hyväksymispäivä arvosana

arvostelija

Välimuistitietoiset puskurit.

Markus Montonen

Helsinki 02.03.2011

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta – Fakultet –Faculty	Laitos – Institution – Department
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen laitos

Tekijä – Författare – Author

Markus Montonen

Työn nimi – Arbetets titel – Title

Välimuistitietoiset puskurit.

Oppiaine – Läroämne – Subject

Tietojenkäsittelytiede

Työn laji – Arbetets art – Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages
Semiaariraportti	02.03.2011	12 sivua

Tiivistelmä – Referat – Abstract

Keskusmuistin kasvaessa, koko tietokanta saattaa mahtua sinne. Pullonkaulaksi on muodostunut sen ja prosessorin välinen tiedonkulku. Hutiosumat prosessorin L1- ja L2-välimuisteissa heikentävät kyselyiden suoritusta. Tässä raportissa käsittelee molempien välimuistien hutiosumien ongelmaa, ja niiden vähentämistä. L1-välimuistin hutiosumia pyritään vähentämään optimoijalla, joka tietyssä tilanteessa otetaan käyttöön. Optimoija on uusi operaatio, joka lisätään kyselyiden väliin ennen suoritusta. Se muuntaa suoritusjärjestystä hutiosumia vähentävään suuntaan. Raportissa on myös kokeellisesti todettu optimoijan parantavan suorituskykyä. L2-välimuistin hutiosumiin ratkaisuksi ehdotetaan prosessorien omistautumista tiettyyn tehtävään kerrallaan, ja tarpeeksi suuria aikaviipaleita koko tehtävän suorittamiseksi kerralla. Puskurit sijoitetaan kyselyiden väliin tehtävässä nopeuttaakseen tehtävän suoritusta rinnakkain.

ACM Computing Classification System (CCS):
B.3.2 Design Styles
B.3.3 Performance Analysis and Design Aids

Avainsanat – Nyckelord – Keywords

prosessorin välimuistit, puskurit, hutiosumien vähentäminen

Säilytyspaikka – Förvaringställe – Where deposited

Muita tietoja – Övriga uppgifter – Additional information

Sisältö

1 Johdanto

Keskusmuistien hintojen lasku on mahdollistanut suurten keskusmuistien käytön. Käyttäjät pystyvät suorittamaan suuria tietokantaoperaatioita kokonaan keskusmuistissa. Keskusmuistissa suorittaminen on tehokkaampaa kuin levyltä luettaessa, koska tieto on lähempänä tiedon prosessointi paikkaa eli prosessoria. Prosessorit ovat hyvin nopeita, ja pullonkaulaksi tietokantaoperaatioiden suorittamisessa on muodostunut muistin ja prosessorin välinen tiedonvälitys. Keskusmuistia nopeampi tiedon säilytyspaikka on prosessorin oma välimuisti, mutta se ei ole niin suuri kuin keskusmuisti. Ongelman ratkaisuksi tutkijat ovat suunnitelleet tietokantaoperaatioista välimuistitietoisia, jolloin ne osaavat käyttää prosessorin sisäistä välimuistia tehokkaasti hyväksi lataamalla usein tarvittavat tiedot välimuistiin operaation suorituksen ajaksi.

Cieslewicz ja kumppanit artikkelissaan [CMR09] tuovat esiin ongelman, jossa useampi eri operaatio suorittaa laskentaa prosessorilla tai prosessoreilla käyttäen jaettua välimuistia (shared cache). Tällöin välimuistin hutiosumat (cache miss) lisääntyvät, koska aikaviipaleen (time slice) käytettyään operaation käyttämä välimuisti on toisen operaation käytössä. Ensimmäisen operaation jatkaessa, joutuu se hakemaan tiedot uudestaan keskusmuistista asti. Ongelma ilmenee vaikka molemmat operaatiot olisivat suunniteltu välimuistitietoiseksi eristyksessä, mutta operaatioiden yhteistyö ei toimi tehokkaasti.

Välimuistin hutiosumia lisäksi tapahtuu, koska välimuistit ovat liian pieniä tietokantaoperaatioiden tiedoille. Varsinkin L1-välimuistissa, joka on lähempänä prosessoria kuin suurempi L2-välimuisti ja mahdollinen L3-välimuisti. Prosessori tutkii ensin L1-välimuistin tietoja tai ohjeita (instructions) hakiessaan tietoa. Jos se epäonnistuu, tietoja yritetään hakea L2-välimuistista ja sitten L3-välimuistista tai keskusmuistista. Tiedon haun kustannus keskusmuistista on kymmenkertainen verrattuna L2- tai L3-välimuistia [ZhR04]. Zhou ja Ross artikkelissaan [ZhR04] ehdottavat L1-välimuistin ongelman ratkaisuksi puskurointioperaatioita, joka muuttaa tietokantaoperaatioiden suoritusjärjestystä. Tsuji ja kumppanit artikkelissaan [TKT10] ovat huomanneet, että puskurointioperaatio vähentää L1-välimuistin hutiosumia vain tietyissä tilanteissa ja jopa lisää niitä toisissa. He esittelevät Zhoun ja Rossin esityksen pohjalta uuden algoritmin, joka valitsee oikean optimoinnin tietokantaoperaatioille riippuen tilanteesta.

