Niko Heikkilä

NOSQL JA HAJAUTETUT JÄRJESTELMÄT

Opinnäytetyö
KESKI-POHJANMAAN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Kesäkuu 2012

KESKI-POHJANMAAN AMMATTIKORKEAKOULU
MELLERSTA ÖSTERBOTTENS YRKESHÖGSKOLA

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Yksikkö	**Aika**	**Tekijä**
Tekniikan ja liiketalouden yksikkö, Kokkola-Pietarsaari	Kesäkuu 2012	Niko Heikkilä
Koulutusohjelma Tietotekniikan koulutusohjelma		
Työn nimi NoSQL ja hajautetut järjestelmät		
Työn ohjaaja DI Sakari Männistö		**Sivumäärä** 29
Työelämäohjaaja DI Sakari Männistö		

Opinnäytetyössä tutkittiin dokumenttipohjaisen tietokannan ja relaatiotietokannan eroja www-sovellusten kehityksessä. Tutkimuksen tarkoituksena oli selvittää, onko dokumenttipohjainen NoSQL tuotteena kannattava valinta perinteisempien SQL-toteutusten tueksi.

Tutkimuksessa viitataan hajautettuun tietotekniikkaan, joka luo pohjan toimintavarmoille ja tehokkaille sovelluksille niin työpöytäympäristössä kuin tietoverkoissakin. NoSQL noudattaa hajautetun tietotekniikan periaatteita tarjoamalla vikasietoisen, nopeasti skaalautuvan ja helposti monistettavan tietokannan usealle laitteelle.

NoSQL:n tuote-esimerkiksi valittiin avoimen lähdekoodin Apache CouchDB -tietokantajärjestelmä pääasiassa selkeytensä ja runsaiden toiminnallisuuksiensa vuoksi. CouchDB tarjoaa sovellusrajapinnan, jonka avulla on helppo luoda HTML5-pohjaisia verkkosovelluksia, joiden kriittinen tieto on jatkuvasti turvattuna ja helposti saatavilla.

NoSQL ei kuitenkaan ole ratkaisu kaikkeen, vaan järjestelmän tietojen viitatessa selkeästi toisiinsa on edelleen varminta käyttää relaatiopohjaisia tietokantoja.

Asiasanat

CouchDB, hajautettu tietotekniikka, HTML5, NoSQL, relaatiotietokannat, replikointi, tietomallit, vikasietoisuus

CENTRAL OSTROBOTHNIA
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Unit	Date	Author
Technology and Business, Kokkola-Pietarsaari	June 2012	Niko Heikkilä

Degree programme	
Information Technology	
Name of thesis	
NoSQL and decentralised systems	
Instructor	**Pages**
Sakari Männistö	29
Supervisor	
Sakari Männistö	

This thesis investigates the differences between relational databases and document-oriented databases. The goal of this investigation was to find out if the document-oriented NoSQL is a profitable solution for supporting traditional SQL databases.

This thesis contains references to the theory of decentralized computing which establishes a base for fault-tolerant and efficient applications in both desktop and network environments. NoSQL follows the principles of decentralized computing by offering distributed, scalable and replicable solution between devices.

Apache CouchDB was chosen as a practical example of NoSQL due to its user-friendliness and rich functionalities. It is open source and offers a powerful application programming interface for developing complex HTML5 based web applications whose critical data remains safely stored and easily accessible.

However, the NoSQL should not be considered to be the solution for each problem. If the data contains multiple references to other data it is better to use relational databases.

Key words

CouchDB, databases, decentralised computing, HTML5, NoSQL, replication, web applications

KÄSITTEIDEN MÄÄRITTELY

CouchDB
- Apachen kehittämä avoimeen lähdekoodiin pohjautuva tietokantajärjestelmä, joka painottaa tiedonhallinnan loogisuutta

hajautettu tietotekniikka
- tietojärjestelmän resurssien jakaminen useampien solmukohtien kesken, joka parantaa vikasietoisuutta ja saavutettavuutta

HTML5
- HyperText Markup Language -kielen uusin standardi, joka toimii WWW-selainten pääasiallisena esitystapana

NoSQL
- hajautettu tietokanta-arkkitehtuuri, jossa tiedon rakenteella on vähäisempi merkitys kuin SQL:ssä ja tietomalli perustuu yksilöityihin dokumentteihin

relaatiotietokannat
- perinteinen rakenteellinen tietokanta-arkkitehtuuri (SQL), jossa tietokannan sisällä voidaan viitata toiseen tietoon avainten avulla

replikointi
- NoSQL-pohjaisen tietokannan metodi, jolla yhden tietokannan sisältö monistetaan kopioksi joko samalle laitteelle tai verkon yli toiselle laitteelle

tietomallit
- abstrakteja malleja, jonka mukaan tietokannan sisältämä tieto on jäsennelty

1 JOHDANTO

Ohjelmistotuotannon suuntaus on varsinkin kuluneilla vuosikymmenillä siirtynyt yhä enemmän kohti verkkosovelluksia. Valtaosa suurten yhtiöiden, kuten Googlen ja Amazonin, palveluista sijaitsee nykyisin verkossa, jossa ne ovat käyttäjien saatavilla sijainnista riippumatta. Oman lukunsa muodostavat sosiaaliset yhteisöpalvelut, kuten Facebook ja Twitter, jotka eivät tarjoa käyttöön perinteisiä työpöytäsovelluksia, vaan palveluita käytetään pääasiassa selaimen kautta.

Verkkosovellusten kehittäminen edellyttää suunnittelijoilta laajaa ja vastuullista tiedonhallinnan osaamista. Verkkoon säilötyn tiedon hyödyntämisessä ja sovellusten vuorovaikutteisessa käytössä kynnyskysymyksiksi nousevat tietoturvan lisäksi laajennettavuus, luotettavuus ja saavutettavuus. Tiedon pitää olla verkossa riittävästi suojattuna, jonka lisäksi on varauduttava tiedon jatkuviin muutoksiin, ja pääsy tietoon ei saa estyä esimerkiksi järjestelmävirheen vuoksi. Nykyaikaiset verkkosovellukset voivat sisältää esimerkiksi tiedonhallintaan liittyviä vaatimuksia, joiden ratkaiseminen perinteisillä tietokantaratkaisuilla on vaikeaa. Tässä työssä tarkastellaan sitä, voisiko yleisesti käytössä olevan relaatiotietomallin tukena käyttää jotain muuta ratkaisua tilanteissa, joissa perinteiset menetelmät koetaan rajoittaviksi.

Työssä lähestytään aihepiiriä teorialähtöisesti, mutta käytännön esimerkkien tukemana. Tekstin seassa esiintyy niin havainnollistavia kuvia kuin koodiesimerkkejäkin. Lukijalta ei vaadita kattavaa ohjelmistoteknistä osaamista, mutta perehtyneisyys tietokantoihin ja -järjestelmiin on suositeltavaa. Opinnäytetyön toisessa luvussa esitellään ei-relaatiomallisen tietokannan - NoSQL:n - ominaisuudet ja käyttökohteet. Kolmannessa luvussa perehdytään tarkemmin avoimeen lähdekoodiin perustuvaan Apache CouchDB -moduuliin. Neljännessä luvussa esitellään tarkemmin, miten sovelluskehitys tapahtuu CouchDB:n avulla, sekä tutustutaan käyttäjien tarinoihin. Viimeisessä luvussa pohditaan työn onnistumista.

2 NOSQL

Tavanomaiset tiedonhallintaratkaisut pohjautuvat usein relaatiomallisiin tietokantoihin, jotka ovat tunnettujen ohjelmistoyhtiöiden, kuten Microsoftin, Oraclen tai Sun Microsystemsin toimittamia. Nopealla aikataululla toteuttavien projektien tarjouksia kilpailutettaessa voi usein jäädä tarkastamatta kaikki tarjolla olevat ratkaisuvaihtoehdot. Väärät ratkaisut kostautuvat asiakkaalle siten, että jälkeenpäin ilmenee uusia ongelmia. Kehitysvaiheessa ilmenevät ongelmat viivästyttävät projektin valmistumista, koska ongelmaa voi joutua lähestymään toisesta näkökulmasta.

Relaatiomallin mukaiset tietokantatuotteet eivät ole ainoa ja kaikkiin käyttötarkoituksiin sopiva vaihtoehto. Uudentyyppisissä verkkosovelluksissa käyttäjämäärät voivat kasvaa hyvin suuriksi. Tieto voi olla myös monimuotoisempaa kuin tyypillisissä yritysjärjestelmissä, joissa relaatiomalliin pohjautuvat tietokannat ovat vallitseva ratkaisu. Tällaisissa suurta skaalautuvuutta vaativissa ja monimuotoista dataa käsittelevissä järjestelmissä dokumenttipohjainen ei-relaatiomalliin pohjautuva tietokanta on lupaava vaihtoehto. Nopeasti suosiotaan kasvattanut dokumenttipohjainen tietokantamalli - NoSQL - tarjoaa hyvän vaihtoehdon relaatiotietokannoille silloin, kun järjestelmien käsittelemän tiedon rakenteelle ei ole asetettu tiukkoja vaatimuksia. (Couchbase 2012.)

