Kurtti, Niko

Saatavuuden huomioiminen toteutettaessa web-sovellusta

Diplomityö

Tarkastaja: Prof. Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan
tiedekuntaneuvoston kokouksessa
6. maaliskuuta 2013

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
Kurtti, Niko: Saatavuuden huomioiminen toteutettaessa web-sovellusta
Diplomityö, 50 sivua
Toukokuu 2013
Pääaine: Ohjelmistotuotanto
Tarkastajat: Prof. Tommi Mikkonen
Avainsanat: Web-sovellukset, Korkea saatavuus, Klusterointi, Virtualisointi, Pilvipalvelut

Yhä useammat tietojärjestelmät ovat toiminnaltaan niin kriittisiä, että niiden palveluiden tulee olla käytännössä aina käytettävissä. Tästä huolimatta palveluiden saatavuusvaatimuksia osataan harvoin arvioida riittävän tarkasti sovellusta toteutettaessa, ja vasta ongelmatilanteissa havaitaan sovelluksen saatavuuden kriittisyys ja merkitys liiketoiminnalle. Palvelun saatavuudelle voidaan kuitenkin laskea rahallinen arvo, ja pyrkiä sen avulla arvioimaan kuinka kriittinen sovellus on organisaation toiminnalle ja suhteuttamaan saatu arvo kuluihin, joita korkean saatavuuden infrasktuurin tuottamisesta aiheutuu.

Tässä diplomityössä esitellään korkean saatavuuden peruskäsitteitä ja esitellään miten saatavuuden arvoa voidaan rahallisesti laskea. Tämän lisäksi tutkitaan millaisia uhkia palveluihin liittyy niiden saatavuuden kannalta, ja miten näihin uhkiin voidaan varautua. Työssä esitellään kuhunkin ongelmaan useita erilaisia ja eri tasoisia ratkaisumalleja, sillä ongelmakohtia voi pyrkiä ratkaisemaan hyvin eri tavalla erilaisissa skenaarioissa.

Korkean saatavuuden palveluja web-sovelluksena tuottaessa tulee ottaa huomioon web-sovellusten erityispiirteet. Tällaisia erityispiirteitä ovat muunmuassa tilaton HTTP-protokolla, palvelun kannalta kriittinen verkkoyhteys asiakkaalle sekä se,että tyypillisesti ulkoisia riippuvuuksia on useita. Useimmat web-sovellukset eivät itse säilytä tietoa, vaan se luetaan jostain ulkopuolisesta tietokannasta tai tietokannoista. Näiden tietokantojen saatavuus on palvelun kannalta yhtä tärkeää kuin itse sovelluksen. Näitä ongelmia ja ratkaisumalleja konkretisoidaan esittelemällä Suomen valtion VAHTI-säännöstön vaatimusten mukainen korkean saatavuuden infrakstuuri web-sovellukselle.

Työn tuloksena saadaan selville, että korkean saatavuuden järjestelmä on useimmiten hyvin monimutkainen, ja sisältää huomattavan määrän erilaisia komponentteja joilla on omat vastuualueensa. Työssä havaittiin, että mitä korkeampiin vaatimuksiin järjestelmä pyrkii vastaamaan sitä monimutkaisempi infrakstuurista tulee, ja monimutkaisuus voi jopa aiheuttaa itsessään uhan järjestelmän saatavuudelle. Varsinkin tiedon synkronoinnin haasteet eri solmujen välillä havaittiin toistuvaksi ongelmaksi arkkitehtuurin eri kerroksissa.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
Kurtti, Niko: Consideration of availability when designing web applications
Master of Science Thesis, 50 pages
May 2013
Major: Software Engineering
Examiners: Professor Tommi Mikkonen
Keywords: Web applications, High availability, Clustering, Virtualization, Cloud services

A growing number of the information systems used these days must always be available to the their users. Despite this, the availability requirements of the systems are rarely addressed carefully enough in the implementation phase. Only after facing first troubles are the applications' real availability requirements and affects of downtime really understood. The monetary value of the services availability can be calculated, and the value can be used to assess how critical the application is to organization's activities. This value can then be reflected to the operating expenses that come from developing and maintaining high availabity systems infrastructure.

In this Master's thesis the basic concepts of high availability and its financial value are represented and demonstrated. In addition to this we explore what kind of risks threaten the availability of our information systems and how we protect the services from them. This thesis presents various approaches for each problem scenario, because there usually are multiple ways to guard from the threats.

When implementing high availability services as web applications one should take into account the special characteristics of web applications. These special characteristics are for example the stateless protocol of HTTP, the required network connection to the client, and the fact that there typically is a number of external dependencies. Most of the web applications themselves do not store information and an external database or databases are required. The availability of these databases are as critical as the web applications. These problems and their proposed solutions are presented by designing a example architecture that fulfils the requirements of the Finnish government's VAHTI -guidelines.

The main finding of this thesis is that all high availability systems are complex and include a significant number of different components which have their own responsibilities. It was observed that the level of complexity increases simultaneously with the availability demands and that the complexity itself becomes a threat to availability of the system. In particular, the challenges of data synchronization between the various nodes were found to be a recurring challenge on different levels of the architecture.

ALKUSANAT

Tämä diplomityö on kirjoitettu syksyn 2012 ja kevään 2013 aikana Solita Oy:lle toimiessani heidän palveluksessaan. Ennen kaikkea haluan kiittää työnantajaani paitsi työn tekemisen tukemisesta rahallisesti niin myös kollegoitani siitä, että he mielellään kertoivat omissa projekteissaan kohtaamistaan haasteista sekä ratkaisuista. Nämä epäviralliset haastattelut toimivat hyvänä pohjana työn tekemiselle, ja antoivat suuntaviittoja mihin eri näkökohtiin tulee kiinnittää erityishuomiota.

Erityiskiitokset kuuluvat kuitenkin työn ohjaajalle Niklas Collinille, joka työkiireistään huolimatta ehti vääntämään rautalankaa, sekä Tommi Mikkoselle joka antoi erittäin tarpeellista palautetta aina pilkkuvirheistä itse työn sisältöön.

Tampereella, 14.03.2013

Niko Kurtti

SISÄLLYS

1. Johdanto . . . 1
2. Tietojärjestelmän saatavuus . . . 3
2.1. Saatavuuden perusongelmat . . . 3
2.1.1. Vikasietoisuus . . . 4
2.1.2. Skaalautuvuus . . . 7
2.2. Saatavuustason määritteleminen . . . 9
2.2.1. Todennäköisyys palvelun käytössäololle . . . 9
2.2.2. Palvelutasosopimukset . . . 11
2.2.3. Verifioinnin ongelmat . . . 12
2.3. Saatavuuden arvo . . . 13
3. Fyysisen infrakstuurin merkitys saatavuuteen . . . 17
3.1. Maantieteellinen sijainti . . . 17
3.1.1. Uhkien kartoitus . . . 17
3.1.2. Maantieteellinen hajautus . . . 18
3.2. Oikeanlaisen laitteiston valinta ja merkitys . . . 19
3.2.1. Klusterointi . . . 19
3.2.2. Levyjärjestelmät . . . 20
3.2.3. Verkko . . . 23
3.3. Virtualisointi . . . 24
3.3.1. Järjestelmän virtualisointi . . . 25
3.3.2. Edut ja riskit . . . 27
3.3.3. Pilvipalvelut . . . 28
4. Korkean saatavuuden web-sovelluksen toteuttaminen . . . 30
4.1. Riippuvuuksien saatavuus . . . 30
4.1.1. Tietokanta . . . 30
4.1.2. Muut riippuvuudet . . . 33
4.2. Tilallisuus ja istunnot . . . 34
4.2.1. Istunnot ja tilatieto . . . 34
4.3. Kuormantasaus ja reititys . . . 35

4.3.1. Nimipalvelut 35
4.3.2. Reititys 36
4.4. Suorituskyvyn takaaminen 36
4.4.1. Automaattinen skaalautuminen 37
4.4.2. Ohjelmiston mukautuminen kuormaan 37
5. Esimerkkitoteutus korkean saatavuuden web-järjestelmän infrastruktuurista . . . 38
5.1. Vaatimukset ja konteksti 38
5.1.1. VAHTI-ohjeen saatavuustasot 38
5.1.2. Järjestelmän vaatima taso 39
5.1.3. Tarjottavan palvelun ohjelmistoarkkitehtuuri 40
5.2. Infrastruktuurin suunnittelu 40
5.2.1. Sovelluksen asettamat haasteet infrastruktuurin suunnittelulle 40
5.2.2. Korkean tason järjestelmäarkkitehtuuri 41
5.3. Toteutuksen verifiointi skenaarioiden avulla 44
5.3.1. Aikataulutettujan päivitysten tekeminen eri osiin 44
5.3.2. Yksittäisten laitteiden vikaantuminen 45
5.3.3. Konesalin laajuinen palvelukatkos 46
5.4. Havainnot arkkitehtuurin vahvuuksista ja heikkouksista 47
6. Yhteenveto 49
Lähteet 51

LYHENTEET JA TERMIT

ASM	Automatic Storage Management. Oraclen tietokannan klusteroimiseen käytetty apuohjelmisto joka muodostaa tiedostojärjestelmän loogisista levylaitteista [1].
CTQ	Critical To Quality, asiakkaan toiminnan kannalta kriittinen ominaisuus [2].
HA	High Availability. Korkea saatavuus.
IP	Internet Protocol. Verkkokerroksella toimiva yhteydetön paketteja välittävä protokolla.
Klusteri	Usean palvelimen muodostama kokonaisuus joka esiintyy loogisesti yhtenä.
LDAP	Lightweight Directory Access Protocol. Protokolla hakemistopalveluja varten. Yleisesti käyttö autentikoimiseen ja autorisointiin erilaisissa järjestelmissä [3].
LUN	Logical Unit Number. Levyjärjestelmien yhteydessä käytetty termi osoitettavalla loogiselle ja yksikäsitteiselle levylaitteelle.
MTBF	Mean time between failures. Keskimääräinen vikaantumisväli.
MTTR	Mean time to repair. Keskimääräinen korjausaika.
RAC	Real Application Cluster. Oraclen tarjoama lisäpaketti Oracle Database -tuotteelle joka sisältää klusterointiominaisuudet.
ROI	Return On Investment, mittari saadun hyödyn ja sijoituksen suhteesta.
RPO	Recovery point objective, määrittelee mihin pisteeseen järjestelmä tulee olla mahdollista palauttaa vikatilanteessa.
RTO	Recovery time objective, kuvastaa toipumiskyvyn nopeutta, eli määrittää missä ajassa järjestelmä on mahdollista palauttaa RPO:n määrittelemään pisteeseen.

SPOF	Single point of failure, mikäli järjestelmän toiminta riippuu jostain yksittäisestä resurssista (esimerkiksi yhdestä verkkoyhteydestä) on se resurssi saatavuuden kannalta SPOF.
Sticky sessions	Tekniikka jossa palvelun käyttäjälle määrätään tietty taustapalvelin johon hänen palvelupyyntönsä aina ohjataan.
TCP-protokolla	Transmission Control Protocol. IP-paketteja käyttävä tilallinen verkkoprotokolla, joka kuljettaa luotettavasti tietoa kahden verkkolaitteen välillä. Lähettää kuittauksen paketin saapumisesta [4].
UDP-protokolla	IP-paketteja käyttävä tilaton verkkoprotokolla. Yleisesti käytössä tilanteissa jossa tietoa pitää saada paljon tai nopeasti lähetettyä, mutta jokaisen paketin saapuminen asiakkaalle ei ole kriittistä [4].

1. JOHDANTO

Nyky-yhteiskunta on yhä enemmissä määrin siirtämässä perinteisiä palveluita, kuten liikenteenohjausta ja viranomaisjärjestelmiä, digitaaliseen muotoon. Tietojärjestelmien yleistymisen lisäksi ohjelmistot ovat yhä monimutkaisempia ja riippuvaisempia muista järjestelmistä. Tästä huolimatta kriittisten tietojärjestelmien tulee olla käytettävissä katkoitta, ja pienetkin katkokset voivat aiheuttaa joko merkittävää liiketaloudellista haittaa tai jopa vaaratilanteita ihmisille. Tällaiset järjestelmät ovat korkean saatavuuden järjestelmiä. Kaikki järjestelmät eivät ole näin kriittisiä, mutta esimerkiksi jo puolen työpäivän katkos verkkokaupalle voi aiheuttaa merkittäviä tulonmenetyksiä kauppiaalle sekä vaikuttaa negatiivisesti asiakkaiden kuvaan kyseisestä palvelusta [2].

Saatavuus on ominaisuus joka, kuvastaa miten suuren osan ajasta palvelu on sen käyttäjien käytettävissä. Saatavuutta kuvataan usein prosenttiluvulla, joka on lähellä sataa. Korkean saatavuuden järjestelmän tavoitteena voi olla esimerkiksi käytettävissä olo 99.9999% ajasta. Järjestelmä, jonka saatavuudeksi on luvattu 99.9999%, on todella korkean saatavuuden järjestelmä, sillä tällainen järjestelmä ei voisi olla kuin noin puoli minuuttia vuodessa käyttämättömissä [2][5].

Korkean saatavuuden järjestelmän suunnittelun osa-alueet voidaan karkeasti jakaa kahteen. Perinteisempi näistä on fyysisen infrastruktuurin suunnittelu, eli missä sovellusta tullaan ajamaan, millaisella laitteistolla, miten fyysiset riippuvuudet kuten sähkö ja verkkoyhteys ovat toteutettu ja niin edelleen. Kuitenkin yhtä tärkeä osa-alue on itse sovelluksen ajaminen. Web-sovellusten suorittamiseen liittyy tyypillisesti sovellusta suorittava sovelluspalvelin, tietokantapalvelin sekä mahdollisesti muita riippuvuuksia. Sovelluspalvelimesta voidaan esimerkkinä mainita Apache Tomcat [6] ja tietokannasta Oracle Database [7]. Näistä ohjelmistoista tulee osata valita ja optimoida kuhunkin käyttötapaukseen parhaiten sopiva, mutta korkean saatavuuden kannalta myös se, joka parhaiten tukee korkean saatavuuden arkkitehtuuria ja vaatimuksia.

Tässä työssä kartoitettiin mitkä ovat oleellisemmat uhat verkkopalvelun saatavuudel-

le, sekä millaisilla ratkaisuilla näiden uhkien toteutumisen mahdollisuutta ja vaikutuksia voidaan minimoida. Tämän lisäksi työssä tutkitaan näiden ratkaisuiden vaikutusta järjestelmän monimutkaisuuteen, eli kuinka paljon erilaiset korkean saatavuuden ratkaisut vaativat ylimääräistä työtä tai muuten monimutkaistavat palvelun ylläpitoa. Työssä esitellään myös korkean saatavuuden web-sovelluksen tarjoamiseen liittyviä erityispiirteitä, kuten tyypillisiä riippuvuuksia sekä tilallisuuden vaikutusta infrakstuurin suunnitteluun.

Oleellinen teema saatavuuden parantamisessa on yksittäisten saatavuutta uhkaavien pisteiden (Single Point Of Failure, SPOF) minimointi. Tämä tapahtuu monistamalla kyseistä resurssia niin, että palvelulla on useita identtisiä ja itsenäisiä riippuvuuksia kuten sähköä, palvelimia tai verkkoyhteyksiä saatavilla. Palvelimissa tällaista monistettua loogista kokonaisuutta kutsutaan klusteriksi. Klusterissa yksittäiset palvelimet voivat vikaantua ilman, että palvelun toiminta häiriintyy, eli kukin klusterin solmu pystyy jatkamaan toimintaa normaalisti. Levyjärjestelmissä tällaista järjestelmää taas kutsutaan RAID:ksi (Redundant Array of Independent Disks), jossa kaksi tai useampi fyysinen levy muodostaa yhden loogisen kokonaisuuden, joka kestää yksittäisten levyjen hajoamisen ilman, että levyjärjestelmän toiminta lakkaisi tai muuten häiriöityisi [4].

Työn muodostuu seuraavasta osista. Luvussa 2 tutustutaan tietojärjestelmän saatavuuteen liittyviin peruskäsitteisiin, sekä käydään läpi saatavuuden arvioimiseen käytettäviä metriikoita. Luvussa 3 esitellään infrakstuurin merkitystä saatavuuteen, sekä siihen liittyviä perusongelmia ja ratkaisumalleja. Luvussa 4 kuvataan korkean saatavuuden web-sovelluksen suunnitteluun liittyviä erityispiirteitä ja haasteita. Luvussa 5 toteutetaan korkean saatavuuden arkkitehtuuri esimerkkisovellukselle, jonka vaatimukset tulevat valtion VAHTI-ohjeistuksesta. Luku 6 kokoaa yhteen työn tulokset ja johtopäätökset.

2. TIETOJÄRJESTELMÄN SAATAVUUS

Tietojärjestelmän saatavuus tarkoittaa tämän tarjoaman palvelun käytettävissä oloa. Saatavuus riippuu sekä ohjelmiston, että sitä ajavan laitteiston virheistä. Varsinkin tietoturvan kontekstissa saatavuudesta käytetään joskus termiä käytettävyys, mutta tässä työssä valittiin selkeyden vuoksi termi saatavuus, sillä varsinkin web-ohjelmien kontekstissa termillä käytettävyys on muitakin tarkoituksia.

Tässä luvussa käsitellään saatavuuden uhkia yleisellä tasolla sekä tutkitaan miten saatavuus voidaan kvantifioida esimerkiksi sopimuksissa määriteltäviksi luvuiksi.

2.1. Saatavuuden perusongelmat

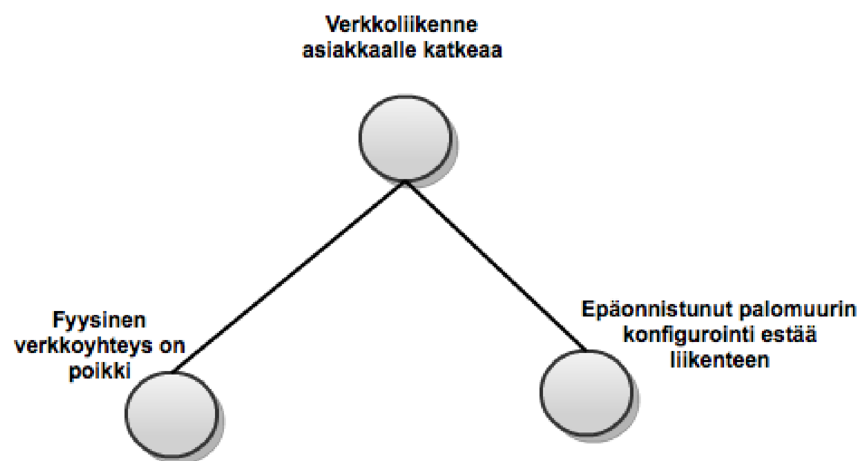
Saatavuuden parantamisessa yleisin lähtökohta on pyrkiä takaamaan korkea vikasietoisuus, eli taatajärjestelmän toiminta virheistä ja ongelmatilanteista huolimatta. Toinen lähtökohta on skaalautuvuus, eli palvelun kyky reagoida kasvaneeseen työkuormaan [8].

Saatavuutta tulisikin suunnitella jo ennen järjestelmän toteuttamista riskianalyysin avulla [2]. Riskianalyysissä kartoitetaan siis:

- palveluun kohdistuvat uhat, esimerkiksi laiterikot
- uhkien todennäköisyydet, eli kuinka todennäköisesti tietty laite hajoaa
- vahingon vaikutus, eli miten esimerkiksi laiterikko vaikuttaa palveluun
- suojautusmenetelmä, eli kuinka estää vahinkojen vaikutus palveluun
- suojautumisen hinta, eli mitä maksaa suojautua palveluun vaikuttavalta vahingolta.

