Petri Sandström

VÄLIMUISTIOLION TUOTTAMISEN KESTON HUOMIOIVA ENNALTAHAKU

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2016

TIIVISTELMÄ

Sandström, Petri
Välimuistiolion tuottamisen keston huomioiva ennaltahaku
Jyväskylä: Jyväskylän yliopisto, 2016, 80 s. + liitteet
Tietojenkäsittelytiede, pro gradu -tutkielma
Ohjaaja: Veijalainen, Jari

Web-teknologiat kehittyvät jatkuvasti ja samalla on nähtävissä suuntaus kohti enemmissä määrin dynaamisempaa ja yksilöidympää web-sisältöä. Yksilöidyn sisällön tuottamiseen vaaditaan tyypillisesti käyttäjän tunnistautuminen web-palveluun, josta johtuva istuntotiedon käyttäminen web-sisällön tuottamisessa tekee lähes jokaisesta sivupyynnöstä yksilöidyn ja siten välimuistitekniikoiden käyttö vaikeutuu. Tästä huolimatta välimuistin osumatarkkuutta on mahdollista tehostaa ennaltageneroimalla osumatarkkuuteen positiivisesti vaikuttavia olioita valmiiksi välimuistiin. Ennaltageneroinnin ja välimuistin yhdistelmä nostaa osumatarkkuutta, jolla pyynnöt osuvat välimuistiin ja siten vähentää sisällön generoinnin viivettä. Tässä tutkimuksessa keskityttiin tekniikoihin, joiden avulla on mahdollista tehostaa web-sisällön tarjoamista käyttäjälle. Tavoitteena oli löytää hyvä malli välimuistin ja ennaltageneroinnin yhdistelmälle, jotta käyttäjän kokemaa viivettä saataisiin vähennettyä ja pyyntöjen välisiä viiveeroja tasoitettua. Tutkimuksessa esiteltiin olemassa olevien ajatusmallien vertailukohdaksi uusi kuormitusta tavoitteleva ahne kustannusfunktio. Tämä maksimaaliseen taustajärjestelmän kuormitukseen tähtäävä kustannusfunktio suunniteltiin valitsemaan välimuistiolion tuottamisen kestoltaan pitkäkestoisimmat ja samaan aikaan eniten välimuistin osumatarkkuutta kasvattavat välimuistioliot. Tutkimuksen konstruktio-osuudessa mallinnettiin ennakoivan välimuistiratkaisun malli ja koostettiin lista keskeisistä vaatimuksista aiempien tutkimusten haasteisiin vastaamiseksi. Kustannusfunktioita vertailevissa simulaatioissa havaittiin, ettei uusi kuormitusta tavoitteleva ahne kustannusfunktio suoriutunut riittävällä tasolla. Tämän lisäksi simulaatiotulokset osoittivat kuormitusta tavoittelevasta ajatusmallista johtuvan sietämättömän suuren kaistankulutuksen. Olemassa olevat ennaltageneroinnin ajatusmallit todettiin kuitenkin ennakoivan välimuistiratkaisun malliin soveltuviksi ja suosittelemisen arvoisiksi kustannusfunktioiksi. Tämän tutkimuksen tuloksia voidaan hyödyntää ennakoivan välimuistiratkaisun toteutuksen lähtökohtana.

Asiasanat: välimuisti, välimuistiolio, ennaltagenerointi, web-sovellus

ABSTRACT

Sandström, Petri
Object Generation Delay Aware Prefetch Caching
Jyväskylä: University of Jyväskylä, 2016, 80 p. + appendices
Computer Science, Master's Thesis
Supervisor: Veijalainen, Jari

Web technologies are constantly evolving, and at the same time a tendency can be seen in page contents being more dynamic and personalized than ever before. With personalized pages comes the need to distinguish content based on session object at the Web application server. This leads to non-overlapping requests and thus complicates the use of Web caching. Regardless of the impact of personalized content, the overall hit ratio of a Web cache can be improved by prefetching some of objects in to cache, based on their contribution towards the cache hit ratio. Hence the combination of Web caching and prefetching results in improvement of cache hit ratio, leading to latency reduction. The objective of this study was to investigate how to reduce the generation delay of Web content on Web application servers. The goal was to find a good model for combination of Web prefetching and caching, in order to decrease the user perceived latency and to achieve more homogeneous delay times between requests. A new stress greedy cost function was introduced to be compared with the existing prefetching paradigms. Aiming at maximum stress towards the backend servers the cost function was designed to select the objects with longest content generation delays and objects that at the same time had the greatest hit rate increase factors. In the construction part of this thesis a cache prefetching model was designed and a list of essential requirements was combined to overcome common obstacles encountered in earlier studies. Comparative simulations between different cost functions resulted in performance deficiency with the new stress greedy cost function. Also an intolerable rate of bandwidth consumption was observed to accompany the stress greedy paradigm. However, the existing paradigms were considered as usable cost functions and were suggested be used in the designed cache prefetching model. The results of this study can be used as a baseline when implementing a cache prefetching mechanism in a Web application.

Keywords: cache, cache object, prefetching, Web application

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Ravi, Yu ja Shi (2009) kertovat nopean vasteajan olevan tärkein menestystekijä web-sovelluksessa. Web-palvelun nopean reagoinnin voidaan nähdä olevan erityisen tärkeää verkkokaupoissa. Käyttäjät ovat jatkuvasti valistuneempia ja kärsimättömämpiä, jolloin web-palvelun hitaus saattaa johtaa ostotapahtumien vähentymiseen tai jopa verkkokaupan hylkäämiseen. Tällaiset negatiiviset seuraukset ovat johdettavissa suoraan merkittäviksi taloudellisiksi tappioiksi. (Chiang, Goes & Zhang, 2006.)

Mehrotra, Nagpal ja Bhatia (2010) pitävät web-sovellusten haasteina suorituskykyä ja skaalautuvuutta, sillä web-sivujen monimutkaisuudella on taipumus kasvaa Internetin käytön lisääntyessä. He kuvailevat web-sivujen luonteen muuttuvan enemmissä määrin kohti dynaamista sisältöä, jotta käyttäjälle saataisiin interaktiivinen ja yksilöllinen käyttäjäkokemus.

Jatkuvat web-teknologioihin liittyvät innovaatiot ovat muuttaneet alkuperäisen dokumenttikeskeisen verkon (engl. document Web) sovelluskeskeiseksi (engl. application Web). Viimeaikoina muutoksen suunta on ollut kohti palvelukeskeistä verkkoa (engl. service Web). Dynaamisen sisällön tarjoaminen vaatii palvelimelta pyyntökohtaisen sisällön generoimista jopa käyttäjäkohtaisesti, ennen kuin vastaus voidaan toimittaa takaisin sivupyynnön suorittaneelle käyttäjälle. Tästä seuraa verkko- sekä palvelinkuormitusta ja lopputuloksena ovat korkeat vasteajat. (Ravi ym., 2009.)

Ravi ym. (2009) toteavat esitettyjen ongelmien vaatineen tiedeyhteisöltä ja teollisuudelta suuria ponnistuksia, jotta dynaamisen sisällön generointia ja jakelua on saatu tehostettua. Ongelmiin on onnistuttu vastaamaan, sillä käyttäjän kokemaa viivettä on saatu laskettua ja web-sovellusten suorituskykyä nostettua. Elintärkeässä teknologisessa roolissa näiden ongelmien ratkaisemisessa ovat olleet erilaiset web-välimuistit, joista saatavien hyötyjen tehostamiseen pyritään tässä tutkielmassa löytämään uusia ratkaisuita.

Tietokonelaitteistojen sisäinen muistihierarkia on käsitteenä osittain rinnastettavissa tässä tutkielmassa tarkasteltavan web-välimuistin hierarkkiseen luonteeseen. Suurimmassa osassa tietokoneita on käytössä monitasoinen muistihierarkia, joissa usein tarvittavaa tietoa säilytetään lähempänä prosessoria.

Esimerkiksi tavanomaisen työaseman prosessorin rekistereissä säilytetään tietoa, jota tarvitaan yhden kellosyklin aikana. Seuraavan tason primäärisen välimuistin hyödyntäminen kestää tyypillisesti yhdestä kahteen kellosykliä. Vastaavasti esimerkiksi kolmannen tason arkistomuistin (engl. off-chip cache) hyödyntäminen vaatii jo noin 20 kellosykliä. Laitteistosta riippuen, kauempana prosessorista sijaitsevat muistihierarkian tasot ottavat enemmän aikaa tiedon noutamisessa. Tällaisia muistihierarkian osia ovat keskusmuisti, levyasemat sekä nauha-asemat. (Scott, 2009.)

Web-välimuisti eroaa käsitteenä laitteiston muistihierarkiasta siinä, että muistihierarkian sijoittuessa tyypillisesti yhden laitteiston sisälle, ulottuvat yhden web-sovelluksen välimuistit lukuisten eri palvelinten kesken, jopa loppukäyttäjän asiakasohjelmiin saakka. Yhden web-sovelluksen välimuisti sijaitsee siten useiden eri laitteistojen web-sovelluskohtaisissa prosesseissa ja muistihierarkioissa. Molemmille vertailtaville käsitteille yhteneväistä on se, että pysyväisluonteisen tiedon tallentamiseksi tai hakemiseksi on käytävä aina alkuperäisen tiedon lähteellä: levyllä tai web-välimuistin tapauksessa tyypillisesti tietokannassa. Scott (2009) kertoo laitteiston muistihierarkian ominaisuudeksi, että välimuistiosumaa (engl. cache hit) pyritään hakemaan mahdollisimman läheltä prosessoria edeten tarvittaessa etäämmälle muistihierarkiassa, kunnes etsitty tieto löydetään. Yhteistä web-välimuistin kanssa on siten se, että hierarkianäkökulmasta tieto on joko heti saatavilla tai se tulee noutaa kauempaa, jopa alkuperäiseltä pysyväisluonteisemmalta tietolähteeltä. Mitä lähempänä tiedon tarvitsijaa tieto on saatavissa, sitä halvemmaksi ja nopeammaksi suoritus muodostuu ja sitä vähemmän joudutaan odottamaan muita resursseja. Web-sovelluksen välimuisteilla pyritäänkin suorituskykyparannuksiin levittämällä välimuistisisältöä mahdollisimman laajalle, jotta tuorein mahdollinen kopio alkuperäisestä tiedosta olisi saatavilla mahdollisimman lähellä loppukäyttäjää.

Mehrotra ym. (2010) kertovat tutkimuksessaan, että välimuistitekniikoita on erittäin laajasti tarjolla, ja että jokaisessa tekniikassa on omat hyvät ja huonot puolensa. Heidän mukaansa markkinoilla olevat välimuistitekniikat ovatkin tyypillisesti suunniteltu jotain tiettyä web-sovellustyyppiä ajatellen. He kertovat myös, että tyypillisimmin web-sovelluksessa käytetään yhden tai useamman välimuistitekniikan yhdistelmää. Ravin ym. (2009) mukaan huomattavia välimuistiratkaisuita on kehitetty kaikkiin web-sovellusten osa-alueisiin: palvelimille, asiakasohjelmiin, välityspalvelimiin sekä web-sovellusten eri arkkitehtuurikerroksiin. He pitävät välimuistin käyttöä erittäin tärkeänä dynaamisen sisällön tapauksessa. Näkemystään he perustelevat sillä, että suuri osa dynaamisen sisällön generointiin käytettävästä lähdetiedosta on tyypillisesti peräisin erilaisista taustajärjestelmistä esimerkiksi tietokannoista.

Scott (2009) mukaan laitteiston muistihierarkiassa lähempänä prosessoria olevat välimuistit ovat kapasiteetiltaan pienempiä, mutta nopeampia kuin kauempana sijaitseva tilavuudeltaan suurempi keskusmuisti. Hän kuvailee laitteiston muistihierarkiassa tunnettua paikallisuusominaisuuden käsitettä (engl. locality of reference): muistihierarkia on suunniteltu hyödyntämään taipumusta, jonka mukaan suurin osa tietokoneohjelmista käyttää samoja tai lähellä toisiaan

olevia muistialueita toistuvasti. Hän jatkaa kertoen, että automatisoimalla näiden muistialueiden sisällön ennakoiva siirtäminen lähemmäs prosessoria mahdollistaa tehokkaita suorituskyvyllisiä parannuksia. Web-välimuistin tapauksessa sisällön ennalta hakeminen välimuistiin on yhtälailla kannattavaa, mutta web-sovellukseen kohdistuvat pyynnöt eivät noudata paikallisuusominaisuuden järjestelmällistä lähekkäin sijaitsevan tiedon hyödyntämisen lähestymistapaa. Web-sovellukseen kohdistuvat pyynnöt saapuvat lukuisilta eri käyttäjiltä ja muodostavat luonteeltaan hajanaisemman hakujen joukon. Web-sovelluksien välimuistien osalta ennakointia kannattaakin tehdä aiempien pyyntöjen historiatietoihin pohjautuen.

Ali, Shamsuddin ja Ismail (2011) toteavat web-sisällön ennaltageneroinnin olevan erittäin tehokas välimuistin käyttöä täydentävä tekniikka. Heidän mukaansa välimuistin käyttö ja ennaltagenerointi ovat avainasemassa web-sovellusten suorituskykyparannuksia etsittäessä, sillä kyseiset tekniikat pitävät jatkossa todennäköisesti tarpeellisia web-sovelluksen olioita lähempänä käyttäjää. Ennaltageneroinnin ja välimuistin yhdistelmä nostaa osumatarkkuutta, jolla pyynnöt kohdistuvat välimuistiin ja siten vähentää sisällön generoinnin viivettä. Kroeger, Long ja Mogul (1997) esittävät tutkimuksessaan, että välimuistin ja ennaltageneroinnin yhdistelmästä on saatavissa jopa 57 % parannus viiveeseen, kun pelkän välimuistin käyttö aikaansaa enintään 23 % parannuksen. Shin, Songin, Dingin, Guun ja Wein (2005) mukaan välimuistin käyttäminen vaikeutuu jatkuvasti, sillä verkkosisällön luonne on enemmissä määrin dynaamista. He toteavat kuitenkin, että ennaltagenerointi mahdollistaa verkkosisällön tarjoamisen optimointimahdollisuuksia ja uskovat, että välimuistin ja ennaltageneroinnin yhdistelmä on lupaavin palvelunlaadun parannuskeino web-sovellusten tulevaisuudessa.

Välimuisti sekä sen yhteydessä tapahtuva ennaltagenerointi ovat tutkittuja ongelma-alueita. Web-tekniikoiden kehittyessä myös tarve näiden tekniikoiden tutkimiselle on jatkuva. Tämän hetken tutkituimmat aiheet liittyvät dynaamisen ja yksilöidyn sisällön tehokkaaseen palvelemiseen. Yksilöidyn sisällön generoimiseen tiiviisti liittyvät istunto-olion tiedot kuitenkin lisäävät haasteita sivugeneroinnin siirtämisessä pois alkuperäiseltä web-sovelluspalvelimelta ja siten rajoittavat skaalautuvuutta. Ajankohtaisena tutkimusalueena on myös P2P-teknologioihin liittyvä välimuisti, jossa tavanomaisesta asiakas-palvelin-arkkitehtuurista on siirrytty tavoittelemaan verkon ja palvelinkuorman näkökulmasta ideaalista tiedon jakamista suoraan kuluttajien kesken.

Tutkielmassa keskitytään tehostamaan web-sisällön tarjoamista käyttäjälle määrittelemällä ennakoivan välimuistiratkaisun vaatimukset ja esittämällä keinot, joilla ennakoiva välimuisti voitaisiin toteuttaa. Tavoitteena on löytää hyvä malli välimuistin ja ennaltageneroinnin yhdistelmälle, jotta käyttäjän kokemaa viivettä saataisiin vähennettyä ja pyyntöjen välisiä viive-eroja tasoitettua. Tällainen oppiva välimuisti päättelisi keräämänsä resurssi- ja tilatiedon avulla välimuistista poistuneen sisällön uudelleengeneroimisesta. Keskeisenä osana ennakointia on enaltagenerointiin käytettävä kustannusfunktio, jolla päätetään

kunkin välimuistiolion ennaltageneroinnista kyseisen välimuistiolion ominaisuuksiin perustuen. Tutkielmassa esitellään uusi kuormitusta tavoitteleva ahne kustannusfunktio, joka huomioi olion generointiin käytetyn työmäärän ja olion aikaansaaman parannuksen välimuistin kokonaisosumatarkkuuteen. Lopulta eri kustannusfunktioiden aikaansaamia vaikutuksia ennakoivan välimuistin toimintaan mitataan simuloidussa ympäristössä.

Tutkimusongelma jakautuu kysymyksiin:

- Mitä vaatimuksia web-sovelluspalvelimella toimivan ennakoivan välimuistin toteuttamiselle on?
- Miten uusi tutkielmassa esitelty kustannusfunktio suoriutuu ja asettuu jo aiemmin tunnettuihin kustannusfunktioihin nähden?
- Millä kustannusfunktiolla viivettä laskeva ja tasoittava ennakoiva välimuistiratkaisu kannattaa toteuttaa yksilöityä sisältöä sisältävässä web-sovelluksessa?

Tutkimuksen teoreettinen perusta pohjautuu kirjallisuuskatsaukseen. Tutkimus on luonteeltaan myös osittain teoriaa testaava. Tämän lisäksi tutkimuksen voidaan katsoa olevan myös suunnittelutieteellinen konstruktiivinen tutkimus, sillä tavoitteena on suunnitella malli ja tehdä sen laskentakaavoihin pohjautuen simuloiva toteutus. Laskentakaavojen relevanttiutta evaluoidaan testaamalla niitä simuloidussa toimintaympäristössä. Tiedonkeruu tapahtuu luomalla kutakin simulaatiota varten riittävä määrä pohjatietoa ja laskemalla oletuksiin pohjautuen kunkin simulaation lopputulos.

Tutkielman aihepiiriin kohdistuneen laajan aiemman tutkimuksen myötä, varsinaisten uusien innovaatioiden kehittäminen on hankalaa. Olemassa olevan teorian testaamiseen sekä uusien toteutusten konstruoiminen eri tutkimustulosten pohjalta on kuitenkin tutkimisen arvoista. Tutkielman voidaan nähdä sulautuvan hyvin ennalta tehtyyn ja ajankohtaiseen tutkimukseen, sillä esitettävä ennaltagenerointiin pohjautuva hyödyntää olemassa olevien tutkimusten tuloksia ja havaintoja.

Lähtökohtana tutkimukselle on taustaoletus, että esitetylle viivettä vähentävälle ja tasaavalle ennakoivalle välimuistiratkaisulle olisi tarvetta. Tutkimustuloksissa on odotettavissa ennakoivan välimuistiratkaisun mallin suhteen suorituskykyhyötyä web-sovelluksen näkökulmasta. Tämän lisäksi tarkastelun kohteena on tavoittelemisen arvoinen, mahdollisimman pieni ja tasainen käyttäjän kokema viive. Tutkielmassa esiteltävä kuormitusta tavoitteleva ahne kustannusfunktio voidaan nähdä arvokkaana, mikäli simulointituloksissa havaitaan käyttäjän kokeman viiveen näkökulmasta parannusta suhteessa kirjallisuuden aiemmin esittämiin kustannusfunktioihin. Tällöin esimerkiksi tietylle suositulle sivulle mentäessä pyydetyn sisällön laskennallisesti raskaat osat olisivat jo prosessoituina valmiiksi ja ne voitaisiin vain noutaa välimuistista.

Web-sovellusten välimuistiratkaisut ovat luonteeltaan eri tilanteisiin sopivia. Ravi ym. (2009) kuvaavat välimuistitekniikoiden luonnetta siten, että tietty välimuistitekniikka voi olla todella tehokas yhdessä sovelluksessa ja vähemmän tehokas toisessa. Onkin oletettavissa, että esitetyn ennaltageneroivan välimuis-

tiratkaisun malli toimii parhaiten vain tietyntyyppisessä web-sovelluksessa. Sama ennakko-oletus pätee myös ehdotetun, työmäärää ja osumatarkkuutta tavoittelevan ahneen kustannusfunktion osalta. Tutkielmassa suoritettavissa simuloinneissa etsitään web-sovelluksen ominaisarvoja, joilla välimuistin ja ennaltageneroinnin avulla saavutettavat hyödyt olisivat esitetyn kustannusfunktion osalta korkeimmat.

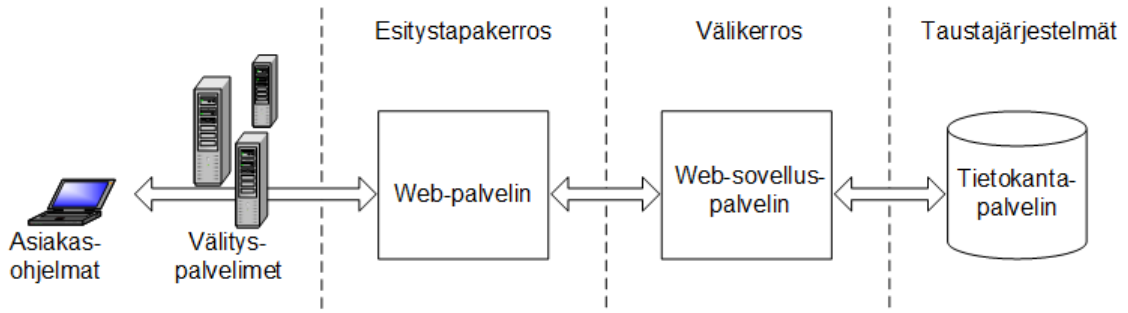
Seuraavassa, tutkielman toisessa luvussa esitellään tarkemmin tutkimuksen aihealueen ongelmakenttää ja kuvataan käytettävät käsitteet. Toinen luku sisältää myös kuvauksen web-sovelluksen arkkitehtuurista. Tämän lisäksi luvussa perehdytään myös web-sovelluksen dynaamiseen sekä yksilöityyn sisältöön sekä niiden haasteisiin. Kolmannessa luvussa syvennytään ennakoivan välimuistin aihepiiriin ja koostetaan välimuistiratkaisun malli määrittelemällä joukko vaatimuksia, joiden pohjalta reaalimaailman ratkaisu olisi mahdollista toteuttaa. Neljännessä luvussa tuodaan esille riittävä teoriaperusta kustannusfunktioiden simulointiympäristön kehittämiselle. Neljännen luvun lopussa esitellään myös uusi kuormitusta tavoitteleva ahne kustannusfunktio, jonka tavoitteena on löytää kuormitukseltaan raskaimmat välimuistioliot. Viides luku kertoo koejärjestelyistä, simulointiympäristöstä, suoritettavista kokeista ja esittelee käytettävät mittarit, joilla simulointituloksia tullaan esittämään. Kuudennessa luvussa esitellään eri kustannusfunktioiden suoriutumista mittaavat simulointitulokset ja seitsemännessä luvussa päädytään johtopäätöksiin analysoimalla tutkimustuloksia ja huomioimalla myös aihealueen aiempien tutkimusten havaintoja. Viimeisessä luvussa tehdään tutkielmasta yhteenveto.

2 VÄLIMUISTI WEB-SOVELLUKSESSA

Tässä luvussa syvennytään johdannossa esitettyyn ongelmakenttään. Tavoitteena on rakentaa tutkimusongelman näkökulmasta riittävä teoreettinen perusta tutkimuksen suoraviivaiselle tulkinnalle.

2.1 Välimuisti ja web-sovelluksen arkkitehtuuri

Tutkimuskysymysten kannalta web-sovelluksen arkkitehtuurivalinta ei ole merkityksellinen. On silti hyvä määritellä tutkielmassa käytettävä termi: *monikerroksinen web-sovellusarkkitehtuuri*. Lu ja Gokhale (2007) kuvailevat web-sovelluksen arkkitehtuurin olevan avainasemassa web-sovellusta suunniteltaessa. Arkkitehtuuri määrittää jokaiselle sovelluksen osalle tarkat toiminnalliset vastuut ja kuinka eri osat vuorovaikuttavat keskenään. Kuviossa (KUVIO 1) on esitettynä Lu ja Gokhale (2007, s. 2) määritelmää mukaileva tyypillinen monikerroksinen web-sovelluksen arkkitehtuuri. Tällaisessa monikerrosarkkitehtuurissa web-sovellus koostuu vähintään kolmesta arkkitehtuurikerroksesta.



KUVIO 1 Web-sovelluksen monikerrosarkkitehtuuri (Lu & Gokhale, 2007, s. 2)

Ravi ym. (2009, s. 945) kuvaavat edellä esitetyn, tyypillisen monikerrosarkkitehtuurin kerrosten vastuut seuraavasti:

> Monikerroksisessa web-sovelluksessa web-palvelin toimii esitystapakerroksella edustapalvelimena (engl. front-end) ja vastaanottaa käyttäjiltä saapuvat HTTP-pyynnöt. Suurin osa dynaamisesta sisällöstä tuotetaan web-sovelluspalvelimilla, välikerroksella (engl. middle tier), jossa sijaitsevat liiketoimintaoliot, jotka toimivat rajapintana käyttäjien ja taustajärjestelmien (engl. back-end) välissä. Taustajärjestelmänä on yleensä tietokanta tai jokin muu järjestelmä. Monikerroksinen web-sovellus voi siten sisältää välimuistiratkaisuita jokaisella sovelluskerroksellaan.

Edellä esitetty määritelmä on valittu tukemaan tämän tutkimuksen etenemistä. Kerroksia voi siis olla useampiakin mutta selkeyden vuoksi web-sovelluksen nähdään koostuvan esitetyistä kolmesta kerroksesta.

Mehrotra ym. (2010) pitävät sijaintipohjaista luokittelua laajasti hyväksyttynä välimuistien luokitteluna. He kertovat, että monikerrosarkkitehtuurissa välimuisteja voidaan sijoittaa useisiin paikkoihin: asiakasohjelmiin, välityspalvelimiin, verkon reunapalvelimille (engl. edge-of-network server), Internet-palveluntarjoajalle, liikeyritysten reunapalvelimille (engl. edge-of-enterprise server), web-palvelimille, web-sovelluspalvelimille sekä tietokantapalvelimille.

2.2 Välimuisti ratkaisuna viiveeseen

Ravin ym. (2009, s. 944) mukaan tavanomaisessa tapauksessa, jossa käyttäjä tekee sivupyynnön web-palveluun, koettu vasteaika koostuu useista eri tekijöistä kaavan (1) mukaisesti.

$$L_t = L_{dns} + L_g + L_{nt} + L_{na} + L_u \quad (1)$$

Kaavassa 1 esitetty kokonaisviive L_t on siten summa seuraavista tekijöistä:

- DNS-selvityksen (engl. DNS resolution) L_{dns} viiveestä,
- web-sovelluspalvelimen sisällön generoinnin L_g kestosta,
- web-sovelluspalvelimen ja käyttäjän liityntäverkon välisestä kahdensuuntaisen verkkoliikenteen L_{nt} viemästä ajasta,
- käyttäjällä käytössä olevan Internet-yhteyden ja muun liityntäverkon aiheuttamasta L_{na} viiveestä ja
- käyttäjän laitteiston prosessointikyvystä L_u johtuvasta viiveestä.

Ravi ym. (2009) jatkavat kertoen, että DNS-selvityksen L_{dns} viiveen nopeuttamiseen on olemassa useita ratkaisuja, mutta esimerkiksi käyttäjän laitteiston aiheuttamalle L_u viiveelle voidaan tehdä vain hyvin vähän. Käyttäjän päätelaitteessa oleva nopea prosessori tuottaa web-sivun nopeammin näkyville kuin hitaampi prosessori. Heidän mukaansa tiedeyhteisö onkin kiinnostunut tehostamaan muuttujista L_g, L_{nt} ja L_{na} aiheutuvia viiveitä ratkaisuilla, jotka paranta-

vat alkuperäisten web-sovelluspalvelinten vasteaikaa, web-palvelun skaalautuvuutta sekä vähentävät verkon siirtokapasiteetin tarvetta.

Tässä tutkimuksessa tavoitteena on vähentää ja tasoittaa sisällön generoinnin L_g viivettä ja siten aikaansaada muutosta myös kokonaisvasteaikaan. Ravi ym. (2009) ovat todenneet, että dynaamisen web-sivun generoinnista aiheutuva viive määräytyy alkuperäisten web-sovelluspalvelinten suorituskyvyn ja skaalautuvuuden mukaan. Koen itse vaikuttaviksi tekijöiksi myös suoritettavien tehtävien kompleksisuuden, algoritmien optimoinnin, mahdolliset web-sovelluspalvelimen välimuistiratkaisut sekä taustajärjestelmistä koostuvan viiveen. He sisällyttävät edellä mainitut lisämuuttujat yhtälössään L_g-viiveeseen. Tämän tutkimuksen näkökulmasta on kuitenkin tärkeää tuoda esille taustajärjestelmistä aiheutuvan viiveen osuus käyttäjän kokemassa kokonaisviiveessä L_t, johon web-sovelluspalvelimella sijaitsevalla ennakoivalla välimuistiratkaisulla voidaan vaikuttaa.

Acharjee (2006) listaa loppukäyttäjien, verkon hallinnoijan ja sisällöntuottajan näkökulmasta kolme houkuttelevaa hyötyä liittyen web-välimuistin käyttöön millä tahansa monikerroksisen web-sovelluksen arkkitehtuurikerroksella: käyttäjän kokeman viiveen vähentyminen, kaistankäytön (engl. bandwidth consumption) säästäminen sekä alkuperäisten web-sovelluspalvelinten kuorman vähentyminen. Kaistalla tarkoitetaan tietoliikenneyhteyden siirtokapasiteettia monikerroksisessa web-sovelluksessa kulloinkin tarkasteltavan arkkitehtuurikerroksen ja sitä seuraavan kerroksen välillä. Esimerkiksi web-sovelluspalvelimen välimuistilla voidaan vähentää tietoliikenneyhteyden kaistankulutusta web-sovelluspalvelimen ja taustajärjestelmän välillä.

2.2.1 Web-sovelluksen dynaamisen sisällön välimuisti

Web-teknologiat kehittyvät ja suuntaus kohdistuu dynaamisempaan ja jatkuvasti yksilöidympään sisältöön. Kehitykselle ominaista on myös siirtyminen sovelluskeskeisestä verkosta palvelukeskeiseen. Mehrotra ym. (2010) ovat listanneet yksilöidyn ja dynaamisen web-sisällön tuottamisesta aiheutuvia, web-palveluiden suorituskykyyn negatiivisesti vaikuttavia ominaisuuksia:

- Muuttuvan sisällön luontikustannukset ovat laskennallisesti kalliita.
- Prosessorikuorma on moninkertainen, staattisen sivun palvelemiseen verrattuna.
- Mikäli sivuun kohdistuu suuria määriä sivupyyntöjä, saattaa dynaamisen sivun palveleminen vaatia merkittäviä erityislaitteistoja.
- Mikäli taustalla olevan tiedon päivittäminen vaikuttaa useisiin dynaamisiin sivuihin, tulee kaikki tällaiset sivut päivittää vastaamaan uutta tietoa.
- Dynaamisen sisällön generointi asettaa merkittävän kuorman web-sovelluspalvelimille ja aikaansaa siten suorituskyvyn suhteen pullonkauloja.

- Dynaamisen sivun generointi vaatii tyypillisesti yhden tai useamman tietokantakyselyn suorittamista tietokantapalvelimella. Tästä johtuen tietokantaan kohdistuvien kyselyiden läpimenoajat voivat kasvaa hallitsemattomasti suurten yhdenaikaisten sivupyyntömäärien aikana.

Esitellyistä negatiivisista vaikutuksista voidaan huomata, että dynaamisen sisällön kustannukset ovat huomattavat. Ravi ym. (2009) asettavat tärkeäksi sisällön generoimisen tehostamisen web-sovelluspalvelimilla, mikäli web-sovellus on interaktiivinen, eikä skaalautuvuuden suhteen tai maantieteellisesti ole mahdollista siirtää suoritettavaa laskentaa lähelle käyttäjää. Dutta, Thomas, Datta ja Keskinocak (2007) kuvailevat välimuistin olevan laajasti omaksuttu mekanismi, jolla saavutetaan suorituskykyä ja skaalautuvuutta modernin informaatioteknologian infrastruktuureissa. He luonnehtivat välimuistia ratkaisuksi, joka mahdollistaa sovelluksen uudelleen käyttää aiemmin generoituja olioita välimuistista ja siten poistaa ylimääräisten uusien olioiden luomisen tarpeen, vähentäen olioiden luomiseen kuluvia kustannuksia.

Ravi ym. (2009) listaavat markkinoiden johtavien valmistajien menestyneitä dynaamisen web-sisällön välimuistitallentamisen mahdollistavia tuotteita kuten: ASP.NET, BEA WebLogic, WebSphere ja Oracle 9iAS. He mainitsevat lisäksi mahdollisuuden toteuttaa sovelluskohtaisia välimuistiratkaisuita websovelluksen kehyksen avulla web-sovelluspalvelimelle. Heidän mukaansa websovelluspalvelimilla sijaitsevilla välimuistiratkaisuilla voidaan vähentää websovelluspalvelimen ja taustajärjestelmien kuormaa sekä parantaa vasteaikoja kovan käyttäjäkuormituksen aikana. He kertovat myös, että tärkein ominaisuus välikerroksen välimuisteissa on tiedon tarjoaminen nopeasti välimuistista ilman, että sitä täytyy jatkuvasti kysellä tietokantapalvelimilta. Heidän mukaansa tällainen välikerroksen välimuisti vähentää kommunikointitarvetta tietokannan kanssa jokaista sivupyyntöä kohden. Siten tietokannan kuorma sekä välikerroksen ja taustajärjestelmien välinen verkkoliikenne vähenee. Dutta ym. (2007) mainitsevat, että olioiden tallentaminen välimuistiin on erittäin suosittua kaikissa moderneissa ohjelmointiympäristöissä. Heidän mainitsemiaan esimerkkejä ovat J2EE-sovelluspalvelimet WebLogic ja WebSphere. Mehrotra ym. (2010, s. 594) puolestaan kuvailevat nykytilannetta näin: ”On olemassa lukuisia toimittajia, jotka tarjoavat web-sovellusten välimuistiratkaisuita, joiden tavoitteena on olla riittäviä jopa pilvipalveluille ja tarjota skaalautuvuusmahdollisuuksia usean palvelimen kesken”. Tällaisiksi välimuistiratkaisuiksi tai niitä sisältäviksi tuotteiksi he mainitsevat vaihtoehtoja kuten: Gear6, NetCache, BranchCache, Akamai, WebSphere ja ASP.NET.

2.2.2 Dynaaminen ja yksilöity sisältö

Dynaaminen web-sisältö koostuu muuttuvasta ei-staattisesta sisällöstä. Esimerkkinä tästä voisi olla uutissivusto, jossa uutiset päivittyvät useita kertoja päivässä. Nämä uutiset näkyvät kaikille käyttäjille yhteisenä yleisenä muuttu-

vana sisältönä. Dynaaminen web-sisältö ja muut web-sovellustekniikat ovat mahdollistaneet esimerkiksi käyttäjä- tai roolipohjaisen web-sisällön tuottamisen. Tästä sisällöstä käytetään tämän tutkielman puitteissa nimitystä yksilöity sisältö. Dynaaminen sisältö voi olla joko yksilöityä tai kaikille käyttäjille yhteistä. Yksilöidyn sisällön tuottamiseen vaaditaan tyypillisesti käyttäjän kirjautuminen web-palveluun. Myös staattinen sisältö voi olla yksilöityä, mutta tämän tutkielman puitteissa yksilöidyn sisällön katsotaan olevan dynaamista ja siten taustajärjestelmiin pohjautuvaa.

Vallitseva suuntaus istuntotiedon käyttämiseen web-sovelluksissa tekee lähes jokaisesta palvelinpyynnöstä yksilöllisen ja siten välimuistitekniikoiden käyttö vaikeutuu. Yleensä istunto sisältää käyttäjään sidottuja tietoja kuten identiteetin, käyttövaltuutuksen ja istuntoavaimen. Nämä yksilöivät tiedot säilyvät web-palvelinten tiedossa eri sivupyyntöjen välillä ja mahdollistavat sen, että peräkkäiset sivupyynnöt voivat vaikuttaa toisiinsa. (Ravi ym., 2009.)

Yksilöidyn sisällön generointi on siis huomattavasti pelkän dynaamisen sisällön generoimista raskaampaa. Istunnon mukanaan tuomien yksilöivien tietojen ollessa rajoitteena, yksilöidyn sisällön prosessointia voidaan tehdä vain web-sovelluspalvelimilla, joilla istunto-olio on tiedossa. Probir ja Rau-Chaplin (2006) kertovat tutkimuksessaan, että yksilöity sisältö heikentää välityspalvelinten välimuistien tehokkuutta. Välityspalvelinten välimuisteista saatava hyöty on yksilöidyn sisällön kohdalla vielä pienempi, kuin pelkän dynaamisen sisällön tapauksessa saatava vastaava hyöty. He kertovat myös, että molemmat: dynaaminen sekä yksilöity sisältö rasittavat huomattavasti web-sovelluspalvelinten suorituskykyä ja vähentävät skaalautuvuutta, sillä lähellä käyttäjää sijaitsevista välimuisteista saatavat hyödyt vähenevät.

Ravi ym. (2009) esittävät, että web-sivut voidaan luokitella karkeasti tietoturvatarpeiden mukaan. Monet uutissivustot vaativat dynaamisesta sisällöstä huolimatta vain vähän tietoturvaa, sillä kaikki käyttäjät näkevät saman sisällön eikä siten ole tarvetta sisäänrakennetuille tietoturvamekanismeille. Kuitenkin heti, kun on tarvetta näyttää yksilöityä sisältöä, on toteutettava jonkinasteisia tietoturvamekanismeja. Esimerkiksi verkkokaupoissa maksutoimintojen yhteydessä tietoturvavaatimukset ovat suuret. He toteavat tiukkojen turvallisuusvaatimusten mukaisten toteutusten tuovan mukanaan rajoituksia myös skaalautuvuuteen.

2.3 Välimuistiin liittyvät haasteet

Tässä alaluvussa perehdytään web-sovelluspalvelimella käytettävien välimuistien haasteisiin. Keskeisimpiä ongelmia ovat sen päätteleminen milloin ja mitä sisältöä välimuistiin tulisi tallentaa. Vastaavasti, mikäli välimuistille määritetty muistialue täyttyy, tulee päätellä sopivimmat välimuistioliot poistettaviksi uusien, parempien välimuistiolioiden tieltä.

