Katsaus hajautettujen web-palveluiden viansietokykyä parantaviin menetelmiin

Turun yliopisto
Tietotekniikan laitos
LuK-tutkielma
Tietojenkäsittelytiede
Toukokuu 2024
Ville Kalliomäki

TURUN YLIOPISTO
Tietotekniikan laitos

Ville Kalliomäki: Katsaus hajautettujen web-palveluiden viansietokykyä parantaviin menetelmiin

LuK-tutkielma, 26 s.
Tietojenkäsittelytiede
Toukokuu 2024

Hajautetut web-palvelut ovat usein keskitettyä ratkaisua monimutkaisempia järjestelmiä, jotka ovat siten myös alttiita suuremmalle kirjolle vikatiloja. Toimintavarmuudeltaan kriittisempien hajautettujen web-palveluiden viansietokykyä voidaan parantaa soveltamalla niiden käyttöönotossa ja kehityksessä erilaisia viansietokykyä parantavia menetelmiä. Erityyppiset vikatilat täytyy kuitenkin ensin pystyä tunnistamaan niin suunnittelussa kuin web-palvelun toiminnankin aikana, jotta niihin pystytään reagoimaan.

Tässä tutkielmassa kartoitetaan, millaisia vikatiloja hajautetut web-palvelut voivat kohdata, ja mitä menetelmiä on olemassa palveluiden viansietokyvyn parantamiseksi. Ennen vikojen luokittelua tai menetelmien esittelyä määritellään mitä hajautetut web-palvelut tarkoittavat tämän tutkielman yhteydessä, ja kerrotaan yleisesti web-palveluiden arkkitehtuurista sekä miten niiden tilaa ylläpidetään.

Viansietokykyä parantavat menetelmät ovat kategorisoitu sen mukaan, mihin vikatilojen luokkaan ne pääasiassa vaikuttavat. Kaksi ensimmäistä vikojen luokkaa ovat hallitut sekä bysanttilaiset viat. Näihin luokkiin soveltuvia viansietokykyä parantavia menetelmiä käsitellään yhteensä viisi, jotka kaikki ovat web-palvelun ympärille tai ulkopuolelle sijoittuvia järjestelmiä mitkä eivät puutu palvelun sisäiseen toteutukseen. Web-palvelinten ylikuormittumisen estämiseen esitetään kolme eri menetelmää, jotka kaikki hyödyntävät asiakkaiden pyyntöjen ohjausta hajautetun web-palvelun eri palvelimille.

Asiasanat: Hajautetut web-palvelut, viansietokyky, vikatilat

Sisällys

1 Johdanto

Web-palveluiden käyttäjämäärien ja resurssitarpeiden kasvaessa keskitetyn arkkitehtuurin rajat skaalautuvuudessa voivat tulla vastaan, jolloin yksi vaihtoehto suorituskyvyn lisäämiseksi on hajauttaa palvelu useammalle erilliselle tietokoneelle. Hajautettu web-palvelu on keskitettyyn verrattuna monimutkaisempi kehittää ja ylläpitää. Syitä ovat esimerkiksi tietokoneiden kommunikointi epäluotettavien verkkoyhteyksien yli, suurempi määrä ylläpidettäviä laitteita sekä palvelun tilan ylläpito hajautetusti.

Yksi konkreettinen esimerkki laajasta viasta hajautetussa web-palvelussa on vuonna 2018 GitHubissa tapahtunut häiriö, jossa yhteys kahden datakeskuksen välillä katkesi 43 sekunniksi, mistä seurasi yli vuorokauden kestävä alentunut palvelutaso [1]. Katkoksen kriittisessä verkkoyhteydessä voidaan olettaa aiheuttavan haittaa, mutta palvelun suunnittelun pitäisi estää häiriön laajeneminen. Toinen esimerkki on vuonna 2021 esiintynyt häiriö Fastlyn hajautetussa CDN-palvelussa, mikä vaikutti sitä käyttäviin järjestelmiin [2].

EU-alueella noin 45 % yrityksistä osti vuonna 2023 pilvipalveluita, jotka sisältävät infrastruktuuria sekä ohjelmistoja [3]. Pilvipalveluiden laitteistoresurssien päälle rakennetaan uusia hajautettuja web-palveluita, jotka ovat alttiita alustan häiriöille. Amazon Web Servicen S3-tallennuspalvelussa tapahtui vuonna 2017 vika, joka vaikutti sitä käyttäviin järjestelmiin [4]. S3 on suosittu tallennuspalvelu, joka on helppo integroida hajautettuihin web-palveluihin.

On tärkeää, että web-palvelut sietävät riittävällä tasolla vikoja, joita niissä itsessään, palvelualustoilla tai verkkoyhteyksissä voi esiintyä. Jos palvelu on suunniteltu ja toteutettu oikein, lyhytkestoiset häiriöt eivät pääse eskaloitumaan useiden tuntien mittaisiksi tai aiheuttamaan ylimääräistä haittaa asiakkaille. Erityisesti kun hajautetulla web-palvelulla on paljon siitä riippuvia järjestelmiä, pienilläkin vioilla voi olla laajoja seurauksia.

Tämän tutkielman tarkoituksena on kartoittaa, millaisia vikoja hajautetut web-palvelut voivat kohdata ja millä eri tavoilla niiden vikasietoisuutta voidaan parantaa. Tavoitteiden pohjalta voidaan muodostaa seuraavat tutkimuskysymykset:

TK1: Millaisia eri vikoja hajautetut web-palvelut voivat kohdata?

TK2: Miten hajautettujen web-palveluiden viansietokykyä voidaan parantaa?

Tutkielman tiedonhaku on toteutettu pääasiassa IEEE- ja Web of Science -hakukoneita käyttäen. Ensimmäisessä ja toisessa luvussa on hyödynnetty julkaisutietokantojen lisäksi myös kaupallisia hakukoneita aiheen taustaa tukevien uutisten, dokumentaation ja tilastoiden haussa. Kolmannen luvun aineistohaku on tehty ainoastaan IEEE:n ja Web of Sciencen tietokantoja käyttäen. Kaikkien kolmannen luvun lähteiden julkaisijoiden JUFO-luokitus on vähintään 1, ja lähes kaikki luvun lähteet ovat IEEE:n julkaisemia. Haku tehtiin useassa eri erässä tutkielman kirjoittamisen aikana.

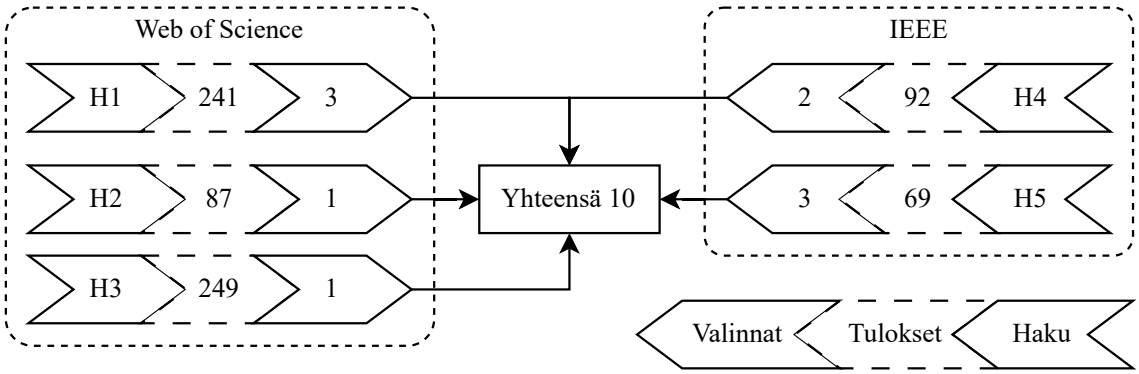
Kuvasta 1.1 on nähtävissä aineistohaussa eri hakulausekkeiden tuottamien tulosten määrä, niistä käytettäväksi valikoituneiden lähteiden määrä, sekä millä hakukoneella haku on tehty. Tuloksia on hyödynnetty pääasiassa kolmannessa luvussa, mutta myös taustaluvussa. Seuraavassa listassa näkyy kuvassa 1.1 esitettyjen hakujen tarkat hakulausekkeet.

1. distributed AND (system* OR service*) AND fault AND tolerance AND web
2. web AND service AND architecture* AND fault* AND (tolerance OR tolerant)

3. distributed AND (system* OR service*) AND (failure OR fault) AND tolerance AND ”load balancing” AND ”high availab*”

4. distributed AND web AND (system* OR service*) AND (failure OR fault) AND tolerance AND client

5. distributed AND web AND service AND load AND failure



Kuva 1.1: Aineistolähteiden tiedonhaku.

Taustaluvussa 2 määritellään, mitä hajautetulla web-palvelulla tarkoitetaan tämän tutkielman yhteydessä, sekä esitellään web-palveluiden arkkitehtuuria korkealla tasolla. Arkkitehtuuria käsittelevässä aliluvussa web-palvelu jaetaan kolmeen eri tasoon, joista jokaiselle määritellään sen oma tehtävä palvelussa. Lisäksi toisessa luvussa kerrotaan, mitä tila ja sen ylläpito tutkielman kontekstissa tarkoittaa, sekä käsitellään myöhemmin esille tulevia protokollia ja teknologioita. Viimeisenä jaetaan web-palveluiden kohtaamat viat kolmeen eri kategoriaan, mitkä toimivat pohjana seuraavan luvun rakenteelle. Kolmas luku sisältää kirjallisuuskatsauksen tutkimuksista, jotka käsittelevät hajautettujen web-palveluiden viansietokykyä parantavia menetelmiä. Luku on jaettu alalukuihin luvussa 2.3 esiteltyjen vikojen kategorioiden mukaisesti. Neljännessä luvussa esitetään yhteenveto työn tuloksista.

