TAMPEREEN TEKNILLINEN YLIOPISTO

JAAKKO TEUHO

USEAMMALLE PALVELIMELLE HAJAUTETUN PALVELUN SYNKRONOINTI JA YLLÄPITO

Diplomityö

Tarkastaja: professori Hannu-Matti Järvinen
Tarkastaja ja aihe hyväksytty Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 3. kesäkuuta 2015

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
Jaakko Teuho: Useammalle palvelimelle hajautetun palvelun synkronointi ja ylläpito
Diplomityö, 47 sivua
Marraskuu 2015
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Hannu-Matti Järvinen
Avainsanat: Tietokannat, hajautetut tietokannat, hajautetut järjestelmät, pilvipalvelut, palvelimet

Useammalle palvelimelle hajautetun palvelun synkronointi ja ylläpito on hankala ongelma, jonka ratkaisemisessa täytyy huomioida useita asioita. Synkronointi on mahdollista keskitetyillä tietokannoilla, mutta vain, jos palvelinten tietokantojen datan ei tarvitse jatkuvasti olla synkronoituna. Mikäli palvelun eri palvelinten tietokantojen synkronoinnin tulisi olla jatkuvaa, tarvitaan tehokkaampia ratkaisumenetelmiä.

Hajautetut tietokannat ovat yksi keino ongelman ratkaisemiseksi. Hajautetun tietokannan toiminnallinen periaate on samanlainen kuin keskitetyillä tietokannoilla, johon on lisätty useamman palvelimen vaikuttaminen tietokannan toimintaan. Erilaiset hajautetut tietokannat käyttävät erilaisia menetelmiä ja kyselyitä tietokannan toteuttamiseen. Kuitenkin monien näistä toiminta perustuu tunnettujen keskitettyjen tietokantojen toimintaan, jolloin ne hyödyntävät samanlaisia menetelmiä kyselyiden suorittamiseen sekä datan varastoimiseen.

Vaikka uusi, juuri kehitetty järjestelmä vaikuttaisi ratkaisevan kohdatut ongelmat yksinkertaisella tavalla, ei siitä ole hyötyä ennen kuin se on otettu käyttöön. Käyttöönotossa on tietysti myös omat huomionarvoiset piirteensä ja riskinsä. Uusi järjestelmä tulisi tutkia ja testata mahdollisimman hyvin ennen sen liittämistä olemassa olevaan palveluun. Myöhemmin, kun järjestelmän toiminta on varmistettu, voidaan järjestelmän toimintaa optimoida erilaisten menetelmien avulla. Mahdolliset uuden järjestelmän käyttöönotosta palvelulle aiheutuvat riskit olisi myös syytä tutkia kunnolla. Riskit voivat vaikuttaa palvelun toimintaan tai käyttäjien saamaan kokemukseen palvelun käytöstä.

Palvelun synkronoinnin ja ylläpidon tekeminen useamman palvelimen välillä tehokkaasti ei ole helppoa, muttei myöskään mahdotonta. Hajautettujen tietokantojen menetelmien opettelu on hyvä askel oikeaan suuntaan. Tärkeintä on kuitenkin edetä tarpeeksi pienin, mutta määrätietoisin askelin kohti päämäärää.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
Teuho, Jaakko: Updating and Synchronizing a service distributed over multiple servers
Master of Science Thesis, 47 pages
November 2015
Major: Software engineering
Examiner: Professor Hannu-Matti Järvinen
Keywords: Database, distributed database, distributed systems, cloud service, servers

Updating and synchronizing a service distributed over multiple servers is a difficult problem which contains numerous points of interest. Keeping servers synchronized is possible with centralized databases but only if the data doesn't have to be continuosly in sync. If continuous synchronization is required one needs to apply a more efficient solutions than centralized databases.

One way of solving this problem is using distributed databases. Easing their use is the fact that their basic functionalities are similar to centralized databases with the addition of operating over multiple servers. Different distributed databases use different methods and queries to create the database. However, being similar to centralized databases allows users to use same methods for building queries to process and store data.

Although the new, recently developed system seems to solve the occurring problems in a simple way it is of no use until it has been integrated into the service. Integration-phase has its own notable points of interest and risks. New system should be thoroughly inspected and tested before integrating it into an existing service. When the integration-phase is over and system testing is over can system optimization begin. All the risks relating to the new system should also be inspected and corrected, since realization of risks can affect the service or user experience received from using the service.

It is not easy to synchronize and update a service distributed over multiple servers, but it isn't impossible either. Learning to use distributed databases in this context is one step into the right direction. However, the most important thing to remember when changing something integral in an existing system is to advance with small but determined steps and always test everything before integration.

ALKUSANAT

Aloittaessani tämän työn tekemisen minulle sanottiin aiheen olevan hyvin hankala. Ja niin se olikin, sillä uudempia lähteitä oli aluksi hyvin vaikea löytää. Mutta useammalle palvelimelle hajautetun palvelun synkronointi ja ylläpito on kuitenkin hyvin ajankohtainen aihe, sillä pilvipalvelut kasvattavat jatkuvasti suosiotaan ja synkronointimenetelmiä tullaan tarvitsemaan kun palvelut laajentuvat useammalle palvelimelle. On mielenkiintoista selvittää, miten tämä synkronointi toteutetaan. Ja alkoihan niitä uudempiakin lähteitä löytyä alkukankeuden jälkeen.

Haluaisin kiittää professori Hannu-Matti Järvistä kommenteista ja neuvoista työn toteutuksen suhteen, Juha Rintalaa kommenteista sekä Timo-Pekka Leinosta työn aiheen esittämisestä. Kaikkein eniten kiitän vanhempiani ja sisaruksiani heidän antamastaan tuesta kirjoitustyön toteutuksessa.

Tampereella 23.11.2015.

Jaakko Teuho

SISÄLLYS

TERMIT JA NIIDEN MÄÄRITELMÄT

Amazonin relaatiotietokantapalvelu (relational database service)	Amazonin tarjoama palvelu palvelimilla sijaitsevien relaatiotietokantojen ylläpitämiseen
Globaalit relaatiot (global relations)	Hajautetun relaatiotietokannan relaatiot, jotka on hajautettu useamman palvelimen välillä.
Hajautettu relaatiotietokanta (distributed relational database)	Relaatiotietokanta, joka toimii hajautettuna useamman solmun sisältämässä järjestelmässä.
Isäntä (Master)	Palvelin, jonka sisältöä voidaan synkronoida eteenpäin.
LAMP	Linux, Apache, MySQL ja PHP, lyhenne palvelimen toiminnan osista.
LINQ (Language-Integrated Query)	Ohjelmointikieleen liittyvä kysely, monadien muoto, jolla selvennetään objektien ja datan väliä.
Lukkiutuminen (Deadlock)	Rinnakkaisesti ajetun ohjelman pysyvä tila, jossa kaikki toiminnot odottavat etenemiseen tarvittavan lukituksen aukeamista, mitä ei tapahdu.
Paikalliset relaatiot (local relations)	Hajautetun relaatiotietokannan relaatiot, jotka on tallennettu kokonaisuudessaan vain yhdelle (paikalliselle) palvelimelle.
Perus- ja Viite-avaimet (Primary and Foreign keys)	Relaatiotietokantatermejä, joilla kerrotaan kannan taulujen välisistä yhteyksistä.
Pilvipalvelu	Palvelu, joka toimii verkossa ilman erillistä asennettavaa sovellusta.
Relaatiokone	Polyphemessä käytetty abstrakti verkon komponentti, joka koostuu kolmesta osasta: nimiavaruus, objektiavaruus sekä kartoitusmekanismi.
Renki (Slave)	Palvelin, joka vastaanottaa sisältöä muilta, ei tuota sitä itse.
Sarakeperhe (column family)	Bigtablen kulunvalvonnan perusosa, koostuu ryhmitellyistä sarakeavaimista.
shared nothing	Hajautetun järjestelmän arkkitehtuurityyli, jossa järjestelmän solmut eivät jaa resursseja toistensa kanssa.
Solmu (Node)	Laite tai komponentti, joka toimii osana hajautettua järjestelmää.
Sovelluksen ohjelmointirajapinta, API	Ohjelmointirajapinta, jonka avulla kehittäjän on mahdollista käyttää sovelluksen sisäisiä toimintoja.

(application programming interface)	
Tietokanta	Järjestelmä, joka mahdollistaa sovelluksen käyttämän datan ja tiedon talletuksen, muokkauksen sekä poistamisen.
Tilakone	Järjestelmä, jonka toiminta perustuu sillä hetkellä olemassa olevaan tilanteeseen. Tiloja on useita, joiden välillä kone vaihtelee tilanteen mukaisesti.
Unioni	Looginen operaatio, joka yhdistää kaksi kokonaisuutta yhdeksi.

1 JOHDANTO

Nykyään palvelut, ohjelmistot ja järjestelmät ovat siirtymässä yhä enemmissä määrin verkossa toimiviksi järjestelmiksi. Tällainen järjestely toimii hyvin mutta ei ongelmitta. Näitä alkaa syntyä kun käyttäjiä on eri puolilla maailmaa eikä vain yhdellä alueella. Mikäli käyttäjä yrittää käyttää palvelua, joka sijaitsee maailman toisella puolella, syntyy palvelun toiminnassa viiveitä. Nämä viiveet voivat pahimmassa tapauksessa estää järjestelmän toiminnan kokonaisuudessaan tai hidastaa toimintaa valtavasti, tehden käytöstä epämukavaa. Tässä työssä tutkitaan menetelmiä usealla palvelimella samanaikaisesti toimivan palvelun tehokkaaseen toimintaan, synkronointiin sekä ylläpitoon.

Työ tehtiin WordDive -nimiselle yritykselle. WordDive on vuonna 2009 perustettu tamperelainen yritys, joka tarjoaa kielten oppimista edistävää palvelua. Tällä hetkellä palvelun käyttökielinä toimivat suomi, englanti, saksa, japani, venäjä, ruotsi, kiina sekä georgia. Opittavia kieliä palvelussa on tarjolla yli 12, mukaan lukien suomi, englanti ja kiina. Opittavat asiat on koottu yhteen kursseiksi ja erilaisia sanastoa ja kielioppia kehittäviä kursseja on kirjoitushetkellä yhteensä yli 200. Erityishuomiona mainittakoon Suomen, Saksan, Ruotsin sekä Venäjän lukioiden abiturienteille suunnatut englannin ja ruotsin kielikurssit, joista WordDive antaa rahat takaisin -takuun, mikäli oppilas ei kurssit suoritettuaan ansaitse vähintään tiettyä arvosanaa. Minimiarvosana riippuu kurssin kohdekielestä, joka suomalaisilla lukioilla on englannissa E ja ruotsissa M. Kirjoitushetkellä palvelu toimii verkon kautta kaikilla nykyaikaisilla selaimilla sekä iPhonen omalla sovelluksella. Vuonna 2014 yrityksen samanniminen sovellus valittiin Suomen parhaaksi sähköisen oppimisen mobiilisovelluksesi. Vuonna 2015 palvelu valittiin Red Herringin järjestämän tilaisuuden Euroopan 100:n parhaimman yrityksen joukkoon. Diplomityön aihe tuli yrityksen tavoitteesta laajentua Euroopan ulkopuolisille markkinoille, mihin tarvitaan uusia palvelimia palvelun käyttöön liittyvän viiveen vähentämiseksi.

Työn tavoitteena on löytää menetelmiä tehokkaaseen ja mieluiten helppokäyttöiseen menetelmään palvelinten välisen datan synkronoimiseen. Kuten luvussa 3 mainitaan, tällä hetkellä käytössä oleva menetelmä on aikaa vievä ja sisältää useita mahdollisuuksia inhimillisiin virheisiin. Nämä mahdollisuudet olisi hyvä saada minimoitua ja prosessi pitäisi saada automatisoida yhtenäiseksi kokonaisuudeksi, jolloin palvelun pitäminen synkronoituna palvelinten välillä helpottuisi huomattavasti. Ongelman ratkaisuvaihtoehtoja pohditaan luvuissa 4 ja 5.

Työ on jaettu lukuihin jotka pitävät sisällään näitä vastaavia kokonaisuuksia. Ensimmäisessä luvussa selostetaan työn tavoitteita sekä kerrotaan yrityksestä, jolle työ tehtiin. Lisäksi kuvaillaan työn muiden lukujen sisältöä lyhyesti. Luvussa 2 annetaan taustatie-

toa keskitetyistä tietokannoista sekä näiden käytöstä nykyaikaisissa tilanteissa. Luku esittelee myös pikaisesti yleisimpiä käytössä olevia tietokantoja. Luvussa 3 esitellään WordDive -palvelun tällä hetkellä toiminnassa olevaa järjestelmän pääpalvelimen ja uuden, Aasian markkinoille suunnatun palvelimen välistä yhteistyötä ja pohditaan näiden palvelinten väliseen toimintaan liittyviä ongelmia. Luvussa 4 esitellään ratkaisumenetelmiä palvelinten välisen synkronoinnin aiheuttamiin ongelmiin. Luvussa 5 pohditaan näiden uusien menetelmien käyttöönottoon liittyviä kohtia sekä pohditaan mahdollisia riskejä ja seuraamuksia, joita uuden järjestelmän käyttöönotosta mahdollisesti seuraa. Luvussa 6 annetaan yhteenveto työstä ja kirjoittajan omia mielipiteitä.

2 TIETOKANNOISTA

Tietokanta on datan hallintaan ja varastointiin tarkoitettu järjestelmä, joista useimmat koostuvat seuraavista komponenteista:

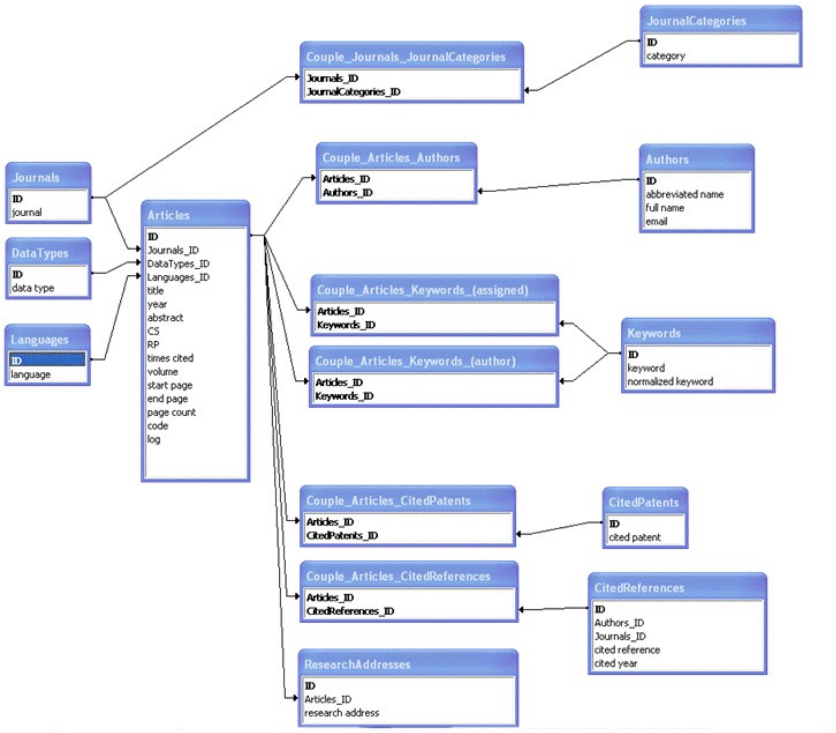
- kaavio, joka kuvaa tallennettavan datan rakennetta
- tietokantaan talletettu data
- käyttösovellukset
- datan paikallistamiseen tarkoitetut hakemistot.

Nämä komponentit muodostavat yhdessä toimivan tietokannan hallintajärjestelmän. [1, s. 45]

2.1 Yleisiä käytössä olevia tietokantoja

Useimmat verkkopohjaiset palvelut ja järjestelmät käyttävät datan hallintaan jonkinlaista tietokantaa. Tietokannan tyyppistä riippumatta sen ensisijainen tehtävä on mahdollistaa datan säilytyksen sekä sen lukemisen, muokkaamisen ja poistamisen. Viisi suosituinta tietokantajärjestelmää kirjoitushetkellä (syksy 2015) ovat Oracle, MySQL, Microsoft SQL SERVER, MongoDB sekä PostgreSQL. Näistä tietokannoista Oracle ja Microsoft SQL SERVER ovat saatavilla maksullisella lisenssillä, kun taas MySQL, MongoDB sekä PostgreSQL ovat vapaan lähdekoodin tarjoamia sovelluksia. [2]

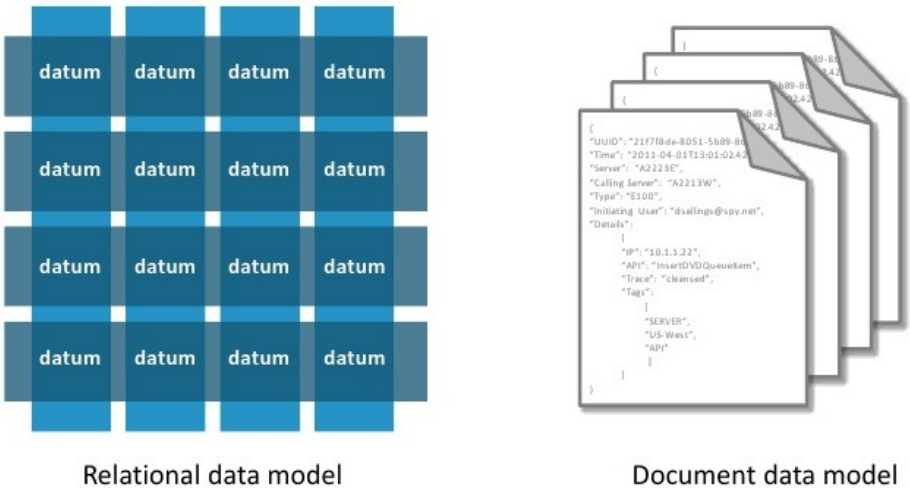
Neljä näistä viidestä tietokantajärjestelmästä kuuluvat relaatiotietokantoihin. Relaatiotietokantojen toiminta perustuu taulurakenteisiin ja näiden tärkeimpiin piirteisiin kuuluvat ennalta määritellyt taulut joihin data varastoidaan. Tauluja on mahdollista ylläpitää useita samanaikaisesti ja niiden välille voidaan luoda erilaisia yhteyksiä. Tästä seuraa näiden tietokantojen yhteinen nimitys, relaatiokanta, eli kanta joka sisältää yhteyksiä. [2] Kuvassa 1 on esitetty esimerkki relaatiokannan tauluista ja taulujen sisältämistä tiedoista sekä näiden välisistä yhteyksistä. Kuten kuvasta näkyy, yksi taululla voi olla yhteydessä useampaan muuhun tauluun samanaikaisesti.



Kuva 1: Esimerkki relaatiokannan rakenteesta [3]

Näistä viidestä kirjoitushetkellä suosituimmasta tietokantajärjestelmästä MongoDB on ainoa dokumenttipohjainen tietokanta. Dokumenttipohjaisten tietokantojen suurimpia eroja relaatiokantoihin verrattuna on datan järjestäminen kaaviottomasti ilman relaatiokantojen kiinteitä tauluja. Datan kaaviottoman järjestämisen myötä se voidaan tallentaa ilman yksiselitteistä rakennetta, eli jokaisella tallenteella voi olla omat tietonsa. Lisäksi jokaisen tallenteen sarakkeet voivat sisältää toisistaan poikkeavantyyppistä dataa ja useamman kuin yhden arvon. Kaikki tämä yhdessä mahdollistaa erittäin muokattavan datan tallennusrakenteen. [2] Kuvassa 2 esitetään relaatiokannan ja dokumenttipohjaisen tietokannan datan tallennukseen liittyviä eroja. Relaatiokannassa tietyn tyyppisen datan paikka on tiukasti rajattu tiettyyn tauluun eikä sitä voi tallettaa muualle. Dokumenttipohjaisessa tietokannassa taas datan tyypillä tai muodolla ei ole väliä vaan ne voidaan kaikki tallentaa vaikka samaan sijaintiin.

Vaikka suurin osa tällä hetkellä suosituimmista tietokantajärjestelmistä onkin relaatiotietokantoja, kasvattavat muunlaiset kantajärjestelmät jatkuvasti suosiotaan. Näistä etenkin MongoDB on jatkuvassa nousussa käyttäjien keskuudessa ja se voi hyvinkin nousta vielä kolmen käytetyimmän järjestelmän joukkoon.