2.3.1 Välimuistiin talletettavien olioiden valinta

Dutta ym. (2007) toteavat, että välimuistitoteutuksen käyttöönotossa tulee tehdä kaksi tärkeää päätöstä. Ensiksi tulee päättää järjestelmän suunnittelun aikana se, mitkä sovelluksen oliotyypit ovat hyviä vaihtoehtoja välimuistiin talletettaviksi. Toiseksi tulee päättää ajon aikana se, mitkä tietyt välimuistioliot tulisi tallettaa tai säilyttää välimuistissa. Jälkimmäinen ongelma yhdistyy tunnettuun välimuistin korvausmenettelyn (engl. replacement policy) haasteellisuuteen, jota käsitellään seuraavassa alaluvussa.

Dutta ym. (2007, s. 815) jatkavat kertoen huonon oliotasolla tehdyn välimuistikäytön aikaansaavan mahdollisia suorituskykyä ja skaalautuvuutta heikentäviä vaikutuksia. He toteavat, että tällainen huono web-sovelluksen suunnittelussa tehty valinta saattaa muuttaa jopa koko välimuistin käytön hyödyttömäksi ja kuvailevat ongelmaa esimerkin avulla:

> Jos ajonaikainen välimuistiolion haku välimuistista vastaa työmäärältään uuden olion generoimiseen vaadittua työmäärää, ei välimuistin käyttö ole kannattavaa. Tällaisessa tapauksessa välimuistin käytöstä aiheutuva turha kuormitus tekee välimuistin käytöstä hyödyltään ennemminkin negatiivista.

Mehrotra ym. (2010) listaavat välimuistiratkaisun toteutukseen liittyviä keskeisiä kysymyksiä:

- Mitä välimuistiin tallennetaan?
- Missä välimuistin tulisi sijaita monikerrosarkkitehtuurissa?
- Minkä kokoinen välimuistin tulisi olla?
- Mikä olisi hyvä välimuistiolion raekoko (engl. granularity)?

Välimuistiolion raekokoa tarkasteltaessa etsitään vastausta kysymykseen, onko kannattavampaa tallettaa esimerkiksi koko sivun sisältö yhteen välimuistiolioon, vai pilkkoa sisältö useampaan loogista kokonaisuutta vastaavaan välimuistiolioon. Mikäli välimuistitila ei ole rajoitteena, on vaihtoehtona myös useamman raekooltaan eriävän, samaa sisältöä sisältävän välimuistiolion säilyttäminen samalla tai eri arkkitehtuurikerroksilla sijaitsevissa välimuisteissa. Tämän voidaan nähdä nopeuttavan web-palvelun vasteaikaa ja saatavuutta, sillä raekooltaan suuren välimuistiolion vanhentuessa, ei kaikkea siihen liittynyttä välimuistisisältöä ole tarvetta tuottaa uudelleen täysin alusta asti. Mehrotra ym. (2010) huomauttavat kuitenkin, että tämä herättää huolen tiedon eheydestä, tietoturvasta sekä johdonmukaisuudesta. Dutta ym. (2007) kuvailevat välimuistitoteutuksen haasteeksi myös sen, etteivät jotkin välimuistioliot säily välimuistissa riittävän pitkään, jolloin ne eivät ehdi olemaan hyödyksi yhdellekään myöhemmälle sivupyynnölle. Toisin sanoen, mikäli jonkin välimuistiolion päivitystiheys on tiiviimpi kuin välimuistiolioon kohdistuvien pyyntöjen tiheys, ei ole kannattavaa tallettaa tällaista välimuistioliota välimuistiin. Myös Probiria ja Rau-Chaplinia (2006) mukaillen voidaan todeta, että välimuistia käytettäessä tulee tehdä valinta välimuistiolion tallentamisen tai hylkäämisen välillä pohjau-

tuen välimuistiolion käyttö- ja päivitystiheyden suhteeseen. Ravi ym. (2009) listaavat avainasioiksi välimuistitoteutuksessa oikean välimuistiin talletettavan sisällön valinnan onnistumisen, välimuistin hyvän sijainnin sovelluksen arkkitehtuurissa sekä välimuistissa olevan sisällön tehokkaan mitätöinnin.

2.3.2 Välimuistiolioiden korvausmenettely

Ali ym. (2011) esittelevät tutkimuksessaan välimuistiolioiden korvausmenettelyä eli säännöstöä, johon pohjautuen välimuistisisällön poistamisesta päätetään. He toteavat, että välimuistin korvausmenettely on ydinasia välimuistitoteutuksessa ja välimuistin rajallisen koon takia on oltava olemassa mekanismi välimuistisisällön hallitsemiseksi. Jos välimuisti on täynnä ja uusi välimuistiolio on tuloillaan välimuistiin, täytyy korvausmenettelyn päätellä, mikä välimuistiolio poistetaan uuden olion tilalta, vai hylätäänkö välimuistiin saapumassa oleva olio. Osumatarkkuuden kannalta optimaalisin web-sovelluspalvelimen välimuistin korvausmenettely tähtää mahdollisimman tehokkaaseen välimuistitilan käyttöön, parantaa välimuistin osumatarkkuutta ja vähentää siten web-sovelluspalvelimen ja taustajärjestelmien kokemaa kuormitusta.

Ayani, Teo ja Ng (2003), Cobb ja ElArag (2008) ja Koskela, Heikkonen ja Kaski (2003) kuvailevat perinteisten välimuistisisällön korvausmenettelyjen olevan rajallisia. Rajallisuuden nähdään johtuvan siitä, että korvausmenettelyt huomioivat päätöksenteossaan vain jonkin tietyn tekijän ja siten muut tekijät jäävät huomioitta. Tällaiseksi perinteiseksi korvausmenettelyksi edellä mainitut tutkimukset esittävät esimerkiksi LRU-korvausmenettelyn (engl. least recently used), joka poistaa välimuistista ne oliot, jotka ovat olleet siellä pisimpään ilman, että niihin on viitattu. Ali ym. (2011) toteavat perinteisten korvausmenettelyiden ongelmaksi, että vain suosituimpaan osaan välimuistiin päätyneistä olioista kohdistuu palvelinpyyntöjä. Tällöin suurin osa välimuistissa olevista välimuistiolioista ei koskaan osu pyyntöjen kohteeksi. Kyseisestä korvausmenettelyihin liittyvää ongelmaa kutsutaan välimuistin saastumisongelmaksi (engl. cache pollution problem).

Alin ym. (2011) mukaan Chen (2008) ja Wong (2006) esittävät, että usean tekijän huomioiminen viisaan korvausmenettelyn toteuttamisessa ei ole helppoa, sillä tekijöillä saattaa olla erilaisia tärkeysasteita tilanteesta tai ympäristöstä riippuen. Näin ollen, ideaalisten välimuistiolioiden löytäminen on edelleen suuri haaste olemassa olevienkin välimuistitekniikoiden keskuudessa.

Ali ym. (2011) jatkavat liittyen korvausmenettelyihin. Heidän mielestään välimuistin korvausmenettelyillä on erittäin tärkeä rooli välimuistitoteutuksissa. Tehokkaan korvausmenettelyalgoritmin suunnittelu on keskeinen vaatimus korkealuokkaisten ja kehittyneiden välimuistimekanismien toteutuksissa.

Korvausmenettelyalgoritmin rooli välimuistin hyödyllisyyden kannalta on merkittävä. Tässä tutkimuksessa ei kuitenkaan tarkastella eri korvausmenettelyalgoritmeja. Sen sijaan tutkimuksessa esiteltävälle ennakoivan välimuistiratkaisun mallille asetetaan vaatimukseksi, että siinä hyödynnettävän korvausmenettelyn tulee olla ennaltageneroinnin kustannusfunktion kanssa yh-

teensopiva. Korvausmenettelyn ja ennaltageneroinnin yhteensopivuutta käsitellään tutkielman seuraavassa luvussa.

2.4 Luvun yhteenveto

Tässä luvussa esiteltiin tutkimuksen kulun kannalta oleellisia kokonaisuuksia ja käsitteitä. Seuraavassa luvussa käsitellään välimuistisisällön ennakointiin pohjautuvaa ennaltagenerointia sekä tutustutaan välimuistin käytön ja ennaltageneroinnin yhteishyötyihin. Tämän lisäksi mallinnetaan välimuistiratkaisun malli esittämällä joukko vaatimuksia välimuistin ja ennakoinnin aihepiirien haasteisiin vastaamiseksi.

3 ENNAKOIVAN VÄLIMUISTIN MALLINTAMINEN

Kolmannessa luvussa käsitellään välimuistiolioiden ennaltageneroinnin aihepiiriä sekä mallinnetaan ennakoivan välimuistiratkaisun malli. Mallintaminen tapahtuu välimuistin komponenttien esittelyllä sekä listaamalla joukko ennakoivalle välimuistiratkaisulle keskeisiä vaatimuksia. Tavoitteena on tuottaa malli, jonka pohjalta on mahdollista rakentaa ennakoivan välimuistiratkaisuntoteutus web-sovelluspalvelimelle. Tässä luvussa olevan mallinnukseen keskittyvän alaluvun tarkoituksena on vastata osatutkimusongelmaan: ”mitä vaatimuksia web-sovelluspalvelimella toimivan ennakoivan välimuistin toteuttamiselle on”.

Seuraavassa alaluvussa käsitellään välimuistitekniikoiden rinnalla käytettyä ennakointiin perustuvaa ennaltageneroitia sekä ennakointiin liittyviä haasteita aikaisemmin tehtyjen tutkimusten pohjalta.

3.1 Välimuistiolioiden ennaltagenerointi

Ennaltageneroinnin tavoitteena on minimoida web-sovelluspalvelimen sisällön generoinnin L_g-viive ja siten pienentää käyttäjän kokemaa kokonaisviivettä. Alin ym. (2011) mukaan ennaltagenerointitekniikoita voidaan toteuttaa alkuperäisille web-sovelluspalvelimille, välityspalvelimille tai asiakasohjelmiin. Asiakasohjelmissa tapahtuva ennaltagenerointi keskittyy ennakoimaan yhden käyttäjän navigointikäyttäytymistä, jolloin navigoidut sivustot voivat jakautua monen eri web-palvelimen kesken. Ennaltageneroinnin tapahtuessa alkuperäisillä web-sovelluspalvelimilla, pyrkimyksenä on kaikkien eri käyttäjien navigointikäyttäytymisen tunnistaminen yhden web-palvelun osalta.

Shi ym. (2005) kertovat, että ennaltageneroinnin aikaansaaman viiveen parannuksen negatiivisena kustannuksena aiheutuu kasvua järjestelmän ja verkon kuormituksessa. Ennaltageneroinnin on siis kohdattava se haaste: kuinka löytää sopiva kompromissi viiveen helpottamisen ja potentiaalisen verkko- ja resurssikuorman välillä. Seuraavassa on annettu kuvaavat ääriesimerkit kyseisestä tasapainottelusta:

> Ensimmäisessä esimerkkitapauksessa noudettaisiin kaikki välimuistioliot paikalliseen välimuistiin, jolloin aikaansaataisiin mahdollisimman hyvä välimuistin käytön osumatarkkuus. Tällöin kuitenkin aiheutettaisiin valtavia määriä tarpeetonta kaistankulutusta haettaessa ja päivitettäessä välimuistiolioita, joita ei kuitenkaan koskaan pyydettäisi käyttäjien toimesta. Toinen ääritapaus olisi jättää ennaltagenerointi kokonaan pois, jolloin välimuisti toimisi täysin passiivisesti pyyntöjen perusteella. Tällainen välimuisti kuluttaisi vähiten kaistaa huonoimmalla mahdollisella välimuistin käytön osumatarkkuudella ja seurauksena olisivat suuret käyttöviiveet. (Jiang, Wu & Shu, 2002, alaluku 4.1)

Edellä esitetyssä ääriesimerkissä, jossa välimuistin esitetään toimivan pelkästään passiivisesti ilman ennaltagenerointia, on kuitenkin mielestäni yleistetty liikaa sitä, että välimuistin osumatarkkuus olisi tällä tavalla huonoin. Osumatarkkuuteen vaikuttaa oleellisesti myös passiivisesti täyttyvän välimuistin korvausmenettely eli se, millä perusteella tarpeettomimpia olioita siivotaan pois välimuistista. Jiang ym. (2002) jättävätkin luultavasti mainitsematta edellä olevassa lainauksessa sen, ettei heidän esittämässään passiivisesti toimivassa välimuistissa ole käytössä kumpaakaan: ennaltageneroivaa mekanismia tai korvausmenettelyä. Vaikka näin olisi, täyttyisi äärellisen kokoinen välimuisti olioiden pyyntötodennäköisyyksiä mukaillen, jolloin silloinkaan ei välimuistiolioiden uudelleenkäytettävyyden osumatarkkuus olisi kaikista alhaisin. Jotta saataisiin aikaan huonoin mahdollinen välimuistin uudelleenkäytettävyyden osumatarkkuus, tulisi välimuistiin ennaltageneroida vähiten välimuistin osumatarkkuutta nostavat välimuistioliot.

Ennaltageneroinnin kustannukset voivat olla suuret, jollei ennaltagenerointiprosessia suunnitella huolella. Ennaltagenerointiin liittyy olennaisesti käyttäjän käyttäytymisen olettaminen tilastollisen tiedon pohjalta. Ravi ym. (2009) kertovat kuitenkin, että käyttäjän todellinen käyttäytyminen voi erota olettamuksista. Lisäksi, jos todellisuus on päinvastainen olettamuksiin nähden, voi suorituskykyhyöty olla jopa negatiivinen, sillä ennakoinnin laskennasta ja ennaltageneroinnista aiheutuva kuormitus osoittautuu turhaksi. Shin ym. (2005) mukaan ennaltageneroinnista saattaa aiheutua haittavaikutuksia, sillä websovelluspalvelinten tulee palvella normaalien pyyntöjen lomassa myös ennaltagenerointipyyntöjä, jotka saattavat saapua ryöppynä, mikäli ennaltagenerointia ei kontrolloida riittävästi. He osoittavat ongelmaksi myös sen, että liian intensiivinen ennaltagenerointi pakottaa rajatun kokoisessa välimuistissa olevia mahdollisesti hyviä välimuistiolioita poistumaan ennaltageneroitavien olioiden tieltä.

On tärkeää muistaa, että ennaltagenerointi auttaa vain käyttäjän kokemaan viiveeseen, eikä ennaltagenerointi aikaansaa kokonaisvaltaista suorituskykyhyötyä järjestelmän näkökulmasta. On kuitenkin mahdollista ajoittaa ennaltagenerointi tasaisemmin ruuhkattomille ajankohdille. Ruuhka-aikoina voidaan prosessoida pelkästään tarvittaessa pyydettävää sisältöä ja palvella osa pyynnöistä välimuistissa olevien olioiden avulla. Suorittamalla ennaltagenerointia vain ruuhkattomina ajankohtina, mahdollistetaan tehokas resurssienkulutus. Myös suorituskyvyn pullonkauloja voidaan siten helpottaa. Järjestelmän näkökulmasta on mahdollista saavuttaa suorituskykyhyötyä ennalta-

generoinnilla, mikäli ennaltagenerointi vähentää järjestelmän ylikuormitushetkille kohdistuvaa työtaakkaa. Joka tapauksessa ennaltagenerointi vie vähintään yhtä paljon kokonaisvaltaisia resursseja ja vaatii kaikesta huolimatta oheislaskentaa.

Mehrotran ym. (2010) mukaan olisi tarvetta palautejärjestelmän kehittämiselle, jossa web-palvelimelta pyydettäviä sivuja ennaltageneroitaisiin ja ennaltagenerointiprosessia hienosäädettäisiin vastaamaan järjestelmän sen hetkistä kuormitusta. Ravi ym. (2009) jatkavat kertoen, että vain vähän tutkimuksia kohdistuu välimuistitoteutusten itse itsensä hallitsemiseen siten, että välimuistitoteutus oppisi ja sopeutuisi käyttäjän käyttäytymismuutoksiin reaaliajassa. He pitävät tällaista tutkimusta erityisen tärkeänä suurten web-palveluiden kohdalla, joissa käyttäjäkanta on suuri ja ennustettavuus vaikeaa. Heidän mukaansa modernien web-sovellusten tyyliin kuuluu kuitenkin se, että käyttäjät on ryhmitelty tiettyihin rooleihin, jolloin ennustettavuutta voidaan parantaa hyödyntämällä tarjolla olevaa semanttista tietoa.

Wu ja Kshemkalyani (2004, luku 1) kuvailevat ennaltagenerointia näin:

> Ennaltageneroinnin tulee hankkia järjestelmän kaikkien mahdollisten välimuistiolioiden luonteenpiireiden perustiedot ja pitää etukäteen vain joidenkin tiettyjen välimuistiolioiden kopioita välimuistissa. Tämän valinnan perusteena ovat tyypillisesti pyyntötiheys, päivitysväli, koko, jne. Esimerkiksi nostaaksemme osumatarkkuutta, haluamme ennaltageneroida ne välimuistioliot, jotka ovat usein pyyntöjen kohteina. Minimoidaksemme kaistankulutuksen haluamme valita ne välimuistioliot, joilla on pisin päivitysväli ja joiden elinikä välimuistissa on siten pisin.

Probir ja Rau-Chaplin (2006) esittelevät tutkimuksessaan ratkaisua, jossa ennaltageneroidaan käyttäjälle todennäköisin seuraavaksi pyydettävä web-sivu. Ennustaminen tehdään tilastollisin menetelmin ja ennaltagenerointi tapahtuu sivupyyntöjen välisenä aikana, jolloin parhaassa tapauksessa ennaltagenerointi vähentää web-sovelluspalvelimen aiheuttaman viiveen nollaan. He kuvaavat ehdottamaansa käyttäjäprofiiliin perustuvaa ennustusmallia seuraavasti:

> Esittämässämme järjestelmässä web-palvelin pitää yllä muistissa lähiajan istuntotietoja käyttäjittäin. Lähetettäessä käyttäjälle vastaus sivupyyntöön, järjestelmä päättää tuleeko ennaltageneroida seuraava, mahdollisimman todennäköinen sivu vaiko ei. Tämän päätöksen järjestelmä tekee ennustamiseen liittyvän tiedon ja muiden järjestelmämuuttujien avulla. Tällaisia järjestelmämuuttujia ovat esimerkiksi: järjestelmän kuormitus, puskurin koko, käyttäjän sivupyynnön tyyppi, jne. (Probir & Rau-Chaplin, 2006, alaluku 2.1)

Tässä tutkimuksessa keskitytään samankaltaiseen tilastolliseen ennustettavuuteen mutta ennustettavien välimuistiolioiden raekoko on hienojakoisempaa verrattuna yllä esiteltyyn, koko sivua koskevaan ennustamiseen. Ennaltagenerointi tapahtuu tilastollisesti, seuraavassa luvussa esiteltävillä kustannusfunktioilla.

Pallis, Vakali ja Pokorny (2008) kirjoittavat, että mikäli välimuisti ja ennaltagenerointi on integroitu tehottomasti, tämä saattaa aiheuttaa kasvavaa verk-

koruuhkaa ja lisätä web-palvelimen kuormaa. Ennen kaikkea, tällöin välimuistin muistitilaa ei käytetä optimaalisesti. Täten onkin tärkeää, että ennaltageneroinnin lähestymistapa suunniteltaisiin huolellisesti, jotta esimerkiksi edellä mainituilta negatiivisilta vaikutuksilta vältyttäisiin. Myös Shi ym. (2005) pitävät välimuistin ja ennaltageneroinnin integroimista viisaana muistinkäytön näkökulmasta. Heidän mielestään myös ennaltageneroinnin tarkkuus eli ennaltageneroitavan olion valinta on tärkeää. He esittävät esimerkin välimuistimallista, jossa osumatarkkuus on huono: kun on paljon turhaa sisältöä ennaltageneroituna, järjestelmän resurssit kuluvat hukkaan. He jatkavat toisella ääriesimerkillä: välimuistin muistimäärän rajallisuus yhdistettynä liian intensiiviseen ennaltagenerointiin aikaansaa sen, että myös hyviä olioita korvataan välimuistista sen täyttymisen myötä.

Shi ym. (2005) esittävät yllä mainittuihin ongelmiin osittaiseksi ratkaisuksi ennaltageneroinnin kontrollointimekanismin. Tällainen mekanismi laskee ennaltageneroinnille kynnysarvon verkko- ja järjestelmäresurssien kuormituksen mukaan reaaliajassa. Mikäli järjestelmän tai verkon kuormitus on suuri, on kynnysarvo korkealla, jolloin ennaltageneroitavien olioiden määrä laskee. Ennaltageneroimalla oliot, joiden kustannusfunktion tuottama arvo on suurempi kuin kynnysarvo, aikaansaadaan tasapaino resurssien kuormituksen ja kulutuksen suhteen. He kertovat myös, että simulaatioista saadut tulokset näyttävät ennaltageneroinnin ja välimuistin kontrollointimekanismiin pohjautuvan vuorovaikutuksen kykenevän hallitsemaan ennaltagenerointia tehokkaasti ja voivan helpottaa verkon ja järjestelmän resurssien ylikuormitusta.

Toinen lähestymistapa kontrollointimekanismin toteuttamiseksi voisi olla web-sovelluspalvelimen ja taustajärjestelmän väliseen todelliseen tiedonsiirtokapasiteettiin pohjautuvan raja-arvon määrittäminen. Kullekin välimuistioliolle voitaisiin lisäksi laskea sen ennaltageneroinnista aiheutuva kaistankulutuksen tarve. Näiden arvojen avulla pystyttäisiin kontrolloimaan ennaltagenerointia huomioiden tiedonsiirron kapasiteetin rajoitteet.

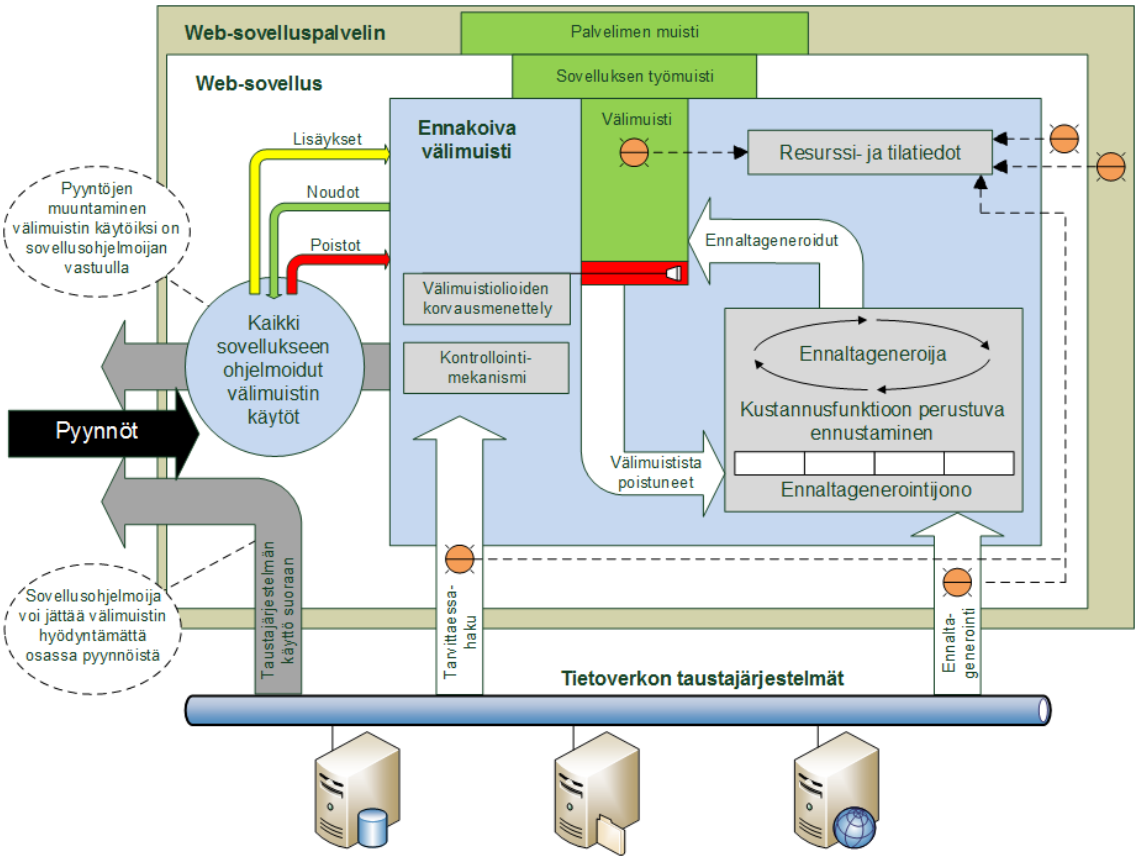
3.2 Ennakoivan välimuistin rakenne ja sijainti osana web-sovellusta

Tässä alaluvussa esitetään ennakoivan välimuistiratkaisun rakenne ja keskeiset komponentit sekä niiden nivoutuminen tyypilliseen web-sovelluksen arkkitehtuuriin.

Ennakoivan välimuistiratkaisun mallintamisessa ei oteta kantaa taustajärjestelmän tyyppiin. Taustajärjestelmä voi täten olla esimerkiksi tietokanta, tiedostojärjestelmä tai jopa REST API (engl. representational state transfer application programming interface) -rajapinta. Ennakoivan välimuistiratkaisun mallin taustaoletuksena on, että kaikki taustajärjestelmäpyynnöt ja niiden paluuarvot ovat sarjallistettavissa ja siten yhdenmukaisia, jolloin ennakoiva välimuistiratkaisu voidaan toteuttaa taustajärjestelmän tyypistä riippumatta.

Käyttäjältä saapuva sivupyyntö voi web-sovelluksen ohjelmoinnista riippuen aikaansaada suurenkin joukon taustajärjestelmäpyyntöjä yhteen tai useampaan taustajärjestelmään, mikäli pyydettävä sivu sisältää esimerkiksi kompleksista tietosisältöä. Tällaisessa tilanteessa, hyödynnettäessä web-sovelluksessa ennakoivaa välimuistiratkaisua, voidaan osa edellä mainituista taustajärjestelmäpyynnöistä palvella välimuistista ja osa sisällöstä joutua noutamaan taustajärjestelmästä asti.

Kuviossa (kuvio 2) nähdään web-sovelluksen looginen rakenne ja ennakoivan välimuistiratkaisun sijainti osana web-sovellusta. Kuviossa esitetty ennakoivan välimuistiratkaisun toimintamalli voidaan kuvata tässä alaluvussa esitettyjen askelten mukaisesti.



KUVIO 2 Ennakoivan välimuistin looginen rakenne ja sijainti osana web-sovellusta

- On olemassa web-sovelluspalvelin, jossa web-sovellus sijaitsee. Web-sovellusta voidaan käyttää HTTP-protokollan avulla ja web-sovellus käyttää web-sivujen sisällön tuottamiseen apunaan taustajärjestelmiä.
- Web-sovellukseen kohdistuu käyttäjiltä saapuvia pyyntöjä, jotka jakautuvat tietoa noutaviin ja tietoa päivittäviin pyyntöihin. Tietoa noutavat pyynnöt hyötyvät välimuistista. Tietoa päivittävät pyynnöt puolestaan aikaansaavat välimuistiolioiden poistoja ja lisäyksiä.

- Web-sovellukseen kohdistuvien tietoa päivittävien pyyntöjen on kuljettava välimuistiratkaisun kautta, jotta tiedetään invalidoida päivitykseen liittyvät välimuistioliot poistamalla ne välimuistista. Sama rajoitus pätee välimuistiratkaisun käyttöönoton jälkeen myös kaikkiin web-sovelluksen hyödyntämiin taustajärjestelmiin, joiden tietojen päivittäminen muuta kautta, kuin välimuistiratkaisun avulla aikaansaa välimuistissa olevan tiedon joutumisen epäeheään tilaan.
- Web-sovelluksen ohjelmakoodiin on sovellusohjelmoijan toimesta ohjelmoitu välimuistin käyttöjä, joita suoritetaan käyttäjiltä saapuvien pyyntöjen mukaisesti. Mahdolliset välimuistin käytöt ovat seuraavat: olioita lisätään välimuistiin, olioita noudetaan välimuistista tai olioita poistetaan välimuistista. Välimuistissa olevan olion päivittämistä tuoreemmalla versiolla ei katsota erillisenä käyttönä, sillä se on poiston ja lisäyksen yhdistelmä.
- Web-sovellukseen ohjelmoidut välimuistin käytöt ovat sovellusohjelmoijan vastuulla, jolloin välimuistin hyödyllisyys riippuu sen käyttöjen määristä sekä käyttöjen sijainneista ohjelmakoodissa. Välimuistin käytöllä kuvataan yhden yksilöllisellä tunnisteella eriteltävissä olevan loogisen välimuistiolion määrittäminen osaksi kaikkien mahdollisten välimuistiolioiden joukkoa. Samalla olio esitellään välimuistiratkaisun näkökulmasta mahdolliseksi ennaltageneroitavaksi välimuistiolioksi.
- Halutessaan sovellusohjelmoija voi jättää välimuistiratkaisun hyödyntämättä osassa web-sovelluksen toiminnallisuuksista. Tällöin pyynnöt suoritetaan välimuistiratkaisun ohi suoraan taustajärjestelmiin. Riskinä poisto-tyyppisten välimuistin käyttöjen toteuttamatta jättämisessä on kuitenkin se, että jonkin taustajärjestelmätiedon päivitysoperaation tapahtuessa välimuistiratkaisun ohi, voivat välimuistissa olevat tiedot joutua epäeheään tilaan.
- Web-sovelluksessa on olemassa työmuisti, jolle on ominaista tietyn muistimäärän varaaminen välimuistiolioiden tallettamista varten.
- Välimuistin täyttyessä välimuistiolioista, tulee välimuistissa olla korvausmenettely, jonka korvausmenettelyalgoritmin avulla poistetaan vähiten sovelluksen toimintaa hyödyttäviä välimuistiolioita välimuistista. Algoritmilla on käytössään ajon aikana välimuistiolioiden ominaisuuksien pohjalta laskettu raja-arvo tai jokin muu päättelysääntö, johon perustuen algoritmi kykenee erottelemaan välimuistin osumatarkkuuden kannalta edulliset ja epäedulliset välimuistioliot.
- Ohjelmakoodista tapahtuvan välimuistin käytön ollessa noutotyyppinen, haetaan välimuistilta tiettyä välimuistiavainta vastaavaa välimuistioliota. Välimuistiosuman tapauksessa, noudettava olio on jo välimuistissa, jolloin se voidaan toimittaa heti pyynnön suorittajalle. Mikäli oliota ei löydetä, joudutaan se noutamaan taustajärjes-

telmästä ja vasta sitten palauttamaan pyynnön suorittajalle. Tyypillisesti tällainen taustajärjestelmästä noudettu olio talletetaan välimuistiin jatkokäyttöä varten. Talletus välimuistiin tehdään vain, mikäli välimuistissa on tilaa ja lasketaan, että korvausmenettelyalgoritmin mukaan kyseisen välimuistiolion on hyödyllistä sijaita välimuistissa.

- Välimuistin rinnalla oleva ennaltageneroija voi oikein toimiessaan tehostaa pyydettävän sisällön löytymistä välimuistista. Jokaista tunnettua välimuistiolioita vastaa kunkin olion ominaisuuksia kuvaava tietue. Välimuistiolion vanhentuessa ja poistuessa välimuistista, kulkee tämä tietue aina ennaltageneroijan kautta. Ennaltageneroija arvioi jokaisen välimuistiolion hyvyyden kustannusfunktion avulla ja päättää, jätetäänkö välimuistiolion ominaisuuksia kuvaava tietue ennaltagenerointijonoon, vai poistetaanko välimuistiolio kokonaan.
- Ennaltageneroija käyttää apunaan web-sovelluspalvelimesta, web-sovelluksesta ja välimuistista kerättyjä resurssi- ja tilatietoja. Näiden tietojen lisäksi ennaltageneroija käyttää kustannusfunktiossaan tietoja, jotka on kerätty välimuistiolioita generoitaessa: joko tarvittaessa hakemalla tai aiemmilla ennaltagenerointikerroilla. Tyypillinen generoimiseen liittyvä kerättävä tieto on esimerkiksi välimuistiolion tuottamiseen käytetty aika.
- Ennaltageneroija tutkii ennaltagenerointijonon sisältöä tietyin väliajoin. Ennaltagenerointijonosta valitaan ennaltageneroitaviksi ne oliot, joiden katsotaan olevan parhaita kustannusfunktion perusteella. Ennaltageneroijan tutkiessa ennaltagenerointijonon sisältöä, voi jokin jo ennaltagenerointijonoon aiemmin valittu välimuistiolion ominaisuuksia kuvaava tietue joutua poistettavaksi, mikäli paremmuusjärjestyksessä, sen eteen on saapunut useita ennaltagenerointiin paremmin soveltuvia välimuistiolioita kuvaavia tietueita.
- Ennaltageneroijaan liittyy poikkeustilanne: välimuistioliota kuvaava tietue voi olla ennaltagenerointijonossa odottamassa ennaltagenerointivuoroaan, josta huolimatta tietueen kuvastama välimuistiolio saatetaan jonotuksen aikana generoida välimuistiin kohdistuneen pyynnön johdosta. Tällainen poikkeustilanne tulee huomioida välimuistioliota ennaltageneroitaessa, jottei tehdä turhaa kaksinkertaista työtä. Tarvittaessa tehtävän haun yhteydessä tulee myös tarkistaa, ettei vastaavan välimuistiolion ennaltagenerointi ole jo käynnissä, jolloin on viisaampaa odottaa sen valmistumista ylimääräisen kaksinkertaisen haun suorittamisen sijasta.
- Web-sovelluksen käytön aiheuttamien päivitysten, korvausmenettelyn sekä ennaltageneroijan yhdessä aikaansaama välimuistiolioiden ja niitä kuvaavien tietueiden liikehdintä välimuistin ja ennaltagenerointijonon välillä on jatkuvaa. Toteuttamalla tai konfiguroi-

malla välimuistin ja ennaltageneroinnin yhdistelmä huolimattomasti on mahdollista aikaansaada silmukka, jossa laskennallisten ominaisuuksiensa puolesta rajalla olevat välimuistioliot siirretään vuoron perään välimuistiin ja jälleen sieltä pois.

3.3 Ennakoivan välimuistin vaatimukset ja mallintaminen

Tässä alaluvussa esitetään vaatimukset, jotka tarkentavat ennakoivan välimuistiratkaisun mallia. Lisäksi esitetään mallin rajaukset sen ulkopuolelle jätettyjen vaatimusten osalta. Mallinnuksessa selvitetään myös, mitä resurssitietoja tarvitaan välimuistiratkaisun toimintaan ja eri kustannusfunktioita varten sekä mistä näitä tietoja mitataan. Esimerkiksi web-sovelluspalvelimelta saadaan muisti, prosessorikuorma ja levykäytön tiedot. Taustajärjestelmien kuormasta saadaan tietoa vain välillisesti: voidaan verrata sitä, kuinka nopeasti kyselyt palvellaan suhteessa niiden entisiin kestoihin. Jos kesto on suurempi kuin normaalisti, vähennetään ennaltagenerointia ja tuodaan siten helpotusta taustajärjestelmän kuormaan. Tämän alaluvun tarkoitus on toimia toteutuksen vaatimusmäärittelyvaiheen pohjana ja tuottaa välimuistitoteutuksen malli.

Seuraavassa taulukossa (taulukko 1) on listattu ennakoivan välimuistin vaatimukset, joiden avulla on mahdollista rakentaa ennakoivan välimuistiratkaisun toteutus. Taulukossa on ilmaistu kullakin rivillä rivikohtainen tunniste, jolla vaatimukseen voidaan jatkossa viitata.

TAULUKKO 1 Ennakoivan välimuistin vaatimukset

V1	Välimuistitoteutuksen tulee kyetä käsittelemään ohjelmalohkomuotoisia parametreja, jotta sillä on mahdollisuus uudelleengeneroida vanhentunutta välimuistisisältöä. Välimuistin käytöt ohjelmakoodissa tulee toteuttaa siten, että välimuistikäytön metodikutsun parametrina välitetään ohjelmalohko, joka sisältää riittävät tiedot välimuistisisällön uudelleen generoimiseksi.
V2	Välimuistitoteutuksen tulee kyetä vastaanottamaan parametrina myös vaatimuksen (V1) mukaiselle, parametrina välitettävälle ohjelmalohkolle tarkoitetut parametrit.
V3	Välimuistitoteutuksen tulee kyetä generoimaan parametrina annetun ohjelmalohkon avulla sitä vastaava välimuistisisältö irrallaan aiemmasta suorituskontekstista.
V4	Välimuistitoteutuksen tulee kyetä pitämään tallessa parametrina saatu ohjelmalohko, sen suorittamiseen vaaditut parametrit sekä ohjelmalohkon avulla generoitu varsinainen välimuistisisältö.
V5	Välimuistitoteutuksen tulee ottaa parametrina välimuistiavain, jolla ohjelmakoodista tapahtuvat välimuistiratkaisun eri käytöt voidaan yksilöidä.
V6	Välimuistitoteutuksen tulee kerätä ajoympäristöstä resurssi- ja tilatietoja ennaltagenerointiprosessin tarpeisiin. Osa tilatiedoista voidaan kerätä esimerkiksi tietyin väliajoin, osa tilatiedoista voidaan saada välimuistitoteutuksen muiden toimintojen tuottamana ja loput tilatiedot hakea reaaliajassa niitä pyydettäessä.
V7	Välimuistitoteutuksen tulee kerätä ja ylläpitää välimuistin sisällöstä oliokohtaista tilatietoa kustannusfunktioon perustuvan ennaltageneroinnin toteuttamiseksi.

