TEKNILLINEN KORKEAKOULU
Tietotekniikan osasto
Tietojenkäsittelyopin laboratorio

Jussi Koskela, jpkoske2@cc.hut.fi
Henri Sivonen, hsivonen@iki.fi

T-106.720 Ohjelmistotekniikan projekti

Dynaamisen HTTP-sisällön tarjoaminen välimuistipalvelimelta

Espoo, 11. toukokuuta 2004

Sisältö

Tiivistelmä

Tässä raportissa kuvataan sisällön eksplisiittiseen invalidointiin perustuva reverse proxy[4] -periaatteella toimiva välimuistijärjestelmä, jonka avulla dynaamista sisältöä voidaan tarjota rinnakkain toimivilta välimuistipalvelimilta kuten staattista sisältöä. Välimuistijärjestelmämme pystyy tehokkaasti hyödyntämään HTTP/1.1-protokollassa määriteltyjä validointimekanismeja pienentäen merkittävästi sekä HTTP-palvelimelle että tietoliikenneverkolle aiheutuvia kuormia.

Välimuistijärjestelmämme toiminta perustuu dynaamisesti generoidun HTTP-sisällön ja generoinnissa käytettyjen resurssien välisten riippuvuuksien seurantaan sekä resurssien muuttuessa suoritettavaan välimuistin eksplisiittiseen invalidointiin.

Menetelmän toimivuus ja soveltuvuus tuotantokäyttöön osoitetaan toteuttamalla geneerinen riippuvuus- ja muutosavainten käyttöön perustuva, sisällön etäinvalidointia tukeva välimuistipalvelin, sekä integroimalla resurssiriippuvuuksien ja resursseihin tehtyjen muutosten rekisteröinti WebCMS-julkaisujärjestelmän[1] yhteyteen.

[1] WebCMS on työnimi Teknillisen korkeakoulun Ohjelmistotekniikan projekti -kurssilla ryhmämme toteuttamalle julkaisujärjestelmälle

1 Johdanto

Tässä luvussa esittelemme HTTP/1.1-protokollan tukemat välimuistimekanismit sekä perustelemme, miksi nämä mekanismit ovat riittämättömiä dynaamisesti generoidun HTTP-sisällön tapauksessa. Luomme myös katsauksen nykyaikaisiin dynaamista sisältöä käsitteleviin välimuistiratkaisuihin sekä esittelemme oman välimuistijärjestelmämme toimintaidean. Lopuksi rajaamme raportissa käsiteltävät asiat.

1.1 HTTP/1.1-protokollan välimuistimekanismit

HTTP/1.1 -protokollassa välimuistin käytöllä pyritään karsimaan tarvetta lähettää HTTP-pyyntöjä sekä tarvetta lähettää kokonaisia HTTP-vastauksia[7]. Välimuistien käyttö keventää HTTP-palvelimelle ja tietoverkolle kohdistuvaa kuormitusta sekä pienentää sivustojen latauksiin kuluvia vasteaikoja. HTTP/1.1-protokollassa välimuistin toiminta perustuu ekspiroitumis- ja validointimekanismien käyttöön.

Ekspiroitumismekanismissa ideana on tallentaa pyydetyn URLin sisältö välimuistiin määrätyksi vakioajaksi. Mekanismia voidaan käyttää hyväksi, mikäli tarjotulle sisällölle pystytään arvioimaan tietty ekspriroitumisajankohta, jota ennen sisältö ei tule muuttumaan, tai jos vanhentuneen sisällön tarjoamisesta ei ole olennaista haittaa. Ekspiroitumismekanismia käytettäessä ei voida taata välimuistin läpinäkyvyyttä, koska välimuistista haettu sisältö ei välttämättä ole aina ajantasalla.

Validointimekanismeissa ideana on se, että välimuistissa oleva sisältö validoidaan oikeaksi alkuperäisestä lähteestä aina ennen kuin sitä tarjotaan eteenpäin. HTTP/1.1-protokollassa validointi tapahtuu joko muutospäivämääräehtoja tai ns. entiteettitageja (ETag) käyttäen. HTTP-pyyntö voidaan tehdä ehdollisena siten, että kokonainen vastaus palautetaan ainoastaan, mikäli pyynnössä tehdyt muutospäivämääriin tai entiteettitageihin liittyvät ehdot täyttyvät.

1.2 Dynaamista sisältöä koskevat ongelmat

Dynaamisen sisällön kohdalla HTTP/1.1-protokollan tarjoamat välimuistimekanismit ovat riittämättömiä. Ekspiroitumismekanismia ei voida käyttää tehokkaasti, koska dynaamisen sisällön muuttumisajankohtia ei pystytä ennakoimaan riittävällä tarkkuudella. Liian myöhäiseksi ajastettu ekspiroituminen johtaa vanhentuneen tiedon esittämiseen. Toisaalta liian aikaisin tapahtuva ekspiroituminen aiheuttaa ylimääräistä kuormaa sekä tietoverkolle että HTTP-palvelimelle.

Validointimekanismien käytössä ongelmana on se, että dynaamisesti generoidulle sisällölle ei voida millään yksinkertaisella menetelmällä määrittää viimeistä muutosajankohtaa tai entiteettitageja. Yleisessä tapauksessa muutosajankohdan tai entiteettitagin määrittäminen ovat yhtä raskaita toimenpiteitä kuin oikean vastauksen generoiminen. Hyödyt rajoittuvat tällöin siis tietoliikenneverkolle

kohdistuvan kuorman pienenemiseen. Yleensä dynaamisesti generoidun HTTP-vastauksen muutosajankohdaksi joudutaan asettamaan HTTP-vastauksen luontiajankohta, mikä tekee validointimekanismista täysin hyödyttömän.

Useilla verkkosivustoilla HTML-sisältö generoidaan kuitenkin dynaamisesti. Dynaamisesti generoidut sivut luodaan uudestaan kutakin HTTP-pyyntöä kohden, ja sisältö joudutaan lähettämään aina kokonaisuudessaan riippumatta siitä, löytyykö kyseinen sisältö jo HTTP-pyynnön tehneen koneen välimuistista. Tarve dynaamista sisältöä tukevalle välimuistijärjestelmälle on siis ilmeinen.

1.3 Nykyaikaiset ratkaisut

Dynaamista sisältöä tukevat välimuistimekanismit ovat olleet Internetin yleistymisen myötä paljolti sekä tiedeyhteisön että teollisuuden kiinnostuksen kohteena. Tehokkailla välimuistijärjestelmillä voidaan merkittävästi pienentää tietoliikenneverkolle sekä HTTP-palvelimille kohdistuvia kuormia ja näin ollen saavuttaa konkreettisia säästöjä tietoliikennekuluissa sekä laitteistoinvestoinneissa.

Dynaamisen sisällön käsittelyyn liittyvä tutkimus on vielä varsin nuorta. Tämä näkyy myös ehdotettujen ja toteutettujen ratkaisujen monimuotoisuudessa. Varsin hyvän yleiskuvan nykyaikaisista välimuistitekniikoista sekä aiheeseen liittyvästä tutkimuksesta saa Barishin ja Obraczkan WWW-välimuisteja käsittelevästä artikkelista [3]. Useimmat kirjallisuudessa esitetyt ratkaisut ovat käyttötarkoitukseemme kuitenkin toteutuksen kannalta turhan raskaita.

Oma välimuistijärjestelmämme muistuttaa kirjallisuudesta löytyvistä ratkaisusta eniten Candanin, Lin, Luon, Hsiugin ja Agrawalin esittelemää tietokantapohjaisille WWW-sivustoille suunniteltua välimuistijärjestelmää[5]. Tämä järjestelmä tosin perustuu riippuvuuksien ja muutosten selvittämiseen suoraan tietokantalokeista, joten järjestelmä on omaa välimuistijärjestelmäämme joustavampi joskin samalla myös raskaampi.

Teollisuuden puolella reverse proxy -tekniikoihin perustuvissa ratkaisuissa alan huippua edustavat muunmuassa Akamain ESI-standardia[11] tukeva hajautettu EdgePlatform-järjestelmä[1], Oraclen OracleAS-WebCache[12], Microsoftin Internet Security & Acceleration Server[9] sekä IBM:n WebSphere-järjestelmä[2]. Kustakin näistä löytyy kokonaisratkaisu dynaamisen sisällön hallintaan sekä käsittelyyn.

Oma välimuistijärjestelmämme ei ole suoraan verrattavissa alan johtaviin järjestelmiin. Kehittyneimmät välimuistijärjestelmät ovat huomattavasti monipuolisempia ja laajempia kuin omamme. Kehittyneimmissä järjestelmissä välimuistit toimivat monitasoisina siten, että välimuistiin tallennetaan erikseen muunmuassa tietokantakyselyiden tuloksia sekä yksittäisiä sivuille liitettäviä palasia sekä näiden lisäksi kokonaisia WWW-sivuja. Kehittyneimmissä järjestelmissä välimuistituki on kiinteä osa muuta järjestelmää. Esimerkiksi Akamain ratkaisussa perinteisesti sovelluspalvelimilta löytyvää logiikkaa on siirretty hajautetusti toimiville proxy-palvelimille.

1.4 Oma ratkaisumme

Oma ratkaisumme dynaamiseen sisältöön liittyvään ongelmaan on reverse proxy-periaatteella toimiva, kevyt, sisällön eksplisiittiseen invalidointiin perustuva välimuistijärjestelmä.

Välimuistijärjestelmämme pitää kirjaa generoidun HTTP-sisällön sekä generoinnissa käytetyn lähdemateriaalin välisistä riippuvuuksista. Riippuvuuksia kuvataan merkkijonopohjaisilla riippuvuusavaimilla, jotka identifioivat riippuvuudet.

Kun järjestelmässä olevaan lähdemateriaalin tehdään muutoksia rekisteröidään vastaavat sisältömuutokset. Sisältömuutoksia kuvataan riippuvuusavaimia vastaavilla muutosavaimilla. Vertaamalla välimuistissa olevien resurssien riippuvuusavaimia muutosta kuvaaviin muutosavaimiin nähdään, mitkä resurssit tulee poistaa välimuistista.

Riippuvuuksien syntyminen sekä muutosten rekisteröiminen joudutaan järjestelmäämme käytettäessä räätälöimään sovelluskohtaisesti. Tässä raportissa esittelemme varsinaisen välimuistijärjestelmän lisäksi sen, kuinka riippuvuuksia hallitaan WebCMS-julkaisujärjestelmän tapauksessa.