2 Hajautetut web-palvelut ja vikatilanteet

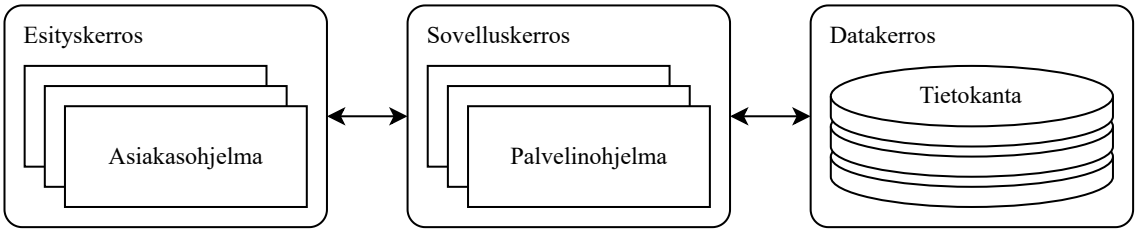
2.1 Arkkitehtuuri

Eräs suosittu tapa jakaa ohjelmia pienempiin osiin on kolmitasoarkkitehtuuri, joka sisältää esitys-, sovellus- ja datakerroksen. Se on sovellettavissa myös web-palveluihin. Esityskerros tarkoittaa asiakkaan käyttämää ohjelmistoa, jonka kautta käytetään taaempien kerroksien toimintoja. Esimerkiksi selaimessa avattava HTML-sivu ja JavaScript-sovellus kuuluu tähän kerrokseen. Sovelluskerroksessa on web-palveluiden kontekstissa palvelimen ohjelmisto, joka sisältää palvelun toiminnan logiikan ja toteuttaa web-palvelun tarjoamat ominaisuudet. Datakerros on pysyvä tallennuspaikka web-palvelun datalle. Esitys- ja datakerros eivät kommunikoi suoraan keskenään, vaan välissä on aina sovelluskerros, joka rajoittaa esityskerroksen pääsyä dataan sekä muita datakerrokseen kohdistuvia operaatioita. [5] Kuvassa 2.1 visualisoidaan eri kerrosten välinen kommunikaatio. Keskitetyssä web-palvelussa olisi kuvasta poiketen yksi palvelinohjelman instanssi ja yksi tietokanta, mutta tässä hajautetussa esimerkissä niitä molempia on useampia. Vaikka kuvassa kerrosten sisällä olevia osia on yhtä monta jokaisessa kerroksessa, todellisuudessa asiakasohjelmia on lähes aina enemmän muiden tasojen osiin verrattuna.

Tässä tutkielmassa web-palvelulla tarkoitetaan verkon yli käytettävää järjestel-

mää, jonka kanssa kommunikoivat laitteet ovat selvästi asiakasohjelman asemassa. Web-palvelun hajauttamisella taas tarkoitetaan sitä, että palvelimen instansseja on enemmän kuin yksi ja instanssit ovat erillisillä tietokoneilla. Tämän tutkielman yhteydessä hajauttamiseksi ei määritellä rinnakkaista laskentaa yhdellä laitteella, eli esimerkiksi HTTP-palvelimen pyyntöjen käsittelyä useammalla prosessorilla samaan aikaan.

Vaikka arkkitehtuurin pystyy jakamaan kolmeen osaan, hajautettu palvelu voi olla sisäiseltä toteutukseltaan ulkopuolista kuvausta monimutkaisempi. Sovellustasolla voi olla useampia kerroksia, erillisiä palveluita, jotka skaalautuvat eri tavoilla, sekä välimuistia, joka voi muistuttaa myös datakerrosta. [6, s. 27–29] Myöskään data- ja esityskerroksen erojen määrittely ei ole täysin yksiselitteistä, koska esimerkiksi JSON Web Tokeneilla (JWT) pystytään kommunikoimaan luotettavasti palvelinten välillä asiakasohjelman kautta. JWT allekirjoitetaan kryptografisesti yhdellä palvelimella, jonka avulla toinen palvelin voi vahvistaa sen aitouden ilman tiedonsiirtoa palvelinten välillä. JWT:in kuljettama data on myös mahdollista salata. [7]



Kuva 2.1: Kolmitasoarkkitehtuurin kerrokset ja komponentit hajautetussa web-palvelussa.

SOA eli Service Oriented Architecture on ohjelmistokehityksen suunnittelutapa, jossa yhden monoliittisen järjestelmän sijasta ohjelmisto on jaettu pienempiin itsenäisiin palveluihin. Mikropalveluarkkitehtuuri muistuttaa SOA-suunnittelua, ja periaate järjestelmän jakamisesta pienempiin osiin on niille yhteistä, mutta mittakaava niiden soveltamisessa on eri. SOA on tarkoitettu palvelun suunnitteluun organisaation tasolla, kun mikropalveluilla voidaan jakaa yksittäinen palvelu vielä

pienempiin osiin. [8] Yhdistämällä SOA ja web-palvelu, voidaan jakaa järjestelmän osat kahteen eri rooliin: asiakkaaseen eli palvelun käyttäjään ja palvelun tarjoajaan. SOA ei rajoitu tiettyyn protokollaan asemien välisessä kommunikaatiossa, ja esimerkiksi HTTP(S), SOAP ja SMTP ovat yhteensopivia. [9] SOA ja mikropalvelut molemmat soveltuvat hyvin hajautettujen web-palveluiden kehitykseen, koska yksittäisiä palvelun osia voidaan päivittää ja skaalata suurelta osin toisistaan riippumatta.

Yksi tapa ajaa web-palvelun osia palvelimilla, esimerkiksi mikropalveluita, on kontitus (engl. containerization). Siinä kaikki ohjelman riippuvuudet pakataan levykuvaan, josta voidaan luoda uusia konttien instansseja. Docker-kontitusjärjestelmässä kontti ei itsessään ole datan pysyvää säilytystä varten, vaan pysyvä data tallennetaan ja asetetaan kontin sisälle saataville jonkin ajurin avulla [10]. Teknologiat siis ohjaavat osaltaan erottamaan data- ja sovelluskerroksen toisistaan, mikä tarkoittaa, että sen ylläpitoa ei tarvitse toteuttaa sovelluskerroksen web-palvelimille yhtä laajasti.

2.2 Tilan ylläpito

Web-palvelun palvelinohjelmien pitää olla suunniteltu toimimaan hajautetussa ympäristössä, eikä kaikkia web-palveluita voida hajauttaa vain ajamalla useampaa kuin yhtä instanssia niistä. Samalla tavoin kuin yksisäikeiseen ajoon suunnitellun ohjelman ajaminen rinnakkain, web-palvelun hajauttaminen useammalle laitteelle voi rikkoa sen sisäisen tilan, tai altistaa sen vioille mitä se ei voisi kohdata keskitetysti ajettuna. Web-palvelun tila tarkoittaa sen sisältämän datan tilannetta jollakin ajan hetkellä, joka määrittyy siitä, mitä web-palvelu on sen vastaanottamien pyyntöjen seurauksena tehnyt. Tietyllä hetkellä olemassa oleva tila voi myös vaikuttaa siihen, miten palvelu pystyy muokkaamaan tilaa tulevaisuudessa sen suunnittelun rajoittamana.

Hajautetuissa web-palveluissa tilaa voidaan pyrkiä ylläpitämään useammalla palvelimella ajantasaisena, tai järjestelmä voidaan suunnitella siten, tietyn tilan osan ylläpitäminen on vain yksittäisen tai muutaman palvelimen vastuulla. Koska kyse on hajautetusta järjestelmästä, yksittäiselläkin palvelimella säilytettävä tila pitää ottaa huomioon koko järjestelmän kontekstissa, esimerkiksi ottamalla suunnittelussa huomioon miten asiakkaat tai muut palvelimet tietävät missä mikäkin tilan osa sijaitsee. Tilaa ja sen muutoksia voidaan siirtää joko web-palvelinten välillä suoraan, niistä ulkoisen tietokannan avulla tai esimerkiksi asiakkaiden kautta käyttämällä kryptografisia menetelmiä tilan eheyden ylläpitämiseksi. Web-palveluiden kohtaamat vikatilat voivat aiheuttaa ongelmia tilan ylläpidossa, esimerkiksi siten, että tilan muutos ei jää pysyvästi palveluun, muutos ei siirry oikein kaikille palvelimille, palvelimille syntyy ristiriitainen käsitys tilasta tai tilan uusimmat muutokset kadotetaan. Erilaiset vikatilat kategorisoidaan tarkemmin luvussa 2.3. Vaihtoehto tilan säilyttämiselle web-palvelun palvelinten instansseissa, on säilyttää sitä palvelimista erillisessä tietokannassa. Silloin voi olla mahdollista siirtää myös vastuu tilan ylläpidosta tietokannalle, mikä saattaa helpottaa hajautetun web-palvelun sovelluskerroksen suunnittelua. [11] Hajautetun web-palvelun käyttämä tietokanta on todennäköisesti myös hyödyllistä ajaa hajautettuna, mutta se jätetään tutkielman ulkopuolelle.

