Tosiaikatietokannat

Tiina Niklander

Tosiaikatietokannat seminaariesitelmä 5.2.2002

Helsingin yliopisto
Tietojenkäsittelytieteen laitos
JP 26 (Teollisuuskatu 23)
FIN-00014 Helsingin yliopisto

1 Johdanto

Lähes kaikissa tietojenkäsittelyjärjestelmissä on mukana tietokantoja. Tietokannat tarjoavat järjestelmän muille komponenteille tietojen pysyvän varaston, jonka avulla myös tiedon eheys on säilytettävissä. Tietokantojen käytössä keskeistä on tapahtuman käsite. Tapahtumien atomisuus helpottaa tietokantaa käyttävän ohjelman laadintaa. Ohjelmoija voi olla varma, että hänen laatimansa toimenpidesarja ei jää kesken. Se joko suoritetaan kokonaan tai mitään sen tekemiä muutoksia ei jää pysyvästi tietokannan tietoalkioihin. Tapahtumien suoritusten eristäminen toisistaan helpottaa tiedon eheyden säilyttämistä ja mahdollistaa sarjallistuvuus-käsitteen käytön tietokantateoriassa. Perinteisiä relaatio- ja oliotietokantoja käytetäänkin paljon. Tietokantojen yleisten ominaisuuksien vuoksi tietokantojen käyttö myös osana tosiaikajärjestelmiä on tärkeää. Varsinkin kun tosiaikajärjestelmien rakenne muuttuu yhä monimutkaisemmaksi ja käsiteltävä tietomäärä kasvaa voimakkaasti.

Perinteisillä tietokannoilla on yksi keskeinen heikkous, joka rajoittaa niiden käytettävyyttä tosiaikajärjestelmissä. Perinteinen tietokanta ei tunne käsitettä 'AIKA', eikä käytä sitä toiminnassaan hyväksi. Tottakai tietokanta-alkioihin usein tallennetaan niiden muutosajankohta ja tätä ajankohtaa saatetaan jopa käyttää alkoiden päivitystilanteissa. Tällä ajankohdalla ei kuitenkaan ole varsinaisesti merkitystä alkioiden arvojen käyttämiselle tapahtumissa. Tapahtumien suoritukseenkaan ei liity ajan käsitettä. Toki tapahtumalla on jokin aloitusajankohta, jolloin sen suoritus kannassa on aloitettu. Vastaavasti tapahtuman päättymiselläkin on ajankohta. Näitä ajankohtia ei voi määrätä ennalta, eikä perinteinen tietokanta pysty takaamaan, että tapahtuma suoritetaan tiettynä ajanhetkenä.

Tosiaikatietokannoissa tapahtumilla on aikarajoitteita ja tietoalkioille on määritelty aikasemantiikka (engl *time semantics*) [SSH99]. Tapahtumille voi olla esimerkiksi määrätty määräaika (*deadline*) ja mahdollisesti myös aloitusaika. Toistuvien tapahtumien periodin pituus on kiinnitetty. Tosiaikatietokannan alkioilla on yleensä kelpoisuusjakso (*validity interval*), jonka kuluessa niiden kulloistakin arvoa saa käyttää.

Tapahtumien suorituksessa on niihin liitetyt aikarajoitteet huomioitava, ja kannan on taattava tapahtumien suoritus annettujen aikarajoitteiden puitteissa. Onnistuneesti suoritettuja tapahtumia tosiaikatietokannassa ovat vain tapahtumat, jotka on suoritettu aikarajoitteidensa puitteissa ja joi-

den käyttämät tietoalkiot ovat kaikki olleet kyseisenä suoritusjaksona kelvollisia. Tämä on varsin erilainen vaatimus kuin perinteisissä kannoissa, joissa usein kaikki sarjallistuvat tapahtumat ovat onnistuneet suoritettuja riippumatta niiden suoritusajankohdasta.

Aikavaatimusten mukaan tulo vaikuttaa tietokannanhallintajärjestelmään joka tasolla. Jotta tosiaikainen tietokannanhallintajärjestelmä pystyisi takaamaan tapahtumien suorituksen aikarajojen puitteissa, on kaikkien sen sisäisten toimintojen oltava myös 'aikatietoisia' (engl. *time-congnizant*). Näin ollen siis samanaikaisuudenhallintaprotokollan täytyy hallita myös tapahtumien ja tietoalkioiden aikarajoitukset. Tapahtumien skedulointiprotokollan pitää käyttää aikarajoja suoritusjärjestyksiä määrättäessä. Pelkkien aikarajojen käyttö ei vielä riitä, vaan skedulointiprotokolla tarvitsee myös varsin luotettavat arviot tapahtumien suoritusten kestosta tai ainakin arviot tietokannan omien toimintojen pisimmästä mahdollisesta kestosta. Jotta tämä arvio olisi likimainkaan realistinen on kaikkien tietokannan palvelujen kestojen oltava mahdollisimman hyvin ennustettavia (engl. *predictable*). Erityisesti tämä koskee mahdollista levy-I/O:ta ja puskurin siivousprotokollaa.