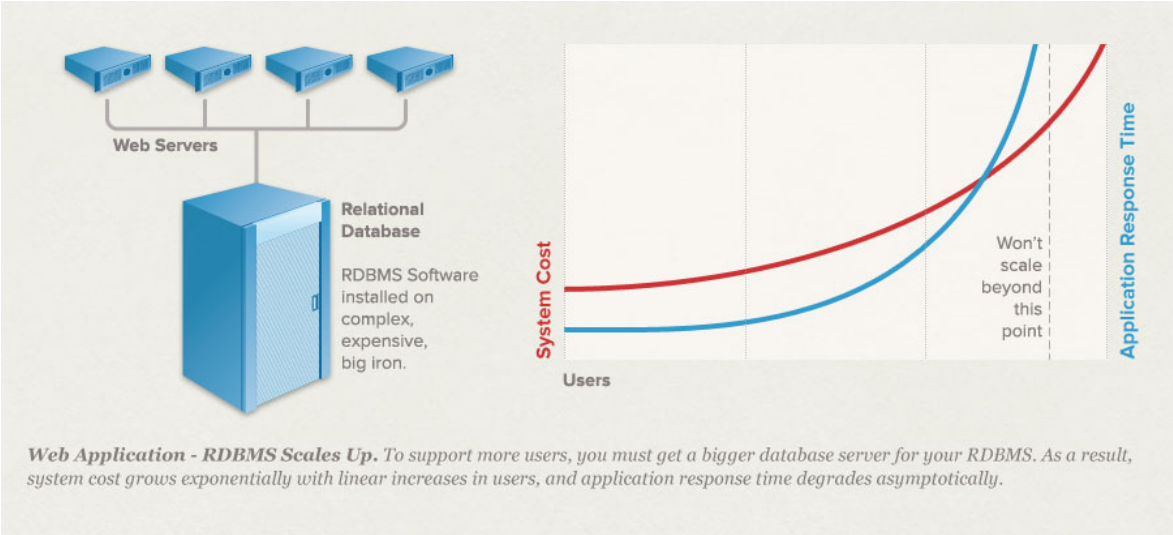
Tiedon määrä internetissä on kasvanut viimeisten vuosien aikana valtavasti. IDC-markkinatutkimusyrityksen tekemän tutkimuksen mukaan vuoden 2011 loppuun mennessä tietoverkoissa oli tallennettuna jo 1,8 zettatavua (1 800 000 000 teratavua). Määrä on 10-kertainen vuoden 2006 lukemaan verrattuna. Nykyään markkinoilla olevissa työpöytäkoneissa vapaata levytilaa on usein yksi teratavu tai enempi. Suurin syy räjähdysmäiseen kasvuun on trendi, jonka mukaan kaikki tallennetaan digitaaliseen muotoon ja usein muutamina eri kopioina. Myös sosiaalisten verkostojen käsittelemään tietoon, kuten valokuviin, videoleikkeisiin ja käyttäjäprofiileihin voidaan viitata suuressa kokoluokassa, arviolta muutamissa sadoissa eksatavuissa. (Daily Tech 2008.)

Suuresta määrästä tietoa, joka sijaitsee verkossa, käytetään nimitystä *Big Data*. IBM:n mukaan arviolta 90% tietoverkkojen käsittelemästä tiedosta on syntynyt viimeisen kahden vuoden aikana. Big Data -teknologiat edustavat alalla uutta sukupolvea, joka on suunniteltu käsittelemään erittäin suurissa tietovarannoissa sijaitsevaa monimuotoista tietoa. Tämän ansiosta mm. modernit pilvijärjestelmät ovat saaneet suosiota, koska ne tarjoavat skaalautuvuutta ja laskentatehoa kustannustehokkaasti. (IBM 2012., Gantz & Reinsel 2011.)

2.1 Skaalautuminen

Tavalliseen relaatiotietomalliin pohjautuvat tietokannat skaalautuvat pystysuunnassa (*vertical scaling*). Kun tarvitaan lisää kapasiteettia joko levytilan tai tehokkuuden suhteen, yleisin ratkaisu on ostaa isompi fyysinen palvelin ja asentaa se laitetilaan. Järjestelmä siis mitoitetaan aina tarpeiden mukaisesti suuremmaksi. Tällainen toimintamalli ei välttämättä sovi pienten ja keskisuurien yritysten budjetteihin laitekustannusten vuoksi. Usein voidaan säästää kustannuksissa hyödyntämällä fyysisten palvelimien sijaan pilvipalveluita. (Couchbase 2012.)

Eräs ratkaisu on investoida virtuaali- ja pilvipalvelimiin. Ne toimivat kevyemmällä kapasiteetilla eivätkä kuormita laitetilaa samalla tavalla kuin tavalliset palvelimet. Palveluiden kapasiteettia voidaan nostaa tai laskea tarpeen tullen. Kuviossa 1 esitetään relaatiomallia käyttävän tietojärjestelmän kustannukset ja järjestelmän vasteaika käyttäjämäärän funktiona. Vaakasuunnassa skaalautuva malli (*horizontal scaling*) on ollut pitkään käytössä sovelluskehityksen parissa, mutta vasta nyt tietokantateknologia on tulossa mukaan. (Couchbase 2011.)



KUVIO 1. Relaatiomallia käyttävän tietojärjestelmän kustannukset ja järjestelmän vasteaika käyttäjämäärän funktiona (Couchbase 2011.)

2.2 Relaatiomalli

Perinteisessä relaatiomallissa tiedon rakenne noudattaa nk. skeemaa eli tiettyä kaavaa, jonka pohjalle tieto rakennetaan. Jokainen otos tietokannasta sisältää arvoja tietystä määrästä sarakkeita. Sarakkeet koostuvat tietotyypeistä, joilla jokaisella on oma tarkoituksensa. Relaatiomalli on pysynyt vallassa jo useiden vuosikymmenien ajan. (Couchbase 2011.)

Relaatiomallissa tiedon normaalimuoto on tärkeää. Normalisoinnissa suurempia kokonaisuuksia sisältävät taulut pilkotaan pienempiin tauluihin. Pienemmät taulut viittaavat toisiinsa avaimilla. Esimerkiksi taulu, johon varastoidaan järjestelmän virhelokista kerätty tieto, sisältää joka rivillä uniikin virhekoodin, virheen tapahtuma-ajan sekä järjestelmän, jossa virhe syntyi. Sen sijaan, että järjestelmien kriittistä tietoa (fyysinen sijainti, IP-osoite) toistettaisiin joka rivillä, voidaan oikealla avaimella viitata sellaisen taulun riviin, joka sisältää jo tiedot oikeasta järjestelmästä. (Couchbase 2011.)

Koska osa relaatiotietokantojen tiedosta on jaettu useampien taulujen kesken, ei kahdenkertaista tietoa synny kantaan. Varjopuolena yksittäiseen riviin tehdyt muutokset lukittavat

useita tietokannan tauluja ristiriitaiselta tiedolta suojautumiseksi. ACID (*Atomicity, Consistency, Isolation, Durability*) -mallin mukaiset operaatiot asettavat haasteita luku- ja kirjoitusoperaatioiden suorituskyvyssä sellaisissa relaatiotietokannoissa, jotka on jaettu useille eri palvelimille. Täten relaatiomalliin pohjautuvaa tietokantaa on vaikea toteuttaa hajautetussa järjestelmässä. (Leavitt 2010.)

2.3 Dokumenttipohjainen tietomalli

Ei-relaatiomallin perustana on dokumenttipohjainen tietomalli. Tällä tarkoitetaan tietokantaa, joka relaatiomallista poiketen sisältää joukon erilaisia havainnollistavassa muodossa olevia tiedostoja. Havainnollistavasta muodosta suosituimpia ovat Javascript Object Notation (JSON)-, HTML- sekä Extensive Markup Language (XML) -formaatit. Tässä asiayhteydessä näistä tiedostoista käytetään nimitystä dokumentti. Seuraavassa esitetään esimerkit JSON- ja XML-syntaksista.

```
{
  "ID": 1,
  "PID": 4580,
  "ERR_MSG": "Out of memory",
  "TIMESTAMP": "2011-01-01T23:59:58:57",
  "SERVER_ADDR": "172.16.48.4",
  "LOCATION": "Lab 2"
},
{
  "ID": 2,
  "PID": 6245,
  "ERR_MSG": "ECC Error",
  "TIMESTAMP": "2011-01-01T23:59:59:20",
  "SERVER_ADDR": "172.16.48.4",
  "LOCATION": "Lab 2"
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<log>
  <error id="1">
        <pid>4580</pid>
        <message>Out of memory</message>
        <time>2011-01-01T23:59:58:57</time>
        <server>172.16.48.4</server>
        <location>Lab 2</location>
  </error>
  <error id="2">
        <pid>6245</pid>
        <message>ECC Error</message>
```

```
        <time>2011-01-01T23:59:59:20</time>
        <server>172.16.48.4</server>
        <location>Lab 2</location>
  </error>
</log>
```

JSON-syntaksissa jokainen kaarisulkeiden väliin jäävä osa on oma kokonaisuutensa eikä riipu muusta tiedosta. Sen vuoksi tietoa on helppo kopioida palvelimelta toiselle, koska ei tarvitse huolehtia, että kriittistä tietoa jää vanhalle palvelimelle. Samalla luku- ja kirjoitusnopeus pysyy siedettävänä, koska tieto on vain yhdessä dokumentissa usean eri taulun sijaan. (Couchbase 2011.)