Loppu raportti etenee seuraavasti: Luvussa kaksi esittelen L1-välimuistin puskuroinnin

ongelman, ja miksi se pitää ratkaista. Luvussa kolme käsittelen Tsujin ja kumppanien ratkaisuehdotuksen ongelmaan. Luvussa neljä esittelen L2-välimuistin ongelmaa sekä Cieslewiczin ja kumppanien ratkaisuehdotuksen. Luvussa viisi käsittelen Cieslewiczin ja kumppanien ratkaisuehdotuksen kokeellisia tuloksia. Luvussa kuusi on yhteenveto raportista.

2 L1-välimuistin puskuroinnin ongelma

Prosessoria lähemmän eli L1-välimuistin hutiosumien haitta on pienempi kuin L2-välimuistin, mutta niiden tapahtuminen on yleisempää [TKT10]. Zhou ja Ross artikkelissaan [ZhR04] ehdottaa ratkaisuksi puskurointioperaatioita, joka asetetaan kahden tietokantaoperaation väliin. Puskurointioperaatio on toteutettu PostgreSQL-järjestelmälle. Oikealle tietokantajärjestelmälle toteutettu ratkaisu ei ole järkevä, koska operaatio parantaa suoritusta vain osassa tapauksia [TKT10]. Puskurointioperaatio perustuu tietoon, että L1-välimuistin koko ei riitä aina kyselyn prosessoinnille. Kun näin tapahtuu, tapahtuu hutiosuma. Esimerkiksi kysely:

SELECT COUNT(*) FROM item,

tarvitsee kaksi tietokantaoperaatiota, selauksen (tablescan) ja koosteen (aggregation). Jokaisella koostamiskäsittelyllä operaatio hakee monikon selausoperaatiolta. Koostamis- ja selausoperaatiolla on vanhempi-lapsi-suhde, missä koostamisoperaatio on selauksen vanhempi. Merkitään koosteoperaatiota kirjaimella P ja selausoperaatioita kirjaimella C. Kahdeksan operaation suorituksen jälkeen järjestys on seuraava: PCPCPCPCPCPCPCPC. Jos yhden operaatioparin PC kokonaiskoko on suurempi kuin L1-välimuistin koko, molemmat operaatiot synnyttävät hutiosuman. Zhoun ja Rossin [ZhR04] ehdotuksessa puskurointioperaatio puskuroi osoittimen (pointer) jälkimmäisen operaation tietoon, jolloin välimuistissa tarvittava tieto pienenee. Puskurointi ei vaikuta normaalien operaatioiden suoritukseen, ja ne pystyvät operoimaan keskeytyksettä. Nyt kahdeksan operaation suorituksen jälkeen järjestys on: PCCCCCCCCPPPPPPP. Sama operaatio suoritetaan peräjälkeen, ja sen tiedosta tallennetaan välimuistiin vain osoitin. Puskurointioperaation hyöty tulee esiin vain, jos operaatiot P ja C ovat liian suuria yhdessä välimuistiin, muussa tapauksessa operaatio heikentää suoritusaikaa [TKT10].

Jotta puskurioperaatiosta saadaan täysi hyöty, tarvitsee se kyselyn optimoijan. Optimoijan tulee valita kyselyä suorittaessa tarvitaanko puskurointioperaatiota käyttää vai ei

perustuen kyselyyn. Optimoijan kehittämisellä on kaksi ongelmaa, jotka tulee ratkaista.

1. Optimoijan tulee tehdä syvä puskurioperaation käyttäytymisanalyysi. Optimoijan tulee tietää milloin puskurioperaation käyttö on hyödyllistä ja milloin ei. Rajat tulee selvittää useilla eri attribuuteilla.
2. Kyselyn optimoijan realisointi puskurioperaatiolla tulee perustua syväanalyysiin.

Tsuji ja kumppanit artikkelissaan [TKT10] ratkaisevat nämä ongelmat PostgreSQL-järjestelmään. He toteuttavat puskurioperaation uutena operaationa, joten järjestelmän omat operaatiot eivät ole muuttuneet. Puskurioperaation suorittaa käynnistäjämoduuli (executor module), ja se toimii kuten perinteiset tietokantaoperaatiot. Puskurioperaatio on Zhoun ja Rossin [ZhR04] kehittämän operaation kaltainen. Puskurioperaatiolla on taulukko (array), joka eroaa Zhoun ja Rossin toteutuksesta siten ettei se ole määrätyn mittainen vaan optimoijan määräämä. PostgreSQL-rajapinta tarjoaa open-next-close-funktiot. Next-funktio toimii ilman puskurioperaatioita niin, että lapsioperaatioita suoritetaan kunnes taulukko on täysi tai lapsioperaatio ei palauta monikkoa. Funktio asettaa taulukkoon osoittimet lapsioperaation monikoihin. Jos taulukko ei ole tyhjä, next-funktio antaa osoittimet vanhempioperaatiolle. Puskurioperaatiolla lisättynä next-funktio tyhjän taulukon tapauksessa suorittaa lapsioperaatioita keskeytymättä. Jos sen sijaan taulukko ei ole tyhjä, suoritetaan vanhempioperaatiota jatkuvasti. Molemmissa tapauksissa L1-välimuistin hutiosumia ei esiinny, koska välimuisti on kokoajan pelkästään yhden operaation käytettävissä.