Perinteisissä tietokannoissa käytetään lähes yksinomaan sarjallistuvuutta tapahtumien oikeellisuuskriteerinä. Tähän usein pyritään myös tosiaikatietokannoissa. Aikavaatimukset ovat kuitenkin tosiaikajärjestelmille tapahtumien sarjallistuvuutta ja kannan eheyttä tärkeämpiä. Siksi tosiaikatietokannoissa usein tingitäänkin sarjallistuvuus- ja/tai tiedon eheysvaatimuksista, jotta aikavaatimuksia voidaan taata paremmin.

Tosiaikatietokantoihin liittyy joitakin virheellisiä käsityksiä. Keskeisimmät väärinkäsitykset ovat [SSH99]:

Laitteistojen nopeutuminen ratkaisee tosiaikaongelmat tietokannoissa.

> Nopeakaan laite ei voi antaa takuita suoritusajoista.

Tietokantatekniikoiden kehittyminen ratkaisee tosiaikaongelmat tietokannoissa.

> Jos nämä tekniikat eivät tue aikarajoitteita, eivät ne voi antaa takuita suoritusajoista tai tietojen ajallisesta oikeellisuudesta.

Tosiaikaisuus on yksinkertaisesti nopeutta.

> Pelkkä nopeus ei riitä, koska tosiaikaisuuden tavoitteena on taata suoritukset aikarajoitteiden puitteissa.

Perinteinen kanta pystyy käsittelemään tosiaikavaatimuksia.

Toki tietokanta-alkioille voidaan määritellä attribuutti voimassaoloaika, jonka arvon kukin tapahtuma voi tarkastaa. Tapahtumat voivat myös seurata omaa ajankäyttöään. Järjestelmä vain ei anna mitään tue tällaista, joten se on suhteellisen tehotonta.

Tavanomainen keskusmuistitietokanta riittää tosiaikajärjestelmälle.

Joskus se riittääkin, vaan ei aina, koska se ei pysty antamaan takuuta suorituksen onnistumisesta määräaikaan mennessä.

Tosiaikatietokannan alkioiden on oltava keskusmuistissa.

Joskus näin onkin, mutta syy on silloin tarve tietojen nopea saatavuus, eikä tosiaikaisuus. Keskusmuistivaatimuksella rajoitetaan suotta tietokannan kokoa.

Tosiaikatietokanta ei voi taata omaa ennustettavuuttaan.

Ei voikaan, mutta se voi antaa tavanomaista kantaa tarkemman arvion omasta käyttäytymisestään, koska tosiaikatapahtumien toiminnallisuus usein tiedetään ennakkoon ja koska se käyttää menetelmiä, joilla epävarmuutta vähennetään.

Tosiaikatietokanta on erikoiskanta.

Tosiaikatietokantaa ei kuitenkaan tarvitse toteuttaa kullekin tosiaikajärjestelmälle erikseen, vaikka järjestelmien vaatimukset vaihtelevatkin. Eihän jokaiselle järjestelmälle toteuteta omaa käyttöjärjestelmää tai tavanomaista tietokantaa, vaikka järjestelmien vaatimukset näille vaihtelevatkin.

Tässä seminaariesitelmässä keskitytään erityisesti ajan käsittelyyn tosiaikatietokannassa ja sen vaikutuksiin tietokannan keskeisiin toimintoihin, kuten samanaikaisuudenhallinta, skedulointi, toipuminen ja vikasietoisuus. Valitettavasti nuo vaikutukset käydään läpi varsin yleisellä tasolla, sillä aika ja tila ei riitä tarkempaan käsittelyyn. Aiheesta on laadittu useita kokoomateoksia (katso mm. [BFW97, BLS97, Son95]), joiden avulla tosiaikatietokantoihin voi tutustua tarkemmin.

2 Aikarajoitteet

Tosiaikatietokantojen kaikkein tyypillisin piirre on niiden suhtautuminen aikaan sekä tietoalkioiden ja tapahtumien ominaisuutena että reaalimaailman kellona. Tietokannassa määritellyt aikarajoitteet kohdistuvat sekä kantaan tallennettuun dataan että sitä käyttäviin tapahtumiin.

2.1 Datan semantiikka

Tosiaikatietokantaan tallennetut data-alkiot usein kuvastavat jotain todellisen maailman ilmiötä, kuten jonkin anturin mitta-arvoa. Tällainen mitta-arvo voi olla vaikkapa auton kiihtyvyys, veden virtaama tai sademäärä. Tällaiselle mitta-arvolla on tärkeää, että se mahdollisimman tarkasti seuraa edustamansa todellisen maailman ilmiön arvomuutoksia. Näitä iliöitä vastaavien tietokanta-alkioiden arvoja yleensä päivitetään jaksollisesti. Päivitysjakson pituus määräytyy arvon tärkeyden ja käyttötapojen mukaan. Auton jarrujen puristusteho tai kaasupolkimen asento on päivitettävä useammin kuin esimerkiksi ulkoilman lämpötila.