Yleisesti web-palveluiden asiakkaiden ja palvelinten välisessä kommunikaatiossa käytetty HTTP-protokolla on tilaton, mutta sillä voidaan ylläpitää istuntoja. Tällä tarkoitetaan, että protokolla itse ei määritä mitään kytköstä eri HTTP-pyyntöjen välille, mutta ne voidaan web-palvelussa liittää toisiinsa esimerkiksi evästeiden avulla käyttäen istuntotunnusta. [12] Pyynnöt voivat liikkua yhden tai useamman välityspalvelimen tai kuormantasaajan kautta matkalla web-palvelimelle. Jos istunnon määrittävä tieto käsitellään vasta web-palvelimella, eikä ole merkitystä millä instanssilla käsittely tehdään, HTTP-protokollan tilattomuus voi tehdä pyyntöjen

käsittelystä yksinkertaisempaa. Kuitenkin jos istunto on sidottu yhteen tiettyyn palvelimeen, pitää istuntoa käyttävän asiakkaan pyynnöt ohjata aina sille kyseiselle palvelimelle.

Web-palvelun tarjoamien resurssien määrittämiseen voidaan käyttää RESTiä (Representational State Transfer), joka on arkkitehtuuri hajautettujen web-palveluiden tarjoamien resurssien kuvaamiselle [13]. REST edellyttää, että yksittäistä pyyntöä koskeva tila on täysin palautettavissa kyseisen pyynnön sisällöstä, eikä tilan palauttamiseen tarvittavaa tietoa pidetä piilossa palvelimella. Se tarkoittaa, että mikä tahansa hajautetun web-palvelun palvelimista pystyy käsittelemään minkä tahansa pyynnön ilman tilan synkronointia palvelinten välillä. Vaikka REST-termiä käytetään nykyään usein web-palveluiden kehityksessä ja suunnittelussa, lopputulos harvoin todellisuudessa seuraa alkuperäistä RESTin määrittelyä. Vaikka sille ei ole olemassa formaalia määritelmää, sen rajoituksia tai ohjeita ei välttämättä kuitenkaan noudateta [14]. REST ei ole ainoa tapa määritellä web-palvelun tarjoamaa dataa ja toimintoja. WSDL (Web Services Description Language) on kieli, millä voidaan kuvata web-palvelu abstraktilla tasolla. Lisäksi se sisältää tiedon siitä mistä palvelu on saavutettavissa ja miten sitä on tarkoitus käyttää. [15] Toinen vanhempi ja erityisesti hajautettuihin ympäristöihin kehitetty SOAP (Simple Object Access Protocol) on tiedonsiirron protokolla, mitä voidaan käyttää useiden muiden eri protokollien päällä. Se koostuu SOAP-viesteistä, joihin voidaan kohdistaa sääntöjä käsittelyn aikana, jotka siirtävät jäsenneltyä tietoa. [16]

2.3 Vikojen luokittelu

Web-palveluiden vikatilat voivat aiheutua useista syistä ja aiheuttaa erilaisia vaikutuksia sen toimintaan. Eräs yksinkertaisimmista vikatiloista on web-palvelun äkillinen sammuminen, jolloin se ei enää vastaa pyyntöihin. Vaikeampia vikoja ovat sisältövirheet, eli sellaiset, joissa palvelu antaa virheellisiä vastauksia, ja sellaiset

missä se toimii poiketen sen suunnittelusta. Ajalliset virheet ovat myös mahdollisia. Tämäntyyppisissä virheissä palvelu toimii muuten oikein, mutta vastaukset tulevat liian aikaisin tai liian myöhään. [17] Se mihin vikatilan alkuperäisen syyn raja vedetään, ei ole aina selvä, ja useamman eri vian yhdistelmä saattaa aiheuttaa ongelmia, jotka eivät olisi yksittäisen vian seurauksena toteutuneet. Web-palvelua käyttävien tahojen näkökulmasta usein ei ole merkitystä, aiheuttiko palvelun toimimattomuuden sähkökatko, verkon häiriö tai web-palvelimen sisäinen virhe, mikä johti sen sammumiseen. Yhteistä näille vioille on se, että ne ovat helposti tunnistettavissa esimerkiksi palautuneesta virheestä tai vastuksen puuttumisesta kokonaan. Tämän tutkielman aikana tämän tyyppiset viat kategorisoidaan *hallituiksi vioiksi*.

Viat voivat myös olla vaikeasti tunnistettavia, eivätkä välttämättä ole aina päätteltävissä vastauksen puuttumisesta, viiveestä tai sen virheellisestä sisällön muodosta. *Bysanttilaiset viat* (engl. Byzantine faults) ovat luonteeltaan sellaisia, että ilmeistä vikatilaa ei välttämättä ole, vaan järjestelmän osapuolet antavat ristiriitaista tietoa. Viat perustuvat bysanttilaisten kenraalien ongelmaan, jossa kuvitteellisessa hyökkäyksessä eri joukkojen kenraalien pitää päästä yhteisymmärrykseen samaa algoritmia noudattamalla, vaikka kaikki heistä eivät ole luotettavia [18].

Bysanttilaiset viat saattavat vaikuttaa niin epätodennäköisiltä, että järjestelmiä ei suunnitella kestämään niitä. Lisäksi niihin liittyy väärinkäsityksiä, jotka voivat johtaa virheelliseen uskomukseen, että ne eivät ole ongelma kehitetyssä järjestelmässä koska jokin toinen viansietokykyä parantava menetelmä tehoaa myös niihin riittävän tehokkaasti. Todellisuudessa perinteiset viansietotekniikat eivät välttämättä estä niiden leviämistä. Bysanttilainen vika määritellään siten, että se esiintyy eri tavoilla eri havainnoitsijoille. Bysanttilainen häiriö on bysanttilaisen vian aiheuttama järjestelmän kaatuminen. Vain sellaiset järjestelmät, joiden täytyy toimiakseen päästä yhteisymmärrykseen jostain yhteisestä tilasta voivat kaatua täysin bysanttilaisen vian seurauksena. [19]

Web-palvelimen ylikuormittumisella tarkoitetaan sitä, että sen prosessointikapasiteetti ei riitä kaikkien sille kohdistettujen pyyntöjen käsittelemiseen. Näkyvin seuraus ylikuormittumisesta on web-palvelun hitaus, mutta se voi myös johtaa niin viivästyneisiin vastauksiin, että asiakasohjelma olettaa web-palvelimen olevan täysin kaatunut. Ylikuormitus voi johtua joko luonnollisesta käyttäjämäärän tai käytön kasvusta, tai tahallisesta hyökkäyksestä, jonka tarkoituksena on web-palvelun ylikuormitus ja siten palvelunlaadun heikentyminen.

3 Viansietokyvyn parantaminen

3.1 Hallitut viat

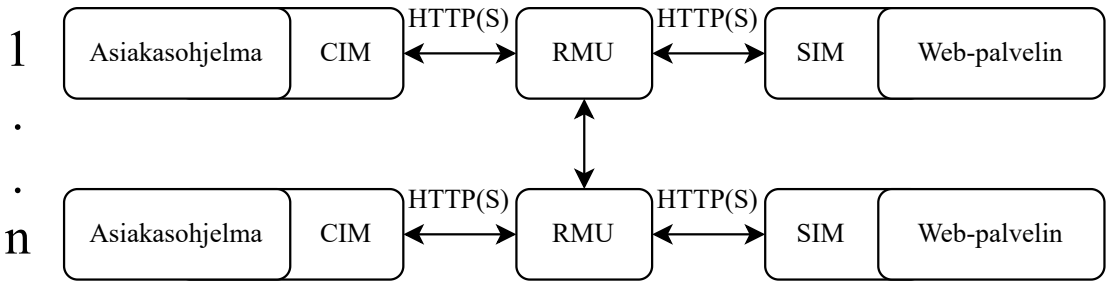
REST-arkkitehtuuriin yhteensopivissa web-palveluissa tilattomuus on usein esille tuotu etu, mutta niiden täytyy silti ylläpitää resurssiensa tilaa palvelimilla sekä asiakkailla vikojen sattuessa. Tila voi kadota vikojen seurauksena, tai se saattaa olla vanhentunut joillain palvelun osapuolista. Palvelun toiminnan jatkaminen voi olla kriittisemmissä järjestelmissä mahdollista ainoastaan, kun oikea tila on saatu palautettua kaikkien osapuolien näkökulmasta. [20]

Anna Kobusińska ja Ching-Hsien Hsu käsittelevät julkaisussaan [20] kehittämäänsä ReServEä, joka on web-palvelun ympäröivä järjestelmä, jolla voidaan vikatilanteissa palauttaa prosessointitila sovelluskerroksen palvelimille sekä esityskerrokselle läpinäkyvällä tavalla. SOA-palvelut toimivat lähettämällä yksittäisiä viestejä asiakkaiden ja palvelun välillä, ja viestit voidaan ReServEllä kaapata siirron aikana. Viesteistä kerätään tiedot lokiin, josta kommunikaation historia nähdään. He ovat myös kehittäneet ReServEstä version, joka on yhteensopiva REST-tyylisten palveluiden kanssa käyttäen GET-, PUT-, POST- ja DELETE-HTTP-metodeja sekä REST-palveluiden resurssien rakennetta. Ainoastaan HTTP-protokolla on tuettu, koska se on laajimmassa käytössä REST-palveluissa. Edellytys ReServEn käytölle on, että palvelun sekä asiakkaiden pitää tuottaa aina sama vastaus yhdestä tietystä pyynnöstä. Kommunikaatio oletetaan luotettavaksi, mikä tässä tarkoittaa sitä,

että pyyntöjä yritetään lähettää uudelleen, jos ne niiden lähetys epäonnistuu. Vaatimusta pyyntöjen perille saapumisen järjestyksestä ei kuitenkaan ole. Käytettävä palautuskeino on määriteltävissä palveluntarjoajan tasolla SOA-kontekstissa, eli jos palvelun kriittisyys ei ole niin tärkeä ominaisuus, se voidaan jättää ReServEn palautumismekanismin ulkopuolelle säästäen resursseja. [20]

