Henri Haverinen

MySQL- ja MongoDB-tietokantojen suorituskykyvertailu

Tietotekniikan pro gradu -tutkielma

15. huhtikuuta 2018

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Henri Haverinen

Yhteystiedot: +358504366096, henri.s.haverinen@student.jyu.fi

Ohjaaja: Antti-Juhani Kaijanaho

Työn nimi: MySQL- ja MongoDB-tietokantojen suorituskykyvertailu

Title in English: Performance comparison between MySQL and MongoDB databases

Työ: Pro gradu -tutkielma

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Sivumäärä: 53+4

Tiivistelmä: Datan määrä ja rakenne on muuttunut vuosikymmenien saatossa huimasti, ja tiedonkäsittely kokee uusia haasteita jatkuvasti. Perinteiset relaatiotietokannat eivät välttämättä enää sovellu ratkaisemaan näitä ongelmia. 2000-luvun puolella vaihtoehdoksi ovat tulleet NoSQL-tietokannat, joiden tavoitteena on tarjota ratkaisukeinoja näihin uusiin haasteisiin. Tässä tutkielmassa käsitellään relaatio- ja NoSQL-tietokantojen taustoja ja eroavaisuuksia. Erityisesti tutkielmassa keskitytään tietokantojen suorituskykyyn. Tutkielmassa mitattiin ja vertailtiin MySQL ja MongoDB tietokantojen suorituskykyä, jotka edustavat vastaavasti relaatio- ja NoSQL-tietokantoja. Suorituskykytesteihin käytettiin avoimen lähdekoodin Yahoo! Cloud Serving Benchmark -työkalua. MongoDB suoriutui suorituskykytesteistä paremmin kuin MySQL-tietokanta, mutta tuloksiin vaikuttavat monet tekijät, eikä vastaus ole kokonaisuudessaan näin yksinkertainen.

Avainsanat: Tietokanta, SQL, NoSQL, MySQL, MongoDB, suorituskyky, YCSB

Abstract: The amount of data and it's structure has changed dramatically in the past years, and thus data processing faces new challenges constantly. Traditional relational databases might not be a valid solution for these problems anymore. In the 21st century NoSQL databases have emerged as an alternative solution. This master thesis covers the basics and differences between relational and NoSQL databases. Especially performance is taken into account. In this thesis performance of two different databases, MySQL which represents tra-

ditional relational database and MongoDB which represents NoSQL database, were tested and compared. Open source tool Yahoo! Cloud Serving Benchmark was used to carry out these tests. MongoDB got better results compared to MySQL database, but it is important to remember that many elements affects to the results and thus the validity of the results is not that straightforward.

Keywords: Database, SQL, NoSQL, MySQL, MongoDB, performance, YCSB

Termiluettelo

ACID	Atomicity, Consistency, Isolation, Durability. Joukko tietokantatransaktion ominaisuuksia, jotka takaavat tietokannan eheänä pysymisen.
BASE	Basically Available, Soft state, Eventual Consistency. Joukko ominaisuuksia, jotka pyrkivät takaamaan korkean tiedon saatavuuden hetkellisellä tietokannan eheyden rikkomisella.
BigData	Tarkoittaa suuria tietomääriä, joita ei voi perinteisillä tiedon prosessointikenoilla käsitellä.
CRUD	Create, Read, Update, Delete. Tietokannan perusoperaatiot: luo, lue, päivitä, poista.
Deadlock	Tilanne, jossa kaksi tietokannan transaktiota pyytävät lukkoja ristiin siten, että molemmat transaktiot jumiutuvat.
Dokumenttitietokanta	NoSQL-tietokannan tyyppi. Data tallennetaan tietokantaan dokumentteina.
Hajautettu järjestelmä	Järjestelmä, joka koostuu useasta itsenäisestä laitteesta tai ohjelmistosta, jotka toimivat ja kommunikoivat keskenään muodostaen järjestelmäkokonaisuuden.
Latenssi	(eng. Latency) Syötteen antamisen ja vasteen saamisen välissä kulunut aika.
MongoDB	Suosittu avoimen lähdekoodin dokumenttipohjainen NoSQL-tietokanta.
MySQL	Suosittu avoimen lähdekoodin relaatiotietokanta.
NoSQL	No-SQL tai Not-only-SQL. Viittaa yleensä tietokantoihin, jotka eivät pohjaudu perinteiseen relaatiomalliin. Termi on kuitenkin melko väljä.
OLTP	(Online) Transaction Processing. Yleisnimitys transaktio-orientoituneille järjestelmille, joilla usein miten tarkoitetaan järjestelmiä, jotka hakevat ja prosessoivat dataa tietokannasta.
Out-of-the-box	Tuote, johon ei ole tehty mitään ylimääräisiä konfiguraatioita

	tai räätälöintejä.
Relaatiotietokanta	Tietokanta, joka pohjautuu relaatiomalliin. Arkikielessä myös SQL-tietokanta.
Skeema	Kuvaa tietokannan rakennetta.
Solmu	(eng. Node) Hajautetun järjestelmän yksittäien laite tai ohjelmisto.
Suoritusteho	(eng. Throughput) Tietokannan suoritusteho, mitataan usein transaktiona / sekunti.
SQL	Structured Query Language. Standardoitu kyselykieli relaatiotietokantojen datan hallintaan.
TPC	Transaction Processing Performance Council. Voittoa tavoittelematon järjestö, joka pyrkii kehittämään ja standardoimaan suorituskykytestejä.
Transaktio	Toimenpide, joka voi koostua useasta yksittäisestä tapahtumasta.
Web 2.0	Viittaa aikakauteen jossa yhä suurempi osa WWW-sivuilla olevasta materiaalista on loppukäyttäjien tuottamaa (esim. Facebook).
YCSB	Yahoo! Cloud Serving Benchmark. Avoimen lähdekoodin työkalu, joka on suunniteltu tietokantojen suorituskyvyn mittaamiseen.

Kuviot

Kuvio 1. Esimerkki relaatiomallista 6
Kuvio 2. Esimerkki dokumenttitietokannasta 12
Kuvio 3. Esimerkki avain-arvo -tietokannasta 12
Kuvio 4. Esimerkki saraketietokannasta 12
Kuvio 5. Esimerkki verkkotietokannasta 12
Kuvio 6. Esimerkki keskiarvon ja persentiilin eroista 14
Kuvio 7. Esimerkki suoritustehon ja latenssin suhteesta 15
Kuvio 8. Esimerkki YCSB:n generoimasta datasta JSON-muotoisena 21
Kuvio 9. Suorituskykytesti 1 - pieni tietokanta 25
Kuvio 10. Suorituskykytesti 1 - iso tietokanta 26
Kuvio 11. Suorituskykytesti 2 - pieni tietokanta 26
Kuvio 12. Suorituskykytesti 2 - iso tietokanta 27
Kuvio 13. Suorituskykytesti 3 - pieni tietokanta 28
Kuvio 14. Suorituskykytesti 3 - iso tietokanta 28
Kuvio 15. Suorituskykytesti 4 - pieni tietokanta 29
Kuvio 16. Suorituskykytesti 4 - iso tietokanta 29
Kuvio 17. MySQL yhdistety tulokset 30
Kuvio 18. MongoDB yhdistetyt tulokset 31

Taulukot

Taulukko 1. SQL- ja relaatiotietokantojen termien väliset yhteydet 5
Taulukko 2. Testilaitteiston tiedot 20
Taulukko 3. Testilaitteen ohjelmistojen tiedot 20
Taulukko 4. Suorituskykytestien luku- ja kirjoitusoperaatioiden suhteet 23
Taulukko 5. Suorituskykytesti 1 tulokset (suoritusteho, operaatioita / sekunti) 47
Taulukko 6. Suorituskykytesti 2 tulokset (suoritusteho, operaatioita / sekunti) 48
Taulukko 7. Suorituskykytesti 3 tulokset (suoritusteho, operaatioita / sekunti) 49
Taulukko 8. Suorituskykytesti 4 tulokset (suoritusteho, operaatioita / sekunti) 50

Sisältö

1 JOHDANTO 1

2 RELAATIOTIETOKANNAT 3
2.1 Relaatiomalli ja käsitteet 3
2.2 Indeksointi 5
2.3 ACID 5

3 NOSQL-TIETOKANNAT 8
3.1 Peruskäsitteet 9
3.2 BASE 10
3.3 Erilaisia NoSQL-tietokantatyyppejä 11

4 TIETOKANTOJEN SUORITUSKYKYJEN MITTAAMINEN 13
4.1 Mittarit 13
4.2 Työkalut 16

5 TUTKIMUSMENETELMÄ 18
5.1 Laitteiston ja ohjelmistojen tiedot 19
5.2 Testidata ja suorituskykytestit 21

6 TULOKSET 24
6.1 Suorituskykytesti 1 24
6.2 Suorituskykytesti 2 25
6.3 Suorituskykytesti 3 26
6.4 Suorituskykytesti 4 27
6.5 Yhdistetyt tulokset 29

7 TULOSTEN ANALYSOINTI 32
7.1 Testien ja työkalujen vaikutus 32
7.2 SQL ja NoSQL sekä datan vaikutus 33
7.3 Laitteiston ja ohjelmistojen vaikutus 34
7.4 Tulokset muissa tutkimuksissa 36
7.5 Tutkimuksen puutteet 37
7.6 Johtopäätökset ja pohdintaa 38

8 YHTEENVETO 40

LÄHTEET 42

LIITTEET 47
A Suorituskykytesti 1 tulokset 47
B Suorituskykytesti 2 tulokset 48
C Suorituskykytesti 3 tulokset 49
D Suorituskykytesti 4 tulokset 50

1 Johdanto

Olemme päivittäin tekemisissä erilaisten tietokantojen kanssa sen kummemmin sitä tiedostamatta. Henkilötietomme löytyvät väestörekisteristä, opiskelijoiden tiedot ovat yliopistojen taustajärjestelmissä ja katuosoitteemme löytyy varmasti monen eri palvelun takaa, kuten esimerkiksi verkkokaupoista tai postin järjestelmistä. Kaikkien järjestelmien takana tieto on tallennettu jonkinlaiseen tietokantaan. Tietokannat ovat käytännössä välttämättömyys, kun tietoa halutaan tallentaa johonkin järjestelmään.

Data lisääntyy maailmassa jatkuvasti. Etenkin Web 2.0 aikakauden myötä ihmiset pystyvät tuottamaan ja tuottavat myös huomaamattaan yhä enemmän dataa. Tätä voimakkaasti kasvavaa dataa kutsutaan usein termillä BigData. Yhä suuremman datamäärän lisäksi data on myös paljon monipuolisempaa ja monimutkaisempaa. Perinteiset (relaatio)tietokannat eivät välttämättä ole enää optimaalisin tai edes mahdollinen vaihtoehto tiedon säilömiseen ja käsittelyyn. Viimeisen kymmenen vuoden aikana markkinoille ovat tehneet tuloaan ei-relaationaaliset, eli NoSQL-tietokannat, joiden tarkoituksena on tarjota ratkaisuja edellämainittuihin haasteisiin.

NoSQL-tietokannat ovat suhteellisen uusia, vaikkakin markkinoilla on jo paljon pitkälle kehitettyjä NoSQL-tietokantoja. NoSQL-tietokantaratkaisuja kehitetään aktiivisesti eteenpäin perinteisiä relaatiotietokantoja unohtamatta - myös niitä kehitetään tarjoamaan ratkaisua näihin uusiin haasteisiin. Esimerkiksi Oracle on kehittämässä MySQL-tietokannalle tukea käyttää sitä dokumenttipohjaisena tietokantana (Oracle Corporation and/or its affiliates 2017). Oraclen tavoittena on yhdistää SQL-kielen tehokkuus ja käyttäjäystävällisyys, sekä NoSQL-tietokantojen tarjoamat uudet mahdollisuudet.

Oikeanlaisen tietokannan valitseminen on erittäin kriittinen tekijä järjestelmä- ja ohjelmistokokonaisuuksien sujuvan tominnan kannalta. Nykypäivänä valinnan tekeminen on vaikeaa, koska eräät tietokannat soveltuvat erilaisiin käyttötarkoituksiin paremmin kuin toiset. Vaikka useampi tietokanta soveltuisikin samaan käyttötarkoitukseen, voi tietokantojen välillä olla silti eroja esimerkiksi suorituskyvyssä. Lisäksi erilaisia tietokantatyyppejä on valtavia määriä, puhumattakaan eri valmistajien tietokantatoteuksista, jolloin parhaan valinnan tekeminen

ei ole itsestäänselvää.

Tutkielmani pohjautuu kandidaatintutkielmaani, joka on kirjallisuuskatsaus erilaisten tietokantojen suorituskykyyn (Haverinen 2016). Tämän tutkielman luvut 2 ja 3 käsittelevät SQL- ja NoSQL-tietokantoja, ja mukailevat pitkälti kandidaatintutkielmaani. Tarkoituksena on antaa lukijalle yleisnäkemys SQL- ja NoSQL-tietokannoista, niiden eroista, käsitteistä ja taustasta. Luku 4 käsittelee tietokantojen suorituskykyjen mittaamista ja pyrkii vastamaan kysymyksiin mitä ja miten suorituskykyä mitamaan, sekä kuinka mittaustuloksia tulkitaan. Luvussa 5 esitellään tutkielman tutkimusmenetelmä ja luvussa 6 on esitelty tutkimuksen tulokset. Luvussa 7 pohditaan tulosten luotettavuutta, käytettävyyttä ja yleistettävyyttä muihin tietokantoihin, ja luku 8 on yhteenveto koko tutkielmasta.

2 Relaatiotietokannat

Tietoa on tallennettu erilaisin tavoin reikäkorteista nykyaikaisiin tietokantoihin aivan tietotekniikan alkuajoista lähtien. Termi *tietokanta* esiteltiin ensimmäisen kerran 1960-luvun alkupuolella (Oxford English Dictionary 2017; Merriam-Webster 2017). Yksinkertaisimmillaan tietokannalla tarkoitetaan kokoelmaa dataa, joka on organisoitu jollakin tavalla. Tietokantojen toiminnan periaatteet ovat pysyneet samana, joskin tekniikkojen ja teknologioiden kehittyessä tietokantatoteutukset ovat hioutuneet yhä paremmiksi ja tehokkaammiksi.

Nykyaikaiset tietokannat alkoivat tulla käyttöön 1960-luvulla, kun Charles W. Bachman kehitti tietokantojen konseptia eteenpäin. Noihin aikoihin alettiin siirtyä pois ajattelumallista, jossa data on sidottu suoraan ohjelmistoon. Sen sijaan data haluttiin erottaa omaksi kokonaisuudeksi, jolloin useat eri ohjelmistot voivat käyttää tuota samaa datalähdettä, eli tietokantaa (Bachman 1973). Suosittuja tietokantatyyppejä olivat tuolloin verkko- ja hierarkiatietokannat.

2.1 Relaatiomalli ja käsitteet

Relaatiomallin kehitti Edgar F. Codd 1970-luvun alussa, ja se on edelleen perusta nykypäivän relaatiotietokannoille (Codd 1970). Relaatiomallissa tieto esitetään ilman tietokoneiden vaatimia teknisiä rakenteita ja sitä on helppo hallita. Relaatiomalli poistaa turhan toistuvuuden ja takaa tiedon ristiriidattomuuden.

Muutama vuosi relaatiomallin julkaisemisen jälkeen Chamberlin ja Boyce (1974) julkaisivat IBM:n alaisuudessa kehittämänsä SEQUEL-kyselykielen, joka nykyisemmin tunnetaan SQL-termillä. Se tarjosi tavan hallita relaatiotietokannassa olevaa dataa. SQL-kyselykieli suunniteltiin alunperin IBM:n kehittämää *System R* -nimistä tietokantaa varten, mutta SQL-kyselykielen todettiin olevan laajemmin hyödynnettävissä. Relaatiotietokannat ovat olleet käytetyimpiä tietokantatyyppejä SQL-kyselykielen lanseerauksen jälkeen.

Relaatiomallissa tieto esitetään eri *relaatioihin*[1] kuuluvina *tietueina*[2], jotka voivat sisältää useita *kenttiä*[3]. Kenttä on tietueen nimetty elementti, esimerkiksi *etunimi* tai *sukunimi*. Relaatio on määritelty joukkona tietueita, joilla on samat kentät. Relaatioita voidaan linkittää toisiinsa tietueiden kenttien perusteella. Kuva 1 havainnollistaa relaatiomallia sekä relaatioiden linkitystä toisiinsa, ja seuraavassa kappaleessa on esimerkin kautta esitettynä relaatiomallin toimintaa. SQL-tietokannoille on syntynyt relaatiomallia vastaavat termit, joita käytetään yleisesti arkikielessä. Taulukossa 1 on kuvattu näiden termien väliset yhteydet.

Esimerkki relaatiomallista: meillä on kaksi relaatiota, *Ihminen* ja *Asunto*. *Ihminen-relaatio* sisältää joukon tietueita, eli eri ihmisiä. Jokainen ihminen on uniikki, mutta jokaisella ihmisella on samat tiedot, eli kentät. Kenttiä tässä esimerkissä voi olla etunimi, sukunimi, sukupuoli ja ikä. *Asunto-relaatio* sisältää vastaavasti joukon eri asuntoja (tietueita), joiden tiedoissa (kentissä) on tieto osoitteesta ja asunnon tyypistä (kerrostalo, omakotitalo jne.). Nämä kaksi relaatiota voidaan linkittää toisiinsa, jolloin jokainen ihminen asuu jossain asunnossa. Linkitys tapahtuu usein uniikilla yksilöivällä tunnisteella.

