Haaga-Helia
ammattikorkeakoulu Oy

Vapaan ohjelmiston HTTP-kiihdyttimien vaikutus verkkosivujen suorituskykyyn

Markus Pyhäranta

Tutkimusraportti
Tietojenkäsittelyn ko.
28.2.2019

Tiivistelmä

Haaga-Helia
ammattikorkeakoulu Oy

Tekijä(t)
Markus Pyhäranta

Koulutusohjelma
Tietojenkäsittely

Raportin/Opinnäytetyön nimi
Vapaan ohjelmiston HTTP-kiihdyttimien vaikutus verkkosivujen suorituskykyyn

Sivu- ja liitesivumäärä
19 + 8

Tutkimus toteutettiin kevään 2019 aikana osana Tutkimusprosessit-kurssia. Tavoitteena oli selvittää vapaan ohjelmiston HTTP-kiihdyttimien vaikutusta staattisten ja dynaamisten verkkosivustojen suorituskykyyn. Tutkimuksessa keskityttiin suorituskyvyn kilpailuttamiseen kolmen HTTP-kiihdyttimen osalta, joiden ominaisuudet ja lisensointi vastasivat tutkimuksen vaatimuksia.

Tutkimusraportin tietoperustassa tutustutaan vapaan ohjelmiston kriteereihin, HTTP-kiihdyttimien, välimuistituksen ja erilaisten välityspalvelimien toimintaan, sekä aiheesta aiemmin tehtyjen tutkimusten tuloksiin. Tietoperustan lopussa esitellään lyhyesti tutkimukseen valitut kolme HTTP-kiihdytintä: Varnish, Squid ja Nginx.

Aineisto- ja tutkimusmenetelmät -osiossa kuvataan, miten testausympäristö luotiin ja millä perustein. Testit suoritettiin kokonaan lähiverkon sisällä palvelin- ja client-tietokoneella, joiden kokoonpanojen konfiguraatiot kuvataan raportin liitteissä yksityiskohtaisesti. Esitettyjen tietojen pohjalta tutkimus onkin tarvittaessa toistettavissa lähes identtisin tuloksin. Osion lopussa kuvataan, mitä dataa eri palvelinkokoonpanojen suorituskyvyn mittaamisessa kerättiin, miten se kerättiin ja kuinka sitä käsiteltiin lopullisten tulosten saamiseksi.

Tulosten pohjalta havaittiin, että verkkosivusisältöjen välimuistittamisesta on suhteellista hyötyä sekä staattisten että dynaamisten verkkosivujen kanssa. Kaiken kaikkiaan parhaimmaksi vapaan ohjelmiston HTTP-kiihdyttimeksi suoriutui Nginx. Heikoiten pärjäsi Squid, vaikkakin se oli ylivoimaisesti tehokkain tilanteissa, joissa HTTP-pyyntöjä ei samanaikaistettu. Yleisimmissä käyttötilanteissa, suositukseni kohdistuu kuitenkin Nginx- ja Varnish-kiihdyttimiin dynaamisten verkkosivujen kiihdyttämisessä.

Lopuksi raportissa tarkastellaan tutkimuksen onnistumista sen tulosten ja tutkimuksessa tehtyjen päätösten osalta. Siinä huomioidaan tutkimuksen aikana tehdyt virheet, jotta ne voidaan välttää vastaavissa tutkimuksissa myöhemmin. Lisäksi esitän suositukseni tulosten hyödyntämiselle ja jatkotutkimuksille.

Asiasanat
HTTP-kiihdytin, staattinen, dynaaminen, verkkosivu, suorituskyky, vapaa ohjelmisto

Sisällys

Symboliluettelo ja määritelmät

Sisällönhallintajärjestelmä (CMS) = Tietojärjestelmä verkkosivujen sisällönhallintaan. Sisällönhallintaan kuuluvat mm. uusien tekstien ja mediaelementtien julkaisu.

Staattinen verkkosivu = Palvelimella olemassa oleva tekstidokumentti. Sivuun tehdyt muutokset tehdään manuaalisesti suoraan tekstidokumenttiin.

Dynaaminen verkkosivu = Tekstidokumentti, joka luodaan palvelimella ohjelmallisesti esimerkiksi tietokannan sisältämistä tiedoista. Dokumenttia voidaan päivittää automatisoidusti tai manuaalisesti muistakin lähteistä, kuin muokkaamalla itse dokumenttia.

WWW-palvelin = Tietokone tai ohjelmisto, joka jakaa HTTP-protokollalla tekstidokumentteja ja mediatiedostoja käyttäjille, jotka ottavat palvelimeen yhteyttä asiakasohjelmalla, kuten verkkoselaimella.

TCP/IP OSI-malli = Internetin arkkitehtuurin kuvaamisessa käytetty tietoliikenneverkkojen viitemalli. Se kuvaa tiedonsiirtoprotokollien yhdistelmän seitsemän eri kerroksen pinona.

Välimuisti (cache) = Pieni, nopea muisti, jonka tehtävänä on nopeuttaa palvelintietokoneen toimintaa. Sinne talletetaan sellaista sisältöä, johon todennäköisesti viitataan usein, mutta ei ole pitkäikäistä.

Localhost = Isäntänimi (nimi, jolla tietokone tunnistaa itsensä lähiverkossa.), joka osoittaa siihen paikalliseen tietokoneeseen, jolla siihen referoidaan. Sitä käytetään, kun yritetään tavoittaa palveluja omalta koneelta.

Hosts-tiedosto = Tekstidokumentti, joka sisältää koneiden isäntänimiä, tai domain-nimiä vastaavat IP-osoitteet. Sillä voidaan simuloida lähiverkossa nimipalvelun toimintaa, tai estää tiettyjä domain-nimiä vastaamasta niiden oikeisiin IP-osoitteisiin.

$ = Merkki, joka kuvaa muuttujaa Linuxin Bash-komentoriviympäristössä. Esimerkiksi komennossa ”ab -n $requests http://localhost/”, $request viittaa muuttujaan, jonka arvo voi vaihtua.

VPS = Pilvessä ylläpidetty, vuokrattava virtuaalipalvelin.

HTTPS = Suojattu versio HTTP-protokollasta, jota verkkoselaimet ja WWW-palvelimet käyttävät tiedonsiirtoon.

SSL/TLS = Salausprotokolla, jolla voidaan suojata tietoliikenne IP-verkkojen yli. Yleisin käyttötarkoitus on suojata verkkosivusisältöjen siirtoa HTTPS-protokollalla.

Kuormantasaus = Työmäärän tasaus useammalle palvelimelle, tai verkkolaitteelle. Sillä optimoidaan yksittäisten resurssien käyttöasteita ja palvelun vasteaikoja. Kuormantasaus on tyypillistä verkkosivuilla, joiden kävijämäärät ovat suuria.

1 Johdanto

Tutkimuksessa selvitettiin vapaan ohjelmiston HTTP-kiihdyttimien vaikutusta verkkosivustojen sisällön lataamisen tehokkuuteen. Vertailtavaksi valittiin kolme tutkimuksen kriteereitä vastaavaa ohjelmistoa, joiden nopeuksia testattiin staattisten ja dynaamisten sivujen lataamisessa. Kriteereinä olivat, että ohjelmistojen tuli olla julkaistu vapaan ohjelmiston lisensseillä, sekä niiden piti toimia välimuistittavina käänteisinä välityspalvelimina. Lisäksi yhteensopivuus Linux-pohjaisen Ubuntu-käyttöjärjestelmän ja Apache www-palvelinohjelmiston kanssa oli välttämätöntä. Ohjelman aktiivinen kehitystyö ja päivitettävyys katsottiin myös merkittäviksi eduiksi.

Tutkimusidea heräsi osittain aiemman Järjestelmäprojekti-toteutuksen pohjalta, jossa käänteinen Varnish-välityspalvelin tarjoili staattisen verkkosivun hitaammin, kuin ilman välityspalvelinta. Päättelimme kyseisen kokeilun pohjalta, että etu tulee näkymään lähinnä dynaamisten sivujen lataamisessa. (Pyhäranta 2017).

Iso osa nykyisistä verkkosivustoista onkin rakennettu dynaamisen sisällönhallintajärjestelmän (CMS), kuten Wordpressin päälle, jonka markkinaosuus CMS-järjestelmistä on 60% ja 33% kaikista julkisen internetin sivuista. (W3Techs 2019). Ylläpidän itse kolmea Wordpress pohjaista verkkosivua, joiden latausnopeuksia olisi tarkoitus kehittää ilman tarvetta sitoutua kolmannen osapuolen lisäosiin.

Tutkimus jakautui kolmeen pääkysymykseen, jotka ovat vielä jaettavissa useaan alakysymykseen:

1. Mitä vapaan ohjelmiston HTTP-kiihdyttimiä on saatavilla?
    a. Mitkä ovat kriteerit, jotta ohjelmisto katsotaan vapaaksi ohjelmistoksi?
2. Mihin niiden toiminta perustuu?
    a. Minkälaiset ominaisuudet omaava ohjelmisto sopii parhaiten tutkimuksen kuvailemaan tarkoitukseen (verkkosivujen ylläpito)?
3. Kuinka merkittäviä vaikutuksia kyseisillä ohjelmistoilla on staattisten ja dynaamisten verkkosivujen suorituskykyyn?
    a. Miten ne vaikuttavat staattisten sivujen suorituskykyyn?
    b. Miten ne vaikuttavat dynaamisten sivujen suorituskykyyn?
    c. Mikä kyseisistä ohjelmistoista suoriutui parhaiten kultakin osa-alueelta?
    d. Mitkä ohjelmisto suoriutui parhaiten kaiken kaikkiaan?