(jatkuu)

Taulukko 1 (jatkuu)

V8	Välimuistitoteutuksen tulee mitata välimuistisisällön generoimiseen käytettyä aikaa oliokohtaisesti, jotta työmäärään perustuva kustannusfunktio voidaan toteuttaa. Sama vaatimus koskee myös muiden kustannusfunktioiden vaatimia oliokohtaisesti mitattavia arvoja. Mitatut arvot tulee olla kustannusfunktioiden hyödynnettävissä liukuvalta tarkasteluväliltä keskimääräisinä arvoina, esimerkiksi: viimeisen vuorokauden keskimääräinen välimuistioliokohtainen elinikä.
V9	Välimuistitoteutuksen kontrollointimekanismin tulee mahdollistaa käytetystä kustannusfunktiosta riippumatta seuraavanlainen toiminta: mikäli olion generoimiseen käytettävä työmäärä on pieni, tulee välimuistitoteutuksen päätellä kannattaako oliota tallettaa tilan säästämiseksi välimuistiin ollenkaan.
V10	Välimuistitoteutuksen kontrollointimekanismin tulee mahdollistaa toiminta, jossa: mikäli vain tarvittaessa pyydettäessä generoitu välimuistiolio on kustannusfunktion mukaan hyvä ja resursseja esimerkiksi vapaata tilaa on välimuistissa riittävästi, talletetaan välimuistiolio välimuistiin.
V11	Välimuistitoteutuksen kontrollointimekanismin tulee toimia siten, että välimuistiolioita ennaltageneroidaan vain jos resursseja on vapaana. Ennaltagenerointi lopetetaan kokonaan jonkin tietyn ennaltageneroinnin ylärajan eli resurssienkäytön prosentin saavuttamisen jälkeen. Tiedonsiirtoverkon kuormitusta on mahdollista rajoittaa asettamalla kontrollointimekanismiin todellista siirtokapasiteettia koskeva raja-arvo ja ennaltageneroida välimuistisisältöä tämän raja-arvon puitteissa, välimuistioliokohtaisten keskimääräisen kaistankäytön tietojen mukaisesti. Resurssien kulutuksessa tulee ottaa mahdollisuuksien mukaan huomioon käytössä olevan laitteiston ja siirtokapasiteetin resurssien lisäksi myös taustajärjestelmien resurssitilanne. Taustajärjestelmien kuormitusta on mahdollista arvioida vertaamalla olioiden keskimääräisiä generointiviiveitä nykyhetken vastaaviin.
V12	Välimuistissa on oltava mekanismi, jotta ennaltagenerointipyynnöt saapuvat rauhallisesti ja hallitusti, eivätkä kaikki kerralla.
V13	Välimuistiratkaisun tulee huolehtia siitä, että ennaltagenerointi sujuu ilman törmäyksiä eri kuluttajien ja tuottajien välillä. Kuluttajia ovat kaikki välimuistiin kohdistuvat pyynnöt sekä ennaltageneroija. Myös ennaltagenerointijonossa olevat välimuistiolioita kuvaavat tietueet tulee poistaa, mikäli ne generoidaan jo toisaalla välimuistiin kohdistuneen pyynnön johdosta. Vastaavasti kuluttajien tulee odottaa välimuistiolion valmistumista, mikäli ennaltagenerointi on jo alkanut.
V14	Välimuistiratkaisun tulee pitää yllä statistiikkatietoa niistäkin välimuistiolioista, jotka eivät tällä hetkellä ole mahtuneet välimuistiin tai eivät jostain muusta syystä ole siellä tällä hetkellä. Välimuististatistiikka tulee ylläpitää kaikille välimuistiolioille, jotka ovat joskus käyneet välimuistissa. Tämä välimuistissa käyneiden välimuistiolioiden joukko muodostaa seuraavassa luvussa esiteltävien kustannusfunktioiden laskukaavoissa tarkasteltavan joukon S. Joukon alkiolla ei tarkoiteta esimerkiksi yhden tietyn välimuistiolion tuoreinta versiota, vaan välimuistiavaimen määrittelemää versioriippumatonta loogista välimuistioliota. Riippuen ajanhetkestä, tätä loogista välimuistioliota vastaa välimuistiolion eri versio.
V15	Kerättävä statistiikkatieto tulee tallettaa välimuistiratkaisuun siten, että sen käyttö on nopeaa ja siten, että sen käyttämä tila on pieni. Välimuistiratkaisun tuottama lisäkuormitus ei saa kumota välimuistin suorituskykyhyötyjä tavoittelevan luonteen periaatetta.

(jatkuu)

Taulukko 1 (jatkuu)

V16	Välimuistitoteutuksen tulee oppia käyttäjien käyttäytymismalleja, jotta uusi web-sivulle ilmestyvä sisältö otetaan huomioon sisällön ennaltageneroinnissa ja vanha sisältö unohdetaan vähitellen. Jotta välimuistitoteutus oppisi web-sovelluksen käyttäjäjoukon käyttäytymistä, tulee tilatietojen osalta ylläpitää vain uusimpia arvoja ja poistaa liukuvasti vanhoja uusien tilatietojen tieltä. Välimuistiratkaisussa käytettävät algoritmit tulee toteuttaa siten, että ne tarkastelevat äärellistä ajanjaksoa historiasta nykyhetkeen. Tällainen liukuva statistiikkatiedon mittausajanjakso voisi olla tunteja tai vuorokausia. Täten esimerkiksi välimuistiolioiden ominaisuudet, kuten keskimääräinen elinikä ja pyydetyksi tulemisen todennäköisyys laskettaisiin liukuvan tarkasteluvälin ajalta. Tähän liittyy oleellisesti myös kustannusfunktioiden laskennassa tarkasteltavan kaikkien mahdollisten välimuistiolioiden joukon rajaaminen liukuvasti mittausajanjakson mukaan. Tällöin, kauan sitten käsitellyt välimuistioliot jätetään tarkasteltavan joukon ulkopuolelle, kunnes käyttäjältä saapuvat kyseisiin olioihin kohdistuvat pyynnöt jälleen aikaansaavat näiden välimuistiolioiden huomioimisen osaksi tarkasteltavaa joukkoa.
V17	Välimuistitoteutuksessa tulee olla korvausmenettely, joka vastaa välimuistitilan optimaalisesta käytöstä. Samalla välimuistitoteutuksen suunnittelussa tulee huolehtia, että korvausmenettely toimii yhtenevästi ennaltagenerointiin käytetyn kustannusfunktion kanssa. Yhteensopiva ennaltagenerointi tuottaa korvausmenettelyn näkökulmasta arvokasta välimuistisisältöä välimuistiin. Tällöin varmistutaan, ettei ennaltageneroinnin kustannusfunktion mukaan arvokkaaksi määriteltyä välimuistisisältöä poisteta korvausmenettelyn toimesta.

Yllä listatut vaatimukset on kuvattu käsitteellisellä tasolla teknologiariippumattomasti. Täten vaatimuslistauksessa jätetään määrittelemättä esimerkiksi missä vaatimuksen V4 tiedot tulisi säilyttää tai miten vaatimuksen V13 kannalta oleellinen pyyntöjen atomisuus saavutettaisiin.

Listattujen vaatimusten lisäksi on muita tämän tutkimuksen ulkopuolelle rajattuja vaatimuksia. Tällaisiksi vaatimuksiksi voidaan nähdä kuuluvan esimerkiksi tarve kerätyn statistiikkatiedon säilymiselle palvelimen uudelleenkäynnistysten ylitse. Tämän lisäksi voisi olla toivottavaa, että web-sovellus aloittaisi välimuistisisällön ennaltageneroinnin välittömästi tallessa olevan statistiikkatiedon pohjalta uudelleenkäynnistyksen jälkeen. Toisaalta vaatimuksena voisi olla myös välimuistisisällön eheyttä tavoitteleva ominaisuus, jonka ansiosta välimuistista palautettava olio olisi aina syväkopio välimuistisisällöstä, jolloin välimuistissa sijaitseva välimuistiolio ei koskaan pääsisi web-sovellusohjelmoijan virheistä johtuen epäeheään tilaan.

Yllä esitetyt ja tutkimuksen ulkopuolelle rajatut vaatimukset olisivat oleellisia, mikäli tämän tutkielman välimuistiratkaisun malliin pohjautuva toteutus otettaisiin käyttöön todellisessa web-sovelluspalvelimessa.

3.4 Luvun yhteenveto

Tässä luvussa esiteltiin ennakoivan välimuistin ennaltageneroinnin liittyvää aihepiiriä sekä listattiin välimuistiratkaisun mallin vaatimukset, joita noudattamalla on mahdollista rakentaa ennaltageneroiva ennakoiva välimuistitoteutus. Seuraavassa luvussa tarkastellaan ennaltageneroitavien olioiden valintaan tähtääviä algoritmeja ja esitellään uusi, osumatarkkuutta ja työmäärää tavoitteleva ahne kustannusfunktio.

4 ENNALTAGENEROITAVIEN OLIOIDEN VALINTA

Tässä luvussa tutustutaan ennakointialgoritmeihin joilla välimuistiolioita on mahdollista valita ennaltageneroitavaksi välimuistiin. Lisäksi esitellään uusi välimuistiolion generointiin käytetyn työmäärän huomioiva kustannusfunktio. Tämä luku luo pohjan tutkimuksen simulaatioille, joissa mitataan eri kustannusfunktioiden suoriutumista toisiinsa nähden.

Seuraavassa alaluvussa tutustutaan historiatietoon pohjautuvaan ennakointiin, jossa pyritään yhden tai useamman käyttäjän aiempien sivupyyntöjen tai niiden järjestyksen perusteella päättelemään todennäköisimpiä seuraavia sivupyyntöjä ja ennaltageneroimaan välimuistisisältöä.

4.1 Historiatietoon perustuva ennakointi

Alin ym. (2011) mukaan Zhijie, Zhimin ja Yu (2009) kertovat historiaan pohjautuvan ennaltageneroinnin olevan oma ennaltageneroinnin kategoria, jossa tulevia käyttäjän pyyntöjä ennustetaan aiemmin havaitun käyttäytymisen pohjalta. Historiatietoon pohjautuva ennakointi voidaan jakaa neljään alikategoriaan:

- Markovin malliin pohjautuviin,
- riippuvuusgraafeja käyttäviin,
- tiedonlouhintaan pohjautuviin sekä
- kustannusfunktioihin perustuviin lähestymistapoihin.

Seuraavissa alaluvuissa on esitettynä kunkin lähestymistavan ominaispiirteet.

4.1.1 Markovin malliin perustuva ennustaminen

Ali ym. (2011) pitävät Markovin malliin pohjautuvan ennustamisen olevan tehokas keino käyttäjän nykyisen ja aikaisemman vierailtujen sivujen sekvenssien yhteen täsmäykselle. Mallin toiminta voidaan kuvata seuraavasti:

> Oletetaan, että on peräkkäisten vierailtujen sivujen sekvenssi $x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_{k-1}, k \geq 2$. Ensimmäisen asteen Markovin malliin pohjautuvassa lähestymisessä seuraava tapahtuma x_k riippuu vain nykyisestä ketjun tilanteesta x_{k-1}. Mikäli tilanteen x_k toteutuminen riippuu viimeisimmistä kahdesta tapahtuneesta tilanteesta $x_{k-2} \rightarrow x_{k-1}$, on kyseessä toisen asteen Markovin malli. Yleisesti, mikäli x_{k+1} riippuu aiemmin tapahtuneista sivukäynneistä $x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_k$, kutsutaan kyseistä mallia k:nnen asteen Markovin malliksi. (Ali ym., 2011, s. 31)

Asteluvultaan pienissä Markovin malleissa ennustus ei yleensä ole kovin tarkkaa, sillä ne eivät katso riittävän kauas aiempaan käyttäjän selaushistoriaan. Seurauksena tästä, asteluvultaan suuret Markovin mallit saavat paremman ennustustarkkuuden, vaikkakin pienemmän kattavuuden kustannuksella. (Nigam & Jain, 2010.) Pienellä kattavuudella tarkoitetaan sitä, että ketjun pituuden kasvaessa voidaan löytää vähemmän hyödyllisiä osumia käyttäjän aiemmasta selauskäyttäytymisestä.

4.1.2 Riippuvuusgraafialgoritmiin perustuva ennustaminen

Alin ym. (2011) mukaan ensimmäisenä web-ennaltageneroinnin riippuvuusgraafialgoritmia käyttivät Padmanabhan ja Mogul (1996). Heidän lähestymistavassaan riippuvuusgraafi rakennetaan seuraavan sivun ennustamista varten. Graafi koostuu solmuista, jotka kuvaavat web-sivuja ja kaarista, jotka kuvaavat mahdollisia siirtymiä sivujen välillä. Jokaiselle kaarelle on painoarvo, joka esittää todennäköisyyttä sille, onko kyseinen kaaren päässä oleva sivu seuraava pyydetty sivu. Riippuvuusgraafiin pohjautuvassa ennaltageneroinnissa haetaan kaikki seuraavat mahdolliset siirrot, joiden todennäköisyys ylittää tietyn kynnysarvon. Ali ym. (2011) kertovat kuitenkin, että tämän ennaltagenerointitavan kustannuksena on kasvava verkon kaistankäyttö. He kertovat myös, että Nanopoulos, Katsaros ja Manolopoulos (2003) esittävät lähestymistavalle huonoksi puoleksi sen alhaisen ennustustarkkuuden, sillä tarkastelun kohteena on vain yhden kaaren pituinen siirtymä. Esimerkiksi aiemmin esitellyn Markovin mallin tapauksessa, riippuvuusgraafin yhden kaaren tarkastelu vastaisi vain ensimmäisen asteen Markovin mallia.

4.1.3 Tiedonlouhintaan perustuva ennustaminen

Tiedonlouhintaan pohjautuvat ennustusmenetelmät ovat ennaltageneroinnin yleisimpiä tekniikoita (Ali ym., 2011). Tiedonlouhintaan pohjautuvat menetelmät voidaan luokitella assosiaatiosääntöpohjaiseen ja klusterointipohjaiseen lähestymistapaan.

Assosiaatiopohjainen tapa ryhmittelee sivut homogeenisiin ryhmiin tiettyjen sivujen, peräkkäin tapahtuvien vierailujen mukaan, yhden käyttäjän istunnon sisällä. Käyttäjän istunto määritellään sivuvierailusekvenssinä yhdessä web-palvelussa. Assosiaatiosäännöissä kannatus ja varmuus ovat kaksi tärkeää assosiaatiosäännön vahvuuden mittaria. Kannatus määräytyy toistuvien sivu-

jen löytymisen perusteella ja varmuus puolestaan määräytyy näiltä toistuvilta sivuilta löytyvien assosiaatiosääntöjen mukaan. (Ali ym., 2011.)

Klusterointipohjainen menetelmä on kehitetty tiedon puolesta yhtenäisten ryhmien eli klustereiden aikaansaamiseksi. Klustereille ominaista on se, että tietoilmentymät yhden klusterin sisällä ovat samankaltaisia mutta klustereiden välisten tietoilmentymien erot voivat olla suuria. Menetelmä aikaansaa sen, että klusterinsisäinen välimatka on lyhyt ja klustereiden välinen välimatka puolestaan suuri. (Ali ym., 2011.)

4.1.4 Kustannusfunktioon perustuva ennustaminen

Kustannuksiin perustuva lähestymistapa toimii siten, että joidenkin laskennallisten tekijöiden perusteella päätetään välimuistiolion ennaltageneroinnista. Kustannusfunktio voi toimia laskukaavana eli osana ennaltegenerointialgoritmia tai vastaavasti korvausmenettelyn algoritmin osana. Tällaisia kustannuksiin perustuvia kustannusfunktioita ovat esimerkiksi välimuistissa olevan välimuistiolion suosioon ja elinikään pohjautuvat funktiot sekä muut, joissa pyritään mahdollisimman suureen hyötyyn hyväksyttävissä rajoissa olevilla oheislaskennan, ylimääräisen resurssienkulutuksen tai kaistankäytön kustannuksilla. (Ali ym., 2011.)

Seuraavassa alaluvussa käsitellään tarkemmin olemassa olevia kustannusfunktioita ja esitellään oma työmäärään pohjautuva kustannusfunktio, jonka hyödyllisyyttä tutkielman edetessä selvitetään.

4.2 Ennaltageneroinnissa käytettävät kustannusfunktiot

Kustannusfunktioille tyypillistä on se, että ne käyttävät apunaan välimuistiolioiden ominaistietoja ja niiden aiempaa käyttöhistoriaa. Tällaisia tietoja voivat olla esimerkiksi välimuistioliokohtainen keskimääräinen elinikä välimuistissa, pyyntötiheys välimuistista sekä tässä tutkielmassa esiteltävä välimuistiolion tuottamiseen käytetty työmäärä.

Ali ym. (2011) kirjoittavat, että jo kyseisenä ajankohtana kirjallisuudessa oli esitetty lukuisia kustannusfunktioihin pohjautuvia ennaltagenerointitekniikoita. Heidän mukaansa kustannusfunktioita hyödyntävät ennaltagenerointitekniikat voidaan jakaa seuraavasti:

- Suosioon (engl. popularity) perustuva ennustaminen. Suosiolla tarkoitetaan olion i viittaustodennäköisyyttä eli olion i todennäköisyyttä p_i tulla pyydetyksi. Suosio katsotaan eduksi, jolloin valitaan n kappaletta suosituinta ehdolla olevaa vaihtoehtoa. Välimuistiolion suosio mitataan siihen kohdistuneiden pyyntöjen määrästä suhteessa kaikkiin web-sovellukseen kohdistuviin pyyntöihin.

- Elinikään (engl. lifetime) perustuva ennustaminen. Välimuistiolion pitkä elinikä katsotaan eduksi, jolloin valitaan n kappaletta pitkäikäisintä ehdolla olevaa vaihtoehtoa. Välimuistiolion elinikä riippuu välimuistiolion päivitystiheydestä. Mikäli välimuistiolioon kohdistuu usein päivitysoperaatioita, on se välimuistissa olon eliniältään lyhytikäinen.
- Hyvään osumaan (engl. good fetch) perustuva ennustaminen. Pitkä elinikä sekä suosio ovat eduksi, jolloin valitaan n kappaletta vaihtoehtoa, joilla on näihin tekijöihin pohjautuen suurin todennäköisyys olla parhaita vaihtoehtoja.
- Tavoitepohjainen ahne (engl. objective-greedy) ennustaminen. Yhden tai useamman välimuistioliokohtaisen ominaisuuden yhdistelmä katsotaan tavoittelemisen arvoiseksi. Tavoitepohjaisessa ahneessa ennaltageneroinnissa valitaankin n kappaletta vaihtoehtoja, jotka antavat parhaan tuloksen kohti yhtä tai yhdistettyä tavoitetta. Esimerkiksi tavoitteeksi voitaisiin valita välimuistin osumatarkkuuden nostaminen, jolloin tähän tavoitteeseen pääsemiseksi pyrittäisiin täyttämään välimuistia niillä olioilla, jotka eniten nostavat välimuistin kokonaisosumatarkkuutta.

Alla on esitetty taulukoituna (taulukko 2) eri kustannusfunktioiden suorituskykyvertailu välimuistin osumatarkkuuden sekä kaistankäytön suhteen. Taulukon numerot kuvaavat paremmuusjärjestystä, jolloin pienempi numero esittää parempaa sijoitusta. Taulukko on osittainen poiminta alkuperäislähteen vertailutaulukosta (Wu & Kshemkalyani, 2006, s. 31).

TAULUKKO 2 Kustannusfunktioiden vertailu (Wu & Kshemkalyani, 2006, s. 31)

Kustannusfunktio	**Välimuistin osumatarkkuus**	**Kaistankäytön säästö**
Suosioon perustuva ennustaminen	2	4
Elinikään perustuva ennustaminen	5	2
Hyvään osumaan perustuva ennustaminen	3	3
Tavoitepohjainen ahne osumatarkkuutta tavoitteleva ennustaminen	1	5
Tavoitepohjainen ahne kaistankäytön säästöä tavoitteleva ennustaminen	4	1

Seuraavissa alaluvuissa esitellään myöhemmin käytettävien matemaattisten muuttujien taustaa sekä tutustutaan tarkemmalla tasolla aiemmissa tutkimuksissa esiteltyjen kustannusfunktioiden toimintaperiaatteisiin. Lisäksi esitetään oma työmäärään ja osumatarkkuuteen perustuva kustannusfunktio. Vertaamalla ehdotettua kustannusfunktiota jo olemassa oleviin ennaltageneroinnin valintamekanismeihin on tarkoitus löytää hyvä malli välimuistin ja ennaltageneroinnin yhdistelmälle, jotta käyttäjän kokemaa viivettä saataisiin vähennettyä ja pyyntöjen välisiä viive-eroja tasoitettua.

4.2.1 Kustannusfunktioiden muuttujat

Taulukossa (taulukko 3) on listattuna kustannusfunktioiden jatkossa käyttävät muuttujat. Esitellyt muuttujat pohjautuvat aiempaan kustannusfunktioiden tutkimukseen tai ovat johdettuja aiempien tutkimusten muuttujien pohjalta. Oheisen taulukon muuttujille ja siten myös niitä hyödyntäville laskentakaavoille on tarpeen määrittää äärellinen tarkasteluväli. Taulukon muuttujat kuvaavat mitattuun historiatietoon pohjautuvia välimuistiolioiden sekä systeemin ominaisuuksia menneestä ajanhetkestä t_{k-n} nykyhetkeen t_k. Tämän tutkimuksen alaluvun 3.3 vaatimuksen V16 mukaan, tällainen äärellinen liukuva mittausajanjakso voisi olla tunteja tai vuorokausia. Mitattaessa kaikki seuraavan taulukon muuttujat liukuvalla tarkasteluvälillä $[t_{k-n},\ t_k]$, $k = nykyhetki$, $n = vuorokausi$, saadaan esimerkiksi muuttujan a arvoksi: ”kuluvan vuorokauden kokonaisvaltainen välimuistin käytön pyyntötiheys, pyyntöä sekunnissa”.

TAULUKKO 3 Kustannusfunktioiden muuttujat

Muuttuja	**Kuvaus**
a	Kokonaisvaltainen välimuistin käytön pyyntötiheys, pyyntöä sekunnissa.
p_i	Olion i todennäköisyys tulla pyydetyksi. Simulaatioissa nämä oliokohtaiset todennäköisyydet noudattavat zipfiläistä jakaumaa. $\mathrm{P}(\text{"}olioon\ i\ viitataan\ aikavälill\text{ä}\ [t_{k-n},\ t_k]\text{"})$
l_i	Olion i keskimääräinen elinikä sekunteina.
s_i	Olion i koko tavuina.
h_i	Olion i todennäköisyys olla välimuistissa olioon i kohdistuvan pyynnön saapuessa. $\mathrm{P}(\text{"}olioon\ i\ kohdistuva\ pyynt\text{ö}\ palvellaan\ v\text{ä}limuistista\ aikav\text{ä}lill\text{ä}\ [t_{k-n},\ t_k]\text{"})$
b_i	Oliosta i aiheutuva kaistankulutus, tavua sekunnissa.
t_i	Olion i generointiin kuluva keskimääräinen aika sekunteina.
w_i	Oliosta i aiheutuva työmäärä. Muuttujan laatu on sekuntia sekunnissa, sillä työmäärä kuvaa muuttujien t_i ja l_i suhdetta.
S	Kaikkien systeemissä sen elinaikana viitattavissa olevien välimuistiolioiden joukko, jonka alkiot voivat tulla tallennetuiksi välimuistiin jossain vaiheessa systeemin elinkaarta. Joukkoon S kuuluu kustakin yksilöitävissä olevasta välimuistioliosta yksi versioriippumaton looginen välimuistiolio. Mikäli joukon S olion i katsotaan olevan välimuistissa, vastaa tällaista loogista oliota i välimuistissa välimuistiolion i tuorein versio.

Jiang ym. (2002) kertovat useiden tutkijoiden todenneen käyttäjien sivupyyntöjen perusteella määritetyn jakauman noudattavan zipfiläistä jakaumaa melko hyvin. Wu ja Kshemkalyani (2004) toteavat siten olion i todennäköisyyden tulla pyydetyksi mukailevan karkeasti zipfiläistä jakaumaa. Kaikkien mahdollisten ennaltageneroitavien välimuistiolioiden joukossa olevan i:nneksi suosituimman olion p_i-arvo voidaan siten määrittää zipfiläisen jakauman perusteella. Olion i todennäköisyys tulla pyydetyksi voidaan siten laskea kaavan (2) mukaisesti yleistetyn zipfiläisen jakauman avulla (Bestavros, Cunha & Crovella, 1995, s. 10).

$$p_i = \frac{1}{\left(\sum_i \frac{1}{i^\alpha}\right) i^\alpha} \quad (2)$$

Kaavan 2 indeksi i kuvaa järjestysnumeroa välillä $1..S_{lkm}$. Joukko S esittää kaikkien viitattavissa olevien välimuistiolioiden joukkoa, joista tyypillisesti vain osa mahtuu välimuistiin. Mitä pienemmällä indeksillä i olion p_i-arvo lasketaan, sitä suurempi zipfiläisen jakauman mukainen olion i todennäköisyys tulla pyydetyksi saadaan.

Muuttujan α arvo vaihtelee tutkimuksittain. Wun ja Kshemkalyanin (2004) mukaan Bestavros, Cunha ja Crovella (1995) ehdottavat, että $\alpha = 0{,}986$. Nishikawa, Hosokawa, Mori, Yoshidab ja Tsujia (1998) puolestaan ehdottavat arvoa $\alpha = 0{,}75$. Jälkimmäinen ehdotetuista arvoista pohjautuu kaksi miljoonaa web-palvelinpyyntöä sisältävän lokitiedoston analyysiin ja on saanut tiedeyhteisössä enemmän kannatusta.

4.2.2 Suosioon perustuva ennustaminen

Suosioon perustuva ennustaminen pohjautuu Markatosin ja Chronakin (1998) tekemään tutkimukseen, jossa ehdotettiin kymmenen suosituimman olion ennaltageneroimista välimuistiin. Suosiolla tarkoitetaan olion viittaustodennäköisyyttä eli olion todennäköisyyttä p_i tulla pyydetyksi. Wun ja Kshemkalyanin (2004) mukaan tässä tekniikassa web-sovelluspalvelin pitää yllä tiedon kaikista mahdollisista välimuistiin talletettavissa olevista olioista. Aina kun jokin kymmenestä suosituimmasta oliosta muuttuu sisältönsä puolesta, viedään muutokset välimuistiin. Tämä mahdollistaa kymmenen suosituimman olion tuoreuden välimuistissa. Esitellystä kymmenen suosituimman välimuistiolion versiosta voidaan helposti tehdä n:nnen suosituimman olion versio, jossa muuttuja n voi muuttua esimerkiksi ympäristön tai verkon kuormituksen mukaan. Toinen vaihtoehto on ennaltageneroida kaikki suosioltaan tietyn kynnysarvon ylittävät oliot välimuistiin.

Wu ja Kshemkalyani (2004, alaluku 2.2) kertovat Jiangin ym. (2002) kuvailevan suosioon perustuvan ennustamisen luonnetta seuraavasti: ”Koska suosituimpiin olioihin kohdistuu enemmän pyyntöjä kuin vähemmän suosittuihin, suosioon pohjautuvan ennustamisen oletetaan aikaansaavan paras välimuistin käytön osumatarkkuus”. Edellisen sivun taulukosta 2 nähtiin, että suosioon perustuvan ennustamisen välimuistin osumatarkkuus on hyvä, mutta ei paras mahdollinen. Vastaavasti, samasta taulukosta nähdään tälle lähestymistavalle olennainen korkea kaistankulutus, sillä kustannusfunktio ottaa huomioon vain olion suosion, eikä välitä muista tekijöistä.

4.2.3 Elinikään perustuva ennustaminen

Wu ja Kshemkalyani (2004) toteavat, että ennaltagenerointi aiheuttaa aina ylimääräistä kaistankulutusta, sillä välimuistiolioiden tuoreena pitäminen vaatii uudelleengenerointia ja siihen liittyvää verkon sekä alkuperäisten web-sovelluspalvelinten kuormitusta. Jiang ym. (2002) esittivät ensikertaa elinikään perustuvan ennustamisen. Toisin kuin yllä esitelty suosioon pohjautuva ennustaminen, tämä tekniikka pyrkii valitsemaan pisimpään välimuistissa olleita olioita. Valinnalla pyritään minimoimaan kaistankäyttö. Aiemmin esitetyssä taulukossa 2 nähdään, että kyseisellä tekniikalla saadaan toiseksi paras kaistankäytön säästö mutta vastaavasti välimuistin käytön osumatarkkuus jää alhaisimmaksi.

4.2.4 Hyvään osumaan perustuva ennustaminen

Hyvään osumaan perustuva ennustaminen tähtää mahdollisimman hyvään ennustustulokseen. Hyvä osuma mitataan todennäköisyydellä, joka määräytyy olion päivitystiheyden sekä suosion mukaan. Mitä korkeampi suosio ja matalampi päivitystiheys oliolla ovat, sitä parempana olio nähdään, jolloin se kannattaa ennaltageneroida välimuistiin. Jiangin ym. (2002) mukaan tämä lähestymistapa pyrkii tasapainottelemaan olioiden suosion ja päivitystiheyden välillä ja siten aikaansaamaan hyvän välimuistin käytön osumatarkkuuden kohtuullisella kaistankulutuksella.

Wu ja Kshemkalyani (2004) kertovat kustannusfunktion rakenteesta. Oletetaan, että kokonaisvaltainen välimuistin käytön pyyntötiheys on a. Olion i pyydetyksi tulemisen todennäköisyys on p_i. Välimuistiolion keskimääräinen elinikä on l_i. Todennäköisyys, jolla olio i tulee käyttöön elinikänsä aikana voidaan esittää kaavan (3) avulla (Wu & Kshemkalyani, 2004, alaluku 2.2).

$$P_{hyvä\_osuma}(i) = 1 - (1 - p_i)^{al_i} \quad (3)$$

Yhtälön 3 muuttujat l_i ja a ovat tarpeen hyvän osuman laskemisessa. Olion keskimääräinen elinikä vaikuttaa oleellisesti siihen, kannattaako sitä ennaltageneroida. Esimerkiksi jos olion i sisältö muuttuu useammin kuin oliota käytetään, on olion ennaltageneroiminen toisinaan turhaa vaikka oliolla olisikin suhteellisen korkea pyydetyksi tulemisen todennäköisyys p_i. Vastaavasti kokonaisvaltainen välimuistin käytön pyyntötiheys a vaikuttaa yhtälön 3 eksponentissa keskimääräiselle eliniälle annettavaan painoarvoon.

Hyvään osumaan perustuva ennustaminen käyttää yhtälössä 3 laskettua todennäköisyyttä hyväksi ja noutaa siten kaikki ne oliot välimuistiin, joilla todennäköisyys ylittää määritetyn kynnysarvon. Nishikawa ym. (1998) väittävät hyvään osumaan pohjautuvan ennustamisen lähestyvän optimaalista algoritmia. Wu ja Kshemkalyani (2004) kuitenkin toteavat algoritmin voivan käyttäytyä tehottomasti joissain haku-päivitys-tilanteissa.

Wun ja Kshemkalyanin (2004) mukaan Jiang ym. (2002) esittävät toisen samankaltaista lähestymistapaa käyttävän algoritmin nimeltään APL-algoritmi, joka perustuu samoihin yhtälössä 3 esitettyihin muuttujiin a, p ja l. APL-algoritmissa laskenta tapahtuu käyttämällä valintakriteerinä muuttujien tuloa $ap_i l_i$. Valinta tapahtuu siten, että $ap_i l_i$-arvoltaan tietyn raja-arvon ylittävät oliot valitaan ennaltageneroitaviksi. Oliolle i laskettu $ap_i l_i$-arvo tarkoittaa kyseisen olion mahdollisia käyttökertoja sen eliniän aikana.

Wu ja Kshemkalyani (2004) todistavat tutkimuksessaan edellä esitettyjen algoritmien ekvivalenssin ja toteavat, että molemmat tavat käyttäytyvät samoin ennaltagenerointiin valittavien olioiden valinnassa. He toteavat myös, että täten voidaan päätellä molempien tässä alaluvussa esitettyjen algoritmien, kaavan 3 laskentatavan sekä APL-algoritmin järjestävän mahdollisten olioiden joukon samaan järjestykseen. Järjestyksen ollessa sama, ennaltagenerointiin valittava joukko on molemmilla tavoilla aina sama.

4.2.5 Osumatarkkuutta tavoitteleva ahne ennustaminen

Wu ja Kshemkalyani (2004) esittivät ensi kertaa tavoitepohjaisiin ahneisiin algoritmeihin pohjautuvan ennaltageneroinnin, joissa tähdätään jonkin tietyn tavoitteena olevan tekijän maksimointiin. Osumatarkkuutta (OT) tavoitteleva ahne (engl. hit rate greedy) ennustaminen kuuluu tällaisiin tavoitepohjaisiin ahneisiin kustannusfunktioihin. Ahneen kustannusfunktion tavoitteena on mahdollisimman suuri osumamäärä, jolloin myös välimuistin voidaan katsoa olevan mahdollisimman hyvin hyödynnetty. Suuren osumatarkkuuden haittapuolena on kuitenkin runsas resurssien kulutus, jolloin kustannusfunktion hyvyyttä voidaan kritisoida.

Wu ja Kshemkalyani (2004) tutkivat välimuistin osumatarkkuutta sekä kaistankäytön säästöä tavoittelevien ahneiden kustannusfunktioiden suorituskykyä välimuistin osumatarkkuuden sekä kaistankulutuksen suhteen. He toteavat, että ahneiden kustannusfunktioiden optimaalisuuden todistamiseen riittää, kun niitä verrataan aiemmin esiteltyihin vastinpareihin: suosioon ja vastaavasti elinikään perustuvaan ennustamiseen. Heidän suorittamansa simulaation tulokset näyttävät, että osumatarkkuutta tavoitteleva ahne ennustaminen suoriutuu aiempaa, suosioon perustuvaa ennustamista paremmin. Samoin kaistankulutuksen säästämistä tavoitteleva ahne ennustaminen onnistuu paremmin kuin elinikään perustuva ennustaminen. Samat tulokset ovat nähtävissä myös aiemmin tässä luvussa esitetyssä taulukossa 2.

Wu ja Kshemkalyani (2004) esittävät osumatarkkuutta tavoittelevaan ahneeseen ennustamiseen liittyvää matemaattista taustaa. Mikäli olio i ei ole ennaltageneroituna välimuistissa, pätee sille ehto: nykyinen pyyntökerta on osuma, mikäli kyseisen välimuistiolion edellisen päivityskerran jälkeen on tapahtunut jo vähintään yksi kyseiseen olioon kohdistunut pyyntö. Tällaisen osuman todennäköisyys voidaan laskea kaavan (4) mukaisesti (Wu & Kshemkalyani, 2004, alaluku 3.1).

$$P_{osuma}(i) = \frac{ap_i l_i}{ap_i l_i + 1} = f(i) \quad (4)$$

Tämä todennäköisyys on olion osumatarkkuus, kun kyseessä on tarvittaessa tehtävä sivupyynnön aikaansaama (engl. on-demand) välimuistiin tallennus. Tästä todennäköisyydestä käytetään myös nimeä *tuoreustekijä* (engl. freshness factor). Tuoreustekijä oliolle i on siten $f(i)$. Välimuistiin jo ennaltageneroiduille olioille osumatodennäköisyys on 1. Olion i osumatarkkuus eli osuman todennäköisyys voidaan siten ilmaista kaavassa (5) näkyvällä tavalla (Wu & Kshemkalyani, 2004, alaluku 3.1).

$$h_i = \begin{cases} \frac{ap_i l_i}{ap_i l_i + 1}, i\ ei\ ole\ ennaltageneroituna \\ 1, i\ on\ ennaltageneroituna \end{cases} \quad (5)$$

Ennaltagenerointialgoritmin kokonaisvaltainen osumatarkkuus voidaan ilmaista kaavan (6) avulla (Wu & Kshemkalyani, 2004, alaluku 3.1).

$$OT_{ennaltagenerointi\_käytössä} = \sum_{i \in S} p_i h_i \quad (6)$$

Wu ja Kshemkalyani (2004) esittävät, että ennaltageneroitaessa olio i, sen aikaansaama panos kokonaisvaltaiseen välimuistin osumatarkkuuteen voidaan laskea kaavassa (7) esitetyllä tavalla (Wu & Kshemkalyani, 2004, alaluku 4.3).

$$OT\_panos(i) = p_i\big(1 - f(i)\big) = \frac{p_i}{ap_i l_i + 1} \quad (7)$$

He toteavat täten, että valittaessa ennaltageneroitaviksi oliot, joilla osumatarkkuutta parantava panos on suurin, tuloksena on oltava paras mahdollinen muutos kokonaisvaltaisen osumatarkkuuden kannalta.

4.2.6 Työmäärän huomioiva ahne ennustaminen

Tässä alaluvussa tuodaan olemassa olevien ennaltageneroinnin kustannusfunktioiden yhteyteen uutena muuttujana kunkin ennaltageneroitavan olion tuottamiseen käytetty aika sekunteina eli olion generoimiseen käytettyä työmäärää kuvaava suure. Työmäärän huomioiva ahne kustannusfunktio pyrkii tasaamaan työläiden ja vähemmän työläiden olioiden generointiin kuluvaa viivettä ennaltageneroimalla työmäärältään raskaita välimuistiolioita välimuistiin. Uuden kustannusfunktion tavoitteena on löytää hyvä ennaltagenerointiin liittyvä logiikka, jotta käyttäjän kokemaa viivettä saataisiin vähennettyä ja pyyntöjen välisiä viive-eroja tasoitettua. Käyttäjälle tällainen web-sivun sisällön ennaltagenerointi näkyisi viiveen suhteen mahdollisimman nopeina ja myös viiveeltään tasakestoisempina sivulatauksina pyydettävästä sivusta riippumatta.

