Eskelinen Juuso-Julius

TIETOKANNAN REPLIKAATIO YKSISUUNTAISIA EI-PYSYVIÄ YHTEYKSIÄ KÄYTTÄEN

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2019

TIIVISTELMÄ

Eskelinen, Juuso-Julius
Tietokannan replikaatio yksisuuntaisia ei-pysyviä yhteyksiä käyttäen
Jyväskylä: Jyväskylän yliopisto, 2019, 61 s.
Tietojärjestelmätiede, Pro gradu -tutkielma
Ohjaajat: Taipalus, Toni & Seppänen, Ville

Tässä pro gradu -tutkielmassa esitetään suunnittelumalli hajautetun tietokannan replikaation toteuttamiseksi käyttäen yksisuuntaisia ei-pysyviä yhteyksiä. Tämän mallin pohjalta on myös tehty replikaation toteutus toimeksiantona Patria Aviation Oy:lle. Replikaation toteutuksessa on otettu huomioon se, että tuloksena syntynyt malli on yhteensopiva erilaisten tietokannanhallintajärjestelmien kanssa, eikä nojaa minkään tietyn tuotteen replikaatio-ominaisuuksiin. Suunnittelua varten on tehty kirjallisuuskatsaus, jossa selvitettiin hajautetun tietokannan suunnitteluun ja replikaatioon liittyviä haasteita ja ratkaisumalleja. Kirjallisuuskatsauksesta selvisi, että hajautetun tietokannan ja replikaation suunnittelu on tasapainottelua suorituskyvyn, datan saatavuuden ja vikasietoisuuden välillä. Tämän lisäksi toimeksiantona toteutetun replikaation rajattu toimintaympäristö asettaa myös useita haasteita replikointistrategian valitsemiselle, sillä yhteyksien yksisuuntaisuuden ja muuttuvan luonteen takia pessimistiset replikaatiostrategiat eivät ole järkevästi toteutettavissa. Tästä syystä tutkimuksen tuloksena luotiin suunnittelumalli, johon on otettu piirteitä aiemmassa kirjallisuudessa käsitellyistä optimistisista replikaatiostrategioista. Mallissa replikaatiotoiminnallisuus on jaettu kahteen osa-alueeseen, joista toisen tehtävänä on jakaa päivitykset, ja toisen korjata mahdollisia puutteita ja virheitä datassa solmujen välillä. Toteutuksen testaus osoitti, että mallin pohjalta replikaatio voidaan toteuttaa kohdejärjestelmään ja se täyttää sille asetetut vaatimukset.

Asiasanat: tietokanta, hajautettu tietokanta, replikaatio, yksisuuntainen yhteys

ABSTRACT

Eskelinen, Juuso-Julius
Database replication using unidirectional temporary connections
Jyväskylä: University of Jyväskylä, 2019, 61 pp.
Information Systems, Master's Thesis
Supervisors: Taipalus, Toni & Seppänen, Ville

The purpose of this master's thesis is to design a model and implement database replication functionality that can be used with unidirectional and temporary connections. This thesis is made for Patria Aviation Oy. Replication of data between nodes in the system is designed the way that it isn't dependent of any particular database management system. All functionality needed for replication is implemented at the application using a database, which makes it possible to create a solution that can be used with a variety of different technologies. A literature review was also made to find out what kind of issues have been raised in the past literature and how they have been solved. The literature review revealed that designing a distributed database and its replication means that compromises must be made between performance, availability of data and fault tolerance of the system. The need to implement replication by using unidirectional temporary connections sets many challenges for the design. Because of these challenges, most of the pessimistic replication methods are not feasible, so the design has taken many characteristics of optimistic replication strategies that has been studied in the past literature. Replication functionality in this model is divided to continuous background replication and event-based replication. Tests confirm that the replication functionality, based on the design model described in this study, can be implemented by using unidirectional temporary connections.

Keywords: database, distributed database, replication, unidirectional connection

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Tietokannat ovat sekä teknisesti että liiketoiminnallisesti kriittinen osa lähes jokaista tietojärjestelmää. On yhä yleisempää, että tietojärjestelmän tietokanta on hajautettu, eli tietokanta sijaitsee maantieteellisesti useammassa solmussa. Hajautus mahdollistaa järjestelmän paremman vikasietoisuuden ja siihen kohdistuvan kuormituksen jakamisen (Özsu & Valduriez, 2011; Rahimi & Haug, 2010). Yksi hajautustekniikoista on tietokannan sisällön kopioiminen eli replikointi, jonka mukaisesti dataa kopioidaan ainakin yhteen muuhun solmuun datan alkuperäisen sijainnin lisäksi.

Tietokantojen hajauttamista ja replikointia on tutkittu runsaasti, ja monet perustavanlaatuisista tutkimuksista on tehty jo useampia vuosikymmeniä sitten (Bernstein & Goodman, 1981; Özsu & Valduriez, 1991). Aikaisempi tutkimus ei ota huomioon 2000-luvulla yleistyneiden ja kehittyneiden hajautustekniikoiden vaikutuksia hajautuksen suunnitteluun. Lisäksi ympäristöihin, joissa tietokantojen hajautusta suunnitellaan ja toteutetaan, voi liittyä rajoittavia tekijöitä, joita aikaisemmassa tutkimuksessa ei ole otettu huomioon.

Tämän tutkimuksen tavoitteena on luoda toimeksiantona Patria Aviation Oy:lle hajautetun tietokannan replikointiin ratkaisu, jonka tulee kyetä toteuttamaan replikaatio järjestelmän solmujen välillä. Hajautuksen kannalta ratkaisuun liittyy kaksi rajoittavaa tekijää. Ensinnäkin järjestelmän yhteydet ovat yksisuuntaisia. Yksisuuntaisuudella tarkoitetaan sellaista yhteyttä kahden dataa sisältävän solmun välillä, jossa dataa replikoidaan vain yhteen suuntaan (esim. solmulta A solmulle B, mutta ei päinvastoin). Toisekseen järjestelmän yhteydet ovat ei-pysyviä, jolla tarkoitetaan, että yhteydet dataa sisältävien solmujen välillä ovat usein käyttämättömissä. Siinä missä yhteyksien katkeaminen on tavallisesti poikkeustilanne, johon hajautusta suunniteltaessa varaudutaan, ovat ei-pysyvät yhteydet tämän tutkimuksen käsittelemässä järjestelmässä pikemminkin normaalitilanne.

Tämän tutkimuksen tavoite on luoda suunnittelumalli tietokannan replikoimiseksi yllä kuvatussa ympäristössä. Suunnittelumalli on yleistettävissä relaatiotietokantoja käyttäville tuotteille. Lisäksi tässä tutkielmassa toteutetaan

toimeksiantajan tarpeiden perusteella tietokannan replikointi luotua suunnittelumallia käyttäen.

Luvuissa 2 ja 3 käsitellään kirjallisuuskatsauksesta esille nousseita haasteita ja tapoja toteuttaa hajautettu tietokanta ja datan replikaatio hajautetussa tietokannassa. Ensin luvussa 2 käsitellään hajautettuja tietokantoja, hajautetun tietokannan eri suunnittelumalleja, niistä saavutettavia etuja sekä datan hajauttamiseen liittyviä ongelmia. Seuraavaksi luvussa 3 esitellään yleisimpiä tapoja toteuttaa replikaatio hajautetun tietokannan solmujen välillä, sekä mitä haasteita näihin eri tapoihin ja replikaatioon ylipäänsä liittyy. Luvussa 4 taas pohditaan aiemmissa luvuissa esitettyjen kirjallisuuteen pohjaavien asioiden pohjalta mitä tulisi ottaa huomioon hajautetun tietokannan ja replikaation suunnittelussa. Luku 5 esittelee tutkimusmenetelmät ja tutkimuksen tavoitteet tarkemmin. Lisäksi käydään läpi tutkimuksen tuloksena syntyvän toteutuksen testausmenetelmät ja -ympäristö. Tämän jälkeen luvussa 6 esitellään tutkimuksen tulokset, joista ensimmäisenä kuvataan luotu replikaation suunnittelumalli. Lisäksi luvussa 6 kuvataan varsinainen toimeksiantona tehty toteutus replikaatiosta, sen testaus ja testien tulokset. Luvussa 7 pohditaan tutkielman tuloksia, jonka jälkeen luvussa 8 esitetään tutkielman rajoitteet. Luku 9 esittää lyhyen yhteenvedon tutkielmasta.

2 HAJAUTETTU TIETOKANTA

Hajautetussa tietokannassa data on fyysisesti hajautettu useamman solmun kesken ja tyypillisesti jokaisessa solmussa on oma tietokannanhallintajärjestelmänsä (Silberschatz, Korth, & Sudarshan, 2001). Tässä ja seuraavassa luvussa käsitellään aiemman kirjallisuuden pohjalta hajautettuja tietokantoja ja niiden replikointia. Datan hajauttamisella järjestelmän eri solmuihin tavoitellaan parempaa datan saatavuutta, vikasietoisuutta sekä tehokkaampia kyselyjä (Rahimi & Haug, 2010). Hajautetut tietokannat ovat loogisesti sidoksissa toisiinsa ja kommunikoivat keskenään verkon ylitse (Özsu & Valduriez, 2011). Seuraavissa alaluvuissa käsitellään lyhyesti hajautetun tietokannan eri suunnittelumalleja, hajautetun tietokannan etuja keskitettyyn tietokantaan verrattuna sekä hajautettuun tietokantaan liittyviä ongelmia.

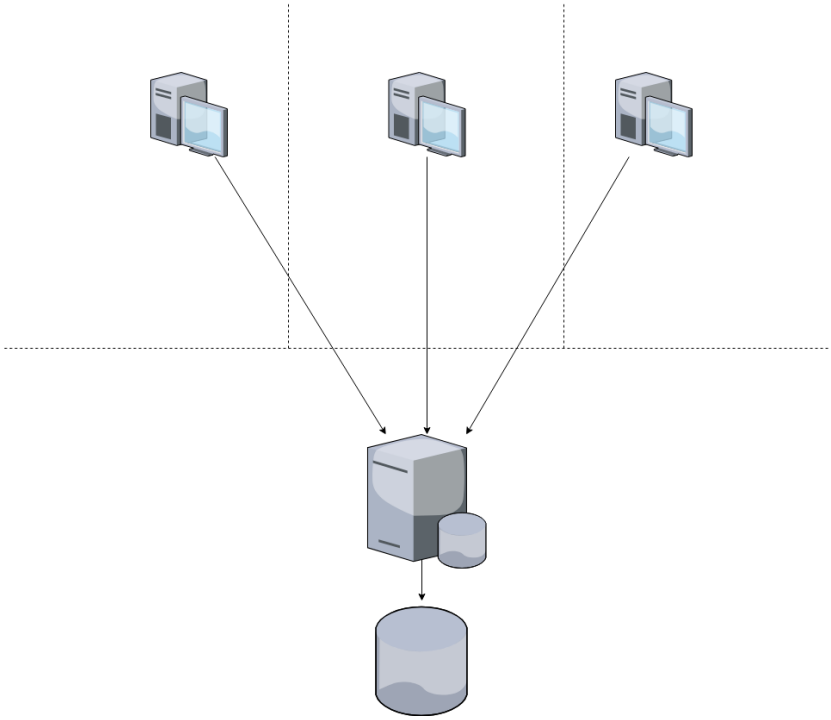
2.1 Datan sijoitus keskitetyssä ja hajautetuissa tietokanta-arkkitehtuurimalleissa

Seuraavissa alaluvuissa esitellään kirjallisuuden pohjalta neljä yleistä tietokanta-arkkitehtuurimallia, joissa datan sijoitus järjestelmän solmujen välillä on tehty eri tavoilla. Rahimi ja Haug (2010) toteavat, että tietokannan voidaan sanoa olevan hajautettu, jos kaikkea sen dataa ei säilytetä yhdessä järjestelmän solmussa. Datan sijoittaminen hajautetussa tietokannassa on tärkeää, sillä datan keskittämisessä, osittamisessa tai hajauttamisessa joudutaan tekemään kompromisseja datan saatavuuden, datan yhteneväisyyden, vikasietoisuuden ja järjestelmän suorituskyvyn välillä, sillä eri datan sijoitusmalleista saadaan erilaisia hyötyjä, mutta myös joudutaan luopumaan tietyistä ominaisuuksista.

2.1.1 Keskitetty tietokanta-arkkitehtuuri

Keskitetyssä tietokanta-arkkitehtuurissa data on yleensä sijoitettu yhteen paikkaan ja sen prosessointi tehdään keskitetysti (Chen, Ng, & Greenfield, 2013).

Kaikki järjestelmän eri solmut tekevät kyselynsä keskitetylle tietokannanhallintajärjestelmälle (Özsu & Valduriez, 2011). Keskitetyssä mallissa kaikki muutokset tehdään samaan tietokantaan, joten tiedon eheyden takaaminen on helpompaa (Eager & Sevcik, 1983), sillä operaatiot voidaan sarjallistaa ja näin välttyä konfliktitilanteilta. Kuitenkin kaikkien pyyntöjen välittäminen yhteen keskitettyyn tietokanta- ja prosessointiyksikköön ei ole aina tehokasta (Özsu & Valduriez, 2011). KUVIO 1 Keskitetty tietokanta-arkkitehtuurikuvastaa keskitettyä tietokantajärjestelmää, jossa kaikki pyynnöt välitetään keskitetylle tietokantaa hallitsevalle komponentille. Vaikka tietokantaan pyyntöjä tekevät asiakkaat tai järjestelmän solmut voivat toimia hajautetusti ja kommunikoida keskitetyn tietokannanhallintajärjestelmän kanssa verkon yli, ei kyseessä kuitenkaan ole hajautettu tietokanta, sillä yksi keskitetty solmu hoitaa tiedon varastoinnin, -



prosessoinnin sekä kyselyihin vastaamisen.

KUVIO 1 Keskitetty tietokanta-arkkitehtuuri

2.1.2 Osioitu tietokanta-arkkitehtuuri

Osioidussa tietokanta-arkkitehtuurissa data on jaettu useamman fyysisen solmun kesken (Navathe, Ceri, Wiederhold, & Dou, 1984). Datan jako on toteutettu siten, että kaikki järjestelmän solmut ylläpitävät ja varastoivat eri dataa (Chen et al., 2013). Data voidaan osittaa jakamalla eri solmuille niiden pääasiassa tarvitsemat taulut tai fragmentoida taulut pienempiin osiin.

Tauluja voidaan fragmentoida vertikaalisesti, horisontaalisesti tai näiden yhdistelmällä (Rahimi & Haug, 2010). Vertikaalisessa fragmentoinnissa taulusta

koostetaan useampia pienempiä kokonaisuuksia ylläpitäviä tauluja, jotka voidaan myöhemmin tarvittaessa koostaa takaisin alkuperäiseen muotoonsa (Rahimi & Haug, 2010). TAULUKKO 1 esitellään esimerkkinä työntekijätaulu, joka on TAULUKKO 2 ja TAULUKKO 3 fragmentoitu vertikaalisesti jakamalla taulu työntekijän palkkatauluun ja työntekijän muihin tietoihin. Esimerkki vastaavasta käyttötapauksesta voisi olla tilanne, jossa pääkonttorissa sijaitseva palkanlaskenta tarvitsee vain työntekijän tunnuksen ja palkan, kun taas yrityksen hallinnossa tarvitaan vain työntekijän muita tietoja, mutta ei palkkatietoja.

HenkilöNo.	Nimi	Toimipiste	Palkka	Syntymäaika	Titteli
12415	Matti	Helsinki	4200,00	14.5.1965	Ohjelmistokehittäjä
92751	Maija	Helsinki	4200,00	11.3.1982	Ohjelmistokehittäjä
28751	Pertti	Tampere	4100,00	24.9.1956	Projektipäällikkö
51212	Milla	Tampere	4400,00	21.3.1977	Ohjelmistokehittäjä

TAULUKKO 1 Alkuperäinen työntekijätaulu

HenkilöNo.	Palkka
12415	4200,00
92751	4200,00
28751	4100,00
51212	4400,00

TAULUKKO 2 Palkkataulu irrotettuna työntekijätaulusta

HenkilöNo.	Nimi	Toimipiste	Syntymäaika	Titteli
12415	Matti	Helsinki	14.5.1965	Ohjelmistokehittäjä
92751	Maija	Helsinki	11.3.1982	Ohjelmistokehittäjä
28751	Pertti	Tampere	24.9.1956	Projektipäällikkö
51212	Milla	Tampere	21.3.1977	Ohjelmistokehittäjä

TAULUKKO 3 Työntekijän muut tiedot irrotettuna työntekijätaulusta

Horisontaalisessa fragmentoinnissa taulun rivit jaetaan eri solmujen kesken (Rahimi & Haug, 2010). TAULUKKO 4 ja TAULUKKO 5 esitellään horisontaalisesti fragmentoitu työntekijätaulu, jossa fragmentointi on tehty työntekijän toimipisteen mukaan. Tällä tavoin jokaisessa toimipisteessä pidetään yllä kyseiseen toimipisteeseen kuuluvien työntekijöiden kaikkia tietoja, mutta ei tietoa muiden toimipisteiden työntekijöistä.

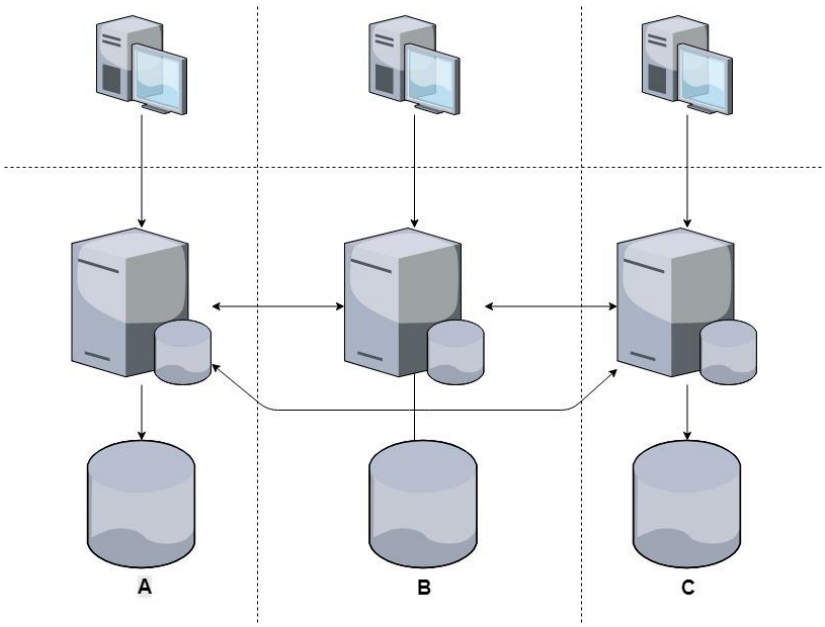
HenkilöNo.	Nimi	Toimipiste	Palkka	Syntymäaika	Titteli
12415	Matti	Helsinki	4200,00	14.5.1965	Ohjelmistokehittäjä
92751	Maija	Helsinki	4200,00	11.3.1982	Ohjelmistokehittäjä

TAULUKKO 4 Toimipisteen mukaan horisontaalisesti fragmentoidun taulun Helsingin toimipisteen fragmentti

HenkilöNo.	Nimi	Toimipiste	Palkka	Syntymäaika	Titteli
28751	Pertti	Tampere	4100,00	24.9.1956	Projektipäällikkö
51212	Milla	Tampere	4400,00	21.3.1977	Ohjelmistokehittäjä

TAULUKKO 5 Toimipisteen mukaan horisontaalisesti fragmentoidun taulun Tampereen toimipisteen fragmentti

Data tulee olla osioitu huolellisesti, jotta järjestelmän suorituskyky voidaan maksimoida (Sacca & Wiederhold, 1985). Jokainen järjestelmän solmu on vastuussa datasta, joka on kyseiselle solmulle relevanttia. Tekemällä ositus huolellisesti, voidaan jakaa kyselyistä aiheutuvaa kuormaa useamman prosessointiyksikön kesken. Osioidussa mallissa jokainen järjestelmän solmu on vastuussa datasta, jota kyseisessä solmussa tarvitaan, ja johon kohdistuu paljon kyselyitä lokaalisti. Tietokantajärjestelmän solmut pyytävät heiltä puuttuvaa dataa tarvittaessa muilta solmuilta, mutta ositus tulisi tehdä siten, että näiden pyyntöjen määrä minimoitaisiin (Chen et al., 2013; Sacca & Wiederhold, 1985). Osioitu tietokanta-arkkitehtuuri voi osoittautua tehokkaaksi lähestymistavaksi, mikäli data on helposti jaettavissa erillisiin kokonaisuuksiin ja suurin osa kyselyistä ei tarvitse dataa muilta järjestelmän solmuilta. KUVIO 2 kuvastaa osioitua tietokanta-arkkitehtuuria. KUVIO 2 tietokanta on jaettu loogisiin kokonaisuuksiin A, B ja C, joista kaikki ovat keskenään erilaisia ja toisistaan riippumattomia.