Kuva 2: Relaatiokannan ja dokumenttipohjaisten tietokantojen rakenteet [4]

Michael Stonebraker Massachusetts Institute of Technologysta on yksi nykyaikaisten tietokantajärjestelmien tärkeimmistä kehittäjistä. Toteuttamastaan kehityksestä modernien tietokantajärjestelmien parissa hänelle myönnettiin arvostettu Turing Award -palkinto. Stonebraker kehitti useita nykyajan tietokantojen käyttämistä menetelmistä sekä esitti kuinka tietokantoja voidaan rakentaa näiden periaatteiden avulla sekä kuinka data säilötään näihin käytännöllisesti. Kaikki hänen kehittämänsä ratkaisunsa julkaistiin avoimen lähdekoodin lisenssillä. Tämä vapaasti saatavaksi asettaminen ja menetelmien tehokkuus datan hallinnan kanssa takasi menetelmille laaja-alaisen käytön ja kehittäjien hyväksynnän. [5, s. 1]

Stonebraker kehitti muun muassa Ingres-nimisen tietokannan, jolla hän pystyi osoittamaan relaatiotietokantojen olevan toteuttamiskelpoisia työkaluja, eikä pelkkiä teorioita. Ingres oli toinen kahdesta ensimmäisenä kehitetystä relaatiotietokannasta ja sen avulla Stonebraker edisti kyselykielten suunnittelua, kyselyjen prosessointitekniikoita, sekä datan ja rinnakkaisen suorituksen hallintaa. Lisäksi hän osoitti, että kyselyiden uudelleenkirjoitus- ja muokkaustekniikoita pystytään käyttämään relaationäkymien ja pääsyn hallintaa. [5, s. 1]

Toisen, Postgres-nimisen tietokantajärjestelmän julkaisullaan Stonebraker esitteli tietokantojen olio-relaatiomallin. Tässä mallissa yhdistettiin olio-ohjelmoinnin tärkeimpiä ideoita relaatiotietokantojen kontekstiin. Postgres laajensi relaatiotietokantamallia, tarjoten käyttäjille mahdollisuuden määrittää, varastoida sekä hallita rikkaita objekteja mutkikkailla tiloilla ja käyttäytymisillä. Postgres ja Ingres järjestelmien esittämiä konsepteja ja menetelmiä käytetään vieläkin lähes kaikissa nykyisissä tietokantajärjestelmissä. [5, s. 1-2]

Stonebraker kehitti myös hajautettujen kyselyiden suoritusmenetelmiä sekä tapahtumien koordinointiprotokollia Distributed Ingres -järjestelmällään, joka oli ensimmäisiä hajautettuja tietokantajärjestelmiä. Stonebraker tutki hajautettuja tietokantoja myös muillakin järjestelmillä, kuten Mariposalla. Se on massiivinen hajautettu tietokantajär-

jestelmä, jolla tutkittiin muun muassa opportunistista datan replikointia sekä hajautettujen kyselyiden hallintaa. Stonebraker asetti suunnan myös skaalautuvien datajärjestelmien suunnittelulle ja oli ensimmäisiä, jotka omaksuivat ”shared nothing” -tyylisen hajautetun tietokanta-arkkitehtuurin. Tässä arkkitehtuurissa jokainen järjestelmän solmu toimii itsenäisesti ja omavaraisesti, eikä yksikään näistä solmuista jaa toimintaresurssejaan muiden kanssa. Tätä lähestymistapaa pidetään yleisesti ainoana tapana ylläpitää järjestelmän skaalautuvuutta ja sitä käytetään lähes kaikissa suurimmissa tietokantajärjestelmissä ja ”big data” -tyylisissä ratkaisuissa. [5, s.2]

Tietokantojen kehityksellä on ollut valtavan suuri vaikutus ohjelmointiin, varsinkin relaatiokannoilla. Ohjelmoijat ovat innokkaasti käyttäneet näitä pitkään kehitettyjä menetelmiä datan hallintaan ja varastointiin. Erilaiset datan välillä olevat yhteydet ja sen organisointiin käytetyt taulut muodostavat tietynlaisen käyttöliittymän datan hallintaan useimmissa tietokannoissa. Nämä ovat oikeastaan matemaattisten konseptien esiintymiä, monadeja, ja tietokantakyselyjä voidaan kääntää monadisiksi operaatioiksi. Ohjelmoijat kutsuvat tätä LINQ:si (Language-Integrated Query, ohjelmointikieleen liittyvä kysely) monadien sijaan ja on olemassa LINQ-sidonnaisia usealle tietokantajärjestelmälle, kuten SQL, Azure Tables, MongoDB ja Hadoop, altistaen mallit kehittäjille upottamalla ne push- ja pull-komennoiksi käyttäen yhteistä rajapintaa. [6, s.8-9]

Relaatiotietokannat ovat hyvin perusteellisia ja mutkikkaita järjestelmiä ja insinööritaidon hedelmiä. Ne toimivat erittäin hyvin niiden suunnittelua vastaavissa tilanteissa, mutta muualla niistä ei ole paljoakaan hyötyä. Objektit puolestaan ovat halpoja, yksinkertaisia sekä monikäyttöisiä ja ne selviävät pilveen siirtymisestä. Pilvipalveluiden aikakausi on monadisen ja comonadisen laskennan ja verbien aikaa, ei datan tai substantiivien. [6, s.9]

Vanhat, 1980-luvun relaatiokannat yrittävät olla liikaa tietokantoja ja liian vähän objekteja eivätkä ne onnistuneet käyttämään vahvuuksiaan edukseen. Pienessä taulujen, deklaratiivisten kyselyiden ja hienostuneiden optimointien maailmassa on vaikea voittaa relaatiokantojen hallintajärjestelmiä näiden omassa lajissaan. Toisaalta, isossa ja avoimessa pilvimaailmassa laajasti saatavilla olevat objektit ja toimijat hallitsevat ja toimivat tehokkaammin kuin perinteiset menetelmät. Helpottaakseen palveluiden siirtymistä pilveen Meijer on koonnut muutamia vaatimuksia tietokannoille:

- Paljasta kaikki tietokantojen mallintajien datan lähteet ohjelmoijille käyttäen monadeja tai LINQejä.
- Luo luokkakirjasto hajautetuista kokoelmista toteutettuna saatavilla olevista ja skaalautuvista palveluista.
- Anna ohjelmoijille pääsy rinnakkaisuuden toteutukseen.
- Käytä jäljitystä, tarkkailua, ongelmienratkontaa sekä diagnosointia osana pilveä.

Monet näistä menetelmistä ovat tällä hetkellä kehittäjien saatavilla. Ainoa puuttuva tekijä on näiden osien liittäminen toisiinsa mahdollisimman toimivassa muodossa. [6, s.9]

2.2 Tietokantojen kehittäjät ja sovellusten ohjelmoijat pilvipalveluissa

Tänä päivänä pilvipohjaiset verkkopalvelut ovat kasvattaneet suosiotaan niin palvelun loppukäyttäjien kuin sen kehittäjienkin keskuudessa. Kun palvelu on verkossa, käyttäjän ei tarvitse asentaa erillisiä ohjelmia tai sovelluksia omalle laitteelleen. Palvelun siirtämisessä pilveen on kuitenkin otettava huomioon useita seikkoja, etenkin tietoturvallisuuden kohdalla.

Datan julkinen esittäminen sekä deklaratiivisten menetelmien käyttö voivat osoittautua pilveen siirryttäessä enemmän haitallisiksi kuin hyödyllisiksi. Objektien tulisi piilottaa datan yksityiset esityksensä, paljastaen ne vain ennalta määritettyjen käyttöliittymien kautta ja vain dataa pyydettäessä. Ohjelmointikielten maailma on aina tukenut avoimen maailman oletuksia asettamalla järjestelmän käyttäytymisen tärkeimmäksi. Helpottaakseen pilvipohjaisiin palveluihin siirtymistä olemassa olevat kirjastot ja työkalut tulisi kehittää toimimaan saatavilla olevina palveluina. [6, s.1]

Tietokantojen kehittäjät (tästä lähtien *kehittäjät*) ja sovellusten ohjelmoijat (tästä lähtien *ohjelmoijat*) näkevät ohjelmoitavan maailman omilla tavoillaan. *Kehittäjille* kaikki toiminta sijoittuu datan ympärille ja koostuu lähinnä substantiiveista, kuten 'Asiakas', 'Tilaus', 'Tunniste', jne. Perus- ja Viite-avaimilla määritellään datan suhdetta toisiinsa tietokannan tauluissa. Perus- ja Viite- avainten muodostamat suhteet vahvistavat tarvetta paljastaa suuri osa alla toimivasta taulurakenteesta, mikä haittaa yksityisen datan piilossa pitämisessä. *Kehittäjät* katsovat tämän piilottamisen puutteen toimivan heidän edukseen, sillä se mahdollistaa improvisoidut tai ennalta suunnittelemattomat tietokantakyselyt, mikä lisää kannan käytön joustavuutta. [6, s.1-2]

Tämä *kehittäjien* substantiivipohjainen sovellusten kehityksen maailmankatsomus näkyy myös heidän suhtautumisessaan tietokonelaskentaan. Heille se on vain yksi uusi malli analysoitavaksi, muutettavaksi, optimoitavaksi sekä tulkittavaksi kyselymoottoreille. Tietokantakyselyt ovat luonteeltaan deklaratiivisia, mutta *kehittäjät* luottavat kyselyjen optimoinnin löytävän tehokkaan suunnitelman suoritukseen. Tämä suunnitelma käyttää hyväkseen tietokannan indeksirakennetta sekä sen muita kehittyneitä menetelmiä nopeuttaakseen toimintaansa. Toteutuakseen tämä suunnitelma vaatisi ideaalisen, suljetun maailman, jossa optimoija voisi pohtia kyselyä kaikkien taulujen avulla ilman viiveitä, poikkeuksia tai muita alhaisempien tasojen ongelmia. Nämä oletukset hajoavat hajautettujen tapahtumakyselyiden yhteydessä hitaalla ja toimimattomalla verkolla, tai kun tietokantataulut kasvavat niin isoiksi etteivät ne enää mahdu yhdelle laitteelle. [6, s.2-3]

Konseptit kuten ”käyttäytyminen”, ”sivuvaikutukset” ja ”korkeamman tason funktiot” ovat *ohjelmoijien* alaa. Heidän ajatusmaailmansa sijoittuu tietokonelaskennan ympärille datan sijaan. *Ohjelmoijille* tärkeitä tekijöitä ovat verbit, kuten 'poistaTiedostot', 'raahauksenJälkeen' ja niin edelleen, sekä vahvojen datan abstraktioiden valvominen.

Sovellusten varsinainen toteutus ja sen käyttämät resurssit ovat tarkasti yksityisiä ja niitä ei paljasteta ulkopuolisille. Tämä mahdollistaa saman syntaksin käytön useammalla eri objektilla, jotka seuraavat samanlaista kaavaa. [6, s.3-4]

Kehittäessään ohjelmia *ohjelmoijat* eivät turvaudu vain ohjelmointikieliin ja kirjastoihin. He käyttävät myös useita erilaisia työkaluja, kuten virheiden jäljittäjiä sekä ohjelmia, jotka mittaavat sovelluksen suoritusaikoja löytääkseen suorituksen hitaimpia kohtia. Puhuttaessa olemassa olevien palveluiden siirtämisestä pilveen jätämme usein huomiotta useita tarpeellisia toteutuksen yksityiskohtia, kuten havainnointi, jäljitys, hajautettu lukittuminen, virheiden hallinta, replikointi ja ongelmatilanteiden menetelmät. Näiden toteuttaminen ei ole helppoa, mutta ne ovat tarpeellisia jotta *ohjelmoijat* voivat menestyä hajautettujen sovellusten parissa. [6, s.8]

3 UUDEN PALVELIMEN LISÄÄMINEN PALVELUUN

Tämä luku käsittelee WordDiven tapausta laajentua Euroopan ulkopuolisille markkinoille. Huomio on kiinnitetty palvelintekniikkaan sekä pääpalvelimen ja uuden palvelimen väliseen yhteyteen sekä näiden väliseen datan synkronointiin ja ylläpitoon.

3.1 Palvelin palvelun sisällönluontia varten

Palvelun kiinankielinen sisältö tuotetaan pääsääntöisesti Kiinassa käyttäen pääpalvelinta. Tämä tuotanto ei kuitenkaan ole täysin vailla ongelmia, sillä pitkät viiveet kiinalaisten työntekijöiden ja Suomessa sijaitsevan palvelimen välillä tekevät työnteosta tuskallisen hidasta. Lisäksi Kiinan palomuuri estää tunnettujen palvelujen lisäosien, kuten JavaScript-kirjasto jQueryn tai Googlen kirjaintyylien latauksen. Tämän seurauksena käyttäjien kokema viive kasvaa ja sivusto ei välttämättä edes toimi halutulla tavalla.

Viiveeseen ja palomuurin aiheuttamaan ongelmaan on kuitenkin olemassa erilaisia ratkaisumahdollisuuksia. Yksi tällainen keino on uuden palvelimen pystyttäminen lähelle Kiinaa ja palvelun sisällönluontiosuuden kopioiminen sinne. Näin saadaan palomuurista ja palvelimen etäisyydestä johtuva viive laskettua mahdollisimman pieneksi.

3.1.1 Palvelimen pystytys

Jotta sisällön luonnissa esiintyvät viiveet saataisiin pienennettyä, hankittiin Kiinan manneralueelta uusi palvelin sisällöntekijöiden käyttöön. Pääpalvelimen käyttämät järjestelmät päätettiin asentaa myös uudelle palvelimelle, jotta palvelimen teknillinen puoli toimisi samalla tavalla. Näin ollen kumpikin palvelin käyttää toiminnassaan LAMP yhdistelmää, joka sisältää Linux käyttöjärjestelmän, Apachen palvelinohjelman, MySQL tietokantana sekä PHP järjestelmän ohjelmointikielenä. Nämä asennettiin uudelle palvelimelle käyttäen Amazonin LAMP opastusta. [7] Järjestelmän versiot pyritään pitämään mahdollisimman uusina ja samoina pääpalvelimen kanssa, jotta vältyttäisiin eri versioiden aiheuttamilta ongelmilta.

Amazonin ohjeita noudattamalla uuden palvelimen käynnistys sujui yksinkertaisesti ja melko nopeasti. Uuden palvelimen järjestelmien päivitys hoidetaan samaan aikaan kuin pääpalvelimenkin.

3.1.2 Tarpeellisen sisällön kopiointi uudelle palvelimelle

Koska tyhjästä palvelimesta ei ole kovinkaan paljoa hyötyä, kopioitiin uudelle palvelimelle sisällönluontiin tarvittavat työkalut pääpalvelimelta. Näin sisältöä pystytään tuottamaan täysin samanlaisin menetelmin ja palveluun ei synny yhteensopimatonta sisältöä.

Tässä tapauksessa kopiointia vaikeuttaa kuitenkin yksi tekijä: Kiinan palomuuri. Kyseinen palomuuri estää tai hidastaa niiden verkkosivujen toimintaa, jotka sisältävät viitteitä tai latauksia palomuurin estolistalla oleviin verkkosivuihin [8]. Listalta löytyvät muun muassa Google ja sen kaikki verkkosivujen lisäosat, esimerkiksi Googlen kehittämät kirjaintyylit, sekä useita JavaScript-kirjastoja, esimerkiksi jQuery. Palomuuri hidastaa palvelun toimintaa todella paljon, koska palvelun nykyinen versio lataa osan näistä kirjastoista ja lisäosista ulkoisina resursseina toimintansa tehostamiseen.

Tähän ongelmaan on onneksi melko yksinkertainen ratkaisu. Koska Kiinan palomuuri estää nimenomaan näiden resurssien ylläpitämien sivustojen toiminnan, ladataan tarvittavat kirjastot ja lisäosat palvelimelle ja muutetaan ulkoiset viitteet sisäisiksi. Näin ollen palvelun toiminnan ulkopuoliset lataukset saadaan minimoitua ja vältytään hidastumisilta.

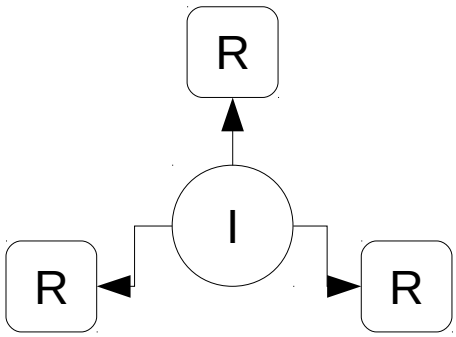
Ratkaisussa on kuitenkin muutamia ongelmia, joiden takia sitä ei voida soveltaa pääpalvelimella. Koska kaikki lataukset tehdään palvelimen sisäisesti, hidastaisi tämä huomattavasti Euroopan ulkopuolella sijaitsevien käyttäjien toimintaa. Tämän seurauksena pääpalvelimen resurssit ladataan pilvipalvelun kautta, jossa tarvittavat resurssit ladataan käyttäjää lähimpänä olevalta palvelimelta käyttäen Amazonin CloudFrontia. Tätä ei kuitenkaan voida soveltaa Kiinan palvelimelle, sillä CloudFront kuuluu Kiinan palomuurin estämien sivujen listaan. Toisena ongelmana on ladattujen lisäosien versioiden päivitys. Linkittämällä lisäosat suoraan valmistajien sivuilta tulisi versiopäivitykset tehtyä automaattisesti, mutta tässä tapauksessa uudet versiot täytyy ladata ja siirtää käsin.

3.1.3 Luodun sisällön synkronointi

Kun palvelin oli toimintakykyinen ja tarvittavat työkalut sisällönluontiin oli asennettu, voitiin uuden palvelimen pääasiallinen tehtävä aloittaa. Sisällönluonti sujui hyvin, ja sen seurauksena ryhdyttiin pohtimaan seuraavaa ongelmakohtaa, eli miten palvelinten välinen data tulisi synkronoida. Synkronointia hankaloittaa se, ettei palvelinten luomaa uutta dataa saisi ylikirjoittaa. Kahden toiminnassa olevan palvelimen synkronointi on hankala ongelma, johon liittyy useampia pohdittavia asioita. Näitä selostetaan enemmän luvuissa 4 ja 5. Tässä kappaleessa tutustutaan tämän uuden palvelimen yhteydessä käytettyyn ratkaisuvaihtoehtoon.

Synkronointi suunniteltiin toteutettavan mahdollisimman yksinkertaisella ja nopeasti tehtävällä tavalla, jotta kiinankielisen sisällön luonti saataisiin mahdollisimman nopeasti tuottamaan palveluun sisältöä ilman ongelmia. Tästä johtuen päädyttiin hieman muokat-

tuun isäntä-renki – suhteeseen pääpalvelimen ja uuden palvelimen välillä. Perinteisessä isäntä-renki –välisessä palvelinten suhteessa isäntä -palvelimella tehdään muutoksia palvelun dataan ja toimintaan, jotka sitten tarpeen tullen kopioidaan renki -palvelimelle. Tätä on kuvattu kuvassa 3, missä isäntä -palvelin on yhdistetty useampaan renki -palvelimeen.



Kuva 3: Perinteinen palvelinten välinen isäntä-renki – suhde

Perinteinen isäntä-renki –suhde ei riitä tämän palvelun vaatimassa tilanteessa. Vaatimuksena oli, että kumpikin palvelin pystyy tuottamaan sisältöä ongelmitta. Tästä johtuen perinteistä mallia päätettiin hieman muuntaa tähän tilanteeseen sopivaksi. Staattisen isäntä-renki –palvelinten välisen suhteen sijaan palvelinten tilaa vaihdetaan riippuen siitä, millä palvelimella uusi, muokattu sisältö kulloinkin sijaitsee. Näin ollen isännästä tulee tarvittaessa renki ja rengistä isäntä. Tämä järjestely ei toimisi, mikäli kummallakin palvelimella tehtäisiin samaa sisältöä samaan aikaan. Tällöin vaarana olisi tuotetun sisällön ylikirjoittaminen. Koska uudella palvelimella tuotetaan vain tiettyä osaa koko palvelun sisällöstä, voitiin tässä tapauksessa soveltaa tätä menetelmää.

Tässä tapauksessa ensisijaisen tärkeää oli saada uusi palvelin toimintakykyiseksi ja uuden sisällön tuotanto käynnistettyä mahdollisimman nopeasti. Sen seurauksena varsinainen sisällön synkronointi toteutettiin käsin käyttäen seuraavanlaista menetelmää. Aina kun isäntänä toimiva palvelin oli valmis synkronoimaan tietonsa eteenpäin, otettiin siellä tehdyistä datan muokkauksista kopiot. Kopiot tarkastettiin ylikirjoitusten varalta ja jotta varmistuttaisiin vain tarvittavien tietojen siirtämisestä. Varmistusten jälkeen nämä kopiot siirrettiin suojatun yhteyden välityksellä sillä hetkellä renkinä toimivalle palvelimelle. Onnistuneen tiedostojen siirron jälkeen tiedot syötettiin rengin tietokantaa. Synkronoinnin jälkeen renkinä toimivasta palvelimesta muodostui isäntä ja siellä voitiin aloittaa muutosten työstäminen. Isännästä puolestaan muodostui renki, jossa vastaavan sisällön tuottaminen lopetettiin kunnes koittaisi jälleen synkronoinnin aika.

