STADIA
HELSINGIN AMMATTIKORKEAKOULU

TEKNIIKAN JA LIIKENTEEN TOIMIALA

Sähkö- ja tietoliikennetekniikka

Ohjelmistotekniikka

HAJAUTETUT JÄRJESTELMÄT

KONSISTENTTISUUS JA REPLIKOINTI

TIETOKESKEISET EHEYSMALLIT JA HAJAUTUSPROTOKOLLAT

Työn tekijät **Erkki Jyrkkänen**
Åke Tyvi

Työ hyväksytty: 2. 3. 2005

Copyright © Erkki Jyrkkänen ja Åke Tyvi

Tämä julkaisu ei ole ilmaisjulkaisu (ks. esipuhe), julkiskaupallinen tai kaupalliseen levitykseen tarkoitettu, eikä sitä saa kaupallisesti hyödyntää esimerkiksi myymällä, vaihtamalla tai muulla tavoin välittämällä ilman julkaisijalta saatavaa kirjallista suostumusta.

Tämän julkaisun tekstin ja kuvien jäljentäminen on sallittua samassa laajuudessa mitä alkuperäislähteiden tekijänoikeussuoja asiasta kertoo. Muita julkaisun kuvia ei saa käyttää ilman julkaisuviitettä. Viitteet toisiin lähteisiin on eritelty julkaisun takaa löytyvässä viiteluettelossa.

Julkaisu perustuu pitkälti Andrew S. Tanenbaumin ja Maarten von Steenin vuonna 2002 ilmestyneeseen kirjaan Distributed Systems and Paradigms ja on sen luvun kuusi kahden kappaleen vapaa suomennos. Kirjan luvussa esitettyjä yleisiä asioita on selvennetty eri lähteistä saadun tiedon ja julkaisua varten tehtyjen lisäkuvien avulla.

Tällä julkaisulla ei ole tieteellistä arvoa ja sen käännösarvo on vähäinen.

Julkaisun kustantaja ei ota vastuuta julkaisussa mahdollisesti esiintyvistä virheistä. Samaa asiaa koskevien myöhempien painosten tai ilmestyvien virheen korjaussivujen takia havaitut virheet ja parannusehdotukset tulisi ilmoittaa sivun lopussa annettuun sähköpostiosoitteeseen.

Kyseessä on elektroninen julkaisu.

1. painos maaliskuu 2005

Asia: Erkki Jyrkkänen ja Åke Tyvi
Taitto: Erkki Jyrkkänen
Kansi: -.
Kannen kuva: -.

ISBN 952-5494-01-2

Omakustanne
Helsinki 2005

Kustantajan yhteystiedot:
Åke Tyvi
Kuusamatie 15
14700 HAUHO

Hinta……... : 2,16EUR (sis.8% alv)
Pankki…….: Nordea
Tiliyhteys… : 158430-22307
Viite……….: 9 52549 40125 tai viesti-kenttään ostajan tiedot.

Julkaisussa havaittujen virheiden ilmoittaminen sähköpostitse osoitteeseen r0100034@curatores.info tai kirjallisesti yllä mainittuun osoitteeseen.

Tämä moniste on Helsingin ammattikorkeakoulun hajautetut järjestelmät –kurssin seminaarityö.

SISÄLLYSLUETTELO

ESIPUHE 3
JOHDANTO 4
1. TIETOKESKEISET EHEYSMALLIT (data-centric consistency models) 6
1.1 TIUKKA EHEYS 7
1.2 LINAARISOITAVUUS JA JAKSOLLINEN EHEYS 8
1.2.1 LINEAARISOINTI 9
1.2.2 ESIMERKKI JAKSOLLISEN EHEYDEN MALLISTA 10
2 HAJAUTUSPROTOKOLLAT (distribution protocols) 13
2.1 MONENNETUN TIEDON SIJAINTIPAIKKA (replica placement) 13
2.1.1 PYSYVÄISLUONTOISET KOPIOT (permanent replicas) 13
2.1.2 PALVELINALOITTEISET KOPIOT (server-initiated replicas) 15
2.1.3 ASIAKAS-/TYÖASEMA-ALOITTEISET KOPIOT (client-server replicas) 20
2.1.4 PÄIVITYSTEN ETENEMINEN (update propagation) 22
2.1.4.1 MITÄTÖIMISPROTOKOLLAN TOIMINNAN TARKASTELU 23
2.1.4.4 TÄSMÄLÄHETYS (unicast) VAI JOUKKOLÄHETYS (multicast)? 28
2.1.5 EPIDEEMISET PROTOKOLLAT (epidemic protocols) 29
2.1.5.1 PÄIVITYSTEN ETENEMISMALLIT (update propagation models) 30
2.1.5.2 TIEDON POISTAMINEN 31
3 YHTEENVETO 33
KIRJALLISUUS- JA VIITELUETTELO 34

ESIPUHE

Tämä julkaisu on tarkoitettu Helsingin ammattikorkeakoulun hajautetut tietojärjestelmät -kurssin tukimateriaaliksi. Julkaisu perustuu ja noudattaa Andrew S. Tanenbaumin ja Maarten von Steenin vuonna 2002 ilmestynyttä kirjaa Distributed Systems and Paradigms [1]. Tukimateriaalina olemme käyttäneet Ilkka Haikalan ja Hannu-Matti Järvisen Käyttöjärjestelmät-kirjasta [2] soveltuvia osia, sekä muita erikseen lähdeluettelossa mainittuja lähteitä. Julkaisussa ei esiinny kirjoittajien omia ehdotuksia tai ratkaisuja tiedon yhdenmukaisuuden eli ristiriidattomuuden takaamiseksi tai tiedon toisintamiseksi (so. replikointi), vaan tämä dokumentti käsittelee ja esittelee olemassa olevia ratkaisuja em. kirjan pohjalta.

Viitatessa tähän julkaisuun tulee viittaajan aina ensin mainita alkuperäisteos [1] ja sen alla maininta tästä seminaarityöstä: Tämä materiaali pohjautuu eri kirjoittajien julkaisuihin, jotka on eritelty täydellisenä Distributed Systems and Paradigms -kirja lähde- ja kirjallisuusluettelossa. Julkaisussa esiintyvien asioiden käsittelyjärjestys noudattaa kirjan luvun kuusi järjestystä.

Vaikka tämä elektroninen julkaisu on tietoverkosta vapaasti saatavissa, niin julkaisun tekemiseen kuluneen käännös- ja prosessointityön takia on sillä nimellinen ’vapaaehtoiseen maksunsuorittamiseen’ perustuva hinta: Aivan kuten te muutkin saatte palkan työstä, jota olette työnantajanne eteen tehneet ja minkä vuoksi todennäköisesti tämän tiedon tietoverkosta käsienne ulottuville olette etsineet, niin myös me olemme palkkion työstämme ansainneet. Vain kommunismissa kuvitellaan asioiden olevan ilmaista ja kaikille yhteistä.
Julkaisua kaupallisesti käyttävän, tähän suomennokseen tai sen avulla viitteitä alkuperäisteoksiin tekevän henkilön tulee aina maksaa korvaus tämän julkaisun käyttämisestä. Maksusuoritusta koskevat tiedot on eritelty kansilehden jälkeisellä sivulla. Hinta on nimellinen 2,16EUR ja sen maksamisesta jää hyvä mieli sekä julkaisun tekijälle ja sen uudelle omistajalle.

Tätä julkaisua voi käyttää veloituksetta Atk-instituutti, Helsingin ammattikorkeakulu ja Amiedu, sekä jokainen allekirjoittaneita opettanut opettaja.

JOHDANTO

Replikointiin on kaksi pääsyytä: luotettavuuden ja suorituskyvyn parantaminen. Replikointi ei ainoastaan paranna saatavuutta, vaan auttaa tasapainottamaan kuormaa komponenttien välillä. Maantieteellisesti laajalle alueelle hajautetussa järjestelmässä on järkevää sijoittaa palvelu mahdollisimman lähelle käyttäjiä palvelun laadun takaamiseksi. Replikoinnilla voidaan myös parantaa järjestelmän luotettavuutta. Jos data on kadonnut tai vioittunut esimerkiksi systeemin vioittumisen aikana, voidaan palvelu ohjata käyttämään replikaa.

Tietokantojen replikointi tarkoittaa tietokantojen automaattista päivitystä paikasta toiseen siten, että vain muuttuneet osat kopioidaan. Jokainen tietokanta voi sijaita usealla eri palvelimella tai yhdessä palvelimessa ja useassa työasemassa. Kun johonkin kantaan tehdään muutoksia, päivittyvät muutokset automaattisesti kaikkiin muihin kantoihin. Replikointi takaa, että kaikkien kantojen tiedot pysyvät ajan tasalla, joten varmuuskopiointi tapahtuu lähes automaattisesti. Jos jokainen käyttäjä voi tehdä muutoksensa aina itseään lähimpään palvelimeen, pysyy myös palvelun nopeus hyvänä.

Toinen esimerkki on World Wide Web, joka on fyysisesti hajautettu lukemattomille palvelimille jotka käsittelevät valtavia määriä palvelupyyntöjä. Jos yhdelle palvelimelle tulee liikaa palvelupyyntöjä, johtaa se suorituskyvyn heikkenemiseen. Tilannetta voidaan parantaa replikoinnilla. Käytännössä tämä tarkoittaa palvelun jakamista toisille palvelimille siten, että eheys säilytetään, kuten edellä mainittiin.

Caching on replikoinnin erikoismuoto. Käytännön esimerkkinä voidaan mainita web-selain. Liikkuessasi web-sivuilla ne tallentuvat tietokoneesi välimuistiin, joten voit liikkua selaimen edellinen- ja seuraava-painonapeilla käymilläsi sivuilla. Sivuissa mahdollisesti tapahtuneet muutokset eivät näy, ennen kuin olet päivittänyt sivun. Erona replikointiin on se, että cachingissä muutoksien päivittymistä valvoo lähinnä resurssin käyttäjä, kun taas replikoinnissa resurssin omistaja.