Lopullisen tutkimuksen tavoitteena oli löytää mainittuihin kysymyksiin vastaukset ja luoda jatkokysymyksiä myöhemmille tutkimuksille.

2 Tietoperusta

2.1 Vapaa ohjelmisto

Vapaaksi ohjelmistoksi kutsutaan ohjelmistoa, joka kunnioittaa käyttäjien vapautta. Sille on määriteltävissä neljä pääperiaatetta: vapaus käyttää ohjelmistoa haluamallaan tavalla, vapaus tutkia ohjelmiston toimintaa lähdekoodista ja muokata sitä, vapaus levittää ohjelmistoa sekä vapaus levittää ohjelmiston muokattua versiota. (GNU 2018). Tutkimukseen valitut HTTP-kiihdyttimet ovatkin käytettävissä vapaan ohjelmiston lisensseillä, kuten FreeBSD ja GNU GPLv2. (Varnish-cache 2017; Squid-Cache Wiki 2018; Nginx 2019).

2.2 HTTP-protokolla

Internetin verkkosivujen käyttö perustuu Hypertext Transfer Protokollaan (HTTP), joka toimii TCP/IP OSI-mallin sovellustasolla. Alkuperäiset versiot HTTP/0.9 ja 1.0 olivat kuitenkin yksinkertaisia, raakaa dataa siirtäviä protokollia, eivätkä ne ottaneet hyvin huomioon välityspalvelimien ja välimuistin hyödyntämisen vaikutuksia, joita uudempi HTTP/1.1 -versio tukee paremmin. (RFC 2616 1999, 7).

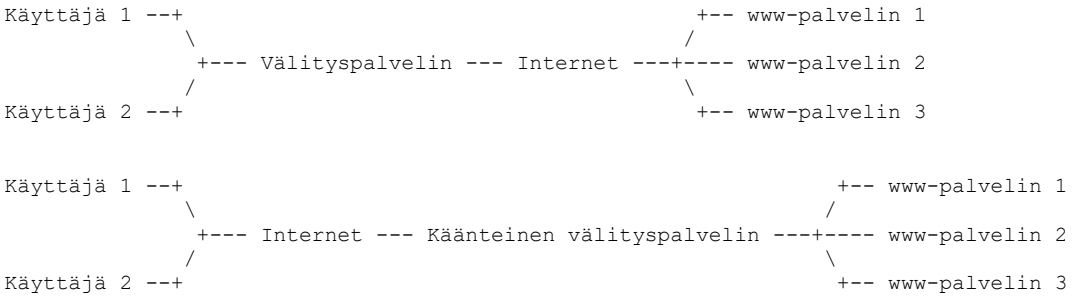
2.3 HTTP-kiihdyttimet

HTTP-kiihdyttimet nopeuttavat verkkosivujen sisällön lataamista toimimalla välimiehenä käyttäjän (client) ja www-palvelimen (web server) välissä. (Nginx 2019). Kiihdytin hyväksyy client-tietokoneen yhteydenoton ja pyrkii vastaamaan sen HTTP-pyyntöön (HTTP GET) lataamalla sivun omasta välimuististaan. Jos sivua ei ole jo valmiiksi välimuistissa, HTTP-kiihdytin tekee saman HTTP-pyynnön www-palvelimelle. Kun kiihdytin saa vastauksen palvelimelta, se tallettaa sen omaan välimuistiinsa ja antaa saman vastauksen, esimerkiksi verkkosivun, client-tietokoneelle. Jatkossa, kun käyttäjät yrittävät ladata samaa sivua, HTTP-pyyntö ei koskaan päädy www-palvelimelle asti, vaan se ladataan suoraan kiihdyttimen välimuistista. (Schroeder, Goddard & Ramamurthy 2000, 44).

HTTP-kiihdyttimet mahdollistavat sekä staattisten että dynaamisten verkkosivujen nopeamman toimittamisen käyttäjille, jolloin www-palvelimen ei itse tarvitse kyetä käsittelemään kaikkia käyttäjäpyyntöjä samanaikaisesti. Koska sivu on jo ladattu kiihdyttimen välimuistiin, se saadaan tarjoiltua sieltä käyttäjälle nopeammin, kuin www-palvelimelta. HTTP-kiihdyttimet voivat myös ennakoida, mitä sisältöä käyttäjät todennäköisesti hakevat, ja säilöä ne valmiiksi välimuistiin. Tällaista sisältöä ovat mm. verkkosivustojen etusivut, eli index.html -dokumentit. (Nginx 2019).

Toinen yleinen tapa kiihdyttää HTTP-liikennettä, on käänteisten välityspalvelimien (reverse proxy) käyttäminen. Erona tavalliseen välityspalvelimeen (proxy tai forward proxy) on se, että kyseessä on palvelin, joka istuu www-palvelimen edessä ja lähettää client-tietokoneiden HTTP-pyynnöt eteenpäin www-palvelimelle, kun taas tavanomaiset välityspalvelimet istuvat client-tietokoneen edessä. Ero on pieni, mutta käänteinen välityspalvelin varmistaa, ettei client koskaan ole suorassa yhteydessä lopulliseen www-palvelimeen. Sen sijaan tavallinen välityspalvelin varmistaa, ettei lopullinen www-palvelin ole ikinä yhteydessä suoraan client-tietokoneeseen. (Cloudflare 2019).

Eri välityspalvelimien toiminta: (Cloudflare 2019).



Kaikki tutkimuksessa testattavat HTTP-kiihdyttimet laitettiin toimimaan käänteisinä välityspalvelimina, jotka säilövät HTTP-pyyntöjen vastaukset välimuistiin. Ratkaisun etuna olivat sekä staattisten että dynaamisten sivujen nopeammat latausajat loppukäyttäjälle. (Nginx 2019). Staattiset sivut ovat nopeampia, kuin dynaamiset. Dynaamisten verkkosivujen sisältö vaihtuu koko ajan, ja siksi niitä ylläpidetään usein sisällönhallintajärjestelmän kautta. Palvelinpään ohjelmointikieli, kuten PHP, lukee sisällön tietokannasta, ja tarjoilee ne sitten www-palvelinohjelmistolle. Prosessissa on huomattavasti enemmän vaiheita, kuin staattisten sivujen lataamisessa, ja lopullinen etusivu koostuu monesti useasta template-tiedostosta, jotka vaikuttavat eri sivuihin. (WSVincent 2018).

Sivut, jotka eivät muutu kovin usein, esimerkiksi etusivujen index.html-tiedostot, kannattaisikin tallentaa välimuistiin staattisena tiedostona. Tällöin www-palvelimen ei tarvitse muodostaa sivua uudelleen dynaamisesta sisällöstä aina, kun uusi käyttäjä pyytää nähdä etusivun. (Copeland & McClain 2019).

Tutkimukseen valittiin kolme vapaan ohjelmiston HTTP-kiihdytintä, joiden lisensointi ja tekniset ominaisuudet vastasivat raportin johdannossa kuvattuja käyttötarkoituksen vaatimuksia. Kyseiset ohjelmistot olivat Varnish, Squid ja Nginx. Muita vastaavia ohjelmistoja olivat HAProxy, Polipo ja Nuster, joita ei kuitenkaan valittu tutkimukseen joko heikon dokumentaation, ylläpidettävyyden tai kehityksen päättymisen vuoksi.

2.3.1 Varnish

Varnish on monikäyttöinen ohjelma, jota voidaan käyttää välimuistitusmoottorina, kuorman tasaajana, www-sovellusten palomuurina, verkon reunan todennuksessa, HTTP-reitityksessä, verkkoresurssien linkityksessä ja DDoS-hyökkäyksien suojana. Se on käänteisenä välityspalvelimena toimiva HTTP-kiihdytin, joka suunniteltiin kiireisille ohjelmointirajapinnoille, raskaille dynaamisille ja korkean kävijämäärän verkkosivuille. Sitä käyttävät monet suuret verkkosivustot, kuten Facebook, Wikipedia ja Twitter. (Varnish Software 2018).

Varnish on julkaistu FreeBSD-lisenssillä (two-clause BSD). (Varnish Cache 2019). Lisenssi mahdollistaa ohjelmiston uudelleenlevittämisen muokattuna tai muokkaamattomana, mikäli uudelleenlevitetyn ohjelmiston lähdekoodi ja dokumentaatio sisältää alkuperäisen ohjelmiston tekijänoikeusmaininnan, ehtoluettelon ja vastuuvapauslausekkeen. (GitHub 2016).

2.3.2 Squid

Squid on HTTP- ja HTTPS-protokollia tukeva välityspalvelin, joka parantaa www-palvelimen vastausaikoja välimuistittamalla usein pyydettyjä sivuja. Se siis toimii Varnishin tavoin HTTP-kiihdyttimenä, kuormantasaajana ja palvelimena sisällönjakeluverkoissa (CDN). Squid optimoi palvelimen ja client-tietokoneen välisen tietovirran, vähentää palvelimen kuormaa ja parantaa sivujen latausaikoja. (Squid-Cache 2019).