Tosiaikatietokantaa voidaan käyttää esimerkiksi tietojen keruuseen ja säilyttämiseen vesivoimalan sulkujen ohjausjärjestelmässä [VT99]. Tietokannan sisältöä ja laskentamalleja käytetään määriteltäessä optimaalinen sulkujen asento sekä ohivirtaamalle että generaattorien käyttämälle vesimäärälle. Kriittisissä tilanteissa, kuten rankkasateiden aikana ja välittömästi niiden jälkeen, tietokantaa ja laskentamalleja käytetään minimoimaan tulvavahinkoja. Ohivirtaaman määrä lasketaan sulkujen asennon ja veden pinnan korkeuden perusteella. Sulkujen uusi asento määrätään tasan 15 minuutin välein. Tämän määrittämiseksi tietokantaan tehdään kysely, joka laskee sademittarien lukemien perusteella arvion saapuvalle vesimäärälle. Erilaisilla mitta-arvoilla on kelpoisuusjakso, jonka aikana ne kuvastavat todellista ilmiötä. Vaikka laskennalle onkin annettu näin pitkä aikarajoite, 15 minuuttia, on kyseessä kuitenkin tosiaikajärjestelmä, sillä järjestelmän on taattava, että laskenta saadaan tehtyä ja sulkujen uuden asennot määrättyä annetun aikarajoituksen mukaisesti.

Tietoalkioiden aikaeheysrajoitukset (engl. *temporal consistency contraints*) liittyvät tietoalkioiden kelpoisuusjaksoon. Absoluuttinen aikaeheys (engl. *absolute temporal consistency*) määrää tietoalkion arvon ja sen kuvastaman todellisen ilmiön välistä suhdetta. Ylläolevassa esimerkissä tällainen suhde

on kaikilla mitta-alkioiden arvoilla suhteessa mitattavaan ilmiöön. Tietokannan alkioita käytettäessä niiden välillä vallitsee myös suhteellinen aikaeheys (engl. *relative temporal consistency*), joka edellyttää että todellista ilmiötä kuvastavien tietoalkioiden päivityshetkien välinen aikaetäisyys on oltava ennaltamäärättyä arvoa pienempi. Esimerkiksi voitaisiin edellyttää, että veden pinnan korkeuksien mittaushetket eri sulkujen kohdalta on oltava korkeintaan 5 minuutin sisällä toisistaan.

Suhteellinen aikaeheys voidaan esittää myös hiukan formaalimmin [PSRS95]. Merkitään käytettävien tietoalkioiden joukkoa kirjaimella R ja yhtä tietoalkiota d kolmikolla $(arvo, abskesto, aikaleima)$. Absoluuttinen kelpoisuuskesto ($abskesto$) kertoo aikajakson pituuden, jona arvo on käyttökelpoinen mittauksen aikaleimasta alkaen. Joukon R suhteellisen kelpoisuusjakson pituus olkoon $R_{suhtkesto}$.

Oletetaan, että $d \in R$. d:n arvo on käyttökelpoinen, joss

1) d_{arvo} on loogisesti eheä eli se täyttää kaikki eheysrajoitukset
2) d on ajallisesti eheä
   - absoluuttinen aikaeheys: $(nykyhetki - d_{aikaleima}) \leq d_{abskesto}$
   - suhteellinen aikaeheys: $\forall d' \in R, |d_{aikaleima} - d'_{aikaleima}| \leq R_{suhtkesto}$

Esimerkiksi, kun $R_{suhtkesto} = 2$ ja $nykyhetki = 100$ sekä $R = \{lampotila, paine\}$ on (a) $lampotila = (347, 5, 95)$ ja $paine = (50, 10, 97)$ aikaeheä joukko, mutta (b) $lampotila = (347, 5, 95)$ ja $paine = (50, 10, 92)$ ei ole, vaikka kumpikin arvo yksinään onkin absoluuttisesti aikaeheä.

Koska data-alkioiden arvot tyypillisesti ovat voimassa vain rajoitetun ajan, voidaan versioinnin avulla hiukan löyhentää järjestelmän sarjallistuvuutta. Tällöin uusi arvon päivitys on mahdollista, vaikka joku tapahtuma vielä käyttäisikin vanhaa arvoa. Toisaalta tapahtumalle voidaan antaa edellinen alkion arvo, jos näin voidaan paremmin taata suhtellinen aikaeheys tapahtuman käyttämien tietoalkioiden välille.

