KARELIA-AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma

Jukka Mustonen

BIG DATA -MÄÄRITELMÄ JA ALUSTAT

Opinnäytetyö
Kesäkuu 2016

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Kesäkuu 2016
Tietotekniikan koulutusohjelma

Karjalankatu 3
80200 JOENSUU
(013) 260 600

Tekijä(t)
Jukka Mustonen

Nimeke
Big Data määritelmä ja alustat

Toimeksiantaja
Karelia-amk

Tiivistelmä

Tässä opinnäytetyön päämääränä oli tutkia Big Dataa ja määritellä mitä se on, miksi sitä tarvitaan, sekä esitellä yleisimpiä Big Data alustoja. Lisäksi työn teknisessä osuudessa toteutettiin ratkaisu sensori datan keräämisestä Wago logiikka moduulista ja kyseisen sensori datan varastoimisesta Big Data alustalle. Raportti käsittelee lyhyesti toteutuksessa käytettyjä tekniikoita ja syitä miksi kyseiset tekniikat on valittu käytettäväksi opinnäytetyön osana toteutetussa ratkaisussa.

Tutkimalla Big Datan määritelmiä ja sitä miten Big Dataa hyödynnetään havaittiin että Big Datan määritelmä vaihtelee hyvin vahvasti lähteestä riippuen. Lisäksi määritelmää sille mitkä alustat luokitellaan Big Data-alustoiksi, oli hankala löytää. Lopulta kuitenkin löydettiin Big Data termille kuitenkin selkeä määritelmä, sekä kaksi erillistä määritelmää Big Data alustalle.

Big Datalla on merkittävä taloudellinen potentiaali ja sen merkitys tulee kasvamaan entisestän tulevaisuudessa. Big Data -alustat eivät kuitenkaan tule korvaamaan tietovarastoja ja muita relationaalisia järjestelmiä kokonaan.

Kieli
suomi

Sivuja 41
Liitteet 0

Asiasanat
Big Data, Hadoop, NOSQL

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
June 2016
Degree Programme in Information Technology

Karjalankatu 3
80200 JOENSUU
FINLAND
+358 13 260 600

Author (s)
Jukka Mustonen

Title
Big Data platforms and definition

Commissioned by
Karelia University of Applied Sciences

Abstract

The goals of this thesis were defining what is Big Data, describing why Big Data is used and describing the main aspects of the most popular Big Data platforms. In addition to the theoretical aspect of this thesis, one of the goals is building a solution to collect sensor data from Wago logic module and store the data on a Big Data platform. This report describes techniques used in the solution and reasons why those particular techniques were chosen for the solution.

It was discovered that definition of Big Data varies greatly and finding out which platforms were considered to be Big Data platforms was challenging. Eventually a proper definition for Big Data was found, and two competing definitions for Big Data platform were discovered.

Big Data has a great economical potential and its role will grow even further in the future. However Big Data platforms will not replace data warehouses or other relational systems completely.

Language	Pages 41
Finnish	Appendices 0

Keywords

Big Data, NOSQL, Hadoop

Sisältö

Käsitteet

ACID on lyhenne sanoista atomicity, consistency, isolation ja durability. ACID ominaisuuksien tarkoitus on taata tietokannan luotettavuus. ACID mallin määrittelee neljä keskeistä ominaisuutta:

- atomicity eli atomisuus tai jakamattomuus,
- consistency oikeellisuus tai eheys,
- isolation eli eristyvyys tai erillisyys,
- durability eli pysyvyys.

Atomicity eli atomisuus tai jakamattomuus asettaa transaktiolle ehdon, että kaikki transaktioon liittyvät toiminnot suoritetaan loppuun asti tai kyseistä transaktiota ei suoriteta ollenkaan. (Techopedia Inc, 2016) (Sandborg, 2012) (Tiwari, 2011) Oikeellisuus ehdon mukaan taas tietokantaan voidaan viedä ainoastaan tietoa, jos se täyttää ennalta määritellyt ehdot. (Tiwari, 2011)

Eristyvyydellä taataan että samanaikaiset transaktiot voivat tapahtua ilman että ne vaikuttavat toistensa toimintaan. Transaktioiden eristyvyys voidaan taata esimerkiksi lukituksella, jossa transaktion sallitaan suorittaa tietokantaan kohdistuvan luku- tai kirjoitusoperaation vain, jos se on lukinnut operaation kohteen asianmukaisella lukolla. Jolloin toinen transaktio odottaa, että ensimmäinen on suoritettu ennen kuin se suoritetaan. Pysyvyys (durability) taas tarkoittaa, että transaktion muutosten onnistuminen varmistetaan ja mikäli esimerkiksi järjestelmävirhe estää muutosten tekemisen kantaan palautetaan kanta transaktiota edeltävään tilaan. (Techopedia Inc, 2016) (Tiwari, 2011)

BASE on lyhenne sanoista “basically available”, “soft state”, “eventual consistency”. BASE on tietojärjestelmien kehitysfilosofia joka korottaa saata-

vuuden johdonmukaisuuden yläpuolelle. Useimmissa tapauksissa tämä tarkoittaa että järjestelmässä joudutaan sallimaan tietynasteinen epätarkkuus. Esimerkiksi pankkisiirroissa voidaan sallia tilisiirron tekemisen ja pankkitilin saldon päivittymisen välille tietty viive. BASE-malli kehitettiin vaihtoehdoksi ACID-mallille ja sen on tarkoitus auttaa kehittämään tehokkaasti skaalautuvia ja edullisempia tietojärjestelmiä ja antaa enemmän vaihtoehtoja toimintojen laajentamiseen. (Techopedia Inc, 2016)

1 Johdanto

Tämän opinnäytetyön tarkoituksena on tutkia mitä tarkoittaa Big Data, mikä sen tarkoitus on ja miksi sitä tarvitaan. Lisäksi tutkitaan yleisimpiä Big Data -alustoja sekä kyseisten Big Data -alustojen välisiä eroavaisuuksia niiden toimintojen osalta sekä niiden rakenteellisten erojen osalta. Lisäksi opinnäytetyön toiminnallisessa osassa toteutetaan pienimuotoinen ratkaisu sonsoritietojen keräämiseen, sekä persistointiin, jossa Wago-logiikkamoduuliin kytketyiltä antureilta kerätään mittausarvoja, jotka tallennetaan ensin tietokantaan, josta ne varastoidaan Big Data -alustalle.

Big Dataa ja sen merkitystä on tutkittu aikaisemminkin muumuassa McKinsey Global Instituten ja IDC:n (International Data Corporation) toimesta. McKinsey Global Institute on julkaissut tutkimuksen Big data: The next frontier for innovation, competition and productivity jossa he määrittelevät mitä Big Data on sekä tutkivat Big Datan mahdollisuuksia. IDC:n tutkimusta, IDC FutureScape: Worldwide Big Data and Analytics 2016 Predictions, ei kuitenkaan ollut mahdollista ottaa käsittelyyn tässä opinnäytetyössä sillä lukuoikeus kyseiseen tutkimukseen olisi maksanut 4500 dollaria.

Big Datalla viitataan lyhyesti sanottuna niin suurin tietomääriin että niiden käsitteleminen perinteisillä tietokantajärjestelmillä on mahdotonta tai erittäin vaikeaa. (Awadallah & Graham, 2014; McKinsey Global Institute, 2011). Big Datan merkitys kasvaa tulevaisuudessa, sillä merkittävä osa kaikesta tiedosta on strukturoimattomassa muodossa, jota perinteiset järjestelmät eivät kykene käsittelemään. (McKinsey Global Institute, 2011).

2 Relationaaliset kannat

Relationaalinen tietokantamalli sai alkunsa 1960-luvun lopulla, kun IBM:n tutkija tri E. F. Codd etsi uusia tapoja käsitellä suuria tietomääriä, koska hän oli tyytymätön silloin käytössä olleisiin tietokantamalleihin. Hänen ajatuksenaan oli käyt-

tää matematiikan oppeja ja rakenteita tietojen käsittelyssä, minkä hän uskoi ratkaisevan muissa silloisissa tietokantamalleissa ilmenneet ongelmat, kuten “ylimääräinen data, liian suuri riippuvuus fyysisestä toteutuksesta ja tietojen heikko eheys.” (Hernandez, 2000.)

Relaatiomalli perustuu kahteen matematiikan osa-alueeseen jotka ovat joukkoteoria sekä ensimmäisen kertaluvun predikaattilogiikka. Relationaalisessa mallissa tiedot viedään tietokantaan relaatioina, jotka esitetään käyttäjälle tauluina, jokainen relaatio eli taulu puolestaan koostuu monikoista eli tietueista sekä attribuuteista eli kentistä. Relationaalisessa mallissa tietueiden ja kenttien fyysisellä järjestyksellä ei ole merkitystä, sillä taulujen sisältämien tietueiden tunnistamiseen käytetään kenttää, joka sisältää muista poikkeavan arvon. Tämä oli suuri etu verrattuna varhaisiin tietokantamalleihin, kuten hierarkkiseen tietokantamalliin sekä verkkotietokantamalliin, joissa käyttäjän on tiedettävä tietueen fyysinen sijainti, jos käyttäjä haluaa hakea tietokannasta tietoa. (Hernandez, 2000.)

Relaatiokantojen suurimmiksi eduiksi nostetaan käytön helppous, siltä osin että yksinkertaisessa ja selkeässä taulukkomuodossa esitettyjen tietojen tarkastelu ja ymmärtäminen on helppoa (Jones, 2015; Amiri, 2013). Relaatiokannassa tieto on vain yhdessä paikassa joten tiedon päivittäminen ja poistaminen on näin ollen helppoa ja kaikki taulut jotka viittaavat muokattuun tauluun käyttävät näin ollen aina ajantasaista tietoa. (Jones, 2015; Amiri, 2013; Karvin, 2016).

Lisäksi eri taulujen tietoja on mahdollista muotoilla ja yhdistellä join- ja project-komennoilla tarvittavien tietojen saamiseksi kannasta. Relaatiokantojen eduksi lasketaan myös tarkkuus, sillä matemaattisten rakenteiden käyttö varmistaa, ettei monimutkaisten taulujen välisten yhteyksien seurauksena aiheudu tietoihin epäselvyyksiä. (Amiri, 2013.)

Relaatiokannoissa myös turvallisuus on otettu hyvin huomioon. Esimerkkinä mainittakoon, että tietokannan käyttäjille voidaan antaa yksityiskohtaisia käyttöoikeuksia tiettyihin tauluihin ja kantoihin, ja näin ollen pääsyä kriittisiin tietoihin voidaan tehokkaasti rajoittaa. (Jones, 2015; Amiri, 2013; Karvin, 2016.) SQL-

kieli on myös erittäin tehokas käytettäessä relationaalista tietokantamallia, mutta toisaalta muuttuu, joko monimutkaiseksi käyttää, tai ominaisuuksiltaan rajoitetuiksi, jos data on strukturoitu jollakin toisella tavalla. (Amiri, 2013;Levitt, 2010).

2.1 Relationaalisten kantojen haasteet

Suurimpana ongelmana tietokantojen kohdalla on, että tieto täytyy aina sovittaa tietokannan käyttämään tietomalliin. (Madden, 2012). Relationaalisten kantojen tapauksessa tämä tarkoittaa sitä että kaikki tieto muutetaan taulujen muotoon (Levitt, 2010), ennen kuin sitä voidaan hyödyntää tekemällä kyseiseen tietoon liittyviä kyselyjä, mikä puolestaan rajoittaa tietokantojen kykyä käsitellä suuria datavirtoja, (Madden, 2012) sekä voi johtaa siihen että tietokannasta saattaa kehittyä monimutkainen, hidas ja kömpelö käyttää. (Levitt, 2010) Toisena keskeisenä ongelmana on se, että monet datavirtojen käsittelyyn liittyvät teknologiat eivät integroidu hyvin relationaalisiin tietokantamoottoreihin. Lisäksi relationaalisten tietokantamoottoreiden tarjoama tuki tietokannan sisäiseen statistiikkaan ja mallintamiseen on hyvin rajallista, ja yleisesti ottaen nämä kyseiset ominaisuudet eivät skaalaudu kovin tehokkaasti massiivisiin tietomääriin. (Madden, 2012.) Lisäksi relationaalisia kantoja ei ole suunniteltu osioimaan toimintojaan, joten niiden linkittäminen toimimaan yhdessä on hankalaa. (Levitt, 2010).