3.1.4 Ongelmia ja parannusehdotuksia

Vaikka tämä edellä kuvattu menetelmä oli nopea toteuttaa ja sillä saatiin melko vaivattomasti uusi palvelin toimintakuntoon ja uuden sisällön luonti toimimaan, on se vielä hyvin kaukana ideaalista ratkaisusta. Suurimpana ongelmana synkronoinnin hoitamisessa edellä mainitulla tavalla on siihen kuluva aika. Kaiken tarvittavan toteuttaminen käsin vie todella paljon aikaa, ainakin 15 minuuttia jokaisella kerralla. Lisäksi voi tapahtua inhimillisiä virheitä tiedostoja siirrettäessä palvelinten välillä.

Näistä seikoista johtuen tulisi järjestelyyn lisätä jonkinlaista automatisointia. Varsinkin tietokannoista tehtävät kopiot ja tiedostojen siirto palvelimelta toiselle voitaisiin tehdä automaattiseksi. Vaihtoehtoisesti siirrot ja kopioinnit voitaisiin toteuttaa erillisellä skriptillä. Näin prosessista saataisiin virtaviivaisempi ja helpompi käyttää.

3.2 Uusi palvelin palvelun käyttöä varten

3.2.1 Testaus ja suositusten hankinta

Manner-Kiinan alueelle ei saa perustaa rahallisesti tuottavaa palvelua ilman tarvittavien lupien hankintaa, mihin voi kulua puolikin vuotta [9, s. 27]. Tästä johtuen uutta, manner-Kiinan sisällä sijaitsevaa sisällöntuotantopalvelinta ei voida heti käyttää kiinalaisten loppukäyttäjien palvelimena. Näinollen yrityksessä päätettiin perustaa uusi palvelin kiinalaisten asiakkaiden käyttöön mahdollisimman läheltä Kiinaa, sillä Kiinan palomuurista johtuen käyttäjät eivät voineet käyttää pääpalvelinta ongelmitta. Kiinan palomuurin aiheuttamat ongelmat on selostettu aliluvussa 3.1.

Uusi palvelin tulisi siis pääsääntöisesti palvelemaan kiinankielistä väestöä, mutta sitä voisi käyttää myös muutkin halukkaat. Kiinan palomuurista johtuen päätettiin siirtää sen estämät resurssit jälleen sisäisesti ladattaviksi. Estetyt resurssit on mainittu aliluvussa 3.1. Palveluun päätettiin myös lisätä enemmän vuorovaikutusta kiinalaisten oman sosiaalisen median kanssa. Tästä johtuen siihen lisättiin kiinalaisten sosiaalisen median sivustojen, Weibon, WeChatin sekä Youkun linkkejä.

Uuden palvelimen toimintakuntoon saattaminen sujui suurin piirtein samalla tavalla kuin manner-Kiinan alueella olevan palvelimen toteutus. Kun kaikki järjestelmät olivat toimintakunnossa, kopioitiin palvelun Kiina -versio uudelle palvelimelle. Kopiointi tehtiin samalla tavalla kuin sisällöntuotannolla. Uusi palvelin tuli kuitenkin asiakkaiden käyttöön, joten muutamia uusia seikkoja oli otettava huomioon dataa kopioitaessa. Yksi näistä oli jälleen tiedostojen synkronointi. Mikäli synkronointiin ei puututtaisi millään tavalla, ylikirjoittaisivat kaikki indekseillä varustetut ja käyttäjiin liittyvät tiedot toisensa tietokannassa.

Jotta palvelin pysyisi mahdollisimman yksinkertaisena, indeksointi päätettiin järjestää uudelleen uudella käyttöpalvelimella. Uusia käyttäjiä koskevien tietojen indeksointi aloitettiin reilusti suuremmasta luvusta kuin pääpalvelimella. Näin vältytään helposti

ylikirjoituksen vaaralta. Ongelmana tosin on se tosiasia että jossain vaiheessa pääpalvelin saa uuden palvelun indeksit kiinni. Tällöin ylikirjoituksen vaara on taas oleellinen seikka. Tästä seikasta johtuen indeksointi tulisi valita mahdollisimman korkeaksi. Toinen vaihtoehto olisi lisätä kaikkiin indeksointia käyttäviin ja käyttäjiin liittyviin tauluihin erillinen sarake käyttäjän alkuperäistä palvelinta tunnistamista varten. Tämän sarakkeen avulla voidaan helposti löytää käyttäjien alkuperäiset palvelimet, mitkä erottavat käyttäjien tiedot toisistaan, ja synkronointi helpottuisi huomattavasti.

Tehtyjen ratkaisujen vuoksi pääpalvelimen ja uuden palvelimen välinen suhde muodostui myös isäntä-renki –tyyppiseksi, vaikka uusi palvelin kerääkin uusien käyttäjien antamia tietoja. Tämä johtuu päätöksestä tehdä koko palvelua muokkaavia muutoksia vain pääpalvelimella, minkä lisäksi synkronointi tapahtuu vain pääpalvelimelta uudelle palvelimelle. Mikäli synkronoinnin suunta halutaan syystä tai toisesta vaihtaa, täytyisi sen hoitamiseen kehittää jokin tehokkaampi keino kuin pelkkä käsin tehtävä tiedostojen siirtäminen.

Vaikka järjestelmässä ja toimintatavoissa on vielä parantamisen varaa, käyttäjät olivat tyytyväisiä viiveiden pienentymiseen ja palvelun parantuneeseen toimivuuteen. Näin ollen kokeilua voidaan pitää onnistuneena prototyyppinä, jota kehitetään tulevaisuuden ja vastaavien tilanteiden varalle.

3.2.2 Uuden version siirtäminen palvelimelle ja datan synkronointi

Koska palvelu toimi hyvin käyttäjien mielestä, päätettiin kehittää palvelun kiinalainen palvelin vastaamaan palvelun uusimman version tuomaa uudistettua ulkoasua. Uusi versio palvelusta julkaistiin heinäkuun aikana ja siihen täytyy tehdä samat toimenpiteet kuin edelliselläkin kerralla. Koska muutoksien vieminen käsin on melkoisen työlästä ja aikaa vievää, yritetään tämä prosessi automatisoida jollain tavalla. Päätettiin toteuttaa uusi skripti-ohjelma, jonka tarkoituksena on muokattujen tiedostojen siirto Kiinan käyttöpalvelimelle. Vastaavanlainen menetelmä on jo käytössä pääpalvelimella muutosten hoitamisessa. Tähän täytyy kuitenkin lisätä jonkinlainen menetelmä tiedostojen turvalliselle siirtämiselle palvelimelta toiselle.

Palvelun tietokantojen synkronointiin aiotaan lähteä samalla suunnitelmalla kuin ennenkin. Tällä kertaa tosin painotetaan synkronoitavuuden helpottamista sekä palveluiden toimivuutta toistensa kanssa. Helpoin keino ilman uusien menetelmien käyttöön ottamista on käyttää indeksointia hyväksi tai lisätä uuteen sarakkeeseen käyttäjän alkuperäisen rekisteröintipalvelimen tunnus. Nämä menetelmät on kuvattu edellisessä aliluvussa tarkemmin. Tärkeintä olisi kuitenkin saada prosessi automaattiseksi tai vaihtoehtoisesti mahdollisimman yksinkertaiseksi.

Järjestelmän uusi versio esitteli myös uudenlaisia ongelmia, jotka ilmaantuivat vastaan kun palvelun uuden version siirtäminen uudelle palvelimelle aloitettiin. Tämä uusi versio esittää palvelun käyttämät ikonit ja kuvakkeet CSS-tyylitiedostojen avulla. Nämä ja muutkin resurssit ladataan suoraan Amazonin tarjoamasta pilvipalvelusta, CloudFron-

tista. CloudFrontia on kuvailtu aliluvussa 3.1.2. Kiinan palomuuri estää tämän palvelun toiminnan, joten resurssit täytyy ladata paikallisesti. Suurimpana ongelmana tästä on tyylitiedostojen vakioina käytetyt resurssien osoitteet.

CSS tukee periaatteellisella tasolla muuttujia tiedostoissaan, sillä kaikki selaimet eivät vielä täysin ymmärrä näitä muuttujia ollenkaan. Tästä johtuen muuttujien käyttäminen on hyvin riskialtista. Ainoa pätevä ratkaisu on muutta osoitteet osoittamaan paikallisiin resursseihin, mikä käsin tehtynä on aivan liian hidasta ja vaivalloista. Ongelma ratkeaa kuitenkin jälleen skriptin avustuksella. Lisätään muutokset siirtävään skriptiin uusi funktio, jonka tehtäväksi tulee muutettavien tiedostojen läpikäynti ja etsiä Cloudfrontiin viittaavia osoitteita. Löydettäessä nämä osoitteet korvataan viitteillä paikallisiin resursseihin.

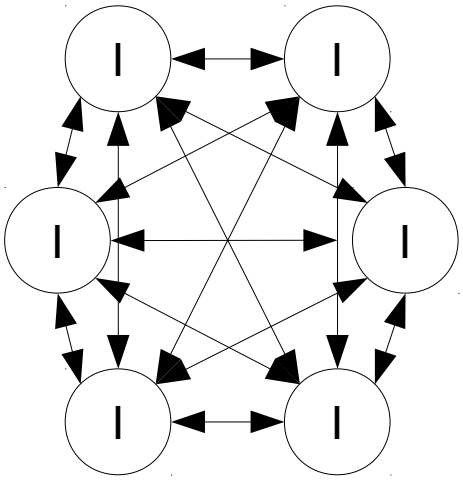
Kaikista parannuksista ja muutoksistaan huolimatta uudet palvelimet ovat vielä varsin alkeellisella tasolla. Suurimmat muutokset tehdään vielä pääpalvelimella, jotka synkronisoidaan muille palvelimille tarvittaessa. Käytössä olevien palvelimien muodostamaa verkkoa tulisi vielä kehittää jotta sen ylläpitämisestä tulisi tehokkaampaa ja helpompaa.

4 KEHITYSVAIHTOEHTOJA

Edellisessä luvussa esitetyt isäntä-renki –tyyppiset ratkaisut eivät ole parhaimpia mahdollisia ratkaisuja palvelinten välisiin suhteisiin liittyviin ongelmiin. Tämäntyyppisissä yhteyksissä palvelimilla tapahtuvia muutoksia pystytään tekemään vain yhdellä palvelimella kerrallaan eikä renki-palvelimilla pystytä tekemään itsenäisiä toimintoja. Lisäksi isäntä-palvelimia voi olla vain yksi kerrallaan koska muuten datan ylikirjoittamisen vaara kasvaa. Jotta kaikki palvelimet pystyisivät toimimaan samalla tasolla yhtenäisesti, tulisi palvelinten välinen suhde muuttaa isäntä-isäntä –tyyppiseksi. Isäntä-isäntä –yhteydessä palvelinten väliset viestit kulkevat kumpaankin suuntaan. Parhaimmassa tapauksessa palvelimia pystytään ylläpitämään useampaa kuin kahta yhtäaikaisesti.

Kuva 4 havainnollistaa isäntä-isäntä –tyyppistä yhteyttä useamman palvelimen välillä. Kuten piirroksesta voidaan havaita, yhteyksiä palvelimien välille tulee moninkertaisesti enemmän verrattuna tavalliseen isäntä-isäntä –yhteyteen, jossa tarvitaan vain yhdensuuntaiset suhteet isännän sekä kaikkien siihen liitettyjen renkien välillä. Isäntä-renki – välisessä suhteessa palvelinten välisiä yhteyksiä on yhteensä $\frac{(n-1)*n}{2}$ kappaletta [10], missä n vastaa palvelinten lukumäärää. Yhteyksien tarvittava määrä kaksinkertaistuu jos yhteyksistä halutaan kaksisuuntaisia, eli isäntä-isäntä –suhteessa.

Palvelinten välisten yhteyksien määrää voidaan pienentää miettimällä uudestaan viestien lähetystä. Mikäli palvelimet lähettäisivät viestejä vain yhdelle palvelimelle kaikkien sijaan, tulisi yhteyksistä huomattavasti kevyempiä. Tästä johtuen keskuspalvelimen perustaminen valomaan palvelinten välistä kommunikointia voisi olla järkevää. Keskuspalvelin tarkistaisi saadut viestit ja lähettäisi tarvittavat toimenpiteet eteenpäin muille palvelimille.



Kuva 4: Havainnollistava kuva isäntä-isäntä –tyyppisestä yhteydestä useammalla palvelimella

4.1 Hajautetut tietokannat

Hajautettua tietokantaa voidaan pitää datana, joka on varastoitu erillisille alueille hajautetussa järjestelmässä. Lisäksi sen voidaan katsoa olevan olemassa vain silloin, kun datan eri alueilla olevat elementit ovat toisistaan riippuvaisia tai alueiden välisen datan käsittely tulee tarpeelliseksi. Tarve hajautetuille tietokannoille nousee vaatimuksesta yhdistää olemassa olevia, maantieteellisesti hajautettuja tietokantoja yksinkertaiseksi ja loogiseksi kokonaisuudeksi sekä vaatimuksista jakaa erittäin suuret tietokannat erillisiksi maantieteellisiksi kokonaisuuksiksi. [1, s. 44]

On väitetty, ettei hajautettu järjestelmä ole kovinkaan hyödyllinen ennen kuin sen käyttämä data on myös hajautettua. Nykyaikaisissa ohjelmistojärjestelmissä data on ainakin yhä tärkeämmässä roolissa. Entisaikoina muuten hajautettu järjestelmä on voinut käyttää keskitettyä tietokantaa. Ne yritykset, jotka ovat käyttäneet jonkinlaista hajautettua tietokantaa, ovat kehittäneet omat järjestelmänsä ja kantansa. [1, s. 44]

Hajautetun tietokannan perustavoitteita on tarjota globaali tietokanta, jota voidaan käsitellä kaikista dataa tarvitsevista järjestelmän osista. Se toimii fyysisesti hajautettuna järjestelmän eri solmujen välillä, jotta data olisi mahdollisimman lähellä aina sitä tarvitsevaa järjestelmän osaa. Jokainen solmu hallitsee omaa, paikallista osaansa tietokannasta. Hallinnollisesta näkökulmasta katsottuna jokainen paikallinen tietokanta tietyllä solmulla toimii paikallisen hallinnan päätöksien tukemana. Esimerkkinä tällaisesta järjestelmästä on pankkisovellus, jonka jokaisella solmulla on omat, paikalliset asiakkaansa. Nämä solmut puolestaan lähettävät keskuspalvelimelle tietoja paikallisten käyttäjien tekemistä tilisiirroista. [1, s. 44-45]

Datan hajauttamisen tavoitteet keskitettyyn datanhallintaan verrattuna ovat yleisesti samanlaiset kuin hajautetuille järjestelmille. Kuitenkin seuraavat tavoitteet ovat erityisen merkityksellisiä tietokantojen kohdalla:

- Suorituskyky – tietokantakyselyille on saatava nopeampia vasteaikoja.
- Hinta – datan viestinnän käyttökustannuksia on vähennettävä.
- Jaettavuus – data tulee jakaa useamman maantieteellisesti erillisen solmun välillä.
- Sijainnista riippumaton käytettävyys – on tarjottava samanlainen käytettävyys ja käyttökokemus sijainnista riippumatta.
- Luotettavuus – järjestelmän toimivuus on varmistettava muutaman solmun lakattua vastaanottamasta.
- Säädettävyys – datan jakaminen ja säilöminen raskaimman käytön lähettyville on mahdollistettava.
- Kasvu – tietokannan kasvu on otettava huomioon olemassa olevilla solmuilla tai uusia solmuja on lisättävä. [1, s. 45-46]

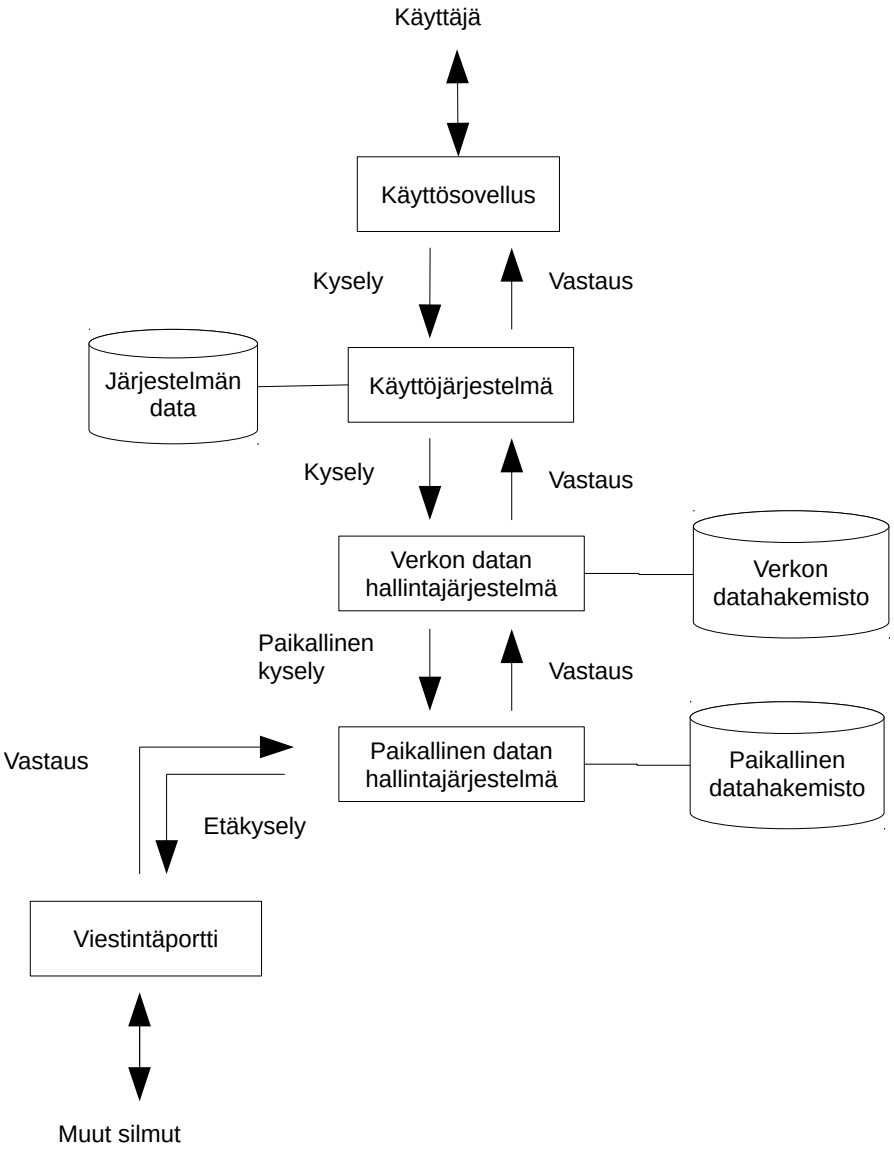
Näistä tavoitteista suorituskyky, luotettavuus, säädettävyys ja kasvu voidaan saavuttaa redundanssin keinoilla. Redundanssi aiheuttaa kuitenkin ongelmia datan eheydelle. Kaikki edellä mainitut tavoitteet esittävät mahdollisia käyttäjille koituvia etuja jotka voidaan saavuttaa käyttämällä hajautettua tietokantajärjestelmää. [1, s. 45-46]

Hajautetussa tietokantajärjestelmässä tarvittavat datan läheiset komponentit sijoitetaan järjestelmän solmuille, mikä herättää kysymyksen sovelluksen, datan, datan hallintajärjestelmän sekä datan määrittelyn jakamisesta. Jokainen näistä voidaan keskittää tai hajauttaa useammalla eri tavalla ja yhdistelmällä. Tarkoituksenmukaista datan määritelmää ja datan käsittelysovellusta tarvitaan, jos data sijaitsee useammalla eri solmulla. Datan hallintatoiminnot tulisi myös hajauttaa mikäli käyttäjäpyynnöt ja data sijaitsevat erillisillä solmuilla. [1, s. 46-47]

Kuva 5 esittää hajautetun tietokantajärjestelmän toiminnallisia komponentteja. Monet näistä komponenteista ovat samoja, joita keskitetty tietokanta käyttää, mutta kommunikaatioportti, verkon datahakemisto sekä verkon datan hallintajärjestelmä ovat hajautetun tietokannan toiminnan kannalta ominaisia osia. Verkon datahakemiston täytyy sisältää loogista informaatiota liittyen muiden järjestelmän solmujen sisältämään dataan. Solmun fyysinen sijainti sekä vaadittavat reititysinformaatiot sijaitsevat viestintäportissa, joka täyttää saman toiminnallisuuden kuin keskitetyssä järjestelmässäkin. Verkon datahallintajärjestelmän täytyy hallita kaikki maantieteellisesti jaetun järjestelmän datan käsittelyyn liittyvät tekijät. Näihin tehtäviin kuuluvat muun muassa [1, s. 47-48]:

- Tulkita koskevatko pyynnöt paikallista dataa.
- Mikäli data ei ole paikallista, käsitellä verkon datahakemistoa ja reitittää pyyntö datan sijaitsemalle solmulle.
- Mikäli pyyntö käsittelee useampaa solmua, koordinoi kaiken prosessoinnin ja vastaukset.

