Rinnakkaishakujen tehokas toteuttaminen keskusmuistitietokannoissa

Lauri Malmi
Teknillinen korkeakoulu, Tietojenkäsittelyopin laboratorio

28. huhtikuuta 1997

1 Johdanto

RAM-muistin hintakehitys kymmenen viime vuoden aikana on muuttanut rajusti ohjelmistokehittäjien toimintamahdollisuuksia. Myös tietokantasovellusten suorituskykyä on tullut mahdolliseksi parantaa huomattavasti sijoittamalla merkittävä osa käsiteltävästä tiedosta keskusmuistitiedostoihin. Tämä ei suinkaan merkitse levytiedostojen syrjäytymistä kokonaan, koska niitä tarvitaan joka tapauksessa tiedon varmentamiseen ja järjestelmän käynnistämiseen. Suuria, satojen tai tuhansien megatavujen tietomääriä käsiteltäessä kustannustekijä myös painaa toistaiseksi vaa'an levypohjaisen järjestelmän puolelle.

Tietorakenne- ja algoritmitutkimuksen kannalta kehityksellä on mielenkiintoisia vaikutuksia. Esimerkiksi hakemistorakenteiden kohdalla nousee enemmän esiin kysymys, mikä tai mitkä rakenteet soveltuvat parhaiten suurten keskusmuistihakemistojen toteuttamiseen. Jotta tähän voidaan vastata, on tarvetta selvittää perusteellisemmin algoritmien kompleksisuusfunktioiden vakiokertoimia. Keskusmuistissa toimivia hakemistorakenteita, kuten taulukkoja, listoja, erilaisia puita ja hajautusrakenteita on kyllä 1950-luvulta lähtien tutkittu vilkkaasti[1], mutta ainakin puurakenteiden osalta tutkimustyö on keskittynyt teoreettiseen analyysiin sekä pahimman tapauksen että keskimääräisen suorituskyvyn osalta. Vakiokertoimia koskevia vertailuja sekä kokeellisia tuloksia on varsin vähän. Se on ymmärrettävää, koska asian monimutkaisuuden tähden on vaikeaa taata tulosten yksiselitteisyys ja yleispätevyys. Kuitenkin käytännön sovellusten kannalta on tarpeen esittää esimerkiksi seuraavankaltaisia kysymyksiä: “Olen päätynyt käyttämään hakemistorakenteena tasapainotettua puuta, mutta mikä niistä mahtaa olla tehokkain, AVL-puu, punamusta puu vai jokin muu? Ovatko erot merkittäviä? Entä miten tuloksiin vaikuttaa prosessorivälimuistin olemassaolo ja mahdollinen säätö?”.

Paitsi algoritmien nopeutta joudutaan vertailemaan myös niiden tilantarvetta. Väheneekö sen merkitys vai kasvaako se? Edellistä vaihtoehtoa voidaan puolustaa argumentilla, että monimutkaisemman mutta tilaa säästävän tietorakenteen valitseminen ei kannata, koska koneeseen saa halvalla lisää muistia;

[1] Tehdessäni väitöskirjatyöhöni liittyvää kirjallisuustutkimusta löysin vaivatta reilusti yli 300 pelkästään tasapainotettuja puita käsittelevää artikkeli- tai raporttiviitettä ja selvitys ei ole lähellekään valmis.

jälkimmäistä vaihtoehtoa taas sen kokemuksen perusteella, että kaikki käytössä oleva muistitila loppuu kuitenkin ja käsiteltävät tietomäärät kasvavat nopeammin kuin muisti halpenee. Tilanne näyttää hyvin samanlaiselta kuin algoritmien aikavaatimusten osalla: tehokkaammat prosessorit tarvitsevat tehokkaampia algoritmeja, kun tietomäärät kasvavat.

Kolmas näkökulma on rinnakkaisuus. Keskusmuistitietokannoissa transaktioiden suoritusaika on tietysti paljon pienempi kuin levypohjaisissa ratkaisuissa ja se sinällään voi riittää tehokkuusvaatimusten täyttämiseksi. On kuitenkin oletettavaa, että kasvavat samanaikaisten transaktioiden määrät ja tiukat reaaliaikaisuusvaatimukset tekevät rinnakkaisalgoritmien käyttöönoton tarpeelliseksi monissa sovelluksissa. Oma tutkimustyöni liittyy tähän alueeseen, erityisesti tilanteisiin, joissa hakemistoon kohdistuu päivityksiä huomattavan paljon. Työ on jatkoa Eljas Soisalon-Soinisen (TKK) ja Otto Nurmen (HY) aiheesta tekemille tutkimuksille ja teen sitä yhteistyössä heidän kanssaan.