2.2 Tietovarastot ja Big Data

On olemassa lukuisia eri määritelmiä mitä Big Datalla tarkoitetaan sekä mitä tietovarastoinnilla tarkoitetaan ja jopa joillakin aktiivikäyttäjillä on virheellinen käsitys niiden merkityksestä. Monet ovat esimerkiksi harhautuneet luulemaan että esimerkiksi Hadoop olisi korvaamassa tietovarastot. Kasvavana trendinä näitä sen sijaan käytetään rinnakkain. (Awadallah & Graham, 2014.)

Bill inmonin mukaan tietovaraston ja Big Datan välillä on selkeä ero. Hän kuvaa Big Dataa työkaluksi ja tietovarastoa arkkitehtuuriksi. Big Data ratkaisussa on

hänen mukaansa kyse työkalusta, jota käytetään suurten tietomäärien varastointiin ja analysointiin. (Inmon, 2013.) Tietovarasto on taas tapa järjestellä tietoa niin, että yrityksellä on käytettävissä “yksi universaali totuus” päätöksentekoa varten eli kun henkilö noutaa tietovarastosta tietoa hän tietää että kaikki muut tekevät päätöksiä saman tiedon pohjalta. (Inmon, 2013; Hovi, 1997).

Inmon perustelee tarvetta molempien järjestelmien olemassa oloon, sillä että Big Dataa tarvitaan siksi, koska monilla yrityksillä on käytössään valtava määrä dataa, joka oikein hyödynnettynä antaa arvokasta tietoa. Tämä puolestaan antaa yritykselle mahdollisuuden laadukkaampaan päätöksentekoon, joka puolestaan parantaa tehokkuutta ja kasvattaa yrityksen tuloja, mikä on jokaisen yrityksen tavoite. Inmonin mukaan tietovarastoa tarvitaan sillä yrityksen johto tarvitsee tietoa joka on luotettavaa ja joka on kaikkien saatavilla, jotta se voi tehdä informoituja päätöksiä ja pysyy selvillä siitä mitä yrityksessä tapahtuu. (Inmon, 2013.)

3 Big Data

3.1 Määritelmä

Big Data on niin suuri tieto massa että sen hallitseminen ja sen sisältämien tietojen analysoiminen on perinteisillä tietokantatyökaluilla joko mahdotonta tai erittäin vaikeaa. (McKinsey Global Institute, 2011; Awadallah & Graham, 2014). McKinsey Global Instituten tutkijoiden mukaan määritelmä on tarkoituksella jätetty joustavaksi sillä tietokantatyökalujen ja laitteiden kehittyessä oletettavasti pystytään suuria tietomääriä käsittelemään entistä tehokkaammin. Lisäksi tietojen käsittelyssä käytettävät työkalut, menetelmät sekä käsiteltävät tietomassat vaihtelevat eri toimialojen välillä. (McKinsey Global Institute, 2011.)

Big Datalle tyypillisiä tunnusmerkkejä ovat ”volume”, ”variety” ja ”velocity” jotka voidaan kääntää karkeasti määrä, monimuotoisuus ja nopeus. (Gartner Inc, 2011). Lisäksi jotkin tahot lisäävät näihin tunnusmerkkeihin lisäksi myös sellaiset piirteet kuten ”veracity” (Villanova University, 2016) eli totuudenmukai-

suus,”variability” (SAS Institute Inc, 2016) eli vaihtelevuus ja ”complexity” (SAS Institute Inc, 2016) eli monimutkaisuus.

Esimerkkinä Big Datan aiheuttamista haasteista, liittyen tiedon valtavaan määrään, voidaan mainita Facebook. Facebook ottaa vastaan joka päivä 500 teratavua uutta tietoa. Toisena esimerkkinä voidaan mainita lentotietojen generointi. Esimerkiksi Boeing 737 tyyppinen lentokone tuottaa yhdellä lennolla yhdysvaltain halki 240 teratavua lentotietoja. (MongoDB Inc, 2016.)

Uuden tiedon syntymisnopeuteen liittyvistä haasteista esimerkkeinä ovat clickstream sekä verkkomainokset. Nämä rekisteröivät käyttäjien toimintaa miljoonien “tapahtumien” sekuntivauhdilla. Toisena esimerkkinä voidaan mainita korkean taajuuden osakekauppa-algoritmit jotka raportoivat markkinoiden muutoksia mikrosekunneissa. (MongoDB Inc, 2016.)

Big Data data luo haasteita myös monimuitoisuutensa takia, sillä se ei ole pelkästään numeroita ja tekstiä. Big Data voi koostua myös esimerkiksi videoista, äänitiedostoista, strukturoimattomasta tekstistä ja sosiaalisen median viesteistä. Perinteisiä tietokantajärjestelmiä ei ole tehty käsittelemään strukturoimatonta tietoa, vaan ne käsittelevät tarkasti määriteltyjä strukturoituja rakenteita. Perinteiset tietokantajärjestelmät ovat myös suunniteltu toimimaan yhdellä palvelimella, mikä tekee kapasiteetin lisäämisestä hankalaa ja kallista. (MongoDB Inc, 2016.)

Bill Inmon määrittelee Big Data alustan niin että kyseinen järjestelmä kykenee säilyttämään suurta määrää tietoa edullisissa tallennuslaitteissa. Lisäksi järjestelmässä tietojen prosessointi tehdään noudattaen “Roman census” metodia ja tiedot tallennetaan alustalle strukturoimattomassa formaatissa. (Inmon, 2013.)

Vastaavasti MongoDB:n esittämän huomattavasti väljemmän määritelmän mukaan Big Datan kohdalla ei puhuta yksittäisestä teknologiasta vaan pikemminkin trendistä joka käsittää suuren määrän eri liike-elämän osa-alueita ja eri teknologioita. Tämän määritelmän mukaan Big Datalla viitataan teknologioihin ja mene-

telmiin, joilla käsitellään sellaisia tietomassoja jotka ovat liian massiivisia, nopeasti muuttuvia tai liian monimuotoisia, perinteisillä teknologioilla, taidoilla ja infrastruktuurilla käsiteltäväksi. (MongoDB Inc, 2016.)

MogoDB:n esittämässä määritelmässä Big Data jaetaan kahteen eri luokkaan, jotka ovat operatiivinen Big Data ja Analyyttinen Big Data (Taulukko 1). Operatiivisen Big Datan järjestelmät on pääasiallisesti tarkoitettu reaaliaikaisten ja interaktiivisten työtehtävien suorittamiseen, joissa tietoa yleensä otetaan vastaan ja tallennetaan. Analyyttisen Big Datan järjestelmät puolestaan on tarkoitettu monimutkaisten analyysien tekemiseen kerätyistä tiedoista jälkeenpäin. Tämän määritelmän mukaan operatiiviset ja analyyttiset järjestelmät ovat toisiaan täydentäviä järjestelmiä, jotka ovat suunniteltu vastaamaan täysin eri haasteisiin, ja joita useimmissa tapauksissa käytetään rinnakkain. (MongoDB Inc, 2016.)

Taulukko 1 Operatiivisen ja analyyttisen Big Datan ominaisuudet (MongoDB Inc, 2016)

	Operational	Analytical
Latency	1 ms - 100 ms	1 min - 100 min
Concurrency	1000 - 100,000	1 - 10
Access Pattern	Writes and Reads	Reads
Queries	Selective	Unselective
Data Scope	Operational	Retrospective
End User	Customer	Data Scientist
Technology	NoSQL	MapReduce, MPP Database

Operatiiviset Big Data järjestelmät ovat kehitetty käyttötapauksiin, joissa tärkeimpiä järjestelmän ominaisuuksia ovat matala vasteaika, sekä kyky käsitellä useita samanaikaisia pyyntöjä. Esimerkkeinä Operatiivisista Big Data alustoista voidaan mainita muun muassa dokumentti kannat, avain-arvo kannat, graph kannat ja wide-column kannat. Useimmat NoSQL kannat on optimoitu tiettyyn

käyttötarkoitukseen. Tietojen käsittelyn lisäksi operatiivisten järjestelmien tulee tarjota jonkinasteista reaaliaikaista tietoa järjestelmän tilasta. (MongoDB Inc, 2016.)

Analyyttiset järjestelmät taas ovat kehitetty pitäen silmällä suoritustehoa, ja kykyä käsitellä poikkeuksellisen monimutkaisia kyselyjä jotka koskevat suurinta osaa, ellei jopa kaikkea järjestelmän sisältämää tietoa. Tällaisia järjestelmiä ovat esimerkiksi MPP tietokantajärjestelmät ja MapReduce ratkaisut. (MongoDB Inc, 2016.)

3.2 Big Datan merkitys

McKinsey Global Instituten mukaan Big Datalla on valtava taloudellinen merkitys niin julkisella kuin yksityiselläkin sektorillakin. Esimerkiksi hyödyntämällä Big Data tehokkaasti Yhdysvaltojen terveydenhuoltojärjestelmän toiminnassa voitaisiin saavuttaa 300 miljardin dollarin vuosittaiset säästöt. Tästä summasta kaksi kolmasosaa tulisi terveyden huolto kustannusten pienentymisestä kahdeksalla prosentilla. Toisena esimerkkinä Big Datan merkityksestä voidaan mainita kehittyneet eurooppalaiset taloudet, jotka voisivat säästää valtion hallinnosta 100 miljardia euroa yksistään parantuneen tehokkuuden myötä. Tähän arvioon ei ole laskettu Big Datan potentiaalisia etuja verotuksen aukkojen, petosten tai virheiden havaitsemisessa. (McKinsey Global Institute, 2011.)

Myös yksityisellä sektorilla Big Datalla on suuri rooli eron saamiseksi kilpailijoihin nähden. Jälleenmyyjä, joka hyödyntää Big Datan koko potentiaalia, voi parantaa liikevoittomarginaalia jopa 60%. Big Data myös tulee luomaan kokonaan uusia kasvun mahdollisuuksia ja yrityksiä, jotka kokoavat ja analysoivat tietoa. (McKinsey Global Institute, 2011.)

On kuitenkin tärkeää pitää mielessä, että Big Datan hyödyntämiseksi tarvitaan huomattavaa analyyttistä ja tietohallinnollista osaamista. Yksistään yhdysvaltojen alueella tullaan tarvitsemaan 140-190 tuhatta henkilöä joilla on vahva analyyttinen osaaminen sekä 1,5 miljoonaa muuta henkilöä hallinnollisiin sekä analyyttisiin tehtäviin. (McKinsey Global Institute, 2011) Big Data -osaajien tarve

tulee kuitenkin kasvamaan tulevaisuudessa. (Marr, 2016). Lisäksi Big Datan hyödyntämistä varten tarvitaan oikeanlainen infrastruktuuri. (McKinsey Global Institute, 2011).

3.3 NoSQL

NoSQL tietokantajärjestelmä on kaikessa yksinkertaisuudessaan ei relationaalinen ja useimmissa tapauksissa hajautettu tietokantajärjestelmä, joka pyrkii tarjoamaan ratkaisun valtavien ja koostumukseltaan vaihtelevien data massojen tehokkaaseen organisointiin ja analysointiin. NoSQL tietokannoista on tullut ensisijainen vaihtoehto relationaalisille kannoille skaalautuvuuden, saatavuuden ja virheidensietokyvyn ansiosta. (DataStax, 2015.) Useimmat NoSQL tietokantajärjestelmät perustuvat myös avoimeen lähdekoodiin. (Sandborg, 2012).

NoSQL tietokantajärjestelmät ovat yleisesti ottaen relationaalisia järjestelmiä joustavampia, sillä useimmat niistä ovat skeemattomia järjestelmiä. NoSQL kannat on suunniteltu niin että ne skaalautuvat horisontaalisesti (DataStax, 2015; Sandborg, 2012.) osittamalla ja replikoimalla tietoa useille laitteille. Useimmissa tapauksissa ositukseen käytettävää menetelmää ei ole mahdollista määritellä etukäteen, vaan yleensä osittaminen tapahtuu automaattisesti. Tyypillisesti osittaminen suoritetaan avaimen perusteella siinä vaiheessa, kun taulun koko kasvaa riittävän suureksi. (Sandborg, 2012.)