- Käsittelee käyttäjän pyynnöt, paikallisen ja etädatan hallinnan sekä järjestelmän hakemistot.
- Tarjoaa komentojen tai datan tulkkaustyökalut heterogeenisessä, hajautetussa järjestelmässä.



Kuva 5: Esimerkki hajautetun järjestelmä tarvitsemista komponenteista [1, s. 47, teksti käännetty suomeksi]

On syytä huomata, että kaikkien järjestelmän solmujen ei tule olla kuvan 5 esittämiä, 'täydellisiä' solmuja. Jotkin solmut voivat esimerkiksi sisältää vain paikallista dataa ilman käyttäjiä.[1, s. 48]

Hajautetut tietokannat vaikuttavat järkevältä ratkaisulta useamman palvelimen synkronoinnin ja ylläpidon aiheuttamaan ongelmaan. Muun muassa Google (Bigtable) ja Amazon (Aurora) ovat toteuttaneet tietokantoja aiemminkin. Lisäksi hajautettuja tieto-

kantoja on tutkittu jo useamman vuoden, kuten seuraavassa aliluvussa esitetty Polypheme, ja niitä kehitetään jatkuvasti eteenpäin, kuten aliluvussa 4.1.2 esitetty Bigtable.

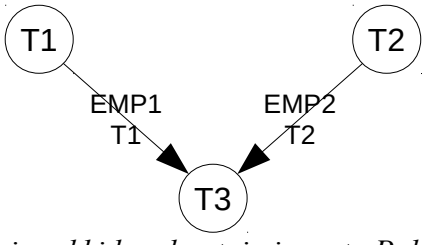
4.1.1 Polypheme, esimerkki hajautetusta tietokannasta

Polyphemen tärkeimpiin osiin kuuluu hajautettu relaatiotietokanta (distributed relational database), joka koostuu usean relaation kokoelmasta. Nämä relaatiot sijaitsevat yksittäisellä palvelimella tai vaihtoehtoisesti relaatiot on hajautettu useammalle palvelimelle tietokoneverkon sisällä [11, s. 68]. Tämänkaltainen relaatioiden jakaminen on yksi edellytys tietokannan hajauttamiselle, sillä jos relaatiot sijaitsevat vain yhden kannan sisällä, ei tietokannan hajauttaminen onnistuisi tai olisi kannattavaa. Relaatiot voidaan jakaa samalla tavalla kuin miten se on tehty Polyphemen tapauksessa: paikalliset relaatiot, jotka on tallennettu kokonaisuudessaan vain yhdelle palvelimelle, sekä globaalit relaatiot, jotka on hajautettu useamman palvelimen välillä [11, s.68]. Relaatioiden jakaminen paikallisiin ja globaaleihin auttaa tietokannan toiminnassa, sillä näin pystytään relaation tyypistä heti näkemään, tarvitseeko siihen liittyvän kyselyn suorittamiseen verkon muita kantoja.

Kuvat 6 ja 7 esittävät esimerkkikyselyä Polyphenen toiminnassa. Kysely on seuraavanlainen:

- Palvelimella 1(T1): EMP1 (NAME, ADDRESS, AGE, SALARY, JOB, PROJ)
- Palvelimella 2(T2): EMP2 (NAME, ADDRESS, AGE, SALARY, JOB, PROJ)
- Palvelimella 3(T3): PROJECT (E-NAME, P-NAME, BUDGET, DEPT)
- GLOBAL-VIEW:
  - UNION (SELECT (EMP1, PROJ='NASA' & JOB='PROGRAMMER'), SELECT (EMP2, PROJ='NASA' & JOB='ANALYST'));
  - = UNION (T1, T2) = T3
- Q_0 = PROJECT (SELECT (GLOBAL-VIEW, SALARY > 10000, NAME, JOB))

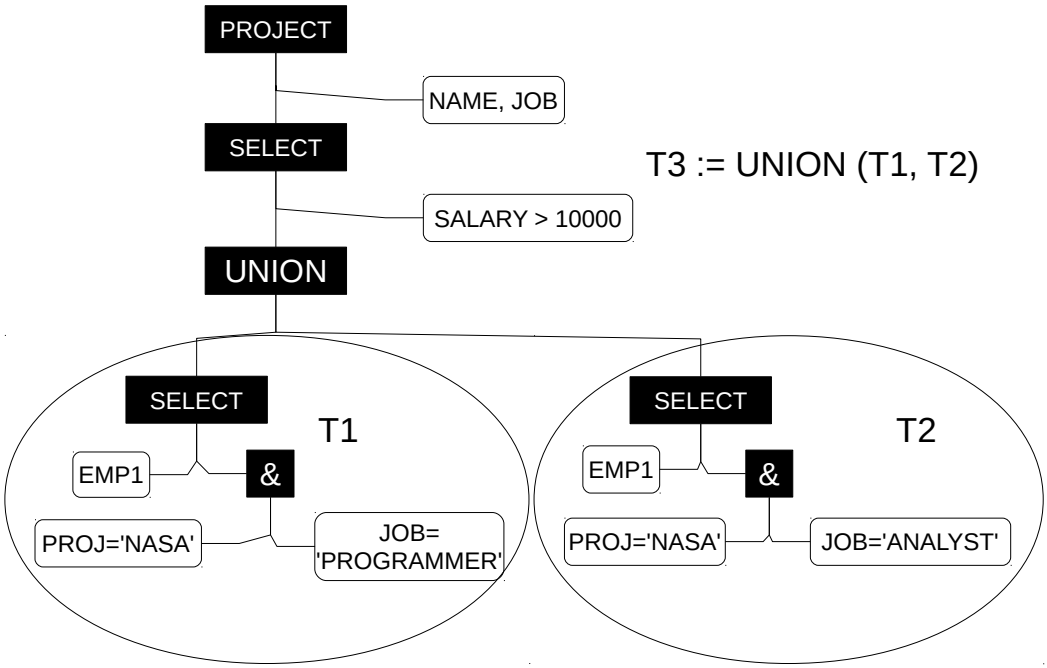
[11, s.70].



Kuva 6: Esimerkki kyselyn toiminnasta Polyphemessä [11, s.71]

Kysely Q_0 siis etsii erillisistä tietokannoista T1 ja T2 sellaisia työntekijöitä, jotka tekevät projektia NASAlle ja joiden työtehtävä on PROGRAMMER T1:ssä tai ANALYST T2:ssa. Kuva 6 näyttää tähän kyselyyn osallistuvien tietokantojen suhdetta toi-

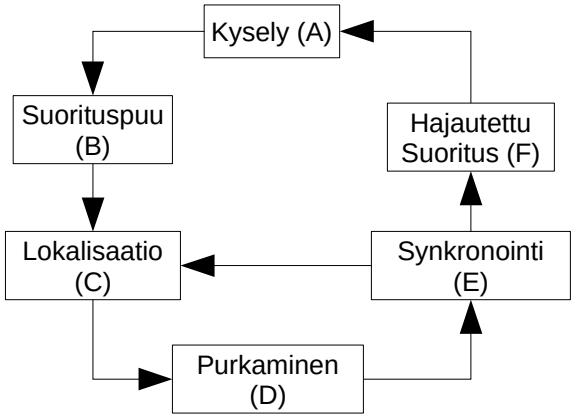
siinsa. Tässä tapauksessa T3:n kanta toimii T1:n ja T2:n kantojen unionina joka tässä tapauksessa sisältää kummankin kannan tiedot halutuilla parametreilla.



Kuva 7: Esimerkki Polyphemen suorittamasta kyselystä [11, s. 71]

Kuva 7 näyttää varsinaisen kyselyn suorituksen. Kuten piirroksesta näkyy, kyselyllä halutaan selvittää selville niiden palvelimilla T1 ja T2 olevien työntekijöiden nimet ja työtehtävät, jotka työskentelevät 'NASA' projektissa tehtävänään 'PROGRAMMER' (S1) tai 'ANALYST' (S2). Lisäehtona tuloksen työntekijöiden SALARY-arvo tulisi olla suurempi kuin 10000.

Koska kyselyt hajautettuihin tietokantoihin ovat hankalampia kuin tavallisiin, tarvitaan kyselyiden suorittamiseen hallintajärjestelmä. Polyphemen kyselynhallintaa hoitavaa järjestelmää on kuvattu kuvassa 8. Prosessi alkaa tietokannan käyttäjän tehdessä relaatiokyselyn omalla päätteellään (A). Kyselylle tehdään syntaksianalyysi samalla, kun siitä koostetaan suorituspuuta (B), jossa solmut esittävät relaatio-operaatioita tai tavallisia operandeja. [11, s. 72]



Kuva 8: Polyphemen kyselynhallintaprosessi [11, s. 73, teksti käännetty suomeksi]

Seuraava vaihe, kyselyn purkaminen (D) koostuu kahdesta osasta. Ensin toteutetaan lokalisaatio käyttäen (relaatio-)hajautusinformaatiota, joka saadaan globaalista lähteestä (C). Tässä hahmotellaan yhden palvelimen puurakenteiden maksimit, ottaen huomioon, että kaikki yksioperandiset operaatiot suoritetaan operandin palvelimella ja kaikki monioperandiset operaatiot suoritetaan samalla palvelimella kuin niiden operandit. Alipuut korvataan globaalissa järjestelmässä väliaikaisilla relaatioilla, joiden on tarkoitus vastaanottaa osatuloksia. Alipuut lähetetään palvelimille, joita ne koskevat, kuten kuvassa 6 S1 ja S2. Tässä vaiheessa käytetään myös sopeutuvaa päätösprosessia käsittelemään seuraavat suoritukset sekä synkronoidaan tiedot. Kun kaikki on valmista, voidaan kyselyt suorittaa hajautetusti oikeilla tietokannoilla (F). [11, s. 72]

Kyselyiden hajautettu suorittaminen on hankalaa, koska suoritusjärjestys täytyy ottaa huomioon. Jos alikyselyt suoritetaan pienimmässäkin määrässä poikkeavassa järjestyksessä, voivat tulokset olla täysin toisenalaiset. Tätä ongelmaa varten Polyphemessä on vuorovaikutteinen, kolmivaiheinen hallintarakenne, nimeltään DOPAGE. Se sisältää operandin ja operaattorin lokalisoinnin, päätösprosessien siirron sekä paikallisten tarpeiden toteutuksen. Mikäli tuloksia on saatavilla, tehdään tilavuuden vertailua nykyisillä raja-arvoilla. Siirrot päätetään tulosten perusteella, jolloin saadaan tulevien operaatioiden suoritusten sijainnit. Suoritus etenee tarvittavilla palvelimilla, jotka aloittavat prosessin jälleen alusta. [11, s. 72]

Koska DOPAGE ei ylläpidä tilastollista dataa, lokalisaatioon ja silmujen siirtoon vaikuttavat päätökset saadaan erillisen funktion avulla, jota kutsutaan ”raja-arvon siirtofunktioksi” (threshold Transfer function). Tämä antaa viitteitä kyselyn käyttäytymisestä annetuissa tilanteissa perustuen ”tehokkaaseen” tietoon, joka on saatu jo aikaisemmin. [11, s. 72]

Polypheme on suunniteltu abstraktien relaatiokoneiden väliseksi tietoverkoksi. Jokainen relaatiokone perustuu kolmeen komponenttiin:

- Nimiavaruus (NAME) koostuen relaatioiden ja attribuuttien kuvauksista.

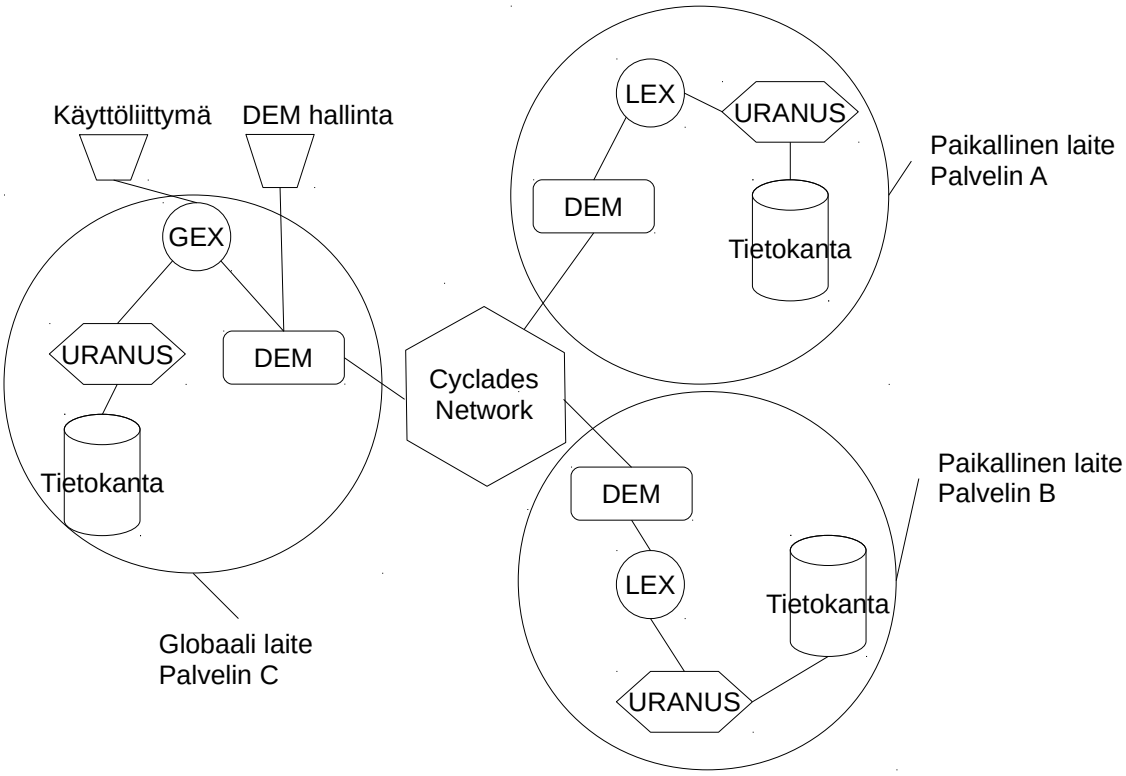
- Objektiavaruus (OBJECT) koostuen nimiavaruudessa kuvattujen relaatioinstanssien suhteista.
- Kartoitusmekanismi (MAPPING), joka toteuttaa objekti- ja nimiavaruuksien välisiä vastaavuuksia.

Jokainen relaatiokone antaa käyttäjälleen datan määrittely- ja hallintakielen perustuen aiemmin mainittuun relaatioalgebraa (Kuvat 6,7 ja 8). [11, s. 77] Näillä pystytään luomaan yhteydet kantojen välille.

On olemassa kahdenlaisia relaatiokoneita:

- Paikallisia (local machine) relaatiokoneita, joissa kaikki kolme yllä esitettyä komponenttia sijaitsevat yhdellä palvelimella. Polyphemessä paikallinen relaatiokone on rakennettu jokaisen yhteistyötä tekevän tietokannan päälle, jotta niiden käytöksestä saadaan homogeenistä.
- Globaaleja (global machine) relaatiokoneita, joissa nimiavaruus sijaitsee yhdellä laitteella samalla, kun objektiavaruus on jaettu useamman laitteen välille. Jokainen näistä laitteista sisältää paikallisen relaatiokoneen.

Globaalien relaatiokoneiden kautta Polyphemen käyttäjä pystyy käsittelemään hajautettua tietokantaa, kuten kuvassa 9 on näytetty. Käyttäjä syöttää globaalille laitteelle tavallisia pyyntöjä tai tapahtuman, joka koostuu sarjasta pyyntöjä. Järjestelmä analysoi jokaisen pyynnön ja muuttaa sen sisäisiksi puurakenteiksi, joille suoritetaan hajautusalgoritmi. Tuloksena saadaan puurakenne, jossa jokainen solmu (relaatio, operaattori) on asetettu kullekin palvelimelle. Näin ollen pystytään päättelemään puurakenteita, jotka voidaan suorittaa vain yhdellä palvelimella. [11, s. 77]



Kuva 9: Polypheme prototyypin arkkitehtuuri [11, s. 78, teksti käännetty suomeksi]

Kuten kuvasta 9 nähdään, jokainen globaali relaatiokone ja paikallinen relaatiokone koostuu kolmesta peruskomponentista, jotka ovat tietokannan hallinta URANUS, hajautettu suoritusmonitori sekä globaali tai lokaali suoritusmonitori.

URANUS on relaatiojärjestelmä, jota voidaan käyttää itsenäisesti tai relaatiokäyttöliittymänä olemassa olevalla verkostoidulla tietokannalla. Se tarjoaa datan määrittely- ja hallinnointikielen, tapahtumienhallinnan (yhtenä kokonaisuutena käsiteltävä sarja yksinkertaisia pyyntöjä), tapahtumien analysoinnin ja jäsentämisen (kaikki muutetaan puurakenteiksi) sekä puurakenteiden tulkinnan. Tämä voidaan yksinkertaistaa kahdeksi tärkeäksi komponentiksi, jäsentäjäksi relaatiopyyntöjen muuntamiseksi puurakenteiksi sekä tulkki tulkitsemaan nämä rakenteet. Kuvassa 9 URANUS toimii tietokannan käsittelijänä suorittaen sekä globaalien että lokaalien relaatiokoneiden antamia käskyjä. [11, s. 78-79]

Hajautettu suoritusmonitori tarjoaa globaaleille ja lokaaleille relaatiokoneille viestintä- ja synkronointivälineet. Yksi hajautettu suoritusmonitori on toiminnassa jokaisella verkon laitteella ja tarjoaa rinnakkaisen etätoiminnan aktivoinnit. Kahden erillisen hajautetun suoritusmonitorin välille voidaan perustaa loogiset yhteydet. Kun tarvittavia käskyjä kutsutaan, hajautettu suoritusmonitori muuntaa ne viesteiksi, jotka lähetetään tutkimukseen tarkoitetun paketinlähettäjän, CYCLADES-verkon yli vastaavalle hajautetulle suoritusmonitorille, joka vuorostaan muuntaa viestit proseduureiksi. [11, s. 79]

Kommunikointi globaalien ja lokaalien laitteiden välillä määritellään Polyphemessä erillisellä protokollalla, Polypheme Procedure Activation Protocol, eli PPAP. Tämä pro-

tokolla näkee lokaalit ja globaalit prosessit kokoelmana aktivoitavia proseduureja. PPAP perustuu hajautetun suoritusmonitorin toimintaan, joka myös määrittelee mitkä proseduurit tulisi aktivoida. Lokaali sekä globaali suoritusmonitori toteuttavat lokaaleja ja globaaleja toimintoja. [11, s. 79-80]

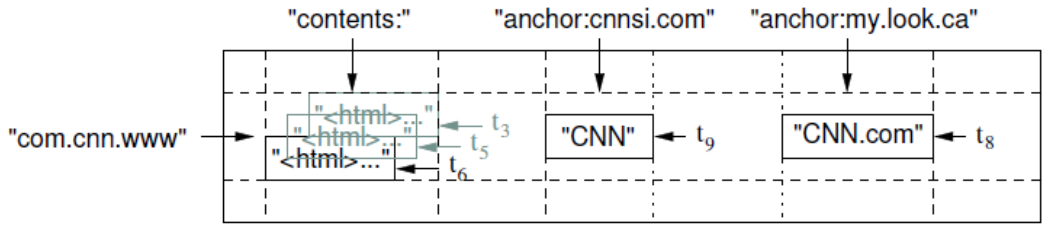
Näiden kaikkien pohjalta on saatu tehtyä Polypheme-prototyyppi hajautetuista tietokannoista. Se on kuitenkin varsin rajoitettu esimerkki, sillä siinä ei ole huomioitu yhtäaikaisia käyttäjiä eikä palautusmenetelmiä [11, s. 80], mutta se esittelee useita tärkeitä ominaisuuksia, joita hajautetuilta tietokannoilta vaaditaan.