Hyvin suunniteltu tietokantarakenne on itsestään selittyvä, ja siinä olevaa dataa on helppo ymmärtää ja käsitellä, mikä onkin relaatiotietokantojen tarkoitus. Tietokantarakenteen voi suunnitella myös huonosti, jolloin vastaavanlaisia hyötyjä ei saavuteta. *Normalisointi* on tekniikka, jonka tarkoituksena on poistaa tietokantarakenteista samojen tietojen turha toistuvuus sekä parantaa datan eheyttä (Codd 1970).

Vaikka arkikielessä puhutaan relaatiotietokannoista, ei todellisuudessa markkinoilla olevista tietokannoista juuri yksikään toteuta täysin Coddin (matemaattista) relaatiomallia. Monet tietokannat sallivat esimerkiksi duplikaatti rivejä saman taulun sisällä, vaikka matemaattinen relaatiomalli ei tällaista salli. SQL-tietokannat, jotka ovat arkipäivän synonyymi relaatiotietokannoille, pohjautuvat kyllä Coddin relaatiomalliin, mutta poikkeavat myös siitä monissa yksityiskohdissa (Codd 1990).

1. eng. relation
2. eng. tuple/record
3. eng. field/attribute

SQL-termi	Relaatiotietokantatermi	Kuvaus
Rivi	Tietue	Kuvaa yhtä tietuetta
Sarake	Kenttä/Attribuutti	Tietueen sisältämä elementti
Pöytä / taulu	Relaatio	Sisältää joukon tietueita

Taulukko 1. SQL- ja relaatiotietokantojen termien väliset yhteydet

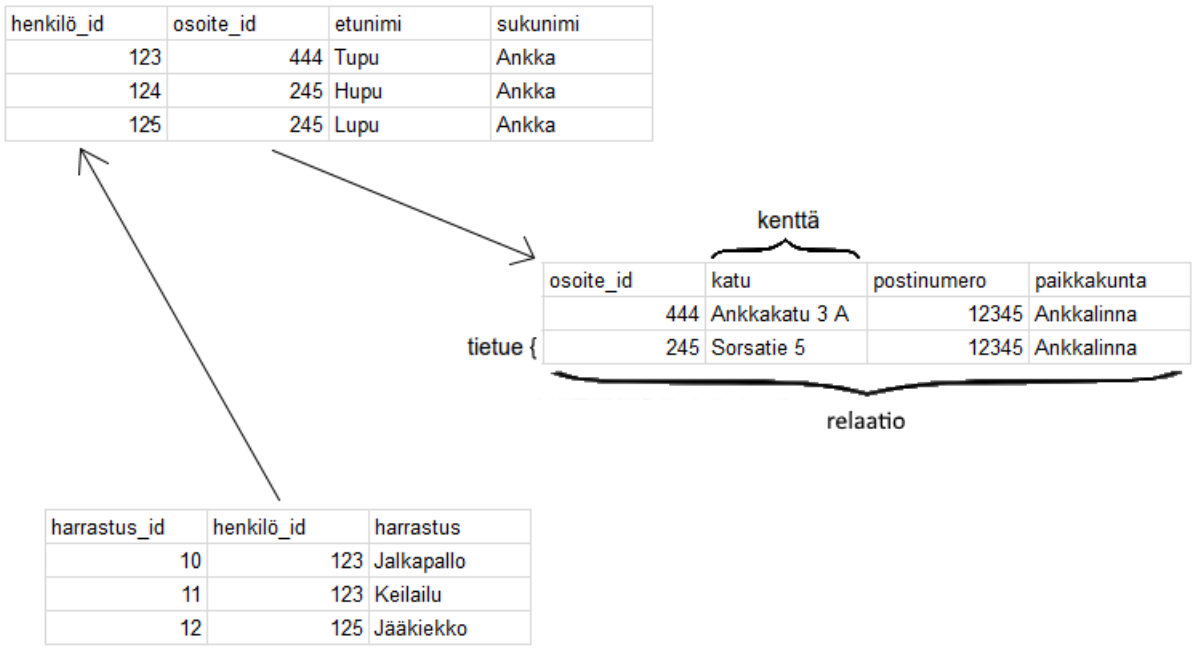
2.2 Indeksointi

Tietokantatermeissä indeksi ei tarkoita vain numeroa, vaan se on datarakenne, jonka tarkoitus on nopeuttaa tiedon hakemista tietokannasta. Indeksointi ei suoranaisesti liity relaatiomalliin, mutta on oleellinen osa moderneja tietokantoja. Indeksit ladataan yleensä suoraan tietokoneen muistiin, mutta tallennus levylle on myös mahdollista. Yleinen lähestymistapa onkin toteuttaa näistä molemmat, koska tällöin indeksin koko voi olla suurempi kuin saatavilla oleva muisti. Tässä tapauksessa muistiin ladataan aina vain tarpeen mukaan indeksejä. Levylle tallentaminen mahdollistaa indeksien säilymisen tietokannan sammumisen aikana, esimerkiksi virtakatkon takia.

Indeksiä voi karkeasti ajatella tietokannan tauluna, josta tiedon hakeminen on nopeampaa. Jokainen indeksin rivi koostuu yhdestä avaimesta, eli uniikista tunnisteesta, sekä yhdestä osoittimesta, joka osoittaa tietokannan alkuperäiseen riviin, josta indeksin rivi luotiin. Indeksien tekeminen ei ole aina järkevää, sillä tietokannan tiedon päivittyessä myös indeksin tietoja on päivitettävä. Uusia rivejä lisättäessä tietokantaan, tehdään niille myös uusi rivi indeksiin. Kun käsitellään suuria datamääriä, on tietokannan nopean toiminnan takaamiseksi käytettävä indeksejä. Silloinkin indeksejä on oltava mahdollisimman vähän, ja näiden indeksien tulee olla kovassa käytössä (O'Neil 1994, luku 7.1). Indeksejä tulee käyttää vasta silloin, kun tarve on riittävän kova.

2.3 ACID

1980-luvun alussa Gray (1981) määritteli transaktioille kolme ominaisuutta: *atomisuus*, *pysyvyys* ja *eheys*. Haerder ja Reuter (1983) kehittivät konseptia edelleen ja lisäsivät omi-



Kuvio 1. Esimerkki relaatiomallista

naisuuksiin myös *eristyneisyyden*. ACID-lyhenne syntyykin näiden neljän ominaisuuden englanninkielisistä sanoista: atomicity, consistency, isolation ja durability. Kyseisten ominaisuuksien tarkoitus on taata tietokannan eheys, myös virhetilanteissa. Haerder ja Reuter (1983) määrittelevät ominaisuudet seuraavasti.

- Atomisuus: transaktio suoritetaan kokonaan tai ei ollenkaan. Vaikka osa transaktion kommitoinneista onnistuisikin, onnistuneita kommitointeja ei säilytetä tietokannassa, mikäli transaktiossa tapahtuu yksikin virhe.
- Eheys: jokaisen transaktion jälkeen tietokannan on edelleen oltava eheä. Siten transaktio kommitoi vain oikeellisia, eli valideja, muutoksia.
- Eristyneisyys: transaktiot suoritetaan aina yksittäin erillään muista transaktioista. Toisin sanoen samaa tietoa käsittelevät yhtäaikaiset transaktiot johtavat samaan lopputulokseen kuin, että transaktiot suoritettaisiin peräkkäin.
- Pysyvyys: kun transaktio on suoritettu loppuun, sen kommitoima tieto myös pysyy tietokannassa. Esimerkiksi tietokanta-alustan kaatuminen tai virheet eivät saa vaikuttaa tiedon palautumiseen.

ACID-ominaisuuksien johdosta relaatiotietokannat soveltuvatkin hyvin sovelluksiin, joissa datan eheydestä ei voida tinkiä. Hyviä esimerkkejä ovat verkkopankit. Verkkopankkien omistajien on oltava varmoja, että asiakkaan siirtäessä rahaa toisen asiakkaan tilille, transaktio on suoritettu vasta, kun rahat ovat oikeasti siirtyneet vastaanottajan tilille. Ilman ACID-transaktioita olisi mahdollista, että transaktion epäonnistuessa rahat jäisivät puolitiehen: siirtäjän tililtä raha olisi lähtenyt, mutta ei olisi päätynyt vastaanottajan tilille.

3 NoSQL-tietokannat

NoSQL-tietokannat ovat olleet käytössä relaatiotietokantoja pidempään, esimerkiksi luvussa 2 mainitut verkko- ja hierarkiatietokannat on kehitetty ennen relaatiotietokantoja. Alunperin tallennettava data ja sen käsittelyn tarpeet olivat hyvin erilaiset verrattuna nykypäivään. Relaatiotietokannat olivat paras vaihtoehto vastaamaan tarpeisiin tuohon aikaan, ja relaatiotietokannat syrjäyttivätkin nopeasti muut tietokantatyypit markkinoilta. Viime aikoina vanhoja tietokantatyyppejä on kuitenkin aloitettu uudelleenkäyttämään, mutta myös uuden tyyppisiä NoSQL-tietokantoja kehitetään aktiivisesti. Etenkin suuret organisaatiot kuten Amazon ja Google, jotka käyttävät paljon rakennettoman datan kanssa, ovat aktiivisesti kehittäneet NoSQL-tietokantoja (Leavitt 2010).

NoSQL-termi voidaan kääntää ei-SQL, joka tarkoittaa ei-relaationaalista tietokantaa. Useissa NoSQL-tietokannoissa on kuitenkin relaatioita tallennettujen tietojen välillä, jolloin Not-only-SQL (ei vain SQL) onkin kenties kuvaavampi termi. NoSQL-termille ei ole tarkkaa määritelmää, mutta mm. NoSQL database (2017) -sivusto kuvailee NoSQL-tietokantoja mm. horisontaalisesti skaalautuviksi, skeemattomiksi ja hajautetuiksi. NoSQL-termi tuli vakiintuneeseen käyttöön vasta 2000-luvun loppupuolella, kun yhä useammalle yritykselle tuli tarve tehokkaampaan ja helpompaan suurten datamäärien käsittelyyn, eikä perinteiset relaatiotietokannat olleet enää järkevä ratkaisu.

Koska NoSQL-tietokannat ovat tulleet esille vasta 2000-luvun alussa, ei aiheesta juuri löydy oppikirjoja tai muita alan perusteoksia. Toisaalta taas aiheesta löytyy paljon tietoa eri WWW-sivuilta blogien, wikien, dokumentaatioiden ynnä muiden muodossa. Aiheesta löytyy myös tieteellisiä julkaisuja, joskin ne ovat usein painottuneet tiettyyn osa-alueeseen. Teknologia ja NoSQL-tietokannat kehittyvät huimaa vauhtia ja siten niihin liittyvä tieto vanhenee nopeasti. Eräs hyvä yleiskatsaus NoSQL-tietokantoihin ja niiden konseptiin on Strauch, Sites ja Kriha (2011) kirjoittama materiaali, vaikka osa sen sisällöstä alkaa olla jo vanhentunutta.

3.1 Peruskäsitteet

Korkealla tasolla NoSQL-tietokannat ovat tietokantoja siinä missä perinteiset relaatiotietokannatkin: ne ovat tiedon tallentamiseen ja hallintaan tarkoitettuja järjestelmiä. NoSQL-tietokannat eroavat relaatiotietokannoista monin tavoin, kuten rakenteeltaan ja käyttökohteiltaan.

NoSQL-tietokannat ovat usein skeemattomia, eli tietokannalle ei tarvitse kuvata datarakennetta etukäteen. Tämän ansiosta tietoa voidaan tallentaa tietokantaan ilman tarkempia määrityksiä, ja se tekee tietokannasta helposti mukautuvan datan rakenteen muuttuessa. Tallennettava data voi olla siis paljon vaihtelevampaa verrattuna skeemallisen tietokannan käyttämään dataan.

Useat NoSQL-tietokannat käyttävät JSON-dataa tiedon tallennusmuotona. JSON-muotoisen datan käyttäminen sovelluksissa (erityisesti web-sovelluksissa) ja rajapinnoissa on tänä päivänä yhä yleisempää, mikä on myös yksi syy NoSQL-tietokantojen suosion kasvuun. Perinteiset relaatiotietokannat eivät juuri käytä JSON-dataa.

NoSQL-tietokannat ovat yleensä suunniteltu skaalautumaan horisontaalisesti. Horisontaalisella skaalautumisella tarkoitetaan samanlaisten laitteistojen ja ohjelmistojen lisäämistä resursseihin, kun taas vertikaalisessa skaalautumisessa yhden laitteiston tehoa parannetaan lisäämällä uusia tehokkaampia komponentteja (esim. keskusmuisti, prosessori). Rinnakkaisten solmujen (eng. node) lisääminen NoSQL-tietokantajärjestelmään on siis helppoa, millä ylletään parempaan suorituskykyyn. Horisontaalisella skaalautumisella ei käytännössä ole rajoja verrattuna vertikaaliseen skaalautumiseen.

NoSQL-tietokannoille ei ole olemassa standardia kyselykieltä. Relaatiotietokannoille löytyy tarkoitettu SQL-kyselykieli. Valmistajasta riippuen tietokannat tukevat erilaisia kyselykieliä tai eri ohjelmointikielille tarkoitettuja rajapintoja, joiden avulla tietokannan dataa voidaan hallita. Eräät tietokannat tukevat jopa SQL-kyselykieltä tai sen kaltaisia kyselykieliä datan hallintaan.

Relaatio- ja NoSQL-tietokantojen väliltä löytyy paljon muitakin eroavaisuuksia, mutta edellämainitut ovat eroista oleellisimpia. Toisaalta valmistajasta riippuen NoSQL-tietokannat

voivat tukea relaatiotietokantojen ominaisuuksia tai päinvastoin, ja siten edellä esitellyt erot eivät ole ns. ”kiveen kirjoitettuja”.

3.2 BASE

BASE on lyhenne englanninkielen sanoista *Basically available*, *Soft state* ja *Eventual consistency*. Näillä ominaisuuksilla kuvataan hajautetun tietokannan oikeellisuusmallia, joka yleensä korostaa parempaa saatavuutta tiedon oikeellisuuden sijaan (Pritchett 2008). Oikeellisuusmallilla tarkoitetaan kokoelmaa vaatimuksia, joita jokaisen transaktion on noudatettava (vrt. ACID ja BASE).

BASE-mallin mukaan hajautettu tietokanta on aina tavoitettavissa (Basically available). Yhden (tai useamman) solmun ollessa saavutettamattomissa, voivat muut toimivat solmut edelleen palvella asiakasta. Pehmeä tila (Soft state) tarkoittaa tilannetta, jossa uusin data ei ole vielä replikoitunut kaikille solmuille. Tällainen tilanne on mahdollinen, koska data on hajautettu useammalle tietokannan solmulle. Tietokannan solmuilla oleva data replikoituu kuitenkin jossain vaiheessa kaikille solmuille, eli tietokannan tila on lopulta oikeellinen (Eventual consistency) (Pritchett 2008; Vogels 2009).

BASE-mallin karkeana vastakohtana voidaan pitää relaatiotietokantojen ACID-mallia, jonka tarkoitus on taata tietokannan eheys jokaisessa tilanteessa. BASE-malli sallii tämän säännön rikkomisen paremman saatavuuden saavuttamiseksi. Eräät NoSQL-tietokannat tarjoavat myös ACID-transaktioille tuen, mutta tuolloin korkean saatavuuden hyöty häviää.

Eric Brewer esitteli vuonna 2000 CAP-teoreeman, jonka myöhemmin Gilbert ja Lynch (2002) todistivat. CAP-teoreeman mukaan hajautetut tietokannat voivat taata samanaikaisesti vain kaksi seuraavasta kolmesta ominaisuudesta.

- *Oikeellisuus* (eng. Consistency). Tietokannan jokainen solmu on aina oikeellisessa tilassa, eli ajantasaisin data on replikoitunut kaikille solmuille (vastakohtana BASE:n soft state).
- *Saatavuus* (eng. Availability). Tietokanta on aina saatavilla, vaikka yksittäinen solmu lopettaisikin toiminnan (vertaa BASEN:n basically available).

- *Osituksen toleranssi* (eng. Partition tolerance). Tietokannan kykyä jatkaa toimintaansa solmujen menettäessä yhteyden keskenään esimerkiksi verkkovirheen vuoksi.

Erityisesti verkon yli hajautettujen järjestelmien on kestettävä verkkovirheitä. Tällöin CAP-teoreemasta valittaviksi ominaisuuksiksi jäävät oikeellisuus ja saatavuus, koska osituksen toleranssia on tuettava. Oikeellisuuden valinta johtaa tietokantaan, jonka toiminta muistuttaa enemmän relaatiotietokantojen toimintaa (vrt. ACID-malli). Saatavuuden valitseminen korostaa NoSQL-tietokantojen tyyliä. Eri tietokantavalmistajat ovat usein valinneet kaksi näistä kolmesta ominaisuudesta, jotka tietokanta toteuttaa.