3 L1-välimuistin puskuroinnin kokeelliset tulokset

Tsuji ja kumppanit [TKT10] analysoivat puskurioperaation tuloksia Pentium 4 – prosessorilla varustetussa Linux-tietokoneessa, jossa keskusmuistia oli 1GB. L1-välimuistin ohjeille tilaa oli 8KB-16KB, L1-tietovälimuistia oli 8KB ja L2-välimuistia 512KB. Tutkijat tarkastelivat puskurioperaation toimintaa kolmella eri tutkimuksella.

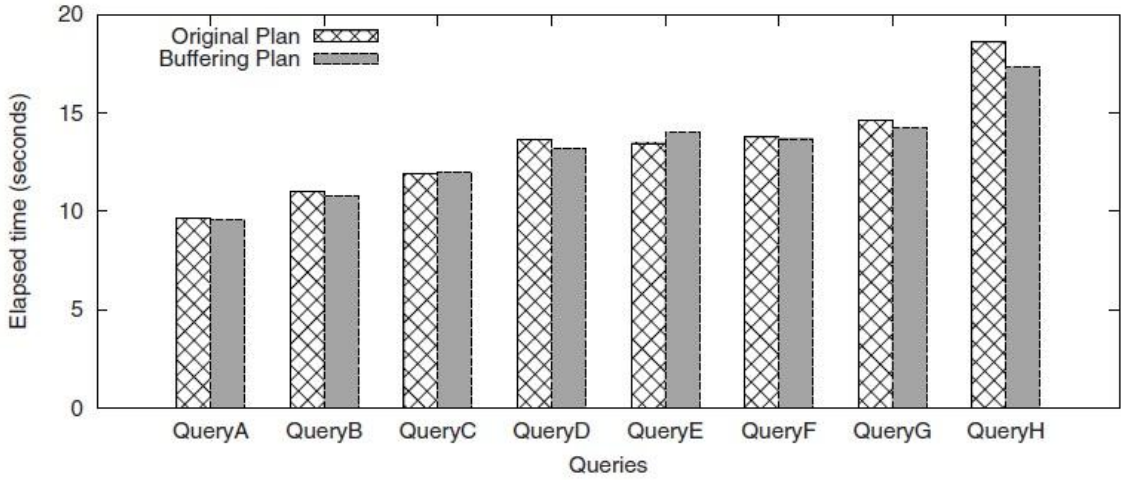
1. Vanhempi- ja lapsioperaation kokonaistilan tarve,
2. puskurointioperaatioiden määrä L2-välimuistissa,
3. puskurointioperaatioiden suoritusten määrä.

Tutkijat suorittivat ensimmäisen tutkimuksen useilla eri kyselyillä, ja he tutkivat olosuhteita missä puskurointioperaatio on hyödyllinen. Vain vanhempi- ja lapsioperaatioiden

kokojen yhteen lisääminen ei riitä, koska operaatioiden koot kääntämisen jälkeen riippuvat laitteistosta, sekä eri operaatioiden moduulit voivat jakaa samoja funktioita. Ensimmäisen tutkimuksen ensimmäisessä vaiheessa tutkijat käyttivät suurikokoisia operaatioita varmistuakseen, että välimuistin hutiosumia tapahtuu ja puskurioperaatio on hyödyllinen kun suoritettavien operaatioiden yhteiskoko ylittää L1-välimuistin koon. Tutkimuksessa Tsuji ja kumppanit [TKT10] vertasivat kooste- ja järjestämisoperaatioiden suoritusaikoja. He käyttivät kahdeksaa eri kyselyä, jotka ovat esitetty kuvassa 1.

ID	SQL
A	SELECT COUNT(*) FROM points WHERE id $<$ 5000000 AND point $\geq$ 0
B	SELECT COUNT(*), AVG(point) FROM points WHERE id $<$ 5000000 AND point $\geq$ 0
C	SELECT COUNT(*), AVG(point), SUM(id) FROM points WHERE id $<$ 5000000 AND point $\geq$ 0
D	SELECT COUNT(*), AVG(point), SUM(id), SUM(id/2) FROM points WHERE id $<$ 5000000 AND point $\geq$ 0
E	SELECT COUNT(*), AVG(point), AVG(id), SUM(id/2) FROM points WHERE id $<$ 5000000 AND point $\geq$ 0
F	SELECT COUNT(*), AVG(point), max(point/2), SUM(id/2) FROM points WHERE id $<$ 5000000 AND point $\geq$ 0
G	SELECT COUNT(*), AVG(point), AVG(id), SUM(id/2), max(point/2) FROM points WHERE id $<$ 5000000 AND point $\geq$ 0
H	SELECT COUNT(*), AVG(point), AVG(id), AVG(id/2), AVG(point/2), AVG(id/3) FROM points WHERE id $<$ 5000000 AND point $\geq$ 0

Kuva 1. Ensimmäisessä tutkimuksessa käytetyt kyselyt [TKT10].