Myös dokumenttipohjaisessa tietomallissa tieto voidaan normalisoinnin tapaan hajauttaa useampiin osiin. Esimerkiksi edellä olevan virhelokin sisältämiä palvelinten IP-osoitteita ja sijaintitietoja voitaisiin säilöä omissa dokumenteissa, johon tästä viitataan. Sama pätee sovellustietoihin ja virhekuvauksiin. Täten tietoa muokattaessa vältyttäisiin työläältä useiden dokumenttien uudelleenkirjoitukselta. (Couchbase 2011.)

2.4 Mallintaminen

Dokumentit ovat keskeinen osa NoSQL:n rakennetta. Esimerkiksi blogijärjestelmässä dokumentteja voi mallintaa artikkeleilla, kommenteilla sekä sivuston staattisella esittelysivulla. Jokaisen artikkelin, kommentin ja sivun tieto on tavallisesti omassa dokumentissaan. Näin ollen dokumentit muistuttavat siis hyvin paljon MVC-arkkitehtuurin malleja (*model*). Mallit sisältävät sovelluksen liiketoimintalogiikan (*business logic*), jonka vastuulla on käytännön työ sovelluksen taustalla. (Couchbase 2011.)

Tärkeä näkökulma mallintamisessa on dokumenttipohjaisen tietomallin parempi semantiikka eli tiedon merkitsevyys. Esimerkiksi yrityksen edustajan käyntikortissa on tavallisesti edustajan nimi, puhelinnumero, sähköpostiosoite ja kuva. Kaikista edustajista ei kuitenkaan ole kuvaa henkilöstön tietokannassa, jolloin se jätetään pois käyntikortista. Sen sijaan, että henkilön kuvan paikalla olisi leikekuva kasvonpiirteistä, voidaan kuva-alue jättää tyhjäksi ja pitää asettelu muuten samanlaisena. Mikäli edustaja ei halua luovuttaa sähköpos-

tiaan julkisuuteen, jätetään maininta koko sähköpostista pois käyntikortista sen sijaan, että kortissa lukisi “Sähköposti: tyhjä“. Käyntikorttia voi siis ajatella omana dokumenttina. Sovelluskehityksessä tällainen mallintaminen antaa suunnittelijalle enempi ilmaisunvaraa, koska ei ole mitään tiettyä skeemaa, jota noudattaa. (Couchbase 2011.)

Relaatiotietomalli kääntyy myös vaivattomasti dokumenttipohjaiseksi. Suunnitteluvaiheessa voidaan tieto koostaa taulujen ja rivien sijasta dokumentteihin ja antaa kullekin dokumentille oma yksilöity tunniste. Myöhemmin esille tuleva Couchbase tekee juuri näin. (Couchbase 2011.)

2.5 Tiedon referointi

NoSQL-tietokannoissa yksittäisiin dokumentteihin viitataan yksilöllisellä pääavaimella (*UUID*), josta tässä asiayhteydessä käytetään nimitystä tunniste. Tunnisteet eivät suuresti poikkea relaatiomallissa esiintyvistä taulujen pääavaimista (*primary keys*). Dokumenttipohjaisessa tietomallissa tunnisteita on yhtä paljon kuin on dokumentteja. Suorituskyvyn parantamiseksi joissain NoSQL-tietokannoissa dokumentit lajitellaan tunnisteiden perusteella siten, että usein luetut tiedon osaset sijaitsevat lähellä toisiaan. Tämä pienentää järjestelmän hakuaikaa ja siten nopeuttaa sovelluksen toimintaa. (Couchbase 2011.)

Kuten moni muukin asia NoSQL:ssä, myös tunnisteiden syntaksi on vapaa: dokumentit voidaan esimerkiksi merkitä käyttämällä kokonaisluvuista poikkeavia tunnisteita, kuten (*user:com:example:123*). Syntaksi on käytössä Google Play -verkkosovelluskaupassa, jossa Google Plus -mobiilisovellukseen viitataan tunnisteella com.google.android.apps.plus. Esimerkistä käy ilmi sovelluksen valmistaja, alusta, kategoria sekä lyhyt nimi, jotka voidaan ohjelmatasolla purkaa helppokäyttöiseen muotoon. CouchDB:n käyttämät dokumenttien tunnisteet ovat hajautusalgoritmeilla laskettuja useita merkkejä pitkiä tarkistussummia (esim. *6e1295ed6c29495e54cc05947f18c8af*). Todennäköisyys, että tällaisella tunnisteella varustettu dokumentti menee tietokannassa päällekkäin toisen kanssa, on häviävän pieni. (Anderson, Lehnardt & Slater 2010.)

Seuraavassa on esimerkki Python-skriptistä, joka lukee tunnisteen ja pilkkoo sen annetun erottimen (piste) perusteella palauttaen lopuksi listan. Pythonissa listaa voidaan käsitellä kuin taulukkoa.

```
>>> id = "com.google.android.apps.plus"
>>> list = id.split(".")
>>> list
['com', 'google', 'android', 'apps', 'plus']
```

Näin ollen tiedon referointi tapahtuu tunnisteiden avulla, mikä helpottaa siirtymistä relaatiotietomallista pois, koska ne muistuttavat paljon relaatiomallisen tietokannan pääavaimia. NoSQL:n ja relaatiotietokantojen suurin eroavaisuus tulee tässä vastaan. Mikäli sovellus vaatii selkeää useiden dokumenttien yhteen liitettävyyttä, on perinteinen relaatiotietomalli parempi ratkaisu. Mitä vähemmän yhteen liitettävyyttä tietokanta tarvitsee, sitä enemmän NoSQL:n tuomat hyödyt korostuvat. (Couchbase 2011.)

2.6 Tiedon hajautuminen ja muokkaaminen

Dokumenttipohjainen tietomalli perustuu hajautettuun tietoon. Tietoa haettaessa on tyypillistä, että sama avainpari voi löytyä useammasta eri dokumentista, mikä vaikeuttaa tiedon muokkaamista ja synkronointia. Esimerkiksi valokuvagalleriassa jokaisella kuvalla on otsake ja yksittäinen valokuva voi esiintyä useassa eri kansiossa, uutisvirrassa tai sosiaalisessa ryhmässä monen muun sijainnin lisäksi. Jokaista kopiota kuvasta voi mallintaa tietokannassa omalla dokumentilla, mutta silloin jokainen dokumentti on päivitettävä erikseen. Satoja kuvia sisältävistä kansioista jokaisen kuvan tietojen läpikäyminen ja muuttaminen on työlästä. (Couchbase 2011.)

Ratkaisu on hajauttaa tietokantaa entisestään laittamalla kuvien sijaintitiedot omaan dokumenttiinsa, johon muut dokumentit viittaavat. Päivittämällä vain sijaintitiedot sisältävää dokumenttia pysyy tieto hallinnassa. Ainoa ero entiseen on, että kuvia hakiessa on tiedot haettava useammasta eri dokumentista, mikä ei nykyisten järjestelmien suorituskyvyllä ole ongelma. (Couchbase 2011.)

Siinä missä relaatiomallissa pyritään tietoa normalisoimaan niin paljon kuin mahdollista, NoSQL:ssä pyritään tiedon täydelliseen hajautukseen. Kumpikaan näistä ei kuitenkaan ole ratkaisu kaikkeen, vaan on korostettava sovelluksen analyysivaiheessa rajattavia tarpeita etenkin tiedonhallinnan osalta. (Couchbase 2011.)

2.7 Tiedon rinnakkaisuus

Koska sovellus sijaitsee verkossa, on hyvin todennäköistä, että sillä on useita käyttäjiä, jotka hakevat ja muokkaavat tietoa samanaikaisesti. Verkkomedioissa ja blogeissa työryhmät lisäävät uusia artikkeleita, mutta harvemmin useampi kuin yksi henkilö kirjoittaa samaa artikkelia. Sivuston vierailijat sen sijaan voivat jättää artikkeleihin kommentteja, joita sekä kirjoittaja itse että sivuston ylläpitäjä voivat muokata. Tieto voi siis mennä päällekkäin eri käyttäjien muokatessa samaa dokumenttia, jolloin on pyrittävä mahdollisimman sujuvaan tiedon rinnakkaisuuteen. (Couchbase 2011.)

Relaatiotietomallin taulu on yksittäinen tietorakenne. Ennen kuin käyttäjä muokkaa tietoa, järjestelmän on varmistettava, ettei kukaan muu ole tekemässä samaa. Tämän varmistaakseen järjestelmä menee lukkoon ensimmäisen käyttäjän toimesta. Lukko avataan väliaikaisesti seuraavalle käyttäjälle, jolloin sitä seuraava käyttäjä odottaa vuoroaan. Eri tuotteissa tietokannan lukitusta hyödynnetään eri tavoin, ja niiden edut ja haitat riippuvat käyttökohteesta. CouchDB:ssä järjestelmän lukituksen sijaan hyödynnetään versionhallintaa. Kun käyttäjä päivittää dokumentin sisältöä, uusi tieto ei ylikirjoita vanhaa vaan järjestelmään tallentuu kokonaan uusi dokumentti, mutta uudella versiotunnisteella merkittynä. Versiotunnisteen syntaksi on seuraava. (Anderson, Lehnardt & Slater 2010.)