2.2 Tapahtuman aikarajoitteet

Tapahtumien aikarajoitteet kuvataan usein määräaikana (engl. *deadline*) tai toistojakson kestona, esimerkiksi tapahtuma suoritetaan joka 5. minuutti. Aikarajoite ei sinänsä määrää tapahtumien suoritusjärjestystä. Toki sen huomioiminen on välttämätöntä. Perinteisessä kannassa, jossa aika ei

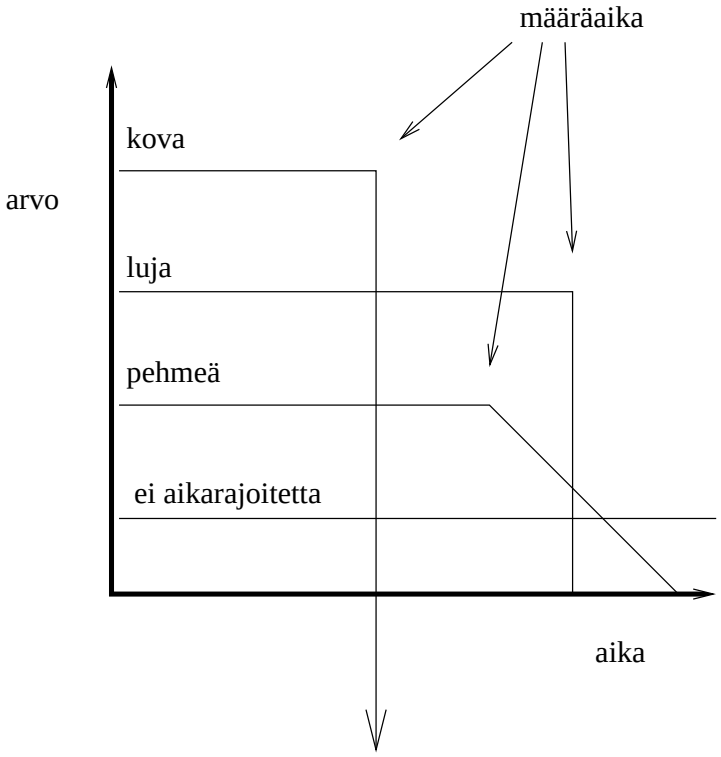
ole kuvioissa mukana, voidaan suoritukseen ottaa mikä tahansa ajokelpoinen tapahtuma. Ajokelpoisuus määräytyy lähinnä samanaikaisuudenhallintamekanismin avulla. Tosiaikatietokannan pitää kuitenkin pyrkiä suorittamaan tapahtumat sellaisessa järjestyksessä, että mahdollisimman moni suoritetaan onnistuneesti aikarajoitteidensa puitteissa. Tämän vuoksi ajokelpoisista tapahtumista pyritään valitsemaan se, joka maksimoi tavoitteen.

Perinteisesti tosiaikajärjestelmät tai tapahtumat yhdessä tosiaikajärjestelmässä usein luokitellaan koviin (*hard*) tai pehmeisiin (*soft*). Tosiaikatietokantojen ja tietokantatapahtumien luokittelussa näiden väliin vielä usein otetaan kolmas luokka: luja (*firm*) [Ram93].

- Kova määräaika edellyttää, että tapahtuma on varmasti suoritettu onnistuneesti määräaikaansa mennessä. Tästä ei voida missään tilanteessa joustaa yhtään. Tätä luokkaa käytetään tosiaikatietokannoissa suhteellisen vähän, koska tietokannanhallintajärjestelmässä on liian paljon epävarmuustekijöitä, jotta tuo onnistuminen voidaan taata ennalta.
- Luja määräaika tarkoittaa, että suoritus voidaan (ja täytyy) lopettaa välittömästi aikarajan täytyttyä. Tällöin esimerkiksi jakso vaihtuu ja seuraava vastaava tapahtuma käynnistyy. Edellinen pitää siivota pois, vaikka laskenta olisikin kesken. Sekä lujassa että pehmeässä on vain pyrkimys saavuttaa tuo annettu määräaika ajoissa.
- Pehmeän määräajan ylityksestä on jonkin verran haittaa, mutta laskenta kannattaa silti pyrkiä suorittamaan valmiiksi.

Näitä luokkia vastaavat kustannuksen arvofunktiot on piirretty kuvaan 1. Kovan arvo menee voimakkaasti negatiiviseksi ($-\infty$), jos aikarajoitusta rikotaan. Luja putoaa nollaan ja laskenta kannattaa keskeyttää. Sen sijaan pehmeän arvo alkaa hitaasti laskea, kun määräaika ohitetaan. Kuvaan on myös piirretty ei-tosiaikaisen tapahtuman arvofunktio, jonka arvo ei siis riipu suorituksen kestosta tai ajankohdasta.

Käyttämällä tarkentuvaa laskentamekanismia tapahtuman suorituksessa voidaan paremmin taata, että tapahtuman aikajakson päättyessä saadaan joku tulos. Tämä tulos on vain approksimaatio todellisesta arvosta. Joissakin tilanteissa on kuitenkin parempi saada approksimaatio kuin ei mitään tulosta. Tätä on käytetty voimalaitos esimerkissä korvaamaan puuttuvien vesimittareiden arvoja sulkujen tulevaa asemaa arvioivassa tapahtumassa [VT99].