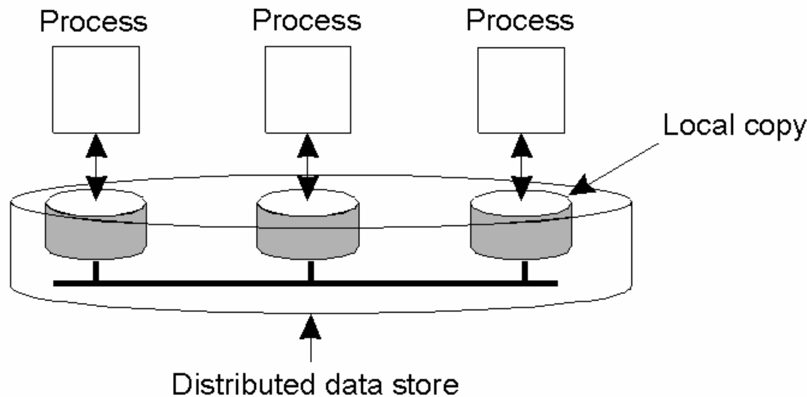
Kummassakin tapauksessa, replikoinnissa ja cachingissa ongelmana on eheyden säilyttäminen. Koska käyttävästä resurssista on olemassa useita kopioita, yhden muuttaminen johtaa muiden muuttamiseen. Kuinka paljon viivettä suvaitaan, riippuu pitkälti

siitä, mitä resurssia käytetään. Webin käyttäjä voi sietää sen, että selain palauttaa välimuistiin talletetun sivun, jota ei ole päivitetty muutamaan minuuttiin. Tilanne on aivan toinen esimerkiksi sähköisessä osakekaupassa. Päivityksen on näyttävä välittömästi kaikissa kopioissa. Lisäksi kaikkien päivityksien on tapahduttava samassa järjestyksessä kuin ne on tehty. Tällaiset tilanteet vaatisivat globaalia synkronointimekanismia ja ns. *tiukkaa eheyttä*. Tällainen mekanismi on kuitenkin mahdoton luoda skaalattavalla tavalla. Replikoinnista saatava taloudellinen hyöty voidaan menettää tiukan eheyden tavoittelemiseen, joten asia on ratkaistava jotenkin toisin. Seuraavassa kappaleessa esitellään erilaisia eheysmalleja [1][2].

1. TIETOKESKEISET EHEYSMALLIT (data-centric consistency models)

- Erkki Jyrkkänen -

Data, johon luku- ja kirjoitusoperaatiot kohdistuvat, voi hajautetuissa järjestelmissä sijaita monessa eri paikassa, kuten jaetussa muistissa, tietokannoissa tai tiedostoissa.



Kuva 1: Fyysisesti hajautettu tietovarasto [1]

On kuitenkin yksinkertaisempaa käyttää näistä paikoista yhteistä termiä, datamuisti (*data store)*, tai vieläkin yksinkertaisempaa suomenkielistä muotoa: tietovarasto. Tietovarasto soveltuu terminä sen vuoksi paremmin käytettäväksi, koska lähdeteoksen eheysmallien määritykset käyttävät termiä: *data item,* jonka suomenkielinen käännös on *tietoalkio.* Yksinkertaistettuna tietovarasto sisältää tietoalkioita, joihin kohdistuu luku- ja kirjoitusoperaatioita. Tilannetta voitaisiin verrata normaalin varastonhallintaan. Varasto on paikka, joka sisältää tavaroita, ts. alkioita. Varaston kirjanpitoon tapahtuu jatkuvasti muutoksia, koska sieltä viedään ja sinne tuodaan jatkuvasti tavaroita. Kuinka pian muutoksen jälkeen tapahtumat kirjautuvat kirjanpitoon riippuu siitä, millaista sopimusta tai mallia varaston työntekijät on velvoitettu noudattamaan. Jos tätä yhteisesti sovittua sopimusta noudatetaan, varaston kirjanpito on jatkuvasti ajan tasalla ja tiedetään varaston tila mahdollisimman lähellä nykyhetkeä. Yleisesti eheysmalleja voitaisiin kuvata tällä yhteisesti sovitulla sopimuksella. Seuraavassa kappaleessa käsitellään näitä malleja.

1.1 TIUKKA EHEYS

Kaikkein vaativin eheysvaatimus on, kuten nimikin jo kuvaa, on tiukka eheys (*strict consistency*). Se on määritelty seuraavalla tavalla:

Jokainen lukuoperaatio tietoalkioon (*data item*) X, palauttaa arvon viimeisimmästä kirjoitusoperaatiosta tietoalkioon X.

Kiteytettynä, kaikki kirjoitustapahtumat muistiin ovat välittömästi näkyvissä kaikille prosesseille ja absoluuttinen globaali kirjoitusoperaatioiden aikajärjestys on säilytetty. Jos arvo on muuttunut, kaikki operaatiot tapahtuvat välittömästi uuteen arvoon, riippumatta siitä kuinka pian muutoksen jälkeen lukuoperaatio tapahtuu tai missä muutos on tapahtunut.

P1: W(x)a
P2: R(x)a

Kuva 2 [1]

P1: W(x)a
P2: R(x)NIL R(x)a

Kuva 3 [1]

Kuvassa 2 tapahtuvat prosessit P1 ja P2 noudattavat tiukan eheyden mallia. Aika akseli on kuvattuna horisontaalisesti, ajan kasvaessa vasemmalta oikealle. W(x)a kuvaa kirjoitusoperaatiota muuttujaan x, antaen sille arvon a. Tämän jälkeen lukuoperaatio R(x)a lukee muuttujan x, viimeisimmän kirjoitetun arvon. Vastaavasti kuva 3 ei noudata tiukan eheyden mallia, koska lukuoperaation pitäisi saada arvo a, jo ensimmäisellä lukukerralla.

Hajautetuissa järjestelmissä asiat ovat kuitenkin monimutkaisempia. Oletetaan, että x on tieto, joka on varastoitu ainoastaan koneeseen B. Kuvitellaan, että prosessi koneessa A, lukee tiedon x ajassa T1. Prosessi A saa siis viimeisimmän kirjoitustapahtuman koneesta B. Hiukan myöhemmin prosessi B, ajassa T2, tekee kirjoitusoperaation tietoon x. Noudattaen tiukkaa eheyttä, lukuoperaation pitäisi aina palauttaa viimeisin arvo, huolimatta siitä missä koneet sijaitsevat tai kuinka pieni on prosessien aikaväli T1 ja T2. Jos aikaero T2-T1 on 1 nanosekunti ja koneet 3 metrin päässä toisistaan, suoriutuakseen viimeisimmän tiedon lukemisesta ennen uutta kirjoitustapahtumaa, signaalin pitäisi kulkea kymmenkertaisella valon nopeudella, mikä rikkoo fysiikan lakeja.

Jotta määritelmän sanalla ”viimeisin”, olisi jotain merkitystä, pitäisi olla olemassa *absoluuttinen globaali aika,* jonka mukaan tapahtumat voidaan asettaa aikajärjestykseen. Jos tapahtumien välit ovat nanosekuntien mittaisia, törmätään hajautetuissa järjestelmissä väistämättä synkronointi ongelmaan. Yksiprosessori järjestelmissä tämä on toteutettavissa, mutta hajautetuissa järjestelmissä sen toteuttamista pidetään mahdottomana. Sitä pidetään ns. *ideaalina mallina.*

1.2 LINAARISOITAVUUS JA JAKSOLLINEN EHEYS

Koska tiukka eheys on mahdoton toteuttaa hajautetuissa järjestelmissä, on pakko käyttää muita tapoja. Jaksollinen eheys (*sequential consistency*) on vaatimusmääritykseltään hiukan heikompi malli kuin tiukka eheys. Sen määritteli Leslie Lamport (1979), tutkiessaan jaetun muistin multiprosessori järjestelmiä. Jaksollisen eheyden määritelmä on seuraavanlainen:

Kaikkien prosessien tietovarastoon kohdistuvat operaatiot (luku ja kirjoitus) suoritetaan jossain jaksollisessa järjestyksessä ja jokaisen yksittäisen prosessin operaatio tapahtuu tässä järjestyksessä, sen oman ohjelman määrittelemällä tavalla.

Käytännössä tämä tarkoittaa, että unohdamme *absoluuttisen globaalin ajan* käsityksen. Tapahtumien järjestys pitää ratkaista ohjelmallisesti erilaisilla synkronointi operaatioilla, kuten semaforeilla. Semaforeilla muuttuja lukitaan siten, että lukuoperaatiota ei voida suorittaa ennen kuin kirjoitusoperaatio on suoritettu loppuun. Tämä malli on käytännön toteutuksissa kaikkein suosituin.

P1:	W(x)a			
P2:		W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)b	R(x)a

Kuva 4 [1]

P1:	W(x)a			
P2:		W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

Kuva 5 [1]

Kuvassa 4 neljä prosessia operoi samaa tietoalkiota x. Ensin P1 suorittaa kirjoitusoperaation tietoalkioon x. Myöhemmin P2 suorittaa myös kirjoitusoperaation, muuttaen tietoalkioon arvon b. Kummatkin prosessit P3 ja P4, lukevat ensin arvon b ja sen jälkeen arvon a. Näyttää siltä, että prosessi P2 olisi tapahtunut ennen prosessia P1, mutta kaikkien prosessien operaatiot noudattavat jaksollisen eheyden määritelmää

Kuvan 5 prosessit eivät noudata jaksollisen eheyden määritelmää. Vaikka prosessin P4 lukuoperaation viimeiseksi arvoksi saadaankin b, prosessien P3 ja P4 operaatiot eivät noudata samaa jaksollista järjestelmää. Huomaa, ettei määritelmässä puhuta enää ajasta mitään. Prosessi näkee kaikkien prosessien kirjoitusoperaatiot, mutta vain omat lukuoperaationsa.

1.2.1 LINEAARISOINTI

Jaksollista eheyttä tiukempi malli on lineaarisointi. Lineaarisointi eroaa jaksollisen eheyden mallista siten, että operaatiot varustetaan aikaleimalla (*time stamp*). Lyhyesti:

Lineaarisointi = jaksollinen eheys + operaatiot varustettuina aikaleimoilla

Lineaarisoinnin määritelmä (*Herlihy ja Wing, 1991*) on sama kuin jaksollisen eheyden mallissa, mutta siihen on tullut seuraava lisäys:

Jos $^{ts}OP1(x) < {}^{ts}OP2(y)$ **, niin OP1(x) suoritetaan ennen OP2(y) suorittamista.**

Toisin sanoen operaation OP1(x) aikaleima on pienempi, joten se suoritetaan ennen operaation OP2(y) suorittamista. Huomioitavaa on, että lineaarisen eheyden mallia noudattava tietovarasto noudattaa myös jaksollisen eheyden mallia.

1.2.2 ESIMERKKI JAKSOLLISEN EHEYDEN MALLISTA

Havainnollistaaksemme jaksollista mallia paremmin, käytämme seuraavaa esimerkkiä.