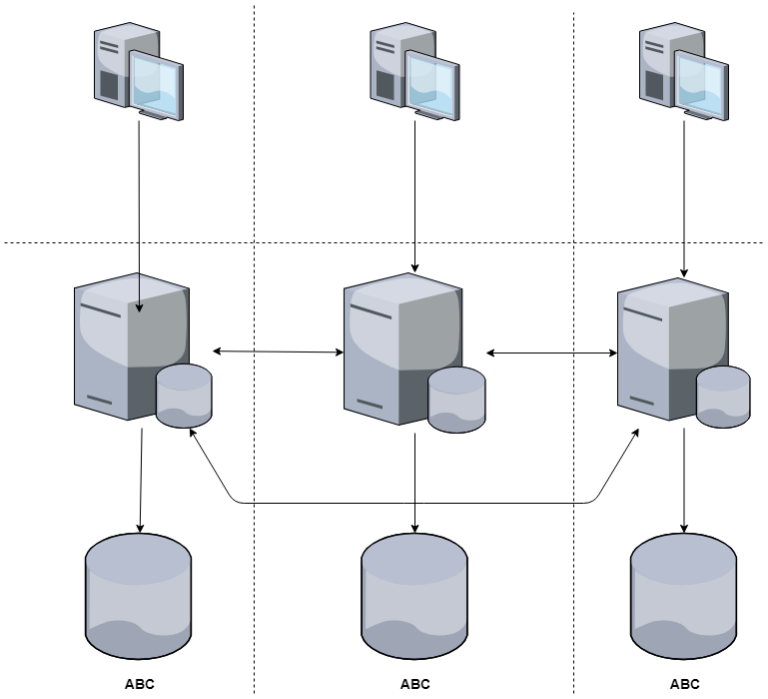
KUVIO 2 Osioitu tietokanta-arkkitehtuuri

2.1.3 P2P-tietokanta-arkkitehtuuri

P2P-tietokanta-arkkitehtuuri (Peer-to-Peer) on terminä moninainen. Sitä on käytetty kuvaamaan erilaisia järjestelmiä, jotka eroavat monelta osin toisistaan (Özsu & Valduriez, 2011). Tässä tutkielmassa P2P-termillä viitattavaan arkkitehtuurimalliin on viitattu myös termeillä replikoitu arkkitehtuuri (RA) (Chen et al., 2013) ja Update Everywhere (Wiesmann, Pedone, Schiper, Kemme, & Alonso, 2000b, 2000a).

P2P-tietokanta-arkkitehtuurilla tarkoitetaankin mallia, jossa kaikki järjestelmän solmut ovat vertaisia keskenään. Vertaisuudella taas tarkoitetaan, että datan päivittäminen ja lukeminen on sallittua kaikista järjestelmän solmuista samanaikaisesti. Kyseisen mallin tärkeimpänä etuna voidaan pitää datan säilyvyyttä verkon tai laitteiston vikatilanteissa, sillä data on fyysisesti tallennettu useampaan järjestelmän eri solmuun (Chen et al., 2013). Malli mahdollistaa kuorman jakamisen solmujen välillä, mutta myös lisää kompleksisuutta eri solmujen datan yhteneväisyyden säilyttämisessä (Özsu & Valduriez, 2011; Wiesmann et al., 2000b). Solmuihin tehdyt päivitykset välitetään muille solmuille, joten kaikki solmut palvelevat asiakkaita paikallisesti kaikella saatavilla olevalla datalla. P2P-arkkitehtuurin tärkeänä hyötynä on myös saatavuus, sillä yhden solmun ollessa poissa käytöstä, pystyvät muut solmut palvelemaan asiakkaita normaalisti (Chen et al., 2013).

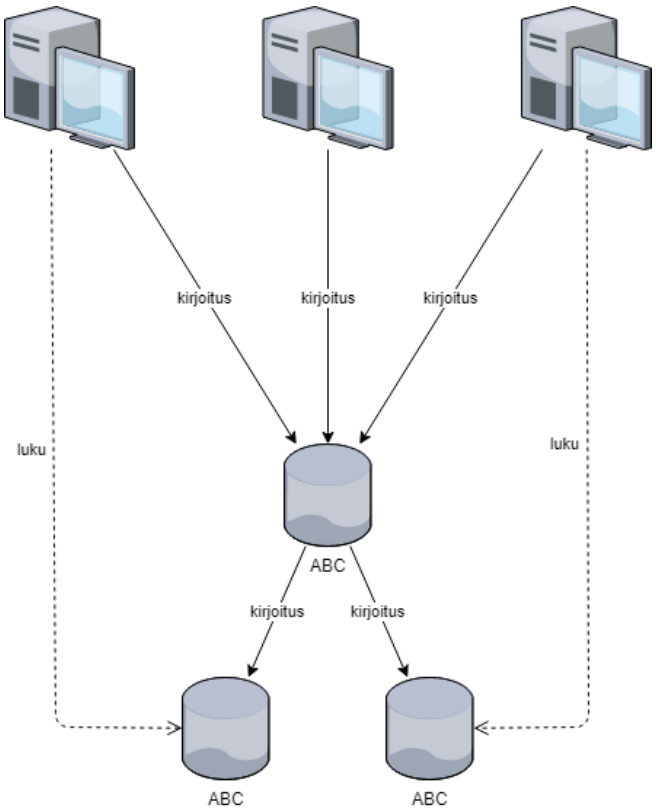
KUVIO 3 kuvaa P2P-arkkitehtuurimallin mukaista hajautetun järjestelmän kokoonpanoa, jossa kaikki tietokannat sisältävät kopion tietueista A, B ja C. Kaikkia näitä tietueita voidaan päivittää ja lukea kaikissa järjestelmän solmuissa ja muutokset kopioon välitetään muille solmuille.



KUVIO 3 P2P-tietokanta-arkkitehtuuri

2.1.4 Master–slave-tietokanta-arkkitehtuuri

Master–slave-tietokanta-arkkitehtuurissa yksi tietokanta toimii pääasiallisena kantana, johon tehdään kaikki päivitykset (Gray, Helland, O'Neil, & Shasha, 1996; Sacha & Dowling, 2007; Wiesmann et al., 2000a, 2000b). Tämän tietokannan vastuulla on jakaa päivitykset muille tietokannoille tietokannan tilan muuttuessa (Sacha & Dowling, 2007). Master–slave -kokoonpanossa konflikteilta eri solmujen välillä voidaan välttyä, sillä kaikki päivitykset tehdään yhteen tietokantaan, joka on vastuussa datan synkronoinnista (Wiesmann et al., 2000a). Tällaisella kokoonpanolla toimivassa järjestelmässä voidaan lukuoperaatiot sallia myös slave-solmuihin, jolloin saadaan suorituskykyä kasvatettua lukuoperaatioiden rinnakkaisella käsittelyllä. Tekemällä päivitykset aina yhteen tietokantaan, voidaankin yksinkertaistaa replikoinnin ja konfliktien hallintaa, mutta samalla myös rajoitetaan kirjoitusoperaatioiden suorituskykyä (Wiesmann et al., 2000b). KUVIO 4 esitellään master–slave-tietokanta-arkkitehtuuria, jossa kaikki päivitykset tehdään master-tietokantaan, joka edelleen jakaa ne slave-tietokannoille. Kaikki tietokannat sisältävät tietueet A, B ja C. KUVIO 4 mallissa lukuoperaatiot ovat sallittuja myös muistakin kuin master-tietokannasta, mutta nämä voivat myös toimia vain varmuuskopioina, jolloin myös lukuoperaatiot tehtäisiin aina master-tietokantaan.



KUVIO 4 Master–slave-tietokanta-arkkitehtuuri

2.2 Syitä hajautetun tietokannan käytölle

Hajautetussa tietokannassa data on hajautettu, jotta saataisiin organisaation kaikki laskentaresurssit hyödynnettyä mahdollisimman tehokkaasti (Rahimi & Haug, 2010). Datan hajauttaminen vaikuttaa koko järjestelmän suorituskykyyn, sillä datan käsittelyyn ja kyselyyn käytetty aika sekä kustannukset vaihtelevat paljon riippuen siitä, miten data on varastoitu (Ceri & Wiederhold, 1987). Datan lokalisointi on yksi hajautetun tietokannan ominaisuus, jolla tarkoitetaan, että data on fyysisesti saatavilla lähempänä sitä järjestelmän solmua, jossa sitä käytetään (Özsu & Valduriez, 2011). Lokalisoinnilla voidaan vähentää etäyhteyksistä johtuvia viiveitä, sillä operaatiot tehdään lokaaliin tai lähimpänä olevaan tietokantaan (Özsu & Valduriez, 2011). Özsu ja Valduriez (2011) toteavat myös, että vaikka verkkoyhteydet ja niiden nopeudet ovat parantuneet huomattavasti, eivät nämä parannukset silti poista maantieteellisestä etäisyydestä aiheutuvia yhteyksien viiveitä, joille tällä hetkellä ei voida mitään. Datan lokalisoinnilla saavutetaan siis vähintään pienempiä viiveitä johtuen vähentyneestä kommunikoinnista verkon ylitse. Abadin (2012) mukaan verkkoliiketoiminnassa viive on kriittinen tekijä. Tietyn tyyppisiin verkkopalveluihin tehdyissä kyselyissä jopa 100 millisekunnin kasvu viiveessä voi laskea huomattavasti todennäköisyyttä sille, että asiakas jatkaa palvelun käyttöä (Abadi, 2012). Tämän lisäksi datan hajauttaminen useaan solmuun lisää rinnakkaisuutta, täten parantaen kokonaisprosessointikykyä, sillä kaikki järjestelmän solmut pystyvät suorittamaan operaatioita samanaikaisesti.

Datan hajauttamisella voidaan myös parantaa koko järjestelmän vikasietoisuutta, sekä turvata datan säilyvyyttä, sillä kun data on varastoitu useampaan kuin yhteen fyysiseen solmuun, on todennäköisyys kaiken datan katoamiselle merkittävästi pienempi (O'Brien et al, 2006; Özsu & Valduriez, 2011). Tilanteessa, jossa järjestelmän yksi solmu on saavuttamattomissa, pystyvät muut solmut silti palvelemaan datalla, joka heillä on saatavilla. Hajautetun tietokannan toteutukseen liittyy myös usein datan replikointi järjestelmän solmuilta toisille, jolloin samaa dataa säilötään useassa fyysisessä sijainnissa ja dataan tehdyt päivitykset välitetään muille kopioille. Luvussa 3 käsitellään syvällisemmin replikaatiota järjestelmän solmujen välillä.

Hajautettu ympäristö tarjoaa myös skaalautuvuutta järjestelmälle. Järjestelmän laajentuessa voidaan yleensä lisätä prosessointi- ja tallennuskapasiteettia yksittäiseen solmuun tai lisätä uusi solmu järjestelmään, jotta pystytään vastamaan laajentumisesta muodostuneisiin prosessoinnin lisätarpeisiin (Özsu & Valduriez, 2011). Keskitetyssä mallissa jouduttaisiin mahdollisesti lisäämään suorituskykyä järjestelmän ainoaan solmuun, joka saattaisi johtaa jopa kyseisen yksikön uusimiseen kokonaan (Özsu & Valduriez, 2011). Jos laajentuminen tapahtuu maantieteellisesti, voidaan lisätä järjestelmään yksi pienemmistä palasista koostuva solmu palvelemaan lisääntynyttä tarvetta. Jos taas laajentuminen tapahtuu operaatioiden määrässä, voidaan päivittää pienempien kokonaisuuksien suorituskykyä siellä missä havaitaan suorituskyky riittämättömäksi. Özsu

ja Valduriez (2011) toteavatkin, että on usein halvempaa koota hajautettu järjestelmä riittävällä suorituskyvyllä pienemmistä osista, kuin yksi keskitetty piste, joka hoitaisi kaiken datan käsittelyn.

Hajautetuilla tietokantaratkaisuilla tavoitellaan siis suorituskykyä, parempaa vikasietoisuutta ja skaalautuvuutta, mutta hajautettuihin tietokantoihin liittyy myös tiettyjä ongelmia ja tarvittavia kompromisseja, joita käsitellään tarkemmin seuraavassa alaluvussa.

2.3 Hajautettuun tietokantaan liittyviä ongelmia

Vaikka hajautetulla tietokannalla saavutetaan soveltuvassa käyttötapauksessa useita etuja keskitettyyn järjestelmään nähden, kuten parempaa suorituskykyä ja vikasietoisuutta, lisää se kuitenkin järjestelmän kompleksisuutta. Özsu ja Valduriez (2011) kuvaavat tätä lisääntynyttä kompleksisuutta kolmen tekijän avulla, jotka ovat replikaatio, yksittäisten solmujen virhetilanteet ja solmujen epätietoisuus toisten solmujen tilasta.

Hajautettuja tietokantoja käyttävät järjestelmät voidaan jakaa synkronisiin- ja asynkronisiin järjestelmiin (Wiesmann et al., 2000b). Synkronisessa järjestelmässä operaatiot tehdään järjestyksessä eikä samaan dataan sallita rinnakkaisia operaatioita, jolloin joudutaan hyväksymään mahdollisesti alentunut suorituskyky ja suuremmat viiveet, jotka johtuvat solmujen välisestä kommunikoinnista (Charron-Bost, Pedone, & Schiper, 2010; Wiesmann et al., 2000b). Synkroniset järjestelmät takaavat vahvan yhteneväisyyden ja niitä vasten voidaan toteuttaa sovelluksia, jotka voivat käyttää jokaista järjestelmän solmua kuin se olisi ainoa olemassa oleva (Charron-Bost et al., 2010). Järjestelmä siis huolehtii siitä, että kaikki lukuoperaatiot kaikista järjestelmän solmuista palauttavat saman arvon ja kaikki kirjoitus- tai päivitysoperaatiot tehdään kaikkiin solmuihin ennen kuin niihin voidaan tehdä uusia operaatioita. Vahvan yhteneväisyyden takaavat järjestelmät saattavat kuitenkin osoittautua pullonkaulaksi koko järjestelmän suorituskyvylle (Charron-Bost et al., 2010), sillä vahvan yhteneväisyyden takeen täyttämiseksi joudutaan kommunikoimaan paljon eri solmujen välillä ja käyttämään lukkoja dataan operaatioiden aikana. Synkronisissa järjestelmissä yhteneväisyyden takaaminen lisää myös järjestelmän kompleksisuutta, sillä jokaista tietokantaan tehtävää operaatiota varten tarvitsee tehdä N*M määrä varmisteluja (N=solmujen lukumäärä, M=tarvittavien viestien lukumäärä/solmu). Tällä voidaan taata, että data järjestelmän eri solmuissa on yhtenevää vielä operaation jälkeenkin ja että kaikkia asiakkaita palvellaan kaikissa tilanteissa samalla datalla eri solmuista. Varmistetuilla tarkoitetaan koordinointia muiden solmujen kanssa, kuten lukkojen hankkimista ennen operaatiota sekä niiden vapauttamista vahvistuksen jälkeen. Järjestelmän tulee myös kyetä selviämään erilaisista virhetilanteista, kuten tilanteesta, joissa jokin solmu ei vastaa tai on muuten saavuttamattomissa, jolloin lukkoja ei mahdollisesti saada hankittua päivitettävälle datalle tai vapautettua operaation jälkeen.

Asynkronisessa järjestelmässä taas solmujen välistä kommunikaatioita ei tarvita transaktion aikana, sillä toiminta luottaa kykyyn ratkaista mahdolliset konfliktit jälkikäteen. Jos operaatioita voidaan tehdä kaikkiin datan kopioihin itsenäisesti ja huolehtia niiden jakamisesta muille kopioille myöhemmin, voidaan suorituskykyä kasvattaa, mutta kopioissa saattaa ilmetä epäyhteneväisyyksiä, jotka myös heijastuvat tietokantoja käyttäville applikaatioille ja asiakkaille (Charron-Bost et al., 2010). Jos operaatioita voidaan suorittaa samanaikaisesti, täytyy ne pystyä erottelemaan toisistaan mahdollisten konfliktien ratkaisemiseksi (Wiesmann et al., 2000b). Kahden samaan dataan tehdyn operaation välillä vallitsee konflikti jos dataan tehdään kaksi samanaikaista operaatiota, joista vähintään toinen on kirjoitusoperaatio (Wiesmann et al., 2000b). Asynkronisuudella saavutettu lisä suorituskyvyssä näkyykin heikentyneenä yhteneväisyyden takeena (Abadi, 2012), sillä rinnakkaisella prosessoinnilla ja asiakkaiden palvelemisella saavutettu lisä suorituskyvyssä kasvattaa kompleksisuutta solmujen välisessä koordinoinnissa ja virheiden korjaamisessa. Järjestelmää suunnitellessa, ei kuitenkaan välttämättä tarvitse luokitella järjestelmää täysin asynkroniseksi tai synkroniseksi, sillä synkronisuuden aste voidaan myös määrittää toimintokohtaiseksi. Esimerkiksi voidaan tehdä kaikki kirjoitusoperaatiot synkronisesti, mutta taas lukuoperaatiot asynkronisesti.

Hajautetun tietokantajärjestelmän vastuulla on myös kyetä valitsemaan lukuoperaatioille yksi kopio pyydetystä datasta, sekä pystyä jakamaan dataan tehdyt päivitykset muille solmuille (Özsu & Valduriez, 2011). Osittain replikoidussa järjestelmässä järjestelmän tulee kyetä valitsemaan solmu, josta dataa kysellään, mikäli lokaali solmu ei sitä kykene tarjoamaan, jolloin järjestelmän tulee myös kyetä valitsemaan suorituskyvyn ja toiminnan kannalta paras saatavilla oleva kopio. Valinnan tekeminen saattaa olla haastavaa, sillä hajautetun tietokantajärjestelmän solmut eivät välttämättä ole jatkuvasti tietoisia toisten solmujen ja niihin varastoidun datan tilasta (Özsu & Valduriez, 2011).