Riskianalyysiä voidaan jatkokäsitellä esimerkiksi uhkapuun muotoon. Uhkapuu on vikaantumispuu-mallista kehitetty uhkien kartoitus menetelmä, jossa pyritään löytämään

uhkia ja pilkkomaan niitä pienempiin osiin mahdollisimman pitkälle [9]. Puun sisäsolmut määrittelevät jonkin osajärjestelmän uhkakokonaisuuden ja toimivat alijuurina, kunnes ollaan päästy pilkkomisessa lehtisolmun tasolle, joka kuvastaa jotakin konkreettista uhkaa. Abstraktiotasolla voidaan siis edetä esimerkiksi verkkosovelluksen saavutettamuudesta ensin verkkoyhteyden katkeamiseen, josta esimerkiksi reititinvikaan, josta vielä esimerkiksi reitittimen laitteistovikaan. Jokaiseen kohtaa sopii monia eri skenaarioita, esimerkiksi reititin saattaa olla toimimaton myös ohjelmistovian vuoksi. Uhkapuu ei kuitenkaan sellaisenaan sovellu kovinkaan tehokkaasti arviointiin, sillä siinä esimerkiksi kustannusten huomioonottaminen on hyvin monimutkaista ja kokonaisen järjestelmän arviointi muodostaisi hyvin valtavia puita [9]. Uhkapuuta voi kuitenkin käyttää soveltaen uhkien kartoittamiseen keskittyen tiettyihin osa-alueisiin, kuten palvelimelle autetikointiin.



Kuva 2.1: Esimerkki yksinkertaisesta uhkapuusta

Kuvassa 2.1 esitetään yksinkertaistettu ja konkreettinen esimerkki uhkapuusta, joka käsittelee tietoliikenneyhteyden katkeamista palveluntuottajan ja sen käyttäjän välillä. Uhkapuun solmuina on eri vaihtoehtoja kyseisen uhan aiheuttamalle tapahtumalle.

2.1.1. Vikasietoisuus

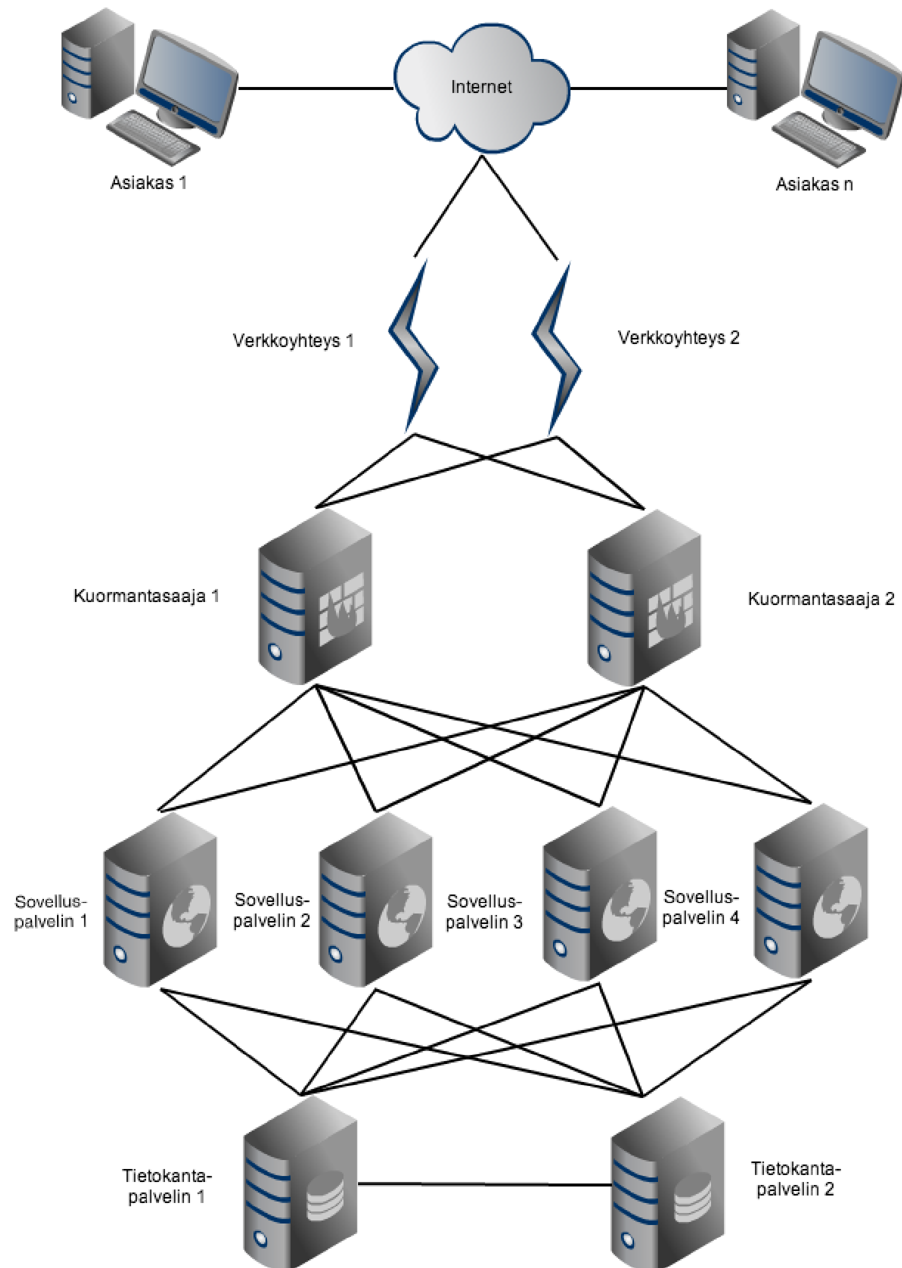
Vikasietoisuus on saatavuuden perinteisin ongelma. Kriittisen tietojärjestelmän ei tulisi koskaan olla riippuvainen mistään yksittäisestä resurssista (Single Point Of Failure, SPOF), eikä sovelluksessa tapahtuvan virhetilanteen tulisi vaikuttaa koko järjestelmään.

Tämän takaaminen on kuitenkin hyvin hankalaa, ja on olemassa osa-alueita missä kahden resurssin yhtäaikainen pettäminen (esimerkiksi levyjärjestelmät) on hyvinkin realistinen uhka.

Vikasietoisuutta voidaan parantaa monistamalla resursseja. Tämä kuitenkin luo monimutkaisuutta järjestelmään, sillä esimerkiksi kaksinkertainen verkkoyhteys vaatii jonkin laitteen valvomaan verkkoyhteyksiä ja ohjaamaan pyynnöt oikeaa reittiä pitkin. Tällaiset konfiguraatiot ovat failover-järjestelmiä. Perusidea on se, että resursseja on monia, mutta vain yksi kerrallaan käytössä. Mikäli yksittäinen resurssi, esimerkiksi verkkoyhteys, katoaa, osaa järjestelmä siirtyä käyttämään toista. Failoverin lisäksi on mahdollista käyttää kuormantasauskonfiguraatiota, jossa on vastaavalla tavalla useita resursseja, mutta kaikkia kuormitetaan jo normaali tilanteessakin. Mikäli jokin resurssi katoaa poistetaan se tasaajasta. Kuormantasauskonfiguraatio takaa sen, että resursseja ei turhaan pidetä käyttämättömänä ja järjestelmä skaalautuu paremmin. Kuormantasaus on kuitenkin huomattavasti monimutkaisempaa. Esimerkiksi mikäli palvelua tarjotaan useammalta palvelimelta ja palvelussa on jonkinlainen tila, kuten web-sovelluksissa usein on, tulee pitää huoli siitä, että tila pysyy samana palvelimien välillä [2].

Kuvassa 2.2 esitetään esimerkkitehtuuri korkean saatavuuden ympäristöstä, joka soveltuisi web-sovelluksen tarjoamiseen. Verkkoyhteydet laitetilaan on kahdennettu, jotta yksittäisen verkkoyhteden katkeaminen huollon tai laiterikon takia ei vaikuta palvelun saatavuuteen. Kun pyyntö on saapunut käsiteltäväksi, ottaa sen vastaan jompikumpi kahdennetuista kuormantasaajista, jotka ohjaavat pyynnön edelleen jollekkin sovelluspalvelimista. Sovelluspalvelimet käyttävät loogista tietokantakokonaisuutta, joka koostuu kahdesta tietokantapalvelimesta. Kaikki arkkitehtuurin palat toimivat itsenäisesti, joten niin kauan kuin kutakin arkkitehtuurin osaa on vähintään yksi toimiva laite jäljellä kykenee palvelua käyttämään.

Esimerkkiarkkitehtuuri tukee myös käytön aikana suoritettavia ohjelmistopäivityksiä, sillä kullekin sovelluspalvelimelle voidaan suorittaa päivityksiä ilman palvelukatkoja, ja koska palvelimia on useita ei suorituskykykään merkittävästi laske. Tämän lisäksi arkkitehtuuri mahdollistaa horisontaalisen skaalautumisen, jonka ansiosta kasvaneeseen kuormaan voitaisiin vastata lisäämällä laitteita rinnalle.



Kuva 2.2: Korkean saatavuuden esimerkkiarkkitehtuuri

Redundanssia, eli vikasietoisuutta, voidaan lisätä ja parantaa monella eri tasolla. Fyysisten laitteiden monistuksen lisäksi voidaan esimerkiksi toiminnot suorittaa moneen kertaan, mikäli ensimmäinen yritys ei mene läpi. Tämä onkin tyypillistä esimerkiksi web-sovelluksen ja tietokannan välisessä kommunikoinnissa, jolloin pyynnöt suoritetaan transaktioissa. Käytetyn verkkoliikenneprotokollan tulisi myös tukea jo alemmalla tasolla pakettien uudelleenlähetystä, eli mikäli operaatioita tai pyyntöjä suoritetaan UDP-protokollaa hyväksikäyttäen tulisi tutkia pystyttäisiinkö sama toteuttamaan myös TCP-

protokollalla. Tietyissä tapauksissa, kuten livevideolähetyksissä, on kuitenkin satunnaiset virheet hyväksyttävämpiä kuin palvelun viivästyminen ja mahdollinen katkeaminen.

Vikasietoisuuden lisääminen lisää monimutkaistaa järjestelmää ja aiheuttaa itsessään uhan vikasietoisuudelle. Kompleksisempi järjestelmä sisältää enemmän komponentteja ja on sitä kautta alttiimpi erilaisille ohjelmointi- ja toimintavirheille. Jokaisen erillisen vikasietoisuutta lisäävän ominaisuuden jälkeen tulisikin kiinnittää erityishuomiota sen testaamiseen ja pyrkiä vikaturvalliseen järjestelmään. Vikaturvallinen järjestelmä tarkoittaa sellaista järjestelmää, joka vikasietoisuutta lisäävässä ominaisuudessa havaitessaan virheen toimii kuin järjestelmässä ei olisi tätä ominaisuutta. Esimerkiksi kuormantasain voisi vikaantuessaan ohjata sisääntulevan liikenteen yhdelle palvelua tarjoavista palvelimista [10].

Web-sovellusten kontekstissa suurin yksittäinen huomioonotettava ongelma on järjestelmien tilallisuus. Mikäli käyttäjien pyyntöihin liittyy istunto on tämä istunto kyettävä joko jakamaan ja ylläpitämään palvelimien välillä. Tämä käytännössä tarkoittaa sitä, että web-palvelimien on kyettävä kommunikoimaan keskenään hyvin nopeasti, ja käyttäjämääristä ja sovelluksesta riippuen myös hyvin intensiivisesti [11]. Usein tätä ongelmaa kierretäänkin sitomalla käyttäjän istunto tiettyyn palvelimeen, jolloin istuntotietoa ei tarvitse jakaa. Tämä kuitenkin pakottaa käyttäjän kirjautumaan uudelleen mikäli kyseinen palvelin vikaantuu. Paras ratkaisu on luonnollisesti pyrkiä tekemään sovelluksesta tilaton niin, että kukin pyyntö on itsenäinen ja sen voi käsitellä mikä tahansa klusterin osista.

Jopa kolmannes suunnittelemattomista katkoista johtuu suoraan inhimillisestä erehdyksestä [4]. Tästä lukemasta on jätetty välilliset ongelmat kuten ohjelmistovirheet pois. Onkin siis kriittistä osata huomioida kompleksisuuden tuomat lisähaitat, sekä pyrkiä automatisoimaan järjestelmän toiminta myös vikatilanteissa. Ihmisen osaamista ei kuitenkaan tule väheksyä, ja siksi onkin oleellista, että kriittisen järjestelmän ylläpidosta on huolehtimassa riittävästi ja riittävän osaavaa henkilökuntaa.

2.1.2. Skaalautuvuus

Saatavuuden kannalta keskeinen haaste heti virhetilanteiden jälkeen on järjestelmän suorituskyky, eli sen kyky vastata saapuneisiin pyyntöihin. Järjestelmä voi olla täysin toimiva, mutta silti olla saavuttamattomissa mikäli siihen kohdistuu suurempaa kuormitusta

kuin mitä se kykenee prosessoimaan. Useimmat ohjelmistot, kuten tietokannata ja web-selaimet, sisältävät itsessään aikamääreitä joiden mukaan ne odottelevat vastausta. Jos vastauksen saamiseen kuluu kauemmin kuin määritelty aika, ei tuloksella tehdä mitään.

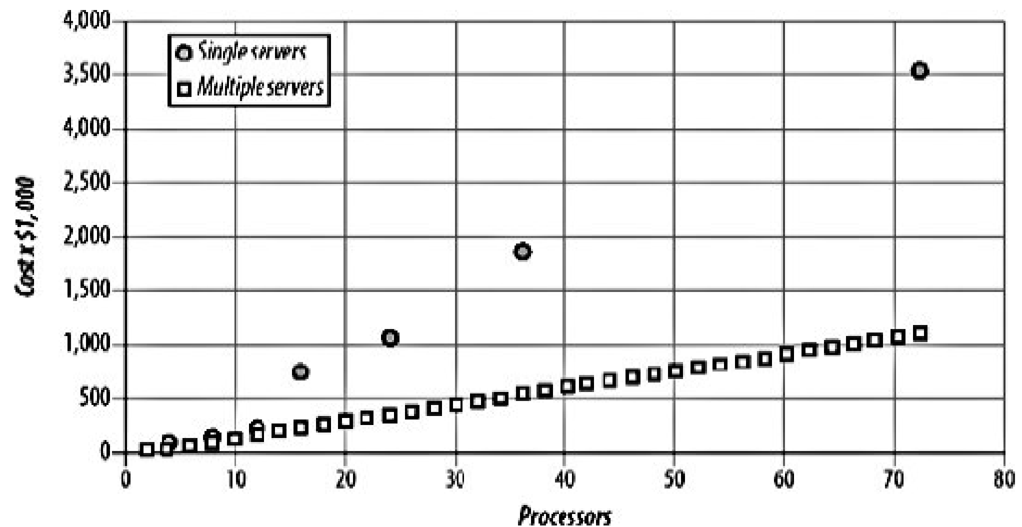
Skaalautuva järjestelmä voidaan määritellä kolmen vaatimuksen avulla [11]. Nämä ovat:

- järjestelmä pystyy mukautuun kasvaneeseen käyttöön
- järjestelmä pystyy mukautuun kasvaneeseen tiedon määrään
- järjestelmä on ylläpidettävä.

Ideaalitilanteessa tietojärjestelmän tuleva kuorma osataan arvioida etukäteen, ja sovelluksen toimintaympäristö sekä optimointi suunnitella sen mukaisesti. Arviointi on kuitenkin hyvin haastavaa, eikä suunnitteluvaiheessa ole helppo luotettavasti arvioida edes kuorman kertaluokkaa oikein.Varsinkin julkisessa verkossa tarjottavien web-sovellusten tulevaa kuormaa on hyvin vaikea etukäteen arvioida, sillä teoreettinen käyttäjien määrä on valtava. Tämän lisäksi haasteena on arvioida käyttäjien toimintatavat, eli kuinka paljon ja milloin he käyttävät palvelussa mahdollisesti olevia raskaampia toimintoja.

Paras tapa parantaa skaalautuvuutta on tehostaa sovelluksen toimintaa. Hajautetuissa ja tilallisissa järjestelmissä tilan monistus itsessään aiheuttaa merkittävää lisäkuormitusta. Mitä vähemmän monistusta tarvitsee suorittaa ja mitä paremmin kuormaa voidaan luonnostaan jakaa järjestelmän eri osille sen tehokkaammin ohjelmisto myös skaalautuu.

Skaalautuvuutta voidaan hallita kahdesta eri suunnasta, horisontaalisesti tai vertikaalisti. Tämä tarkoittaa sitä, että resursseja voidaan lisätä monistamalla nykyisiä (esimerkiksi palvelimia) rinnakkain, tai parantamalla nykyistä resurssia, esimerkiksi päivittämällä nykyisen palvelimen suoritin [12]. Vertikaalinen lähestymistapa on kaikista helpoin, mutta aiheuttaa usein katkoksen palveluun päivityksen ajaksi. Vertikaalinen skaalautuvuus on myös rajallisempaa kuin horisontaalinen, ja sen kustannukset nousevat hyvin nopeasti, kuten kuvassa 2.3 on esitetty [11].



Kuva 2.3: Vertikaalisen ja horisontaalisen skaalaamisen kustannuserot palvelinlaitteistossa suorittimiin suhteuttaen [11]

Horisontaalinen skaalautuvuus tekee järjestelmästä monimutkaisemman, mutta tarjoaa samaan aikaan myös parempaa vikasietoisuutta laitteiston osalta. Horisontaalinen skaalaaminen luo tarvetta myös kasvaneille ylläpito- ja sovelluskehityskuluille. Esimerkiksi kuvan 2.2 mukaisessa ympäristössä yksittäinen sovelluspalvelin yksittäisen kantapalvelimen kanssa ei tarvitsisi kuormantasaajia, eikä esimerkiksi eri palvelimien tietokantojen synkronoinnista tarvitsisi huolehtia.

2.2. Saatavuustason määritteleminen

Saatavuutta voidaan arvioida useista eri näkökulmista ja metriikoilla. Saatavuustasoa määriteltäessä onkin tärkeää osata arvioida kunkin osa-alueen merkitys omalle sovellukselle. Tässä kohdassa käydään läpi tähän liittyvät perusmetriikat ja tutkitaan niiden hyödyllisyyttä ja luotettavuutta.

2.2.1. Todennäköisyys palvelun käytössäololle

Saatavuutta kuvataan prosenttiarvolla, joka koostuu palvelun normaalitilan ajasta (uptime) suhteessa vertailtavaan ajanjaksoon (uptime+downtime). Tätä on kuvattu kaavassa 2.1.

$$saatavuus = \frac{uptime}{uptime + downtime} \tag{2.1}$$

Jos kaavaa käytetään kuvastamaan historiatietoa, on se hyvin tarkka ja kuvastava metriikka. Tämä arvo toimii hyvin myös saatavuustason määrittelemiseen. Esimerkiksi kriittisen järjestelmän vaatimukseksi voitaisiin asettaa, että järjestelmä saa olla käyttämättömissä maksimissaan tunnin kuukaudessa, eli järjestelmän tulisi olla käytettävissä 719 tuntia 720:sta, josta saadaan 99,86%.