Process P1	Process P2	Process P3
x = 1; print (y, z);	y = 1; print (x, z);	z = 1; print (x, y);

Kuva 6: Kolme samanaikaisesti suoritettavaa prosessia [1]

Kuvassa 6 on kolme samanaikaisesti suoritettavaa prosessia P1, P2 ja P3. Tietoalkioina on kolme kokonaislukumuuttujaa x, y ja z, jotka on sijoitettu jaksollisen eheyden mallia noudattavaan, hajautettuun tietovarastoon. Jokaisen muuttujan arvo on aluksi nolla. Kirjoitusoperaation jälkeen seuraa lukuoperaatio, jossa luetaan operaation kaksi argumenttia. Esimerkissä lukuoperaatiota kuvaa print-käsky. Kaikkien lauseiden (*statements*) oletetaan olevan jakamattomia. Koska prosessit ovat samanaikaisia, voi ohjelman suoritusjaksot limittyä usealla eri tavalla. Kuuden itsenäisen lauseen mahdollisia suoritusjaksoja on kaikkiaan 720, eli 6! kappaletta. Useat jaksoista kuitenkin rikkovat jaksollisen eheyden mallia.

Oletetaan, että 120 (5!) jaksoa alkaa kirjoitusoperaatiolla x = 1. Puolet jaksoista suorittaa operaation print(x,z), ennen kuin y = 1. Puolet jaksoista on myös suorittanut operaation print(x,y), ennen kuin z = 1. Kaikki edellä mainitut jaksot rikkovat ohjelman suoritusjärjestystä. Operaatioilla x = 1, y = 1 ja z = 1 alkavista jaksoista, on jokaisella 30 luvallista aloitustapaa, eli jaksoista ¼ on kelvollisia. Siten suoritusjärjestystä noudattaa yhteensä 90 jaksoa. Kuvassa 7 esitetään näistä neljä.

Kuvan 7(a) prosessit suoritetaan järjestyksessä P1, P2 ja P3. Muissa kolmessa esimerkissä esitetään erilaisia, mutta yhtä lailla kelvollisia operaatioita, jotka limittyvät ajan suhteen. Jokainen kolmesta prosessista tulostaa kaksi muuttujaa. Koska jokaisella muuttujalla voi olla alkuarvo 0 tai asettu arvo 1, jokainen prosessi tuottaa 2-bittisen merkkijonon. ”Prints” sanan perässä olevat numerot kuvaavat päätelaitteeseen tulostuvaa merkkijonoa. Merkkijono on 6-bittinen, jossa jokainen kaksi bittiä kuvaa tulostusta siinä järjestyksessä, kuin jokaisessa jaksossa (a), (b) ja (c) operaatiot on esitetty.

x = 1;	x = 1;	y = 1;	y = 1;
print ((y, z);	y = 1;	z = 1;	x = 1;
y = 1;	print (x,z);	print (x, y);	z = 1;
print (x, z);	print(y, z);	print (x, z);	print (x, z);
z = 1;	z = 1;	x = 1;	print (y, z);
print (x, y);	print (x, y);	print (y, z);	print (x, y);
Prints: 001011	Prints: 101011	Prints: 010111	Prints: 111111
Signature: 001011	Signature: 101011	Signature: 110101	Signature: 111111
(a)	(b)	(c)	(d)

Kuva 7: Neljä kelvollista ohjelman suoritusjaksoa, jotka on kuvattu kuvassa 6. Aikajärjestys (suoritusjärjestys) kuvataan pystyakselilla. [1]

Jos puramme kuvan 7(a) Prints-tulostetta, saamme merkkijonon 001011. Kaksi ensimmäistä bittiä 00, saadaan operaatiosta print(y,z). Koska ainoastaan muuttujaan x on sijoitettu arvo 1, ja muuttujissa y ja z on alkuarvot 0, tulostuu operaatiosta print(y,z) merkkijono 00. Vastaavasti operaatiosta print(x,z) tulostuu merkkijono 10, koska ennen sitä on suoritettu operaatio x = 1 ja muuttujan z arvoa ei ole vielä muutettu alkuarvostaan 0. Käytyämme läpi vastaavalla tavalla muut jaksot (a), (b) ja (c) aikajärjestyksessä, saamme merkkijonot 001011, 101011 ja 111111.

Jos noudatamme jaksollisen eheyden mallia, joka tässä tapauksessa noudattaa ohjelmajärjestystä siten, että kirjoitustapahtuma on tapahduttava ennen lukutapahtumaa ja lukutapahtuma noudattaa kuvassa 6 esitettyä prosessijärjestystä P1, P2 ja P3, saamme jaksossa (c) erilaisen merkkijonon. Merkkijono on kuvattu kuvan 7 kohdassa ”Signature”. Jos lähdemme purkamaan jakson 7(c) Signature-merkkijonoa 110101, ensimmäiset kaksi bittiä 11, kuvaavat prosessin P1 lukuoperaatiota print(y,z). Tulostuksessa huomioidaan vain ennen lukuoperaatiota tapahtuneet kirjoitusoperaatiot siten, että noudatetaan ohjelmajärjestystä P1, P2 ja P3. Koska operaatiota print(y,z) ennen on jakson aikana muuttujien arvot muutettu arvoihin y = 1 ja z = 1, on merkkijonon kaksi ensimmäistä bittiä 11. Vastaavasti prosessin P2 lukuoperaation muuttujat x ja z ovat saaneet arvot 0 ja 1, kuten myös operaation P3 muuttujat.

6-bittisen merkkijonon esittämiseen on 64 eri tapaa. Kaikki näistä ei kuitenkaan ole sallittuja, kuten aikaisemmin todettiin. Esimerkkinä kielletystä merkkijonosta on

001001. Jos noudatamme edelleen jaksollisen eheyden mallia, sen kaksi ensimmäistä bittiä 00 tarkoittaa, että prosessien P2 ja P3 muuttujat olivat nollia, kun prosessin P1 lukuoperaatio suoritettiin. Tapahtuma on mahdollinen ainoastaan silloin, kun P1 on tehnyt kirjoitusoperaationsa x = 1 ja lukuoperaatioonsa print(y,z), ennen kuin P2 ja P3 ovat käynnistyneet. Seuraavat kaksi bittiä 10 tarkoittavat sitä, että prosessin P2 täytyy käynnistyä sen jälkeen kun P1 on käynnistynyt, mutta ennen kuin P3 käynnistyy. Viimeiset bitit 01 tarkoittaa, että prosessin P3 pitää olla suoritettuna ennen kuin P1 käynnistyy, mutta olemme jo todenneet, että aloittavan prosessin täytyy olla P1. Tämän perusteella merkkijono 001001, on todettava kelvottomaksi.

Kaiken kaikkiaan lähdeteoksemme kuvaa jaksollista eheyttä ohjelmoija ystävälliseksi malliksi ja sen vuoksi myös käytännön toteutuksissa suosituimmaksi. Kuitenkin siinäkin on vakavia ongelmia. Lipton ja Sanberg (1988) todistivat, että jos lukuaika on *r*, kirjoitusaika on *w* ja minimiaika paketin kuljettamiseen solmun päästä päähän on *t,* voidaan kirjoittaa: $r + w \geq t$. Toisin sanoen, jaksollista eheyttä noudattavassa mallissa, protokollan muuttaminen lukunopeuden parantamiseksi, johtaa kirjoitusnopeuden hidastumiseen. Väitteen paikkaansa pitävyyttä ei kirjassa todisteta, joten emme käsittele sitä tämän enempää. Tämän vuoksi tutkijat ovat kehittäneet muita, heikompia eheysmalleja. Niitä emme myöskään tässä käsittele. Sivulla olevassa kuvassa [1] on yhteenveto kyseisistä malleista. Kaikki esittelemämme mallit ovat niin sanottuja synkronoimattomien operaatioiden malleja. Kaikista malleista voi lukea tarkemmin käyttämästämme lähdeteoksesta: Distributed Systems: Principles and Paradigms.

Summary of Consistency Models

Consistency	Description
Strict	Absolute time ordering of all shared accesses matters.
Linearizability	All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp
Sequential	All processes see all shared accesses in the same order. Accesses are not ordered in time
Causal	All processes see causally-related shared accesses in the same order.
FIFO	All processes see writes from each other in the order they were used. Writes from different processes may not always be seen in that order

(a)

Consistency	Description
Weak	Shared data can be counted on to be consistent only after a synchronization is done
Release	Shared data are made consistent when a critical region is exited
Entry	Shared data pertaining to a critical region are made consistent when a critical region is entered.

(b)

a) Consistency models not using synchronization operations.
b) Models with synchronization operations.

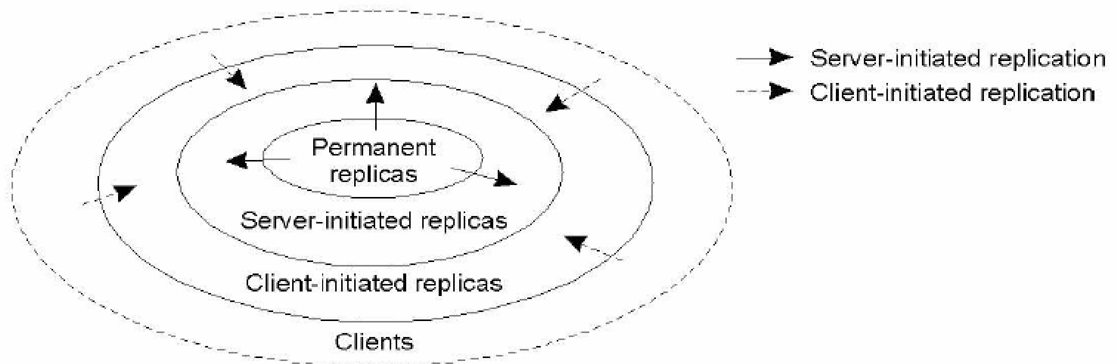
2 HAJAUTUSPROTOKOLLAT (distribution protocols)

- Åke Tyvi –

Tarkastelemme seuraavaksi erilaisista tiedon kopiointitapoja ilman tiedon eheysvaadetta, johon perehdymme myöhemmin tässä materiaalissa.

2.1 MONENNETUN TIEDON SIJAINTIPAIKKA (replica placement)

Hajautettujen tietovarastojen osalta suurin suunnittelupäätös liittyy tietovarastojen sijaintiin. Koska sijoitetaan tietoa, minne ja kenen toimesta? Distributed Systems -kirja viittaa kolmeen erilaiseen loogisesti järjestettävissä olevaan kopiotyyppiin, jotka on esitetty nimettyinä alla olevassa kuvassa.