4.1.2 Bigtable: Googlen käyttämä hajautettu tiedonhallintajärjestelmä

Google on jo useamman vuoden ajan kehittänyt ja käyttänyt itse tekemäänsä järjestelmää nimeltä Bigtable datan tallentamiseen ja käyttämiseen. Se on suunniteltu skaalautumaan luotettavasti petabitteihin asti kasvavalle datalle tuhansilla eri laitteilla. Bigtable on onnistunut useissa tavoitteissaan, sillä se on laajasti sovellettavissa ja saatavilla. Lisäksi Bigtablea on mahdollista skaalata ja se on toiminnassaan tehokas. Google käyttää Bigtablen ominaisuuksia yli kuudessakymmenessä sovelluksessa, mukaan lukien Google Analytics ja Personalized Search. Nämä palvelut käyttävät Bigtablea useilla raskailla kuormituksilla. [12, s. 1]

Bigtable muistuttaa monella tavalla tietokantoja ja käyttää useita samoja käyttöönottostrategioita. Se ei kuitenkaan täysin tue relaationaalista datamallia. Sen sijaan Bigtable tarjoaa dynaamista hallintaa datan asettelusta ja ulkoasusta. Data indeksoidaan käyttäen rivien ja sarakkeiden nimiä, jotka koostuvat merkkijonoista. Asiakasohjelmat pystyvät hallitsemaan datan paikallisuutta skeemojensa päätöksillään. Bigtablen skeeman parametrit käyttäjät voivat määrätä ladataanko data suoraan järjestelmän muistista vai levyltä [12, s. 1]

Bigtable on niukka, hajautettu, yhtämittainen ja moniulotteinen lajiteltu kartta, jota indeksoidaan rivillä, avaimella, sarakkeen avaimella sekä aikaleimalla. Tähän datamalliin kehittäjät päätyivät tarkasteltuaan useita erilaisia käyttömahdollisuuksia vastaavanlaisella järjestelmällä. Yksi tärkeimmistä suunnittelua ohjaavista esimerkeistä oli mahdollisuus säilöä suuri määrä erilaisia verkkosivuja sekä näihin liittyvää tietoa, jota voitaisiin käyttää useassa erilaisessa projektissa. Tätä tiettyä taulua nimitetään *Webtableksi*, joka on kuvattu kuvassa 10. Taulun riviavaimet ovat sattumanvaraisia merkkijonoja (enintään 64 kB). Jokainen luku- ja kirjoitusoperaatio yhden rivin alueella on atominen, jotta rinnakkaisten tietopäivitysten hallinnan helpottamiseksi. [12, s. 2]



Kuva 10: Osa Bigtablen taulua, johon tallennetaan verkkosivujen tietoa [12, s. 2]

Datan hallitsemiseksi Bigtable käyttää ylläpitämiseen sanakirjamaista järjestystä lajiteltuna riviavainten mukaisesti. Rivietäisyys taulukossa on dynaamisesti partitioitu ja jokaista rivietäisyyttä kutsutaan *tablet*iksi, joka on hajautuksen ja kuormituksen tasauksen yksikkö. Tästä johtuen lyhyen rivietäisyyden luku on tehokasta ja se vaatii tyypillisesti kommunikointia vain muutamalta verkoston koneelta. Esimerkiksi yllä mainitussa Webtablessa saman domainin alla olevat verkkosivut on ryhmitelty yhteen vierekkäisiksi riveiksi kääntämällä URL:n isäntäkomponentit. Avain **maps.google.com/index.html** tallennetaan siis avaimen **com.google.maps/index.html** alle. [12, s.2]

Sarakeavaimet on ryhmitelty yhteen joukoiksi nimeltä sarakeperheet, jotka muodostavat kulunvalvonnan perusosan. Kaikki sarakeperheeseen säilötty data on yleensä samantyyppistä. Sarakeperhe on luotava, ennen kuin siihen kuuluviin sarakeavaimiin on mahdollista tallentaa dataa. Sarakeavainten täytyy olla tulostettavia merkkijonoja, mutta muuten ne voivat koostua sattumanvaraisista merkeistä. Esimerkki sarakeperheestä Webtablessa on kieli, johon tallennetaan jokaisen sivun kielitunniste. Kulunvalvonta ja levyn sekä muistin hallinta suoritetaan sarakeperheen tasolla. [12, s.2]

Jokainen Bigtablen solu voi sisältää useita eri versioita samasta datasta. Nämä indeksoidaan käyttäen aikaleimaa, joka on 64-bittinen kokonaisluku. Bigtable voi määrittää leiman, jolloin se esittää ”oikeaa” aikaa mikrosekunneissa. Vaihtoehtoisesti käyttäjäohjelma asettaa oman leimansa datalle. Sovellusten, jotka haluavat välttää datan törmäämistä, tulee käyttää yksilöllisiä aikaleimoja. Solun eri versiot tallennetaan laskevassa aikaleimajärjestyksessä. Jotta datan hallinnasta tulisi kevyempää, on Bigtablessa roskienkeruujärjestelmä, jolle käyttäjäohjelmat voivat määritellä kuinka monta uusinta versiota tietystä datasta säästetään. [12, s. 2-3]

Bigtablen sovelluksen ohjelmointirajapinta, API tarjoaa funktioita tietokantataulujen ja sarakeperheiden luomiseen ja tuhoamiseen sekä muokkaamaan näihin liittyvää metadataa kuten käyttöoikeuksia. Asiakasohjelmat voivat kirjoittaa ja poistaa arvoja, etsiä yksittäisiä rivejä tai iteroida datan osajoukkoa taulussa käyttäen C++-ohjelmointikieltä. Kuva 11 tarjoaa esimerkin datan kirjoittamisesta Bigtableen käyttäen RowMutation-abstraktiota usean päivityksen toteuttamiseen. Applyn kutsuminen tekee atomisen mutaation Webtableen poistaen yhden ankkurin ja lisäten uuden, www.cnn.comin. Kuva 12 puolestaan esittää datan lukemista Bigtablesta käyttäen Scanner-abstraktiota iteroimaan

kaikkien ankkureiden yli tietyllä rivillä. On olemassa useita tapoja rajoittaa skannauksen tuottamia rivejä, sarakkeita ja aikaleimoja. Käyttäjä voi esimerkiksi rajoittaa kuvan 12 esittämän skannauksen etsimään vain ankkureita, joiden sarakkeet vastaavat yleistä ilmausta **anchor:*.cnn.com**, tai vain niitä, joiden aikaleima sijoittuu 10 päivän sisälle nykyisestä päivämäärästä. Bigtable tukee useita muitakin menetelmiä, jotka mahdollistavat datan käsittelyn mutkikkaammilla tavoilla, ja sitä voidaan käyttää muun muassa MapReducella, joka on Googlen yhtäaikaiseen suuren skaalan laskentaan kehitetty viitekehys.[12, s. 3]

```
// Open the table
Table *T = OpenOrDie("/bigtable/web/webtable");

// Write a new anchor and delete an old anchor
RowMutation r1(T, "com.cnn.www");
r1.Set("anchor:www.c-span.org", "CNN");
r1.Delete("anchor:www.abc.com");
Operation op;
Apply(&op, &r1);
```

Kuva 11: Datan kirjoittamisen esimerkki Bigtablessa.[12, s. 3]

```
Scanner scanner(T);
ScanStream *stream;
stream = scanner.FetchColumnFamily("anchor");
stream->SetReturnAllVersions();
scanner.Lookup("com.cnn.www");
for (; !stream->Done(); stream->Next()) {
  printf("%s %s %lld %s\n",
         scanner.RowName(),
         stream->ColumnName(),
         stream->MicroTimestamp(),
         stream->Value());
}
```

Kuva 12: Datan lukeminen Bigtablesta käyttäen Scanner -ominaisuutta[12, s. 3]

Bigtable on rakennettu useamman muun Googlen kehittämän järjestelmän infrastruktuurin päälle käyttäen Google File Systemsiä loki- ja datatiedostojen säilömiseen. Bigtable-ryhmittymä toimii tyypillisesti jaetussa laitteistolähteessä, joka suorittaa myös monenlaisia muita hajautettuja sovelluksia. Bigtable on riippuvainen ryhmittymän hallintajärjestelmästä, prosessointiajan järjestelemisestä, jaettujen laitteiden resurssien hallinnasta, laitteistojen virhetilanteiden hallinnasta sekä laitteiston tilan valvonnasta. Googlen SSTable-tiedostorakennetta käytetään sisäisesti Bigtablen datan varastoimiseen. Lisäksi Bigtable nojaa suuresti saatavilla olevaan, yhtämittaiseen hajautettujen järjestelmien lukkojen hallintaan nimeltä Chubby. Chubby koostuu viidestä aktiivisesta repli-

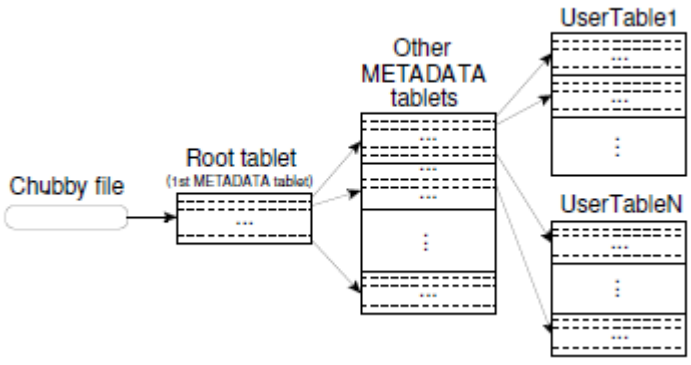
kaatiosta, joista yksi valitaan toimimaan isäntäjärjestelmänä ja vastaamaan aktiivisesti järjestelmän pyyntöihin. [12, s. 3-4]

Bigtablen toteutus sisältää kolme tärkeää komponenttia:

- jokaiseen asiakasohjelmaan yhdistettävä kirjasto
- yksi isäntäpalvelin
- useita tablet-palvelimia.

Tablet-palvelimia voidaan lisätä tai poistaa ryhmittymästä dynaamisesti kuormituksen sisällyttämiseksi. Isäntä on vastuussa taulukoiden asettamisesta tablet-palvelimille, tarkastellen tablet-palvelimien lisäys- ja poistotarpeita sekä tasapainottaen tablet-palvelinten kuormitusta ja roskienhallintaa. Jokainen tablet-palvelin hallitsee osaa järjestelmän taulukoista, mikä tyypillisesti on noin 10 – 1000 kappaletta. Palvelin huolehtii näihin taulukoihin tehtävistä kirjoitus- ja lukuoperaatioista sekä jakaa taulukoita, mikäli ne kasvavat liian suuriksi. Kuten monissa muissakin yhden isännän hajautetuissa varastointijärjestelmissä, asiakkaiden data ei kulje isännän kautta. Sen sijaan asiakkaat kommunikoivat suoraan tablet-palvelinten kanssa kirjoitus- ja lukuoperaatioita tehtäessä. Tästä johtuen useimmat asiakasjärjestelmät eivät koskaan kommunikoi isäntä-palvelimen kanssa, keventäen sen kuormitusta. [12, s. 4]

Taulukoiden sijaintitietoihin Bigtable käyttää kolmikerroksista hierarkiaa vastaavasti kuin B+-puu. Rakenne on esitettynä kuvassa 13. Ensimmäisellä tasolla on tiedosto tallennettuna Chubbylle, joka sisältää root tabletin sijainnin. Tämä sisältää kaikkien sen vastuulle kuuluvien taulukoiden sijainnit erillisessä METADATA-taulukossa. Jokainen näistä METADATA-taulukoista sisältää ryhmän käyttäjien taulukoita. Ensimmäinen näistä taulukoista on Root tablet, mutta sitä ei koskaan jaeta pienemmiksi, jotta taulujen sijaintihierarkiassa ei olisi kolmea kerrosta enempää. [12, s. 4-5]



Kuva 13: Bigtablen puurakenne taulukoiden sijaintitiedon säilyttämiseen [12, s. 4]

Jokainen taulukko kuuluu aina yhden tablet-palvelimen alaisuuteen. Isäntäpalvelin ylläpitää listaa käytössä olevista tablet-palvelimista sekä mille palvelimelle mikin taulukko kuuluu ja mitä taulukoita ei ole vielä asetettu palvelimille. Mikäli taulukkoa ei ole

asetettu millekään palvelimelle ja tarpeeksi tilaa sisältävä palvelin on saatavilla, asettaa isäntä taulukon lähettämällä taulukon latauspyynnön tablet-palvelimelle. Bigtable käyttää Chubbya tablet-palvelinten seuraamiseen. Aina uuden palvelimen käynnistyessä se luo yksilöllisen lukituksen ennalta määriteltyyn Chubbyn tiedostokansioon. Isäntäpalvelin seuraa tätä kansiota valvoakseen tablet-palvelimia. [12, s. 5]

Mitatakseen Bigtablen suorituskykyä kehittäjät loivat Bigtable-ryhmittymän joka sisälsi N kappaletta tablet-palvelimia. Palvelimet asetettiin käyttämään 1 GB muistia ja kirjoittamaan Googlen tiedostojärjestelmän soluun, joka koostui 1786 laitteesta. Näistä laitteista jokaisessa oli kaksi 400 GB IDE kovalevyä, kaksi kaksiydin Opteron 2GHz:n prosessoria, tarpeeksi fyysistä muistia kaikille käynnissä oleville prosesseille sekä yksi gigabitin Ethernet linkki. N käyttäjälaitetta loi testissä käytetyn Bigtablen kuormituksen. Varmistaakseen ettei käyttäjistä aiheutunut järjestelmän pullonkaulaa, käytettiin yhtä monta palvelinta kuin käyttäjääkin. Laitteet järjestettiin kaksikerroksiseen puumuotoiseen verkkoon, jolla oli noin 100-200 Gps:n kaistanleveys. Kaikki laitteet sijaitsivat samassa tilassa, jolloin minkä tahansa laitteiden väliseen viestintään kulunut aika oli alle millisekunti. R on testissä käytettyjen, erillisten Bigtablen riviavainten lukumäärä. R valittiin siten, että jokainen luku- ja kirjoitusoperaatio käsittelisi vähintään 1 GB:n sisältämää dataa. Taulukossa 1 esitetään vertailukokeiden tulokset jokaisen palvelimen tekeminä operaatioina sekuntia kohden. [12, s. 8]

Taulukko 1: Bigtablen vertailukokeiden tuloksia, jokaisen tablet-palvelimen tekemien 1000-bittisten arvojen luku-ja kirjoitusoperaatioiden määrä sekunnissa. [12, s. 9, käännetty suomeksi]

	Tablet -palvelinten lukumäärä			
Koe	**1**	**50**	**250**	**500**
Satunnainen luku	1212	593	479	241
Satunnainen luku (muistista)	10811	8511	8000	6250
Satunnainen kirjoitus	8850	3745	3425	2000
Peräkkäinen luku	4425	2463	2625	2469
Peräkkäinen kirjoitus	8547	3623	2451	1905
Luku Scan-operaatiolla	15385	10526	9524	7843

Peräkkäisen kirjoittamisen (sequential writes) vertailu käytti 0:sta R-1:een nimettyjä riviavaimia, joka partitioitiin 10 N yhtä suureen etäisyyteen. Nämä etäisyydet asetettiin N:lle asiakkaalle keskusvuorottimella, joka asetti seuraavat saatavilla olevat etäisyydet asiakkaalle niin pian kun asiakas oli saanut edellisen asetetun etäisyyden prosessoitua. Tämänkaltainen dynaaminen etäisyyksien asettaminen auttoi pienentämään laitteistolla

suoritettavien muiden prosessien aiheuttamia suorituksen varioinnin vaikutuksia. Testissä jokaisen riviavaimen alle kirjoitettiin yksi merkkijono. Merkkijonot luotiin satunnaisesti ja yksilöllisiksi, jotta rivien välinen pakkaus ei olisi mahdollista. *Satunnaisen kirjoittamisen* (random write) vertailu toimi muuten samalla periaatteella, mutta riviavain oli hajautettu modulo R ennen kirjoitusta. [12, s. 9]

Peräkkäisen lukemisen (sequential read) vertailu loi riviavaimia täsmälleen samalla tavalla kuin peräkkäisen kirjoittamisen vertaus, mutta riviavaimelle kirjoittamisen sijaan se luki merkkijonon riviavaimen alla. Merkkijonot luotiin kirjoitusvertailussa. *Satunnaisen lukemisen* (random read) vertailu toteutettiin vastaavasti samalla tavalla, mutta seuraten satunnaisen kirjoittamisen merkkijonoja. [12, s. 9]

Skannaus (scan) toimi samantyylisenä vertailuna kuin peräkkäinen lukeminen, mutta käytti Bigtablen API:n tarjoamaa tukea lukemaan kaikki arvot tietyllä riviväliillä. Skannauksen käyttö vähentää toteutettavien etäproseduurikutsujen määrää, sillä yhdellä kutsulla noudetaan suuria arvosarjoja tablet-palvelimelta. [12, s. 9]

Satunnainen luku muistista (random read (mem)) vertailu on saman tyylinen kuin satunnainen luku, mutta lokaalisuusryhmä, joka sisältää vertailudatan, on merkitty muistiin ja lukuoperaatiot toimivat tablet palvelimen muistista globaalin tiedostojärjestelmän sijaan. Jotta data mahtuisi järjestelmän muistiin, datan määrä laskettiin palvelinta kohti 1 GB:sta 100 MB:iin. [12, s. 9]

Bigtable -järjestelmän koottu suoritusteho kasvaa dramaattisesti kun tablet-palvelinten määrää järjestelmässä nostetaan yhdestä 500:aan. Esimerkiksi satunnainen lukeminen muistista nousee lähes 300-kertaiseksi. Tämä kasvu voidaan selittää vertailussa olevalla pullonkaulalla, joka johtuu yksittäisten tablet-palvelinten prosessointiyksiköissä. Suoritusteho ei kuitenkaan kasva lineaarisesti ja useimmissa vertailuissa palvelinkohtaiset läpikäynnit laskevat siirryttäessä yhdestä palvelimesta 50:een. Tämän tehon putoamisen aiheuttaa kuormituksen epätasapaino usealla erillisellä palvelinkonfiguraatiolla. [12, s. 9-10]

Elokuussa 2006 Bigtable oli jo käytössä 388 ei-kokeellisessa ryhmittymässä erilaisilla Googlen laitteistojärjestelmillä. Järjestelmää käytetään muun muassa Google Analyticsissä, Google Earthissä sekä henkilökohtaisessa hakukoneessa. [12, s. 10-11] Kirjoitushetkellä Bigtable on saatavilla ilmaisella kokeiluajalla kaikille halukkaille kehittäjille beta-versiona.