{versionumero}-{version tarkistussumma}
2-de0ea16f8621cbac506d23a0fbbde08a

Tietokannan mentyä lukkoon eri tuotteilla on eri algoritmeja, joiden avulla lasketaan voittava eli tietokantaan jäävä versio. CouchDB vertaa merkkijonojen pituuksia, jolloin pitemmän tunnisteen sisältävä dokumentti voittaa. (Anderson, Lehnardt & Slater 2010.)

2.8 Päätelmät

Kuka hyötyy NoSQL:stä ja miten? Seuraavassa on pääkohdat edellisistä kappaleista.

Dokumenttipohjaisen tietomallin mukainen tietokanta skaalautuu tavallisten palvelinten, virtuaalikoneiden sekä pilvipalveluiden välillä vaivattomasti. Tiedon mallintaminen ei vaadi tiedolta ehdotonta rakennetta, vaan tietoa voidaan lisätä ja poistaa vapaasti. Tämä on hyödyllistä etenkin järjestelmissä, joissa tietorakenteet muuttuvat usein. Joustava mallinnustekniikka sallii monimutkaisenkin tiedon esittämisen, hakemisen ja muokkaamisen. (Couchbase 2011.)

Tiedon hajautus vaatii normaalia järjestelmää enempi levytilaa, mutta vastineeksi järjestelmä hyötyy lisääntyneestä suorituskyvystä, skaalautuvuudesta ja joustavuudesta. Edellä mainitut asiat tekevät dokumenttipohjaisesta tietomallista ja NoSQL:stä varteenotettavan ehdokkaan mille tahansa verkkosovellukselle. (Couchbase 2011.)

3 APACHE COUCHDB

CouchDB on tällä hetkellä MongoDB:n ja Cassandran ohella suosituin NoSQL-tuote. CouchDB:tä hyödyntävät mm. AOL, BMW, Cisco sekä Honda Motors, jotka ovat havainneet NoSQL:n tuomat edut tavanomaisten tietokantatuotteiden rinnalla. CouchDB-moduulin lisäksi Couchbasen tuotteisiin kuuluu Membase- ja Memcached-järjestelmät. Näitä ei tässä työssä käsitellä. (Couchbase 2012.)

CouchDB on puhtaasti www-käyttöä varten suunniteltu tietokanta. Tieto kirjataan järjestelmään JSON-muotoisena, sitä haetaan ja muokataan HTTP-rajapinnan yli selaimella ja koostetaan erilaisia näkymiä Javascriptia hyödyntäen. Inkrementaalisen replikoinnin ansiosta varmuuskopioiden ottaminen on helppoa ja tiedoista säilyy omat versionsa loogisessa järjestyksessä. Myös usean käyttäjän yhtäaikainen muokkaaminen on automaattisen konfliktien selvittelyn ansiosta turvallista. Edellä mainittuihin ominaisuuksin perehdytään tarkemmin seuraavissa luvuissa. (Anderson, Lehnardt & Slater 2010.)

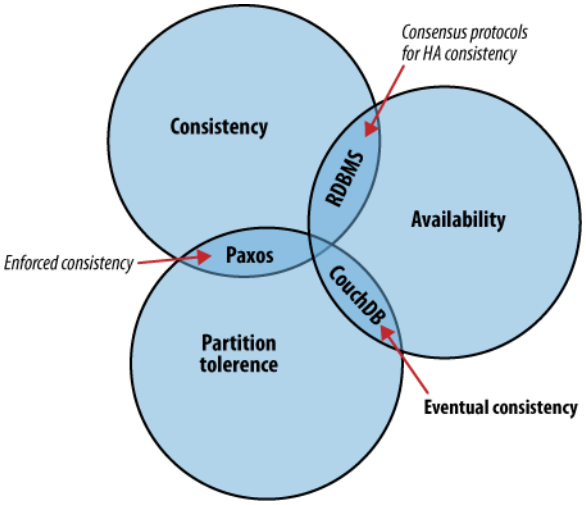
3.1 Hajautetut järjestelmät

Ymmärtääkseen CouchDB:n logiikkaa ylipäätään on tiedettävä, mitä tarkoitetaan hajautetuilla järjestelmillä. Hajautettujen järjestelmien puolesta on puhunut mm. Berkeleyn yliopiston professori Eric A. Brewer, jota pidetään yhtenä suurimmista vaikuttajista hajautettuun tietotekniikkaan. Hajautettu järjestelmä toimii pienten osien summana toiminnallisuuksien jakautuessa verkon eri osiin. Kirjassa *CouchDB: The Definitive Guide* on kuvattu Brewerin lause, toiselta nimeltään CAP-lause, jonka mukaan mikään järjestelmä ei voi samanaikaisesti taata yhtäpitävyyttä (*consistency*), saatavuutta (*availability*) ja vikasietoisuutta (*partition tolerance*). Järjestelmän yhtäpitävyys tarkoittaa, että kaikki sovellukset näkevät saman tiedon. Saavutettavuus tarkoittaa, että kaikki sovellukset pääsevät käsiksi tietoon, vaikka se ei olisikaan uusinta. Vikasietoisuus sallii tietokannan hajauttamisen useammalle laitteelle ilman tiedon hävikkiä. Kuvio 2 havainnollistaa näiden kolmen keskinäistä suhdetta. Kaikissa tapauksissa kolmesta eri vaihtoehdosta joutuu valitsemaan kaksi

ja uhraamaan yhden. Sovellusta määriteltäessä on siis tärkeää valita juuri oikea työkalu oikeaan tehtävään. (Anderson, Lehnardt & Slater 2010.)

Tiedon hajauttaminen palvelimen kuormituksen kasvaessa on hyvä keino lisätä vikasietoisuutta ja saatavuutta, mutta kääntöpuolena tiedon yhtäpitävyys kärsii. Jos yksi palvelin on verkon ulottumattomissa, tietokanta ei ole enää synkroninen. CouchDB:n ratkaisu on nk. lopullinen yhtäpitävyys (*eventual consistency*), jossa tietokannan eri solmukohdat (*nodes*) eivät lue tai kirjoita tietoa ennen kuin ne kaikki ovat ajan tasalla. Käyttäjälle tämä näkyy yleensä vain millisekuntien viiveenä järjestelmässä eikä täten ole este käytölle. (Anderson, Lehnardt & Slater 2010.)

Useiden käyttäjien hajautetuissa järjestelmissä on varauduttava siihen, ettei palvelimien sisältämä tieto ole aina yhtäpitävää. Se kuinka paljon yhtäpitävyydestä voidaan joustaa riippuu ohjelmiston suunnittelijoista ja suorituskykyvaatimuksista. (Vogels 2008.)



KUVIO 2. Brewerin lause

3.2 Sovellusrajapinta

Tässä luvussa käydään läpi keskeisimmät Couchbase-järjestelmän komponentit sovelluskehityksen kannalta. Couchbase sisältää neljä tärkeää komponenttia, jotka yhdessä muodosta-

vat sen sovellusrajapinnan. Rajapinta toimii sovelluskehityksen tukirunkona. Komponentit ovat palvelin, tietokanta, dokumentti ja replikaatio. (Anderson, Lehnardt & Slater 2010.)

3.2.1 Palvelin

Palvelimella tarkoitetaan järjestelmää, jossa CouchDB:tä ajetaan. Palvelulle on varattu yksi TCP-portti, joka on tavallisesti 5984. Jos palvelimeen yritetään päästä käsiksi selaimen kautta, aukeaa Futon. Futon on graafinen käyttöliittymä tietokannan hallintaa varten. Sieltä käsin voi suorittaa samat toimenpiteet kuin komentoriviltäkin. Komentoriviltä kutsuttuna vastaus näyttää seuraavalta: (Anderson, Lehnardt & Slater 2010.)

```
{"couchdb" : "Welcome", "version" : "0.10.1"}
```

Kutsu palauttaa JSON-merkkijonon, joka sisältää palvelimen tervehdyksen ja ohjelmiston versionumeron. JSON on natiivia Javascript-ohjelmointikieltä, jonka avulla mallinnetaan olioita. Javascriptin avulla yllä olevaa oliota voidaan käsitellä sellaisenaan, ja se on yhteensopiva muiden ohjelmointikielien kanssa. Python-kielen sanakirja (*dictionary*), Rubyn *hash* ja PHP:n assosiatiivinen taulukko ovat rakenteeltaan samankaltaisia ja muunnettavissa JSON-muotoon. Näiden kielien lisäksi JSON:n kotisivuilla on dokumentoitu kirjaston käyttö useita muita kieliä varten. (JSON.org 2012.)

Palvelin toimii käytännössä siten, että asiakasohjelma muodostaa ja lähettää HTTP-pyynnön palvelimelle, joka palauttaa vastauksen JSON-muodossa. Eroa tavallisen websivun lataamiseen on vain palautuvan dokumentin muoto. Tavallisesti palvelin palauttaa dokumentin HTML-muodossa, jonka selain tulkitsee käyttäjälle.

