hyväksymispäivä arvosana

arvostelija

NoSQL-tietokannat

Joel Sandborg

Helsinki 27.3.2012

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta – Fakultet –Faculty	Laitos – Institution – Department
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen laitos

Tekijä – Författare – Author

Joel Sandborg

Työn nimi – Arbetets titel – Title

NoSQL-tietokannat

Oppiaine – Läroämne – Subject

Tietojenkäsittelytiede

Työn laji – Arbetets art – Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages

Tiivistelmä – Referat – Abstract

Avainsanat – Nyckelord – Keywords

Säilytyspaikka – Förvaringställe – Where deposited

Muita tietoja – Övriga uppgifter – Additional information

Sisältö

1 Johdanto

Tietojenkäsittelyn alalla on viimeaikoina kohistu paljon NoSQL-tietokantajärjestelmistä. Näistä järjestelmistä on kirjoitettu runsaasti alan lehdistössä sekä Internetin blogikirjoituksissa. NoSQL-tietokantajärjestelmistä on julkaistu artikkeleita esimerkiksi ACM SIGMOD Record, Communications of the ACM, Computer ja Linux-journal lehdissä [Cat10, Sto10, Lea10, Ler10]. Aiheen tiimoilta on myös järjestetty konferensseja eri puolilla maailmaa, kuten esimerkiksi NoSQL Now! Yhdysvalloissa [NOS11] ja SDEC 2011 Etelä-Koreassa [SDE11]. Kyseiseen paradigmaan tai lähestymistapaan pohjautuvia ratkaisuja ovat ottaneet 2000-luvun aikana käyttöön esimerkiksi Internetissä toimivat suuret yritykset, kuten Google, Facebook ja Amazon. Näiden yritysten tarjoamat palvelut, kuten Google hakukone [Goo12c], Facebook [Fac12] ja Amazonin verkkokauppa [Ama12] ovat miljoonien ihmisten käytettävissä ympäri maailmaa. Tämän kaltaiset palvelut edellyttävät valtavien hajautettujen tietomäärien käsittelyä ja varastoimista. Tiedon pitää olla hyvin saatavilla. Tietokantajärjestelmältä edellytetään myös hyvää suorituskykyä, vaikka tietoon kohdistuvien operaatioiden määrä vaihtelisi. Suorituskyvyn ylläpitämiseksi järjestelmän täytyy skaalautua niin, että tarpeen vaatiessa järjestelmään voidaan lisätä enemmän resursseja. Lisäksi tietokannan rakenteen tulee olla joustava ja helposti muokattavissa.

Perinteiset relaatiotietokantajärjestelmät eivät enää ole riittäneet vastaamaan edellä mainittujen kaltaisten sovellusten tarpeisiin. Näille perinteisille tietokantajärjestelmille onkin viime vuosina kehitetty muita vaihtoehtoja. Näitä järjestelmiä on alettu kutsua ns. NoSQL-tietokantajärjestelmiksi. NoSQL-tietokantajärjestelmät käyttävät usein jotain relaatiomallia yksinkertaisempaa tietomallia, kuten avain-arvo-mallia [Cat10]. Sen lisäksi näiden järjestelmien käyttämä tietokannan skeema on usein hyvin joustava, tai tietokanta saattaa olla jopa kokonaan skeematon. NoSQL-tietokantajärjestelmien rakenne onkin usein niin yksinkertainen, että tietojen analysointiin ei tarvita kaikkia perinteisen SQL-kielen hienouksia.

NoSQL-tietokantajärjestelmien ominaispiirteisiin kuuluu myös, että ne eivät noudata kaikkia transaktiokäytäntöjen ACID-ominaisuuksia [SKS11, s. 866]. Näistä periaatteista luopumalla tähdätään nimenomaan parempaan skaalautuvuuteen, saatavuuteen ja suorituskykyyn. Näiden tavoitteiden saavuttamiseksi NoSQL-tietokantajärjestelmät ovat jou-

tuneet tekemään toimintaansa vaikuttavia kompromisseja. Kyseisten järjestelmien transaktio-, taululiitos- ja indeksointikäytännöt ovat usein rajoittuneita verrattuna perinteisiin relaatiotietokantajärjestelmiin.

Google [CDG06, CDG08] ja Amazon [DHJ07] julkaisivat 2000-luvulla artikkelit niiden omien palvelujensa perustaksi kehittämistä tietokantajärjestelmistä sekä niiden arkkitehtuuriratkaisuista. Näiden suurten Internet-yritysten esiteltyä omat NoSQL-tietokantajärjestelmänsä, uusia NoSQL-tietokantajärjestelmiä on kehitetty ja julkaistu yhä enemmän esimerkiksi avoimeen lähdekoodiin perustuvina –projekteina [Tiw11]. Kehitteillä on myös uusia kyselykieliä. Nämä uudet tietokantajärjestelmät ovat herättäneet paljon kiinnostusta sovelluskehittäjien sekä eri kokoisten yritysten keskuudessa. Uudenlaisten tietokantajärjestelmien tarjonnan lisääntyessä voi kuitenkin olla vaikeaa hahmottaa niiden käyttötarkoitusta ja soveltuvuutta. Esille saattaa nousta kysymys miten sovellukset ja yritystoiminta voivat hyötyä NoSQL-paradigman soveltamisesta, vai soveltuvatko nämä järjestelmät kenties vain suurten Internet-yhtiöiden tarpeisiin? Kuten jo edellä todettiin, NoSQL-tietokantajärjestelmät poikkeavat keskeisiltä ominaisuuksiltaan relaationtietokannoista, sen lisäksi nämä järjestelmät saattavat olla myös keskenään hyvin erilaisia. Tiettyjen yhteisten ominaispiirteiden lisäksi näiden uusien järjestelmien ominaisuudet voivat poiketa toisistaan suuresti.

Tässä tutkielmassa pyritään selventämään NoSQL-tietokantajärjestelmien tallennusratkaisun ja tiedon käsittelyn periaatteita, eroja relaatiotietokantoihin sekä millaiseen käyttöön nämä tietokantajärjestelmät oikein soveltuvat. Tutkielmassa esitellään myös joitakin NoSQL-tietokantajärjestelmien suosituimpia tietomalleja ja yksittäisiä järjestelmiä sekä vertaillaan niiden ominaisuuksia. Tutkielmasta rajataan pois sellaiset relaatiomallista poikkeavat tietokantajärjestelmät, joita ei oltu alun perin kehitetty laajan mittakaavan Internet-sovellusten käyttöön, kuten XML-tietokannat. Tutkielman tuloksen toivotaan selventävän kuvaa NoSQL-tietokantajärjestelmien käyttökohteista, ongelmista, puutteista sekä mahdollisesta soveltuvuudesta myös tavanomaisen yrityksen tiedonhallintaan. Tutkielma perustuu pääosin aikaisemmin aiheesta laadittuun kirjalliseen materiaaliin, kuten lehti- ja konferenssiartikkeleihin sekä kirjoihin.

2 Tiedon käsittelyn ja varastoinnin uudet haasteet ja vaatimukset

Digitaalisen tiedon määrä kasvaa koko ajan. Tietoa käsitellään ja varastoidaan yhä enemmän ja kiihtyvällä vauhdilla. Tähän on osaltaan vaikuttanut Internetin nopea kehitys 1990-luvulta lähtien laajaksi maailmanlaajuiseksi tietoverkoksi. Internetissä toimivia palveluja voivat käyttää miljoonat ihmiset ympäri maailmaa. Internetin toimintamalli ja käyttötavat ovat silti jo muuttuneet hyvin lyhyessä ajassa. Internetissä toimivat palvelut olivat alun perin pelkästään staattisia web-sivustoja, joihin käyttäjät eivät itse voineet paljon vaikuttaa. Nykyään monet tahot tarjoavat sen sijaan käyttäjilleen uusia käyttäjäkeskeisiä, interaktiivisia sekä sosiaalista verkostoitumista edistäviä palveluja, jotka toimivat Internetin välityksellä [Mur07]. Näitä palveluja ovat esimerkiksi monet yhteisöllisyyttä tarjoavat sovellukset, kuten Facebook [Fac12], Flickr [Yah12], ja YouTube [Goo12d]. Edellä mainittuja sovelluksia yhdistää usein se, että käyttäjät voivat myös itse tuottaa sisältöä palveluun. Käyttäjät voivat esimerkiksi tuottaa ja ladata tietoa palveluihin digitaalisessa muodossa, kuten blogikirjoituksia, tilapäivityksiä, linkkejä, kuvia, musiikkia ja videoita. Kaikki nämä lataukset lisäävät Internetissä olevaa digitaalisen tiedon määrää.

Näitä palveluja myös tarjotaan hyvin laajalle käyttäjäkunnalle, jopa maailman laajuisesti [Mur07]. Kyseisten palvelujen odotetaan olevan käytettävissä joka paikassa ja milloin tahansa. Tämän kaltaisia sovelluksia kutsutaan usein *Web 2.0 -sovelluksiksi* (Web 2.0 applications). Web 2.0 -käsite on kuitenkin laajempi kokonaisuus ja sisältää edellä mainittujen sovelluksien lisäksi teknologisia ratkaisuja, yritysten liiketoimintastrategioita sekä sosiaalisia trendejä. Web 2.0 -käsite on siis käyttötapaa ja teknologiaa koskeva paradigma, jolla usein tarkoitetaan Internetin toista kehitysvaihetta, jonka aikana Internetin koko toimintamalli kehittyi tiettyyn suuntaan. Web 2.0 tarkoittaa esimerkiksi joustavaa web-sivujen suunnittelua ja ylläpitoa, interaktiivisia käyttöliittymiä, mahdollisuutta tuottaa sisältöä itse ja yhdistellä tietoa eri lähteistä sekä sosiaalisia verkostoja.

Digitaalisen tiedon suuri ja koko ajan kasvava määrä sekä laaja eri puolille maailmaa sijoittuva käyttäjäkunta asettavat aivan uudenlaisia vaatimuksia myös tietojärjestelmille [SKS11, s. 862-863]. Useat Internetissä toimivat tunnetut palvelut, kuten Googlen hakukone [Goo12a], Amazon-verkkokauppa [Ama12] ja Facebook [Fac12] edellyttävätkin valtavien tietomäärien käsittelyä ja varastoimista. Google paljasti vuonna 2009 käsitte-

levänsä jo peräti yli 20 petatavua tietoa päivässä [DeG08]. Facebookin mukaan taas käyttäjät olivat ladanneet vuoteen 2009 mennessä sen palveluun jo yli 15 biljoonaa valokuvaa, jotka veivät yhteensä jopa 1.5 petatavua levytilaa [Vaj09]. Tiedon tulee olla myös koko ajan saatavilla monessa eri paikassa. Esimerkiksi Amazon tarjoaa maailmanlaajuisia lähestulkoon aina käytettävissä olevia web-palvelujaan jopa kymmenille miljoonille yhtäaikaisille käyttäjille, joille se pyrkii takaamaan erittäin korkean toimintavarmuuden [DHJ07]. Tietojärjestelmän pitää siis olla suorituskykyinen, luotettava ja tehokas. Sen tulee myös mukautua kasvavaan tietomäärään. Edellä mainitut vaatimukset edellyttävät tietojärjestelmältä usein hajautettua arkkitehtuuriratkaisua. Tiedot varastoidaan yleensä ympäri maailmaa sijoitettuihin konesaleihin, joissa toimii tuhansia toisiinsa yhteydessä olevia tavanomaisia tietokoneita. Tällaista järjestelmää voidaan kutsua *hajautetuksi tietokantajärjestelmäksi* (distributed database system). Samaa tietoa *toisinnetaan* (replicate) usealle eri koneelle tiedon saatavuuden varmistamiseksi. Toisinnus tarkoittaa saman tiedon tallentamista moneen eri paikkaan esimerkiksi tiedon saatavuuden varmistamiseksi. Näiden palvelujen taustalla toimivan tietokantajärjestelmän tulee myös *skaalautua* (scale) käyttöasteen mukaan eli tietokantaan kohdistuvien operaatioiden lisääntyessä kapasiteettia tulee voida lisätä. Lisäksi tiedon ja tietorakenteiden ominaisuuksiin voi tulla muutoksia, joten tietokannan rakenteen tulee olla joustava ja helposti muokattavissa.