Kuva 1: Tapahtumien aikarajoitteiden kustannusfunktiot [Ram93]

3 Skedulointi ja rinnakkaisuuden hallinta

Aikarajoitteiden lisäksi tapahtumien suoritusjärjestykseen voivat vaikuttaa myös muut tapahtuman ominaisuudet, esimerkiksi tapahtuman tärkeys (engl. *criticality*) ja tapahtuman arvioitu kesto [YWLS94]. Tapahtuman keston arvioiti usein perustuu ns. pahimman tilanteen kestoon (*worst-case execution time*).

Kovat tapahtumat on suoritettava varmemmin kuin lujat tai pehmeät. Tällöin ei välttämättä pelkän aikarajoitteen, kuten määräaika, käyttö riitä, vaan tarvitaan useampia kriteerejä. Usein tapahtumien suoritusjärjestystä kuvaamaan käytetään prioriteettia. Korkeimman prioriteetin tapahtuma on aina suoritusvuorossa. Tällöin tapahtuman aikarajoitteet ja muut ominaisuudet muunnetaan prioriteetiksi.

Tosiaikatietokannan käyttötavoista ja tapahtumien luonteesta riippuen voidaan tapahtumien suo-

ritusjärjestyksen määräämiseen käyttää useita erilaisia algoritmeja. Jo vuonna 1973 on osoitettu, että jaksollisten tapahtumien skedulointiin ovat optimaalisia menetelmiä rate-monotonic (RT) ja earliest-deadline-first (EDF) [LL73]. Satunnaisesti ilmaantuvat tapahtumat ovat ongelmallisempia. Näitä varten on esitelty useita erilaisia menetelmiä. Esimerkiksi Adaptive access parameter [DMK+96] pyrkii päättämään tapahtuman hyväksymisen ja skeduloinnin samanaikaisesti ennen tapahtuman suorituksen aloittamista. Varsin yleisesti käytetty skedulointimenetelmä on EDF. Se sopii myös satunnaisille tapahtumille, vaikka ei ole niille optimaalinen. EDF:ää käytettäessä korkein prioriteetti annetaan tapahtumalle, jonka määräaika on ajallisesti lähimpänä.

Suunnitteluaikainen suoritusjärjestyksen määrääminen ja analyysi on erittäin tärkeää koville järjestelmille, koska vain siten voidaan varmistaa, että järjestelmän teho ja suorituskyky riittää suunniteltujen tapahtumien suorittamiseen. Tällöin sidotaan kaikki suoritettavat tapahtumat jo ennen järjestelmän käyttöönottoa ja muutokset edellyttävät uuden järjestelmän käyttöönottoa tai ainakin analyysia uudesta suoritusjärjestyksestä ennen muutoksia. Staattinen ratkaisu ei sovellu kaikkiin järjestelmiin. Usein kaivataan enemmän dynaamisuutta, vaikkapa vapaamuotoisia kyselyjä tosiaikatietokantaan. Tosiaikatietokannoissa myös tapahtumien väliset konfliktit vaikeuttavat staattista skedulointia.

Tietokannoissa tapahtumien suoritusjärjestys ei määräydy yksinomaan skedulointimekanismilla, vaan tietokannan käyttämä rinnakkaisuudenhallintamekanismi vaikuttaa siihen myös. Rinnakkaisuudenhallintamekanismit jaetaan lukitseviin ja optimistisiin. Lukitsevat mekanismit estävät konfliktoivien tapahtumien etenemisen, kunnes edellinen on sitoutunut. Optimistiset tarkistavat konfliktit sitoutumisvaiheessa ja abortoivat konfliktoivia tapahtumia tarpeen mukaan.

Tunnetuin ja yleisimmin käytetty lukitseva menetelmä, kaksivaiheinen lukitus (2PL), ei toimi aikarajoitteisten tapahtumien kanssa. Lukon saa se joka ensin pyytää ja lukko pysyy varattuna kunnes pyytäjä sitoutuu. Jos lukon on saanut alemman prioriteetin tapahtuma, joutuu korkeamman prioriteetin tapahtuma odottamaan. Tällöin syntyy tilanne, joka kutsutaan käänteiseksi prioriteetiksi (engl. *priority inversion*). Näitä tilanteita pyritään estämään, koska se voi aiheuttaa, että korkeampiprioriteettinen tapahtuma ei valmistukaan määräajassa jouduttuaan odottamaan vähempiarvoisia tapahtumia. Jos kyseessä on vielä kova määräaika, niin tällainen ei yksinkertaisesti ole sallittua. On-

gelma voidaan korjata käyttämällä protokollaa, joka tunnistaa prioriteetit ja joko sallii korkeamman prioriteetin tapahtuman edetä tai nostaa matalamman prioriteetin tapahtuman prioriteettia, jotta korkeampi voi edetä nopeammin [YWLS94].

Optimististen menetelmien suurin ongelma on myöhään tapahtuva konfliktien tarkastus. Jos lukitsevien menetelmien ongelma on odotus, niin optimististen ongelma on uudelleenkäynnistys tai abortointi vasta suorituksen jälkeen. Optimistisiin menetelmiin on myös yhdistetty aikaleimoja, joilla pyritään abortoimaan tapahtuma mahdollisimman aikaisin konfliktin vuoksi. Esimerkiksi OCC-TI ([LS93]) käyttää aikavälejä, joilla määritellään kunkin tapahtuman sallittu sitoutumisaika. Sitoutumisajankohdan pitää sijoittua sallitulle aikavälille tai tapahtuma abortoidaan.