3.2.2 Tietokanta ja dokumentit

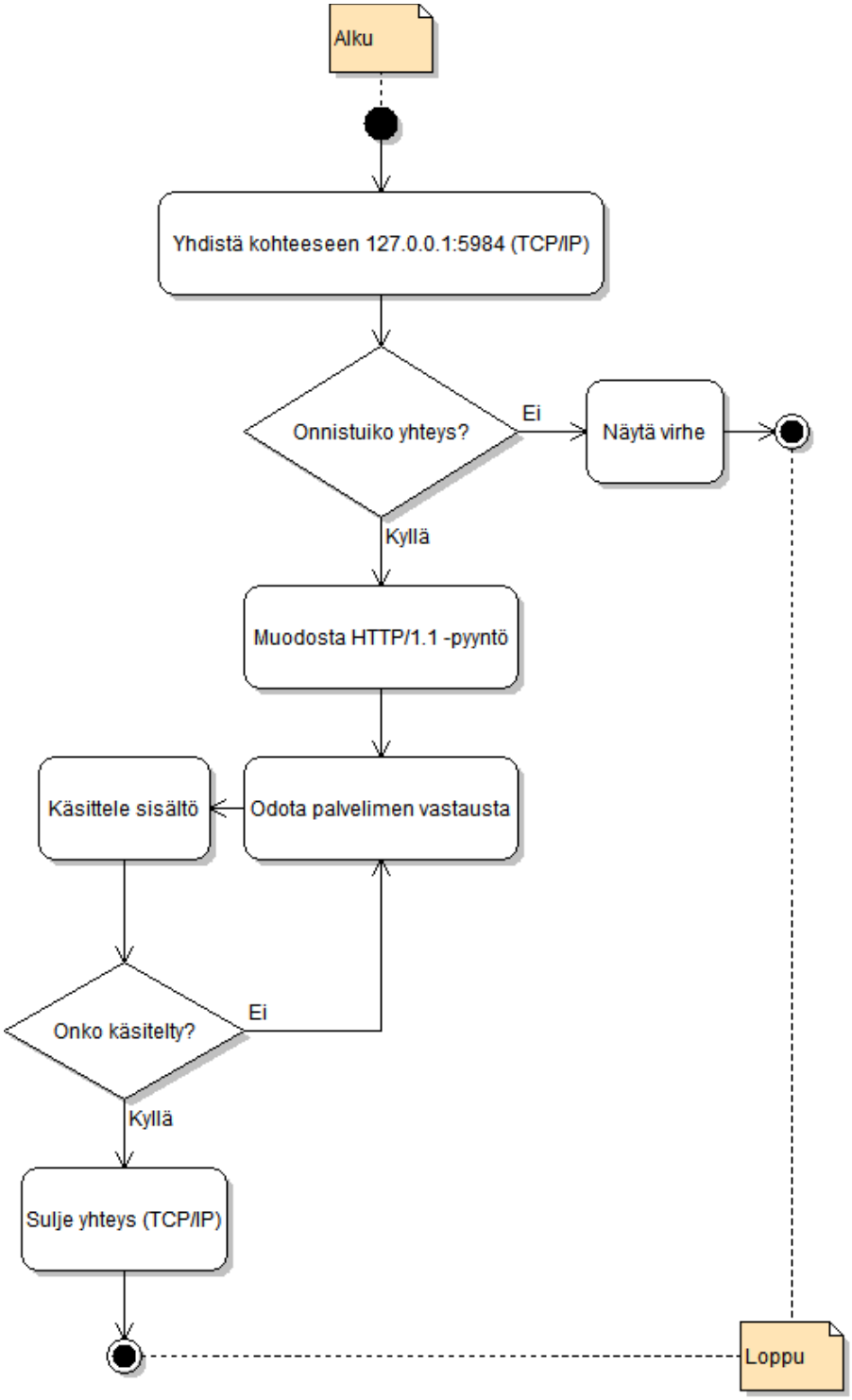
Lähes kaikki liikenne verkossa on sovelluksen ja palvelimen välistä liikennöintiä, poikkeuksena vertaisverkot (P2P, *peer-to-peer*). HTTP (*Hypertext Transfer Protocol*) on protokolla, joka määrittelee viestinvaihdon palvelimen ja sovelluksen välillä. HTTP-liikenne on

jaettu pyyntöihin ja vastauksiin. Tavallisesti selain muodostaa ja lähettää sovelluksen määräämät pyynnöt, joihin palvelin antaa vastaukset. Vastaus tulkitaan, ja sovellus suorittaa määrätyt toiminnot vastauksen sisällön mukaan. Esimerkki pyynnöistä ja vastauksista on Internet-selaimen käyttö. Selain lähettää pyynnön palvelimelle, joka palauttaa www-sivun vastauksena. Selain tulkitsee vastauksen ja näyttää sivun käyttäjälle sekä suorittaa mahdollisesti mukana tulevaa selainkoodia. (Anderson, Lehnardt & Slater 2010.)

HTTP-pyyntöjä on useaa eri tyyppiä. GET- ja POST-pyynnöt sekä hakevat että lähettävät tietoa, mutta jälkimmäinen lähettää tiedon koodattuna palvelimelle, jolloin ulkopuolisen on vaikeampi tulla yhteyden väliin. CouchDB:ssä GET-metodia käytetään tiedon hakemiseen ja POST-metodia dokumenttien muokkaamiseen. PUT-metodi luo uuden tietueen tyhjästä, ja DELETE-metodi poistaa tietueen. Jokainen tietokantaoperaatio kulkee REST-rajapinnan (*Representational State Transfer*) välityksellä. REST-rajapintaa käytetään paljon esimerkiksi Googlen tuotteiden omissa sovellusrajapinnoissa. HTTP-pyyntö sisältää tyypin lisäksi protokollan versionumeron ja otsakkeet. Pyynnön otsakkeissa on määritelty lisätietoja palvelinta varten. HTTP-vastaus sisältää protokollan versionumeron, numeerisen tilakoodin, otsakkeet sekä varsinaisen vastauksen sisällön. Tilakoodeja on useita kymmeniä, ja ne ovat merkitseviä sovelluslogiikan kannalta. Taulukkoon 1 on listattu tärkeimmät koodiryhmät. REST:n välityksellä tehtävän operaation kulkua havainnollistamaan on laadittu vuokaavio (Kuvio 3). (Anderson, Lehnardt & Slater 2010.)

2xx	Kaikki OK.
3xx	Resurssi on siirretty muualle. (301 = pysyvästi, 302 = tilapäisesti)
4xx	Ei pääsyä. (403 = estetty, 404 = puuttuu)
5xx	Virhe palvelimen koodissa ajon aikana.

TAULUKKO 1. HTTP-vastauksen tilatietojen yleisimmät virhekoodit ja selitykset



KUVIO 3. HTTP -pyynnön kulku vuokaavion avulla esitettynä

3.2.3 Replikointi

Tiedon yhtäpitävyys CouchDB:ssä on toteutettu replikoinnin eli monistamisen avulla. Monistaminen on prosessi, jossa kaksi kopiota tietokannasta synkronoidaan siten, että ne sisältävät saman tiedon. Jatkuva monistaminen sallii sen, että käyttäjän tietokanta voi sijaita missä tahansa, ja tietoon käsiksi pääsyssä ei synny merkittävää viivettä. Perinteisissä järjestelmissä varmuuskopioita otetaan usein yhdestä tietokannasta ja ne sijoitetaan samalle levypalvelimelle. CouchDB:n käyttämässä hajautetussa ajattelutavassa tiedon voi synkronoida esimerkiksi yhteydetöntä käyttöä varten mobiililaitteelle. Palattuaan yhteyden ääreen käyttäjä voi synkronoida tiedot takaisin työasemalle ja jatkaa työskentelyä normaalisti. CouchDB:n ulkopuolella paljon käytetty metodi on Google Chrome -selaimen synkronointi. Käyttäjä voi tallettaa verkon pilvipalvelimelle selaimen sovellukset, kirjanmerkit ja asetukset. Käytännössä tällöin selain kulkee käyttäjän mukana, kunhan vaaditut komponentit on asennettu. (Google 2012,. Anderson, Lehnardt & Slater 2010.)

Replikointi CouchDB:ssä tapahtuu yksinkertaisesti lähettämällä POST-pyyntö palvelimelle. Pyynnössä on määritelty lähde- ja kohdepalvelin, joiden välillä monistaminen tapahtuu. Jos pyynnössä on määritelty tietokannaksi URL-osoite, käsitellään kyseistä tietokantaa nk. etätietokantana, joka sijaitsee lähiverkon ulkopuolella. Ilman URL-osoitetta tietokanta on lokaali ja sijaitsee samalla palvelimella. Seuraavassa on kaksi esimerkkiä sekä paikallisesta että etäismonistuksesta.

```
POST /_replicate HTTP/1.1
{"source":"database","target":"database2"}

POST /_replicate HTTP/1.1
{"source":"database","target":"http://example.org/database2"}
```

Etätietokannan tunnistaa HTTP-protokollan etuliitteestä *http://* tai SSL-suojattuna *https://*, ilman sitä tietokantaa käsitellään paikallisena. (Anderson, Lehnardt & Slater 2010.)

