų nusakančių įmonę, projektą ir vartotoją. Yra ryšiai tarp šių objektų: vartotojas turi atitinkamą imonę ir projektų masyvą.



2 pav. Objektų schema

Releacinės duomenų bazės modelio implementacijoje 2 pav., užklausos turi daugiau negu vieną lentelę ir reikalauja sujungimo (join) operacijos. Pavyzdžiui: užklausa apimanti vartotoją ir imonę, reikalauja sujungimo(join) tarp vartotojo ir imonės lentelių, pagrįsto atitinkama imonės ir ID reikšme. Kadangi SQL sistemos (MsSQL ir MySQL) turi schemą, informacija yra traukiama, atnaujinama ir trinama susiejant lenteles paprasta „Join" operacija. Tačiau MongoDB ir Cassandra duomenų bazių valdymo sistemose, tokios užklausos reikalauja kopijuoti imonės duomenis į Vartotojo kolekciją(„nested" užklausa) arba paieškos pagrįstos raktine reikšme. Taip pat pabrėžtina yra tai, kad duomenų bazės implementacija 2 pav., reikalauja 3 lentelių kiekvienam duomenų objektui. Siejant lenteles, papildomos lentelės sutinka santykiu M:N, tarp vartotojų ir projektų objektų. Ši papildoma lentelė pakeičia projektų masyvą vartotojų lentelėje ir vartotojų masyvą projektų lentelėje

3 pav. iliustruoja imonių, vartotojų ir projektų skaičių, kiekvienam iš keturių testavimo atvejų (test cases). Išviso, panaudotų tuple[7] kiekis svyruoja nuo 148 (pirmame Testavimo atvejyje), iki 12416 (ketvirtame testavimo atvejyje). Select'ų skaičius bus aptartas vėliau.

Test case	Companies	Users	Projects	Selected times
1	4	128	16	100
2	4	256	64	200
3	16	1024	512	100
4	128	4096	8192	200

3 pav. Objektų skaičius testavimo atvejuose

Eksperimentams buvo sukuros atskiros C# aplikacijos, kurios naudojo tą patį kodą visoms sistemoms. Įrašymo kodas skyrėsi tik nuorodomis į skirtingas duomenų bazes. Metodai skirti atnaujinimui ir užklausoms yra labai skirtingi savo implementacijomis. Releacinė duomenų bazė naudoja SQL sintaksę, kol tuo tarpu MongoDB ir Cassandra DBVS naudoja Javascript Objektinę notaciją (JSON) ir CQL. Bendrinės funkcijos taip pat skiriasi SQL ir NoSQL implementacijose, dėl MapReduce funkcijų, kurios prideda sudėtingumo.

Testavimui buvo įrašyti duomenys, kaip kad parodyta 3 pav. Sąryšiai tarp duomenų objektų, kaip kad imonės ir vartotojo objektas, buvo implementuoti MongoDB ir Cassandra saugojant nuorodą į kitą dokumentą, o ne plečiant dokumentą. Pavyzdžiui nuoroda į imonę buvo saugoma vartotojo objekte.

Atnaujinimams buvo panaudoti trys skirtingi atnaujinimų tipai. Pirmasis tipas įtraukė vienos dešimtosios imonių vartotojų vardų atnaujinimą, pagrįstą jų pirminiu raktu. Antrasis tipas atnaujino pavardes visų vartotojų, kurių vardas sutapo su pasirinktuoju. Trečiasis tipas atnaujimo vieną ketvirtąją projektų pavadinimų, pagrįstų jų raktu. Testavimo atvejuose (test cases) buvo testuojami visi trys atnaujinimų tipai.

Taip pat buvo testuojamos septynios skirtingos Select užklausos. Select'ai buvo suskirstyti į dvi grupes: paprastus ir sudėtingus. Paprastos užklausos įtraukė tik vieno tipo duomenų traukimą. Sudėtingos užklausos įtraukė kelių tipų objektus, „nested" užklausas ir agregavimo funkcijas. Testavimo atvejai 1 ir 3 buvo vykdomi 100 kartų, o testavimo atvejai 2 ir 4 buvo vykdomi 200.

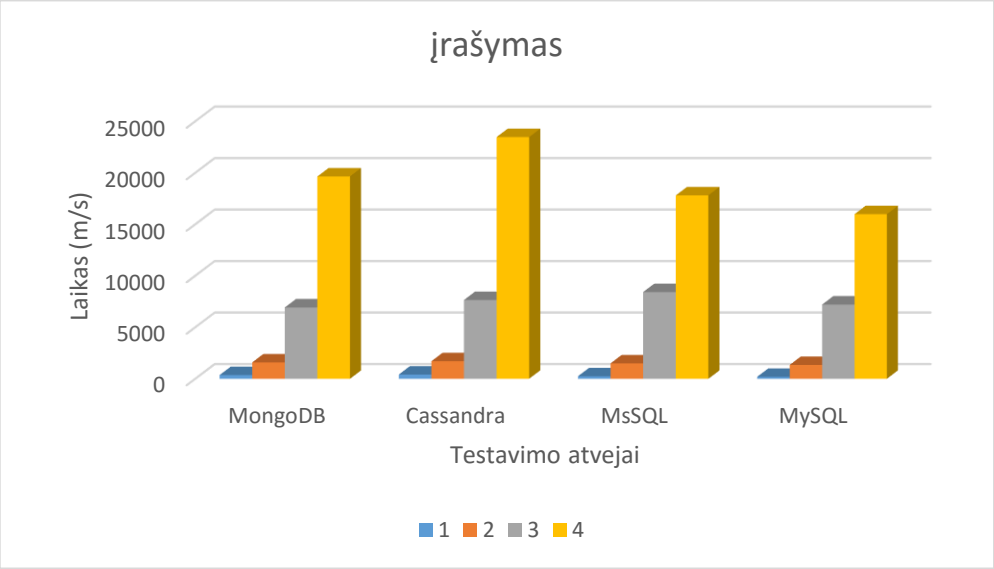
Testavimui buvo sugeneruotos keturios skirtingos paprastos select užklausos. Pirmoji select užklausa gavo imonę, pagal pirminį raktą. Antroji select užklausa gavo imonę, pagal atsitiktinai pasirinktą pavadinimą. Visos imonės turėjo unikalius pavadinimus ir pavadinimas garantavo, kad bus rastas vienas atitikmuo imonių kolekcijoje. Trečia select užklausa gavo duomenis vartotjų kurių vardas sutapdavo su atsitiktiniu vardu. Šis atsitiktinai parinktas vardas galėjo neturėti atitikmenų kolekcijoje.