Suurin osa tosiaikatietokanta tutkimuksesta onkin keskittynyt tapahtumien skedulointiin ja rinnakkaisuudenhallintaan. Tämä on erittäin keskeinen ongelma, joka ratkaisee ehditäänkö tapahtumat suorittaa määräaikaan mennessä. Algoritmeja ja niiden muunnoksia onkin julkaistu monia, mutta mikään ei toistaiseksi ole osoittautunut muita paremmaksi tai edes samanarvoiseksi kaikissa tilanteissa. Perinteisten tietokantojen puolella tällainen yleinen menetelmä on juuri 2PL. Olisi toivottavaa, että tosiaikapuolelle muotoutuisi vastaavanlainen yleismekanismi. Tämä tosin saattaa jäädä vain toiveeksi.

Rinnakkaisuudenhallintamekanismilla on varsin paljon mahdollisuuksia löyhentää sarjallistuvuutta. Se voi käyttää epsilon-sarjallistuvuutta eli sallia tietyn määrän ajallista epäeheyttä kyselyjen tietoalkioiden kesken. Myös erilaisiin tietoalkioiden ja tapahtumien semantiikkaan perustuen voidaan sallia yleisesti sarjallistumattomia tapahtumien suorituksia. Esimerkiksi semantic-based concurrency control [PLN97] käyttää laajennettua konfliktitaulua, jossa siis sallitaan useammanlaisia rinnakkaisia luku- ja kirjoitusoperaatioita samoihin tietoalkioihin käyttäen operaatioiden semanttista tietoa hyväksi.

4 Tietokannan rakenne

Tietoalkioiden pysyvyyden takaamiseksi ne pitää varastoida siten, että niiden arvot säilyvät myös kun tietokannanhallintajärjestelmä vikaantuu. Tämän vuoksi useimmat tietokannat pitävät pysy-

viä arvoja tallessa levyllä ja levykirjoituksia suojatakseen tallentavat vielä viimeisimmät muutokset erilliselle lokille. Levypohjaisen järjestelmän sijasta tietokanta voi myös pitää alkioiden ensisijaista kopiota tallessa keskusmuistissa. Tällöinkin levyllä on yleensä pysyvyyden vuoksi tallessa arkistokopio tai vain loki. Vaikka keskusmuistin koko on kasvanut, on myös tietokantoihin tallennettavan tiedon määrä kasvanut, eivätkä kaikki (tosiaika)tietokannat ole riittävän pieniä mahtuakseen kokonaan keskusmuistiin.

Eräs yleisimpiä väärinkäsityksiä on väittää, että kaikki tosiaikatietokannat ovat keskusmuistitietokantoja tai että kaikki keskusmuistitietokannat ovat tosiaikatietokantoja [SSH99]. Toki joitakin keskusmuistitietokantoja käytetään osata tosiaikajärjestelmiä, koska tosiaikatietokantoja ei ole välttämättä saatavilla ja koska pehmeät aikarajoitukset sallivat tavanomaisen tietokannan käytön. Vastaavasti osa tosiaikatietokannoista on toteutettu keskusmuistipohjaisena, koska saantiaikavaatimukset eivät salli levyoperaatioita tapahtumien suorituksessa. Tällainen tosiaikainen tietokantaratkaisu on esimerkiksi MDARTS [LSK00], joka uhraa tiedon pysyvyyden sen nopean saatavuuden vuoksi. MDARTS on suunniteltu vain lyhyen kelpoisuusjakson tietoalkioille, joiden arvoja päivitetään säännöllisesti. Tietoalkioiden kelpoisuusjakso on niin lyhyt, että kanta ei ehdi toipua tuon jakson aikana ja siksi sitä ei edes tavoitella. Vikaantumisesta toivutaan yksinkertaisesti vain käynnistämällä tietokannanhallintajärjestelmä ja odottamalla hetki, että kaikkien tietoalkioiden uuden arvot on taas päivitetty.

Tosiaikatietokannan toteuttaminen levytietokantana tuo omat ongelmansa, vaikka tiedon pysyvyys näin voidaankin taata. Levy I/O aiheuttaa runsaasti ongelmia tapahtumien keston arviointiin. Perinteisesti levypyynnöt vain laitetaan jonoon ja levynkäsittelyprotokolla suorittaa ne haluamassaan järjestyksessä. Tämä aiheuttaa ennustamatonta vaihtelua suorituspyyntöjen kestoille. Joitakin levynkäsittelyprotokollia, jotka huomioivat määräajat [CSKT91] tai prioriteetin [AGM90, CJL89], on esitelty. Artikkelien mukaan nämä parantavat levyoperaatioiden keston ennustettavuutta ja erityisesti tukevat tärkeiden pyyntöjen nopeampaa suoritusta. FDSCAN [AGM90] palvelee levypyynnöt niiden aikarajoitusten mukaisessa järjestyksessä. Aivan kuin perinteinen SCAN se poimii levypään siirtyessä matkan varrelle osuvat muut pyynnöt saman tien. SSEDO ja SSEDV [CSKT91] pyrkivät minimoimaan lukupään siirtoja ja pyyntöjen järjestelyjä. Korkeimman prioriteetin saa ly-

hyimmän aikarajoituksen pyyntö, mutta jos pidemmän aikarajoituksen pyyntö on hyvin lähellä lukupään nykyistä kohtaa se saakin korkeimman prioriteetin, jolloin lukupäätä ei tarvitse siirtää edestakaisin.