1.5 Käsiteltävien asioiden rajaus

Raportin pääpaino pidetään dynaamisen sisällön tukemiseen vaadittujen välimuistimekanismien tarkastelussa. Pyrimme antamaan mahdollisimman hyvän kokonaiskuvan toteuttamastamme välimuistijärjestelmästä sivuuttaen teknisten ratkaisujen yksityiskohtaisen käsittelyn. Itse välimuistipalvelimen toteutukseen liittyy monia mielenkiintoisia kysymyksiä koskien säikeistyksestä, synkronointia sekä algoritmivalintoja. Emme kuitenkaan tämän raportin yhteydessä perehdy näihin asioihin pintaa syvemmältä, vaan esittelemme järjestelmän enemmänkin käsitteellisellä tasolla.

Välimuistijärjestelmien yhteydessä tehokkuuskysymykset ovat keskeisessä asemassa. Tehokkuuden kannalta olennaista olisi määrittää se, kuinka paljon järjestelmässämme tapahtuu turhia invalidointeja ja mikä on välimuistijärjestelmän osuma/huti-suhde tai keskimääräinen vasteaika. WebCMS-julkaisujärjestelmä ei ole kuitenkaan tämän raportin kirjoittamisen aikaan vielä tuotantokäytössä, joten oikean mittausdatan puuttuessa emme pysty esittämään luotettavia lukuja toteuttamamme välimuistijärjestelmän tehokkuudesta. Näin ollen tehokkuustarkastelu joudutaan sivuuttamaan tässä raportissa.

2 Kontekstin kuvaus

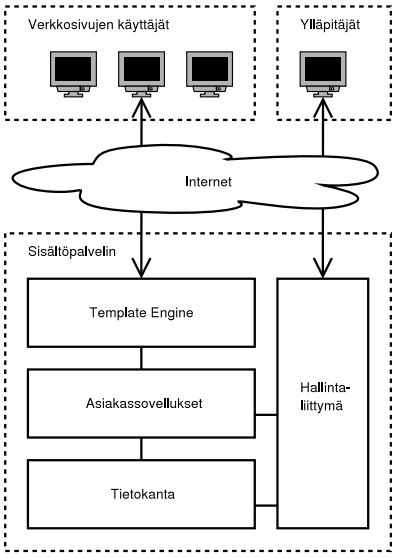
Tässä luvussa esitellemme lyhyesti WebCMS-julkaisujärjestelmän, jonka osaksi välimuistijärjestelmä rakennetaan. Julkaisujärjestelmän toimintaperiaatteet esitetään pelkistettyinä, mutta kuitenkin sillä tarkkuudella, mikä on tarpeen välimuistijärjestelmän toiminnan ymmärtämiseksi.

2.1 Projektin tausta

Välimuistijärjestelmä toteutetaan Teknillisen korkeakoulun T-106.720 Ohjelmistotekniikan projekti -kurssilla ryhmätyönä tehdyn WebCMS-julkaisujärjestelmän yhteyteen. Pyrimme rakentamaan välimuistijärjestelmän mahdollisimman geneeriseksi siten, että sen käyttö on mahdollista myös muissa sovelluksissa. Tässä tapauksessa WebCMS-julkaisujärjestelmä toimii referenssijärjestelmänä, jonka kautta pääsemme lähestymään ongelmaa konkreettisella tavalla.

2.2 WebCMS-julkaisujärjestelmä

WebCMS on Javalla toteuttu operaattorikäyttöön tarkoitettu WWW-pohjainen julkaisujärjestelmä. WebCMS-julkaisujärjestelmän avulla voidaan hallita, julkaista ja tarjota WWW-sivustoja, joille voidaan staattisen sisällön lisäksi liittää dynaamiseen sisältöön perustuvia lisäpalveluita (mm. uutispalvelu, tapahtumakalenteri, kuvapankki). WebCMS-julkaisujärjestelmää hallitaan kokonaisuudessaan WWW-käyttöliittymän kautta.



Kuva 1: WebCMS-julkaisujärjestelmän rakenne ennen välimuistituen lisäämistä

WebCMS-julkaisujärjestelmä koostuu kuvan 1 mukaisesti neljästä osajärjestelmästä.

Template Engine näkyy ulospäin HTTP-palvelimena. Sisäisesti Template Engine toimii template-ohjattuna sisältöintegraattorina, joka yhdistelee asiakassovellusten tarjoaman sisällön sivukokonaisuuksiksi.

Asiakassovelluksiin on toteutettu julkaisujärjestelmään liitettyjen palveluiden logiikka. Asiakassovellukset suorittavat Template Engineltä saamiensa pyyntöjen perusteella kyselyt tietokantaan ja muotoilevat tietokannasta saadut vas-

taukset Template Enginen ymmärtämään muotoon. Kutakin palvelua kohden tarvitaan erillinen asiakassovellus.

Tietokanta toimii taltiona, johon voidaan tallentaa sekä staattisia tiedostoja (esim. template-tiedostot ja kuvat) että sovelluskohtaisia sisältöalkioita (mm. uutispalvelun uutiset tai tapahtumakalenterin tapahtumat). Taltion sisältöä voidaan käsitellä vain ja ainoastaan tietokantajärjestelmän rajapinnan kautta.

Hallintajärjestelmällä hallitaan WebCMS-julkaisujärjestelmän toimintaa. Hallintajärjestelmä jakautuu eri sidosryhmille tarkoitettuihin osiin, joista kullekin tarjotaan rajatut mahdollisuudet julkaisujärjestelmän tietokantasisällön käsittelyyn.

3 Ratkaisulta vaaditut ominaisuudet

Seuraavassa listaamme sekä projektin tilaajan että julkaisujärjestelmän muiden osioiden välimuistijärjestelmälle asettamia vaatimuksia. Lisäksi esittelemme kriteerit, jotka ohjaavat välimuistijärjestelmän suunnittelussa ja toteutuksessa tehtäviä ratkaisuja.

3.1 Ulkoiset vaatimukset

Välimuistijärjestelmään kohdistuu seuraavat projektin tilaajan asettamat vaatimukset:

- järjestelmän on kyettevä tarjoamaan dynaamista sisältöä välimuistista
- järjestelmän on tuettava ns. virtuaalihostausta
- järjestelmän tulee olla skaalautuva
- järjestelmän tulee toimia Linux-alustalla
- järjestelmän toteutuksessa tulee käyttää vain ilmaisia komponentteja
- välimuistin on toimittava läpinäkyvästi

3.2 Sisäiset vaatimukset

Jotta välimuistijärjestelmä voidaan integroida WebCMS-julkaisujärjestelmän yhteyteen tulee ratkaisussa huomioida seuraavat kohdat:

- välimuistijärjestelmän käyttämisen on oltava muun järjestelmän kannalta mahdollisimman läpinäkyvää
- välimuistijärjestelmän tukeminen ei saa hidastaa sivujen generointia merkittävästi

- välimuistijärjestelmä on pystyttävä integroimaan Javalla toteutettuun sisältöpalvelimeen
- välimuistista on tarjottava tiedostoja oikeilla MIME-tyypeillä (tiedot WebCMS-järjestelmältä)

3.3 Muita kriteerejä

- välimuistijärjestelmään yritetään toteuttaa mahdollisimman hyvä suojaus palvelunestohyökkäksiä vastaan
- ennustettavissa olevat HTTP-virhevastaukset (esim. 404) pyritään tallentamaan välimuistiin
- suunnitelussa tärkeintä on sisältöpalvelimen laskentakuorman keventäminen
- sisältöpalvelimen ja välimuistipalvelinten väliselle tietoliikennemäärälle ei ole tiukkoja rajoituksia
- sisältöpalvelin ei näy verkossa ulospäin, joten lyhytaikaisesta sisältöpalvelimen ja välimuistipalvelimen välisestä epäkonsistenttiudesta ei ole merkittävää haittaa
- välimuistipalvelimilla on riittävästi prosessointitehoa, joten tehokkuutta ei tarvitse hakea koodia optimoimalla
- järjestelmän suunnittelussa pyritään siihen, että välimuistipalvelimen tarvitsee lähettää ns. full-content mahdollisimman harvoin
- välimuistipalvelimelta on varattu riittävä määrä levytilaa, jotta merkittävä osa sisältöpalvelimen sisällöstä voidaan pitää yhtä aikaa välimuistissa
- sisällön turhasta invalidoinnista välimuistipalvelimelta ei ole merkittävää haittaa, kunhan koko välimuistin sisältöä ei jouduta tyhjentämään usein
- toteutus pyritään pitämään yksinkertaisena

4 Ratkaisuvaihtoehtoja

Tässä luvussa tarkastelemme erilaisia ratkaisuvaihtoehtoja välimuistijärjestelmän eri osien toteutukseen. Pohdimme myös mahdollisuuksia valmiiden komponenttien käyttämiseen.

4.1 Järjestelmän rakenne

Välimuistijärjestelmän rakenteelle on kaksi ilmeistä rakennevaihtoehtoa: välimuistijärjestelmä voidaan joko integroida suoraan sisältöpalvelimen yhteyteen

tai välimuistijärjestelmä voidaan hajauttaa erillisille välimuistipalvelimille. Ensimmäinen vaihtoehto tehostaisi välimuistin toimintaa pienentyneinä yleiskustannuksina. Asiakkaan vaatimuksien mukaan WebCMS-järjestelmän on kuitenkin oltava hyvin skaalautuva, joten vain jälkimmäinen vaihtoehto on varteenotettava. Välimuistijärjestelmä suunnitellaan kuitenkin sellaiseksi, että välimuistipalvelinta voidaan tarvittaessa pyörittää fyysisesti samalla koneella kuin sisältöpalvelinta.

4.2 Välimuistipalvelin

Välimuistipalvelinohjelmistoksi voimme valita jonkin valmiin ohjelmiston, jossa on tarvittava tuki URLien etäinvalidointiin. Vaihtoehtoisesti voimme tarvittavin osin laajentaa jotakin avoimeen lähdekoodiin perustuvaa välimuistiohjelmistoa tai HTTP-palvelinta.

Tarkastelumme perusteella riittävä tuki sisällön etäinvalidointiin löytyy ainoastaan kehittyneemmistä kaupallisista ratkaisuista (esim. OracleAS Web Cache[12], Microsoft ISA Server 2004[9], Akamai EdgePlatform[1]). Projektin tilaajan vaatimuksien mukaisesti emme voi kuitenkaan käyttää kaupallisia komponentteja, joten joudumme hylkäämään nämä vaihtoehdot.

Ilmaisista välimuistiohjelmistoista kehittynein on GPL-lisenssillä jaettava Squid[13]. Squid tarjoaa perusmekanismin yksittäisten URLien invalidointiin Squid-asiakasohjelman kautta. Haluamme kuitenkin käyttää kehittyneempää, riippuvuus- ja muutosavaimiin perustuvaa invalidointimekanismia, joten emme pysty hyödyntämään Squidia sellaisenaan.