Visos trys skirtingos sudėtingos select užklausos naudojo sujungimo(join) operaciją. Pirmoji užklausa gaudavo duomenis visų imonių kurioms priklauso vienas arba daugiau vartotojų su atsitiktinai pasirinktu vardu. Kadangi vardai buvo suvesti atsitiktinai, tikėtina, kad užklausa rezultatų negrąžins. Antroji užklausa gaudavo visus vartotojus kurie dirbo prie bet kurio iš trijų atsitiktinai pasirinktų projektų. Kadangi vartotojai būdavo atsitiktinai priskiriami projektams, šios užklausos

rezultatas taip pat galėjo neturėti validžių atitikemenų. Paskutinė užklausa gaudavo vidutini amžių darbuotojų, kurie dirbo prie trijų atsitiktinai pasirinktų projektų (panaudota agregavimo funkcija average()). Rezultatas gali būti 0, jeigu nėra vartotojų priskirtų projektui.

4.1.DBVS palyginimo rezultatai

Tyrimo rezultatai pavaizduoti 4 pav. – 13 pav. diagromose. Kiekviena diagrama palygina vidutinį SQL ir NoSQL implementacijų vykdymo laikus milisekundėmis, atliekant sudarytus 4 Testavimo atvejus. Buvo naudojamas automatizuotas įrankis, kuris paleido kiekvieną iš testų nuo 100 iki 200 kartų, kad turėti vidutinį laiką ir sumažinti duomenų iškreiptumo tikimybę.