Hajautetun järjestelmän tulee myös pystyä jakamaan päivitykset tilanteissa, jossa yksi tai useampi järjestelmän solmu on väliaikaisesti saavuttamattomissa (Özsu & Valduriez, 2011). Kaikki päivitykset, jotka tietueeseen tehdään, tulee lopulta tehdä kaikkiin tietueen kopioihin, mukaan lukien myös päivityksen tapahtumahetkellä saavuttamattomissa olevien solmujen kopiot. Jos päivityksiä ei jaettaisi kaikille solmuille heti kun mahdollista, kasvaisi riski tietueiden päivittymisestä samanaikaisesti omissa haaroissaan ja asiakkaita palveltaisiin eri versioilla tietueista, joka taas hankaloittaisi konfliktin ratkaisemista entisestään. Asynkronisessa hajautetun tietokannan järjestelmässä eri kopioita samasta tietueesta voidaan päivittää samanaikaisesti, mikä lisää kopioiden ja transaktioiden synkronoinnin kompleksisuutta verrattuna keskitettyyn tietokantajärjestelmään (Özsu & Valduriez, 2011). Näitä synkronoinnin ongelmia on ratkaistu erilaisilla replikaatiostrategioilla ja rinnakkaisuuden hallinnalla (Özsu & Valduriez, 1991; Wiesmann et al., 2000b), mutta niissäkin ratkaisuissa joudutaan tasapainottelemaan synkronoinnin varmuuden ja heikomman suorituskyvyn välillä.

Hajautettuun tietokantaan liittyvät haasteet pohjautuvat kasvaneeseen järjestelmän kompleksisuuteen, joka näkyy saman datan rinnakkaisessa käytössä tai rinnakkaisen käytön estämisessä. Hajautetun tietokannan toteuttamisen haasteita on tutkittu paljon ja yksi ratkaisu niihin on erilaiset replikaatiostrategiat, joita esitellään seuraavassa luvussa.

3 TIETOKANNAN REPLIKAATIO

Hajautetussa tietokannassa tietokantojen eri kopiot voidaan nähdä yhtenä loogisena kokonaisuutena. Vaikka yksittäiset tietokannat huolehtivat itse siitä, että samasta datasta voi olla olemassa vain yksi versio kerrallaan, jää kaikkien solmujen yhteneväisyyden huolehtiminen järjestelmässä tietokantoja korkeammalle taholle. Solmujen välinen kommunikaatio voidaan toteuttaa replikoinnin avulla, jolla voidaan kasvattaa järjestelmän saatavuutta, suorituskykyä ja skaalautuvuutta (Özsu & Valduriez, 2011; Saito & Shapiro, 2005). Datan replikaatio on tärkeä osa-alue hajautettujen tietokantajärjestelmien toteuttamisessa, sillä sen avulla pystytään ratkaisemaan monia hajautetun tietokannan ongelmia ja heikkouksia. Replikoimalla päivitykset tietokantojen välillä, voidaan pienentää vasteaikoja ja toimia tehokkaasti laajallakin maantieteellisellä alueella, sillä hajautetun tietokannan lähin kopio pystyy palvelemaan asiakkaita omalla kopiollaan datasta sen sijaan, että data välitettäisiin pyynnöstä keskitetystä tietokannasta. Replikaatioon liittyy myös ongelmatilanteita, kuten samanaikaisiin operaatioihin liittyvät ongelmat (Eager & Sevcik, 1983), jotka ovat erilaisia riippuen valitusta replikaatiostrategiasta. Replikoinnin ongelmat ovat nykypäivänä jo kattavasti tutkittuja ja niiden ratkaisemiseksi on luotu erilaisia replikaatiostrategioita (Chen et al., 2013; Guy et al., 1999; Saito & Shapiro, 2005; Wiesmann et al., 2000b). Tässä luvussa esitellään replikoinnin hyötyjä, yleisimmät replikaatiostrategiat sekä niille ominaiset ongelmat ja hyödyt. Lisäksi alaluvussa 3.2 käsitellään konfliktien hallintaa replikoidussa järjestelmässä.

3.1 Yleisimmät replikaatiostrategiat

Replikoiduissa hajautetuissa tietokannoissa on tärkeää pohtia miten saman tietueen eri kopioihin tehdyt päivitykset käsitellään, sillä jos päivityksiä ei saada tehtyä kaikkiin kopioihin kerralla, saattaa tietueen kopiot päivittyä eri tavalla samanaikaisesti (Guy et al., 1999). Replikaatiostrategiat kuvaavat mitä dataa replikoidaan sekä miten, koska ja minne data replikoidaan (Charron-Bost et al.,

2010). Tässä alaluvussa esitellään kaksi yleisesti käytettyä replikaatiostrategiaa, optimistinen ja pessimistinen replikaatio, sekä niihin liittyviä ominaispiirteitä ja ongelmia. Edellä mainituille replikaatiostrategioille löytyy tutkimuksesta useita eri termejä ja selityksiä, vaikka kuitenkin usein puhutaan samasta asiasta tai sen osa-alueesta. Vastaavan kaltaisia strategioita on muun muassa kutsuttu myös laiskaksi (lazy) ja ahneeksi (eager) replikaatioksi (Özsu & Valduriez, 1991; Wiesmann et al., 2000b).

3.1.1 Optimistinen replikaatio

Optimistisessa replikaatiomallissa päivityksiä voidaan tehdä lokaaliin tietueeseen ilman, että vaaditaan lukkoja muille kopioille tietueesta (Guy et al., 1999; Saito & Shapiro, 2005; Wiesmann et al., 2000b). Optimistisen replikaatiomallin avulla voidaan parantaa datan saatavuutta ja malli skaalautuu pessimististä replikaatiomallia paremmin sekä parantaa suorituskykyä (Guy et al., 1999; Saito & Shapiro, 2005). Optimistisen replikaatiomallin hyötynä on myös se, että eri instanssien välisten yhteyksien ei tarvitse olla luotettavia, toisin kuin pessimistisessä replikaatiossa, sillä malli ei vaadi toisten kopioiden päivittämistä välittömästi. Optimistinen replikaatiomalli soveltuu siis hyvin ympäristöihin, joissa yhteydet ovat epäluotettavia, hitaita tai viiveet suuria, koska operaatioiden ei tarvitse odottaa vahvistusta replikaation onnistumisesta muilta kopioilta. Optimistinen malli tarjoaa usein suorituskykyä ylitse muiden mallien, sillä viestien määrä eri solmujen välillä on vähäistä verrattuna muihin malleihin ja soveltuukin erittäin hyvin järjestelmiin, joissa tietokantaan tehdyt operaatiot ovat suurelta osin kyselyjä (Wiesmann & Schiper, 2005).

Optimistiseen replikaatioon liittyy toki myös varjopuolia. Kuten on mainittu aiemmin, saattavat tietueen eri kopiot päivittyä eri tavalla eri solmuissa. Tämä voi johtaa tilanteeseen, jossa kopiot lähtevät päivittymään useassa eri haarassa, jolloin järjestelmän eri solmut ja niiden datan historia eivät ole keskenään yhteneviä. Saito ja Shapiro (2005) toteavat, että optimistinen replikaatio on datan saatavuuden ja yhteneväisyyden vaihtokauppa. Tällä vaihtokaupalla tarkoitetaan sitä, että samalla kun parannetaan datan saatavuutta, joudutaan heikentämään taetta tiedon yhteneväisyydestä. Replikaatiota ei tarvitse kuitenkaan suunnitella täysin optimistista tai pessimististä strategiaa käyttäen, vaan esimerkiksi jo kirjoitusoperaatioiden järjestäminen riittää takaamaan datan eheyden solmujen välillä, jolloin lukuoperaatioita voidaan tehdä samanaikaisesti.

Optimistisen replikaation yhteydessä puhutaan usein lopulta yhtenevästä (eventually consistent) hajautetusta tietokannasta. Lopulta yhteneväisyys tarkoittaa sitä, että jos tietuetta ei päivitetä, lopulta kaikki hajautetun tietokannan solmut sisältävät samanlaisen version tietueesta (Saito & Shapiro, 2005; Vogels, 2009). Lopulta yhtenevä malli sallii operaatioiden tekemisen kaikkiin kopioihin, ja muutokset välitetään muille kopioille taustalla, kun se on mahdollista tai soveltuvaa. Tämä malli mahdollistaa myös operaatioiden tekemisen tilassa, jossa yhteyttä muihin järjestelmän solmuihin ei ole kyseisellä hetkellä muodostettu. Lopulta yhtenevän replikaation tulee kyetä ratkaisemaan konflikteja itsenäisesti,

sillä päivitykset saattavat saapua kopioille eri järjestyksessä kun ne ovat tapahtuneet (Charron-Bost et al., 2010). Konfliktien hallintaa voidaankin pitää yhtenä optimistisen replikaation tärkeimpänä yksittäisenä toimintona (Saito & Shapiro, 2005; Wiesmann et al., 2000b). Järjestelmän tulee kyetä valitsemaan ristiriidassa olevista päivityksistä voittaja (Wiesmann et al., 2000b), jotta tietokannat saadaan lopulta saatettua takaisin yhtenevään tilaan ja mahdolliset eri haaroissa päivittyneet datat yhdistettyä.

Koska optimistinen replikaatio ei varmista, että kaikki kopiot ovat päivittyneet ennen muutosten vahvistamista, voidaan sitä pitää heikon yhteneväisyyden (weak consistency) mallina. Vaikka optimistiset ja lopulta yhtenevät replikaatiomallit eivät ole kaikkiin järjestelmiin parhaita kandidaatteja, tarjoavat ne kuitenkin verrattain parempaa suorituskykyä ja saatavuutta, kasvattaen niiden suosiota pessimistisiin malleihin nähden (Bailis & Ghodsi, 2013).

3.1.2 Pessimistinen replikaatio

Pessimistisessä replikaatiossa tarkoituksena on estää samanaikaiset päivitykset tietueen eri kopioihin (Guy et al., 1999). Pessimistinen replikaatio takaa datan atomisuuden, mutta on verrattain hidas ja altis deadlock-tilanteille (Wiesmann & Schiper, 2005).

Yksi paljon tutkittu tapa toteuttaa pessimistinen replikaatio on hajautettu lukitus. Hajautetussa lukituksessa tietueen kopioille pyydetään lukitusta operaation ajaksi ja lukot avataan vasta kun kaikki kopiot ovat tehneet päivityksen tai päivitys kumotaan (Wiesmann et al., 2000b). Lukitsemalla kaikki kopiot operaation ajaksi voidaan välttyä konflikteilta, mutta se ei poista kaikkia ongelmia. Tietueet, joiden päivitys- tai lukuväli on tiheä, ovat usein lukittuna ja näin myös usein saavuttamattomissa. Myös tilanteessa, jossa yhdellä tai useammalla kopiota hallitsevalla solmulla kestää vastata, odottavat kaikki muut kopiot vahvistusta operaation onnistumisesta. Hajautettuun lukitukseen turvaavat replikaatioprotokollat toimivat hyvin nopeiden yhteyksien varassa, mutta kun tiedonsiirron kustannukset kasvavat, laskee myös näiden protokollien suorituskyky nopeasti (Kemme & Alonso, 2000).

Myös muita pessimistisiä replikaatioprotokollia on tutkittu ja ehdotettu kirjallisuudessa, kuten ryhmäkommunikaatioon perustuvat mallit (Alonso, 1997; Pedone, Guerraoui, & Schiper, 1998; Schiper & Raynal, 1996). Vaikka näillä malleilla saadaan parannettua suorituskykyä huomattavasti verrattuna niin sanottuihin perinteisiin hajautetun lukituksen malleihin, ovat ne silti suorituskyvyssä optimistisia replikaatiomalleja heikompia, vaikkakin suotuisissa olosuhteissa saattavat päästä lähelle (Wiesmann & Schiper, 2005).

Pessimistinen replikaatiomalli soveltuukin tilanteisiin, joissa datan eheys on tärkeämpää kuin saatavuus ja suorituskyky. Pessimistisen replikaation avulla voidaan taata vahva datan yhteneväisyys (Charron-Bost et al., 2010).), sillä pessimistisessä replikaatiossa tulee tehdä kaikki muutokset kaikkiin kopioihin, tai jos tämä ei ole mahdollista niin ei tehdä mitään muutoksia yhteenkään kopioon, edellyttäen, että myös lukuoperaatiot toteutetaan järjestetysti. Yhteneväi-

syyden takaamisella on suorituskyvyllisiä seuraamuksia, sillä tiettyyn tietueeseen tehtävän operaation aikana kaikki muut kyseistä tietuetta tavoittelevat operaatiot joutuvat odottamaan. Suorituskyvyn heikentyminen johtuu myös osittain kasvaneesta liikenteestä eri solmujen välillä, mikä lisää viivettä kyselyjen käsittelyssä. Koska pessimistinen replikaatio varmistaa, että kaikki kopiot ovat päivittyneet tai lukuoperaatio on valmis ennen kuin muilla on oikeus päästä käsiksi lukittuun dataan, voidaan sitä pitää vahvan yhteneväisyyden (strong consistency) takaavana mallina.

3.2 Konfliktien hallinta

Saito ja Shapiro (2005) toteavat, että tehokkain lähestymistapa konfliktien hallintaan on niiden välttäminen, mitä pessimistiset replikaatiostrategiat toteuttavat estämällä operaatioita sekä kumoamalla ne tarvittaessa. Konflikteja ei kuitenkaan voida kaikissa järjestelmissä välttää, joten niiden hallinta ja ratkaiseminen tulee pystyä toteuttamaan myös vaihtoehtoisella menetelmällä. Saito ja Shapiro (2005) esittelevätkin neljä eri lähestymistapaa konfliktien hallintaan, jotka ovat: (1) konfliktien ehkäiseminen, (2) konfliktien sivuuttaminen, (3) konfliktien vähentäminen sekä (4) konfliktien havaitseminen ja ratkaiseminen. Kaikille edellä mainituille lähestymistavoille on omat käyttötarkoituksensa. Esimerkiksi hajautetuissa järjestelmissä, joissa on käytössä master–slave-kokoonpano, voidaan kirjoituskonfliktit välttää, mutta lukuoperaatioissa saattaa silti konflikteja esiintyä (Saito & Shapiro, 2005). Kuitenkin asynkronisissa järjestelmissä riski konflikteille on yleensä suuri, ja jos konflikteja ei voida sivuuttaa tai niiltä ei voida välttyä, tulee ne pystyä havaitsemaan ja ratkaisemaan luotettavasti. Konfliktit voidaankin ratkaista manuaalisesti tai automaattisesti, joista manuaalisessa ratkaisumallissa käyttäjän vastuulla on yhdistää kaksi tietueen versiota tai valita voittaja niiden väliltä (Saito & Shapiro, 2005). Automaattiset konfliktin ratkaisut taas pyrkivät päättelemään kahdesta tai useammasta tietueen versiosta voittajan tai miten ne voitaisiin yhdistää. Tämänhetkisissä suosituissa tietokanannanhallintajärjestelmissä replikaatiosta aiheutuvat konfliktit ratkaistaan deterministisesti, jolloin järjestelmä pystyy ennalta määriteltyjen sääntöjen avulla päättelemään kumpi päivityksistä on oikeampi tai miten kaksi päivitystä voidaan mahdollisesti yhdistää (Microsoft, 2017; Oracle, 2019). Deterministinen konfliktien ratkaisu ei tarvitse kommunikaatiota eri järjestelmän solmujen välillä, sillä kaikki käyttävät samaa sääntöjoukkoa konfliktien ratkaisemiseen.

Järjestelmässä, jossa kirjoitus- ja lukuoperaatiot ovat sallittuja kaikissa solmuissa, on hankalaa välttyä konflikteilta kokonaan. Tietueen samanaikaiseen päivitykseen liittyvät konfliktit voidaankin siis jakaa kirjoitus–kirjoitus-konfliktiin sekä luku–kirjoitus-konfliktiin. Kirjoitus–kirjoitus-konfliktista on kyse silloin kun kaksi tai useampi solmua päivittää tietueen arvoja samanaikaisesti, johtaen tilanteeseen, jossa eri solmuissa olevat datat muuttuvat eri tavalla. Toinen konfliktityyppi on luku–kirjoitus-konflikti, jossa samanaikaisesti toinen

solmu päivittää tietuetta, kun taas toinen solmu suorittaa tietueeseen lukuoperaation. Tämä lukuoperaatio palauttaa vielä tietueesta muuttumattoman version, jolloin kyselyn tuloksena saatu versio tietueesta ei ole enää palautushetkellä yhtenevä muiden solmujen kanssa. Molemmat konfliktit voivat johtaa tilanteisiin, jossa samaa dataa päädytään päivittämään eri tavalla kahdessa tai useammassa solmussa.

Yhtenä paljon tutkittuna ja sovellettuna keinona välttää konflikteja ja ratkaista niiden aiheuttamia ongelmia on MVCC (Multiversion Concurrency Control) eli rinnakkaisuudenhallinta, joka sallii tietueelle useamman eri version samanaikaisesti (Berenson et al., 1995). Bernstein ja Goodman (1983) kuvaavat, että jokainen tietueeseen tehty päivitys muodostaa uuden kopion tietueesta ja samaan aikaan tapahtuvat lukuoperaatiot voivat käyttää näistä kopioista soveltuvinta. Tällaisessa rinnakkaisuuden hallinnassa ei siis tarvitse lukita tietuetta operaation ajaksi, sillä eri operaatiot muodostavat uuden version kopiosta tai lukevat tiettyä versiota. Bernstein ja Goodman (1983) toteavat myös, että MVCC tarjoaa tietokannanhallintajärjestelmälle enemmän joustoa luku- ja kirjoitusoperaatioiden kontrolloinnissa. Useat tietokantatuotteet tukevat nykyään Snapshot Isolation rinnakkaisuudenhallinta-algoritmia. Snapshot Isolation -algoritmia käyttävät operaatiot tehdään aina vedokseen datasta, joka on luotu ennen operaation alkua (Berenson et al., 1995). Tällä tavoin eri operaatiot eivät näe toisten operaatioiden tekemisiä kesken suorituksen. Yksi tapa ratkaista samanaikaisista kirjoitusoperaatioista voittaja, on noudattaa ”First Committer Wins” –sääntöä (Fekete, Liarokapis, O’Neil, O’Neil, & Shasha, 2005). Jos transaktion päättyessä joku muu transaktio on päivittänyt dataa transaktion suoritusaikana, valitaan se voittajaksi tätä sääntöä noudattaessa. Daudjee ja Salem (2006) toteavat, että Snapshot Isolationin suurin etu onkin se, että lukuoperaatioita ei ikinä estetä, joka lisää rinnakkaisuutta ja suorituskykyä lukuintensiivisissä järjestelmissä.

4 ARKKITEHTUURIMALLIN JA REPLIKAATIOSTRATEGIAN VALINTA JÄRJESTELMÄN SUUNNITTELUSSA

Luvuissa 2 ja 3 tutkittiin kirjallisuuteen pohjaten erilaisia tapoja toteuttaa hajautettu tietokanta ja datan replikaatio järjestelmän solmujen välillä. Tässä luvussa pohditaan tutkitun pohjalta mitä tulisi ottaa huomioon hajautettua tietokanta-arkkitehtuurimallia ja replikaatiostrategiaa suunnitellessa.

Hajautetun tietokannan ja datan hajauttamisen suunnitteluprosessi on ylhäältä alaspäin (top-down) suuntautuva prosessi, jossa liikkeelle lähdetään selvittämällä tietokannan käyttävien tahojen vaatimukset datan saatavuudelle (Rahimi & Haug, 2010). Molempien, niin datan sijoitusmallin kuin replikaatiostrategian valinnassa joudutaan tekemään erilaisia kompromisseja, eikä tämänhetkinen tutkimus tarjoa yhtä ja oikeaa ratkaisua kaikkien ongelmien ratkaisemiseksi. Tästä syystä on tärkeää selvittää ne muuttujat, jotka vaikuttavat valintaan ja ymmärtää mistä joudutaan luopumaan, jotta voidaan tarjota tietokantoja käyttäville tahoille mahdollisimman soveltuva palvelu.