Säilyvän muistin käyttöä tavanomaisen katoavan muistin sijasta on myös tutkittu. Esimerkiksi sijoittamalla koko tietokantapuskuri tällaiseen muistiin voidaan tapahtumien sitoutumista nopeuttaa [BDH+93], koska vikatilanteissa viimeisimmät muutokset säilyvät muistissa, josta ne ovat käytettävissä kun tietokannanhallintajärjestelmä on toipunut.

Tosiaikatietokantoja on lähdetty tekemään kahdella toisistaan poikkeavalla tavalla. Toisaalta, on suunniteltu puhtaalta pöydältä täysin tosiaikainen tietokanta, jossa kaikki komponentit ja mekanismit hallitsevan ajankäsittelyn ja tukevat aikakriittisten tapahtumien suorittamista. Esimerkkinä voidaan mainita StarBase [KLS96]. Se käyttää lujia aikarajoja eikä tapahtumien suoritusten kestoa tunneta etukäteen. Aikarajansa ylittänyt tapahtuma keskeytetään. Toisaalta, olemassa oleviin perinteisiin tietokantoihin on lisätty tosiaikaisuutta tukevia piirteitä. Esimerkiksi Genesis tietokannasta on tehty RT-Genesis lisäämällä tapahtumille aikarajoitteet [AGN+96]. Tietoalkioilla ei kuitenkaan ole aikaeheyttä. Pehmeiden tosiaikatapahtumien suoritus perustuu niiden aikarajoitteista laskettuihin prioriteetteihin. Myös lukitus- ja puskurointimekanismeja on muutettu tukemaan aikakäsitystä.

5 Vikasietoisuus ja saatavuus

Tosiaikatietokantoja käytetään aina osana jotain laajempaa tosiaikajärjestelmää. Järjestelmän toiminnalle on sen luonteesta johtuen usein asetettu erilaisia luotettavuus- ja vikasietoisuusvaatimuksia. Koko järjestelmä on siis suunniteltava siten, että nuo vaatimukset täyttyvät. Tämä koskee myös järjestelmän osana olevaa tosiaikatietokantaa. Voimalaitosesimerkissä edellytetään, että sulkuja säädetään 15 minuutin välein. Vikasietoisuuden vuoksi voidaan esimerkiksi sallia yhden yksittäisen säädön tekemättömyys, mutta ei sen enempää. Kun tämä vaatimus siirretään tosiaikatietokantaan asti, on selvää, että tietokannan tietojen pitää olla saatavilla silloin kun niitä tarvitaan. Erityisesti vikaantuneen kannan pitää toipua niin nopeasti, että korkeintaan yksi säätö jää väliin, eli tietokanta ei missään olosuhteissa saa käyttää yli 15 minuuttia toipumiseensa. Tähän 15 minuuttiin pitää siis

mahtua vian paikallistaminen, vikaantuneen komponentin vaihto, tietokannanhallintajärjestelmän uudelleenkäynnistys ja tietokannan tilan palautus. Aika varmaankin riittää, kunhan komponenttia ei tarvitse noutaa kovin kaukaa ja tietokannan tilan palautus on suunniteltu toimimaan aikarajoituksen mukaisesti.

Vielä julkaisematon artikkeli [SSS02] esittelee lokeihin perustuvan toipumismekanismin tosiaikatietokannoille. Tietoalkiot jaetaan kahteen päätyyppiin niitä käyttävien tapahtumien tärkeyden perusteella. Kriittiset tietoalkiot varmistetaan säilyvään keskusmuistiin ja ei-kriittiset tietoalkiot varmistetaan levylle. Kriittisistä tietoalkioista erotetaan vielä kaksi aliryhmää, joiden lokitallennus- ja toipumismekanismi on erilainen. Muuttuvalla kriittisellä datalla (*critical variant data*) on jokin voimassaolojakso, jonka kuluessa kukin arvo on käytettävissä. Pysyvällä kriittisellä datalla (*critical invariant data*) ei yksittäiseen arvoon liittyvää voimassaolorajoitetta ole. Toipumisvaiheessa nämä käyttäytyvät eritavoin, sillä muuttuvista datoista voidaan palauttaa vain ne, joiden arvot ovat vielä kelvollisia. Kelvottomien arvojen tilalle saadaan uudet arvot, kun päivitystapahtuma asettaa ne. Menetelmän keskeinen piirre on toipumisajan ennustettava kesto kriittisten datojen osalta. Ei-kriittiset datat palautetaan, kun tietokannan kriittinen osa on jo käytössä.