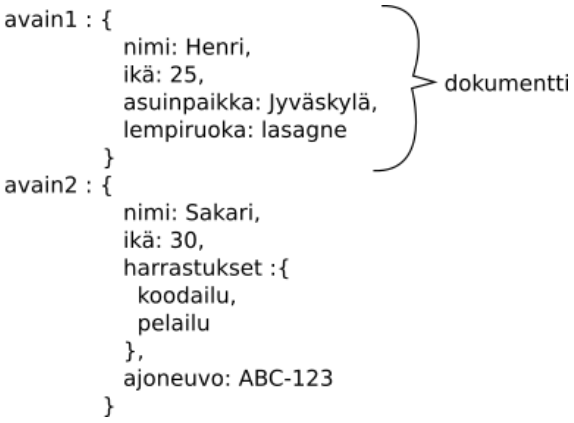
3.3 Erilaisia NoSQL-tietokantatyyppejä

Erilaisia NoSQL-tietokantoja on NoSQL database (2017) -sivuston mukaan yli 200, ja niitä voidaan luokitella monella eri tavalla. Useimmiten luokittelu tapahtuu datamallin mukaan. Datamallin mukaan luokiteltuna yleisimpiä tietokantoja ovat:

- *Avain-arvo -tietokanta* (eng. Key-Value store). Sen rakenne koostuu nimensä mukaisesti avain-arvo pareista (vrt. map ja dictionary). Jokainen tallennettu avain on uniikki, ja sen avulla tietokannasta voidaan hakea siihen liitettyä tietoa.
- *Saraketietokanta* (eng. Column store). Saraketietokanta on samankaltainen kuin avain-arvo -tietokanta, mutta lisäksi sarakkeeseen kuuluu aikaleima. Aikaleimaa käytetään esimerkiksi kertomaan mikä hajautetun tietokantaympäristön solmujen tiedoista on uusin.
- *Verkkotietokanta* (eng. Graph database). Verkkotietokannassa data esitetään solmuina, joita voidaan linkittää toisiin solmuihin. Solmuihin sekä solmujen välisiin suhteisiin voidaan tallentaa dataa, usein avain-arvo -pareina, jotka kertovat kohteista ja niiden välisistä linkityksistä.
- *Dokumenttitietokanta* (eng. Document store). Kyseinen tietokanta pohjautuu myös avain-arvo -tietokantaan, jossa uniikki avain identifioi tallennetun tiedon. Data tallennetaan usein tietyssä formaatissa, kun taas avain-arvo -tietokannassa tallennettu data voi periaatteessa olla millaisessa muodossa tahansa. Dokumenttipohjaiset tietokannat käyttävät myös dokumenttien tietoja enemmän hyödyksi suorituskyvyn optimoin-

nissa. Dokumenttitietokannat tarjoavat usein kattavat rajapinnat tietojen hakuun myös dokumenttien sisällön perusteella. Yleisimpiä dokumenttien tallennusformaatteja ovat JSON, YAML, XML ja BSON.

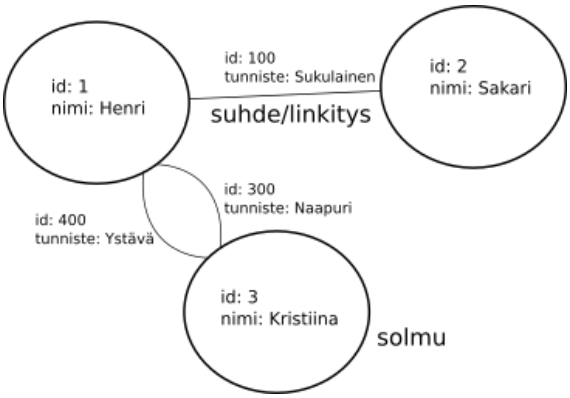
Kuvat 2, 3, 4 ja 5 havainnollistavat edellämainittujen tietokantatyyppien rakennetta. Kuvat mukailevat pitkälti WWW-sivuilta jo runsaasti löytyviä esimerkkikuvia tietokantarakenteista.



Kuvio 2. Esimerkki dokumenttitietokannasta

Kuvio 3. Esimerkki avain-arvo -tietokannasta

Kuvio 4. Esimerkki saraketietokannasta



Kuvio 5. Esimerkki verkkotietokannasta

4 Tietokantojen suorituskykyjen mittaaminen

Tietokantojen suorituskykyjen vertaileminen on olennainen osa sopivan tietokannan valitsemisprosessia. Nopea tietokanta on tärkeä perusta hyvin toimivalle ohjelmistokokonaisuudelle. Tietokantojen suorituskykyjen mittaaminen ja tulosten julkaiseminen on siis tärkeää, sillä siitä on hyötyä monille eri tahoille.

Tietokantojen suorituskyvyn mittaamiselle sekä mittareille ei ole olemassa yksiselitteistä tapaa tai suuretta. On olemassa monia erilaisia tapoja testata ja mitata suorituskykyä, mikä luonnollisesti hankaloittaa eri tulosten keskinäistä vertailua. Samasta syystä aiheesta on vaikea löytää perusteoksia. WWW-sivuilta löytyy onneksi paljon aiheeseen liittyvää informaatiota erilaisista blogeista, foorumeilta sekä yritysten (etenkin tietokantavalmistajien ja erilaisten tietokantapalveluiden tarjoajien) kotisivuilta. Vaikka menetelmiä on useita, ovat toiset niistä käytetympiä.

4.1 Mittarit

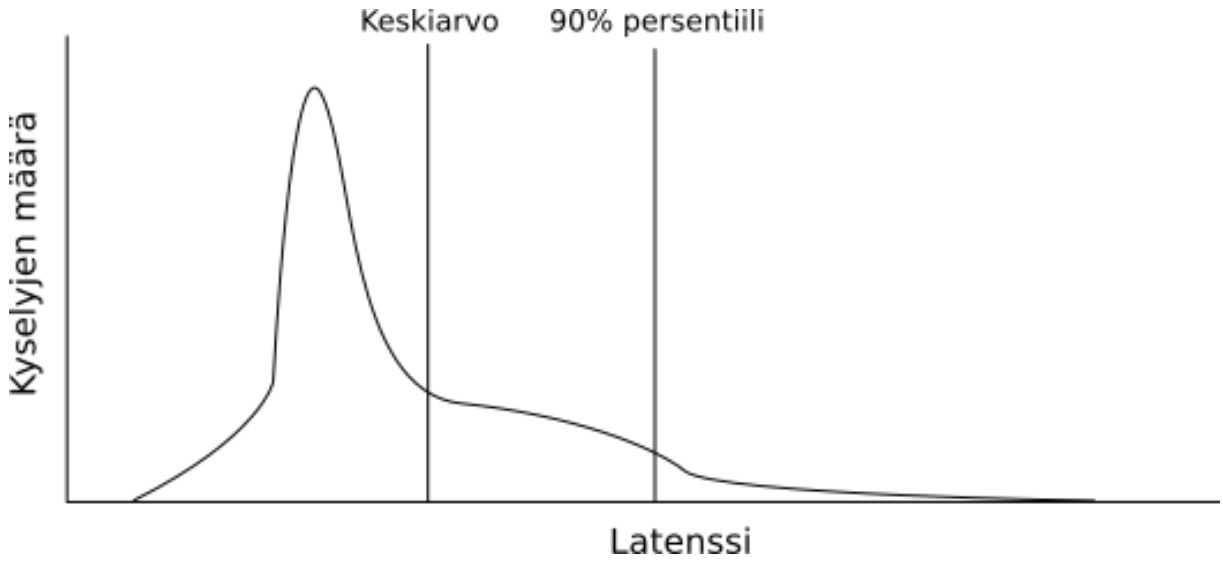
Yleisin mittari tietokannan suorituskyvylle on suoritusteho (eng. throughput). Sitä mitataan suoritettuna työnä ajan suhteen, ja usein se ilmaistaan tehtyinä operaatioina/transaktiona per sekunti (Schwartz 2018; Smartbear 2018; Oracle 2018; Merriam-Webster 2018). Suoritusteho kertoo, kuinka tehokkaasti tietokanta pystyy suorittamaan tietokantaan tehtyjä kyselyjä. Useat tietokantavalmistajat ilmoittavat tietokantojensa suoritustehoja. Pelkästään niiden perusteella tietokantojen vertaileminen on vaikeaa, koska ei ole olemassa standardia transaktiota. Transaktio voi koostua esimerkiksi useista erilaisista operaatioista (Bitton ym. 1985).

Toinen usein tarkasteltu mittari on tietokantakyselyihin kulunut aika eli kyselyaika tai vastausaika (eng. query time tai response time). Kyselyaika on nimensä mukaisesti aika, joka kuluu yksittäisen tietokantakyselyn vastaanottamiseen asiakkaalta, sen suorittamiseen ja vastauksen palauttamiseen asiakkaalle (Logicalread 2018; Carter 2018). Kyselyajasta puhutaan myös luku- ja kirjoituslatenssina, eli aikana joka kuluu vastaavasti luku- ja kirjoitusoperaatioihin. Kyselyaikaan vaikuttavat tietokannan toiminnan lisäksi muun muassa I/O- sekä verkkolatenssi (Petkovic 2018). Hidas laitteisto tai verkkoyhteys lisäävät latenssia eli kyselyyn

kulunutta aikaa, mikä ei ole tietenkään toivottua. Kyselyaika riippuu myös miltä laitteelta latenssia mitataan. Latenssia voidaan esimerkiksi mitata suoraan tietokannasta tai kyselyn tehneeltä asiakkaalta. Asiakkaan laitteelta tai ohjelmistosta mitattaessa on syytä muistaa, että latenssia kasvattaa esimerkiksi ohjelmiston huono toteutus tai hidas verkkoyhteys, mikäli tietokantaan otetaan yhteys verkon ylitse (Packet-foo 2018; Petkovic 2018).

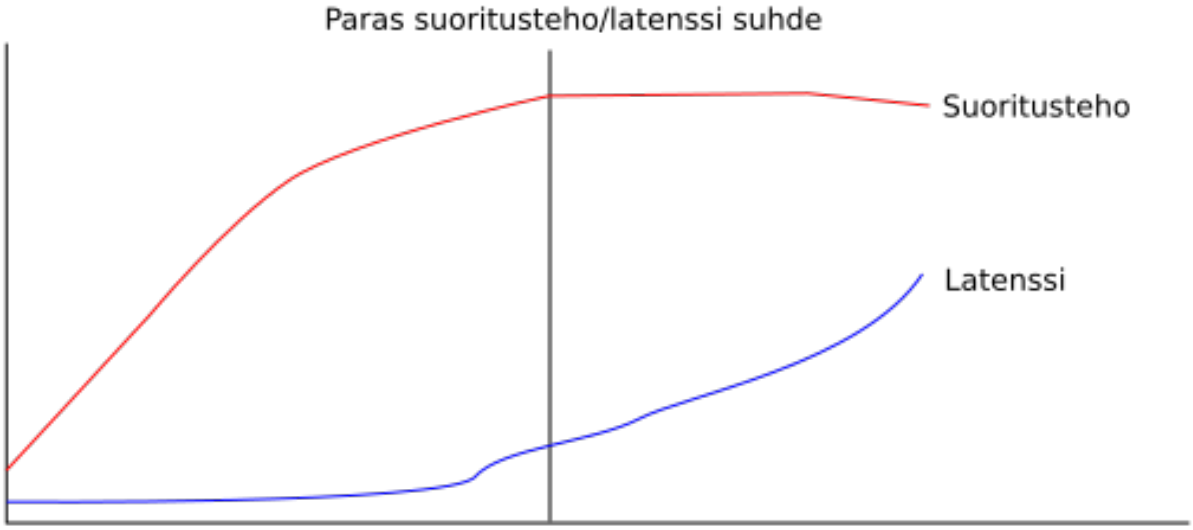
Latenssien tulkitsemiseen käytetään usein persentiilejä. Persentiili kuvaa arvoa, jonka alle kuuluu tietty osa mitatusta populaatiosta (Statistics How To 2018; Treat 2018). Esimerkiksi jos mitattujen tietokantakyselyiden suoritusajan 95% persentiili on 100ms, tällöin 95% tietokantakyselyistä suoriutui 100 millisekunnissa tai nopeammin. Persentiilin avulla pyritään hankkiutumaan eroon yksittäisistä suurista mittausarvoista, jotka vääristävät tuloksia. Lisäksi on usein kiinnostavampaa tietää, mikä on suurin suoritusaika tietyltä persentiililtä verrattuna vaikkapa keskiarvoon tai mediaaniin (Kopp 2018; Tong 2018). Usein käytettyjä persentiilejä ovat 90%, 95%, 99% ja jopa 99.9%.

Kuvassa 6 on esimerkki kuinka keskiarvo ja persentiili voivat erota toisistaan. Niiden tulkitseminen väärin voi johtaa helposti myös huonoihin päätöksiin jatkotoimenpiteiden kannalta. Vaikka keskiarvo näyttäisi hyvältä, saattaa silti suuri määrä kyselyistä suoriutua hyvin hitaasti, mikä voi olla ongelma.



Kuvio 6. Esimerkki keskiarvon ja persentiilin eroista

Yleensä suoritustehoa ja latenssia mitattaessa halutaan löytää kohta, jossa tietokannan suoritusteho ei enää nouse ja latenssit pysyvät alhaisina (Hussain 2018; Pandey 2018). Järjestelmästä ja tarpeista riippuen alhainen latenssi on arvoltaan hyvin epätarkka, ja se täytyykin määrittää aina tapauskohtaisesti. Suorituskykytesteissä, joissa tietokantaan kohdistuu tasainen rasitus tietyn ajan, suoritusteho saturoituu yleensä tietylle alueelle. Saturaatiopisteen läheisyydessä latenssit alkavat kasvamaan, koska tietokanta ei voi vastata kyselyihin nopeammin (Hussain 2018; Pandey 2018). Saturaatiopistettä pidetäänkin yleensä suorituskykytestin parhaana tuloksena. Kuva 7 havainnollistamaan suoritustehon ja latenssin käyttäytymistä, sekä osoittaa pisteen, jossa tietokanta saavuttaa optimaalisimman suoritustehon ja latenssin suhteen.



Kuvio 7. Esimerkki suoritustehon ja latenssin suhteesta

Mittarina voidaan käyttää myös hinta/suoritusteho -suhdetta (Gray 1992; Bitton ym. 1985), mikä voi olla hyödyllisin mittari, valittaessa kustannustehokkainta tietokantaa. Hintana voidaan käyttää esimerkiksi vuosittaista kustannusarviota, johon kuuluvat mahdolliset lisenssit, alustavuokrat ja ylläpitomaksut.

4.2 Työkalut

Suorituskykyä ja latensseja voidaan mitata useilla eri tavoilla. Yksinkertaisimmillaan suorituskykytesti voi olla jokin ohjelmakoodi, joka tekee ennaltamäärätyn määrän tietokantakyselyjä tietokantaan. Tietokantakyselyiden määrä jaetaan suoritukseen kuluneella ajalla, ja näin saadaan tietokannan suoritusteho. Latenssi voidaan vastaavasti mitata jokaiselle yksittäiselle kyselylle, ja laskea lopuksi esimerkiksi näiden keskiarvo. Kaikki suorituskykytestit pohjautuvat enemmän tai vähemmän edellä esitettyihin menetelmiin. Tietokantojen suorituskyvyn mittaamiseen tarkoitettuja työkaluja ja niiden tuloksia on julkaistu paljon WWW-sivuilla.

Suorituskykytestit voidaan luokitella sen perusteella ovatko ne synteettisiä, vai pohjautuvatko ne johonkin oikeaan systeemiin. Synteettisiä, eli keinotekoisia, suorituskykytestejä on usein helpompi toteuttaa, mutta ne eivät välttämättä kuvasta hyvin oikeita systeemejä ja niiden käyttötapauksia. Realistiset suorituskykytestit vaativat usein paljon monimutkaisempia järjestelmiä sekä suunnittelua, mutta todennäköisesti kuvastavat paremmin suorituskykyä oikeassa käytössä.

Eräitä suosittuja kehyksyiä suorituskykyjen mittaamiseen ovat TPC-organisaation julkaisemat suorituskykytestit ja suorituskykytestien määritelmät. Transaction Processing Performance Council (TPC) on voittoa tavoittelematon järjestö, joka pyrkii kehittämään standardisoituja suorituskykytestejä ja suorituskykytestien määritelmiä. TPC on julkaissut useita suorituskykytestejä eri tarkoituksiin, kuten päätöksenteon (Decision support), Big Datan ja IoT-laitteiden testaamiseen.[1] TPC:llä on tällä hetkellä yhteensä 16 aktiivista suorituskykytestiä ja -määritelmää (TPC 2017).