Squid on vapaata ohjelmistoa ja se on julkaistu GNU GPLv2 -lisenssillä, mikä mahdollistaa ohjelmiston uudelleenlevittämisen muokattuna tai muokkaamattomana, kunhan sen mukana toimitetaan alkuperäiset maininnat tekijänoikeudesta, vastuuvapauslausekkeesta ja GNU GPLv2-lisensoinnista. (GitHub 2019).

2.3.3 Nginx

Nginx on monikäyttöinen ja skaalattava vapaan ohjelmiston HTTP-palvelin, käänteinen välityspalvelin ja IMAP/POP3 -välityspalvelin. Sen etuina on vähäinen ja ennakoitava muistinkulutus johtuen sen tavasta käsitellä HTTP-pyyntöjä. (Nginx 2019). Nginx on tunnetuin Apachen kaltaisena HTTP-palvelimena, mutta se voidaan konfiguroida clientin ja www-palvelimen välille välimuistittavaksi välityspalvelimeksi. Nginxissä onkin monia sisäänrakennettuja kuormantasaus- ja välimuistitusominaisuuksia. (Ryandel 2018).

Nginx on julkaistu Varnishin tavoin vapaan ohjelmiston FreeBSD-lisenssillä (two-clause BSD). Näin ollen, sen uudelleenlevittäminen on sallittua muokattuna tai muokkaamattomana, mikäli uudelleenlevitetyn ohjelmiston lähdekoodi ja dokumentaatio sisältää alkuperäisen ohjelmiston tekijänoikeusmaininnan, ehtoluettelon ja vastuuvapauslausekkeen. (Nginx 2019).

2.4 Aiemmat tutkimukset

Dynaamisen sisällön muuntamista staattiseksi on tutkittu aiemminkin. Tero Karvinen tutki jo vuonna 2011 dynaamisen sisällön tallentamista välimuistiin staattisina sivuina. Karvinen totesi Apache www-palvelinohjelmiston palvelleen staattisia sivuja 400 kertaa nopeammin, kuin dynaamisia Wordpress-sivuja. (Karvinen 2011). Testausmenetelmät olivat kuitenkin julkaisussa epäselvät, eikä niitä toteutettu täysin puhtaalla palvelinasennuksella. Kyseessä oli myös Karvisen oma välimuistia hyödyntävä ohjelma, jonka teknistä toimintaa tai hyödynnettävyyttä laajemmassa mittakaavassa ei oltu kuvattu. Hän otti kuitenkin testauksessa huomioon mahdolliset pullonkaulat verkkoyhteyksissä, ja toteutti testit puhtaasti lähiverkossa.

Sen sijaan esimerkiksi Irina Vasilievan testit Varnish HTTP-kiihdyttimen nopeuksista käsitellä palvelupyyntöjä tehtiin kokonaan Amazonin AWS-pilvipalvelussa (Vasilieva 2018), jolloin lopulliset tulokset ovat riippuvaisia useista tekijöistä, ja saattavat muuttua jatkuvasti testauksen aikana. Kyseinen testaustapa toimii HTTP-kiihdyttimen integroimiseksi jo olemassa olevaan www-palvelimeen, mutta ei testatessa useiden vastaavien ohjelmistojen riippumatonta suorituskykyä toisiinsa nähden.

Kumpikaan tutkimuksista ei kuitenkaan vertaillut useiden HTTP-kiihdyttimien suorituskykyä toisiinsa erilaisten, oikeiden verkkosivusisältöjen lataamisessa. Wenyuan Jiang mittasi kolmen eri HTTP-kiihdyttimen välimuistin nopeutta yksinkertaisen /helloworld URL:in lataamisessa. Testissä Wenyuan vertasi kolmea tässäkin tutkimuksessa alustavasti mukana olevaa HTTP-kiihdytintä hänen kehittämäänsä Nuster-kiihdyttimeen. (Wenyuan 2018). Tutkimuksessa HTTP-kiihdyttimet ja www-palvelin olivat asennettuna samalle palvelimelle. Testien perusteella, kiihdytinohjelmistot sijoittuivat nopeusjärjestyksessä nopeimmasta hitaimpaan seuraavasti: Nuster, Nginx, Varnish, Squid. (Wenyuan 2018). Tutkimuksen tuloksia kannattaa kuitenkin tarkastella kriittisesti, sillä hän on ohjelman kehittäjä. Nuster ei kuulunut tässä tutkimuksessa testattaviin HTTP-kiihdyttimiin, mutta minua kiinnosti tietää, sijoittuvatko testaamani Varnish, Squid ja Nginx nopeuksien suhteen vastaavalla tavalla.

3 Aineisto ja tutkimusmenetelmät

3.1 Fyysinen testausympäristö ja ohjelmistot

Dynaamisina verkkosivuina käytettiin paikallista kopiota markuspyharanta.com Wordpress-blogista. Staattiset sivut tehtiin MIT-lisensoidun Landing Page -Bootstrap-teeman pohjalta. (Start Bootstrap 2014). Molemmat sivut toimivat testauksessa vakiona, eivätkä ne muuttuneet palvelinohjelmistosta riippumatta. Staattinen sivu oli kooltaan 8929 bittiä ja dynaaminen sivu 51241 bittiä. Jotta Wordpress-pohjainen sivu saatiin toimimaan lähiverkossa oikein, nimipalvelun toimintaa simuloitiin sekä palvelimen että clientin päässä hosts-tiedostolla. Palvelimella isäntänimi markusproto.fi osoitettiin palvelimen localhost-osoitteeseen, kun taas client-tietokoneella markusproto.fi osoitettiin palvelimen sisäverkon IP-osoitteeseen 192.168.1.8.

Palvelintietokoneen virkaa toimitti Lenovo ThinkPad T480 -kannettava (I5-8250U, 16GB RAM, Gigabit ethernet). Vakiona konfiguraatiossa oli LAMP-pino, joka koostui Xubuntu 18.04.1 LTS -käyttöjärjestelmästä, Apache 2.4.38 www-palvelinohjelmistosta, MariaDB 10.3.12 -tietokannasta ja PHP 7.3.1 ohjelmointikielestä. Apache asennettiin kuuntelemaan porttia 8080 ja vaihdettiin mpm_prefork -modulista mpm_worker -moduliin. Modulin konfiguraatio muokattiin mahdollistamaan jopa 1000 samanaikaista yhteydenottoa. Myös PHP optimoitiin suorituskykyisemmäksi PHP-FPM FastCGI prosessimanagerilla.

Muuttujina toimivat lopulliseen testaukseen valitut kolme HTTP-kiihdytinohjelmistoa, jotka olivat täyttäneet niille alustavassa arvioinnissa esiasetetut kriteerit. Valitut HTTP-kiihdyttimet, Varnish 5.2.1, Squid 3.5.27 ja Nginx 1.14.0, asennettiin palvelintietokoneelle kuuntelemaan porttia 80. Ohjelmistot testattiin vuorotellen siten, että vain yhden ohjelmiston palvelut olivat aktiivisesti käynnissä palvelinkoneella. Muut HTTP-kiihdyttimet eivät samanaikaisesti kuluttaneet järjestelmäresursseja, tai vaikuttaneet sisällön välimuistittamiseen. Kunkin ohjelmiston konfiguraatiotiedostot ja asennusvaiheet ovat tämän tutkimusraportin liitteissä 2-15. Sekä Apachea että HTTP-kiihdyttimiä ajettiin kaikilla neljällä prosessoriytimellä, ilman rajoituksia.

Rasitustestauksessa käytettiin tehokasta pöytäkonetta (I7-6700K, 16GB RAM, Gigabit Ethernet). Testaustyökaluna toimi koko tutkimuksen ajan ab-ohjelmaa, joka on suunniteltu Apache www-palvelimen suorituskyvyn arviointiin. (Apache 2019).

Molempia tietokoneita varten tarvittavat ohjelmat asennettiin suoraan raudan ja käyttöjärjestelmän päälle, ilman virtualisointia. Lisäksi koneet olivat sisäverkossa kiinni Zyxel GS-

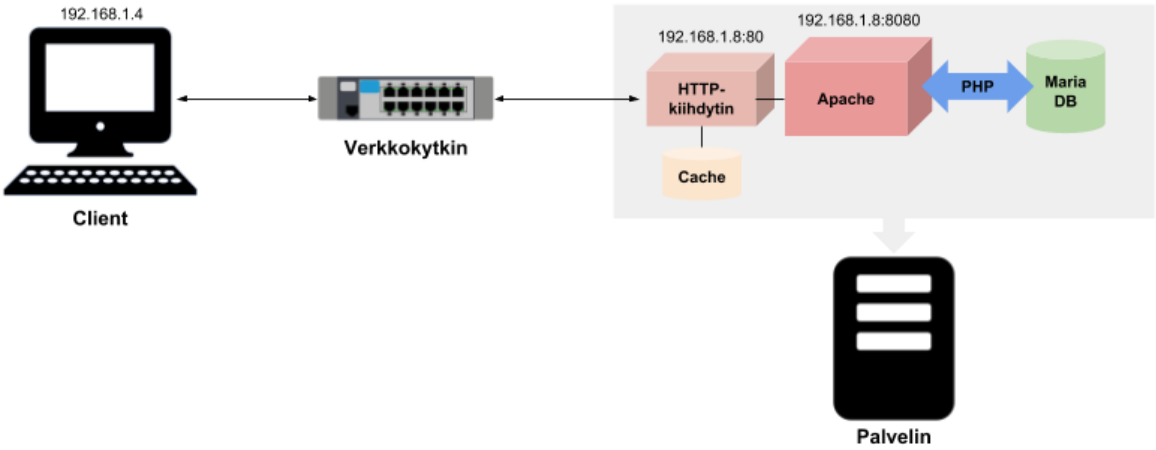
108B v3 -kytkimessä, joka tuki koneiden verkkokorttien tavoin gigabitin verkkoyhteyksiä. (Zyxel 2019). Tutkimus toteutettiinkin paikallisessa lähiverkossa, josta ei ollut pääsyä julkiseen internettiin. Tähän oli neljä syytä:

1. Testauksessa käytetyt rasitustyökalut ovat tulkittavissa palvelunestohyökkäykseksi. Testattaessa palvelinohjelmistojen kykyä tarjoilla verkkosisältöä, sivuja ladattiin samanaikaisesti satoja kertoja. Tämä simuloi realistista tilannetta, jossa sivustolla vierailisi oikeita käyttäjiä. Lailliset ja eettiset ongelmat vältettiin toteuttamalla testausympäristö puhtaasti sisäverkon sisään. Yhteys julkiseen verkkoon oli auki ainoastaan palvelin- ja muiden ohjelmistopakettien asennusvaiheessa.

2. Sisäverkossa vältettiin vaikeammin ennakoitavat pullonkaulat verkkoyhteyksissä. Vaikka testausta suorittavan client-tietokoneen ja pilvessä olevan virtuaalipalvelimen yhteyksissä ei olisikaan ollut vikaa, mikään ei olisi taannut, etteikö yhteysongelmia olisi voinut olla jossain niiden välillä. Testaus julkisessa verkossa olisi toki ollut lähempänä realismia, mutta muuttujia ja mahdollisia rajoittavia tekijöitä olisi myös ollut liikaa, jotta vertailusta olisi saatu kaikkien ohjelmistojen osalta reilu.

3. Kustannustehokkuus. Sisäverkossa testaaminen ei maksanut mitään, jolloin tutkimuksen kustannuksiksi tuli nolla euroa laitteiston ollessa jo valmiina.

4. Lähiverkossa testaaminen ilman julkista verkkoyhteyttä varmisti, ettei kaistaa kulunut muihin internet-yhteydestä riippuviin prosesseihin. Tällöin koko verkon kapasiteetti oli testauksessa käytettyjen työkalujen käytössä.



Kuva 1. Testausverkon arkkitehtuurikuvaus.

3.2 Aineiston keruu

Tutkimuksessa kerättiin ensin pohjatulokset Apache www-palvelimen suorituskyvystä ilman HTTP-kiihdytintä. Ab-ohjelman syntaksi oli seuraavanlainen, jossa parametri **-n** viittaa yhteydenottojen kokonaismäärään ja **-c** samanaikaisten yhteydenottojen määrään:

```
ab –n $requests –c $concurrency http://markusproto.fi/ >
/home/markus/default/static/n$requestsc$concurrency.csv
```

Kussakin testissä suorituskykyä mitattiin ab-ohjelmalla eri parametriarvoin. Ensin testattiin pelkällä -n -parametrilla, jonka jälkeen komentoon lisättiin parametri -c siten, että yhteydenottojen kokonaismäärän ja samanaikaisten yhteydenottojen suhde oli 5:1. (Taulukko 1).

Taulukko 1. Ab-ohjelmalla suoritetut testit.

$requests	$concurrency	Tiedostonimi
10	-	n10.csv
100	-	n100.csv
250	-	n250.csv
500	-	n500.csv
1000	-	n1000.csv
2500	-	n2500.csv
10	2	n10c2.csv
100	20	n100c20.csv
250	50	n250c50.csv
500	100	n500c100.csv
1000	200	n1000c200.csv
2500	500	n2500c500.csv

Tulokset vietiin suoraan CSV-tiedostossa testikäyttäjän kotihakemistossa sijaitseviin static tai dynamic -kansioihin, riippuen kumpaa sivua testattiin.

Näin saatiin oletusarvot Apache www-palvelinohjelmiston suorituskyvystä. Seuraavaksi suorituskykyä mitattiin samoilla testeillä eri HTTP-kiihdyttimien kanssa. Järjestys, jossa kiihdyttimiä testattiin, oli seuraava: Varnish > Squid > Nginx. Lopulta tutkimuksessa oli siis suoritettu testit siten, että kahden sivun suorituskykyä mitattiin neljällä eri palvelinkokoonpanolla ja 12:lla eri komennolla. Täten tutkimuksen aikana muodostui yhteensä 96 CSV-tiedostoa dataa. Dataa käsiteltiin lopulliseen muotoon, jonka jälkeen ne yhdistettiin yhdeksi isoksi Excel-taulukoksi, joka löytyy raportin liitteestä 1.

4 Tulokset

HTTP-kiihdyttimien suorituskykyä testattiin 12 testillä taulukon 1 mukaisesti. Testien tarkat tulokset löytyvät raportin liitteestä 1. Jokaisen kiihdyttimen tuloksia verrattiin prosentuaalisesti oletuskonfiguraation suorituskykyyn. Tulosten esittämisen helpottamiseksi, prosenttiosuuksista laskettiin keskiarvo.

4.1 Staattiset sivut

Tutkimustulokset osoittivat, että staattisten sivujen kohdalla, Squid HTTP-kiihdytin suoriutui keskiarvoisesti parhaiten tilanteissa, joissa HTTP-pyyntöjä ei tehty samanaikaisesti. Squid ei kuitenkaan pärjännyt testeissä, joissa pyyntöjä samanaikaistettiin. Niissä parhaat tulokset jakautuivat Nginxin ja Varnishin kesken. Erot oletuskonfiguraation (ei HTTP-kiihdytintä) eivät kuitenkaan olleet kovin suuria (keskiarvo alle 11%).

4.1.1 Testeihin kulunut kokonaisaika

Tarkastelemalla testeihin kulunutta kokonaisaikaa, kun pyyntöjen samanaikaisuutta ei otettu huomioon, Squid käsitteli HTTP-pyynnöt keskimäärin 26,8% nopeammin verrattuna oletuskonfiguraatioon. Seuraavana oli Nginx 20,72% nopeammin ja viimeisenä Varnish, joka käsitteli pyynnöt 10,61% hitaammin, kuin oletuskonfiguraatio. Samanaikaisuuden kanssa Nginx oli oletuskokoonpanoa 7,39% nopeampi, Varnish 2,85% nopeampi ja Squid 62,44% hitaampi.

4.1.2 Käsiteltyjen pyyntöjen lukumäärä sekunnissa

Tarkastelemalla käsiteltyjen pyyntöjen lukumäärää sekunnissa, ilman pyyntöjen samanaikaisuutta, Squid käsitteli sekunnissa pyyntöjä keskimäärin 37,01% enemmän, kuin oletuskonfiguraatio. Seuraavana oli Nginx 26,43% parannuksella, ja viimeisenä Varnish, jonka käsittelemien pyyntöjen lukumäärä oli 8,87% huonompi verrattuna oletuskonfiguraation. Testeissä, joissa huomioitiin samanaikaisuus, Varnish pärjäsi parhaiten käsittelemällä pyyntöjä 9,87% enemmän. Seuraavana oli Nginx 8,79%:n parannuksella ja viimeisenä Squid, joka käsitteli pyyntöjä sekunnissa 31,96% vähemmän.

4.1.3 Siirtonopeus

Siirtonopeudessa parhaiten suoriutui Squid 38,89% parannuksella oletukseen, kun testeissä ei huomioitu pyyntöjen samanaikaisuutta. Seuraavana oli Nginx 26,7%:lla ja viimeisenä Varnish, jonka siirtonopeudet olivat keskiarvoisesti 8,15% oletuskonfiguraatiota huonommat. Samanaikaisuus huomioiden, Varnish oli paras parantaen tuloksia 10,76%:lla.

Seuraavaksi tuli Nginx 9,03% parannuksella, ja viimeiseksi Squid, jonka siirtonopeudet olivat 31,02% huonommat.

4.1.4 HTTP-pyyntöön käytetyn ajan keskiarvo

Ab:n tulokset sisältävät kaksi arvoa, joilla mitattiin yhteen pyyntöön käytettyä aikaa millisekunteina. Kun testi (-n 1000) ei sisällä samanaikaisuutta, arvot ovat identtiset:

```
Time taken for tests:   1.308 seconds
Time per request:       1.308 [ms] (mean)
Time per request:       1.308 [ms] (mean, across all concurrent
requests)
```

Samanaikaisuuden (-n 1000 -c 200) kanssa niillä on kuitenkin eroa. Esimerkiksi:

```
Time taken for tests:   0.088 seconds
Time per request:       17.539 [ms] (mean)
Time per request:       0.088 [ms] (mean, across all concurrent
requests)
```

Ensimmäinen arvo, Time per request (mean), kuvaakin aikaa, joka keskimäärin kuluu yksittäisen pyynnön käsittelyyn, jos se olisi suoritettu ilman samanaikaisuutta, tai pikemminkin eristetyssä ympäristössä nykyisellä samanaikaisuudella. Käytännössä se kuvaa parhaiten yhden pyynnön käsittelyn viivettä. (ServerFault 2011). Tällä osa-alueella parhaiten pärjäsi Squid 26,73% parannuksella, kun samanaikaisuutta ei huomioitu. Seuraavana oli Nginx 20,37% ja viimeisenä Varnish 10,76% hitaampana. Samanaikaisuus huomioiden, Nginx voitti 5,74% oletusta nopeampana, sitten Varnish 1,41% tuloksella ja viimeisenä Squid, joka oli 63,96% hitaampi.