Monet näistä yhtiöistä, kuten esimerkiksi Google ja Amazon hyödyntävät rakentamaansa laajan mittakaavan hajautettua järjestelmää tarjoamalla sitä myös muiden yritysten Web 2.0 -sovelluksien alustaksi ns. *pilvipalveluna* (cloud service) [SKS11, s. 861-862]. *Pilvilaskennan konsepti* (cloud computing concept) tarkoittaa tietotekniikan, kuten tietokoneiden ja palvelimien tarjoamista asiakkaalle palveluna. Asiakas voi palvelussa esimerkiksi itse määritellä kuinka paljon kapasiteettia, kuten laskentatehoa tai levytilaa se tarvitsee. Pilven kapasiteettia voidaan myös lisätä tai vähentää asiakkaan omien tarpeiden mukaan eli kyseessä on siis skaalautuva palvelu. Pilvilaskentaa on olemassa useampaa eri tyyppiä. Pilvilaskennan erilaisissa konseptimalleissa asiakas voi sopimuksen mukaan esimerkiksi asentaa ja/tai ylläpitää omia sovelluksiaan pilvilaskentaa tarjoavan yrityksen palvelimilla. Vaihtoehtona on myös, että pilvilaskentaa tarjoava yritys tarjoaa itse sekä sovellukset että niiden ylläpidon palveluna. Monet pilvipalveluja tarjoavat yritykset tarjoavat esimerkiksi tietovarastointia, karttapalveluja sekä muita palveluja, joita asiakas voi hyödyntää tarjotun web-palvelun *ohjelmointirajapinnan* (application prog-

ramming interface, API) kautta. Ohjelmointirajapinta on lähdekoodiin perustuva määritelmä, jonka määrittelemän rajapinnan avulla ohjelmat voivat kommunikoida toistensa kanssa. Tämän kaltaisia tietovarastointia tarjoavia pilviperustaisia tietovarastointijärjestelmiä kutsutaan Silberschatz ja kumppanit [SKS11, s. 861-862] kirjassa *pilviperustaisiksi tietokannoiksi* (cloud based databases).

3 Hajautetun tietokantajärjestelmän ja rinnakkaiskäsittelyn toimintaperiaatteita

Laajan mittakaavan Web 2.0 -sovellukset sekä massiiviset tietomäärät edellyttävät yleensä hajautetun tietokantaratkaisun käyttöä. Tietokannan tiedot tallennetaan useammalle eri tietokoneelle, jotka ovat yhteydessä toisiinsa nopean tietoverkon ja Internetin välityksellä. Tietokannan skaalautuvuus ja tehokkuuden säilyttäminen taas edellyttävät usein tietokantaoperaatioiden käsittelyä rinnakkain. Tämän tekee mahdolliseksi tietokoneiden prosessorien ja levyjen käyttäminen rinnakkain. Järjestelmää jonka suorituskykyä parannetaan rinnakkain suoritettavilla operaatioilla voidaan kutsua myös *rinnakkaisjärjestelmäksi* (parallel system) [SKS11, s. 777]. Tässä luvussa esitellään hajautetun tietokantajärjestelmän rakenteen ja toiminnan periaatteita sekä sitä, miten tietokannan tietoja voidaan käsitellä rinnakkain samaan aikaan.

3.1 Tietokoneryväs

Tietokoneryväs (computer cluster) on joukko toisiinsa yhteydessä olevia tietokoneita, jotka sijaitsevat jollain tietyllä samalla alueella, esimerkiksi konesalissa [SKS11, s. 867]. Nämä tietokoneet voivat jakaa tehtäviä tai toimia toistensa sijalla, jos jokin niistä ei ole käytettävissä. Tietokonerypäitä kutsutaan usein hajautetun tietokantajärjestelmän *pisteiksi* (site) .

Hajautetun tietokannan tiedot on sijoitettu hajautetun tietokantajärjestelmän eri pisteiden tietokoneisiin [SKS11, s. 827]. Tietojen sijoittaminen voi olla tilapäistä tai enemmän pysyvämmän luonteista. Pisteisiin sijoitetut tiedot ovat yleensä pysyvämpiä. Pisteeseen voidaan sijoittaa pysyvästi esimerkiksi sellaiset tiedot, joita käytetään useammin juuri siinä pisteessä johon ne on sijoitettu. Näin ollen lähimmästä pisteestä löytyvät tiedot vähentävät verkon kautta tehtäviä hakuja muihin pisteisiin. Yksittäiselle rypään tietokoneelle tiedot saattaa sen sijaan olla sijoitettu vain tilapäisesti. Tietoja voidaan siirtää

yksittäiseltä tietokoneelta toiselle esimerkiksi tietokoneiden kuormituksen tasaamiseksi [SKS11, s. 865]. Tietokoneet saattavat myös olla ajoittain poissa käytöstä tietokoneen kaatumisen tai verkkovian takia. Tämän takia niiden palveluja toisinnetaan eri koneille. Yksittäisen tietokoneen poistuessa käytöstä, sen tehtäviä voi alkaa hoitaa joku toinen vielä toiminnassa oleva rypään tietokone.

Tietokoneryväs on skaalautuva yksikkö eli tiedon ja operaatioiden lisääntyessä suorituskyky voidaan pitää ennallaan lisäämällä rypääseen uusia tietokoneita tarpeen mukaan (BDH03). Nämä tietokoneet ovat usein tavanomaisia ja suhteellisen halpoja tietokoneita, jotka eivät välttämättä ole yhtä luotettavia kuin kalliimmat ja tehokkaammat palvelintietokoneet. Ratkaisu perustuu siihen, että myös laadukkaammat tietokoneet kaatuvat joskus. Hajautetun tietokantajärjestelmän pitää siis joka tapauksessa varautua laitevikoihin. Kuten edellä jo mainittiin tietoja toisinnetaan monelle eri koneelle esimerkiksi laitevikojen varalta. Tietokantajärjestelmän virhetilanteet myös pyritään löytämään ja käsittelemään mahdollisimman automaattisesti. Näiden lisäksi hajautetussa tietokantajärjestelmässä operaatioita voidaan käsitellään aidosti rinnakkain useammalla eri tietokoneella samaan aikaan, joten yksittäisten tietokoneiden vasteajoilla ei ole suurta merkitystä. Tämä tietojen *rinnakkaiskäsittely* (parallel processing) tarkoittaa siis operaation jakamista osatehtäviin, jotka voidaan käsitellä rinnakkain useammalla eri tietokoneella samaan aikaan [SKS11, s. 401].

Tietokonejärjestelmät jotka suorittavat rinnakkaiskäsittelyä voidaan organisoida niiden yhteisten resurssien mukaan. Tietokoneryväs on yleensä organisoitu joko *yhteislevyjärjestelmäksi* (shared-disks system) tai *yksityislevyjärjestelmäksi* (shared-nothing system) [SKS11, s. 781]. Yhteislevyarkkitehtuurissa jokaisella tietoja käsittelevillä yksiköllä eli prosessorilla on oma yksityinen keskusmuisti, mutta se käyttää kaikille yhteisiä levyjä [SKS11, s. 783, Rah93]. Jokaisella prosessorilla on näin ollen pääsy erityisen *kytkentäverkon* (interconnection network) välityksellä ulkopuolisiin levyihin, joihin tietokannan tiedot on varastoitu. Tässä arkkitehtuuriratkaisussa tietokannan operaatioiden suorittaminen rinnakkain saattaa kuitenkin olla monimutkaista [SKS11, s. 802]. Yksittäinen tietokannan sivu voi olla samaan aikaan useamman prosessorin puskurissa, koska jokainen prosessori puskuroi tietokannan sivuja omassa puskurissaan. Kun prosessori toteuttaa tietokantaoperaatioita eli sillä oleva transaktio haluaa käyttää prosessorin puskuroitua sivua, on varmistuttava siitä että sivu on sen viimeisin versio. Tätä kutsutaan myös

välimuistin yhteneväisyysongelmaksi (cache-coherency problem). Tämän lisäksi prosessorien on tehtävä tiettyjä tietokannan luotettavuuden varmistavia toimenpiteitä. Eri prosessoreilla olevat transaktiot varaavat esimerkiksi lukkoja samaan tietokantaan. Transaktiot myös kirjataan yhteiseen transaktiolokiin. Jotta tietokannan luotettavuus säilyisi, tietoja käsittelevien prosessorien on koordinoitava toimintojaan. Koordinointi edellyttää viestien välitystä eli kommunikointia eri prosessorien välillä. Koska prosessorit liikennöivät toistensa välillä kytkentäverkon kautta, niin kommunikointi voi olla hitaampaa kuin esimerkiksi käytettäessä prosessorien välistä yhteistä muistia [SKS11, s. 783]. Jos tietokantajärjestelmä suorittaa paljon levyoperaatioita, yhteislevyjärjestelmän ongelmaksi voi myös muodostua yhteys yhteisiin levyihin. Koska prosessorit liikennöivät yhteisiin levyihin kytkentäverkon kautta, niin yhteys levyihin voi hidastua liikenteen lisääntyessä. Yhteislevyjärjestelmä on kuitenkin *vikasietoinen* (fault tolerant) eli se kestää hyvin prosessori- tai muistivikoja. Jos joku prosessori tai sen muisti lakkaa toimimasta, muut prosessorit voivat alkaa hoitaa sen tehtäviä.

Yksityislevyarkkitehtuurissa tietoja käsittelevät yksiköt toimivat itsenäisinä *solmuina* (node), joilla on kullakin oma prosessori, muisti ja yksi tai useampia levyjä [SKS11, s. 783, Rah93]. Jokainen solmu toimii sen levylle tai levyille sijoitettujen tietojen palvelimena. Solmut on kytketty toisiinsa nopean kytkentäverkon kautta. Solmun paikallisille levyille kohdistuvat tietokantaoperaatiot ovat hyvin nopeita, koska niiden ei tarvitse kulkea kytkentäverkon kautta. Tietokantaoperaatiot ja niiden tulokset jotka eivät koske solmun omille levyille tallennettuja tietoja, välitetään silti kytkentäverkon kautta. Solmujen välinen kommunikointi edellyttää myös molempien osallisena olevan solmun toimintaa ohjaavan järjestelmän vuorovaikutusta kummassakin päässä. Yksityislevyjärjestelmässä tietojen saanti muilta kuin solmun omilta levyiltä ja kommunikointi solmujen välillä on siis hitaampaa kuin yhteislevyjärjestelmässä. Yksityislevyjärjestelmien kytkentäverkko on kuitenkin usein suunniteltu skaalautuvaksi. Kytkentäverkon välityskapasiteettia voidaan lisätä sitä mukaan kun siihen liitetään lisää solmuja. Tällä tavoin yksityislevyjärjestelmään voidaan liittää suuri määrä solmuja suorituskyvyn siitä kärsimättä. Yksityislevyjärjestelmä onkin yhteislevyjärjestelmää skaalautuvampi.