Normaalin suorituksen aikana ReServE tallentaa sen kautta kulkevat viestit sen RMU-palvelimille (Recovery Management Unit). Palvelimille kohdistettu liikenne kaapataan palvelun välittäjämoduuleilla (Service Intermediary Module), joita on yksi jokaista erillistä palvelua kohti. RMU sijaitsee kuormantasaajan tavoin esitys- ja sovelluskerroksen välissä. Myös jokaisella asiakasohjelmalla on oma välittäjämoduulinsa (Client Intermediary Module). Jokainen pyyntö, jonka asiakasohjelma lähettää, kulkee sen oman välittäjämoduulin kautta, joka palauttaa vastauksen sen välimuistista tai ohjaa sen eteenpäin RMU:lle, jos sitä ei ole saatavilla. RMU tarkistaa, onko sen pysyvässä lokissa vastausta asiakkaan pyyntöön, ja joko palauttaa sen sieltä tai ohjaa pyynnön palvelun välittäjämoduulin kautta palvelulle. Palvelun vikatilan sattuessa kyseisen palvelun välittäjämoduuli ilmoittaa siitä RMU:lle, joka toistaa pyynnöt lokin perusteella toiselle instanssille. Koska vastaukset kulkevat RMU:n kautta, se voi estää duplikaattien lähettämisen asiakkaalle, jos vioittunut instanssi palauttaa vastauksen vasta sen jälkeen, kun pyyntö on jo ehditty ohjata toiselle instanssille. [20] Kuvassa 3.1 esitellään ReServEn komponenttien tyypit ja niiden yhteydet toisiinsa. Kuvassa jokaisesta komponenttityypistä on yhtä monta instanssia, mutta todellisuudessa asiakasohjelmia on web-palvelimia enemmän, eikä RMU- ja web-palvelinten määrän tarvitse olla sama.

Nicolas Salatge ja Jean-Charles Fabre [21] ovat kehittäneet SOAP-protokollaa käyttäville web-palveluille liitinohjelman, joka ReServEen verrattuna ei vaadi asiakasohjelmaan tai web-palvelimeen liitettäviä komponentteja, vaan se toimii läpinäkyvästi niiden välillä. Liitinohjelman tarkoituksena on välittää jokin haluttua epä-



Kuva 3.1: ReServEn komponentit ja niiden välinen kommunikaatio. Yksinkertaistettu ja suomennettu versio lähteen kuvasta. [20]

vakaampi web-palvelu asiakkaille vikasietoisempana. Se sisältää mekanismit sisään tai ulos liikkuvien pyyntöjen tarkistamiseen, vikatiloista palautumiseen ja kohteena olevien web-palveluiden toiminnan valvomiseen. Palautumiskeinot edellyttävät, että kohteena olevasta web-palvelusta löytyy usea replikoitu instanssi, jotka vastaavat riittävällä tasolla toisiaan. Palvelut kategorisoidaan keskenään identtisiin, joiden WSDL-dokumentit ovat samat mutta instanssit sijaitsevat eri osoitteissa, sekä ekvivalentteihin, joissa dokumentit eivät vastaa toisiaan. Replikaatiostrategia vikatiloista palautumiseen valitaan web-palvelun ominaisuuksien perusteella. Passiivisen replikaation strategioissa vain yksi web-palvelun instanssi käsittelee pyynnön, ja virheen sattuessa pyyntö ohjataan toiselle instanssille. Toinen vaihtoehto ovat aktiiviset replikaatiostrategiat, joissa pyyntö lähetetään useammalle instanssille, joista ensin saatu vastaus palautetaan asiakkaalle. Molempien kategorioiden sisällä voidaan valita sopiva strategia riippuen siitä, onko web-palvelu tilaton vai tilallinen ja miten se mahdollistaa tilan säilyttämisen sekä palauttamisen. Tilallisissa palveluissa pyyntöjen järjestyksellä on merkitystä, koska pyyntöjen käsittely eri järjestyksissä voi johtaa eri tiloihin käsittelyn lopussa. Liitinohjelma ei ota huomioon sitä, missä palvelininstanssilla sen kautta kulkeviin SOAP-viesteihin liittyvä tila sijaitsee web-palvelussa, vaan sen replikointi jätetään web-palvelun vastuulle. [21]

Liitinohjelma tunnistaa web-palvelimen vikatilan vastauksen puuttumisesta, jostain muusta kuin SOAP-tyyppisestä vastauksesta, SOAP-virheviestistä tai käyttä-

jän määrittelemästä ajonaikaisesta tarkistuksesta [21]. ReServEn yhteydessä vikatiloja ei määritellä web-palvelimen täyttä kaatumista tarkemmin [20]. Ne pystyvät siis tunnistamaan ja käsittelemään onnistuneesti vain sellaisia virheitä, jotka web-palvelu itse tunnistaa tai jotka johtavat palvelun kaatumiseen. ReServE vaikuttaa olevan enemmän suunnattu tahoille, jotka työskentelevät web-palvelun kehityksessä, koska se vaatii toimiakseen komponenttien liittämisen jokaiseen asiakasohjelmaan ja web-palvelimeen. Liitinohjelma toimii minkä tahansa edellytykset täyttävän web-palvelun kanssa ilman aktiivista yhteistyötä sitä kehittävän ja ylläpitävän tahon kanssa. Koska yksi ReServEn versio tukeutuu toiminnassaan REST-arkkitehtuurin noudattamiseen web-palvelussa, RESTin tyyliset palvelut, jotka eivät kuitenkaan seuraa sen täyttä määrittelyä saattavat aiheuttaa ongelmia menetelmän käytössä.

3.2 Bysanttilaiset viat

Bysanttilaisia vikoja sietävät algoritmit ja järjestelmät kestävät myös yksinkertaisimpia vikoja, kuten web-palvelimen kaatumisen tai virheen kohtaamisen pyyntöön vastatessa. Lisäksi ne kestävät myös vaikeammin tunnistettavia sattumanvaraisia tai mielivaltaisia vikatiloja, kuten vaarantuneen tietoturvan aiheuttamia hyökkäyksiä ja web-palvelun suunnitellusta poikkeavaa käyttäytymistä [22]. Erityisen kriittisten replikoitujen web-palveluiden täytyy pystyä ylläpitämään oikea tila palvelininstanssien välillä, sekä pystyä takamaan oikeamuotoisten pyyntöjen suoritus [23].

Ympäristöissä missä kommunikaatio on synkronista, f määrän bysanttilaisia vikoja kestävään järjestelmään vaaditaan vain $f + 1$ replikaa. Verkossa, missä web-palvelut toimivat, synkroninen kommunikaatio ei ole tehokasta vaihtelevien ja usein suurienkin viiveiden takia. Takaamalla että pyynnöt saapuvat aina lopulta perille uudelleenlähetyksillä, voidaan asynkronisessa kommunikaatiossa kestää f bysanttilaista vikaa $3f + 1$ replikalla. CLBFT-protokollalla (Castro–Liskov Practical Byzantine Fault Tolerance protocol) voidaan toteuttaa palveluita, joissa osapuolet on jaet-

tu asiakkaisiin ja palvelimiin. Lisäksi se ei edellytä synkronista kommunikaatiota ja tarjoaa hyvän suorituskyvyn, eli se on sopiva käytettäväksi web-palveluissa. Se ei kuitenkaan tue ulkoisten palveluiden käyttämistä, eli se ei ole itsellään sopiva toisistaan riippuvien web-palveluiden kanssa, jos bysanttilaisten vikojen sietokykyä vaaditaan järjestelmältä. [22]

Michael G. Meridethin ym. kehittämä Thema on web-palvelun ohjelmistoon lisättävä välikappale, jonka tarkoitus on lisätä bysanttilaisten vikojen sietokyky monitasoisiin web-palveluihin. Se sijoittuu SOAP-moottorin ja käyttöjärjestelmän väliin, eli SOAP-viestit ohjataan kulkemaan välikappaleen kautta, ennen kuin ne päätyvät käyttöjärjestelmän verkkototeutuksen kautta eteenpäin. Se tarkoittaa, että sen käyttöönotto ei edellytä muutoksia web-palvelun sisäiseen logiikkaan tai toteutukseen. Monitasoinen web-palvelu saattaa siis asiakkaan pyynnön vastaanottamisen jälkeen lähettää sen käsittelyyn liittyen pyynnön toiseen web-palveluun. Thema on yhteensopiva järjestelmissä, joissa kaikki tasot eivät ole bysanttilaisia vikoja kestäviä, eli kaikkiin asiakasohjelmiin tai web-palveluihin ei välttämättä tarvitse lisätä välikappaletta, jos se ei ole toimintavarmuuden kannalta tarpeellista. [22]

Tiettyjen oletusten pitää täyttyä, jotta Themalla voidaan saavuttaa sen takaama vikasietoisuus. Ympäristössä missä Themaa käytetään, määritellään että web-palvelut voivat koostua myös muista pienemmistä web-palveluista, muodostaen yksittäisen laajemman palvelun. Vioittuneille palvelimille ei ole muita oletuksia, kuin että ne eivät pysty ohittamaan kryptografisia salauksia tai allekirjoituksia. Yhteismuotoisten (engl. common-mode) vikojen syntymistä useilla replikoilla pitää jollain menetelmällä ehkäistä, tapauskohtaisesti esimerkiksi n-versioiden päällekkäisyydellä (engl. n-version redundancy), eli ajamalla useita eri versioita samasta ohjelmasta. Lisäksi jokaisella järjestelmään kuuluvalla laitteella on oma julkisen ja yksityisen avaimen pari, joiden levitys hoidetaan Theman ulkopuolella. [22]