Kuva 2. Ensimmäisen tutkimuksen tulokset [TKT10].

Kuvassa 2 on esitetty palkeilla jokaisen kuvassa 1 esitetyn kyselyn suoritusaika ilman puskurioperaatiota ja sen kanssa. Koosteoperaation kokoon vaikuttaa sen monimutkaisuus, ja tutkijat ovat valinneet kyselyiksi erikokoisia koosteoperaatioita. Tulosten johtopäätökset ovat seuraavat: Puskurioperaatiosta on hyötyä kun kyselyssä on kolme tai enemmän koosteoperaatioita sisältäen keskiarvon ja summan koostamisen tai laskettiin vain viiden tai enemmän monikon keskiarvoa tai summaa. Puskurioperaatiosta oli vain joskus hyötyä, kun laskettiin keskiarvoa ja summaa.

Järjestysoperaation tutkimiseen Tsuji ja kumppanit [TKT10] käyttivät kyselyä:

SELECT * FROM points WHERE id < 5000000 AND point >= ORDER BY point DESC.

Kyselyn alkuperäinen suoritusaika oli 91.63 sekuntia, ja vastaava puskurioperaatiolla 90.93 sekuntia. Puskurioperaation tuottama hyöty oli siis 0.76 %. Kun järjestysoperaatio oli lapsioperaationa, suoritusaika kasvoi.

Tutkijat käyttivät pienikokoisten operaatioiden tutkimiseen seuraavaa kyselyä:

SELECT * FROM names n, (SELECT COUNT(*) FROM points GROUP BY point) AS t WHERE t.count = n.id.

Kyselyn alkuperäinen suoritusaika oli 184.6 sekuntia, ja puskurioperaatiolla 186.9 sekuntia. Puskurioperaation käyttäminen heikensi suorituskykyä 1.25 %.

Toinen Tsujin ja kumppanien [TKT10] tekemä tutkimus käsitteli puskurioperaatioiden määrää L2-välimuistissa. Puskurioperaatioiden lisääntyessä L1-välimuistin hutiosumat vähentyvät, mutta jos puskurioperaatio ei mahdu L2-välimuistiin tapahtuu siellä hutiosuma. Tutkijat testasivat puskurioperaatioiden määrän vaikuttamista suorituskykyyn lisäämällä ensin yhden ja sen jälkeen useita puskurioperaatioita suunnitelmapuuhun (plan tree). Yhden puskurioperaation tapauksessa tutkijat lisäsivät kyselyyn neljä koosteoperaatiota, jotta L1-välimuisti täyttyisi. Kyselyn suoritusaika parani kun puskurinkoko oli yli sata ja alle 2500. L1-välimuistin hutiosumat vähentyivät puskurikoon lisääntyessä, ja L2-välimuistin hutiosumat lisääntyivät kun puskurikoko kasvoi yli tuhannen. Useiden puskurioperaatioiden tapauksessa tutkijat käyttivät kyselyä, jossa oli kaksi liitosoperaatioita (join operator). Kyselyn suorituskyky parani, kun puskurikoko ylitti sadan.

Kolmannessa tutkimuksessaan Tsuji ja kumppanit [TKT10] tutkivat puskurioperaatioiden suoritusten määrää. Vaikka puskurioperaatio on tehokas, sen käyttäminen kuluttaa resursseja. L1-välimuistin hutiosumien vähentäminen pitäisi tapahtua kohtuullisella kustannuksella. Tutkimuksen kysely oli:

SELECT COUNT (*) FROM item.

Tuloksista selviää, että kun laskettavien monikoiden määrä ylittää 600, alkuperäisen kyselyn suorituskyky laskee. L1-välimuistin hutiosumat puskurioperaatiolla vähenivät verrattuna alkuperäiseen suunnitelmaan kun laskettavien monikoiden määrä ylitti 300, ja L2-välimuistin hutiosumat alkoivat vähentyä monikoiden ylittäessä 500.

Saaduista tuloksista Tsuji ja kumppanit ehdottavat kyselyn optimoijaa, joka lisää pusku-

rioperaation suunnittelupuuhun kun sen käyttö on hyödyllistä. Se on hyödyllistä vain silloin kun käsiteltäviä monikoita on yli 500 ja ainakin yksi seuraavista ehdoista pätee:

a) vanhempisolmu (parent node) sisältää enemmän kuin kolmen tyyppisiä kooste-operaatioita, ja lapsisolmu (child node) on päämoduuli poislukien järjestysoperaatio,

b) vanhempisolmu on liitosoperaatio, ja lapsisolmut ovat päämoduuleja poislukien järjestysoperaatio,

c) vanhempisolmu on järjestysoperaatio, ja lapsisolmu on päämoduuli.