Suuremman suorituskyvyn, skaalautuvuuden ja saatavuuden saavuttamiseksi NoSQL kantojen kohdalla on usein jouduttu luopumaan monista ACID transaktioiden ominaisuuksista. Usein NoSQL kannat noudattavatkin väljempää BASE oikeellisuusmallia. Toisin kuin ACID mallissa, jossa tietokannan tulee aina olla oikeellinen, BASE mallissa on riittävää, että oikeellisuus saavutetaan jossakin vaiheessa. (Sandborg, 2012.)

Transaktioiden suorituksessa hajautetut järjestelmät usein käyttävät kaksivaiheista transaktioiden sitoutumiskäytäntöä (two-phase commit) 2PC, jonka avulla pyritään varmistamaan hajautetun transaktion ACID ehtojen mukainen luotetta-

vuus. Toteutus tapahtuu kahdessa eri vaiheessa. Ensimmäisessä vaiheessa transaktioita koordinoiva taho varmistaa, että muut klusterin laitteet ovat valmiit sitoutumaan. Jos kaikki transaktioon osallistuvat laitteet ovat valmiita, koordinoiva taho pyytää toisessa vaiheessa jokaista osapuolta sitoutumaan. Mikäli jokin osapuoli joutuu keskeyttämään transaktion tai päätöstä ei saada määräajassa, koordinoiva taho peruuttaa transaktion. (Sandborg, 2012.)

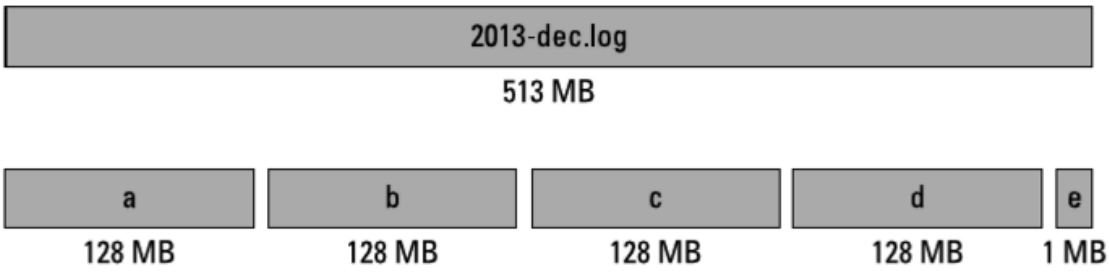
4 Big Data -alustojen vertailu

4.1 Hadoop

Hadoop on Apache Software Foundationin sponsoroima (Middleton, 2011) avoimen lähdekoodin ohjelmistokehys, joka mahdollistaa suurien tietomäärien rinnakkaisen prosessoinnin useilla laitteilla samanaikaisesti. (Apache Software Foundation, 2015) Hadoop rakentuu neljästä olennaisesta osasta jotka ovat: Hadoop common, Hadoop distributed filesystem, Hadoop YARN sekä Hadoop MapReduce. (Apache Software Foundation, 2015). Hadoop projektin alkuperäisenä tavoitteena oli luoda avoimen lähdekoodin MapReduce sovellus. Hadoopin käyttämät HDFS tiedostojärjestelmä ja MapReduce ratkaisu pohjautuvat Googlen GFS tiedostojärjestelmään ja Googlen MapReduce järjestelmään, lukuunottamatta sitä seikkaa että Hadoop on kehitetty Java kielellä, toisin kuin Googlen kehittämät ratkaisut joiden ohjelmointikielenä on C++. (Middleton, 2011.) Hadoop oli alun perin tarkoitettu käytettäväksi Linux klustereissa, (Middleton, 2011) mutta versiosta 2.2 eteenpäin Hadoop sisältää natiivisti tuen Windows käyttöjärjestelmille, (Hadoop Wiki, 2014) ja esimerkiksi Hortonworks Data Platform on saatavilla Windows käyttöjärjestelmille. (Bakhshi, 2014)

Hadoop Common, joka tunnetaan myös nimellä hadoop core, on paketti joka sisätää Hadoop ohjelmistokehyksen eri osien yhteisesti käyttämiä kirjastoja sekä työkaluja, kuten Hadoopin alla olevan käyttöjärjestelmän ja sen tiedostojärjestelmän abstraktoinnin, lisäksi hadoop common sisältää mm. tiedostot ja scriptit joita tarvitaan hadoopin käynnistämiseen. Edelllä mainittujen elementtien lisäksi Hadoop common pakettiin sisältyy lähdekoodi, dokumentaatio sekä contribution osio joka sisältää Hadoop yhteisön tuottamia projekteja. (Janalta Interactive Inc, 2015.)

Hadoop sisältää hajautetun tiedostojärjestelmän nimeltä Hadoop Distributed File System eli lyhennettynä HDFS, joka pohjautuu GFS eli Google File System tiedostojärjestelmään. Hadoopin käyttämä HDFS tiedostojärjestelmä noudattaa master/slave tyyppistä rakennetta mikä muodostuu yhdestä master laitteesta, jonka tehtävänä on hallita tiedostojärjestelmän nimiavaruutta, sekä hallita käyttäjien pääsyä tiedostoihin. Tätä laitetta kutsutaan nimellä Namenode. Lisäksi HDFS järjestelmään kuuluu Datanodeja. Datanodet vastaavat tietojen hallinnoinnista omalla laitteellaan. (Middleton, 2011.) HDFS poikkeaa perinteisistä tiedostojärjestelmistä, niin että toisin kuin perinteisissä tiedostojärjestelmissä, HDFS tiedostojärjestelmässä tallennettavat tiedot hajotetaan pieniin osiin, joita kutsutaan blokeiksi (Kuva 1). Oletusarvoisesti jokainen blokki on kooltaan 128 megatavua, ja kaikki blokit ovat aina saman kokoisia viimeistä lukuun ottamatta. Blokkien kokoa voidaan HDFS järjestelmässä muuttaa, joko vaihtamalla järjestelmän oletusarvoa tai määrittämällä se tiedostokohtaisesti. (deRoos;Zikopoulos;Melnyk;Brown;& Coss, 2014.) Sitten blokit jaetaan Hadoop klusteriin kuuluvien laitteiden kesken. HDFS tiedostojärjestelmä ottaa huomioon vikatilanteiden mahdollisuuden ja oletusarvoisesti tallentaa samat tiedot kolmelle klusterin laitteelle. Redundanssia voidaan HDFS järjestelmässä säätää tiedosto tasolta aina koko ympäristön tasolle. (IBM, 2016.)

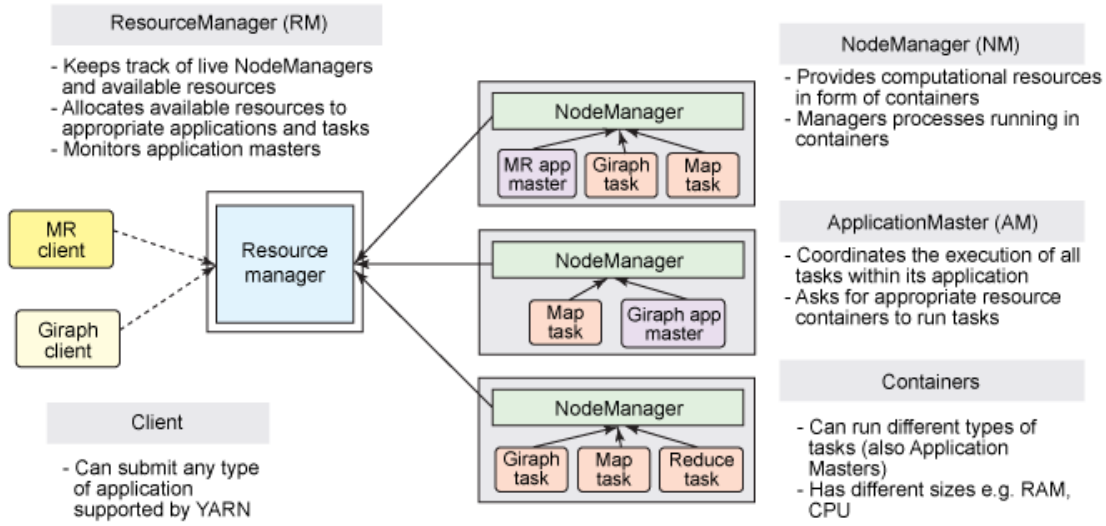


Kuva 1. Tiedoston jakaminen blokeiksi (deRoos;Zikopoulos;Melnyk;Brown;& Coss, 2014)

Apache Hadoop YARN on lyhenne sanoista Yet Another Resource Negotiator. YARN on klusterin hallinta teknologia, jota alkuun kutsuttiin yksinkertaisesti uudelleen suunnitelluksi resurssien hallinta työkaluksi. Nykyisin YARN:ia kutsu-

taan suuren mittakaavan hajautetuksi käyttöjärjestelmäksi Big Data sovelluksille. (TechTarget, 2013.)

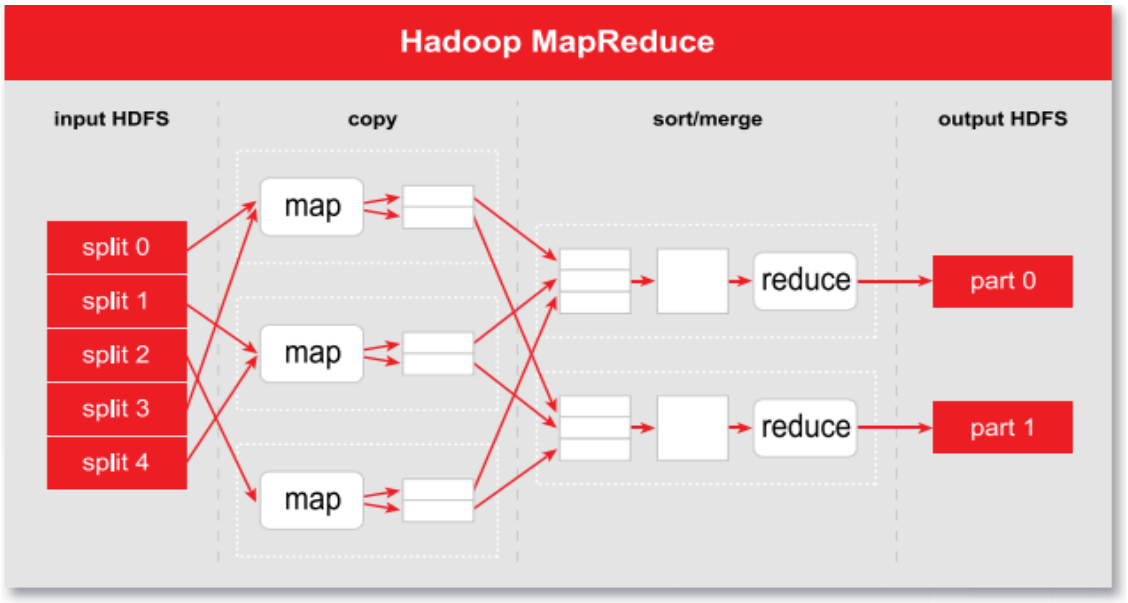
YARN arkkitehtuurissa Resource Manager toimii master prosessina, joka järjestelee resursseja suoritettaville prosesseille (Kuva 2). Sen tehtäviin kuuluu myös pysyä selvillä klusteriin kytketyistä laitteista ja käytettävissä olevista resursseista, ja päättää mitkä käyttäjän syöttämät tehtävät saavat käyttöönsä näitä resursseja ja milloin ne saavat niitä käyttöönsä. (IBM, 2014.)