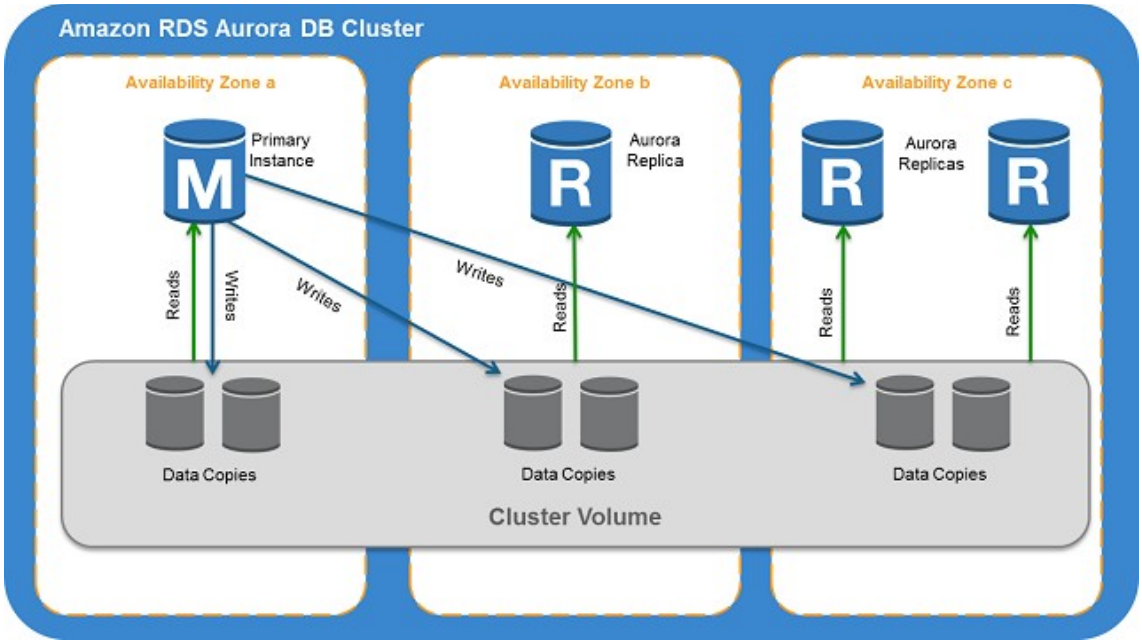
4.1.3 Amazon Aurora

Amazon Aurora on Amazonin kehittämä, ja palvelintensa käyttäjille tarjoama tietokantapalvelu. Se on MySQL-yhteensopiva relaatiotietokantajärjestelmä, joka yhdistää markkinoiden parhaiden tietokantojen nopeuden ja saatavuuden avoimen lähdekoodin yksinkertaisuuteen ja hinta-laatusuhteeseen. Lisäksi Aurora toimii tehokkaammin kuin tavallinen MySQL samalla laitteistolla [13]. Kaikki menetelmät, työkalut, ohjelmistokoodit sekä sovellukset, jotka toimivat MySQL:n yhteydessä, ovat myös yhteensopivia

Amazon Auroran kanssa. Amazonin tarjoama relaatiotietokantapalvelu hoitaa automaattisesti tietokantojen varaamisen, paikkaukset, varmuuskopiot, palautukset, vianhallinnan sekä tarvittavat korjaukset. [13]

Amazon Auroran instanssia luotaessa luodaan tietokantaryhmittymä, joka koostuu yhdestä tai useammasta instanssista sekä tilavuuden ryhmittymästä, joka hallitsee näiden instanssien sisältämää dataa. Auroran tilavuuden ryhmittymä on virtuaalinen tietokantavarasto, joka käsittää useita saatavuusalueita, joista jokaisella sijaitsee kopio ryhmitellystä datasta. Kahdenlaiset instanssit määrittävät Auroran tietokantaryhmän: ensisijainen instanssi sekä yksi tai useita Auroran replikaatioita. Ensisijainen instanssi, joita on yksi jokaisella Auroran tietokantainstanssilla, tukee luku- ja kirjoitusoperaatioiden kuormitusta sekä suorittaa kaikki datan muokkaukset tilavuuden ryhmittymälle. Auroran replikaatiot tukevat ainoastaan lukuoperaatioita. Jokainen tietokantaryhmittymä voi sisältää enintään 15 replikaatiota ensisijaisen instanssin lisäksi. Useat Auroran replikaatiot jakavat datan lukemisesta seuraavaa kuormitusta ja sijoittamalla Auroran replikaatiot erillisille saatavuusalueille pystytään lisäämään tietokannan yleistä saatavuutta. [13]

Kuva 14 esittää Auroran ensisijaisen instanssin ja replikoiden välistä suhdetta useamman saatavuusalueen välillä sekä näiden mahdollisten operaatioiden toteutumista. Saatavuusalueella A oleva tietokanta toimii ensisijaisena instanssina ja B sekä C toimivat A:n replikaatioina. Tarvittava data kopioidaan replikaatioille, mistä ne voivat lukea sitä helposti. Viittaushetkellä Amazon Aurora on saatavilla itä- ja länsi-Yhdysvalloissa sekä Euroopassa. Palvelimet sijaitsevat Pohjois-Virginiassa, Oregonissa sekä Irlannissa. [13]



Kuva 14: Amazon Auroran Ensisijaisen instanssin ja Replikaatioiden välinen suhde [13]

Jokaisella Auroran tietokantaryhmittymällä on ryhmittymän päätepiste, johon käyttäjä voi ottaa yhteyden. Päätepiste koostuu verkkoalueen nimestä sekä portista, esimerkiksi **mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com:3306**. Ensisijaisella instanssilla sekä kaikilla replikaatioilla on omat yksilölliset päätepisteensä, joiden tunnisteessa ei ole mukana sanaa cluster. Ryhmittymän päätepiste yhdistää käyttäjän ensisijaiseen tietokantaryhmittymän instanssiin ja päätepistettä käyttäen pystytään tekemään kirjoitus- ja lukuoperaatioita. Ensisijaisella instanssilla on myös uniikki päätepiste, sillä ryhmittymän päätepiste osoittaa aina ensisijaiseen instanssiin. Mikäli tämä instanssi lakkaa vastaanottamasta, ryhmittymän päätepiste siirretään uudelle ensisijaiselle instanssille. Tietokannan asiakasohjelmat pystytään ohjaamaan eri replikaatioille Auroran tietokantaryhmittymässä käyttäen replikaatioiden yksilöllisiä päätepisteitä. [13]

Esimerkkinä toimikoon Amazon Auroran tietokantaryhmittymä, joka sisältää kaksi Auroran replikaatiota eri saatavuusalueella kuin ensisijainen instanssi. Avaamalla yhteyden ryhmittymän päätepisteeseen käyttäjä pystyy lähettämään sekä luku- että kirjoitusoperaatioita ensisijaiselle instanssille. Käyttäjä voi myös avata yhteyden jollekin replikalle ja lähettää kyselyitä suoraan näille tietokannan instansseille. Mikäli ensisijainen instanssi tai siihen liittyvä saatavuusalue lakkaa vastaanottamasta viestejä, kohottaa Amazonin relaatiotietokantapalvelu jonkin replikoista uudeksi ensisijaiseksi instanssiksi ja päivittää ryhmittymän päätepisteen DNS-rekisterin osoittamaan kohti uutta instanssia. Sovellus pystyy jatkamaan kirjoitus- ja lukuoperaatioiden lähettämistä Auroran tietokantaryhmittymälle minimaalisella häiriöllä. [13]

Amazon Auroran data säilötään ryhmittymän varastoon, joka on yksittäinen, virtuaalinen varasto puolijohdelevyasema SSD:tä käyttäen. Ryhmittymän sisältämä data kopioidaan jokaiselle järjestelmään kuuluvalle saatavuusalueelle, jotka muodostavat yhdessä ryhmittymän käyttämän tietokannan. Koska data replikoidaan automaattisesti useammalle saatavuusalueelle, on se näin ollen kestävää ja hävikin mahdollisuus on häviävän pieni. Replikointi mahdollistaa myös datan saatavuuden kasvattamisen virhetilanteiden tapahtuessa, sillä datan kopioita on jo valmiiksi olemassa eri saatavuusalueilla ja tietokantaryhmittymän pyyntöjen palvelemista voidaan jatkaa eri instansseille. Ryhmittymän tilavuus kasvaa automaattisesti samalla, kun tietokannan datan määrä lisääntyy. Suurin mahdollinen tilan määrä on 64 TB. [13]

Auroran tehokkuus perustuu virtuaalisen SSD-varastointikerroksen läheiseen tietokantajärjestelmän integrointiin. Varastointi on vikasietoinen sekä itsensä korjaava. Levyjen hajoamiset korjataan taustalla ilman tietokannan toimivuuden kärsimistä, sillä Aurora on suunniteltu havaitsemaan tietokannan kaatumiset automaattisesti ja käynnistymään uudelleen ilman kaatumisten palautusta tai tietokannan välimuistin uudelleenrakentamista. Mikäli koko järjestelmä lakkaa toimimasta, Aurora siirtää toiminnan yhdelle olemassa olevista replikaatiosta, mikäli käyttäjä on ottanut tietokannan replikoinnin käyttöön. [13]

Taatakseen tietokannan toiminnan ja tehokkuuden Amazon Aurora käyttää useita erilaisia ohjelmisto- ja laitteistotekniikoita. Kirjoitus- ja tulostusoperaatiot käyttävät hajautettujen järjestelmien tekniikoita kuten quorumia parantaakseen suorituksen yhtenäisyyttä. Käyttäjät voivat tehdä Aurora replikoita jakaakseen suurista lukuoperaatioista seuraavaa kuormitusta useammalle palvelimelle. Auroran replikat käyttävät samaa varastointia kuin lähdeinstanssi laskien järjestelmän kulutusta sekä välttäen replikasolmuilla tapahtuvia kirjoitusoperaatioita. Tämä vapauttaa prosessointitehoja lukutoimitusten tekemiseen sekä vähentää replikan kokemaa viivettä parhaimmillaan jopa muutamaan millisekuntiin. [13]

Amazonin relaatiokantapalvelu valvoo jatkuvasti sekä Auroran tietokannan että palvelimen tilaa. Mikäli tietokanta lakkaa vastaanottamasta, käynnistää Amazon RDS sen ja siihen liittyvät muut prosessit uudelleen automaattisesti. Jos uudelleenkäynnistys ei korjaa ongelmaa, Aurora pyrkii korvaamaan tietokannan jollain sen replikoista. Jos replikoita ei ole olemassa, Aurora yrittää luoda uuden tietokantainstanssin automaattisesti. Aurora ei tarvitse järjestelmän kaatumisraportteja, mikä vähentää huomattavasti uudelleenkäynnistykseen kuluvaa aikaa. Aurora myös eristää tietokannan välimuistipuskurin tietokantaprosesseista, jolloin välimuisti pystyy selviämään tietokannan uudelleenkäynnistyksestä. [13]

Käyttäjät pystyvät määrittämään erilaisia käyttäjäryhmiä, joilla on omat oikeutensa Auroran tietokannan tietojen muokkaamiseen. On mahdollista luoda esimerkiksi kaksi tietokantainstanssia, ”kehitys” ja ”tuotanto”, joista ”kehitystä” pystyvät muokkaamaan järjestelmän kehittäjät ja ylläpitäjät ja ”tuotantoa” vain ylläpitäjät. Aurora käyttää datan siirronaikaiseen salaukseen SSL (AES-256) sertifikaattia. [13]

Aurora vaikuttaa pätevältä valinnalta tietokantojen datan synkronoimiseen. Varsinkin päätepisteiden käyttö mahdollistaa monia käytännön asioita, kuten toisella saatavuusalueella olevan järjestelmän pyytämän kirjoituksen pääpalvelimelle ja sitä kautta synkronoituna muille palvelimille. Tietokannan replikat auttavat myös datan säilyvyydessä sekä laajentavat tietokannan saatavuusaluetta.

4.2 Hajautetun ohjelmoinnin semanttisuudesta

Yleisesti hajautetut järjestelmät ohjelmoidaan alhaisella tasolla, jokainen solmu erikseen. Solmut koostuvat usein tilakoneesta, joka välittää muille solmuille paikallisia tapahtumia, kuten solmun omaa tilaa koskevia tietoja sekä paikallisen tilan muutoksia. Suunnittelijat ovat dkuitenkin kiinnostuneita vain järjestelmän korkeatasoisista ominaisuuksista, kuten saatavuudesta, tulosten johdonmukaisuudesta sekä järjestelmän kyvystä sietää virheitä. Näitä korkean tason ominaisuuksia ei voida valitettavasti yksikäsitteisesti määritellä tai tarkkailla, koska ne ilmaantuvat järjestelmän yksittäisten silmujen yhteistyön seurauksena. [14] Tämä on yksi hajautettujen järjestelmien luomisen vaikeimmista ongelmista. Vaikka järjestelmän pohjana ovat useammat yksittäiset solmut, koko-

naistuloksena tärkeintä on kuitenkin niiden yhteistyön sujuvuus ja viestien välittäminen muille järjestelmän solmuille.

Järjestelmää tehtäessä ohjelmoijan täytyy määritellä alhaisen tason toiminnot jokaiselle paikalliselle tapahtumalle yrittäen samalla saavuttaa korkeamman tason globaaleja ominaisuuksia. Tämä semanttinen ero matalan ja korkean tason ohjelmoinnin välillä vaikeuttaa järjestelmien käyttöönottoa, virheidenkorjausta sekä valvontaa. [14]

Druschel yrittää lähentää korkean ja matalan tason ohjelmointia deklaratiivisen verkonkäytön avulla. Ohjelmoijat käyttävät laajennettua tietokantakieltä ohjelmoidakseen deklaratiivisesti. Näin ohjelmista saadaan ytimekkäitä, voidaan määritellä ei-lokaaleja ominaisuuksia suoraan sekä abstrahoida useita matalan tason yksityiskohtia. Varsinainen ohjelma muistuttaa myös sen suunnitteluun käytettyä pseudo-koodia, ja siihen voidaan lisätä ajonaikaisia kyselyitä valvomaan järjestelmän tilaa. [14]

Deklaratiivisen verkonkäytön tekniikat ovat vielä kuitenkin varhaisessa vaiheessa ja tekniikoiden valmistuminen käyttökelpoisiksi voi viedä paljon aikaa. Mutta mikäli tekniikka saadaan kehitettyä tarpeeksi tehokkaaksi, helpottaisivat deklaratiivisen verkonkäytön tekniikat kehitys- ja ajattelutapa hajautettujen järjestelmien kehittämistä.

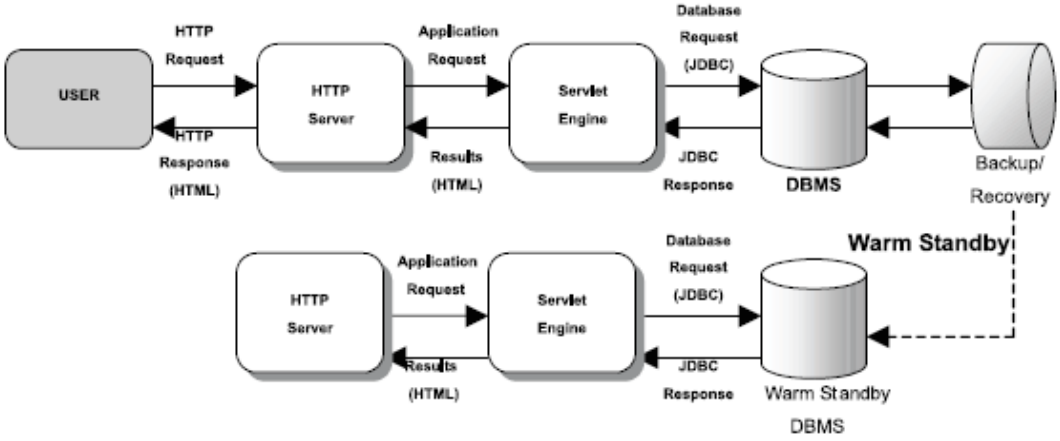
4.3 Tietokanta palveluna

Nykyään datan tehokas käsittely on tavallisimpia ja tärkeimpiä asioita kaikissa tieteellisissä, akateemisissa sekä taloudellisissa järjestöissä, jotka asentavat ja hallitsevat erilaisia tietokannan hallintajärjestelmiä tyydyttääkseen datan käsittelytarpeensa. On toki mahdollista ostaa tarvittavat laitteistot ja palkata työntekijät huolehtimaan järjestelmän toiminnasta, mutta tästä on tulossa kallis ja epäkäytännöllinen lähestymistapa, kun tietokantajärjestelmät ja niiden ongelmat kasvavat suuremmiksi ja mutkikkaammiksi. Tietokannan varmuuskopiointi, palautus sekä uudelleenorganisointi ja varsinkin uuteen järjestelmäversioon siirtyminen kuluttavat suuresti järjestön resursseja. [15, s. 1]

Tietokannan käyttö palveluna -paradigma haastaa perinteisen mallin datan hallinnasta, jota nykyiset järjestöt suosivat. Tietokantapalvelun tarjoaja tarjoaa saumattoman menetelmän luoda, tallentaa ja käsitellä tietokantoja, pitäen päävastuun asioiden sujuvuudesta itsellään. Halutessaan käsitellä dataa palvelun, käyttäjät käyttävät sen tarjoamia laitteistoja ja ohjelmia. Kaikki tietokannan toimivuuteen liittyvät velvollisuudet jätetään palvelun tarjoajan vastuulle. [15, s. 1-2]

Tämä lähestymistapa datan hallintaan nostaa esille uusia haasteita. Ennen kaikkea ongelmaksi nousee datan yksityisyyden suoja. Useimmat yritykset pitävät dataansa erittäin arvokkaana resurssina, joten tietokantapalvelun tarjoajan tulee taata datan säilyvyys ja turvallisuus. Toisena ongelmana on suorituskyky, johon vaikuttaa yhteyden muodostaminen tietokantaan verkon yli. Myös tarvittavan ja kunnollisen käyttöliittymän luominen voi osoittautua haasteeksi. [15, s. 2]

Hacigümüs et al. kehittivät NetDB2, kokeellisen tietokannan, jota käytetään palvelun tavoin. Tätä on käytetty mm. yliopistoissa helpottamaan tietokantakurssien opettamista eri paikoissa. NetDB2 tarjoaa tietokantapalveluita käyttäjilleen Internetin välityksellä, kuten työkaluja sovellusten kehitykseen, taulukoiden luomiseen ja lataamiseen sekä kyselyiden ja tapahtumien tekemiseen. Järjestelmän käyttöliittymä toimii erillisellä sovelluksella tai selaimen kautta. Kuvassa 15 esitetään NetDB2:en kolmikerroksista järjestelmäarkkitehtuuria, joka koostuu esitys-, sovellus- ja datan hallintakerroksista. Kerroksiin jakamisesta on kahdenlaista hyötyä: ohjelmistokomponentit pystytään eristämään kerroksittain ja saavutetaan parempi yhteentoimivuus sekä suurempi skaalautuvuus. [15, s. 2-3]



Kuva 15: NetDB2:n arkkitehtuurikuva [15, s. 2]

NetDB2:n esityskerros koostuu käyttäjän verkkoselaimesta sekä NetDB2:n HTTP-palvelimesta, jossa verkkoselain on vastuussa käyttäjälle näytettävästä käyttöliittymästä ja palvelin hallitsee kommunikointia. Sovelluskerros koostuu Java servleteistä, joita hallitsee servlet moottori. HTML-sivut luodaan ja ohjataan esityskerrokselle perustuen käyttäjältä tulleisiin komentoihin. Datan hallintakerros koostuu tietokannan hallitsijasta sekä varmuuskopiointi- ja palautuspalvelimesta. Servlet-moottori kommunikoi tietokannan kanssa käyttäen JDBC-protokollaa. Tietokantapalvelin ja varmuuskopiointi- ja palautus-palvelin kommunikoivat turvatun, korkeanopeuksisen sekä yksityisen yhteyden yli ilman käyttäjien vaikutusta. [15, s. 3]

Käyttäjät pääsevät käsiksi NetDB2:n tietokantaan Internetin kautta ja heidän tekemiensä kyselyiden tulokset palaavat samaa reittiä takaisin. Tästä aiheutuu yleisrasitetta jota Hacigümüs et al. kutsuvat palvelun lähetysrangaistukseksi. He tutkivat NetDB2:en suoritustehokkuutta käyttäen alan standardin TPC-H:n mittapuuta asteikolla 0,1, 1 ja 10. Yleisrasitetta ei ilmaantunut lainkaan kun kyselyt tehtiin NetDB2:ta suorittavalla palvelimella. Kun käytettiin NetDB2:n käyttöliittymää verkon välityksellä, TPC-H testit

osoittivat palvelun lähetysrangaistuksen olevan 28 % asteikolla 0,1, 8 % asteikolla 1 ja 1 % asteikolla 10. [15, s. 4]

Datan tietosuoja on yksi suurimmista huolenaiheista Internetin palveluissa. Datan tulisi olla suojattua sekä liikkuessaan että paikalla ollessaan. Paikallaan olevan datan suojausta on tutkittu vähemmän, mutta se on tärkeää kun tietokantaa käytetään palveluna. Datan suojausta täytyy kehittää siten, etteivät ulkopuoliset 1) pääse siihen käsiksi ja 2) jos pääsevät, data on heille käyttökelvotonta. Datan salaaminen kryptaamalla on hyvä keino käyttökelvottomaksi tekemiseksi ja se voidaan toteuttaa joko ohjelmisto- tai laitteistotasolla. Järjestelmän suorituskyky tosin kärsii kun dataa joudutaan jatkuvasti salaamaan ja avaamaan tarvittaessa. Näistä kahdesta salausvaihtoehdosta Hacigümüs et al. havaitsivat ohjelmallisen tason salauksen hidastavan suoritustehoa suuresti. [15, s. 5-6]

Tietokannan käyttö palveluna on vielä kehityksensä alkuvaiheessa ja sen kehittämiseen turvalliseksi seä virheettömäksi palveluksi kuluu vielä aikaa. Jos se saadaan toimivaksi ja turvalliseksi menetelmäksi, tulee siitä yksi mahdollinen tapa ratkaista useamman palvelimen tietokantojen yhteystyöhön liittyvät ongelmat, sillä jokainen palvelin pystyy ottamaan yhteyden samaan, verkossa sijaitsevaan tietokantaan samalla menetelmällä.

5 UUDEN TIETOKANTAJÄRJESTELMÄN KÄYTTÖÖNOTTO

Uuden tietokantajärjestelmän käyttöönottoon liittyy monta huomioon otettavaa asiaa. Mikäli keskitetystä tietokannasta siirrytään hajautettuun, tulee tietokannan synkronointi ja mahdollisten replikoiden luominen huomioida. Rinnakkaisessa käytössä myös lukkiutumisten hallinta on elintärkeää palvelun toiminnan kannalta ja pilvipalveluihin siirtymisestä voi koitua ongelmia varautumattomalle käyttäjälle. Uuden järjestelmän optimointia on myös hyvä pohtia, kuten myös järjestelmän käyttöönottoon liittyviä riskejä.