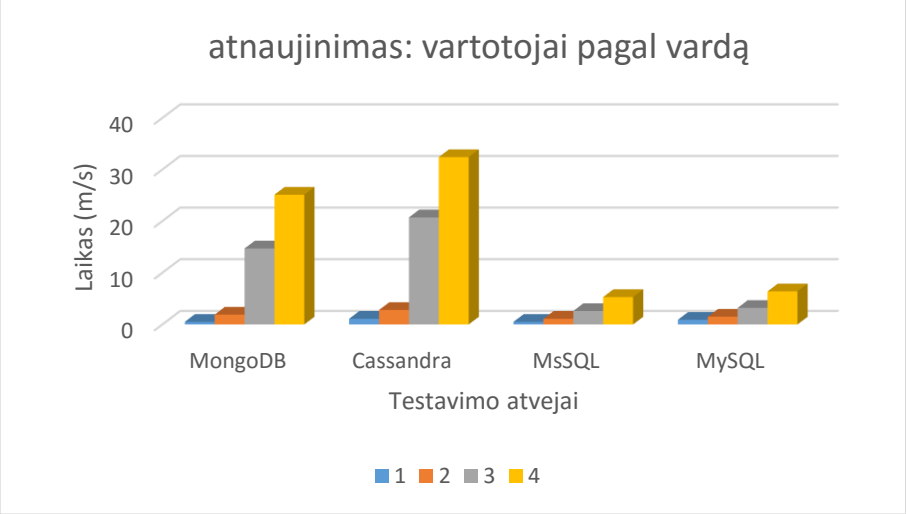


4 pav. Įrašymas

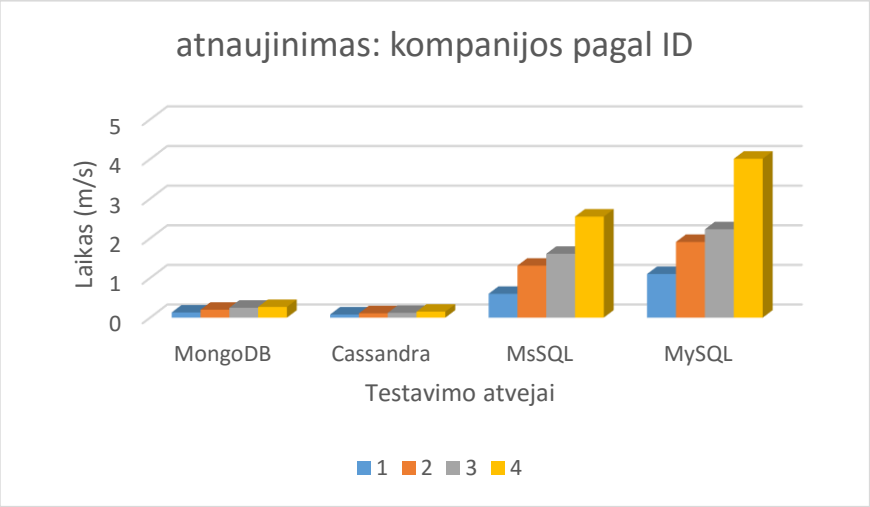
4 pav. vaizduoja įterpimo greitį, tarp keturių duomenų bazių sistemų. Bendrai vertinant MySQL testuose pasirodė geriausius rezultatus, likusios trys sistemos parodė skirtingus rezultatus. MsSQL yra greitesnė už MongoDB ir Cassandra Testavimo atvejuose 1, 2 ir 4, tačiau MongoDB yra greitesnė už SQL atliekant 3 Testavimo atvejį, taip pat MongoDB pasirodė geriau už Cassandra visuose keturiuose testavimo atvejuose. MongoDB ir Cassandra duomenų įrašymo implementacijoje objektų ir kolekcijų susiejimai buvo įgyvendinti naudojant nuorodas(reference), todėl nereikėjo daryti papildomų duomenų įrašymų. MongoDB buvo 52% lėtesnė už MySQL ir 32% lėtesnė nei MsSQL atliekant pirmą testavimo atvejį, tačiau 15% greitesnė už Cassandrą. Antrame testavimo atvejyje skirtumas tarp visų keturių sistemų vykdymo laikų buvo santykinai mažiausias, reultatai svyravo nuo 1351 ms. iki 1712 ms. Greičiausiai testą įvykdė MySQL, o lėčiausiai Cassandra. Trečiame testavimo atvejyje MsSql rezultatai buvo 14% blogesni negu MySql, 9% blogesni negu Cassandra ir 18 % blogesni negu MongoDB. Ketvirtame teste, SQL DBVS pasirodė geriau negu NoSQL: Mysql atliko

testą greičiausiai, o MsSQL buvo dešimtadaliu greitesnė už MongoDB ir ketvirtadaliu greitesnė už Cassandra. Nors MySQL rezultatai šiuose testuose buvo geriausi, negalima tvirtai teigti, jie geriausi bendrai dirbant su nedideliais kiekiais duomenų visų keturių sistemų rezultatai nesiskiria stabiliai – trečiame testavimo atvejyje MongoDB aplenkė.

5 pav., 6 pav. ir 7 pav. diagramos vaizduoja laikus milisekundėmis, kiekvienam iš skirtingų atnaujinimų. Atnaujimas 5 pav. vaizduoja atnaujinimą pagrįstą vardu, o vardas nebuvo niekaip indeksuojamas. Abi NoSQL sistemos parodė blogesnius rezultatus nei SQL sistemos. Kadangi atnaujinimai vyko remiantis neindeksuotu stulpeliu, MongoDB ir Cassandra užklausos turėjo atlikti sudėtingas paieškas kiekvienam įrašui, todėl pareikalavo daugiau laiko. Lyginant tarpusavyje, abi SQL sistemos parodė labai panašius rezultatus ir abi NoSql sistemos, taip pat parodė panašius rezultatus. MsSQL parodė neženkliai, bet visuose testuose geresnius rezultatus už MySQL. Taip pat pastebimas dalykas yra tai, kad didėjant duomenų kiekiui NoSQL sistemų rezultatai santykinai blogėjo labiau negu SQL sistemų. Nors Cassandra rezutatai stabiliai blogesni negu MongoDB visuose keturiuose testuose, didėjant duomenų kiekiui rezultatai darėsivis panašesni, esant didesnei duomenų imčiai, Cassandra atliktų testus geriau. Pirmame testavimo atvejyje, esant mažam duomen kiekiui, skirtumas tarp visų keturių sistemų nėra didelis, svyruoja nuo 0,5 ms. iki 1,1ms. Antrame testavimo atvejyje MySql įvykdė atnaujinimus vidutiniškai 27% lėčiau, bet 21% greičiau negu MongoDB ir 44% greičiau negu Cassandra.



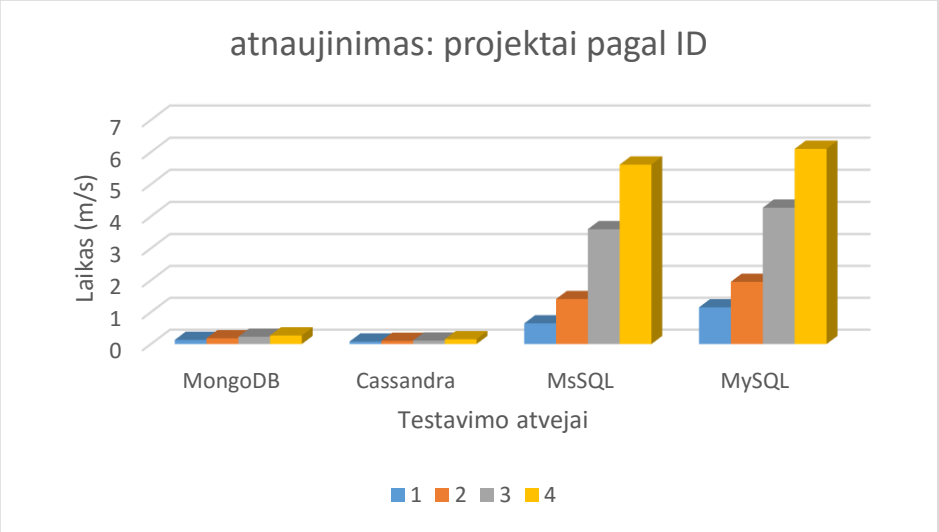
5 pav. Vartotojų atnaujinimas pagal vardą