Kuva 2 YARN sovelluksen arkkitehtuuri (IBM, 2014)

Käyttäjän lähetettyä sovelluksen käynnistetään prosessi nimeltä ApplicationMaster, joka koordinoi kaikkia sovellukseen liittyviä tehtäviä. Tämä pitää sisällään tehtävien valvonnan, epäonnistuneiden tehtävien uudelleenkäynnistämisen, sekä hitaiden prosessien spekulatiivisen suorittamisen. The ApplicationMaster ja sen hallinnoimaan sovellukseen kuuluvat tehtävät on eristetty omaksi yksikökseen, jota kutsutaan nimellä “container”. NodeManager taas vastaa näiden “containerien” hallinnoinnista. Näiden containerien koko riippuu siitä, kuinka paljon niihin on sijoitettu resursseja. (IBM, 2014.)

YARN arkkitehtuuriin kuuluva ApplicationMaster sovellus kykenee suorittamaan minkä tyyppisen tahansa tehtävän containerin sisällä. Esimerkikiksi MapReduce applicationMaster kykenee suorittamaan map tai reduce tehtäviä ja Giraph ApplicationMaster kykenee suorittamaan Giraph tehtäviä. Käyttäjät voivat myös luoda omia ApplicationMaster sovelluksia suorittamaan käyttäjän määrittelemiä tehtäviä. (IBM, 2014.)



Kuva 3 MapReducen tietovirta (Middleton, 2011)

MapReduce on järjestelmäarkkitehtuuri, joka on suunniteltu suurten tietomäärien prosessointiin ja analysointiin klusteroiduilla laitteilla. Google kehitti alkuperäiseen MapReduce ratkaisuunsa työnteon tehokkuuden parantamista varten Sawzall ohjelmointikielen. Sawzall kieltä vastaava ratkaisu Hadoopissa on Yahoon kehittämä Pig ohjelmointikieli. (Middleton, 2011.)

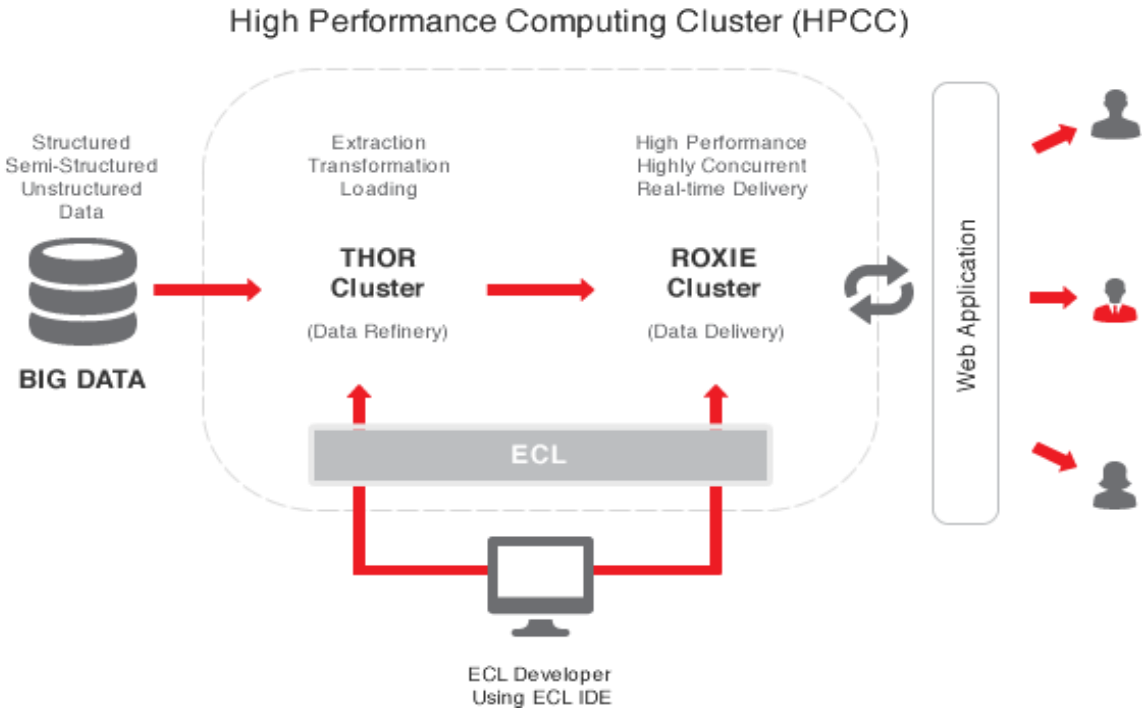
MapReduce sovellukset toimivat vaiheittain (Kuva 3). Ensimmäisessä vaiheessa etsitään prosessoitavat tiedot HDFS järjestelmän sisältämistä blokeista. Sen jälkeen käydään jokainen prosessoitavien tietojen sisältämä tietue läpi. Tätä kutsutaan map vaiheeksi ja sovelluksen osaa, joka suorittaa tämän vaiheen sanotaan mapperiksi. Tämän jälkeen voidaan paikallisesti yhdistellä yksittäisten mapperien tulokset, mutta tätä vaihetta ei aina suoriteta. Tämän jälkeen mapperien tulokset ryhmitellään järjestelmän ositussääntöjen mukaisesti. Sitten tiedot siirretään MapReduce sovelluksen reducer osien käsiteltäväksi, jotka tuottavat

mapperien vastauksista yhden yhtenäisen tuloksen. (deRoos;Zikopoulos;Melnyk;Brown;& Coss, 2014.)

Maddenin mukaan Mapreduce ja Hadoop skaalautuvat hyvin suuriin tieto määriin, mutta toteaa että Mapreduce ja Hadoop tarvitsevat enemmän kapasiteettia saman tietomäärän käsittelemiseen, kuin tietokantapohjaiset järjestelmät, koska kyseisistä järjestelmistä puuttuvat tietokantajärjestelmien kehittyneet kyselyjen prosessointi menetelmät. Hän myös toteaa, että nämä ratkaisut kärsivät samanlaisista rajoituksista kuin relationaaliset järjestelmät. Yhtenä suurena ongelmana kyseisissä järjestelmissä on se, että Hadoop ja Mapreduce tarjoavat tehokkaan infrastruktuurin tietojen käsittelyyn, mutta ei sen hallinnointiin. Tämä tarkoittaa hänen mukaansa, että nämä järjestelmät tarjoavat pääsyn tiedostokokoelmiin, mutta on kokonaan käyttäjän vastuulla varmistaa tiedostojen eheys, ja ylläpitää niitä ja varmistaa että sovellukset jotka hyödyntävät kyseisiä tietoja toimivat siinäkin tapauksessa, että tietojen rakenne muuttuu. Monia edellä mainittuja ongelmia on mahdollista kuitenkin korjata lisäämällä tuki tietojen hallinnointia varten Hadoop alustan päälle, esimerkkeinä tällaisista ratkaisuista voidaan mainita Hive ja Hbase järjestelmät. Ongelmallista näiden järjestelmien kannalta on myös se että kehittäjät eivät ole lähteneet ratkaisemaan Big Datan ytimessä olevia ongelmia vaan he ovat pikemminkin rakentaneet uudelleen tietokanta-kantajärjestelmiä. Lisäksi kyseiset järjestelmät on suunniteltu käsittelemään suuria määriä replikoitua levypohjaista tietoa, mistä johtuen kyseiset järjestelmät kärsivät suuresta latenssista. (Madden, 2012.)

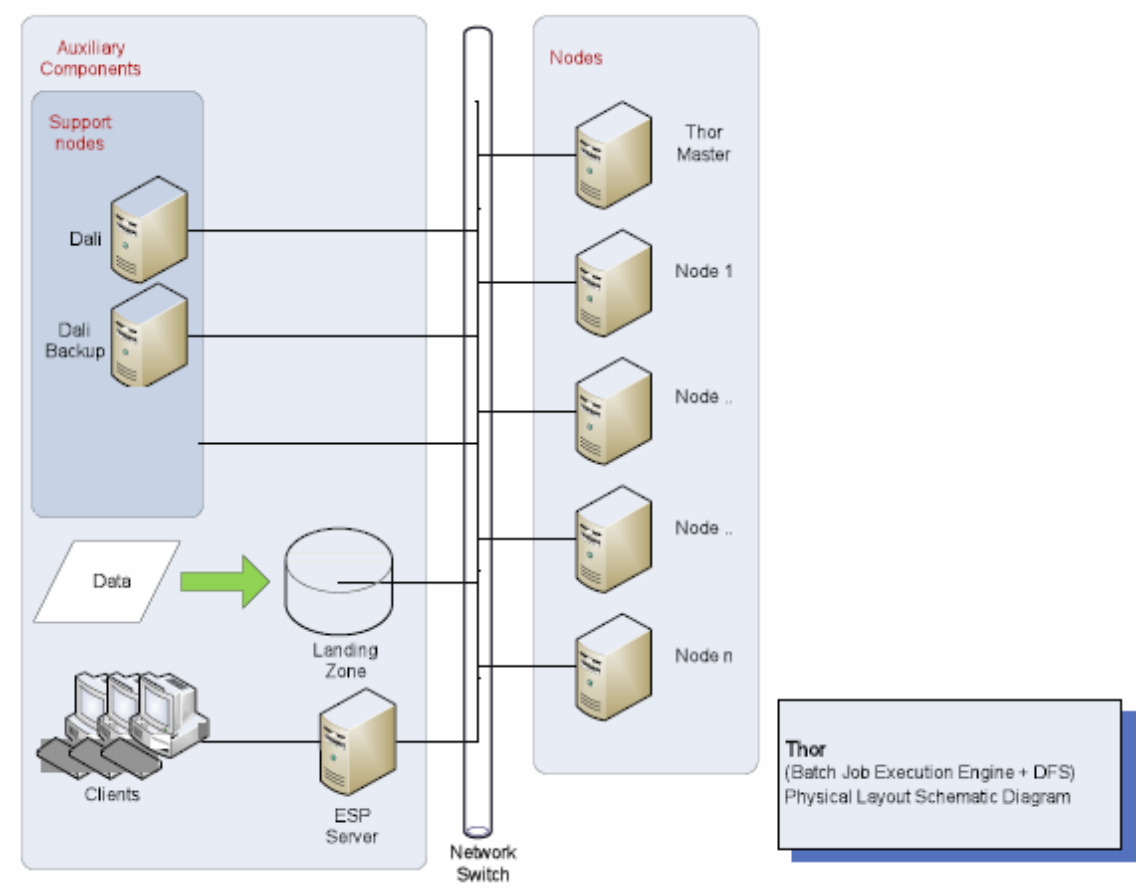
4.2 HPCC

HPCC Systems platform koostuu kahdesta integroidusta mutta eri tarkoitukseen tehdystä klusterista (Kuva 4). Molemmat järjestelmät toimivat hajautetusti ja molemmilla on oma erillinen tiedostojärjestelmä. Sekä Thor että Roxie käyttävät ECL:ää eli Enterprise Control Languagea, Thorin tapauksessa analytiikka tehtävien luomiseen ja Roxien tapauksessa kyselyjen tekemiseen. ECL kielinen koodi kääntyy optimoiduksi C++:si ja sen ominaisuuksia voidaan laajentaa tarvittaessa käyttäen C++ kirjastoja. (HPCC Systems, 2016.)