Kolmantena vaihtoehtona on kokonaan uuden välimuistijärjestelmän rakentaminen valmiin HTTP-palvelimen päälle. Pohjana voidaan käyttää mitä tahansa HTTP-palvelinta, joka on toteutettu tarpeeksi modulaarisesti siten, että välimuistipalvelintoiminnallisuuden toteuttaminen on mahdollista. Julkaisujärjestelmän muihin osioihin valittiin käytettäväksi Mortbayn Jetty-HTTP-palvelin[10], joten sen käyttämistä voidaan pitää etusijalla myös välimuistipalvelimen toteutuksessa. Jetty on toteutettu Javalla ja sen lähdekoodia jaetaan ilmaiseksi avoimen lähdekoodin -periaatteella, joten se on myös tältä osin soveltuva välimuistipalvelinalustaksi.

Toteuttamamme välimuistijärjestelmä tulee toimimaan varsin erilaisilla periaatteilla kuin perinteiset välimuistijärjestelmät. Näin ollen voidaan olettaa että Squid-välimuistipalvelimeen täytyisi tehdä varsin mittavia muutoksia. Toisaalta Jetty-HTTP-palvelin ei sisällä valmiina käyttötarkoitukseemme sopivia välimuistitoimintoja, joten Jettyn laajentamiseen joudutaan asettamaan suurempi työpanos. Aikaisempi Jettyn tuntemus ja suurempi toteutuksellinen vapaus kääntää asetelman kuitenkin Jettyn eduksi. Näin ollen päädymme toteuttamaan täysin uuden välimuistijärjestelmän Jetty-HTTP-palvelimen päälle.

4.3 Riippuvuuksien kerääminen

Sisältöpalvelin kerää HTTP-sisällön generoinnin aikana sisällön generoinnissa tarvittavista resursseista sisältöön kohdistuvat riippuvuudet. Ratkaistavana on, missä komponenteissa ja millä tavoin riippuvuudet kerätään. Riippuvuudet voidaan kerätä periaatteessa tietokannan SQL-kyselyistä tai tietokantarajapintaan abstrahoiduista resurssikyselyistä. Vastuu riippuvuuksien rekisteröinnistä voidaan jättää myös asiakassovelluksille.

Järjestelmän sisäisten vaatimusten mukaisesti välimuistijärjestelmän käyttämisen pitäisi olla mahdollisimman läpinäkyvää muun järjestelmän toiminnan kannalta. Riippuvuuksien rekisteröiminen suoraan asiakassovelluksista olisi toteutusteknisesti erittäin virhealtista, joten riippuvuudet kannattaa mieluummin rekisteröidä automatisoidusti yhdessä paikassa.

SQL-kyselyistä päätellyillä riippuvuuksilla päästäisiin esitetyistä vaihtoehdoista hienoimpaan riippuvuusgranulariteettiin, mutta tämän vaihtoehdon toteuttaminen ei ole mahdollista projektiin allokoiduilla resursseilla. Hyvään ratkaisuun päästään onneksi myös muuttamalla tietokantarajapintaan abstrahoidut korkeamman tason tietokantakyselyt resurssipohjaisiksi[2] riippuvuuksiksi. Näin ollen riippuvuuksien kerääminen päädytään integroimaan tietokantarajapinnan yhteyteen.

4.4 Riippuvuuksien tallentaminen

Välimuistipalvelimella tallessa olevan HTTP-sisällön resurssiriippuvuuksista on pidettävä yllä tietoa, jotta resurssien muuttuessa osattaisiin invalidoida vastaava välimuistipalvelimella oleva HTTP-sisältö. Riippuvuuksien tallentamiseen on kaksi olennaisesti erilaista mallia. Riippuvuudet voidaan pitää tallessa joko sisältöpalvelimella tai ne voidaan siirtää välimuistipalvelinten ylläpidettäviksi.

Riippuvuuksien tallentaminen sisältöpalvelimelle vaikuttaa mielekkäältä ratkaisulta. Käytettäessä tätä vaihtoehtoa riippuvuuksia ei tarvitse turhaan siirtää välimuistipalvelimille. Lisäksi riippuvuudet voidaan tallentaa persistentisti tietokantaan. Käytännössä riippuvuudet tallennettaisiin [riippuvuusavain, URL]-pareina ja kun riippuvuusavaimen osoittamaan resurssiin kohdistuisi muutos, lähetettäisiin välimuistipalvelimelle pyyntö tuhota URLia vastaava välimuistissa oleva sisältö. Tämän ratkaisuvaihtoehdon kohdalla välimuistipalvelimena voitaisiin lähtökohtaisesti käyttää Squidia sellaisenaan.

Ongelmaksi osoittautuu kuitenkin virhepyyntöjen ja vanhentuneiden riippuvuuksien ylläpitäminen. Sisältöpalvelimen ei ole mahdollista tietää, missä vaiheessa välimuistipalvelimen välimuisti on täyttynyt, ja mitä sisältöä tämän takia on jouduttu poistamaan välimuistista. Näin ollen sisältöpalvelimella olevien riippuvuuksien määrä voisi kasvaa mielivaltaisen suureksi. Jotta ongelmalta vältyttäisiin, tulisi riippuvuudet ja samalla kaikkien välimuistipalvelinten sisältö tyhjentää säännöllisin väliajoin. Tätä emme kuitenkaan halua.

[2] Resurssipohjainen riippuvuus määritellään luvussa 8.3

Jos riippuvuudet sen sijaan tallennetaan välimuistipalvelimille, saadaan riippuvuuksien tuhoaminen automatisoitua välimuistin vapauttamisen yhteyteen (vrt. välimuistin täyttyminen). Tällä mallilla riippuvuuksia kertyy siis vain rajoitettu määrä. Valitsemme jatkokäsittelyyn mallin, jossa riippuvuudet siirretään ja tallennetaan erikseen kullekin välimuistipalvelimille.

4.5 Riippuvuuksien välittäminen välimuistipalvelimelle

HTTP-vastauksen generoinnin aikana syntyvät riippuvuudet voidaan välittää välimuistipalvelimelle joko HTTP-vastauksen mukana käyttäen hyväksi HTTP-vastaukseen määrettelemiämme laajennusotsakkeita tai erillisen tietoliikenneyhteyden avulla käyttäen hyväksi HTTP-POST-metodia tai jotakin muuta tiedonsiirtoprotokollaa.

Riippuvuusavainten välittäminen HTTP-laajennusotsakkeissa varsinaisen HTTP-sisällön mukana on edellämainituista vaihtoehdoista toteutuksellisesti huomattavasti elegantimpi ratkaisu. Riippuvuuksien tallentaminen välimuistipalvelimelle voidaan tällöin toteuttaa luonnolliseen tapaan varsinaisen sisällön tallentamisen yhteyteen. Ongelmana tässä lähestymistavassa on kuitenkin se, että välimuistijärjestelmän riippumattomuus sisältöpalvelinohjelmiston toteutuksesta kärsii. Käytännössä tämä tarkoittaa sitä, että sisällöngeneroijan on pystyttävä manipuloimaan HTTP-otsakkeita ja rakennettava sivu kokonaisuudessaan ennen HTTP-otsakkeiden lähettämistä, koska HTTP-otsakkeisiin liitettävät riippuvuudet saadaan selville vasta sivun generoinnin aikana. Tämä ei kuitenkaan ole ongelma ainakaan WebCMS-julkaisujärjestelmän tapauksessa, joten riippuvuudet päädytään välittämään välimuistipalvelille varsinaisen HTTP-vastauksen laajennusotsakkeissa.

4.6 Sisältömuutosten rekisteröinti

Sisältömuutosten rekisteröinti suoritetaan tietokantarajapinnassa vastaavalla tavalla kuin riippuvuuksien rekisteröinti. Muutosten rekisteröinti halutaan tehdä konsistentisti riippuvuuksien rekisteröintien kanssa, joten muunlaista ratkaisua ei ole järkevää edes harkita.

4.7 Invalidointien välittäminen välimuistipalvelimille

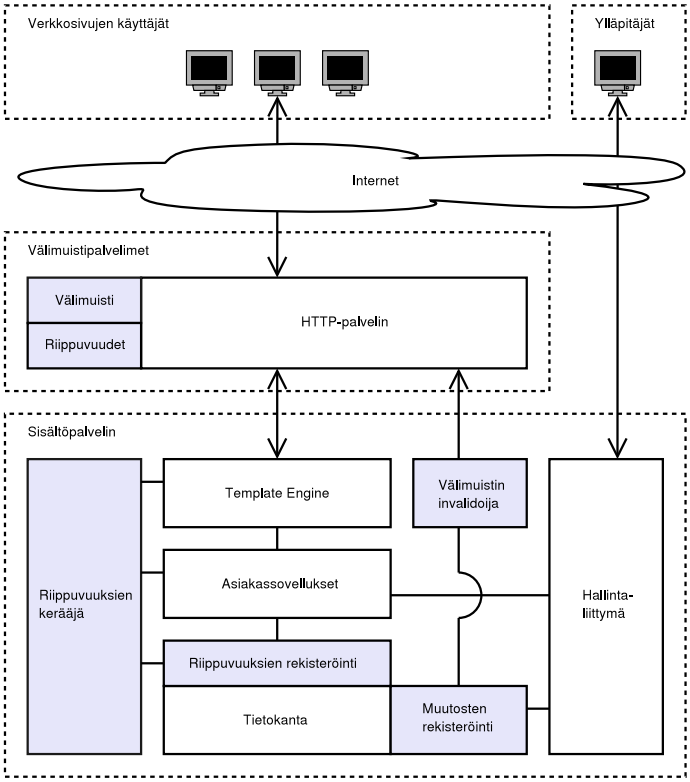
Sisällön invalidoinnin yhteydessä muutosavaimet on välitettävä sisältöpalvelimelta välimuistipalvelimille. Käytännössä tämä voidaan voidaan tehdä joko HTTP-POSTin tai jonkin muun protokollan avulla. HTTP-DELETEä ei voida käyttää, koska sisältöpalvelimella ei ole tiedossa invalidoitavia URLeja, vaan ainoastaan muutosavaimet. Välimuistipalvelimelta löytyy valmiina tuki HTTP-POST-metodin käyttöön, joten sen valinta invalidointiavainten siirtoon on melko ilmeinen.

5 Välimuistijärjestelmän yleiskuvaus

Tässä luvussa kuvaamme välimuistijärjestelmän rakenteen sekä välimuistijärjestelmän yleiset toimintaperiaatteet. Tarkemmman katsauksen välimuistipalvelimen toteutukseen teemme luvussa 6. Sisältöpalvelimen välimuistitukeen tutustumme luvussa 7.