5.1 Hajautettuihin tietokantoihin siirtyminen

5.1.1 Tietokantojen synkronointi

Kenties yksi kaikkien tärkeimmistä teknisistä ongelmista hajautettuja tietokantoja käyttävissä järjestelmissä on tietokantojen synkronointi. Synkronointi on rinnakkaisuuden hallinnan laajennus keskitettyjen tietokantojen vaatimuksista. Se on mekanismi, joka ylläpitää datan loogista jatkuvuutta ympäristössä, jossa useampi lähde päivittää yhteistä dataa. Keskitetyissä järjestelmissä rinnakkaisuutta hallitaan lukitsemalla päivitykseen liittyvä data estääkseen datan epäjohdonmukaisuuksia. Kun tapahtuma yrittää lukea lukittua dataa, se yksinkertaisesti odottaa lukon aukeamista. [1, s. 219]

Hajautetussa järjestelmässä ongelma on paljon suurempi, sillä päivitykset voivat saapua usealta eri solmulta erilaisissa järjestyksissä. Päivitykset voivat lisäksi aiheuttaa globaalisti epäjohdonmukaisia tuloksia, vaikka paikallisesti data olisi johdonmukaista. Menetelmää, jolla hajautettujen tietokantojen johdonmukaisuutta ylläpidetään, kutsutaan synkronoinniksi. Kaikkien kannan päivitysten tulisi tapahtua synkronoinnin avulla rikkomatta datan johdonmukaisuutta. [1, s. 219]

Sekä partitioidut että replikoidut hajautetut tietokannat vaativat globaalia synkronointia. Partitioiduissa tapauksissa paikallinen jatkuvuuden hallinta voi vaikuttaa riittävältä, mutta todellisuudessa asia ei ole näin yksinkertainen. Esimerkiksi tapahtuma voi joutua lukemaan useita erilaisia arvoja, joista jokainen sijaitsee erillisellä solmulla. Jos näitä arvoja päivitettäisiin samalla hetkellä, tapahtuman tulokset olisivat epäjatkuvia. Synkronointi on välttämätön tapahtumien jatkuvuuden kannalta. Replikoiduissa tietokannoissa synkronoinnin tarve on vielä selvempi, sillä päivitykset voivat tapahtua samaan aikaan useammalla eri solmulla.[1, s. 220]

Hajautettujen tietokantojen päivitysprotokollat voidaan luokitella käytetyn hallintamenetelmän mukaan. Yksi protokollaluokka käyttää keskitettyä hallintaa muodossa, jossa kaikki päivityspyynnöt välitetään yhden hallintapisteen kautta. Tässä pisteessä pyynnöt myös validoidaan. Toinen luokka käyttää hajautettua hallintaa, missä vastuu päivityspyyntöjen validoinnista ja suorituksesta jaetaan järjestelmän solmujen kesken. [1, s. 221]

5.1.2 Tietokantojen replikointi

Horisontaalinen partitiointi on yleisin käytetty tapa skaalata tietokantoja ajettavaksi useammalle laitteelle. Asettamalla partitiot eri solmuille saadaan aikaiseksi lähes lineaarista nopeuden kasvua. Kasvu näkyy erityisesti analyyttisillä kyselyillä, joissa jokainen solmu voi tutkia omaa partitiotansa yhtäaikaisesti. Skaalautuvuuden lisäksi partitiointi voi parantaa saatavuutta takaamalla muiden partitioiden toimivuuden mikäli yksi partitio vioittuisikin. [16, s. 48]

Partitiointimenetelmiä on useita erilaisia, mutta yleisimmät niistä ovat kiertovuorottelu- (lähettämällä jokainen peräkkäinen rivi eri partitiolle), etäisyys- (jakamalla rivit predikaattijoukon mukaisesti) sekä hajautustaulu-partitiointi (aseta rivit osille käyttämällä hajautustaulua). Kaikki nämä menetelmät ovat tehokkaita analyyttisille kyselyille, jotka tutkivat suuria tietomääriä. Valitettavasti yksikään näistä ei toimi kovinkaan hyvin pienistä tapahtumista muutamassa tallenteessa koostuvista kuormista. Jos käsitellään useampaa kuin yhtä riviä, kiertovuorottelu- ja hajautustaulu-partitiointi vaativat usein pääsyä useammalle eri laitteelle. Hajautettujen kyselyjen suorittaminen vähentää suoritustehokkuutta verrattuna kyselyiden lokaaliin suoritukseen. Etäisyyspartitioinnilla on mahdollista saada aikaiseksi parempia tuloksia, mutta tämän menetelmän käyttö vaatii etäisyyksien tarkkaa määrittämistä, joka voi olla käsin toteutettuna vaikeaa. Partitiointiongelmasta tulee vielä hankalampi, mikäli kyselyt vaikuttavat useampaan tietokantatauluun. [16, s. 48]

Curino et al. kehittivät Schism-järjestelmää partitioinnissa ilmaantuvien ongelmien ratkaisemiseksi. Schism on ennennäkemätön datapohjainen partitiointijärjestelmä tapahtumien kuormitukselle esittäen tietokannan ja sen kuormituksen graafina. Tähän sovelletaan graafin partitiointialgoritmeja jotta löydettäisiin mahdollisimman tasapainoiset partitiot minimoiden leikkaavien reunojen painoarvon. Tämä painoarvo vastaa likimäärin useammalla laitteella suoritettavien tapahtumien minimointia. [16, s. 48-49]

5.2 Lukkiutumisten hallinta

Ongelmia syntyy, mikäli kaksi eri prosessia pyrkivät muokkaamaan samaa osaa tietokannan datasta samaan aikaan. Tämän seurauksena ensimmäisen prosessin tulee lukita data muokkauksen ajaksi. Kohdatessaan lukituksen muut prosessit odottavat sen aukeamista. Lukitus voidaan avata kun lukitusta koskeva prosessi on suoritettu loppuun. Tä-

män jälkeen muut prosessit pääsevät käsiksi näihin tietoihin. Ongelmia kuitenkin ilmenee, mikäli näitä lukituksia ei saada auki.

Lukkiutuneen tilanteen selvittämiseksi täytyy lukkiutuminen havaita ja siihen liittyvä tilanne ratkaista. Lukkiutumisen havainnointialgoritmi katsotaan oikeaksi, mikäli se täyttää kaksi ehtoa: 1) jokainen lukkiutuminen havaitaan lopulta sekä 2) jokainen havaittu lukkiutuminen on oikeasti olemassa, eli havainnoidaan vain aitoja lukkiutumistilanteita. [17, s. 80] On myös syytä huomata, että ratkaistut lukkiutumiset eivät enää ole ongelma, joten niitä ei tulisi edes havaita.

Kuten Krivokapić et al. mainitsevat [17, s. 79], maailmanlaajuinen tietokoneiden välinen yhteys aiheuttaa ongelmia datan luotettavuudelle. Tästä johtuen tarvitaan jatkuvasti laajempia, hajautettuja ratkaisuja. MARIPOSA prototyyppi on järjestelmä, jolla näitä vaatimuksia on tarkoitus täyttää. Tulevaisuudessa tällaiset järjestelmät voivat joutua ylläpitämään satojen tai jopa tuhansien sivustojen sekä miljoonien käyttäjien aiheuttamia vaatimuksia. [17, s. 79] On selvää, että tämänlaiset kasvut aiheuttavat haasteita nykyisten järjestelmien skaalautuvuuksille.

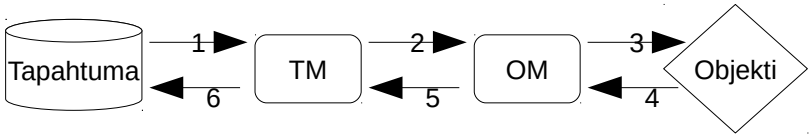
Lukkiutuminen voi tapahtua jokaisessa tietokantajärjestelmässä, joka jakaa resursseja useammalle laitteelle ja joka mahdollistaa näiden resurssien rinnakkaisen käytön. Pessimistinen määrittely eri tilanteissa on yleisimpiä syitä ilmiön tapahtumiselle, kuten esimerkiksi lukitus- tai avausprotokollissa. [17, s. 79] Tästä johtuen on ensiarvoisen tärkeää määritellä lukitukseen käytettävät protokollat mahdollisimman tarkoiksi ja huolehtia virhetilanteista mahdollisimman tehokkaasti.

Krivokapić et al. ehdottamassa prototyyppialgoritmissa [17, s. 79-80] tutkitaan lukkiutumista sekä niistä aiheutuvia tilanteita, päämääränä estää lukkiutumisten syntyminen. Mikäli lukkiutumisia kuitenkin havaitaan, pystytään niihin reagoimaan algoritmin avustuksella. Algoritmissa tapahtumien suorituksesta ja eteenpäin viennistä vastaa tapahtumien hallinta, kun taas objekteista vastaa objektien hallinta. Kumpiakin esiintyy järjestelmässä useita kappaleita. Tapahtumien hallinnat lähettävät toimintapyyntöjä objektien hallinnoille koskien tämän hallitsemia objekteja. Objektien hallinta lähettää objektien antamat vastaukset takaisin toimintapyynnön tehneelle tapahtumien hallinnalle. Yksinkertaisimmassa tilanteessa kumpikin hallinta hallitsee vastaavasti vain yhtä tapahtumaa ja objektia, jolloin voidaan käyttää tapahtumaa merkitsemään tapahtumien hallintaa, ja objektia tarkoittamaan objektien hallintaa. [17, s. 80]

Krivokapić et al.:n kehittämässä algoritmissa jokaisella objektilla ja tapahtumalla on yksilöllinen tunniste. Useat lukkiutumisen havainnointialgoritmit vaativat lukitusten ratkaisemiseksi tapahtumien täydellisen järjestyksen ja olettavat pystyvänsä käyttämään tapahtumien tunnisteita. Mikäli tapahtuma joudutaan keskeyttämään, täytyy sille antaa uusi tunniste uudelleenkäynnistyksen yhteydessä. Muuten järjestelmässä on samanaikaisesti usea tapahtuma samalla tunnisteella aiheuttaen mahdollisesti järjestelmän epäjohdonmukaisuutta. Tämä taas voi vaikuttaa tapahtumien suoritusjärjestykseen, esimerkiksi vanhimmasta tapahtumasta tulee suoritusjärjestyksessä nuorin, millä voi olla ennalta ar-

vaamattomia seurauksia. Välttääkseen tällaisia tilanteita algoritmiin käyttämiin tapahtumiin lisättiin aikaleima hetkestä, jolloin tapahtuma käynnistyi järjestelmässä. Tätä leimaa ei alusteta uudelleen, jolloin sitä voidaan käyttää tapahtumien järjestyksen selvittämiseen. [17, s. 80]

Tapahtumat ja objektit kommunikoivat viestien välityksillä asynkronisesti. Tapahtumat lähettävät viestejä niistä vastaavan tapahtuman hallinnan kautta objekteja vastaaville objektien hallinnalle. Nämä lähettävät objekteilta saamansa ilmoitukset ja mahdolliset tulokset takaisin tapahtumien hallinnoille, jotka palauttaa ne eteenpäin varsinaisille tapahtumille. Kommunikointi oletetaan virheettömäksi, jolloin jokainen lähetetty viesti saapuu määränpäähänsä muuttumattomana. [17, s.80] Tapahtumien kulkua algoritmissa on havainnollistettu kuvassa 16.



Kuva 16: Selventävä piirros tapahtuman käsittelystä Krivokapić et al.:n kehittämässä algoritmissa. [17, s. 80, käännetty suomeksi]

Oletetaan kaksivaiheinen lukitusprotokolla, jonka ensimmäisessä vaiheessa tapahtumien hallinnat pyytävät operaation suorituksia (mahdollisesti) eri objekteilta. Objektien hallinnat ylläpitävät syntyviä lukituksia odottavien tapahtumien puolesta. Toisessa vaiheessa tapahtumien hallinnat aloittavat sitoutumisprosessin, jonka jälkeen lukko aukaistaan. Objektien hallinnat ylläpitävät listaa kaikista tapahtumista, jotka haluavat suorittaa jotain sen ylläpitämälle objektille. Listaa aikataulutetaan pyyntöjen mukaisesti. Keräämiensä tietojen perusteella objektin hallinta pystyy päättämään, voidaanko haluttu toiminto päästää lukituksen valvonnassa olevalle alueelle. Mikäli pyyntöä ei sallita, operaation kutsua viivästytetään, jolloin tapahtuman toiminnallisuus on estetty lukkojen aukeamiseen asti. [17, s. 80] Tällä lukitusprotokollalla on mahdollista estää ainakin osa lukkiutumiseen johtavista oireista.

Krivokapić et al.:n algoritmi soveltaa semanttista lukitusta, joka on joustavampi kuin eksklusiivinen tai luku ja kirjoitus-lukitusmenetelmä. Eksklusiivisella lukitusmenetelmällä objektia pystyy käsittelemään vain yksi tapahtuma kerrallaan. Luku ja kirjoitus-lukitusmenetelmä puolestaan antaa useammalle lukijalle pääsyn kerrallaan, mutta kirjoittamiseen tarvitaan eksklusiivista lukitusta. Semanttinen lukitusmalli käyttää hyväksi operaatioiden semanttisuutta lisätäkseen mahdollisien rinnakkaisten suoritusten määrää. Tällöin useammat dataa päivittämässä olevat tapahtumat voivat toimia lukitusalueella samanaikaisesti. [17, s. 81]

Lukkiutumisen ratkaisustrategiat määrittävät, mitkä tapahtumat hylätään lukkiutumisen ratkaisemiseksi. Krivokapić et al.:n tekemistä simulaatioista selvisi kaksi ehtoa, jotka jokaisen ratkaisustrategian tulisi taata: 1) takuu ainakin yhden toiminnon valmistumi-

sesta järjestelmässä, sekä 2) yhtään tapahtumaa ei hylätä (tai uudelleenkäynnistetä) loputtomasti. [17, s. 82]

Yksi mahdollinen ratkaisu hylättävän tapahtuman löytämiseen on suoritusajan päättyminen, jossa tapahtuma asettaa itselleen viimeisen mahdollisen suoritushetken. Mikäli se ei pääse suorittamaan toimintojaan haluttuun aikaan mennessä, tapahtuma katsoo olevansa lukkiutuneessa tilassa ja hylkää toimintansa. Tämänlainen algoritmi on yksinkertainen toteuttaa, eikä se aiheuta verkolle suuria rasituksia. Samalla se kuitenkin hylkää aivan liian monta tapahtumaa sekä mahdollisesti myös sellaisiakin, jotka eivät oikeasti ole lukkiutuneet vaan odottavat vuoroaan suoritukseen. Lisäksi suoritusaika täytyy virittää yksilöllisesti jokaisen tapahtuman mukaan. [17, s. 82]

Luotausalgoritmit toimivat myös ratkaisuna lukkiutumistilanteisiin. Nämä eivät toimi suoranaisella tavalla, vaan erityisten viestien, *luotainten*, kautta. Luotausalgoritmit voidaan jakaa kahteen tyyppiin, edge-chasin- sekä diffusing computation-tyyppeihin. Edge-chasing-tyyppi perustuu luotainten lähettämiseen aina, kun jokin tapahtuma pysähtyy odottamaan lukituksen aukeamista reitillään. Luotain lähetetään jokaiselle odottavalle tapahtumalle, jotka aloittavat laskennan. Luotaimen saaneen tapahtuman tulee lähettää se eteenpäin muille lukkoa odottaville tapahtumille. Mikäli luotain palaa alkuperäiselle tapahtumalle, täytyy tilanteen olla lukkiutunut ja voidaan ryhtyä toimiin sen aukaisemiseksi. Diffusing computationissa sen sijaan tapahtuma aloittaa diffuusiolaskennan aina, kun sen joutuu odottamaan lukon aukeamista. Laskennan päättyminen on merkki tilanteen lukkiutumisesta. [17, s. 84]

Hajautetut tietokantajärjestelmät tuovat lukkiutumiseen vielä omat ongelmansa mukaan. Tieto voi olla vanhaa tai epäsäännöllistä, mikä johtaa olemattomien lukkiutumisten havainnointiin. Tämän välttämiseksi globaalien lukkiutumisten havainnointialgoritmit pyrkivät etsimään lukkiutumisia kokonaiskuvasta. [17, s. 84]

Krivokapić et al.:n algoritmi käyttää apunaan lukkiutumisen havainnointitoimijoita. Globaalin tilan informaatio jaetaan näille lukkiutumisten havainnointitoimijoille siten että vain yhdellä niistä on järjestelmän kokonaiskuva yhteen tapahtumaan liittyen. Näin ollen samaa lukkiutumista ei havaita kahdesti. Kun järjestelmään lisätään komponentteja ja tapahtumia, lisätään vastaava määrä myös lukkiutumisten havainnoititoimijoita. [17, s.85]

5.3 Suorituksen mallintamista

Brumber et al. kehittivät simulaatiomallinnuksen, jonka tarkoitus oli arvioida erilaisten laitteistojen ja kuormitusten intensiteettien vaikutusta hajautetun olio -periaatteen tietokantajärjestelmään. Simuloidakseen suuria konfiguraatioita mielenkiintoisella ajanjaksolla he rajoittivat pienten yksityiskohtien määrää, jonka seurauksena järjestelmän solmuja käsitellään yksinkertaisesti olioviitteiden lähteinä ja kohteina. Malli jaettiin kah-

teen osaan, konfiguraatiomalliin, sisältäen laitteiston ja tietokannat, sekä työtaakkamalliin, sisältäen järjestelmän resursseja käyttävän laskennan ja hallinnan). [18, s. 22]

Järjestelmän konfiguraatiomalli koostuu lähiverkostolla toisiinsa liitetyistä solmuista. Solmut puolestaan koostuvat laitteistoresursseista, kuten prosessoreista ja levyistä, sekä prosesseista, jotka mallintavat käynnissä olevia aktiviteetteja solmussa. Palvelimeksi määritelty solmu voi myös pitää sisällään verkkoyhteyksille tarvittavaa tietoa etäsolmuista. Solmut on suunniteltu tarpeeksi mukautuviksi, jotta niillä voidaan mallintaa erilaisia järjestelmän osia. Yksinkertaisimmillaan solmut eivät tarvitse kuin yhden prosessorin. [18, s. 23]

Järjestelmän toinen osa, kuormitusmalli, sisältää kaikki järjestelmän suoritettavat tehtävät. Näiden tehtävien katsotaan olevan peräisin järjestelmän käyttäjiltä. Simulaatiossa prosessit mallintavat käyttäjien käyttäytymistä ja jokainen käyttäjäprosessi on liitetty solmuun. Solmulla olevien käyttäjien määrää ei ole rajattu. Käyttäjät vaihtelevat tilaansa odottamisen ja järjestelmän suoritettavien prosessien luomisen välillä. Käyttäjien mahdollisia toimintoja on neljä erilaista: paikallisia viitteitä, globaaleja viitteitä, sisään- ja uloskirjautumistoimintoja sekä suorituksia. Käyttäjä voi suorittaa useita toimintoja yhden toimintatilan aikana. [18, s. 23]

Hyvin suunniteltu hajautettu järjestelmä mahdollistaa toimintojen suorituksen yhtäaikaisesti aina kun tämä on mahdollista. Tästä seuraa suurempi laitteiden hyötykäyttö ja toimintojen suoritusmäärä. Joissain järjestelmissä yhtäaikainen suoritus on kuitenkin rajattu, esimerkiksi objekteja tulee hakea ja täytyy tarkastaa, mitä muita tarvitaan operaation loppuun suorittamiseen. Brumber et al.:n mallinnus ottaa huomioon kummankin tyylisen rinnakkaisen suorituksen [18, s. 23]

Tätä mallinnusta sovellettiin hajautettuun, olio-tyyliseen tietokantajärjestelmä ORION-2een kahdella tavalla. Ensimmäisessä, hajautetussa mallissa, kaikki solmut toimivat palvelimina. Toisessa, palvelinmallissa, vain yhden solmun sallitaan toimivan palvelimena. Mallilla tehtiin kaksi erillistä koetta, joista toisessa arvioitiin järjestelmän suorituskykyä erilaisilla silmujen määrillä, ja toisessa arvioitiin suorituskykyä globaalien kyselyiden vaihtelevalla pyyntötahdilla. Kumpaakin suoritusta jatkettiin 600 sekunnin ajan. [18, s. 27-28]

Kokeiden tulosten perusteella järjestelmässä olevien solmujen määrä hidastaa kumpaakin mallia. Pienemmillä määrillä hidastus on hyvin samankaltaista, mutta suuremmilla, yli 50 solmun määrillä, hajautetun mallin suoritusnopeus laskee merkittävästi. Tämä hidastuminen on selitettävissä verkon käytön määrällä. 50 solmulla verkkoa käytetään jo 81%, kun taas käyttö nousee 100% 75 solmun kohdalla. Tästä seuraa myös kyselyiden odotuksen lisääntyminen, sillä keskiarvoisesti 50:llä silmulla odottaa 76 viestiä, kun taas 75 silmulla viestien määrä nousee 1000:en. Palvelinmallissa taas verkon käyttö ei koskaan noussut 10% yläpuolelle ja viestejä oli silmulla jonossa vain yksi kerrallaan. [18, s. 28-29]

Globaalien kyselyiden määrää testattiin vain hajautetulla mallilla. Kuten odotettua, tälläkin oli vaikutusta järjestelmän suoritustehoon: mitä tiheämmin globaaleja kyselyjä ilmaantui, sitä hitaammin järjestelmä toimi. Solmujen määrällä pystyttiin kuitenkin pienentämään tätä globaaleista kyselyistä aiheutuvaa hidastumista. [18, s. 31-32]

Tästä mallinnuksesta voidaan päätellä, että solmujen eli toisiinsa yhteydessä olevien laitteiden lisäämisellä voi olla suuria seurauksia järjestelmän suoritusnopeuteen. Liian paljon yhteen liitettyjä laitteita ja kyselyt voivat hidastua liian paljon kun vastauksia etsitään useammasta laitteesta. Liian vähän solmuja, ja globaalit kyselyt voivat helposti hidastaa koko järjestelmän toiminnan. Laitteiden määrä tulisi arvioida jokaiseen tilanteeseen uudelleen.