Ennen työmäärän (TM) huomioivan ahneen ennustamisen tarkempaa esittelemistä tutustutaan aiemmissa tutkimuksissa esiteltyyn kaistankulutukseen (KK), sillä se on luonteeltaan työmäärää vastaava välimuistiolion ominaisuus. Olion kaistankulutus on laskettavissa kaavan (8) avulla (Wu & Kshemkalyani, 2004, alaluku 3.2).

$$b_i = \begin{cases} ap_i\big(1 - f(i)\big)s_i \text{ , } i \textit{ ei ole ennaltageneroituna} \\ \dfrac{s_i}{l_i} \text{ , } i \textit{ on ennaltageneroituna} \end{cases} \tag{8}$$

Yhtälössä oleva muuttuja s_i kertoo olion i koon. Olion ollessa ennaltageneroituna, muuttujien s_i ja l_i suhde eli olion koon suhde sen keskimääräiseen elinikään määräävät kaistankulutuksen. Mikäli olio ei ole ennaltageneroituna, vaikuttavat siihen lähinnä sen käyttötiheys ja koko. Voidaankin todeta, että kokonaisvaltainen kaistankulutus saadaan selville kaavan (9) avulla (Wu & Kshemkalyani, 2004, alaluku 3.2).

$$KK_{ennaltagenerointi\_k\ddot{a}yt\ddot{o}ss\ddot{a}} = \sum_{i \in S} b_i \tag{9}$$

Wu ja Kshemkalyani (2004) esittävät kaistankulutusta tavoittelevan ahneen kustannusfunktion laskennassa käytetyn kaavan (10). He kertovat tätä yhtälöä hyödyntävän kustannusfunktion tavoittelevan mahdollisimman pientä olion ennaltageneroinnista aiheutuvaa ylimääräistä kaistankulutusta. (Wu & Kshemkalyani, 2004, alaluku 4.4).

$$KK\_panos(i) = \frac{s_i}{l_i}\big(1 - f(i)\big) = \frac{s_i}{ap_i l_i^2 + l_i} \tag{10}$$

Yhtälössä 10 on esitetty olion i kaistankulutuksen panos. Esitellään uusi muuttuja: olion i generointiin kuluva keskimääräinen aika t_i. Aika voidaan ilmaista esimerkiksi sekunteina. Aika t_i on luonteeltaan olion kokoa s_i vastaava välimuistiolion ominaisuus. Kaava (10) voidaan nyt kirjoittaa uudelleen, muuttamalla se työmäärää tavoittelevaan muotoon. Tuloksena saadaan kaava (11), jolla pystytään laskemaan olion i ennaltageneroinnin aiheuttama työmäärän panos.

$$TM\_panos(i) = \frac{t_i}{l_i}\big(1 - f(i)\big) = \frac{t_i}{ap_i l_i^2 + l_i} \tag{11}$$

Tässä luvussa aiemmin esitettyjen kaavojen pohjalta voidaan päätyä myös olion i työmäärän ilmaisevaan kaavaan (12) sekä edelleen kaavassa (13) ennaltagenerointialgoritmin aikaansaamaan kokonaistyömäärään.

$$w_i = \begin{cases} ap_i\big(1 - f(i)\big)t_i \text{ , } i \textit{ ei ole ennaltageneroituna} \\ \dfrac{t_i}{l_i} \text{ , } i \textit{ on ennaltageneroituna} \end{cases} \tag{12}$$

$$TM_{ennaltagenerointi\_käytössä} = \sum_{i \in S} w_i \tag{13}$$

Jotta voitaisiin hyödyntää yhdessä molempia: yhtälössä 7 esiteltyä olion i aikaansaamaa osumatarkkuuteen kohdistuvaa panosta ja yhtälössä 11 esiteltyä olion i panosta työmäärään, tulee ensin esitellä Jiangin ym. (2002) määrittelemä osumatarkkuuden ja kaistankulutuksen suhdetta mittaava OT/KK-suhde sekä sen osatekijät. Seuraavassa tarkasteltava tarvittaessa tehtävä välimuistiolion generointi tarkoittaa tilannetta, jossa ennaltagenerointi ei ole käytössä. Tarvittaessa tehtävälle välimuistiolion generoinnille on voimassa seuraavat, kaavojen (14) ja (15) esittämät ominaisuudet (Wu & Kshemkalyani, 2006, s. 13).

$$OT_{tarvittaessa} = \sum_{i \in S} p_i f(i) \tag{14}$$

$$KK_{tarvittaessa} = \sum_{i \in S} \frac{s_i}{l_i} f(i) \tag{15}$$

Kaavassa 14 on esitetty tarvittaessa generoitavien välimuistiolioiden osumatarkkuus ja kaavassa 15 määritetään vastaavasti tarvittaessa generoitavien välimuistiolioiden kaistankulutus. Aiemmin todettujen, muuttujien t_i ja s_i yhtenevien ominaisuuksien puolesta voidaan päätyä kaavaan (16), jossa esitetään tarvittaessa generoitavien välimuistiolioiden työmäärä.

$$TM_{tarvittaessa} = \sum_{i \in S} \frac{t_i}{l_i} f(i) \tag{16}$$

Jiangin ym. (2002, alaluku 5.2) mukaan osumatarkkuuden ja kaistankulutuksen suhde saadaan laskettua kaavan (17) avulla. Kaavassa esitetty OT/KK-suhde kertoo kuinka paljon osumatarkkuuden kasvun ja lisääntyvän kaistankäytön kustannusten tasapainottamiseen tähtäävää hyötyä ennaltagenerointialgoritmit voivat aikaansaada verrattuna vain tarvittaessa tehtävään olion generointiin. Toisin sanoen, kaavan osoittajassa oleva jakolasku vertaa osumatarkkuuden parantumista ennaltageneroinnin ollessa käytössä tilanteeseen, jossa ennaltagenerointi on pois käytöstä. Samankaltainen ennaltageneroinnin käytössä olon ehto pätee myös yhtälön nimittäjän jakolaskuun. Jiang ym. (2002) kertovat myös, että OT/KK-suhteen kaavaan voidaan lisätä eksponentti k, jonka arvoa muuttamalla voidaan vaikuttaa tavoiteltavaan osumatarkkuuden ja kaistankulutuksen suhteeseen. Esimerkiksi kasvattamalla muuttujan k arvoa, kasvatetaan osumatarkkuuden merkitsevyyttä yhtälössä. (Jiang ym., 2002, alaluku 5.2).

$$OT^k/KK = \frac{\left(OT_{ennaltagenerointi\_käytössä}/OT_{tarvittaessa}\right)^k}{\left(KK_{ennaltagenerointi\_käytössä}/KK_{tarvittaessa}\right)} \tag{17}$$

Wu ja Kshemkalyani (2004) ehdottavat tutkimuksessaan ahnetta algoritmia, jonka he väittävät aikaansaavan parhaan OT/KK-suhteen. He kutsuvat esittämäänsä algoritmia OT/KK-ahneeksi kustannusfunktioksi. He kuvaavat ehdotetun ahneen kustannusfunktion valitsevan kaikkien mahdollisten olioiden joukosta n kappaletta olioita, jotka eniten hyödyttävät OT/KK-suhdetta. He esittelevät tutkimuksessaan olion valinnan vaikutuksia kuvastavan *kasvutekijän* (engl. increase factor). Kasvutekijää käytetään välimuistiolioiden valintakriteerinä OT/KK-ahneessa algoritmissa. Kasvutekijän laskemisesta nähdään esimerkki kaavassa (18) (Wu & Kshemkalyani, 2004, alaluku 4.2).

$$\frac{OT_{tarvittaessa} + OT_{panos(i)}}{KK_{tarvittaessa} + KK_{panos(i)}} = \frac{\sum_{j \in S} p_j f(j) + p_i\big(1 - f(i)\big)}{\sum_{j \in S} \frac{s_j}{l_j} f(j) + \frac{s_i}{l_i}\big(1 - f(i)\big)}$$

$$= \left(\frac{OT_{tarvittaessa}}{KK_{tarvittaessa}}\right) \times \frac{\left(1 + \frac{p_i\big(1 - f(i)\big)}{\sum_{j \in S} p_j f(j)}\right)}{\left(1 + \frac{\frac{s_i}{l_i}\big(1 - f(i)\big)}{\sum_{j \in S} \frac{s_j}{l_j} f(j)}\right)} \tag{18}$$

$$= (OT/KK)_{tarvittaessa} \times kasvutekijä_{OT/KK}(i)$$

Esitän kaavaan 18 pohjautuvan osumatarkkuuden kasvuun ja mahdollisimman suureen välimuistisisällön generoinnin viiveeseen tähtäävän osumatarkkuustyömäärä ($OTTM$) -ahneen kustannusfunktion. Ennaltageneroinnissa valitaan siten tarjolla olevien mahdollisten olioiden joukosta n kappaletta olioita, joilla on suurin $OTTM$-ahneen kustannusfunktion kasvutekijä. Aiemmin esitelty KK–muuttuja on luonteeltaan negatiivinen ja sen arvoa pyritään minimoimaan, jolloin se on luonnollisesti sijoitettava nimittäjään, jotta aikaansaadaan mahdollisimman suuri OT/KK-kasvutekijä. Kaavan (19) muuttuja TM on kuitenkin esitellyn kustannusfunktion puitteissa positiivinen ja siten tavoittelemisen arvoinen suure. Täten kyseisen ahneen kustannusfunktion $OTTM$-kasvutekijän laskennassa hyödynnetään jakolaskun sijasta OT- ja TM-kasvutekijöiden tuloa, joka kaavassa 19 ilmaistaan käyttämällä TM-kasvutekijän käänteislukua.

$$\frac{\sum_{i \in S} p_i f(i) + p_j\big(1 - f(j)\big)}{\left(\frac{1}{\sum_{i \in S} \frac{t_j}{l_i} f(i) + \frac{t_j}{l_j}\big(1 - f(j)\big)}\right)}$$

$$= \frac{OT_{tarvittaessa} + OT_{panos(j)}}{\left(\frac{1}{TM_{tarvittaessa} + TM_{panos(j)}}\right)} \tag{19}$$

$$= \big(OT_{tarvittaessa} + OT_{panos(j)}\big) \times \big(TM_{tarvittaessa} + TM_{panos(j)}\big)$$

$$= \left(OT_{tarvittaessa} + OT_{panos(j)}\right)TM_{tarvittaessa} + \left(OT_{tarvittaessa} + OT_{panos(j)}\right)TM_{panos(j)}$$

$$= OT_{tarvittaessa}TM_{tarvittaessa} + OT_{panos(j)}TM_{tarvittaessa} + OT_{tarvittaessa}TM_{panos(j)} + OT_{panos(j)}TM_{panos(j)}$$

$$= (OTTM)_{tarvittaessa} \times \left(1 + \frac{TM_{panos(j)}}{TM_{tarvittaessa}} + \frac{OT_{panos(j)}}{OT_{tarvittaessa}} + \frac{OT_{panos(j)} \times TM_{panos(j)}}{(OTTM)_{tarvittaessa}}\right)$$

$$= (OTTM)_{tarvittaessa} \times \left(1 + \frac{\frac{t_j}{l_j}\left(1 - f(j)\right)}{\sum_{i \in S}\frac{t_j}{l_i}f(i)} + \frac{p_j\left(1 - f(j)\right)}{\sum_{i \in S}p_i f(i)} + \frac{\left(p_j\left(1 - f(j)\right)\right) \times \left(\frac{t_j}{l_j}\left(1 - f(j)\right)\right)}{\left(\sum_{i \in S}p_i f(i)\right) \times \left(\sum_{i \in S}\frac{t_j}{l_i}f(i)\right)}\right)$$

$$= (OTTM)_{tarvittaessa} \times kasvutekijä_{OTTM}(j)$$

Wu ja Kshemkalyani (2004, alaluku 4.2) esittävät algoritmin, jolla on mahdollista valita n kappaletta OT/KK-suhteen puolesta parasta vaihtoehtoa. Seuraavaksi on esiteltynä algoritmi (algoritmi 1), joka on muunnelma heidän esittämästään menettelystä. Tämä muunneltu algoritmi valitsee kaikkien mahdollisten olioiden joukosta n kappaletta olioita, joilla on suurimmat $OTTM$-ahneen kustannusfunktion kasvutekijän arvot.

Algoritmi 1. *OTTM*-ahneen kustannusfunktion käyttö ennaltageneroinnissa

```
OTTM_AHNE_ENNALTAGENEROINTI(S, n, a)
{

   # PARAMETRIT:
   # S,  joukko olioita, joista jokaisesta on
   #     tiedossa muuttujat p_i, l_i ja t_i
   # n,  ennaltageneroitava oliomäärä
   # a,  välimuistin käytön pyyntötiheys

   for each i ∈ S do
      laske tuoreustekijä: f(i) = ap_i l_i / (ap_i l_i + 1)
   end for

   laske kokonaisvaltainen vain tarvittaessa
   generoitaessa kertyvä osumatarkkuus:

   OT_tarvittaessa = Σ_{i∈S} p_i f(i)

   laske kokonaisvaltainen vain tarvittaessa
   generoitaessa kertyvä työmäärä:

   TM_tarvittaessa = Σ_{i∈S} (t_i / l_i) f(i)

   for each j ∈ S do
      laske kasvutekijä_OTTM(j)
   end for

   järjestä joukon S oliot laskevaan järjestykseen
   kasvutekijä_OTTM(j)-arvojen perusteella

   ennaltageneroi järjestyksessä ensimmäiset n
   kappaletta olioita välimuistiin

}
```

Tässä tutkimuksessa esitetty *OTTM*-ahne kustannusfunktio perii osumatarkkuutta tavoittelevan ahneen ennustamisen hyviä ja huonoja puolia. Mahdollisesti suuri, kustannusfunktion laskukaavassa huomioimatta jäävä kaistankäyttö on siten myös tämän kustannusfunktion huonona puolena. Mahdollisena hyvänä puolena kuitenkin on, että esitetty kustannusfunktion pyrkii tasoittamaan pyyntöjen viiveitä huomioimalla ennaltagenerointiin käytetyn työmäärän. Työläät ja samalla osumatarkkuudeltaan parhaat oliot palvellaan välimuistista ja vähemmän työläät osumatarkkuutta vähiten parantavat oliot generoidaan vasta pyynnön saavuttua. Äärettömän kokoisessa välimuistissa ennaltagenerointialgoritmi tuottaisi kaikki mahdolliset oliot välimuistiin ja keskimääräinen sisällön generoinnin viive L_g saataisiin lähelle nollaa. Rajallisen kokoisessa välimuistissa

vain osa mahdollisista olioista mahtuu kerralla välimuistiin. Esitetty työmäärän huomioiva kustannusfunktio pyrkii mahdollisimman pieneen ja tasaiseen L_g-viiveeseen. Tutkielman simulointituloksissa nähdään, miten ehdotettu kustannusfunktio suoriutuu L_g-viiveen tasoittamisen suhteen muihin kustannusfunktioihin nähden.

Esitetyn kustannusfunktion kaistankäyttöön kohdistuvan taakan huomiointi jätetään välimuistiratkaisun vastuulle. Välimuistiratkaisun tulee ennaltageneroida sisältöä vain silloin, kun se on ennaltageneroinnista aiheutuvan kuormituksen kannalta kannattavaa. Ainakin osittain kaistankäyttö on perusteltua, sillä kaistankäyttö on väistämätöntä osumatarkkuutta runsaasti nostavien olioiden tapauksessa. Välimuistitoteutuksen ennaltageneroinnin raja-arvot tulee kuitenkin asettaa riittävän tiukoiksi, jottei ennaltageneroinnista koidu liian suuria haittoja suhteessa siitä saataviin hyötyihin. Tämän tutkimuksen simulointitulokset havainnollistavat eri kustannusfunktioiden kaistankäytönkulutusta suhteessa ennaltageneroinnista saataviin hyötyihin.

4.3 Luvun yhteenveto

Tässä luvussa tutustuttiin tarkemmin kustannusfunktioihin pohjautuvaan ennaltagenerointiin. Luvussa esiteltiin olemassa olevia eri hyötyjä tavoittelevia kustannusfunktioita sekä uusi, osumatarkkuutta ja työmäärää tavoitteleva ahne kustannusfunktio. Tarkastellut kustannusfunktiot toimivat laskentakaavoina seuraavan luvun pohjalta suoritettavissa simulaatioissa. Seuraavassa luvussa esitellään koejärjestelyt, simulointiympäristö ja kokeet, joiden avulla tämän tutkielman simulointiosion tutkimustulokset tullaan tuottamaan.

5 KOEJÄRJESTELYT

Tässä luvussa kuvataan koejärjestelyt, jonka lisäksi esitellään kokeiden kulku sekä välimuistimallin simuloinnin toteutustapa. Koejärjestelyissä mitattavan artefaktin ei ole tarkoitus olla käytettävä web-sovellus, vaan se simuloi mahdollisimman realistisesti web-sovelluksen ominaisuuksia. Tehdyt valinnat esimerkiksi raja-arvojen ja muiden oletusten osalta perustellaan. Nämä valinnat muovaavat web-sovelluksen tyyppiominaisuuksia, joita tässä tutkimuksessa ollaan etsimässä. Raja-arvoja ja oletuksia pyritään siten siirtämään suuntaan, jossa esitetystä $OTTM$-ahneesta kustannusfunktiosta saadaan paras hyöty. Tästä huolimatta simuloitavan web-sovelluksen tulee täyttää perustellusti tietyn reaalimaailman web-sovellustyypin tunnusmerkit.

Seuraavassa alaluvuissa esitellään koeympäristö, simulaattori sekä kuvataan yhden simulointiajon suoritus. Lisäksi määritellään suoritettavien kokeiden mittarit sekä simulointiajoissa voimassa olevat oletukset sekä rajaukset.

5.1 Koeympäristö

Koeympäristönä toimii täysin simuloitu kustannusfunktioiden laskentaan pohjautuva ohjelmoimalla toteutettu parametroitu ajo. Kukin simulaatioajo pyrkii vastaamaan yhteen tutkimusongelman kannalta oleelliseen kysymykseen. Kunkin simulaatioajon pohjalta syntyy useampia kaavioita, joista osa esitetään seuraavan luvun simulointituloksina ja osa tämän tutkielman liitteinä.

Simulaatiossa voidaan määrittää välimuistiin talletettavien olioiden kokonaismääräksi esimerkiksi 10^5 oliota. Tämän jälkeen kullekin välimuistioliokandidaatille luodaan satunnaisesti ominaisuuksia ennalta määritettyjen vaihteluvälien pohjalta. Alaluvussa 5.3 kuvataan vaihteluvälit, joiden pohjalta välimuistiolioiden ominaisuudet valitaan. Tällaisia ominaisuuksia ovat: välimuistiolion keskimäärinen elinikä, koko ja sen generoimiseen kuluva keskimääräinen työmäärä. Näiden ominaisuuksien lisäksi jokainen välimuistiolio vaatii suosioon liittyvän oliokohtaisen ominaisarvon laskemisen zipfiläistä jakaumaa hyödyn-

täen. Edellä esitettyjen oliokohtaisten ominaisuuksien lisäksi simulaatio hyödyntää muita, taulukossa 3 esitettyjä aiempien tutkimusten hyväksi havaitsemia muuttujia. Simulaatioissa ei ole tarpeen kiinnittää tarkasteltavan mittausajanjakson $[t_{k-n},\ t_k]$ muuttujan n arvoa, sillä tarkasteltavien olioiden ja muiden muuttujien arvot alustetaan simulaatioiden alussa mittaamisen sijasta.

Simulaatiot suoritetaan laskemalla, joten esimerkiksi kaistankulutusta simuloitaessa ei ole tarvetta huolehtia riittävästä siirtokapasiteetista välimuistiratkaisun ja taustajärjestelmien välillä. Toisin sanoen kaistankulutuksen simulaatio kertoo kustannusfunktioiden siirtokapasiteetin tarpeen ja esittää siten, mitkä kustannusfunktiot ovat riskialttiimpia kaistankulutuksen suhteen. Simulaatiot olettavat, että kaistankulutuksesta selvitään, eikä tiedonsiirrolle ole tarvetta määrittää raja-arvoja. Komponenttien välistä tiedonsiirtoa on oleellista rajoittaa siinä vaiheessa, kun reaalimaailman järjestelmää ollaan toteuttamassa.

Kukin simulaatio ajetaan kolmesti, jotta satunnaislukugeneroinnin aiheuttamat mahdolliset yllättävät vaikutukset voidaan sulkea pois. Myös välimuistiolioiden suuri määrä vähentää simuloinnissa käytettävän satunnaislukugeneroinnin vaikutuksia kuvaajien painotuksiin. Tämä on havaittavissa tämän tutkielman liitteissä, joissa esimerkiksi 10^3 kokoisella välimuistiolioiden joukolla kaavioiden viivakuvaajat eivät ole välttämättä täysin johdonmukaisia.

Simulaatio etenee laskemalla tarkasteluun valittua kustannusfunktioita hyödyntämällä eri välimuistin täyttöasteilla saavutettavat hyödyt, mittaamalla kussakin simulaatiossa mielenkiinnon kohteena olevaa suuretta. Yhteistä eri simulaatioajoille on kahden ääritapauksen välillä liikkuminen: ”ei yhtään välimuistioliota ennaltageneroituna välimuistiin” sekä vastaavasti ”kaikki välimuistioliot ennaltageneroituna välimuistiin” -tapaukset. Peilattaessa simulaatioita reaalimaailman järjestelmään, voidaan esimerkiksi tarkastella simulaatioita 20 % välimuistin täyttöasteen kohdalta, jolloin tämä vastaa reaalimaailman systeemin kykyä saada 20 % kaikista mahdollisista välimuistiolioista välimuistiin.

Tutkimuksen koeympäristön tarpeisiin on ohjelmoitu simulaattori C# kielellä .NET -sovelluskehyksen avulla. Kukin simulaatio koostuu yhdestä parametroidusta ohjelman ajosuorituksesta, jonka lopputuloksena saadaan simuloidun välimuistioliojoukon ominaisuuksien pohjalta piirretty graafinen kuvaaja. Simulaattoriin on luvun 4 laskukaavojen pohjalta toteutettu välimuistioliota ja sen ominaisuuksia mallintava luokka sekä kustannusfunktioiden laskukaavat.

Seuraavana esitetyssä algoritmissa (algoritmi 2) on kuvattu yhden parametroidun simulaatioajon suoritus. Algoritmi käyttää osittain seuraavien alalukujen 5.2 ja 5.3 mukaista terminologiaa, suureita sekä muuttujien oletusarvoja. Algoritmissa esiintyvä parametri m kuvaa kulloinkin suoritettavan simulaation mittaria. Mittaria vaihtamalla saadaan samaan simulaatioon useita eri perspektiivejä. Yhdessä simulaatioissa voidaan tarkastella esimerkiksi pelkästään eri kustannusfunktioiden kykyä vaikuttaa välimuistin osumatarkkuuteen eri välimuistin täyttöasteilla. Vaihtamalla muuttujan m arvoksi kaistankulutusta mittaava mittari, saadaan esille kustannusfunktioiden ennaltageneroinnin aiheuttama siirtokapasiteetin tarve eri välimuistin täyttöasteilla. Simulaatioissa käytettävät mittarit on esitelty seuraavassa alaluvussa.

Algoritmi 2. Simulaatioajon suoritus

```
SIMULAATIOAJO(S_lkm, a, t_max, l_max, s_max, m, n)
{
    # PARAMETRIT:
    # S_lkm,   simulaation koko
    # a,       välimuistin käytön pyyntötiheys
    # t_max    keskimääräisen generoinnin keston maksimiarvo
    # l_max    keskimääräisen eliniän maksimiarvo
    # s_max,   keskimääräisen koon maksimiarvo
    # m,       tarkasteltava mittari
    # n,       näytteenottotiheys (esimerkiksi 10)

    alusta kaikkien mahdollisten välimuistiolioiden joukko S

    laske zipfiläinen jakauma Z välillä 1..S_lkm

    for each i ∈ S_lkm do
       alusta uusi välimuistiolio i
       aseta välimuistioliolle i satunnaiset arvot:
          keskimääräinen generoinnin kesto väliltä [1, t_max]
          keskimääräinen elinikä väliltä [0, l_max]
          keskimääräinen koko väliltä [1, s_max]
       aseta välimuistioliolle i todennäköisyys p_i = Z[i]
       lisää välimuistiolio i joukkoon S
    end for

    alusta lopputuloksille taulukot L_1..k[n + 1]

    aseta kullekin kustannusfunktiolle omaan taulukkoonsa L_1..k
    indeksiin 0 vain tarvittaessa generoitaessa kertyvä
    lähtöarvo tarkasteltavan mittarin m osalta

    for each j ∈ n do
       merkitse välimuistiin ennaltageneroiduksi joukosta
       S seuraavaksi parhaat S_lkm/n välimuistioliota
       kaikilla eri kustannusfunktioiden oliojärjestyksillä

       aseta kullekin kustannusfunktiolle omaan taulukkoonsa
       L_1..k indeksiin j kaikkien välimuistissa olevien olioiden
       yhteenlaskettu mittarin m mukainen panos tai kasvutekijä

       lisää kullekin kustannusfunktiolle omaan taulukkoonsa
       L_1..k indeksiin j vain tarvittaessa generoitaessa kertyvä
       lähtöarvo tarkasteltavan mittarin m osalta
    end for

    piirrä taulukkojen L_1..k arvojen mukainen graafinen kuvaaja

}
```

5.2 Suoritettavat kokeet ja niiden mittarit

Kokeiden mittareina toimii joukko aiempien tutkimusten esittelemiä mittareita sekä tämän tutkimuksen tutkimusongelmaa palvelevat, esimerkiksi pyynnön palvelemisen kestoon keskittyvät mittarit. Seuraavassa listauksessa on esiteltynä kokeissa käytetyt mittarit sekä avattuna tarkemmin, mihin kysymykseen mikäkin mittareista pyrkii vastaamaan. Lisäksi kunkin mittarin osalla on esitettynä reaalimaailman esimerkki, johon kyseisen mittarin tavoitetta voitaisiin hyödyntää.

- OT-mittari mittaa välimuistin osumatarkkuutta kullakin välimuistin täyttöasteella. Mittari vastaa kysymykseen: kuinka eri kustannusfunktiot vaikuttavat osumatarkkuuteen. Tätä mittaria hyödynnettäessä käytännön tavoitteena voisi olla selvittää, millä kustannusfunktiolla saadaan aikaan paras osumatarkkuus ennaltageneroimalla vain n kappaletta olioita välimuistiin.
- KK-mittari mittaa kustannusfunktioista aiheutuvaa kaistankulutusta kullakin välimuistin täyttöasteella. Mittari vastaa kysymykseen: kuinka paljon kaistankulutusta aiheutuu eri kustannusfunktioiden tavasta valita n kappaletta ennaltageneroitavia välimuistiolioita. Tätä mittaria hyödynnettäessä käytännön tavoitteena voisi olla selvittää, millä kustannusfunktiolla välimuisti voidaan täyttää puolilleen minimoimalla kuitenkin aiheutuva kaistankulutus.
- TM-mittari mittaa välimuistissa olevien olioiden panosta kokonaistyömäärään kullakin välimuistin täyttöasteella. Mittari vastaa kysymykseen: kuinka eri kustannusfunktioiden tavat valita n kappaletta ennaltageneroitavia välimuistiolioita vaikuttavat siihen, paljonko työmäärän panosta välimuistiin valituilla olioilla on, suhteessa tunnettujen olioiden joukon S kokonaistyömäärään. Tämän mittarin käytännön sovelluksena voitaisiin selvittää, millä kustannusfunktiolla välimuistiin saadaan valittua eniten taustajärjestelmiä kuormittavat välimuistioliot.
- Pyynnön palvelemiseen kuluvan keskimääräisen ajan mittari vastaa kysymykseen: millä kustannusfunktiolla saadaan tehokkaimmin pienennettyä keskimääräistä L_g-viivettä. Oletuksena tämän mittarin simuloinnissa on, että yhtä pyyntöä vastaa web-sovelluksessa vain yksi välimuistiolio. Lisärajauksena on se, ettei pyynnön palvelemisen aikaan lasketa mukaan muuta kuin tarkastelun kohteena oleva L_g-viive.
- Pyynnön palvelemiseen kuluvan ajan jakaumaa esittävä mittari vastaa kysymykseen: millä kustannusfunktiolla saadaan aikaiseksi L_g-viiveen arvojen mahdollisimman tiivis jakauma, mahdollisimman pienellä välimuistin käytön täyttöasteella. Oletuksena tämänkin mittarin simuloinnissa on, että yhtä pyyntöä vastaa web-

sovelluksessa vain yksi välimuistiolio ja, ettei pyynnön palvelemisen aikaan lasketa muuta kuin tarkastelun kohteena oleva L_g-viive.

5.3 Oletukset ja raja-arvot

Tämän alaluvun tavoitteena on esitellä yleisesti kokeissa voimassa olevat oletukset sekä määritellä alustavat raja-arvot, joiden vaihtelua suoritettavissa kokeissa tarkastellaan. Seuraavassa on listattuna keskeiset muuttujat sekä niiden oletusarvoiset vaihteluvälit.

- Tunnettujen välimuistiolioiden joukon S kokoa varioidaan simuloinneissa neljällä eri arvolla: 10^3, 10^4, 10^5, 10^6.
- Välimuistiolioiden oliokohtaiset ominaisuudet valitaan satunnaisesti seuraavilta vaihteluväleiltä:
    - Olion i keskimääräinen elinikä l_i vaihtelee $0 - 10^5$ sekunnin välillä.
    - Olion i koko s_i saa arvoja $1 - 10^6$ tavun väliltä. Vaihtelemalla olioiden kokoa päästään lähemmäs reaalimaailman vastaavuutta kaistankulutuksen arvioinnissa.
    - Olion i generointiin kuluva keskimääräinen aika t_i vaihtelee $1 - 20$ sekunnin välillä.
    - Olion i todennäköisyys tulla pyydetyksi p_i noudattaa zipfiläistä jakaumaa, jonka avulla jokaiselle oliolle lasketaan todennäköisyys. Jakauman laskennassa hyödynnetään arvoa $\alpha = 0{,}75$ sillä sen on nähty Nishikawan ym. (1998) mukaan vastaavan eniten reaalimaailman tilanteita.
- Simulaation kokonaisvaltainen välimuistin käytön pyyntötiheys a saa arvon $0{,}01$ pyyntöä sekunnissa.
- Simulaatioissa joissa lasketaan pyynnön palvelemiseen kuluvaa keskimääräistä aikaa tai aikojen jakaumia, käytetään laskennassa lukumäärältään $10\,S$ kokoista pyyntöjoukkoa. Nämä pyynnöt jakautuvat välimuistiolioiden joukon S kesken zipfiläiseen jakaumaan pohjautuen, jolloin jokainen välimuistiolio tulee pyydetyksi simulaatiossa vähintään kerran.

Yllä olevien muuttujien oletusarvot ja vaihteluvälit on valittu perustuen aiempiin aihepiirin tutkimuksiin. $OTTM$ -kustannusfunktiolle edullisia web-sovelluksen tyyppiominaisuuksia etsittäessä tulee kuitenkin tarkastella vaihteluvälien valintoja laajemmin. Tästä johtuen yllä olevien muuttujien oletusarvoja varioidaan simulaatioissa myös alla olevan listauksen mukaisesti. Tavoitteena on löytää web-sovelluksen tyyppiominaisuudet, joissa esitetyn kustannusfunktion hyödyt suhteessa esimerkiksi kaistankulutukseen olisivat edullisimmat. Alla olevien muuttujien variointien simulaatiot suoritetaan kiinteällä oliomää-

rällä 10^5. Kerrallaan tutkitaan vain yhden muuttujan vaihtelun aiheuttamaa yksittäisvaikutusta.

- Olion i keskimääräinen elinikä $l_i$valitaan satunnaisesti oletusarvon $0 - 10^5$ lisäksi myös väleiltä $0 - 10^4$ ja $0 - 10^6$.
- Oletusarvoista välimuistin käytön pyyntötiheyttä a muunnellaan käyttäen arvoja: $a = 0{,}005$; $a = 0{,}1$; $a = 1$ pyyntöä sekunnissa.
- Olion i generointiin kuluva keskimääräinen aika t_i valitaan satunnaisella kaavalla oletusarvosta poiketen myös pidemmältä $1 - 100$ sekunnin väliltä. Tämän lisäksi satunnaisesta kaavasta poikkeavasti, oliokohtainen t_i arvo valitaan myös nousevalla sekä laskevalla painotetulla jakaumalla. Kulmakertoimena painotetuissa jakaumissa toimii nousevalle jakaumalle arvo 1 ja laskevalle arvo -1. Täten esimerkiksi nousevassa painotetussa jakaumassa vain murto-osa olioista saa arvoltaan pienen generointiin kuluvan keskimääräisen ajan. Tällöin keskimääräisen generointiajan kasvaessa $1 - 20$ sekunnin välillä yhä suurempi osa olioista saa osakseen suuruusluokkaa vastaavan keskimääräisen generointiajan t_i.

Tässä alaluvussa esitettyjen oletusten ja rajoitusten lisäksi on hyvä mainita, että simulaatioissa, joissa lasketaan pyynnön palvelemiseen kuluvaa keskimääräistä aikaa tai aikojen jakaumia ovat voimassa seuraavat rajaukset:

- Yksi pyyntö kohdistuu vain yhteen välimuistiolioon, joka voi olla välimuistissa tai sen ulkopuolella. Täten generoinnin viiveeseen voi vaikuttaa vain yksi välimuistiolio.
- Yksi välimuistiolio voi olla usean eri pyynnön kohteena. Täten esimerkiksi suosittuun välimuistiolioon kohdistuu useita pyyntöjä.
- Tutkimuksen mielenkiinnon kohteena on pelkkä olion generointiin kohdistuva viive, joten pyyntöjen palvelemisen viiveiden arvioinnissa ei tämän lisäksi huomioida muita mahdollisia L_g-viiveen aiheuttajia.

Seuraavan luvun simulointituloksissa havaitaan, että web-sovelluksen normaali käyttö ilman ennaltagenerointia tuottaa tietyn määrän olioita välimuistiin, jonka ansiosta välimuistista saatava passiivinen hyöty nousee. Simulaatiot osoittavat myös, että web-sovelluksen ennaltageneroiva välimuisti voidaan nähdä olevan tehokkaimmillaan tilanteessa, jossa pelkkien käyttäjiltä saapuvien pyyntöjen aiheuttamat vain tarvittaessa tehtävät välimuistiintallennukset eivät aikaansaa korkeaa välimuistin käytön pyyntötiheyttä ja siten lähtötasoltaan korkeaa välimuistin osumatarkkuutta. Tästä johtuen simulaatioissa, joissa lasketaan pyynnön palvelemiseen kuluvaa keskimääräistä aikaa tai aikojen jakaumia pätee lisäksi seuraava rajaus:

- Simulaatiot eivät huomioi vain tarvittaessa välimuistiin tapahtuvista välimuistiintallennuksista saavutettavia viiveen hyötyjä. Täten molempien mittareiden simulaatiot kuvaavat tilannetta, jossa välimuistia täydennetään pelkästään ennaltageneroinnin avulla, jolloin erot eri kustannusfunktioiden välillä ovat selkeiten nähtävissä.

5.4 Luvun yhteenveto

Tässä luvussa esiteltiin simulointiympäristön toimintaperiaate sekä suoritettavat kokeet. Tämän lisäksi kuvattiin simulointiajojen tarkastelun kohteena olevat mittarit. Suoritettaviksi esitetyt kokeet ja niitä vastaavien mittareiden tuottamat simulointitulokset ovat nähtävissä seuraavassa simuloinnin koetuloksia käsittelevässä luvussa.

6 SIMULOINNIN KOETULOKSET

Tässä luvussa esitellään simulaatioiden tuottamat tutkimustulokset jaoteltuina edellisessä luvussa esitettyjen mittareiden mukaan. Tämän lisäksi kutakin tarkasteltavaa mittaria kohden varioidaan tunnettujen olioiden joukon S kokoa. Mittarikohtaisten kokeiden lisäksi esitetään simulointitulokset myös muiden tarkastelun kohteina olevien muuttujien varioimisen vaikutuksista.

Kukin simulaatiotulos koostuu graafisesta kuvaajasta, joka esittää kulloinkin vain tietyn mittarin mittaaman suureen muutosta suhteessa välimuistiin ennaltageneroituun oliomäärään. Keskeisimpänä tarkastelun kohteena on tässä tutkielmassa esitetty välimuistiolion osumatarkkuuden ja keskimääräisen tuottamisen keston huomioiva ahne kustannusfunktio *($OTTM$-ahne)*. Simulaatioihin on valittu vertailukohdiksi aiemmin kirjallisuuskatsauksessa esitetyt kustannusfunktiot:

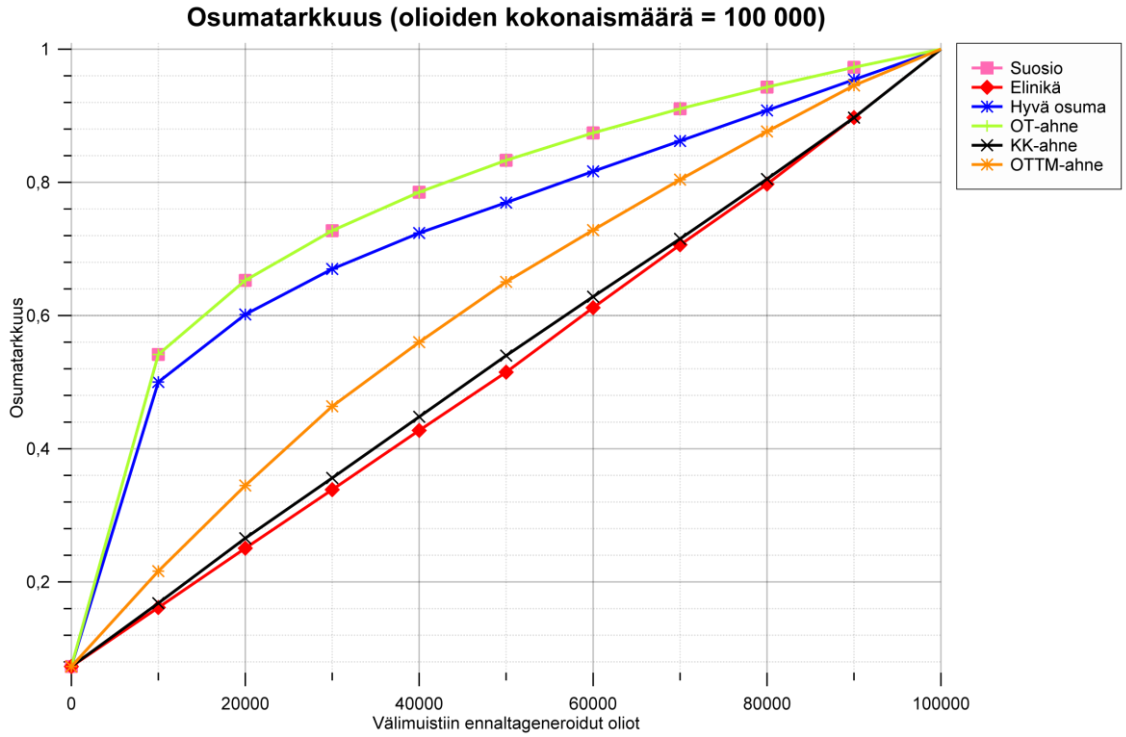
- Suosioon perustuva ennustaminen *(Suosio)*
- Elinikään perustuva ennustaminen *(Elinikä)*
- Hyvään osumaan perustuva ennustaminen *(Hyvä osuma)*
- Ahne osumatarkkuutta tavoitteleva ennustaminen *(OT-ahne)*
- Ahne kaistankäytön säästöä tavoitteleva ennustaminen *(KK-ahne)*

Seuraavissa alaluvuissa on esitettynä tutkimustuloksia välimuistioliomäärän ollessa 10^5. Loput simulointitulokset oliomäärillä 10^3, 10^4, 10^6 ovat nähtävissä tämän tutkielman liitteinä.