Tässä arkkitehtuuriratkaisussa tietokannan operaatioiden suorittaminen rinnakkain ei välttämättä kaikilta osin ole yhtä monimutkaista kuin yhteislevyarkkitehtuurissa [Rah93, SKS11, luvut 17 ja 18]. Tietokannan eheys voidaan varmistaa yksinkertaisem-

min ja tehokkaammin ainakin paikallisten transaktioiden osalta. Jokainen solmu on nimittäin mahdollisimman itsenäinen yksikkö. Jokainen solmu puskuroi vain omien levyjensä tietokantasivuja, joten välimuistin yhteneväisyydestä ei tarvitse varmistua yhteislevyjärjestelmän tapaan. Jokaiseen solmuun on myös sijoitettu transaktioiden hallinnan edellyttämät lokitiedostot sekä katoavat keskusmuistirakenteet, kuten lukkotaulu ja tietokantapuskuri. Paikallisten transaktioiden hallinta ei siis edellytä kommunikointia eri yksiköiden välillä samaan tapaan kuin yhteislevyjärjestelmässä. Sen sijaan, mikäli nämä transaktiot voivat kohdistua useammassa solmussa sijaitseviin tietoihin, niitä joudutaan koordinoimaan samaan tapaan kuten hajautettuja transaktioita yleensä koordinoidaan hajautetun tietokantajärjestelmän eri pisteiden välillä.

3.2 Tiedon saatavuus ja tiedon toisintaminen

Sovellukset jotka hyödyntävät hajautettua tietokantajärjestelmää edellyttävät usein, että tietokantajärjestelmä on aina käytettävissä [SKS11, s. 847]. Hajautetussa ympäristössä tämä on kuitenkin erityisen haasteellista. Hajautetun tietokantajärjestelmän pisteet ja solmut eivät välttämättä ole aina saatavilla erilaisten verkko- ja laitevikojen takia. Laajan mittakaavan ympäristö on lisäksi usein levittäytynyt ympäri maailmaa ja sen piirissä saattaa olla tuhansia tietokoneita ja verkkolaitteita, joten erilaiset viat ovat yleensä hyvin todennäköisiä. Esimerkiksi hajautetun tietokantajärjestelmän pisteet ja solmut toisiinsa kytkevän verkkolaitteen katkos saattaa eristää järjestelmän pisteet eri osiin, esimerkiksi aliverkkoihin, jotka eivät voi enää kommunikoida toistensa kanssa. Tätä kutsutaan nimellä *verkon pirstoutuminen* (network partition). Hajautetun tietokantajärjestelmän pitää siis osata löytää ja korjata erilaisia virheitä sekä toipua niistä ja jatkaa toimintaansa mahdollisimman automaattisesti. Tällaista hajautetun tietokantajärjestelmän vikasietoisuutta ja kykyä jatkaa toimintaansa kutsutaan tietokantajärjestelmän *kestävyydeksi* (robustness).

Hajautetun tietokantajärjestelmän tietokannan tiedoista on usein varastoitu kopio useampaan kuin yhteen solmuun tai pisteeseen [SKS11, s. 826-827]. Tätä varten tietokannan relaatio toisinnetaan eli relaation kopio tallennetaan kahteen tai useampaan solmuun tai pisteeseen. Tietokannasta on mahdollista tehdä myös *kokonaan toisinnettu* (full replication), jolloin relaation kopio on tallennettu hajautetun tietokantajärjestelmän jokaiseen solmuun tai pisteeseen. Tämän menettelyn tarkoituksena on parantaa tieto-

kannan saatavuutta. Jos joku solmu tai piste ei ole enää käytettävissä, voidaan relaation tietoja edelleen hyödyntää sen solmun tai pisteen kautta, jonne relaation *toisinne* (replica) on tallennettu. Hajautetun tietokantajärjestelmän toimintaa voidaan myös tehostaa suorittamalla tietokannan lukuoperaatioita saman relaation eri solmuihin tai pisteisiin tallennetuissa toisinteissa rinnakkain samaan aikaan. Jos lukuoperaation tulos saadaan mahdollisimman lähellä operaatiota suorittavasta pisteestä, se yleensä vähentää liikennettä muihin pisteisiin.

Tietokantajärjestelmän pitää myös varmistaa, että kaikki tietokannan relaation toisinteet ovat johdonmukaisia alkuperäisen relaation kanssa [SKS11, s. 826-827]. Jos alkuperäistä relaatiota päivitetään, niin kaikki päivitykset pitää myös *levittää* (propagate) relaation kaikkiin toisinteisiin. Levittäminen tarkoittaa ajan tasalla olevien tietojen kopioimista ja päivittämistä tietojärjestelmän kaikille osapuolille. Hajautetussa tietokantajärjestelmässä tämä saattaa olla monimutkaista ja tehotonta, koska kaikki pisteet joissa relaation toisinteet sijaitsevat eivät ole välttämättä aina tavoitettavissa. Järjestelmän samanaikaisuuden hallinta onkin monimutkaisempaa kuin keskitetyssä tietokantajärjestelmässä.

3.3 Tiedon osittaminen

Tietokannan relaatio voidaan *hajauttaa* (decluster) *osittamalla* (partition) se useammalle levylle [SKS11, s. 798-799, 827]. Ositus tarkoittaa tietokannan relaation jakamista pienempiin osiin useammalle tietokannan eri levylle jollain tiedonositusmenetelmällä. Näitä relaation osarelaatioita kutsutaan usein myös *paloiksi* (fragment tai tablet). Relaation paloissa tulee olla kaikki tarvittava tieto, jolla alkuperäinen relaatio voidaan mahdollisesti rakentaa uudelleen. Hajautetuissa ja rinnakkaistietokannoissa tiedonositusmenelmänä käytetään yleisimmin *vaakasuoraa ositusta* (horizontal partitioning). Vaakasuorassa osituksessa relaatio hajautetaan useamman levyn kesken niin, että jokainen relaation monikko sijaitsee vain yhdellä levyllä. Relaation r monikot hajautetaan n:lle levylle D_0, D_1…, D_{n-1} yhtä monessa osarelaatiossa r_0, r_1,…, r_{n-1} niin, että jokaisen osarelaation r_i monikot sijoitetaan vastaavalle levylle D_i, kun jokainen osarelaatio r_i on alkuperäisen relaation r osa ja osarelaatiot ovat toisistaan erillisiä.

Vaakasuoraositus voidaan toteuttaa *kiertovuoro-osituksella* (round-robin), *hajautusosituksella* (hash partitioning) tai *osaväliosituksella* (range partitioning) [SKS11, s. 798-802]. Kiertovuoro-osituksessa relaation monikot sijoitetaan tasaisesti riippumatta niiden

tietosisällöstä kaikille tietokannan eri levyille eli jokainen levy saa suurin piirtein saman määrän monikoita. Hajautusosituksessa taas käytetään jotain relaation attribuuttia tai useampaa attribuuttia hajautusfunktiossa ositusattribuuttina, jonka perusteella monikot sijoitetaan tai hajautetaan mahdollisimman tasaisesti tietokannan eri levyille. Vastaavasti osaväliosituksessa monikot sijoitetaan levyille joltain ositusattribuutin arvoväliltä. Näin jokaiselle levylle tallennetaan vain tietyn arvovälin monikot. Yleensä osituksen tavoitteena on jakaa monikot mahdollisimman tasaisesti tietokannan levyille. Osaväliosituksen ongelmana on *vinouma* (skew), joka johtuu siitä, että jonkun relaation osavälien koko saattaa poiketa toisistaan jolloin monikot jakautuvat kyseisen relaation osalta epätasaisesti levyille. Sama ongelma saattaa esiintyä myös hajautusosituksessa, jos hajautusattribuutti esiintyy relaatiossa hyvin monta kertaa.

Ositettujen relaatioiden palojen hajauttaminen tekee myös mahdolliseksi hajautetun tietokantajärjestelmän pisteiden ja solmujen kuorman tasauksen [CDG06, CDG08]. Paloja voidaan tarvittaessa siirtää enemmän kuormitetuilta solmuilta vähemmän kuormitetuille solmuille. Alkuperäisessä relaatiossa peräkkäin sijoittuneet monikot päätyvät hyvin suurella todennäköisyydellä hajautuksessa hakuavaimen perusteella tehdyn vaakasuoran osituksen tuloksena samaan tai muutamaan palaan. Pientä riviväliä koskevat lukuoperaatiot voidaan siis kohdistaa tehokkaasti vain muutamaan palaan, jotka on todennäköisesti tallennettu vain muutamaan rypään solmuun. Samalla tavalla ositusta voidaan hyödyntää jonkun tietyn relaation tietoja koskevan ositusattribuutin, kuten sijaintipaikkakunnan osalta. Jos relaatio ositetaan sijaintipaikkakunnan perusteella, kyseistä paikkakuntaa koskevat kyselyt voidaan kohdistaa tehokkaasti vain tiettyihin paloihin.

Tietokantajärjestelmän *sirpalointi* (sharding) tarkoittaa tietokannan jakamista pienempiin osiin hajauttamalla ja osittamalla relaatioita [IMS10]. Tavoitteena on sijoittaa ne tiedot joihin kohdistuu operaatioita usein samaan aikaan lähekkäin toisiaan samaan tai muutamaan palaan. Tämän lisäksi tavoitteena on relaation hajauttaminen eri tietokantapalvelimille tai fyysiseen sijaintiin niin, että paloihin kohdistuvat kyselyt kohdistuvat mahdollisimman tasaisesti jokaiselle palvelimelle. Sirpaloinnissa relaation osituksessa muodostettu pala muodostaa osan *sirpaleesta* (shard). Sirpale taas muodostuu useammasta samaan paikkaan tallennetusta palasta. Hajautetuissa tietokantajärjestelmissä tietokannan sirpalointi voidaan toteuttaa hajauttamalla tietokannan relaatio eri pisteisiin ja

yksityislevyjärjestelmän solmuihin.

3.4 Relaation rinnakkaiskäsittely ja rinnakkaisliitos

Relaatioiden osittaminen mahdollistaa tietokannan tietojen rinnakkaiskäsittelyn [SKS11, s. 799]. Relaation monikoita voidaan käsitellä samaan aikaan rinnakkain kaikilla niillä levyillä D_i, joille on hajautettu relaation r monikoita sisältäviä paloja r_i. Vastaavasti kun relaatio hajautetaan osittamalla, relaation r palaset r_i voidaan kirjoittaa levyille D_i samaan aikaan rinnakkain. Relaatioon r kohdistuvaa tietokantaoperaatiota voidaan nopeuttaa jakamalla se rinnakkain suoritettaviksi osaoperaatioiksi, jotka suoritetaan samaan aikaan osarelaatioissa r_i [SKS11, s. 804-814]. Tästä käytetään termiä *operaationsisäinen rinnakkaisuus* (intraoperation parallelism). Tietokantakysely on mahdollista rinnakkaistaa riippuen sen sisältämistä tietokantaoperaatioista. Yksittäinen operaatio, kuten lajittelu, valinta, projektio ja liitos voidaan rinnakkaistaa jakamalla se osaoperaatioiksi, joiden tulokset voidaan helposti yhdistää koko operaation tulokseksi. Vastaavasti kyselysuunnitelman sisältämä useampi eri operaatio voidaan rinnakkaistaa laskemalla toisistaan riippumattomat eri operaatiot rinnakkain. Tästä käytetään sanontaa *operaatioiden välinen rinnakkaisuus* (interoperation parallelism). Jos taas operaatiot ovat toisistaan riippuvaisia, toisen operaation tulos voidaan *putkittaa* (pipeline) toisen operaation syötteeksi.

Rinnakkaiskyselyjen tehokkuuden taso riippuu relaation osittamiseen käytetystä menetelmästä [SKS11, s. 799-800]. Esimerkiksi yksinkertaiset koko relaation läpi käyvät *taulukyselyt* (table scan) käsitellään rinnakkain lukemalla relaation kaikki palaset rinnakkain. Tämän kaltaiset kysely ovat tehokkaita kiertovuoro-ositetuissa relaatioissa, koska relaation monikot ovat jakautuneet tasaisesti kaikille levyille. Jonkun attribuutin perusteella tiettyyn monikkoon kohdistuvat *pistekyselyt* (point query) taas käsitellään kohdistamalla kysely siihen relaation palaseen, joka sisältää osavälin johon arvo kuuluu tai jolle hajautusfunktio kuvaa haettavan arvon. Jos relaatioita ei ole ositettu osaväleittäin tai hajauttamalla, joudutaan lukemaan relaation kaikki palaset rinnakkain läpi. *Osavälikyselyjen* (range query) käsittely riippuu siitä onko relaatio ositettu sen attribuutin perusteella jonka osaväliä kysellään. Jos näin on ja relaatio on osaväliositettu, voidaan kysely toteuttaa lukemalla rinnakkain kaikki ne relaation palaset jotka leikkaavat kyseistä osaväliä. Jos relaatio on ositettu jonkun muun attribuutin mukaan, joudutaan taas lu-

kemaan relaation kaikki palaset rinnakkain läpi.