Tiivistettynä monistus onnistuu minkä tahansa paikallisen ja etätietokannan yhdistelmästä ellei palvelinta ole määritelty estämään se. Paikallinen tietokanta voi sijaita esimerkiksi

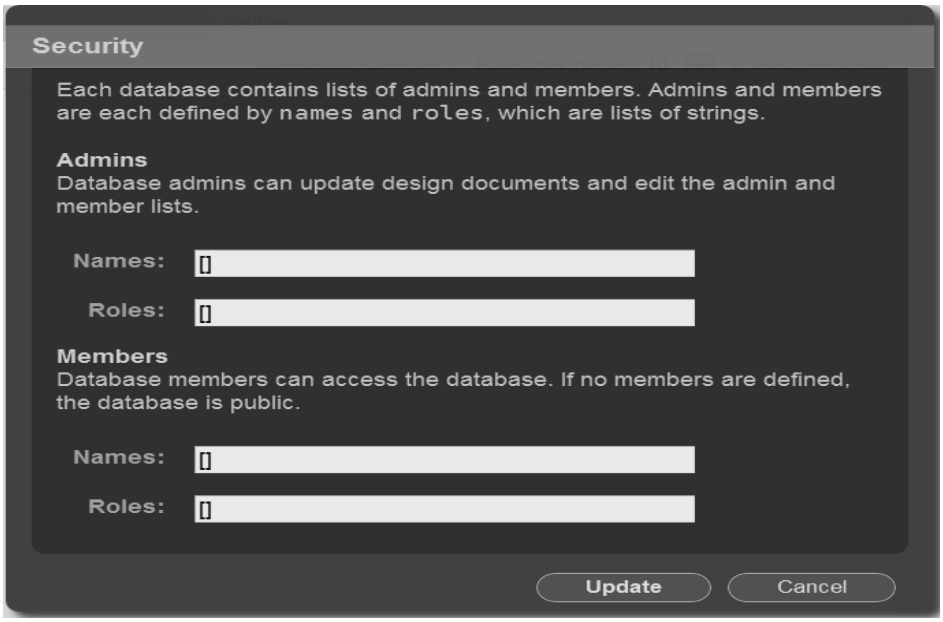
mobiililaitteessa ja etätietokanta palvelimella. Monistus on yksisuuntainen operaatio, joten tiedon synkronointiin vaaditaan kaksi kutsua.

4 SOVELLUSTEN KEHITYS JA KÄYTTÄMINEN

Tässä luvussa käsitellään CouchDB-sovellusten käyttöä tuotantoympäristössä ja sitä miten kehittäjien tulee sitä varten valmistautua. Luvussa esitellään myös kaksi käyttötarinaa eri yrityksiltä CouchDB:n käytöstä.

4.1 Tietoturva

Asennuksen jälkeen CouchDB:ssä ei ole määriteltynä käyttäjärooleja. Tämä tarkoittaa, että kuka tahansa saa lisätä, muokata tai poistaa dokumentteja. Oletuksena CouchDB kuitenkin vastaa vain kutsuihin, jotka tulevat palvelimelta itseltään *(localhost*). Ensimmäinen askel, jota kehittäjää kehotetaan tekemään, on luoda tietokantaan uusi ylläpitäjä. CouchDB:ssä on kaksi muistettavaa termiä: ylläpitäjä (*admin*) sekä oikeudet (*privileges*). Oletuksena kaikki ovat ylläpitäjiä. Ylläpitäjä voi luoda tietokantaan jäseniä, joilla on rajoitetut oikeudet. Seuraavassa kuvakaappaus on jäsenten ja käyttöoikeuksien luomiseen tarkoitetusta työkalusta Futonissa (Kuvio 4).



KUVIO 4. Futon-hallintapaneelin tietoturvaikkuna

Käyttäjien salasanat suojataan 128-bittisellä suolauksella (*salt*) ja niistä lasketaan SHA1-algoritmia käyttäen tarkistussumma. Tarkistussuma on ainoa salasanaan viittaava merkkijono, joka näkyy tietokannan ylläpitäjälle. Käyttäjän tunnistautuminen on mahdollista suorittaa selaimeen tallennettavalla evästeellä, jonka voimassaoloaika on oletuksena 10 minuuttia. (Anderson, Lehnardt & Slater 2010.)

Tavanomaisissa sovelluksissa tieto siirretään salaamatta käyttäjän ja palvelimen välillä. CouchDB voidaan asentaa esimerkiksi suojatun VPN-tunnelin päähän, jolloin vain halutut käyttäjät pääsevät käsiksi tietokantaan. Tämä on suositeltavaa erityisesti julkisissa langattomissa verkoissa, joissa nk. “Mies välissä” -hyökkäykset muodostavat merkittäviä tietoturvariskejä. CouchDB:tä kuten muitakin järjestelmiä ei tulisi ajaa palvelimella pääkäyttäjän oikeuksilla. Tämä asettaa etenkin UNIX-pohjaisissa järjestelmissä suuren riskin, mikäli CouchDB-palvelimeen kohdistuu tietomurto. Paras ratkaisu on luoda oma käyttäjänsä sovellukselle, jolla on vain välttämättömät käyttöoikeudet järjestelmään. (Anderson, Lehnardt & Slater 2010.)

CouchDB:n tietoturvallisuutta kehitetään jatkuvasti. Tuotantokäytössä on harkittava tietoturvallisuutta eri näkökulmista, kuten sitä keskusteleeko järjestelmä julkisen verkon yli ja kuinka sisäverkko on varustettu mahdollisia hyökkääjiä varten. Versiosta 1.1.0 alkaen Couchbase tukee Secure Sockets Layer (SSL) -suojausta. Tietokanta voidaan monistaa laitteille, jotka ovat SSL-välityspalvelimen (*proxy*) takana. (Couchbase 2012; Anderson, Lehnardt & Slater 2010.)

4.2 Tietojoukon koostaminen

Toistaiseksi on käsitelty ainoastaan yksittäisen dokumentin hakemista tunnisteella tai kaikkien dokumenttien näyttämistä kannasta. Usein kuitenkin tulee tarve koostaa tieto helposti käsiteltäväksi näkymäksi, joka rajaa koko tietokannasta suppeamman otoksen tietyin ehdoin. Tällöin puhutaan Map/Reduce-algoritmista, jossa ohjelmoidun funktion avulla koostetaan tietoa. Esimerkiksi verkkokaupan tietokannassa voi olla useita dokumentteja, joille on yhteinen tietomalli. Toinen dokumentti kuvaa asiakkaan tietoja ja toinen myytävän tuotteen tietoja. Seuraavassa on esimerkki yksinkertaisesta koostefunktiosta.

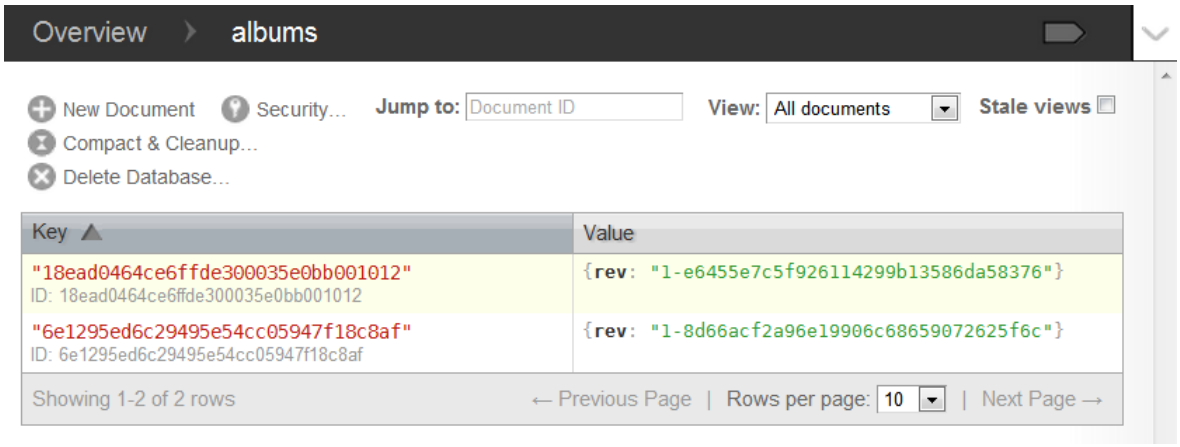
```
function (doc)
{
    if (doc.type === "product" && doc.price)
    {
        emit(doc.id, doc.price);
    }
}
```

Funktio hakee tietokannasta kaikki ne dokumentit, jotka sisältävät avainkentän price, ja joiden type-avainkentän arvo on “product”. Toisin sanoen haetaan tietokannasta ne dokumentit, joille on määritelty hinta ja jotka kuvaavat tuotteita. Lopuksi näiden dokumenttien tunniste ja hinta lähetetään käsittelyä varten eteenpäin. Koostefunktion välittämä tieto käsitellään toisessa funktiossa, joka tässä tapauksessa laskee annettujen dokumenttien hinnat yhteen ja palauttaa lasketun summan, esimerkiksi:

```
function (keys, prices)
{
    return sum(prices);
}
```

Edellä olevat ovat yksinkertaisimpia esimerkkejä CouchDB:n dokumenttien koostamisesta. Koostaminen on tehokas tapa löytää kannasta haluttu tieto. Koostefunktioita ei tarvitse määritellä joka kerta uudestaan, vaan funktiot voidaan tallentaa pysyviksi näkymiksi (*view*) uutta käyttöä varten. Varjopuolena koostamisessa on näkymien hidas luonti ohjelman suorituksen aikana. (Anderson, Lehnardt & Slater 2010.)