6.1 Välimuistin osumatarkkuus välimuistin täyttöasteittain

Tässä alaluvussa esitetään OT-mittarin simulointitulokset. Seuraavassa kuviossa (kuvio 3) nähdään eri kustannusfunktioiden vaikutukset välimuistin osumatarkkuuteen eri välimuistin täyttöasteilla, kun kokonaisoliomäärä on 10^5 välimuistioliota. Tulokset esitetään viivakaaviossa, jossa vaaka-akselilla kuvataan

välimuistiin ennaltageneroitujen olioiden lukumäärää ja pystyakselilla välimuistiin osuvien pyyntöjen osumatarkkuutta. Välimuistiin ennaltageneroidun oliomäärän ollessa vielä nollassa, saavat kaikki kustannusfunktiot arvokseen nollasta poikkeavan osumatarkkuuden arvon. Tämä pohjautuu kaavassa 14 esitettyyn tarvittaessa generoitavien välimuistiolioiden osumatarkkuuteen, joka lasketaan osumatarkkuutta mitattaessa lähtöarvoksi, jonka päälle ennaltageneroinnin aikaansaamaa osumatarkkuuden parannusta kerrytetään. Websovelluksen normaali käyttö ilman ennaltagenerointia tuottaa täten tietyn määrän olioita välimuistiin, jonka ansiosta välimuistista saatava passiivinen hyöty nousee.



KUVIO 3 Eri kustannusfunktioiden osumatarkkuus välimuistin täyttöasteittain

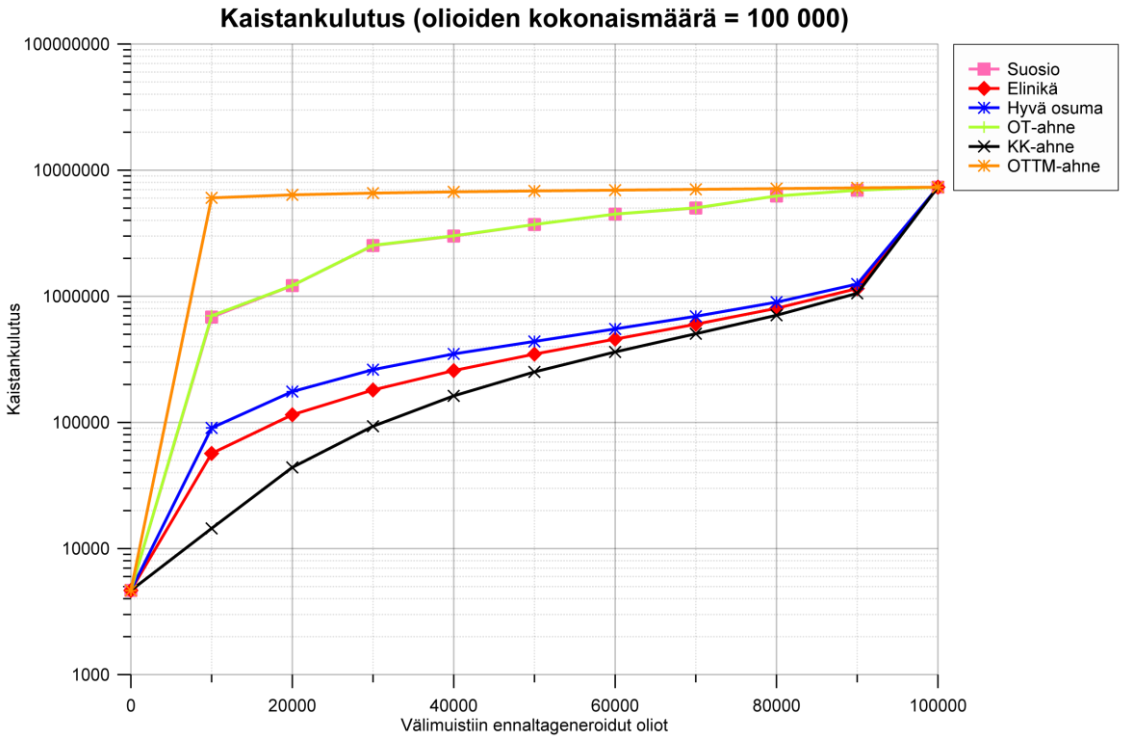
Simulointitulokset osoittavat, että olion todennäköisyyteen p_i tulla pyydetyksi pohjautuvat kustannusfunktiot: OT-ahne, suosioon pohjautuva ja hyvään osumaan pohjautuva saavat aikaan parhaat välimuistin osumatarkkuuden tulokset. Tämän lisäksi suosioon pohjautuva sekä OT-ahne kustannusfunktio näyttävät saavan lähelle toisiaan sijoittuvia arvoja. Wu ja Kshemkalyani (2004) todistavat kuitenkin, että osumatarkkuutta tavoitteleva ahne kustannusfunktio saavuttaa näistä kahdesta aina korkeimman osumatarkkuuden. Esimerkiksi varioitaessa muuttujan a arvoa, saadaan näkyvää eroa näiden kahden kustannusfunktion välille.

Tarkastelun kohteena oleva $OTTM$ -ahne kustannusfunktio käyttäytyy puoliksi OT-mittarin mukaisesti, johtuen siitä että puolet kustannusfunktion laskukaavan painotuksesta tulee OT-ahneen laskennan pohjalta. Muut kustannusfunktiot eivät tavoittele laskennassaan olion todennäköisyyttä p_i tulla pyy-

detyksi, joten niiden kuvaajat etenevät lineaarisesti simulaation satunnaislukugeneroinnista johtuvaa vaihtelua lukuun ottamatta. OT-mittarin simulointitulokset välimuistin muilla oliomäärillä ovat nähtävissä liitteessä (LIITE 1).

6.2 Aiheutuva kaistankulutus välimuistin täyttöasteittain

Kaistankulutuksen mittaamiseen tähtäävä KK-mittari kuvaa kustannusfunktioista aiheutuvaa kaistankulutusta. Eri kustannusfunktiot aiheuttavat eri määrän kaistankulutusta perustuen niiden tapaan valita n kappaletta ennaltageneroitavia välimuistiolioita. Seuraavassa kuviossa (kuvio 4) on esitettynä eri kustannusfunktioista aiheutuva kaistankulutus 10^5 olion kokonaismäärällä. Simulaation tulokset esitetään viivakaaviossa, jossa vaaka-akselilla kuvataan välimuistiin ennaltageneroitujen olioiden lukumäärää ja pystyakselilla ennaltageneroinnin aiheuttamaa kaistankulutusta, jonka mittayksikkönä on tavua sekunnissa. Kuten edellisessäkin simulaatiossa, tämänkin mittarin osalta pystyakselin arvot poikkeavat nollasta tilanteessa, jossa välimuistiin ei ole vielä ennaltageneroitu yhtään olioita. Myös kaistankulutuksen mittari perustuu siihen, että laskennan lähtöarvoksi huomioidaan kaavan 15 mukaan määräytyvä, vain tarvittaessa generoitavien välimuistiolioiden kaistankulutus. Tämä tarkoittaa ennaltageneroinnista riippumatonta välimuistiolioiden generointia ja siitä aiheutuvaa kaistankulutusta.



KUVIO 4 Eri kustannusfunktioiden kaistankulutus välimuistin täyttöasteittain

Simulaation tutkimustuloksia tarkasteltaessa tulee huomata pystyakselin logaritminen asteikko ja siten eri kustannusfunktioiden sijoittuminen kaistankulutuksensa suhteen kauas toisistaan. Kaistankulutus koetaan negatiivisena välimuistin ennaltageneroinnin ominaisuutena, joten kaavion mukaan selkeästi vähiten ylimääräistä kuormaa aiheuttavat välimuistiolion keskimääräiseen elinikään pohjautuvat kustannusfunktiot: KK-ahne, suoraan keskimääräiseen elinikään pohjautuva sekä hyvään osumaan pohjautuva.

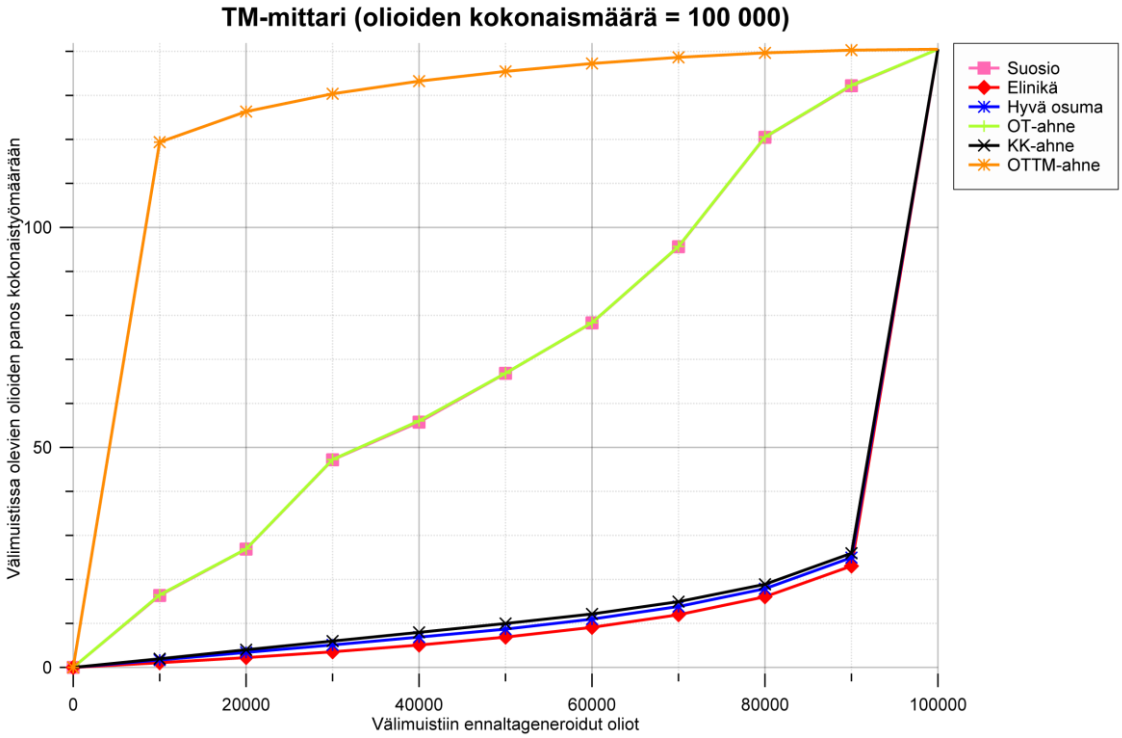
Logaritmisesta asteikosta huolimatta $OTTM$-ahne kustannusfunktio saa aikaan jo 10 % välimuistin täyttöasteen kohdalla lähes maksimaalisen kaistankulutuksen. Vertailukohtana voidaan käyttää esimerkiksi OT-ahnetta kustannusfunktiota, joka myös aikaansaa kaistankulutusta mutta sekin monta kertaluokkaa vähemmän. Aiemmat tutkimukset kritisoivat jo ennalta OT-ahneen kustannusfunktion kaistankäyttöä, joten monta kertaluokkaa korkeampi $OTTM$-ahneen kustannusfunktion kaistankulutus on sitäkin kritisoitavampaa. Wu ja Kshemkalyani (2004) toteavatkin, että OT-ahnetta kustannusfunktiota tulisi välttää, ellei tasapainotukseen hyödynnetä OT/KK-ahnetta kustannusfunktiota ylimääräiseltä kaistankäytöltä välttymiseksi. Tämän mittarin simulointitulokset välimuistin muilla oliomäärillä ovat nähtävissä liitteessä (LIITE 2).

6.3 Välimuistiolioiden panos kokonaistyömäärään

Tässä alaluvussa olevat simulaatiotulokset näyttävät millä kustannusfunktiolla välimuistiin saadaan valittua eniten sisällön generoinnin viiveen L_g kasvua aikaansaavat välimuistioliot. Tarkasteltava TM-mittari esittää välimuistissa olevien olioiden panosta kokonaistyömäärään kullakin välimuistin täyttöasteella.

Seuraavassa kuviossa (kuvio 5) nähdään eri kustannusfunktioiden käyttäytyminen kokonaistyömäärän panoksen suhteen. Simulaation tulokset esitetään viivakaaviossa, jossa vaaka-akselilla kuvataan välimuistiin ennaltageneroitujen olioiden lukumäärää. Kaavion pystyakseli esittää lukua, joka saavuttaa huippuarvossaan tunnettujen välimuistiolioiden joukon S sisältämän laskennallisen maksimityömäärän. Pystyakselin mittayksikkönä on sekuntia sekunnissa, sillä se kuvaa suhdetta t_i/l_i eli olioiden tuottamiseen kuluvan ajan suhdetta olioiden elinikään välimuistissa. Aiempien tämän luvun mittareiden tavoin, myös TM-mittarin laskennassa huomioidaan työmäärän lähtötaso, joka saadaan kaavan 16 avulla. Tämä lähtötaso esittää vain tarvittaessa generoitavien välimuistiolioiden työmäärää tilanteessa, jossa välimuistiin ei ole ennaltageneroitu vielä yhtään oliota. Kaava tuottaa kuitenkin varsin mitättömiä arvoja kokonaistyömäärään verrattuna, eikä niiden merkitsevyys edes välity kaavioita tarkasteltaessa.

Tutkimuksen mielenkiinnon kohteena oleva $OTTM$-ahne kustannusfunktio on tarkasteltavan kustannusfunktiojoukon ainoa, joka ottaa laskennassaan kantaa välimuistiolion tuottamiseen kuluvaan keskimääräiseen aikaan t_i. Täten olikin oletettavissa, että kyseinen kustannusfunktio on työmäärän panosta tavoittelevan TM-mittarin suhteen hyödyllinen.



KUVIO 5 Eri kustannusfunktioiden sijoittuminen työmäärän panoksen suhteen

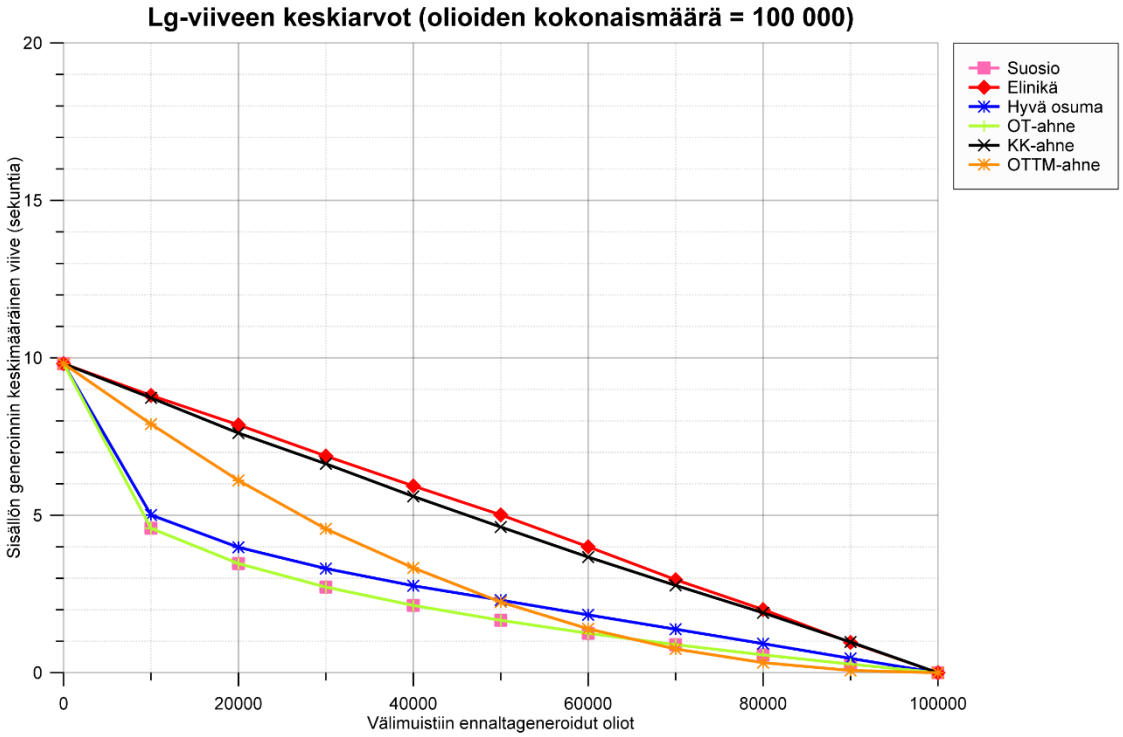
$OTTM$-ahneen kustannusfunktion toinen puoli hyödyntää TM-ahnetta laskentaa ja pyrkii löytämään eniten työmäärän panosta aikaansaavat välimuistioliot. Tästä syystä välimuistioliot, joilla esimerkiksi keskimääräinen elinikä on alhainen, ovat jatkuvan muuttuvuutensa ansiosta helposti työmäärän panokseltaan korkeita. Edellä todettuun verratessa, toisena ääripäänä ovat olion keskimääräiseen elinikään pohjautuvat kustannusfunktiot, joilla pyritään valitsemaan mahdollisimman pitkään välimuistissa muuttumattomina säilyviä välimuistiolioita. TM-mittaria kuvaavassa kaaviossa nähdään, kuinka keskimääräiseen elinikään pohjautuvat kustannusfunktiot tavoittelevat $OTTM$-ahneeseen kustannusfunktioon nähden täysin päinvastaista asiaa ja sijoittuvatkin siten kaavion pohjalle.

Simulaation tuloksissa OT-ahne sekä suosioon pohjautuva kustannusfunktio saavat lähes lineaarisesti sijoittuvat arvot, joihin vaihtelua aiheuttaa ainoastaan simulaation oliokohtaisten ominaisuuksien satunnaisuus. TM-mittarin simulointitulokset välimuistin muilla oliomäärillä ovat nähtävissä liitteessä (LIITE 3).

6.4 Pyynnön palvelemiseen kuluvat keskimääräiset ajat

Tässä alaluvussa esitetään pyynnön palvelemiseen kuluvan keskimääräisen ajan mittarin simulointitulokset. Tutkimustuloksista voidaan nähdä kustannusfunktiot, joilla saadaan tehokkaimmin pienennettyä keskimääräistä L_g-viivettä. Oletuksena tämän mittarin simuloinnissa on, että yhtä pyyntöä vastaa web-

sovelluksessa vain yksi välimuistiolio ja ettei pyynnön palvelemisen aikaan lasketa mukaan muuta kuin tarkastelun kohteena oleva L_g-viive. Mittarin simuloinnissa tarkasteltava pyyntöjoukko kohdistetaan joukon S olioihin hyödyntämällä zipfiläisen jakauman mukaista pyyntöjen hajontaa eri olioiden kesken. Seuraavassa kuviossa (kuvio 6) esitetään sisällön generoinnin keskimääräiset viiveet. Viivakaavion vaaka-akselilla nähdään välimuistiin ennaltageneroitujen olioiden lukumäärä.



KUVIO 6 Eri kustannusfunktioiden vaikutus sisällön generoinnin keskimääräiseen viiveeseen

Kuviossa pystyakselilla kuvataan olion generoimisen keskimääräinen L_g-viive sekunteina. Kuvion 6 simulointituloksista nähdään, että pyynnön palvelemisen viive laskee huomattavasti kustannusfunktioilla, joiden laskenta pohjautuu olion todennäköisyyteen p_i tulla pyydetyksi. Merkittävin tekijä näiden kustannusfunktioiden tehokkuuden takana on p_i-arvon pohjautuminen zipfiläiseen jakaumaan. Ennaltageneroimalla välimuistiin ne oliot, jotka tulevat todennäköisimmin pyydetyiksi, saadaan keskimääräistä sisällön generoinnin viivettä laskettua, sillä ennaltageneroidut oliot palvellaan välimuistista ajassa $L_g = 0$.

Tutkimuksessa tarkastelun kohteena oleva $OTTM$-ahne kustannusfunktio pohjautuu p_i-arvoon vain toisen osatekijänsä osalta, joten sen nähdään sijoittuvan pyyntöjen keskimääräisten viiveiden vähentämisessä kohtalaisesti. Zipfiläiseen jakaumaan pohjautuvan p_i-arvon vaikutus vähenee välimuistin täyttöasteen kasvaessa jakauman muodosta johtuen. Välimuistin täyttöasteen ylittäessä noin 60 % raja-arvon, saa $OTTM$-ahne kustannusfunktio parhaan tuloksen keskimääräisen sisällön generoinnin viiveen laskemisen suhteen.

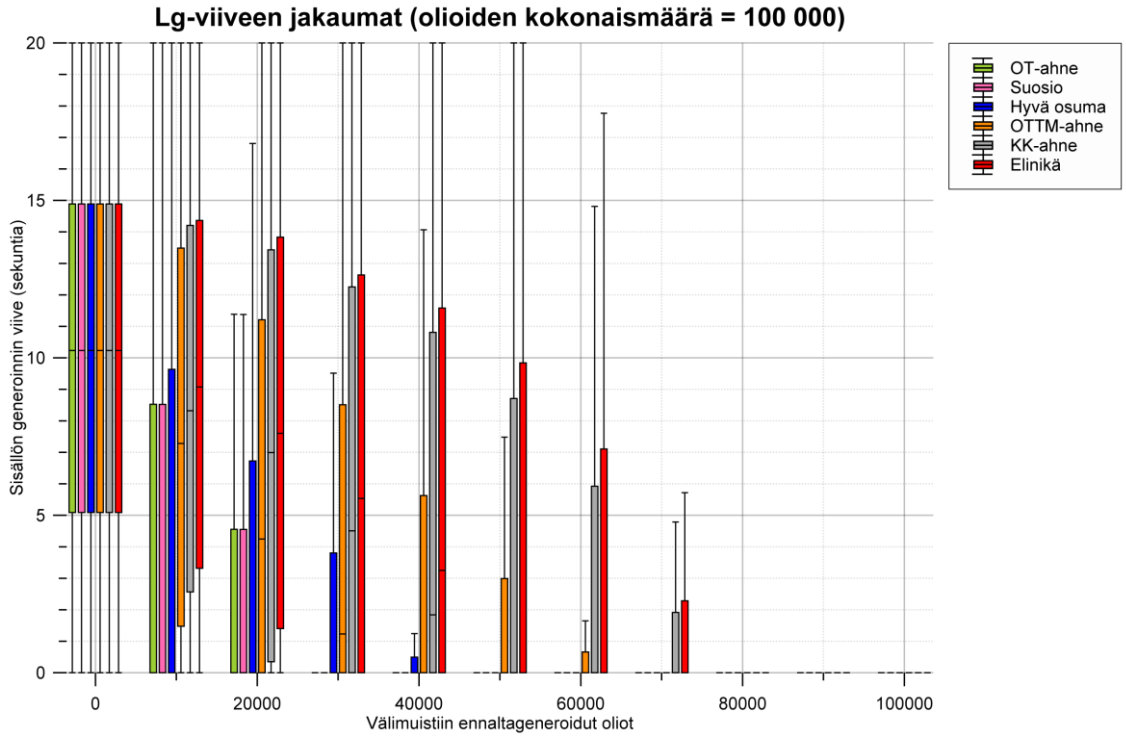
Lineaarisesti välimuistiolioiden generoinnin keskimääräistä viivettä laskevat kustannusfunktiot: KK-ahne sekä keskimääräiseen elinikään pohjautuva. Nämä eivät tähtää välimuistin osumatarkkuuden parantamiseen, vaan pyrkivät täyttämään välimuistia, haittavaikutuksiltaan mahdollisimman kevyillä olioilla. Edellä mainittujen kustannusfunktioiden kuvaajissa on havaittavissa pelkästään simulaation satunnaislukugeneroinnin aiheuttamaa vaihtelua. Tässä alaluvussa tarkastellun mittarin simulointitulokset välimuistin muilla oliomäärillä ovat nähtävissä liitteessä (LIITE 4).

6.5 Pyynnön palvelemiseen kuluvan ajan jakaumat

Tässä alaluvussa esitetään simulaatiotulokset sisällön generoinnin keston jakaumia esittävän mittarin osalta. Simulaatiotulokset näyttävät millä kustannusfunktiolla saadaan aikaiseksi mahdollisimman tiivis L_g-viiveen arvojen jakauma, mahdollisimman pienellä välimuistin täyttöasteella. Edellisen alaluvun tavoin, myös tämä simulaatio pohjautuu zipfiläiseen jakaumaan, jota mukaillen pyynnöt kohdistetaan tunnettujen välimuistiolioiden joukon S olioiden kesken. Oletuksena tämänkin mittarin simuloinnissa on, että yhtä pyyntöä vastaa websovelluksessa vain yksi välimuistiolio ja ettei pyynnön palvelemisen aikaan lasketa muuta kuin tarkastelun kohteena oleva L_g-viive.

Seuraavalla sivulla esitetyssä kuviossa (kuvio 7) nähdään välimuistiolioiden generointien viiveiden jakaumat laatikkodiagrammina, jossa vaaka-akselilla kuvataan välimuistiin ennaltageneroitujen olioiden lukumäärää. Kaavion pystyakselilla esitetään sisällön generoinnin viivettä sekunteina. Lähtötilanteessa yhtään oliota ei ole ennaltageneroituna välimuistiin, jolloin kaikki kustannusfunktiot saavat saman tuloksen. Samassa nähdään, ettei satunnaisesti tuotettu lähtötilanteen välimuistiolioiden generointiviivettä esittävä jakauma ole täysin identtinen pystyakselin asteikon kanssa: mediaanit poikkeavat luvusta 10 sekä tämän lisäksi ylä- ja alakvartiilien viiveet poikkeavat hieman arvoista 15 ja 5. Tällä ei kuitenkaan ole vaikutusta tutkimuksen kannalta, sillä simulaation tavoitteena on vertailla eri kustannusfunktioiden keskinäistä suoriutumista monenlaisilla eriävillä lähtöarvojen joukoilla. Lisäksi satunnaislukugeneroinnin aiheuttamia vaikutuksia arvioidaan ajamalla simulaatioajot useaan kertaan sekä käyttämällä riittävän suurta määrää välimuistiolioita.

Edellisen alaluvun simulointituloksissa havaittiin, että olion i todennäköisyydellä p_i tulla pyydetyksi on suuri vaikutus keskimääräiseen pyynnön palvelemisen kestoon. Sama suuntaus on nähtävissä myös L_g-viiveen arvojen jakaumien kanssa. Esimerkiksi jo 20 % välimuistin täyttöasteella OT-ahne sekä suosioon pohjautuva kustannusfunktio saavat simulaatiossa L_g-viiveen mediaanikseen arvon 0.



KUVIO 7 Eri kustannusfunktioiden vaikutus sisällön generoinnin viiveen jakaumiin

Laatikkodiagrammista on selvästi nähtävissä p_i-arvon merkitsevyys osana kustannusfunktion laskentakaavaa. Vain osittain p_i-arvoa hyödyntävä työmäärän panoksen huomioiva $OTTM$-ahne kustannusfunktio saa sisällön generoinnin viiveen mediaaniarvokseen arvon 0 vasta noin 40 % välimuistin täyttöasteen paikkeilla. Nähdään ettei $OTTM$-ahne kustannusfunktio kykene tasoittamaan käyttäjän kokemaa viivettä, eikä myöskään vähentämään sitä tehokkaammin kuin muut p_i-arvoon pohjautuvat kustannusfunktiot. Merkittävin syy on zipfiläisen jakauman mukainen web-sovellukseen kohdistuvien pyyntöjen jakauma ja samaan jakaumaan pohjautuvien kustannusfunktioiden vastaavuus p_i-arvojen osalta.

Edellisen alaluvun tutkimustulosten tavoin KK-ahneen sekä keskimääräiseen elinikään pohjautuvien kustannusfunktioiden jakaumat noudattavat lineaarista kaavaa, eivätkä ole siten viiveen tasaamisen näkökulmasta tehokkaita. Tässä alaluvussa tarkastellun mittarin simulointitulokset välimuistin muilla oliomäärillä ovat nähtävissä tutkielman lopussa, liitteessä (LIITE 5).

6.6 Keskeisten simulaatiomuuttujien varioinnin vaikutus

Tämän alaluvun simulointitulokset esittävät eri muuttujien varioinnin vaikutukset. Seuraavissa alakohdissa tarkastellaan välimuistiolion keskimääräisen eliniän, välimuistiolion generointiin kuluvan keskimääräisen ajan sekä väli-

muistin käytön pyyntötiheyden vaihteluiden vaikutuksia simulointituloksiin eri mittareittain.

6.6.1 Olion kesimääräinen elinikä

Olion i keskimääräinen elinikä l_i valitaan satunnaisesti oletusarvon $0 - 10^5$ lisäksi myös väleiltä $0 - 10^4$ ja $0 - 10^6$, jotta nähdään keskimääräisen eliniän muutoksen vaikutukset.

Simulaatiotulokset osoittavat, että olion keskimääräisen eliniän kasvattamisella on havaittavissa kaistankulutuksen säästöä. Samassa nähdään, että välimuistin osumatarkkuus on sitä korkeampi, mitä pidempiä keskimääräiset olioiden eliniät ovat, kun ennaltagenerointi ei ole käytössä. Wu ja Kshemkalyani (2004) päätyvät myös samoihin lopputuloksiin OT- ja KK-mittareiden simulointituloksissa olion keskimääräisen eliniän arvoavaruutta varioitaessa. Heidän mukaansa vain tarvittaessa generoitaessa kertyvä osumatarkkuus on olioiden eliniän noustessa korkeampi, sillä välimuistiolioilla on tästä johtuen korkeammat tuoreustekijät.

Välimuistiolioiden sisällön generoinnin viiveen L_g keskimääräisissä kestoissa ei ole havaittavissa muutoksia olion keskimääräisen eliniän muutosten johdosta, sillä kyseinen mittari ei pohjaudu välimuistiolion keskimääräiseen elinikään. Työmäärän panosta mittaava mittari saa välimuistiolioiden elinikään suoraan verrannollisesti suhteutuvia arvoja. Nämä työmäärän panokset ovat oletusarvoon verraten joko kymmenen kertaa pienempiä tai suurempia, vaihdeltaessa keskimääräisen eliniän maksimiarvoa 10^4, 10^5, 10^6 sekunnin välillä. Verrannollisuus pohjautuu siihen, että välimuistissa olevan välimuistiolion vanhetessa syntyy uudelleengeneroinnin tarve, joka puolestaan kasvattaa kustakin välimuistioliosta johtuvaa ennaltageneroinnin aiheuttamaa ylimääräistä työmäärän panosta.

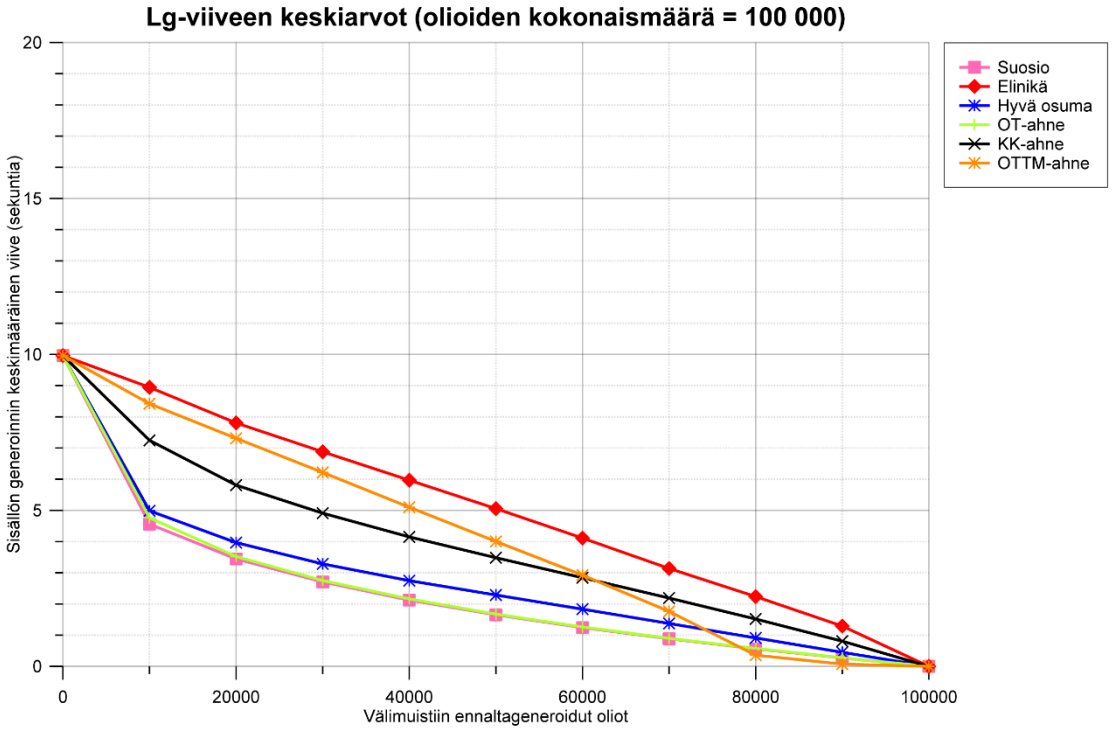
6.6.2 Välimuistin käytön pyyntötiheys

Oletusarvoista välimuistin käytön pyyntötiheyttä a muunnellaan käyttäen arvoja: $a = 0{,}005$; $a = 0{,}1$; $a = 1$ pyyntöä sekunnissa. Tässä alaluvussa esitetään pyyntötiheyden vaikutukset välimuistin ja ennaltageneroinnin simuloinnissa.

Välimuistin käytön pyyntötiheydellä nähdään olevan selkeä vaikutus vain tarvittaessa tehtävään välimuistiolioiden generointiin ja siitä johtuviin tarkastelun kohteina olevien mittareiden mittaamiin panoksiin. Muuttujan a varioinnin simulointitulokset osoittavat, että esimerkiksi vain tarvittaessa tehtävien generointien osumatarkkuuden panos kasvaa pyyntötiheyden nousun myötä. Osumatarkkuuden panoksen nousun selittää se, että välimuistissa jo olevaan välimuistiolioon kohdistuu sitä todennäköisemmin osuma, mitä enemmän websovellukseen saapuu pyyntöjä. Simulointitulokset osoittavat myös, että kaistankulutuksen lähtöarvo on korkeammalla muuttujan a arvon ollessa korkeampi. Tämä johtuu siitä, että vain tarvittaessa tehtävien välimuistiolioiden generointien kaistankulutuksen panos suurenee kokonaispyyntömäärän kasvaessa.

Myös Wu ja Kshemkalyani (2004) päätyvät samoihin johtopäätöksiin muuttujan a varioinnin osalta OT- ja KK-simulaatioissaan. Erot eri kustannusfunktioiden suoriutumisessa muiden tarkasteltavien mittareiden osalta ovat pienet, kun muuttujan a arvo on joko 0,005 tai 0,1. Nostettaessa välimuistin käytön pyyntötiheyttä aina arvoon $a = 1$ saakka, nähdään seuraavissa kuvioissa esitettyjä havaintoja.

Sisällön generoinnin keskimääräisen L_g-viiveen mittarin simulointituloksissa kuviossa (kuvio 8) nähdään, että KK-ahne kustannusfunktio saa $OTTM$-ahnetta kustannusfunktiota parempia keskimääräisen sisällön generoinnin viiveen arvoja.



KUVIO 8 Pyyntötiheyden kasvattamisen ($a = 1$) vaikutus sisällön generoinnin keskimääräiseen viiveeseen

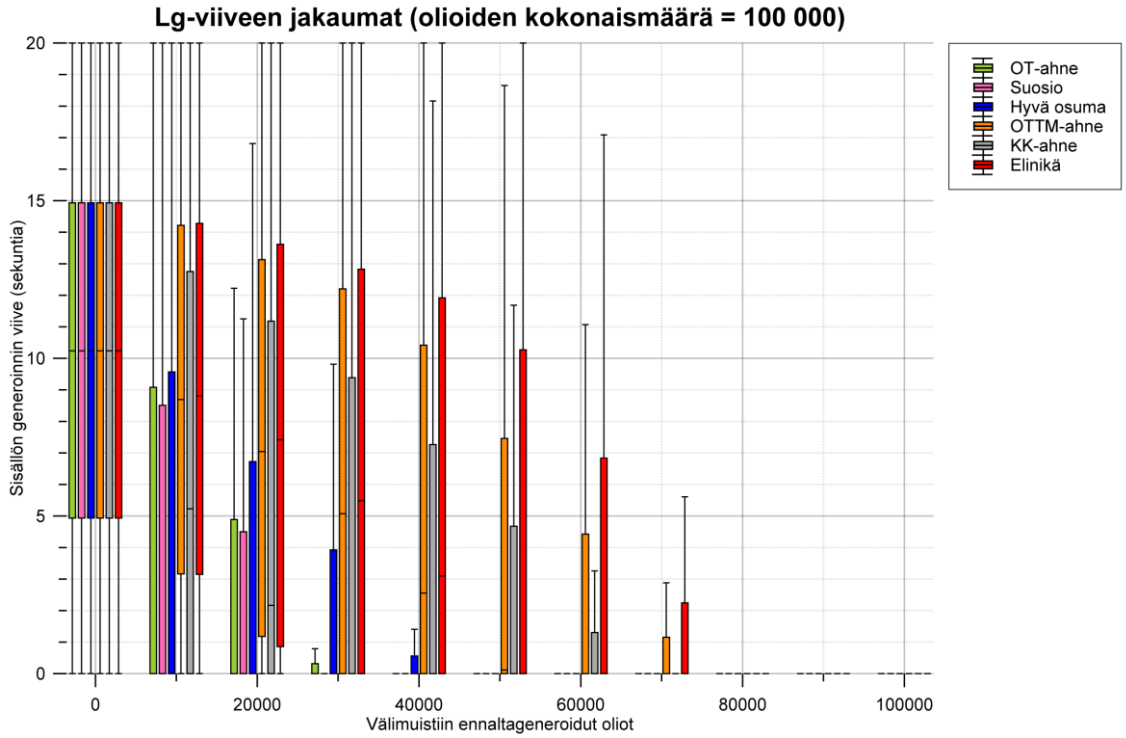
KK-ahneen kustannusfunktion paremman suoriutumisen taustalla on tutkielman alaluvussa 4.2.5, yhtälössä 4 esitetty tuoreustekijä, jonka yhtälön mukaan tuoreustekijän arvot lähenevät lukua 1 kun muuttujan a arvo kasvaa. KK-ahneessa kustannusfunktiossa olioiden ennaltagenerointijärjestys määräytyy oliokohtaisten kaistankulutuksen panosten pohjalta pienimmästä aloittaen. Alaluvun 4.2.5 yhtälössä 10 esitetty olion i kaistankulutuksen panos saa sitä pienempiä arvoja, mitä lähempänä lukua 1 tuoreustekijä on. Täten tuoreustekijältään suurten välimuistiolioiden kaistankulutuksen panos pienenee, jolloin juuri nämä oliot päätyvät KK-ahneen kustannusfunktion ennaltagenerointijärjestyksessä kärkipäähän. Muuttujan a kasvattaminen vaikuttaa suureen määrään välimuistiolioita, joiden kaikkien tuoreustekijöiden arvot lähenevät lukua 1. Kysymys miksi tästä oliojoukosta valitaan KK-ahneessa kustannusfunktiossa ensin

ennaltageneroitaviksi oliot, jotka pienentävät keskimääräistä L_g-viivettä selittyy yhtälössä 10 nähtävällä muuttujien a ja p_i tulolla. Muuttujan a arvon kasvaessa muuttujan p_i merkitsevyys kasvaa. Aiemmissa L_g-viiveen keskiarvon simulointituloksissa havaittiin, että pyynnön palvelemisen viive laskee huomattavasti kustannusfunktioilla, joiden laskenta pohjautuu olion todennäköisyyteen p_i tulla pyydetyksi.