Myös relaatioiden liitoksien laskenta voidaan rinnakkaistaa [SKS11, s. 806-811]. Tämä onnistuu jos kyseessä on yhtäläisyys- tai luonnollinen liitos. Liitos voidaan laskea *ositettuna rinnakkaisliitoksena* (partitioned parallel join). Molemmat liitettävät relaatiot ositetaan hajautusosituksella samalla hajautusfunktiolla eri prosessoreille. Jokainen prosessori saa osituksen tuloksena palan molemmista relaatioista, joiden liitoksen se voi laskea itsenäisesti jollakin tavanomaisella liitosmenetelmällä. Näin kaikki prosessorit voivat laskea omien palojensa liitokset samaan aikaan rinnakkain. Lopullinen tulos saadaan aikaan yhdistämällä operaation lopuksi kaikkien prosessorien tulokset.

Tietokantajärjestelmän kapasiteettia voidaan lisätä lisäämällä tietokannan rinnakkaiskäsittelyä [SKS11, s. 779-780]. Tietokantaoperaatioiden rinnakkaiskäsittelyn tavoitteena onkin yleensä tietokantajärjestelmän suorituskyvyn säilyttäminen, vaikka tietokannan koko ja operaatioiden lukumäärä kasvaisi. Rinnakkaistamalla tietokantaoperaatioiden laskentaa voidaan vaikuttaa tietokantaoperaatioiden kustannuksiin jakamalla työmäärää suoritettavaksi samaan aikaan useammalle eri taholle, kuten eri levyille tai prosessoreille. Tämän ansiosta operaatioihin kuluvaa aikaa voidaan vähentää. Rinnakkaislaskennasta syntyy kuitenkin myös lisäkustannuksia, jotka täytyy ottaa huomioon operaatioiden kokonaiskustannuksia laskettaessa. Operaatioiden käynnistämiseen usealla eri prosessorilla aiheuttaa *käynnistyskustannuksia* (startup cost). Käynnistyskustannukset voivat olla merkittävän suuria isoissa, jopa tuhansien operaatioiden rinnakkaisoperaatioissa. Myös useiden eri osaoperaatioiden *kilpailu resursseista* (interference) saattaa aiheuttaa viivästyksiä. Rinnakkain suoritettavat prosessit kilpailevat usein samoista jaetuista resursseista, kuten väyläohjain, jaetut levyt tai tietokannan lukot. Nämä jaetut resurssit saattavat olla jo jonkun toisen prosessin käytössä, joten resurssin vapautumista joudutaan odottamaan. Lisäksi työkuorman jakelussa saattaa esiintyä vinoumaa, joka aiheuttaa enemmän työmäärää joillekin prosessoreille. Tasaisesti jakautunut työkuorma voidaan laskea tehokkaammin rinnakkain, joten mahdollinen vinouma vaikuttaa negatiivisesti rinnakkaisen laskennan tehokkuuteen. Jos taas laskennan lopuksi pitää koota lopullinen tulos useammalta prosessorilta, se aiheuttaa *koontikustannuksia* (cost of assembling).

4 Tietokannan laatuun vaikuttavat ominaisuudet

Tietokannan laatuun vaikuttavat tietyt ominaisuudet, joita tietokantajärjestelmä noudat-

taa toiminnassaan. Esimerkiksi tietokannan luotettavuus voidaan taata noudattamalla transaktioiden ACID-ominaisuuksia. Hajautetut tietokantajärjestelmät eivät aina noudata kaikkia transaktioiden ACID-ominaisuuksia. Näistä ominaisuuksista luopumalla tähdätään parempaan skaalautuvuuteen, saatavuuteen ja suorituskykyyn. Tässä luvussa käydään läpi edellä mainitut ACID-ominaisuudet sekä esitellään joitakin vaihtoehtoisia malleja, joita usein käytetään hajautetun tietokannan laadun takaamiseen.

4.1 Transaktioiden ACID-ominaisuudet

Transaktion ACID-ominaisuudet takaavat tietokannan luotettavuuden [SKS11, luku 14]. ACID-ominaisuudet koostuvat seuraavista neljästä eri ominaisuudesta:

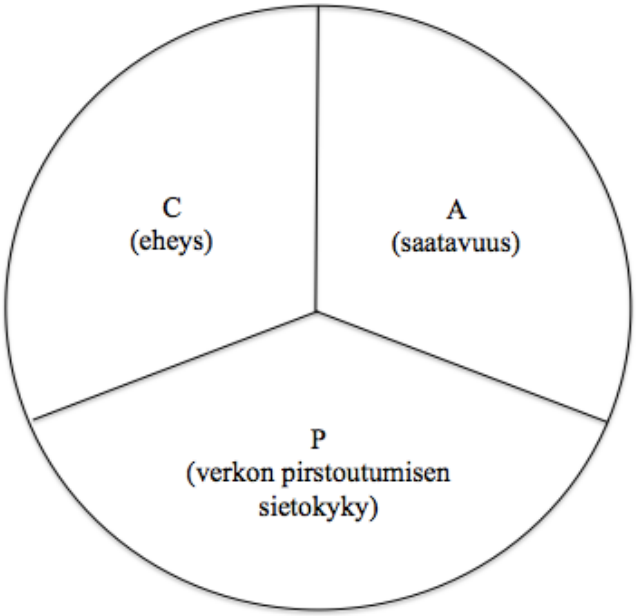
- atomisuus tai jakamattomuus,
- oikeellisuus tai eheys,
- eristyvyys tai erillisyys,
- pysyvyys.

Atomisuus (atomicity) tarkoittaa, että kaikki transaktion suorittamat operaatiot suoritetaan loppuun asti tai niitä ei suoriteta ollenkaan. *Oikeellisuus* (consistency) taas tarkoittaa, että tietokannan tila säilyy eheänä transaktion suorituksen jälkeenkin. *Eristyvyydellä* (isolation) tarkoitetaan, että muut tapahtumat eivät saa vaikuttaa yksittäisen transaktion suoritukseen. Transaktioiden eristyvyys taataan lukituksella, jossa transaktion sallitaan suorittaa tietokantaan kohdistuvan luku- tai kirjoitusoperaation vain, jos se on lukinnut operaation kohteen asianmukaisella lukolla. *Pysyvyys* (durability) taas tarkoittaa. että transaktion aikaansaamat muutokset tietokantaan ovat pysyviä.

Hajautettujen transaktioiden kohdalla käytetään usein *kaksivaiheista transaktioiden sitoutumiskäytäntöä* (two-phase commit) 2PC, varmistamaan jokaisen hajautetun transaktion osapuolen ACID-ominaisuuksien mukainen luotettavuus. Nimensä mukaisesti käytännön toteutus tapahtuu kahdessa eri vaiheessa. Ensimmäisessä vaiheessa transaktioita koordinoiva taho kysyy jokaiselta osapuolelta ovatko ne valmiita sitoutumaan. Jos kaikki osapuolet ovat valmiita sitoutumaan, koordinoiva taho pyytää toisessa vaiheessa jokaista osapuolta sitoutumaan. Jos joku osapuoli haluaa keskeyttää transaktion tai päätöstä ei saada määräajassa, koordinoiva taho keskeyttää ja peruuttaa transaktion.

4.2 CAP-teoreema

Californian Berkeley yliopiston professori Eric A. Brewer [Bre00] esitteli vuonna 2000 kehittämänsä ns. CAP-teoreeman (CAP Theorem). CAP-teoreeman mukaan hajautettu tietojärjestelmä voi taata vain kaksi seuraavista ominaisuuksista: *eheys* (consistency), *saatavuus* (availability) ja *verkon pirstoutumisen sietokyky* (tolerance to network partitions). Kuvassa 1 CAP-teoreeman ominaisuudet on esitetty niiden englanninkielisistä nimistä johdettujen lyhenteiden C, A ja P avulla. Hajautetun tietokantajärjestelmän takaamat CAP-teoreeman mukaiset ominaisuudet esitetään yleensä näillä kirjainyhdistelmillä. Esimerkiksi järjestelmällä joka takaa saatavuuden ja verkon pirstoutumisen sietokyvyn on ominaisuudet AP.



Kuva 1: CAP-teoreema.

CAP-teoreemassa esitetyt ominaisuudet voidaan tulkita hajautettujen tietokantojen osalta seuraavalla tavalla [Sto10]. Eheyden saavuttamiseksi useaan eri pisteeseen kohdistuvat tietokantaoperaatiot tulee suorittaa loppuun saakka tai peruuttaa kokonaan. Myös tietokannan toisinteiden tulee olla eheitä. Tämän seurauksena kaikki hajautetun tietokantajärjestelmän solmut näkevät kaikki samat tiedot samaan aikaan. Saatavuus taas edellyttää, että tietokantajärjestelmän tulee pysyä aina toiminnassa eli kaikki normaalit luku- ja kirjoitusoperaatiot ovat mahdollisia virheistä huolimatta, esimerkiksi tietokannan toisinteiden avulla. Kolmas vaihtoehto eli verkon pirstoutumisen sietokyky tarkoittaa, että verkon pirstoutuessa tietojen käsittelyn tulee edelleen jatkua solmuissa.

4.3 BASE-oikeellisuusmalli

CAP-teoreemaa on usein käytetty perusteena tarpeelle transaktioiden ACID-ominaisuuksia heikentävään luotettavuusmalliin [Pri08, Bre00]. Kuten edellä jo todettiin, CAP-teoreeman mukaan jaettu järjestelmä voi taata vain kaksi seuraavista ominaisuuksista: eheys, saatavuus ja verkon pirstoutumisen sietokyky. ACID-ominaisuuksien mukaisen transaktioiden oikeellisuuden ja eristyvyyden takaaminen saattaa kuitenkin vaikuttaa negatiivisesti hajautetun tietokantajärjestelmän saatavuuteen. Esimerkiksi kaksivaiheisessa transaktioiden sitoutumiskäytännössä saatavuuteen vaikuttaa jokaisen transaktioon osallistuvan osapuolen saatavuus. Hajautetun tietokannan saatavuuteen voi myös vaikuttaa tietoalkioiden lukitseminen eristyvyyden takaamiseksi.

BASE-*oikeellisuusmallin* (BASE consistency model) avulla pyritään takaamaan hajautetulle järjestelmälle mahdollisimman korkea saatavuus ja tehokkuus. Tämä saavutetaan heikentämällä ACID-ominaisuuksista oikeellisuutta ja eristyvyyttä. BASE-oikeellisuusmallin ominaisuudet koostuvat seuraavista kolmesta eri ominaisuudesta:

- periaatteessa käytettävissä,
- ei aina oikeellinen,
- lopulta oikeellinen.

BASE-mallia noudattamalla päämääränä on, että järjestelmä on *periaatteessa käytettävissä* (basically available) ja järjestelmän *ei tarvitse olla aina oikeellinen* (soft state), mutta se päätyy oikeelliseen tilaan aina *jossain vaiheessa* (eventually consistent). Tämä tarkoittaa, että tietokannan ei tarvitse olla oikeellisessa tilassa jokaisen transaktion jälkeen, vaan riittää että tietokannan oikeellisuus saavutetaan jossain myöhemmässä vaiheessa. Kyseinen menettely on ACID-ominaisuuksien vastainen. BASE-oikeellisuusmalli siis sallii tietokannan tietojen olevan jossain tilanteessa vanhentuneita.