4.1 Hajautetun tietokanta-arkkitehtuurin valinta

Tietokanta-arkkitehtuurin valintaan vaikuttaa kokonaisjärjestelmän käyttötarkoitus. Jos järjestelmää käytetään keskitetysti maantieteellisesti pienellä alueella, saattaa keskitetty tietokanta-arkkitehtuuri olla perusteltu vaihtoehto. Kuitenkin jatkuvasti enemmissä määrin verkottuvassa maailmassa järjestelmiltä vaaditaan kykyä skaalautua ja toimia tehokkaammin muuttuvassa ympäristössä (Abadi, 2012; Chen et al., 2013). Useassa tilanteessa siis yhden keskitetyn tietokannan käyttäminen koko organisaatiossa tai järjestelmässä ei ole järkevää, vaan joudutaan toteuttamaan hajautettu versio. Arkkitehtuurimallia valitessa täytyykin pohtia, haetaanko hajauttamisella ensisijaisesti parempaa suorituskykyä, parempaa vikasietoisuutta, datan saatavuutta vai jotain näiden väliltä.

Osittamalla data järjestelmän eri solmuihin, saadaan suorituskykyä parannettua, mutta yhden solmun pettäessä, on siellä sijainnut data saavuttamattomissa. Vaikka yksi solmuista pettäisi, kykenevät muut silti palvelemaan normaalisti saatavilla olevalla datalla. Osioidussa mallissa lokaaleihin tietokantoihin kohdistuvat kyselyt ja päivitykset ovat tehokkaita. Resursseja kuluttavia operaatioita ovat taas ne, jotka liittyvät usean solmun sisältämään dataan, tai dataan, jota ei löydy ollenkaan lokaalista tietokannasta.

Jos taas järjestelmän vikasietoisuus ja datan saatavuus järjestelmän eri solmuista on tärkeää toiminnan kannalta, tarjoaa P2P-arkkitehtuurimalli näitä ominaisuuksia. P2P-mallissa data toisinnetaan kaikkiin solmuihin, joten yhden solmun pettäessä, muut ovat kykeneviä palvelemaan koko datalla. P2P-malli tarjoaa myös vikasietoisuuden lisäksi parempaa suorituskykyä lukuoperaatioille, mutta kirjoitusoperaatiot ovat enemmän resursseja kuluttavia.

On siis vaikea sanoa yhtä ehdotonta mallia kaikkiin tilanteisiin, joka olisi soveltuvampi kuin muut. Valintaan vaikuttavia tekijöitä ovat muun muassa tietokantaoperaatioiden arvioitu määrä, operaatioiden painottuminen luku- tai kirjoitusoperaatioihin, järjestelmän rakenne, suorituskykyvaatimukset sekä vaatimukset vikasietoisuudelle ja datan saatavuudelle kustakin järjestelmän solmusta. Valintaa tehdessä täytyy punnita edellä mainittuja kriteerejä ja selvittää minkä toteutuminen on kokonaisuudessa järjestelmän toiminnan ja suorituskyvyn kannalta oleellista.

4.2 Replikaatiostrategian valinta

Replikaatiostrategian valinnassa on myös tärkeää vertailla eri vaihtoehtoja. Eri strategioita on olemassa paljon ja niiden vertailu on usein hankalaa (Charron-Bost et al., 2010). Strategioiden vertailussa tulee ottaa huomioon niin järjestelmän kuin myös sen toimintaympäristön tarpeet ja rajoitteet. Usein kuitenkin riittää, että järjestelmä vaikuttaa vahvasti yhtenevältä loppukäyttäjille ja sovelluskehittäjille (Charron-Bost et al., 2010). Joissakin tilanteissa voidaan siis hyväksyä tietyntasoinen epäyhteneväisyys datassa solmujen välillä tai viive yhteneväisyyden saavuttamisessa, jotta saadaan järjestelmän suorituskyky pidettyä korkealla tasolla. Tätä ei kuitenkaan voida yleistää kaikkiin järjestelmiin, sillä on myös olemassa järjestelmiä, joissa tiedon yhteneväisyys eri solmujen välillä on kriittistä.

Jos data ja sen yhteneväisyys eri solmujen välillä on kriittistä ja epäyhteneväisyyksiä ei voida järjestelmässä sallia, voi olla hyödyllistä suosia vahvan yhteneväisyyden malleja. Usein ei kuitenkaan tarvitse valita vain toista ääripäätä, vaan valitaan malli, joka takaa riittävän yhteneväisyyden suurimpaan osaan tilanteista, esimerkiksi estämällä samanaikaiset kirjoitusoperaatiot kopioihin, mutta sallimalla rinnakkaiset lukuoperaatiot.

Järjestelmän toimintaympäristö saattaa myös asettaa rajoitteita tai vaatimuksia replikaatiostrategian valinnalle. Jos replikaatio täytyy esimerkiksi pystyä toteuttamaan yksisuuntaisia tai ei-pysyviä yhteyksiä käyttäen, ei voida hal-

littua lukitusta ja vahvaa yhteneväisyyttä toteuttaa järkevällä tavalla. Myös hajautetun tietokannan solmujen välinen maantieteellinen etäisyys, yhteyksien nopeus ja -laatu sekä saatavilla olevat resurssit voivat vaikuttaa strategian valintaan.

5 TUTKIMUKSEN TOTEUTUS

Tässä luvussa kuvaillaan tutkimuksen tavoite, toimintaympäristön haasteet ja ominaisuudet sekä tutkimuksen menetelmät ja tulosten verifiointi. Ensin kuvaillaan tutkimuksen tavoitteita, jonka jälkeen alaluvussa 5.2 kerrotaan, miten tutkimuksen tuloksena syntynyttä ratkaisua testataan ja tulokset vahvistetaan.

5.1 Tutkimuksen tavoite

Luvuissa 2 ja 3 esiteltiin kirjallisuuskatsauksen pohjalta hajautetun tietokannan eri arkkitehtuurimalleja, ongelmia, tavoiteltuja hyötyjä sekä eri replikaatiostrategioita. Katsauksesta nousi esille se, että suunnittelussa ei usein ole mahdollista saavuttaa optimaalisinta suorituskykyä samalla säilyttäen tae datan konstistentistä tilasta eri solmujen välillä. Tämän tutkimuksen konstruktiivisen osion tavoitteena on luoda toimeksiantona suunnittelumalli hajautetun tietokannan replikoinnille sekä toteuttaa suunnitellun mallin pohjalta ohjelmistopohjainen ratkaisu tähän ongelmaan. Suunnittelussa on otettu huomioon kirjallisuuskatsauksessa esille nousseet ongelmat ja niihin esitetyt ratkaisut.

Ensimmäisenä konstruktiivisen osion tavoitteena on siis luoda suunnittelumalli, jota voidaan hyödyntää hajautetun tietokannan replikoinnin suunnittelussa. Suunnittelumallia testataan toteuttamalla ohjelmistopohjainen malliin perustuva ratkaisu toimeksiantajalle. Tämä ratkaisun tulee toimia ympäristössä, jota kuvataan tarkemmin luvussa 6. Toteutus tehdään myös siten, että se on mahdollisimman riippumaton minkään tietyn tietokannanhallintajärjestelmän replikaatioon liittyvistä toiminnallisuuksista. Vaikka suunnittelumalli on toteutettu ensisijaisesti toimeksiantajan kohdeympäristöä ja tarpeita ajatellen, kuvailee se myös yleisesti yhden tavan ratkaista optimistinen tietokantareplikaatio ympäristössä, jossa tietyt resurssit, kuten kaksisuuntaiset yhteydet, eivät ole käytettävissä.

5.2 Toteutuksen testausmenetelmät ja tulosten vahvistus

Seuraavissa alaluvuissa kuvaillaan millä menetelmillä replikaatiota testataan ja miten testauksen tulokset vahvistetaan. Ensimmäiseksi kuvaillaan testiympäristöä ja testaukseen käytetyn testisovelluksen sekä järjestelmän yleisen rajapinnan kyvykkyyksiä. Lisäksi kuvataan, miten testataan sellaisia tapauksia, joita ei kyetä toistettavasti toteuttamaan testisovelluksen ja yleisen rajapinnan avulla. Lopuksi kerrotaan, miten testauksen tulokset varmennetaan.

5.2.1 Testausympäristö

Testausympäristössä varmistetaan, että tietokannat eivät kykene kommunikoimaan keskenään muuten kuin niitä hallitsevien sovellusten kautta, jotta testien aikana kunkin tietokantainstanssin tilaa muuttaa vain sitä hallitseva sovellus. Testeissä järjestelmän solmut pystytetään omiin CentOS 7 virtuaalikoneisiin, joissa jokaisella solmulla on käytettävissä oma lokaali Oracle-11g XE tietokantainstanssi. Toteutettu ratkaisu ei ole riippuvainen edellä mainituista teknologioista, joten testit voitaisiin tehdä myös toisenlaisella kokoonpanolla. Tietokantainstanssit alustetaan jokaisessa testissä kyseiselle testille soveltuvaan tilaan, jotta varmistutaan siitä, että aikaisemmat ajot eivät vääristä tuloksia.

5.2.2 Testisovellus ja sen kyvykkyys

Kohdeympäristössä on käytössä Python-ohjelmointikielellä kirjoitettu testikirjasto, jonka avulla pystytään ajamaan testejä solmuille yleisen rajapinnan kautta. Tämän rajapinnan kautta pystytään testisovelluksella suorittamaan jokaiseen solmuun operaatioita, jotka lisäävät, päivittävät ja poistavat dataa. Testisovelluksella pystytään edellä mainittujen operaatioiden suoritusta satunnaistamaan siten, että testiohjelma arpoo jokaisen operaation alkaessa, onko seuraava operaatio lisäys, päivitys vai poisto sekä minkä taulun dataan se vaikuttaa. Sovelluksessa on myös käytössä unittest-viitekehys, jonka avulla pystytään luomaan useista operaatiosta koostuvia testitapauksia ja ajamaan niitä rinnakkain usealla asiakasohjelmalla ja satunnaisessa järjestyksessä. Replikaatiota testaavien testitapausten lisäksi ajetaan samalla myös muita järjestelmän osia testaavia tapauksia samanaikaisesti, jotta saadaan käsitys siitä, miten taustalla tapahtuva replikointi mahdollisesti vaikuttaa muihin operaatioihin tai yleiseen suorituskykyyn.

5.2.3 Rajapinta ja sen kyvykkyys

Järjestelmän yleinen rajapinta tarjoaa mahdollisuuden lähettää viestejä datan lisäämiseksi, päivittämiseksi, poistamiseksi, datan hakemiseksi sekä muutosten kuuntelemiseksi. Testisovellukseen saadaan ilmoituksia muutoksista dataan, jos rajapinnan kautta rekisteröidytään kuuntelemaan tiettyä objektia tai data-

tyyppiä. Rajapinnan kautta testatessa voidaan myös tehdä rasitustestejä, jolloin testisovellus käynnistää useita prosesseja ajamaan viestejä rajapinnan läpi rinnakkain. Replikaation ansiosta lokaalin solmun dataan tehty muutos on myös kuunneltavissa muista solmuista, jolloin rajapinnan kuunteluominaisuuksia käyttämällä voidaan todentaa tiedon replikoituneen, sekä varmistaa, että tieto on oikeellista. Rajapinnan läpi saadaan myös tieto mahdollisista konflikteista, jos sovellus on rekisteröitynyt kuuntelemaan konfliktissa olevaa objektia.

5.2.4 Verkkohäirintä ja manuaaliset testit

Replikaatiolle tehdään myös muita testejä testisovelluksen ajamisen lisäksi, sekä sen ajon aikana. Yksi tällainen testitapaus on häiritä verkkoliikennettä virtuaalikoneiden välillä testien ajon aikana. Verkon häirintä toteutetaan WANem-ohjelmalla, jonka avulla voidaan muun muassa satunnaisesti katkoa yhteyksiä, pudottaa ja korruptoida paketteja, sekä lisätä viivettä. Edellä mainituista ominaisuuksista replikointia testatessa kiinnostaa lähinnä yhteyksien katkominen ja viiveen lisääminen, sillä näiden avulla saadaan aikaan tilanteita, joissa päivityksiä ei saada toimitettua toiselle solmulle välittömästi. Lisäksi tehdään myös manuaalisia testejä, kuten estetään väliaikaisesti tapahtumaperustainen replikaatio, jolloin pystytään testaamaan tilannetta, jossa operaation jälkeen lähetetyt replikaatioviestit eivät tavoita toisia solmuja lainkaan. Toinen manuaalinen testaustapa on alustaa eri solmujen tietokannat konfliktissa olevaan tilaan ennen sovellusten käynnistymistä. Tällä saadaan aiheutettua erityistapauksia, jotka on vaikeita toteuttaa rajapinnan läpi testattaessa, mutta niiden esiintyminen lopullisessa käytössä on kuitenkin mahdollista.

5.2.5 Tulosten varmistustavat

Testien tulokset varmistetaan käyttämällä rajapinnan tarjoamaa kuunteluominaisuutta, sekä tarkastelemalla tietokantainstanssien tilaa testitapausten jälkeen, jotta voidaan todentaa replikaation onnistuneen ja että data on yhtenevää testin jälkeen. Kaikista testisovelluksella tehtävistä testitapauksista kirjataan myös ylös suoritukseen kulunut aika. Lisäksi kirjataan ylös, miten operaatiot ovat hajautuneet eri operaatiotyyppien ja taulujen välillä sekä mahdolliset virheet.

6 TULOKSET

Tässä luvussa esitellään tutkielman tulokset. Ensiksi alaluvussa 6.1 esitellään luotu suunnittelumalli replikaation toteuttamiseksi yksisuuntaisia ei-pysyviä yhteyksiä käyttäen. Sen jälkeen alaluvussa 6.2 kuvataan replikaation toteutusta toimeksiantajan kohdejärjestelmään ja viimeiseksi alaluvussa 6.3 kerrotaan millä tavoin replikaatiota testattiin ja mitä näistä testeistä selvisi.

6.1 Replikaation suunnittelumalli

Tässä alaluvussa esitellään suunnittelumalli hajautetun tietokantajärjestelmän replikoinnin toteuttamiseksi yksisuuntaisia ei-pysyviä yhteyksiä käyttäen. Ensiksi esitellään millaisessa toimintaympäristössä mallin avulla replikaatio pitäisi pystyä toteuttamaan ja toiseksi malliin liittyviä komponentteja sekä niiden vastuualueita. Tämän jälkeen kuvataan konfliktien hallintaa ja ratkaisemista sekä miten tietokantoihin tehdyt päivitykset jaetaan mallissa solmujen välillä.

6.1.1 Toimintaympäristön rajoitteet suunnittelumallille

Toimintaympäristö, jota varten tämä malli on luotu, koostuu useammasta hajautetun järjestelmän solmusta, jotka kaikki ylläpitävät omaa tietokantaa. Kaikkia eri solmuissa sijaitsevia tietokantoja voidaan päivittää ja kaikki solmut hallitsevat samaa dataa, joten kyseessä on P2P-tietokanta-arkkitehtuurin toteuttava järjestelmä, jossa data on täysin replikoitua. Haasteita tietokantojen replikoinnille ja synkronoinnille tuo kohdeympäristön rajoitteet. Isoimpana rajoitteena kohdeympäristössä on se, että yhteydet solmujen välillä saattavat olla yksisuuntaisia ja luonteeltaan ei-pysyviä. Alaluvussa 6.1.2 kuvataan yksisuuntaisia yhteyksiä ja niiden replikaatiolle aiheuttamia rajoitteita. Ei-pysyviä yhteyksiä ja niiden haasteita replikaatiolle kuvataan alaluvussa 6.1.3.

6.1.2 Yksisuuntaiset yhteydet ja niiden tuomat rajoitteet

Yksisuuntaisella yhteydellä tarkoitetaan yhteyttä, jossa data kulkee vain toiseen suuntaan. Datan kulun yksisuuntaisuus varmistetaan fyysisesti esimerkiksi data diodin avulla. Tämän mallin suunnittelussa oletetaan, että kaikissa yhteyksissä solmujen välillä datan siirtäminen kahteen suuntaan samaa yhteyttä käyttäen ei ole fyysisesti mahdollista. Solmujen välillä voi joissain käyttötapauksissa olla olemassa myös yhteys molempiin suuntiin, mutta tämä ei ole aina mahdollista.

On siis mahdollista, että solmulta toiselle on olemassa vain yksi yksisuuntainen yhteys, jolloin kommunikointi onnistuu vain toiseen suuntaan. Tästä syystä replikaation toteutus ei voi luottaa siihen, että toinen solmu pystyy aina sille lähettyihin viesteihin antamaan vasteen, eikä varmistamaan, että viestit ovat saapuneet määränpäähänsä. Koska yhteydet eivät välttämättä ole kaksisuuntaisia, eivät pessimistiset replikaatiostrategiat ole järkevästi toteutettavissa. Replikaatio onkin tässä mallissa toteutettu optimistista strategiaa noudattaen, joka antaa lupauksen, siitä että tietokannat ovat lopulta yhteneviä, jos solmujen välillä on toimiva yhteys. Lopulta yhteneväisyys voidaan kuitenkin taata vain silloin, kun solmujen välillä on olemassa yhteydet molempiin suuntiin, sillä mikäli solmujen välillä on toiminnassa yhteys vain toiseen suuntaan, ei taustareplikaatio pysty korjaamaan mahdollisia epäyhteneväisyyksiä datassa. Solmu pystyy kuitenkin lähettämään muutokset toiselle solmulle sokkona, vaikka yhteys olisi vain toiseen suuntaan olemassa, mutta lähetys ei kuitenkaan pysty takaamaan sitä, että toinen solmu saa viestin perille.

6.1.3 Ei-pysyvät yhteydet ja niiden tuomat rajoitteet

Toisena toimintaympäristön tuomana rajoitteena toteutukselle on se, että yhteydet solmujen välillä eivät ole luonteeltaan pysyviä. Koska yhteydet eivät ole pysyviä, ei replikaatioviestien lähettäminen ole mahdollista kaikkina ajanhetkinä kaikille järjestelmän solmuille. Järjestelmän yksittäistä solmua voidaankin siis käyttää verkkoon liitettynä, tai siten, että sillä ei ole lainkaan yhteyttä toisiin solmuihin. Jokainen yhteys solmujen välillä saattaa olla ajoittain toiminnassa tai poissa käytöstä, joten solmujen väliset yhteydet ovat luonteeltaan ei-pysyviä. Yhteyksien muuttuminen tai katkeaminen solmujen välillä ajon aikana on kohdejärjestelmälle normaalia käytöstä, joten sitä ei tulkita virhetilanteeksi, kuten useissa hajautetun tietokannan ratkaisuissa. Yhteyksien ei-pysyvän luonteen takia datan replikointi ei aina ole mahdollista, joten replikaation toteutuksen tulee kyetä synkronoimaan solmujen data myös jälkikäteen, sillä tietokantojen tilaa tulee pystyä muuttamaan myös silloin kun solmua käytetään ilman yhteyttä muihin solmuihin. Datan replikaation vastuulle jääkin solmun uudelleen verkkoon liittyessä aloittaa synkronointiprosessi muiden sillä hetkellä saavutettavissa olevien solmujen kanssa. Kuitenkin, kuten mainittu edellisessä alaluvussa, tarvitsee tämä synkronointiprosessi toimiakseen yhteyden solmujen vä-

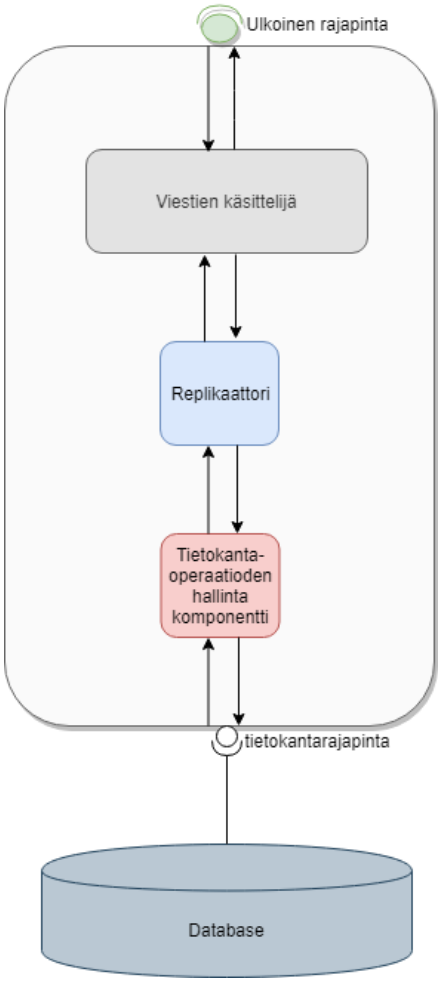
lillä kumpaankin suuntaan, joten pelkän yhden yksisuuntaisen yhteyden palauttamisella synkronointi ei onnistu.