Poiketen Themasta joka käyttää taustalla CLBFT-protokollaa, Perpetual on Sa-

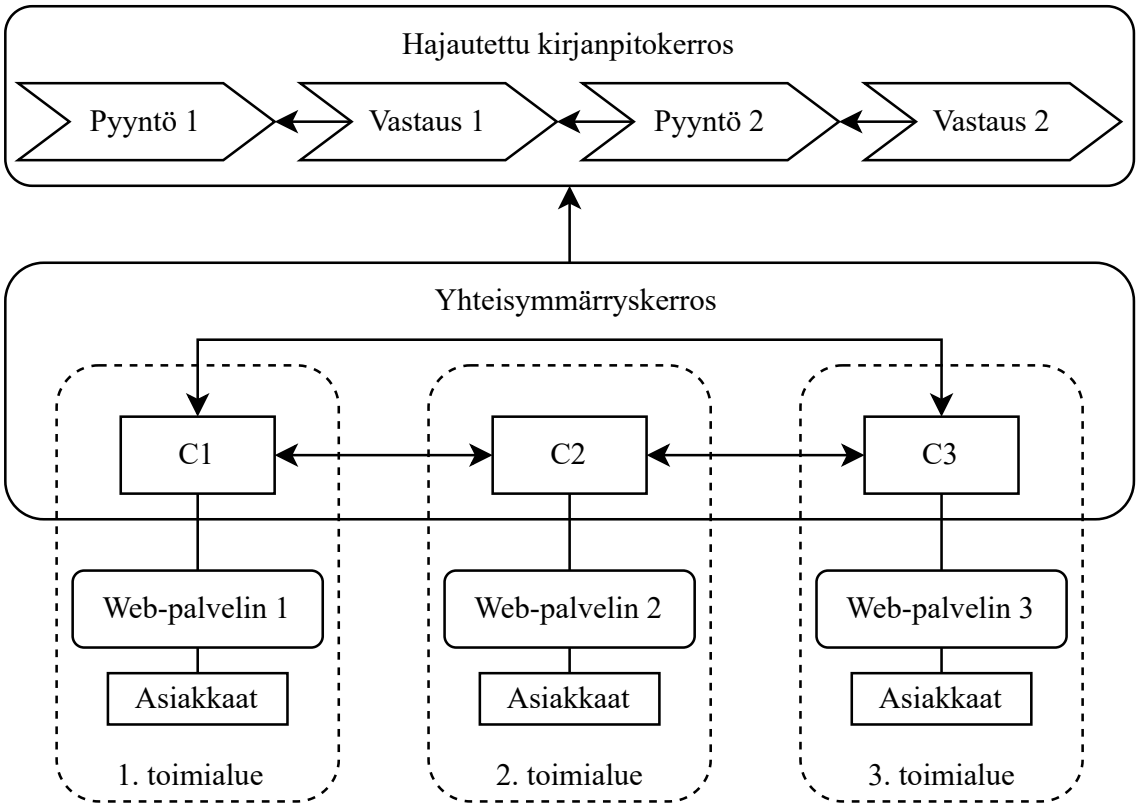
jeeva L. Pallemullen, Haraldur D. Thorvaldssonin ja Kenneth J. Goldmanin julkaisussa [23] esitelty CLBFT:hen pohjautuva protokolla, joka on suunniteltu lisäämään web-palveluihin kyvyn kestää bysanttilaisia vikatiloja. Sen olennaisimmat tavoitteet ovat yhteensopivuus eri määrin replikoitujen web-palveluiden välillä, tuki pitkään suoritettaville säikeille sekä vikatilojen eristäminen. Verrattuna Perpetualia edeltäviin bysanttilaisia vikoja sietäviin protokolliin, se takaa myös pyyntöjä lähettävien palveluiden turvallisuuden ja eloisuuden kohdepalvelun vioittuessa. Lisäksi se tukee asynkronista pyyntöjen käsittelyä, mikä on puuttunut muilta Perpetualin kehityksen aikana olemassa olevilta protokollilta. Itse algoritmin lisäksi sen kehittävät ovat luoneet sitä hyödyntävän välikappaleen, mikä voidaan yhdistää Apache Axis2 web-palvelinta käyttäviin palveluihin, ilman että koko protokollaa pitää toteuttaa itse.

Thema pystyy lähettämään ja vastaanottamaan pyyntöjä osapuolten välillä, kunhan vain toinen niistä on replikoitu, mutta Perperual tukee myös kommunikoinnin kahden replikoidun osapuolen välillä. [23] Theman ja Perpetualin välillä on myös yhtenäisyyksiä. Ne molemmat käyttävät viestien todennuskoodeja (engl. message authentication codes, MAC) niiden aitouden tarkistamiseen, mikä on ainakin algoritmien kehittämisen aikana kolme kertaa nopeampi kuin muiden ratkaisujen käyttämät digitaaliset allekirjoitukset. [22], [23]

Gowri Sankar Ramachandranin ym. suunnittelema DeWS viitekehys on Themaa ja Perpetualia uudempi menetelmä vikasietoisuuden parantamiseen web-palveluissa, joissa bysanttilaisten vikojen syntyminen halutaan minimoida. Se on kehitetty erityisesti sellaisille web-palveluille, jotka muodostuvat useiden eri toimijoiden tarjoamista pienemmistä palveluista. Heidän määritelmänsä mukaan perinteiset keskitettyä arkkitehtuuria noudattavat web-palvelut käyttävät pyyntö–laskenta–vastausmallia asiakasohjelmien pyyntöjä käsitellessä, kun DeWS käsittelee pyynnöt mallilla pyyntö–laskenta–yhteisymmärrys–lokikirjaus–vastaus. Olennainen heikkous monilla aikaisemmilla ratkaisuilla on ollut, että asiakkaiden täytyy luottaa niiden pyyntö-

jä käsittelevään web-palvelimeen sekä palvelinalustaa ylläpitävään tahoon. Lisäksi useiden menetelmien käyttö on edellyttänyt muokkausten tekemistä asiakasohjelmaan, tai jonkin lisäosan yhdistämistä siihen, mikä on mahdollistanut yhteensopivuuden bysanttilaisia vikoja sietävän web-palvelun kanssa. DeWS:n tarkoituksena on myös välttää muutosten tekeminen web-palvelimille ja niiden käyttämille perinteisille viitekehyksille. Menetelmän arkkitehtuurissa web-palvelu jaetaan kuvan 3.2 mukaisesti useaan erilliseen toimialueeseen mitkä kuvaavat eri organisaatioita, jossa jokaisessa on yksi web-palvelin, joka palvelee yhtä tai useampaa asiakasta, sekä yksi yhteisymmärryspalvelin. Jokaisesta toimialueessa on palvelin, joka on osana niin sanottua yhteisymmärryskerrosta, jonka kautta kaikki pyynnöt pitää käsitellä ennen kuin ne vastaanottanut web-palvelin voi vastata asiakkaalle. Käsittelyssä tilan mahdolliset muutokset välitetään muille web-palvelun toimialueille ja kirjataan kirjanpitokerrokseen pysyvästi. Kirjanpitoon voidaan lisätä uusia tilan muutoksia ainoastaan sen loppuun, eikä sinne aikaisemmin kirjattuja voida poistaa tai muokata. Yksittäisen toimialueen web-palvelin saattaa käsitellä sen vastaanottamia pyyntöjä joko tarkoituksella tai vian takia väärin, mikä voi johtua joko bysanttilaisesta virheestä tai normaalista palvelimen kaatumisesta. Siksi ennen kuin ne voidaan kirjata pysyvästi, vähintään kaksi kolmasosaa yhteisymmärryspalvelimista pitää hyväksyä ne. [24]

DeWS:n oikea toiminta edellyttää useiden oletusten täyttymisen. Ensimmäisenä kaikki pyynnöt ja vastaukset tulee tallentaa kirjanpitokerroksen lokiin, myös tapauksissa, joissa yhteisymmärrystä ei saavuteta. Kun oikein toimiva web-palvelin käsittelee oikein toimivalta asiakkaaltaan saadun pyynnön, joka muuttaa web-palvelun tilaa, täytyy muutokset replikoida myös kaikille muille oikein toimiville web-palvelimille yhteisymmärryksen toteutumisen jälkeen. Web- ja yhteisymmärryspalvelinten pyyntöjen käsittelyaikojen täytyy olla rajattu, eli niiden täytyy siirtää pyynnön käsittelyn prosessi seuraavaan vaiheeseen määritellyn aikarajan kuluessa. Viimeisenä,



Kuva 3.2: Lähteeseen perustuva yksinkertaistettu malli DeWS viitekehyksen arkkitehtuurista. [24]

kaikki asiakkaiden ja web-palvelinten viestintä pitää olla salattu, ja asiakkaita täytyy vaatia autentikoitumaan palvelimille. DeWS ei pyri ratkaisemaan luotettavaa kommunikaatiota, vaan se pyrkii estämään web-palvelinten viallisen tai tahallisesti väärän laskennan. [24]