Saatavuutta tutkiessa on hyvä lähtökohta selvittää mikä on vikojen esiintymistiheyden keskiarvo (Mean time between failures, MTBF) ja vikojen korjausajan keskiarvo (Mean time to repair, MTTR). Järjestelmän saatavuutta voidaan myös arvioida ennen sen käyttöönottoa MTBF:n ja MTTR:n avulla kaavan 2.2 [13] mukaisesti.

$$saatavuus = \frac{MTBF}{MTBF + MTTR} \tag{2.2}$$

Tämä lähestymistapa on erittäin toimiva, mikäli MTBF ja MTTR voidaan määrittää luotettavasti esimerkiksi identtisen vanhan järjestelmän avulla. On kuitenkin huomioitava, että keskiarvojen käyttämisessä on merkittävät riskit. Esimerkiksi kiintolevyn valmistaja voi ilmoittaa kiintolevyn toimivan keskimäärin 200,000 tuntia, eli 23 vuotta, mutta tämä arvo ei tarkoita sitä, että kiintolevyt toimisivat keskimäärin 23 vuotta ja sen jälkeen hajoaisivat. Keskiarvoon vaikuttavat merkittävästi ääripäät ja esimerkiksi uusien kiintolevyjen tapauksessa tällaiset luvut ovat testaamiseen pohjautuvia arvioita [4]. Käyttäjä ei voikaan tähän valmistajan eksaktiin numeroon kovinkaan paljoa luottavaa, vaan se toimiii lähinnä suuntaa antavana arviona.

Tällä arviointitavalla nähdään myös miten oleellisesti MTTR vaikuttaa järjestelmän saatavuuteen, jota on kuvattu kaavassa 2.3 [2].

$$\frac{10 \times MTBF}{10 \times MTBF + MTTR} = \frac{MTBF}{MTBF + MTTR/10} \tag{2.3}$$

Huomattavaa on,että tässä tapauksessa virheellä tarkoitetaan järjestelmän pysäyttävää virhettä. RAID-varmistetun levyjärjestelmän yksittäisen levyn hajoaminen esimerkiksi ei vaikuta näihin laskuihin, mutta mikäli levyjä olisi vain yksi olisi se pysäyttävä virhe [4].

2.2.2. Palvelutasosopimukset

Palvelusopimus (SLA, Service Level Agreement) asiakkaalle tarjotun palvelun palvelutason määrittelevä sopimus. SLA:n tarkoitus on taata selkeät reunaehdot palvelun tarjoamiselle, jotta asiakas osaa arvioida saamansa palvelun toimivuutta mahdollisimman tarkoin. Sopimuksesta on apua myös palveluntarjoajalle sillä se toimii tietyiltä osin spesifikaationa sille mitä asiakkaalle käytännössä tullaan tarjoamaan [14].

Palveluehtosopimus muodostuu palvelun, prioriteettien,vastuiden ja takuiden määrittelystä. Sopimuksen tulee myös määritellä miten palvelutasoa seurataan ja mitataan, sekä palvelutason alittamisesta seuraavat sanktiot [14].

Taulukko 2.1: Esimerkkejä SLA-sopimuksen tasoista [2]

SLA %	24 x 7		24 x 6		14 x 5	
	per kk	per vuosi	per kk	per vuosi	per kk	per vuosi
99,0%	7,3h	3,7	6,3h	3,1d	3,0h	1,5d
99,5%	3,7h	1,8d	3,1h	1,6d	1,5h	18,3h
99,8%	1,5h	17,5h	1,3h	15,0d	36,6min	7,3h
99,9%	43,8min	8,8h	37,6min	7,5h	18,3min	3,7h
99,99%	4,4min	52,6min	3,8min	45,1min	1,8min	21,9min
99,999%	26,3min	5,3min	22,6s	4,5min	11,0s	2,2min
99,9997%	7,9s	1,6min	6,8s	1,4min	3,3s	39,4s

Saatavuuden kannalta oleellisinta on sen palvelutason tavoitteen (SLO, Service Level Objective) riittävä ja oikeanlainen määrittely. Taulukossa 2.1 on esitetty esimerkkejä yleisimmin käytetyistä prosenttiarvoista ja niiden vaikutuksista reaalisaatavuuteen. Taulukossa kuvataan kunkin prosenttiarvon kohdalla sen sallima palvelun seisonta-aika, eli aika jona palvelu ei ole käytettävissä. Korkean saatavuuden järjestelmiä käsiteltäessä pyritään useimmiten kolmen desimaalin tarkkuuteen (99,999%) tai jopa tarkempaan. Jos järjestelmän on oltava käytettävissä vuorokauden jokaisena hetkenä, tarkoittaa se sitä, että

sallittu seisonta-aika on vain hieman yli viisi minuuttia vuodessa. Tämä on todella lyhyt aika, mikäli virhe vaatii ihmisen toimia, ja siksi tällaiset järjestelmät ovatkin useampikertaisesti monistettu ja automatisoitu. Automatisoidulle valvontajärjestelmällekin tämä voi aiheuttaa vuositasolla ongelmia, sillä pitkällekään automatisoitu valvontajärjestelmä ei huomaa välttämättä vikaa aivan välittömästi. Mikäli vuodessa aiheutuu useita esimerkiksi kymmenen sekunnin katkoksia, kumuloituvat ne helposti useaksi minuutiksi.

Katkojen esiintymisväli onkin katkon keston rinnalla hyvin oleellinen metriikka. Edellämainitun kumuloitumisefektin lisäksi, voi useat lyhyet katkot aiheuttaa enemmän harmia palvelun asiakkaalle kuin esimerkiksi yksittäinen tunnin katko kuussa [2]. Onkin tärkeää, että tämä osataan ottaa huomioon palvelua suunniteltaessa.

Huomioitavaa on myös se, että seisonta-aikoihin pitää sisältyä myös normaalit huoltotoimenpiteet, kuten tietoturvapäivitykset. Käytännössä kaikki tällaisissa järjestelmissä kaikki komponentit onkin monistettu niin, että yksittäinen komponentti voidaan kerrallaan päivittää tuottamatta katkosta palveluun.

Mikäli palvelu koostuu useista eri komponenteista joille on laskettu oma saatavuusarvonsa (kuten kantapalvelimet,sovelluspalvelimet,verkko jne.) on huomioitava, että esimerkiksi seitsemän 99,99% saatavuustason järjestelmän käyttö ei tarkoita, että niiden muodostaman palvelun saatavuus olisi 99,999% vaan $99,999^7\%$ eli 99,993%. Tämä on merkittävästi heikompi taso ja sallii esimerkiksi vuositasolla alle tunnin seisonta-ajan sijaan yli kuuden tunnin seisonta-ajan.

2.2.3. Verifioinnin ongelmat

Edellä mainittujen arvojen laskeminen on hyvin yksinkertaista, mutta oikeiden lukuarvojen määrittäminen voi olla vaikeaa. Varsinkin projektin suunnitteluvaiheessa on uudella alustalla tuotetun uuden ohjelmiston saatavuuden arvioiminen hyvin haastavaa, sillä ei ole välttämättä ollenkaan historiatietoa järjestelmäkokonaisuuden luotettavuudesta.

Järjestelmän käyttöönoton jälkeenkin tulisi mitattavan ajanjakson olla riittävän pitkä, jotta satunnaisuus ei vaikuta oleellisesti laskelmiin. Käytännössä esimerkiksi viikon ajalta otettu mittausdata on hyödytöntä, sillä jo yksittäinen laiterikko vaikuttaa pienellä aikavälillä tutkiessa oleellisesti laskelmaan.

Toisaalta myös mittauskohteen määritteleminen on hankalaa. Asiakkaalle kaikkein tär-

keimpien ominaisuuksien (CTQ, critical-to-quality) kartoittamiseen tulisikin kiinnittää erityistä huomiota, ja tätä listaa päivittää tarpeen tullen. Mikäli tämän listan suunnittelussa onnistutaan, on sen pohjalta hyvä muodostaa määrittelyt oleellisille virheille. Edellisessä luvussa käsitelty virheiden pituuden ja esiintymistiheyden kriittisyys voidaan päätellä CTQ-listan pohjalta [4].

Varsinkin web-palveluiden tapauksessa mittaamiseen tuo lisähaastetta globaali verkko ja monet toimijat asiakkaan ja palveluntarjoajan välillä. Kenellä on vastuu, mikäli suomalainen asiakas ei pääse käyttämään Irlannissa olevaa palvelua, koska reitti Irlannista mantereelle on katki poikki? Tällaisten ongelmien ennakointi on haastavaa tapauksissa, jossa koko reitti asiakkaalta palveluntarjoajalle ei ole omassa hallinnassa, mutta ne tulisi silti kyetä ottamaan huomioon riskisuunnitelmissa.

2.3. Saatavuuden arvo

Korkean saatavuuden järjestelmän tuottaminen ei ole halpaa. Järjestelmien monistaminen, henkilöstöltä vaadittu lisäkoulutus, järjestelmien ylimääräinen testaaminen sekä päivystäminen on usein hyvin kallista. Saatavuustaso onkin usein kompromissi budjetin suhteen. Järjestelmää suunniteltaessa on osattava arvioida millä tavoin suunnittelemattomat katkokset vaikuttavat liiketoimintaan ja mikä on häiriön tuottama rahallinen tappio.

Taulukko 2.2: Esimerkkejä seisonta-ajan kustannuksista [15]

Sektori	Tuotto/Tunti	Tuotto/Miestyötunti
Energia	$2 817 846	$568,20
Tietoliikenne	$2 066 245	$186,98
Valmistus	$1 610 654	$134,24
Rahoitus	$1 495 134	$1079,89
Tietotekniikka	$1 344 461	$184,03
Vakuutus	$1 202 444	$370,92
Kauppa	$996 802	$244,37
Kuljetus	$668 586	$107,78
Sairaanhoito	$636 030	$142,58
Media	$340 432	$119,74

Taulukossa 2.2 on esitetty esimerkkejä eri aloilla saatavasta tuotosta. Tietyissä tapauksissa, kuten finanssialalla, jo muutamien sekunttien katkokset voivat aiheuttaa suoraan merkittäviä tappioita. Toisilla aloilla taas kyse on jopa ihmishengistä, esimerkiksi sairaaloissa tiettyjen tietojärjestelmien toimiminen on kriittistä hoidon kannalta. Epäsuorat seuraamukset kuten tyytymättömät asiakkaat, median huomio tai jopa oikeusseuraamukset tulee myös ottaa huomioon. Mikäli työntekijöiden työ riippuu jostakin tietojärjestelmä voi sen jatkuvat saatavuusongelmat myös vaikuttaa työmoraaliin.

Käytännössä sijoitetun pääoman ja saadun hyödyn laskeminen voi kuitenkin olla erittäin hankalaa. Varsinkin epäsuorien seuraamuksien rahallisen arvon määrittäminen on haastavaa. Sijoitetun pääoman tuottoa (Return On Investment, ROI) voi kuitenkin pyrkiä laskemaan säästöjen kautta, eli

$$S = R_B - R_A$$

jossa S on säästö, R_B riski ennen saatavuustoimia, ja R_A riski toimien jälkeen. Tästä laskemme vielä ROI:N säästöjen ja järjestelmän saatavuuskustannusten (C_M) kautta [4]

$$ROI = \frac{S}{C_M}.$$

Yksittäinen riskielementti (E) voidaan määritellä sen kolmen osa-alueen avulla, eli todennäköisyyden (L), keston(D) ja vaikutuksen(I), josta saadaan

$$E = L \times D \times L.$$

Kokonaisriski ennen saatavuuden huomioimista (R_b) voidaan määritellä näiden summan ja seisonta-ajan kustannuksen (C_D) avulla eli:

$$R_b = C_D \times (E_b 1 + E_b 2 + E_b 3 + ..E_n b).$$

R_a:n, eli kokonaisriski saatavuuden huomioimisen jälkeen, voidaan laskea samalla kaavalla, mutta lisäämällä siihen muutoksien kustannukset C_m :

$$R_a = C_M + (C_D \times (E_a 1 + E_a 2 + E_a 3 + ..E_a n)).$$

Esimerkki. Lasketaan palvelimen tuplauksesta koituva hyöty edellisen luvun arvoilla. Otetaan reunaehdoiksi kolmen vuoden ajanjakso, ja että tässä. mainitut lisäkustannukset tuplauksesta ovat paikkaansapitävät.

Seisonta-ajan kustannukset C_D:

- 75eur/minuutti.

Lisäkustannukset lisätystä saavutettavuudesta C_M:

- Toinen palvelin: 3000eur.
- Ohjelmistot toiseen palvelimeen: 800eur.
- Toisen palvelimen asennus ja klusteroinnin toteutus: 1000eur.
- Ylimääräinen testaus klusteroinnin vuoksi: 1000eur.
- Ylläpitäjän koulutus klusterointituotteen käyttöä varten: 2000eur.

C_M on siis $3000 + 800 + 1000 + 1000 + 2000 = 7800$eur.

Taulukko 2.3: Riskielementit ennen klusterointia [4]

Riski	**Kesto**	**Todennäköisyys**	**Vaikutus**	**E**
Yllättävä reboot	10min	3	100%	E_{b1} = 30
Ajoitettu reboot	10min	10	10%	E_{b2} = 10
Vakava laitteistorikko	60min	1	100%	E_{b3} = 60
Ohjelmistovika	10min	20	100%	E_{b4} = 200
Huoltoikkuna	180 min	3	10%	E_{b5} = 54
Kokonais seisonta-aika				= 354 min (/3vuotta)

Taulukko 2.4: Riskielementit klusteroinnin jälkeen [4]

Riski	**Kesto**	**Todennäköisyys**	**Vaikutus**	**E**
Yllättävä reboot	10min	3	10%	E_{b1} = 3
Ajoitettu reboot	10min	10	0%	E_{b2} = 0
Vakava laitteistorikko	60min	1	40%	E_{b3} = 24
Ohjelmistovika	10min	20	20%	E_{b4} = 40
Huoltoikkuna	240 min	3	10%	E_{b5} = 72
Kokonais seisonta-aika				= 139 min (/3vuotta)

Taulukon 2.3 mukaan R_b on siis

$$75\text{eur} \times 354\text{min} = 26550\text{eur},$$

ja taulukon 2.4 mukaan R_a on

$$7800\text{eur} + 75\text{eur} \times 139\text{min} = 18225\text{eur}.$$

Säästöä (S) kertyi siis

$$S = 26550 - 18225 = 8325\text{eur}$$

ja ROI oli

$$ROI = \frac{S}{C_M}$$

eli

$$ROI = \frac{8325}{7800} = 106,7\%.$$

Vaikka ROI ei tässä laskelmassa olekaan erityisen iso, on muistettava epäsuorat hyödyt. Kyseessä voisi olla esimerkiksi verkkokauppa, jolle parantuneella saatavuudella olisi selkeä merkitys asiakastyytyväisyyteen ja maineeseen.

3. FYYSISEN INFRAKSTUURIN MERKITYS SAATAVUUTEEN

Tässä luvussa käsitellään saatavuuden kannalta perinteisintä näkökulmaa, eli fyysistä infrakstuuria, ja käydään läpi siihen liittyvät peruskomponentit ja -periaatteet. Luvussa myös esitellään kunkin komponentin muodostamat uhat ja mahdollisuudet saavutettavuuden kannalta.

3.1. Maantieteellinen sijainti

Palvelua tarjoavan laitteiston maantieteellisen sijaintiin tulee kiinnittää huomiota kriittisten palveluiden kohdalla. Sijainti voi usein olla palvelun saatavuuden kannalta SPOF (Single Point Of Failure), eli yksittäinen kohta, jonka ongelmien vuoksi palvelua ei pystytä tuottamaan. Esimerkiksi mikäli laitetilan verkko- tai sähköyhteydet katkeavat ei paikallisista klusterointiratkaisuista tai vastaavista ole hyötyä vaan koko palvelu on pois käytöstä.

3.1.1. Uhkien kartoitus

Laitetila on riippuvainen monista ulkopuolisista resursseista. Laitetilan sijainnin suunnittelussa onkin otettava huomioon ennenkaikkea seuraavat riippuvuudet [16]:

- sähkön saatavuus
- jäähdytyksen tarve
- fyysinen turva
- verkkoyhteyksien saatavuus ja luotettavuus.

Vaikka näihin kaikkiin kohtiin voi laitetilan omistaja itsekin vaikuttaa, esimerkiksi hankkimalla omia turvamiehiä ja järeän generaattorin isolla polttoainesäiliöllä, on kustannustehokkainta valita paikka, jossa infrakstuuri on mahdollisimman hyvin toimintaa tukeva ja kustannustehokas. SANS -instituutio suosittelee, että laitetilan paikan valinnassa kannattaa kiinnittää erityistä huomiota viranomaisten läheisyyteen, jotta esimerkiksi tulipalon tai rikoksen sattuessa vasteaika olisi mahdollisimman pieni [16]. He myös suosittelevat ettei laitetilaa kannata perustaa paljon liikennöiden reitin varteen jotta kulkuyhteys olisi mahdollisimman esteetön. Tietyissä tapauksissa voi olla myös tarpeen tarjota henkilökunnalle majoitus- ja ruokatilat kriisitilanteiden varalta, jossa poistuminen tiloista ei ole mahdollista [16]. Tällaisia kriisitilanteita voivat olla esimerkiksi mellakat laitesalin ympärillä tai tulvat.

Edellä mainittujen seikkojen lisäksi tulee nykyaikana kiinnittää erityistä huomiota verkkoyhteyksien toimivuuteen. Laitetila tarjoaa useimmiten palvelujaan tilojen ulkopuolelle, ja siksi on kriittistä, että verkkoyhteydet toimivat myös poikkeustilanteissa. Laitetilaan kannattaakin tuoda useita verkkoyhteyksiä eri suunnista, jotta esimerkiksi tietyllä kadulla tehtävät tietyöt eivät pysäytä palvelujen tarjoamista. Kriittisten palveluiden kohdalla kannattaakin luoda pisteestä pisteeseen yhteyksiä omien tilojen välillä, jotta riippuvaisuus muista toimijoista vähentyy [17].

3.1.2. Maantieteellinen hajautus

Hurrikaani Sandy iski Yhdysvaltojen itärannikolle syksyllä 2012. Myrskyn jälkiseurauksina New Yorkin alueella monet alueet jäivät ilman sähköä päiviksi, pahimmat alueet jopa viikoiksi [18]. Varavirtalaitteistot, kuten akkujärjestelmät tai generaattorit, on tarkoitettu tuottamaan sähköä ongelmatilanteissa, mutta vain harvat ovat varautuneet ajamaan generaattoreita useita päiviä. Palveluntarjoaja Datagram, joka vastasi muunmuassa Gawkerin, Gizmodon ja The Huffington Postin kaltaisista suurista verkkopalveluista, ei ollut varautunut luonnonkatastrofeihin monistamalla palveluita eri lokaatioissa oleviin konesaleihin, ja edellä mainitut palvelut olivatkin myrskyn aikana käyttämättömissä [19]. Muutamaa kuukautta aikaisemmin myrsky Yhdysvaltojen itärannikolla aiheutti sähkökatkoksen muunmuassa Amazonin pilvipalveluita tarjoavassa laitetilassa Virginiassa. Seuraamuksena useissa palveluissa kuten Netflix, Instagram ja Pinterest havaittiin katkoksia. Amazon

kärsi katkoksen seurauksena myös tiedostojärjestelmäongelmista.

Luonnonkatastrofit ja sodat ovat verrattain harvinainen ongelma, mutta edellisessä kohdassa esitellyt laitetilan riippuvuudet voivat osoittautua ongelmallisiksi muutenkin. Sähköverkoissa voi ilmetä suuria ongelmia, sää voi osoittautua poikkeuksellisen lämpimäksi, mellakat voivat estää työntekijöiden pääsyn tilaan tai esimerkiksi rakennustyö katkaista tietoliikenneyhteydet. Nämä ongelmat aiheuttavat useimmiten pitkiä katkoksia joihin korkean saatavuuden SLA-sopmukset eivät anna varaa. Tästä syystä myös laitetilat tulisi usein monistaa, jotta yhden laitetilan ongelmat eivät aiheuta palvelukatkosta [20].