5.1 Arkkitehtuuri

Luvussa 4 käydyn tarkastelun pohjalta päädyttiin kuvan 2 mukaiseen välimuistijärjestelmän arkkitehtuuriin (vrt. toteutus ilman välimuistitukea kuvassa 1).



Kuva 2: Välimuistijärjestelmän arkkitehtuuri

Järjestelmä koostuu edustalla toimivasta kuormantasaajasta (ei kuvassa), mielivaltaisen monesta välimuistipalvelimesta sekä taustalla toimivasta sisältöpalvelimesta. Tässä raportissa emme ota erityisemmin kantaa kuormantasaajan toteutukseen.

5.2 Toimintaperiaatteet

Ulkoapäin tulevat HTTP-pyynnöt ohjataan kuormantasaajan kautta jollekin välimuistipalvelimista (esim. Round-Robin DNS:n avulla). Mikäli pyydetty resurssi löytyy välimuistipalvelimelta, se palautetaan välittömästi asiakkaalle. Muussa tapauksessa välimuistipalvelin suorittaa vastaavan HTTP-pyynnön sisältöpalvelimelle. Sisältöpalvelin käsittelee pyynnön keräten samalla joukon riippuvuuksia, joista ilmenee mitä resursseja vastauksen generoinnissa käytettiin. Sisältöpalvelin välittää riippuvuusavaimet sekä mahdollisen sisällön vanhentumisajankohdan HTTP-vastauksen mukana välimuistipalvelimelle. Välimuistipalvelin rekisteröi riippuvuudet, tallentaa vastauksen välimuistiin ja palauttaa vastaavan HTTP-vastauksen asiakkaan WWW-selaimelle.

Välimuistipalvelimelle tallennettu sisältö on voimassa niin pitkään kunnes yksi tai useampi generoinnissa käytetyistä resurseista muuttuu tai sisällön vanhentumisajankohta saavutetaan. Resurssien muuttumista seurataan rekisteröimällä sisältöpalvelimelle tallennettuun sisältöön tehdyt muutokset ja lähettämällä tiedot muutoksista kaikille välimuistipalvelimille. Välimuistipalvelimet vertaavat muutosavaimia oman sisältönsä riippuvuusavaimiin ja poistavat muuttuneen sisällön välimuistista. Kukin välimuistipalvelin huolehtii itsenäisesti omassa välimuistissaan olevan vanhentuneen sisällön invalidoinnista.

6 Välimuistipalvelin

Tässä luvussa perehdymme välimuistipalvelimen rakenteeseen ja toimintaperiaatteisiin sekä esittelemme välimuistipalvelimen teknistä toteutusta.

6.1 Rakenne

Välimuistipalvelimen pohjana käytetään Mortbayn Jetty-HTTP-palvelinta. Itse palvelimen runkoon ei tehdä mitään muutoksia, vaan välimuistituki lisätään Jettyyn toteuttamamme HTTP-pyyntöjen käsittelijän avulla. HTTP-pyyntöjen käsittelijä peritään Jettyn AbstractHttpHandler-luokasta, joka tarjoaa peruspalvelut HTTP-pyyntöjen käsittelylle. Välimuistitoteutuksen sisältävä HTTP-pyyntöjen käsittelijä rekisteröidään Jettyn HTTP-kontekstiin, joka välittää HTTP-palvelimelle tulevat HTTP-pyynnöt omalle HTTP-pyyntöjen käsittelijällemme. Jettyn HttpHandler-rajapinta välittää käsittelypyynnön yhteydessä myös HTTP-vastaus-olion, johon HTTP-vastaus voidaan generoida.

HTTP-pyyntöjen kuuntelemista varten Jettylle avataan kaksi palvelinistukkaa, joista toinen kuuntelee järjestelmän ulkopuolelta tulevia HTTP-sisältöpyyntöjä ja toinen sisältöpalvelimelta tulevia sisällön invalidointipyyntöjä. Sisältöpyyntöjä ja invalidointipyyntöjä kuunnellaan eri porteista, jotta invalidointipyyntöjen tuleminen järjestelmän ulkopuolelta voidaan suojata palomuurin avulla.

Toiminnallisesti välimuistitoteutuksesta voidaan erottaa seuraavat komponentit: sisältötaltio, sisältöpyynnön käsittelijä, sisällön lataaja, invalidointipyynnön

käsittelijä ja ekspiroitumisten tarkkailija. Edellämainitut komponentit ovat toteutusteknisistä syistä hyvin kiinteässä yhteydessä toisiinsa, eikä komponenttien jako ole käytännössä näin selvä.

6.2 Sisältötaltio

Sisältötaltio koostuu itse välimuistissa olevasta HTTP-sisällöstä, HTTP-sisältöön assosioiduista riippuvuusavaimista sekä molempien edellämainittujen tehokkaan käsittelyn mahdollistavista indeksoinneista. HTTP-sisältö tallennetaan välimuistiin URL-perusteisesti siten, että yhtä sisältöpalvelimelta saatua HTTP-vastausta vastaa aina yksi sisältötaltioon tallennettu resurssi.

Välimuistissa olevat resurssit indeksoidaan hajautustaulukkoon käyttäen hakuavaimena HTTP-pyyntöä vastaavaa URL-osoitetta. Näin ollen HTTP-pyynnön saapuessa HTTP-pyyntöä vastaava resurssi löydetään muistista vakioajassa. Tämän lisäksi resurssit indeksoidaan toiseen hajautustauluun käyttäen hakuavaimina resurssiin liittyviä riippuvuusavaimia. Täten myös invalidointipyynnön invalidointiavaimet pystytään yhdistämään vakioajassa invalidoitaviin resursseihin.

Sisältötaltion resursseja kuvataan erillisillä keskusmuistissa säilytettävillä resurssiolioilla. Resurssioliot sisältävät HTTP-vastauksen generointiin tarvittavat otsaketiedot, jotka ovat sisällöltään identtiset sisältöpalvelimelta saadun HTTP-vastauksen kanssa (pl. ekspiroitumisajankohta). Välimuistipalvelimen toiminnan kannalta olennaisia otsaketietoja ovat sisällön tyyppi, sisällön koko, HTTP-vastauskoodi, sisällön viimeinen muutospäivämäärä sekä mahdollinen uudelleenohjaussijainti. Otsaketietojen lisäksi resurssiolioihin tallennetaan resurssin riippuvuusavainjoukko sekä tiedot resurssin sisäisestä tilasta (mm. lukijoiden määrä ja resurssin latauksen vaihe).

Varsinainen HTTP-vastauksen sisältö tallennetaan välimuistipalvelimen massamuistiin siten, että kutakin välimuistissa olevaa resurssia vastaa yksi massamuistissa oleva tiedosto. Tiedostot tallennetaan erilliseen välimuistipalvelimen käyttöön varattuun hakemistoon käyttäen tiedostonimenä juoksevaa kirjainnumerotunnistetta. Tällaisella nimeämiskäytännöllä vältetätään mahdollisesta URL-osoitteiden ja levyjärjestelmän tukemien merkistöjen välisistä eroavaisuuksista johtuvat ongelmat.

6.3 Sisältöpyyntöjen käsittely

Sisältöpyynnön käsittelijän tehtävänä on ottaa vastaan asiakkailta tulevia HTTP-pyyntöjä ja palauttaa vastauksena pyyntöä vastaava sisältö. HTTP-pyynnön saavuttua sisältöpyynnön käsittelijä pyytää sisältötaltiolta jaetun lukuoikeuden sisältötaltiossa olevaan resurssiin. Resurssi suojataan ekslusiivisella lukolla lukuoikeuden pyytämisen ajaksi. Mikäli kyseinen resurssi ei ole ennestään sisältötaltiossa, käynnistetään taustalle erillinen säie, joka hakee HTTP-pyyntöä vastaavan sisällön sisältöpalvelimelta. Sisältötaltio pitää resurssin lukittuna eksplisiittisesti kunnes resurssiin liittyvät HTTP-otsakkeet on saatu haettua sisältöpalvelimelta ja kunnes sisällön lataus on käynnistetty. Jos resurssi vastaavasti

löytyy jo välimuistista, resurssin ekslusiivinen lukko voidaan vapauttaa heti antaen samalla sisältöpyynnön käsittelijälle resurssin lukuoikeus.

Edellä kuvatulla lukituskäytännöllä estetään se, että erillisillä, samaa URL-osoitetta koskevilla HTTP-pyynnöillä ei haeta turhaan samaa resurssia sisältöpalvelimelta. Sisältötaltio pitää huolen siitä, että sisällön hakeminen sisältöpalvelimelta käynnistyy ainoastaan ensimmäisen samaa resurssia koskevan HTTP-pyynnön yhteydessä. Tätä seuraavat HTTP-pyynnöt odottavat ekslusiivisen lukon takana kunnes resurssi on alustettu siihen tilaan, että HTTP-otsakkeet on vastaanotettu. Tämän jälkeen välimuistipalvelimelta lähetetään asiakkaalle HTTP-otsakkeista sekä HTTP-pyynnöstä riippuen samanlainen vastaus kuin mitä sisältöpalvelin olisi lähettänyt (sisältäen myös persistentit virhevastaukset sekä mahdolliset uudelleenohjaukset).

HTTP-vastauksen varsinainen sisältöä ei lähetetä, mikäli HTTP-pyyntö oli HEAD-tyyppinen tai mikäli HTTP-pyyntö on tehty ehdollisena ja ehtojen mukaan sisältöä ei kuulu lähettää. Muussa tapauksessa HTTP-vastauksen sisältö luetaan massamuistissa jo olevasta tai massamuistiin samanaikaisesti taustalla tallentuvasta tiedostosta ja ohjataan kahdeksan kilotavun kokoisen muistipuskurin kautta HTTP-vastauksen tulostevirtaan.

Kun HTTP-sisältö on lähetetty kokonaisuudessaan HTTP-pyynnön tekijällä, sisältöpyynnön käsittelijä vapauttaa resurssiin liitetyn lukusalvan. Sama toimenpide suoritetaan myös, mikäli HTTP-pyynnön käsittely keskeytyy jostakin syystä (esim. yhteyden katkeaminen tai odottamattomat virhetilanteet). Välimuistin toiminnan kannalta on ensiarvoisen tärkeää, että välimuistijärjestelmä pystyy toipumaan kaikista virhetilanteesta. Näin ollen poikkeustenkäsittelyyn liittyviin mekanismeihin on kiinnitetty toteutuksessa erityistä huomiota.