Toinen arvo, Time per request (mean, across all concurrent requests), taas kuvaa paremmin suorituskykyä. (ServerFault 2011). Esimerkin mukaisesti, yksittäisen pyynnön käsittelyyn meni samanaikaiset pyynnöt sallittaessa 0,088 ms, joka on nopeampi, kuin 1,308 ms. Pyyntöjen samanaikaistaminen on siis nopeampaa, kuin niiden käsitteleminen yksitellen. Ilman samanaikaisuutta, Squid oli nopein 26,73% parannuksella oletuskonfiguraatioon. Seuraavana oli Nginx 20,37% parannuksella ja viimeisenä Varnish 10,76% heikommalla tuloksella. Kuitenkin samanaikaisuuden kanssa Nginx voitti 5,74% paremmalla suorituskyvyllä. Varnish suoriutui 1,54% paremmin ja Squid 63,88% hitaammin.

4.1.5 Pelkistetyt tulokset

Kun staattisen verkkosivun suorituskykyä testattiin ilman samanaikaisia HTTP-pyyntöjä, Squid-kiihdyttimen suorituskyky oli ylivoimaisesti paras. Samanaikaisuuden kanssa, tulokset jakautuivat Nginxin ja Varnishin kesken, joista Nginx oli keskiarvon perusteella tehokkaampi. (Taulukko 2.)

Taulukko 2. Parhaat HTTP-kiihdyttimet staattisten sivujen suorituskyvyn parantamiseen.

Ab:n mittausyksikkö	**Keskiarvoisesti parhaat**	
Tietue:	Ilman samanaikaisuutta:	Samanaikaisuuden kanssa:
Time taken for tests [sec]	Squid	Nginx
Requests per second [#/sec]	Squid	Varnish
Transfer rate [Kbytes/sec]	Squid	Varnish
Time per request [ms] (mean)	Squid	Nginx
Time per request [ms] (mean, across all concurrent)	Squid	Nginx
Keskiarvoisesti paras:	Squid	Nginx

4.2 Dynaamiset sivut

Tutkimustulokset osoittivat, että Nginx tarjoili dynaamiset Wordpress-sivut keskiarvoisesti tehokkaimmin. Kun HTTP-pyyntöjä ei samanaikaistettu, Nginx voitti kaikki suoritetut testit. Samanaikaisuuden kanssa, testien parhaat tulokset jakautuivat tasaisesti Nginxin ja Varnishin välille. Kun tuloksia verrattiin prosentuaalisesti oletuskonfiguraatioon, Nginx oli kuitenkin keskiarvon perusteella parempi.

4.2.1 Testeihin kulunut kokonaisaika

Tarkastelemalla testeihin kulunutta kokonaisaikaa, kun pyyntöjen samanaikaisuutta ei otettu huomioon, Nginx käsitteli HTTP-pyynnöt keskimäärin 96,49% nopeammin verrattuna oletuskonfiguraatioon. Seuraavana oli Squid 95,49% nopeammin ja viimeisenä Varnish 94,58% parannuksella oletuskonfiguraatioon nähden. Samanaikaisuuden kanssa Nginx oli oletuskokoonpanoa 95,71% nopeampi, Varnish 95,41%, ja Squid 95,24%. Tulokset olivat siis kaikki hyvin lähellä toisiaan.

4.2.2 Käsiteltyjen pyyntöjen lukumäärä sekunnissa

Tarkastelemalla käsiteltyjen pyyntöjen lukumäärää sekunnissa, ilman pyyntöjen samanaikaisuutta, Nginx käsitteli pyyntöjä keskimäärin huimat 2751,71% enemmän, kuin oletuskonfiguraatio. Seuraavaksi tuli Squid 2115,06% parannuksella, ja viimeisenä Varnish, jonka käsittelemien pyyntöjen lukumäärä oli 1743,95% suurempi verrattuna oletuskonfiguraation. Testeissä, joissa pyyntöjen samanaikaisuus huomioitiin, Nginx pärjäsi parhaiten käsittelemällä pyyntöjä 2246,22% enemmän. Seuraavana oli Varnish 2174,78%:n parannuksella ja viimeisenä Squid, joka käsitteli sekunnissa pyyntöjä 2083,93% oletuskonfiguraatiota enemmän.

4.2.3 Siirtonopeus

Siirtonopeudessa parhaiten suoriutui Nginx 2523,24%:n parannuksella oletukseen, kun testeissä ei huomioitu pyyntöjen samanaikaisuutta. Seuraavana oli Squid 1943,39%:lla ja viimeisenä Varnish, jonka siirtonopeudet olivat keskiarvoisesti 1599,33% oletuskonfiguraatiota paremmat. Samanaikaisuus huomioiden, Nginx oli paras parantaen tuloksia 2058,37%:lla. Seuraavaksi tuli Varnish 1996,48% parannuksella, ja viimeiseksi Squid, jonka siirtonopeudet olivat 1914,76% oletusta paremmat.

4.2.4 HTTP-pyyntöön käytetyn ajan keskiarvo

Kuten tutkimusraportin osiossa 4.1.4 esitettiin, Ab-työkalu antaa kaksi eri arvoa HTTP-pyyntöön käytetylle ajalle. Tarkasteltaessa ensimmäistä, Time per request (mean), arvoa, kun samanaikaisuutta ei huomioitu, Nginx käsitteli pyynnöt keskiarvoisesti 96,48% oletuskonfiguraatiota nopeammin. Seuraavaksi tuli Squid 95,48%:n parannuksella ja viimeisenä Varnish 94,57% oletusta nopeampana. Samanaikaisuus huomioiden, Nginx paransi oletuskonfiguraation tuloksia 95,67%, Varnish 95,42% ja Squid 95,23%.

Toisen ”Time per request (mean, across all concurrent requests)” -arvon perusteella, kun samanaikaisuutta ei huomioitu, Nginx oli taas nopein 95,67%:n parannuksella oletuskonfiguraatioon. Seuraavana oli Squid 95,48%:n parannuksella ja viimeisenä Varnish 94,57% oletusta paremmalla tuloksella. Samanaikaisuuden kanssa Nginx voitti taas 95,67% nopeammalla tuloksella. Varnish suoriutui 95,42% oletusta paremmin ja Squid 95,23% nopeammin.

4.2.5 Pelkistetyt tulokset

Kun dynaamisen verkkosivun suorituskykyä testattiin ilman samanaikaisia HTTP-pyyntöjä, Nginx-kiihdyttimen suorituskyky oli ylivoimaisesti paras. Nginx johti tuloksia myös samanaikaisuuden kanssa, vaikkakin Varnish voitti joitain yksittäisiä testejä. Loppuen lopuksi, Nginx oli kuitenkin keskiarvon perusteella tehokkain kaikilla osa-alueilla. (Taulukko 3.)

Taulukko 3. Parhaat HTTP-kiihdyttimet dynaamisten sivujen suorituskyvyn parantamiseen.

Ab:n mittausyksikkö	**Keskiarvoisesti parhaat**	
Tietue:	Ilman samanaikaisuutta:	Samanaikaisuuden kanssa:
Time taken for tests [sec]	Nginx	Nginx
Requests per second [#/sec]	Nginx	Nginx
Transfer rate [Kbytes/sec]	Nginx	Nginx
Time per request [ms] (mean)	Nginx	Nginx
Time per request [ms] (mean, across all concurrent)	Nginx	Nginx
Keskiarvoisesti paras:	Nginx	Nginx

5 Johtopäätökset ja suositukset

Tutkimus osoitti, että verkkosivusisältöjen välimuistittamisesta on suhteellista hyötyä sekä staattisten että dynaamisten verkkosivujen kanssa. Kaiken kaikkiaan parhaimmaksi vapaan ohjelmiston HTTP-kiihdyttimeksi suoriutui Nginx.

Jos tarkoituksena on nopeuttaa staattisia verkkosivuja HTTP-kiihdyttimellä, suosittelen Nginxin asentamista välimuistittavaksi, käänteiseksi välityspalvelimeksi. Vaihtoehtoisesti Varnish on myös hyvä ja helpommin konfiguroitava vaihtoehto. Jos taas keksii käyttötarkoituksen, jossa staattista sisältöä ladataan paljon ilman pyyntöjen samanaikaisuutta, Squid on suorituskyvyltään ylivoimaisesti paras.