5.4 Hajautetun järjestelmän optimointia

Optimointistrategiat globaaleille kyselyille ovat riippuvaisia eri laitteiden välisen verkon yhteysnopeudesta sekä kirjoitusnopeudesta. Erityisesti yhteysnopeus vaikutti ennen viestien välitykseen niin paljon, että optimoinnin kannalta paikallisen prosessoinnin voitiin katsoa olevan lähes ilmaista. Nykyään verkon siirtonopeudet ovat kuitenkin kehittyneet niin suuriksi, ettei tämä oletus voi enää pitää paikkaansa. Nopeuksien optimointitärkeydet ovat pikemminkin päinvastaiset, sillä siirtonopeus voi olla paljon suurempi kuin paikallisen prosessoinnin vaatima. Toisaalta, jos laitteiden väliset suhteet ovat pirstoutuneita ja kantojen välisiä liitoksia ja puoliliitoksia suoritetaan, tulee datan siirtokustannukset ottaa huomioon. [19, s. 12]

Wiggins tutki kahta erilaista menetelmää optimoida kyselyt hajautettujen tietokannanhallintajärjestelmien välillä. Ensimmäisessä menetelmässä käytettiin tietokantojen puoliliitoksia, joilla voidaan vähentää kyselyyn kuuluvia suhteita ja pirstoutumista. Toisessa menetelmässä toistettiin tehtävä kysely kaikille kyselyyn osallisena olevilla laitteilla. Optimointitekniikassa huomioidaan lähinnä paikalliset verkostot, joissa siirtonopeus on vähintään 10 Mbit/s. Puoliliitosstrategian seurauksena laitteiden välinen verkkoliikenne kasvaa, mutta vähentyneet suhteet laskevat kuitenkin paikallisen prosessoinnin määrää. Toistamisstrategiassa taas vähennetään laitteiden välistä datan siirtoa, mutta paikallinen prosessointi kasvaa. [19, s. 12-13]

Vaikka Wiggins ei mainitse tässä dokumentissaan strategioiden yksityiskohtia, voidaan näistä tehdä päätelmiä. Mikäli tiedonsiirto on rajoitettua laitteiden välillä, voidaan toistaminen valita optimoidummaksi ratkaisuksi. Mutta jos tiedonsiirto voidaan taata nopeaksi ja virheettömäksi, mutta paikallinen prosessointi ei ole kovin tehokasta, kannattaa päätyä puoliliitosten suunnitteluun.

5.5 Pilvipalveluihin siirtyminen

Kehittäessään järjestelmiä ohjelmoijat eivät käytä pelkästään ohjelmointikieliä ja kirjastoja, vaan myös muita monenlaisia työkaluja kuten virheenhallintaa ja profilointia. Pu-

huttaessa palvelun siirtämisestä pilveen jätetään yleensä huomiotta useita tärkeitä yksityiskohtia tehtäessä yritysluokan palveluita. Tällaisia ovat

- monitorointi
- jäljitys
- hajautettu lukkiutuminen
- virheiden hallinta
- replikointi
- suunnitelmat virheiden varalle.

Monet näistä voidaan kuitenkin osoittaa pilvipalvelun tarjoamiksi objekteiksi. Hajautetun järjestelmän testauksen, ymmärtämisen ja valvonnan ylläpito ei ole helppo tehtävä, mutta se on välttämätön ehto, jos ohjelmoijat haluavat menestyä hajautettujen järjestelmien maailmassa. [6, s. 8]

Laitetilatiedot ja palvelun tehokas suoritusaika tulisi myös huomioida pieniä järjestelmiä isommiksi yhdistettäessä. Jopa yksittäisessä Windows-laitteessa moniajon miettiminen on välttämätöntä, sillä esimerkiksi käyttöliittymän päivittämiseksi tulee toimia käyttöliittymän omalla säikeellä. Käytännöllisen alustan hajautettujen järjestelmien rakentamiseen täytyy tarjota abstraktioita, jotka tekevät rajojen paikallistamisesta ja asettamisesta täsmällisiä, ja tarjoavat kenties myös paikallista atomisuutta sekä takuuta luotettavuudesta. Yhden laitteen kohdalla ohjelmoijat ja tietokantojen mallintajat usein ylenkatsovat tällaisia ongelmia, mutta hajautetussa maailmassa näiden ongelmien löytymisestä tulee oleellinen ja tärkeä tehtävä sekä ohjelmoijille että tietokantojen mallintajille. [6, s. 8]

5.6 Uuden järjestelmän käyttöönottoon liittyviä riskejä

Uusiin menetelmiin ja järjestelmiin käyttöönotossa on aina omat riskinsä. Näihin riskeihin olisi hyvä varautua ennakkoon ja mieluiten hyvissä ajoin. Tietokannan vaihtaminen aiheuttaa myös riskialttiita tilanteita.

5.6.1 Vaihtoprosessi

Vaihtoprosessiin on hyvä varata aikaa, sillä uuden järjestelmän käyttö ja ylläpito voivat poiketa vanhasta melkoisen paljon. Prosessi tulisi suunnitella alusta asti mahdollisimman huolellisesti. Erityisesti palvelun loppukäyttäjät tulisi huomioida vaihtoa tehtäessä sekä minimoida näille aiheutuva vaihdosta johtuva haitta. Pienillä kokeilla on hyvä aloittaa, esimerkiksi vaihto voidaan suorittaa ensin lokaalisti ja tarkistaa, että uusi järjestelmä toimii halutulla tavalla. Usean palvelimen toimintaa voidaan simuloida yhdistämällä joukko paikallisia laitteita samaan verkkoon ja kokeilemalla näiden välistä kommunikointia.

Palvelun toiminta ei saisi katketa pitkäksi aikaa järjestelmän vaihdon yhteydessä. Näin ollen toiminnan jatkuvuuden tulisi olla yksi tärkeimpiä prioriteetteja järjestelmää

uudistettaessa. Erillinen palvelin uuden järjestelmän kokeilemiseen auttaa vaihdon toimivuuden tarkastamisessa, mutta tärkeämpää on luoda palautuspiste ennen uuden järjestelmän käyttöön ottamista. Palautuspisteen avulla palvelu voidaan palauttaa toimivaksi vanhalla järjestelmällä, mikäli uudessa ilmenee odottamattomia ongelmia tai vikoja. Järjestelmän vaihtamista voidaan yrittää uudelleen kun esiintyvät ongelmat on selvitetty.

5.6.2 Tietokannan vaihtaminen

Tietokannan käyttöliittymän ja komentojen muuttuminen aiheuttavat riskitekijöitä. Entiset käyttökomennot eivät toimi enää ja ne tulee korvata uusilla, mikä voi olla suurikin prosessi. Prosessi tulisi automatisoida jollain keinolla, sillä automatisointi vähentää inhimillisten virheiden määrää.

Tietokannan datarakenne voi myös olla selvästi erilainen kuin aiemmin. Esimerkiksi relaatiokantatyyppinen MySQL säilöö datansa hyvin erilaisella tavalla kuin dokumenttipohjainen MongoDB. Tästä voi seurata datan muokkaamista, mistä voi mahdollisesti aiheutua useita ongelmia, kuten vääränlaisen datan sijoittamista väärään kohteeseen. Tietokannan vaihtoon voidaan varautua huomioimalla uuden tietokannan asettamat vaatimukset datan suhteen tai vaihtoehtoisesti valitsemalla uudeksi tietokannaksi mahdollisimman samankaltainen järjestelmä kuin entinen, jotta vaihto sujuisi mahdollisimman turvallisesti.

Hajautetut tietokannat tekevät vaihdosta vielä mutkikkaampaa. Ennen ainoastaan yhdelle järjestelmälle tietojaan tarjoavan tietokannan tilalle tulee uusi järjestelmä, joka esittää dataansa useammalle laitteelle samanaikaisesti ja samanlaisena. Ongelmia syntyy kun useammalla palvelimella yritetään muuttaa samaa dataa samanaikaisesti.

Toinen ongelmia aiheuttava tekijä hajautetuissa tietokannoissa on lisääntynyt verkkohyökkäysten kohteeksi joutumisen riski. Koska tietokantaan saadaan yhteys kahdelta tai useammalta palvelimelta, pääsevät verkkohyökkäyksen tekijät käsiksi samaan dataan ottamalla kohteekseen heikoimmin turvatun palvelimen. Palvelinten tietoturvallisuus ja verkkohyökkäyksiin varautuminen tulisi näin ollen hoitaa kummallakin palvelimella tehokkaasti. Datan suojaaminen ja käyttäjien yksityisyyden varjeleminen ulkopuolisten käsiltä ovat korkeimman prioriteetin arvoisia tietoturvatekijöitä.

5.6.3 Pilveen siirtyminen ja useamman palvelimen käyttöönotto

Pilvipalveluissa esiintyy myös erilaisia riskejä. Jatkuva yhteys Internetiin asettaa palvelun alttiimmaksi verkkohyökkäyksille, kuten esimerkiksi palvelunestohyökkäykselle. Näihin hyökkäyksiin pystytään parhaiten varautumaan valvomalla palvelimella tapahtuvaa liikennettä ja estämällä niiden palvelua käyttävien laitteiden IP-osoitteet, joilta toistuvasti tulee epämääräisiä, palveluun liittymättömiä pyyntöjä. Tietokantakyselyt on myös syytä suojata mahdollisten tietokantainjektiohyökkäysten varalta siten, ettei palvelun tietokantoja pysty kukaan kehittäjien ulkopuolinen henkilö muokkaamaan.

Jatkuva Internet-yhteyden vaatiminen voi myös rajoittaa palvelun käyttömahdollisuuksia joillekin käyttäjille. Syynä yhteysvaatimukselle voi olla esimerkiksi palvelimelta ladattava data, jota tarvitaan järjestelmän toimintaan. Vaikka nykymaailmassa Internet on saatavilla lähes kaikkialla, langattomasti tai langallisesti, on myös olemassa alueita, joissa Internet ei ole saatavilla tai sen käyttö on rajoitettua. Tällaisilla alueilla olevat henkilöt jäävät pois järjestelmän mahdollisista käyttäjistä. Tätä ongelmaa pystytään lieventämään tekemällä palvelusta erillinen offline-versio, joka toimii Internetin kuuluvuusalueen ulkopuolella.

Pilvipalvelun päivitys sujuu yleensä saumattomasti ja helposti. Vain tiedoston siirto ja käyttäjät pääsevät käsiksi uusimpaan kehitettyyn versioon. Kaikki käyttäjät eivät tosin välttämättä pidä uudesta versiosta. Jokin entinen palvelun ominaisuus tai toiminto on saattanut poistua tai muuttua, käyttäjä on tottunut käyttämään vanhaa versiota, uuden palvelun ulkoasu ei miellytä tms. Näihin tilanteisiin ei voida varautua kunnolla muilla keinolla kuin kuuntelemalla järjestelmän käyttäjien kommentteja ja ottamalla heidät huomioon palvelua kehitettäessä. Palautteen avulla palvelua pystytään kehittämään suuntaan joka miellyttää kaikkia palveluun liittyviä henkilöitä, niin käyttäjiä kuin ohjelmoijiakin.

6 JOHTOPÄÄTÖKSET

Useammalle palvelimelle jaetun palvelun synkronointi ja ylläpito ei ole yksinkertainen toimenpide. Tavallisella, keskitetyllä tietokannalla dataa ei mitenkään saada pysymään jatkuvasti synkronoituna kaikilla järjestelmän palvelimilla. Mitä enemmän dataa muutetaan tai lisätään tietokantaan, sitä vaikeammaksi kantojen pitäminen synkronoituina tulee. Mikäli tietokantojen synkronoinnista halutaan tehdä mahdollisimman tehokas ja käyttäjäystävällinen sekä minimoida synkronoinnista aiheutuvat virhetilanteet ja ylikirjoitukset, täytyy keskitettyjen tietokantojen tilalle löytää toinen vaihtoehto. Jokin menetelmä, jolla tietokannat pystytään pitämään ajan tasalla järkevässä ajassa ja mahdollisimman tehokkaasti.

Yksi keino ratkaista nämä edellä mainitut ongelmat on ottaa käyttöön jokin hajautettu tietokanta, kuten Google Bigtable tai Amazon Aurora. Hajautetulla tietokannalla palvelun käyttämä data saadaan pidettyä järkevästi synkronoituna palvelinten välillä. Suurin ongelma luultavasti tulee hajautettuun tietokantaan siirtymisessä, mihin kuuluu uuden kannan menetelmien ja toimintojen opetteleminen sekä järjestelmän asennus ja käyttöönotto nykyisellä palvelulla. Palvelinten tietokantojen välille täytyy myös muodostaa synkronoinnin vaatimat yhteydet, mikä vie tilanteesta riippuen myös oman aikansa ja työpanoksensa.

Ennen kuin hajautetun tietokannan käyttöön voidaan siirtyä, täytyy tehdä valinta mahdollisista järjestelmistä. Suurimmat hajautetun tietokannan valintaan vaikuttavat tekijät ovat tietokannan ominaisuudet ja saatavuus. Mikäli järjestelmässä on jo käytössä keskitetty tietokanta, kannattaa uusi, hajautettu kanta valita mahdollisimman samankaltaiseksi. Tällöin siirtymisvaiheeseen kuluva aika saadaan minimoitua. Jokaisessa hajautetussa tietokantajärjestelmässä on tietenkin omat hyvät ja huonot puolensa ja toiset järjestelmät saattavat soveltua tiettyihin palveluihin paremmin kuin toiset. Tärkeä huomioitava tekijä on myös järjestelmän valmiusaste. Esimerkiksi Google Bigtablen käyttäjille tarjottu versio on tämän työn kirjoitushetkellä hetkellä vielä beta-vaiheessa, joten sen käyttäminen ei välttämättä ole aivan ongelmatonta, kun taas Amazon Auroraa tarjotaan täysin toimivana versiona.

Toinen vaihtoehto olisi käyttää jotain tietokantapalvelua. Tämän palvelun avulla pystyttäisiin ottamaan yhteys useammalla palvelimella samalle tietokannalle, jolloin synkronointiongelma häviäisi kokonaan. Tähän menetelmään liittyy tosin useampia ongelmia, joista suurimpana ongelmana on kuitenkin tämän teknologian kehitysaste, joka on vielä vasta alkuvaiheessa ja palveluja ei ole toiminnassa muita kuin kokeellisia tarkoi-

tuksia varten. Lisäksi palvelun tietoturvallisuus tulisi olla kunnossa ennen palvelun varsinaista käyttöönottoa.

Uuden järjestelmän käyttöönottoon liittyvät ongelmat ja menetelmät tulee myös huomioida uudistettaessa järjestelmiä. Tietokantojen tarvittavat synkronointi- ja replikointimenetelmät on hyvä hallita ennen varsinaista järjestelmän käyttöönottoa. Järjestelmän toimivuutta voi kokeilla pienemmillä testeillä ennen varsinaista uuden järjestelmän käyttöönottoa. Yksittäisten laitteiden muodostamassa paikallisessa verkossa on hyvä kokeilla järjestelmän vaadittavien komponenttien kommunikointia ja yhteentoimivuutta. Myös tarvittavien tietokantareplikoiden toteutus ja käyttö on hyvä suunnitella ennen käyttöönottoa.

Palvelun uudistamisessa on omat riskinsä, jotka kehittäjien tulee tiedostaa ja huomioida ennen uudistamisen toteuttamista. Riskien arvo kasvaa etenkin silloin, kun järjestelmässä vaihdetaan kriittisiä osia, kuten keskitetyn tietokannan vaihto hajautettuun. Valmistautumisella ja tekemällä uudistukset pienissä osissa riskien toteutumisen mahdollisuutta voidaan pienentää.

Vaikka useammalle palvelimelle jaetun palvelun synkronointi ja ylläpito on hyvin hankala ongelma, se ei kuitenkaan ole mahdoton toteuttaa. Huolellisella valmistautumisella sekä oikeiden menetelmien ja järjestelmien valitsemisella palvelinten välinen synkronointi voidaan toteuttaa ongelmitta.

LÄHTEET

[1] Champine, George A. et al., Distributed Computer Systems: Impact on Management, Design and Analysis, 380 s., 1980, North-Hollan Publishing Company, Amserdam, New York, Oxford, viitattu 10.10.2015

[2] DB-Engines Ranking, viitattu 5.8.2015, http://db-engines.com/en/ranking

[3] Heimeriks.net, viitattu 5.8.2015, saatavilla: http://heimeriks.net/wp-content/uploads/2011/10/ISI-Relational-database.gif

[4] Gigaom.com, viitattu 5.8.2015, saatavilla: https://gigaom.com/wp-content/uploads/sites/1/2011/07/unql-1.jpg?quality=80&strip=all

[5] Shriver, B., Gentile, M., ACM Turing Award goes to Pioneer in Database Systems Architecture, NY, 25.3.2015, viitattu 5.8.2015, saatavilla: http://www.acm.org/press-room/news-releases/2015/pdfs/turing-award-14b.pdf

[6] Meijer, E., All Your Database Are Belong to Us, ss. 1-10, Databases, Volume 10, Issue 7, 23.7.2012, viitattu 14.8.2015, saatavilla https://queue.acm.org/detail.cfm?id=2338507

[7] Tutorial: Installing a LAMP Web Server on Amazon Linux, viitattu 14.5.2015, saatavilla http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/install-LAMP.html

[8] Online Censorship in China, viitattu 12.11.2015, saatavilla: https://en.greatfire.org/

[9] Jiang, M., Internet Companies in China: Dancing between the Party Line and the Bottom Line, Ifri, Center for Asian Studies, January 2012, viitattu: 12.11.2015, saatavilla: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1998976

[10] The On-Line Encyclopedia of Integer Sequences, viitattu 28.8.2015, http://oeis.org/

[11] Adiba, M. et al., Polypheme: An Experience in Distributed Database System Design and Implementation, ss. 67 - 84
Distributed Data Bases, 367 p., 1980, edited by C. Delobel, W. Litwin, viitattu 19.6.2015

[12] Chang F. et al, Bigtable: A Distributed Storage System for Structured Data, 14 s., OSDI 2006, viitattu 3.9.2015, saatavilla http://static.googleusercontent.com/media/research.google.com/fi//archive/bigtable-osdi06.pdf

[13] Amazon Aurora, viitattu 17.9.2015, saatavilla https://aws.amazon.com/aurora, http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Aurora.html

[14] Druschel, P. Narrowing the Semantic Gap In Distributed Programming, s.86, Communications of the ACM, Vol.52, No.11, November 2009, Viitattu 10.8.2015, saatavilla cacm.acm.org/magazines/2009/11/48436-technical-perspective-narrowing-the-semantic-gap-in-distributed-programming/fulltext

[15] Hacigümüs, H. et al, Providing Database as a Service, 10s., Proceedings of the 18th International Conference on Data Engineering (ICDE'02), IEEE, 2002, viitattu 17.9.2015, saatavilla: http://ieeexplore.ieee.org/

[16] Curino, C. et al, Schism: a Workload-Driven Approach to Database Replication and Partitioning, ss. 48 – 57, Proceedings of the VLDB Endowment, Volume 3, Number 1, 2010, viitattu 27.8.2015, saatavilla http://db.csail.mit.edu/pubs/schism-vldb2010.pdf

[17] Krivokapić, N. Et al., Deadlock detection in distributed database systems: a new algorithm and a comparative performance analysis, ss. 79 – 100, The VLDB Journal – The International Journal on Very Large Data Bases, Volume 8, Issue 2, lokakuu 1999, viitattu 21.7.2015 saatavilla: http://dl.acm.org/citation.cfm?id=765509.765510&coll=DL&dl=ACM&CFID=652119671&CFTOKEN=58021783

[18] Brumfield, J. A. et al. Performance modeling of distributed object-oriented database systems, ss.22-32, Proceeding DPDS '88 Proceedings of the first international sympsium on Databases in parallel and distributed systems, IEEE Computer Society Press Los Alamitos, CA, USA, 1988, viitattu 6.8.2015, saatavilla: http://dl.acm.org/citation.cfm?id=62597.62602&coll=DL&dl=ACM

[19] Wiggins, R., A comparison of two techniques for generating near optimal query strategies in a distributed relational database management system, ss. 12 – 13, Proceeding ACM '85 Proceedings addendum of the 1985 ACM annual conference on The range of computing: mid-80s perspective, ACM New York, NY, USA, 1985, viitattu 21.7.2015, saatavilla: http://dl.acm.org/citation.cfm?id=324409.324414&coll=DL&dl=ACM