2 Rinnakkaisuus ja puurakenteiden tasapainotus

Mikäli keskusmuistihakemistossa päädytään puurakenteen käyttöön[2], on ratkaistava kysymys, miten puu pidetään tasapainossa. Tavanomaisia vaihtoehtoja on kolme: luotetaan siihen, että päivitykset eivät vinouta puuta liikaa; tasapainotetaan puu sopivin väliajoin täydellisesti käyttäen jotain algoritmia, joka rakentaa koko puun uudelleen (esim. [3, 7, 13]; tai käytetään jotain tasapainotettua tai itseorganisoituvaa puurakennetta. Viimeksi mainittuja on esitetty tutkimuksessa paljonkin, mutta tunnetuimpia lienevät AVL-puu [1], punamusta puu [4] ja splay-puu [12].

Jos tietorakenteeseen kohdistuviin vaatimuksiin lisätään rinnakkaisuus, tulee em. tasapainotusratkaisuissa vastaan ongelmia. Toiminnan oikeellisuudesta tinkimättä samanaikaisuutta pitäisi sallia mahdollisimman paljon eli prosessien odotusaikojen tulisi olla lyhyitä. Tavanomainen ratkaisu samanaikaisuuden hallintaan on puun solmujen lukitus, jolloin kukin prosessi varmistaa toimintansa oikeellisuuden lukitsemalla muilta prosesseilta pääsyn käsittelemäänsä osaan puuta. Jotta muiden samanaikaisten prosessien toiminta ei liiaksi häiriintyisi, tulisi tarvittavien lukkojen määrän olla mahdollisimman pieni, mielellään vakiomääräksi rajoitettu. Puuta globaalisti muuttavat algoritmit eivät sovellu tähän tarkoitukseen lainkaan. Tasapainotetuissa puissa voidaan puolestaan joutua lukitsemaan koko polku juuresta päivitettävään lehteen, jos tasapainotus tehdään päivityksen jälkeen bottom-up-periaatteella. Rakenteita voidaan tasapainottaa myös ennaltaehkäisevästi top-down-algoritmeilla, jolloin joudutaan lukitsemaan vain pieni osa polusta. Niiden haittana on kuitenkin ainakin punamustissa ja AVL-puissa monimutkaisuus[3] sekä se, että nämä algoritmit tekevät usein tarpeetonta ylimääräistä työtä.

Ongelman ratkaisemiseksi Nurmi, Soisalon-Soininen ja Wood ovat kehittäneet punamustien puiden, AVL-puiden ja B-puiden käsittelyyn uuden periaateratkaisun, ns. *relaksoidun tasapainoituksen*, jossa päivitys- ja tasapainotuso-

[2]En ole käsitellyt digitaalisia puurakenteita.

[3]Itse en ole edes löytänyt AVL-puulle top-down-periaatteella toimivaa poistoalgoritmia.

peraatiot on irrotettu toisistaan [8, 9, 10]. Päivitykset tapahtuvat puussa lähes samaan tapaan kuin alkuperäisissäkin tietorakenteissa. Ne jättävät kuitenkin päivityskohdassa sijaitseviin puun solmuihin tiedon siitä, millä tavalla päivitys on muuttanut puuta. Yksi tai useampi erillinen tasapainotusprosessi käy puuta läpi etsien sellaisia solmuja, joissa ko. tasapainoinformaatio ilmaisee puun olevan paikallisesti epätasapainossa. Löydettyään tällaisen *konfliktin* tasapainottaja pyrkii poistamaan sen muuntamalla puuta ko. solmun ympärillä, tyypillisesti tekemällä jonkin kierron ja/tai tasapainoinformaation muunnoksen. Äärellinen määrä tällaisia *tasapainotusoperaatioita* muuttaa puun takaisin vastaavaksi tasapainotetuksi puuksi eli tasapainottaa koko puun, mikäli uusia päivityksiä ei tapahdu.