6 pav. Kompanijos atnaujinimas pagal ID

6 pav. diagramoje vaizduojami kompanijos atnaujinimo pagal kompanijos ID, testų rezultatai. Visuose keturiuose testuose NoSQL sistemos pasirodė ženkliai geriau negu SQL. Taip yra todėl, kad užklausoje reikšmės buvo ieškomos pagal raktinę reikšmę. Lyginant sistemas tarpusavyje matosi, kad SQL pusėje, MsSQL rezultatai yra stabiliai geresni nei MySQL, o NoSQL pusėje – Cassandra rezultatai geresni negu MongoDB. Abiejose NoSQL sistemose laiko skirtumas tarp testavimo atvejų – didėjant duomenų kiekiui yra mažas: MongoDB svyruoja nuo 0,13 ms. (1 testavimo atvejis) iki 0,27 (4 testavimo atvejis), o Cassandra nuo 0,08 ms. (1 testavimo atvejis) iki 0,15 (4 testavimo atvejis). MsSQL atliekant testus buvo lėtesnis nuo penkių kartų (1 testavimo atvejis), iki dešimties kartų (4 testavimo atvejis) lyginant su MongoDB, bet greitesnis už MySQL 38-43%.

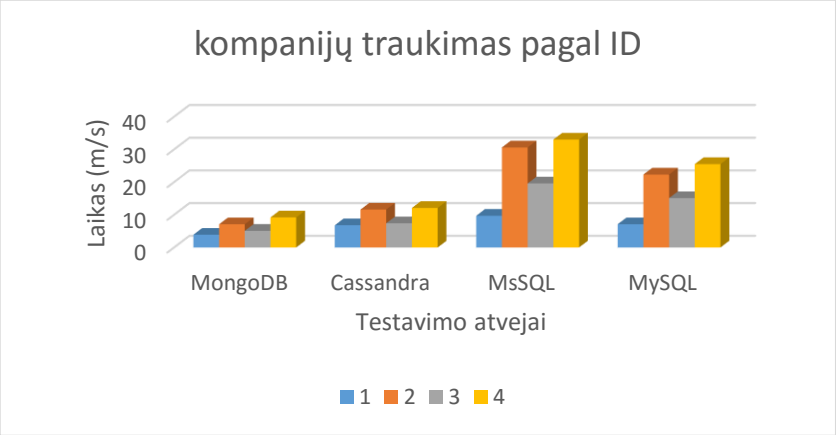
7 pav. diagramoje vaizduojami projekto atnaujinimo pagal projekto ID, testų rezultatai yra labai panašūs į Diagram3 diagramos. Lyginant tarpusavyje, tarp SQL sistemų, MySQL padarė atnaujinimo testus stabiliai greičiau negu MsSQL, o lyginant tarpuavyje NoSQL sistemas – Cassandra užduotis atliko greičiau negu Mongo DB. Kadangi atnaujinimai, kaip ir 6 pav. diagramoje vyko, pagal raktinę reikšmę, NoSQL DBVS pasirodė ženkliai geriau negu SQL, Cassandra rezultata isvyravo nuo 0,08 ms. (1 testavimo atvejyje) iki 0,15 ms. (4 testavimo atvejyje), o MongoDB nuo 0,13 (1 testavimo atvejyje) iki 0,27 (4 testavimo atvejyje). Taip yra todėl, kad NoSQL duomenų bazių valdymo sistemos turi iš anksto paruoštą (pre-build) indeksą pirminiam dokumento raktui, kuris yra greitesnis negu SQL serverio pirminio rakto klasterizuotas indeksas.



7 pav. Projektų atnaujinimas pagal ID

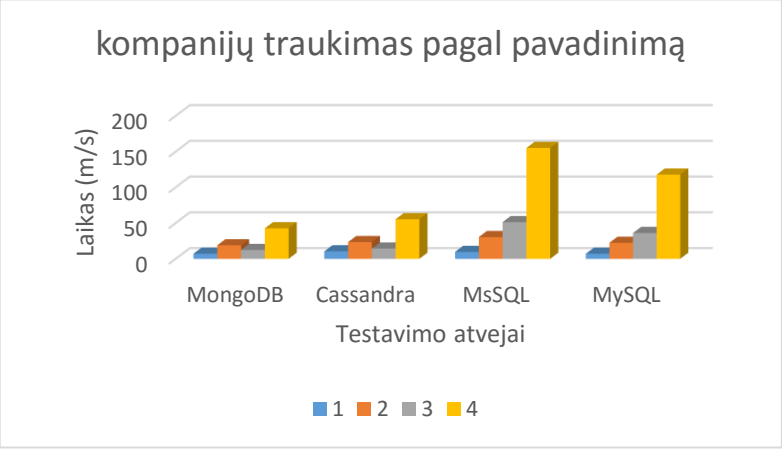
8 pav., 9 pav., 10 pav., 11pav., 12pav ir 13 pav. diagramos iliustruoja laiką kurio prireikė įvykdyti select užklausas: 8 pav., 9 pav., ir 10 pav. „paprastų" užklausų rezultatai, 11 pav., 12 pav. Ir 13. pav „sudėtingų" užklausų rezultatai. Paprastos užklausos ieškojo pagal lentelės(dokumento) priminį raktą arba vieną iš kitų reikšmių. Sudėtingos užklausos įtraukė reikšmes iš skirtingų lentelių. Šioms užklausoms įgyvendinti reikėjo naudoti grupavimą, objektų sąsajas tarpusavio santykiu M:N ir duomenų agregavimo funkcijas.