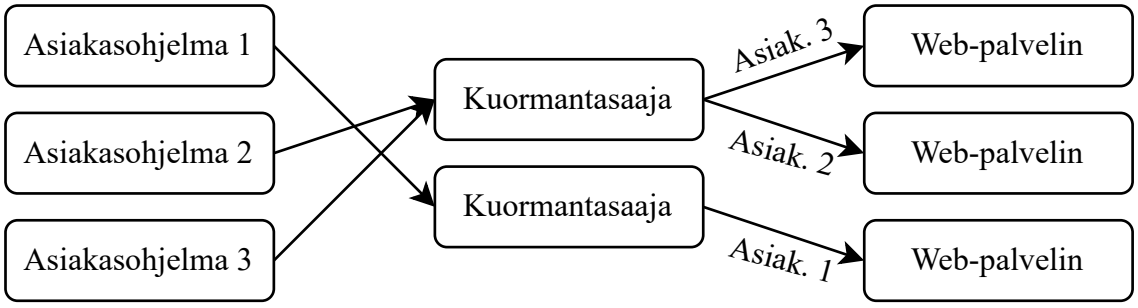
3.3 Web-palvelinten ylikuormitus

Perinteinen tapa jakaa asiakasohjelmien liikenne useammalle eri palvelimelle on DNS-kuormantasaus Round Robin -algoritmilla, jossa yhdellä verkkotunnuksella voi olla useampi IP-osoite. Käytännössä DNS-palvelin vastaa asiakkaalle yhdellä IP-osoitteella kaikkien mahdollisten joukosta, jolloin asiakasohjelman ei tarvitse mallintaa web-palvelua hajautettuna. Mei Lu Chin ym. ovat kehittäneet version Round

Robin-algoritmista, joka pystyy ottamaan huomioon asiakasohjelmille palauttamissaan IP-osoitteissa yksittäisten palvelinten kuorman kapasiteetin. Sen avulla pyyntöjen jakauma palvelimille vastaa paremmin niiden vapaata suorituskykyä. Palvelimet vastaavat itse oman suorituskykynsä arvioinnista, ja raportoivat ne jakauman ajastimeen (engl. distribution scheduler). Jos palvelin ei lähetä tietoa ajoissa sen oletetaan olevan vikatilassa eikä sen IP-osoitetta lähetetä asiakasohjelmille. [25] Vaikka kehitetyn algoritmin pääasiallinen tavoite ei ole vikasietoisuuden parantaminen, sillä voidaan suodattaa pois vioittuneet palvelimet aikaisessa vaiheessa asiakasohjelman ja web-palvelun kanssakäymisessä. Lisäksi suorituskyvyn ja kuorman huomioon ottava Round Robin saattaa laskea vikoja, jotka johtuvat palvelimien ylikuormittumisesta.

Yksittäistä kuormantasaajaa voidaan käyttää kiinteänä päätepisteenä useammalle sen takana olevalle web-palvelimelle, joihin se ohjaa asiakkaiden pyyntöjä. Silloin palvelu voi sietää tilanteen, jossa osa web-palvelimista kaatuu tai eivät enää vastaa riittävän nopeasti pyyntöihin ylikuormituksen takia, mutta on kuitenkin täysin riippuvainen yhdestä kuormantasaajan palvelimesta. Yksi ratkaisu on lisätä kuormantasaajien määrää, jolloin palvelu ei kaadu täysin vaikka yksi tai osa kuormantasaajista vioittuisi tai sen kapasiteetti pyyntöjen välittämiseen web-palvelimille ylittyy. Kuvassa 3.3 on kuvattu web-palvelun rakenne, jossa ensin asiakasohjelma saa yhden kuormantasaajan IP-osoitteen Round Robin -algoritmilla. Kyseinen kuormantasaaja välittää sitten asiakkaan pyynnön sen valitsemalle web-palvelimelle, välttäen niitä, mitkä on jonkin mekanismin avulla todettu vioittuneiksi tai liian hitaiksi. Asiakasohjelmalle on myös mahdollista palauttaa lista kuormantasaajien IP-osoitteita, joka mahdollistaa suuremman viantietokyvyn, mikäli asiakasohjelma pystyy itsenäisesti tunnistamaan vioittuneen kuormantasaajan ja vaihtamaan toiseen. [26]

Yksinkertainen satunnainen jakauma ja Round Robin -algoritmi ovat suunniteltu toimimaan erityisesti perinteisemmissä web-palveluissa, missä sisältö on suurelta



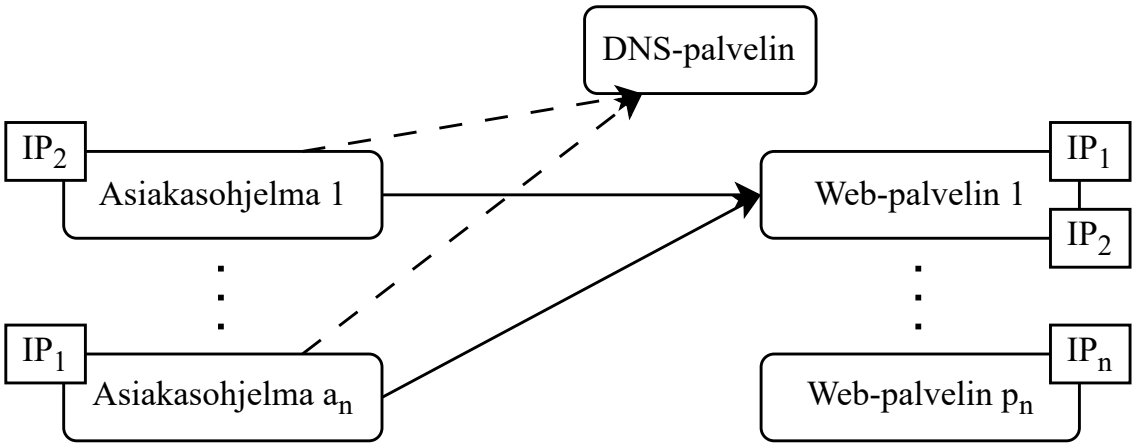
Kuva 3.3: Lähteeseen perustuva yksinkertaistettu kuva Round Robin -algoritmin ja kuormantasauspalvelinten yhdistelmästä. [26]

osin staattista, eivätkä ne ota huomioon missä kontekstissa kukin pyyntö on lähetetty. Lisäksi joissain tapauksissa web-palvelimen kuorman kasvaessa, kilpailu sen resurssien käytöstä saattaa johtaa heikompaan suorituskyvyn hyödyntämiseen. Eli palvelimen kuorman lähestyessä maksimiaan, tietyn suuruinen kasvu siinä voi heikentää suorituskykyä enemmän kuin samansuuruinen kuorman kasvu alemmalla kapasiteetin käytöllä. Zhao Tianhain ehdottaman algoritmin tavoitteena on ottaa asiakkaiden pyyntöjen ohjauksessa huomioon myös web-palvelinten prosessoinnin hyötysuhde. Siinä mahdollisimman optimaalisen jakauman määrittämisessä yritetään välttää objektien kopiointia ohjaamalla samaa objektia koskevat pyynnöt samalle palvelimelle, jossa pyyntöjen kohteena olevaa objektia on jo aikaisemminkin käsitelty. Siten voidaan välttää objektin luominen uudelleen, kun pyyntö ohjataan palvelimelle, missä se on jo valmiiksi olemassa. Objekti ja web-palvelin eivät kuitenkaan ole pysyvästi liitettyinä toisiinsa, vaan tarpeen vaatiessa pyyntöjä voidaan ohjata myös siten että ne johtavat objektien luomiseen uudelleen. Uudelleenohjauksilla on myös oma hintansa, joka riippuu pääasiassa web-palvelinten verkon topologiasta. Kokeissa algoritmi tuotti paremman suorituskyvyn verrattuna satunnaiseen pyyntöjen jakamiseen palvelimille. [27]

Tavallinen kuormantasaaja on selkeästi erillinen komponentti web-palvelun palvelimista. Kaiken palvelun liikenteen täytyy kulkea kuormantasaajan kautta, joka

tekee siitä kriittisen web-palvelun häiriöiden estämisen kannalta. Pahimmillaan se voi muodostaa yhden keskitetyn pisteen asiakkaiden pyyntöjen käsittelyssä, joka vikatilassa voi estää koko palvelun käytön. Lisäksi palvelimilla, joilla kuormantasaajia ajetaan, kuorma täytyy pitää riittävän alhaisena ja niiden välillä sopivan tasaisesti jakautuneena, jotta kuormantasaajan ylikuormitus ei häiritse palvelun käyttöä. Kunihiko Toumura ja Naokazu Nemoto ovat kehittäneet vaihtoehdon tavalliselle kuormantasaukselle, missä liikenteen ei tarvitse liikkua kuormantasaajan läpi vaan se voidaan ohjata suoraan ja hallitusti web-palvelimille IP-reitityksellä. Menetelmässä jokaisella web-palvelimella on suuri määrä IP-osoitteita, ja web-palvelinten kuorman muuttuessa epätasaiseksi niiden kesken, palvelimet voivat ”varastaa” muilta niiden käytössä olevia IP-osoitteita itselleen, ohjaten myös niihin kohdistuvan liikenteen itselleen. Tieto palvelinten kuormasta jaetaan vain kunkin palvelimen naapureiden kanssa, ja siten IP-osoitteet voivat liikkua vain yhdessä ulottuvuudessa. Asiakasohjelmien olisi monimutkaista päättää listasta IP-osoitteita mille palvelimelle pyyntö olisi järkevintä lähettää, ja tieto kaikkien eri palvelinten vapaana olevasta kapasiteetista olisi myös työläs välittää asiakkaille. Siksi DNS palauttaa asiakasohjelmalle vain yhden satunnaisen osoitteen. Kuvasta 3.2 näkee, että verrattuna perinteiseen kuormantasaukseen, tässä menetelmässä hyödynnetään olemassa olevaa IP-reititystä, eikä palvelinten tilasta tarvitse pitää kirjaa keskitetysti, koska ne kommunikoivat IP-osoitteiden siirrosta toistensa kanssa. Kuvassa IP-osoitteet 1 ja 2 ovat päätyneet asiakasohjelmille, ja tilanteessa missä web-palvelimen 1 kuorma kasvaa liikaa, palvelin 2 voi ottaa siltä IP-osoitteen, ja siten osoitteeseen kohdistuneen kuorman itselleen. [28]