6.1.4 Malliin liittyvät komponentit

KUVIO 5 esittää yksinkertaistetun mallin hajautetun tietokantajärjestelmän yhdestä solmusta ja siihen liittyvistä komponenteista. Jokaisella solmulla on olemassa jonkinlainen rajapinta, johon pystytään liittymään, lähettämään päivityksiä dataan sekä tekemään pyyntöjä solmulta. Jokainen solmu sisältää myös viestien käsittelijän, jonka tehtävänä on välittää viestit oikeille vastaanottajilleen. Viestien käsittelijä ottaa vastaan kaikki rajapinnan läpi tulevat viestit ja lähettää ulospäin lähtevät viestit oikeisiin osoitteisiin. Kaikki tietokantaa koskevat operaatiot välitetään komponentin läpi, jota kutsutaan replikaattoriksi. Replikaattori taas välittää viestit komponentille, joka on vastuussa tietokantaoperaatioista. Replikaattorin vastuulla on myös lähettää muille järjestelmän solmuille tieto onnistuneesta operaatiosta, mikäli kyseessä on tietokannan tilaa muuttava operaatio, ja se suoritetaan onnistuneesti.

Replikaattori on tässä mallissa väliohjelmisto, jonka vastuulla on sen läpi kulkevan datan perusteella hoitaa synkronointi ja virheen korjaus järjestelmän solmujen välillä. Replikaattori toimii tietokantarajapinnan yläpuolella osana tietokantaa käyttävää sovellusta. Koska replikaattori toimii väliohjelmistona osana sovelluksen sisäistä viestiketjua, on sillä käytössä sovelluksen muut resurssit, joita voidaan hyödyntää viestien välityksessä ja replikoinnissa. Replikaattori-instanssien välinen kommunikaatio tapahtuu samoja väyliä pitkin kuin muukin solmujen välinen ja sisäinen kommunikaatio.

Tietokantaoperaatioiden hallintakomponentilla tässä mallissa tarkoitetaan sitä ohjelman osaa, joka on lopulta vastuussa tietokantakyselyjen ja kirjoitusten tekemisestä. Näitä komponentteja voi olla järjestelmässä useita tai ne voivat olla itse osa jotain järjestelmän suurempaa komponenttia. Tietokannan hallintakomponentin vastuulla on tietokantaoperaatioiden lisäksi konfliktien hallinta ja ratkaiseminen sekä näistä raportoiminen replikaattorille. KUVIO 5 on yksinkertaistettu esitys hajautetun tietokantajärjestelmän yhden solmun rakenteesta ja vähimmäiskokoonpanosta.



KUVIO 5 Hajautetun tietokantajärjestelmän yksittäisen solmun rakenne

6.1.5 Päivitysten jakaminen mallissa

Resursseiltaan rajoitetun toimintaympäristön takia tässä replikoinnin suunnittelumallissa päivitysten jakaminen on jaettu kahteen osa-alueeseen: tapahtumaperustaiseen- ja taustareplikaatioon. Tapahtumaperustaisen replikaation tehtävänä on pyrkiä jakamaan päivitykset kaikille saatavilla oleville järjestelmän solmuille välittömästi tapahtuman jälkeen. Kuitenkaan tämä ei vielä riitä luotettavaan tiedon jakoon, sillä lähetyistä viesteistä ei saada vahvistusta, joten ei voida varmistua siitä, että toiset solmut ovat saaneet viestin ja pystyneet muuttamaan omaa tilaansa. Taustareplikaation tehtävänä onkin pyrkiä korjaamaan tilanteet, joissa jokin päivitys ei saavuta kaikkia solmuja ja data ei ole jostain syystä yhtenevää solmujen välillä.

6.1.6 Konfliktien ratkaiseminen suunnittelumallissa

Tämän mallin pohjalta toteutetussa replikaatiossa konflikteilta ei voida varmuudella välttyä, sillä kaikki operaatiot tehdään ilman kommunikaatiota mui-

den solmujen välillä, jonka takia on tärkeää, että konfliktit pystytään havaitsemaan, kun niitä pääsee muodostumaan. Konflikteja ei kuitenkaan voida havaita välittömästi niiden muodostuessa, sillä replikaatio tapahtuu vasta operaation jälkeen, eivätkä solmut odota toisilta solmuilta minkäänlaista vastetta replikaatioviesteihin. Tästä syystä replikaatioviestin vastaanottavan pään tulee kyetä luotettavasti ja itsenäisesti havaitsemaan sekä ratkaisemaan konfliktit, joita rinnakkaisesta operoinnista mahdollisesti syntyy.

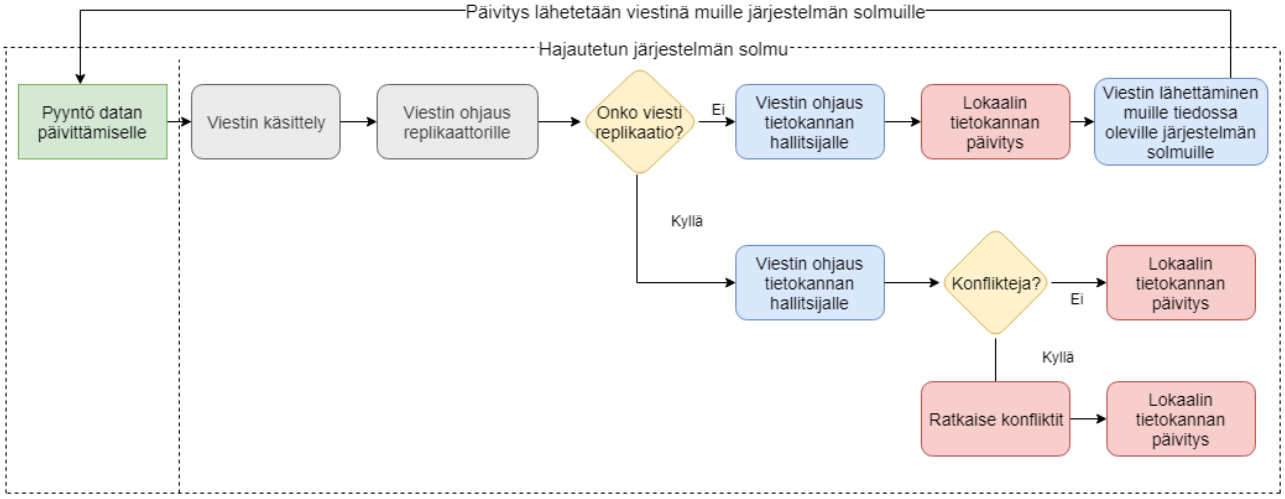
Kuten taustareplikaatio, myös konfliktien ratkaiseminen on tässä mallissa toteutettu deterministisesti, joten kaikki järjestelmän solmut kykenevät itse päättelemään miten konfliktit tulisi ratkaista ilman kommunikointia muiden solmujen kanssa. Konfliktien ratkaisussa päättely perustuu ennalta määriteltyyn sääntöjoukkoon. Nämä ovat jokainen yksinkertaisten ehtojen avulla toisiaan pois sulkevia sääntöjä. Yhtenä sääntönä voisi olla, että viimeisin päivitys voittaa, jolloin vertailtaisiin aikaleimoja. Tilanteissa, joissa sääntö ei pysty konfliktia ratkaisemaan (esimerkiksi jos aikaleimat ovat täsmälleen samoja), tarkastellaan konfliktissa olevia tietoja seuraavan säännön avulla, kunnes konflikti lopulta saadaan ratkaistua. Sääntöjä voi olla useita, mutta tärkeintä on, että lopulta pystytään jokaisessa tilanteessa ratkaisu tekemään. Automaattisen konfliktien ratkaisun tarkoituksena on mahdollistaa järjestelmän itsenäinen toiminta, mutta se ei kykene ottamaan kantaa päivitysten sisältöjen oikeellisuuteen konfliktin ratkaisussa, joten konflikteista tulee myös ilmoittaa järjestelmän käyttäjälle tai ylläpitäjälle.

Jotta konfliktit pystytään ratkaisemaan deterministisesti, täytyy konfliktissa olevat tietueen eri versiot kyetä erottelemaan toisistaan. Tähän yksilöintiin voidaan käyttää tietoa siitä, mikä solmu kyseistä tietoa on päivittänyt. Tietoa voidaan käyttää konfliktien ratkaisemisessa yhdistämällä se muihin tietoihin, kuten aikaleimaan tai revisiotietoon. Konfliktien ratkaisusäännöt sekä konfliktissa olevien tietueen versioiden tunnistamiseen soveltuvat tiedot riippuvat datan luonteesta. Alaluvuissa 6.2.5 ja 6.2.6 käsitellään, miten konfliktien hallinta ja ratkaiseminen on toteutettu edellä kuvatulla tavalla kohdejärjestelmään

6.1.7 Tapahtumaperustainen replikaatio

Tapahtumaperustainen replikaatio on päivitysten levitysmalli, joka nimensäkin mukaisesti tapahtuu aina onnistuneen operaation jälkeen. Tapahtumaperustainen replikaatio aktivoituu, kun järjestelmää pyydetään lisäämään tai päivittämään dataa. Päivitykset tehdään ensin lokaaliin tietokantaan ja samat muutokset lähetään operaation onnistuessa kaikille saatavilla oleville solmuille. Replikaattorin tehtävänä on tapahtumaperustaisessa replikaatiossa tietokantaoperaation onnistumisen jälkeen vain kopioida ja lähettää kyseinen päivitys kaikille solmuille, jotka sillä hetkellä ovat järjestelmään rekisteröityneet. Tähän viestiin lisätään mukaan tieto siitä, mikä solmu dataa on muuttanut ja tätä voidaan käyttää apuna konfliktitilanteissa. KUVIO 6 esitellään tapahtumaperustaisen replikaatioviestin käsittelyä järjestelmässä. Sama päättelyketju käydään läpi kaikissa solmuissa, mukaan lukien solmu, jossa dataa alun perin päivitettiin.

Erottavana tekijänä alkuperäisessä päivityspyynnössä ja replikaatioviestissä on viestiin liitetty tieto siitä, onko viesti replikaatio, ja jos se on, tehdään sille konfliktitarkastus ennen tietokantaan kirjoittamista. Replikaatioviestejä ei myöskään enää replikoida eteenpäin. KUVIO 6 viestien käsittelyä kuvaavan vuokaavion vaiheet on värikoodattu vastaamaan KUVIO 5 komponentteja. Vaiheen väri kertoo operaatiosta vastuussa olevan komponentin.



KUVIO 6 Vuokaavio tapahtumaperustaisesta replikaatiosta

6.1.8 Taustareplikaatio

Taustareplikaatio eroaa tapahtumaperustaisesta replikaatiosta monella tavalla. Sen tarkoitus on toimia korjaavana ja varmistavana toimenpiteenä, kun taas tapahtumaperustaisen replikaation tehtävänä on jakaa päivityksiä optimistisesti. Taustareplikaatiolla tarkoitetaan ajoittain tapahtuvaa toimenpidettä, jonka tehtävänä on etsiä mahdollisia eroavaisuuksia datassa järjestelmän eri solmujen välillä. Taustareplikaatio ei ole siis käyttäjän tai tietokannan ylläpitäjän aloittama toimenpide, vaan taustalla tapahtuva prosessi.

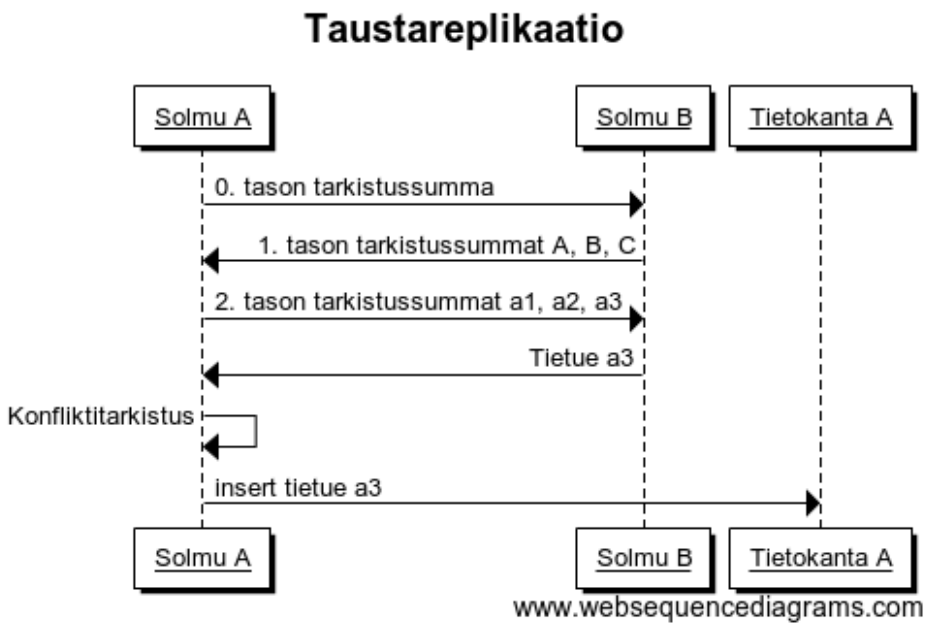
Taustareplikaation voi aloittaa kaksi eri tapahtumaa: (1) Järjestelmään liittyy uusi solmu tai solmu palaa takaisin verkkoon oltuaan tavoittamattomissa tai (2) ajoitettu tarkistus. Molemmat tapahtumat aiheuttavat samanlaisen viestiketjun, joka alkaa tarkistussumman lähettämisestä kaikille solmuille, jotka ovat kyseisellä hetkellä järjestelmään liittyneenä. Muiden solmujen tilasta ei ole reaaliaikaista tietoa, sillä kaikki viestit lähetetään olettaen, että yhteydet solmujen välillä ovat yksisuuntaisia. Tästä syystä taustareplikaatio ei takaa onnistumista jokaiselle synkronointitapahtumalle vaan antaa lupauksen lopulta yhtenevästä tilasta. Solmut aloittavat kaikki kyseisen prosessin itsenäisesti, joten taustareplikaatiotapahtumia saattaa olla useita käynnissä samanaikaisesti.

Taustareplikaatioprosessin viestien välitys koostuu pääasiallisesti eritasoisten tarkistussummien lähettämisestä. Tarkistussummien avulla replikaattorit pystyvät havaitsemaan mitkä tietueet eroavat toisistaan, mutta eivät pysty päättelemään millä tavalla. Yksittäisen taustareplikaatiotapahtuman viimeinen

vaihe onkin lähettää ketjun aloittaneelle solmulle oma versio eroavasta tietueesta, jonka perusteella vastaanottava solmu pystyy päättelemään miten pitää toimia. Tarkistussummia lähetetään edes takaisin varsinaisen datan sijasta, kunnes saadaan selville, mikä tai mitkä tietueet eroavat, jotta voidaan minimoida verkon kuormitus ja turha datan prosessointi. Tarkistustasot ja niiden määrä määräytyy kohdeympäristön ja siellä käsiteltävän datan rakenteen perusteella.

Taustareplikaatio lähtee siis liikkeelle tilanteesta, jossa järjestelmän solmu lähettää kaikille muille saatavilla oleville solmuille korkeimman tasoisen tarkistussummansa. Seuraavassa vaiheessa vastaanottajat vertailevat saatua tarkistussummaa summaan, joka on laskettu vastaanottajan päässä olevasta datasta. Jos tarkistussummat vastaavat toisiaan, loppuu prosessointiketju tähän, eikä ensimmäisen viestin lähettäjälle lähetetä minkäänlaista vastausta. Jos taas tarkistussummat eroavat toisistaan, lähetetään takaisin seuraavan tason tarkistussummat. Tätä toistetaan, kunnes on saavutettu riittävä tarkkuus, jotta voidaan erottaa tietueet, joissa on eroavaisuuksia solmujen välillä. Kun eroavat tietueet on löydetty, lähetetään ne ketjun aloittaneelle solmulle, joka yrittää lisätä ne omaan tietokantaansa. Lisäyksen yhteydessä tehdään tarkastelu konfliktien varalta. Kaikki solmut osaavat ratkaista konfliktit itsenäisesti ennalta määriteltyjen ratkaisusääntöjen perusteella, joten tässä vaiheessa ei enää tarvita kommunikointia solmujen välillä.

KUVIO 7 kuvataan sekvenssikaaviolla solmujen välistä viestinvälitystä taustareplikaation aikana. Kuviossa esitetään yhden solmun aloittama viestiketju. Kaikki solmut voivat aloittaa vastaavanlaisen ketjun samanaikaisesti. Viestiketjut eivät siis ole toisistaan riippuvaisia. Kun solmu vastaanottaa viestin, se päättelee viestin tason ja tarkistussummien perusteella tarvittavat toimenpiteet. Mikäli tarkistussummat eivät täsmää, lähetetään joko seuraavan tasoinen tarkistussumma tai lopulta itse tietue. Viestiketjun eteneminen ja viesteihin vastaaminen perustuu deterministiseen toimintaan. Yksisuuntaiset yhteydet hankaloittavat kommunikaatiota, joten deterministisellä tavalla toteutettuna voivat kaikki solmut hoitaa yhden viestiketjun operaation kerrallaan ja lähettää mahdollisen vastauksen sokkona takaisin. Jos vastausta lähetettyyn viestiin ei tule, ei ole löydetty eroavaisuuksia tai on tapahtunut virhetilanne. Lähettäjä ei kuitenkaan kykene varmistamaan onko viesti mennyt onnistuneesti perille, joten myöskään vastausta ei odoteta erikseen. On siis mahdollista, että jokaisella kerralla viestiketju ei mene loppuun asti. Tämä on ratkaistu sillä, että taustareplikaation aloittavia viestejä lähetetään ajoitetusti, luottaen siihen, että jossain vaiheessa operaatio saadaan suoritettua loppuun. Jos solmujen välillä ei ole olemassa kahta erisuuntaista yhteyttä, ei taustareplikaatio kykene tarkistusta tekemään, sillä toisella solmulla ei ole tapaa vastata ketjun aloittaneelle solmulle. KUVIO 8 esitetään yksinkertaistettu tietorakenne, joita solmut ylläpitävät. Tietorakenne on hierarkkinen ja jokaiselle eri tasolle on laskettu oma tarkistussumma, johon on sisällytetty kaikki alemmat tasot. Kaikki muutokset heijastuvat siis ylöspäin tarkistussummissa, joten kaikki muutokset datassa huomataan. Jokainen tarkistustaso karsii kandidaatteja eroavaisuuksille, joten lopulta päädytään lähettämään verkon yli vain ne tietueet, joissa on eroa.