8 pav. diagramoje vaizduojami įmonės duomenų traukimo (select), pagal įmonės ID, testų rezultatai. Visuose keturiuose testavimo atvejuose MongoDB pasirodė stabiliai geriausiai. Cassandra pirmą testavimo atvejį įvykdė 43% lėčiau, antrą testavimo atvejį 39% lėčiau, trečią 31% lėčiau ir ketvirtą 24% lėčiau. Nors visus testavimo atvejus atliko lėčiau, galima pastebėti tendenciją, kad didėjant duomenų kiekiui ivykdymo laiko skirtumas keičiasi Cassandra DBVS naudai. Iš to galima daryti prielaidą, kad aplikacijos duomenų imtis buvo per maža, kad atsiskleistų Cassandra našumas. Vis dėlto, Cassandros rezultatai stabiliai geresni negu SQL sistemų: pirmamą testavimo atvejį atliko vidutiniškai 4.5% greičiau negu MYSQL ir 30% greičiau negu MsSQL, antrą testavimo atvejį atitinkamai 48% ir 62% greičiau, trečią testavimo atvejį 51% ir 63%, bei ketvirtą testavimo atvejį atliko 53% ir 64% greičiau. Nors duomenų kiekis didėjo nuo pirmo testavimo atvejo iki ketvirto testavimo atvejo, bet veikimo greitį įtakoja ir select'ų traukimų skaičius, kas atsispindi lyginant antrą testavimo atvejį ir trečią testavimo atvejį.



8 pav. Kompanijų traukimas pagal ID

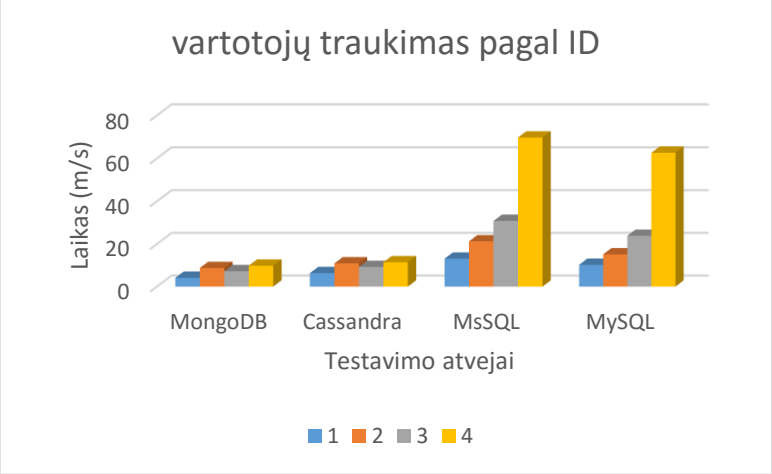
9 pav. diagrama iliustruoja laiką reikalingą ištraukti įmones pagal pavadinimą. Nors bendru atveju NoSQL rezultatai yra geresnis negu SQL duomenų bazių valdymo sistemų, negalima vienareikšmiškai daryti išvadų, nes pirmame testavimo atvejyje su mažu kiekiu duomenų MongoDB ir MySQL rezultatas buvo vienodas dešimtųjų milisekundžių tikslumu. Vis dėlto didėjant duomenų kiekiui, skirtumas aiškėja. Pirmą testavimo atvejį kone vienodu greičiu (7.1ms.) atliko MongoDB ir MySQL, Cassandra užklausas įvykdė 33% lėčiau, o MsSQL 27% lėčiau. Antrąjame testavimo atvejyje MySQL atsiliko nuo MongoDB 16%, Cassandra atsiliko 19%, o MsSQL vykdymo laikas užtruko 38% ilgiau. Trečiajame testavimo atvejyje NoSql sistemos pasirodė ženkliai geriau: MongoDB užtruko vidutiniškai 12,4ms., Cassandra 14,3 ms., kol tuo tarpu MySQL 36,1 ms., o MsSQL 51,3ms. Ketvirtame testavimo atvejyje rezultatai tarp Nosql ir SQL sistemų skyrėsi dar labiau: MongoDB vykdymo laikas buvo 3 kartus trumpesnis negu MySQL ir net 4 kartus trumpesnis negu MsSQL



9 pav. Kompanijų traukimas pagal pavadinimą

10 pav. diagramoje pavaizduotas vykdymo laikas užklausos kurioje yra traukiami vartotojai pagal vartotojų ID. 10 pav. diagramos rezultatai yra panašūs į 8 pav., kur buvo traukiamos kompanijos

pagal kompanijų ID. Dėl didesnio įrašų kiekio (vartotojų sistemoje yra daugiau negu kompanijų) skirtumas tarp SQL ir NoSQL yra dar didesnis. Visuose keturiuose testavimo atvejuose NoSQL sistemos buvo greitesnės. Taip pat galima pastebėti, jog nors Cassandra testavimo atvejus atliko lėčiau, kylant duomenų kiekiui rezultatas krypo šios sistemos naudai. Pirmą testavimo atvejį greičiausiai atliko MongoDB (per 4,1ms.), Cassandra vykdė 35% ilgiau, o MySQL ir MsSQL atitinkamai daugiau negu 2 ir 3 kartus ilgiau. Antrame testavimo atvejyje Cassandra užtruko 20% ilgiau už MongoDB, tačiau 49% negu MsSQL ir 38% greičiau negu MsSQL. Trečią testavimo atvejį Cassandra atliko 22% lėčiau negu MongoDB ir 70% greičiau negu MsSQL, bei 62% greičiau negu MySQL. Ketvirtame testavimo atvejyje abi NoSQL sistemos parodė panašius rezultatus ir įvykdė užklausas daugiau negu 6 kartus greičiau už SQL sistemas. Remiantis 8 pav. ir 10 pav. diagramų rezultatais, galima pastebėti, jog esant mažam duomenų kiekiui, releacinėje lentelėje individualių atributų paieška atima panašiai laiko kaip ir NoSQL sistemose atributų paieška kolekcijose. Vis dėlto, kai duomenų kiekis yra didelis (testavimo atvejis 4), SQL vykdymo laikas yra didesnis negu NoSQL.