5 Relaatiotietokantajärjestelmät

Relaatiotietokantajärjestelmät ovat yhä edelleen kaikkein yleisimmin käytettyjä tietokantajärjestelmiä. Kyseisiä järjestelmiä käytetään hyvin yleisesti kaiken kokoisten yritysten liiketoiminnan sovelluksissa. Tämän lisäksi relaationtietokantajärjestelmillä on paljon muitakin käyttökohteita. Laajan mittakaavan Web 2.0 -sovellukset ja massiiviset

tietomäärät ovat kuitenkin asettaneet näille perinteisille tietokantajärjestelmille aivan uudenlaisia haasteita. Tässä luvussa käsitellään millaisiin tarkoituksiin relaatiotietokantajärjestelmiä on käytetty sekä millaisia haasteita laajan mittakaavan sovellukset niille asettavat. Luvun lopuksi vielä esitellään minkälaista kritiikkiä relaatiotietokantajärjestelmät ovat saaneet osakseen.

5.1 Relaationtietokantajärjestelmien käyttökohteet

Alun perin E. F. Coddin kehittämään relaatiomalliin perustuvat relaatiotietokantajärjestelmät kehitettiin jo 1970-luvulla [VMZ10]. Sen jälkeen relaatiotietokantajärjestelmät ovat hallinneet kaupallisiin, akateemisiin ja tieteellisiin tarkoituksiin kehitettyjen tietokantajärjestelmien markkinoita aina näihin päiviin asti. Kaupallisia relaatiotietokantojen valmistajia ovat esimerkiksi Oracle, Microsoft ja IBM. Avoimeen lähdekoodiin perustuvista relaatiotietokantajärjestelmät tunnetuimpia ovat MySQL [Ora12a] ja PostgreSQL [Pos12]. Pienyrityksen ja henkilökohtaiseen käyttöön on kehitetty sen kaltaisia ohjelmia kuten Microsoftin Access, kun taas suuren mittaluokan käyttöön on kehitetty tehokkailla tietokantapalvelimilla toimivia ohjelmia, kuten Oracle, Microsoftin SQL-Server tai IBM:n DB2.

Relaatiotietokantajärjestelmien hyödylliset ominaisuudet ovat tehneet niistä suosittuja [Ler10]. Relaationtietokannat ovat esimerkiksi suhteellisen vakaita ja turvallisia. Niiden noudattamat transaktioiden ACID-ominaisuudet pyrkivät takaamaan tietokannan luotettavuuden. Relaatiotietokannan relaatioihin eli tauluihin on taas ollut suhteellisen yksinkertaista mallintaa tietoja tavanomaisissa tapauksissa. Relaationtietokantojen käyttämä SQL-kyselykieli on suurimmaksi osaksi deklaratiivinen kieli, jolla voidaan kuvailla intuitiivisesti mitä halutaan tehdä. Sen takia SQL-kieltä on myös helpompi ymmärtää. Näiden lisäksi relaatiotietokantajärjestelmiin on usein kehitetty uusia ominaisuuksia tarpeen niin vaatiessa. Näitä ovat esimerkiksi mahdollisuus tallentaa tietokantaan xml-muotoista tietoa sekä erilaiset analysointi- ja raportointiominaisuudet.

Relaatiotietokantajärjestelmät ovat osoittautuneet alusta lähtien toimiviksi ja luotettaviksi ratkaisuiksi esimerkiksi erilaisten perinteisten *liiketoimintaan liittyvien tietojenkäsittelytoimintojen* (business data processing), kuten kirjanpidon, laskutuksen ja varastohallinnan sovelluksien käytössä [StC11]. Relaatiotietokantajärjestelmiä on silti käytetty ja kehitetty myös muihin tarkoituksiin. Näihin järjestelmiin on esimerkiksi kehitetty

uusia versioita, päivityksiä ja laajennososia, joiden avulla niiden käyttömahdollisuudet ovat entisestään laajentuneet. Relaatiotietokantajärjestelmiä on kehitetty esimerkiksi tietovarastointiin, tieteellisen tiedon käsittelyyn ja varastointiin sekä web-sivustojen, kuten sosiaalisen median ja pelisivustojen sovellusten käyttöön. Myös hajautettuun ja rinnakkaiseen suuren mittaluokan käyttöön on kehitetty relaatiotietokantajärjestelmiä. Esimerkiksi Oraclen Real Application Clusters (RAC) [Ora12b] on yhteislevyarkkitehtuuriin perustuva ohjelma, jonka avulla Oraclen relaatiotietokantajärjestelmää voidaan käyttää hajautetusti tietokonerypäissä. IBM:n DB2 Parallel Edition (PE) taas on yksityislevyarkkitehtuuriin perustuva sekä tietokantajärjestelmän loogisista solmuista koostuva relaatiomalliin perustuva rinnakkaistietokantajärjestelmä [BaF95].

Tietokantajärjestelmiä käyttävät sovellukset voivat toimia hyvin eri tavoin ja vaatia järjestelmältä toisistaan poikkeavia ominaisuuksia. Tietokantajärjestelmiä käyttävät sovellukset voidaan jaotella esimerkiksi niiden suorittamien operaatioiden tyypin ja määrän mukaan. Stonebraker ja kumppanit [StC11] toteavat, että esimerkiksi tietovarastoinnin sovellukset ovat keskittyneet lukuoperaatioihin ja koostuvat enimmäkseen monimutkaisista tuhansien kohteiden luku- tai kirjoitusoperaatioista. Liiketoiminnan operatiiviset tietojenkäsittelysovellukset, joita nykyään kutsutaan online transaction prosessing eli OLTP-järjestelmiksi, ovat taas keskittyneet kirjoitusoperaatioihin ja koostuvat enimmäkseen yksinkertaisista vain muutamien kohteiden luku- tai kirjoitusoperaatioista. Uusien Web 2.0 -sovellusten, kuten yhteisöpalvelujen sovelluksien operaatiot sijoittuvat edellä mainittujen sovellusten välimaastoon ja koostuvat enimmäkseen yksinkertaisista luku- ja kirjoitusoperaatioista. Stonebraker ja kumppanit [StC11] esittävät, että kaikkien merkittävien sekä kaupallisten että avoimeen lähdekoodiin perustuvien relaatiotietokantajärjestelmien toteutusten väitetään olevan soveltuvia kaikkien eri operaatiotyyppejä ja määriä suorittavien sovellusten tarpeisiin. Stonebraker ja kumppanit [StC11] kutsuvat tämän kaltaisia järjestelmiä *yksi koko sopii kaikille –järjestelmiksi* (one-size-fits-all).

5.2 Relaatiotietokantojen haasteet laajan mittakaavan ympäristössä

Laajan mittakaavan Web 2.0 -sovellukset ja massiiviset tietomäärät ovat asettaneet relaatiotietokantajärjestelmille isoja haasteita [Tiw11]. Nämä haasteet liittyvät tehokkaaseen tietojenkäsittelyyn, tietokantaoperaatioiden rinnakkaistamiseen, järjestelmän skaalautuvuuteen sekä kustannuksiin. Relaatiotietomalli edellyttää, että tietokantaan tallen-

nettava tieto on rakenteellista, tiheää ja pääosin yhtenäistä. Oletuksena on myös, että tiedon ominaisuudet ja tietokannan rakenne voidaan määritellä etukäteen. Eri tietokokonaisuuksiin liittyvien keskinäisten suhteiden tulee olla pysyviä. Näiden lisäksi tietokantakyselyjen tehostamiseksi relaatiotietokannan tietokokonaisuuksiin tulee olla mahdollista rakentaa oikeellisia indeksejä. Laajan mittakaavan Web 2.0 -sovelluksilla ja massiivisilla tietomäärillä edellä mainitut ominaisuudet eivät välttämättä toteudu. Tieto on usein harvaa ja semirakenteellista tai jopa kokonaan rakenteetonta. Web 2.0 –sovellukset saattavat esimerkiksi usein vaatia paljon attribuutteja, joista vain osa on käytössä. Tiedon ja tietorakenteiden ominaisuuksiin voi tulla muutoksia jälkikäteen. Myös indeksien toteuttaminen ja ylläpito saattaa olla hankalaa.

Relaatiomallin edellyttämien taululiitosten toteuttaminen voi olla hankalaa laajan mittakaavan hajautetussa järjestelmässä. Jos esimerkiksi käsiteltävänä on iso määrä ja paljon keskinäisiä liitoksia vaativia hajautettuja suuria tauluja, näiden taulujen liitosten laskenta voi olla tehotonta ja viedä esimerkiksi liian kauan aikaa [Lea10, VMZ10]. Sen lisäksi semirakenteellisen tai rakenteettoman tiedon sovittaminen relaatiomalliin saattaa tehdä tietokannan rakenteesta monimutkaisen ja sitä kautta tehottoman [Lea10]. Relaatioiden hajauttaminen ja tietokantaoperaatioiden rinnakkaislaskennan toteuttaminen voi myös olla vaikeaa, koska relaatiotietokantajärjestelmää ei oltu alun perin suunniteltu tiedon osittamista varten.

Relaatiotietokantajärjestelmät noudattavat operaatioissaan transaktioiden ACID-ominaisuuksia [Aba10]. Kaikkien ACID-ominaisuuksien noudattaminen saattaa kuitenkin olla monimutkaista ja tehotonta, jos tiedot on varastoitu suurelle määrälle eri tietokoneita ja tavoitteena on tietokantajärjestelmän hyvä skaalautuvuus ja korkea saatavuus. Esimerkiksi tiedon oikeellisuutta on hankalaa ylläpitää, jos samaa tietoa pitää ylläpitää monella eri koneella tiedon korkean saatavuuden varmistamiseksi.

5.3 Kritiikkiä relaatiotietokantajärjestelmiä kohtaan

Suosituimmat kaupalliset relaatiotietokantajärjestelmät perustuvat jo 25 vuotta sitten suunniteltuun arkkitehtuuriin [SMA07]. Oracle, Microsoftin SQL-Server ja IBM:n DB2 ovat kaikki tavalla tai toisella perustuneet tai saaneet vaikutteita IBM:n jo 1970-luvulla kehittämästä System R –tietokantajärjestelmästä. System R:n jälkeen tietojenkäsittelytieteen alalla on tapahtunut huimaa kehitystä. Prosessorien nopeus ja muistien tallen-

nuskapasiteetti ovat moninkertaistuneet. Tietokantajärjestelmien markkinat ovat myös kehittyneet. Ensimmäiset kaupalliset relaatiotietokantajärjestelmät oli kehitetty perinteisten liiketoimintaan liittyvien yritystoimintojen tarpeiden kannalta. Tämän jälkeen tietokantajärjestelmien markkinat ovat kuitenkin kehittyneet ja rinnalle on tullut esimerkiksi tietovarastoja, *tekstin hallintaa* (text management), *tietovirtojen käsittelyä* (stream processing) ja tieteellisiä tietokantoja. 1970-luvulla tietokoneen ensisijainen käyttöliittymä oli yksinkertainen *terminaali-ikkuna* (terminal prompt), jonka kautta tietokoneelle annettiin käskyjä. Nykyään Internetin, maailmanlaajuisen tietoverkon ja siihen kytkettyjen tehokkaiden tietokoneiden myötä, tietokannan operaatioita voidaan suorittaa websivujen välityksellä. Kuten on jo tullut esille, relaatiotietokantajärjestelmiin on kehitetty vuosien varrella uusia ominaisuuksia vastaamaan näihin muuttuneisiin vaatimuksiin. Kyseisten järjestelmien arkkitehtuuri perustuu silti jossain määrin edelleen aivan eri aikakauden vaatimuksiin.