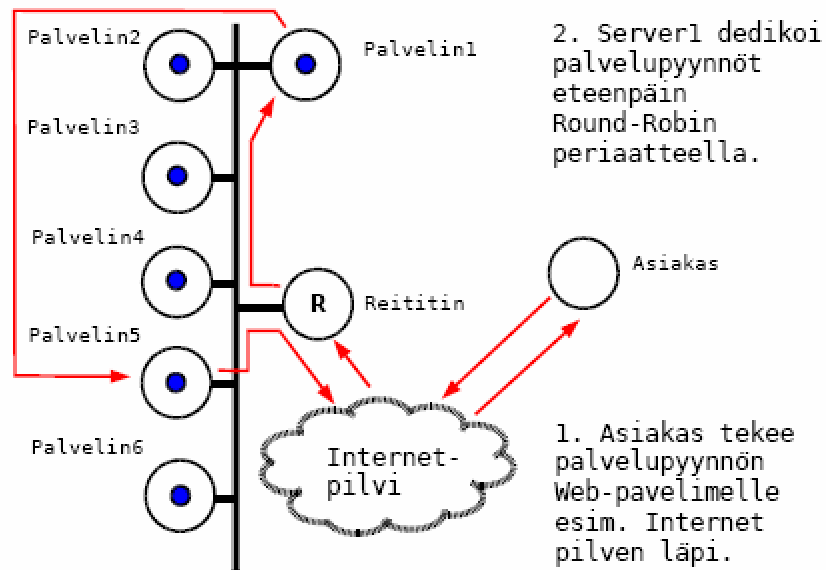
Kuva 8. Kopioitujen tietovarastojen looginen järjestys kolmena sisäkkäisenä kehänä esitettynä [1] (kuva s.326). Pysyväisluontoiset kopiot (permanent replicas), server-initiated replicas (palvelinpyyntöiset kopiot), asiakas-/työasemapyyntöiset kopiot (client-initiated replicas) ja asiakkaat (clients).

Tutkimme seuraavaksi erilaiset kopiotyypit ja selvitämme mitä niillä tarkoitetaan. Selvitämme samalla kopioiden sijaintipaikat (replica placement) merkityksineen.

2.1.1 PYSYVÄISLUONTOISET KOPIOT (permanent replicas)

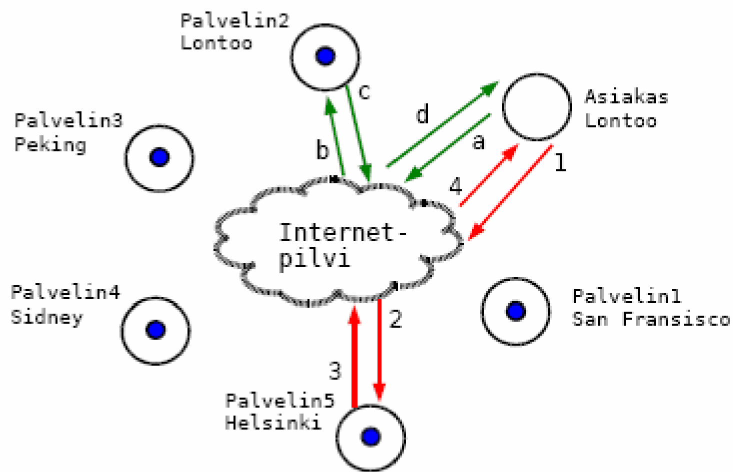
Pysyväisluontoisilla kopioilla tarkoitetaan alkuperäiskopioita, jotka muodostavat itse hajautetun tietovaraston (distributed data store) ja jossa alkuperäiskopioiden määrä on yleensä pieni. Esimerkkitapauksena Distributed Systems –kirja käsittelee Web-palvelimen hajauttamisen, josta se esittää kaksi erilaista toimintamallia. Näistä ensimmäisessä Webpalvelimen muodostavat tiedostot kopioidaan usealle saman paikal-

lisverkon palvelimelle. Aina palvelupyynnön saapuessa Web-palvelimelle ohjataan se yhdelle palvelimista Round Robin -periaatteen mukaisesti.



Kuva 9 Paikallisverkkoon replikoitu Round-Robin -dedikointiperiaatteella toimiva Web-pavelin.

Toisessa mallissa Web-palvelimella sijaitsevat tiedostot kopioidaan rajoitetulle määrälle peilauspalvelimia (mirror sites), joista asiakas tavanomaisesti itse valitsee haluamansa palvelimen. Rypästetyille (clustered servers) ja peilatuille Web-palvelimille on yhteistä staattisessa kokoonpanossa (static configured) olevien kopioiden pieni määrä. Toimintamalli on kuvattu seuraavalla sivulla graafisesti. Edellä esitettyä palvelua käyttävät useat ohjelmistotoimittajat ja he antavat käyttäjä itse valita palvelimen, jolta tiedoston lataaminen (down-load) omalle laitteistolle suoritetaan. Molemmille malleille yhteistä on lukumääräisesti suhteellisen pieni staattisten replikaattien määrä.



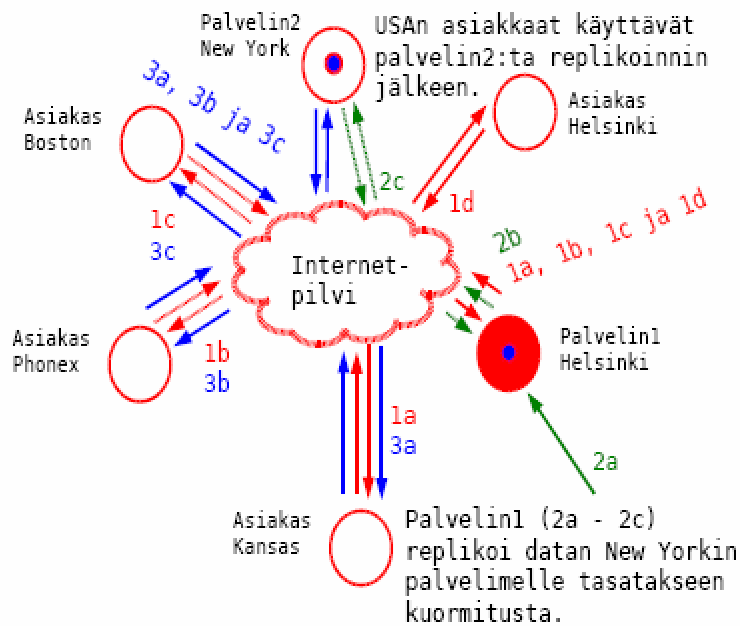
Asiakas selaa Helsingissä sijaitsevan Web-pavelimen sivustoja Lontoosta, josta hän valitsee käyttöönsä lähinnä häntä sijaitsevan Web-palvelimen.

Kuva 10 Pysyväisluontoisten replikaattion peilaaminen.

2.1.2 PALVELINALOITTEISET KOPIOT (server-initiated replicas)

Palvelinpyyntöisten replikaattien idea perustuu tietovaraston omistajan (data store owner) tarpeeseen parantaa ja tasata järjestelmän suorituskykyä. Tietovarastoon kohdistuvia hakupurskeista johtuvaa kuormitusta pyritään tasaamaan riittävällä määrällä väliaikaisia monistettuja tietovarastoja, joiden tarkoitus on toimia lähettävänä välimuistina (push cache) piilottaen tietovaraston lähelle sen käyttäjää.

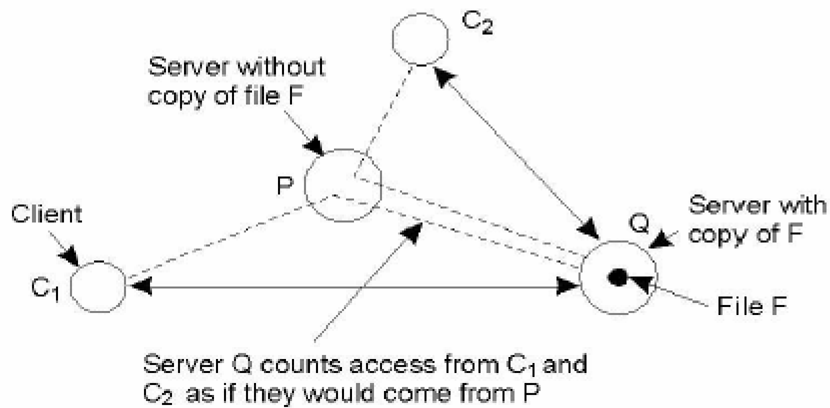
Palvelinpyyntöinen replikointi on tuttua Web-palvelinten maailmasta, jossa staattisia sivuja voidaan tallentaa muiden palvelinten välimuisteihin. Palvelinpyyntöisten kopioiden keskeisin ongelma liittyy replikaattien luomiseen ja poistamiseen: minne maantieteelliseen tai loogiseen paikkaan ja milloin se tulee luoda tai mistä poistaa. Rabinovitch et al. on esittänyt Web-palvelimen dynaamisen replikointialgoritmin Web-sivujen replikointiin [1].



Kuva 11 Palvelin monentaa tietovaraston tasatakseen purskekuormituksen. Tämän jälkeen palvelupyynnöt ohjautuvat kahdelle eri palvelimelle.

Web-palvelimelle tyypillinen ominaisuus on tiedostojen staattisuus, koska Web-sivuston sivuihin kohdistuu harvoin päivitysoperaatioita. Palvelimeen kohdistetaan suureksi osaksi vain lukuoperaatioita. Web-sivuston dynaaminen replikointialgoritmin vähentää Web-palvelimen kuormitusta monistamalla itsensä tai palvelimen kuormitetut tiedostot. Replikointi voidaan toteuttaa joko peilaamalla tai monistamalla ko. sivusto tai tiedosto(t) kyselyitä lähettäneen asiakkaan läheisyyteen.

Kukin palvelin pitää kirjaa tiedostokohtaisista saantipyynnöistä ja mistä saantipyyntö tuli. Lisäksi kukin palvelin pystyy päättelemään mikä palvelimista sijaitsee lähinnä saantipyynnön esittänyttä asiakasta (esim. kuvassa 12 saantipyynnön esittänyttä asiakasta C1 lähinnä on palvelin P).



Kuva 12 Palvelinaloitteiset replikaatit. Asiakkaiden saantipyyntöjen (access requests) laskeminen [1] (kuva s. 328).

Etäisyyden päättelemiseksi tarvittava tieto on saatavissa esimerkiksi reititystaulukosta (routing table). Asiakkaiden C1 ja C2 saantipyyntöjen ohjautuessa palvelimen P kautta jakavat he saman heitä lähimpänä olevan palvelimen P. Kaikki saantipyynnöt tiedostoa F koskien rekisteröidään palvelimella Q yhdeksi saantipyynnöksi (access count) **$cnt_0(P,F)$**. Tiedostoa F koskevien saantipyyntöjen määrän laskiessa palvelimessa S alle poistamiskynnyksen (deletion threshold) voidaan tiedosto F poistaa S:ltä ***del(S,F)***.Tiedoston poistamisen seurauksena muilla palvelimilla oleva kuormitus saattaa kasvaa palvelimen S liikenteen ohjautuessa muille palvelimille.

Käytettäessä algoritmiä tulee varmistaa ainakin yhden tiedostokopion olemassaolo.

Replikointikynnys ***rep(S,F)***, joka valitaan aina korkeammaksi kuin poistamiskynnys, kertoo milloin tiedostoa S koskevien saantipyyntöjen määrä on riittävän suuri, jotta tiedosto S kannattaa replikoida toiselle palvelimelle.