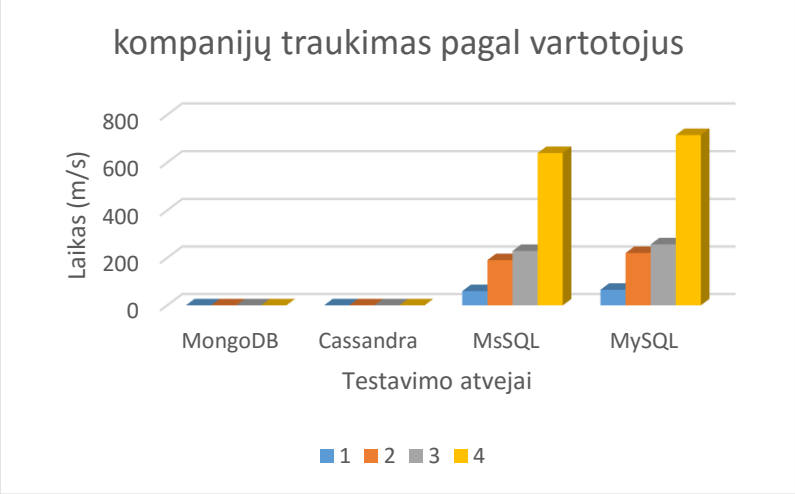


10 pav. Vartotojų truakimas pagal ID

11 pav. diagrama vaizduoja kompanijų traukimo pagal vartotojus rezultatus. NoSQL sistemų rezultatai yra ženkliai geresni negu SQL sistemų. Lyginant tarpusavyje, abi SQL sistemų rezultatai labai panašius ir atitinkamai NoSQL sistemų rezultatai taip pat tarpusavyje panašius. Abi NoSQL sistemos atliko visus keturis testavimo atvejus taip greit, kad diagramoje negalima įžvelgti duomenų. Laiko skirtumas vykdant trumpiausią testavimo atvejį (pirmą) ir ilgiausiai trukusį (ketvirtą), abiejų sistemų atveju nesiekia 0,2 ms. Pirmą testavimo atvejį MongoDB atliko per 0,17 ms., o Cassandra per 0,26. Antrą testavimo atvejį MongoDB atliko per 0,19 ms., o Cassandra per 0,26. Trečią testavimo

atvejį MongoDB atliko per 0,23 ms., Cassandra užtruko 0,28 ms. Paskutinį, patį sunkiausią testavimo atvejį MongoDB įvykdė, per 0,31 ms., o Cassandra 0,34 ms. Kylant duomenų kiekiui NoSQL sistemų vykdymo laikas kito labai mažai.

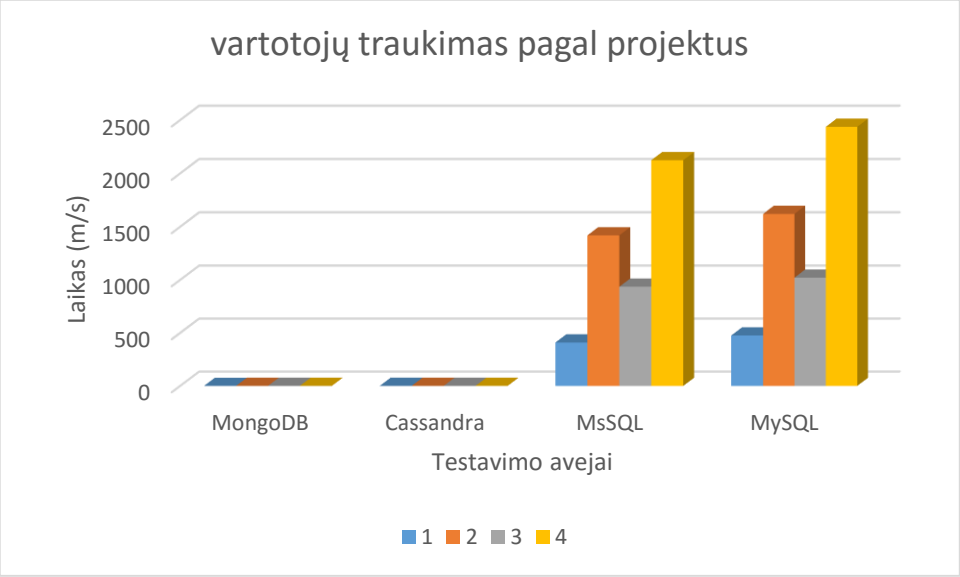
MsSQL visuose keturiuose testavimo atvejuose pasirodė 10% geriau negu MongoDB, tačiau lyginant su betkuria iš NoSQL sistemų, rezultatai blogesni šimtais kartų. Mažiausias skirtumas yra pirmame testavimo atvejyje, nes ten yra mažai duomenų, ten MsSQL užtruko 59,9 ms., o MySQL 65,5 ms. Didžiausias skirtumas ketvirtame testavimo atvejyje, kur MsSQL rezultatai viršyja 600, o MySQL net daugiau negu 700 ms.