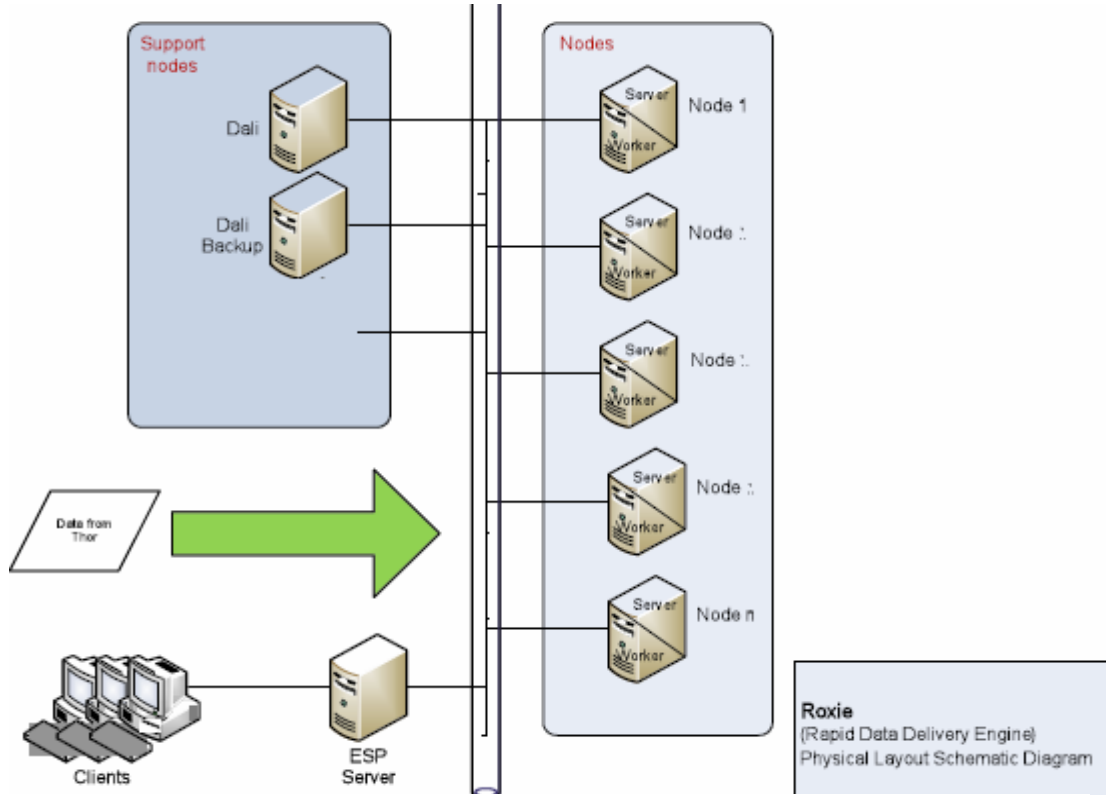
Kuva 4 HPCC järjestelmän rakenne (Middleton, 2011)

Back-end-klusteria kutsutaan nimellä Thor. Thor-klusterin tehtävänä on Big Datan puhdistaminen, muokkaaminen, linkittäminen, ja indeksointi. Thor klusteri kykenee rinnakkaisprosessointiin ja klusteri skaalautuu yksittäisestä laitteesta aina tuhansiin laitteisiin asti. Thor klusteri noudattaa tyypillistä master-slave-rakennetta, jossa orjalaitteet tarjoavat tallennuskapasiteettia ja laskentatehoa, ja master vastaa orjien hallinnoinnista, tehtävien koordinoinnista sekä tilannetietojen raportoinnista. (HPCC Systems, 2016.)



Kuva 5 Thor klusterin rakenne (Middleton, 2011)

Thor toimii hajautetulla tietue pohjaisella tiedostojärjestelmällä. Kun klusteri ottaa vastaan joko määrämittaisista tai muuttuvan mittaisista tietueista koostuvan tiedoston, se pilkotaan klusterin laitteille, niin että kaikki laitteet saavat suunnilleen saman määrän tietoja. Tietueita ei kuitenkaan koskaan hajoteta. Thor klusterin tiedostojärjestelmä kestää hyvin vikatilanteita tiedostojen replikoinnin ansiosta. Oletusarvoisesti jokaisesta tiedoston osasta on ainakin yksi kopio, mutta tiedostojen replikointia on mahdollista säätää tarpeen mukaan. (HPCC Systems, 2016.) Thor-klusterin tiedostojärjestelmän nimipalvelut ja tallennettujen tiedostojen metatiedot säilytetään erityisellä Dali-palvelimella (Kuva 5). Thorin erityispiirteenä on mahdollisuus jakaa klusterin sisältämä data uudelleen, esimerkiksi tietyn avaimen tai kentän perusteella, luomalla ECL tehtävä. Dali nimipalvelin tallentaa tiedostojärjestelmän tiedot dynaamiseen varastoon, jossa tiedot on järjestetty tiedostojärjestelmän rakennetta vastaavaan hierarkkiseen malliin. (Middleton, 2011.)

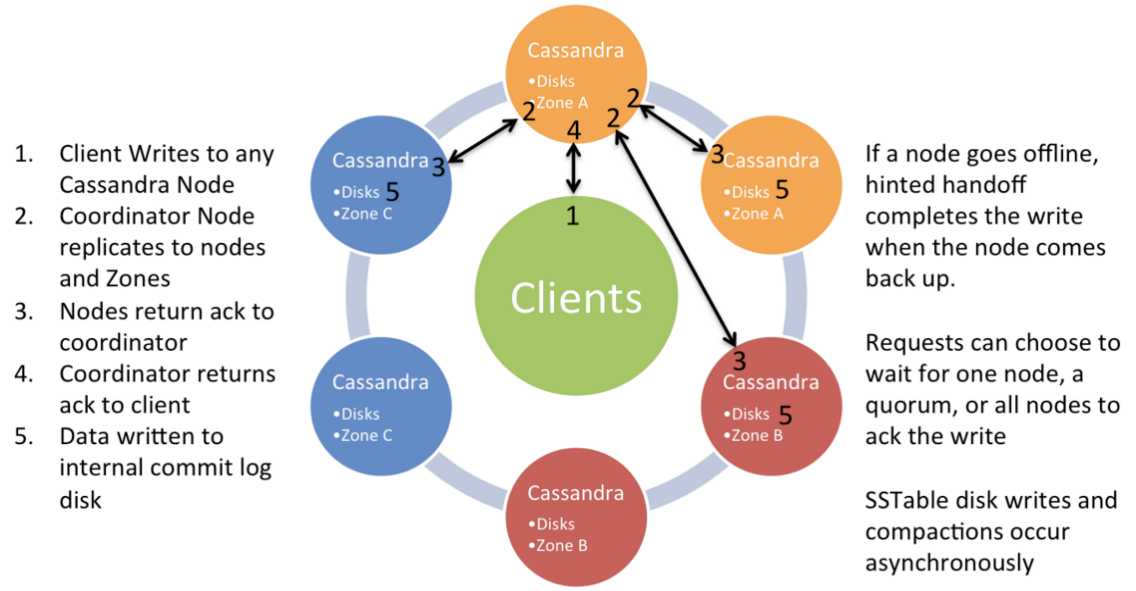


Kuva 6 Roxie klusterin rakenne (Middleton, 2011)

Toista front-end klusteria kutsutaan nimellä Roxie. Roxien tehtävänä on tarjota mahdollistaa reaaliaikaiset kyselyt prosessoiduista tiedoista ja tarjota tietovarastointi palveluita. Käyttäjän on myös mahdollista halutessaan hyödyntää Roxien reaaliaikaisia kyselyjä The Enterprise Services Platform (ESP) webpalvelualustan läpi käyttäen tuettuja SOAP, XML, HTTP, and REST protokollia. Jokaisella Roxie klusterin laitteella toimii Server prosessi ja Agent prosessi. Server prosessin tehtävänä on käsitellä käyttäjiltä tulevat kysely pyynnöt, jakaa prosessointi tehtävät oikeille Agent prosesseille klusterissa sekä koota tulokset ja palauttaa vastaus asiakaskoneelle. Tiedot Roxie klusterille Thor klusterilta jossa ne ensin indeksoidaan (Kuva 6). Roxie klusteri käyttää hajautettua indeksipohjaista tiedostojärjestelmää, joka pohjautuu mukautettuun B+ puu rakenteeseen. Roxielle tehdyt kyselyt voivat sisältää join komentoja, tai muita monimutkaisia tietojen transformaatioita. (HPCC Systems, 2016.)

4.3 Cassandra

Cassandra on avoimen lähdekoodin tietokannan hallintajärjestelmä, joka on suunniteltu käsittelemään suuria määriä tietoa hajautetussa järjestelmässä. Alun perin järjestelmän kehittämisestä vastasi Facebook, mutta nykyisin järjestelmän kehittäminen on siirtynyt Apache Software Foundationin vastuulle. Cassandran tietokantamalli on tyypiltään Wide-column store ja pohjautuu Amazonin Dynamo sekä Googlen Big Table järjestelmiin. (DataStax, 2015.) Cassandra on toteutettu Java ohjelmointikielellä. (DB-Engines, 2016).



Kuva 7 Cassandran rakenne ja tietovirrat (DataStax, 2015)

Cassandra klusterin arkkitehtuuri eroaa tyypillisestä siinä, että Cassandra klusteriin kytkettyjä laitteita ei ole jaettu master ja slave tyyppeihin, vaan kaikki laitteet ovat tasa-arvoisia, ja klusteri noudattaa kehä rakennetta (Kuva 7). (DataStax, 2015) Tämä arkkitehtuuri pohjautuu Amazonin DynamoDB:hen, joka kuten Cassandra käyttää symmetristä rakennetta, jossa kaikki klusterin laitteet ovat saman arvoisia. (Grehan, Review: Cassandra lowers the barriers to big data, 2014) Cassandran käyttämän arkkitehtuurin ansiosta sillä ei ole ”single point of failurea”. (DataStax, 2015) Single point of failure tarkoittaa sellaista järjestelmän osaa, jonka kaatuminen pysayttää järjestelmän toiminnan. (Techopedia Inc, 2016) Cassandra myös käyttää Dynamon kaltaista hashing menetelmää tietojen osiointiin ja replikointiin klusterin laitteille. Cassandran käyttämässä hashing menetelmässä luodaan 128 bittinen hash joka määrittelee tietojen sijainnin klusterissa. Lisäksi tietojen sijoittamisessa auttaa sovellus jota kutsutaan nimellä ”snitch”, sen tehtävänä on pitää kirjaa verkon laitteiden IP-osoitteista, fyysisestä sijainnista ja replikoitaessa tietoja varmistaa, että tiedot replikoidaan eri räkissä olevalle laitteelle, jotta vikatilanteessa tietoja ei häviä vaan käytettävissä on aina kopio. (Grehan, Review: Cassandra lowers the barriers to big data, 2014.)

Cassandran suurimpina etuina ovat arkkitehtuuri, joka tekee uusien laitteiden lisäämisen klusteriin helpoksi ja suoraviivaiseksi, (Grehan, Review: Cassandra lowers the barriers to big data, 2014) sekä CQL kieli, joka muistuttaa läheisesti paljon käytettyä SQL kieltä (Kuva 8), mikä madaltaa huomattavasti oppimiskynnystä Cassandran kohdalla. (Asay, 2014)

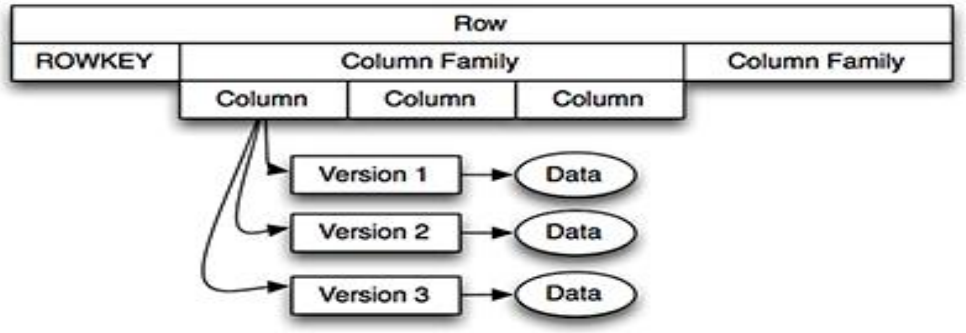
```
-- Inserts or updates
INSERT INTO Standard1 (KEY, col0, col1)
VALUES (key, value0, value1)

                    vs.

-- Inserts or updates
UPDATE Standard1
SET col0=value0, col1=value1 WHERE KEY=key
```

Kuva 8 Esimerkki CQL ja SQL kielten yhteneväisyyksistä (Evans, 2011)

4.4 Hbase

Hbase on avoimen lähdekoodin NOSQL tietokantajärjestelmä. (Haines, 2014). Samoin kuin Cassandra, Hbase on Googlen Bigtable-järjestelmään pohjautuva (Asay, 2014) Wide-column store -tyyppinen tietokantajärjestelmä (DB-Engines, 2016), ja se on suunniteltu skaalautumaan tehokkaasti. Hbasen suosiota osittain selittää se, että se on osa Hadoop-projektia. (Asay, 2014)