Edellä kuvattuun kaistankulutuksen panokseen verrattavissa oleva, $OTTM$-ahneessa kustannusfunktiossa hyödynnettävä työmäärän panos on nähtävissä kaavassa 11. Tarkasteltava $OTTM$-ahne kustannusfunktio pyrkii valitsemaan toisen osatekijänsä osalta mahdollisimman suuria työmäärän panoksen arvoja saavuttavat välimuistioliot. Kuten aiemmin tarkastellussa yhtälössä 10, myös yhtälössä 11 muuttujan a arvon kasvaessa muuttujan p_i merkitsevyys kasvaa, jolloin yhä suurempi osa olioista saa pieneneviä työmäärän panoksen tuloksia. Samoista syistä johtuen myös kaavan 7 avulla laskettavat OT-panoksen tulokset saavat pienempiä lähemmäs toisiaan osuvia arvoja. OT-panoksen kaava kuvaa oliokohtaista ylimääräistä ennaltageneroinnin aiheuttamaa työmäärän panosta, joka on sitä pienempi, mitä useammin välimuistiolio päätyy pyyntöjen aikaansaamana jo valmiiksi välimuistiin.

Kuviossa 8 nähtiin $OTTM$-kustannusfunktion lähes lineaarinen kuvaaja, sillä suuri joukko kustannusfunktion laskemista oliokohtaisista $OTTM$-kasvutekijöistä saa lähelle toisiaan sijoittuvia arvoja. Yleisesti ottaen huomataan, että yhtälössä 19 esitetty $OTTM$-ahneen kustannusfunktion laskukaava menettää muuttujan a arvon kasvaessa erottelukykyään. Sama erottelukyvyn heikkeneminen on havaittavissa myös muiden ahneiden kustannusfunktioiden panosten laskukaavoissa nostettaessa pyyntötiheyttä huomattavasti.

Kuviossa (kuvio 9) nähdään muuttujan a arvon ollessa 1 esiin tuleva $OTTM$-ahneen kustannusfunktion heikentynyt kyky pienentää ja tasoittaa sisällön generoinnin viivettä. Muuttujan a variointi sisällön generoinnin keston jakaumien suhteen mukailee aiemmin tässä alaluvussa tarkastellun pyyntöjen keskimääräisten viiveiden simulaation havaittua kaavaa sekä taustalla olevia päätelmiä. $OTTM$-ahneen kustannusfunktion menettäessä erottelukykyään pyyntötiheyden noustessa, lähenee se suoriutumisensa suhteen keskimääräiseen elinikään pohjautuvan kustannusfunktion kuvaajaa. Ero on havaittavissa esimerkiksi aiemmin $OTTM$-ahnetta kustannusfunktiota heikommin suoriutuneeseen KK-ahneeseen kustannusfunktioon nähden, joka nyt suoriutuu näistä kahdesta paremmin.



KUVIO 9 Pyyntötiheyden kasvattamisen ($a = 1$) vaikutus sisällön generoinnin viiveen jakaumiin

Nostettaessa muuttujan a arvoa yhä korkeammalle huomataan, että vain tarvittaessa pyydettäessä tehtävien välimuistiin hakuja kuvaavat panokset kasvavat huomattavasti. Tästä voidaan päätellä, että kaavojen 4, 10, 11 kuvaamat oliokohtaiset ennaltageneroinnista saatavat lisähyödyt eli panosten erotteluky-vyt menettävät tehokkuuttaan. Esimerkiksi muuttujan a arvon ollessa 10 nähdään, että vain tarvittaessa tehtävien välimuistiin hakujen lähtöpanos osuma-tarkkuuden osalta on jo noin 80 % kokonaispanoksesta. Samoin kaistankulutus joka aiheutuu web-sovelluksen normaalista toiminnasta, ennaltageneroinnista riippumatta on jo hyvin korkealla. Täten voidaan todeta, etteivät ennalta-generoinnilla saatavat hyödyt ole enää kovin merkittäviä web-sovelluksen pyyntötiheyden kasvaessa.

6.6.3 Olion generointiin kuluva keskimääräinen aika

Olion i generointiin kuluva keskimääräinen aika t_i valitaan satunnaisella kaa-valla myös pidemmältä sekuntiväliltä $1 - 100$. Tämän lisäksi satunnaisesta kaavasta poikkeavasti olion i generointiin kuluva keskimääräinen aika valitaan myös nousevalla sekä laskevalla painotetulla jakaumalla.

$OTTM$-ahneen mittarin tarkastelun näkökulmasta on mielenkiintoista va-rioida olion generointiin kuluvaa keskimääräistä aikaa. Keskimääräisen gene-rointiin kuluvan ajan jakaumien muuntelu voisi olla keino löytää web-sovellustyypin ominaisuudet, joissa $OTTM$-ahne kustannusfunktio suoriutuisi muita kustannusfunktioita paremmin esimerkiksi sisällön generoinnin viivei-

den jakaumien tasaamisen suhteen. Seuraavana esitetään olion generointiin kuluvan keskimääräisen ajan jakauman varioinnin vaikutukset aiemmin esiteltyjen mittareiden näkökulmasta.

Simulaatiot osoittavat, ettei L_g-viiveen satunnaisjakauman varioinnilla $1-20$ ja $1-100$ sekunnin välillä ole havaittavissa muutoksia eri kustannusfunktioiden keskinäisen suoriutumisen tai niiden kuvaajien suhteen. Jakaumat ovat samanmuotoiset ja siten ennaltageneroinnin painotus pysyy järjestykseltään samana simuloinnin satunnaislukugeneroinnista johtuvaa vaihtelua lukuun ottamatta.

Seuraavana simuloinnin kohteena on nouseva painotettu jakauma, jossa vain murto-osa olioista saa arvoltaan pienen generointiin kuluvan keskimääräisen ajan ja lineaarisesti edeten yhä suurempi osa olioista saa suuremman sisällön generointiin kuluvan keskimääräisen ajan. Tämän simulaation tuloksissa selviää, ettei kustannusfunktioiden keskinäinen suoriutuminen muutu huomattavasti yhdenkään mittarin suhteen sisällön generoinnin kestojen jakauman painotuksen myötä. Havaittavana muutoksena on korkeampi sisällön generoinnin viiveen keskiarvo lähtötilanteessa kustannusfunktiosta riippumatta. Lisäksi sisällön generoinnin viiveiden jakaumat sijoittuvat oletusarvoista jakaumaa korkeammalle. Vastaavasti simulaatioissa havaitaan, ettei laskeva painotettu jakauma myöskään muuta huomattavasti kustannusfunktioiden keskinäistä suoriutumista. Vaikutuksia on havaittavissa lähinnä TM-mittarin kokonaistyömäärän pienenemisen suhteen. Lisäksi havaitaan L_g-viiveen keskimääräisten arvojen pienenemistä ja yhtälailla L_g-viiveen jakaumien keskittymistä pienempiin arvoihin.

Tämän alaluvun simulointituloksissa huomattiin, ettei simulaatioissa alustettavan L_g-viiveen jakauman variointi tuottanut tutkimusongelman näkökulmasta lisäarvoa. Variointi ei esimerkiksi tuottanut oletettuja tutkimustuloksia, joissa $OTTM$-ahne kustannusfunktio olisi suoriutunut muita kustannusfunktioita paremmin sisällön generoinnin viiveiden jakaumien tasaamisen suhteen, kun käytössä olisi ollut esimerkiksi tietyllä tavalla painotettu L_g-viiveiden jakauma.

6.7 Luvun yhteenveto

Tässä luvussa esiteltiin simulointiympäristön tuottamat simulointitulokset graafisina kuvaajina. Kukin simulaatio vastasi tiettyä mittaria, jonka tuottamat simulaatiotulokset esiteltiin ja havaittuihin poikkeamiin johtavat syyt perusteltiin. Lisäksi tutkittiin simulaatioiden keskeisten muuttujien variointien vaikutukset, joilla pyrittiin etsimään web-sovelluksen tyyppiominaisuuksia, joissa $OTTM$-ahneesta kustannusfunktiosta saataisiin mahdollisimman suuret hyödyt.

Web-sovelluksen tyyppiominaisuuksia varioivista simulaatioista huolimatta simulaatiotulokset eivät osoittaneet $OTTM$-ahneen kustannusfunktion hyödyllisyyttä kirjallisuuden jo aiemmin tuntemiin kustannusfunktioihin nähden. Esimerkiksi tutkimusongelman kannalta keskeisin, sisällön generoinnin

viiveen jakaumien varioinnin vaikutusten vähyys $OTTM$-ahneen kustannusfunktion suoriutumiseen osoittaa muiden variointiin pohjautuneiden simulointitulosten ohella, ettei web-sovelluksen tässä tutkimuksessa tehdyillä tyyppiominaisuuksien muuntelulla ole merkittävää vaikutusta esitetyn kustannusfunktion hyvyyteen.

Seuraavassa luvussa keskitytään kirjallisuuskatsauksen tutkimustulosten sekä tässä luvussa esitettyjen simulaatioiden tulosten tarkempaan analysointiin.

7 JOHTOPÄÄTÖKSET

Tässä luvussa perehdytään tutkielman tutkimustuloksiin ja verrataan niitä aiempien tutkimusten havaintoihin sekä tarkastellaan ennakko-oletusten paikkansapitävyyttä. Luvun tavoitteena on myös arvioida ehdotetun $OTTM$-ahneen kustannusfunktion hyödyllisyyttä osana, tässä tutkimuksessa mallinnettua ennakoivan välimuistiratkaisun mallia. Tämän lisäksi tehdään johtopäätöksiä ennakoivan välimuistiratkaisun mallin ja sen pohjalta rakennettavissa olevan reaalimaailman toteutuksen hyödyllisyydestä. Edellä kuvattujen pohdintojen ohessa vastataan tutkimustulosten pohjalta myös osatutkimusongelmiin:

- Miten uusi tutkielmassa esitelty kustannusfunktio suoriutuu ja asettuu jo aiemmin tunnettuihin kustannusfunktioihin nähden?
- Millä kustannusfunktiolla viivettä laskeva ja tasoittava ennakoiva välimuistiratkaisu kannattaa toteuttaa yksilöityä sisältöä sisältävässä web-sovelluksessa?

Lopulta tässä luvussa esitellään tutkielman aihealueeseen kohdistuvia erillisiä johtopäätöksiä, jotka osaltaan nostavat esiin jatkotutkimuksen kannalta mielenkiintoisia kysymyksiä.

7.1 Havainnot ja aiempi tutkimus

Tutkielman havainnot suhtautuvat aiempaan aihepiirin tutkimukseen hyvin. Esimerkiksi tämän tutkimuksen sekä Wun ja Kshemkalyanin (2004) simulointien suhteen saadut lopputulokset ovat identtisiä molemmissa tutkimuksissa esiintyneiden yhteisten kustannusfunktioiden osalta.

Myös Dutta ym. (2007) esittävät tutkielman simulaatiotulosten kanssa yhdensuuntaisia ajatuksia. Heidän mielestään ennaltageneroinnin haasteena oli, etteivät jotkin välimuistioliot säily välimuistissa riittävän pitkään, jolloin ne eivät ehdi olemaan hyödyksi yhdellekään myöhemmälle sivupyynnölle. Toinen,

yhtälailla aiempien tutkimusten suhteen yhdensuuntainen havainto on Probirin ja Rau-Chaplinin (2006) toteamus: mikäli jonkin välimuistiolion päivitystiheys on tiiviimpi kuin välimuistiolioon kohdistuvien pyyntöjen tiheys, ei ole kannattavaa tallettaa tällaista välimuistioliota välimuistiin. $OTTM$-ahne kustannusfunktio ei suoranaistesti hae hyötyjä näiden aiemmissa tutkimuksissa esitettyjen havaintojen puitteissa, vaan käyttää laskennassaan lähes päinvastaista tavoitetta, kuormitusta. Täten molemmat aiempien tutkimustulosten havainnot ovat nähtävissä $OTTM$-ahneen kustannusfunktion osakseen saamaa kritiikkiä tukeviksi havainnoiksi. Myös muiden tutkimusten simulaatioista riippumattomat päätelmät tukevat tämän tutkimuksen havaintoja. Esimerkiksi havainnot $OTTM$-ahneen kustannusfunktion kaistankulutuksen kritiikin suhteen ovat yhteneviä.

Tutkielman kirjallisuuskatsauksessa olevaan ja seuraavassa esiteltyyn Kroegerin ym. (1997) väitteeseen ei kuitenkaan voida luoda vertailukelpoista näkemystä. He esittivät, että välimuistin ja ennaltageneroinnin yhdistelmästä on saatavissa jopa 57 % parannus viiveeseen, kun taas pelkän välimuistin käyttö aikaansaa enintään 23 % parannuksen. Tämän tutkimusten havaintojen suhteen vertailua voitaisiin suorittaa ainoastaan, mikäli tiedossa olisi heidän koejärjestelyissä käytössä ollut kokonaisvaltainen välimuistin käytön pyyntötiheys a. He saavat välimuistin ja ennaltageneroinnin osumatarkkuuden arvoksi luvun $0{,}57$. Tämän tutkimuksen koetulokset osoittavat kuitenkin, että muuttujan a lähestyessä ääretöntä, vain tarvittaessa tehdyn välimuistiolion generoinnin aikaansaama osumatarkkuus lähestyy arvoa 1. Täten, tuntematta käytettyä muuttujan a arvoa, ei voida rakentaa vertailupohjaa edellä mainittuihin tutkimustuloksiin. Tämän lisäksi heidän tutkimuksessaan mitataan laskennassa mukaan muukin, kuin pelkkä välimuistisisällön osuus L_g-viiveestä. He mainitsevatkin, että loput 43 % eli jäljelle jäänyt viive koostuu osista, joita ei ole mahdollista saada välimuistiin, eikä ennaltageneroinnin avulla nollata. Tällaisiksi asioiksi he mainitsivat esimerkiksi välimuistiolioihin kohdistuvat ensimmäiset pyynnöt, joita ei koskaan voida osata ennaltageneroida kustannusfunktiopohjaisella lähestymistavalla. He sisällyttivät tähän 43 % osuuteen myös web-sovelluksesta itsestään johtuvan viiveen. Tästä johtuen tämän tutkimuksen tutkimustuloksia muihin tutkimuksiin verratessa tulee huomioida esimerkiksi tutkielman simuloinneissa tehty rajaus, jossa oletettiin, että L_g-viive on mahdollista saada poistettua kokonaisuudessaan välimuistin ja ennaltageneroinnin avulla.

Simulaatioissa havaittiin, että web-sovelluksen normaali käyttö ilman ennaltagenerointia tuottaa tietyn määrän välimuistiolioita välimuistiin, jonka ansiosta välimuistista saatava passiivinen hyöty nousee. Web-sovelluksen ennaltageneroivan välimuistin nähtiinkin olevan tehokkaimmillaan tilanteessa, jossa pelkkien käyttäjiltä saapuvien pyyntöjen aiheuttamat vain tarvittaessa tehtävät välimuistiintallennukset eivät aikaansaaneet liian korkeaa välimuistin käytön pyyntötiheyttä ja siten jo lähtötasoltaan korkeaa välimuistin osumatarkkuutta. Havainnot osoittivat siten, että ennaltageneroinnista saatavat hyödyt vähenevät, mitä suurempi välimuistin käytön pyyntötiheys a on. Samalla havaittiin ahneiden kustannusfunktioiden menettävän muuttujan a arvon kasvaessa erottely-

kykyään. Erottelukyvyn taustalla olevaksi tekijäksi osoittautui tuoreustekijä, jonka arvojen nähtiin lähenevän lukua 1 muuttujan a arvon kasvaessa. Simulaatioissa havaittiin myös, että web-sovelluksen normaalista toiminnasta aiheutuva ennaltageneroinnista riippumaton kaistankulutus on jo hyvin korkealla välimuistin käytön pyyntötiheyden ollessa korkealla.

Toinen osatutkimusongelma: ”miten uusi tutkielmassa esitelty kustannusfunktio suoriutuu ja asettuu jo aiemmin tunnettuihin kustannusfunktioihin nähden” saa vastauksen tässä tutkimuksessa suoritettujen simulaatioiden avulla. Simulaatiotuloksissa päädyttiin lopulta siihen, ettei ehdotettu $OTTM$-ahne kustannusfunktio kykenekään vähentämään eikä tasoittamaan käyttäjän kokemaa viivettä tehokkaammin kuin kirjallisuuden jo aiemmin tuntemat kustannusfunktiot. Edellisen luvun simulaatiotulosten kuvioihin 3 ja 4 pohjautuen ehdotettu $OTTM$-ahne kustannusfunktio sijoittuu järjestysluvullisilla arvosanoilla $3 > OT > 4$ sekä $KK = 6$, kirjallisuuskatsauksessa esitetyn taulukon (taulukko 2) sijoituksiin suhteutettuna. Kuitenkaan näin pintapuolisesti $OTTM$-ahneen kustannusfunktion suoriutumista ei tule arvioida, sillä muut johtopäätöksissä käsiteltävät kritiikinaiheet painavat edellä esitettyä järjestysluvullista sijoittumista enemmän. Tutkielmassa pyrittiin simulaatioiden keskeisiä muuttujia varioimalla löytämään web-sovelluksen tyyppiominaisuuksia, joissa $OTTM$-ahne kustannusfunktio olisi ollut muita kustannusfunktioita hyödyllisempi. Simulaatiotulokset kuitenkin osoittivat, ettei esitetty kustannusfunktio, muuttujien varioinneista huolimatta suoriutunut muita vertailtavia kustannusfunktioita paremmin.

$OTTM$-ahneen kustannusfunktion ansiosta keskimääräinen L_g-viive lähenee nollaa kaikkien mahdollisten olioiden kesken tasaisesti. Tämä tasainen viiveiden pieneneminen ei kuitenkaan vastaa reaalimaailman tilannetta, jossa web-sovellusta kuormitetaan zipfiläiseen jakaumaan pohjautuvalla pyyntöjoukolla. Haasteena aiempia kirjallisuuden tunnistamia kustannusfunktioita paremman laskentakaavan löytämiseen on siinä, että web-sovelluksen käyttö perustuu aina muuttujan p_i jakaumaan. Tämä vaikuttaa suoraan siihen, mitkä oliot ovat kannattavimpia ennaltageneroida välimuistiin tapauksissa, joissa osumatarkkuus, viiveen pienentämiseen tai jopa viiveen tasoittamiseen tähtäävät asiat ovat tavoitteena.

Riittämättömän hyödyllisyytensä lisäksi $OTTM$-ahne kustannusfunktio saa kritiikkiä ylivoimaisesti suurimman kaistankäyttönsä suhteen. Simulaatiotuloksissa nähtiin, että $OTTM$-ahne kustannusfunktio on kaistankäytön kannalta erittäin kallis. Esimerkkinä kaistankulutuksesta $OTTM$-ahne kustannusfunktio saa aikaan jo 10 % välimuistin täyttöasteen kohdalla lähes maksimaalisen kaistankulutuksen. Vertailukohtana voidaan käyttää esimerkiksi OT-ahnetta kustannusfunktiota, joka myös aikaansaa kaistankulutusta, mutta monta kertaluokkaa $OTTM$-ahnetta kustannusfunktiota vähemmän. Simuloinnin koetuloksissa todettiinkin OT-ahneen kustannusfunktion kaistankäytön saaneen jo osakseen kritiikkiä, joten monta kertaluokkaa korkeampi $OTTM$-ahneen kustannusfunktion kaistankulutus voidaankin nähdä tätäkin haitallisempana. Simulointitulosten yhteydessä todettiin myös, että Wu ja Kshemkalyani (2004) kehottivat vält-

tämään OT-ahneen kustannusfunktion käyttöä, ellei sitä tasapainotettaisi ylimääräiseltä kaistankäytöltä välttymiseksi esimerkiksi OT/KK-ahneella kustannusfunktiolla. Lisähuomiona mainittakoon, että esimerkiksi nykyisissä pilviympäristöissä laskutusperusteena on usein käytetyt resurssit, kuten verkkoliikenne ja käytetty laskentakapasiteetti. Täten kuormitusta tavoitteleva kustannusfunktio voidaankin nähdä jo kustannussyistäkin erittäin haitallisena.

$OTTM$-ahne kustannusfunktio esiteltiin tämän tutkielman edetessä. Samassa yhteydessä mainittiin esitetyn kustannusfunktion aikaansaaman kaistankäytön rajoittamisen vastuun jäämisestä välimuistiratkaisulle. Kustannusfunktion esittelyn yhteydessä mainittiin myös, että kaistankäyttö olisi osittain perusteltua, sillä kaistankäyttö on väistämätöntä osumatarkkuutta runsaasti nostavien välimuistiolioiden kohdalla. Tämä ennakko-oletus pitää edelleen paikkaansa, mutta simulaatiotulokset näyttivät esitetyn kustannusfunktion kaistankäytön nousevan huippulukemiin logaritmisesta asteikosta huolimatta jo hyvin pienillä ennaltageneroiduilla oliomärillä. Tämä herättää huolen siitä, että miten yksikään ennakoivan välimuistiratkaisun sisältämä kaistankäytöllisen kulutuksen rajoittamiseen keskittynyt mekanismi kykenisi toimimaan näiden havaintojen valossa ilman, että ennaltagenerointi olisi jatkuvasti rajoitettuna lähes kokonaan pois käytöstä. $OTTM$-ahneen kustannusfunktion esittelyn yhteydessä mainittiin myös, että: ”välimuistitoteutuksen ennaltageneroinnin raja-arvot tulee asettaa riittävän tiukoiksi, jottei ennaltageneroinnista koidu liian suuria haittoja suhteessa siitä saataviin hyötyihin”. Tutkimustulosten valossa kuitenkin nähdään, etteivät $OTTM$-ahneen kustannusfunktion tapauksessa saavutettavat hyödyt ole muihin kustannusfunktioihin nähden paremmat. Samoin nähdään, että tarkasteltavan kustannusfunktion haitat ovat jo pienillä ennaltageneroitavilla oliomäärillä huomattavat. Tämän kappaleen pohdinnan pohjalta voidaan todeta, ettei raja-arvojen määrittämistä nähdä kovinkaan käyttökelpoisena mekanismina $OTTM$-ahneen kustannusfunktion kohdalla. Myös muut tutkimustulokset viittaavat siihen, että $OTTM$-ahne kustannusfunktio voidaan nähdä reaalimaailman web-sovelluksen kannalta ennakoivassa välimuistiratkaisussa haastavana, ellei lähes mahdottomana ennaltageneroitavien välimuistiolioiden valintaperusteena.

Tutkimuksen kolmas osatutkimusongelma: ”millä kustannusfunktiolla viivettä laskeva ja tasoittava ennakoiva välimuistiratkaisu kannattaa toteuttaa yksilöityä sisältöä sisältävässä web-sovelluksessa” saa vastauksen näissä johtopäätöksissä esitetyn pohdinnan avulla. Ennaltageneroivan välimuistiratkaisun kustannusfunktioksi kannattaakin valita OT/KK-ahne kustannusfunktio, jotta OT-ahneen kustannusfunktion osumatarkkutta tavoittelevat hyödyt saadaan käyttöön kuitenkin siten, että kaistankulutusta voidaan hallita.

Tutkielmassa esitetyt välimuistiratkaisun mallin vaatimukset ovat hyvä pohja reaalimaailman ratkaisun rakentamisen tueksi esimerkiksi vaatimusmäärittelyn pohjaksi. Kuitenkin suorituskyky ja resurssienkulutus tulee ottaa huomioon välimuistiratkaisua toteutettaessa. Simulaatioissa laskentaa tehtiin miljoonilla olioilla, joiden arvoja ei tarvinnut mitata tai laskea, sillä esimerkiksi keskimääräiset arvot voitiin olettaa tai satunnaistaa. Välimuistiratkaisun haas-

teena ja suurena riskinä onkin oheislaskennasta aiheutuva kasvava sisällön generoinnin viive, joka voi johtua web-sovelluspalvelimen laskentatehon hyödyntämisestä ennakoivan välimuistin ylläpitämiseen käyttäjien pyyntöjen palvelemisen sijasta. Mikäli edustapalvelinten prosessorikuormaa on vapaana ja tyypillinen L_g-viive koostuu taustajärjestelmien odottelusta, voidaan resurssienkäytön kannalta nähdä hyödyllisenä edustapalvelimen vapaan prosessorikapasiteetin hyödyntäminen ennaltageneroinnin vaatimaan laskentaan. Tällöin voitaisiin olla välittämättä esimerkiksi tutkimuksessa täysin huomiotta jätetyn ennakoivan välimuistiratkaisun laskennasta aiheutuvan kuormituksen vaikutuksista ennaltageneroinnista saatavaan kokonaissuorituskyvyn hyötyyn.

Ennakoivan välimuistiratkaisun malli sai tutkimuksen edetessä osakseen kritiikkiä myös arkkitehtuurillisista näkökulmista, sillä sen nähtiin heikentävän web-sovelluksen piirteitä sovellusarkkitehtuurin laadullisista näkökulmista. Suurimmaksi haasteeksi havaittiin web-sovellukseen tiiviisti integroidun ratkaisun negatiiviset vaikutukset web-sovelluksen testattavuuteen sekä muuntautumiskykyyn. Toisena arkkitehtuurillisena haasteena esitetyssä mallissa havaittiin rajoitus, etteivät taustajärjestelmien tiedot saaneet päivittyä muutoin, kuin web-sovelluksen välimuistiratkaisun toimesta. Välimuistiratkaisun ohi tapahtuneet taustajärjestelmätietojen muutokset aikaansaivat web-sovelluksen välimuistin joutumisen epäeheään tilaan.

Tässä tutkimuksessa nähtiin kustannusfunktioita, jotka pärjäsivät TM-mittarin suhteen hyvin. Nämä kustannusfunktiot voidaan nähdä taustajärjestelmille haitallisina sekä myös kaistankulutukseltaan erittäin raskaina. Tämä johtopäätös voidaan kääntää myös toisinpäin tutkimalla TM-mittarin, kuvion 5 simulointituloksia. Kuviosta nähdään, että olion keskimääräiseen elinikään pohjautuva, KK-ahne sekä hyvään osumaan perustuvat kustannusfunktiot saavat työmäärien panoksiensa suhteen taustajärjestelmille erittäin suosiollisia tuloksia. Täten nähdäänkin aiempien tutkimusten esittämien havaintojen lisäksi uudesta näkökulmasta se, että juuri tämä kustannusfunktiojoukko on taustajärjestelmien kuormituksen kannalta edullinen. Merkittävimpinä taustatekijöinä tämän löydöksen takana olevien kustannusfunktioiden laskennassa ovat: pitkän välimuistiolion elinajan tavoittelu, todennäköisimmin käytettäviksi päätyvien olioiden ennaltagenerointi välimuistiin tai näiden molempien taustatekijöiden yhdistelmä.

7.2 Jatkotutkimusaiheet

Edellisessä alaluvussa esitettyjen johtopäätösten ja havaintojen lisäksi tämä tutkimus herättää laajasti jatkotutkimusaiheita sekä parannusmahdollisuuksia, jopa tässä tutkimuksessa aiemmin tehtyihin esimerkiksi ennakoivan välimuistiratkaisun mallin arkkitehtuurillisiin lähtökohtiin. Seuraavana on listattuna laajempi joukko ajatuksia siitä, miten tämän tutkimuksen tuloksia voitaisiin hyödyntää.

Yleisesti havaitaan, että web-sisältö on talletettavissa välimuistiin helpommin, mikäli se ei ole yksilöityä. Tällaisessa tilanteessa välimuisteja voi sijaita kaikilla monikerroksisen web-sovelluksen arkkitehtuurin kerroksilla: käyttäjän ja alkuperäisen tiedonlähteen välillä. Päinvastaisessa tilanteessa, vain yhtä käyttäjää koskevan yksilöidyn sisällön välimuistiin tallettamisesta saatavat hyödyt ovat erittäin pienet, sillä yksilöityyn välimuistiolioon kohdistuu yleiskäyttöisempää välimuistioliota huomattavasti vähemmän osumia. Jotta voitaisiin vastata yksilöidyn sisällön aiheuttamiin haasteisiin ennaltageneroinnin ja välimuistin yhdistelmällä, tulee ennaltagenerointia ja välimuisteja rakentaa lähemmäs tiedon alkuperää esimerkiksi web-sovelluksen ja taustajärjestelmien välille, jossa esimerkiksi tietokantakyselyt voidaan helpommin irrottaa käyttäjäsidonnaisuudesta. Tällainen web-sovelluksen ja taustajärjestelmän välissä sijaitseva ennaltageneroiva välimuisti voisi toimia irrallisena itsenäisenä komponenttina, vastaanottaen syötteenä pelkästään web-sovelluksen sekä taustajärjestelmän suunnista tulevia viestejä sekä ohjaustietoja.

Aiemman kirjallisuuden pohjalta voidaan nostaa esiin Alin ym. (2011) määritelmä, jonka mukaan ennaltagenerointitekniikoita voidaan toteuttaa alkuperäisille web-sovelluspalvelimille, välityspalvelimille tai asiakasohjelmiin. Mielestäni, taustajärjestelmään ja sieltä takaisin kulkevan pyyntöliikenteen pohjalta muotoutuva oppiva taustajärjestelmäkohtainen ennakoiva välimuistiratkaisu olisi Alin ym. esittämään listaukseen nähden uusi erillinen innovaatio. Tällaisen ennakoivan välimuistin kannattavuus pohjautuu siihen oletukseen, että ennaltageneroinnista saatavat hyödyt paranevat lähempänä taustajärjestelmiä, kun kyseessä on dynaaminen tai yksilöity sisältö. Huomioitavaa on myös se, että mitä lähemmäs taustajärjestelmiä välimuisti sijoitetaan, sitä vähemmän kokonaisvaltaista kaikkien järjestelmäosien suhteen mitattavaa hyötyä välimuistista on saatavissa, sillä välimuistiolioon kohdistunut osuma tapahtuu vasta pyynnön kuljettua ensin usean arkkitehtuurikerroksen läpi.

Yllä esitettyyn pohjautuen seuraavien kappaleiden pohdinnoissa esitetyt kysymykset voisivat olla jatkotutkimuksen kannalta mielenkiintoisia ongelmia ja havaintoja.

Saavutettaisiinko parempi ennakoivan välimuistiratkaisun malli, mikäli ennakoiva välimuistiratkaisu toteutettaisiinkin web-sovelluksesta erillisenä, modulaarisena komponenttina, joka ei ottaisi kantaa web-sovelluksen toteutukseen, vaan toimisikin irrallisena kokonaisuutena ja ainoastaan mukautuisi komponentin itsensä läpi kulkevaan liikenteeseen. Tällä modulaarisella mallilla saataisiin vaatimuslistauksen taulukon (taulukko 1) vaatimukset V1, V2, V3 sekä V4 huomioitua ilman, että välimuistiratkaisun toteuttaminen monimutkaistaisi web-sovelluksen arkkitehtuuria ja toteutusta. Haasteena tutkielman luvussa 3 esitetyssä mallissa on myös se, että siitä saatava hyöty on täysin riippuvainen web-sovelluksen ohjelmoijan työnlaadusta. Ennakoivan välimuistiratkaisun integrointi web-sovellukseen voi pahimmillaan epäonnistua ja aikaansaadaan enemmän suorituskyvyn menetystä kuin hyötyjä. Haasteena on myös inhimillisen komponentin mukanaan tuoma virheen mahdollisuus, jossa web-sovellusohjelmoija voi aiheuttaa tiedon eheyden kannalta virhetilanteita.

Web-sovellukseen tiiviisti integroitu ennakoiva välimuistiratkaisu vaikeuttaa myös web-sovelluksen testattavuutta sekä muuntautumiskykyä ja siten heikentää web-sovelluksen piirteitä laadullisista näkökulmista.

Myös Dutta ym. (2007) toteavat tutkimuksessaan, että välimuistiin mahdollisesti talletettavat ehdolle tuotavat oliot joudutaan esittämään hyvinkin yksioikoisin perustein pohjautuen pelkästään web-sovelluksen ohjelmoijan tai ylläpitäjän tekemiin esivalintoihin. Tästä poiketen, voitaisiin web-sovelluksen ja taustajärjestelmän välille sijoittuvassa ennaltageneroivassa välimuistissa vaatimusten V1-V4 sijasta käyttää ennaltageneroinnin ohjaustietoina sekä välimuistiavaimina esimerkiksi xml-muotoisia välimuistiin kohdistuvia pyyntöjä sekä välimuistiavaimina näiden pyyntöjen pohjalta laskettua tiivistettä (engl. hash). Taustajärjestelmän käyttö tulisi tässä tapauksessa suunnitella siten, että pyynnöt olisi rakennettu mahdollisimman yleispäteviksi. Kuitenkin niin, ettei saatava tulosjoukko olisi liian suuri tai kysely muuten liian kuormittava taustajärjestelmien kannalta. Nämä yleiset taustajärjestelmiin kohdistuvat pyynnöt saavuttaisivat korkeampia p_i -todennäköisyyksiä ja siten tälle web-sovellusarkkitehtuurin tasolle sijoitettavasta ennaltageneroivasta välimuistista voisi olla saavutettavissa suhteessa enemmän hyötyä muihin arkkitehtuurin tasoihin nähden.

Jatkotutkimuksen kannalta oleellisina kysymyksinä näenkin: ennaltageneroivan välimuistiratkaisun optimaalisimman sijoituspaikan määrittämisen monikerroksisessa web-sovellusarkkitehtuurissa sekä ennaltageneroivan välimuistiratkaisun modulaarisen toteutuksen suunnittelun. Käytännön toteutuksessa tällaisen modulaarisen ennakoivan välimuistin voisi rakentaa taustajärjestelmän, esimerkiksi Web Services -rajapinnan oheen siten, että ennakoivan välimuistiratkaisun perustoteutus generoitaisiin esimerkiksi .NET-sovelluskehyksen ohjelmakoodigeneroinnilla Web Services -rajapinnan WSDL-kuvauksen (engl. web service description language) pohjalta. Tämän toteuttamiseen tarvittaisiin .NET-sovelluskehyksen avulla mallinnettava ohjelmakoodigeneraattori, jonka parametriksi annettaisiin tietyn taustajärjestelmän rajapinnan WSDL-kuvaus. Koodigeneroinnin lopputuotteena syntyisi esimerkiksi irrallisena Windows Service -palveluna suoritettavissa oleva välimuistiratkaisu, jonka rajapintaa kutsuttaisiin todellisen taustajärjestelmän sijasta. Tällöin kaikki kyseisen taustajärjestelmän pyynnöt kulkisivat välimuistiratkaisun kautta. Täten ennaltageneroivasta välimuistista tulisi web-sovelluksen ja taustajärjestelmän välille täysin irrallinen pyyntöjä välittävä ja lisäarvoa tuottava modulaarinen komponentti.

Edellä esitetyn lisäksi oleellisena jatkotutkimusaiheena on modulaarisen, web-sovelluksen kontekstista irrotetun ennaltageneroivan välimuistiratkaisun välimuistisisällön tehokkaan mitätöinnin tutkiminen: millä tavalla välimuistiolioiden invalidointi kannattaa rakentaa ja miten välimuistiratkaisu voisi oppia tunnistamaan välimuistisisältöön kohdistuvia muutoksia? Seuraavissa alakohdissa on pohdittuna alustavasti eri lähestymistapoja:

- Käytetään nykyistä tässä tutkimuksessa esitettyä tapaa, jossa sovellusohjelmoija huolehtii välimuistisisällön invalidoinnista viemällä muu-

tokset välimuistiin. Tämän lähestymistavan huonona puolena on se, että ratkaisu on arkkitehtuurillisesti kuormittava ja haastava toteuttaa. Lähestymistavan käyttöönoton yhteydessä rajoitteeksi tulee myös se, ettei taustajärjestelmätietoa saa päivittää välimuistiratkaisulle muutoksista viestimättä. Lisäksi tällä tavoin toteutettu ennakoiva välimuistiratkaisu on vaikea monistaa saati tuotteistaa, sillä toteutustapa integroituu tiiviisti kuhunkin yksilölliseen web-sovellukseen ja sen keskeisiin sovelluskohtaisiin toimintoihin. Monistettavuutta rajoittaa myös se, ettei tiiviisti integroitu toteutustapa ole alusta- tai teknologiariippumaton.

- Invalidointi hoidetaan siten, että taustajärjestelmä tuo muutokset modulaariseen välimuistiin. Tämä lähestymistapa on kuitenkin vaikea toteuttaa, sillä tyypillisesti taustajärjestelmät eivät toimi itse aktiivisina interaktion osapuolina. Taustajärjestelmät eivät myöskään tarjoa mekanismeja muuttuneen tiedon välittämiseksi välimuistiratkaisulle.
- Toteutetaan invalidointi siten, että taustajärjestelmästä noudetaan olioiden keskimääräisen eliniän pohjalta muutoksia välimuistiratkaisun toimesta. Tällöin riskinä on, että välimuistissa säilytettävä tieto on vanhentunutta. Suorituskykyhyödyt ovat kuitenkin tässä ratkaisussa korkeat, sillä taustajärjestelmää ei ole aina tarvetta kuormittaa, jos esimerkiksi on havaittavissa muutakin palvelussa tapahtuvaa kuormitusta. Tämä välimuistin invalidoinnin toimintamalli ei kuitenkaan tiedon eheyttä korostavien vaatimuksen näkökulmasta sovi kovinkaan moneen operatiiviseen järjestelmään.