Tulee kuitenkin huomioida, että testeissä osoitetut staattisten sivujen latausnopeudet eivät vastaa oikeaa tilannetta, jossa käyttäjä vierailee sivustolla. Verkkoselaimet hyödyntävät myös omaa välimuistiaan, ja staattisen sivun lataaminen on aina nopeampaa sieltä, kuin HTTP-kiihdyttimen välimuistista. Siksi koenkin, ettei HTTP-kiihdyttimistä ole tarpeeksi hyötyä staattisten sivujen kohdalla, jotta sellaisen käyttöönotto olisi välttämättä kannattavaa. Realistisessa käyttötarkoituksessa palvelin joutuu käsittelemään useita HTTP-pyyntöjä (kävijöitä) samanaikaisesti. Tällöinkin Nginx käsitteli pyynnöt vain alle 6% oletuskonfiguraatiota nopeammin.

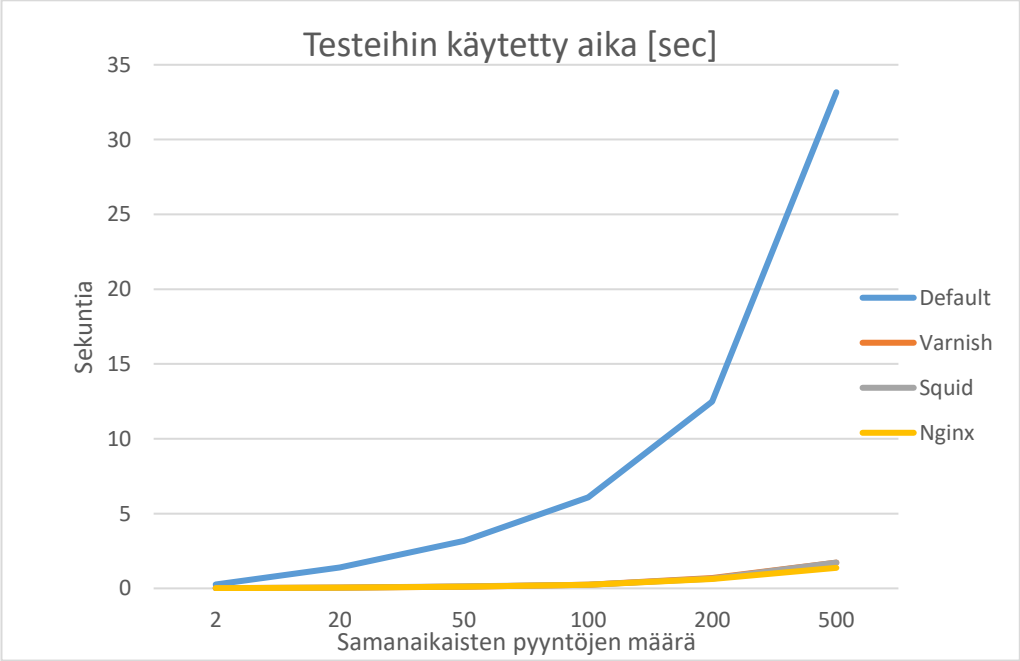
HTTP-kiihdyttimien hyödyt dynaamisten verkkosivujen tarjoamisessa, ovat kuitenkin selvät. Suosittelen Nginxiä myös dynaamisen sisällön suorituskyvyn parantamiseen. Varnish pärjäsi myös ihan hyvin yksittäisissä testeissä, joissa samanaikaisten yhteydenottojen määrät olivat 20-100 välillä. Nginx oli kuitenkin tehokkaampi sitä suuremmilla ja pienemmillä määrillä HTTP-pyyntöjä. Sen tulosten keskiarvot olivat parhaat kaikkien Ab-testien mittausyksikköjen osalta. Raskaimmassakin testissä (2500 pyyntöä, joista 500 suoritettiin samanaikaisesti), Nginx tarjoili dynaamiset Wordpress sivut noin 24 kertaa nopeammin, kuin oletuskonfiguraatio, jossa ei ollut välimuistittavaa HTTP-kiihdytintä.

Tuloksia tarkasteltaessa, tulee kuitenkin huomioida, etteivät tutkimuksen tulokset vastaa realistisessa ympäristössä odotettavia tuloksia. Oikea www-palvelin tavoitettaisiin internetin välityksellä, jolloin viive on huomattavaa verrattuna yhden gigabitin lähiverkkoon. Tutkimuksen pohjalta ei siis voida suoraan sanoa, kuinka paljon kiihdytin nopeuttaa pilvessä ylläpidettyä VPS-palvelinta, mikä ei toisaalta ollutkaan tutkimuksen tavoitteena. Tulokset pikemminkin osoittavat, kuinka tutkimuksessa testatut kolme HTTP-kiihdytintä sijoittuvat suorituskyvyssä toisiinsa nähden.

Oleellinen tekijä testausympäristön konfiguroimisessa oli saada Apache sallimaan vähintään 500 samanaikaista yhteydenottoa. Rajoitus vaikutti nimittäin oleellisesti oletuskonfiguraation suorituskyvyn mittaamiseen, ja täten olisi voinut väärin konfiguroituna vääristää eri HTTP-kiihdytin kokoonpanojen tuloksia suhteessa oletuskonfiguraatioon.

Tutkimuksessa tapahtui kuitenkin valitettava vastaava virhearviointi, sillä MariaDB-tietokannalla on myös rajoitus samanaikaisten yhteydenottojen määrästä. Tätä rajoitusta ei ymmärretty muokata, jolloin samanaikaiset pyynnöt tietokannasta oli rajoitettu oletukseen, eli 151 pyyntöön. (MariaDB 2019). Näin ollen, tutkimustulokset vääristyivät dynaamisten verkkosivujen suorituskyvyn osalta, koska samanaikaisia yhteydenottoja ei pystytty kaikissa testeissä suorittamaan määrätyn muuttujan mukaisesti. Virhe on siis vaikuttanut niihin kahteen testiin, joissa oletuskonfiguraation suorituskykyä ladata dynaamisia sivuja mitattiin yli 151 samanaikaisella tietokantapyynnöllä (-n 1000 -c 200 ja -n 2500 -c 500).

Esimerkiksi, kun tarkastellaan testeihin kulunutta kokonaisaikaa viivadiagrammissa, Varnishin, Squidin ja Nginxin käyrät nousevat paljon lineaarisemmin verrattuna oletuskonfiguraatioon, joka taas lähtee jyrkkään kasvuun joskus 100:n samanaikaisen pyynnön jälkeen.



Kuva 2. Testeihin käytetty aika sekunteina.

Virhe ei vaikuttanut HTTP-kiihdyttimillä suoritettuihin testeihin, sillä niissä sivut ladattiin kiihdyttimen välimuistista, eikä tietokantapyyntöjä tehty. Se kuitenkin vääristi niitä prosentuaalisia arvoja, joilla kiihdyttimien suorituskykyä verrattiin oletuskonfiguraatioon nähden.

Tässä tutkimuksessa käytetyt HTTP-kiihdytin konfiguraatiot eivät kuitenkaan ole suoraan hyödynnettävissä realistisiin ympäristöihin, sillä niissä ei huomioida esimerkiksi välimuistin tyhjentämisen käytännöllisyyttä tosielämän tilanteissa. Tyypillisesti siihen pitäisi olla helppo ratkaisu esimerkiksi sisällönhallintajärjestelmän kautta, eivätkä nämä tulokset ota siihen kantaa millään tavalla. Kiihdyttimet konfiguroitiin suoraan tutkimuksen tarpeisiin, eikä oikeasti verkkosivujen aktiiviseen ylläpitoon. Kunkin kiihdyttimen konfiguraatiot ovat paremmin optimoitavissa yksilöllisiin käyttötarkoituksiin, mutta ne toimivat sellaisenaan hyvänä pohjana muokkauksille.

Aiheen tutkimusalue ei kuitenkaan jää tähän, sillä tutkimuksen pohjalta herää uusia kysymyksiä. Nykyinen trendi on siirtymässä HTTPS-protokollaan myös yksinkertaisten sivujen käytössä. Google Chrome on jo vuoden 2018 heinäkuusta saakka merkannut kaikki tavallista HTTP-protokollaa käyttävät sivustot ”ei turvalliseksi”. (Google Security Blog 2018). Tukevatko tutkimuksessa testatut HTTP-kiihdyttimet myös suojattua protokollaa, ja vaikuttaako se eri tavalla niiden suorituskykyyn? Jos eivät toimi, niin mitä vaaditaan TLS-yhteyden muodostamiseksi? Ainakin Nginx tukee itsessään SSL/TLS-protokollaa, mutta esimerkiksi Varnishin kanssa tulisi ilmeisesti integroida vielä erillinen TLS-proxy, jotta HTTPS-yhteyden kiihdyttäminen olisi mahdollista. Hitch on Varnish Softwaren kehittämä välityspalvelin juuri tähän tarkoitukseen, eli SSL/TLS:n terminointiin. Lisäksi, miten nämä teknologiat skaalautuvat laajempiin infrastruktuureihin, joissa hyödynnetään kuormantasausta (load balancing)?