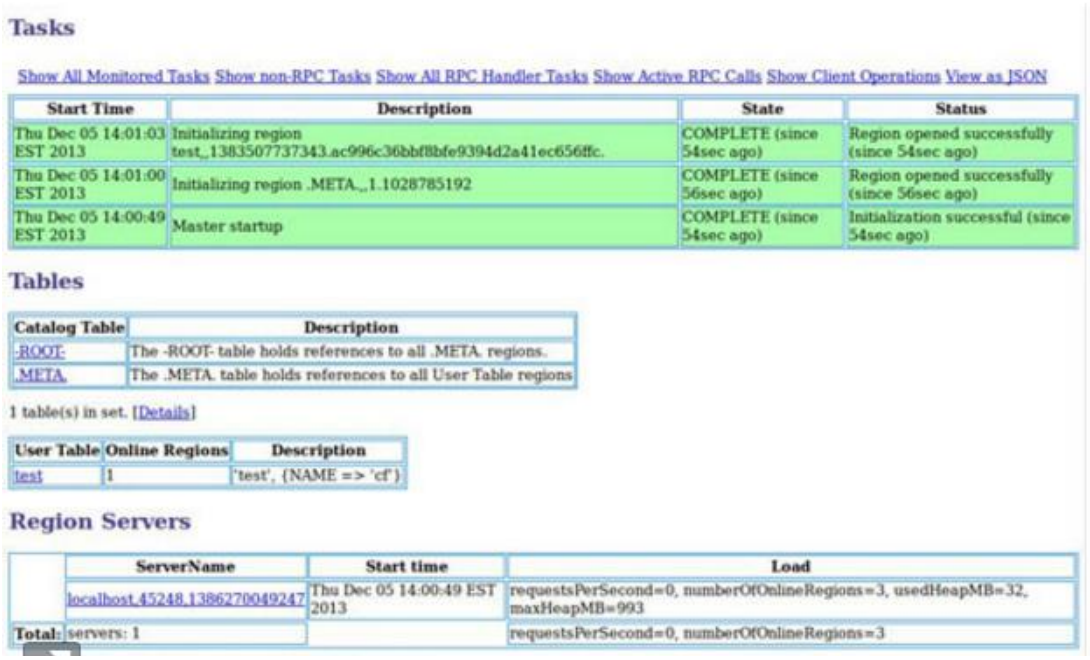
Hbase noudattaa neliulotteista tietokantamallia (Kuva 9). Kyseisessä mallissa tietorakennetta määrittää neljä keskeistä attribuuttia, jotka ovat ”rowkey”, ”column family”, ”column qualifier” ja versio. (Haines, 2014.)



Kuva 9 HBasen neliulotteinen tietomalli (Haines, 2014)

Hbasen tärkeimpiä ominaisuuksia ovat skaalautuvuus, luotettavuus ja skeeman joustavuus. Vaikka Hbasen ja Cassandran tyyppisissä tietokantajärjestelmissä taulut ja column familyt täytyy määritellä ennakkoon, voidaan sarakkeita lisätä aina lennosta. Hbase tarjoaa rivi tasolla vahvan menetelmän tietojen yhdenmukaisuuden varmistamiseen, (Grehan, Review: HBase is massively scalable -- and hugely complex, 2014.) ja täyttääkin ACID määritelmän vaatimukset rivitasolla. (Grehan, Big data showdown: Cassandra vs. HBase, 2014), Lisäksi Hbase sisältää sisäänrakennetun versioinnin ja relationaalisten kantojen käyttämiä stored procedureja muistuttavan coprocessor ominaisuuden. (Grehan, Review: HBase is massively scalable -- and hugely complex, 2014)

Hbase järjestelmässä on kahta eri tyyppiä olevia Coprocessoreja. Ensinnäkin on observer Coprocessoreita ja sitten on endpoint Coprocessoreita. Observer on käyttäjän määrittämä Java luokka, joka määrittelee mitä metodeja suoritetaan missäkin tilanteessa. Näin ollen Observer muistuttaa läheisesti relationaalisten järjestelmien trigger-ominaisuutta. Endpoint tyyppinen Coprocessor toimii taas hyvin samankaltaisella tavalla kuin stored procedure, eli kyseessä on funktio, jota voidaan kutsua esimerkiksi Observerin kautta. Coprocessor toimii niin että käyttäjän ohjelmoimaa koodia suoritetaan osana Hbasen Region Server ja Master prosesseja. (Grehan, Review: HBase is massively scalable -- and hugely complex, 2014.)Hbase päinvastoin kuin Cassandra on optimoitu tietojen



Kuva 10 Hbasen web-käyttöliittymä (Grehan, Big data showdown: Cassandra vs. HBase, 2014)

such as the node's execution history, tables managed by the node, and region servers in the master's domain.

lukemista eikä kirjoittamista varten, sillä Hbase on ensisijaisesti tarkoitettu suurista datamassoista tehtäviin kyselyihin ja toisin kuin Cassandra, joka on ”eventually consistent”, Hbase pyrkii varmistamaan tietojen eheyden, mikä hidastaa kirjoitusnopeutta. (Grehan, Review: HBase is massively scalable -- and hugely complex, 2014.)

Hbase käyttää erillistä Zookeeper sovellusta laitteiden väliseen kommunikointiin, toisin kuin Cassandra, jossa on sisäänrakennettu Gossip protokolla. Hbasen riippuvuus erillisestä sovelluksesta saattaa heikentää järjestelmän luotettavuutta ja haitata vianmääritystä. (Grehan, Big data showdown: Cassandra vs. HBase, 2014) Lisäksi Hbase master palvelimella on Web pohjainen käyttöliittymä, joka tarjoaa tietoja klusterin tilasta (Kuva 10). (Grehan, Review: HBase is massively scalable -- and hugely complex, 2014.)

4.5 MongoDB

MongoDB on avoimen lähdekoodin dokumenttipohjainen tietokanta. MongoDB tallentaa tiedot kokoelmina, jotka taas muodostuvat dokumenteista. (Mei, 2013) Dokumenttien tietorakenne muodostuu JSON tyyppisistä kenttä ja avain pareista (Kuva 11). MongoDB:n käyttämästä dokumenttiformaatista käytetään nimitystä BSON. BSON eli binary JSON dokumentti ovat JSON:in binäärimuotoinen esitys johon on lisätty ylimääräisiä data tyyppien määrittelyjä. (MongoDB Inc, 2016.) Poiketen Cassandrasta ja Hbasesta MongoDB käyttää AGPL lisenssiä. (DB-Engines, 2016)

Kuva 11. Esimerkki BSON dokumentista (MongoDB Inc, 2016)

MongDB:n suurimpiin etuihin lukeutuu se että se säilyttää monia SQL:n käyttäjäystävällisistä ominaisuuksista esimerkiksi dynaamiset kyselyt sekä indeksit. (Kovacs, 2016). Huomattavaa on myös ohjelmistokehittäjän näkökulmasta se että MongoDB:n käyttämät dokumentit ovat yhteneviä useiden ohjelmointikielien käyttämien tietorakenteiden kanssa, sekä mahdollisuus taulukkojen ja toisten dokumenttien sisällyttämiseen dokumentin kenttien arvoiksi, mikä vähentää tarvetta relationaalisille kannoille tyypillisten join operaatioille. (MongoDB Inc, 2016).

Skeeman suunnittely määrittelee, kuinka järjestelmä käsittelee tietoa. Perinteiset relationaaliset järjestelmät vaativat, että skeema määritellään ennen kuin mitään tietoa voidaan tallentaa kantaan. Monet NoSQL tietokannat kuten MongoDB sallivat tietojen syöttämisen ilman ennalta määriteltyä skeemaa (MongoDB Inc, 2016.), minkä ansiosta tietokantaa hyödyntävien ohjelmistojen kehittäminen on paljon helpompaa. Lisäksi relationaalisten järjestelmien tiukka skeema aiheuttaa sen, että kaikki kantaan syötettävät tiedot on ensin sovitettava skeeman määrittelemään muotoon. Kehittämistä helpottavat muu muassa sellaiset seikat kuten se, että skeeman suunnitteluun ei tarvitse projektia aloittaessa käyttää aikaa, tämä on merkittävä etu erityisesti suuremmissa projekteissa, joissa tietokannan rakenteeseen saatetaan joutua tekemään muutoksia useita kertoja. Lisäksi mikäli joudutaan tekemään muutoksia skeemaan tarkoittaa se, että muutoksia joudutaan myös tekemään tietokantaa käyttävien sovellusten koodiin. (Lachita, 2014.)

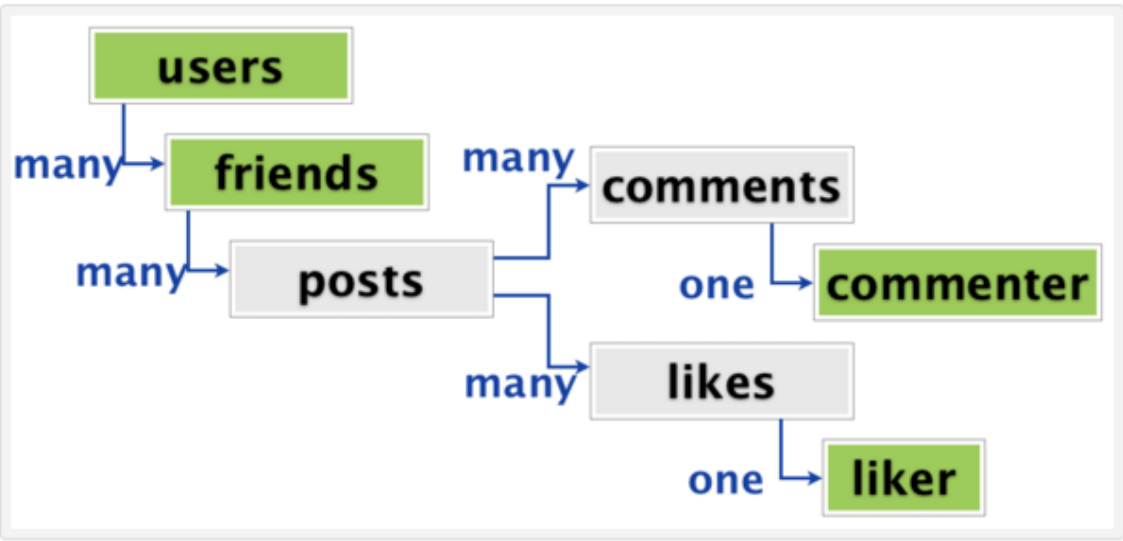
MongoDB ei kuitenkaan ole aina ideaalinen ratkaisu kaikista vahvuuksistaan huolimatta. Ensinnäkin skeeman puuttuminen voi muodostua ongelmalliseksi siitä syystä, että MongoDB:n kokoelmassa olevilla kahdella dokumentilla voi olla eri kentät, ja saman nimiset kentät voivat sisältää eri tietotyypin. Esimerkkinä tästä tilanne, jossa on artists kokoelma, jossa on kaksi rakenteeltaan poikkeavaa dokumenttia (Kuva 12). Kuten kuvasta käy ilmi

```
var artists = [
{
first: "David",
last: "Bowie",
age: "sixty-six"
},
{
first: "John",
middle: "Winston",
last: "Lennon",
age: 30
}
]
```

Kuva 12 Rakenteeltaan poikkeavat dokumentit (Coe, 2013)

vain jälkimmäisessä dokumentissa on middle niminen kenttä ja ensimmäisessä dokumentissa age kentän sisältämä tieto on esitetty tekstinä, kun taas toisessa dokumentissa vastaava tieto on esitetty numeerisessa muodossa. Siltikin kumpikin on MongoDB:n mielestä täysin kelvollisia dokumentteja. Näin ollen MongoDB:n tapauksessa käyttäjälle jää suuri vastuu tiedon johdonmukaisuuden säilyttämisestä. (Coe, 2013.)