4 L2-välimuistin puskurointi

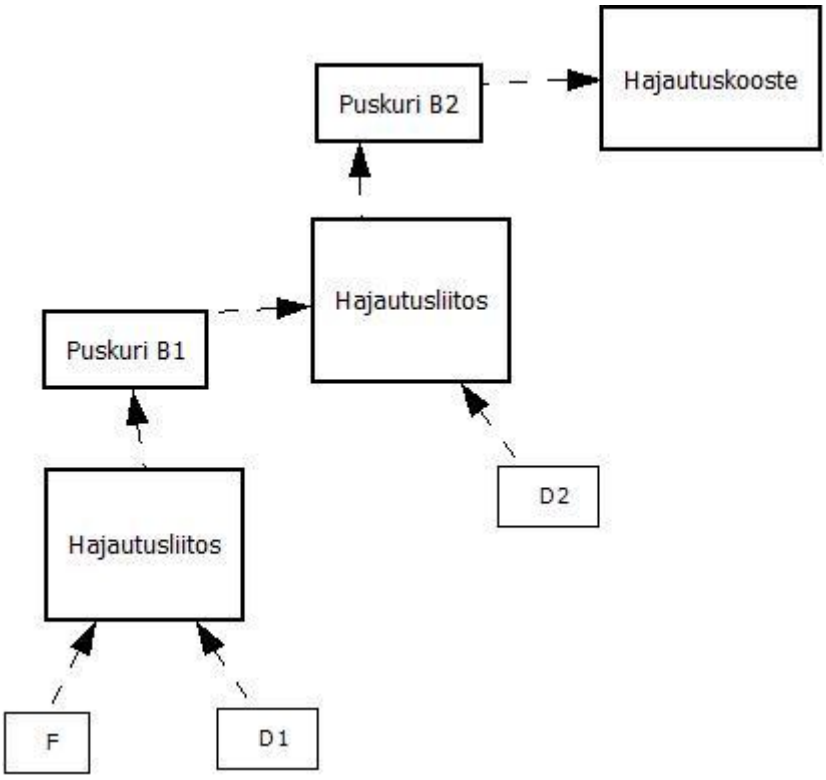
Cieslewicz ja kumppanit artikkelissaan [CMR09] tutkivat L2-välimuistin hutiosumia erityisesti kun useampi kuin yksi operaatio on suorituksessa. Yksittäiset operaatiot ovat hiottu käyttämään välimuistia tehokkaasti, mutta usean operaation tapauksessa yhteistyö ei toimi. Esimerkiksi suoritetaan operaatiot A ja B. Molemmat operaatiot tarvitsevat suuren tietorakenteen, kuten hajautustaulun. Jos operaation A tietorakenne mahtuu välimuistiin, muutaman suorituksen jälkeen sen tietoa pidetään siellä pysyvästi. Se parantaa suorituskykyä, koska tiedon haku ei kestä niin kauan kuin keskusmuistista. Mutta jos operaation B suoritus lomittaa operaation A suorituksen, B:n tietorakenne vie välimuistista A:n tietorakenteen tilan, ja hutiosumia esiintyy.

Tietokantaoperaation hutiosumien määrä on sen aliprosessien hutiosumien kooste. Esimerkiksi järjestämisoperaatio lukee syötteen, suorittaa järjestelyn ja kirjoittaa tulosteen. Jokainen operaation vaihe käyttää omaa tietorakennetta, ja niillä on eri hutiosumat. Operaation suorituksen kuluessa sen tietoja jätetään välimuistiin hakuajan pienentämiseksi. Hutiosumien esiintyvyys on joko nopeasti häviävää, hitaasti häviävää tai ne eivät häviä ollenkaan. Nopeasti häviävä on pieni tietorakenne, hitaasti on suuri välimuistiin mahtuva tietorakenne ja häviämättömällä ei ole väliaikaista sijaintia toisin sanoen se ei mahdu välimuistiin ollenkaan. Kaikkien operaatioiden tulee saada aikaviipale reilusti ja kohtuullisella viiveellä. Aikataulualgoritmit voivat tasapainottaa viivettä ja suoritustehoa tarpeeksi pienillä aikaviipaleilla [CMR09]. Käyttöjärjestelmissä aikaviipale on yleisesti 10-100ms [SGG03], mutta Cieslewicz ja kumppanit [CMR09] käyttävät pienempiä aikaviipaleita operaatioille. Esimerkiksi nopeasti ja hitaasti häviävät hutiosumat suoritetaan tarpeeksi pitkässä aikaviipaleessa, jotta hutiosumien määrä häviää suurimmassa

osassa monikoita. Liian lyhyellä aikaviipaleella hutiosumia esiintyy aina uudestaan ja uudestaan samoilla monikoilla.