Relaksoidun tasapainotuksen soveltuvuus rinnakkaisten haku- ja päivitysoperaatioiden yhteyteen perustuu siihen, että tasapainottaja lukitsee kerrallaan vain pienen vakiomäärän solmuja, jolloin sen tuottama viivytys muille prosesseille on pieni. Aiheen tutkimusta on jatkettu sekä teoreettisella että kokeellisella puolella. Boyar ja Larsen ovat laskeneet ylärajat tarvittavien tasapainotusoperaatioiden määrälle relaksoitujen punamustien puiden eli *värillisten puiden* ja relaksoitujen AVL-puiden tapauksessa [2, 5]. Molemmille puille ne ovat luokkaa $O(Mlog(N+M))$, jossa M on punamustaan tai AVL-puuhun tehtyjen päivitysten määrä ja N puun solmujen määrä.

Omassa työssäni olen kehittänyt värillisille puille algoritmin, joka tasapainottaa koko puun ajassa $O(Mlog(N+M))$, mikäli uusia päivityksiä ei tapahdu [6]. Tavanomaiset binääripuun tasapainotusalgoritmit vaativat aikaa $\Omega(N)$, joten ne toimivat hitaammin, jos $M \ll N$. Kokeellisessa vertailuissa totesin uuden algoritmin toimivan aiemmin esitettyjä algoritmeja nopeammin, kun M oli suuruusluokkaa 10–20% N:stä[4]. Uusi algoritmi perustuu siihen, että jokainen päivitysprosessi merkitsee puuhun sen polun, jota pitkin päivittäjä etenee. Tasapainotusalgoritmin tarvitsee käydä läpi vain nämä polut, koska konflikteja on voinut syntyä vain niille. Asialla on erityisesti merkitystä silloin, jos päivitykset ovat kohdistuneet vain pieneen ja kapeaan osuuteen puuta. Tällöin algoritmi nopeutuu merkittävästi, koska suuri osa näistä merkityistä poluista menee päällekkäin.

Mainittu algoritmi ei sellaisenaan toimi samanaikaisesti muiden prosessien kanssa. Siihen on kuitenkin helposti liitettävissä sopiva lukitusjärjestely, joka mahdollistaa tasapainotuksen yhtä aikaa hakujen ja lisäysten kanssa. Samanaikaisuus poistojen kanssa on vaikeammin toteutettava asia. Kokeellisia tuloksia tällaisen rinnakkaisalgoritmin suorituskyvystä keskusmuistirakenteissa ei vielä ole. Relaksoiduilla B-puilla tehdyissä kokeissa, tosin käyttäen toisia algoritmeja, on sen sijaan raportoitu hyviä tuloksia [11]. Tässä selvityksessä tutkittiin ryhmäpäivitysten toteuttamista tekstitietokannassa samanaikaisesti hakujen kanssa ja todettiin, että relaksoitua B-puuta käytettäessä keskimääräiset ja maksimaaliset samanaikaisten hakuprosessien läpimenoajat putosivat jopa kertaluokilla verrattuna tavallista B-puuta päivittävään top-down-algoritmiin.

[4]Testiympäristönä Macintosh IIfx ja toteutuskielenä Think Pascal.

3 Toteutus

Tutkimustyöni apuvälineeksi olen toteuttanut laajan ohjelmiston, jonka avulla on mahdollista graafisesti muokata erilaisia tasapainotettuja puurakenteita, niiden relaksoituja muunnoksia ja visualisoida tasapainotusalgoritmeja. Välineellä on ollut huomattava merkitys algoritmien kehittelyssä, koska mutkikkaiden rakenteiden piirtely paperille on hyvin työlästä. Samassa yhteydessä ohjelmisto pystyy laskemaan erilaisia algoritmien suorituskykyarvoja, kuten keskusyksikköajan kulutusta, tehtyjen kiertojen määrää, viitattujen solmujen määrää yms. Nykyisellään ohjelma on toteutettu Macintoshilla. Opiskelijatyönä on tekeillä X- ja Unix-ympäristössä toimiva algoritmianimoija, jota voitaisiin käyttää mm. tietorakenteiden ja algoritmien opetuksessa.

4 Jatkotutkimusten tavoitteet

Jatkossa on seuraavia tavoitteita.