Web-palvelun ylikuormitus voi johtua myös tahallisesti aiheutetusta hajautetusta palvelunestohyökkäyksestä (engl. distributed denial of service attack), eli DDoS-hyökkäyksestä. Anmol Kumar ja Mayank Agarwal ovat tutkineet millä tavoilla tämän tyyppisten hyökkäysten haittoja voitaisiin minimoida erityisesti mikropalve-



Kuva 3.4: Lähteestä suomennettu kuva IP-reititykseen perustuvasta kuormantasauksesta. [28]

luista ja konteista koostuvissa web-palveluissa. Menetelmässä puolustusmekanismina hyökkäyksille toimii asiakasohjelmien lähettämien pyyntöjen jaottelu oikeiden käyttäjien sekä hyökkäykseen osallistuvien lähettämiin, jotka ohjataan eri mikropalveluja suorittaville konteille. Mikropalveluiden tarjoamat web-sivut jaetaan korkean ja matalan resurssienkäytön kategorioihin, riippuen siitä miten paljon laskentaa pyyntöihin vastaamiseen pitää käyttää. Tavoitteena tutkimuksessa on ollut luoda aikaisempia samankaltaisia ratkaisuja parempi vaihtoehto, joka olisi riittävän tehokas vastaamaan myös suuremman mittakaavan hyökkäyksiin. Aikaisemmissa ratkaisuissa hyökkäykset johtavat suurempaan määrään tarpeettomasti estettyihin hyvänlaatuisiin pyyntöihin, kun pyyntöjen eteneminen kuormantasaajalta on estetty täysin. Uuden menetelmän tavoitteena siis on ylläpitää selvästi luokiteltavien oikeiden käyttäjien pyyntöjen alhaiset vasteajat, samalla minimoiden virheellisesti hyökkääjien lähettämäksi kategorisoitujen pyyntöjen aiheuttama haitta. Käytännössä luokittelu tehdään pyynnön kohteena olevan web-sivun vastaanottamien pyyntöjen määrän ja sivun aktiivisten yhteyksien määrällä. Kuormantasaajan vastaanottaessa uuden pyynnön se, tarkistaa ensin onko pyyntö kohdistettu alhaisen voi korkean kategorian web-sivulle. Jos aktiivisten yhteyksien määrä on alempi tai yhtä suuri kuin asetettu raja, pyyntö ohjataan mikropalvelun instanssille, joka on määritetty vastaanotta-

maan pyyntöjä vain oikeilta asiakkailta. Jos taas raja ylittyy, pyyntö ohjataan mahdollisten hyökkääjien pyynnöt käsittelevälle instanssille. Konttipohjainen suunnittelu siis mahdollistaa pyyntöjen ohjaamisen yksittäisiin mikropalveluihin. Kokeissa suuren simuloidun hyökkäyksen aikana oikeilla asiakkailla vasteajat pysyivät alhaisina ja täysin epäonnistuneet pyynnöt olivat harvinaisia. Tuloksista selvisi, että rajan asettaminen oikein on tärkeää toivotun suorituskyvyn saavuttamiseksi. Menetelmät tunnistetut heikkoudet olivat kohtalaisen suuri resurssien tarve, sekä heikko kyky torjua hyökkäyksiä, joissa pyyntöjen tiheys oli alhainen. Raja-arvon asettamisella voidaan kuitenkin vaikuttaa siihen, miten pieni hyökkäys tunnistetaan, mutta siinä tehdään aina kompromissi oikeiden käyttäjien tunnistamisessa. [29]

Vain yksi kolmesta esitellystä menetelmästä ottaa huomioon sen, että jos asiakkaiden pyyntöjen ohjauksessa sovelluskerroksen palvelimille ei tiedetä, missä palvelimella pyyntöä koskeva tila sijaitsee, sen täytyy olla kaikilla palvelimilla saatavilla. Zhao Tianhain julkaisussa [27] esitelty menetelmä hyödyntää juuri tilan kopioinnin ja siirtämisen minimointia ylikuormituksen välttämisessä. IP-reititystä käyttävä menetelmä ei pysty huomioimaan tilan sijaintia, koska IP-osoitteiden siirto perustuu täysin viereisten web-palvelinten väliseen kokonaiskuorman eroon [28]. Anmol Kumarin ja Mayank Agarwalin ratkaisu jakaa web-sivut korkean ja matalan resurssienkäytön kategorioihin, mutta siinä ei myöskään huomioida missä kontissa tila sijaitsee tai mihin se olisi tehokkainta siirtää [29].

4 Yhteenveto

Tutkimuskysymys TK1 käsittelee sitä, millaisia vikoja hajautetut web-palvelut voivat kohdata. Tutkielmassa erilaiset viat jaettiin kolmeen eri kategoriaan: hallittuihin vikoihin, bysanttilaisiin vikoihin sekä web-palvelinten ylikuormitukseen. Hallitut viat sisältävät vikatiloja, jotka web-palvelin pystyy itse tunnistamaan ja reagoimaan niihin esimerkiksi sammumalla hallitusti. Ne on siten helppo tunnistaa, eikä niiden yhteydessä oleteta olevan vihamielistä tekijää järjestelmän sisäpuolella. Bysanttilaiset viat ovat vaikeammin tunnistettavia vikoja, koska ne esiintyvät eri havainnoitsijoille eri tavoilla, eikä palvelin välttämättä lopeta toimintaansa kohdatessaan vian, vaan jatkaa virheellistä toimintaa. Hallittuihin vikoihin verrattuna bysanttilaiset viat kattavat suuremman kirjon erilaisia vikatiloja, koska mikä tahansa suunnitellusta poikkeava datan muutos tai laitteistovika voi aiheuttaa sellaisen. Lisäksi bysanttilaisten vikojen määritelmä sisältää kaikki hallitut viat. Viimeisenä luokkana on palvelinten ylikuormitus, joka johtaa niiden hitaampaan toimintaan asiakkaan näkökulmasta, tai voivat jopa täysin lakata käsittelemästä pyyntöjä liiallisen kuorman seurauksena. Ylikuormitus voi aiheutua täysin normaalista palvelun käytöstä, tai tahallisesti aiheutettuna esimerkiksi palvelunestohyökkäyksestä.

Toinen tutkimuskysymys TK2 koski sitä, millä tavoilla hajautettujen web-palveluiden viansietokykyä voidaan parantaa. TK2:en vastauksissa ei käsitelty web-palvelun sisäistä suunnittelua tai logiikkaa viansietokyvyn näkökulmasta, vaan esitellyt menetelmät on tarkoitettu jo olemassa oleviin web-palveluihin, vaikka TK2:n

voidaan tulkita sisältävän myös sisäisen toteutuksen. Siihen vastatessa ainoastaan hallitut viat huomioon ottavia menetelmiä käsiteltiin tutkielmassa kaksi. Ensimmäinen niistä, ReServE, auttaa tilan ylläpidossa REST-arkkitehtuuria noudattavissa web-palveluissa. Sen komponentit pitää integroida tiiviisti asiakasohjelmiin sekä palvelimiin, ja se vaatii niiden väliin pyyntöjä välittävän ja kaappaavan osan. Toinen käsitelty menetelmä oli liitinohjelma, joka toimii SOAP-protokollaa käyttävissä web-palveluissa, ja on suunnattu myös käytettäväksi vain asiakkaan toimesta, eikä vaadi muutoksia web-palveluun.

Tutkielmassa käsiteltiin kolmea eri bysanttilaisten vikojen sietokykyä parantavaa menetelmää. Menetelmistä kaksi, Thema sekä Perpetual olivat vanhempia, joista Thema on SOAPia käyttäville web-palveluille suunniteltu. Perpetual on Themaa sopivampi eri tasoisesti replikoitujen web-palveluiden yhdistämisessä. Kolmas DeWS menetelmä on uudempi, ja arkkitehtuuriltaan sekä tavoitteiltaan hieman erilainen verrattuna kahteen aikaisempaan. DeWS on kehitetty toimimaan useiden eri tahojen web-palvelimien muodostamassa palvelussa, joista jokainen kuluu omaan toimialueeseensa. Kaikki kolme menetelmää toimivat muuttamatta web-palvelun sisäistä logiikkaa ja toimintaa, sekä oikein toimiakseen vaativat tiettyjen oletusten toteutumisen. Koska tutkielmassa ei käsitelty menetelmien algoritmien yksityiskohtaista toimintaa, tulosten perusteella ei pystytä vertailemaan onko menetelmien välillä eroja siinä, miten hyvin ne sietävät bysanttilaisia vikoja tai kattaako niiden suunnittelu kaikki määritelmän mahdolliset vikatilat. Vertailun kohteena olikin pääasiassa, millaisissa web-palveluissa menetelmiä on mahdollista käyttää ja miten niiden käytön edellytykset eroavat toisistaan. Menetelmiä esittelevissä julkaisuissa se, että web-palvelun sisäistä toteutusta ei tarvitse muokata oli usein kerrottu olevan etuna muihin tapoihin verrattuna.