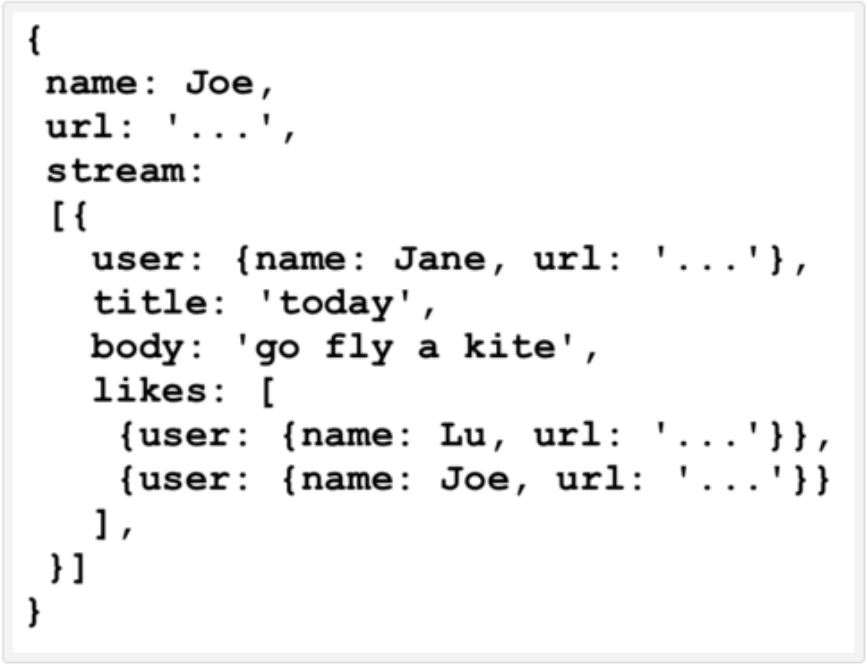
Toinen ongelma on JOIN toiminnon puuttuminen. Esimerkiksi Sarah Mei käsittelee MongoDB:n tietorakenteisiin liittyviä ongelmia blogissaan Why you Should Never Use MongoDB. Hänen tapauksessaan MongoDB tietokantajärjestelmää yritettiin käyttää Facebookin kaltaisessa sosiaalisen median järjestelmässä ja kyseisen projektin aikana havaittiin, että tietyissä tapauksissa MongoDB:n dokumenttipohjainen tietorakenne muodostui haitaksi, sillä se ei tue relationaalisten järjestelmien JOIN operaatioita, eikä se tarjoa vastaavankaltaista ratkaisua. Tässä tapauksessa ongelmana oli tapahtumasyötteen (activity stream) luominen. Kun käyttäjä on kirjautunut järjestelmään, hänelle on tarkoitus näyttää tapahtumasyöte, jossa esitetään kaikkien hänen seuraamiensa henkilöiden postaukset. Jokainen postaus sisältää taas erilaisia tietoja, kuten kuvia, tykkäyksiä ja kommentteja. Tapahtumasyötteen rakenne näkyy Kuva 13. Tapahtumavirran rakenteen haastavaksi tekee se että, jokaisella käyttäjällä on ystäviä, jotka ovat myös tässä tapauksessa käyttäjiä ja myös kommentoijat sekä tykkääjät ovat käyttäjiä. Tämä muodostui ongelmaksi siksi että eri kohdissa dokumenttia viitataan samaan konseptiin eli tässä tapauksessa samaan käyttäjään, sillä sama käyttäjä joka tykkäsi postauksesta voi olla sama käyttäjä joka kommentoi toista



Kuva 13 Tapahtumasyötteen rakenne (Mei, 2013)

postausta tapahtumasyötteessä. (Mei, 2013.)

MongoDB:ssä kyseinen tietorakenne voidaan toteuttaa kahdella eri tavalla, mutta kummassakin on omat ongelmansa. Ensimmäisenä vaihtoehtona on käyttäjän tietojen kopioiminen kaikkiin niihin paikkoihin, joissa käyttäjän tietoja tarvitaan (Kuva 14). Kuten kuvassa näkyy, Joe nimisen käyttäjän tapahtumasyötteessä näkyy Jane nimisen käyttäjän postaus, josta käyttäjät Lu ja Joe ovat tykänneet, ja kaikkien näiden käyttäjien tiedot on kopioitu tapahtumasyötteeseen. Tämän lähestymistavan etuna on se että kaikki tiedot löytyvät aina sieltä missä niitä tarvitaan. Huonona puolena on se että mikäli käyttäjän tietoja päivitetään, täytyy kaikki tapahtumasyötteet käydä läpi, jotta käyttäjän tiedot saadaan päivitettyä ajan tasalle. (Mei, 2013.)



Kuva 14. Ongelman ratkaisu käyttäjän tiedot kopioimalla (Mei, 2013)

Toisena vaihtoehtona on viittausten tekeminen dokumentteihin. Kuten Kuva 15 näkyy, kaikille käyttäjille on annettu id-atribuutti, ja sen sijaan että kaikki käyttäjän tiedot olisi talletettu kaikkiin postauksiin, kommentteihin ja tykkäyksiin, merkitään niihin vain käyttäjän id, joka viittaa tiettyyn käyttäjään. Tämän lähestymistavan etuna on se, että mikäli käyttäjän tietoja muutetaan, tarvitsee muuttaa ainoastaan yhtä dokumenttia. Tämä luo kuitenkin sellaisen ongelman, että koska tietoa on siirretty pois tapahtumasyötteestä, ei sitä voida toteuttaa enää yhdessä dokumentissa, mikä tekee sen toteuttamisesta huomattavasti vaikeampaa ja monimutkaisempaa. (Mei, 2013.)

Kuva 15. Ongelman ratkaiseminen viittauksilla (Mei, 2013)

5 Toteutus

Opinnäytetyön toteutusvaihe aloitettiin perehtymällä työssä käytettävään Wago 750-881 -logiikkamoduuliin ja sen toimintoihin, sekä mahdollisiin rajapintoihin, joilla sensoridataa voidaan ottaa laitteesta ulos. Tähän vaiheeseen saatiin ohjeistusta Karelia-amk:n lehtorilta Miska Piiraiselta.

Vaikka varsinaisten kytkentöjen tekeminen ei sisälly opinnäytetyöhön tehtiin toteutettavan tiedonkeruu menetelmän testausta ja toimintojen mallintamista varten pienimuotoinen sensorikytkentä, joka sisältää valaistusta ja lämpötilaa mittaavat analogiset sensorit, jotka on liitetty Wago-logiikkamoduuliin, sekä kytkimen, jonka kautta itse logiikkamoduuli on kytkettynä tietokoneeseen.

Selvitettäessä mahdollisia työkaluja ja menetelmiä sensoridatan keräämistä varten havaittiin, että laitevalmistajalla on oma ohjelmointiympäristö nimeltä Wago-I/O-PRO. Mahdollisten yhteensopivuusongelmien välttämiseksi päädyttiin käyttämään nimenomaan tätä laitevalmistajan omaa ohjelmointiympäristöä, samalla havaittiin, että tiedon siirtämistä varten on valmistajan tarjoamassa ohjelmointiympäristössä automatisoitu toiminto nimeltä MODBUS Master Configurator, joka on helppokäyttöinen, sekä myös valmistajan toimesta erittäin kattavasti dokumentoitu. Tästä johtuen tiedonsiirtoon päädyttiin käyttämään Modbus-protokollaa Ethernet-yhteyden yli.

Jotta Modbus-protokollaa voitaisiin käyttää, oli Wago-logiikkamoduuli tarpeellista liittää verkkoon. Liitettäessä tietokonetta ja Wago-logiikkamoduulia verkkoon ilmeni, että Wago on konfiguroitu väärään IP-osoite alueeseen. Selvitettäessä ratkaisua kävi ilmi, että verkkokonfiguraatioiden tekemiseen on käytettävissä valmistajan oma sovellus Wago Ethernet settings, jolla verkkoasetukset voidaan määrittää usb-yhteyden kautta.

Ensin yritettiin säätää Wago noutamaan automaattisesti IP-osoite dhcp-palvelimelta. Syystä, jota ei pystytty määrittämään, tämä ei kuitenkaan onnistunut, joten Wago-logiikkamoduulille määritettiin manuaalisesti staattinen IP-osoite samasta verkkoalueesta, johon työpiste on kytkettynä.

Wago-logiikkamoduulista tiedot päädyttiin hakemaan hyödyntäen phpmodbus-kirjastoa, ja tallettamaan tiedot MySQL-palvelimelle. Phpmodbus-kirjaston valintaan päädyttiin ensinnäkin siitä syystä, että se ei vaadi muiden kirjastojen ottamista käyttöön, kuten esimerkiksi Python-kielellä toteutettu Modbustk-kirjasto, mikäli halutaan käyttää tiedonsiirtoon Ethernet-yhteyttä. Toisena syynä phpmodbus-kirjaston valintaan oli se että anturitietojen tallettamista varten tarvittiin tietokantaratkaisu ja käytettäessä LAMP-pakettia saadaan sekä PHP-jakelu, jota voidaan käyttää modbus masterin luomiseen, että MySQL-palvelin tietokantaa varten, joten LAMP-ratkaisun myötä PHP:llä tehty Modbus Master oli järkevin ratkaisu yhdessä MySQL-palvelimen kanssa. LAMP-ratkaisun puo-

lesta puhuu myös se, että LAMP-jakeluita on tarjolla sekä Windows- että Linux-käyttöjärjestelmiin.

LAMP-jakeluista valittiin käytettäväksi XAMPP, koska Modbus Masterina toimivalla tietokoneella oli käytössä Windows-käyttöjärjestelmä, ja monien LAMP-pakettien käyttöönoton kanssa on omakohtaisen kokemuksen perusteella usein ongelmia nimenomaisesti Windows-käyttöjärjestelmässä, ja omien käyttökokemusten perusteella XAMPP on kaikkein toimintavarmin ja helppokäyttöisin LAMP jakelu Windows ympäristössä. Lisäksi XAMPP-paketin etuihin kuuluu erityisesti sen graafinen hallintaliittymä, XAMPP Control Panel, jonka kautta on helppo käynnistää ja sammuttaa Apache- ja MySQL-palvelimet, sekä päästä käsiksi tärkeisiin konfiguraatiotiedostoihin.

Tämän jälkeen työpisteellä olleelle laitteelle luotiin MySQL-kanta, joka konfiguroitiin tietoturvan parantamiseksi kuuntelemaan vain loopback-osoitetta, sekä Hadoop-palvelimen IP-osoitetta. Modbus-protokollalla kerätyt tiedot ajettiin modbus-masterina toimineen laitteen MySQL-kantaan PHP-skriptillä, josta ne Hadoop-palvelimelle asennetulla Sqoopilla haetaan ja tallennetaan Hadoop palvelimen Hive-kantaan. Ensimmäisellä kerralla tämä tehtiin käsin, jotta voitiin hyödyntää Sqoopin hive-import toimintoa, joka luo automaattisesti Hiveen vastaavat taulut ja ajaa MySQL-kannan tiedot sisään. Myöhempiä tiedonsiirtoja varten luotiin Pig job, joka hakee automaattisesti määräajoin MySQL kannasta ne tiedot jotka eivät vielä ole Hive-kannassa ja tallettaa ne Hiveen.

6 Pohdinta

Opinnäytetyön tutkivan osuuden lopputulos tarjoaa yleismallisen katsauksen siihen mitä Big Datalla tarkoitetaan ja millaiset teknologiat määritellään Big Data alustoiksi. Lisäksi tutkivassa osuudessa käytiin läpi useita Big Data alustoja ja niiden eri teknisiä ominaisuuksia. Tutkivan osuuden suurimmaksi haasteeksi muodostuivat hyvinkin ristiriitaiset näkemykset siitä mitä Big Datalla tarkoitetaan ja mitkä eri tekniset ratkaisut luokitellaan Big Data alustoiksi. Monissa lähteissä

ei tarjottu näille minkäänlaista määritelmää tai määritelmä oli hyvin ylimalkainen. Toisaalta toisen ääripään esimerkkejäkin löytyi, joissa Big Data ilmausta käytettiin yksiselitteisesti synonyyminä Hadoop nimiselle alustalle.