6.4 Sisällön hakeminen sisältöpalvelimelta

Sisällön lataaja hakee sisältöä sisältöpalvelimelta käyttäen hyväksi HTTP-protokollaa. Sisällön hakeminen tehdään kahdessa vaiheessa. Ensimmäisessä vaiheessa HTTP-pyyntöä käsittelevä säie ottaa yhteyden sisältöpalvelimeen ja odottaa, että välimuistipalvelin saa sisältöpalvelimelta HTTP-vastauksen otsaketiedot. Otsaketiedot tallennetaan muistiin, jotta samat tiedot voidaan välittää edelleen välimuistipalvelimelta ulospäin. Välimuistipalvelin poimii otsaketiedoista myös resurssiin liitetyt riippuvuusavaimet sekä mahdollisen aikariippuvuuden. Riippuvuusavaimet tallennetaan sekä resurssia kuvaavan olion riippuvuusjoukkoon, että riippuvuusavainpohjaiseen resurssiolioiden hajautustauluun. Aikariippuvuus rekisteröidään ajastetuista invalidoinneista huolehtivan säikeen invalidointijonoon. Otsaketietojen saavuttua välimuistiin tallennettu resurssi on esialustettu ja se voidaan tarjota muidenkin HTTP-pyyntöjä käsittelevien säikeiden käyttöön.

Ensimmäisen vaiheen suorituksen jälkeen HTTP-pyyntöä käsittelevä säie käynnistää taustalle erillisen lataussäikeen, joka huolehtii varsinaisen HTTP-sisällön lataamisesta sisältöpalvelimelta. Lataajasäie lukee sisältöpalvelimelta tulevaa sisältövirtaa muistipuskuriinsa maksimissaan kahdeksan kilotavun lohkoissa. Muistipuskurin täytyttyä (kokonaan tai osittain) sisällön lataaja pyytää sisältötal-

tiolta muistipuskurissa olevan sisällön kokoa vastaavan määrän tilaa massamuistista. Sisältötaltio pitää huolen siitä, että massamuistissa olevien tiedostojen yhteiskoko ei kasva yli ennaltamäärätyn rajan. Jos tämä raja ylitetään, sisältötaltio aloittaa resurssien invalidoinnin LRU-periaatteen mukaisesti. Tilapyynnön käsittelyn jälkeen lataajasäie kirjoittaa muistipuskurissa olevan sisällön resurssitiedoston loppuun, ja signaloi sisältöä odottaville HTTP-pyynnön käsittelijöille tiedon siitä, että tiedostoon on tullut lisää materiaalia.

Edellä kuvatun järjestelyn ansiosta sisältöpyynnön käsittelijä voi siirtää sisältöpalvelimelta tulevaa sisältöä asiakkaalle jo ennen kuin sisältö on kokonaisuudessaan ladattu välimuistipalvelimelle. Lisäksi useampi sisältöpyyntöjen käsittelijä voi käsitellä tulevaa sisältövirtaa samanaikaisesti. Tästä järjestelystä on hyötyä lähinnä silloin, kun sisältöpalvelimelta haetaan isoja tiedostoja: liian pitkät viiveet HTTP-pyynnön vastaanottamisen ja HTTP-vastauksen lähettämisen välillä aiheuttaisivat turhia yhteyksien katkaisemisia ja näin ollen myös ylimääräistä tietoliikennettä. Haarautetun streamauksen ansiosta sisältöpalvelimelle ei myöskään koskaan suoriteta samanaikaisesti turhia HTTP-pyyntöjä.

6.5 Invalidointien käsittely

Välimuistipalvelin kuuntelee sisältöpalvelimelta tulevia invalidointipyyntöjä erilliseen invalidointiporttiin sidotussa palvelinistukassa. Invalidointipyynnöt välitetään sisältöpalvelimen ja välimuistipalvelimen välillä käyttäen hyväksi HTTP-POST-metodia, jonka hyötykuorma koostuu pilkulla erotetusta listasta muutosavaimia. Invalidointien käsittelijän tehtävänä on verrata muutosavaimia välimuistissa olevien resurssien riippuvuusavaimiin ja poistaa välimuistista sellaiset resurssit, joiden kohdalla riippuvuusavainjoukon ja muutosavainjoukon leikkaus on ei-tyhjä joukko. Yksittäisen muutosavaimen vertailu voidaan tehdä koko resursijoukolle vakioajassa käyttäen hyväksi riippuvuusavaimia resurssien hakuavaimena käyttävää hajautustaulua.

Invalidoinnin yhteydessä resurssi poistetaan välittömästi resurssit indeksoivasta hajautustaulusta, jotta seuraavat HTTP-pyynnöt eivät voi saada enää kyseistä resurssia käyttöönsä. Resurssia ei kuitenkaan tuhota heti, jos resurssin lukeminen on vielä kesken yhdellä tai useammalla sisältöpyynnön käsittelijällä. Resurssin tuhoaminen jää lopulta sen sisältöpyynnön käsittelijän vastuulle, joka lopettaa resurssin lukemisen viimeisenä. Mikäli resurssin invalidoinnin ja resurssin tuhoamisen välisenä aikana samaa URL-osoitetta vastaavaa resurssia pyydetään uudestaan, käynnistetään päivitetyn resurssin lataaminen sisältöpalvelimelta kuten normaalitilanteessa. Näin ollen välimuistipalvelimella voi olla samasta resurssista tallessa samanaikaisesti useampia instansseja.

Erityishuomiota välimuistipalvelimella joudutaan kiinnittämään sellaiseen suoritusjärjestyksen kannalta poikkeukselliseen tilanteeseen, jossa invalidointipyyntö vastaanotetaan jonkun välimuistiin tallennettavan resurssin esialustuksen aikana. Mikäli resurssia koskeva HTTP-pyyntö on jo lähetetty sisältöpalvelimelle, mutta sisältöpalvelin ei ole vielä palauttanut HTTP-vastauksen otsakkeita, välimuistipalvelimella ei ole vielä invalidointipyynnön saapuessa tietoa siitä, tuleeko kyseinen resurssi invalidoida. Näin ollen invalidointipyynnön muutosavaimet

joudutaan jokaisen invalidointipyynnön yhteydessä lisäämään kaikkien sellaisten resurssien muutosavainjoukkoihin, joiden esialustus on vielä kesken. Tällaiset resurssit invalidoidaan esialustuksen lopuksi, mikäli otsaketiedoissa olleen riippuvuusavainjoukon ja esialustuksen aikana saapuneiden muutosavainten joukon leikkaus on ei-tyhjä joukko.

6.6 Ajastetut invalidoinnit

Välimuistipalvelin huolehtii ajastetuista invalidoinneista erillisellä säikeellä. Invalidoitavat resurssit tallennetaan invalidointiajankohdan mukaisesti järjestettyyn prioriteettijonoon, josta invalidointisäikeen on helppo poimia invalidoitavat resurssit tehokkaasti. Jos jokin invalidointijonossa oleva resurssi invalidoidaan resurssin muuttumisen takia, tämän resurssin ajastettu invalidointi peruuntuu ja resurssi poistetaan välittömästi invalidointijonosta.

6.7 Toiminta välimuistin täyttyessä

Sisältötaltiossa olevien resurssien määrä sekä resurssien kuluttama kokonaislevytila voidaan rajoittaa konfiguraatiotiedoston avulla tiettyihin maksimiarvoihin. Näillä rajoittimilla saadaan optimoitua välimuistin suorituskyky välimuistipalvelimen keskusmuistin ja massamuistin määrälle sopivaksi.

Resurssimäärän tai kokonaislevytilavaatimuksen kasvettua liian suureksi välimuistista poistetaan tallessa olevia resursseja LRU-periaatteen mukaisesti. LRU-algoritmin käyttäminen on perusteltua, koska sen implementoiminen on helppoa ja koska sen on todettu toimivan riittävän hyvin vastaavissa järjestelmissä[6]. Välimuistin täytyttyä uuden resurssin tallentaminen välimuistiin käynnistää aina yhden tai useamman resurssin invalidoimisen. Invalidoinneista aiheutuva kuormitus jakautuu näin ollen tasaisesti kullekin HTTP-pyynnölle, joten kuormituspiikkejä ei pääse syntymään samalla tavalla kuin, jos invalidoinnit tehtäisiin koottuina eräajoina. Toisaalta invalidointien suorittaminen eräajoina saattaisi vähentää HTTP-vastauksen generoinnin keskimääräistä kestoa.

7 Sisältöpalvelin

Tässä luvussa tarkastelemme välimuistijärjestelmän toteutusta sisältöpalvelimen päässä. Esittelemme toteutuksen mahdollisimman yleisellä tasolla, emmekä ota kantaa WebCMS-julkaisujärjestelmää koskeviin erikoispiirteisiin. Sisältöpalvelimen integrointi WebCMS-järjestelmään esitellään erikseen luvussa 8.

7.1 Rakenne

Välimuistijärjestelmä voidaan yhdistää Javalla toteutettuihin HTTP-palvelimiin integroimalla palvelimen yhteyteen sisältöriippuvuuksien kerääminen sekä sisäl-

tömuutosten rekisteröinti. Välimuistijärjestelmän toteutus ei ole ohjelmointikieliriippuvainen, joten sisältöpalvelinosuus voidaan toki toteuttaa uudelleen muillakin ohjelmointikielillä.

Sisältöriippuvuuksien kerääminen tapahtuu käyttäen erillistä riippuvuuksien kerääjää, jolle voidaan tallentaa HTTP-vastauskohtaiset riippuvuusavaimet sekä mahdollinen sisällön ekspriroitumisajankohta. Sisältömuutosten rekisteröinti tehdään erilliselle välimuistin invalidoijalle, jonka tehtävänä on välittää tiedot muutoksista kullekin välimuistipalvelimelle. Edellämainittuihin komponentteihin on lisäksi toteutettava dynaamiset riippuvuudet huomioiva riippuvuusavainten luontilogiikka.

7.2 Riippuvuus- ja muutosavaimet

Riippuvuusavaimet ovat merkkijonoja, joilla identifioidaan sisältöpalvelimen palauttaman HTTP-vastauksen riippuvuudet. Riippuvuusavaimia vastaavilla muutosavaimilla voidaan sen sijaan ilmaista tarve invalidoida kyseinen HTTP-vastaus. Riippuvuus- ja muutosavainten semantiikka rajoittuu avainten välisen ekvivalenttisuuden vertailuun.

Välimuistimallimme ei rajoita avaimilla kuvattavien riippuvuuksien tai vastaavien muutosten luonnetta, vaan nämä voidaan määritellä täysin sovelluskohtaisesti. Luvussa 8 esittelemme WebCMS-julkaisujärjestelmässä esiintyvät riippuvuudet sekä näytämme, kuinka niiden perusteella muodostetaan tarvittavat riippuvuus- ja muutosavaimet.