Cieslewicz ja kumppanit [CMR09] esittävät ratkaisuksi tarpeeksi suuria suoritusjoukkoja (batch of work), jotta sama operaatio ehtii suorittamaan mahdollisesti koko kyselynsä. Tutkijat lisäävät kyselysuunnitelmaan (query plan) puskureita, joihin talletetaan tuloksia käytettäväksi operaation syötteenä. Estämättömille (nonblocking) eli lukkimattomille operaatioille puskuriin tallennetaan myös operaation tulostus. Jokainen operaatio aikataulutetaan suorittamaan sen koko syötepuskuri kerralla. Moniytimisellä prosessorilla kaikki säikeet suorittavat samaa operaatioita aikaviipaleen ajan, jotta säikeet voivat käyttää jaettua L2-välimuistia kokonaisuudessaan.

Esimerkiksi kysely, joka suorittaa viiteavaimen perusteella hajautusliitoksia (hash join) taulun F, D1 ja D2 kanssa. F on faktataulu (fact table) ja D1 sekä D2 ovat dimensiotauluja. Kysely tarvitsee kaksi liitosta ja koosteen. Ensimmäiseen liitokseen ei tarvita puskuria, koska F on kantarelaatio (base relation). Liitoksessa on tulostepuskuri B1, jota käytetään toisen liitoksen syötepuskurina. Toisen liitoksen tulostepuskuri B2 toimii koosteen syötepuskurina. Kuvassa 3 on esitetty tiedon kulku esimerkki kyselyssä.



Kuva 3. Kyselysuunnitelma puskureilla.

Ensimmäisen liitoksen tehokkuus riippuu puskurin B1 koosta, ja kuinka hyvin se pystyy

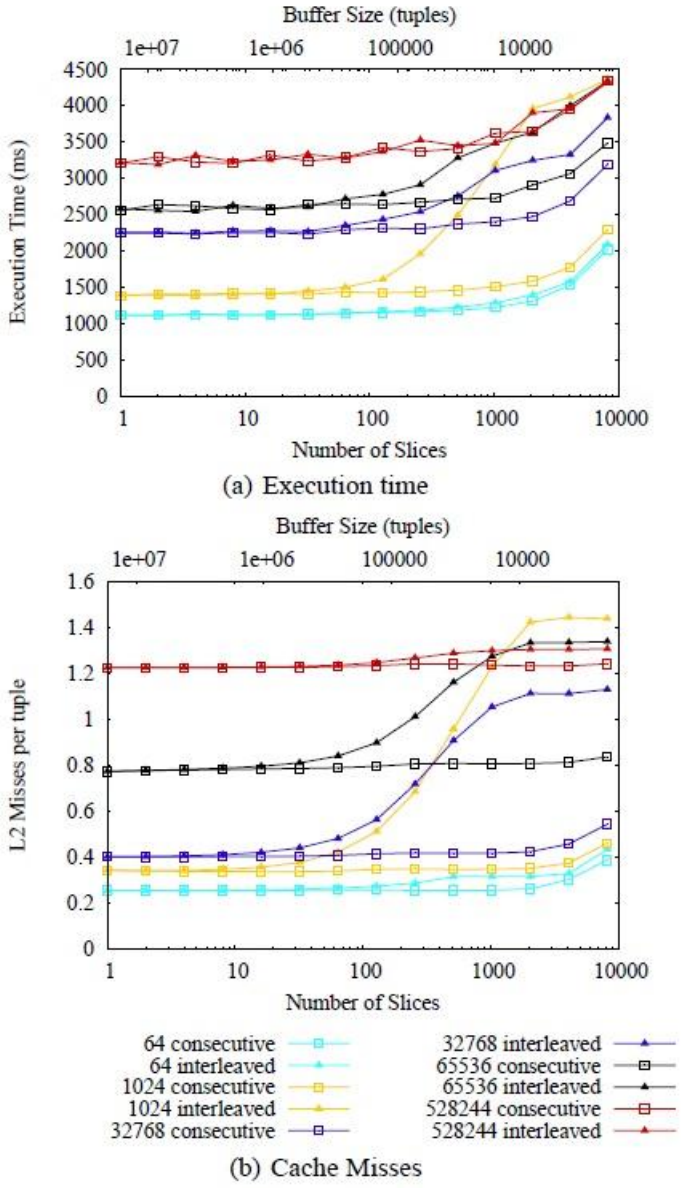
vähentämään taulusta D1 lukemisen kustannusta. Toisen liitoksen tehokkuus riippuu molemmista puskureiden B1 ja B2 koista. Kooltaan pienempi puskuri on rajoite kuinka monta kertaa operaatioita voi ajaa yhdessä suorituksessa. Koosteen tehokkuus riippuu puskurin B2 koosta, ja miten hyvin se pystyy vähentämään hakukustannuksia hajautustaulun avaimiin ja arvoihin. Jotta kokonaissuoritus olisi tehokasta, pitää ainakin yhden operaation olla aina valmiina aikataulutukseen. Operaatio on valmiina aikataulutukseen, kun sen käyttämä syötepuskuri (jos olemassa) on ainakin puoliksi täynnä, ja sen käyttämä tulostepuskuri (jos olemassa) on ainakin puoliksi tyhjä. Monen operaattorin ketjussa kuten kuvassa 3, tämä on aina voimassa koska: Jos ensimmäinen operaatio ei ole valmis on sen tulostepuskuri ainakin puoliksi täynnä. Induktiolla saadaan, että jos jokin operaatio O_i ei ole valmis, on sen tulostepuskuri B_{i+1} ainakin puoliksi täynnä (i = 1,.,n-1). Jos siis tulostepuskuri B_n on enemmän kuin puoliksi täynnä on sitä seuraava operaatio O_n valmis aikataulutukseen. Tästä seuraa, että joukkosuoritus voi olla suuri, eikä sen sisällä tapahdu odottelua.