KUVIO 7 Taustareplikaation yksittäinen viestiketju

Solmu A

```
root 0x3473DAB1 (tason 0 tarkistussumma) *
  |
  |--A  0x13092585 (tason 1 tarkistussumma) *
  |   |
  |   |-a1  0x6CE14823 (tason 2 tarkistussumma)
  |   |-a2  0xF5E81999
  |   |-a3  0x82EF290F *
  |
  |--B  0xC85CD159
  |   |
  |   |-b1  0x6E646C97
  |   |-b2  0xF76D3D2D
  |
  |--C   0x451FAC5B
      |
      |-c1  0x3E611DAB
      |-c2  0xA7684C11
      |-c3  0xD06F7C87
      |-c4  0x4E0BE924
```

Solmu B

```
root  0xE25DCD0C (tason 0 tarkistussumma) *
  |
  |--A  0x8E95B930 (tason 1 tarkistussumma) *
  |   |
  |   |-a1  0x6CE14823 (tason 2 tarkistussumma)
  |   |-a2  0xF5E81999
  |   |-a3  0x5321245D *
  |
  |--B  0xC85CD159
  |   |
  |   |-b1  0x6E646C97
  |   |-b2  0xF76D3D2D
  |
  |--C   0x451FAC5B
      |
      |-c1  0x3E611DAB
      |-c2  0xA7684C11
      |-c3  0xD06F7C87
      |-c4  0x4E0BE924
```

* Tarkistussummat eroavat toisistaan solmujen välillä

KUVIO 8 Esimerkki tietorakenteesta

6.2 Replikoinnin toteutus kohdejärjestelmässä

Suunnittelumallin mukainen replikaatio on kohdejärjestelmässä toteutettu alustalle, jossa on jo olemassa omat mekanismit viestien välittämiselle järjestelmässä niin sisäisesti, kuin myös solmujen välillä. Alusta osaa myös huolehtia viestijonoista ja muista itse viestien välittämiseen liittyvistä toiminnallisuuksista, joten niihin ei mennä syvällisemmin tässä tutkielmassa. Replikaatio tapahtuu replikaattori-väliohjelmiston tietokannan hallintakomponentin avulla, jotka on toteutettu C++-kielellä. Tietokantarajapintana toimii ODBC (Open DataBase Connectivity), sillä tutkielman kirjoitushetkellä se tukee useamman eri valmistajan tietokantatuotteita ja on yhteensopiva niin Windows-, macOS-, kuin Linux käyttöjärjestelmien kanssa. Seuraavissa alaluvuissa kuvataan, miten replikointi on toteutettu kohdejärjestelmässä, sekä miten sen toimivuus on todennettu.

6.2.1 Viestien käsittely

Jokaisen solmun viestinkäsittelyssä replikaattori-väliohjelmisto toimii välietappina pyynnöissä, joilla haetaan tietokannasta dataa tai muutetaan sen tilaa. Näiden viestien käsittely tapahtuu yhdessä säikeessä, jotta vältytään tilanteilta, joissa yhden solmun sisällä olisi mahdollista luoda konfliktitilanne rinnakkaisen viestien käsittelyn ja tietokannan operoinnin takia. Kaikki solmulle saapuvat viestit noudetaan samasta jonosta, oli kyseessä sitten replikaatioviesti tai lokaali viesti, joten yksi solmu ei voi samanaikaisesti suorittaa kahta samaa dataa muuttavaa operaatiota. Tämä vaikuttaa myös suorituskykyyn, sillä replikaatioviestit ja paikallisesti tehdyt operaatiot kilpailevat saman säikeen suoritusajasta. Saman jonon käyttäminen kaikille viesteille kuitenkin suoraviivaistaa viestien käsittelyä ja karsii pois konfliktitilanteita, sillä kaikki viestit käsitellään samanarvoisina ja omalla vuorollaan.

6.2.2 Datan rakenne ja versioiden hallinta

Tässä toteutuksessa tietokantaan tallennettava data on revisioitua. Data on revisioitu, jotta kaikki muutokset dataan säilyisivät tallessa ja historia olisi myöhemmin analysoitavissa, joka on tarpeellista kohdejärjestelmän toiminnan kannalta. Tämä tarkoittaa sitä, että yhdenkään tietokannan rivin sisältö ei muutu sen olemassaolon aikana, vaan päivitys tiettyyn tietueeseen tarkoittaa uuden revision, eli rivin, luomista. Tietokantaan tallennettaville tietueille annetaan kaikille oma UUID (Universally Unique Identifier), jokaiselle tietueen revisiolle kasvava revisionumero sekä kyseisen revision luoneen solmun yksilöllinen prioriteetti ja identiteetti. Näiden parametrien avulla pystytään yksilöimään jokainen revisio ja jokaisen revision mahdolliset konfliktissa olevat eri versiot. Yksisäikeisen viestien käsittelyn ansiosta voidaan varmistaa, että lokaalit päivitykset dataan tehdään aina järjestetysti ja tietueen revisionumero kasvaa jokaisella päivityksellä. Koska järjestelmä huolehtii siitä, että samaan tietueeseen ei

voida tehdä yhdessä solmussa kahta riviä samalla revisionumerolla, muodostuvat konfliktit ainoastaan tilanteissa, joissa useampi solmu tekee päivityksen samaan dataan ennen kuin muutokset on saatu replikoitua. Kaikki solmut pystyvät siis tekemään vain yhden version tietystä revisiosta, joten pahimmassa tapauksessa tietystä revisiosta voi olla yhtä monta versiota, kun järjestelmässä on solmuja päivityksen hetkellä.

6.2.3 Vanhojen revisioiden poistaminen

Tietueiden vanhempia revisioita poistetaan ajoittain tietokannasta automaattisesti. Poisto tietokannasta tehdään vasta, kun revisio on vanhentunut. Revisio on vanhentunut silloin, kun sen aikaleima ja revisionumero ovat riittävän vanhoja. Pelkkä aikaleima ei riitä oikeuttamaan revisioiden poistamista, sillä aika ei ole kyseisenlaisessa hajautetussa järjestelmässä luotettava ja täysin synkronoitavissa oleva määre. Tietueista säilytetäänkin aina tietty vähimmäismäärä revisioita tietokannassa, mutta revisiot, jotka ovat riittävän vanhoja niin aikaleimaltaan kuin myös revisionumeroltaan, voidaan poistaa. Vanhentunutta dataa ei myöskään replikoida, ettei toinen solmu samanaikaisesti poista dataa, jota toinen replikoi sille takaisin.

6.2.4 Taustareplikaatio

Taustareplikaation viestinvälitys on toteutuksessa nelijakoinen. (1) Ensimmäisellä tasolla lähetetään muille solmuille tarkistussumma, joka on laskettu kaikista tietokannassa olevista tiedoista. Tarkistussummat lasketaan aina rivin lisäyksen yhteydessä. Taustareplikaatioviestit aiheuttavatkin jokaisessa solmussa ajoittain vain yhden CRC32-tarkistussumman vertaamista vastaavaan, jos tapahtumaperustainen replikaatio on onnistuneesti jakanut aiemmat päivitykset tai dataan ei ole tullut muutoksia edellisen tarkistuksen jälkeen. (2) Toinen taustareplikaation tarkistustaso on jaoteltu taulukohtaisesti, jolloin replikaattori lähettää erillisinä viesteinä jokaisen taulun tarkistussumman tarkistusketjun aloittaneelle solmulle. Toisen vaiheen viesti lähetetään vain siinä tapauksessa, että ylimmän tason tarkistussummat eivät täsmänneet. Tämän tarkistelun avulla tapahtuman osapuolet saavat selville taulun tai taulut, joiden data eroaa toisistaan solmujen välillä. (3) Kolmas taustareplikaation taso on tietuekohtainen tarkistustaso, eli yhteenlaskettu tarkistussumma jokaisesta tietueen revisiosta ja revision mahdollisista eri versioista. Replikaattori lähettää taulusta, jonka tarkistussummat eivät täsmänneet, kaikkien tietueiden UUID:t ja niitä vastaavat tarkistussummat tapahtuman toiselle solmulle. Tämän tarkistusvaiheen seurauksena tapahtumaan osallistuvat solmut kykenevät selvittämään, mikä tai mitkä tietueet eroavat toisistaan tai mahdollisesti puuttuvat toiselta osapuolelta. (4) Neljäs taustareplikoinnin vaihe on lähettää eroavien tietueiden kaikki revisiot ketjun aloittaneelle solmulle, joka pystyy niiden perusteella päättelemään tarvittavat toimenpiteet.

Taustareplikaatiotapahtuman aloittamisesta on vastuussa replikaattorikomponentti, jolle on lisätty replikaatiotapahtuman aloittamisesta vastaava työsäie. Tämän säikeen tehtävänä on aloittaa ennalta määritellyn ajastuksen mukaisesti edellä kuvattuja datan yhteneväisyyden tarkistusketjuja lähettämällä oman solmunsa korkeimman tason CRC32-tarkistussumma kaikille järjestelmään rekisteröityneille muille solmuille. On mahdollista, että data muuttuu sillä välin, kun toinen solmu käsittelee lähetettyä tarkistussummaa, jolloin tarkistussummien vertailu ei enää kerro todellista tilannetta solmujen yhteneväisyydestä. Data saadaan kuitenkin lopulta saatettua yhtenevään tilaan, vaikka se ei tapahtuisi ensimmäisellä tarkistuskierroksella.

6.2.5 Konfliktien hallinta

Konfliktien havaitseminen ja lukuoperaatioissa eri revision versioiden järjestäminen tapahtuu tietokannanhallintakomponentissa. Tämän komponentin vastuulla on tarkistaa aiheuttaako saatu päivityspyyntö konfliktin jo olemassa olevan datan kanssa, sekä huolehtia siitä että lukuoperaatioissa palautetaan aina viimeisin ja voittava revisio sitä pyytäneelle taholle. Jos replikaationa saapunut päivitys on konfliktissa lokaalin datan kanssa, otetaan kaikki konfliktissa olevan revision versiot talteen ja ilmoitetaan konfliktista järjestelmän käyttäjälle. Käyttäjä voi tämän jälkeen manuaalisesti suorittaa korjaavia toimenpiteitä, jos automaattinen konfliktin ratkaisu ei saavuttanut tyydyttävää lopputulosta datan sisällön kannalta. Kuitenkin järjestelmän autonomisen toiminnan kannalta käyttäjän toimet eivät ole välttämättömiä konfliktin ratkaisussa, sillä tietokantaa hallitseva komponentti kykenee valitsemaan voittajan itsenäisesti. Seuraavassa alaluvussa kuvataan konfliktien ratkaisusäännöt, joita käytetään voittajan valinnassa.

6.2.6 Konfliktien ratkaisusäännöt

Tässä toteutuksessa data on revisioitua, joten se näkyy myös konfliktin ratkaisun säännöissä. Konfliktit kyetään ratkaisemaan käyttämällä vain kahta yksinkertaista sääntöä. (1) Ensimmäinen konfliktin ratkaisusääntö on, että suurin reivisionumero voittaa aina. Eli jos jokin muu kuin viimeisin revisio on konfliktissa, ei tehdä muuta kuin otetaan talteen kaikki revision versiot ja hylätään duplikaatit. Konfliktissa olevat häviävät versiot revisiosta otetaan talteen, sillä vaikka automaattinen konfliktien ratkaisu pystyykin valitsemaan voittajan, saattaa joskus olla tarpeen kumota tuo päätös manuaalisesti. Jos tarkastelussa häviävän version data olisikin ollut käyttäjän kannalta hyödyllisempää, voi käyttäjä luoda tiedosta uuden revision hävinneen version sisällöllä. (2) Toinen sääntö on, että revisio, jonka on luonut korkeamman prioriteetin solmu, voittaa. Solmuille annetaan käynnistyessä yksilölliset prioriteetit, joita vertaamalla voidaan konflikti ratkaista. Toista sääntöä käytetään myös hyväksi silloin, kun pyydetään historiadataa, jolloin voidaan myös vanhan tiedon lukuoperaatiossa päätellä, mikä revision versio on oikeellisin. Nämä kaksi sääntöä riittävät rat-

kaisemaan konfliktit, sillä konfliktissa olevia eri versioita yhdestä revisiosta voi muodostua vain eri solmujen välillä ja solmujen prioriteetit ovat yksilöllisiä. Konfliktit pystytään ratkaisemaan edellä mainittuja sääntöjä soveltaen kolmen yksilöivän parametrin avulla: (1) UUID, (2) revisionumero ja (3) solmun prioriteetti.

6.3 Replikaation testaus

Kaikki replikaation testit on toteutettu Lenovon P50-kannettavalla, jota käytettiin pyörittämään CentOS 7-virtuaalikoneita VMware-virtualisointialustalla. Kullakin virtuaalikoneella oli käytössään 2 Gt keskusmuistia ja kaksi virtuaalista prosessoriydintä. Pääasiallisesti testit ajettiin kolmen solmun kokoonpanolla, mutta suunniteltu malli mahdollistaa myös useamman solmun käytön. Osaan testeistä pystytettiin myös lisäksi virtuaalikone, jossa pyöri WANem OS, jonka kautta solmujen välinen liikenne reititettiin. Testausjakson aikana nousi esiin myös joitain bugeja ja kehittämisen kohteita, jotka liittyivät konfliktien hallintaan ja duplikaattien käsittelyyn. Nämä korjattiin välittömästi, jonka jälkeen ajettiin samat testit uudestaan, jotta korjaus voitiin todentaa. Liitteessä 3 on kuvattu kaikki testausjakson aikana tehdyt testitapaukset. Testitapaukset on luotu siten, että ne pystytään myöhemmin automatisoimaan osaksi järjestelmän automaatiotestausta. Seuraavissa alaluvuissa käsitellään ajettuja testejä ja pohditaan niistä saatuja tuloksia.

6.3.1 Replikaation testaus testisovelluksella ilman verkkohäirintää

Ensimmäiseksi replikaatiota testattiin ympäristössä, jossa yhteydet eivät muutu testin aika, eikä yhteyksiä häiritty tässä vaiheessa muillakaan tavoin. Näillä testeillä haettiin viitteitä suorituskyvystä suotuisissa olosuhteissa, jotta voidaan vertailla miten paljon eri tekijät, kuten ei-pysyvät yhteydet, vaikuttavat suorituskykyyn. Näiden testien avulla myös todennettiin replikaation ja konfliktien ratkaisun toimintalogiikka erilaissa dataan liittyvissä erikoistapauksissa.

Osa testeistä ajettiin ilman keskeytyksiä testisovelluksella, jolloin lyhyessä ajassa kaikkiin solmuihin ajettiin rinnan tuhansia operaatioita ja tarkkailtiin, onnistuuko replikaatio solmujen välillä. Näissä testeissä ei noussut esille ongelmia suorituskyvyn tai replikoinnin luotettavuuden suhteen, sillä tapahtumaperustainen replikaatio kykeni välittämään päivitykset välittömästi operaation onnistumisen jälkeen muille solmuille.

Osa testeistä taas ajettiin siten, että tapahtumaperustainen replikaatio oli estettynä kokonaan. Kaikki tiedon synkronointi tapahtui viiveellä taustareplikaation toimesta, koska tieto muutoksista ei päätynyt muille solmuille tapahtumaperustaisen replikaation toimesta. Näiden testien tarkoituksena oli mallintaa tilannetta, jossa solmu on pitkäaikaisesti muiden solmujen saavuttamattomissa, jolloin dataan on saatettu tehdä paljon päivityksiä ilman synkronointia.

Taustareplikaatiotapahtumien välillä kaikkiin solmuihin ajettiin tuhansia operaatioita ja testien aikana taustareplikaation synkronointiväli oli asetettu tapahtumaan 100 sekunnin välein. Tämä nosti esiin suorituskyvyn alenemisia ajoittain, sillä kaikki solmut käsittelivät tuhansien tietojen synkronointia samanaikaisesti. Rajapinnan kautta ajetuissa operaatioissa näkyi jopa sekuntien viiveitä, koska operaatioiden suoritusjonot täyttyivät taustareplikaation tapahtumista. Solmut saivat lopulta taustareplikaation avulla synkronoinnin tehtyä. Jos operaatioita ei enää tullut lisää, tarvittiin yleensä 1-3 tarkistuskierrosta, jotta kaikki solmut olivat yhteneviä.

Replikaatiota testattiin myös ajamalla pidempikestoisia testikokonaisuuksia suotuisissa olosuhteissa, sillä sekunneissa tuhansien operaatioiden ajaminen ei vastaa lopullista käyttöä, vaikka sillä voidaankin montaa asiaa hyvin testata. Pidempiaikaisissa testeissä kaikkiin solmuihin ajettiin operaatioita satunnaisella 5-60 sekunnin päivitysvälillä ja testejä ajettiin kutakin vuorokauden ajan. Näitä testejä tehtiin kaksi, joista toisessa oli tapahtumaperustainen replikaatio poistettu käytöstä. Pidempiaikaisten testiajojen jälkeen data solmujen tietokannoissa oli yhtenevää. Joissakin tilanteissa testeissä, joissa tapahtumaperustainen replikaatio oli estetty, taustareplikaatio tarvitsi useamman kierroksen ennen kuin data oli yhtenevää. Tämä johtui siitä, että solmut A, B ja C kaikki olivat keskenään epäyhteneviä taustareplikaatioprosessien alkaessa ja solmu A aloitti synkronoinnin solmun B kanssa, jolloin ne saivat datan yhtenäistettyä. Tämän jälkeen solmu C aloitti synkronoinnin solmun A kanssa, jonka jälkeen solmu A ei ollut enää yhtenevä solmun B kanssa ja niin edelleen. Kuitenkin joka kierroksella epäyhtenevien tietueiden määrä väheni ja lopulta kaikki solmut saavuttivat keskinäisen yhteneväisyyden. Kaikkiaan testit, jotka ajettiin jatkuvasti pystyssä olevia yhteyksiä käyttäen, antoivat viitteitä suorituskyvystä suotuisissa verkko-olosuhteissa ja niillä pystytiin vahvistamaan replikaation toimintalogiikka. Estämällä tapahtumaperustainen replikaatio, saatiin selvitettyä, millaisia määriä epäyhteneväisyyksiä taustareplikaatio pystyy käsittelemään ennen kuin se alkaa näkymään suorituskyvyssä.

6.3.2 Replikaation testaus testisovelluksella verkkohäirinnän kanssa

Seuraavana vaiheena testauksessa oli ottaa testaukseen mukaan WANem-työkalu. WANem pystytettiin omaan virtuaalikoneeseen, jonka kautta reititettiin kaikki liikenne kahden solmun välillä. Testejä tehtiin muuttamalla viivettä yhteydessä sekä konfiguroimalla WANem katkomaan yhteyttä satunnaisesti ajon aikana. Lisäksi WANem konfiguroitiin hukkaamaan ja korruptoimaan paketteja, mutta lähinnä edellä mainituista ominaisuuksista keskityttiin yhden katkomiseen, sillä se tuo suurimmat haasteet replikaation kannalta lopullisessa käyttöympäristössä.