7.3 Aikariippuvuus

HTTP-vastaukset voidaan varustaa erillisellä ekspiroitumissajankohdalla, joka kertoo milloin sisältö tulee poistaa välimuistista. Jos verkkosivuilla näytetään esimerkiksi päivän mietelausetta tai muuta vastaavaa ajoitettua sisältöä, on aikariippuvuuksien käyttäminen välttämätöntä kätkökonsistenssin säilyttämiseksi.

7.4 Riippuvuuksien kerääminen

Sisältöpalvelin luo kutakin HTTP-pyyntöä varten oman riippuvuuksien kerääjä -olion. Riippuvuuksien kerääjään rekisteröidään HTTP-vastauksen generoinnin aikana syntyvät dynaamiset riippuvuudet sekä mahdollinen sisällön ekspiroitumisajankohta. Riippuvuuksien kerääjä muuttaa rekisteröidyt riippuvuudet sovelluskohtaisen logiikan mukaisesti riippuvuusavaimiksi ja lisää ne ylläpitämäänsä riippuvuusavainjoukkoon. Mikäli riippuvuuksien kerääjälle rekisteröidään useampi kuin yksi ekspiroitumisajankohta, vain aikaisin näistä ajankohdista huomioidaan.

7.5 Riippuvuuksien välittäminen välimuistipalvelimelle

Kun sisältöpalvelin on saanut generoitua HTTP-vastauksen sisällön valmiiksi, on riippuvuuksien kerääjään kertynyt joukko riippuvuusavaimia sekä mahdollinen sisällön ekspiroitumisajankohta. Riippuvuusavaimet (ja ekspiroitumisajankohta) välitetään välimuistipalvelimelle yhdessä varsinaisen sisällön kanssa käyttäen hyväksi HTTP-vastaukselle määriteltyjä laajennusotsakkeita:

X-Cache-Dependencies: riippuvuusavaimet pilkulla erotettuna listana
X-Cache-Expires: ekspiroitumisajankohta RFC2616:n määrittämässä full date-muodossa

7.6 Muutosten rekisteröinti

Sisältöpalvelimen on informoitava välimuistipalvelimia sellaisista sisältöpalvelimen tilassa tapahtuvista muutoksista, joilla voi olla vaikutusta välimuistissa olevaan sisältöön. Informointi tapahtuu rekisteröimällä muutokset sisältöpalvelimen yhteydessä toimivalle välimuistin invalidoijalle, joka välittää tiedot muutoksista edelleen kullekin välimuistipalvelimelle.

Muutosten rekisteröinnin yhteydessä luodaan muutosta kuvaavat muutosavaimet sekä lisätään ne invalidointia ohjaavan kontrollisäikeen lähetysjonoon. Tällä järjestelyllä taataan se, että muutosten rekisteröinti on mahdollisimman nopeata rekisteröinnin suorittavan säikeen (ja lopulta myös WWW-hallintaliittymän käyttäjän) kannalta (vrt. järjestelmän sisäiset vaatimukset).

Kontrollisäie välittää muutosavaimia lähetysjonostaan jatkuvalla syötöllä invalidointitietoliikenteestä huolehtiville tietoliikennesäikeille. Kukin tietoliikennesäie vastaa tämän jälkeen siitä, että muutosavaimet saadaan välitettyä omalle välimuistipalvelimelle.

7.7 Muutosavainten välitys välimuistipalvelimille

Tietoliikennesäikeet lähettävät muutosavaimet välimuistipalvelimille tekstimuodossa käyttäen hyväksi HTTP-POST-metodia. Muutosavainten lähettäminen tapahtuu säännöllisin väliajoin siten, että yhteen lähetykseen saadaan tarvittaessa kerättyä muutosavaimia useammasta peräkkäin tehdystä muutoksesta. Tämä on tarkoituksenmukaista, sillä sisältöpalvelimen sisältöön tehtävät muutokset suoritetaan tyypillisesti ryppäissä. Muutosavainten lähettäminen erikseen yksittäisien HTTP-pyyntöjen mukana aiheuttaisi huomattavan hukkakuorman saavutettuun hyötykuormaan verrattuna. Toisaalta muutaman sekunnin viiveellä tapahtuvasta invalidoinnista ei ole välimuistipalvelimella olevan sisällön tuoreuden kannalta huomattavaa merkitystä, joten pienen viiven käyttäminen on perusteltua. Tällainen ratkaisu on myös vaatimuksissa esitettyjen kriteerien mukainen.

Muutosavainten lähettämisen lisäksi sisältöpalvelin voi pyytää välimuistipalvelinta tyhjentämään koko välimuistinsa. Tämä tulee kysymykseen silloin, kun tie-

toliikennesäikeen lähetysjonoon on kertynyt liikaa muutosavaimia (esim. tietoliikenneongelmien takia). Kaikki välimuistit tyhjennetään myös sisältöpalvelimen käynnistyksen yhdeydessä. Tällä varmistetaan se, että välimuistipalvelimille ei jää missään tilanteessa vanhentunutta sisältöä.

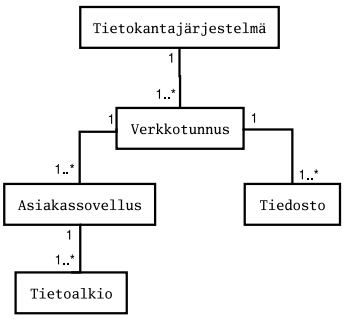
8 Integrointi WebCMS-julkaisujärjestelmään

Tässä luvussa esittelemme, kuinka välimuistijärjestelmän sisältöpalvelinosuus saadaan integroitua WebCMS-julkaisujärjestelmän yhteyteen. Esittelemme tietokannan resurssihierarkian sekä riippuvuusrelaation, jolla kuvaamme resurssipohjaisen dynaamisen sisällön generoinnissa syntyviä riippuvuuksia. Näiden perusteella muodostamme WebCMS-julkaisujärjestelmässä esiintyvät resurssiriippuvuudet ja näytämme, kuinka riippuvuudet syntyvät ja kuinka resursseissa tapahtuvat muutokset rekisteröidään invalidoinneiksi.

8.1 Toteutuksen läpinäkyvyys

Kaikkien riippuvuuksien rekisteröinti (pl. aikariippuvuus) suoritetaan kootusti tietokantarajapinnan takana. Näin ollen erillisten asiakassovellusten tai julkaisujärjestelmän hallintaliittymän ei tarvitse olla tietoisia riippuvuuksien rekisteröintiin liittyvästä logiikasta. Tällä järjestelyllä saavutetaan toteutuksellinen läpinäkyvyys, mikä esitettiin luvussa 3 eräänä järjestelmään kohdistuvana vaatimuksena. Mahdollinen aikariippuvuus joudutaan rekisteröimään erikseen asiakassovelluksesta käsin.

8.2 Tietokantasisällön rakenne



Kuva 3: Tietokannan resurssihierarkia

Kaikki WebCMS-julkaisujärjestelmässä julkaistava materiaali tallennetaan talteen tietokantarajapinnan kautta. Tietokannassa oleva tietosisältö on organisoitu kuvan 3 mukaisella tavalla. Tietokantaan tallennetut tiedostot assosioidaan aina tiettyyn verkkotunnukseen. Myös asiakassovelluksista (esim. uutispalvelu,

tapahtumakalenteri) luodut asiakassovellusinstanssit ovat verkkotunnuskohtaisia. Asiakassovellusinstansseihin voidaan edelleen assosioida yksittäisiä tietoalkioita (esim. uutisartikkeli, tapahtumakalenterin tapahtuma). Jatkossa nimitämme verkkotunnuksia, tiedostoja, asiakassovelluksia sekä asiakassovellusten tietoalkioita yhteisellä nimellä resursseiksi.

8.3 Resurssiriippuvuuden määritelmä

Kuvaamme seuraavan mallin avulla WebCMS-järjestelmän resurssipohjaisia riippuvuuksia.

Sisältöpalvelimen generoimalla HTTP-vastauksella on riippuvuus sisältöpalvelimen resurssiin R, jos ja vain jos:

- HTTP-vastaus muuttuu, kun resurssi R lisätään sisältöpalvelimelle
- HTTP-vastaus muuttuu, kun resurssi R poistetaan sisältopalvelimelta
- HTTP-vastaus muuttuu, kun resurssi R muuttuu sisältöpalvelimella

8.4 Riippuvuustyypit

WebCMS-julkaisujärjestelmää varten määrittelemme seuraavat resurssiriippuvuudet: verkkotunnusriippuvuus, tiedostoriippuvuus, asiakassovellusriippuvuus, asiakassovellusjoukkoriippuvuus, alkioriippuvuus sekä alkiojoukkoriippuvuus. Riippuvuustyyppien valinnassa on pyritty sellaiseen granulariteettiin, jolla turhien invalidointien määrä saadaan pidettyä alhaisena, mutta jolla aktiivisten riippuvuuksien joukko pysyy kuitenkin helposti hallittavissa.

Verkkotunnusriippuvuus

Sisällöllä on riippuvuus tiettyyn verkkotunnukseen, mikäli tietokannasta pyydettiin sisällön generoinnin aikana mitä tahansa kyseiselle verkkotunnukselle spesifistä tietoa.

Tiedostoriippuvuus

Sisällöllä on riippuvuus tiettyyn verkkotunnukseen liitettyyn tiedostoon, mikäli tietokannasta pyydettiin sisällön generoinnin aikana kyseisen verkkotunnuksen kyseistä tiedostoa.

Asiakassovellusriippuvuus

Sisällöllä on riippuvuus tiettyyn verkkotunnukseen liitettyyn asiakassovellukseen, mikäli tietokannasta pyydettiin sisällön generoinnin aikana mitä tahansa kyseisen verkkotunnuksen kyseiselle asiakassovellukselle spesifistä tietoa.

Asiakassovellusjoukkoriippuvuus

Sisällöllä on riippuvuus tietyn verkkotunnuksen asiakassovellusjoukkoon, mikäli tietokannasta pyydettiin sisällön generoinnin aikana kyseisen verkkotunnuksen asiakassovellusjoukkoa.

Alkioriippuvuus

Sisällöllä on riippuvuus tiettyyn verkkotunnukseen liitetyn asiakassovelluksen tiettyyn tietoalkioon, mikäli tietokannasta pyydettiin sisällön generoinnin aikana kyseisen verkkotunnuksen ja asiakassovelluksen kyseistä tietoalkiota yksilöivällä avaimella.