5 L2-välimuistin puskuroinnin kokeelliset tulokset

Cieslewicz ja kumppanit [CMR09] tutkivat puskurin toimintaa L2-välimuistin hutiosumien määrän avulla niin sanotulla musta laatikko -tyylillä (black-box fashion), jolloin ulospäin ei näy miten hutiosumat tapahtuvat. He käyttivät tutkimiseen Sunin T1000 palvelinta, jossa on 32 säiettä neljällä UltraSPARC T1 prosessorilla. Yksi säie vastaa operaatioiden aikataulutuksesta sekä tilastojen keräämisestä. Muut 31 säiettä ovat omistettuina aikataulutetulle operaatioille kokonaan. Operaatio suoritetaan rinnakkain, jotta säikeet voivat käyttää tehokkaasti yhteistä jaettua L2-välimuistia. Operaation suoritus ei myöskään riipu muista järjestelmän tapahtumista. Tutkijat tarkastelivat kooste- ja hajautusliitosoperaatioita (hashjoin) neljässä eri kokeessa. Ensimmäisessä kokeessa he testasivat hutiosumien esiintymisen välimuistissa kahden yhtä aikaa ajettavan operaation aikana, toisessa puskurin koon vaikutusta hutiosumiin, kolmannessa operaation syötteen eri jakautumista välimuistiin ja neljännessä hajautusliitoksen hutiosumien määrää.

Ensimmäisessä kokeessa tutkijat käyttivät 2^{24} (noin 16 miljoonaa) monikkoa syötteenä kahdeksalle samanlaiselle koostekyselylle. Prosessointi jaettiin viipaleihin, joiden koko oli monikoiden määrä jaettuna puskurin koolla. 31 säiettä oli omistettu yhdelle viipaleelle kerrallaan. Ensimmäisellä kerralla kaikki viipaleet käsiteltiin yksi kerrallaan lop-

puun asti ennen seuraavaan siirtymistä. Toisella kerralla viipaleiden suorituksien annettiin olla päällekkäisiä. Ensimmäisen kerran tapauksessa välimuistiin haetut tiedot olivat samoja, jos viipaleet kuuluivat samaan kyselyyn. Toisen kerran tapauksessa välimuistissa ei todennäköisesti ole valmiina oikeita tietoja, koska päällekkäiset suoritukset sekoittavat tietoa välimuistissa. Tutkimuksessa mitattiin suoritusaika ja hutiosumat. Kuvassa 4 on esitetty tulokset tutkimuksesta. Neliöllä varustetut viivat ovat peräkkäin suoritetut viipaleet ja kolmiolla päällekkäin suoritetut. Tutkimuksessa käytettiin eri määriä monikoita yhdessä viipaleessa, ja ne on kuvattu eri väreillä. Suurilla puskurikooilla ei eroa synny, mutta pienemmillä hutiosumien määrä ja suorituskyky kasvavat päällekkäisillä kyselyillä. Varsinkin kun monikoita oli 1024 (kuvassa keltainen), ero on huomattava.



Kuva 4. Viipaleiden määrän vaikutus suorituskykyyn ja hutiosumiin [CMR09].

Toisessa kokeessa tutkijat halusivat katsoa onko toiset operaatiot herkempiä puskurin koolle kuin toiset. Kokeessa käytettiin kymmentä 64 monikon ja kymmentä 1024 monikon koostekyselyä. Edellisen kokeen perusteella 64 monikon kyselyt ovat tehokkaita riippumatta puskurinkoosta, kun sen sijaan puskurinkoko vaikuttaa suuresti 1024 monikon kyselyihin. Tutkijat antavat 64 monikon kyselyille 8192 monikon puskurikoon, joka on pienin mahdollinen hyötyä antava koko. Loppu muisti annetaan 1024 monikon kyselyille. Tämä paransi kokonaissuoritusta 3.3 % - 8.4 %. Jos kaikki kyselyt tarvitsevat muistia, on tasajako toimiva ratkaisu. Jos vain osa tarvitsee muistia, on se suorituskykyä heikentävää antaa sitä kaikille. Kokeen johtopäätös on että tasajako on hyvä lähtökohta, mutta 8.4 % lisähyöty on saavutettavissa jakamalla muistia sitä tarvitseville.