Suosituin TPC:n julkaisema suorituskykytestin määritelmä lienee TPC-C, jolla mitatataan transaktio-orientoituneiden (OLTP, Online transaction Processing) järjestelmien suorituskykyä.[2] Transaktio-orientoituneet järjestelmät ovat järjestelmiä, jotka lukevat ja prosessoivat dataa tietokannasta. Termi transaktio-orientoitunut järjestelmä pitää sisällään kaiken siihen liittyvän, eli ei pelkästään tietokantaa (Gray ja Reuter 1992). TPC-C suorituskykytestiä käytetään kuitenkin paljon myös pelkkien tietokantojen suorituskykyjen testaamiseen, koska

1. TPC-DS suorituskykytesti päätöksenteolle TPCx-BB suorituskykytesti Big Datalle TPCx-IoT suorituskykytesti IoT-laitteille

2. TPC-C suorituskykytesti

oleellinen osa transaktion toimenpiteistä tapahtuu tietokannassa.

TPC:n suorituskykytesteissä ja yleensäkin transaktio-orientoituneista järjestelmistä puhuttaessa transaktiolla tarkoitetaan suurempaa kokonaisuutta: toimenpidettä, joka koostuu yleensä useammasta kuin yhdestä operaatiosta. Siten transaktiolla ja operaatiolla on suuri ero, ja niiden vertaaminen keskenään ei ole mielekästä.

TPC-C (kuten muutkin TPC-organisaation suorituskykytestit) mallintaa hyvin oikeita systeemejä. Tämä antaa todenmukaisemman kuvan mitattavan järjestelmän suorituskyvystä verrattuna synteettisiin suorituskykytesteihin. TPC-C on vain määritelmä suorituskykytestille, eikä TPC-organisaatio tarjoa siihen valmista implementaatiota. TPC-C suorituskykytestistä löytyy kuitenkin paljon implementaatioita esimerkiksi google-haulla.

Toinen suosittu työkalu on Yahoo! Cloud Service Benchmark (YCSB) (Cooper ym. 2010). Se on kehitetty erilaisten NoSQL-tietokantojen (tarkemmin ottaen avain-arvo-pohjaisten tietokantojen) suorituskykyjen testaamiseen, ja se onkin kerännyt jonkin verran suosiota epävirallisena standardina NoSQL-tietokantojen suorituskyvyn mittaamiseen. Työkalulla voi myös testata relaatiotietokantojen suorituskykyä, joten työkalu on hyvin monikäyttöinen.

YCSB koostuu periaatteessa kahdesta osasta: testidatan generoinnista ja lataamisesta tietokantaan sekä suorituskykytestien suorittamisesta. YCSB on tarkoitettu helppokäyttöiseksi sekä helposti konfiguroitavaksi ja laajennattavaksi. YCSB:n generoima testidata on kuitenkin geneeristä (pelkkää binääridataa), ja suorituskykytestit ovat hyvin synteettisiä, vaikka niillä pyritäänkin kuvastamaan oikeita systeemejä.[3]

Synteettisissä suorituskykytesteissä on myös puolensa. Niillä voidaan saada pienellä vaivalla tietokannoille jokin vertailutaso, jonka pohjalta voidaan lähteä rakentamaan tarkempaa analyysia. Synteettiset suorituskykytestit ovat myös yksinkertaisuudensa ansiosta yleensä helposti eristettävissä koskemaan tiettyä järjestelmän osaa tai tietynlaisia operaatioita, mikä mahdollistaa tarkemman, pienempien kokonaisuuksien ja osa-alueiden testaamisen.

3. YCSB:n mukana tulevat suorituskykytestit ja niiden kuvaukset

5 Tutkimusmenetelmä

Tutkielmassani pyrin selvittämään suorituskykyeroja MySQL- ja MongoDB-tietokantojen välillä. MySQL-tietokanta edusti perinteistä relaatiotietokantaa ja MongoDB edusti NoSQL-tietokantaa. Eri valmistajien tietokantoja on satoja erilaisia, joten rajaus verrattavien tietokantojen välillä oli tehtävä. Valitsin kyseiset kaksi tietokantaa pääasiassa siksi, että ne ovat omissa tietokantatyypeissään käytetyimpien tietokantojen joukossa (DB-Engines 2017). Molemmista tietokannoista on kattavat dokumentaatiot, ja niistä löytyy paljon tietoa ja apua mm. foorumeilta ja muilta yhteisöjen keskustelupaikoilta.

Lähtökohtana tutkimukselleni sovelsin PICOC-ajattelumallia. PICOC-ajattelumalli on lähtöisin lääketieteen tutkimuksesta, mutta Kitchenham ja Charters (2007, kpl. 5.3.2) ovat kirjoittaneet materiaalin, jossa kuvataan kuinka ajattelumallia sovelletaan ohjelmistokehityksen parissa. PICOC on joukko kriteerejä, joiden tarkoitus on helpottaa tutkimuskysymyksen muotoilemista. *Populaatio* kertoo keihin henkilöihin tai ryhmiin tutkimus vaikuttaa. *Interventio* on vaihtoehtoinen tapa tehdä tutkittava asia. Tätä verrataan *vertailukohtaan* (vrt. lääketieteessä kontrolliryhmä). *Mittari* kertoo mitä tutkimuksessa mitataan, ja minkä perusteella vertailut ja johtopäätökset tehdään. *Konteksti* kuvaa millaisessa ympäristössä tutkimus tehdään. Esimerkiksi teollisuudessa ja akateemisessa ympäristössä tehdyt tutkimukset ja niiden tavoitteet ovat hyvin erilaisia. Tässä tutkimuksessa olen soveltanut PICOC-ajattelumallia seuraavasti:

- *Populaationa* (eng. Population) toimii ohjelmistokehittäjät ja -arkkitehdit.
- *Interventiona* (eng. Intervention) toimii NoSQL-tietokanta, tämän tutkielman tapauksessa MongoDB.
- *Vertailukohtana* (eng. Comparison) toimii perinteinen relaatiotietokanta, tässä tapauksessa MySQL.
- *Mittarina* (eng. Outcome) toimii suorituskyky.
- *Konteksti* (eng. Context) on tutkimuksen tapauksessa akateeminen.

Tutkimustavoitteenani oli selvittää, onko NoSQL-tietokannan suorituskyky huomattavasti parempi verrattuna perinteiseen relaatiotietokantaan. Kontekstin ollessa akateeminen, ja tut-

kimuksen ollessa pienimuotoinen, tuloksia ei voida juuri yleistää, ei ainakaan täydellä varmuudella. Toivon kuitenkin tutkimukseni antavan ohjelmistokehittäjille ja -arkkitehdeille tuloksia, joiden pohjalta voi suorittaa kattavampaa analyysia esimerkiksi sopivan tietokannan valitsemiseen.

Tutkimuksessani käytin Yahoo! Cloud Serving Benchmark -työkalua (YCSB) suorituskykytestien generointiin ja mittaamiseen. Valitsin YCSB:n työkaluksi omaan tutkimukseeni, koska YCSB-työkaluun on tehty implementaatiot sekä MySQL että MongoDB tietokantojen testaamiseen. YCSB on myös helppokäyttöinen, joten käyttöönotto ja testiympäristön pystyttäminen oli suoraviivaista. YCSB:tä on käytetty myös useissa muissa suorituskykytesteissä, joten tuloksia oli helppo vertailla muiden samalla työkalulla tehtyjen tutkimustulosten kanssa.

Hypoteesinani oli, että MongoDB suoriutuu suorituskykytesteistä paremmin kuin MySQL. Syy tälle oletukselle oli pääasiassa se, että NoSQL-tietokannat ovat yleensä suunniteltu korkean suorituskyvyn järjestelmiksi. Toki perinteisistä relaatiotietokannoista pyritään kehittämään niin tehokkaita kuin mahdollista, mutta usein relaatiotietokannoille ominaisten piirteiden, kuten ACID-transaktioiden, toteuttaminen johtaa kompromisseihin suorituskyvyn kanssa. Lisäksi testiasetelma suosii MongoDB-tietokantaa, koska YCSB on pääasiassa suunniteltu NoSQL-tietokantojen suorituskyvyn testaamiseen.

5.1 Laitteiston ja ohjelmistojen tiedot

Suorituskykytestit tehtiin yhdellä ja samalla tietokoneella. Tietokoneen ja käytettyjen ohjelmistojen tarkemmat tiedot ovat esiteltynä taulukoissa 2 ja 3.

MongoDB ja MySQL-tietokantaohjelmistot asennettiin Debianin *apt* pakettimanagerityökalun avulla. Ohjelmistoja käytettiin *out-of-the-box* periaatteella, eikä niiden oletuskonfiguraatioihin tehty mitään muutoksia. YCSB asennettiin sen wiki-sivuilla olevien ohjeiden mukaan.[1]

1. https://github.com/brianfrankcooper/YCSB/wiki/Getting-Started

Laitteisto	
Tietokone	Lenovo Thinkpad X220 (tuotenumero: 4291Y2S)
Prosessori	Intel® Core™ i5-2540M CPU @ 2.60GHz x 4 (L1 cache 64KiB, L2 cache 256KiB, L3 cache 3Mib)
Keskusmuisti	Samsung 2x4GiB DDR3 1333Mhz (tuotenumero: M471B5273DH0-CH9)
Kovalevy	Hitachi 320Gib ATA HDD (tuotenumero: HTS72323) EXT4 volume
Käyttöjärjestelmä	Ubuntu 16.04 LTS 64bit (kernel versio: 4.4.0-96-generic x86_64 GNU/Linux)

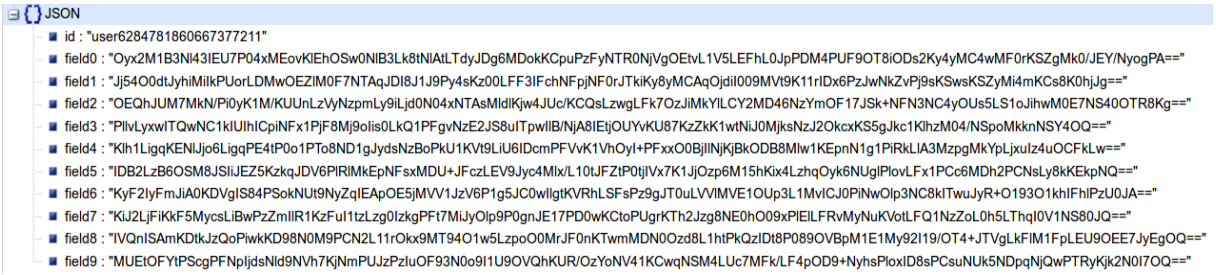
Taulukko 2. Testilaitteiston tiedot

Ohjelmistot	
YCSB	release 0.12.0
Java	OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
MongoDB	versio 3.4.9 community edition, MongoDB Java ajurin versio: 3.0.3
MySQL	Versio 5.7.19 for Linux on x86_64 (MySQL Community Server (GPL)), JDBC ajurin malli ja versio: MySQL Connector/J 5.1.44

Taulukko 3. Testilaitteen ohjelmistojen tiedot

5.2 Testidata ja suorituskykytestit

Aloitin tutkimukseni käyttämällä YCSB-työkalua datan generointiin ja lataamiseen tietokantoihin. YCSB:n generoima data on hyvin geneeristä (binääridataa), joka on hyvä huomioida tuloksia analysoidessa. Kuvassa 8 on esimerkki YCSB:n generoimasta datasta JSON-dokumenttina. Yksi dokumentti tai rivi koostuu avaimesta sekä kymmenestä datakentästä. Datan generoinnissa ja lataamisessa käytettiin YCSB:n vakioasetuksia, koska tällöin muihin samalla työkalulla tehtyihin tuloksiin vertaaminen on järkevämpää.



Kuvio 8. Esimerkki YCSB:n generoimasta datasta JSON-muotoisena

Tein yhteensä neljä erilaista suorituskykytestiä. Näistä kolme oli YCSB:n oletustestejä, ja yksi oli itse konfiguroitu testi. Suoritin testit kahteen erikokoiseen tietokantaan, kymmenen tuhannen ja sadan tuhannen rivin (MySQL) tai dokumentin (MongoDB) kokoisille tietokannoille. Tässä tutkielmassa viittaan näihin tietokantoihin *pienenä* ja *isona tietokantana*. Jokaista testiä ajettiin kahden minuutin ajan, ja tältä ajalta otettiin suorituteshon keskiarvo (suoritetut operaatiot / sekunti) sekä operaatioista riippuen 99% persentiilin luku- ja kirjoituslatenssit. Asiakassäikeiden määrää lisättiin yhdellä säikeellä jokaisen testiajon jälkeen, aloittaen yhdestä säikeestä ja päättyen 12 säikeeseen. Yhdelle suorituskykytestille yhteen tietokantakokoon tuli siis yhteensä 12 testiajoa. Asiakassäikeiden määrää lisäämällä pyrittiin nostamaan kuormaa niin pitkään, kunnes tietokannan suoritusteho saturoituu johonkin pisteeseen.

Yhteensä testiajoja tehtiin 48 kappaletta yhtä tietokantaa ja tietokantakokoa kohden[2]. Lopulliseksi luvuksi testiajoja tietokantaa kohden tuli 96[3], eli yhteensä 192[4].

2. 12 testiajoa x 4 suorituskykytestiä
3. 48 testiajoa x 2 tietokantakokoa
4. 96 testiajoa x 2 tietokantaa

Suorituskykytesteissä käytettiin vain luku (Read) ja päivitys (Update) -operaatioita. Lisäys (Insert) ja poisto (Delete) -operaatioita ei näissä suorituskykytesteissä käytetty, koska ne muuttavat tietokannan kokoa, jolloin tietokanta olisi jouduttu alustamaan aina uudelleen yhden testiajon jälkeen. Tämä olisi lisännyt huomattavasti työmäärää sekä vaikeuttanut suorituskykytestien automatisointia.

Luettavien ja muokattavien tietueiden valinnassa käytettiin tasajakaumaa. Tasajakauma ei mallinna parhaiten oikeita systeemejä, mutta näin pieniä tietokantoja testattaessa en usko sen vaikuttavan oleellisesti tuloksiin. Lisäksi halusin välttää liiallista samojen tietueiden käsittelyä, erityisesti kun testattavat tietokannat olivat kooltaan varsin pieniä.

Ensimmäisessä suorituskykytestissä operaatioiden keskinäinen suhde oli puolet ja puolet. Toisessa suorituskykytestissä vain 5% operaatioista oli kirjoitusoperaatioita. Kolmas suorituskykytesti koostui pelkästään lukuoperaatioista, ja neljäs suorituskykytesti puolestaan koostui pelkistä kirjoitusoperaatioista. Suorituskykytestien luku- ja kirjoitusoperaatioiden prosentuaaliset osuudet on myös taulukoituna taulukossa 4. Erilaisilla luku- ja kirjoitusoperaatioiden suhteilla pyrin saamaan testeihin ja tuloksiin monipuolisuutta. Ne auttavat ymmärtämään kuinka luku- ja kirjoitusoperaatiot kuormittavat tietokantaa, ja kuinka ne toimivat yhdessä. Lisäksi ne mallintavat oikeita systeemejä paremmin, joka auttaa hahmottamaan mahdollisia käyttökohteita ja -tarkoituksia.

Suorituskykytesti 1 kuvastaa esimerkiksi session säilömistä. Palvelimelta kysytään ja päivitetään session tietoja sitä mukaan kun asiakas toimii tietyllä sivustolla. Esimerkki tällaisesta sivustosta on verkkokauppa. Suorituskykytesti 2 on esimerkki sosiaalisesta mediasta: 95/5 luku- ja kirjoitusoperaatioiden suhde ei ole ehkä todenmukaisin, mutta keskimäärin käyttäjät näkevät ja lukevat enemmän muiden sisältöä kuin tuottavat sitä itse. Suorituskykytesti 3 on esimerkki välimuistista jonka sisältö on staattista, tai se päivitetään jotain muuta kautta. Välimuisti rakentuu esimerkiksi sovelluksen käynnistyessä, ja sitä käytetään vain tiedon lukemiseen. Suorituskykytesti 4 kuvastaa esimerkiksi logien tallentamista. Lukuoperaatioita tapahtuisi harvoin, ja ne tehtäisiin muuta kautta.

Suorituskykytestit 1, 2 ja 3 ovat YCSB:n mukana tulevia suorituskykytestejä. Vastaavat suo-

	Lukuoperaatiot %	**Kirjoitusoperaatiot %**
Suorituskykytesti 1	50	50
Suorituskykytesti 2	95	5
Suorituskykytesti 3	100	0
Suorituskykytesti 4	0	100

Taulukko 4. Suorituskykytestien luku- ja kirjoitusoperaatioiden suhteet

rituskykytestit löytyvät YCSB:n wikisivuilta[5] nimillä workoload A, B ja C. Neljänteen suorituskykytestiin muutin konfiguraatioista operaatioiden osuudet seuraavanlaiksi.

```
readproportion=0
updateproportion=1
scanproportion=0
insertproportion=0
```

YCSB:n mukana tulee useita rajapintoja erilaisten tietokantojen testaamiseen. MongoDB:lle tarjolla oli kaksi: synkroninen ja asynkroninen -rajapinta. Käytin näistä ensimmäistä, koska ajattelin sen poistavan yhden kerroksen kompleksisuutta testiasetelmasta. MySQL tietokannan testaamisessa hyödynsin JDBC (Java Database Connectivity) rajapinnan päälle tehtyä toteutusta, jossa käytin MySQL-tietokannan natiivia JDBC-ajuria.[6]

Kaikki testit ajettiin single-user tilassa, jotta testattavat ohjelmistot saivat mahdollisimman paljon tietokoneen resursseja käyttöönsä, eivätkä muut ohjelmistot aiheuttaisi häiriöitä.