Saantipyyntöjen määrän ollessa poistokynnyksen ja replikointikynnyksen välimaastossa voidaan tiedosto ainoastaan siirtää (migrate) palvelimelle P. Kyseistä tiedostoa koskevien kopioiden määrä tulisi pitää samana. Palvelin Q tarkistaa saantipyynnöt päättäessä laskea uudelleen (reevaluate) varastoimiensa tiedostojen sijaintipaikat.

Mikäli tiedostoa F koskevat saantipyynnöt palvelimella Q laskee alle tuhoamiskynnykset ***del(Q,F)***, eikä tiedosto F ole viimeinen kopiokappale, niin se poistetaan palve-

limelta Q. Palvelinta P pyydetään vastaanottamaan kopio F:stä, jos jollekin vastaavalle palvelimelle kuin P **cnt_Q(P,F)** ylittää puolet kaikista saantipyynnöistä Q:lla eli F:n kopio siis siirretään P:lle. Tiedoston F siirtäminen ei aina onnistu P:lle. Syynä tähän saattaa olla levytilan loppuminen tai palvelimen P liiallinen kuormitus. Tällöin Q pyrkii kopioimaan F:n muille palvelimille.

Tiedoston kopioiminen palvelimelta Q tapahtuu F:ää koskevien saantipyyntöjen ylittäessä replikointikynnyksen ***rep(Q,F)***. Toiminnallisesti palvelin Q tarkistaa kaikki Web-palvelua tarjoavat palvelimet kauimmaisesta lähimpänä olevaan. Palvelimen R ***cnt_Q(R,F)*** ylittäessä Q:lla lasketun vertailuarvon F:lle yrittää palvelin Q kopioida F:n palvelimelle R. Palvelinaloitteista kopioimista voidaan käyttää, mikäli voidaan taata vähintään yhden kopion F olemassaolo. Palvelimella Q on kahdentumisaika tiedostolle F palvelimessa P, jonka määrää tekijät ***rep(Q,F)***, ***del(Q,F)***, ***cnt_Q(Palvelimet,F)*** ja ***cnt_P(Q,F)***. Tieto replikoidaan palvelimelle P, jos ***cnt_P(Q,F) > 0.5 x cnt_Q(Palvelimet,F)***.

Poistamissääntö:

cnt_Q(P,F) < del(P,F), jossa on huomioitava viimeisen kopion poistamatta jättäminen.

Migraatiosääntö:

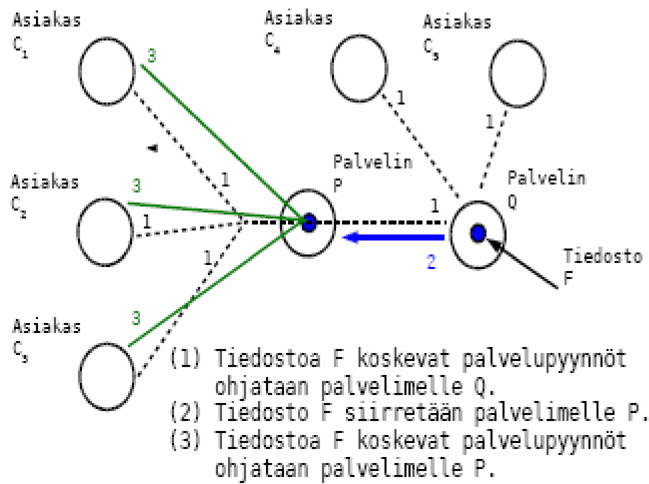
rep(S,F) > del(P,F).

Replikointisääntö:

rep(P,F) > del(P,F) ja ***cnt_P(Q,F) > 0.5 x cnt_Q(Palvelimet,F)***.

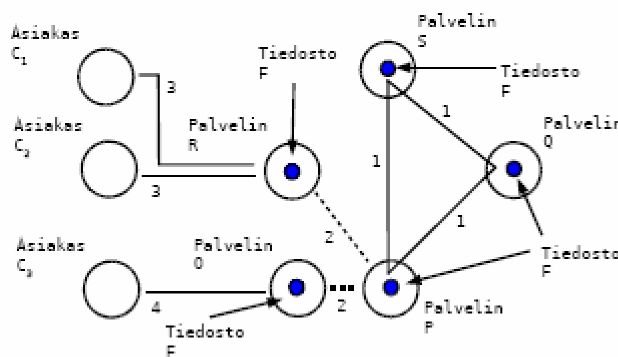
Kuvan 13 jokainen asiakas C1-C5 on viitannut palvelimella Q sijaitsevaan tiedostoon F kerran. Kuvassa oletettu migraatiosääntö ***rep(P,F) > 2*** täyttyy, jolloin tiedosto kopioidaan palvelimelle P. Tiedostoon F tehtyjen viittausten määrä asiakkailla C1 - C3 P:ltä Q:lle ylittää puolet kaikista tiedostoon tehdyistä viittauksista Q:lla säännön ***cnt_P(Q,F) > 0.5 x cnt_Q(Palvelimet,F)*** eli ***3 > 0.5 x 5*** mukaan, joten P:ltä Q:lle tiedostoa F koskevat palvelupyynnöt ohjataan palvelimelle P. Palvelin P vastaa asiakkaiden C1 – C3 palvelupyyntöihin koskien tiedostoa F.

$rep(P,F) > del(P,F)$ ja $cnt_P(Q,F) > 0.5 \times cnt_Q(Palvelimet_{P,Q},F)$



Kuva 13 Palvelinaloitteisen replikoinnin toiminta.

Tiedon johdonmukaisuuden (data consistency) turvaamiseksi pysyväisluonteisia kopioita käytetään yleensä varmuuskopioina, tai ainoana sellaisena tietovarastona, johon voidaan kohdistaa muutoksia. Palvelinaloitteista kopioimista käytetään pysyväisluonteisen kopioinnin lisänä ja sen avulla toimitetaan vain luettavaksi tarkoitetuista tiedostoista kopio asiakasta lähellä sijaitsevalle palvelimelle.



Pysyvät replikaatit (1), joista on otettu lukukopiot (2) kahta eri asiakasryhmää (C_1-C_2 ja C_3) palvelemaan. Lukukopioihin (R ja O) ei voida kohdistaa päivitysoperaatioita, vaan ne tehdään pysyvissä replikaateissa Q, S ja P.

Kuva 14 Pysyväisluonteiset ja lukureplikaatit. Ainoastaan pysyviin replikaatteihin voi kohdistua muutoksia.

Kuvassa 14 palvelimet Q, S ja P muodostavat pysyvien replikaattien ryhmän, joissa sijaitsevaan tiedostoon F voidaan kohdistaa muutoksia.

Pysyväisluonteisen replikaattipalvelimen P tiedostosta F on otettu asiakaspalvelupyyntöjä varten lukukopio replikaattipalvelimelle O ja R.

Replikaattipalvelimilla R ja O olevaan tiedostoon F voidaan kohdistaa asiakaspyyntö tiedoston lukemiseksi. Asiakaspalvelupyynnöt C1 ja C2 ohjautuvat palvelimelle R ja C3 palvelimelle O tasaten näin replikaattipalvelimiin kohdistuvaa kuormitusta.

2.1.3 ASIAKAS-/TYÖASEMA-ALOITTEISET KOPIOT (client-server replicas)

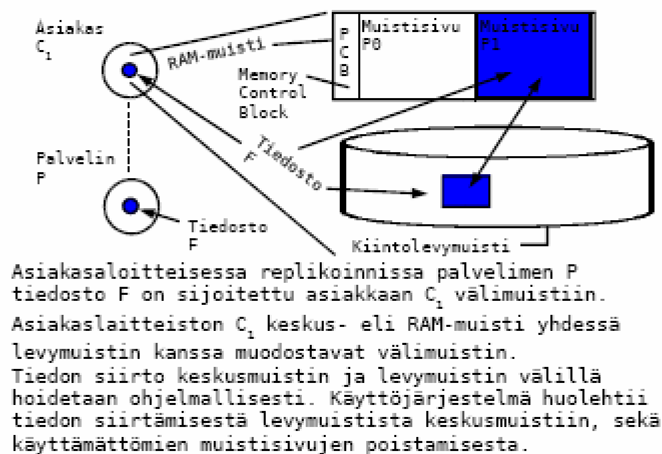
Asiakas- tai työasema-aloitteiset replikaatit tunnetaan paremmin työaseman välimuistikopiona (client cache replica) ja replikointimuodolla lyhennetään tiedon saantiviivettä. Työasema varastoi eli puskuroi muistiinsa hetkeä aikaisemmin pyytämänsä tiedon (data). Kun tietoon seuraavan kerran kohdistuu saantipyyntö, niin tieto on saatavissa työaseman tai samassa paikallisverkossa olevan laitteiston välimuistista, eikä sitä enää jouduta erikseen palvelimelta työasemalle noutamaan.

Työasema ja paikallisverkon laite vastaa itsenäisesti välimuistinsa toiminnasta, eikä sen sisältämän tiedolla ole mitään tekemistä tiedon ristiriidattomuuden kanssa. Tiedon eheydestä vastaavat palvelimet, eivät asiakkaat tai paikallisverkkoon kytketty muu laite. Useassa replikointitapauksessa työasema voi luottaa tietovaraston ilmoitukseen puskuroidun tiedon vanhenemisesta (stale). Asiakas-aloitteinen replikointi on varsin toimiva idea kunhan noudettu tieto (fetched data) ei ole muuttunut palvelimelta välimuistiin ja sieltä elvyttämisen kuluessa tai sen jälkeen.

Työasema-aloitteisen replikoinnin onnistuminen riippuu pitkälti jaettavaksi tarkoitetun tietovaraston (data store) tyypistä. Tiedostojärjestelmien osalta ei tiedostoja juurikaan jaeta, joten niiden jakaminen tällä tavoin on turhaa. Websivujen osalta työasema-aloitteisen välimuistin käyttäminen on osoittautunut hyödylliseksi, joskin sen merkitys vähentyy asteittain, sillä asiakaspyynnön seurauksena tapahtuu harvemmin laitteisto-

jen välimuistin jakamista Web:ssä olevan sivumäärän kasvaessa (so. samoihin sivuihin tehdään harvemmin viittauksia kokonaissivumäärän kasvaessa).

Välimuistista säilytetään tietoa tietty aika. Käyttöjärjestelmän muistinkäsittely- eli sivutusalgoritmi sekä käyttöjärjestelmän parametrisointi vastaavat vanhentuneiden välimuistisivujen poistamisesta tai uusimisesta.



Kuva 15 Väli- eli cache-muistin toimintaperiaatekuva.