3.2. Oikeanlaisen laitteiston valinta ja merkitys

Tässä kohdassa esitellään infrakstuurin tyypiliisimmät osa-alueet konesalitasolla, eli mistä laitteisto palveluita tarjoa infrasktuuri useimmiten koostuu. Kohdassa käydään läpi kuhunkin osa-alueeseen liittyvät erityishuomiot, ja esitellään joitain esimerkkejä ratkaisuja, joille saatavuutta voidaan parantaa.

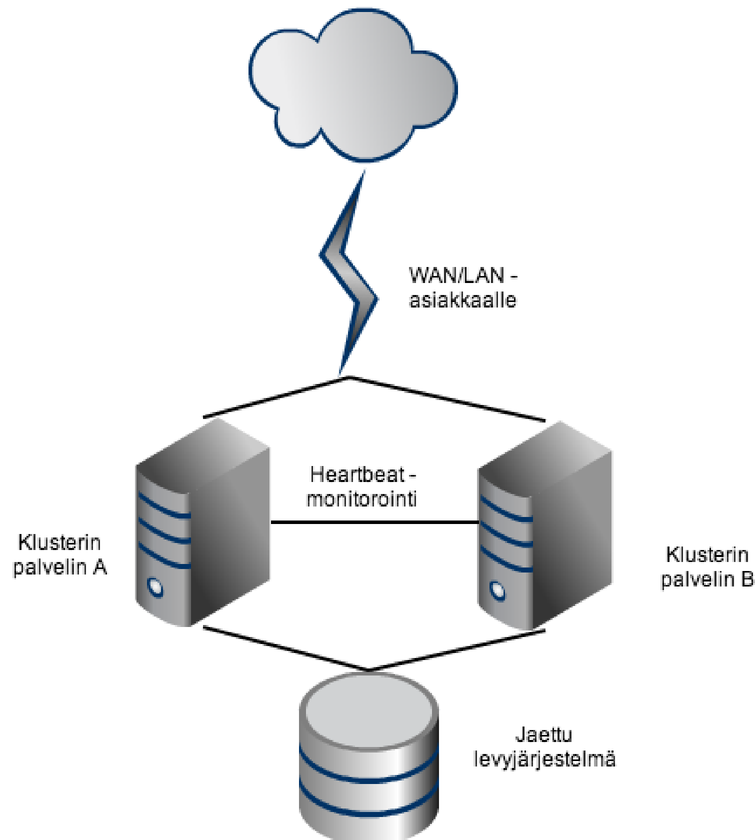
3.2.1. Klusterointi

Klusteroinnilla tarkoitetaan, että pääasialliselle järjestelmälle on olemassa rinnakkaisjärjestelmä, joka automaattisesti ensijaisen järjestelmään vikaantuessa osaa ottaa sen tehtävät hoidettavakseen [4]. Tyypillisesti tässä tilanteessa havaitaan pieni palvelukatkos kun varajärjestelmä havaitsee katkoksen ja siirtyy palvelemaan asiakkaita. Tällaisessa järjestelmässä on siis aktiivisia ja passiivisia osia. Rinnakkaisjärjestelmiä voi olla myös useita ja niitä voidaan käyttää tasaamaan normaalitilanteessa kuormaa. Kuormantasauksen tapauksessa ovat kaikki osat aktiivisia koko ajan.

Klusterin oleellisia ominaisuuksia on se, että sen palat on yhdistetty verkon ylitse ja että se näkyy yhtenä loogisena kokonaisuutena ulkopuolelle [21]. Tyypillinen moderni klusteri rakentuu kahdesta tai useammasta palvelimesta, ja jaetusta levyjärjestelmästä. Alunperin myös käyttömuisti oli jaettu eri palvelimien välillä, mutta tämä on verrattain harvinaista nykyaikana.

Kuvassa 3.1 on havainnollistettu kahden palvelimen klusteria. Palvelimet on yhdistetty verkon yli toisiinsa heartbeat -monitorointia varten. Tämän lisäksi on omat linkit jaettuun

levyjärjestelmään ja palvelurajapintaan. Jokainen verkkoyhteys on hyvä toteuttaa erillisenä fyysisenä yhteytenä, jotta vikatilanteessa osataan reagoida oikein juuri tiettyyn vikaan.



Kuva 3.1: Esimerkkitoteutus kahden palvelimen klusterista

Nykyaikana klustereita käytetään ennen kaikkea korkeaa suorituskykyä vaativissa laskentapalvelimissa. Korkean saatavuuden osalta klusteroinnin tuomaa etua on korvattu virtualisoinnilla ja monitoroinnilla palvelun, ei laitteiston, tasolla.

3.2.2. Levyjärjestelmät

Korkean saatavuuden tietojärjestelmän oleellisin osa on huolehtia tiedon eheydestä ja saatavuudesta. Levyjärjestelmät ovatkin korkean saatavuuden järjestelmien yksi tärkeimmistä osista, sillä ne tulee ottaa huomioon lähes joka skenaariossa, myös sellaisissa järjestelmissä jotka eivät olisi yhteyksissä konesalin ulkopuolelle.

Erilaiset tavat tarjota tallennustilaa palvelimelle voidaan karkeasti jakaan seuraavaan kolmeen tapaan [2]:

- DAS (Direct Attached Storage) tarkoittaa fyysisesti ja suoraan kytkettyä tallennustilaa, kuten palvelimen sisäisiä kiintolevyjä.

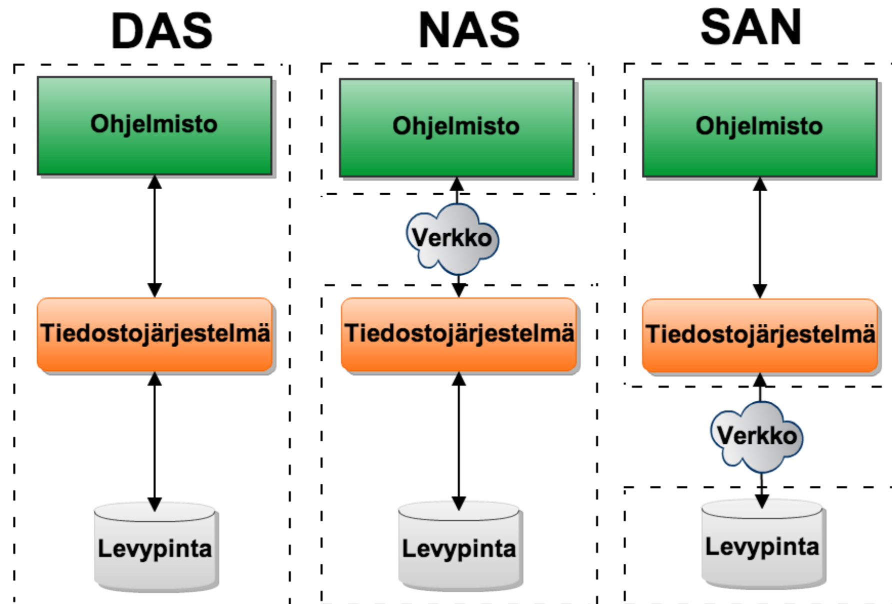
- NAS (Network Attached Storage) käsittää verkon yli jaetut talllennustilat joissa on valmiina jo tiedostojärjestelmä.
- SAN (Storage Area network) tarkoittaa verkon yli käytettävää loogista tallennustilaa, eli käytännössä virtuaalisia levyjä joita ole alustettu.

SAN ja NAS eroavat siten, että NAS:ssa tiedostojärjestelmä sijaitsee etälevyjärjestelmässä. Kohdejärjestelmään tallennustila tuodaan siis jonkin levyjärjestelmän osaa jakavan ohjelmiston avulla. Suosituimpia tähän käyttöön ovat Samba- [22] ja NFS (Network File System)-ohjelmistot [23]. NAS voi käyttää samaa verkkoa muiden palveluiden kanssa, mutta SAN on lähes poikkeuksetta omassa eristetyssä verkossaan aina fyysistä infrakstuuria myöden. Perinteisesti SAN:ia varten olikin omat dedikoidut valokuituyhteydet ja oma protokollansa, mutta nykyään vastaava jako voidaan tehdä TCP/IP:n päällä esimerkiksi iSCSI-protokollalla [2].

Useimmiten palvelimella käytettävät tietojärjestelmät koostuvat yhdistelmästä DAS:ia, NAS:ia ja SAN:ia. Esimerkiksi klusteri voi käynnistyä palvelimeen liitetyiltä levyiltä (DAS), mutta itse käyttöjärjestelmä sijaita esimerkiksi iSCSI:lla tuodulla SAN:lla. Palveluun liittyvä data taas voi sijaita NAS:lla, joka on jaettu esimerkiksi NFS:llä useammalle laittelle.

Näiden kolmen eri tavan erot on havainnellistettu kuvassa 3.2. Oleellisempana erona on tiedostojärjestelmän sijainti. DAS:n tapauksessa kaikki levyjärjestelmään liittyvät osat sijaitsevat paikallisena, NAS:ssa taas tiedostojärjestelmä tuodaan verkon, yli ja SAN:n tapauksessa tiedostojärjestelmä luodaan paikallisesti laitteelle [4].

Keskitetty levyjärjestelmä, jolta tarjotaan kaikille palvelimille tallennustila, on yleisesti käytössä [4]. SAN muodostaa useimmiten kokonaisuuden, jonka takana on moninkertaistettu verkko sekä laitteisto, vaikkei se varsinaiseen määritelmään kuulukaan. Keskittämällä levyjärjestelmä yhteen kokonaisuuteen pystytään sen hallintaa tehostamaan. Palvelinlaitteistot muuttuvat paljon 3-5 vuoden ajanjakson aikana, esimerkiksi levyjen tyyppien osalta, mutta levyjärjestelmän ovat pidempään käytössä ja levyt siksi pidempään samanlaisia. Levyjärjestelmästä voidaan myös allokoida dynaamisesti ja tehokkaasti asiakkaille heidän tarvitsemiaan resurssejaan ilman, että varsinaiseen fyysiseen laitteistoon tarvitsee koskea.



Kuva 3.2: Erot eri levyjärjestelmätyyppien välillä [24]

SAN myös tarjoaa läpinäkyvän tavan tarjota vikasietoista tallennusjärjestelmää asiakkaalle, sillä esimerkiksi levyrikot tai klusteroidun levyjärjestelmän yksittäisen solmun vaihto ei näy kohdekoneelle asti mitenkään. Myöskään varmuuskopiointi ei kuluta kohdejärjestelmän resursseja laisinkaan, sillä se voidaan suorittaa suoraan itse levyjärjestelmää vasten.

DAS:n tapauksessa verkon vikaantuminen ei vaikuta saatavuuteen, mutta toisaalta kunkin koneen oma levyjärjestelmä on oma kokonaisuutensa. Fyysisiin levyrikkoihin voidaan vaikuttaa käyttämällä RAID:ia (redundant array of independent disks). Useimmiten käytetyt RAID-tasot ovat

- RAID0 - Lomitus, 0 levyrikon kesto, n kokonaiskapasiteetti
- RAID1 - Peili, n/2 levyrikon kesto, n/2 kokonaiskapasiteetti
- RAID5 - Pariteettijärjestelmä, yhden levyrikon kesto, n-1 kokonaiskapasiteetti
- RAID6 - Pariteettijärjestelmä, kahden levyrikon kesto, n-2 kokonaiskapasiteetti.

Ylläolevassa listassa on mukana myös RAID0, joka ei tarjoa laisinkaan parannusta saatavuuteen, vaan päinvastoin huonontaa sitä. RAID0 on tekniikka, jolla useita levyjä voidaan lomittaa yhdeksi loogiseksi kokonaisuudeksi. Dataa kirjoitetaan (ja luetaan)

satunnaiselta levyltä ja RAID0-pakka näyttää yhdeltä loogiselta laitteelta. RAID0:n teoreettinen nopeus on levyjen yhteenlaskettu nopeus, mutta tähän käytännössä harvemmin päästään, sillä ei ole takeita että haettua dataa voitaisiin lukea kaikilta levyiltä rinnakkain.

Muut RAID-tekniikat tarjoavat lisävarmuutta, sillä ne mahdollistat levyjärjestelmän katkottoman käytön vaikka yksittäinen levy pettäisikin. RAID1 perustuu puhtaasti siihen, että datasta on aina täysi kopio toisella levyllä, mutta RAID5/6 ovat pariteettien laskemiseen perustuvia tekniikoita. Levyrikot ovat kuitenkin verrattain harvinaisia ja siksi useamman kuin kahden levyn RAID-järjestelmissä tyypillisesti käytetään joko sisäkkäistä RAID0- ja RAID1-toteutusta (ns. RAID10), tai sitten pariteetteihin perustuvia tekniikoita (RAID5/6). Näin saadaan käyttövarmuutta, mutta ei menetetä yhtä paljon kokonaiskapasiteettia kuin peilausta käytettäessä [2].

Korkean saatavuuden palveluissa usein myös levyjärjestelmän tulee olla klusteroitu. Oracle on kehittänyt OCFS2-tiedostojärjestelmän (Oracle Cluster File System Version 2) tähän tarkoitukseen, joka on nykyisin integroitu myös Linux-kerneliin [25]. Vaihtoehtoinen ratkaisu olisi esimerkiksi klusteroitu NFS-jako, jossa replikoinnista huolehtii Linuxin kernelistä löytyvä Distributed Replicated Block Device -moduuli. Perusperiaate klusteroiduissa levyjärjestelmissä on se, että on yksittäinen isäntä-palvelin, joka huolehtii kirjoittamista ja replikoinnista muille klusterissa oleville laitteille. Kirjoitukset siis tehdään verkon yli päätoimisen järjestelmän kautta, mutta lukuoperaatiot voidaan suorittaa paikallisina [26].

3.2.3. Verkko

Nykyaikaisissa sovelluksissa verkko on yhä oleellisemmassa osassa. Palvelut siirtyvät yhä enemmissä määrin verkkoon, ja perinteisetkin sovellukset kuten toimistotyökalut hyödyntävät yhä enemmissä määrin verkkoa. Web-sovellusten tapauksessa verkon toiminta on kriittistä. Korkean saatavuuden verkkosovellusta suunniteltaessa onkin osattava ottaa huomioon oman sovelluksen ja sen ajoympäristön lisäksi myös sen saatavuus asiakkaalle asti [4].

Suunniteltaessa korkean saatavuuden verkkoratkaisuita on ongelmakenttä esimerkiksi levyjärjestelmiin verrattuna hieman erilainen. Laiterikkojen ja ohjelmistongelmien on huomioitava, että koko toimitusputki asiakkaalta palveluun harvemmin on omassa hal-

linnassa. Tämä käytännössä näkyy siinä, että on osattava ennakoida esimerkiksi käyttäjien operaattoreiden toimia ja heikkouksia [5]. Tämän lisäksi on huomioitava, että verkkoa vastaan voidaan vihamielisesti hyökätä ulkoapäin, ja että verkko onkin ensimmäinen kriittinen polku usein sovelluksen käyttämisen kannalta. Mikäli hyökkääjä voi tukkia verkkoyhteyden esimerkiksi palvelunestohyökkäyksellä on palvelun tarjoaminen mahdotonta vaikka itse ympäristö olisikin täysin käyttökelpoinen palomuurien takana.

Täysin robustin verkon suunnittelu ei ole kuitenkaan mahdotonta. Oleellista on tunnistaa palveluun kohdistuvat uhat, ja pyrkiä minimoimaan niiden vaikutuksia. Peruslähtökohta on sama kuin palvelimissa, pyrkiä poistamaan yksittäiset laitteet joiden vikaantuminen voi aiheuttaa katkoksen palveluun. Tällaisia laitteita voivat olla esimerkiksi palvelun edessä olevat palomuurit,reitittimet tai kytkimet. Jopa laitesalin käyttämä yhteys voidaan laskea tällaiseksi, ja todella kriittiissä palveluissa laitesaliin tulisikin tuoda kaksi fyysisesti täysin erillistaä yhteyttä. Korkean saatavuuden ympäristöissä onkin tyypillistä, että jokainen verkkolaite on vähintään tuplattu, ja jokaisesta verkkolaitteesta on redundantit verkkoyhteydet siihen liittyviin laitteisiin. Käytännössä tämä siis tarkoittaa, että mikäli HA-ympäristössä on kaksi redundanttia reititintä jotka ovat yhteydessä kahteen redundanttiin kytkimeen on näiden välillä neljä verkkopolkua, kaksi kustakin reitittimestä kumpaankin kytkimeen[2].

Verkkohyökkäyksiltä puolustautuminen on haastavaa, sillä ne saattavat usein muistuttaa laillista liikennettä. Esimerkiksi palvelunestohyökkäyksiä on hyvin hankala kyetä kokonaan torjumaan, mutta niiden sietäminen verkkoyhteystasolla ja suodattaminen palvelimilta on mahdollista. Reitityspolkuja muuttamalla voidaan liikennettä myös ohjata kulkemaan eri reittiä pitkin kuin mitä hyökkäys käyttää [2].

3.3. Virtualisointi

Virtualisointi on terminä on terminä peräisin 1960-luvulta, ja sitä käytetään kuvastamaan erilaisten virtuaalikoneiden ajamista fyysisen laitteiston päällä ohjelmistolla emuloituna. Tämä virtuaalikone näkyy käyttäjälle useimmiten itsenäisenä kokonaisuutena jolla ei ole tietoa sitä ajavan virtuaali-isännän tilasta. Tässä diplomityössä käsitellään vain myös fyysisellä laitteistolla ajettavien käyttöjärjestelmien virtualisointia [27].

Tässä kohdassa käydään käyttöjärjestelmävirtualisoinnin perusperiaatteet läpi, sekä

esitellään niiden hyödyt korkean saatavuuden palveluiden tarjoamisen kontekstissa. Kohdassa esitellään myös pilvipalvelujen perusperiaatteet, sillä ne hyödyntävät erityisesti virtualisoinnin tarjoamia mahdollisuuksia.

3.3.1. Järjestelmän virtualisointi

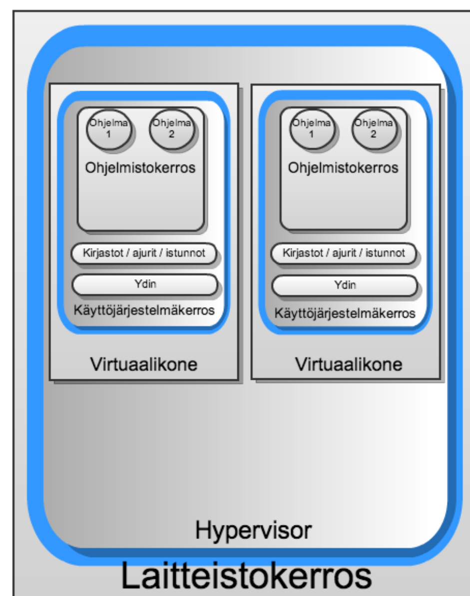
Virtualisointitekniikoita on kehitetty erilaisia kymmenien vuosien aikana. Tällä hetkellä yleisin tekniikka on niin sanottu täysi virtualisointi, ja siinä käyttäjälle esitetään isänkoneesta riippumaton täysi järjestelmä. Tämä virtuaalikone kykenee ajamaan muokkaamattonta käyttöjärjestelmää, kuten Microsoft Windowsia. Täyttä virtualisointia tukevia alustatyyppejä on kahdenlaisia: hypervisor ja isännöity. Täysi virtualisointi prosessorilta oman laajennoksensa (Intelillä VT-x ja AMD:llä AMD-V). Nämä tulivat käyttöön vuonna 2005, ja ne ovat nykyisin mukana kaikissa näiden valmistajien x86 -prosessoreissa [28].