5. https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads
6. https://dev.mysql.com/downloads/connector/j/

6 Tulokset

Tässä luvussa on esitelty suorituskykytestien tulokset. Kuvioissa 9–16 on esitetty suorituskykytestien suoritustehot sekä luku- ja kirjoituslatenssit. Kuvioissa 17 ja 18 on esitetty MySQL ja MongoDB-tietokantojen suorituskykytestien yhdistetyt tulokset tietokantojen sisäisen vertailun helpottamiseksi. Liitteissä A–D on esitetty suorituskykytestien suoritustehojen (operaatioita / sekunti) tarkat arvot taulukoituna. Tulokset on pyöristetty lähimpään kokonaislukuun. Luku- ja kirjoituslatenssien tarkkoja arvoja ei ole tässä tutkielmassa esitetty, koska mitatut arvot ovat pieniä, eikä niiden tarkat luvut tuo oleellisesti lisäarvoa tutkielmalle.

Kokonaisuudessaan tulokset olivat varsin odotetut. MongoDB pärjäsi suorituskykytesteissä huomattavasti paljon paremmin kuin MySQL. MongoDB:n saavuttamat suoritustehot olivat moninkertaisia verrattuna MySQL:n suoritustehoihin. MySQL:n suorituskyky jättää paljon toivomisen varaa, ja tulokset olivatkin MySQL-tietokannan osalta yllättävän huonot. MySQL-tietokannan suorituskykytestien aikana syntyi myös muutamia deadlockeja, mutta niiden määrä oli marginaalinen eikä estänyt testaamista. YCSB ei myöskään ota huomioon epäonnistuneita transaktioita tulosraporteissaan, joten ne eivät vaikuttaneet testituloksiin.

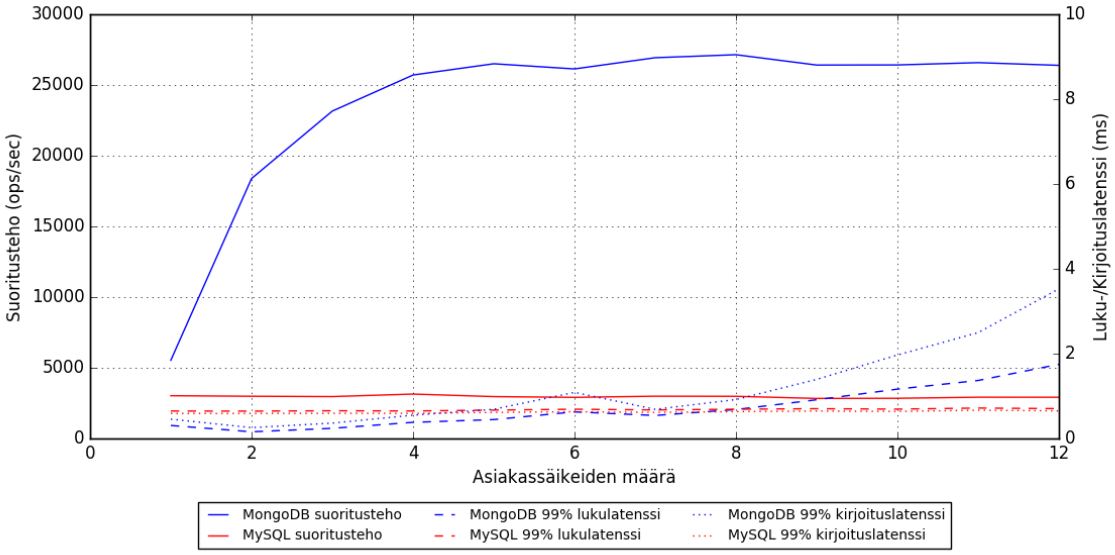
Etenkin kirjoitusoperaatioita sisältävät testit osoittautuivat MySQL-tietokannalle todella raskaiksi. Pelkästään marginaalisen osuuden lisääminen kirjoitusoperaatioita alensi suoritustehoa huomattavasti, kuten suorituskykytestejä 2 ja 3 vertailemalla voi hyvin havaita. 5% lisääminen kirjoitusoperaatioita laski kokonaissuoritustehoa jopa neljäsosaan verrattuna suorituskykytestiin joka koostui kokonaan lukuoperaatioista.

Tuloksiin vaikuttavia tekijöitä on monia. Niitä sekä pohdintaa tuloksien merkityksellisyydestä käsitellään tarkemmin luvussa 7.

6.1 Suorituskykytesti 1

Ensimmäisessä suorituskykytestissä MongoDB saavutti huomattavasti suuremman suoritustehon molempien tietokantakokojen kanssa verrattuna MySQL-tietokantaan. MongoDB säilytti myös suoritustehon varsin hyvin suuremmankin tietokannan kanssa, kun taas MySQL

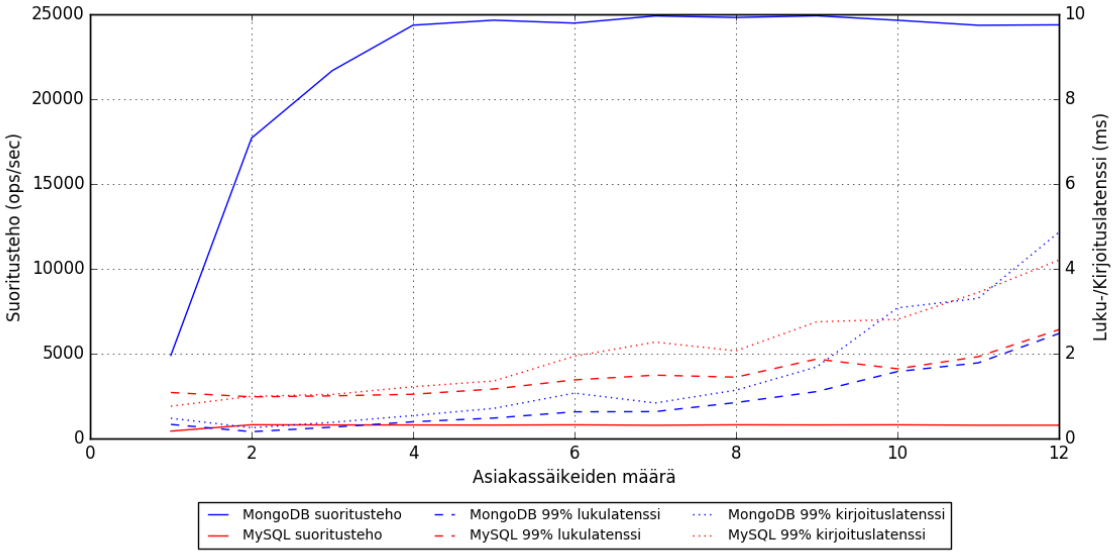
hidastui selvästi verrattuna pienemmässä tietokannassa saatuihin tuloksiin. Molempien tietokantojen luku- ja kirjoituslatenssit pysyivät varsin alhaisina, joka oli odotettavaa. Sekä YCSB -asiakas että tietokanta sijaitsivat samalla fyysisellä laitteella, jolloin verkkolatenssia ei käytännössä ollut. Latenssit kuitenkin nousivat selvästi asiakassäikeiden määrää lisätessä, ja trendiä pystyy päättelemään jo näistä tuloksista.



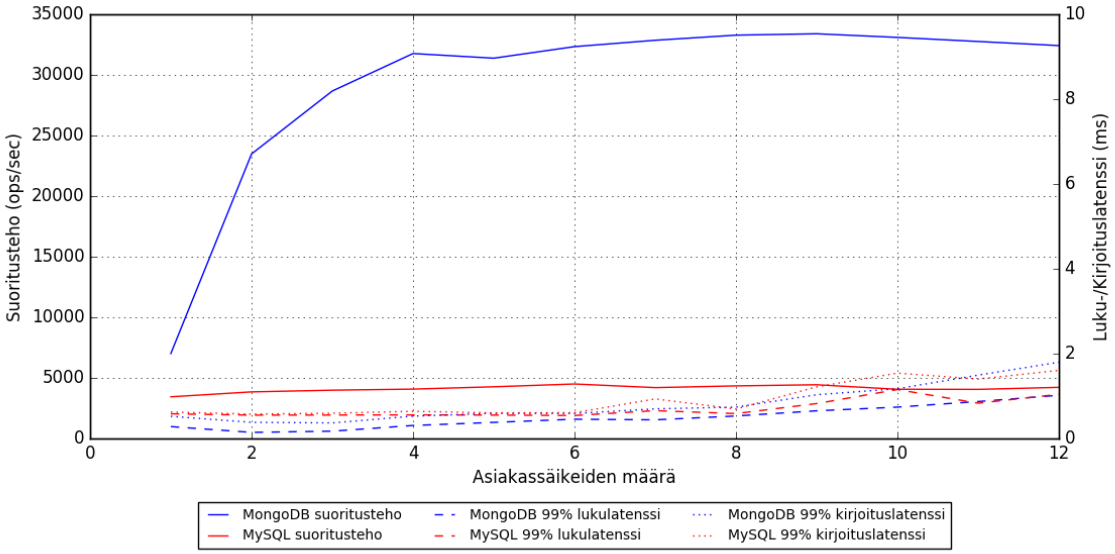
Kuvio 9. Suorituskykytesti 1 - pieni tietokanta

6.2 Suorituskykytesti 2

Toisessa suorituskykytestissä MongoDB saavutti myös suuremman suoritustehon molempien tietokantakokojen kanssa verrattuna MySQL-tietokantaan. Molemmat tietokannat saavuttivat korkeamman suoritustehon verrattuna suorituskykytesteihin 1 ja 4. Tämä selittyy vähäisten kirjoitusoperaatioiden määrällä. Selvää näyttäisi kuitenkin olevan, että kirjoitusoperaatioiden vähäinenkin lisääminen suorituskykytesteihin laskee kokonaissuoritustehoa. Tämä on havaittavissa vertaamalla tuloksia suorituskykytestin 3 tuloksiin. Molempien tietokantojen luku- ja kirjoituslatenssit pysyivät alhaisina myös tässä suorituskykytestissä.



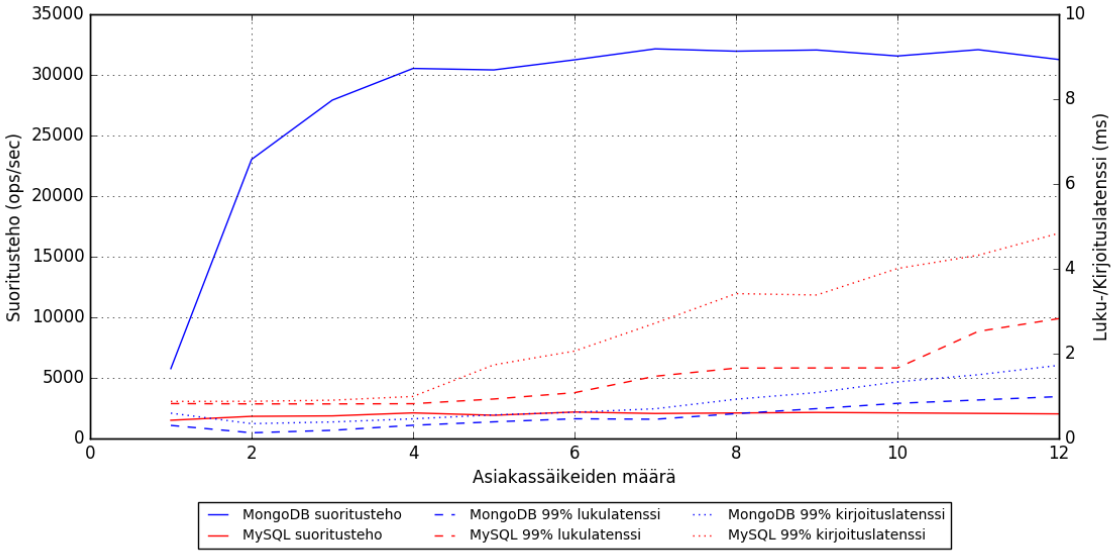
Kuvio 10. Suorituskykytesti 1 - iso tietokanta



Kuvio 11. Suorituskykytesti 2 - pieni tietokanta

6.3 Suorituskykytesti 3

Kolmannessa suorituskykytestissä MongoDB ja MySQL saavuttivat parhaimmat tulokset kaikista neljästä suorituskykytestistä sekä pienellä että isolla tietokannalla. Pelkkien lukuoperaatioiden suorittaminen oli selvästi kevyempää verrattuna kirjoitusoperaatioihin, tai näi-

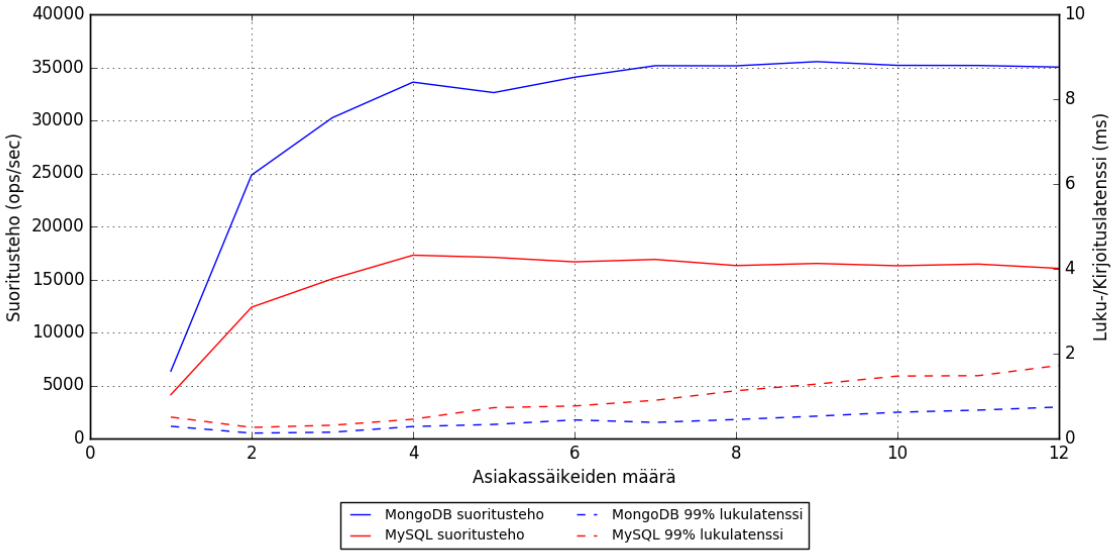


Kuvio 12. Suorituskykytesti 2 - iso tietokanta

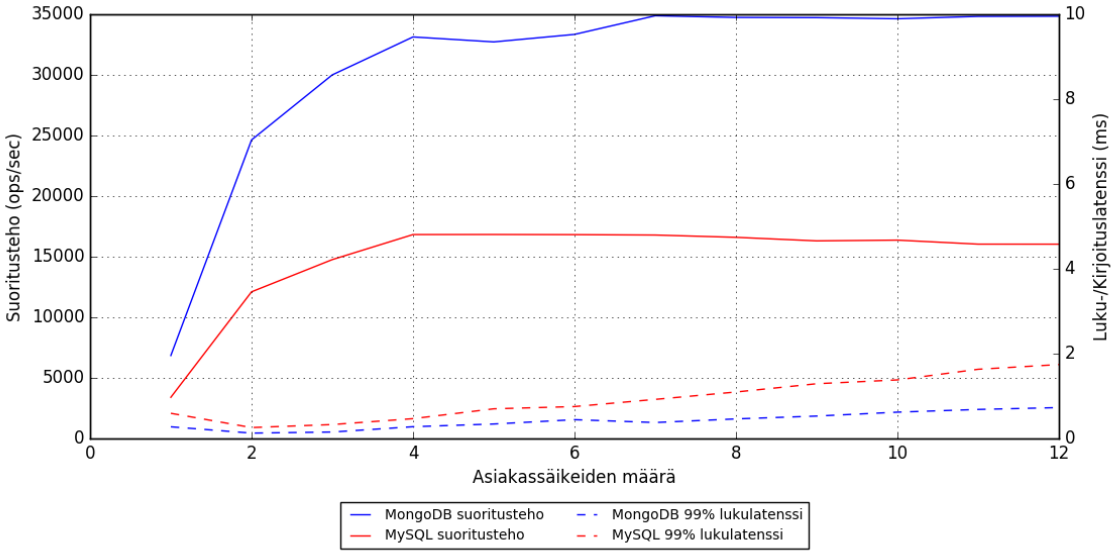
den sekoitukseen. MySQL saavutti myös noin 3 kertaa paremman suoritustehon verrattuna muiden suorituskykytestien MySQL-tietokannan tuloksiin. Myös lukulatenssit pysyivät molempien tietokantojen kohdalla alhaisimpina tässä suorituskykytestissä. Vaikka MySQL-tietokannan suoritusteho oli tässä suorituskykytestissä korkein MySQL-tietokannan tuloksista, oli se silti noin puolet hitaampi verrattuna MongoDB-tietokantaan. On mielenkiintoista huomata, että suurimmat saavutetut suoritustehot ovat sekä pienellä että isolla tietokannalla lähes samat, kun taas muissa suorituskykytesteissä suuremman tietokannan käsittely hidasti molempia tietokantoja, mutta etenkin MySQL-tietokantaa.