Näihin kysymyksiin voidaan etsiä vastauksia tutkimustulosten pohjalta. Ensin kuitenkin suorittaisin korjaavat testit tästä tutkimuksesta, jossa MariaDB:n rajoitukset otettaisiin huomioon. Tekisin myös enemmän testejä HTTP-pyyntöjen samanaikaisuudella, enkä antaisi yhtä paljon arvoa testeille, joissa sitä ei ollut. Lisäksi, nostaisin samanaikaisten pyyntöjen lukumääriä. Suorittaisin kunkin testin vähintään kolme kertaa, joista sitten laskisin testin osalta keskiarvon. Siihen ei kuitenkaan lähdetty tässä tutkimuksessa, jotta datamäärät pidettäisiin hallinnassa tutkimuksen aikataulu huomioon ottaen. Samalla olisin myös halunnut seurata palvelinresurssien kuormaa testien aikana, jotta lähtökohdat tuloksille olisivat mahdollisimman yhtenäiset kussakin testissä. Tutkimukseen voisi myös lisätä HTTP-kiihdyttimeksi Apache www-palvelimen, joka on Nginxin tavoin konfiguroitavissa välimuistia hyödyntäväksi, käänteiseksi välityspalvelimeksi.

Korjaavien testien jälkeen, lähtisin ensisijaisesti testaamaan suorituskykyä HTTPS-protokollalla, jonka jälkeen tutkisin ratkaisujen skaalautuvuutta kuormantasausta hyödyntävissä infrastruktuureissa, kuten sisällönjakeluverkoissa.

Lähteet

Apache 2019. Ab – Apache HTTP server benchmarking tool. Luettavissa: https://httpd.apache.org/docs/2.4/programs/ab.html. Luettu 10.2.2019.

Cloudflare 2019. What Is A Reverse Proxy? | Proxy Servers Explained. Luettavissa: https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/. Luettu 31.1.2019.

Copeland, G & McClain, M. 2019,1. Web Caching With Dynamic Content. IBM. Luettavissa: https://jimgray.azurewebsites.net/HPTS99/papers/Copeland_McClain.pdf. Luettu 31.1.2019.

Github 2016. LICENCE. Luettavissa: https://github.com/varnishcache/varnish-cache/blob/master/LICENSE. Luettu 23.2.2019.

Github 2019. READ ME. Luettavissa: https://github.com/squid-cache/squid. Luettu 23.2.2019.

GNU 2018. What is free software? Luettavissa: https://www.gnu.org/philosophy/free-sw.en.html. Luettu 31.1.2019.

Google Security Blog 2018. A secure web is here to stay. Luettavissa: https://security.googleblog.com/2018/02/a-secure-web-is-here-to-stay.html. Luettu 1.2.2019.

Karvinen 2011. Static Advantage – Could Wordpress Be 400 Times Faster? Luettavissa: http://terokarvinen.com/2011/static-advantage-could-wordpress-be-400-times-faster. Luettu 31.1.2019.

MariaDB 2019. Server System Variables. Luettavissa: https://mariadb.com/kb/en/library/server-system-variables/#max_connections. Luettu 27.2.2019.

Netcraft 2018. April 2018 Web Server Survey. Luettavissa: https://news.netcraft.com/archives/2018/04/26/april-2018-web-server-survey.html. Luettu 29.1.2019.

Nginx 2019. LICENCE. Luettavissa: http://nginx.org/LICENSE. Luettu 31.1.2019.

Nginx 2019. Welcome to Nginx Wiki. Luettavissa: https://www.nginx.com/resources/wiki/. Luettu 23.2.2019.

Nginx 2019. What is Web Acceleration? Luettavissa: https://www.nginx.com/resources/glossary/web-acceleration/. Luettu 31.1.2019.

Pyhäranta 2017. Varnish-käänteisproxy ja webpalvelimen suorituskyvyn testaus Apache-Benchillä. Luettavissa: https://markuspyharanta.com/2017/05/14/varnish-kaanteisproxy-ja-apachebenchmark/. Luettu: 28.1.2019.

RFC 2616 1999, 7. Hypertext Transfer Protocol – HTTP/1.1. Luettavissa: https://www.rfc-editor.org/rfc/pdfrfc/rfc2616.txt.pdf. Luettu 31.1.2019.

Ryandel 2018. How to cache your website using NGINX Reverse Proxy and Proxy-Cache in CentOS7. Luettavissa: https://www.ryadel.com/en/nginx-reverse-proxy-cache-centos-7-linux/. Luettu 14.2.2019.

Schroeder, T., Goddard, S. & Ramamurthy, B. 2000, 44. Scalable Web Server Clustering Technologies. DigitalCommons@University of Nebraska – Lincoln. Nebraska. Luettavissa: http://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1083&context=csearticles. Luettu 31.1.2019.

ServerFault 2011. Apache ab: please explain the output. Luettavissa: https://serverfault.com/questions/274252/apache-ab-please-explain-the-output. Luettu 21.2.2019.

Squid-Cache 2019. Squid: Optimising Web Delivery. Luettavissa: http://www.squid-cache.org/. Luettu 23.2.2019.

Squid-Cache Wiki 2018. What's the legal status of Squid? Luettavissa: https://wiki.squid-cache.org/SquidFaq/AboutSquid. Luettu 31.1.2019.

Start Bootstrap 2014. Landing Page – A simple, elegant, and beautifully responsive landing page theme for Bootstrap 4 websites. Luettavissa: https://start-bootstrap.com/themes/landing-page/. Luettu 10.2.2019.

Varnish-Cache 2017. Introduction to Varnish. Luettavissa: https://varnish-cache.org/intro/. Luettu 31.1.2019

Varnish Software 2018. Varnish Web Developer Wiki. Luettavissa: https://www.varnish-software.com/wiki/. Luettu 31.1.2019.

Vasilieva, I 2018. Introduction of an Advanced Caching Layer Leveraging the Varnish Technology Stack and Integrating It to the Existing Web Platform. Barcelona School of Informatics. Barcelona. Luettavissa: https://upcommons.upc.edu/bitstream/handle/2117/121674/135073.pdf?sequence=1&isAllowed=y. Luettu 31.1.2019.

W2Techs 2019. Usage statistics and market share of Wordpress for websites. Luettavissa: https://w3techs.com/technologies/details/cm-wordpress/all/all. Luettu 28.1.2019.

Wenyuan, J 2018. Web cache server performance benchmark: nuster vs nginx vs varnish vs squid. Luettavissa: https://github.com/jiangwenyuan/nuster/wiki/Web-cache-server-performance-benchmark:-nuster-vs-nginx-vs-varnish-vs-squid#results. Luettu 31.1.2019.

Wordpress 2019. Plugins – Browse: Popular. Luettavissa: https://wordpress.org/plugins/browse/popular/page/2/. Luettu 28.1.2019.

WSVincent 2018. Static vs Dynamic Websites: Pros and Cons. Luettavissa: https://wsvincent.com/static-vs-dynamic-websites-pros-and-cons/. Luettu 1.2.2019.

Zyxel 2019. 8-Port Desktop Gigabit Ethernet Switch. Luettavissa: https://www.zyxel.com/fi/fi/products_services/8-Port-Desktop-Gigabit-Ethernet-Switch-GS-108B-v3/. Luettu 18.2.2019.

Liitteet

Liite 1. Tutkimustulokset. (Liite aukeaa kaksoisklikkaamalla. Tiedoston muokkaus täytyy sallia ensin.)

Tutkimustulokset - Markus Pyhäranta.xl

Liite 2. Oletuskonfiguraation asennusvaiheet.

```
sudo apt update
sudo apt upgrade

sudo ufw allow 80/tcp
sudo ufw allow 8080/tcp
sudo ufw enable

sudo apt install apache2

sudo a2enmod userdir
sudo systemctl restart apache2

cd
mkdir public_html
cd public_html
mkdir static
mkdir dynamic

cd /etc/apache2/sites-available/
sudo cp 000-default.conf markus.conf
sudoedit markus.conf

sudo a2dissite 000-default.conf
sudo a2ensite markus.conf
sudo systemctl reload apache2
sudo systemctl restart apache2

sudo apt install php7.2 libapache2-mod-php7.2 php7.2-mysql

sudoedit /etc/apache2/mods-available/php7.2.conf
sudo systemctl restart apache2

sudo apt install php7.2-fpm
sudo a2enmod proxy_fcgi
sudo systemctl restart apache2

cd /etc/php/7.2/fpm/pool.d/
sudo nano markus.conf

sudo systemctl restart php7.2-fpm.service

cd /etc/apache2/sites-available/
sudoedit markus.conf
sudo systemctl restart apache2

sudo apt install mariadb-client mariadb-server
```

```
sudo mysql_secure_installation

sudo mariadb -u root -p
SHOW DATABASES;
CREATE DATABASE wordpress;
GRANT ALL ON wordpress.* TO markuswp@localhost IDENTIFIED BY 'password';
exit

wget https://wordpress.org/latest.tar.gz
tar xzvf latest.tar.gz
rm latest.tar.gz
cd wordpress
cp wp-config-sample.php wp-config.php
nano wp-config.php

find /home/markus/public_html/static/ -type d -exec chmod 755 {} \;
find /home/markus/public_html/static/ -type f -exec chmod 644 {} \;
find /home/markus/public_html/dynamic/wordpress/ -type d -exec chmod 755
{} \;
find /home/markus/public_html/dynamic/wordpress/ -type f -exec chmod 644
{} \;

cd /etc/apache2/mods-available/
sudo a2dismod php7.2
sudo a2dismod mpm_prefork
sudo a2enmod mpm_worker
sudo systemctl restart apache2

sudoedit mpm_worker.conf
sudo systemctl restart apache2
```

Liite 3. Apachen VirtualHost-konfiguraatio /etc/apache2/sites-available/markus.conf.

```
<VirtualHost *:8080>

    ServerAdmin webmaster@localhost
    DocumentRoot /home/markus/public_html/dynamic/wordpress/
    #DocumentRoot /home/markus/public_html/static/

    <Directory /home/markus/public_html/dynamic/wordpress/>
    #<Directory /home/markus/public_html/static/>
        AllowOverride All
        Require all granted
        Options -Indexes
    </Directory>

    <FilesMatch \.php$>
        SetHandler "proxy:unix:/var/run/php/php7.2-fpm-
markus.sock|fcgi://localhost/"
    </FilesMatch>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

Liite 4. Apachen /etc/apache2/ports.conf.