11 pav. Kompanijų traukimas pagal vartotojus

12 pav. diagramoje yra vaizduojami Vartotojų traukimo pagal projektus rezultatai. Rezultatai yra labai panašūs į 11 pav. diagramos rezultatus: NoSQL rezultatai buvo daug geresni visuose keturiuose testavimo atvejuose lyginant su SQL sistemomis. Lyginant tarpusavyje NoSQL sistemas, rezultatai tarp MongoDB ir Cassandra panašūs, ilgiausiai trukęs vykdymas (4 testavimo atvejis) nesiekė 1 ms. Pirmame testavimo atvejyje MongoDB rezultatas vidutiniškai buvo 0,33 ms. , o Cassandra 0,57ms. Antrame testavimo atvejyje užtruko atitinkamai 0,41 ms. Ir 0,71, trečiame testavimo atvejyje 0,49ms. ir 0,79. Ketvirtame testavimo atvejyje skirtumas sumažėjo: MongoDB vidutiniškai užtruko 0,62ms. , o Cassandra 0,91 ms.

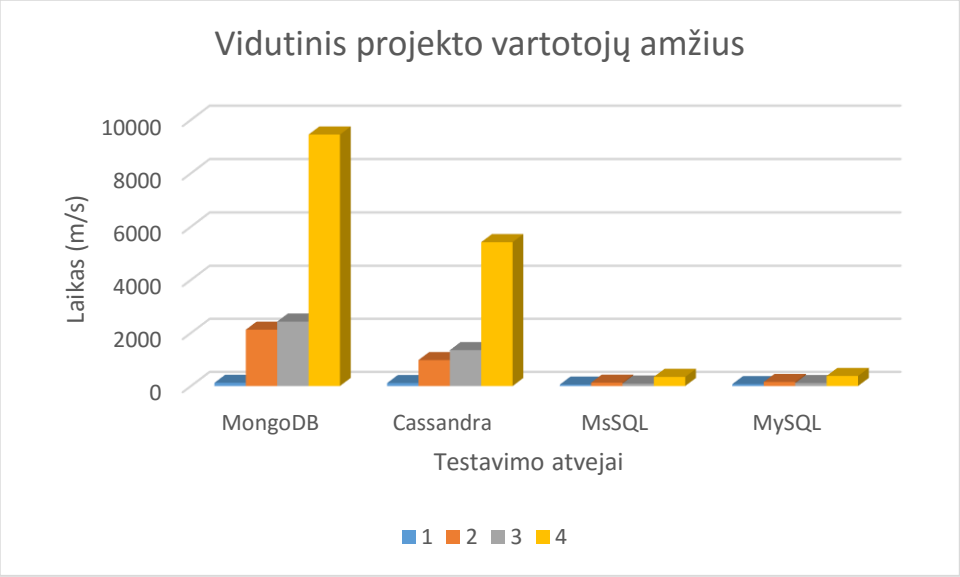
SQL sistemų rezultatai panašūs kaip ir 11 pav. diagramoje. MsSQL visuose keturiuose testavimo atvejuose vidutiniškai užtruko 12% trumpiau negu MySQL. Pirmąjį testavimo atvejį įvykdė per 410ms, kol tuo tarpu MySQL per 476ms. Antrą testavimo atvejį, MsSQL įvykdė per 1419,5 ms., o MySQL per 1620ms. Trečiąjį testavimo atvejį MsSQL įvykdė per 936 ms., o MySQL per 1021. Paskutiniame testavimo atvejyje, su daug duomenų, rezultatai stipriai šoktelėjo į viršų: MsSQL užtruko 2124 ms., MySQL 2437 ms.



12 pav. Vartotojų traukimaspagal projektus

13 pav. diagramoje yra vaizduojama vidutinio projekto vartotojų amžiaus duomenų traukimo ir agregavimo rezultatai. Abi SQL sistemos parodė stabiliai geresnius rezultatus nei NoSQL. Lyginant SQL sistemas tarpusavyje MsSQL rezultatai 20% geresni negu MySQL. Pirmame testavimo atvejyje MsSQL sugaišo 52,1 ms., kol tuo tarpu MySQL 72,2 ms. Antrame testavimo atvejyje MsSQL užtruko 125,3 ms, o MySQL 152,3, trečiame testavimo atvejyje atitinkamai: 92,6 ms. ir 112 ms., ketvirtajame atvejyje MsSQL užtruko 346 ms., o MySQL 380,2 ms.

NoSQL sistemos dažnu atveju neturi daug „tikrų" duomenų agregavimo mechanizmų kuriuos turi SQL. Vis dėlto, abi testuojamos sistemos turi reikiamą funkciją implementuotą ir teikia kaip bazinį funkcionalumą vartotojui. Agregavimo funkcija „average", kaip ir kitos agregavimo funkcijos yra sukurtos per „MapReduce" metodą ir nėra tokios efektyvios kaip atitinkami SQL metodai. Lyginant tarpusavyje, Cassandra rezultatai šioje imtyje buvo geresni: pirmame testavimo atvejyje rezultatai buvo panašūs ir skyrėsi dvigubai nuo SQL sistemų. Antrame testavimo atvejyje Cassandra pasirodė 45% geriau negu MongoDB, tačiau 8 kartus lėčiau už SQL sistemas. Trečią testavimo atvejį Cassandra įvykdė 44% grei2iau nei MongoDB ir daugiau negu 10 kartų lėčiau už SQL sistemas. Paskutiniame testavimo atvejyje abiejų NoSQL sistemų rezultatai ženkliai šokteli į viršų: MongoDB rezultatai 43% didesni negu Cassandra ir apie 30 kartų didesni nei SQL sistemų