Verkkohäirinnän kanssa pitkäkestoisia testejä ajettiin yhteensä yli seitsemän vuorokauden edestä ja tämän lisäksi ajettiin lyhytkestoisempia testejä, joilla testattiin tiettyjä ominaisuuksia. Yhteyden katkeamisen jälkeen solmut kykenivät palauttamaan yhteyden ja taustareplikaatio saattamaan solmut yhtene-

vään tilaan. Näissä testeissä taustareplikaation suorituskykykään ei noussut erityisemmin esille, sillä osan ajasta yhteys oli eheä ja katkokset olivat ajoittaisia, jolloin tapahtumaperustainen replikaatio kykeni välittämään ison osan päivityksistä välittömästi operaation jälkeen.

6.3.3 Manuaaliset testit

Liitteessä 3 kuvattujen manuaalisten tietokannan sisällön manipuloinnilla tehtyjen testien tarkoitus oli varmistaa, että replikaation konfliktien havainnointi ja ratkaiseminen toimii kuten on suunniteltu. Näissä testeissä luotiin erilaisia konfliktitilanteita, joita saattaa tulla vastaan lopullisessa käytössä ja seurattiin miten solmut konfliktitilanteet hoitavat. Vaikka muissakin testeissä saatiin konflikteja luotua testisovelluksen avulla, pystyttiin näillä testeillä luomaan konflikteja toistetusti ja halutulla tavalla. Solmut pystyivät kaikki konfliktit ratkaisemaan, joita testeissä luotiin, mutta kolmen solmun kokoonpanolla tietyissä tilanteissa taustareplikaatiotapahtumia tarvittiin useita, joka johtui vastaavanlaisesta tilanteesta kuin alaluvussa 6.3.1 mainittu esimerkkitilanne kolmen solmun välisestä synkronoinnista. Kaikkiaan manuaaliset testit osoittivat, että taustareplikaatio ja konfliktien ratkaisu toimii halutulla tavalla ja kykenee ratkaisemaan rinnakkaisuuden aiheuttamat konfliktit ja raportoimaan niistä rajapinnan kautta kuunteleville asiakkaille.

7 POHDINTA

Tässä tutkielmassa esitelty suunnittelumalli ja sen pohjalta tehty toteutus hajautetun tietokannan replikoimiseksi on tehty toimeksiantona ja siten myös toimeksiantajan tarpeet edellä. Kuitenkin suunnittelumalli on hyödynnettävissä myös muissa järjestelmissä, joiden toimintaympäristöissä on vastaavanlaisia piirteitä ja rajoitteita. Tarkoituksena oli myös luoda ratkaisu, joka on mahdollisimman riippumaton tietyn valmistajan tietokantatuotteista. Tämän tutkielman mallissa replikaatio toteutetaan tietokantaa käyttävän sovellusohjelman puolella, joten sen näkökulmasta ei ole merkitystä mikä tietokannanhallintajärjestelmä alemmalla kerroksella on käytössä. Tietokantatuoteriippumattomuus voidaan toteuttaa tietokantaa hallitsevassa komponentissa käyttämällä toteutuksessa tietokantarajapintaa, kuten toimeksiannossa käytetty ODBC-rajapinta, joka tarjoaa yhteisen rajapinnan tietokannan operointiin asiakassovelluksesta. Kaikki tietokantaoperaatiot tehdään käyttämällä tämän rajapinnan kutsuja, jotka ovat yhteisiä kaikille sen tukemille eri tuotteille.

Kohdejärjestelmän rajoitteiden takia tässä mallissa ei voitu ottaa huomioon kaikkia luvussa 4 mainittuja tekijöitä, joita tulisi suunnitteluvaiheessa pohtia, kuten tiedon yhteneväisyyden tärkeyttä tietokantaa käyttäville sovellusohjelmille, sillä pessimistinen replikaatio ei ole järkevästi toteutettavissa, vaikka datan yhteneväisyys olisikin kriittistä kaikilla ajan hetkillä. Kuitenkin suunnittelumallin mukainen kaksijakoinen replikaatio pyrkii paikkaamaan lopulta yhtenevän strategian puutteita datan eheyden varmistamisessa tarjoten samalla myös pienempiä prosessointi- ja kyselyviiveitä tietokantaoperaatioihin, joita saavutetaan rinnakkaisella tietokantojen operoinnilla eri solmuissa.

Replikaation testausvaiheessa nousi esiin tilanteita, joissa solmun lokaalit operaatiot joutuivat odottamaan replikaatiosta aiheutuneita viestejä, mikä näkyi, suurilla tiuhaan ajetuilla datamäärillä, ajoittaisena suorituskyvyn alenemisena. Tämä on yksi niistä kompromisseista, joita joudutaan tekemään, kun hajautetun tietokannan tulee kyetä replikoimaan muutokset yksisuuntaisten yhteyksien yli ja mahdollistaa tietokantojen päivitys ilman yhteyttä muihin solmuihin. Kuitenkaan testaukseen käytetty laitteisto ei vastaa suorituskyvyltään lopullista ympäristöä, jossa huomattavasti testeissä käytettyä kannettavaa tietokonetta

suorituskykyisemmät palvelinkoneet ovat vastuussa operaatioiden suorittamisesta. Testeissä ajettiin myös yhdellä fyysisellä kannettavalla tietokoneella useampaa virtuaalikonetta, jolloin eri solmut kilpailivat samoista prosessointiin saatavilla olevista resursseista. Testeissä myös ajettiin operaatioita paljon lopullista käyttöä suuremmalla intensiteetillä. Testauksen pohjalta voidaankin siis olettaa toteutuksen suorituskyvyn olevan riittävä lopulliseen käyttöön toimeksiantajan kohdejärjestelmässä. Replikaatio kykeni myös saattamaan tietokannat lopulta yhtenevään tilaan, vaikka verkkoliikennettä häirittiin ja manuaalisesti aiheutettiin erilaisia konflikteja ja epäyhteneväisyyksiä dataan.

Monessa P2P-järjestelmässä tapahtumaperustainen replikaatio olisi riittävä tapa toteuttaa replikointi, mutta tässä toteutuksessa taustareplikaation rooli korostuu, sillä on todennäköistä, että tapahtumaperustaiset replikaatioviestit eivät aina saavuta kaikkia solmuja. Tästä syystä testijaksolla testattiin replikaatiota myös ottamalla tapahtumaperustainen replikaatio kokonaan pois käytöstä, jolloin pystyttiin luomaan testejä, jotka vastaavat pahinta mahdollista tilannetta. Deduktiivinen taustareplikaatio osoittautui testeissä toimivaksi ratkaisuksi, vaikkakin joissain tilanteissa synkronointi vaati useamman tarkistuskierroksen.

Mikäli solmujen välillä on yhteydet toiminnassa siten, että tapahtumaperustainen replikaatio kykenee jakamaan kaikki päivitykset muille solmuille, on taustareplikaatiotapahtuma hyvin kevyt luonteeltaan, sillä se aiheuttaa vain yhden tarkistussumman vertaamista toiseen. Näissäkin tilanteissa kuitenkin taustareplikaatiota tarvitaan varmistamaan tietojen yhteneväisyys, sillä sokkona lähetetyt tapahtumaperustaiset replikaatiopaketit saattavat hukkua matkalla, koska niiden perille saapumisesta ei saada tietoa lähettävään päähän. Mikäli eroavaisuuksia datassa on paljon ja taustareplikaatioviesteissä lähetettävä revisiomäärä massiivinen, voidaan taustareplikaation lisätä vielä yksi vaihe, jossa vertaillaan revisiotasoisia eroja tai lähettää tietueet jaettuna useampaan eri viestiin. Tällä ei kuitenkaan ole taustareplikaation toimintaperiaatteen kannalta merkitystä, vaan eroja saatetaan nähdä suorituskyvyssä, mikäli suurten pakettien käsittely kuluttaa resursseja liikaa yhdellä ajanhetkellä. Taustareplikaatiossa tarkistussummien käyttämisellä eroavaisuuksien selvittämiseen parannetaan myös toteutuksen tietoturvaa, sillä taustareplikaatiossa ei solmujen välillä lähetetä tietoa, jolle ei ole oikeaa tarvetta.

Vaikka replikaation testit ajettiin enimmäkseen kolmella solmulla, mahdollistaa malli kuitenkin loogisesti rajoittamattoman solmujen määrän käytön. Yhtenä testinä ajettiinkin seitsemää solmua samanaikaisesti vuorokauden ajan. Tässä testissä kuitenkin näkyi usein se, että monissa konfliktitilanteissa solmujen välillä synkronointiin tarvittiin useita kierroksia, sillä kaikki solmut olivat pahimmillaan keskenään epäyhtenevässä tilassa ja kaikki solmut yrittivät synkronoida dataa kaikkien solmujen kesken. Vaikka malli loogisesti kykenisikin toteuttamaan taustareplikaation rajoittamattomalla määrällä solmuja, tulee kuitenkin jossain vaiheessa suorituskyvyssä raja vastaan. Kuitenkin luotu malli mahdollistaa myös erilaisten kokoonpanojen luomisen useampaa solmua käytettäessä, sillä jokaisen solmun ei tarvitse kyetä kommunikoimaan jokaisen solmun kanssa, eikä jokaisen solmun välillä tarvitse olla molempiin suuntiin

yhteyttä. Voitaisiin esimerkiksi luoda verkko, jossa on useita muutaman solmun muodostamia ryppäitä, joista jokaisesta ryppäästä yksi solmu hoitaa synkronoinnin muiden ryppäiden kanssa.

Kaikkiaan testien perusteella voidaan todeta, että tässä tutkielmassa kuvatun mallin pohjalta voidaan toteuttaa hajautetun tietokannan replikaatio käyttäen yksisuuntaisia ei-pysyviä yhteyksiä. Malli mahdollistaa myös replikaation toteuttamisen ilman riippuvuuksia tietystä tietokantatuotteesta, sillä kaikki replikaation toimintalogiikka on toteutettu tietokantaa käyttävän sovellusohjelman puolella.

8 TUTKIELMAN RAJOITTEET

Tutkielman yhtenä rajoitteena voidaan pitää suunnitellun toteutuksen epätavallista toimintaympäristöä. Toimintaympäristö asettaa tutkimuksessa esitellylle suunnittelumallille rajoitteita ja vaatimuksia, jotka saattavat heikentää mallin suorituskykyä verrattuna muihin malleihin tilanteessa, jossa kaikkia näitä rajoitteita ei ole. Toimintaympäristön, johon perustuen malli on luotu, tuomat rajoitteet rajaavat suunnittelumallin yleistettävyyttä ja sen toimivuus onkin testattu vain tutkimuksessa kuvatuissa olosuhteissa. Vaikka mallia pystyisikin hyödyntämään myös vähemmän rajoitetuissa ympäristöissä, ei tämä tutkimus tarjoa testituloksia sen soveltamisesta tai suorituskyvystä vähemmän rajatussa toimintaympäristössä verrattuna muihin malleihin. Kuitenkin suunnittelumallia voidaan käyttää viitekehyksenä suunnittelulle myös erilaisissa toimintaympäristöissä ja soveltaa sitä niiltä osin kuin se on kyseisen toimintaympäristön kannalta järkevää.

Toisena tutkimuksen rajoitteena on replikaatiototeutuksen testauksen kattavuus. Vaikka esimerkkitoteutusta testattiin simuloidussa ympäristössä ottaen huomioon monenlaisia testitilanteita, ei varsinaisesta toteutuksen loppukäytöstä ole pitkäaikaista dataa saatavilla tutkielman kirjoitushetkellä. Testien perusteella ei siis voida täysin todentaa ratkaisun lopullista suorituskykyä, sillä ei ole saatavilla dataa lopullisesta käytöstä ja testiympäristö ei vastaa suorituskyvyltään lopullista toimintaympäristöä. Kuitenkin testien avulla pystytään arvioimaan missä suhteessa suorituskyky heikkenee, kun datan tai konfliktien määrä kasvaa.

9 YHTEENVETO

Tämän tutkielman tavoitteena oli selvittää, miten hajautetun tietokannan replikointi voidaan toteuttaa yksisuuntaisia ei-pysyviä yhteyksiä käyttäen. Lisäksi suunnittelun tavoitteena oli luoda malli, joka ei perustu mihinkään tiettyyn olemassa olevan tietokannanhallintajärjestelmän tarjoamaan palveluun replikoinnin toteuttamiseksi, vaan olisi yleistettävissä eri tietokannanhallintajärjestelmiä käyttäviin toteutuksiin. Suunnittelua varten toteutettiin myös kirjallisuuskatsaus, jossa selvitettiin aikaisemman tutkimuksen perusteella hajautettuun tietokantaan ja replikaatioon liittyviä ongelmia ja ratkaisumalleja. Kirjallisuuskatsauksesta selvisi, että niin datan sijoittamisessa solmujen välillä kuin myös hajautetun tietokannan- ja replikaation suunnittelussa joudutaan tekemään kompromisseja datan saatavuuden, vikasietoisuuden ja suorituskyvyn suhteen. Kirjallisuuskatsauksessa nousi myös esille tekijöitä, jotka tuli ottaa huomioon suunnittelussa.

Kirjallisuuskatsauksen ja kohdejärjestelmän tarpeiden perusteella luotiin malli, jonka avulla voidaan toteuttaa hajautetun tietokannan replikaatio käyttäen yksisuuntaisia ei-pysyviä yhteyksiä. Suunnittelumalli toteutettiin siten, että replikaatioon liittyvä logiikka on kaikki toteutettu tietokantaa käyttävän sovellusohjelman puolella, jolloin esitellyn mallin kannalta ei ole väliä, mitä tietokannanhallintajärjestelmää käytetään. Lopuksi toteutettiin toimeksiantona Patria Aviation Oy:lle suunnittelumalliin perustuva toteutus replikaatiosta. Kyseistä toteutusta testattiin useissa erilaisissa tilanteissa, joissa verkkoa häirittiin tai luotiin muita erikoistapauksia. Testauksen pohjalta selvisi joitain tapauksia, joissa suorituskyky alenee, mutta ei kuitenkaan siten, että se muodostuisi kyseisessä toteutuksessa ongelmaksi. Replikaatio onnistui selviämään kaikista testitapauksista ja lopulta saattamaan eri solmujen tietokannat yhtenevään tilaan, joten malli todettiin toimivaksi tavaksi toteuttaa hajautetun tietokannan replikaatio yksisuuntaisia ei-pysyviä yhteyksiä käyttäen.

LÄHTEET

Abadi, D. (2012). Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. *Computer*, *45*(2), 37–42. https://doi.org/10.1109/MC.2012.33

Alonso, G. (1997). Partial Database Replication and Group Communication Primitives.

Bailis, P., & Ghodsi, A. (2013). Eventual consistency today: Limitations, extensions, and beyond. *Queue*, *11*(3), 20.

Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E., & O'Neil, P. (1995). A Critique of ANSI SQL Isolation Levels. *SIGMOD Rec.*, *24*(2), 1–10. https://doi.org/10.1145/568271.223785

Bernstein, P. A., & Goodman, N. (1981). Concurrency Control in Distributed Database Systems. *ACM Comput. Surv.*, *13*(2), 185–221. https://doi.org/10.1145/356842.356846

Bernstein, P. A., & Goodman, N. (1983). Multiversion Concurrency Control&Mdash;Theory and Algorithms. *ACM Trans. Database Syst.*, *8*(4), 465–483. https://doi.org/10.1145/319996.319998

Ceri, S., & Wiederhold, G. I. (1987). Distributed Database Design Methodologies. *Proceedings of the IEEE*, *75*(5).

Charron-Bost, B., Pedone, F., & Schiper, A. (2010). *Replication: Theory and Practice* (Vol. 5959). https://doi.org/10.1007/978-3-642-11294-2

Chen, S., Ng, A., & Greenfield, P. (2013). A performance evaluation of distributed database architectures. *Concurrency and Computation: Practice and Experience*, *25*(11), 1524–1546. https://doi.org/10.1002/cpe.2891

Daudjee, K., & Salem, K. (2006). Lazy Database Replication with Snapshot Isolation. In *Proceedings of the 32Nd International Conference on Very Large Data Bases* (pp. 715–726). VLDB Endowment. Retrieved from http://dl.acm.org/citation.cfm?id=1182635.1164189

Eager, D. L., & Sevcik, K. C. (1983). Achieving Robustness in Distributed Database Systems. *ACM Trans. Database Syst.*, *8*(3), 354–381. https://doi.org/10.1145/319989.319992

Fekete, A., Liarokapis, D., O'Neil, E., O'Neil, P., & Shasha, D. (2005). Making Snapshot Isolation Serializable. *ACM Trans. Database Syst.*, *30*(2), 492–528.

https://doi.org/10.1145/1071610.1071615

Gray, J., Helland, P., O'Neil, P., & Shasha, D. (1996). The dangers of replication and a solution. *ACM SIGMOD Record*, *25*(2), 173–182.

Guy, R., Reiher, P., Ratner, D., Gunter, M., Ma, W., & Popek, G. (1999). Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication. In Y. Kambayashi, D. L. Lee, E.-P. Lim, M. K. Mohania, & Y. Masunaga (Eds.), *Advances in Database Technologies* (pp. 254–265). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-49121-7_22

Kemme, B., & Alonso, G. (2000). A New Approach to Developing and Implementing Eager Database Replication Protocols. *ACM Trans. Database Syst.*, *25*(3), 333–379. https://doi.org/10.1145/363951.363955

Microsoft. (2017). Advanced Merge Replication - Conflict Detection and Resolution. Haettu 2.4.2019, osoitteesta https://docs.microsoft.com/en-us/sql/relational-databases/replication/merge/advanced-merge-replication-conflict-detection-and-resolution?view=sql-server-2017

Navathe, S., Ceri, S., Wiederhold, G., & Dou, J. (1984). Vertical Partitioning Algorithms for Database Design. *ACM Trans. Database Syst.*, *9*(4), 680–710. https://doi.org/10.1145/1994.2209

O'Brien, J. A., Marakas, G. M., & others. (2006). *Management information systems* (Vol. 6). McGraw-Hill Irwin.

Oracle. (2019). Conflict Resolution Concepts and Architecture. Haettu 2.4.2019, osoitteesta https://docs.oracle.com/cd/B28359_01/server.111/b28326/repconflicts.htm#i26513

Özsu, M. T., & Valduriez, P. (1991). Distributed database systems: where are we now? *Computer*, *24*(8), 68–78.

Özsu, M. T., & Valduriez, P. (2011). *Principles of distributed database systems*. Springer Science & Business Media.

Pedone, F., Guerraoui, R., & Schiper, A. (1998). Exploiting atomic broadcast in replicated databases. In D. Pritchard & J. Reeve (Eds.), *Euro-Par'98 Parallel Processing* (pp. 513–520). Berlin, Heidelberg: Springer Berlin Heidelberg.

Rahimi, S. K., & Haug, F. S. (2010). Data Distribution Alternatives. In *Distributed Database Management Systems: A Practical Approach*. IEEE. https://doi.org/10.1002/9780470602379.ch2

Sacca, D., & Wiederhold, G. (1985). Database Partitioning in a Cluster of Processors. *ACM Trans. Database Syst.*, *10*(1), 29–56.

https://doi.org/10.1145/3148.3161

Sacha, J., & Dowling, J. (2007). A Gradient Topology for Master-Slave Replication in Peer-to-Peer Environments. In G. Moro, S. Bergamaschi, S. Joseph, J.-H. Morin, & A. M. Ouksel (Eds.), *Databases, Information Systems, and Peer-to-Peer Computing* (pp. 86–97). Berlin, Heidelberg: Springer Berlin Heidelberg.