Kuviossa 5 on kaapattu Futonin näkemä tieto yhdestä tietokannasta *(albums*). Kuvion oikean yläreunan View-pudotusvalikosta voi tarkastella joko kaikkia dokumentteja tai itse määriteltyä näkymää.



KUVIO 5. Tietokannan dokumenttien listaus Futon -hallintapaneelissa

4.3 Sovellusrajapinnan käyttö

CouchDB on yksinkertaisen logiikkansa ja joustavan tietorakenteensa ansiosta helppo mallintaa lähes mille tahansa ohjelmointikielelle. Sovellusrajapinnan edellytyksenä on tuki HTTP-pyynnöille verkon yli sekä JSON-parsinnalle. Edellä mainitut ominaisuudet ovat nykypäivänä lähes kaikkien usein käytettyjen kielten vakiokomponentteja. Tässä luvussa

käydään läpi lyhyt esimerkkiohjelma, joka on kirjoitettu Python-kielellä. Ohjelmassa on hyödynnetty Pythonin omaa CouchDB-moduulia, jonka uusin versio opinnäytetyön kirjoitushetkellä on 0.8. Tämä yhdessä muiden moduulien kanssa löytyy Pythonin keskitetystä pakettivarastosta. (Python Software Foundation 2011.)

Ensimmäisenä toimenpiteenä CouchDB-moduuli liitetään ohjelmaan Pythonin import-käskyllä. Moduuli sisältää kaikki tarvittavat toiminnallisuudet tietokannan tietojen käsittelyyn. Seuraavassa on määritelty funktio, jolla otetaan yhteys tietokantaan. Jollei toisin käsketä, yhteys kohdistetaan samalle palvelimelle, jossa koodi sijaitsee (*localhost*). Funktion palautusarvo onnistuessaan on tietokantayhteyttä kuvaava objekti tai virheen sattuessa negatiivinen totuusarvo.

```
import couchdb

def couchdb_connect(username, password, host="127.0.0.1", session=None):
    """
    Connect to CouchDB server and return an instance object
    of the database connection or 'False' on error.
    """
    try:
        # Connect and send credentials.
        server = couchdb.Server(host)
        server.resource.credentials = (username, password)
        return server
    except Exception, e:
        print "Fatal error: %s" % str(e)
        return False
```

Ohjelman päärutiinissa otetaan yhteys palvelimeen ja välitetään parametreina hallinnointiin vaadittavat käyttäjätunnus ja salasana.

```
if __name__ == "__main__":
    connection = couchdb_connect("admin", "secret")

    if not connection:
        print "Error connecting to server, exiting."
        exit()
```

Jos yhteys muodostettiin onnistuneesti, viedään tietokanta muuttujaan. Tietokannan nimi tässä esimerkissä on *reports*. Esimerkkidata muodostetaan Pythonin sanakirjan avulla, joka käyttää samaa syntaksia kuin JSON.

```
# Select database called reports.
db = connection['reports']

# Construct data to send.
document = {

    'author': "Yritys",
    'classification': "confidential",
    'date': "2012-05-04",
    'subject': "Osavuosikatsaus Q1/2012",

}
```

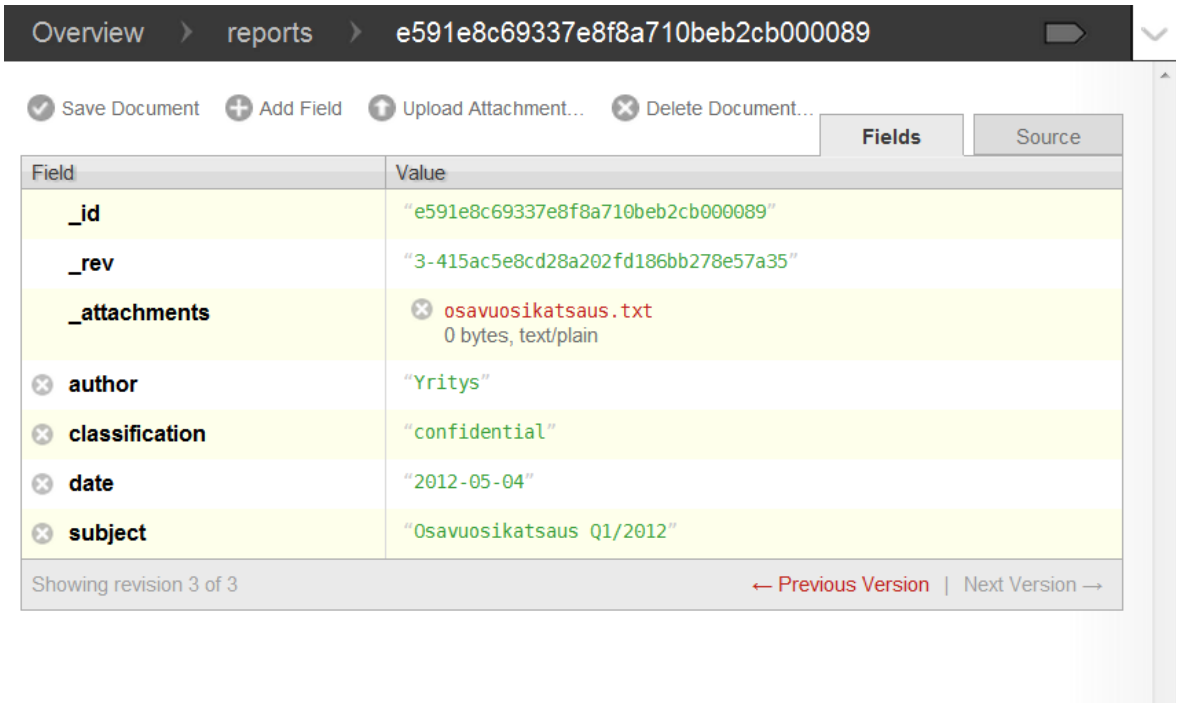
Tietokantaobjekti sisältää metodin, jonka avulla tieto tallennetaan. Metodi hyödyntää POST-pyyntöä, ja sillä voidaan luoda uusi dokumentti tyhjästä tai antamalla dokumentin tunniste, jolloin päivitetään vanhaa dokumenttia. Metodi palauttaa aina kaksi arvoa: dokumentin tunnisteen ja versionumeron. Nämä tallennetaan alla omiin muuttujiinsa Pythonin tuple-tietotyyppiä hyödyntämällä. Lopuksi ohjelmassa haetaan luotu dokumentti GET-pyynnöllä antamalla äskettäin saatu tunnistenumero ja tulostetaan sisältö. Kuviossa 6 on esitetty näkymä dokumentista ohjelman suorituksen jälkeen. Kuviossa näkyvä liitetiedosto (*attachment*) on lisätty myöhemmin hallintapaneelista käsin.

```
# Fetch document ID and revision.
doc_id, doc_rev = db.save(document)

# Fetch row with given id (=primary key)
row = db[doc_id]

# Print it out
for key in row:
    print "%s: %s" % (key, row[key])
```

Esimerkkiohjelma kuvaa lyhyesti kaksi tietokannan perustoiminnallisuutta: tiedon säilömisen ja hakemisen. Moduulista löytyy metodit myös tiedon poistamiselle, päivittämiselle ja replikoinnille muiden lisäksi. Kehittäjän ei tarvitse tyytyä mukana toimitettaviin metodeihin, vaan laajennus voidaan suorittaa helposti periyttämällä uusi aliluokka. Keskeistä kaikille toiminnallisuuksille on HTTP-pyyntöjen käsittely, JSON-tiedon lukeminen ja TCP-protokollan käyttö.



KUVIO 6. Dokumentin sisältö Futon-hallintapaneelissa

4.4 Asiakkaiden käyttötarinat

CouchDB:tä on onnistuneesti käytetty useissa yrityksissä maailmalla. Tässä luvussa käydään läpi muutama case, joissa yritys on hyötynyt ei-relaatiomallisesta infrastruktuurista liiketoiminnassaan.

4.4.1 Case A: Puhelinvaihteen analysaattori

Nu Echo on kanadalainen puheentunnistusohjelmistoihin erikoistunut yritys, jonka VoiceXML-ohjelmistokehystä käytetään pääasiassa asiakaspalvelun kantaan saapuvien puheluiden analysointiin. Nu Echo valitsi CouchDB:n ensisijaisesti palvelemaan puheluiden varastointia. Aiemmassa toteutuksessa kantaan saapuneiden puheluiden tiedot tallennettiin levylle tekstitiedostoon, joka myöhemmin vietiin SQL-tietokantaan. Tiedon monimutkaisen rakenteen vuoksi sovelluksen tietokanta koostui noin 15 taulusta, joiden hallinta yksit-

täisessä tekstitiedostossa oli hyvin vaikeaa. Tekstitiedosto koostui sarkaimella erotetuista kentistä, ja osa sisällöstä oli koodattu JSON-muotoiseksi. (Nu Echo 2011.)