Alkiojoukkoriippuvuus

Sisällöllä on riippuvuus tiettyyn verkkotunnukseen liitetyn asiakassovelluksen tietoalkiojoukkoon, mikäli tietokannasta pyydettiin sisällön generoinnin aikana kyseisen verkkotunnuksen ja asiakassovelluksen jotakin tietoalkiota muulla kuin yksilöivällä avaimella.

Aikariippuvuus

Sisällöllä on aikariippuvuus, mikäli jokin asiakassovellus rekisteröi sisällölle sisällön generoinnin aikana ekspiroitumisajankohdan.

Riippuvuusavaimet

WebCMS-julkaisujärjestelmässä käytetyt riippuvuusavaimet luodaan hajautusfunktion H avulla taulukon 1 mukaisesti verkkotunnus-, tiedosto-, asiakassovellus- ja alkiotunnuksista muodostetuista merkkijonoista. Prefiksejä D, F, A, B, I ja S käytetään, jotta riippuvuusavainten generoinnissa ei syntyisi ylimääräisiä avainten päällekkäisyyksiä.

Riippuvuus	Riippuvuusavain
Domain-riippuvuus	H(D:verkkotunnus)
Tiedostoriippuvuus	H(F:verkkotunnus:tiedosto)
Asiakassovellusriippuvuus	H(A:verkkotunnus:asiakassovellus)
Asiakassovellusjoukkoriippuvuus	H(B:verkkotunnus)
Alkioriippuvuus	H(I:verkkotunnus:asiakassovellus:alkio)
Alkiojoukkoriippuvuus	H(S:verkkotunnus:asiakassovellus)

Taulukko 1: Riippuvuusavainten luonti

Hajautusfunktion avulla riippuvuusavaimet hajautetaan 32-bittiseen numeroavaruuteen, josta ne edelleen palautetaan merkkijonoesitysmuotoon siirtoa varten. Avaimista saadaan hajautuksen ansiosta huomattavasti alkuperäisiä merkkijonoavaimia lyhyempiä. Tämä on tärkeää, jotta avaimia voidaan käsitellä tehokkaasti ja jotta riippuvuuksien tallentamiseen kohdistuvat muistivaatimukset eivät kasva liian suuriksi.

Hajautusfunktion käyttämisen myötä riippuvuusavaimet eivät ole enää yksilöllisiä. Päällekkäisyyksien kohdalla invalidointiprosessi saattaa poistaa välimuistista myös sellaisia tiedostoja, joita ei tarvitsisi poistaa. Tämä ei kuitenkaan ole ongelmallista, koska kätkökonsistenssi saadaan edelleen säilytettyä. Päällekkäisyydet ovat myös verrattain harvinaisia, joten hajautusfunktion käyttäminen välimuistijärjestelmän yleisen tehokkuuden parantamiseksi on perusteltua.

8.5 Riippuvuuksien rekisteröinti

Riippuvuuksien rekisteröinti suoritetaan tietokantarajapinnassa tallentamalla riippuvuuksien kerääjään tietokantakyselyitä vastaavat riippuvuusavaimet.

Riippuvuuksia rekisteröidessä luodaan riippuvuus aina kysyttyyn resurssiin tai resurssijoukkoon sekä kaikkiin resurssihierarkiassa (ks. kuva 3) ylempänä oleviin resursseihin. Riippuvuus on resurssikohtainen, mikäli kysely tehdään yksilöllisellä avaimella. Muussa tapauksessa riippuvuudesta tulee resurssijoukkokohtainen. Resurssikohtainen riippuvuus on erotettu resurssijoukkokohtaisesta riippuvuudesta, jotta turhien invalidointien määrä saadaan minimoitua.

Seuraavassa esittelemme, kuinka riippuvuudet muodostetaan tietokantarajapinnan tarjoamissa palveluissa edellä kuvattujen periaatteiden mukaisesti.

Tiedoston pyytäminen verkkotunnuksen ja tiedostotunnisteen perusteella:

- luodaan tiedostoriippuvuus avaimella H(F:verkkotunnus:tiedosto)
- luodaan verkkotunnusriippuvuus avaimella H(D:verkkotunnus)

Asiakassovellustyypin pyytäminen verkkotunnuksen ja asiakassovellustunnisteen perusteella:

- luodaan asiakassovellusriippuvuus avaimella H(A:verkkotunnus:asiakassovellus)
- luodaan verkkotunnusriippuvuus avaimella H(D:verkkotunnus)

Asiakassovellusinstanssijoukon nimitunnisteiden pyytäminen verkkotunnuksen perusteella:

- luodaan asiakassovellusjoukkoriippuvuus avaimella H(B:verkkotunnus)
- luodaan verkkotunnusriippuvuus avaimella H(D:verkkotunnus)

Tietoalkion pyytäminen verkkotunnuksen sekä asiakassovellus- ja tietoalkiotunnisteiden perusteella:

- luodaan alkioriippuvuus avaimella H(I:verkkotunnus:asiakassovellus:alkio)
- luodaan asiakassovellusriippuvuus avaimella H(A:verkkotunnus:asiakassovellus)
- luodaan verkkotunnusriippuvuus avaimella H(D:verkkotunnus)

Tietoalkion tai -alkioiden pyytäminen verkkotunnuksen sekä asiakassovellustunnisteen perusteella:

- luodaan alkiojoukkoriippuvuus avaimella H(S:verkkotunnus:asiakassovellus)
- luodaan asiakassovellusriippuvuus avaimella H(A:verkkotunnus:asiakassovellus)
- luodaan verkkotunnusriippuvuus avaimella H(D:verkkotunnus)

8.6 Muutosten rekisteröinti

Kaikki tietokantaan kohdistuvat muutokset rekisteröidään tietokantarajanpinnan takana. Muutokset voivat koskea joko resurssin poistamista tai lisäämistä tietokantaan. Resurssin muuttaminen tapahtuu suorittamalla poistamis- ja lisäämisoperaatiot peräkkäin, joten emme käsittele tätä tapausta erikseen.

Resurssin poistaminen ja lisääminen ovat muutoksen rekisteröinnin kannalta identtisiä tapahtumia. Kun resurssi poisteteaan tietokannasta, joudutaan välimuistista poistamaan kaikki sellainen sisältö, jonka generoinnissa kyseistä resurssia on käytetty. Tällöin siis rekisteröidään muutokset sekä kyseiseen resurssiin että resurssia vastaavaan resurssijoukkoon, mikäli sellainen on olemassa. Resurssin lisääminen tietokantaan muuttaa vastaavasti resurssijoukon koostumusta, joten tällöin joudutaan rekisteröimään resurssijoukkomuutos. Resurssin puuttuminen on vaikuttanut myös sellaisiin tietokannasta tehtyihin kyselyihin, jotka on tehty resurssin yksilöllisen avaimen perusteella, joten joudumme rekisteröimään myös lisättävää resurssia koskevan muutoksen.

Alla esittelemme kuinka muutokset rekisteröidään kuvaamallamme tavalla tietokantarajapinnassa.

Verkkotunnuksen ottaminen käyttöön/pois käytöstä verkkotunnuksen perusteella:

- invalidoidaan verkkotunnusriippuvuus avaimella H(D:verkkotunnus)

Tiedoston lisääminen/poistaminen verkkotunnuksen ja tiedostotunnisteen perusteella:

- invalidoidaan tiedostoriippuvuus avaimella H(F:verkkotunnus:tiedosto)

Asiakassovellusinstanssin lisääminen/poistaminen verkkotunnuksen ja asiakassovellustunnisteen perusteella:

- invalidoidaan asiakassovellusriippuvuus avaimella H(A:verkkotunnus:asiakassovellus)
- invalidoidaan asiakassovellusjoukkoriippuvuus avaimella H(B:verkkotunnus)

Verkkotunnus- ja asiakassovelluskohtaisen tietoalkion lisääminen/poistaminen:

- invalidoidaan alkioriippuvuus avaimella H(I:verkkotunnus:asiakassovellus:alkio)
- invalidoidaan alkiojoukkoriippuvuus avaimella H(S:verkkotunnus:asiakassovellus)

9 Ratkaisun soveltuvuus tuotantokäyttöön

9.1 Toimivuus

Välimuistijärjestelmämme osoittautui ainakin konseptuaalisesti erittäin toimivaksi ratkaisuksi. Kaikki vaatimusmäärittelyssä esitetyt sisäiset ja ulkoiset vaatimukset pystyttiin täyttämään. Järjestelmä saatiin myös kokonaisuutena toimimaan odotetulla tavalla. Välimuistijärjestelmästä saatiin toteutettua kohtuullisen kevyt, joten se soveltuu käytettäväksi pienemmissäkin järjestelmissä. Toisaalta useammalla rinnakkain toimivalla välimuistipalvelimella voidaan palvella suurempaakin käyttäjäkuntaa.

9.2 Tehokkuus

Välimuistijärjestelmän todellista tehokkuutta ei päästy vielä mittaamaan, koska WebCMS-julkaisujärjestelmä ei ole vielä tässä vaiheessa otettu tuotantokäyttöön. Mielenkiintoisinta tehokkuuden kannalta olisikin selvittää, kuinka hyvin invalidointimekanismimme osaa valita invalidoitavat resurssit ja kuinka paljon turhia invalidointeja näin ollen ilmenee. Riippuvuudet ovat nykyisessä toteutuksessa jo melko hienojakoisia, joten voimme perustelluin syin olettaa, että turhia invalidointeja tapahtuu vain varsin kohtuullinen määrä.

Implementaation tehokkuutta mittaavissa testeissä osoittautui, että välimuistijärjestelmämme pystyi käsittelemään jo välimuistissa olevia URLeja koskevia HTTP-pyyntöjä jokseenkin samaan tahtiin kuin, millä Apache-HTTP-palvelin käsittelee staattista sisältöä koskevia HTTP-pyyntöjä. Implementaatiokohtainen tehokkuus ei kuitenkaan HTTP-välimuisteissa ole kovinkaan merkittävässä asemassa, joten odotamme mielenkiinnolla sitä, että saamme käyttöömme mittausdataa järjestelmän todellisesta käyttötilanteesta.

9.3 Vikasietoisuus

Välimuistijärjestelmän toteutuksessa pyrittiin huomioimaan mahdollisimman hyvin kaikenlaisten virhetilanteiden käsitteleminen. Varsinkin välimuistipalvelinohjelmisto on monisäikeisyyden ja synkronointimekanismiensa takia kuitenkin siinä määrin monimutkainen, että kaikkia mahdollisia skenaarioita ei pystytty mallintamaan etukäteen. Järjestelmän toiminnan oikeellisuutta ei myöskään ole verifioitu formaalisti, vaan toimivuus on todettu pelkillä testiajoilla. Ennen kuin välimuistijärjestelmää voidaan käyttää tuotantokäytössä näemme tarpeelliseksi, että välimuistijärjestelmää testataan vielä kattavammin.