Saito, Y., & Shapiro, M. (2005). Optimistic Replication. *ACM Comput. Surv.*, *37*(1), 42–81. https://doi.org/10.1145/1057977.1057980

Schiper, A., & Raynal, M. (1996). From Group Communication to Transactions in Distributed Systems. *Commun. ACM*, *39*, 84–87. https://doi.org/10.1145/227210.227230

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2001). DatabaseSystem Concepts.

Vogels, W. (2009). Eventually Consistent. *Commun. ACM*, *52*(1), 40–44. https://doi.org/10.1145/1435417.1435432

Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., & Alonso, G. (2000a). Database replication techniques: a three parameter classification. In *Proceedings 19th IEEE Symposium on Reliable Distributed Systems SRDS-2000* (pp. 206–215). https://doi.org/10.1109/RELDI.2000.885408

Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., & Alonso, G. (2000b). Understanding replication in databases and distributed systems. In *Proceedings 20th IEEE International Conference on Distributed Computing Systems* (pp. 464–474). https://doi.org/10.1109/ICDCS.2000.840959

Wiesmann, M., & Schiper, A. (2005). Comparison of database replication techniques based on total order broadcast. *IEEE Transactions on Knowledge and Data Engineering*, *17*(4), 551–566. https://doi.org/10.1109/TKDE.2005.54

LIITE 1 TIEDON JAON MALLIEN TERMISTÖ

Termi	Lyhyt kuvaus	Tutkimukset
Tiedon jaon mallit, joissa data on täysin toisinnettua		
Peer-to-Peer	P2P-mallissa kaikki järjestelmän solmut ovat vertaisia keskenään. Solmujen hallinoivissa tietokannoissa dataan voidaan tehdä päivityksiä ja lukea kaikissa solmuissa.	Özsu & Valduriez (2011) Chen et al. (2013) Guy et al. (1999)
Update anywhere/everywhere	Update anywhere -mallissa kaikkiin järjestelmän solmuihin voidaan tehdä päivityksiä ja kaikki solmut sisältävät kopion samasta datasta.	Wiesmann et al. (2000b, 2000a) Wiesmann & Schiper (2005) Alonso (1997)
Multi-Master/multimaster replication architecture	Kuten P2P-mallissa ja Update anywhere -mallissa, myös multi-master mallissa kaikki solmut käsittelevät kaikkea dataa ja sallivat datan päivittämisen ja lukemisen kaikista järjestelmän solmuista ja kaikki päivitykset jaetaan kaikille kopioille datasta.	Saito & Shapiro (2005) Oracle (2019)
Replicated architecture	Replicated arcitechture termillä viitataan malliin, jossa järjestelmän solmut ylläpitävät kaikki samaa dataa. Tietokannat voivat olla osittain replikoituja tai täysin replikoituja.	Chen et al. (2013) Rahimi & Haug (2010)
Replicated Database Model	Replikoidussa tietokantamallissa jokainen prosessi/solmu hallitsee omaa kopiotaan samasta datasta ja päivitykset dataan jaetaan muille kopioille.	Pedone et al. (1998)
Master-Slave replication architecture	Tässä mallissa yksi tietokanta toimii hallitsevana kantana, johon tehdään päivitykset. Tähän kantaan tehdyt päivitykset jaetaan hallittaville kannoille.	Abadi (2012) Rahimi & Haug (2010)

Primary-copu	Tässä mallissa yksi tietokanta toimii ensisijaisena kantana, johon tehdään päivitykset. Tähän kantaan tehdyt päivitykset jaetaan toissijaisille kannoille.	Wiesmann et al. (2000b, 2000a) Wiesmann & Schiper (2005)
Muut tiedon jaon mallit		
Centralized model/architecture	Keskitetyssä mallissa data on fyysisesti sijoitettu yhteen pisteeseen, josta palvellaan kaikki järjestelmän osa-alueita	Chen et al. (2013) Özsu & Valduriez (1991) Özsu & Valduriez (2011) Eager & Sevcik (1983) Ceri & Wiederhold (1987) Rahimi & Haug (2010)
Partitioned database	Osioidussa tietokanta-arkkitehtuurissa data on jaettu eri solmujen kesken. Eri solmut sisältävät ylläpitävät eri dataa. Osiointi voidaan tehdä vertikaalisesti tai horisontaalisesti.	Chen et al. (2013) Rahimi & Haug (2010) Özsu & Valduriez (2011) Navathe et al. (1984) Saccà & Wiederholf (1985)
Fragmented model	Fragmentoidussa hajautetun tietokannan mallissa järjestelmän eri solmut sisältävät ja ylläpitävät eri dataa. Fragmentoitu data voidaan koostaa takaisin laajemmaksi kokonaisuudeksi. Datan fragmentointi voidaan tehdä horisontaalisesti tai vertikaalisesti.	Ceri & Wiederhold (1987) Rahimi & Haug (2010)

LIITE 2 REPLIKOINNIN TERMISTÖ

Termi	Lyhyt kuvaus	Tutkimukset
Replikointistrategiat		
Optimistiset replikaatiostrategiat		
Optimistic replication	Optimistisessa replikaatiossa luotetaan siihen, että konfliktit pystytään havaitsemaan korjaamaan. Konflikteilta ei siis pyritä välttymään samalla tavalla kuin pessimistisissä strategioissa, esimerkiksi odottamalla vahvistusta tai lukituksia käyttämällä.	Guy et al. (1999) Saito & Shapiro (2005)
Lazy replication	Laiskalla replikaatiolla tarkoitetaan replikaatiota, jossa päivitykset jaetaan vasta operaation jälkeen, kun tämä on mahdollista. Laiska replikaatio toimii sateenvarjoterminä, ja sen alle on sijoitettu usein myös tarkemmin strategiaa kuvaavat termit: Lazy Primary Copy ja Lazy Update EveryWhere.	Wiesmann et al. (2000b, 2000a) Wiesmann & Schiper (2005) Özsu valduriez (2011)
Asynchronous replication	Asynkronisessa replikaatiossa muiden kopioiden vastausta ei tarvitse odottaa, sillä replikaatioviestit lähetetään operaation jälkeen muille kopioille, joten kaikkien kopioiden päivittymistä ei taata.	Abadi (2012) Rahimi & Haug (2010) Bailis & Ghodsi (2013) Alonso (1997)
Pessimistiset replikaatiostrategiat		
Pessimistic replication	Pessimistisessä replikaatiossa varmistetaan, että muutokset tehdään kerralla kaikkiin kopioihin tai ei mihinkään niistä. Ei luoteta kykyyn havaita ja ratkaista kaikki konflikteja niiden sattuessa.	Wiesmann et al. (2000b, 2000a) Saito & Shapiro (2005)

Eager replication	Innokkaalla replikaatiolla tarkoitetaan replikaatiota, jossa päivitykset jaetaan kaikkiin tai ei mihinkään kopioon datasta transaktion aikana. Innokas replikaatio toimii sateenvarjoterminä, ja kuten laiskassa replikaatiossa, sen alle on sijoitettu usein myös tarkemmin strategiaa kuvaavat termit: Eager Primary Copy ja Eager Update EveryWhere.	Wiesmann et al. (2000b, 2000a) Wiesmann & Schiper (2005) Özsu valduriez (2011)
Conservative replication	Konservatiivisessa replikaatiossa estetään kaikki samanaikaiset päivitykset samaan dataan. Samanaikaiset päivitykset estetään, joten data on lukittuna päivityksen ajan.	Guy et al. (1999)
Synchronous replication	Synkronisessa replikaatiossa replikaatioviestin lähettänyt solmu odottaa vahvistusta, että kaikki muut kopiot ovat tehneet päivityksen. Päivitys tehdään joko kaikkiin kopioihin tai ei yhteenkään.	Abadi (2012) Rahimi & Haug (2010) Bailis & Ghodsi (2013) Oracle (2019) Alonso (1997)

LIITE 3 TESTITAPAUKSET

Testit vakailla yhteyksillä käyttäen molempia replikaatiomenetelmiä		
Testi No.	**Testin kuvaus**	Solmujen lkm.
1	Testissä ajettiin yhteen solmuun 10 000 satunnaistettua operaatiota.	3
2	Testissä ajettiin kaikkiin solmuihin rinnakkain jokaiseen 10 000 satunnaistettua operaatiota.	3
3	Testissä ajettiin kahteen solmuun kumpaankin 10 000 satunnaistettua operaatiota rinnan.	3
4	Testissä ajettiin yhteen solmuun 10 000 satunnaistettua operaatiota ja samalla toiseen solmuun ajettiin muita satunnaisia operaatioita. Näistä muista operaatioista osa käsitteli samaa dataa kuin ajetut 10 000 operaatiota.	2
5	Testissä ajettiin kaikkiin solmuihin satunnaistettuja operaatioita 5-60 sekunnin välein vuorokauden ajan. Seuraavan operaation ajankohta arvottiin edellistä suorittaessa edellä mainitulta aikaväliltä.	3
Testit ilman tapahtumaperustaista replikaatiota vakailla yhteyksillä		
Testi No.	**Testin kuvaus**	Solmujen lkm.
6	Testissä toiseen solmuun ajettiin 1000 satunnaista operaatiota ja odotettiin taustareplikaation synkronointia.	2
7	Testissä molempiin solmuihin ajettiin kumpaankin 1000 satunnaista operaatiota ja odotettiin taustareplikaation synkronointia.	2

8	Testissä lähtötilanteessa kaikki tietokannat olivat toisistaan eroavia. Yhteen solmuun ajettiin heti käynnistyessä 1000 operaatiota ja odotettiin taustareplikaation synkronointia.	3
9	Testissä lähtötilanteessa kaikki tietokannat olivat toisistaan eroavia. Kahteen solmuun ajettiin heti käynnistyessä 1000 operaatiota kumpaankin ja odotettiin taustareplikaation synkronointia.	3
10	Tässä testissä ajettiin kaikkiin solmuihin jokaiseen 1000 satunnaista operaatiota rinnakkain. Tällä testattiin taustareplikaation suorituskykyä, sillä kaikki solmut aloittivat lähes samaan aikaan synkronoinnin erilaisilla datoilla.	3
11	Tässä testissä ajettiin molempiin solmuihin 1000 satunnaista operaatiota rinnakkain. Molemmilla solmuilla oli tietokannoissaan jo valmiiksi useampi tuhat riviä, joista osa ei ollut synkronoitu solmujen välillä.	2
12	Testissä ajettiin kaikkiin solmuihin satunnaistettuja operaatioita 5-60 sekunnin välein vuorokauden ajan. Seuraavan operaation ajankohta arvottiin edellistä suorittaessa edellä mainitulta aikaväliltä.	3
13	Tässä testissä laitettiin yhteen solmuun ajoon satunnaisia operaatioita niin nopeasti kuin Python client niitä pystyi rajapinnan läpi lähettämään. Hetken päästä alettiin ajamaan toiseen solmuun operaatioita, jolloin taustareplikaatio ja asiakasoperaatiot kilpailivat keskenään.	3
14	Testissä yhdellä solmulla tietokantaan ei ollut yhteyttä. Muihin solmuihin tehtiin satunnaisia operaatioita 10 minuutin ajan. Testissä todennettiin, että vaikka yksi solmu ei toimi oikein, pystyvät muut päivittämään ja lukemaan dataa rinnakkain.	3
15	Testissä yhdellä solmulla tietokantaan ei ollut yhteyttä. Muihin solmuihin tehtiin satunnaisia operaatioita 10 minuutin ajan. Tietokantayhteys palautettiin kesken ajon solmulle, jolloin varmistettiin, että synkronointi jatkuu solmujen välillä virhetilanteen jälkeen.	3
Python testikirjastolla tehdyt testit verkkohäirinnän kanssa		
Testi No.	**Testin kuvaus**	Solmujen lkm.

16	Tässä pitkäkestoisessa testissä kumpaankin solmuun ajettiin satunnaisia operaatioita satunnaistetulla aikavälillä, joka vaihteli 5-60 sekunnin välillä. Testin ajo kesti kaksi vuorokautta. Koko testin ajan solmujen liikenne ajettiin WANem -verkkosimulaattorin läpi, joka oli konfiguroitu seuraavalla tavalla: -**Random connection disconnect**: Type: TCP-restart, MTTF Low : 1, MTTF High :1000, MTTR Low : 1, MTTR High : 1000 -**Delay**: 100ms, Jitter :20ms -**Loss**: 2%, Correlation: 25% -**Corruption**: 1%	2
17	Tässä pitkäkestoisessa testissä kumpaankin solmuun ajettiin satunnaisia operaatioita satunnaistetulla aikavälillä, joka vaihteli 5-60 sekunnin välillä. Testin ajo kesti kaksi vuorokautta. Koko testin ajan solmujen liikenne ajettiin WANem:verkkosimulaattorin läpi, joka oli konfiguroitu seuraavalla tavalla: -**Random connection disconnect**: Type: TCP-restart, MTTF Low : 20, MTTF High :5000, MTTR Low : 1, MTTR High : 3000 -**Delay**: 120ms, Jitter :30ms -**Loss**: 1%, Correlation: 25% -**Corruption**: 2%	3
18	Tässä testissä kumpaankin solmuun ajettiin satunnaisia operaatioita, sekä samanaikaisesti muita viestejä rajapinnan läpi. Koko testin ajan solmujen liikenne ajettiin WANem:verkkosimulaattorin läpi, joka oli konfiguroitu seuraavalla tavalla: -**Random connection disconnect**: Type: TCP-restart, MTTF Low : 1, MTTF High :10, MTTR Low : 1, MTTR High : 100 -**Delay**: 120ms, Jitter :30ms -**Loss**: 1%, Correlation: 25% -**Corruption**: 1%	2

19	Tässä testissä kumpaankin solmuun ajettiin satunnaisia operaatioita, sekä samanaikaisesti muita viestejä rajapinnan läpi. Testissä myös välillä käynnistettiin uudelleen toisen solmun prosessi. Koko testin ajan solmujen liikenne ajettiin WANem:verkkosimulaattorin läpi, joka oli konfiguroitu seuraavalla tavalla: -**Random connection disconnect**: Type: TCP-restart, MTTF Low : 1, MTTF High :10, MTTR Low : 1, MTTR High : 100 -**Delay**: 120ms, Jitter :30ms -**Loss**: 1%, Correlation: 25% -**Corruption**: 1%	2
20	Tässä testissä kumpaankin solmuun ajettiin satunnaisia operaatioita, sekä samanaikaisesti muita viestejä rajapinnan läpi. Kesken testiajon järjestelmään liittyi uusi solmu. Koko testin ajan solmujen liikenne ajettiin WANem:verkkosimulaattorin läpi, joka oli konfiguroitu seuraavalla tavalla: -**Random connection disconnect**: Type: TCP-restart, MTTF Low : 1, MTTF High :10, MTTR Low : 1, MTTR High : 100 -**Delay**: 120ms, Jitter :30ms -**Loss**: 1%, Correlation: 25% -**Corruption**: 1%	3
21	Tässä testissä kaikkiin solmuihin ajettiin satunnaisia operaatioita kolmen vuorokauden ajan. Verkko solmujen välillä oli konfiguroitu siten, että solmujen A ja B välillä oli häiritsemätön yhteys molempiin suuntiin koko ajan. Yhteyksiä C -> A, C -> B, A -> C ja B -> C häirittiin WANem:n avulla. Tämän lisäksi solmun A tietokanta sijaitsi toisella palvelimella ja yhteyttä tähän palvelimeen häirittiin myös WANem:n avulla. Testien aikana välillä solmuja myös sammutettiin ja käynnistettiin uudelleen. Koko testin ajan solmujen liikenne ajettiin WANem:verkkosimulaattorin läpi, joka oli konfiguroitu seuraavalla tavalla: -**Random connection disconnect**: Type: TCP-restart, MTTF Low : 1, MTTF High :10, MTTR Low : 10, MTTR High : 1000 -**Delay**: 100ms, Jitter :30ms -**Loss**: 1%, Correlation: 25%	3

	-**Corruption**: 2%	
22	Tässä testissä kaikkiin solmuihin ajettiin satunnaisia operaatioita vuorokauden ajan. Testissä oli käytössä 7 solmua ja tapahtumaperustainen replikaatio oli kaikissa estettynä testin ajan. Koko testin ajan solmujen liikenne ajettiin WANem:verkkosimulaattorin läpi, joka oli konfiguroitu seuraavalla tavalla: -**Random connection disconnect**: Type: TCP-restart, MTTF Low : 1, MTTF High :1000, MTTR Low : 10, MTTR High : 1000 -**Delay**: 100ms, Jitter :20ms -**Loss**: 2%, Correlation: 25% -**Corruption**: 2%	
Manuaaliset testit		
Testi No.	**Testin kuvaus**	Solmujen lkm.
23	Tässä testissä tietokannat alustettiin siten, että toisella solmulla tietokanta oli käynnistyessä tyhjä, kun taas toisella oli tietokannassa tuhansia rivejä.	2
24	Tässä testissä tietokannat alustettiin siten, että toisen solmun tietokannasta poistettiin satunnaisesti rivejä, jonka jälkeen käynnistettiin solmut.	2
25	Tässä testissä luotiin solmujen välille tietueisiin päivityskonflikteja. Näissä konflikteissa tietueiden joistakin revisioista oli eri versiot solmuilla. Tämän jälkeen solmut käynnistettiin.	2
26	Tässä testissä luotiin solmujen välille tietueisiin poistokonflikteja. Näissä konflikteissa tietueiden joistakin revisioista toinen solmu oli tehnyt poisto-operaation kyseisellä revisionumerolla ja toinen päivityksen. Osassa revisioista poistajan prioriteetti oli suurempi ja osassa päivittäjän. Tämän jälkeen solmut käynnistettiin.	2
27	Tässä testissä luotiin solmujen välille tietueisiin päivitys- ja poistokonflikteja, sekä poistettiin satunnaisesti revisioita tietueiden historiasta. Tämän jälkeen solmut käynnistettiin.	2

28	Tässä testissä tietokannat alustettiin siten, että yhdellä solmulla tietokanta oli käynnistyessä tyhjä, kun taas muilla oli tietokannassa tuhansia rivejä.	3
29	Tässä testissä tietokannat alustettiin siten, että kahdella solmulla tietokanta oli käynnistyessä tyhjä, kun taas yhdellä oli tietokannassa tuhansia rivejä.	3
30	Tässä testissä tietokannat alustettiin siten, että yhden solmun tietokannasta poistettiin satunnaisesti rivejä, jonka jälkeen käynnistettiin solmut.	3
31	Tässä testissä tietokannat alustettiin siten, että kahden solmun tietokannoista poistettiin satunnaisesti rivejä, jonka jälkeen käynnistettiin solmut.	3
32	Tässä testissä luotiin solmujen välille tietueisiin päivityskonflikteja kaikkien solmujen tietokantoihin. Näissä konflikteissa tietueiden joistakin revisioista oli eri versiot solmuilla. Tämän jälkeen solmut käynnistettiin.	3
33	Tässä testissä luotiin solmujen välille tietueisiin poistokonflikteja. Näissä konflikteissa tietueiden joistakin revisioista yksi solmu oli tehnyt poisto-operaation kyseisellä revisionumerolla ja kaksi muuta päivityksen. Osa konflikteista oli myös tehty siten, että ne olivat kahden solmun välillä ja kolmannella ei ollut tietoa kummastakaan versiosta käynnistyessä. Osassa revisioista poistajan prioriteetti oli suurempi ja osassa päivittäjän. Tämän jälkeen solmut käynnistettiin.	3
34	Tässä testissä luotiin solmujen välille tietueisiin päivitys- ja poistokonflikteja, sekä poistettiin satunnaisesti revisioita tietueiden historiasta. Tämän jälkeen solmut käynnistettiin.	3