Vaikka kaupalliset relaatiotietokantajärjestelmät oli alun perin suunniteltu edellä mainittujen perinteisten yritystoimintojen tarpeisiin, on niitä sittemmin käytetty lähes kaikkiin tietojenkäsittely- ja varastointitarpeisiin [HeJ11]. Näin on tehty huolimatta siitä, vaikka varastoitavat tiedot eivät sopisi kovin hyvin yhteen relaatiotietokantajärjestelmien käyttämän relaatiomallin kanssa. Stonebraker ja kumppanit [SMA07] esittävät artikkelissaan, että perinteiset kaupalliset relaatiotietokantajärjestelmät eivät välttämättä ole paras ratkaisu kaikkiin erilaisiin tietojenkäsittelyn ja varastoinnin vaatimuksiin. Sen sijaan, että yritetään ratkaista kaikkia uusia vaatimuksia vanhaan arkkitehtuuriin perustuvalla yksi koko sopii kaikille -tietokantajärjestelmällä, tulee kehittää kokonaan uusia vain tiettyihin tarkoituksiin erikoistuneita tietokantajärjestelmiä. Nämä uudet järjestelmät tulee kehittää täysin puhtaalta pöydältä ilman vanhaan arkkitehtuuriin perustuvia rasitteita esimerkiksi vanhaa koodia.

6 NoSQL-tietokantajärjestelmät

Laajan mittakaavan Web 2.0 -sovelluksien käyttöön on 2000-luvulla kehitetty hajautettuja tietokantajärjestelmiä, jotka eivät perustu relaatiomalliin. Tässä luvussa käsitellään näiden tietokantajärjestelmien ominaisuuksia sekä toimintaperiaatteita. Sen lisäksi tutkitaan näiden tietokantajärjestelmien syntyä ja kehitystä sekä esitellään miten näitä tietokantajärjestelmiä on luokiteltu. Tässä luvussa esitellään myös MapReduce-ohjelmointi-

paradigma, jonka avulla näissä tietokantajärjestelmissä voidaan suorittaa rinnakkaislaskentaa. Luvun lopuksi esitellään vielä minkälaista kritiikkiä nämä tietokantajärjestelmät ovat saaneet osakseen.

6.1 NoSQL-tietokantajärjestelmien ominaispiirteet

Relaatiotietokantajärjestelmien saatavuus, skaalautuvuus ja suorituskyky on usein koettu riittämättömäksi laajan mittakaavan Internet-ympäristössä. Perinteisille relaatiotietokantajärjestelmille onkin viime vuosina kehitetty muita vaihtoehtoja. Näitä vaihtoehtoisia tietokantajärjestelmiä on kehitetty laajan mittakaavan Web 2.0 -sovelluksien tarpeita varten ja ne perustuvat usein avoimeen lähdekoodiin. Näitä järjestelmiä on alettu kutsua ja luokitella NoSQL-tietokantajärjestelmiksi.

Näiden tietokantajärjestelmien tietokannan rakenne on suunniteltu hyvin yksinkertaiseksi ja joustavaksi. NoSQL-tietokantajärjestelmien käyttävätkin pääosin jotain relaatiomallia paljon yksinkertaisempaa tietomallia, kuten *avain-arvo-mallia* (key-value model) [Cat10]. Avain-arvo-malli perustuu avain ja arvo pariin, jossa varastoitu arvo löydetään indeksoitavan avaimen perusteella. Näiden järjestelmien tietokannan skeema on usein hyvin joustava tai tietokanta saattaa olla kokonaan skeematon. Tietokannan rakennetta ei siis tarvitse olla määritelty kokonaan etukäteen. Sovelluskehittäjät voivat usein tehdä muutoksia skeemaan jälkikäteen, esimerkiksi ohjelmointirajapinnan kautta.

NoSQL-tietokantajärjestelmien ominaispiirteisiin kuuluu myös, että ne skaalautuvat osittamalla ja toisintamalla tietoa monelle eri koneelle. Ositus tapahtuu pääsääntöisesti vaakasuoraan osittamalla. Ositukseen käytettävää hajautusfunktiota ei usein ole mahdollista määritellä etukäteen [SKS11, s. 865]. Tämän takia tietokannan taulu ositetaan (usein avaimen perusteella) yleensä automaattisesti, kun taulun koko kasvaa tarpeeksi suureksi, suhteellisen pieniksi vain muutamien satojen megatavujen taulujen paloiksi.

NoSQL-tietokantajärjestelmät eivät yleensä noudata kaikkia transaktioiden ACID-ominaisuuksia [SKS11, s. 866]. Näitä ominaisuuksia löyhentämällä tähdätään parempaan skaalautuvuuteen, saatavuuteen ja suorituskykyyn. ACID-ominaisuuksien mukaista tietojen oikeellisuutta on löyhennetty korkean saatavuuden ja suorituskyvyn saavuttamiseksi. NoSQL-tietokantajärjestelmät noudattavat usein löyhempää BASE-oikeellisuusmallia, jolloin tietokannan ei tarvitse olla aina oikeellinen, vaan riittää että tietokannan oikeellisuus saavutetaan jossain myöhemmässä vaiheessa. Joku transaktio-

naalinen ACID-ominaisuus voidaan silti taata jollekin tietylle tietokannan osalle. Esimerkiksi transaktion operaatioiden atomisuuden toteutuminen voidaan taata vain rivikohtaisesti. NoSQL-tietokantajärjestelmät eivät usein voi ylläpitää toissijaisia transaktionaalisesti oikeellisia indeksejä, koska niitä saattaa olla hankalaa ja tehotonta ylläpitää kun tiedot on hajautettu ja ositettu jo avaimen perusteella. Toissijaisen indeksin muodostavaan attribuuttiin kohdistuvat lisäykset ja päivitykset pitää nimittäin indeksin päivittämistä varten levittää edelleen useampaan hajautetun tietokantajärjestelmän pisteeseen.

NoSQL-tietokantajärjestelmien tietokantaoperaatioiden laskentamenetelmät perustuvat hajautetun tietokantajärjestelmän pisteiden solmuissa rinnakkain suoritettaviin osaoperaatioihin, joiden tulokset lopuksi yhdistetään. Rinnakkaisessa laskennassa hyödynnetään samoja rinnakkaiskäsittelyn menetelmiä, kuin perinteisissä relaatiotietokantajärjestelmissä. NoSQL-tietokantajärjestelmien tietorakenteet ovat kuitenkin usein niin yksinkertaisia, että tietojen analysointiin ei välttämättä tarvita kaikkia perinteisen SQL-kielen monimutkaisimpia ominaisuuksia [PDG05]. NoSQL-tietokantajärjestelmille on kehitetty omia paljon yksinkertaisempia kyselykieliä. Käytössä oleva kyselykieli on usein rajoittunut yksittäisten tietoalkioiden lukemiseen ja päivittämiseen [SKS11, s. 863]. Johonkin avaimeen liittyvä arvo voidaan varastoida tietokantaan esimerkiksi yksinkertaisella put(key, value) komennolla. Vastaavasti tietyllä avaimella varastoituun arvoon päästään käsiksi get(key, value) komennolla. Joissakin NoSQL-tietokantajärjestelmissä arvoja voidaan hakea myös avainten *arvovälillä* (range query). Sovelluskehittäjille tarjotaan yleensä hyvin pelkistetty ohjelmointirajapinta näiden yksinkertaisten tietokantaoperaatioiden suorittamiseen.

6.2 NoSQL käsitteenä ja ilmiönä

NoSQL-termiä käytettiin jo vuonna 1998. Carlo Strozzi [Str12, Pat99] kutsui kehittämäänsä avoimeen lähdekoodiin perustuvaa järjestelmää nimellä NoSQL. Kyseessä on kuitenkin relaatiotietokantajärjestelmä, jossa ei käytetä SQL:ää kyselykielenä. Termin NoSQL otti uudelleen käyttöön Eric Evans [Eva09, Ler10] vuonna 2009 järjestetyssä tapahtumassa, jossa käsiteltiin avoimeen lähdekoodiin perustuvia tietokantajärjestelmiä. NoSQL ei siis ole käsitteenä kaikkein paras mahdollinen käsitteen tulkinnanvaraisuuden vuoksi. Voidaan esimerkiksi tulkita, että NoSQL tarkoittaa *ei SQL* (no SQL) tai *ei*

SQL:lle (no to SQL). Joissain lähteissä termistä käytetään tulkintaa *ei relaationaalinen (*non relational). Yleisesti käytetty määritelmä NoSQL-termille on kuitenkin, että NoSQL tarkoittaa *ei vain SQL* (not only SQL).

Internetin ja Web 2.0:n kehityksen lisäksi 2000-luvun NoSQL-ilmiön syntyyn ovat vaikuttaneet vahvasti suuret Internetissä toimivat yhtiöt Google ja Amazon, jotka kehittivät omien palvelujensa perustaksi omat avain-arvo-malliin perustuvat tietokantajärjestelmänsä. Googlen Bigtable on rakenteellisen tai puolirakenteellisen tiedon hallintaan ja varastointiin suunniteltu suuren mittakaavan hajautettu tietokantajärjestelmä, joka on kehitetty Googlen Internetissä toimivien palvelujen perustaksi [CDG06, CDG08]. Näitä palveluja ovat esimerkiksi Google hakukone [Goo12c], YouTube [Goo12d], Google Earth [Goo12b] ja Google+ [Goo12a]. Google julkaisi vuonna 2006 sen omien työntekijöiden laatiman artikkelin [CDG06, CDG08], jossa kuvaillaan Bigtablen rakennetta ja toimintaa. Bigtablen toiminta on riippuvainen muista Googlen järjestelmistä. Bigtable hyödyntää toiminnassaan Googlen omaa *GFS-tiedostojärjestelmää* (The Google File System, GFS) ja *Chubby-lukituspalvelua* (Chubby Lock Service). Googlen tietokantajärjestelmässä voidaan myös hyödyntää MapReduce-ohjelmointiparadigmaa. MapReduce on ohjelmointikehys, jonka avulla voidaan suorittaa rinnakkaislaskentaa laajan mittakaavan hajautetussa ympäristössä.

Googlen tietokantajärjestelmän arkkitehtuurin julkistamisen jälkeen on kehitetty useita samankaltaiseen arkkitehtuuriin perustuvia tietokantajärjestelmiä [Tiw11]. Ensimmäisenä näiden uusien järjestelmien joukossa oli avoimeen lähdekoodiin perustuva Lucene-hakukone, jolla oli samoja Googlen tietokantajärjestelmän arkkitehtuuria muistuttavia ominaisuuksia. Tämän jälkeen Lucenen kehittäjät auttoivat Yahoota kehittämään *Hadoop-ohjelmistokehyksen* (Hadoop Software Framework), jonka avulla Googlen tiedostojärjestelmää muistuttavassa ympäristössä voidaan suorittaa MapReducen kaltaista rinnakkaislaskentaa. Nykyään Hadoop on Yahoon lisäksi käytössä esimerkiksi Facebookilla.

Idea NoSQL-tietokantajärjestelmistä ja NoSQL-termi nousi jälleen pinnalle Hadoopin kehityksen yhteydessä [Tiw11]. Kiinnostukseen tämän kaltaisia järjestelmiä kohtaan vaikutti omalta osaltaan, kun myös ison verkkokaupan omistava Amazon päätti julkistaa oman tietokantajärjestelmänsä arkkitehtuurin Googlen esimerkkiä seuraten. Amazonin Dynamo on Amazonin korkeaa saatavuutta edellyttävää Amazon-verkkokauppaa

[Ama12] varten kehitetty hajautettu tietokantajärjestelmä [DHJ07]. Amazon julkaisi vuonna 2007 Dynamoa kuvaavan artikkelin. Kumpikaan isojen Internet-yhtiöiden tietokantajärjestelmistä ei siis perustunut relaatiomalliin. Näiden yhtiöiden sekä niistä mallia ottaneiden muiden yhtiöiden relaatiomallista poikkeavat ratkaisut ovat herättäneet paljon kiinnostusta sovelluskehittäjien sekä eri kokoisten yritysten keskuudessa. NoSQL-tietokantajärjestelmien tallennusratkaisun ja tiedon käsittelyn periaatteista sekä soveltuvuudesta eri käyttötarkoituksiin on haluttu saada enemmän tietoa. Lisäksi esille on noussut kysymys miten sovellukset ja yritystoiminta voivat hyötyä NoSQL-paradigman soveltamisesta? NoSQL-tietokantajärjestelmien tietojenkäsittelyn ja varastoinnin toimintaperiaatteet ja ideat ovat levinneet laajalle hyvin lyhyessä ajassa. Pienempien yritysten lisäksi hyvin monet tunnetut yritykset, kuten Facebook, Ebay, IBM ovat kehittäneet NoSQL-paradigmaan perustuvia sovellusten laajennoksia sekä aivan uusia sovelluksia. Suuri osa näistä sovelluksista on julkaistu myös avoimena lähdekoodina.