Välimuistijärjestelmän kyky kestää erityyppisiä palvelunestohyökkäyksiä jäi vielä pitkälti selvittämättä. Toteutuksessa keskeisessä asemassa oli se, että palvelunestohyökkäykset vaikuttaisivat mahdollisimman vähän varsinaisen sisältöpalvelimen toimintaan. Muunmuassa virhesivut tallennetaan välimuistiin, joten toistuvasti virheellisellä URL-osoiteteella suoritettavat HTTP-pyynnöt eivät

pääse sisältöpalvelimelle asti aiheuttamaan ylimääräistä kuormaa. Suojamekanismi kuitenkin pettää, mikäli HTTP-pyynnöt suoritetaan aina erilaisella URL-osoitteella. Tällöin vaarana on lisäksi se, että välimuisti täyttyy lopulta pelkistä virhesivuista. Virhesivujen tallentaminen välimuistiin suojaa sisältöpalvelinta kuitenkin ylimääräiseltä kuormitukselta esimerkiksi virheellisten linkitysten tapauksessa.

9.4 Toteutuksen geneerisyys

Välimuistijärjestelmä on ainakin periaatteessa integroitavissa muidenkin kuin WebCMS-julkaisujärjestelmän yhteyteen. Tässä kohden on kuitenkin huomioitava se tosiasia, että WebCMS-julkaisujärjestelmän tietokantarajapinta on tietokantakyselyiden suhteen jokseenkin rajoittunut. Monimutkaisempien tietokantakyselyiden tapauksessa riippuvuuksien määritteleminen ja kannassa tapahtuvien muutosten seuraaminen eivät ole läheskään yhtä selkeitä toimenpiteitä kuin mitä esimerkkitapauksessamme antaa helposti ymmärtää. Näin ollen välimuistijärjestelmämme integrointia muiden julkaisujärjestelmien yhteyteen ei voida pitää missään tapauksessa triviaalina tehtävänä.

10 Jatkotutkimus ja jatkokehitys

10.1 Dynaamisuuden lisääminen

Nykyisellään välimuistijärjestelmämme tukee dynaamisuutta ainoastaan julkaisujärjestelmän hallintaliittymän suunnalta. Mikäli julkaisujärjestelmään halutaisiin lisätä sellaisia palveluita, joiden sisältöön voisivat vaikuttaa myös WWW-palvelun käyttäjät (esim. keskustelufoorumit, chatit yms.), järjestelmän toiminnallisuutta jouduttaisiin laajentamaan. Tällaisen toiminnallisuuden toteuttamiseksi vaadittaisiin lisätutkimusta muunmuassa session hallintaan liittyen. Riippuvuus- ja muutosavaimiin perustuvilla invalidoinneilla voidaan periaatteessa hallita myös yleisön tuottamaa dynaamista sisältöä.

10.2 Suojatun sisällön tukeminen

Myös suojatun sisällön tukemisen osalta välimuistijärjestelmässä on tilaa jatkotutkimukselle ja myöhemmin jatkokehitykselle. Tätä varten tulee määritellä menetelmät käyttäjän tunnistamiseen sekä suojatun sisällön turvalliseen tallentamiseen välimuistiin. Lisäksi mahdollisena tutkimuksen kohteena voidaan pitää personoitujen WWW-sivujen tallennusta välimuistiin. Tästä aihepiiristä löytyy jo kattavasti kirjallisuutta, mutta konseptien siirtäminen omaan välimuistijärjestelmäämme ei tule olemaan kokonaan erilaisten alkuperäisten lähestymistapojen takia vaivatonta.

10.3 Tehokkuuden parantaminen

Välimuistijärjestelmää voidaan tehostaa entisestään valitsemalla tilan vapauttamiseen joku LRU:ta sofistukoituneempi algoritmi. Valintaa varten tulee analysoida välimuistijärjestelmän todellisia käyttöskenaarioita. Simuloidun testidatan käyttö on tehokkuustesteissä välimuistijärjestelmien kohdalla käytännössä hyödytöntä, joten algoritmeja kannattaa testailla tarkemmin vasta, kun järjestelmä on ollut tuotantokäytössä.

Lisää tehokkuutta voidaan saada aikaiseksi myös pitämällä useimmiten käytettyjen resurssejen sisältöä massamuistin sijaan välimuistipalvelimen keskusmuistissa. Nykyisessä ratkaisussa luotetaan siihen, että levyjärjestelmän välimuistit toimivat tehokkaasti. Tämä ei kuitenkaan ole välimuistipalvelimen tehokkuuden kannalta välttämättä kaikkien optimaalisin ratkaisu. Toisaalta levyjärjestelmän käyttöön liittyvät optimoinnit ovat konekohtaisia, joten parhaan kompromissiratkaisun löytäminen voi olla vaikeata.

Sisältöpalvelimelle kohdistuvan kuormaa voitaisiin pienentää parhaimmillaan suhteessa välimuistipalvelinten määrään, mikäli välimuistipalvelimet kommunikoisivat keskenään ja levittäisivät sisältöpalvelimelta saamaansa sisältöä toisilleen push-tekniikoita hyväksi käyttäen. Tällainen ratkaisu pienentäisi sekä sisältöpalvelimen kuormaa että välimuistipalvelinten ja sisältöpalvelimen välisen tietoliikenneyhteyden kuormaa. Haittapuolena olisi kuitenkin välimuistipalvelinten toteutuksen tuleminen monimutkaisemmaksi sekä sisällön levittämisestä välimuistipalvelimelle aiheutuva ylimääräinen kuorma.

11 Yhteenveto

Olemme esitelleet tässä raportissa reverse proxy -periaatteella toimivan välimuistijärjestelmän, jonka toiminta perustuu välimuistissa olevan sisällön eksplisiittiseen invalidointiin. Välimuistijärjestelmämme pystyy tarjoamaan dynaamista sisältöä suoraan välimuistista samaan tapaan kuin staattista sisältöä. Täten järjestelmä pystyy hyödyntämään täysin myös HTTP/1.1-protokollan tarjoamia validointimekanismeja pienentäen merkittävästi sekä tietoliikenneverkolle että sisältöpalvelimelle aiheutuvia kuormia.

Välimuistijärjestelmä jakautuu erilliseen sisältöpalvelimeen ja yhteen tai useampaan välimuistipalvelimeen. Sisältöpalvelinten yhteyteen integroidaan sisältöriippuvuuksien kerääminen sekä sisältömuutosten rekisteröiminen. Riippuvuuksia ja muutoksia kuvataan riippuvuus- ja muutosavaimilla, joita vertailemalla voidaan selvittää mitkä resurssit on poistettava välimuistista. Esittelimme tässä raportissa myös sen, kuinka riippuvuudet on mallinnettu WebCMS-julkaisujärjestelmän tapauksessa.

Kerroimme myös kuinka muunsimme Jetty-HTTP-palvelimen avainperusteista etäinvalidointia tukevaksi välimuistipalvelimeksi sekä tutustuimme lyhyesti itse välimuistipalvelimen tekniseen toteutukseen.

Projektin lopputuotosten perusteella päättelimme, että esittelemäämme välimuistitekniikkaa voidaan tietyin edellytyksin käyttää hyväksi myös tuotanto-

käytössä. Välimuistijärjestelmämme toiminnallisuus on kuitenkin vielä tässä vaiheessa kohtalaisen rajoittunutta, joten mikäli järjestelmän kautta halutaan esimerkiksi tarjota yleisön suuntaan interaktiivisia palveluita tai suojattuja sivustoja, jatkotutkimusta ja -kehitystä joudutaan vielä suorittamaan.

Viitteet

[1] Akamai Technologies, Inc. EdgePlatform. [viitattu 10.5.2004] URL: http://www.akamai.com/en/html/technology/edgeplatform.html.

[2] Bakalova R., Chow A., Fricano C., Jain P., Kodali N., Poirier D., Sankaran S., Shupp D. WebSphere Dynamic Cache: Improving J2EE application performance. IBM Systems Journal, Vol 43. NO 2, 2004. [viitattu 10.5.2004]. URL: http://research.ibm.com/journal/sj/432/bakalova.pdf.

[3] Barish Greg, Obraczka Katia. World Wide Web Caching: Trends and Techniques. 2000. [viitattu 8.5.2004]. URL: http://inrg.cse.ucsc.edu/katia-pubs/cache-survey.pdf.

[4] Cooper I. RFC 3040: Internet Web Replication and Caching Taxonomy. Tammikuu 2001. [viitattu 9.5.2004]. URL: http://www.ietf.org/rfc/rfc3040.txt.

[5] Candan K. Selcuk, Li Wen-Syan, Luo Qiong, Hsiung Wang-Pin, Agrawal Divyakant. Enabling Dynamic Content Caching for Database-Driven Web Sites. 2001. [viitattu 10.5.2004] URL: http://www.cs.ust.hk/~luo/papers/invalidation_sigmod01.pdf.

[6] Dilley John, Arlitt Martin, Perret Stephane. Enhancement and Validation of Squid's Cache Replacement Policy. 1999. [viitattu 8.5.2004] URL: http://www.hpl.hp.com/techreports/1999/HPL-1999-69.ps.

[7] Fielding Roy T, Gettys James, Mogul Jeffrey C., Nielsen Henrik Frystyk, Masinter Larry, Leach Paul J. & Berners-Lee Tim. RFC 2616: Hypertext Transfer Protocol – HTTP/1.1. Kesäkuu 1999. [viitattu 2.4.2004]. URL: http://www.ietf.org/rfc/rfc2616.txt.

[8] Jacobs Larry, Ling Gary, Liu Xiang. Technical Specification: ESI Invalidation Protocol 1.0. 2001. [viitattu 8.5.2004]. URL: http://www.esi.org/invalidation_protocol_1-0.html.

[9] Microsoft Corporation, Microsoft Internet Security and Acceleration Server. [viitattu 8.5.2004] URL: http://www.microsoft.com/isaserver/.

[10] Mort Bay Consulting. Jetty:// WebServer & Servlet Container. [viitattu 8.5.2004]. URL: http://www.mortbay.org/jetty/index.html.

[11] Oracle Corporation, Akamai Technologies Inc. Edge Side Includes. [viitattu 8.5.2004]. URL: http://www.esi.org/.

[12] Oracle Corporation. OracleAS Web Cache 10g. [viitattu 8.5.2004]. URL: http://otn.oracle.com/products/ias/web_cache/index.html.

[13] Squid Web Proxy Cache. [viitattu 8.5.2004]. http://www.squid-cache.org/.