Voitaisiinko tässä alaluvussa jo esiin tuodulla pohdinnalla vastata seuraaviin aiemman tutkimuksen esittämiin aihepiirin ongelmiin? Mehrotra ym. (2010) ovat listanneet dynaamisen ja yksilöidyn web-sisällön tuottamisesta aiheutuvia web-sovellusten suorituskykyyn negatiivisesti vaikuttavia ominaisuuksia. He toteavat, että mikäli taustalla olevan tiedon päivittäminen vaikuttaa useisiin dynaamisiin sivuihin, tulee kaikki tällaiset sivut päivittää vastaamaan uutta tietoa. Tämän lisäksi he mainitsevat dynaamisen sivun generoinnin vaativan tyypillisesti yhden tai useamman tietokantakyselyn suorittamista tietokantapalvelimella. Tästä johtuen tietokantaan kohdistuvien kyselyiden läpimenoajat voivat kasvaa hallitsemattomasti suurten yhdenaikaisten sivupyyntömäärien aikana. Tuomalla ennakoiva välimuisti lähelle taustajärjestelmää voidaan tuottaa apua kyseisen taustajärjestelmän suoriutumiseen, mikäli taustajärjestelmään kohdistuvista pyynnöistä saadaan osumatarkkuudeltaan riittävän hyviä. Liian yksilöidyiksi muodostuneet taustajärjestelmien kutsut eivät kuitenkaan ole ennaltageneroivan välimuistiratkaisun näkökulmasta edullisia, sillä niiden osumatarkkuudet jäävät liian alhaisiksi.

Tutkielman kirjallisuuskatsauksen noteeraamat aiempien tutkimusten esiintuomat kysymykset saavat osittain vastauksia tässä luvussa esitettyjen pohdintojen pohjalta. Esimerkiksi Mehrotra ym. (2010) listaavat välimuistiratkaisun toteutukseen liittyviä keskeisiä kysymyksiä:

- Missä välimuistin tulisi sijaita monikerrosarkkitehtuurissa?
- Minkä kokoisia välimuistiolioiden tulisi olla raekooltaan?

Ensimmäisen kysymyksen osalta voitaisiin pohtia, voisiko esitetty taustajärjestelmän ja web-sovelluksen välissä oleva modulaarinen ennakoiva välimuisti olla hyödyllinen jopa yksilöidyn web-sisällön tapauksessa?

Vastausta toisen kysymyksen välimuistiolion raekokoon liittyen voisi lähteä etsimään tässä luvussa esitetyn modulaarisen välimuistimallin pohjalta: tutkimalla pyynnön sisällön mukaan laskettavan tiivisteeseen pohjautuvan välimuistiavaimen merkitystä. Tässä ajatusmallissa välimuistiavain määrittää myös raekoon, joten taustajärjestelmäpyyntöjen määrittäessä välimuistiavaimet, siirtyy kiinnostus näiden pyyntöjen raekoon määrittelyyn. Raekoko tulisi määrittää sellaiseksi, että jopa yksilöityä sisältöä sisältävän web-sovelluksen taustajärjestelmäpyynnöt saisivat riittävästi osumia, jotta niiden pitäminen välimuistissa olisi kannattavaa.

8 YHTEENVETO

Tutkielmassa tutustuttiin web-sovellusten välimuistien toimintaan ja niistä saataviin hyötyihin. Tutkimuksessa havaittiin, että huolimattomasti suunnitellut välimuistitekniikat voivat olla web-sovelluksen kannalta jopa haitallisia. Välimuistitekniikoiden aihepiirissä syvennyttiin välimuistin käytön ja ennaltageneroinnin yhteishyötyihin. Tarkastelussa oli ennakoivan välimuistin ennaltagenerointiin liittyvä aihealue sekä kustannusfunktioihin pohjautuva ennaltagenerointi. Tutkielmassa esiteltiin olemassa olevia eri hyötyjä tavoittelevia kustannusfunktioita sekä uusi työmäärää ja osumatarkkuutta tavoitteleva ahne kustannusfunktio.

Tutkimus toteutettiin kirjallisuuskatsaukseen pohjautuvana suunnittelutieteellisenä konstruktiivisena tutkimuksena. Tutkimuksessa suunniteltiin ennakoivan välimuistiratkaisun malli ja suoritettiin kustannusfunktioihin pohjautuen simulointeja, joiden avulla tarkasteltiin eri kustannusfunktioiden hyötyjä ja haittoja. Tarkastelun kohteena ollut tutkimusongelma jakautui kysymyksiin:

- Mitä vaatimuksia web-sovelluspalvelimella toimivan ennakoivan välimuistin toteuttamiselle on?
- Miten uusi tutkielmassa esitelty kustannusfunktio suoriutuu ja asettuu jo aiemmin tunnettuihin kustannusfunktioihin nähden?
- Millä kustannusfunktiolla viivettä laskeva ja tasoittava ennakoiva välimuistiratkaisu kannattaa toteuttaa yksilöityä sisältöä sisältävässä web-sovelluksessa?

Ensimmäisen osatutkimusongelman vastaukseksi syntyi kirjallisuuskatsauksen pohjalta luvussa 3 esitelty ennakoivan välimuistiratkaisun malli ja vaatimuslistaus eli joukko ennakoivalle välimuistiratkaisulle keskeisiä vaatimuksia. Listatut vaatimukset nähtiin hyvänä pohjana reaalimaailman ratkaisun rakentamiseksi. Ennakoivan välimuistiratkaisun kritiikiksi nostettiin suorituskyky ja resurssienkulutus, jotka molemmat on otettava huomioon ennakoivaa välimuistiratkaisua toteutettaessa. Haasteena ja suurena riskinä nähtiin oheislaskennasta aiheutuva kasvava sisällön generoinnin viive, jonka esitettiin johtuvan web-

sovelluspalvelimen laskentatehon hyödyntämisestä ennakoivan välimuistin ylläpitämiseen käyttäjien pyyntöjen palvelemisen sijasta. Esitetyn ennakoivan välimuistiratkaisun mallin haasteeksi havaittiin myös web-sovellukseen tiiviisti integroidun ratkaisun negatiiviset vaikutukset web-sovelluksen testattavuuteen sekä muuntautumiskykyyn. Täten esitetyn mallin nähtiin heikentävän web-sovelluksen piirteitä sovellusarkkitehtuurin laadullisista näkökulmista. Mallin arkkitehtuuriseksi rajoitukseksi havaittiin myös, ettei taustajärjestelmätietojen päivityksiä sallittaisi mallin käyttöönoton jälkeen muiden, kuin välimuistiratkaisun toimesta, jotteivat välimuistitiedot joutuisi epäeheään tilaan.

Jatkotutkimuksen kannalta tutkielmassa esitetyn ennakoivan välimuistiratkaisun mallin hyvyyttä tulisi kyseenalaistaa esimerkiksi esittelemällä jollain tapaa hyödyllisempi tai vähemmän haasteellisempi malli. Jatkotutkimuksessa tulisi pohtia saavutettaisiinko parempi ennakoivan välimuistiratkaisun malli, mikäli ennakoiva välimuistiratkaisu toteutettaisiinkin web-sovelluksesta erillisenä modulaarisena komponenttina, joka ei ottaisi kantaa web-sovelluksen toteutukseen, vaan toimisikin irrallisena kokonaisuutena. Tällainen web-sovelluksen ja taustajärjestelmän välissä sijaitseva ennakoiva välimuisti toimisi vastaanottaen syötteenä pelkästään web-sovelluksen sekä taustajärjestelmän suunnista saapuvia viestejä sekä ohjaustietoja ja mukautuisi automaattisesti komponentin itsensä läpi kulkevaan liikenteeseen. Edellä esitetyn ennakoivan välimuistin kannattavuus pohjautuu oletukseen, että ennaltageneroinnista saatavat hyödyt kasvavat lähempänä taustajärjestelmiä tarkasteltaessa dynaamista sekä yksilöityä sisältöä. Yksilöity web-sisältö vaatii lähes aina jonkin verran taustajärjestelmätietoa sekä istuntotiedon, josta johtuen sisällön generointi tapahtuu web-sovelluspalvelimella. Tähän tarpeeseen on etsittävä edelleen tapoja, joilla saadaan tehostettua sisällön generointia alkuperäisillä web-sovelluspalvelimilla. Jatkotutkimuksen kannalta mielenkiintoinen hypoteesi olisikin: ”Ennaltageneroinnin hyödyt kasvavat web-sovelluksen ja taustajärjestelmän välillä, mikäli taustajärjestelmäpyyntöjä säädetään yleiskäyttöisemmiksi siten, että aikaansaadaan suurempia välimuistiolioiden osumatarkkuuksia, jolloin välimuistiin tallennetaan raekooltaan optimaalisia välimuistiolioita”.

Tutkielman luvussa 3 esitetty ennakoivan välimuistiratkaisun malli kärsii lisäksi siitä, että suurin osa välimuistiolioiden osumatarkkuuksista on hyvin alhaisia. Suurimmat vaikuttavat tekijät ovat dynaaminen sekä yksilöity web-sisältö. Jotta voitaisiin vastata paremmin dynaamisen sekä yksilöidyn sisällön aiheuttamiin haasteisiin ennaltageneroinnin ja välimuistin yhdistelmällä, tulisi ennaltagenerointia ja välimuisteja viedä kenties lähemmäs tiedon alkuperää esimerkiksi web-sovelluksen ja taustajärjestelmien välille, jossa esimerkiksi tietokantakyselyt voitaisiin helpommin irrottaa käyttäjäsidonnaisuudesta. Tulisikin tutkia, voitaisiinko lähempänä taustajärjestelmää sijaitsevalla välimuistilla aikaansaada suurempia välimuistiolioiden osumatarkkuuksia tallentamalla välimuistiin raekooltaan optimaalisia välimuistiolioita. Optimaalinen välimuistiolio olisi riittävän yleinen, jotta osumatarkkuus olisi riittävän korkea mutta kuitenkin riittävän suppea, jottei siihen kohdistuisi liikaa invalidointeja taustajärjestelmässä tapahtuvien tietojen muutosten johdosta. Jatkotutkimuksen kan-

nalta oleellista olisikin selvittää ennakoivan välimuistiratkaisun hyödyllisin sijaintipaikka web-sovelluksen arkkitehtuurissa. Edellä esitetyn lisäksi, näen oleellisena jatkotutkimusaiheena modulaarisen web-sovelluksen kontekstista irrotetun ennaltageneroivan välimuistiratkaisun välimuistisisällön tehokkaan mitätöinnin tutkimisen: millä tavalla välimuistiolioiden invalidointi tulisi rakentaa ja miten välimuistiratkaisu voisi oppia tunnistamaan taustajärjestelmässä tapahtuvia välimuistisisältöön kohdistuvia muutoksia?

Tutkimuksen toisen osatutkimusongelman osalta päädyttiin simulaatiotuloksissa lopputulokseen, ettei ehdotettu $OTTM$-ahne kustannusfunktio kykene ennakko-oletuksista huolimatta vähentämään eikä tasoittamaan käyttäjän kokemaa viivettä tehokkaammin, kuin kirjallisuuden jo aiemmin tuntemat kustannusfunktiot. Lisäksi simuloinneissa pyrittiin keskeisiä muuttujia varioimalla löytämään web-sovelluksen tyyppiominaisuuksia, joissa $OTTM$-ahne kustannusfunktio olisi ollut muita kustannusfunktioita hyödyllisempi. Simulaatiotulokset osoittivat kuitenkin, ettei esitetty kustannusfunktio muuttujien varioinneista huolimatta suoriutunut muita vertailtavia kustannusfunktioita paremmin. Suurimmaksi taustalla olevaksi tekijäksi osoittautui zipfiläiseen jakaumaan pohjautuva olion i todennäköisyys p_i tulla pyydetyksi. Laskennaltaan muuttujaan p_i pohjautuvien kustannusfunktioiden havaittiin olevan ylivoimaisia suuren osumatarkkuuden aikaansaamisen suhteen. $OTTM$-ahneen kustannusfunktion simulointituloksissa keskimääräisen L_g-viiveen havaittiin lähenevän nollaa kaikkien mahdollisten olioiden kesken tasaisesti. Tämän tasaisen viiveiden pienemisen ei kuitenkaan todettu vastaavan reaalimaailman tilannetta, jossa web-sovelluksen kuormituksen tiedettiin tapahtuvan zipfiläiseen jakaumaan pohjautuvalla pyyntöjoukolla. Täten $OTTM$-ahneen ennaltageneroinnin tunnistettiin tuottavan välimuistiin sisältöä, jota ei todellisuudessa koskaan välimuistista käytetty.

Kirjallisuuden ennalta tuntemia kustannusfunktioita heikomman suoriutumisen lisäksi $OTTM$-ahne kustannusfunktio sai osakseen runsaasti kritiikkiä ylivoimaisesti suurimman kaistankäyttönsä suhteen. Simulointituloksissa nähtiin myös, että esitetyn kustannusfunktion haitat olivat jo pienillä ennaltageneroitavilla oliomäärillä huomattavat. Tutkimustulosten johtopäätöksissä todettiinkin, että esitetty kustannusfunktio voidaan nähdä reaalimaailman websovelluksen kannalta ennakoivassa välimuistiratkaisussa haastavana, ellei lähes mahdottomana ennaltageneroitavien välimuistiolioiden valintaperusteena.

Kolmanteen osatutkimusongelmaan vastaus löytyi aiemman aihepiirin kirjallisuuden pohjalta, sillä esitetty $OTTM$-ahne kustannusfunktio ei kyennyt haastamaan jo tunnettuja kustannusfunktioita. Tämän tutkimuksen pohjalta ennaltageneroivan välimuistiratkaisun kustannusfunktioksi nähdään parhaaksi valinnaksi OT/KK-ahne kustannusfunktio, jossa OT-ahneen kustannusfunktion osumatarkkutta tavoittelevat hyödyt saadaan käyttöön kuitenkin siten, että liiallista kaistankulutusta voidaan kontrolloida. Tutkimuksen simulaatiotulokset osoittivat kuitenkin, että ennaltageneroivasta välimuistiratkaisusta saatavat hyödyt laskevat, mitä enemmän välimuistin käytön pyyntötiheys nousee, sillä välimuistista saatava passiivinen hyöty kasvaa.

LÄHTEET

Acharjee, U. (2006). *Personalized and Artificial Intelligence Web Caching and Prefetching*. Master's thesis. University of Ottawa.

Ali, W., Shamsuddin, S. M. & Ismail, A., S. (2011). A Survey of Web Caching and Prefetching. *International Journal of Advances in Soft Computing and Its Applications, 3(1),* 18-44.

Ayani, R., Teo, Y., M. & Ng, Y., S. (2003). Cache pollution in Web proxy servers. Teoksessa *17th International Parallel and Distributed Processing Symposium IPDPS'03 Nice, France, April 22-26* (s. 248). DC, Washington: IEEE Computer Society.

Bestavros, A., Cunha, C., R. & Crovella, M., E. (1995). *Characteristics of WWW Client-based Traces* (Report BU-CS-95-010), Boston University, Department of Computer Science.

Chen, H., T. (2008). *Pre-fetching and Re-fetching in Web caching systems: Algorithms and Simulation*. Master's thesis. Trent University.

Chiang, R., Goes, P., B. & Zhang, Z. (2006). Periodic cache replacement policy for dynamic content at application server. *Journal of Decision Support Systems, 43(2)*, 336-348.

Cobb, J. & ElAarag, H. (2008). Web proxy cache replacement scheme based on back-propagation neural network. *Journal of System and Software, 81(9),* 1539-1558.

Dutta, K., Thomas, H., Datta, A. & Keskinocak, P. (2007). Linearized model of object caching and heuristic solution. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 37(5)*, 815-826.

Jiang, Y., Wu, M.-Y. & Shu, W. (2002). Web Prefetching: Costs, Benefits and Performance. Teoksessa *7th International Workshop on Web Content Caching and Distribution (WCW) Boulder, Colorado, August 12-14.*

Koskela, T., Heikkonen, J. & Kaski, K. (2003). Web cache optimization with nonlinear model using object feature. *Elsevier Computer Networks Journal, 43(6)*, 805-817.

Kroeger, T. M., Long, D. D. E. & Mogul, J. C. (1997). Exploring the Bounds of Web Latency Reduction from Caching and Prefetching. Teoksessa *Proceedings of the USENIX Symposium on Internet Technologies and Systems Montrey, California, December* (s. 13-22). Berkley, CA: USENIX Association Berkeley.

Lu, J. & Gokhale, S., S. (2008). *Hierarchical Performance and Availability Analysis Methodology for Multi-tiered Web Applications*. A Dissertation. Connecticut University.

Markatos, E., P. & Chronaki, C., E. (1998). A Top-10 approach to prefetching on the Web, Teoksessa *Proceedings of INET'98 Geneva, Switzerland, July* (s. 276-290).

Mehrotra, D., Nagpal, R., & Bhatia, P. (2010). Designing metrics for caching techniques for dynamic web site. Teoksessa *2010 International Conference on Computer and Communication Technology (ICCCT) Allahabad, Uttar Pradesh, India, September 17-19* (s. 589-595). Allahabad: Motilal Nehru National Institute of Technology.

Nanopoulos, A., Katsaros, D., & Manolopoulos, Y. (2003). A data mining algorithm for generalized Web prefetching, *IEEE Trans on Knowledge and Data Engineering, 15(5),* 1152-1169.

Nigam, B. & Jain, S. (2010). Analysis of Markov model on different web Prefetching and caching schemes. Teoksessa N. Krishnan (toim.), *2010 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC) Indore, India, December 28-29* (s. 1-6). Coimbatore: Tamilnadu College of Engineering.

Nishikawa, N., Hosokawa, T., Mori, Y., Yoshidab, K. & Tsujia, H. (1998). Memory Based Architecture with Distributed WWW Caching Proxy, *Computer Networks*, *30(1-7)*, 205-214.

Padmanabhan, V., N. & Mogul, J., C. (1996). Using predictive prefetching to improve World Wide Web latency, *Computer Communication Review, 26(3)*, 22–36.

Pallis, G., Vakali, A. & Pokorny, J. (2008). A clustering-based prefetching scheme on a Web cache environment. *Computers and Electrical Engineering, 34(4)*, 309-323.

Probir, G. & Rau-Chaplin, A. (2006). Performance of Dynamic Web Page Generation for Database-driven Web Sites. Teoksessa A. Ajith, Y. H. Sang (toim.)*, Proceedings of the International Conference on Next-Generation Web Services Practices Seoul, Korea, September 25-28* (s. 56-63). Washington, DC: IEEE Computer Society.

Ravi, J., Yu, Z. & Shi, W. (2009). A Survey on Dynamic Web Content Generation and Delivery Techniques, *Elsevier Journal of Network and Computer Applications, 32(5),* 943-960.

Scott, M. L. (2009). *Programming language pragmatics* (3. painos). Burlington (MA): Morgan Kaufmann.

Shi, L., Song. B., Ding, X., Gu, Z. & Wei, L. (2005). Web Prefetching Control Model Based on Prefetch-Cache Interaction. Teoksessa S. Kawada (.toim), *International Conference on Semantics, Knowledge and Grid (SKG 2005) Beijing, China, November 27-29* (s. 30-35). Washington, DC: IEEE Computer Society.

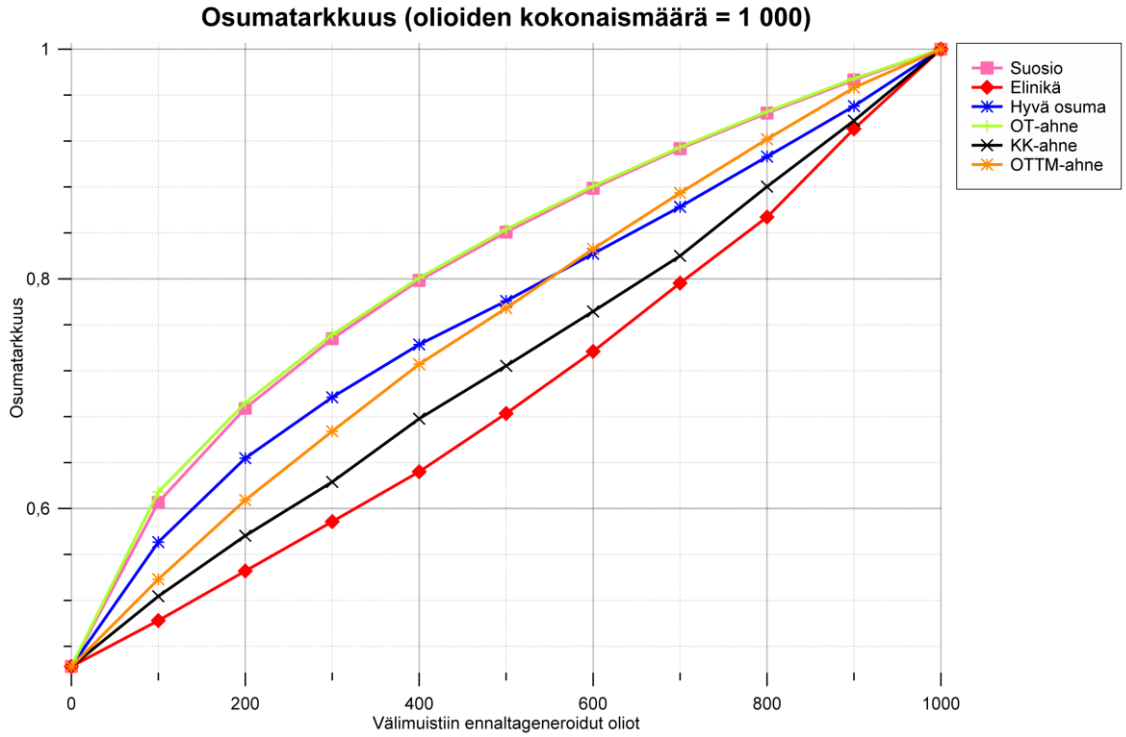
Wong, A., K., Y. (2006). Cache Replacement Policies: A Pragmatic Approach. *IEEE Network magazine, 20(1),* 28–34.

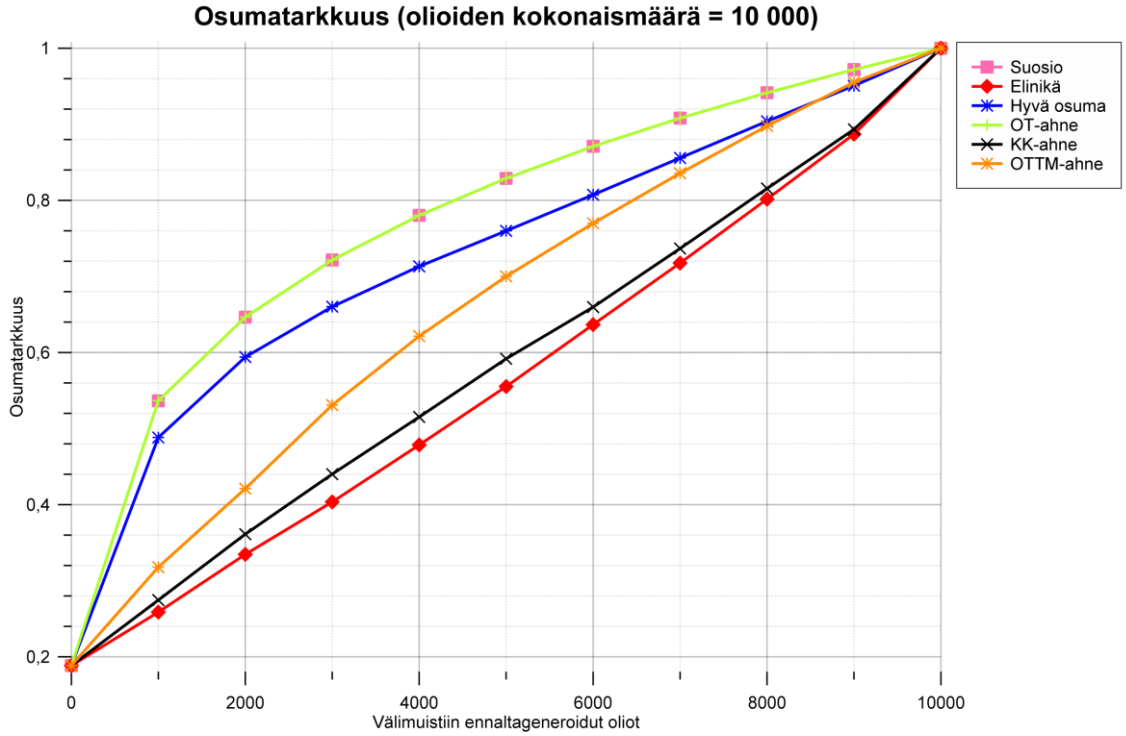
Wu, B. & Kshemkalyani, A. D. (2004). Objective-greedy algorithms for long-term Web prefetching. Teoksessa *Proceedings of the IEEE Conference on Network Computing and Applications (NCA)* (s. 61-68). Piscataway: IEEE Computer Society.

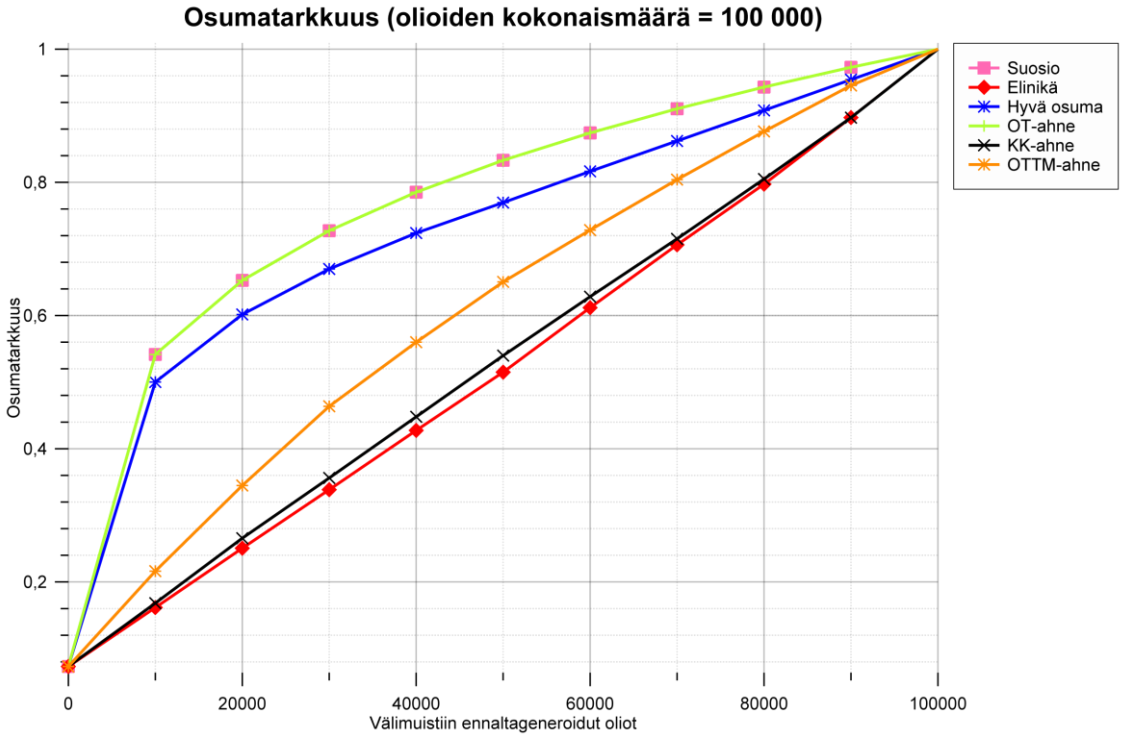
Wu, B. & Kshemkalyani, A. D. (2006). Objective-Optimal Algorithms for Long-Term Web Prefetching. *IEEE Transactions on Computers, 55(1)*, 2-17.

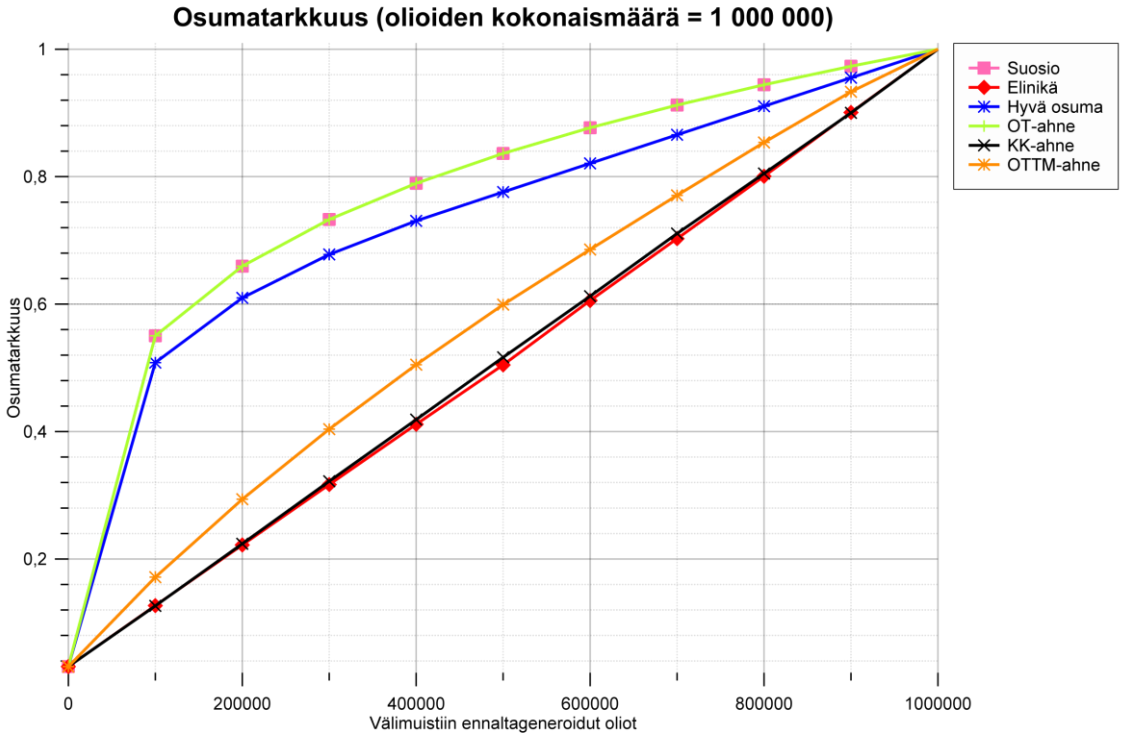
Zhijie, B., Zhimin, G. & Yu, J. (2009). A Survey of Web Prefetching. *Journal of computer research and development, 46(2)*, 202-210.

LIITE 1 OSUMATARKKUUS

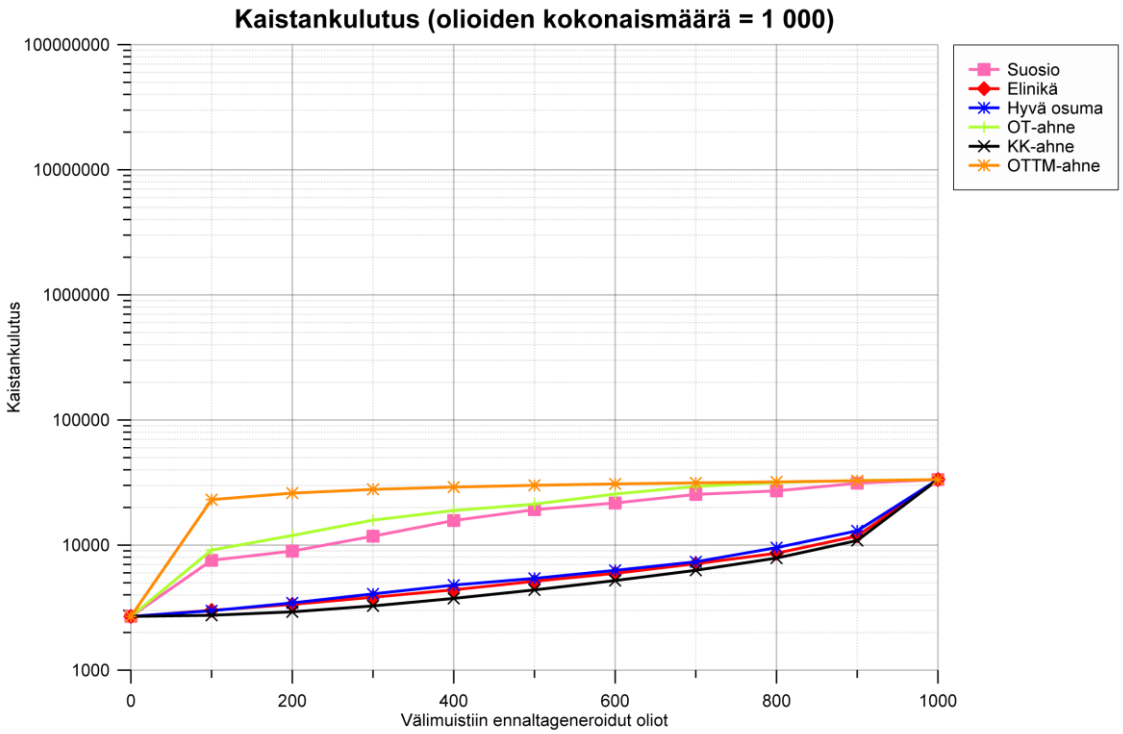


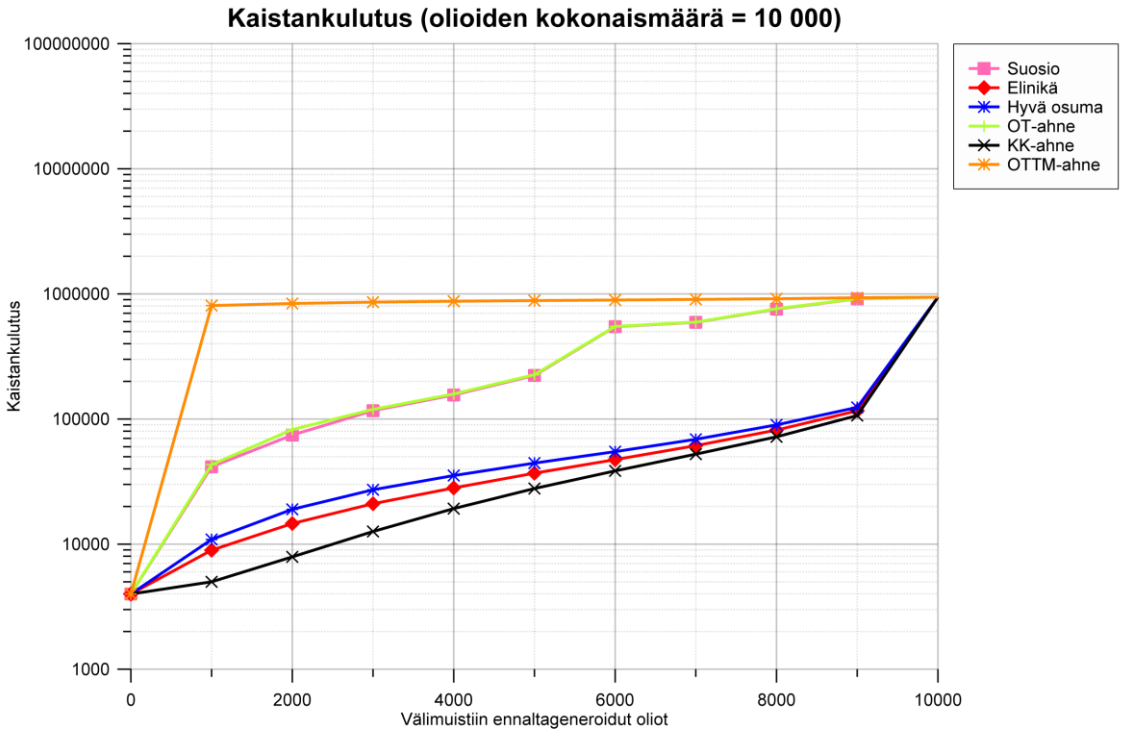


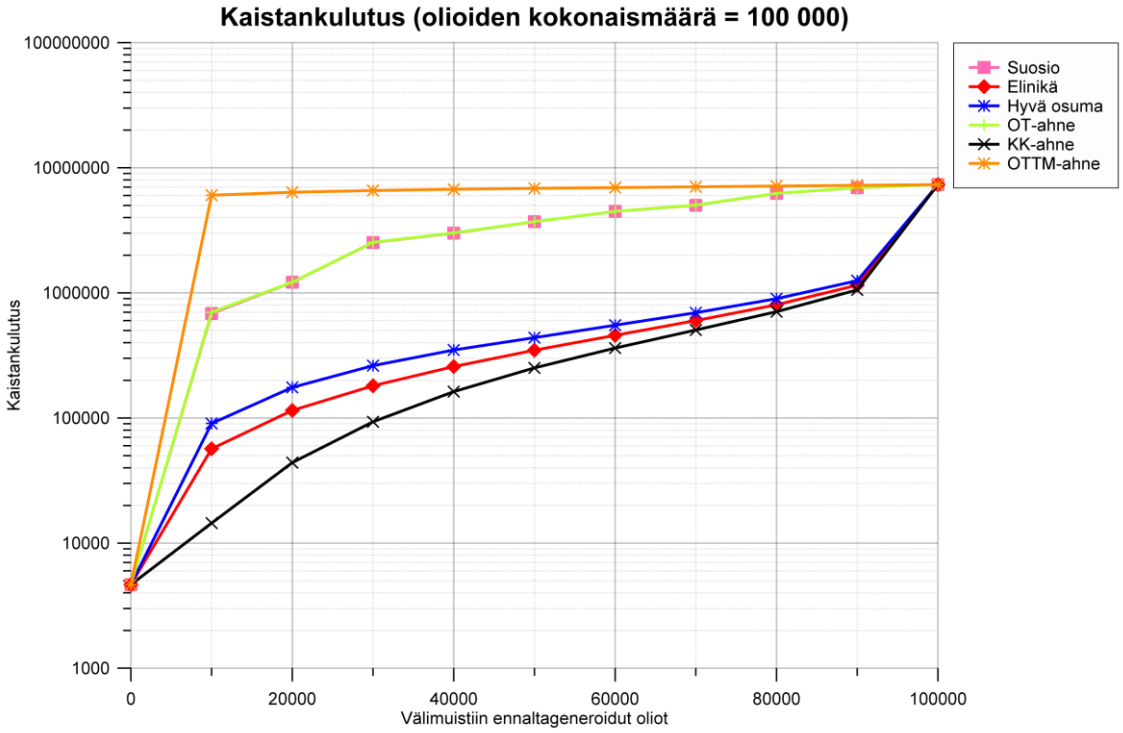

Osumatarkkuus (olioiden kokonaismäärä = 100 000)
Osumatarkkuus
1
0,8
0,6
0,4
0,2
0
20000
40000
60000
80000
100000
Välimuistiin ennaltageneroidut oliot
Suosio
Elinikä
Hyvä osuma
OT-ahne
KK-ahne
OTTM-ahne