6.3 NoSQL-tietokantajärjestelmien luokittelu

Tietokantajärjestelmiä joilla on NoSQL-tietokantajärjestelmien ominaispiirteitä on kehitetty paljon viime vuosina. Tietokantajärjestelmiä joita kutsutaan jollain perusteella NoSQL-tietokantajärjestelmäksi on tällä hetkellä olemassa useampia kymmeniä ellei jopa satoja [TuB11]. Näillä tietokantajärjestelmillä on yleensä tiettyjä yhteisiä perusominaisuuksia, joita käsiteltiin jo kohdassa 6.1. Kyseiset järjestelmät saattavat silti poiketa ominaisuuksiltaan ja toteutustavoiltaan hyvin paljon toisistaan.

NoSQL-tietokantajärjestelmiä ei ole luokiteltu tieteellisesti tai edes kovin täsmällisesti. NoSQL-tietokantajärjestelmien luokitteluun on käytetty useita erilaisia toisistaan vaihtelevia malleja. Seuraavassa kuvataan muutamia erilaisia tapoja luokitella näitä järjestelmiä. Luokitteluryhmät kuvataan tässä hyvin suppealla tavalla. Tarkoituksena on lähinnä tuoda esille olemassa olevia erilaisia malleja. Tässä työssä jatkossa käytettävä luokittelumalli kuvataan myöhemmissä luvuissa paljon kattavammin.

NoSQL-tietokantajärjestelmät voidaan luokitella esimerkiksi tietokantajärjestelmän noudattaman tietomallin mukaan. Usein käytetyssä luokittelumallissa NoSQL-tietokantajärjestelmät on luokiteltu neljään eri ryhmään seuraavalla tavalla [HeJ11]:

- avain-arvo-varastot (Voldemort, Redis ja Membase),

- dokumenttivarastot (Riak, MongoDB ja CouchDB),
- sarakeperhevarastot (Cassandra, HBase ja HyperTable) ja
- verkkotietokannat (Sesame, BigData, Neo4J, GraphDB ja FlockDB).

Avain-arvo-varastot (key value stores) perustuvat nimensä mukaisesti avain-arvo-tietomalliin [HeJ11]. *Dokumenttivarastot* (document stores) taas perustuvat avain-arvo-pareista koostuvista dokumenttien kokoelmista. Avain-arvo-parin avain on yksilöivä tunnus dokumentin sisällä. Sen lisäksi jokaisella dokumentilla on oma yksilöllinen tunnus, jolla dokumentti tunnistetaan. *Sarakeperhevarastojen* (column family stores) tietomallin rakennetta voidaan verrata ison matriisin tai taulukon kaltaiseen tietorakenteeseen, joka koostuu riveistä ja sarakkeista. Avain-arvo-parit varastoidaan kyseisen taulun riveille ja sarakkeisiin. Sarakeperhevarastojen tietomallissa on yleensä myös kolmas ulottuvuus, nimittäin aika. Näiden tietokantajärjestelmien tauluihin voidaan tallentaa samasta tiedosta useita eri versioita. *Verkkotietokantojen* (graph databases) tietomallia voidaan taas verrata verkkomaiseen rakenteeseen. Verkon muodostavat solmut ja niitä yhdistävät kaaret. Avain-arvo-parit on tässä tapauksessa varastoitu verkon solmuihin ja kaariin.

Stonebraker ja kumppanit [StC11] ja Rick Cattel [Cat10] luokittelevat NoSQL-tietokantajärjestelmät tietomallin mukaan merkittävimpiin ryhmiin seuraavasti:

- avain-arvo-varastot (Dynamo, Voldemort, Membase, Membrain, Scalaris, Riak, Redis ja Tokyo Tyrant),
- dokumenttivarastot (CouchDB, MongoDB, SimpleDB ja Terrastore) ja
- laajennettavat arvo -varastot (BigTable, Cassandra, HBase, HyperTable ja PNUTS).

Stonebraker ja kumppanit [StC11] kuvaavat avain-arvo-varastoja *oliokokoelmiksi* (collection of objects). Tämän kuvauksen mukaan jokaisella olioilla on avain ja arvo. Tietokantajärjestelmällä ei ole mahdollisuutta liittää arvoihin mitään erityistä semantiikkaa tai tulkita arvoja attribuuteiksi. Dokumenttivarastojen tietomalli on kuvattu oliokokoelmaksi, joka tässä kuvauksessa koostuu olioista joilla on useampia attribuutteja. Rakenne voi myös koostua sisäkkäisistä olioista. *Laajennettavat arvo-varastot* (extensible record stores) sisältävät artikkelin mukaan arvojoukkoja, joiden leveys voi vaihdella. Nämä

arvojoukot voidaan osittaa hajautetun tietokantajärjestelmän eri solmuille joko vaakasuoraan tai pystysuoraan.

Relaatiomallista poikkeavat tietokantajärjestelmät voidaan jakaa ryhmiin myös eri abstraktiotasolla. Stefan Edlich [EDL12] jaottelee nämä järjestelmät ylemmälle abstraktiotasolle *ydin NoSQL -järjestelmiksi* (core NoSQL systems) ja *soft NoSQL -järjestelmiksi* (soft NoSQL systems). Ydin NoSQL –järjestelmiä ovat Web 2.0 -sovellusten tarpeisiin kehitetyt tietokantajärjestelmät. Näitä ovat jo edellä käsitellyt avainarvo-varastot, dokumenttivarastot, sarakeperhevarastot ja verkkotietokannat. Niiden lisäksi Edlich luokittelee ydin NoSQL -järjestelmiksi tietokantajärjestelmät, joilla on useampien erilaisten tietokantajärjestelmien ominaisuuksia. Näitä hän kutsuu *multimallitietokannoiksi* (multimodel databases). Näitä ovat esimerkiksi OrientDB ja AlchemyDB. Tähän ryhmään hän luokittelee järjestelmät, joilla on esimerkiksi jokin yhdistelmä edellä mainittujen ydin NoSQL-järjestelmien ominaisuuksia. Tämän lisäksi multimallitietokannoilla voi olla relaatiotietokantojen ominaisuuksia, ja ne voivat noudattaa transaktioiden ACID-ominaisuuksia. Soft NoSQL –järjestelmiksi taas luokitellaan Edlichin mukaan muut relaatiomallista poikkeavat tietokantajärjestelmät, joita ei oltu kehitetty Web 2.0 –sovellusten tarpeisiin, kuten esimerkiksi *oliotietokannat* (object databases), kuten Db4o, Versant ja Objectivity sekä XML-tietokannat.

Kuten luvun alussa jo todettiin NoSQL-tietokantajärjestelmien luokittelutapa ei ole vielä vakiintunut. Tätä kuvaa hyvin miten Internetin eri lähteissä, kuten blogikirjoituksissa [HOF12, YEN09, EDL12] ja Wikipediassa [WIK12] on luokiteltu relaatiomallista poikkeavia tietokantajärjestelmiä erilaisiin ryhmiin, esimerkiksi tietomallin tai toimintatavan mukaan. Näitä järjestelmiä saatetaan kutsua esimerkiksi NoSQL-tietokannoiksi tai *NoSQL-toteutuksiksi* (NoSQL implementations). Usein näitä erilaisia luokitteluryhmiä ja luokittelun syitä ei ole selostettu sen tarkemmin. Taulukkoon 1 on koottu näitä edellä mainittuja järjestelmiä eri luokitteluryhmiin.

Luokitteluryhmä	**Järjestelmä**
avain-arvo-varastot	Keyspace, Flare, Schema Free, RAMCloud
avain-arvo-varastot levyllä (key-value stores on solid state or rotating disk)	BigTable, CDB, Keyspace, LevelDB, Membase, MemcacheDB, MongoDB,

	Virtuoso, Tarantool, Tokyo Cabinet, TreapDB, Tuple space
avain-arvo-välimuistit (key-value-cache)	Memcached, Virtuoso, Coherence, Redis, Hazelcast, Tuple Space, Repcached, Infinispan, EXtreme Scale, Jboss Cache, Velocity, Terracoqa
hierarkkiset avain-arvo-varastot (hierarchical key-value store)	GT.M, InterSystems Cache
järjestetyt avain-arvo-varastot (ordered key-value stores)	Berkeley DB, IBM Informix C-ISAM, InfinityDB, MemcacheDB, NDBM, Tokyo Tyrant, Lightcloud, NMDB, Luxio, Actord
lopulta oikeelliset avain-arvo-varastot (eventually consistent key-value store)	Cassandra, Dynamo, Hibari, Virtuoso, Voldemort, Riak, Dynomite, SubRecord, Mo8onDb, Dovetaildb
ylläpitopalvelut (hosted services)	Freebase, Virtuoso, Google Appengine Datastore
verkkotietokannat (graph databases)	AllegroGraph, DEX, FlockDB, InfiniteGraph, Neo4j, Virtuoso, OrientDB, Pregel, Sones GraphDB, InfoGrid, HyperGraphDB, Trinity, BrightstarDB, Bigdata
tietorakennepalvelin (data-structures server)	Redis
oliotietokannat	Db4o, GemStone/S, InterSystems Cache, JADE, NeoDatis ODB, ObjectDB, ObjectivityDB, ObjectStore, Virtuoso, Versant Object Database, Wakanda, ZODB, ZopeDB, Shoal, Starcounter, Perst, NEO
dokumenttivarastot	Virtuoso, OrientDB, SimpleDB, Terrastore, CouchDB, MongoDB, Jackrabbit,

	XML-tietokannat, ThruDB, CloudKit, Perservere, Riak, Basho, Scalaris, BaseX, Clusterpoint, Exist, Lotus Notes, Lotus Domino, MarkLocig Server, RavenDB, SisoDB
laajat sarake-varastot (wide columnar store)	BigTable, Hadoop, Hbase, Cassandra, Hypertable, KAI, OpenNeptune, Qbase, KDI, Amazon SimpleDB
taulukko-varastot (tabular store)	BigTable, Hadoop, Hbase, Hypertable, Mnesia, Virtuoso
monikko-varastot (tuple store)	Gigaspaces, Coord, Apache River, Virtuoso, Tarantool

Taulukko 1: relaatiomallista poikkeavien tietokantajärjestelmien luokittelua.

Taulukossa 1 esiintyvistä luokitteluryhmistä ja järjestelmistä huomataan, että sama järjestelmä voi olla luokiteltuna useampaan eri ryhmään. MongoDB löytyy esimerkiksi avain-arvo-varastot levyllä sekä dokumenttivarastot ryhmistä. OpenLinkin Virtuoso löytyy myös monesta eri ryhmästä. BigTable on luokiteltu avain-arvo-varastoksi levyllä, laajaksi sarake-varastoksi ja taulukko-varastoksi. Joissakin lähteissä kaikkia BigTablen kaltaisia järjestelmiä taas kutsutaan *BigTable-toteutuksiksi* (BigTable implementations) [WIK12]. Osalla näistä järjestelmistä on myös relaatiotietokantojen ominaisuuksia, kuten Virtuosolla, tai niitä ei ollut kehitetty alun perin Web 2.0 –sovellusten tarpeisiin, kuten XML-tietokannat. Tästä tutkielmasta rajataan pois nämä jo aikaisemmin Stefan Edlichin [EDL12] Soft NoSQL –järjestelmiksi määrittelemät järjestelmät, ja keskitytään vain Web 2.0 -sovellusten tarpeisiin kehitettyihin uusiin tietokantajärjestelmiin.