Hypervisor -tyyppinen virtualisointi tarkoittaa sitä, että isäntälaitteistoon (eli fyysiseen laitteistoon) asennetaan minimaalinen käyttöjärjestelmä, joka tarjoaa pelkästään virtualisointia varten tarvittavan alustan. Tällaisia käyttöjärjestelmiä tarjoavat muunmuassa VMware [29] ja Citrix [30]. Hypervisor -pohjaista virtualisointia käytetään tyypillisesti palvelinympäristöissä, joissa kaikki palvelut sijaitsevat virtuaalikoneilla ja suoraan fyysisen laitteiston päällä toimivan käyttöjärjestelmän ainoa funktio on toimia virtuaalikoneiden pohjana.

Isännöity virtualisointi taas kuvaa järjestelmää, jossa on jokin olemassaoleva normaali käyttöjärjestelmä, jonka päälle asennetaan virtualisointia suorittava ohjelma. Tällainen sovellus on esimerkiksi Oraclen Virtualbox, joka toimii Windows-, Linux- ja OSX-käyttöjärjestelmissä [27]. Tyypillisesti isännöityä virtualisointia käytetään työasemissa, jolloin isäntäkäyttöjärjestelmä suorittaa muitakin ohjelmia kuin virtualisointisovellusta.

Kuva 3.3: Fyysinen laitteisto

Kuva 3.4: Isännöity virtualisointi



Kuva 3.5: Hypervisor -virtualisointi

Kuvissa 3.3, 3.4 ja 3.5 on esitelty isännöidyn ja hypervisor -pohjaisen virtualisoinnin erot. Oleellisimpana erona siis on se, että isännöidyssä virtualisoinnissa virtuaalikoneita isännöivä käyttöjärjestelmä on normaali käyttöjärjestelmä, joka suorittaa muitakin ohjelmia virtualisointialustan ollessa vain yksi näistä. Hypervisor-pohjaisessa taas ei ole mitään muuta kuin virtuaalikoneita suorituksessa.

3.3.2. Edut ja riskit

Suurimmat virtualisoinnin hyödyt liittyvät ketterämpään ja tehokkaampaan fyysisten resurssien käyttöön. Laitteistoja voidaan ylivarata olettamalla, että virtuaalikoneet eivät useimmiten käytä esimerkiksi CPU:ta täydellä teholla yhtä aikaa, ja toisaalta uusia virtuaalikoneita voidaan luoda lisää niin kauan kuin olemassaolevalla laitteistolla on joutukäyntiä. Tämä mahdollistaa nopeamman palveluiden tarjoamisen, mutta toisaalta myös laskee kustannuksia tehostamalla laitteistojen hyväksikäyttöä. Mitä paremmin fyysistä laitteistoa voidaan hyväksikäyttää, sen vähemmän tarvitaan tarvitaan myös paitsi laitteistoja niin myös tilaa,sähköä ja laitteistoja niiden ajamiseen [27].

Perinteisen korkean saatavuuden ratkaisu palvelinlaitteistojen saatavuuden parantamiseen on niiden monistaminen. Useimmiten kuitenkaan kuormanjakoratkaisut eivät ole tarpeellisia, joten käytännössä varalaitteistot vain odottavat vikatilanteiden tapahtumista käyttämättömänä. Tämä on huomattava kustannus suuremmissa järjestelmissä, ja yksi virtualisoinnin tuomista eduista onkin se, että voidaan perinteisen fyysisten laitteistojen sijaan monistaa virtuaalikoneita joita ajetaan muutamalla fyysisellä laitteistolla [28].

Virtuaalikoneet on lähtökohtaisesti suunniteltu niin, että ne samassa fyysisessä laitteistossa suoritettavat virtuaali-instanssit eivät näe toisiaan eivätkä isäntälaitteistoa suoraan. Joitakin poikkeuksia, on kuten tiettyjen resurssien jakaminen isäntälaitteistolla, mutta näitä ratkaisuja käytetään yleensä lähinnä työasemakäytössä. On kuitenkin huomattava, että isäntälaitteisto kykenee lukemaan virtuaalikoneiden resursseja kuten muistia, levyä tai verkkoa. Mikäli isäntäkoneeseen päästään murtautumaan on mahdollista murtautua myös virtuaalikoneisiin ja päästä käsiksi niiden käsittelemään dataan. Isäntäkoneen suojaaminen on siis pidettävä vähintään yhtä tärkeänä asiana kuin tärkeimmmän virtuaalikoneen suojaaminen [31].

Yksi virtualisoinnin suurimpia riskejä ovatkin virtualisointikerroksen tietoturvaongelmat. Toinen merkittävä tekijä on virtualisointikerroksen toteutuksen taso, eli varmistus siitä, että esimerkiksi yksittäiseen virtualisointikoneeseen kohdistuva palvelunestohyökkäys ei estä muita virtualisoitua palveluita toimimasta normaalisti [26].

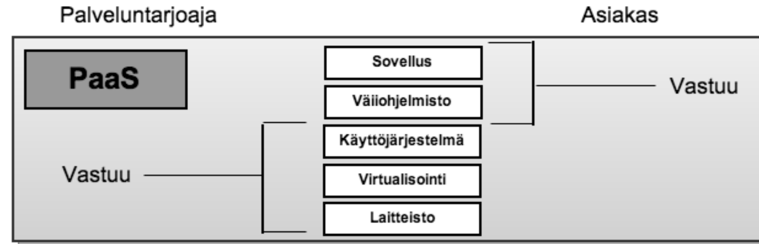
3.3.3. Pilvipalvelut

Pilvi ja pilvipalveluiden määritelmä vaihtelee huomattavasti eri lähteissä. Eksaktin määritelmän puute onkin luonut hyvin erilaisia, niin uusia kuin vanhoja, malleja tarjota palveluita ja resursseita. Oleellisin tekijä on kuitenkin pilvipalvelujen luonteen dynaamisuus, sekä infrakstuurissa hyödynnyttävä virtualisointi, joka osaltaan sen mahdollistaa. Pilvelle ominaista on sen ketterä skaalautuvuus sekä käyttöpohjainen laskutus [32]. Ketterä skaalautuvuus tarkoittaa siis sitä, että laskentatehoa ja muita pilvipalveluita pystytään automaattisesti luomaan lisää (tai poistamaan olemassa olevia), eikä perinteisellä tavalla manuaalisesti [33].

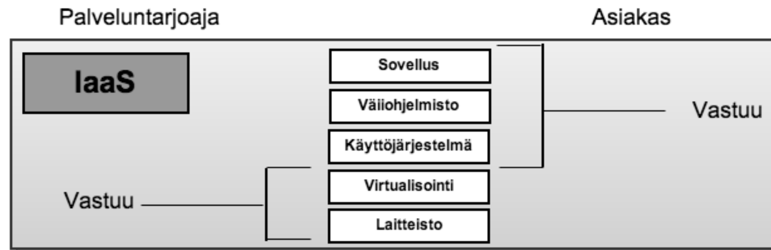
Pilvipalvelut voidaan jakaa kolmeen eri kategoriaan niiden tarjoaman palvelun perusteella. Karkeasti jakaen nämä tarjoamat ovat valmiit palvelut (Software as a Service, SaaS), sovellusalusta (Platform as a Service, PaaS) ja infrasktuuri (Infrastructure as a Service, IaaS) [28].

Vastuujaot kussakin kategoriassa on esitelty kuvissa 3.6, 3.7 ja 3.8. SaaS -mallissa palveluntarjoaja huolehtii koko sovelluksen toiminnasta, joka tasolla, aina fyysisestä laitteisto itse ohjelmistoon. PaaS:ssa palveluntarjoaja vastaa sovelluksen alustan toiminnasta, yleisimmin fyysisen laitteiston toiminnasta sovellusta suorittavan virtuaalikoneen käyttöjärjestelmään asti. IaaS:ssa asiakkaalle tarjotaan virtualisoitu ympäristö, joka emuloi fyysisten laitteiden hankkimista. IaaS:n tapauksessa asiakkaan ei tarvitse huolehtia fyysistä laitteistoista, mutta käyttöjärjestelmän asennus ja ylläpito on hänen vastuullaan [28].

Kuva 3.6: Sofware-as-Service (SaaS). Valmis ohjelmisto



Kuva 3.7: Platform-as-a-Service (PaaS), Alusta ohjelmistolle



Kuva 3.8: Infrastructure-as-a-Service (IaaS), Virtuaalikone/koneita

Pilvipalveluiden oleellisin hyöty niiden käyttäjälle on käyttöön perustuva laskutus ja nopea skaalautuvuus. Resursseja voidaan ostaa vasta kun niitä tarvitaan, eikä palvelua käyttöönottaessa tarvitse varata kuin minimiresurssit. Tällainen skaalautuvuus mahdollistaa merkittäviä säästöjä, ja toisaalta parantaa palvelun loppukäyttäjienkin saamaa kokemusta sillä automaattisella skaalautuvuudella pystytään tehokkaasti vastaamaan piikkimäisiin kuormiin [33].

4. KORKEAN SAATAVUUDEN WEB-SOVELLUKSEN TOTEUTTAMINEN

Tässä luvussa käydään kuvataan korkean saatavuuden web-sovelluksen tuottamiseen liittyvät korkean saatavuden haasteet, sekä käydään läpi näiden haasteiden ratkaisumalleja.

4.1. Riippuvuuksien saatavuus

Web-sovelluksille on ominaista riippuvuudet muista järjestelmistä. Tyypillisellä web-sovelluksella on taustallaan tietokanta, joka on useimmiten SQL-pohjainen relaatiotietokanta. Tämän lisäksi riippuvuuksia on usein erillisiin integraatiorajapintoihin, tai esimerkiksi käyttäjätietokantaan, kuten LDAP-hakemistoon (Lightweight Directory Access Protocol) [3]. Nämä erilaiset riippuvuudet heikentävät web-sovelluksen saatavuutta, sillä esimerkiksi käyttäjähakemisto ja relaatiotietokanta muodostavat usein sellaisia riippuvuuksia jotka estävät itse web-sovelluksen käytön.

Korkean saatavuuden kannalta on siis oleellista huomioida myös niiden palveluiden monistaminen joista oma palvelu on riippuvainen. Tiedon monistamisen perinteinen ongelma, eli eheys solmujen välillä, on kuitenkin tässä tapauksessa otettava huomioon jokaisen riippuvuuden kohdalla erikseen.

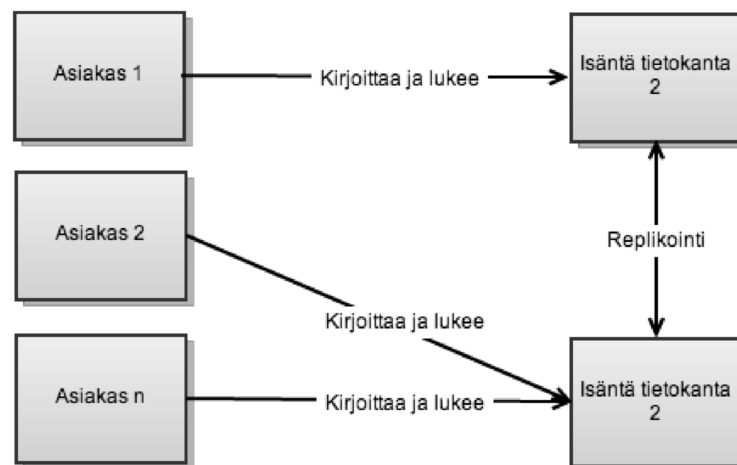
4.1.1. Tietokanta

Web-palveluiden käyttämä tieto on tyypillisesti tallennettu tietokantaan. Tällaisten sovellusten kannalta tietokanta on kriittinen sillä niiden toiminta perustuu juuri tämän tiedon käsittelyyn, eivätkä ne kykene toimimaan ilman sitä. Tietokannan saatavuus ja luotettavuus ovatkin siksi kriittisessä osassa suunniteltaessa korkean saatavuuden sovellusta.

Ideaalitilanteessa tieto sijaitsee useassa paikassa samanaikaisesti. Tämä asettaa kuitenkin haasteita tiedon eheydelle, sillä tiedon tulee olla identtistä jokaisena ajanhetkenä

jokaisessa sijainnissaan. Tietokantojen klusterointiominaisuudet pyrkivät vastaamaan tähän tarpeeseen tarjoamalla sisäänrakennetun tuen tiedon monistamiselle kullekkin klusterin osalle, sekä tiedon eheydestä huolehtimisen [34]. Oraclella on tähän tarpeeseen tuote nimeltä Oracle RAC (Real Application Cluster), joka on Oraclen Database -tuotteen klusterointiominaisuuspaketti [35]. Se tarjoaa klusterointiin tarvittavia apuohjelmistoja, kuten levyjärjestelmän hallintaan ja klusterin monitorointiin liittyviä osia.

Nykyaikaiset tietokantaklusterit lähestyvät tiedon jakamista kahdesta eri arkkitehtuurillisesta mallista. Oracle RAC käyttää tiedon jakamiseen jaettuun levyjärjestelmään, eli tietokanta sijaitsee loogisesti yhdessä yhteisessä paikassa jota kaikki klusterin osat käyttävät. Useista kilpailevista tietokantaratkaisuista, kuten PostgreSQL:stä [36] ja MySQL:stä [37], löytyy kuitenkin tuki myös itsenäisille klusterin solmuille, jotka eivät jaa mitään resurssia.

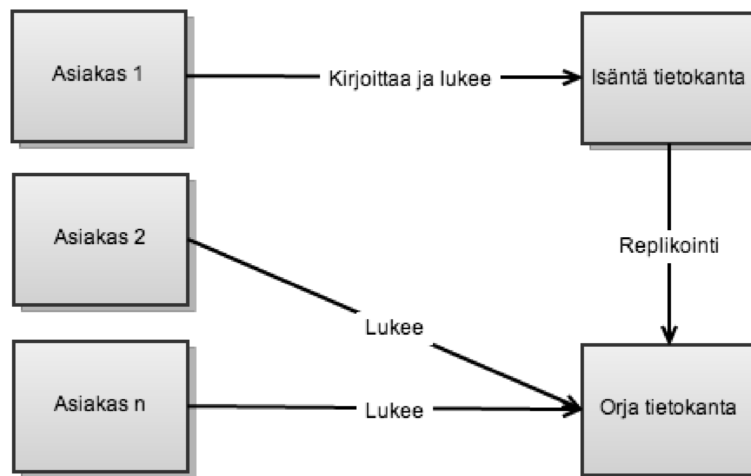


Kuva 4.1: Kaikki tietokannat isäntätilassa -arkkitehtuuri

Tietokantasovellustason tiedon replikaatio tarjoaa täysin itsenäiset tietokantasolmut, jotka eivät ole riippuvaisia levyjärjestelmätason toteutuksesta. Tämän arkkitehtuurin suurin ongelma on kuitenkin mahdollisuus tilanteisiin jossa solmujen tietokannat eivät ole enää keskenään eheitä, eli tietoa on muokattu kahdessa paikassa yhtäaikaa ja sovellus on epäonnistunut sen synkronoimisessa. Myös suorituskyvyssä on eroa, sillä jaetun levyjärjestelmän tapauksessa tietokantamuutos on tehtävä vain kerran levyjärjestelmään, mutta tietokantatason replikoinnissa se on sen jälkeen vielä viestittävä muille solmuille ja niiltä saatava myös kuittaus päivityksen onnistumiselle. Tämä hidastaa merkittävästi tietokannan kirjoitusnopeutta [36]. Kuvassa 4.1 on esitetty esimerkki tästä arkkitehtuurista, jossa

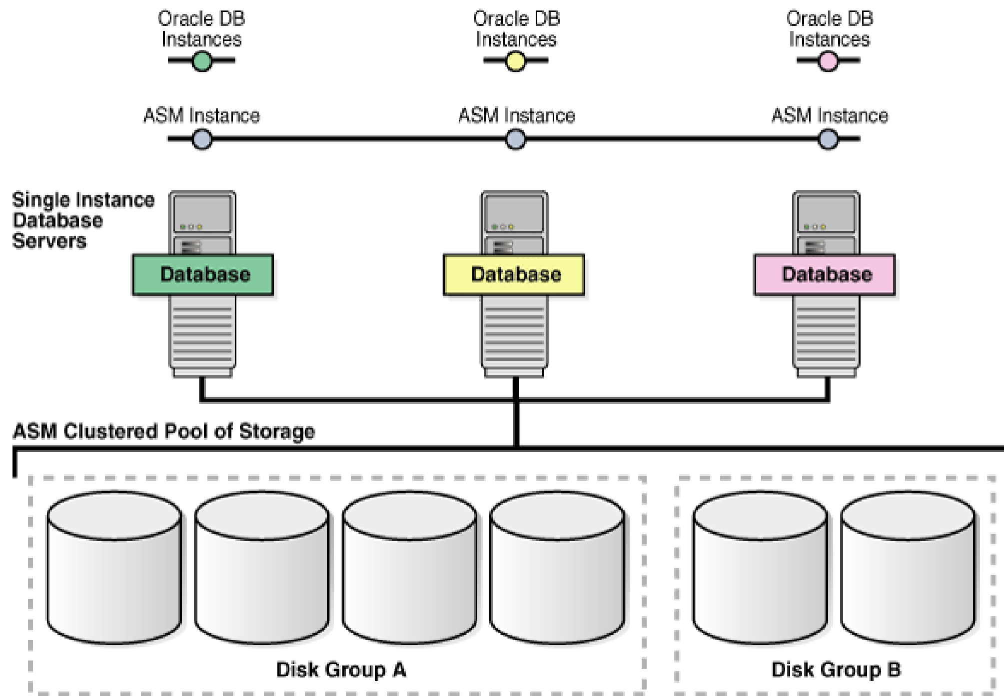
jokainen tietokantasolmu tarjoaa sekä kirjoitus- että lukuoperaatioita asiakkaille.

Tietokantatason replikointia voidaan tehdä myös niin, että yksittäinen tietokantasolmu toimii isäntätilassa ja ainoastaan tähän solmuun kohdistuu kirjoitusoperaatiot. Rinnalla voi olla useita orjatilassa toimivia solmuja, joista käyttäjä voi lukea tietoa mutta joihin ei kohdistu kirjoitusoperaatioita. Mikäli käytössä ollut isäntäsolmu vikaantuu, ottaa jokin orjasolmuista isännän roolin [11]. Tämä operaatio on esitetty kuvassa 4.2, jossa asiakas 1 tekee muutoksia tietokantaan, sekä lukee tietoa sieltä, mutta muut asiakkaat vain lukevat. Asiakas 1 on ohjattu kirjoituksen vuoksi isäntätietokantaan, ja muut asiakkaat voivat käyttää orjatietokantaa. Tämä arkkitehtuuri parantaa lukunopeutta ja saatavuutta, mutta kirjoitusnopeus jää maksimissaan yhden tietokannan tasolle.



Kuva 4.2: Isäntä ja orja -tietokanta-arkkitehtuuri

Jaettua levyjärjestelmää käyttävä tietokantaklusteri on siis lähtökohtaisesti nopeampi ja yksinkertaisempi kuin tietokantatasolla replikaatiota tekevä järjestelmä. Levyjärjestelmä kuitenkin voi aiheuttaa itsessään uhan saatavuudelle mikäli sen monistamisesta ja luotettavuudesta ei ole huolehdittu. Levyjärjestelmän ei siis tule olla yksi yksittäinen sijainti tai esimerkiksi fyysinen levy, vaan klusteroitu useampaan paikkaan. Tätä varten tarvitaan klusteroitu tiedostojärjestelmä. Oracle RAC tarjoaa ASM-ohjelmiston (Automatic Storage Management), joka abstrahoi tämän tarpeen [1]. ASM:lle tarjotaan loogisia levylaitteita (LUN, Logical Unit Number), joista se muodostaa hajautetun levyryhmän. Tätä levyryhmää käyttävät RAID:n tapaan kaikki tietokantaklusterin osat [38]. ASM:n arkkitehtuuria on esitelty kuvassa 4.3.