6.4 Suorituskykytesti 4

Neljännessä suorituskykytestissä MongoDB oli selvästi nopeampi kuin MySQL. Etenkin suuremman tietokannan suorituskykytesteissä MySQL-tietokannan suoritusteho oli todella alhainen. Myös MongoDB saavutti tässä suorituskykytestissä alhaisimman suoritustehon kaikkien neljän suoritusykytestin osalta. Kirjoitusoperaatiot olivat siis selvästi raskaampia verrattuna lukuoperaatioihin. Huonommista suoritustehoista huolimatta molempien tietokantojen latenssit pysyivät maltillisina kuten muissakin suorituskykytesteissä. Suuremman tietokannan suorituskykytesteissä latenssit alkoivat nousemaan selvästi asiakassäikei-

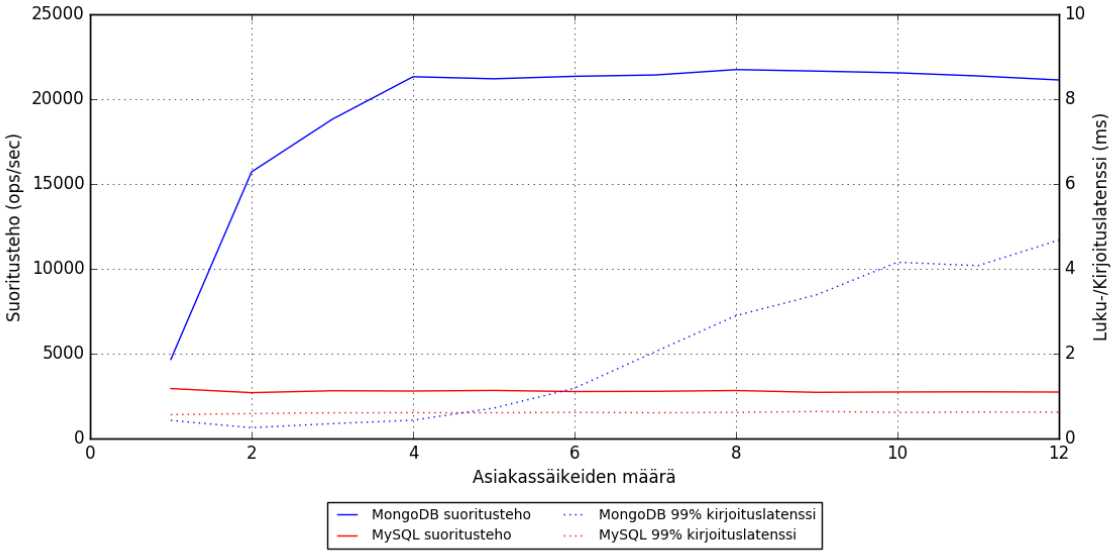


Kuvio 13. Suorituskykytesti 3 - pieni tietokanta

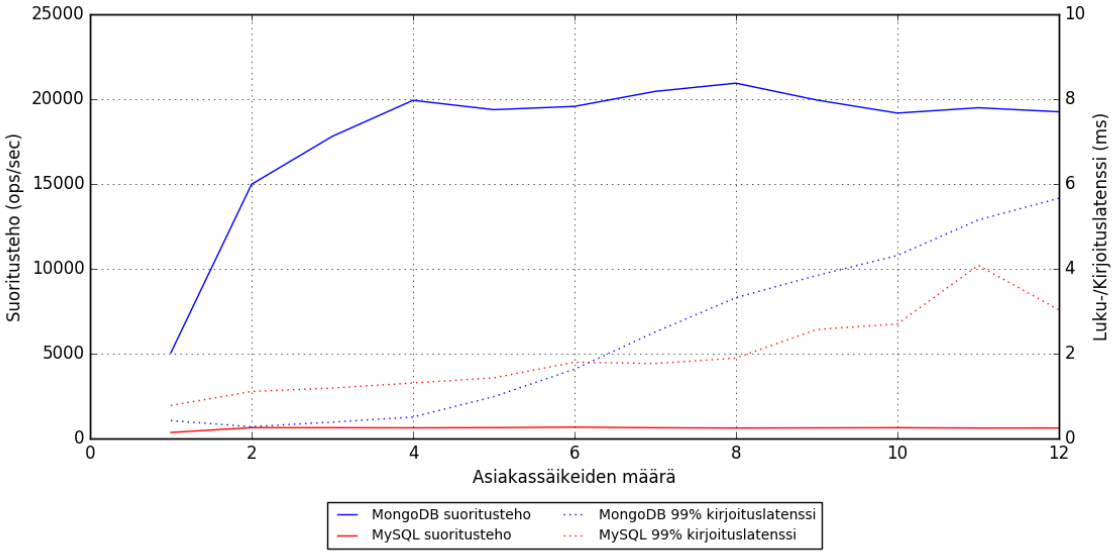


Kuvio 14. Suorituskykytesti 3 - iso tietokanta

den määrää lisätessä, etenkin MongoDB-tietokannalla.



Kuvio 15. Suorituskykytesti 4 - pieni tietokanta



Kuvio 16. Suorituskykytesti 4 - iso tietokanta

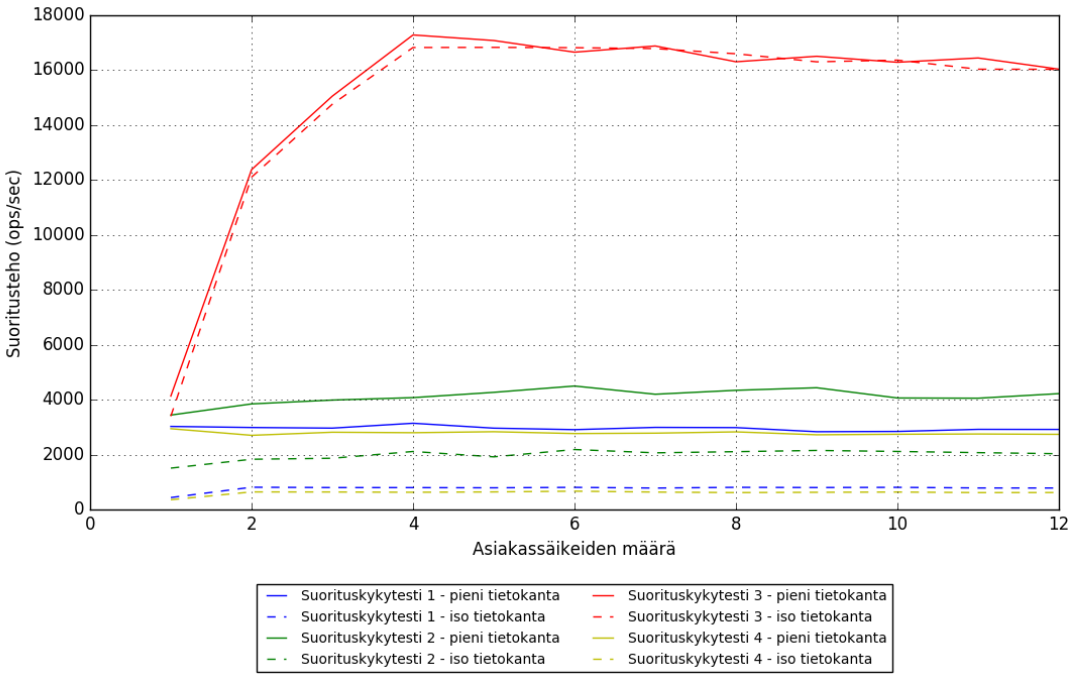
6.5 Yhdistetyt tulokset

Kuvassa 17 on MySQL-tietokannan kaikkien suorituskykytestien tulokset ja kuvassa 18 on MongoDB-tietokannan kaikkien suorituskykytestien tulokset. Tasaisella viivalla on esitetty pieneen tietokantaan tehdyt suorituskykytestit, kun taas katkoviivalla on esitetty isoon tieto-

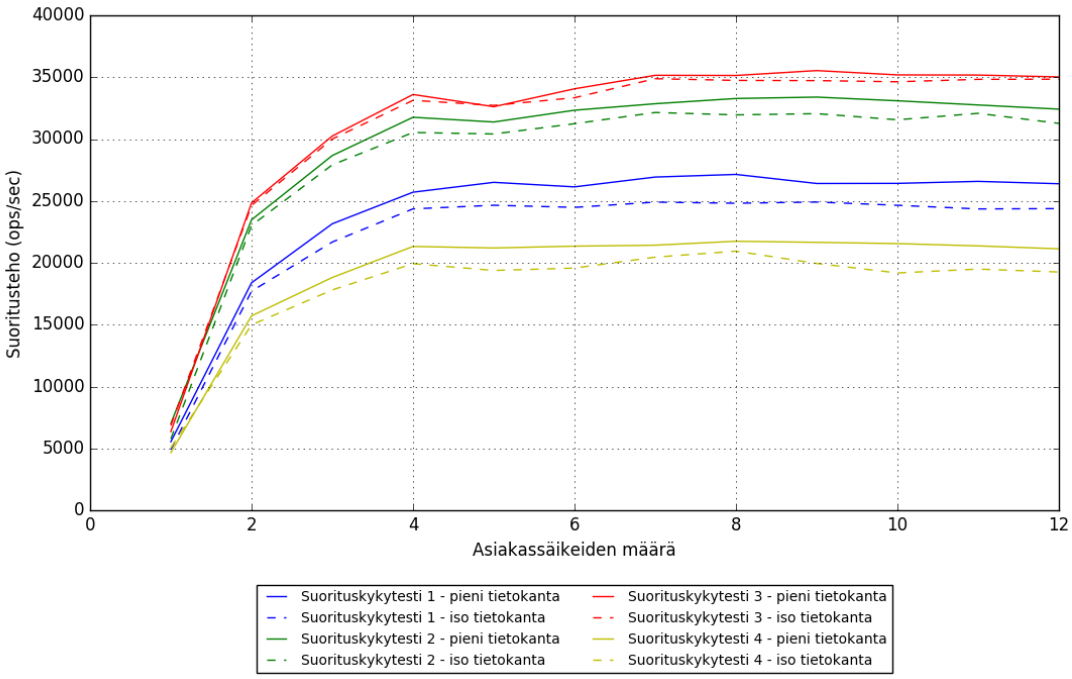
kantaan tehdyt suorituskykytestit. Värit kuvastavat yhtä suorituskykytestiä.

Kuvista on helppo havaita, että MySQL ja MongoDB-tietokannan sisäiset eroavaisuudet suorituskykytestien kesken ovat samanlaiset. MySQL-tietokanta oli selvästi hitaampi kirjoitusoperaatioiden suorittamisessa verrattuna MongoDB-tietokantaan. MySQL-tietokanta koki myös suhteellisesti paljon suuremman tiputuksen suoritustehossa kirjoitusoperaatioiden kohdalla verrattaessa suorituskykytestejä pienen ja ison tietokannan välillä. Siinä missä MongoDB kokee noin 10% pudotuksen suoritustehossa, niin MySQL-tietokannan vastaava pudotus on jopa 50%.

Lukuoperaatioissa tietokantojen sisäisissä eroissa ei ole juuri eroavaisuuksia. Molempien tietokantojen kohdalla tietokannan koko näyttäisi vaikuttavan suoritustehoon vain marginaalisesti. MongoDB-tietokanta oli lukuoperaatioissa noin 60% nopeampi kuin kirjoitusoperaatioissa, mikäli verrataan suorituskykytestejä 3 ja 4. MySQL-tietokannan vastaava ero oli vajaa 600%.



Kuvio 17. MySQL yhdistety tulokset



Kuvio 18. MongoDB yhdistetyt tulokset

7 Tulosten analysointi

Tässä luvussa on tarkemmin pohdittu tuloksiin vaikuttavia tekijöitä ja tulosten oikeellisuutta, sekä vertailtu tuloksia muihin tieteellisiin julkaisuihin. Luvun lopussa on esitetty myös johtopäätökset sekä pohdintaa aiheesta.

7.1 Testien ja työkalujen vaikutus

Testityökalulla on vaikutusta tuloksiin. YCSB on wiki-sivujensa mukaan suunniteltu mittaamaan ja arvioimaan erilaisten avain-arvo -tietokantojen suorituskykyä. Lähtökohta perinteisen relaatiotietokannan mittaamiselle YCSB-työkalulla on huono. Relaatiotietokantoja ei ole suunniteltu avain-arvo -tietokannoiksi, ja kaiken tiedon säilöminen yhdessä tietokantataulussa ei ole optimaalista. Dokumenttipohjaiset tietokannat pohjautuvat avain-arvo -tietokantoihin, joten tässä tutkimuksessa MongoDB:llä oli selvästi etulyöntiasema MySQL-tietokantaan nähden.

YCSB:n generoima kuorma on synteettistä. Todellisessa maailmassa tietokannat kokevat usein hetkellisiä kuormapiikkejä ja tasaisemman kuorman aikoja. YCSB tuottaa tasaista kuormaa niin paljon kuin tietokanta pystyy käsittelemään, eikä tämä välttämättä anna todellista kuvaa tietokannan suorituskyvystä, mikä voi johtaa suuriinkiin arviointivirheisiin, kuten esimerkiksi Boza ym. (2017) tutkimuksessaan totevat. Toisaalta pitkäkestoiset tasaisen kuorman testit antavat hyvän perustan jatkotutkimusten tekemiselle. Lisäksi tulokset periaatteessa kertovat, millaista kuormaa tietokanta jaksaa käsitellä pitkiäkin aikoja.

YCSB:n mukana tulevien implementaatioden välillä voi olla eroja eri tietokantojen kohdalla. Koska YCSB on avoimen lähdekoodin projekti, on hyvin todennäköistä, että tässäkin tutkimuksessa käytetyt implementaatiot ovat eri henkilöiden tekemiä. Vaikka projektilla on ylläpitäjä sekä koodikatselmoinnit ovat käytössä (pull-request), on todennäköistä että kaikki koodi ei ole parhaalla mahdollisella tavalla optimoitua. Tästä voi seurata hyvinkin eriäviä tuloksia verrattuna esimerkiksi muilla tietokantojen suorituskykytestityökaluilla saatuihin tuloksiin. Tässä tutkielmassa en perehtynyt tarkemmin YCSB:n mukavan tulevien implementaatioiden eri toteutuksiin tai versioihin, vaan käytin viimeisintä julkaisua työkalusta sellai-

senaan.

Vaikka YCSB tukeekin Create-, Read-, Update- ja Delete- (CRUD) sekä scan-operaatioiden testaamista, ei tutkielman suorituskykytesteissä käytetty kuin Read- ja Update operaatioita. On hyvin mahdollista, että Create-, Delete- ja scan-operaatiot olisivat antaneet erilaisia tuloksia. Uskon kuitenkin, että MongoDB olisi suoriutunut näistä kaikista operaatioista paremmin kuin MySQL, tässä tutkielmassa saatujen tulosten perusteella.

CRUD-operaatiot ovat tietokantaoperaatioiden perusta, ja siksi onkin yleistä testata ja vertailla niiden suorituskykyä eri tietokantojen kesken. Tämä ei kuitenkaan aina välttämättä anna reilua vertailukohtaa, sillä nykypäivänä tietokannat kykenevät paljon monimutkaisempiinkin operaatioihin ja tehtäviin. Reniers ym. (2017) nostavatkin esille, että yleensä vaatimukset ylittävät pelkkien yksinkertaisten operaatioiden tarpeen. YCSB:n tuottamat suorituskykytestit eivät testaa tietokantojen kehittyneempiä ominaisuuksia, mikä on syytä muistaa ottaa huomioon niin tämän tutkielman, kuin muidenkin tutkimusten tuloksissa.

7.2 SQL ja NoSQL sekä datan vaikutus

Käsiteltävän datan määrällä sekä rakenteella ja monimuotoisuudella on vaikutusta tietokantojen suoritustehoon. Mitä enemmän käsiteltävää dataa on, sitä enemmän se vaatii prosessointia. Kyselyt suurempiin tietokantoihin ovat siis hitaampia, joka on havaittavissa myös tämän tutkimuksen tuloksista. Tietokannat saavuttivat korkeamman suoritustehon pienemmillä tietokannoilla verrattuna suurempiin tietokantoihin.

Datan rakenteella sekä monimuotoisuudella voi olla sekä positiivisia että negatiivisia vaikutuksia suorituskykyyn. Relaatiotietokannat ovat suunniteltuja jakamaan data loogisiin relaatioihin ja näiden välisiin linkityksiin. Hyvin suunniteltu relaatiotietokannan skeema tarjoaa usein myös paremman suorituskyvyn (kaiken muun hyödyn lisäksi) verrattuna huonosti suunniteltuun skeemaan, jossa esimerkiksi kaikki data tallennettaisiin vain yhteen relaatioon.