Tutkittaessa onnistuttiin löytämään yleisesti hyväksytty ja selkeä määritelmä Big datalle, joskin määrittelyn löytäminen sille, mitkä ratkaisut ovat Big Data -alustoja, osoittautui huomattavasti vaikeammaksi ja kilpailevia näkemyksiä löytyi kaksi. Suurimpana erona näiden välillä on se, että Bill Inmonin tiukempi määritelmä sulkee useimmat NoSQL-tekniikat määrittelyn ulkopuolelle, sillä Inmon määrittelee, että Big Data -alustalle tiedot talletetaan strukturoimattomassa muodossa, kun taas useimpiin NoSQL-kantoihin tiedot viedään semi-structured-muodossa. Tutkimusosiossa myös havaittiin, että useiden eri lähteiden mukaan Big Data ei ole korvaamassa tietovarastoja tai muita relationaalisia järjestelmiä kokonaan vaan Big Data on pikemminkin muita järjestelmiä täydentävä ratkaisu ja kasvavana trendinä Big Data -ratkaisuja käytetään rinnakkain relationaalisten järjestelmien, kuten tietovarastojen kanssa.

Vaikka opinnäytetyön toiminnallisessa osuudessa toteutettu ratkaisu oli toimiva, kävi opinnäytetyö prosessin edetessä selväksi, että ratkaisu olisi voitu tehdä paremmin. Esimerkiksi ratkaisussa käytetty MySQL-palvelin oli täysin turha. MySQL-palvelimen ja Sqoopin sijasta olisi voitu hyödyntää Hadoopin data streaming -ominaisuuksia. Tämä olisi tosin myös vaatinut käytettävän modbus master/slave -ratkaisun muuttamista. Lisäksi käytetyn Hadoop ratkaisun sijasta, olisi todennäköisesti Cassandra ollut parempi vaihtoehto, sillä kerätyt sensoritiedot olisivat sopineet erittäin hyvin taulumuotoon. Lisäksi Cassandra on optimoitu erityisesti kirjoitustehtäviä varten, joten Cassandralla toteutettu ratkaisu skaalautuisi erittäin tehokkaasti.

Tämän opinnäytetyön teknisessä osuudessa ei ollut mahdollisuutta toteuttaa kaikkia alkuperäisessä suunnitelmassa määriteltyjä ominaisuuksia. Tähän oli syynä se, että projektia tehtiin oppilaitoksen laboratoriotilassa, ja toteutuksessa käytetty testikytkentä purettiin tuntemattoman henkilön toimesta ilmeisesti siitä syystä, että osia siitä tarvittiin toiseen tarkoitukseen. Lisäksi oppilaitoksessa

tehtävät säännölliset huoltotoimenpiteet pyyhkivät ratkaisussa käytetyt ohjelmat laitteilta joulutauon aikana. Projektille aiheutunutta vahinkoa olisi voitu pienentää varmuuskopioinnista huolehtimalla.

Jatkokehityksen kannalta mielenkiintoisia näkökulmia tarjoaisi Big Datan käytännön hyödyntäminen kuten geospatiaalinen data, clickstream analysointi sekä machine learning. Myös Big Datan kaupalliset mahdollisuudet olisivat hyvä tutkimusaihe. Lisäksi jatkokehitysmahdollisuutena olisi alkuperäiseen suunnitelmaan kuuluneet graafinen käyttöliittymä, sekä ohjausarvojen lähettäminen Wago-logiikkamoduulille.

Lähdeluettelo

Amiri, S. M. (16. 4 2013). *Advantage & Disadvantage of relational database*. Haettu 19. 4 2016 osoitteesta http://smhamiri.blogspot.fi/2013/04/advantage-disadvantage-of-relational.html

Apache Software Foundation. (2015). *Welcome to Apache™ Hadoop®!* Haettu 7. 12 2015 osoitteesta https://hadoop.apache.org/

Asay, M. (19. 11 2014). *MongoDB, Cassandra, and HBase -- the three NoSQL databases to watch*. Haettu 20. 5 2016 osoitteesta http://www.infoworld.com/article/2848722/nosql/mongodb-cassandra-hbase-three-nosql-databases-to-watch.html

Awadallah, A.;& Graham, D. (2014). *Hadoop and the Data Warehouse: When to Use Which.* Haettu 20. 5 2016 osoitteesta http://assets.teradata.com/resourceCenter/downloads/WhitePapers/EB-6448.pdf?processed=1

Bakhshi, R. (21. 1 2014). HOW TO INSTALL HADOOP ON WINDOWS WITH HDP 2.0. Haettu 22. 5 2016 osoitteesta http://hortonworks.com/blog/install-hadoop-windows-hortonworks-data-platform-2-0/

Coe, B. (2013). To MongoDB, or Not to MongoDB. *Code Magazine*. Noudettu osoitteesta http://www.codemag.com/article/1309051

DataStax. (2015). *What is Apache Cassandra*. Haettu 20. 5 2016 osoitteesta http://www.planetcassandra.org/what-is-apache-cassandra/

DataStax. (2015). *What is NoSQL*. Haettu 20. 5 2016 osoitteesta http://www.planetcassandra.org/what-is-nosql/

DB-Engines. (2016). *System Properties Comparison*. Haettu 20. 5 2016 osoitteesta http://db-engines.com/en/system/Cassandra%3BHBase%3BMongoDB

deRoos, D.;Zikopoulos, P. C.;Melnyk, R. B.;Brown, B.;& Coss, R. (2014). *Hadoop for Dummies.* New Jersey: John Wiley & Sons, Inc. Haettu 22. 5 2016

Evans, E. (2011). *CQL:SQL for Cassandra*. Haettu 18. 5 2016 osoitteesta http://www.slideshare.net/jericevans/cql-sql-in-cassandra

Gartner Inc. (27. 6 2011). *Gartner Says Solving Big Data Challenge Involves More Than Just Managing Volumes of Data*. Haettu 18. 4 2016 osoitteesta https://www.gartner.com/newsroom/id/1731916

Grehan, R. (2. 4 2014). *Big data showdown: Cassandra vs. HBase*. Haettu 22. 5 2016 osoitteesta http://www.infoworld.com/article/2610656/database/big-data-showdown--cassandra-vs--hbase.html

Grehan, R. (24. 3 2014). *Review: Cassandra lowers the barriers to big data*. Haettu 20. 5 2016 osoitteesta http://www.infoworld.com/article/2610676/database/review--cassandra-lowers-the-barriers-to-big-data.html

Grehan, R. (31. 3 2014). *Review: HBase is massively scalable -- and hugely complex*. Haettu 22. 5 2016 osoitteesta http://www.infoworld.com/article/2610709/database/review--hbase-is-massively-scalable----and-hugely-complex.html

Hadoop Wiki. (27. 8 2014). Hadoop2OnWindows. Haettu 22. 5 2016 osoitteesta https://wiki.apache.org/hadoop/Hadoop2OnWindows

Haines, S. (27. 10 2014). *Introduction to HBase, the NoSQL Database for Hadoop*. Haettu 22. 5 2016 osoitteesta http://www.informit.com/articles/article.aspx?p=2253412

Hernandez, M. J. (2000). *Tietokannat.* Helsinki: Oy Edita AB.

Hovi, A. (1997). *Data Warehousing.* Espoo: Suomen Atk-kustannus.

HPCC Systems. (2016). *How It Works*. Haettu 19. 4 2016 osoitteesta https://hpccsystems.com/why-hpcc-systems/how-it-works

IBM. (12. 8 2014). *Introduction to YARN*. Haettu 23. 5 2016 osoitteesta https://www.ibm.com/developerworks/library/bd-yarn-intro/

IBM. (2016). *What is the Hadoop Distributed File System (HDFS)?* Haettu 19. 4 2016 osoitteesta https://www-01.ibm.com/software/data/infosphere/hadoop/hdfs/

Inmon, B. (7. 11 2013). *Big Data Implementation vs. Data Warehousing*. (B-eye-Network) Haettu 20. 5 2016 osoitteesta http://www.b-eye-network.com/view/17017

Janalta Interactive Inc. (2015). *What is Hadoop Common?* Haettu 8. 12 2015 osoitteesta https://www.techopedia.com/definition/30427/hadoop-common

Jones, D. (12 2015). *Advantages of a relational database*. (Teach-ICT.com Limited) Haettu 19. 4 2016 osoitteesta http://www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_9/database_design/miniweb/pg8.htm

Karvin, B. (2016). *What are some reasons to use traditional RDBMS over NoSQL?* (Quora.com) Haettu 20. 5 2016 osoitteesta https://www.quora.com/What-are-some-reasons-to-use-traditional-RDBMS-over-NoSQL

Kovacs, K. (2016). *Cassandra vs MongoDB vs CouchDB vs Redis vs Riak vs HBase vs Couchbase vs OrientDB vs Aerospike vs Neo4j vs Hypertable vs ElasticSearch vs Accumulo vs VoltDB vs Scalaris vs RethinkDB comparison*. Haettu 20. 5 2016 osoitteesta http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis

Lachita, R. (26. 3 2014). *Benefits of schema-less database*. Haettu 20. 5 2016 osoitteesta http://fuzzzyblog.blogspot.fi/2014/03/benefits-of-schema-less-database.html

Levitt, N. (2010). Will NoSQL Databases Live Up to Their Promise. *Computer, 43*(2). Haettu 20. 5 2016 osoitteesta http://www.leavcom.com/pdf/NoSQL.pdf

Madden, S. (2012). From Databases to Big Data. *Computer*. Haettu 21. 5 2016 osoitteesta https://www.computer.org/csdl/mags/ic/2012/03/mic2012030004.pdf

McKinsey Global Institute. (2011). *Big data: The next frontier for innovation, competition and productivity.* Haettu 19. 5 2016 osoitteesta http://www.mckinsey.com/~/media/McKinsey/Business%20Functions/Business%20Technology/Our%20Insights/Big%20data%20The%20next%20frontier%20for%20innovation/MGI_big_data_full_report.ashx

Mei, S. (11. 11 2013). *Why You Should Never Use MongoDB*. Haettu 21. 5 2016 osoitteesta http://www.sarahmei.com/blog/2013/11/11/why-you-should-never-use-mongodb/

Middleton, A. M. (2011). Introduction to HPCC. LexisNexis Risk Solutions. Haettu 20. 5 2016 osoitteesta http://cdn.hpccsystems.com/pdf/hpccsystems_technical_brochure.pdf

MongoDB Inc. (2016). *Dynamic Schema Design*. Haettu 22. 5 2016 osoitteesta https://www.mongodb.com/scale/dynamic-schema-design

MongoDB Inc. (2016). *Introduction to MongoDB*. Haettu 22. 5 2016 osoitteesta https://docs.mongodb.com/manual/introduction/

MongoDB Inc. (2016). *What Is Big Data?* (MongoDB Inc) Haettu 20. 5 2016 osoitteesta https://www.mongodb.com/big-data-explained

Sandborg, J. (27. 3 2012). NoSQL Tietokannat. Helsingin Yliopisto. Tietojenkäsittelytieteen laitos. Haettu 22. 5 2016 osoitteesta https://www.cs.helsinki.fi/u/paakki/Semik12-Sandborg.pdf

SAS Institute Inc. (2016). *Big Data What it is and why it matters*. Haettu 19. 4 2016 osoitteesta https://www.sas.com/en_us/insights/big-data/what-is-big-data.html

Techopedia Inc. (2016). *Atomicity Consistency Isolation Durability (ACID)*. Haettu 25. 5 2016 osoitteesta https://www.techopedia.com/definition/23949/atomicity-consistency-isolation-durability-acid

Techopedia Inc. (2016). *Basically Available, Soft State, Eventual Consistency (BASE)*. Haettu 25. 5 2016 osoitteesta https://www.techopedia.com/definition/29164/basically-available-soft-state-eventual-consistency-base

Techopedia Inc. (2016). *Single Point of Failure (SPOF)*. Haettu 21. 5 2016 osoitteesta https://www.techopedia.com/definition/4351/single-point-of-failure-spof

TechTarget. (12 2013). *Apache Hadoop YARN (Yet Another Resource Negotiator)*. Haettu 23. 5 2016 osoitteesta http://searchdatamanagement.techtarget.com/definition/Apache-Hadoop-YARN-Yet-Another-Resource-Negotiator

Tiwari, S. (2011). *Professional NoSQL.* Indianapolis: John Wiley & Sons, Inc.

Villanova University. (2016). *What is Big Data*. Haettu 18. 4 2016 osoitteesta http://www.villanovau.com/resources/bi/what-is-big-data/