Kuva 4.3: Oracle Automatic Storage Management (ASM) -arkkitehtuuri [35]

Oracle tietokannan kanssa voidaan myös ASM:n sijaan käyttää Oraclen omaa OCFS2 klusteroitua tiedostojärjestelmää. MySQL:n tai PostgreSQL:n kanssa voidaan käyttää esimerkiksi DRBD:tä (Distributed Replicated Block Device) [39], joka on saatavilla Linux-käyttöjärjestelmissä. DRBD eroaa Oraclen toteutuksesta siinä mielessä, että se ei ole itsessään tiedostojärjestelmä, vaan luo virtuaalisen levylaitteen, jonka päällä voidaan käyttää perinteisiä Linuxin tiedostojärjestelmiä kuten esimerkiksi EXT4:sta [40]. Levyjärjestelmien välisen linkin on oltava nopea, ja siksi esimerkiksi Oracle RAC -tuotteen kanssa suositellaan dedikoitujen kuituyhteyksien käyttämistä [1].

4.1.2. Muut riippuvuudet

Tietokannan lisäksi web-sovelluksilla voi olla riippuvaisuuksia muihinkin toiminnan kannalta kriittisiin järjestelmiin. Esimerkiksi autentikaatiosta ja autorisaatiosta saattaa huolehtia LDAP-hakemisto, jota ilman käyttäjät eivät voi kirjautua järjestelmään. LDAP:n -hakemistorakenteen vuoksi sen monistaminen muistuttaa hyvin paljon relaatiotietokannan monistamista. OpenLDAP tarjoaa itsessään MirrorMode -toiminnallisuuden, joka muistuttaa tietokantason replikointia. Siinä LDAP-hakemistoklusterin solmut viestivät muutoksista toisilleen, ja kokonaisuus on täysin toiminnallinen niin kauan kuin yksikin

solmu on toiminnassa [3].

4.2. Tilallisuus ja istunnot

Web-sovellusten käyttämä HTTP-protokolla (Hypertext Transfer Protocol) on tilaton protolla, eli jokainen pyyntö on oma itsenäinen kokonaisuutensa, jolla ei ole tietoa edellistä [41]. Siitä huolimatta HTTP:tä käyttävät nykyaikaiset web-sovellukset hyödyntävät ja käyttävät yhä enemmissä määrin tilallisuutta. Yleisin käyttökohde tilallisuudelle on käyttäjän toimintojen seuraaminen istunnon avulla, esimerkiksi autorisointiroolin tai verkkokaupassa olevan ostoskorin vuoksi [42].

4.2.1. Istunnot ja tilatieto

Istunto muodostaa yhteyden käyttäjän selaimen ja palvelimen välille useamman HTTP-pyynnön ajaksi. Istuntotietoon tallennettavat tiedot riippuvat sovellukesta, mutta esimerkiksi verkkokaupan tapauksessa istuntoon voi olla yhdistettynä käyttäjän ostoskori. Tämä tieto tarvitsee tallettaa sellaiseen paikkaan, että se on sovelluksen käytettävissä [4]. Tyypillisesti tämä paikka on ollut joko sovellusta suorittavan palvelimen muisti. Mikäli tilatieto on tärkeää tai oltava useamman sovelluspalvelimen käytettävissä voidaan tilatieto tallentaa erilliseen tietokantaan [43]. Istuntotiedot voitaisiin myös kokonaisuudessaan tallentaa evästeenä käyttäjän selaimeen, mutta tämä aiheuttaa merkittäviä tietoturvauhkia [41].

Korkean saatavuuden palvelun tapauksessa asiakasta palvelee useita eri palvelemia. Kun asiakkaan pyyntö saapuu palvelimelle, on palvelimen osattava yhdistää käyttäjä hänen istuntoonsa. Jos asiakkaan istunto on avattu toisella palvelimella ja istuntoa ei ole jaettu, ei asiakkaan tietoja voida näyttää. Tätä ongelmaa on kierretty pakottamalla käyttäjän yhteys aina samalle palvelimelle istunnon ajaksi (engl. sticky sessions). Tätä tekniikkaa käyttäessä voi sovelluspalvelin itsenäisesti huolehtia asiakkaan istunnosta, sillä muut palvelimet eivät saa asiakkaan pyyntöjä tämän istunnon aikana. Korkean saatavuuden kannalta tämä kuitenkin aiheuttaa ongelmia, sillä tämän yksittäisen sovelluspalvelimen vikaantuessa katoaa myös asiakkaan istunto. Tätä ratkaisua käytettäessä istuntotiedoissa ei saisi olla mitään tietoja, jotka eivät saisi kadota, mutta esimerkiksi autentikoivassa järjestelmässä käyttäjä joutuu kuitenkin luomaan uuden istunnon kirjautumalla uudelleen [4].

Istuntotiedon tallentaminen tietokantaan mahdollistaa aidon kuormantasauksen asiakkaita palvellessa, sillä jokaisella palvelimella on identtinen ja ajantasainen istuntotieo saatavilla. Toisaalta tämä aiheuttaa ylimääräistä liikennettä sovelluspalvelimen ja tietokantapalvelimen välillä. Useimmiten tähän tarkoitukseen käytetäänkin normaalista tietokannasta erillistä välimuistitietokantaa kuten Memcachedia [44].

4.3. Kuormantasaus ja reititys

Asiakkaat kommunikoivat web-sovelluksen kanssa käyttäen TCP/IP-protokollaa. Asiakkaan selaimen kannalta on merkityksentä kuinka monimutkainen ja monikäsitteinen palvelun rakenne on, sillä selain tarvitsee vain yhden päätepisteen (eli IP-osoitteen), johon ohjata pyyntönsä [45]. Korkean saatavuuden kannalta on oleellista, että tämä päätepiste on aina saatavissa, ja ajantasalla. Mikäli verkkoyhteys asiakkaalta palveluun on katki, ei monimutkaine HA-järjestelmä laitesalissa palvele asiakasta.

4.3.1. Nimipalvelut

Kun asiakas tekee pyynnön haluamaansa web-palveluun käyttää hän tyypillisesti URL-osoitteessa (Uniform Resource Identifier) nimeä, eikä suoraan palvelun ip-osoitetta. Käyttäjän selain tekee ensimmäisenä nimipalvelupyynnön annetulle osoitteelle, esimerkiksi www.tut.fi -osoitteelle, ja saa takaisin käyttämältään nimipalvelimeltaan sitä vastaavan ip-osoitteen. DNS-järjestelmä kuitenkin mahdollistaa useamman osoitteen palautuksen samassa pyynnössä, ja palautuksien järjestys voi olla myös satunnainen. Selaimet ja muut ohjelmat pyrkivät käymään tätä listaa läpi järjestyksessä, joten mikäli ensimmäiset osoitteet eivät vastaa käytetään jälkimmäisiä. Listan järjestyksen satunnaistaminen auttaa myös kuorman tasaamisessa sillä, teoriassa asiakkaille jaettavat ensimmäiseet osoitteet jakautuvat tasan, eli kaikkia käytetään ensimmäisenä yhtä paljon [45].

Korkean saatavuuden kannalta DNS mahdollistaa tehokkaan kuormantasausta yksinkertaisesti. Ongelmallista on kuitenkin se, että DNS-järjestelmä perustuu vahvasti välimuistin käyttämiseen. Käyttäjien tietokone, paikallinen nimipalvelin sekä muut matkalla käytettävät nimipalvelimet saattavat kukin pitää muistissaan vanhaa tietoa. Jokaiselle tietueelle on kuitenkin määritelty voimassaoloarvo (TTL, Time To Live) sekunneissa, jonka ummettua kunkin laitteen tulisi uusia tietonsa. TTL:t ovat useimmiten pieniä, noin 5-10

minuutin mittaisia,mutta tämäkin voi olla pitkä aika kriittisissä palveluissa muutoksien näkymiseksi asiakkaalle asti [45].

DNS-kuormantasauksella on kuitenkin huonot puolensa. Lyhyet TTL:t lisäävät DNS-liikennettä eivätkä kaikkien operaattorien paikalliset DNS-palvelimet noudata annettuja TTL-arvoja vaan säästääkseen resursseja kasvattavat näitä jopa tunneilla. DNS-kuormantasaus vaatii myös tietoa järjestelmän tilasta, jotta se osaa mukautua tiettyjen resurssien lisääntyneeseen kuormaan ja tarjota näitä uusille asiakkaille harvemmin [45].

4.3.2. Reititys

Reitityksellä tarkoitetaan IP-pakettien reittiä asiakkaalta kohde IP-osoitteeseen. Tämä reitti kulkee erilaisten reitittimien kautta jotka kukin ohjaavat paketin seuraavalle reitittimelle. Reitityksen muokkaaminen mahdollistaa pakettien ohjaamisen eri tilanteissa eri fyysisiin pisteisiin esimerkiksi poikkeustilanteissa (ohjataan paketit varalaitetilaan), tai normaalitilanteessa jossa halutaan ohjata käyttäjät palveluntarjoajan maantieteellisesti lähimpään laitetilaan.

Internetissä tapahtuva reititys perustuu reititystauluihin. Nämä taulut voivat olla staattisia ja manuaalisesti luotoja tai dynaamisia jotka muodostuvat ja päivittyvät kun reitittimet kommunikoivat erilaisilla protokollilla, kuten BGP (Border Gateway Protocol), keskenään [45]. Tarjoamalla vaihtoehtoisia reittejä useamman operaattorin kautta kyetään estämään yksittäisen operaattorin, reittimen ja fyysisen linkin katkeamisen aiheuttama katkos.

4.4. Suorituskyvyn takaaminen

Julkisessa verkossa olevat web-sovellukset ovat käytännössä kenen tahansa ja milloin tahansa käytettävissä. Julkisten palveluiden käyttäjämääriä ja käyttöintensiteettiä on usein hyvin vaikeaa tarkasti etukäteen arvioida. Esimerkiksi sosiaalisen median palvelu Facebook kasvoi alle vuodessa miljoonan käyttäjän palveluksi, ja seuraavan vuoden aikana vielä kuusinkertaisti käyttäjämääränsä kuuteen miljoonaan [46]. Näin nopea kasvu vaatii erityishuomiota palvelun skaalautumiseen.

4.4.1. Automaattinen skaalautuminen

Web-sovellusta ajavan infrasktuurin skaalautuminen kuorman vaatimaan tasoon on yleistynyt pilvipalveluiden myötä. Tässä tapauksessa esimerkiksi sovellus- ja tietokantapalvelimia monistetaan, tai käytönaikaisesti päivitetään nopeampiin niin kauan, että ympäristö ei ole ylikuormittunut. Usein tällaisissa tapauksissa myös toiseen suuntaan skaalautuminen on mahdollista hiljaisina ajankohtina [47].

Infrasktuurin skaalautuminen on helpointa ja tehokkainta vertikaalisesti. Vertikaalinen skaalautuminen, eli palvelinten ja resurssien monistaminen, on kuitenkin sovelluksen kannalta haastavampaa sillä, se vaatii sovelluksen osilta (sovelluspalvelimet, tietokannat jne.) tukea. Suuret kokonaisuudet myös monimutkaistavat palvelun rakennetta replikoinnin ja hajauttamisen suhteen [11].

Yksittäisten raskaiden operaatioiden nopeuttaminen vertikaalisesti skaalautumalla on hyvin haastavaa, sillä se vaatii toiminnolta säikeistystukea. Web-sovellusten tapauksessa vertikaalinen skaalautuminen kuitenkin toimii lähtökohtaisesti hyvin sillä skaalautumistarpeet tulevat pyyntöjen lisääntyessä, ja pyyntöjä voidaan hajauttaa eri sovelluspalvelimille ja niiden takana oleville tietokantapalvelimille. Yksittäiset operaatiot eivät siis muutu raskaammiksi, vaan operaatioiden määrä kasvaa [11].

4.4.2. Ohjelmiston mukautuminen kuormaan

Sen lisäksi, että ohjelmistoa suorittavaa ympäristöä voidaan muokata lennosta, voidaan sovelluksesta luoda kuormaan mukautuva. Tämä tarkoittaa sitä, että ohjelmisto osaa muokata itseään kevyemmäksi mikäli palvelupyyntöjä alkaa odottamattomasti tulla enemmän kuin mitä järjestelmä kykenee käsittelemään.

Tällainen tilanne saattaisi ilmetä esimerkiksi uutissivustolla jonkin erityisen kiinnostavan uutisen vuoksi, ja pyynnöt luultavasti kohdistuisivat tähän uutiseen. Tässä tilanteessa sovellus voisi siirtyä yksinkertaisempaan tilaan ja poistaa käyttäjien näkyviltä esimerkiksi tiettyjä tyylitiedostoja tai kehittyneitä toimintoja. Tässä tilanteessa palvelun käyttökokemus olisi heikentynyt, mutta käyttäjät pääsisivät kuitenkin käsiksi haluamaansa materiaaliin.

5. ESIMERKKITOTEUTUS KORKEAN SAATAVUUDEN WEB-JÄRJESTELMÄN INFRASTRUKTUURISTA

Tässä luvussa esitellään esimerkkitoteutus korkean saatavuuden järjestelmän HA-arkkitehtuurista. Luvussa käydään läpi tyypillisen Web-sovelluksen esimerkin kautta korkean saatavuuden perinteiset haasteet ja niihin käytetyt ratkaisumenetelmät. Ratkaisut ja haasteet perustuvat luvussa 4 esitettyihin kohtiin.

5.1. Vaatimukset ja konteksti

Esimerkkijärjestelmä ei pohjaudu mihinkään reaalimaailman järjestelmään, mutta sen vaatimukset ja vaatimustaso mukailevat hyvin läheisesti erään julkishallinnon korkean saatavuuden järjestelmäkokonaisuutta.Tämän vuoksi esimerkkinä vaatimuksista käytetään valtion määrittelemää VAHTI-ohjeistusta [48]. Järjestelmä on nimetty "Karo":ksi siihen viittaamisen helpottamiseksi.

5.1.1. VAHTI-ohjeen saatavuustasot

Suomen valtion ICT-järjestelmien saatavuuteen liittyviä tasoja ja ohjeistuksia on kerätty VAHTI -ohjeistukseen. Ohjeistus on hyvin kattava, ja keskittyy pitkälti tämän työn ulkopuolisiin asioihin kuten tietoturvaan ja strategiaan. VAHTI-ohjeistuksessa kuitenkin määritellään viisi eri vaatimustasoa julkishallinnon järjestelmille, jotka ovat avoin taso, perustaso, korotettu taso, korkea taso ja erityistaso.

VAHTI-ohjeen ICT-varautumistasot ovat apuväline varautumistoimenpiteiden yhtenäistämiseen valtion järjestelmissä. Se myös antaa viitteellisen arvion siitä, miten vikasietoinen käytössä oleva järjestelmä on, sillä korkeimpiin tasoihin liittyy tason vahvistava auditointi. Jokaisen julkishallinnon viranomaisorganisaation on saavutettava näistä vähin-

tään perustaso.

Avoin taso on siirtymävaiheisiin tarkoitettu taso, jossa järjestelmän luokittelu ja toteutus oikealla tasolle on kesken. Siinä sallitaan muista tasoista poiketen mm. yleisten pilvipalveluiden käyttö.

Perustaso on tarkoitettu verkostoituneeseen sähköiseen asiointiin. Perustason palvelut ovat tyypillisesti virka-aikaan käytettäviä järjestelmiä, ja palautumisen virhetilanteista on sallittu kestävän jopa seuraavaan työpäivään.

Korotettu taso on organisaation kriittisten toimintojen minimitaso. Tällaisia palveluita voivat olla esimerkiksi yhteiskunnan elintärkeitä toimintoja tukevat tai kansalaiselle poikkeustilanteissa keskeiset palvelut. Korotetun tason palveluita seurataan ympärivuorokautisesti, ja niiden on toimittava vaikka tietoliikenneyhteydet Suomesta ulkomaille olisivat lamautuneet. Korotetusta tasosta lähtien kaikkien tasojen saavuttaminen auditoidaan.

Korkean tason järjestelmiä ovat esimerkiksi hallinnon taruvallisuusverkko ja turvallisuusviranomaisten operatiiviset järjestelmät. Tämän tason järjestelmien pienetkin katkot aiheuttavat joko vakavia toiminnallisia häiriöitä tai isoja taloudellisia vahinkoja. Korkean tason järjestelmän oleellisimmat erot korotettuun tasoon ovat tilaaja- ja käyttäjäorganisaatioilta vaaditut järjestelyt joilla taataan kyky nopeaan päätöksentekoon virhetilanteissa sekä käytännönläheisempänä se,että yksittäisen konesalin tai tietoliikenneyhteyden tuhoutuminen ei saa lamauttaa järjestelmää.

Erityistaso on sellaisia kriittisiä palveluita varten joiden toiminnan luonteen vuoksi on jouduttu soveltamaan korotetun ja korkean tason vaatimuksia ja käyttämään näistä poikkevia toimintatapoja. Järjestelmää ohjaava ministeriö päättää erityistaso luokituksesta, mutta sen hyväksyy valtiovarainministeriö. Tälläkin tasolla on pakollinen auditointi kansallisen tietoturvaviranomaisen (NCSA-FI) toimesta.

5.1.2. Järjestelmän vaatima taso

Esimerkkijärjestelmän vaatimuksena on korkean tason vaatimusten täyttäminen. Tämän tason oleellisimmat huomioon otettavat asiat ovat siis seuraavat:

- Riippumattomuus yksittäisestä konesalista tai tietoliikenneyhteydestä.
- Suunnitelma palveluiden siirtämisestä toiseen tilaan, mikäli alkuperäinen muuttuu käyttökelvottomaksi.

- Järjestelmän toimiminen tilanteessa, jossa Suomen tietoliikenneyhteydet ulkomaille ovat lamaantuneet.
- Pienetkin katkokset aiheuttavat vakavia toiminnallisia häiriöitä johonkin palveluun.
- Tilaaja- ja käyttäjäorganisaatiolla oltava valmius nopeisiin toimiin.

Näiden kohtien lisäksi on useita tarkempia säännöksiä, kuten esimerkiksi varautuminen neljän tunnin sähkökatkoksiin [48]. Näitä spesifejä vaatimuksia käydään läpi myöhemmissä luvuissa soveltuvissa kohdissa. Ohjeistuksen määrittelemät vaatimukset ovat kuitenkin yleisesti ottaen hyvin korkealla tasolla, ja niiden täyttämiseen on monia erilaisia tekniikoita.

5.1.3. Tarjottavan palvelun ohjelmistoarkkitehtuuri

Esimerkkijärjestelmänä käytetään tyypillistä Java-pohjaista web-sovellusta. Sovellus tarvitsee toimiakseen relaatiotietokannan, ja sen tulee tukea useaa yhtäaikaista käyttäjää, jotka autentikoidaan ja autorisoidaan keskitetyssä tunnushallintajärjestelmässä.

Sovellus ei näy julki-internettiin, ja voidaankin olettaa, että siihen tulevat yhteydet tulevat vain sitä käyttävän organisaation tietystä verkkoalueesta.