6.4 MapReduce-ohjelmointiparadigma

6.5 Kritiikkiä NoSQL-tietokantoja kohtaaan

7 Tietomalleja ja toteutuksia

7.1 Avain-arvo-varastot

Avain-arvo-varastojen tietomallia voidaan verrata matriisin tai hakemiston kaltaiseen tietorakenteeseen, jossa arvot varastoidaan yksilöivän avaimen taakse [HeJ11]. Näiden järjestelmien tietomalli perustuu siis yksinkertaiseen avain-arvo-malliin. Avain-arvo-varastojen arvoilla ei yleensä ole mitään tietokantajärjestelmälle näkyvää semantiikkaa. Arvot ovat myös itsenäisiä ja eristettyjä toisistaan. Arvojen mahdollisia keskinäisiä suhteita on ylläpidettävä sovellusohjemassa. Tietokannan rakenne on hyvin joustava. Eri tyyppisiä arvoja voidaan lisätä ilman suuria muutoksia olemassa olevaan tietokantaan. Avain-arvo-varastojen tietokanta onkin usein täysin skeematon. Tietorakenteen tietojen indeksointi ja hakutoiminnot perustuvat yksilöivään avaimeen. Avain-arvoja-pareja voidaan yleensä myös ryhmitellä jollakin tavalla.

7.2 Sarakeperhevarastot

Sarakeperhevarastojen (column family stores) tietomallin rakennetta voidaan verrata matriisin tai taulukon kaltaiseen tietorakenteeseen, joka koostuu riveistä ja sarakkeista [HeJ11]. Avain-arvo-parit varastoidaan kyseisen taulun riveille ja sarakkeisiin. Sarakeperhevarastojen tietomallissa on yleensä myös kolmas ulottuvuus, nimittäin aika. Näiden tietokantajärjestelmien tauluihin voidaan tallentaa samasta tiedosta useita eri versioita. Periaatteena on, että kun tietoja muokataan, siitä tallennetaan aina uusi versio. Vanha versio jää myös talteen. Nämä versiot indeksoidaan sekä erotellaan toisistaan *aikaleiman* (timestamp) perusteella. Sarakeperheiden tietomallia voidaan kutsua rivien, sarakkeiden ja aikaleimojen takia moniulotteiseksi. Tietomalli tukee hyvin tietojen osittamista. Sarakeperhevarasto soveltuu laajan mittakaavan Web 2.0 -sovelluksien käyttöön.

7.3 Dokumenttivarastot

Dokumenttivarastot (document stores) perustuvat avain-arvo-pareista koostuvista do-

kumenttien kokoelmista [HeJ11]. Avain-arvo-parin avain on yksilöivä tunnus dokumentin sisäisesti. Sen lisäksi jokaisella dokumentilla on oma yksilöllinen tunnus, jolla dokumentti tunnistetaan. Dokumenttivarastot sopivat yleensä avain-arvo-varastoja paremmin monimutkaisten tietorakenteiden käsittelyyn. Näitä tietokantajärjestelmiä käytetään usein pienten web-sivujen kävijöiden analysointi- ja seurantatoiminnoissa sekä tietojen varastona.

7.4 Verkkotietokannat

8 Vertailua ja analysointia

8.1 Tallennusratkaisujen ja tiedonkäsittelytapojen vertailua

8.2 Ongelmia, puutteita ja kehityskohteita

8.3 Käyttökohteet ja soveltuvuus

9 Yhteenveto

Lähteet

Aba10 Abadi, Daniel, The problems with ACID, and how to fix them without going NoSQL, 31.8.2010. http://dbmsmusings.blogspot.com/2010/08/problems-with-acid-and-how-to-fix-them.html. [20.11.2011]

Ama12 Amazon.com, Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more. http://www.amazon.com. [26.3.2012]

BaF95 Baru, Chaitanya, Fecteau, Gilles, An overview of DB2 parallel edition. Proceedings of the 1995 ACM SIGMOD international conference on Management of data (SIGMOD ’95), San Jose, California, Yhdysvallat, sivut 460-462.

BDH03 Barroso, Andre, Luiz, Dean, Jeffrey, Hölzle, Urs, Web search for a planet: the Google cluster architecture. IEEE Micro, 23, 2 (2003), sivut 22-28.

Bre00 Brewer, Eric, Towards robust distributed systems. Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing (PODC ’00), Portland, Oregon, Yhdysvallat, 2000, sivu 7. [Myös:http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf].

Cat10 Cattell, Rick, Scalable SQL and NoSQL data stores. ACM SIGMOD Record, 39, 4 (December 2010), sivut 12-27.

CDG06 Chang, Fay, Dean, Jeffrey, Ghemawat, Sanjay, Hsieh, Wilson, Wallach, Deborah A., Burrows, Michael, Chandra, Tushar, Fikes, Andrew, Gruber, Robert, Bigtable: a distributed storage system for structured data. Proceedings of the 7th symposium on Operating Systems Design and Implementation (OSDI'06), Seattle, Yhdysvallat, 2006, sivut 205–218.

CDG08 Chang, Fay, Dean, Jeffrey, Ghemawat, Sanjay, Hsieh, Wilson, Wallach, Deborah A., Burrows, Michael, Chandra, Tushar, Fikes, Andrew, Gruber, Robert, Bigtable: a distributed storage system for structured data. ACM Transactions on Computer Systems, 26, 2 (2008), sivut 4:1-4:26.

DeG08 Dean, Jeffrey, Ghemawat, Sanjay, MapReduce: simplified data processingon large clusters. Communications of the ACM – 50th anniversary issue: 1958 – 2008, 51, 1 (2008), sivut 107-113.

DHJ07 DeCandia, Giuseppe, Hastorun, Deniz, Jampani, Madan, Kakulapati, Gunavardhan, Lakshman, Avinash, Pilchin, Alex, Sivasubramanian, Swaminathan, Vosshall, Peter, Vogels, Werner, Dynamo: amazon's highly available key-value store. Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles (SOSP ’07), Stevenson, Washington, Yhdysvallat, 2007, sivut 205-220.

EDL12 Edlich, Stefan, NoSQL, your ultimate guide to the non - relational universe!. http://nosql-database.org/. [24.3.2012]

Eva09 Evans, Eric A., NOSQL 2009, 12.5.2009. http://blog.sym-link.com/2009/05/12/nosql_2009.html. [4.3.2012]

Fac12 Facebook Inc., Facebook. http://fi-fi.facebook.com. [26.3.2012]

Goo12a Google, Google+: reaaliaikaisen sisällön jakamisen vallankumous verkossa. https://plus.google.com. [26.3.2012]

Goo12b Google, Google earth. http://www.google.com/intl/fi/earth/index.html. [26.3.2012]

Goo12c Google, Google suomi. http://www.google.fi. [26.3.2012]

Goo12d Google, YouTube - Broadcast Yourself. http://www.youtube.com. [26.3.2012]

HeJ11 Hecht, Robin, Jablonski, Stefan, NoSQL evaluation: A use case oriented survey. International Conference on Cloud and Service Computing (CSC ’11), Hong Kong, Kiina, 2011, sivut 336-341.

HOF09 Hoff, Todd, A Yes For A NoSQL Taxonomy. http://highscalability.com/blog/2009/11/5/a-yes-for-a-nosql-taxonomy.html, 5.11.2009. [24.3.2012]

IMS10 Ishizaki, Kazuaki, Mizuno, Ken, Suganuma, Toshio, Silva, Daniel, Koseki, Akira, Komatsu, Hideaki, Ueda, Yohei, Nakatani, Toshio, Parallel programming framework for large batch transaction processing on scale-out systems. Proceedings of the 3rd Annual Haifa Experimental Systems Conference (SYSTOR ’10), Haifa, Israel, 2010, sivut 15:1-15:14.

Lea10 Leavitt, Neal, Will NoSQL databases live up to their promise?. Computer, 43, 2 (February 2010), sivut 12-14.

Ler10 Lerner, Reuven M., At the forge: NoSQL? I'd prefer SomeSQL. Linux Journal, 2010, 192 (April 2010), artikkeli nro 5.

Mur07 Murugesan, San. Understanding Web 2.0. IT Professional, 9, 4 (July-August 2007), sivut 34-41.

NOS11 NoSQL Now! 2011, San Jose, Yhdysvallat, 2011. http://nosql2011.wilshireconferences.com. [24.3.2012]

Pat99 Paterno, Giuseppe, NoSQL Tutorial: A comprehensive look at the NoSQL database. Linux Journal, 1999, 67es (November 1999), artikkeli nro 23.

Pos12 PostgreSQL Global Development Group, PostgreSQL. http://www.postgresql.org. [26.3.2012]

Pri08 Pritchett, Dan, BASE: An acid alternative. Queue - Object-Relational Mapping, 6, 3 (May, 2008), sivut 48-55.

Ora12a Oracle Corporation, MySQL: The world's most popular open source database. http://www.mysql.com. [26.3.2012]

Ora12b Oracle Corporation, Oracle real application clusters. http://www.oracle.com/us/products/database/options/real-application-clusters/index.html. [26.3.2012]

Rah93 Rahm, Erhard, Parallel query processing in shared disk database systems. ACM SIGMOD Record, 22, 4 (December 1993), sivut 32-37.

SDE11 Seoul Data Engineering Camp (SDEC ’11), Soul, Etelä-Korea, 2011. http://www.sdec.kr. [24.3.2012]

SKS11 Silberschatz, Abraham, Korth, Henry F., Sudarshan, S., *Database System Concepts*, *Sixth Edition*, International Edition 2011, McGraw-Hill, 2011.

SMA07 Stonebraker, Michael, Madden, Samuel, Abadi, Daniel J., Harizopoulos, Stavros, Hachem, Nabil, Helland, Pat, The end of an architectural era: (it's time for a complete rewrite). Proceedings of the 33rd international conference on Very large data bases (VLDB ’07), Wien, Itävalta, 2007, sivut 1150-1160.

StC11 Stonebraker, Michael, Cattell, Rick, 10 rules for scalable performance in

'simple operation' datastores. Communications of the ACM, 54, 6 (June 2011), sivut 72-80.

Sto10 Stonebraker, Michael, In search of database consistency. Communications of the ACM, 53, 10 (October 2010), sivut 8-9.

Str12 Strozzi, Carlo, NoSQL A Relational Database Management System. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page. [4.3.2012]

Tiw11 Tiwari, Shashank, *Professional NoSQL*, Wiley, 2011.

TuB11 Tudorica, Bodgan George, Bucur, Cristian, A comparison between several NoSQL databases with comments and notes. Roedunet International Conference (RoEduNet ‘11), Iasi, Romania, 2011, sivut 1-5.

Vaj09 Vajgel, Peter, Needle in a haystack: efficient storage of billions of photos. http://www.facebook.com/note.php?note_id=76191543919. [18.3.2012]

VMZ10 Vicknair, Chad, Macias, Michael, Zhao, Zhendong, Nan, Xiaofei, Chen, Yixin, Wilkins, Dawn, A comparison of a graph database and a relational database: a data provenance perspective. Proceedings of the 48th Annual Southeast Regional Conference (ACM SE '10), Oxford, Yhdysvallat, 2010, artikkeli nro 42.

WIK12 Wikipedia, NoSQL. http://en.wikipedia.org/wiki/NoSQL. [24.3.2012]

Yah12 Yahoo! Inc., Welcome to Flickr - photo sharing. http://www.flickr.com. [26.3.2012]

YEN09 Yen, Stephen, NoSQL is a horseless carriage. http://dl.getdropbox.com/u/2075876/nosql-steve-yen.pdf. [24.3.2012]