Onnistuneita tiedonhakupyyntöjä cache-muistista mitataan välimuistiosumilla (cache-hit -arvo), jonka arvon voidaan kuvitella lähenevät esimerkiksi lukua yksi sitä enemmän mitä onnistuneemmin sivut on onnistuttu noutamaan välimuistista. Välimuistiosuma-arvoa voidaan kasvattaa määrittelemällä paikallisverkkoon kytkettyjen laitteiden välimuisti yhtenäiseksi. Laitteiden välistä yhtenäistä jaettua välimuistia käytetään tilanteissa, joissa oletetaan käyttäjien palvelinpyyntöjen kohdistuvan yhteiseen data-avaruuteen.

Tavallisesti cache- eli välimuisti määritellään asiakaslaitteistolle tai paikallisverkossa olevalle jaetulle laitteistolle (shared machine). Välitasojen luonti on mahdollista ja toisinaan järjestelmähallinto (system administrator) saattaa asettaa jaettu välimuisti osastojen, organisaatioiden tai eri alueiden välille.

Toisaalta aivan toinen lähestymistapa on sijoittaa cache-palvelimet alueverkon haluttuihin paikkoihin ja antaa asiakkaan navigoida itsensä häntä lähinnä olevalle palvelimelle.

2.1.4 PÄIVITYSTEN ETENEMINEN (update propagation)

Päivitysoperaatiot suoritetaan tavallisesti asiakkaasta käsin yhteen replikaateista, josta se etenee (propagate) muihin replikaatteihin johdonmukaisesti (consistecy). Päivitysten levittämisessä tulee huomioida erilaisia toteutuksen suunnitteluun liittyviä asioita, joihin seuraavaksi perehdymme.

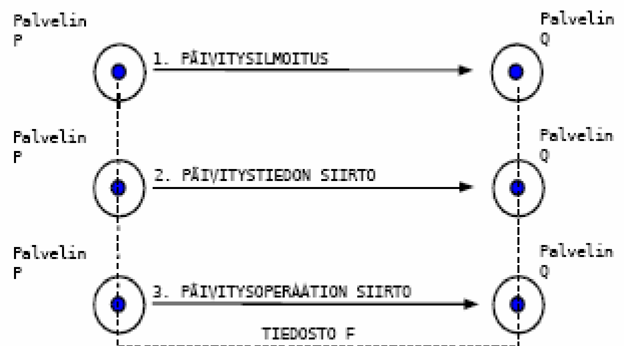
Päivityksen tila vastaan päivitysoperaatiot:

Suunnittelussa on tärkeätä huomioida mitä päivitetään.

1. Levitä vain päivitystä koskeva ilmoitus (update notification).

2. Siirrä tieto yhdestä kopiosta toiselle

3. Levitä päivitysoperaation muille kopioille



Päivityksestä voidaan ilmoittaa kolmella (1-3) tavalla palvelinten P ja Q välillä:
1. Tekemällä muille palvelimille päivitysilmoitus.
2. Siirtämällä päivitettävä tieto P:ltä Q:lle.
3. Siirtämällä päivitysoperaatio P:ltä Q:lle.

Kuva 16 Päivitysten etenemisen kolme eri toteutustapaa.

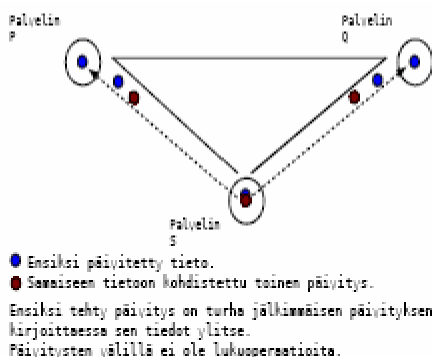
Mitätöimisprotokolla (invalidation5,6 protocols) kertoo muille replikaateille, ettei päivitysoperaation kohteena oleva tietovaraston tieto ole enää kelvollista (valid). Protokolla voi kertoa mikä tietovaraston osa tai sen osoitettavissa oleva tietoyksikkö on päivitetty niin, että osa sen sisältämästä tiedosta on mitätöity.

Replikaattien välillä siirretään vain tieto tiedon mitätöimisestä. Asetettaessa mitätöityyn tietoon kohdistuva palvelupyyntö päivitetään tieto aina ajan tasalle ennen vasta-

uksen palauttamista pyynnön esittäneelle osapuolelle, riippuen siitä millaista tiedon yhtenevyysmallia käytettävä protokolla tukee.

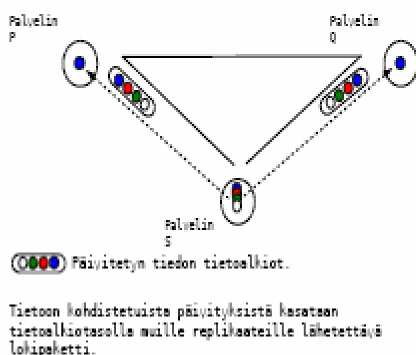
Ainoa siirtyvä tieto on mitätöityä tietoa koskeva erittely (spefication), minkä takia mitätöimisprotokolla toimii pienellä kaistaleveydellä. Mitätöimisprotokolla toimii parhaiten päivitysoperaatioiden määrän ollessa suuri lukuoperaatioihin nähden. Tällöin lue-kirjoittaaksesi –operaatioiden (read-to-write) määrä on suhteellisen pieni.

2.1.4.1 MITÄTÖIMISPROTOKOLLAN TOIMINNAN TARKASTELU



Kuva 17 Sarjallisen tiedon päivitys.

1. Tarkasteltaessa esimerkkitilannetta, missä päivitystä koskevat tiedot lähetetään kaikille replikaateille, saatetaan joutua tilanteeseen, missä sarjallisesti kaksi tai useampi päivitys kohdistuu tietoon (data) ilman siihen välillä kohdistettavaa lukuoperaatiota. Tällöin ensiksi tehdyn mitätöinnin päivittäminen tietovarastoon olisi turha jälkimmäisen päivitysoperaation kirjoittaessa tarpeetta yli ensiksi tehty päivitys. Pelkän tiedon lähettäminen tiedon muuttumisesta muille replikaattipalvelimille olisi ollut toiminnallisesti tehokkaampi toimintatapa.

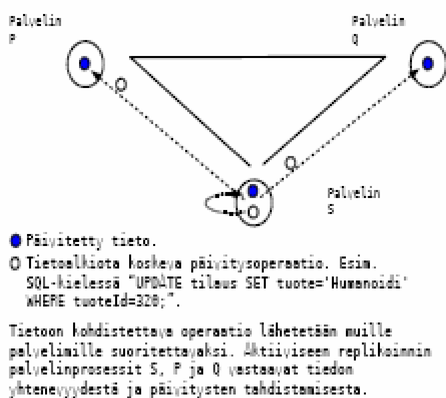


Kuva 18 Sarjallisten tiedon päivittäminen tietoalkioilla.

2. Muutetun tiedon eli tietoalkioiden (data item) siirtäminen replikaattien välillä olisi toinen toimintatapa ja käytännöllinen lue-kirjoittaaksesi -operaatioiden määrän ollessa korkea. Todennäköisyys päivityksen tehokkaasta (effective) voimaansaattamista on korkea johtuen muutettuun tietoon kohdistetuista lukuoperaatiosta ennen siihen seuraavaksi

kohdistettavaa päivitystä. Vaihtoehtoisesti järjestelmä pitää lokikirjaa tietoon kohdistetuista muutoksista ja protokolla siirtää vain muutoslokit replikaattien välillä säästäen kaistaleveyttä.

Muutokset kasataan usein monta muutosta käsittäviksi paketeiksi, jotka välitetään yhtenä viestinä replikaattien välillä vähentäen näin viestiliikenteen ylikuormittumista (communication overhead).



Kuva 19 Sarjallinen päivittäminen päivitysoperaatioilla.

3. Kolmas mitätöimisprotokollan muoto on kertoa jokaiselle replikaatille minkä päivitysoperaation sen tulee suorittaa. Tätä toimintatapaa kutsutaan myös aktiiviseksi replikoinniksi (active replication). Aktiiviseen replikointiin osallistuvalla replikalla on prosessi, jonka oletetaan kykenevän pitämään siihen liitetyt tietoalkiot replikaattien tasolla ajan tasalla.

Mikäli välitettävien parametrien (parameters) määrä on suhteellisen pieni, niin päivitykset eivät juuri kuluta tietoliikennekaistaa. Replikoissa suoritettavien operaatioiden monimutkaisuudesta johtuen saatetaan palvelimissa tarvita muita vaihtoehtoja, kuten enemmän prosessointitehoa (processing power).

2.1.4.2 TYÖNNÄ-KÄYTÄNTÖ (push protokolla)

On makuasia tapahtuuko tiedon päivittäminen työntämällä (push) vai vetämällä (pull) tarvittava tieto asiakkaan ja palvelimen välillä. Työntökäytännössä (push protocol), jota myös palvelinpohjaiseksi käytännöksi (server based protocol) kutsutaan, päivitykset työnnetään toisille palvelimille ilman heiltä tulevaa pyyntöä.

Työntökäytäntö on yleinen pysyväisluoteisessa (permanen replicas) ja palvelinaloitteisessa replikoinnissa (server-initiaited replicas). Käytäntöä voidaan soveltaa asiakaslaitteistoille tapahtuvissa välimuistipäivityksissä. Palvelinpohjaista käytäntöä yleensä

käytetään ylläpidettäessä suhteellisen korkeata tiedon yhdenmukaisuusastetta – toisin sanoen silloin, kun kopioidun tiedon eli replikoiden tulee olla identtisiä sen kaikilta osin. Pysyväisluoteiset ja palvelinaloitteiset replikaatit, sekä suuret jaetut välimuistit (large shared cache) on usein jaettu monen asiakkaan kesken, jotka lähinnä suorittavat tietoon kohdistuvia lukuoperaatioita.

Lue-kirjoittaaksesi (read-to-write) suhdeluku (ratio) jokaisessa kopiossa on suhteellisen korkea. Näissä tapauksissa työnnä-käytäntö (push protocol) on tehokas, sillä jokaista replikaateille työnnettyä tiedon päivitystä kohti on oletettavissa yksi tai useita päivitettyyn tietoon kohdistuvia lukuoperaatioita. Työnnä-käytännöllä saadaan ristiriidaton (consistent) tieto kätten ulottuville heti sitä kysyttäessä ilman mitään lisätoimenpiteitä.

2.1.4.3 VEDÄ-KÄYTÄNTÖ (pull protocol)

Vedä-pohjaisessa käytännössä (pull-based protocol) palvelin tai asiakas pyytää toista palvelinta lähettämään sillä ajan hetkellä kaikki saatavilla olevat päivitykset. Vedä-käytäntö tunnetaan myös nimellä asiakas-pohjainen käytäntö (clien-based protocols) ja asiakkaiden välimuistin hallinta usein noudattaa kyseistä käytäntöä.