5.2. Infrastruktuurin suunnittelu

Tässä kohdassa käydään läpi ne keinot joilla esimerkkipalvelu saadaan täyttämään edellisessä alikohdassa esitetyt korotetun tason oleellisimmat vaatimukset. Useimpiin vaatimuksiin ja niiden implikaattisiin vaatimuksiin on olemassa useita erilaisia ratkaisutapoja, ja tässä kohdassa niistä pyritään esittelemään ne vaihtoehtoiset ratkaisut jotka eivät yksittäisinä paloina vaikuta muuhun arkkitehtuuriin.

5.2.1. Sovelluksen asettamat haasteet infrastruktuurin suunnittelulle

Ennen varsinaisen arkkitehtuurin suunnittelua on kiinnitettävä huomiota sovelluksen vaatimiin erityispiirteisiin, jotta ne osataan etukäteen huomoida infrastruktuurissa.

Autorisaatio/autentikointi. Järjestelmä ei itsessään sisällä käyttäjätietoa, vaan hakee nämä ulkopuolisesta käyttäjähakemistosta. Tämän vuoksi myös tämä käyttäjähakemisto on sisällytettävä suunnitteluun, sillä ilman sitä käyttäjä ei pääse kirjautumaan järjestelmään ja näinollen käyttämään sitä. Olemassaoleva käyttäjähakemisto on kuitenkin LDAP-palvelu, joka tukee replikointia eri noodien välillä.

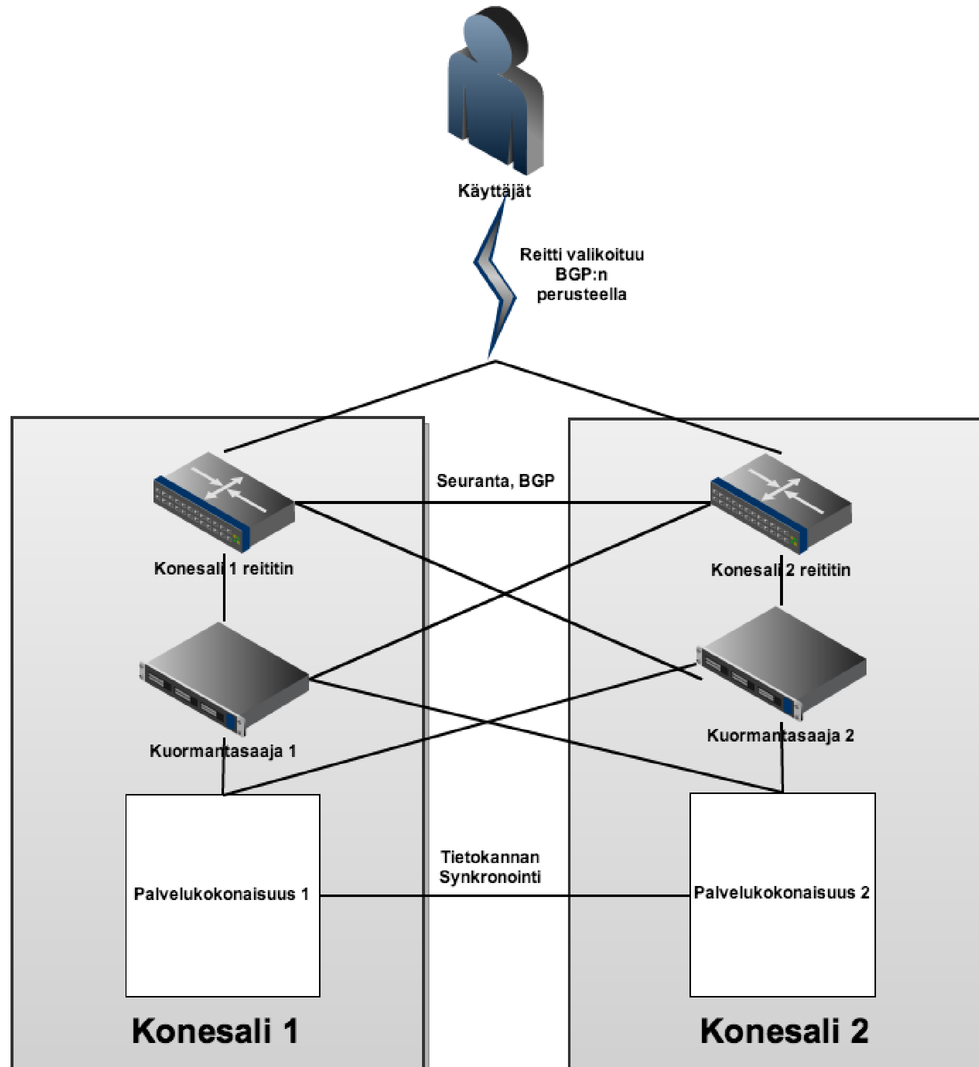
Istunto. Karon käyttäjälle tarjotut toiminnot riippuvat käyttäjäoikeuksista. Järjestelmän on tuettava autentikaatiota ja autorisaatiota, joten näiden vuoksi on ylläpidettävä istuntoa. Tämän lisäksi on käyttäjien suorittamat toiminnot oltava monitoroitavissa ja yhdistettävissä niiden tekijään. Tyypillisesti Java-pohjaisissa sovelluksissa istuntunnosta vastaa sovellus itse, ja istuntotieto sijaitsee sovelluspalvelimessa muistinvaraisesti. Karo kuitenkin halutaan monistaa itsenäisiin osiin kahteen eri laitetilaan, joten sovelluspalvelimiakin on useita. Istuntotieto voitaisiin jakaa sovelluspalvelimien välillä tai esimerkiksi persistoida jaettuun tietokantaan, mutta tämä vaatisi jatkuvaa kommunikointia laitteiden välillä, sekä monimutkaistaisi sovelluksen rakennetta jaetun istunnonhallinnan vuoksi. Tästä syystä Karon arkkitehtuurissa ollaankin päädytty sticky sessions -tekniikkaan. Sticky sessionia hyväksikäyttämällä käyttäjän tekemät HTTP-pyynnöt päätyvät saman istunnon aikana aina samalle sovelluspalvelimelle [41].

Tietokannan synkronointi. Järjestelmän käyttämä tietosisältö on monistettava kumpaankin konesaliin toimivuusvaatimuksien vuoksi. Relaatiotietokannassa oleva tietosisältö ei kuitenkaan ole staattista, vaan jatkuvasti käyttäjien toimien mukaan muokkaantuvaa. Tästä syystä on kyettävä pitämään kumpikin tietokanta jatkuvasti ajantasalla niin, että käyttäjien käytettävissä on aina uusin mahdollinen tieto, ja häiriötilanteissa tietoa ei huku tai korruptoidu. Useimmat modernit tietokannat, kuten Oracle Database, MySQL tai PostgreSQL tarjoavat tuen synkronoinnille sisäänrakennettuna.

5.2.2. Korkean tason järjestelmäarkkitehtuuri

VAHTI-järjestelmän korkea taso määrittelee, että järjestelmän tulee olla kahdennettu useampaan konesaliin. Tästä syystä tämä järjestelmä tulee toteuttaa niin, että kummastakin laitesalista löytyy täysin identtiset laitteet ja ohjelmistot. Kumpikin laitesali toimii siis itsenäisenä kokonaisuutena, ja näiden kahden laitesalin välillä replikoidaan palvelun käyttämä tietosisältö. Toisen laitesalin tai esimerkiksi sen tietoliikenneyhteyden hävites-

sä voidaankin siis automaattisesti siirtyä käyttämään toista. Tämä ratkaisu mahdollistaa myös kuormantasaamisen normaalissa käyttötilanteessa, sekä tarjoaa mahdollisuuden latenssiriippuvaisissa palveluissa tarjota käyttäjille aina optimaalisin sijainti.

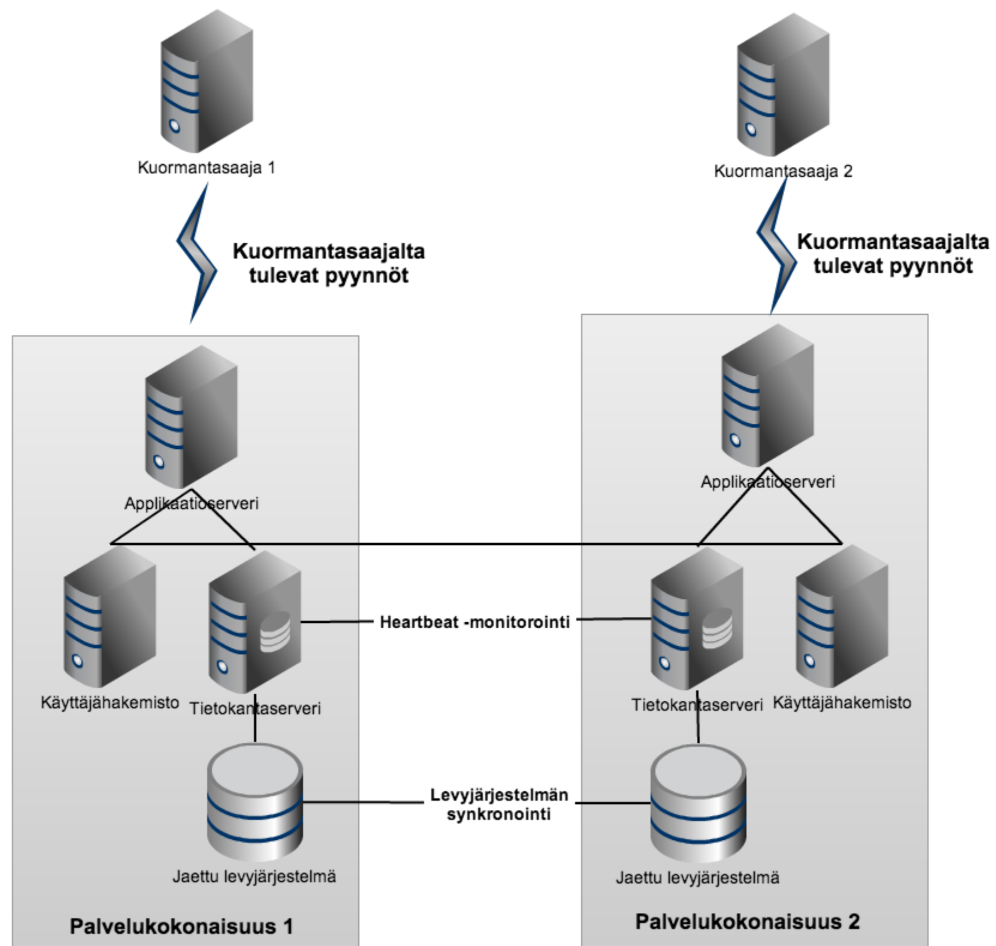


Kuva 5.1: Korkean tason järjestelmäarkkitehtuuri

Kahden laitesalin ratkaisu on esitetty kuvassa 5.1. Verkkotasolla korkean saatavuudenden ratkaisuun on siis päästy seuraavilla menetelmillä:

- BGP:n hyödyntäminen reitityksessä.
- Tilan seuranta reitittimissä virhetilanteiden tunnistamiseksi.
- Asiakkaiden tilan seuranta reitittimissä sticky sessionin mahdollistamiseksi.
- Sticky sessioneiden käyttö istuntojen replikoinnin välttämiseksi.

Palvelinympäristö voidaan toteuttaa käyttämällä joko perinteistä mallia jossa fyysisiä laitteita hankitaan kiinteästi määriteltyihin laitesaleihin. Vaihtoehtoinen ratkaisu on virtualisoida kaikki palvelun tarjoamiseen liittyvät palvelinalustat ja näin pitää järjestelmä ketterästi siirrettävänä laitesalista ja fyysisestä laitteistosta toiseen. Virtuaalisoidun ympäristön skaalautuvuusedut ovat myös merkittävät.



Kuva 5.2: Palvelukokonaisuuden rakenne

Palvelukokonaisuus on avattu kuvassa 5.2, jossa näkyy miten kumpikin laitesali sisältää identtiset järjestelmäkokonaisuudet. Itse sovellus sijaitsee sovelluspalvelimilla joihin HTTP-pyynnöt ohjataan kuormantasaajilta. Tämä malli mahdollistaa sovelluspalvelimien päivittämisen itsenäisesti ilman muutoksia reititykseen. Kuormantasaajat voidaan määritellä ohjaamaan kaikki tulevat pyynnöt töiseen konesaliin, ja palauttaa tilanne päivitysten valmistuttua. Konesali kerrallaan tehtävät päivitykset mahdollistavat rutiinipäivitysten tekemisen ilman katkoja palveluun. Mikäli päivityksessä ilmenee ongelmia, voi-

daan myös väliaikaisesti jatkaa toisen konesalin käyttämistä kunnes ongelma on ratkaistu. Kuormantasaajan tässä ratkaisussa käytetään HAProxy -ohjelmistoa, jota ajetaan normaalissa palvelinlaitteistossa [49]. Sovelluspalvelimeksi on valittu Apache Tomcat, joka huolehtii Java-pohjaisen ohjelmiston ajamisesta [6]. Käyttäjähakemistosta vastaa OpenLDAP -ohjelmisto [3], jota suoritetaan klusterointitilassa käyttäen Mirror Mode -ominaisuutta.

Tietokannan toteuttamiseen on valittu Oracle RAC [35]. Tietokantakokonaisuus vaatii kaksi erilaista heartbeat-pohjaista valvontamekanismia. Käytettäessä Oraclen RAC -tuotetta tietokantasovellukset valvovat kutakin olemassaolevaa noodia UDP-pohjaisilla paketeilla, ja aktiiviset noodit päätetään äänestämällä niiden saatavuuden mukaan (eli palautuuko vastaus lähetettyihin paketteihin). Tämän lisäksi useamman laitesalin ratkaisussa tulee huolehtia itse datan replikoinnista, johon Oracle RAC:n kanssa tässä ratkaisussa käytetään Oracle ASM-ohjelmistoa [1].

5.3. Toteutuksen verifointi skenaarioiden avulla

Tässä kohdassa käydään läpi tyypillisimmät saatavuuteen liittyvät häiriötilanteet ja näiden vaikutukset palvelun saatavuuteen loppukäyttäjän näkökulmasta.

5.3.1. Aikataulutettujan päivitysten tekeminen eri osiin

Jokaisen järjestelmäkokonaisuuden tulee varautua päivitysten tekemiseen. Karon tapauksessakin tämä tulee ottaa huomioon, sillä Karoa päivitetään noin kerran vuodessa ja mahdollisia muita kompontteja kuten sovelluspalvelinta ja tietokantaohjelmistoa tarpeiden mukaan. Nämä päivitykset pyritään pääosin suorittamaan virka-ajan ulkopuolella, mutta Karon kriittisten saatavuusvaatimusten vuoksi nämäkään päivitykset eivät saa lamauttaa koko järjestelmää pitkiksi ajoiksi. Päätavoite on, että komponetit voidaan päivittää aina niin, että järjestelmä pysyy päivityksenkin ajan käytössä.

Tähän tavoitteeseen päästään esimerkkiarkkitehtuurilla sillä, että käyttäjät voidaan läpinäkyvästi ohjata kahdella tasolla eri laitesaleihin ja sovellusympäristöihin. Ensimmäinen tapa on reititys, eli BGP-reitin hinnan muuttaminen, ja toinen tapa on tehdä asia kerrosta alempana kuormantasaajan tasolla. BGP-reittien muuttaminen ei tule voimaan välittömästi, joten esimerkiksi sovellusympäristön päivitysten tapauksessa ohjaukset kannattaa suorittaa kuormantasaajan tasolla.

Ainoa varsinainen ongelmakohta tässä päivitysmallissa on se, että mikäli halutaan tehdä tietokantamuutoksia sovellukseen tulee ajossa olevien sovellusten olla samassa versiossa muutosten valmistuttua. Tällaisten muutoksien tulisi kuitenkin olla erityisen harvinaisia.

5.3.2. Yksittäisten laitteiden vikaantuminen

Infrakstuurikokonaisuus koostuu reittimistä, kuormantasaajista, sovelluspalvelimista, käyttäjähakemistoista ja tietokantaservereistä sekä näiden välisistä yhteyksistä. Siitä huolimatta, että jokainen näistä laitteista on kahdennettu tulee tutkia miten järjestelmä kokonaisuutena käyttäytyy kun yksittäinen laite vikaantuu.

Reititin. Tässä arkkitehtuurissa on yksinkertaisuuden vuoksi oletettu, että reitittimiä on vain yksi jokaisessa konesalissa. Luultavasti niitä oikeassa käyttötapauksessa olisi useita, mutta mikäli ne kokonaisuutena (tai tässä tapauksessa yksilöittäin) vikaantuisi olisi verkkoyhteys kyseiseen saliin lamaantunut.

Kuormantasaaja. Kuormantasaaja voi vikaantua laite- tai ohjelmistovian vuoksi. Kuormantasaaja saatetana ottaa myös väliaikaisesti pois käytöstä esimerkiksi ohjelmistopäivitystä varten. Yksittäisen kuormantasaajan käytöstä poistuminen havaitaan reitittimessä ja joka ohjaa tulevat pyynnöt toiselle kuormantasaajalle. Ratkaisu ei aiheuta palvelukatkosta, ja on hyvin nopeasti toipuvainen (sekunneissa), mutta käyttäjien aktiiviset istunnot joudutaan luomaan uudelleen. Mikäli palvelun käyttötaso on korkea voi myös palvelun responsiivisuus heiketä sillä ylimääräinen kuorma yhdessä laitesalissa voi aiheuttaa hidastusta. Myös ylimääräinen hyppy palvelinsalista toiseen aiheuttaa lievän lisäviiveen, mutta tämä tuskin on havaittavissa sillä salien väliset viiveet ovat tyypillisesti muutamia millisekuntteja.

Sovelluspalvelin. Sovelluspalvelimen vikaantuminen aiheuttaa kyseisen laitesalin poistamisen käytöstä. Vikaantumisen havaitsee kuormantasaaja, joka alkaa ohjata pyyntöjä toiseen saliin. Tässäkin tapauksessa käyttäjä joutuu kirjautumaan uudelleen. Reagointi ongelmaan ja siitä toipuminen vie tässäkin tapauksessa joitain sekuntteja.

Käyttäjähakemisto. Käyttäjähakemistopalvelimen vikaantuminen aiheuttaa ongelmia kirjautumiseen. Mikäli katkos on lyhyt, ei se aiheuta mitään oiretta jo kirjautuneille käyttäjille, mutta uudet kirjautumiset voivat hetken epäonnistua. LDAP-klusteroinnin vuoksi

OpenLDAP kuitenkin havaitsee ongelman lähes välittömästi ja siirtyy ohjaamaan pyynnöt ehjälle palvelimelle. OpenLDAP:n klusterointi toimii isäntä-orja -tilassa joten tiedon synkronoinnin kanssa ei tule ongelmia. Hakemistoon kirjoitetaan myös verrattain harvoin.

Tietokantapalvelin. Tietokantapalvelimen vikaantuminen havaitaan tietokantaklusterin toisessa osassa. Tietokantapalvelimeen viitataan virtuaaliselle IP:llä joten ainoastaan vikaantumisen aikana vikaantuneessa palvelimessa olleet pyynnöt epäonnistuvat ja muut ohjataan välittömästi toiselle tietokantaklusterin puoliskolle. Käyttäjälle tämän vikaantumisen ei tulisi suoraan näkyä juuri laisinkaan, korkeintaan yhtenä epäonnistuneena pyyntönä, mutta mikäli yhden tietokantapalvelimen suorituskyky ei riitä palvemaan kaikkia asiakkaita voi palvelu hidastua.