Palvelinten ylikuormituksen estämisen menetelmät keskittyivät pitkälti asiakkailta tulevien pyyntöjen kehittyneempään ohjaamiseen web-palvelimille. Perintei-

sestä Round Robin -algoritmista paranneltu versio, joka ottaa huomioon palvelinten kuorman oli yksi vaihtoehto. Web-palvelinten prosessoinnin hyötysuhdetta voidaan käyttää niiden kuorman kasvun ennakoimiseen. Sama pyyntöjen määrä saattaa kuluttaa enemmän resursseja joissain tapauksissa web-palvelimella, jonka kuorma on ennestään suuri, verrattuna sellaiseen, jonka kapasiteetista on suurempi osa vapaana. Yksi menetelmistä ehdotti kuormantasaajien poistamista kokonaan web-palvelun arkkitehtuurista, ja toteuttamaan kuormantasauksen hyödyntämällä IP-reititystä. Viimeinen keino pyrki estämään mikropalveluihin tahallisesti kohdistettuja palvelunestohyökkäyksiä kategorisoimalla asiakkailta vastaanotetut pyynnöt.

Vaikka osa esitellyistä menetelmistä on vanhempia, niiden kehityksessä käytettyjä periaatteita ja ideoita voi olla mahdollista soveltaa myös moderneihin web-palveluihin. Toisaalta edelleen on käytössä vanhempia web-palveluita, joiden viansietokykyä voidaan haluta parantaa puuttumatta suuremmin niiden sisäiseen toteutukseen ja logiikkaan. Aineistohaussa löydetyt menetelmät eivät ottaneet tarkemmin kantaa siihen, miten tilaa pitäisi ylläpitää web-palvelun tai sen tietokannan sisällä, esimerkiksi millä palvelimilla tilaa säilytetään, mistä tiedetään tilan osien sijainnit tai miten monta kopiota tilasta pitäisi olla. Nämä puutteet voisivat olla hyviä jatkotutkimusideoita. On mahdollista, että koska aineistohaussa tulokset rajattiin hajautettuihin web-palveluihin eikä hajautettuihin järjestelmiin yleisesti, web-palvelinten tai tietokantojen sisäisten tilojen ylläpidosta ei löytynyt aineistoa viansietokyvyn kontekstissa. Eli hajautettujen järjestelmien algoritmiset ja arkkitehtuuriset valinnat koskisivat useimmiten kaikkia hajautettuja järjestelmiä, eivätkä ainoastaan web-palveluita.

Lähdeluettelo

[1] GitHub. ”October 21 post-incident analysis”. (2018), url: https://web.archive.org/web/20240205164855/https://github.blog/2018-10-30-oct21-post-incident-analysis/ (viitattu 05. 02. 2024).

[2] Fastly. ”Summary of June 8 outage”. (2021), url: https://web.archive.org/web/20240205172645/https://www.fastly.com/blog/summary-of-june-8-outage (viitattu 05. 02. 2024).

[3] Eurostat. ”Cloud computing - statistics on the use by enterprises”. (2023), url: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Cloud_computing_-_statistics_on_the_use_by_enterprises (viitattu 05. 02. 2024).

[4] Amazon Web Services. ”October 21 post-incident analysis”. (2017), url: https://web.archive.org/web/20240205171840/https://aws.amazon.com/message/41926/ (viitattu 05. 02. 2024).

[5] IBM. ”What is three-tier architecture?” (2024), url: https://web.archive.org/web/20240217132209/https://www.ibm.com/topics/three-tier-architecture (viitattu 17. 02. 2024).

[6] I. Gorton, *Foundations of Scalable Systems: Designing Distributed Architectures*. O’Reilly Media, Inc., 2022.

[7] Internet Engineering Task Force (IETF). ”JSON Web Token (JWT)”. (2015), url: https://datatracker.ietf.org/doc/html/rfc7519 (viitattu 18. 02. 2024).

[8] Red Hat. ”What is service-oriented architecture (SOA)?” (2020), url: https://web.archive.org/web/20240224200735/https://www.redhat.com/en/topics/cloud-native-apps/what-is-service-oriented-architecture (viitattu 24. 02. 2024).

[9] M. P. Papazoglou ja W.-J. van den Heuvel, ”Service oriented architectures: approaches, technologies and research issues”, *VLDB JOURNAL*, vol. 16, nro 3, s. 389–415, heinäkuu 2007, ISSN: 1066-8888. DOI: 10.1007/s00778-007-0044-3.

[10] Docker Inc. ”Manage data in Docker”. (2024), url: https://web.archive.org/web/20240204030302/https://docs.docker.com/storage/ (viitattu 20. 02. 2024).

[11] X. Song, N. Jeong, P. Hutto, U. Ramachandran ja J. Rehg, ”State management in Web services”, teoksessa *Proceedings. 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, 2004. FTDCS 2004.*, 2004, s. 21–27. DOI: 10.1109/FTDCS.2004.1316589.

[12] MDN contributors. ”An overview of HTTP”. (2023), url: https://web.archive.org/web/20240221132038/https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview (viitattu 21. 02. 2024).

[13] R. T. Fielding, ”Architectural Styles and the Design of Network-based Software Architectures”, tohtorinväitöskirja, University of California, Irvine, 2000. url: https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (viitattu 02. 05. 2024).

[14] A. Archip, C.-M. Amarandei, P.-C. Herghelegiu, C. Mironeanu ja E. şerban, ”RESTful Web Services – A Question of Standards”, teoksessa *2018 22nd In-*

ternational Conference on System Theory, Control and Computing (ICSTCC), 2018, s. 677–682. DOI: 10.1109/ICSTCC.2018.8540763.

[15] W3C. "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language". (2007), url: https://www.w3.org/TR/wsdl20/ (viitattu 08. 04. 2024).

[16] W3C. "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)". (2007), url: https://www.w3.org/TR/soap12/ (viitattu 08. 04. 2024).

[17] Z. Maamar, Q. Z. Sheng, S. Tata, D. Benslimane ja M. Sellami, "Towards an approach to sustain web services high-availability using communities of web services", *INTERNATIONAL JOURNAL OF WEB INFORMATION SYSTEMS*, vol. 5, nro 1, s. 32+, 2009, ISSN: 1744-0084. DOI: 10.1108/17440080910947303.

[18] L. Lamport, R. Shostak ja M. Pease, "The Byzantine Generals Problem", *ACM Transactions on Programming Languages and Systems*, s. 382–401, heinäkuu 1982. url: https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/.

[19] K. Driscoll, B. Hall, M. Paulitsch, P. Zumsteg ja H. Sivencrona, "The real Byzantine Generals", teoksessa *The 23rd Digital Avionics Systems Conference (IEEE Cat. No.04CH37576)*, vol. 2, 2004, s. 6.D.4–61. DOI: 10.1109/DASC.2004.1390734.

[20] A. Kobusińska ja C.-H. Hsu, "Towards increasing reliability of clouds environments with RESTful web services", *Future Generation Computer Systems*, vol. 87, s. 502–513, 2018, ISSN: 0167-739X. DOI: 10.1016/j.future.2017.10.050.

[21] N. Salatge ja J.-C. Fabre, "Fault Tolerance Connectors for Unreliable Web Services", teoksessa *37th Annual IEEE/IFIP International Conference on De-*

pendable Systems and Networks (DSN'07), 2007, s. 51–60. DOI: 10.1109/DSN.2007.48.

[22] M. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou ja P. Narasimhan, "Thema: Byzantine-fault-tolerant middleware for Web-Service applications", teoksessa *24TH IEEE SYMPOSIUM ON RELIABLE DISTRIBUTED SYSTEMS, PROCEEDINGS*, 24th IEEE Symposium on Reliable Distributed Systems (SRDS 2005), Orlando, FL, OCT 26-28, 2005, IEEE Comp Soc; ACM SIGMICRO, 2005, s. 131–140, ISBN: 0-7695-2463-X. DOI: 10.1109/RELDIS.2005.28.

[23] S. L. Pallemulle, H. D. Thorvaldsson ja K. J. Goldman, "Byzantine Fault-Tolerant Web Services for n-Tier and Service Oriented Architectures", teoksessa *2008 The 28th International Conference on Distributed Computing Systems*, 2008, s. 260–268. DOI: 10.1109/ICDCS.2008.94.

[24] G. S. Ramachandran, T. T. L. Tran ja R. Jurdak, "DeWS: Decentralized and Byzantine Fault-tolerant Web Services", teoksessa *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2023, s. 1–9. DOI: 10.1109/ICBC56567.2023.10174949.

[25] M. L. Chin, C. E. Tan ja M. I. Bandan, "Efficient load balancing for bursty demand in web based application services via domain name services", teoksessa *8th Asia-Pacific Symposium on Information and Telecommunication Technologies*, 2010, s. 1–4.

[26] J. E. C. de la Cruz ja I. C. A. R. Goyzueta, "Design of a high availability system with HAProxy and domain name service for web services", teoksessa *2017 IEEE XXIV International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*, 2017, s. 1–4. DOI: 10.1109/INTERCON.2017.8079712.

[27] Z. Tianhai, "Web service request distribution based on object set", teoksessa *IEEE Conference Anthology*, 2013, s. 1–5. DOI: 10.1109/ANTHOLOGY.2013.6784869.

[28] K. Toumura ja N. Nemoto, "A Scalable Server Load Balancing Method Using IP Address Stealing", teoksessa *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*, 2013, s. 599–603. DOI: 10.1109/COMPSACW.2013.81.

[29] A. Kumar ja M. Agarwal, "Preserving Service Availability Under DDoS Attack in Micro-Service Based Cloud Infrastructure", teoksessa *2023 16th International Conference on Security of Information and Networks (SIN)*, 2023, s. 1–8. DOI: 10.1109/SIN60469.2023.10474720.