13 pav. Vidutinio projekto vartotojų amžiaus ištraukimas

Rezultatai ir Išvados

1. Buvo atlikta dabartinių SQL ir NoSQL DBVS analizė, bei palyginti sistemų duomenų modeliai, nuoseklumo ir saugojimo mechanizmai.
2. Populiarių SQL ir NoSQL sistemų palyginimas parodė, kad:
   - 2.1. vykdant įrašymą, atnaujinimus ir paprastas užklausas pagal neraktines reikšmes, visų keturių sistemų rezultatai su maža duomenų apimtimi yra panašūs, tačiau didėjant duomenų kiekiui, MongoDB ir Cassandra sistemų vykdymo laikas auga greičiau negu MySQL ir MsSQL.
   - 2.2. vykdant įrašymą, atnaujinimus ir paprastas užklausas pagal raktines reikšmes, MongoDB ir Cassandra pasirodo geriau visais atvejais. Didejant duomenų kiekiui, MySQL ir MsSQL sistemų vykdymo laikas auga sparčiau.
   - 2.3. Vykdant sudėtingas užklausas, be agregavimo funkcijų, MongoDB ir Cassandra sistemos pasirodo geriau visais atvejais. Didejant duomenų kiekiui, MySQL ir MsSQL sistemų vykdymo laikas auga sparčiau.
   - 2.4. Vykdant sudėtingas užklausas su agregavimo funkcijomis, MySQL ir MsSQL sistemos parodo geresnius vykdymo rezultatus. Didėjant duomenų kiekiui MongoDB ir Cassandra sistemų vykdymo laikas auga sparčiau.

3. NoSQL ir SQL sistemos pasižymi greitu veikimu skirtingomis sąlygomis.

   NoSQL savybės:

   3.1. Geresni rezultatai dirbant su dideliais duomenų kiekiais, atliekant paprastas užklausas, kurios naudoja raktines reikšmes,
   3.2. Sudėtingo tipo užklausose be agregavimo, kur naudojamos raktinės reikšmės, greičiau įvykdo duomenų traukimą, atnaujinimą ir kopijavimą
   3.3. Geresni rezultatai dirbant su dinaminiais laukais, bei kitais dinaminiais schemos objektais

   SQL savybės:

   3.4. Geresni rezultatai dirbant su agregavimo funkcijomis
   3.5. Geresni rezultatai dirbant su mažais kiekiais duomenų, kur nenaudojamos raktinės reikšmės

Šaltiniai

[AB13] - V. Abramova, J. Bernardino „NoSQL databases: MongoDB vs cassandra” (2013) p.14-22

[AJ15] - A. Anderson, S. D. Johnson: „Bigtable: A Distributed Storage System for Structured Data” (2015)

[BCM+14] - R. Brown, S. Cribbs, C. Meiklejohn, S. Elliott „Riak DT map: a composable, convergent replicated dictionary” (2014)

[CS10] - R. Cattell, M.Shiler „Scalable SQL and NoSQL data stores” (2010)

[DPM14] - A. Dalonne, M.Parker, R. Mongomer „BigTable storage – big data concurency solutions” (2014)

[JPY+13] - T. Jörg, R. Parvizi, H. Yong, S. Dessloch „Hypertable: a distributed storage system for structured data” (2013)

[KCM+14] - W. Kim, I. Cho, D. Min, C. Sung „Design of data backup on distributed memory system based on key-value store using hot/cold data management (2014)

[LSR14] - T. Lamekova, S. Shusaper, S.Rekliott „Scalable relational data bases” (2014) p. 18-29

[MKM12] - Y. Mao, E. Kohler, R. T. Morris „Cache craftiness for fast multicore key-value storage” (2012) p.183-196

[MT13]- J. Marter, J.Thomagan „Extensible record data stores” (2015) p. 16-18

[PPV13] - Z. Parker, S. Poe, S. V. Vrbsky „Comparing NoSQL MongoDB to an SQL DB (2013)

[SSY+11] - T. Sliwkanich, D. Schneider, A. Yong, M. Home, D. Barbosa „management for document stores” (2011)

[SV13] - A. Al-Shishtawy, V. Vlassov „autonomic elasticity manager for cloud-based key-value stores” (2013) p. 115-116

[WZS+15] - L. Wang, S. Zhang, J. Shi, L. Jiao, O. Hassanzadeh, C. Wangz „Schema management for document stores” (2015)

[ZNH+15] - K. Zhenshenby, S. Neverbicky, M. Harbert „NoSQL big data – mapReduce the solution for aggregations .” (2015)

Sąvokų apibrėžimai:

Podėlis[1] - ang. Cache

Skaliuojamumas[2] – ang. scalability

Išplėsto įrašo saugyklos[3] - ang. Extensible Record Stores

Šardinimas[4] – (ang. Sharding) horizontalus duomenų paskirstymas duomenų bazėje arba paieškos variklyje

Bucket - duomenų tipas, „hashing" algoritme, kuris grupuoja objektus.

REST[5] – (ang.:Representational state Transfer) kliento su suerveriu komunikavimo protokolas

Backup[6] - atsarginis duomenų grąžinimas

Nested - The nested type is a specialised version of the object datatype that allows arrays of objects to be indexed and queried independently of each other

Tankinimas – ang. compaction

RDBMS - releacinė duomenų bazių valdymo sistema

ACID transanctions – duomenų bazių tranzakcijų rinkinys.

Tuple[7] – nesurikiuotas rinkinys žinomų reikšmių su vardais (labai panašu į duomenų bazės eilutę).

Deadlock - Tranzakcijų vykdymo užsiblokavimas, kuris įvyksta tuo atveju,kai dvi arba daugiau tranzakcijos bando imti tą pačią duomenų bazės dalį.

Testavimo atvejis – ang. Test Case