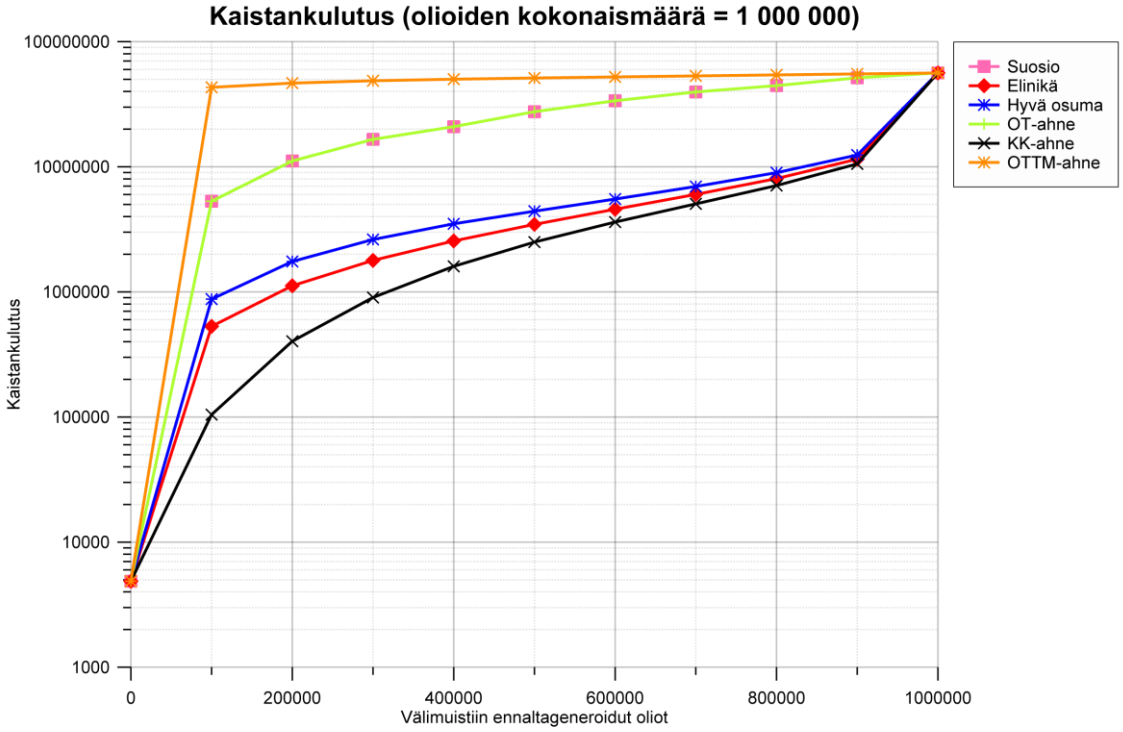

Osumatarkkuus (olioiden kokonaismäärä = 1 000 000)
Osumatarkkuus
1
0,8
0,6
0,4
0,2
0
200000
400000
600000
800000
1000000
Välimuistiin ennaltageneroidut oliot
Suosio
Elinikä
Hyvä osuma
OT-ahne
KK-ahne
OTTM-ahne

LIITE 2 KAISTANKULUTUS

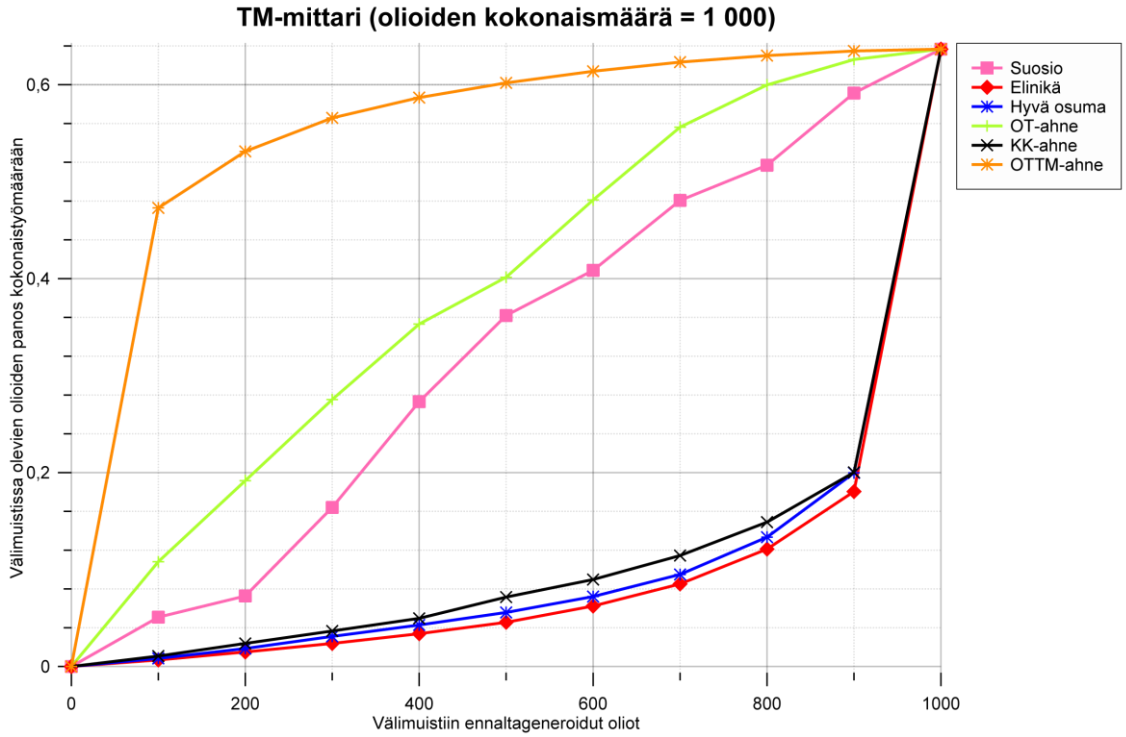


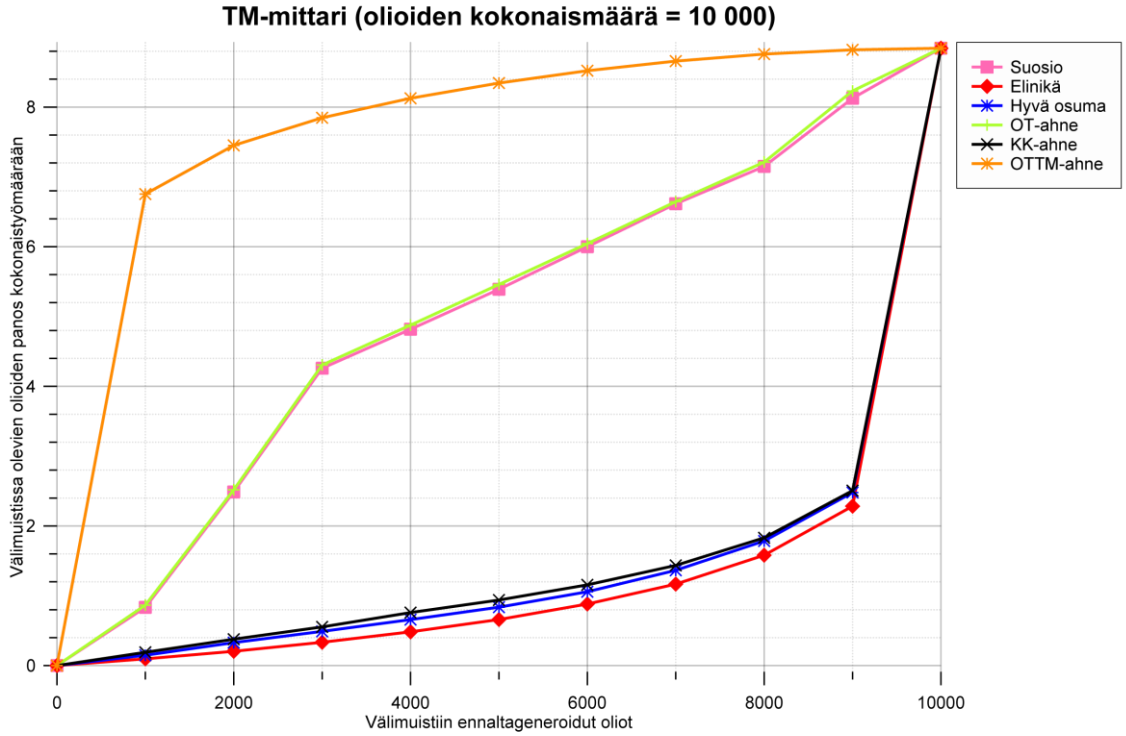


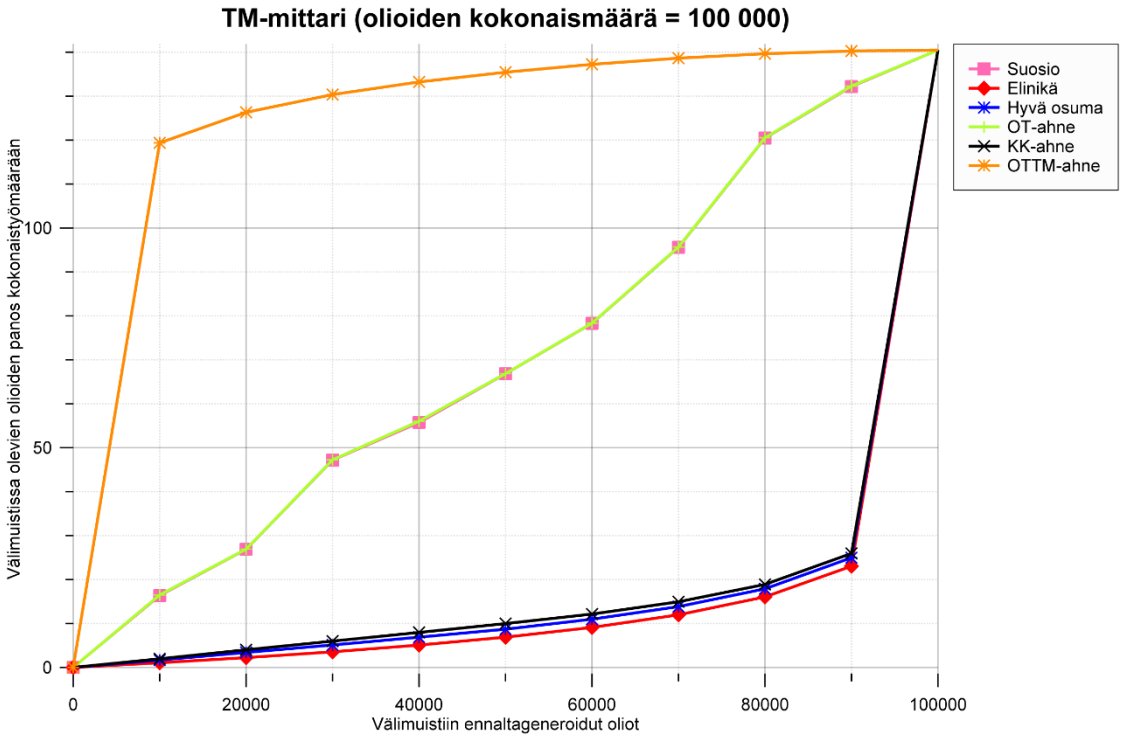

Kaistankulutus (olioiden kokonaismäärä = 100 000)
Kaistankulutus
100000000
10000000
1000000
100000
10000
1000
0
20000
40000
60000
80000
100000
Välimuistiin ennaltageneroidut oliot
Suosio
Elinikä
Hyvä osuma
OT-ahne
KK-ahne
OTTM-ahne

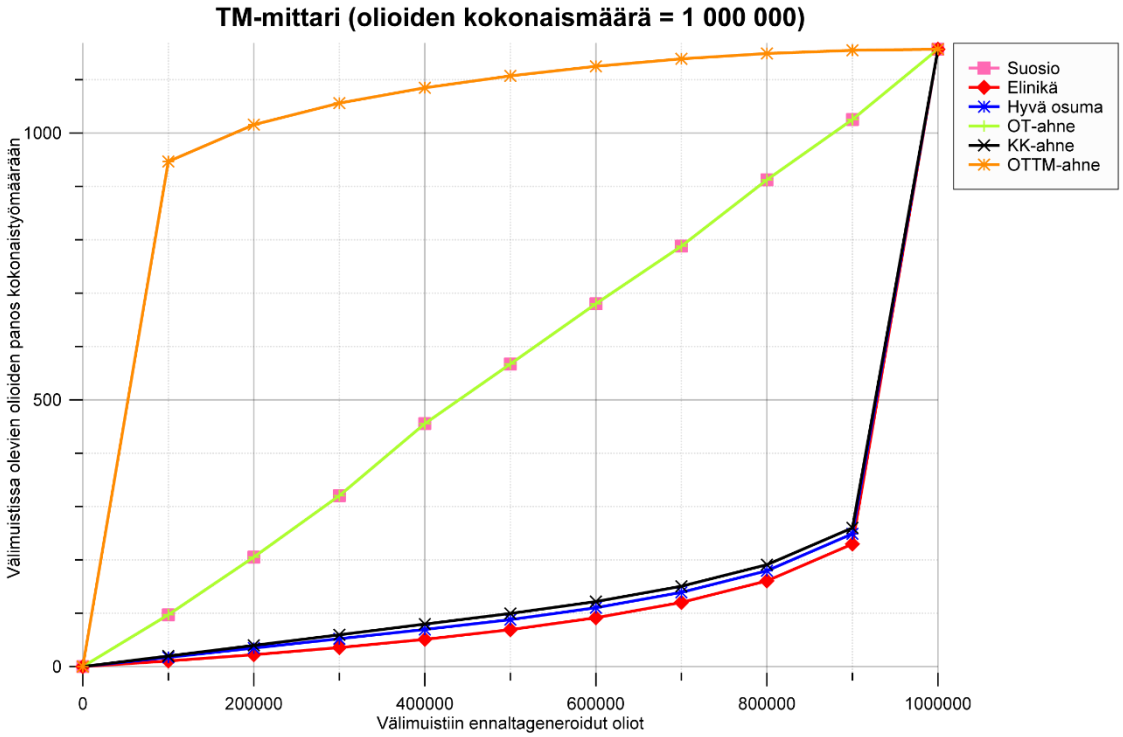

Kaistankulutus (olioiden kokonaismäärä = 1 000 000)
Kaistankulutus
100000000
10000000
1000000
100000
10000
1000
0
200000
400000
600000
800000
1000000
Välimuistiin ennaltageneroidut oliot
Suosio
Elinikä
Hyvä osuma
OT-ahne
KK-ahne
OTTM-ahne

LIITE 3 TM-MITTARI

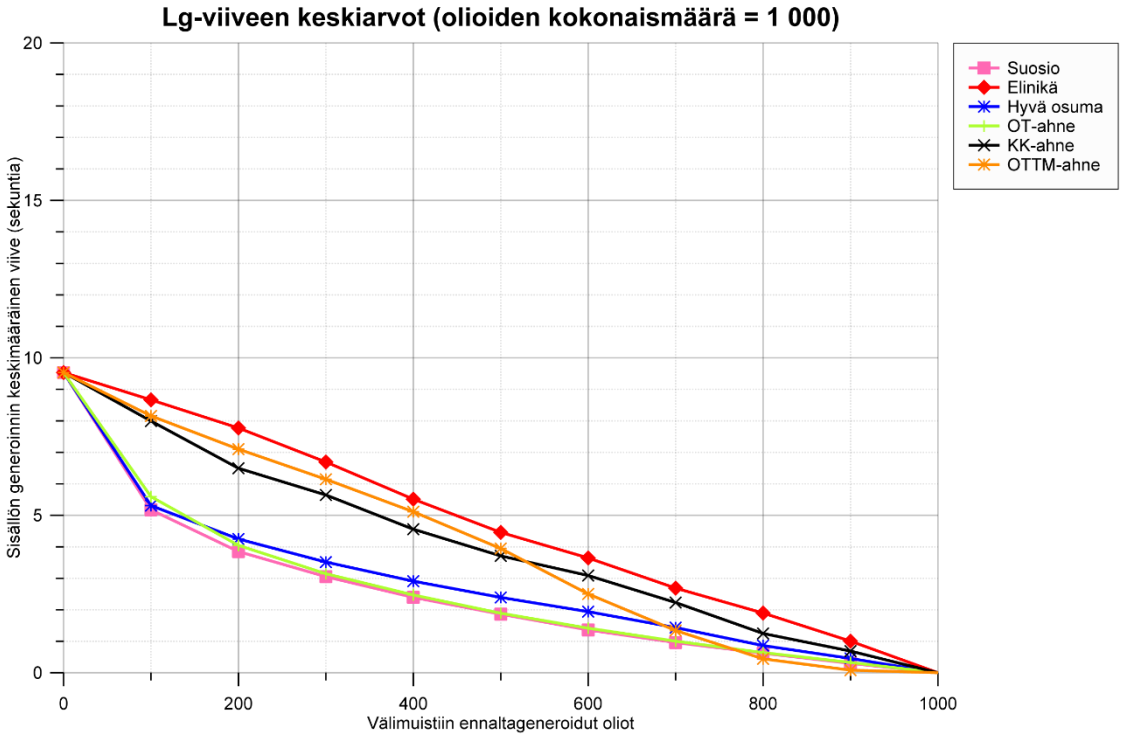


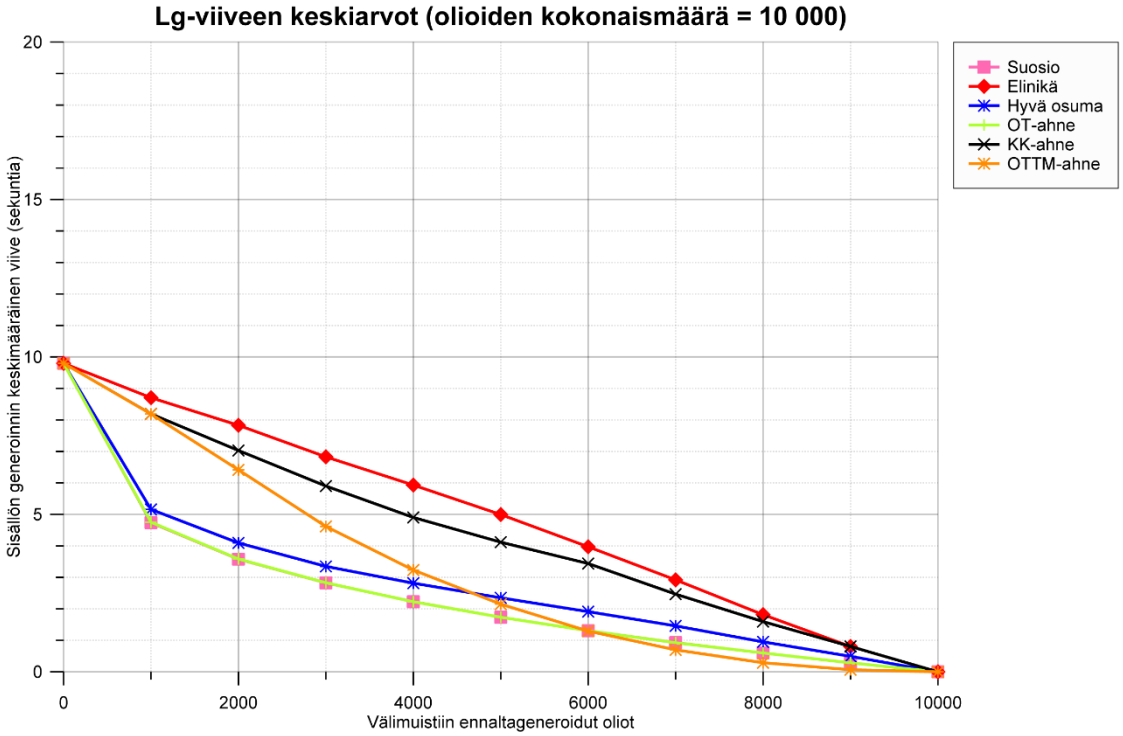


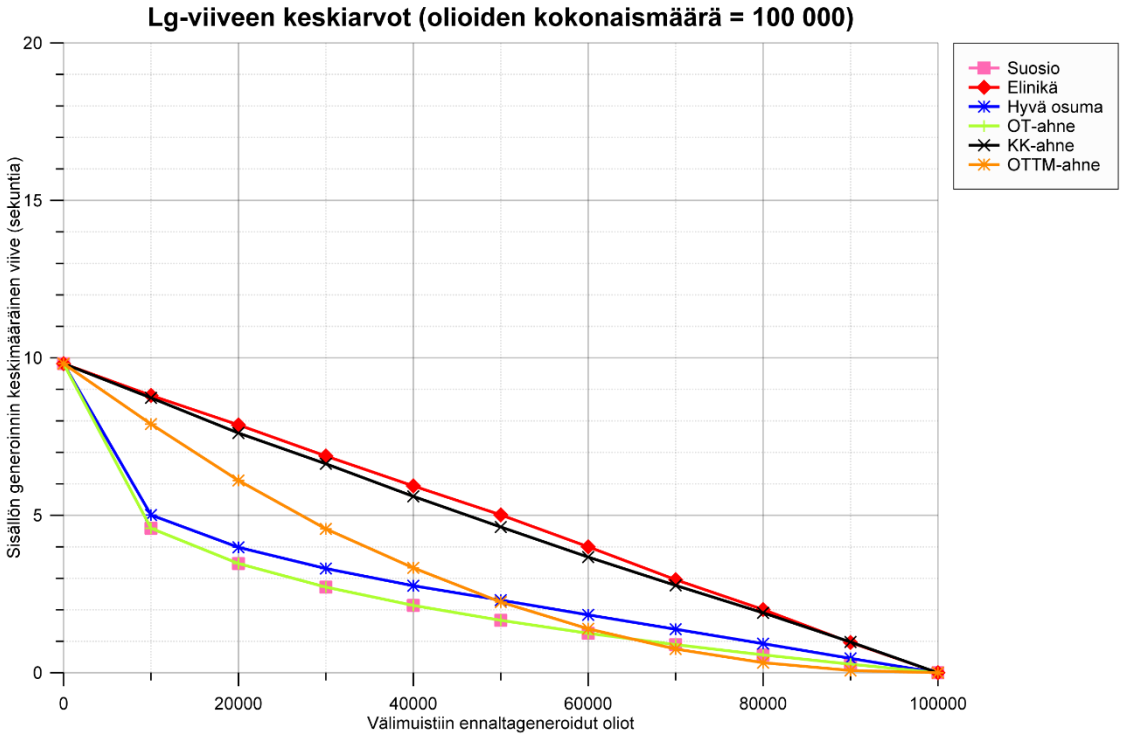

TM-mittari (olioiden kokonaismäärä = 100 000)
Välimuistissa olevien olioiden panos kokonaistyömäärään
0
50
100
0
20000
40000
60000
80000
100000
Välimuistiin ennaltageneroidut oliot
Suosio
Elinikä
Hyvä osuma
OT-ahne
KK-ahne
OTTM-ahne

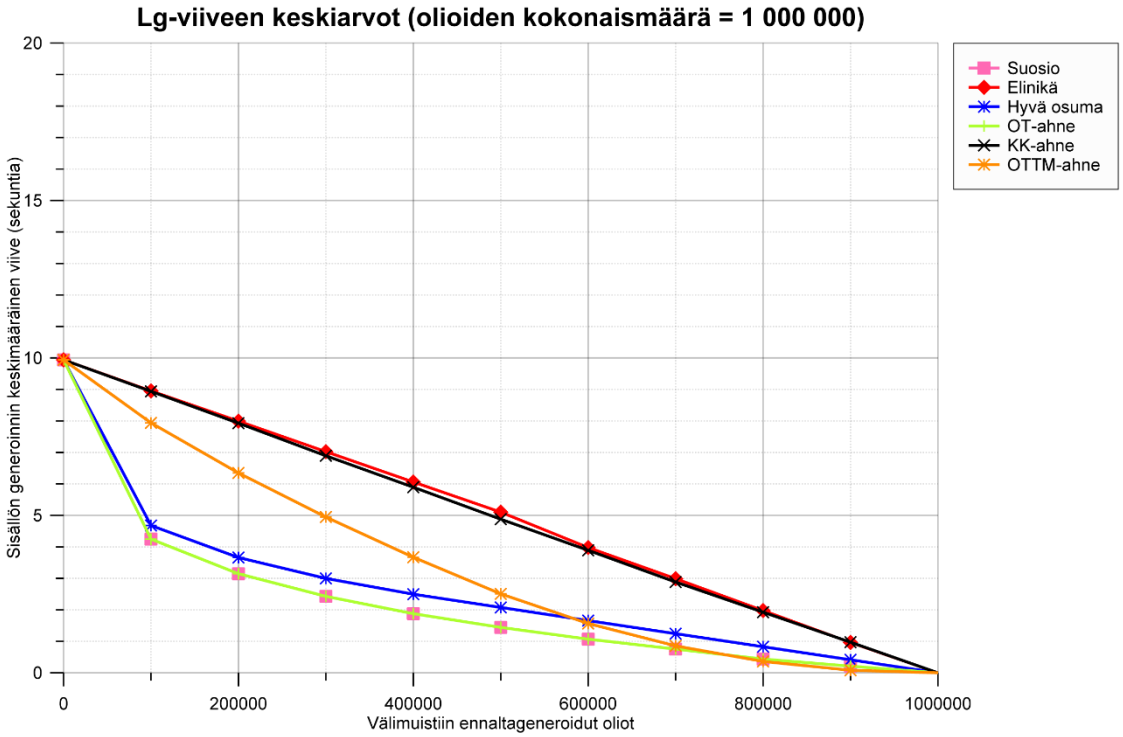

TM-mittari (olioiden kokonaismäärä = 1 000 000)
Välimuistissa olevien olioiden panos kokonaistyömäärään
0
500
1000
0
200000
400000
600000
800000
1000000
Välimuistiin ennaltageneroidut oliot
Suosio
Elinikä
Hyvä osuma
OT-ahne
KK-ahne
OTTM-ahne

LIITE 4 LG-VIIVEEN KESKIARVOT

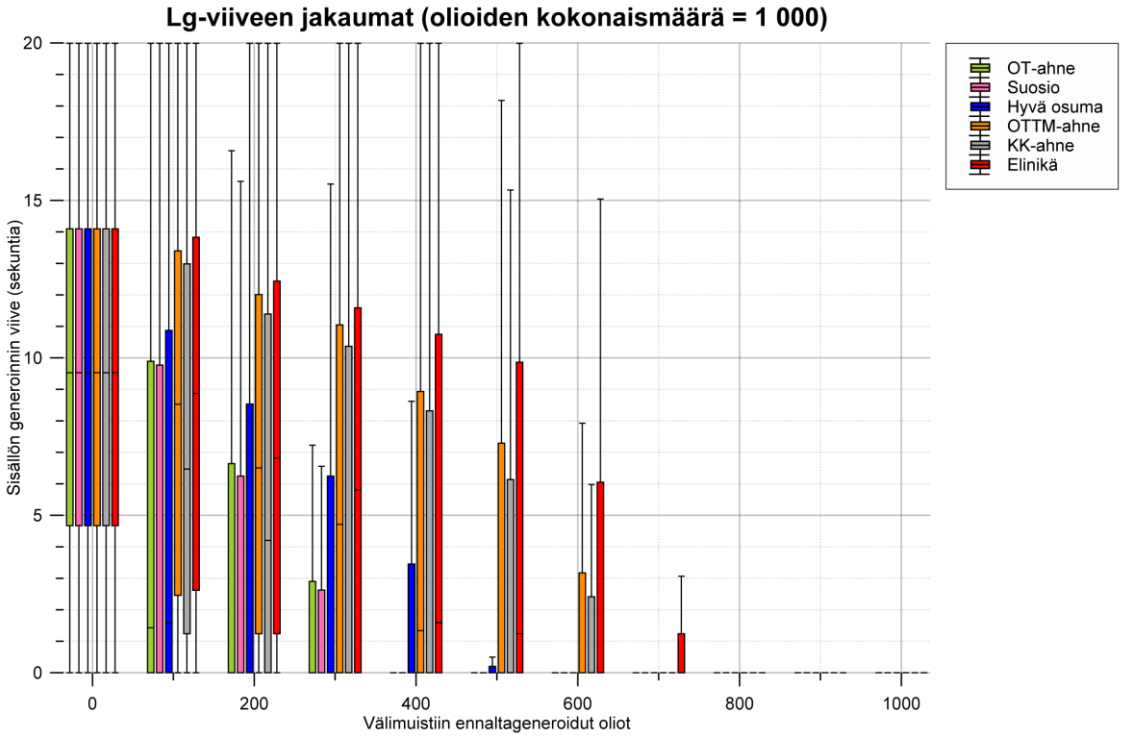


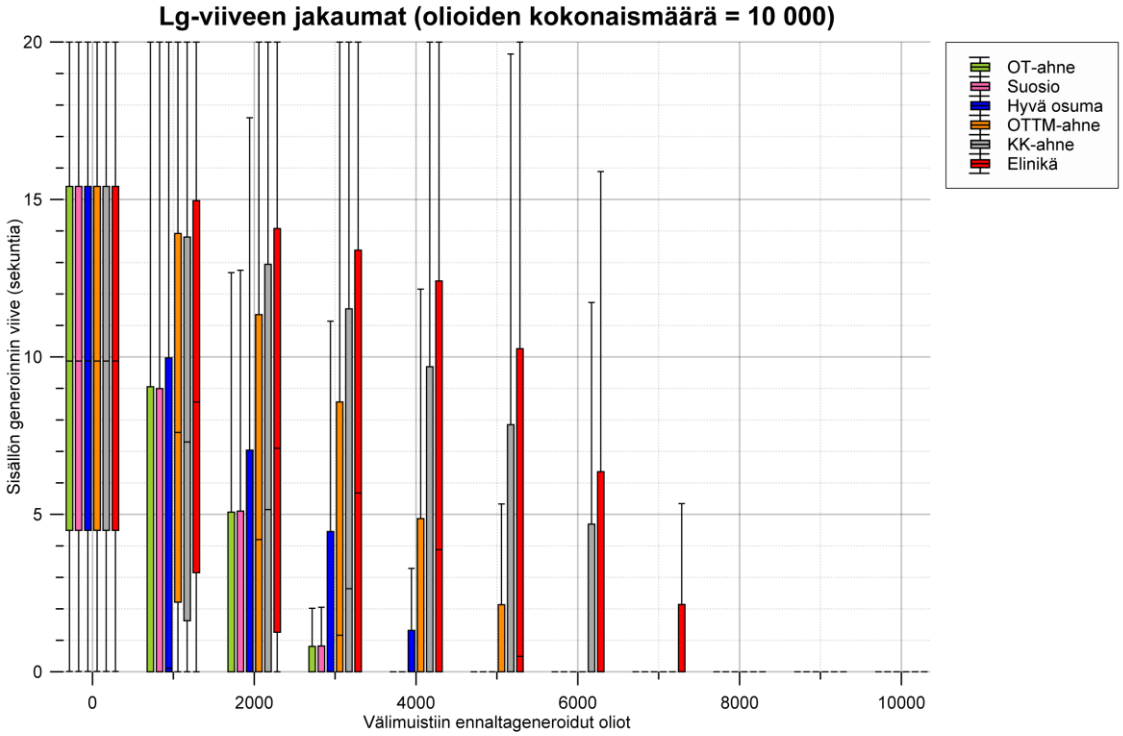


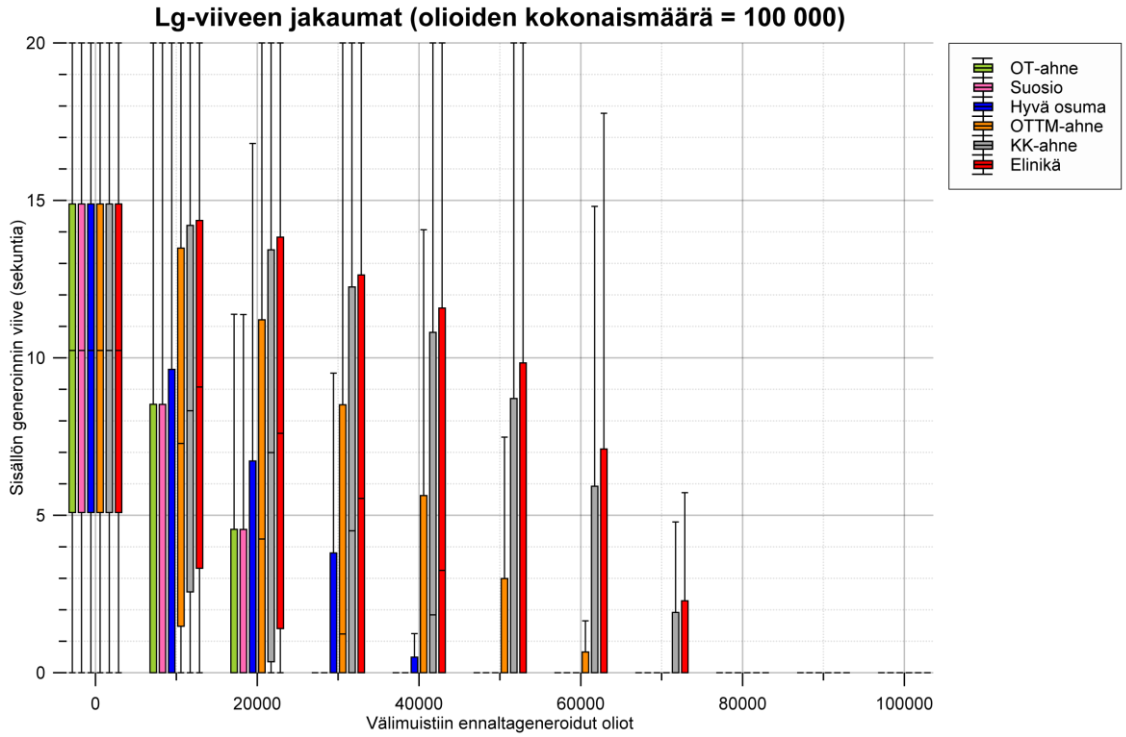

Lg-viiveen keskiarvot (olioiden kokonaismäärä = 100 000)
Sisällön generoinnin keskimääräinen viive (sekuntia)
Välimuistiin ennaltageneroidut oliot
0
5
10
15
20
0
20000
40000
60000
80000
100000
Suosio
Elinikä
Hyvä osuma
OT-ahne
KK-ahne
OTTM-ahne

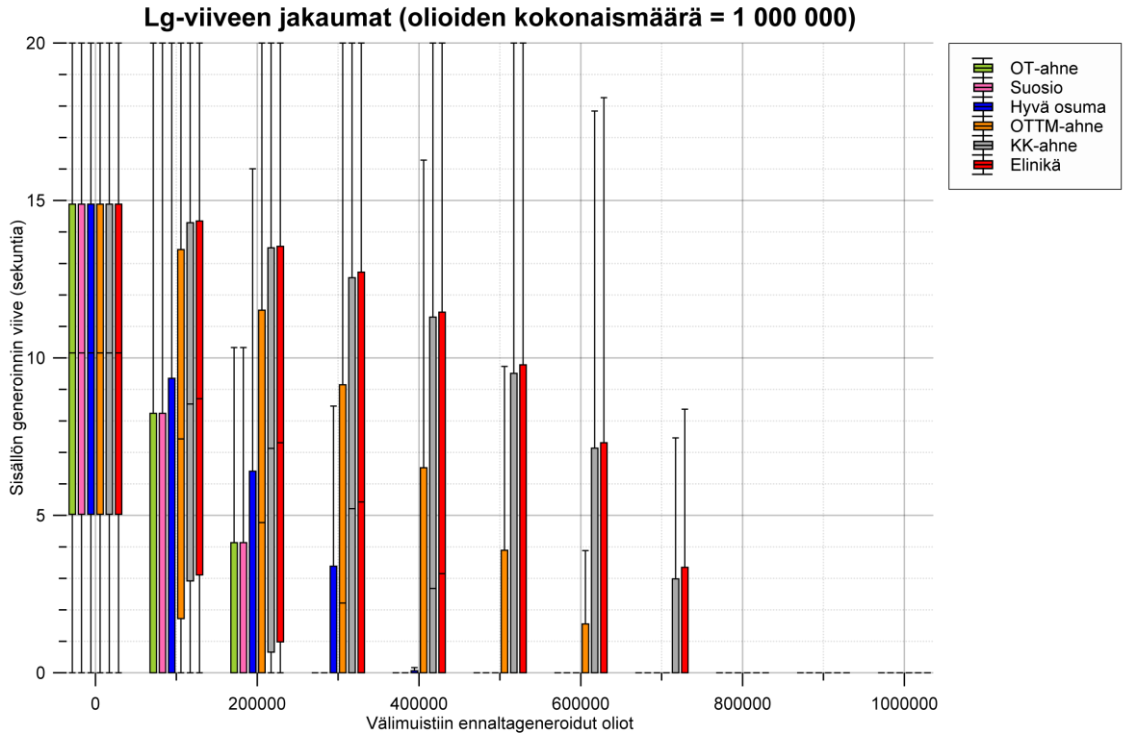

Lg-viiveen keskiarvot (olioiden kokonaismäärä = 1 000 000)
Sisällön generoinnin keskimääräinen viive (sekuntia)
Välimuistiin ennaltageneroidut oliot
0
5
10
15
20
0
200000
400000
600000
800000
1000000
Suosio
Elinikä
Hyvä osuma
OT-ahne
KK-ahne
OTTM-ahne

LIITE 5 LG-VIIVEEN JAKAUMAT






Lg-viiveen jakaumat (olioiden kokonaismäärä = 100 000)
Sisällön generoinnin viive (sekuntia)
0
5
10
15
20
0
20000
40000
60000
80000
100000
Välimuistiin ennaltageneroidut oliot
OT-ahne
Suosio
Hyvä osuma
OTTM-ahne
KK-ahne
Elinikä


Lg-viiveen jakaumat (olioiden kokonaismäärä = 1 000 000)
Sisällön generoinnin viive (sekuntia)
0
5
10
15
20
0
200000
400000
600000
800000
1000000
Välimuistiin ennaltageneroidut oliot
OT-ahne
Suosio
Hyvä osuma
OTTM-ahne
KK-ahne
Elinikä