YCSB:n generoima data on hyvin geneeristä eikä sillä ole mitään vastaavaa oikean systeemin käyttötarkoitusta. Lisäksi data generoidaan tietokantatyypistä riippuen esimerkiksi yhteen dokumenttiin tai relaatioon. Relaatiotietokannan kohdalla tämä ei ole optimaalinen ratkaisu,

ja se on yksi syy, miksi MySQL-tietokanta suoriutui suorituskykytesteissä heikommin kuin MongoDB.

MySQL-tietokannalla oli myös enemmän ongelmia tietokantalukkojen kanssa verrattuna MongoDB-tietokantaan. Molempien tietokantojen lukitusperiaate on samanlainen: MySQL lukitsee rivin, kun taas MongoDB lukitsee dokumentin. Lukkototeutuksissa on varmasti eroja, ja MongoDB näyttäisikin pystyvän käsittelemään tietokantalukkoja paremmin. On toki muistettava, että MySQL tukee ACID-transaktioita, jotka voivat olla yksi syy tietokannan hitauteen. Toisaalta taas myös MongoDB tukee osittaista, dokumenttitason, ACID-transaktioita. Koska tehdyissä suorituskykytesteissä ei ole transaktioita jotka muokkaisivat useaa dokumenttia kerralla, ei MongoDB saa hyötyä ACID-transaktioiden uupumisesta.

7.3 Laitteiston ja ohjelmistojen vaikutus

Kirjoitusoperaatioiden raskauteen vaikuttaa suuresti testilaitteen kovalevy. Nopeammalla kovalevyllä, kuten Flash-muistitekniikkaan pohjautuvalla SSD-levyllä, kirjoitusoperaatioiden suoritustehojen pitäisi teoriassa olla korkeampia, koska levylle kirjoittamiseen kuluu vähemmän aikaa. Tämän tutkielman suorituskykytestien tietokannat olivat kooltaan varsin pieniä, ja mahtuivatkin kokonaan tietokoneen välimuistiin. Tämä selittää myös lukuoperaatioiden suurta nopeutta. Suorituskykytesti 3:n tulokset selittyvätkin sillä, että kaikki tulokset voidaan käyttännössä palauttaa suoraan muistista. Nämä havainnot eivät kuitenkaan selitä MongoDB ja MySQL-tietokantojen suoritustehojen eroja, vaan pikemminkin miksi tuloksissa olevat erot luku- ja kirjoitusoperaatioiden suoritustehoissa olivat niin suuret kaikissa tehdyissä suorituskykytesteissä.

Tuloksiin vaikuttaa myös kuormaa tekevien asiakkaiden ja tietokannan sijaitseminen samalla laitteella. Asiakkaat ja tietokanta kilpailevat samoista resursseista, mikä ei ole optimaalisin ratkaisu tietokannan suorituskykyä mitattaessa. Lisäksi asiakkaan toteutus vaikuttaa paljon resurssien käyttöön. Mikäli asiakas on huonosti toteutettu tai optimoitu, saattaa se suuren resurssien käytön lisäksi turhaan varata niitä, jolloin tietokannalla on vähemmän resursseja käytettävissään. Toisaalta asiakkaiden ja tietokannan sijaitseminen samalla laitteella poistaa verkkoliikenteessä tapahtuvan latenssin, jota syntyisi mikäli asiakkaat ja tietokanta sijaitsisi-

vat eri laitteilla. Matalat latenssit näkyvätkin hyvin tutkimustuloksissa. Korkeimmillaankin latenssit ovat 8ms suuruusluokkaa, joka on varsin alhainen lukema.

Myös käyttöjärjestelmällä voi olla vaikutusta suorituskykyyn. Tietokantojen Windows- ja Linux-jakelut pohjautuvat eri arkkitehtuureille, jolloin niiden suorituskyvyissä on mitä todennäköisimmin eroja. Lisäksi eri käyttöjärjestelmät toimivat eri laitteistolla paremmin tai huonommin. Uskon kuitenkin, että tässä tutkimuksessa mahdollinen ero on häviävän pieni.

Testeissä käytettyjen tietokanta-ajureiden vaikutus on hyvä huomioida. MySQL-tietokannan kanssa käytettiin natiivia ajuria, joka on varmasti käytetyin ajuri universaalisti. Täten tulokset ovat vertailukelpoisempia muiden tutkimusten kanssa. Ajureita on kuitenkin olemassa useita, ja esimerkiksi CData -niminen yritys on keskittynyt tehokkaiden tietokanta-ajureiden valmistamiseen. Heidän suorituskykymittausten perusteella CData-ajuri on selvästi nopeampi kuin natiivi MySQL-tietokannan ajuri (CData 2017). Testituloksiin kannattaa kuitenkin suhtautua pienellä varauksella, etenkin kun tutkimus on itse ajureita valmistavan yrityksen julkaisema.

Esimerkkinä laitteiston ja ohjelmiston vaikutuksesta nostan Tang ja Fan (2016) tekemän tutkimuksen NoSQL-tietokantojen suorituskykyvertailusta. Vaikka tutkimuksessa käytetään samaa työkalua kuin tässä tutkielmassa, eli YCSB:tä, ja jopa samoja suorituskykytestejä, ovat tulokset hyvin erilaiset. On mielenkiintoista huomata, kuinka Tang ja Fan (2016) tekemässä tutkimuksessa MongoDB ei saavuta läheskään yhtä korkeita suoritustehoja kuin tekmässäni tutkimuksessa, vaikka heidän käyttämänsä laitteisto näyttäisi olevan huomattavasti tehokkaampi. Julkaisusta ei tosin käy ilmi kaikkia tarvittavia tietoja, jotta testi olisi mahdollista toistaa mahdollisimman tarkasti.

Tässä tutkimuksessa tietokantoja testattiin out-of-the-box periaatteella. Molemmat tietokannat asennettiin asennusohjeiden mukaisesti, eikä tarkempia konfiguraatioita tai asetuksia tehty. Testattavalle tietokantataululle ei luotu edes indeksejä, joka olisi todennäköisesti vaikuttanut tietokantojen suoritustehoon positiivisella tavalla. Myös muilla asetuksilla ja optimoinneilla olisi pystynyt vaikuttamaan tuloksiin positiivisella tai negatiivisella tavalla. Tällainen optimointi vaati paljon syvällisempää ammattitaitoa ja perehtymistä kuin tämän tutkielman aikana oli mahdollista toteuttaa. Lisäksi halusin pyrkiä pitämään koeasetelman mahdolli-

simman yksinkertaisena, jotta kokeiden mahdollinen replikoiminen olisi mahdollisimman tarkasti sekä helposti toteutettavissa.

7.4 Tulokset muissa tutkimuksissa

Erilaisia suorituskykytestejä eri tietokannoille on tehty lukuisia. Osa on lähestymistavaltaan akateemisempia, ja tutkimuksista on yleensä tehty julkaisuja, kun osa on teollisuuden kontekstissa toteutettuja, joita löytyy esimerkiksi eri tietokantavalmistajien kotisivuilta.

Rapa (2016) vertailee pro gradu -tutkielmassaan MySQL ja MongoDB-tietokantojen suorituskykyeroja. Suorituskykytesteissä ei käytetä mitään valmista työkalua, vaan eri kokoisiin tietokantoihin on suoritettu kolmea tietokantakyselyä, joiden suoritusaikaa on mitattu. Tuloksissa on selviä eroja tämän tutkielman tuloksiin. MySQL selviytyy useassa testissä paremmin kuin MongoDB, etenkin datamäärän kasvaessa, joka on juuri päinvastainen tulos verrattuna tämän tutkielman tuloksiin.

Rapan tutkielmassa on käytetty huomattavasti monimutkaisempia tietokantakyselyitä, jotka testaavat paremmin ja monipuolisemmin tietokannan kehittyneempia ominaisuuksia. Tämän tutkielma suorituskykytestit sen sijaan mittaavat enemmän tietokannan *raakatehoa* suorittamalla mahdollisimman paljon yksinkertaisia tietokantakyselyitä tietokantaan. Lisäksi Rapan tutkimusasetelma kuvastaa paremmin oikeaa systeemiä, ja tietokantarakenteet sekä MySQL- että MongoDB-tietokannoille on suunniteltu sopiviksi, kun tämän tutkielman tutkimusasetelmassa etu oli selvästi MongoDB:n puolella. Yhteistä tuloksissa kuitenkin on, että datamäärän kasvaessa suorituskyky laskee.

Nämä tutkimukset kuvaavat hyvin erilaisten tutkimusten vertailtavuutta keskenään. Tutkimustuloksiin vaikuttavat niin monet asiat, että on hyvin vaikeaa löytää yksiselitteistä totuutta kysymykseen *mikä tietokanta on nopein*. Lisäksi edellä esitetty kysymys on monitulkintainen, ja suorituskykyäkin voidaan mitata monella eri tavalla. Useita tutkimuksia vertailemalla voi kuitenkin löytää yhteneväisyyksiä tuloksissa, ja muodostaa käsityksen tuloksien yleisestä trendistä.

7.5 Tutkimuksen puutteet

Tutkimus on mielestäni pääsääntöisesti onnistunut, mutta siitä löytyy useita puutteita. Sigplan (2018) ylläpitää erittäin hyvää muistilistaa asioista, jotka tulisi ottaa huomioon (ohjelmistotekniikkaan) liittyvissä empiirisissä kokeissa. Blackburn ym. (2016) ovat julkaisseet hyvän artikkelin, joka käsittelee empiiristä testausta ja tulosten käsittelyä. Artikkeli käsittelee tarkemmin samoja asioita, kuin edellä mainitussa muistilistassa. Tässä luvussa olen nostanut esiin asioita muistilistasta ja artikkelista, jotka olisi ollut hyvä huomioida tämän tutkimuksen empiirisiä testejä tehdessä. Muutamaa asiaa olen jo sivunnut luvun 7 muissa alaluivuissa.

Testissä käytetyn ohjelmiston tulisi olla mahdollisimman hyvin testitapaukseen sopiva sekä mielellään standardoitu. YCSB ei ole kumpaakaa näistä. YCSB ei ole mikään virallisesti vahvistettu standardi, vaikka sitä paljon käytetäänkin. Lisäksi se ei sovellu perinteisten relaatiotietokantojen testaamisen, mikä johtaa vääristyineisiin tuloksiin.

Samat testiajot tulisi tehdä riittävän usein, jotta mittausvirheiden mahdollisuus vähenisi. Tässä tutkimuksessa jokainen testi ajettiin vain kerran, vaikka parempi tapa olisi ajaa testit useaan kertaan, ja laskea tuloksista eri tunnusluvut, kuten keskiarvo. Ajamalla testit useasti olisimme myös nähneet, ovatko tulokset ollenkaan toistettavissa olevia.

Tilastollinen analyysi voisi antaa kuvaavampia tuloksia, joiden avulla testattujen ohjelmistojen vertailua olisi helpompi ja järkevämpi tehdä. Tässä tutkimuksessa tilastollista analyysia ei kuitenkaan tehty, vaan saatuja tuloksia verrattiin suoraan toisiinsa.

Testausympäristön tulisi vastata mahdollisimman hyvin testattavan sovelluksen todellisia käyttöympäristöjä. Tietokantoja käytetään pääasiassa palvelinympäristöissä, jolloin tämän tutkimuksen testiympäristönä toiminut kannettava tietokone ei ole lähimpänä kyseistä ympäristö, mikä voi vääristää tuloksia.

Tutkimuksessa olisi ollut järkevää ja mielenkiintoista tutkia testiympäristön järjestelmäresursseja testiajojen aikana. Tämä olisi voinut valoittaa lisää saatuja tuloksia, sillä järjestelmän tila vaikuttaa tuloksiin mitä todennäköisimmin. Tässä tutkimuksessa nämä tärkeät muuttujat on jätetty kuitenkin tutkimatta.

7.6 Johtopäätökset ja pohdintaa

Tulosten perusteella MongoDB pärjäsi suorituskykytesteissä selvästi paremmin verrattuna MySQL-tietokantaan. MongoDB vaikuttaisi siis soveltuvan hyvin sovelluksiin, joissa tarvitaan tehokasta ja nopeaa datan käsittelyä. Tuloksia on kuitenkin aina muistettava tarkastella kriittisesti, sillä tuloksiin vaikuttavat monet asiat kuten ympäristö, testidata sekä työkalut (Hazelhurst 2010). Tässä tutkielmassa testattiin vain yksinkertaisia operaatioita. Esimerkiksi monimutkaisemmilla hakukyselyillä tietokantojen suorituskykyerot voisivat kaventua, tai olla jopa päinvastaiset. Testattavien tietokantojen koot olivat myös hyvin pieniä verrattuna oikeisiin systeemeihin. Näiden ja muiden edellä mainittujen asioiden nojalla tuloksia ei voi yleistää kovin laajalle alalle, ei ainakaan täydellä varmuudella.

Käytettävyydeltään MongoDB-tietokanta yllätti positiivisesti. Käytön aloittaminen on helppoa ja dataa pystyy tallentamaan ilman skeeman suunnittelua. Samasta syystä esimerkiksi uusien tietueiden lisääminen dokumentteihin on helppoa. Nämä ominaisuudet yhdistettynä hyvään suorituskykyyn tekevät MongoDB:stä erinomaisen valinnan esimerkiksi prototyyppiprojekteihin, erityisesti jos käyttäjille JSON-mallinen data on ennestään tuttua. Lisäksi sekä MongoDB- että MySQL-tietokannoille on laaja yhteisö ja kattava dokumentaatio WWW-sivuilla, joten apua ongelmatilanteisiin on helppo löytää.

Vaikka MongoDB vaikuttaakin tämän tutkielman pohjalta edottomasti paremmalta valinnalta MySQL-tietokantaan verrattuna, on tärkeää muistaa, että molemmat tietokannat edustavat erilaisten tietokantojen arkkityyppejä. Mikäli datan eheydestä ei voida tinkiä, ja tallennettavan datan muoto on tiedossa sekä tiukasti toisiinsa sidoksissa (relaation linkitykset), on relaatiotietokanta todennäköisesti parempi vaihtoehto. Jos datan ei tarvitse jokaisella tietokantakyselyllä olla täysin ajantasaista ja sen tilalle halutaan korkeaa suorituskykyä sekä tietokannan helppokäyttöisyyttä, voi NoSQL-tietokanta tarjota silloin enemmän.

Kokemukset Yahoo! Cloud Service Benchmarkista työkaluna ovat hieman ristiriitaiset. Työkalua oli helppo käyttää, suorituskykytestien konfigurointi oli vaivatonta sekä mukana tulee useita implementaatioita eri tietokantojen testaamiseen. Kuitenkin työkalussa on keskeneräisyyden tunne. Lisäksi vaikuttaisi, että järjestelmällinen kehitys ja versioiden julkaiseminen on hidastunut. Viimeisin julkaisu, jota tässäkin tutkielmassa käytettiin, on julkaistu joulu-

kuussa 2016. Uusi julkaisu näyttäisi olevan kuitenkin tulossa lähiaikoina, sillä tutkielman kirjoittamishetkellä julkaisuehdokas on ainakin kasattu wiki-sivujen mukaan.[1]

YCSB:n generoimat suorituskykytestit ovat myös varsin synteettisiä, joka laskee tulosten arvoa ja paikkaansapitävyyttä verrattaessa oikeisiin systeemeihin ja niiden käyttötarkoituksiin. YCSB on pääasiassa suunniteltu avain-arvo- ja NoSQL-tietokantojen suorituskykyjen testaamiseen, joten perinteisiä relaatiotietokantoja tällä työkalulla ei kannata testata. Lisäksi on syytä huomioida, että myös NoSQL-tietokantojen toteutuksissa on paljon eroja. Erot yhdistettynä synteettisiin suorituskykytesteihin eivät mielestäni anna parasta mahdollista pohjaa reilulle suorituskykyjen vertailulle.

Nään kuitenkin YCSB:ssä paljon potentiaalia ja toivon, että sen kehitystä jatketaan aktiivisesti. YCSB on tapa mitata tietokantojen suorituskykyä helposti ja edes jotenkin standardoidulla tavalla. YCSB:ssä on myös helposti implementoitava rajapinta, joten uusien tietokantojen lisääminen ja testaaminen on helppoa. Tämä on erityisen hyödyllistä tietäen, kuinka paljon eri tietokantoja nykyään on olemassa.

1. https://github.com/brianfrankcooper/YCSB/releases

8 Yhteenveto

Tässä pro gradu -tutkielmassa käsiteltiin relaatio- ja NoSQL-tietokantojen eroavaisuuksia. Erityisesti tutkielmassa tutkittiin näiden tietokantojen suorituskykyeroja. Luvuissa 2 ja 3 käsiteltiin niin relaatio- kuin NoSQL-tietokantojen perusteita, taustaa ja eroavaisuuksia. Luku 4 käsitteli tietokantojen suorituskyvyn teoriaa ja yleisiä suorituskyvyn mittaamiseen liittyviä käytänteitä. Aiheet ovat laajoja, mutta tässä yhteydessä tarkoituksena oli antaa lukijalle tarvittava yleisnäkyms aiheesta, jotta tutkielman loppuosan ymmärtäminen on mahdollista.