Kolmannessa kokeessa Cieslewicz ja kumppanit [CMR09] tutkivat kyselyn syötteen jakautumisen vaikuttamista operaatioon. Esimerkiksi jos koostekyselyn arvot ovat yleisiä, tarvitsee kysely enemmän väliaikaista paikallisuutta (temporal locality) eli välimuistia saavuttaakseen parhaan tehokkuuden. Tutkijat suorittivat koostealgoritmin eri jakeluilla, jotka on kuvattu Cieslewiczin ja Rossin aikaisemmassa artikkelissa [CiR07]. He mittasivat hutiosumat ja suorituskyvyn eri puskurikoilla. Tulokset kertovat, että päällekkäin suoritetuissa kyselyissä esiintyy huomattavasti enemmän hutiosumia kun viipaleiden määrä ylittää sadan kaikilla muilla kuin aivan pienillä monikon määrillä. Hutiosumien esiintyminen sen sijaan vaikuttaa suorituskykyyn alentavasti, ja päällekkäin suoritettujen kyselyiden suoritusajat ovat sadan viipaleen jälkeen heikompia.

Neljännessä ja viimeisessä kokeessa tutkijat tarkastelivat hajautusliitosoperaatioita. Käytetty kysely on viiteavaimen mukainen liitos, jossa liittävä monikko liittyy vain ja ainoastaan yhteen monikkoon liitettävässä taulussa. Liittävän taulun kokoa muutettiin, mutta liitettävä taulu oli tasan liitettävien monikoiden kokoinen. Hajautustaulu, jossa on noin 150 000 monikkoa, mahtuu juuri L2-välimuistiin. Kokeen tuloksena liitos vahvisti koostekokeen tuloksen. Suorituskyky parani keskisuurilla monikko määrillä, kun aikaviipaleita oli yli sata. Vaikka liitoskokeessa hutiosumien määrä pieneni, ei se vaikuttanut niin suuresti suorituskykyyn. Välimuistiin mahtuva tietorakenne on tehokkaampi puskurioperaatiolla vain, jos se mahtuu välimuistiin ja se prosessoi tarpeeksi montaa monikkoa. Pienillä monikko määrillä ei puskuroinnilla ole vaikutusta, koska kaikki tieto mahtuu välimuistiin.

6 Yhteenveto

Prosessorien L1- ja L2-välimuistien hutiosumat kasvattavat suoritettavan kyselyn suoritusaikaa, koska tiedon hakuun kestää enemmän aikaa. Tsuji ja kumppanit artikkelissaan [TKT10] esittävät ongelman L1-välimuistin hutiosumista. He esittelevät Zhoun ja Rossin optimoijaan [ZhR04] parannuksia, sekä kokeellisesti testaavat sitä. Testien perusteella he esittelevät oman optimoijan, joka toisin kuin Zhoun ja Rossin optimoija, ei ole käytössä joka kyselyllä. Heidän optimoijalla on tutkimukseen perustuvia ehtoja, joiden perusteella käyttöönotto valitaan.

Cieslewicz ja kumppanit artikkelissaan [CMR09] keskittyvät sen sijaan L2-välimuistin hutiosumiin. He kiinnittävät huomiota kyselyiden päällekkäiseen suoritukseen, milloin kyselyiden tietoalkiot ovat samassa välimuistissa. Tällöin välimuistissa ei aina ole yhden kyselyn tarvitsemia tietoja ja syntyy hutiosuma. Vaikka kyselyt ovat optimoitu käyttämään välimuistia tehokkaasti eristyksissä, ei suoritus ole tehokasta useamman kyselyn ollessa suorituksessa rinnakkain. Cieslewicz ja kumppanit [CMR09] ratkaisevat ongelman puskuroinnilla kyselyiden välissä, sekä tarpeeksi suurilla aikaviipaleilla jotta yksi kysely ehtii suorittamaan itsensä loppuun. Tällöin kaikki mahdolliset prosessorit suorittavat vain yhtä tehtävää, ja jaettu välimuisti on kokonaan tämän tehtävän käytettävissä. Koska jaetussa välimuistissa on vain yhden tehtävän tietoja, siellä on todennäköisemmin tehtävän tarvitsemia tietoja. Se tarkoittaa vähemmän hutiosumia.

Molempien välimuistien hutiosumien vähentäminen kokeiden perusteella lisäsi tehokkuutta muutamia prosentteja, ja siten helpotti pullonkaulaa joksi keskusmuistin ja prosessorin välinen tiedonkulku on muodostunut [CMR09].

Lähteet

CiR07 John Cieslewicz ja Kenneth A. Ross: Adaptive Aggregation on Chip Multiprocessors. VLDB 2007, sivut 339-350.

CMR09 John Cieslewicz, William Mee ja Kenneth A. Ross: Cache-Conscious Buffering for Database Operators with State. Damon 2009, sivut 43-51.

SGG03 Abraham Silberschatz, Peter Baer Galvin ja Greg Gagne: Operationg System Concepts. John Wiley & Sons, Inc., 6. painos, 2003.

TKT10 Yoshishige Tsuji, Hideyuki Kawashima ja Ikuo Takeuchi: Optimization of Query Processing with Cahce Conscious Buffering Operatior. DNIS 2010, sivut 65-79.

ZhR04 Zhou, J. Ross, K.A: Buffering Database Operations for Enhanced Instruction Cache Performance. SIGMOD 2004, sivut 191-202.