```
#NameVirtualHost *:8080
Listen 8080

<IfModule ssl_module>
    Listen 443
</IfModule>

<IfModule mod_gnutls.c>
    Listen 443
</IfModule>
```

Liite 5. Apache-moduulin /etc/apache2/mods-available/mpm_worker.conf konfiguraatio.

```
<IfModule mpm_worker_module>
    ServerLimit     40
    StartServers              2
    MinSpareThreads      25
    MaxSpareThreads      75
    ThreadLimit          64
    ThreadsPerChild      25
    MaxRequestWorkers     1000
    MaxConnectionsPerChild   0
</IfModule>
```

Liite 6. PHP-FPM konfiguraatio /etc/php/7.2/fpm/pool.d/markus.conf.

```
[markus]
user = markus
group = markus
listen = /var/run/php/php7.2-fpm-markus.sock
listen.owner = www-data
listen.group = www-data
listen.mode = 0660

pm = dynamic
pm.max_children = 5
pm.start_servers = 2
pm.min_spare_servers = 1
pm.max_spare_servers = 3
```

Liite 7. Varnish asennus.

```
sudo apt install varnish

sudo nano /etc/default/varnish
sudo nano /lib/systemd/system/varnish.service
sudo systemctl daemon-reload
systemctl restart varnish
```

```
sudo nano /etc/varnish/default.vcl
sudo systemctl restart varnish

cd /etc/apache2/mods-available/
sudo a2enmod rewrite
sudo systemctl restart apache2

sudo systemctl reload varnish
sudo systemctl restart varnish
```

Liite 8. Varnishin pääkonfiguraatio /etc/default/varnish

```
# Should we start varnishd at boot?  Set to "no" to disable.
START=yes

# Maximum number of open files (for ulimit -n)
NFILES=131072

# Maximum locked memory size (for ulimit -l)
MEMLOCK=82000

DAEMON_OPTS="-a :80 \
             -T localhost:6082 \
             -f /etc/varnish/default.vcl \
             -S /etc/varnish/secret \
             -s malloc,1024m"
```

Liite 9. Varnish-servicen määritykset /lib/system/system/varnish.service.

```
[Unit]
Description=Varnish HTTP accelerator
Documentation=https://www.varnish-cache.org/docs/4.1/ man:varnishd

[Service]
Type=simple
LimitNOFILE=131072
LimitMEMLOCK=82000
ExecStart=/usr/sbin/varnishd -j unix,user=vcache -F -a :80 -T lo-
calhost:6082 -f /etc/varnish/default.vcl -S /etc/varnish/secret -s mal-
loc,1024m
ExecReload=/usr/share/varnish/varnishreload
ProtectSystem=full
ProtectHome=true
PrivateTmp=true
PrivateDevices=true

[Install]
WantedBy=multi-user.target
```

Liite 10. Varnishin VCL-konfiguraatio /etc/varnish/default.vcl.

```
# Default backend definition. Set this to point to your content server.
```

```
backend default {
    .host = "127.0.0.1";
    .port = "8080";
}
sub vcl_recv {
    # Happens before we check if we have this in cache already.
# exclude wordpress url
if (req.url ~ "wp-admin|wp-login") {
    return (pass);
}

#unsetting wordpress cookies
  set req.http.cookie = regsuball(req.http.cookie, "wp-settings-
\d+=[^;]+(; )?", "");
  set req.http.cookie = regsuball(req.http.cookie, "wp-settings-time-
\d+=[^;]+(; )?", "");
  set req.http.cookie = regsuball(req.http.cookie, "word-
press_test_cookie=[^;]+(; )?", "");
  if (req.http.cookie == "") {
    unset req.http.cookie;
  }
}

sub vcl_backend_response {
    # Happens after we have read the response headers from the backend.
    if (beresp.ttl == 120s) {
            set beresp.ttl = 1h;
    }
}

sub vcl_deliver {
    # Happens when we have all the pieces we need, and are about to send
the
    # response to the client.
}
```

Liite 11. Squid asennus.

```
sudo systemctl stop varnish
sudo systemctl stop varnishncsa

sudo apt install squid
sudoedit /etc/squid/squid.conf
sudo systemctl restart squid
```

Liite 12. Squidin konfiguraatio /etc/squid/squid.conf.

```
http_port 80 accel defaultsite=markusproto.fi no-vhost ignore-cc

cache_peer markusproto.fi parent 8080 0 no-query originserver name=myAc-
cel

acl our_sites dstdomain markusproto.fi
http_access allow our_sites
cache_peer_access myAccel allow our_sites
cache_peer_access myAccel deny all
```

```
cache_mem 1024 MB
```

Liite 13. Nginx asennus.

```
sudo systemctl stop squid

sudo apt install nginx
sudoedit /etc/nginx/nginx.conf

sudo systemctl nginx restart
```

Liite 14. Nginxin konfiguraatio /etc/nginx/nginx.conf.

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 768;
}

http {

        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 65;
        types_hash_max_size 2048;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2; # Dropping SSLv3, ref: POO-
DLE
        ssl_prefer_server_ciphers on;

        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;

        gzip on;

    proxy_buffering on;

    proxy_buffers 256 16k;
    proxy_buffer_size 32k;

    proxy_cache_path /var/www/ levels=1:2 keys_zone=edge-cache:10m inac-
tive=600m max_size=1024m;
    proxy_temp_path /var/www/tmp;
    proxy_cache_key $scheme$host$request_uri;

    proxy_cache_lock on;

    proxy_cache_revalidate on;
```

```
    proxy_cache_min_uses 3;

    proxy_cache_use_stale error timeout updating http_500 http_502
http_503 http_504;
    proxy_cache_background_update on;

    proxy_connect_timeout 600;
    proxy_send_timeout 600;
    proxy_read_timeout 600;
    send_timeout 600;

    proxy_cache_valid 200 302   1h;
    proxy_cache_valid 301        1h;
    proxy_cache_valid any        1m;

    proxy_http_version       1.1;

        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;

    add_header  X-Cache-Status $upstream_cache_status;

    server {
        listen 80;
        #root /home/markus/public_html/static/;
        root /home/markus/public_html/dynamic/wordpress/;
        # define nginx variables
        set $do_not_cache 0;
        set $skip_reason "";
        set $bypass 0;

        # security for bypass so localhost can empty cache
        if ($remote_addr ~ "^(127.0.0.1)$") {
            set $bypass $http_secret_header;
        }

        # skip caching WordPress cookies
        if ($http_cookie ~* "comment_author_|word-
press_(?!test_cookie)|wp-postpass_" ) {
            set $do_not_cache 1;
            set $skip_reason Cookie;
        }

        # Don't cache URIs containing the following segments
        if ($request_uri ~* "/wp-admin/|/xmlrpc.php|wp-
.*.php|/feed/|sitemap(_index)?.xml") {
            set $skip_cache 1;
            set $skip_reason URI;
        }



        location / {
        # comment out proxy_redirect if get login redirect loop
            proxy_redirect off;
            proxy_cache edge-cache;
            proxy_cache_revalidate on;
            proxy_ignore_headers  Expires Cache-Control Set-Cookie;

            # CACHE CONFIGURATION result
            proxy_cache_bypass $bypass $do_not_cache;
            proxy_no_cache $do_not_cache;
```

```
            # httpoxy exploit protection
            proxy_set_header Proxy "";
            # add forwarded for header
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            # add the Wordpress hostname to avoid Wordpress canonical re-
direct
            proxy_set_header Host $host;
            # proxy_set_header Host www.edge-hostname.com;
            proxy_set_header Connection "";

            # pass requests to the origin backend
            proxy_pass http://markusproto.fi:8080;
        }

        location ~* .(css|js|png|jpe?g)$ {
            expires 600h;
            add_header Cache-Control "public";
            add_header      X-Cache-Status $upstream_cache_status;

            proxy_redirect off;
                        proxy_cache edge-cache;
                        proxy_cache_revalidate on;
                        proxy_ignore_headers  Expires Cache-Control Set-
Cookie;

                        # CACHE CONFIGURATION result
                        proxy_cache_bypass $bypass $do_not_cache;
                        proxy_no_cache $do_not_cache;

                        # httpoxy exploit protection
                        proxy_set_header Proxy "";
                        # add forwarded for header
                        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
                        # add the Wordpress hostname to avoid Wordpress
canonical redirect
                        proxy_set_header Host $host;
                        # proxy_set_header Host www.edge-hostname.com;
                        proxy_set_header Connection "";

                        # pass requests to the origin backend
                        proxy_pass http://markusproto.fi:8080;
        }
    }

}
```

Liite 15. Apache benchmark, Ab-työkalun asennus.

```
sudo apt install apache2-utils
```