1. Tehdä mittauksia kehittämääni tasapainotusalgoritmia käyttävän värillisen puun suorituskyvystä tilanteessa, jossa puussa on useita haku- ja päivitysprosesseja meneillään, verrattuna tasapainotettuihin puihin, kuten punamustaan puuhun, AVL-puuhun ja top-down splay-puuhun vastaavissa tilanteissa. Mielenkiinnon kohteena ovat ainakin yksittäisten haku- ja päivitysprosessien maksimaaliset ja keskimääräiset suoritusajat, sekä kokonaisaika, joka tarvitaan erityyppisten haku- ja päivitysoperaatiojoukkojen läpimenoon. Tasapainotusprosessien määrää ja niille annettua keskusyksikköajan osuutta vaihtelemalla pyritään löytämään parhaat vaihtoehdot eri testikuormilla.

2. Laajentaa tuloksia muihin relaksoituihin puihin ja kehittää uusia relaksoituja tietorakenteita.

Tutkimustyön hyväksikäytöstä on keskusteltu sekä Nokia Telecommunicationsin kanssa että TKK:n tietojenkäsittelyopin laboratoriossa meneillään olevan TEKESin rahoittaman SHADOWS-projektin [14] kanssa. Viimeksi mainitussa kehitetään uudenlaista rinnakkaiseen varjosivutukseen (concurrent shadow paging) perustuvaa tietokantaratkaisua, jossa keskusmuistirakenteilla tulee olemaan suuri merkitys.

Viitteet

[1] G.M. Adel'son-Vel'skii, Y.M. Landis, An Algorithm for the Organisation of Information. *Doklady Akademi Nauk* **146** (1962), pp. 263–266. English translation in *Soviet Math. Dokl.* **3**, pp. 1259–1263.

[2] J. Boyar, K.S. Larsen, Efficient Rebalancing of Chromatic Search Trees, *Proc. of Algorithm Theory – SWAT'92. Lecture Notes in Computer Science* **621**, (Springer-Verlag, 1992).

[3] H. Chang, S.S. Iyengar, Efficient Algorithms to Globally Balance a Binary Search Tree. *Communications of the ACM* **27**:7 (1984), pp. 695–702.

[4] L.J. Guibas, R. Sedgewick, A Dichromatic Framework for Balanced Trees. *Proc. of the 19th Symposium on Foundations of Computer Science* (1983), pp. 8–21.

[5] K. Larsen, AVL Trees with Relaxed Balance. Report DAIMI PB - 428, Computer Science Department, Aarhus University, Denmark (1992).

[6] L. Malmi, Efficient Rebalancing of Tree-Type Main Memory Dictionaries. Report B107, Helsinki University of Technology, Department of Computer Science, Finland (1993).

[7] A. Moitra, S.S. Iyengar, Maximally Parallel Algorithms to Balancing Binary Search Trees. *IEEE Transactions on Computers* **34**:6 (1985), pp. 563–565.

[8] O. Nurmi, E. Soisalon-Soininen, D. Wood, Concurrency Control in Database Structures with Relaxed Balance. *Proc. of Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (1987), pp. 170–176.

[9] O. Nurmi, E. Soisalon-Soininen, Uncoupling Updating and Rebalancing in Chromatic Binary Search Trees. *Proc. of Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (1991), pp. 192–198.

[10] O. Nurmi, E. Soisalon-Soininen, Concurrent Balancing and Updating of AVL Trees. Report B76, Helsinki University of Technology, Department of Computer Science, Finland (1992).

[11] K. Pollari-Malmi, E. Soisalon-Soininen, T. Ylönen, Batch Updates and Concurrency in B-trees. Report B106, Helsinki University of Technology, Department of Computer Science, Finland (1993).

[12] D.D. Sleator, R.E. Tarjan, Self-adjusting Binary Search Trees. *Journal of the ACM* **32** (1985), pp. 652–686.

[13] Q.F. Stout, B.L. Warren, Tree Rebalancing in Optimal Time and Space. *Communications of the ACM* **29**:9 (1986), pp. 902–908.

[14] T. Ylönen, Concurrent Shadow Paging: A New Direction for Database Research. Report B86, Helsinki University of Technology, Department of Computer Science, Finland (1992).