Tutkielman tutkimusosuus, luvut 5-7, koostuivat relaatio- ja NoSQL-tietokantojen suorituskykyjen vertailusta. Eri valmistajien tarjoamien tietokantojen suuren määrän vuoksi tarkasteltavat tietokannat rajattiin yhteen molemmista tietokantatyypeistä. Relaatiotietokantaa edusti MySQL-tietokanta, ja NoSQL-tietokantaa edusti MongoDB-tietokanta. Kyseiset tietokannat ovat erittäin suosittuja, joten niistä löytyi hyvin dokumentaatiota sekä keskustelua WWW-sivuilta. Lisäksi tulokset ovat todennäköisesti mielenkiintoisempia suuremmalle yleisölle edellä mainituista syistä.

Suorituskykytesteihin käytettiin Yahoo! Cloud Service Benchmark -työkalua, joka on avoimen lähdekoodin projekti. Työkalu on suunniteltu erilaisten avain-arvo -tietokantojen testaamiseen, mutta siihen on myös toteutettu implementaatiot useille erityyppisille tietokannoille, kuten esimerkiksi relaatiotietokannoille.

Suorituskykytestien tulokset olivat odotetut. MongoDB suoriutui huomattavasti paremmin MySQL-tietokantaan verrattuna. Lisäksi käyttökokemukset MongoDB-tietokannasta olivat positiiviset. MongoDB- ja MySQL-tietokantojen tuloksissa oli myös yhteneväisyyksiä. Molemmilla tietokannoilla kirjoitusoperaatioiden suorittaminen oli raskaampaa lukuoperaatioihin verrattuna. Erityisesti MySQL-tietokannan kohdalla ero oli huomattavasti suurempi.

Tuloksia on kuitenkin syytä muistaa tarkkailla kriittisesti. Tuloksiin vaikuttavat useat asiat kuten testiympäristön ohjelmistot, laitteisto ja konfiguraatiot. Testattujen tietokantojen koot olivat myös hyvin pieniä verrattuna oikeiin systeemeihin. Tuloksia ei kannatakaan ottaa absoluuttisena totuutena, vaan käyttää osana muita lähteitä kartoittaessa eri tietokantojen suorituskykyä.

Jatkossa olisi mielenkiintoista toteuttaa samoille tietokannoille suorituskykytestit käyttäen erilaista testityökalua tai -tekniikkaa, kuten esimerkiksi TPC-organisaation standardoimaa suorituskykytestiä. Tässä tutkielmassa tutkittiin suorituskykyä vain perusoperaatioiden suorittamisen kannalta. Lisäksi tutkimusasetelma oli MongoDB-tietokannan eduksi. Monimutkaisemmat operaatiot ja MySQL-tietokantaa suosiva tutkimusasetelma voisivat antaa erilaisia tuloksia, ja näitä tuloksia yhdistelemällä testattujen tietokantojen todellisesta suorituskyvystä saisi kattavamman kuvan.

Toinen mielenkiintoinen näkökulma olisi yrittää parantaa tietokantojen suorituskykyä erilaisilla konfiguraatioilla kuten esimerkiksi indeksoinnilla. Tuloksia vertaamalla tämän tutkielman tuloksiin nähtäisiin, kuinka tärkeää tietokantojen oikeaoppinen konfigurointi on. Lisäksi voitaisiin saada suuntaa antava käsitys, kuinka paljon kyseisten tietokantojen suorituskykyä on mahdollista parantaa out-of-the-box -suorituskykyyn nähden.

Tämä tutkielma on kontribuutio jo lukuisten suorituskykytestien ja -tutkimusten joukossa. Suorituskyvyn mittaaminen ja tutkiminen on käytännössä loputon kaivo, sillä erilaisia testiasetelmia voidaan rakentaa lukemattomia määriä. Suorituskykyä voidaan myös lähestyä monelta kannalta, joten ei ole olemassa absoluuttista vastausta kysymykseen *mikä on tehokkain tietokanta*.

Lähteet

Bachman, Charles W. 1973. “The Programmer As Navigator”. *Commun. ACM* (New York, NY, USA) 16, numero 11 (marraskuu): 653–658. ISSN: 0001-0782. doi:10.1145/355611.362534. http://doi.acm.org/10.1145/355611.362534.

Bitton, Dina, Mark Brown, Rick Catell, Stefano Ceri, Tim Chou, Dave DeWitt, Dieter Gawlick ym. 1985. “A Measure of Transaction Processing Power”. *Datamation* (Newton, MA, USA) 31, numero 7 (huhtikuu): 112–118. ISSN: 0011-6963. http://dl.acm.org/citation.cfm?id=13900.18159.

Blackburn, Stephen M. et al, Amer Diwan, Matthias Hauswirth, Peter F. Sweeney, Jose Nelson Amaral, Tim Brecht, Lubomir Bulej ym. 2016. “The Truth, The Whole Truth, and Nothing But the Truth: A Pragmatic Guide to Assessing Empirical Evaluations”. *ACM Trans. Program. Lang. Syst.* (New York, NY, USA) 38, numero 4 (lokakuu): 15:1–15:20. ISSN: 0164-0925. doi:10.1145/2983574. http://doi.acm.org/10.1145/2983574.

Boza, E. F., C. San-Lucas, C. L. Abad ja J. A. Viteri. 2017. “Benchmarking Key-Value Stores via Trace Replay”. Teoksessa *2017 IEEE International Conference on Cloud Engineering (IC2E),* 183–189. Huhtikuu. doi:10.1109/IC2E.2017.11.

Carter, Breck. 2018. “Bottleneck: Latency Versus Throughput”. Viitattu 1. huhtikuuta. http://sqlanywhere.blogspot.fi/2011/11/bottleneck-latency-versus-throughput.html.

CData. 2017. “Comparison of JDBC Drivers for MySQL”. Viitattu 7. marraskuuta. https://www.cdata.com/kb/articles/jdbc-mysql-comparison.rst.

Chamberlin, Donald D., ja Raymond F. Boyce. 1974. “SEQUEL: A Structured English Query Language”. Teoksessa *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control,* 249–264. SIGFIDET ’74. Ann Arbor, Michigan: ACM. doi:10.1145/800296.811515. http://doi.acm.org/10.1145/800296.811515.

Codd, E. F. 1970. “A Relational Model of Data for Large Shared Data Banks”. *Commun. ACM* (New York, NY, USA) 13, numero 6 (kesäkuu): 377–387. ISSN: 0001-0782. doi:10.1145/362384.362685. http://doi.acm.org/10.1145/362384.362685.

———. 1990. *The Relational Model for Database Management: Version 2.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-201-14192-2.

Cooper, Brian F., Adam Silberstein, Erwin Tam, Raghu Ramakrishnan ja Russell Sears. 2010. “Benchmarking Cloud Serving Systems with YCSB”. Teoksessa *Proceedings of the 1st ACM Symposium on Cloud Computing,* 143–154. SoCC ’10. Indianapolis, Indiana, USA: ACM. ISBN: 978-1-4503-0036-0. doi:10.1145/1807128.1807152. http://doi.acm.org/10.1145/1807128.1807152.

DB-Engines. 2017. “DB-Engines Ranking”. Viitattu 19. lokakuuta. https://db-engines.com/en/ranking.

Gilbert, Seth, ja Nancy Lynch. 2002. “Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services”. *SIGACT News* (New York, NY, USA) 33, numero 2 (kesäkuu): 51–59. ISSN: 0163-5700. doi:10.1145/564585.564601. http://doi.acm.org/10.1145/564585.564601.

Gray, Jim. 1981. “The Transaction Concept: Virtues and Limitations (Invited Paper)”. Teoksessa *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7,* 144–154. VLDB ’81. Cannes, France: VLDB Endowment. http://dl.acm.org/citation.cfm?id=1286831.1286846.

———. 1992. *Benchmark Handbook: For Database and Transaction Processing Systems.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 1558601597.

Gray, Jim, ja Andreas Reuter. 1992. *Transaction Processing: Concepts and Techniques.* 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 1558601902.

Haerder, Theo, ja Andreas Reuter. 1983. “Principles of Transaction-oriented Database Recovery”. *ACM Comput. Surv.* (New York, NY, USA) 15, numero 4 (joulukuu): 287–317. ISSN: 0360-0300. doi:10.1145/289.291. http://doi.acm.org/10.1145/289.291.

Haverinen, Henri. 2016. “SQL- ja NoSQL-tietokantojen suorituskykyerot”. Kandidaatintutkielma, Jyväskylän yliopisto, informaatioteknologian tiedekunta. http://urn.fi/URN:NBN:fi:jyu-201605032410.

Hazelhurst, Scott. 2010. “Truth in advertising: Reporting performance of computer programs, algorithms and the impact of architecture”. Teoksessa *South African Computer Journal 46,* 24–37. Joulukuu. doi:10.18489/sacj.v46i0.50.

Hussain, Ali. 2018. “Little’s Law- An insight on the relation between latency and throughput”. Viitattu 1. huhtikuuta. http://blog.flux7.com/blogs/benchmarks/littles-law.

Kitchenham, B., ja S Charters. 2007. *Guidelines for performing Systematic Literature Reviews in Software Engineering*.

Kopp, Michael. 2018. “Why Averages Suck and Percentiles are Great”. Viitattu 1. huhtikuuta. https://www.dynatrace.com/news/blog/why-averages-suck-and-percentiles-are-great/.

Leavitt, N. 2010. “Will NoSQL Databases Live Up to Their Promise?” *Computer* 43, numero 2 (helmikuu): 12–14. ISSN: 0018-9162. doi:10.1109/MC.2010.58.

Logicalread. 2018. “Response Time Analysis: How to Improve Database Performance by Measuring User Experience”. Viitattu 1. huhtikuuta. https://logicalread.com/response-time-analysis/#.WsIzyZdRUuU.

Merriam-Webster. 2017. “Database | Definition of database by Merriam-Webster”. Viitattu 6. marraskuuta. https://www.merriam-webster.com/dictionary/database.

———. 2018. “Throughput | Definition of Throughput by Merriam-Webster”. Viitattu 1. huhtikuuta. https://www.merriam-webster.com/dictionary/throughput.

NoSQL database. 2017. “NoSQL Databases”. Viitattu 29. syyskuuta. http://nosql-database.org.

O’Neil, Patrick. 1994. *Database Systems: Principles, Programming, Performance.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 1-55860-219-4.

Oracle. 2018. “Transaction throughput”. Viitattu 1. huhtikuuta. https://docs.oracle.com/cd/E17276_01/html/programmer_reference/transapp_throughput.html.

Oracle Corporation and/or its affiliates. 2017. “Chapter 19 Using MySQL as a Document Store”. Viitattu 19. elokuuta. https://dev.mysql.com/doc/refman/5.7/en/document-store.html.

Oxford English Dictionary. 2017. “database, n. : Oxford English Dictionary”. Viitattu 6. marraskuuta. http://www.oed.com/view/Entry/47411.

Packet-foo. 2018. “How millisecond delays may kill database performance”. Viitattu 1. huhtikuuta. https://blog.packet-foo.com/2014/09/how-millisecond-delays-may-kill-database-performance/.

Pandey, Vaibhaw. 2018. “How to Benchmark MongoDB with YCSB?” Viitattu 1. huhtikuuta. https://scalegrid.io/blog/how-to-benchmark-mongodb-with-ycsb/.

Petkovic, Dusan. 2018. “How System Resources Affect SQL Server Performance”. Viitattu 1. huhtikuuta. https://logicalread.com/system-resources-affect-sql-server-performance-mc03/#.WsuKwZdRUuU.

Pritchett, Dan. 2008. “BASE: An Acid Alternative”. *Queue* (New York, NY, USA) 6, numero 3 (toukokuu): 48–55. ISSN: 1542-7730. doi:10.1145/1394127.1394128. http://doi.acm.org/10.1145/1394127.1394128.

Rapa, Antti. 2016. “Relaatio- ja epärelaatiotietokantojen suorituskykyvertailu : MySQL ja MongoDB”. Pro gradu -tutkielma, Jyväskylän yliopisto, informaatioteknologian tiedekunta. http://urn.fi/URN:NBN:fi:jyu-201606173190.

Reniers, Vincent, Dimitri Van Landuyt, Ansar Rafique ja Wouter Joosen. 2017. “On the State of NoSQL Benchmarks”. Teoksessa *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion,* 107–112. ICPE ’17 Companion. L’Aquila, Italy: ACM. ISBN: 978-1-4503-4899-7. doi:10.1145/3053600.3053622. http://doi.acm.org/10.1145/3053600.3053622.

Schwartz, Baron. 2018. “The four fundamental performance metrics”. Viitattu 1. huhtikuuta. https://www.percona.com/blog/2011/04/27/the-four-fundamental-performance-metrics/.

Sigplan. 2018. “Empirical Evaluation Guidelines”. Viitattu 8. huhtikuuta. http://www.sigplan.org/Resources/EmpiricalEvaluation/.

Smartbear. 2018. “Measuring SQL Server Performance”. Viitattu 1. huhtikuuta. https://support.smartbear.com/loadcomplete/docs/use-cases/measuring-sql-server-performance.html.

Statistics How To. 2018. “Percentiles, Percentile Rank & Percentile Range: Definition & Examples”. Viitattu 1. huhtikuuta. http://www.statisticshowto.com/probability-and-statistics/percentiles-rank-range/.

Strauch, Christof, Ultra-Large S. Sites ja Walter Kriha. 2011. “NoSQL databases”. http://www.christof-strauch.de/nosqldbs.pdf.

Tang, E., ja Y. Fan. 2016. “Performance Comparison between Five NoSQL Databases”. Teoksessa *2016 7th International Conference on Cloud Computing and Big Data (CCBD),* 105–109. Marraskuu. doi:10.1109/CCBD.2016.030.

Tong, Zachary. 2018. “Averages Can Be Misleading: Try a Percentile”. Viitattu 1. huhtikuuta. https://www.elastic.co/blog/averages-can-dangerous-use-percentile.

TPC. 2017. “TPC Homepage”. Viitattu 1. marraskuuta. http://www.tpc.org.

Treat, Tyler. 2018. “Everything You Know About Latency Is Wrong”. Viitattu 1. huhtikuuta. https://bravenewgeek.com/everything-you-know-about-latency-is-wrong/.

Vogels, Werner. 2009. “Eventually Consistent”. *Commun. ACM* (New York, NY, USA) 52, numero 1 (tammikuu): 40–44. ISSN: 0001-0782. doi:10.1145/1435417.1435432. http://doi.acm.org/10.1145/1435417.1435432.

Liitteet

A Suorituskykytesti 1 tulokset

Asiakas-säikeiden määrä	MySQL		MongoDB	
	pieni tietokanta	**iso tietokanta**	**pieni tietokanta**	**iso tietokanta**
1	3025	435	5534	4899
2	2986	810	18386	17705
3	2963	798	23154	21681
4	3141	797	25706	24359
5	2962	790	26502	24652
6	2908	806	26136	24480
7	2991	780	26930	24904
8	2982	807	27142	24813
9	2833	799	26410	24911
10	2843	806	26419	24649
11	2919	783	26580	24353
12	2916	782	26390	24379

Taulukko 5. Suorituskykytesti 1 tulokset (suoritusteho, operaatioita / sekunti)

B Suorituskykytesti 2 tulokset

Asiakas-säikeiden määrä	MySQL		MongoDB	
	pieni tietokanta	**iso tietokanta**	**pieni tietokanta**	**iso tietokanta**
1	3439	1506	7011	5765
2	3845	1829	23479	23026
3	3985	1866	28670	27917
4	4075	2112	31762	30532
5	4267	1917	31379	30411
6	4497	2184	32332	31243
7	4198	2064	32857	32148
8	4341	2106	33279	31955
9	4434	2150	33393	32054
10	4061	2114	33093	31559
11	4053	2071	32756	32081
12	4220	2031	32419	31268

Taulukko 6. Suorituskykytesti 2 tulokset (suoritusteho, operaatioita / sekunti)

C Suorituskykytesti 3 tulokset

Asiakas-säikeiden määrä	MySQL		MongoDB	
	pieni tietokanta	**iso tietokanta**	**pieni tietokanta**	**iso tietokanta**
1	4132	3399	6350	6846
2	12378	12104	24843	24637
3	15055	14760	30245	30003
4	17284	16826	33602	33131
5	17081	16832	32616	32722
6	16656	16819	34063	33342
7	16883	16787	35153	34879
8	16307	16598	35140	34741
9	16507	16305	35535	34724
10	16289	16367	35186	34632
11	16445	16036	35174	34827
12	16038	16033	35021	34840

Taulukko 7. Suorituskykytesti 3 tulokset (suoritusteho, operaatioita / sekunti)

D Suorituskykytesti 4 tulokset

Asiakas-säikeiden määrä	MySQL		MongoDB	
	pieni tietokanta	**iso tietokanta**	**pieni tietokanta**	**iso tietokanta**
1	2942	356	4662	5049
2	2702	640	15712	14982
3	2814	637	18813	17807
4	2797	627	21321	19930
5	2833	640	21199	19379
6	2766	667	21345	19572
7	2780	634	21423	20457
8	2827	615	21741	20941
9	2721	626	21653	19947
10	2740	636	21552	19177
11	2750	614	21367	19492
12	2737	617	21130	19255

Taulukko 8. Suorituskykytesti 4 tulokset (suoritusteho, operaatioita / sekunti)