Ensimmäiseksi Nu Echo suunnitteli tiedon sarjallistamisen uudelleen. Sarjallistamisessa tieto muutetaan muotoon, jossa se on helppo tallentaa tekstitiedostoksi. Tekstitiedostoista luovuttiin ja tilalle otettiin pelkästään JSON-muotoisia dokumentteja, jotka merkittiin omalla yksilöllisellä tunnisteella (ID). Nämä dokumentit ajettiin CouchDB-tietokantaan kerrallaan 100 kpl:n erissä HTTP:n PUT-metodia ja CouchDB:n sovellusrajapintaa käyttäen. Toiseksi Nu Echo suunnitteli tietokantaan omat rakenteelliset näkymät, jotka suodattivat koko tietokannasta halutun otoksen Map/Reduce-algoritmeja käyttäen. Näin tiedon liikuttelu muistissa helpottui huomattavasti verrattuna vanhaan selkotekstiin pohjautuvaan toteutukseen. Nu Echon toteutus hyödytti yritystä kolmella tapaa. Suorituskyky kasvoi, tiedon rakenne muuttui joustavammaksi, ja dokumentteihin pystyi tallentamaan runsaasti erityyppistä tietoa. (Nu Echo 2011.)

Tiedon säilöminen CouchDB:ssä on moninkertaisesti nopeampi prosessi kuin SQL:ssä, mikä etenkin puheluiden nauhoittamisen aikana osoittautui suureksi eduksi. Kiireisimpinä tunteina vaihteeseen saattaa kohdistua jopa tuhansia puheluita. JSON-rakenteen lisäksi oli helppoa varastoida myös binaarista tietoa kantaan, tässä tapauksessa puheluiden äänitteet. CouchDB:ssä nämä toteutetaan dokumenttien liitetiedostoina, ja niillä on oma linkkinsä, jotta resurssit pystytään avaamaan esimerkiksi selaimessa ja tallentamaan mille tahansa koneelle. (Anderson, Lehnardt & Slater 2010.)

4.4.2 Case B: Reseptitietokanta

Syksyllä 2010 AllRecipes.com-sivuston ylläpito alkoi pohtia uutta tietokantaratkaisua suuren suosion saavuttaneelle reseptitietokantasivustolleen. Vuotta aikaisemmin sivustolle oli laadittu moottori, joka tallensi sivuja välimuistiin nopeaa hakua varten. Kävijämäärien lisääntyessä tekniikan rajat tulivat kuitenkin vastaan, ja seurauksena oli kasvavissa määrin sivuston aikakatkaisuja sekä muita suorituskykyyn liittyviä ongelmia. (AllRecipes 2012.)

Vaatimuksena uuden sivuston tekniikalle oli mahdollisimman vaivaton sulautuminen jo käytössä olevaan Windows-infrastruktuuriin. CouchDB oli ylläpidon valinta toteutuksen yhteensopivuuden, skaalautuvuuden ja helpon muokattavuuden vuoksi. Koska CouchDB:n sovellusrajapintaa oli mahdollista käyttää C#:n avulla, säästyi vanha koodi suurilta muutoksilta. Konkreettinen hyöty sivuston tekijöille CouchDB:stä tuli ilmi sivuston kasvaneessa suorituskyvyssä ja palvelinvikojen laskussa. Lisäksi sivuston uusien toiminnallisuuksien kehittäminen sujuu yksinkertaisemmin, koska kehitysympäristö on helppo replikoida suoraan tuotantopalvelimelta testipalvelimelle. (AllRecipes 2012.)

5 POHDINTA

Opinnäytetyön alkuperäisenä suunnitelmana oli tutkia eri sovellusten mahdollisuuksia käsitellä ja tallentaa tietoa yhteydetöntä tilaa varten. Haasteeksi suunnitelmassa osoittautui tiedon yhtäpitävyys, kun suuri osa asiakassovelluksista oli ollut ilman verkkoyhteyttä esimerkiksi vierailtaessa verkon katvealueilla. Kuinka tieto pysyisi synkronisena ja mitä haasteita se asettaisi kehittäjälle, kun asiakassovelluksia olisi kymmeniä ellei jopa satoja? Synkronoinnin lisäksi tärkeää oli varmistaa eheys tietoturvan näkökulmasta. Verkkoliikenteen uhkana ovat salakuunteleminen ja tiedon haitallinen muokkaaminen. Tätä varten tuli harkita siirtoyhteyksien salaamista jollain helppokäyttöisellä protokollalla. Synkronoinnin tuli olla mahdollista toteuttaa kahden solmukohdan välille pystytetyllä *Virtual Private Network* (VPN) -tunnelilla. VPN:llä toteutetussa yhteydessä kaksi fyysisesti kaukana toisistaan olevaa laitetta keskustelee keskenään kuin ne olisivat samassa lähiverkossa. Toinen vaihtoehto on tietoliikenteen salakirjoitus jollain tunnetulla salausalgoritmilla, esimerkiksi AES:llä tai Blowfishillä.

Tutkittuani asiaa päädyin Couchbasen verkkosivuille. Pian tutustuin Apachen CouchDB-moduuliin ja huomasin sille löytyvän ohjelmakirjastoja useista eri ohjelmointikielistä. Lisäksi CouchDB keskusteli laitteiden kanssa suoraan HTTP-rajapinnan välityksellä, ja siirtoformaattina oli helppolukuinen JSON. Mielenkiinnosta perehdyin tarkemmin moduulin tekniikkaan ja asensin verkon pilvipalvelimelle koeympäristön. Havainnollistavat oppaat johtivat tutustumaan laajemmin NoSQL:ksi kutsuttuun arkkitehtuuriin. Pitäydyin työssäni kuitenkin tiiviisti CouchDB:n parissa, vaikka esimerkiksi samanhenkiset MongoDB, Redis ja Cassandra olivat myös sovelluskehittäjien suosiossa.

Työ onnistui omasta mielestäni hyvin, ja aihepiirin käytännönläheisyydestä huolimatta mukaan sopi myös eri vaihtoehtojen tarkastelua ja kritiikkiä. Selkeänä haittana työtä tehdessä oli kirjallisuuslähteiden vähäinen saatavuus etenkin suomeksi, mutta sitäkin enempi verkosta löytyi aihetta tarkastelevia artikkeleita ja oppaita englanniksi.

LÄHTEET

- AllRecipes, 2012. Couchbase is the Secret Ingredient for Scaling Needs at Allrecipes.com. WWW-dokumentti. Saatavissa: http://www.couchbase.com/case-studies/allrecipes. Luettu 7.5.2012.

- Anderson, J. Chris, Lehnardt, J. & Slater N. 2010. CouchDB: The Definitive Guide. O'Reilly Media, Inc. Sähköinen painos saatavissa: http://guide.couchdb.org/.

- Couchbase, 2011. Navigation the Transition From Relational to NoSQL Database Technology. PDF-tiedosto. Saatavissa: http://info.couchbase.com/Relational-to-NoSQL.html. Luettu 27.4.2012.

- Couchbase, 2012. Why NoSQL? WWW-dokumentti. Saatavissa: http://www.couchbase.com/why-nosql/nosql-database. Luettu 27.4.2012.

- Daily Tech, 2008. World's Data to Reach 18 Zettabytes by 2011. WWW-dokumentti. Saatavissa: http://www.dailytech.com/Worlds+Data+to+Reach+18+Zettabytes+by+2011/article11055.htm. Luettu 27.4.2012.

- Gantz, J. & Reinsel, D., 2011. IDC: Extracting Value from Chaos. PDF-tiedosto. Saatavissa: http://idcdocserv.com/1142. Luettu 26.5.2012.

- Google, 2012. Why sign in to Chrome. WWW-dokumentti. Saatavissa: http://support.google.com/chrome/bin/answer.py?hl=en&answer=165139. Luettu 1.6.2012.

- IBM, 2012. Bringing Big Data to the Enterprise. WWW-dokumentti. Saatavissa http://www-01.ibm.com/software/data/bigdata. Luettu 31.5.2012.

- JSON.org, 2012. Introducing JSON. WWW-dokumentti. Saatavissa: http://json.org. Luettu 26.5.2012.

- Leavitt, N., 2010. Technology News: Will NoSQL Databases Live Up to Their Promise. PDF-tiedosto. Saatavissa: http://www.leavcom.com/pdf/NoSQL.pdf. Luettu 6.5.2012.

- Nu Echo, 2011. CouchDB for call analysis data – a case study. WWW-dokumentti. Saatavissa: http://blog.nuecho.com/2011/05/18/couchdb-for-call-analysis-data-a-case-study. Luettu 5.5.2012.

- Python Software Foundation, 2011. Python-kielen pakettivaraston verkkosivut. Saatavissa: http://pypi.python.org/pypi. Luettu 26.5.2012.

- Vogels, W., 2008. All Things Distributed: Eventually Consistent – Revisited. WWW-dokumentti. Saatavilla: http://www.allthingsdistributed.com/2008/12/eventually_consistent.html. Luettu 6.5.2012.