Monitorointi- ja synkronointiyhteydet. Eri ohjelmistojen väliset monitorointi- ja synkronointiyhteydet ovat elintärkeitä niiden oikealla toimimiselle. Tällaisten yhteyksien katkeaminen tarkoittaa sitä, että kumpikin itsenäinen osapuoli tekee itsestään isännän ja palvelee asiakkaita. Kun yhteys myöhemmin palautuu saattavat eri puoliskot olla käsitelleet tietosisältöä eri muotoon ja syntyy konflikti. Oracle RAC -ohjelmistossa tämä ongelma on kierretty jaetun levyjärjestelmän avulla. Jos tietokantojen välillä oleva yhteys katkeaa, ongelman ensimmäisenä havaitseva estää toiselta puoliskolta kirjoittamisen jaettuun levyjärjestelmään [1]. OpenLDAP:ssa on automaattinen tiedon yhdistämistoiminto, jolla eri tilassa olevian solmujen tiedot voidaan yhdistää. Konfliktien tapauksessa, esimerkiksi jos samaa tietoa on muutettu kummallakin isännällä, joutuu käyttäjä käsin korjaamaan kumman solmun tieto tullaan hyväksymään.

5.3.3. Konesalin laajuinen palvelukatkos

Mahdollisia syitä tällaisia katkoille voivat olla ongelmat konesalin tietoliikenneyhteydessä tai viallinen sähkönsyöttö. Sähkönjakelun katkeaminen konesaliin ei kuitenkaan tulisi tällaista tilannetta aiheuttaa, sillä VAHTI-ohjeistus määrittelee neljän tunnin varasähkövaatimuksen. Sähkön jakelu tiettyyn laitekaappiin voi kuitenkin vikaantua tästä huolimatta.

Konesalin laajuinen palvelukatkos aiheuttaisi jäljellejääneen reittimen pääasiallisen käytön, sillä toinen reitti merkattuuntuisi kuolleeksi. Tämä tapahtuu automaattisesti, eikä aiheuttaisi kuin joidenkin sekunttien katkoksen loppukäyttäjille.

Tietokantatasolla synkronointi lakkaisi toimimasta, ja toisen konesalin palatessa tuleekin tarkkailla jaetun levyjärjestelmän toipumista tilanteesta. Tällaisessa tilanteessa konflikteja ei tulisi kuitenkaan syntyä, sillä ristiriitaisia kirjoituksia ei ehtisi tapahtua. Mikäli ongelmia esiintyisi voitaisiin siirtyä käyttämään toista konesalia aina

5.4. Havainnot arkkitehtuurin vahvuuksista ja heikkouksista

Järjestelmä on hyvin vikasietoinen sillä kummankin laitesalin resursseja voidaan tehokkaasti käyttää ristiin. Tästä huolimatta aktiivisia laitteita on hyvin paljon, joka näkyy järjestelmän suhteellisen suurena kompleksisuutena. Monimutkainen konfiguraatio useisiin laitteisiin, sekä jaettujen resurssien synkronoin voi jopa aiheuttaa enemmän haittaa kuin hyötyä. Järjestelmäarkkitehtuurin raskaus näkyy myös tarvittujen itsenäisten monitorointi- ja synkronointiverkkoyhteyksien isossa määrässä. Tämä onkin merkittävin rajoittava tekijä infrasktuurin siirrolle tai monistamiselle useamaan konesaliin.

Järjestelmän siirtäminen konesalitasolla tapahtuvaan isäntä-orja -tilaan yksinkertaistaisi monia tähän liittyviä ongelmia. Tässä arkkitehtuurissa normaalitilanteessa toinen laitesali on kuitenkin täysin käyttämättömänä loppukäyttäjän näkökulmasta, joka taas aiheuttaa paineita yksittäisen konesalin laitteiden suorituskyvylle.

Sticky Session -tekniikan käyttö yksinkertaistaa sovelluspalvelimilta vaadittua älykkyyttä. Se kuitenkin aiheuttaa käyttäjille näkyvän katkoksen aina kun alunperin valittu sovelluspalvelin poistuu käytöstä. Käytännössä ongelma on kuitenkin hyvin lievä sillä, varsinaista käyttökatkosta palveluun ei tule vaan käyttäjä joutuu ainoastaan kirjautumaan uudellaan palveluun.

Tietokannan toimiessa isäntä-isäntä -tilassa on mahdollisuuksia ristiriitaisille kirjoituksilla. Tällaista voisi tapahtua esimerkiksi tietokantaverkkoyhteyden vikaantuessa niin, että samanaikaisesti kummassakin salissa olisi tapahtumassa kirjoituksia ja valvontajärjestelmä ei ole vielä havainnut ongelmaa. Tämä voitaisiin ratkaista siirtymällä isäntä -orja -tilaan, eli määrittelemällä toinen tietokanta sellaiseksi johon kaikki kirjoitukset kohdistuvat ja toinen vain lukutilaan. Isäntätietokannan vikaantuessa voisi lukutilassa ollut tietokanta siirtyä isäntärooliin [1].

Niin sanottujen Split-Brain -tilanteiden, eli tilanteiden, joissa synkronointiyhteyden katkeamisen vuoksi on useampi isäntäsolmu toisistaan tietämättä, välttämis-

tä voitaisiin parantaa erillisellä käyttöjärjestelmätason monitorointiyhteydellä. Linux-käyttöjärjestelmässä tähän tarkoitukseen on Pacemaker-ohjelmisto, joka voidaan määritellä monitoroimaan yhteyttä. Pacemaker voisi oman yhteytensä kautta havaita, että laite ja ohjelmisto ei oikeasti olekkaan poissa käytöstä ja sammuttaa toisen solmun ohjelmisto kunnes synkronointiyhteys on korjattu [3]. Mikäli klusteri olisi laajempi, voitaisiin hyväksikäyttää jopa erillistä valvontapalvelinta tai muodostaa valvontaohjelmistoista neuvosto, joka äänestäisi kulloinkin mikä solmu toimii isäntänä, mikä orjana ja mitkä tulee mahdollisesti sammuttaa.

6. YHTEENVETO

Web-sovelluksilla ohjataan ja viestitään monia yhteiskunnan toiminnan kannalta kriittisiä asioita. Tällaisten sovellusten jatkuva ja luotettava toiminta on äärimmäisen tärkeää. Tämän lisäksi kuluttajat käyttävät yhä enemmissä määrin palveluita verkossa, ja odottavat, että esimerkiksi verkkokauppa on käytettävissä aina tarvittaessa.

Näihin vaatimuksiin joudutaan yhä järeämmin keinoin vastaamaan. Useamman palvelimen, tai jopa konesalin, ratkaisut eivät ole enää erikoistapauksia, vaan jo lähtökohtia useimmille isoille palveluille. Nämä ratkaisut kuitenkin muodostavat hyvin monimutkaisia kokonaisuuksia, joiden hahmottaminen ja ylläpito voi osoittautua hyvinkin työlääksi. Järjestelmien saatavuus ja luotettavuus voikin jopa heiketä, mikäli järjestelmän toteuttuja ei ole tietoinen kaikista ratkaisujen tuomista haasteista.

Työssä havaittiin, että yhtä yleistä korkean saatavuuden haastetta ei nouse muiden yli, vaan tyypillisesti ratkaisut lähtevät matalalta tasolta ja nousevat budjetin salliessa korkeammalle. Matalalla tasolla voidaan ottaa huomioon esimerkiksi yksittäisten kiintolevyjen hajoaminen johon voidaan varautua rakentamalla vikasietoisia loogisia levyjärjestelmiä, tai mikäli tätä ei koeta riittäväksi voidaan sovelluspalvelin monistaa niin, että toisen vikaantuessa toinen edelleen palvelee asiakkaita. Tästä korkeammalle tasolle päästessä aletaan usein monistamaan kokonaisia infrakstuureja aina verkkoyhteyksistä palvelimiin mukaanlukien riippuvuudet kuten tietokannat ja integraatiorajapinnat. Mitä korkeammalle tasolle tässä noustaan sitä monimutkaisempi arkkitehtuurista tulee, ja sitä kalliimmaksi sen tuottaminen muodostuu.

Esimerkkiarkkitehtuuria toteutettaessa kävi selväksi, että resursseja monistaessa oleelliseksi ongelmaksi nousee käytetyn tiedon yhtenäisyys ja ajantasaisuus kussakin sitä käyttävässä laitteessa. Tiedon määrä ja vaihtuvuus on hyvin suurta, joten käytännössä tietoa joudutaan säilyttämään yhdessä paikassa, tai jakamaan solmujen välillä. Jakamisen aiheuttamat haasteet, kuten sen vaatima verkkokaista ja virhetilanteissa mahdollisesti tapahtuvat konfliktit, ovat kuitenkin erittäin tarkasti huomioonotettavia toteutettaessa kor-

kean saatavuuden järjestelmää. Esimerkkijärjestelmän arkkitehtuuria suunniteltaessa tämä korostui, sillä tällä järjestelmällä oli useita riippuvuuksia eri paikoissa sijaitsevaan tietoon.

Esimerkkijärjestelmällä, kuten tyypillisillä web-sovelluksilla yleensäkin, on riippuvuuksia useisiin tietokantoihin, sekä sen tulee ylläpitää ajantasaista ja muuttuvaa istuntotietoa asiakkaasta. Näitä tietoja käytettiin sekä säilöttiin eri solmuissa, joten oli ilmiselvää, että mahdolliset synkronointi- ja eheysongelmat tulivat monistamisen jälkeen ajankohtaisiksi. Näitä ongelmia pyrittiin rajaamaan estämällä rinnakkaisten järjestelmien toiminta, muunmuassa tahmeilla istunnoilla ja isäntä-orja -mallilla, mutta nämä kiertotavat taas heikensivät järjestelmän skaalautuvuutta.

Yhteenvetona voidaan kuitenkin todeta, että esimerkkijärjestelmän infrakstuuri toimii hyvänä lähtökohtana korkean saatavuuden web-sovelluksen toteutukselle. Se esittelee tyypillisiä ratkaisumalleja ja näiden heikkouksia, sillä tavoin, että eri ratkaisuja voidaan modulaarisesti valita erilaisiin tarkoituksiin.

LÄHTEET

[1] Murali Vallath. *Oracle 10g RAC Grid, Services & Clustering*. Digital Press, 2006.

[2] Klaus Schmidt. *High Availability and Disaster Recovery: Concepts, Design, Implementation*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[3] Ralf Haferkamp. OpenLDAP in High Availability Environments. http://ldapcon.org/downloads/Haferkamp-paper.pdf, 2011. Viitattu: 07.04.2013.

[4] Evan Marcus and Hal Stern. *Blueprints for High Availability: Designing Resilient Distributed Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[5] Floyd Piedad and Michael W. Hawkins. *High Availability: Design, Techniques and Processes (Harris Kern's Enterprise Computing Institute Series)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2008.

[6] Tomcat. Apache Tomcat. http://tomcat.apache.org/, 2013. Viitattu: 07.04.2013.

[7] Oracle. Oracle Database 11g. http://www.oracle.com/us/products/database/overview/index.html, 2013. Viitattu: 07.04.2013.

[8] Pasi Vainio. Korkean Saatavuuden Verkkopalvelut. http://publications.theseus.fi/bitstream/handle/10024/5945/Vainio_Pasi.pdf, 2009. Vaasan ammattikorkeakoulu.

[9] Ari Juhani Korhonen. Uhkapuut. http://www.cs.hut.fi/~archie/publications/Uhkapuut.pdf, 1997. Tampereen teknillinen korkeakoulu.

[10] Kilpilinna Jani. Paavilainen Juhani. Huhtanen Karri., Karjalainen Reetta. and Vatiainen Heikki. Tietoturvallinen Ohjelmointi. http://www.cs.tut.fi/~8306000/TK-TT/tt_ohjelmointi.pdf, 2004. Tampereen teknillinen yliopisto.

[11] Cal Henderson. *Building Scalable Web Sites: Building, Scaling, and Optimizing the Next Generation of Web Applications*. O'Reilly Media, Inc., 2006.

[12] John Allspaw. *The Art of Capacity Planning: Scaling Web Resources*. O'Reilly Media, Inc., 2008.

[13] Espen Braastad. Management of High Availability Services Using Virtualization. https://oda.hio.no/jspui/bitstream/10642/522/2/Braastad_Espen.pdf, 2006. University of Oslo.

[14] Mikko Kautto. Palvelutasosopimukset (SLA). http://www.cs.helsinki.fi/group/cinco/teaching/2009/soc-seminaari/abstracts/kautto_abstract.pdf, 2009. Helsingin yliopisto.

[15] Meta Group. IT Performance Engineering & Measurement Strategies: Quantifying Performance Loss. http://www.metagroup.com/cgi-bin/inetcgi/jsp/displayArticle.do?oid=18750, 2000. Viitattu: 07.04.2013.

[16] SANS Institute. Requirements for the Design of a Secure Data Center. http://www.sans.org/reading_room/whitepapers/recovery/requirements-design-secure-data-center_561, 2002. Viitattu: 07.04.2013.

[17] Meta Group. A Comprehensive View of High-Availability Data Center Networking. http://www.synegi.com/docs/HA_Data_Center.pdf, 2004. Viitattu: 07.04.2013.

[18] Paul Venezia. After Hurrican Sandy: Lessons for the Data Center. http://www.infoworld.com/d/data-center/after-hurricane-sandy-lessons-the-data-center-206304, 2012. Viitattu: 07.04.2013.

[19] Cade Metz. Hurricane Sandy Topples New York Data Center, Gawker, Gizmodo. http://www.wired.com/wiredenterprise/2012/10/hurricane-sandy-datagram/, 2012. Viitattu: 07.04.2013.

[20] Gerry Smith. Amazon Power Outage Exposes Risks of Cloud Computing. http://www.huffingtonpost.com/2012/07/02/amazon-power-outage-cloud-computing_n_1642700.html, 2012. Viitattu: 07.04.2013.

[21] David Bader. Cluster Computing Applications. http://www.cc.gatech.edu/~bader/papers/ijhpca.pdf, 1996. The International Journal of High Performance Computing Applications, Volume 15, No. 2, Summer 2001, pp. 181-185.

[22] Richard Sharpe. Just what is SMB? http://www.samba.org/cifs/docs/what-is-smb.html, 2002. Viitattu: 07.04.2013.

[23] Christopher M. Smith. NFS, FAQ. http://nfs.sourceforge.net/, 2013. Viitattu: 07.04.2013.

[24] Uni Lore. Comparative, NAS,SAN,DAS. http://commons.wikimedia.org/wiki/File%3ACompingles2.svg, 2009. Viitattu: 07.04.2013.

[25] Oracle. Project: OCFS2. https://oss.oracle.com/projects/ocfs2/, 2009. Viitattu: 07.04.2013.

[26] Christian Lunden. Evaluation of Different SAN Technologies for Virtual Machine Hosting. http://eternity.iu.hio.no/theses/master2009/christian.pdf, 2009. University of Oslo.

[27] Iikka Jaakkola. Virtuaalikone ja -verkkoympäristön Hyödyntäminen Tietoverkkotekniikan Tutkimus- ja opetusympräistöjen Rakentamisessa. http://lib.tkk.fi/Dipl/2010/urn100221.pdf, 2010. Aalto-yliopiston teknillinen korkeakoulu.

[28] Federico Calzolari. *High Availability Using Virtualization*. PhD thesis, 2006. University of Pisa.

[29] Inc VMware. VMware Virtualization. http://www.vmware.com/, 2013. Viitattu: 07.04.2013.

[30] Citrix Systems Inc. VMware Virtualization. http://www.citrix.com/products/xenserver/overview.html, 2013. Viitattu: 07.04.2013.

[31] Joel Kirch. Virtual Machine Security Guidelines, Version 1.0. http://www.lasr.cs.ucla.edu/classes/239_1.fall10/papers/CIS_VM_Benchmark_v1.0.pdf, 2007. The Center for Internet Security.

[32] Tuomas Kommeri. Pilvilaskennan Suorituskyky – Erityistarkastelussa Drupal-sisällönhallintasovellus Amazonin, Rackspacen ja GoGridin Pilvipalvelimilla. https://jyx.jyu.fi/dspace/bitstream/handle/123456789/27104/URN:NBN:fi:jyu-2011053110946.pdf, 2011. Jyväskylän yliopisto.

[33] Joyce Mlawanda. A comparative Study Of Cloud Environments and the Development of a Framework for the automatic Deployment of Scalable Cloud-Based Applications. http://scholar.sun.ac.za/Fbitstream/Fhandle/F10019.1/F19994/Fmlawanda_comparative_2012.pdf, 2012. Stellenbosch University.

[34] Ilari Moilanen. Tietokantaklusterit. http://www.cs.helsinki.fi/u/jplindst/ps/seminaari_moilanen.pdf, 2007. Helsingin yliopisto.

[35] Oracle Corportation. Oracle® Database Storage Administrator's Guide 11g Release 1 (11.1). http://docs.oracle.com/cd/B28359_01/server.111/, 2007. Viitattu: 07.04.2013.

[36] Shaun Thomas. High Availability with PostgreSQL and Pacemaker. http://wiki.postgresql.org/images/0/07/Ha_postgres.pdf, 2012. Viitattu: 07.04.2013.

[37] Oracle Corportation. MySQL and DRBD Guide. http://downloads.mysql.com/docs/mysql-ha-drbd-en.a4.pdf, 2013. Viitattu: 07.04.2013.

[38] Oracle Corporation. Oracle Automatic Storage Management and Thin Reclamation. http://www.oracle.com/technetwork/database/

oracle-automatic-storage-management-132797.pdf, 2010. Viitattu: 07.04.2013.

[39] LINBIT HA-Solutions GmbH. DRBD - What is DRBD. http://www.drbd.org/, 2013. Viitattu: 07.04.2013.

[40] Nicolas Kaiser. Ext4. http://kernelnewbies.org/Ext4, 2011. Viitattu: 07.04.2013.

[41] Santeri Paavolainen. Skaalautuvuuden ABC, Osa 6: Istunnot. http://blog.codento.com/2010/05/skaalautuvuuden-abc-osa-6-istunnot/, 2010. Viitattu: 07.04.2013.

[42] Gunter Ollman. Web Based Session Management – Best practises in managing HTTP-based client sessions. http://www.technicalinfo.net/papers/WebBasedSessionManagement.html, 2007. Viitattu: 07.04.2013.

[43] Bjørn Hansen. Real World Web: Performance & Scalability. http://www.scribd.com/doc/2569319/Real-World-Web-Performance-Scalability, 2008. Viitattu: 07.04.2013.

[44] Dormando. memcached - a distributed memory object caching system. http://memcached.org/, 2013. Viitattu: 07.04.2013.

[45] Sven Ingebrigt. High-Level Load Balancing for Web Services. http://eternity.iu.hio.no/theses/pdf/master2006/sven-thesis-20060520.pdf, 2006. University of Oslo.

[46] Facebook. Facebook Newsroom. http://newsroom.fb.com/Timeline, 2013. Viitattu: 07.04.2013.

[47] Martti Vasar. A Framework for Verifying Scalability and Performance of Cloud Based Web Application. http://comserv.cs.ut.ee/forms/ati_report/downloader.php?file=9b77daf847970506bdcf6e3d98f7f0a29d6fa360, 2012. University of Tartu.

[48] Valtionhallinnon tietoturvallisuuden johtoryhmä. ICT-varautumisen Vaatimukset. https://www.vm.fi/vm/fi/04_julkaisut_ja_asiakirjat/01_julkaisut/076_ict/20120925ICTvar/vahti_2_2012_NETTI_PDF.pdf, 2012. Viitattu: 07.04.2013.

[49] HAproxy community. HAproxy - The Reliable, High Performance TCP/HTTP Load Balancer. http://haproxy.1wt.eu/, 2013. Viitattu: 07.04.2013.