Yleinen strategia on ensin tarkistaa onko välimuistissa oleva tietoalkio vielä ajan tasalla. Välimuistin saadessa pyyntö paikallisesti saatavilla olevasta tietoalkiosta tarkistaa välimuisti alkuperäiseltä tiedon sisältävältä palvelimelta onko pyynnön kohteena olleita tietoalkioita muutettu sitten niiden välimuistiin tuomisen jälkeen. Törmättäessä muutokseen siirretään muuttunut tieto ensin palvelimen välimuistiin, jonka jälkeen se palautetaan palvelupyynnön tehneelle asiakkaalle. Ellei tarkistettavaan tietoon ollut kohdistunut muutoksia, niin asiakkaan välimuistissa oleva tieto palautetaan kyselyn tehneelle osapuolelle. Asiakas kiertokyselee (poll) palvelimelta selvittääkseen josko välimuistin tieto on päivitettävä ajan tasalle.

Vedä-tyyppinen käytäntö on tehokas lue-kirjoittaaksesi -suhdeluvun (read-to writeratio) ollessa suhteellisen matala. Ei jaetun asiakasvälimuistin (nonshared client cache) tämä on tyypillisesti vallitseva tilanne. Jaetun asiakasvälimuistin osalta vedä-käytäntö voi osoittautua myös tehokkaaksi, mikäli välimuistin tietoalkioita ei jaeta.

Suurin takaisku vedä-käytäntöä noudatettaessa on vasteaikojen (response time) kasvu etsittävän tiedon puuttuessa välimuistista. Työnnä- ja vedä-käytännön välillä valitseminen on aina kaupankäyntiä hyvien ja huonojen puolien valitsemisen välillä. Alla taulukossa koottuna vertailu vedä ja työnnä-käytännön eduista ja haitoista palvelin-asiakaskeskeisessä järjestelmässä.

Asia	Työnnä-käytäntö	Vedä-käytäntö
Palvelimen tila	Asiakasreplikaateista ja välimuisteista lista	Ei mitään
Viesti lähtee	Päivitys (ja mahdollisesti päivityksen nouto myöhemmin)	Pollaa ja päivitä
Asiakkaan vaste-aika	Välitön vastine (tai päivityksen noutamiseen kuluva aika)	Päivityksen noutamiseen kuluva aika

Yhden palvelimen ja usean asiakkaan järjestelmässä toteutettu vertailu työnnä- (push-based protocol) ja vedä-käytännön (pull-based protocol) välillä.

Kuva 20 Työnnä- ja vedä-käytännön vertailutaulukko [1] (s. 332 kuva 6-26 suomennos).

Seuraavaan kolmeen asiaan tulee kiinnittää huomiota työntö- ja vedä-käytäntöjä vertailtaessa:

1. Työnnä-käytännössä joutuu palvelin ylläpitämään listaa kaikista asiakasvälimuisteista, mistä saattaa aiheutua ylimääräistä palvelinkuormitusta. Kuormitusongelma tulee vastaan suurissa asiakas-palvelin -järjestelmissä. Tällöin palvelin joutuu käymään läpi asiakaslistansa eli kirjanpidon asiakkaiden välimuisteissa sijaitsevista sivuista, joissa päivitettävä sivu sijaitsee.

2. Palvelimen ja asiakkaan kesken välitettävä viestisisältö eroaa vallitsevan käytännön mukaisesti. Työnnä-käytännössä palvelin ainoastaan lähettää päivityksen kullekin asiakkaalleen. Välitettävän päivitystiedon ollessa vain mitätöintiviestejä (invalidation) tarvitaan lisäkommunikaatiota asiakkaan noutaessa vaadittua tietoa. Vedä-pohjaisessa

käytännössä asiakaan täytyy kiertokysellä aika ajoin palvelinta ja tarvittaessa noutaa muutettu tieto asiakaslaitteen välimuistiin.

3. Jokaiselta asiakkaalta saatu vasteaika eroaa toisistansa. Palvelimen työntäessä muutettua tietoa asiakaslaitteistolle on asiakaslaitteiston vasteaika nolla. Työnnettäessä mitätöintiviestejä asiakkaalle on vasteaika sama kuin vedä-käytäntöä noudatettaessa, jolloin sen määrää muutetutun tiedon noutoaika palvelimelta.

Kaupankäynti eri hyötyjen ja haittojen välillä on johtanut vuokra-aikasopimus eli (lease) hybridi-mallin (hybrid model) syntymiseen päivitysten levittämisessä. Vuokra-aikasopimus mallissa palvelin lupautuu työntämään päivitykset asiakaslaitteistolle tietyn ajan. Vuokra-aikasopimuksen päätyttyä asiakkaan on kiertokyseltävä palvelinta päivitysten varalta ja tarvittaessa vedä-käytännön mukaisesti noudettava muuttunut tieto laitteistolleen. Vaihtoehtoisesti asiakas voi pyytää uutta vuokra-aikasopimusta palvelimelta saadakseen palvelimen jatkamaan työntö-käytännön mukaisesta päivitetyn tiedon automaattista päivittämistä.

Duvvuri et al. (2000) kuvaa joustavan vuokra-aika -järjestelmän (flexible lease system), jossa vuokra-ajan pituus voidaan dynaamisesti valita eri vuokrauskriteerien perusteella. Joustava vuokra-aikajärjestelmä erottelee kolme eri tekijää vuokra-aikasopimukseen liittyen:

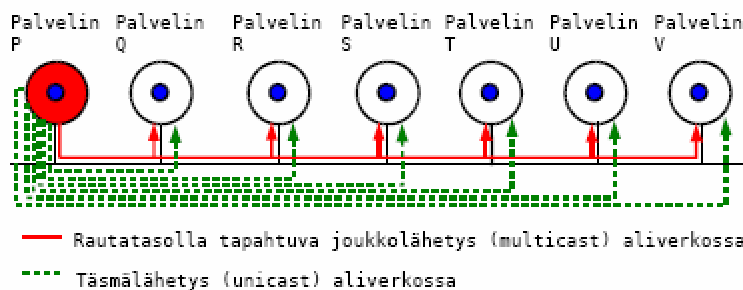
1. Vuokra-aikasopimuksen ikäpohjaisuus, joka peruu tiedon viimeiseen muuttamisaikaleimaukseen. Mallin perusidea on oletus, jonka mukaisesti tieto, mikä ei ole muuttunut pitkän ajan kuluessa voidaan olettaa pysyvän muuttumattomana vielä jonkin aikaa tulevaisuudessa. Pitkäaikaisten vuokra-aikasopimusten myöntäminen pitkään muuttumattomana pysyvälle tiedolle vähentää merkittävästi päivitysviestejä.

2. Toinen vuokra-aikasopimustyypin valintaa vaikuttava tekijä perustuu asiakkaiden välimuistin päivitysfrekvenssiin. Palvelin myöntää pitkäaikaisen vuokra-aikasopimuksen useasti välimuistipäivitystä tarvitsevalle asiakkaalle ja lyhytaikaisen vuokra-aikasopimuksen harvoin välimuistipäivitystä pyytävälle asiakkaalle. Palvelin pitää kirjaa vain niistä asiakkaista, joissa sen sisältämä tieto on suosittua ja joille tarjotaan suuri tiedon ristiriidattomuusaste (high degree of consistency).

3. Kolmas kriteeri liittyy palvelimen tila-avaruudessa (state-space) olevaan ylikuormitukseen (overhead). Palvelimen ymmärtäessä tulevansa asteittain kuormitetuksi vähentää se asiakkaille myöntämiensä uusien vuokra-aikasopimusten kestoaikaa. Vuokra-aikasopimusten päättyessä (expired) nopeammin palvelimen tarvitsee pitää kirjaa pienemmästä asiakasmäärästä. Palvelin siis dynaamisesti kytkee (dynamic switching) itsensä enemmän tilattomaan olotilaan suoritettavien prosessien osalta vähentäen omaa kuormitustansa menettelemällä em. tavoin.

2.1.4.4 TÄSMÄLÄHETYS (unicast) VAI JOUKKOLÄHETYS (multicast)?

Päivitysviesti voidaan lähettää vastaanottajalle joko täsmälähetyksenä tai joukkolähetyksenä. Täsmälähetyksessä tietovarastona tai sen osalta toimina palvelin lähettää päivitysviestinsä N:lle muulle palvelimelle. Joukkolähetyksessä tietoverkko vastaan viestin toimittamisesta usealle vastaajalle ja se hyödyntää verkko-osoiteavaruutta, sekä verkossa tapahtuvaa viestivälitystä.



Joukkolähetyksen ja täsmälähetyksen ero palvelimien, tai asiakkaiden ja palvelimen ollessa kytkettynä samaan verkko-segmentiin.

Täsmälähetys kuormittaa verkkokaistaa malliesimerkissä viisi kertaa enemmän verrattuna yhden kerran tehtävään joukkolähetykseen.

Kuva 21 Täsmä- ja joukkolähetyksen välinen ero päivitysviestiä välitettäessä.

Useissa tapauksissa on edullisempaa käyttää joukkolähetystä. Erikoistapauksena voimme tarkastella tilannetta, jossa kaikki palvelimet sijaitsevat samassa aliverkossa ja niille tehtävä viestinvälitys voidaan hoitaa yhdellä ainoalla yleislähetys-viestillä (broadcast message).

Täsmälähetyksenä tehdyn viestin kustannus olisi tällöin huomattavasti suurempi verrattuna yhteen ainoaan verkon päästä päähän matkaavaan yleislähetysviestiin, eikä viestimuoto olisi taloudellinen palvelimen tarvitseman prosessointiajan tai tietoverkkokaistan kannalta. Joukkolähetys voidaan usein yhdistää työnnä-käytäntöön päivitystietoa levitettäessä. Tällöin palvelin, joka lähettää päivitystiedon levitykseen, käyttää yhtä joukkolähetysryhmää (multicast group) päivityksen levittämiseen. Vastavuoroisesti työnnä-käytännössä tavallisesti vain yksi asiakas tai palvelin pyytää replikaattiaan päivitettäväksi, jolloin täsmälähetys saattaa olla tehokkain toimintavaihtoehto.

2.1.5 EPIDEEMISET PROTOKOLLAT (epidemic protocols)

Useille tietovarastoille riittää tiedon tarjoaminen vain tapahtumaristiriidattomassa tilassa (eventual consistent state). Tietovaraston on taattava kaikkien replikaattien identtisyys ja tiedon ristiriidaton tila sitä päivitettäessä.

Päivitysten levittäminen tapahtumaeheisiin tietovarastoihin implementoidaan (implement) toteutuksiin yleensä käyttämällä epideemisen-algoritmiluokan (class of algorithms) ilmentymiä, jonka tarkoitus on vasta päivityksen levittämisestä replikaateille mahdollisimman pienellä viestien määrällä. Päivityksen yleensä kootaan yhdeksi viestiksi, jota sitten vaihdetaan kahden palvelimen välillä. Selittääksemme algoritmien yleiset periaatteet oletamme jokaisen tietoon kohdistuvan päivitysten tulevan yhdeltä palvelimelta välttääksemme kirjoita-kirjoita konfliktin (write-write conflict) käsittelyn.

Kuten nimestäkin voi päätellä, niin epideeminen-käytäntö (epidemic protocols) perustuu tarttuvien tautien leviämisteoriaan, mutta tarttuvien tautien sijaan replikoivat tietokannat levittävät päivityksiä ympäristöönsä. Toisin kuin lääkärikollegat, jotka yrittävät estää tarttuvien tautien leviämisen, epideemisten-algoritmien suunnittelijat yrittävät tartuttaa (infect) uusilla kopioilla kaikki replikaatit mahdollisimman nopeasti hajautettuihin tietovarastoihin.

Alan termeillä ilmaistuna hajautetun tietovaraston osana olevaa palvelinta kutsutaan tartuttajaksi (infective), mikäli se yrittää levittää päivitystä toisille palvelimille. Päivi-

tykselle altistumatonta palvelinta kutsutaan päivitykselle alttiiksi (susceptible). Tapahtumasarjan lopuksi päivitettyä palvelinta, joka ei kykene tai halua levittämään päivitystä, kutsutaan poistetuksi (removed).

2.1.5.1 PÄIVITYSTEN ETENEMISMALLIT (update propagation models)

Tämä kappale käsittelee kaksi etenemismallia, joista ensimmäinen on antientropia (anti-entropy) ja toinen huhunlevittämismalli (rumour spreading).

Anti-entropia -malli (anti-entropy model) ja sen toiminta

Anti-informaatioastetta koskevan suositun etenemismallin mukaisesti antientropiasta (anti-entropy) palvelin P poimii satunnaisesti palvelimen Q vaihtaakseen tämän kanssa päivityksiä koskevaa tietoa:

1. P vain työntää omat päivityksensä Q:lle
2. P vetää Q:lta itselleen uudet päivitykset
3. P ja Q lähettelevät päivityksiä toisillensa (vedä-työnnä -käytännön mukaisesti

Päivityksiä levitettäessä vain työnnä-käytäntö osoittautuu huonoksi vaihtoehdoksi, sillä päivitykset voidaan levittää vain tartuttajina toimiviin palvelimiin. Usean palvelimen toimiessa tartuttaja (infective) on pieni todennäköisyys sille, että jokainen niistä onnistuisi valitsemaan kumppanikseen päivitykselle alttiin palvelimen. Tietty palvelin pysyy todennäköisesti pitkän aikaa päivitykselle alttiina (suspectible) vain sen takia, ettei se tule valituksi tartuttajana toimivan palvelimen taholta.

Vastakohtaisesti vedä-käytäntö toimii paremmin useiden palvelinten toimiessa tartuttajina. Tällöin päivitysten levittämisestä vastaavat päivitykselle alttiiksi viritetyt (triggered) palvelimet. Suurella todennäköisyydellä viritetty ja päivitykselle alttiiksi joutuva palvelin ottaa yhteyttä tartuttajana toimivaan palvelimeen tarkoituksena vetää päivitykset itselleen ja tullakseen näin myös itse tartuttajaksi.

Huhunlevittämismalli (rumour spreading model)

Eräs edellä esitetyn mallin variaatio on huhunlevittämismalli tai toiselta nimeltään juoruilumalli (gossiping model) ja se toimii seuraavalla tavalla: Jos palvelin P on juuri päivitetty tietoalkion X osalta, niin se ottaa yhteyttä mielivaltaisesti palvelimeen Q ja yrittää työntää päivityksen Q:lle. Jos Q on kuitenkin jo päivitetty jonkin toisen palvelimen toimesta, niin tällöin P saattaa menettää mielenkiintonsa yrittää päivittää Q:ta enää tulevaisuudessa vaikkapa todennäköisyydellä 1/k. Loppujen lopuksi Q poistuu P:n päivityslistasta.

Juoruilu on todella tehokas tapa levittää päivityksiä, mutta sen avulla ei voida taata kaikkien palvelinten päivittymistä. Voidaan toteen näyttää palvelinmäärän ollessa suuri päivittymättömien tai päivitykselle alttiina olevien palvelinten määrää kuvaavan murtoluvun s noudattavan yhtälöä $S = e^{-1(k+1)(1-s)}$, *missä tn*=1/ k .

Anti-entropia -mallin yhdistäminen juoruilumalliin takaa päivityksen leviämisen kaikille palvelimille. Prosessien välinen synkronointimäärä pysyy pienenä verrattuna muihin levitysmalleihin. Epideemisen algoritmien etu on niiden skaalautuvuus.

2.1.5.2 TIEDON POISTAMINEN

Epideemiset algoritmit soveltuvat käytettäväksi hyvin päivitysten levittämiseen tapahtumaristiriidattomassa tietovarastossa (eventual-consistent data store). Algoritmissä on kuitenkin yksi aika outo sivuvaikutus, nimittäin tiedon poistamisesta kertovan päivityksen tekeminen on vaikeata. Tämä perustuu siihen tosiasiaan, että tietoalkion poistaminen poistaa kaiken alkiota koskevan tiedon. Kun tieto on poistettu palvelimelta, niin se saa ennen pitkään vanhoja kopioita tietoalkiosta ja tulkitsee ne jonkin sellaisen tiedon päivitykseksi, jota palvelimella ei aiemmin ole ollut. Poisto tulee tehdä siis päivityksenä, jossa kerrotaan tiedon olevan ”poistettu”. Tällöin vanhat kopiot samasta tietoalkiosta tulkitaan poisto-operaation päivittämiksi. Poistaminen hoidetaan lähettämällä ja tallettamalla kuolintodistus (death certificates). Kuolintodistusten kuittaamien tie-

tojen poistamiseksi on kehitetty malli, jossa poistettavia tietoja koskevat kuolintodistukset aikaleimataan ja tieto siirretään uinuvaan (dormant) tilaan. Tietyn ajan kuluessa poistoa koskeva päivitys on levinnyt kaikkiin palvelimiin, jolloin poistettu tieto voidaan lopullisesti hävittää.

Jotta voidaan varmistua tiedon poistamisesta, niin vain muutamiin palvelimiin jätetään kuolintodistukset pysyviksi. Jos jostain syystä palvelin P saa tietoalkiota X koskevana kuolintodistuksen säilyttäjänä päivityksen, niin se yksinkertaisesti levittää uudelleen X:ää koskevan kuolintodistuksensa.

3 YHTEENVETO

- Åke Tyvi –

Julkaisun ensimmäisessä luvussa tarkastelimme tietokeskeisten eheysmallien tiukkaa eheyttä. Sitten selvitimme linaarisoitavuuden käsitteen ja jaksollisen eheyden.
Tiukasta eheysmallista totesimme, ettei mallia voida käytännössä toteuttaa. Toteutukset siis noudattavat muita eheysmalleja.
Tiukassa eheysmallissa jokainen lukuoperaatio tietoalkioon X palautti arvon viimeisimmästä kirjoitusoperaatiosta tietoalkioon X. Jaksollisessa eheydessä kaikki prosessien tietovarastoon kohdistuvat operaatiot (luku- ja kirjoitus) suoritetaan jossain jaksollisessa järjestyksessä ja jokaisen yksittäisen prosessin operaatio tapahtuu tässä järjestyksessä ohjelman määräämällä tavalla.
Luvussa yksi tulisi lisäksi käsitellä kausaalinen ristiriidattomuus (causal consistency), FIFO ristiriidattomuus (FIFO consistency), synkronointimuuttujilla toteutettu heikko ristiriidattomuus (weak consistency), semaforia käyttävä vapauta-lukko - ristiriidattomuus (release consistency) ja kriittistä lohkoa hyväksikäyttävä sisäänpääsykäytäntöä noudattava ristiriidattomuusmalli (entry consistency), joita ajanpuutteen vuoksi ei käsitellä tässä julkaisussa.

Toisessa luvussa kävimme läpi hajautusprotokollat. Aluksi selvitimme kopioitujen tietovarastojen kolme loogista sijaintipaikkaa. Nämä olivat pysyväisluontoiset, palvelinpyyntöiset ja asiakas-/työasemapyyntöiset kopiot.
Sitten tarkastelimme päivitysten etenemistä kolmea mahdollista käytäntöä noudattaen: Kuten muistamme, niin päivityksen tila voidaan levittää palvelimien välillä joko päivitysilmoituksella, siirtämällä itse päivitettävä tieto tai sen päivitysoperaatio.
Päivityksen etenemistä tarkasteltiin ja sitä havainnollistettiin kolmen mitätöimiskäytäntöesimerkin avulla, jonka jälkeen päivitysten toimintatavoista esitettiin työnnä- ja vedä-käytäntö. Osion viimeinen asiakokonaisuus käsitteli päivityksen tekotapaa joukkolähetyksenä tai täsmälähetyksenä.

Viimeinen luvun kaksi käsiteltävä asia oli epideemiset protokollat. Asiakokonaisuus aloitettiin määrittelemällä termi ’epideeminen käytäntö’, jota seurasi päivitysten etenemismallien esittely (anti-entropia- ja huhunlevittämismalli). Lopuksi selvitimme miten vaikeata tiedon poistaminen on, ellei sitä tehdä kuolintodistuskäytännön avulla.

KIRJALLISUUS- JA VIITELUETTELO

[1] Andrew S. Tanenbaum, Maarten van Steer, Distributed Systems Principles and Paradigms, 2002
[2] Ilkka Haikala, Hannu-Matti Järvinen, Käyttöjärjestelmät, 2003
[3] , The Wordsworth Dictionary of Science & Technology, 1998
[4] Raija Hurme, Riitta-Leena Malin, Olli Syväoja, Suomi-englanti-suomi sanakirja, 2001
[5] Annukka Aikio, Rauni Vornanen, Uusi sivistyssanakirja, 1994
[6] Atk-sanasto, http://www.kolumbus.fi/linnala/janne/atksanas.htm [2.3.2005]

Runo oikeudelle:

Tämä on Kabbalani.

Koska en ole oikeutta saanut,
olet sitä toisin hakenut.

Sillä olen sinut Kabbalaani kironnut.

Tämä on seinä, jonka olen luonut
pitämään sinut erossa minusta.

Kaikki mitä ikinä olen saanut,
on Jumala minulle palauttava.

Sillä niin minut oli petetty,
ettei omaisuuteni takaisin tullut
ilman vääryyttä.

Minä olen teille Saatana.

Jumalalta Åkelle