Aina ei aikaa ole käytettävissä yhtään tai vain niin vähän, että pelkkään uudelleenkäynnistykseenkin ilman toipumista kuluu liikaa aikaa. Silloin joudutaan turvautumaan toisenlaiseen lähestymistapaan. Tietokannan sisältö täytyy monistaa useampaan paikkaan. Tällöin yhden kopion vikaantuminen ei estä koko järjestelmän käyttöä. Tällaista monentamista käytetään joskus myös kuormantasaukseen. Jos tietokannan kopiot osallistuvat palveluntuottamiseen palvelemalla omia saapuvia tapahtumia, kutsutaan tietokantarakennetta toisinnetuksi kannaksi. Jos kopiot osallistuvat laskentaan toimimalla synkronoidusta pääsolmun kanssa, on kyseessä aktiivinen monentaminen. Jos kopiot eivät osallistu laskentaa vaan vain seuraavat pääsolmun tilanmuutoksia on kyseessä passiivinen monentaminen. Tällöin kopioita kutsutaan peilisolmuiksi.

Passiivinen monentaminen soveltuu tietokannoille, koska tietokanta pääsolmussa tuottaa muutenkin tietoa pysyvistä tilamuutoksistaan eli tietoalkioiden arvojen muuttumisista. Pääsolmu tarjoaa tietokantapalvelun kannan asiakkaille ja peilisolmu keskittyy pitämään oman kantakopionsa yhdenmukaisena pääsolmun kanssa. Kaupallinen tietokantajärjestelmä Clustra [THK96] käyttää lokitietoa

pääsolmusta tiedon siirtämiseksi peilisolmuille. Järjestelmästä löytyy tietoa myös yrityksen verkkosivustosta *http://www.clustra.com/*. Clustra ei varsinaisesti ole tosiaikatietokanta, koska se ei tue tietoalkioiden aikarajoitteita. Korkean saatavuutensa ja tapahtumien pehmeiden aikarajojen käytön vuoksi se soveltuu ja sitä käytetään joidenkin tosiaikajärjestelmien osana. Clustrassa kukin tietokannan laskentasolmu, toimii osalle kantaa pääsolmuna ja toiselle osalle kantaa peilisolmuna. Näin kustakin kannan alkiosta on vain yksi pääkopio, mutta kaikkia kannan solmuja voidaan käyttää myös tietokantakyselyjen suorittamiseen.

Passiivinen lokeihin pohjaava monentaminen ei riitä, jos suorituksessa olevaa tapahtumaa pitää kesken sen suorituksen jatkaa peilisolmussa pääsolmun vikaannuttua. Tällöin tarvitaan joko aktiivista monentamista tai peilisolmuun pitää toimittaa tietoa myös suorituksessa olevien tapahtumien tilojen muutoksista. Tällöin lähennytään voimakkaasti yleistä tosiaikajärjestelmien vikasietoisuusmallia, jossa mikä tahansa tosiaikasovellus voidaan monentaa vikasietoisuuden parantamiseksi (katso [ZJ99]).

6 Yhteenveto

Tosiaikatietokannoista on hyötyä joidenkin tosiaikajärjestelmien toiminnalle. Esimerkiksi erilaisissa ohjausjärjestelmissä kertyvä mittaustieto voidaan tallentaa tosiaikatietokantaan käytettäväksi eri-osissa järjestelmää. Ilman tietokantaa kukin järjestelmän osio joutuisi itse noutamaan tarvisemansa lukemat antureilta.

Tosiaikatietokannan hallintajärjestelmä pyrkii maksimoimaan määräajassaan onnistuneesti päättyvien tapahtumien lukumäärän. Tällöin joitakin vähemmän tärkeitä tapahtumia saatetaan joutua uhraamaan tärkeämpien tieltä.

Varsin harvoja tosiaikatietokantoja on tarjolla kovia määräaikoja vaativien tosiaikajärjestelmien avuksi. Useimmat näistä ovat joutuneet luopumaan joistakin tietokantapiirteistä voidakseen taata tapahtumien suoritukset niiden määräaikoihin mennessä. Esimerkiksi saapuvien tapahtumien joukko on sidottu jo suunnitteluvaiheessa, tietokannan alkioita ei tallenneta mihinkään pysyvään muistiin, tai tapahtumien sarjallistuvuutta on löyhennetty.

Tosiaikatietokannat ovat edelleen suhteellisen aktiivin tutkimuksen kohteena ja niihin liittyy vielä paljon ratkaisemattomia asioita. Erityisesti nuo joustot kaipaisivat teoreettisempaa mallia. Myös tietoalkioiden aikakäsitteiden yleinen malli helpottaisi tosiaikatietokantojen käyttäjiä. Lähes kaikki tietokantojen yleiset piirteet täytyy arvioida uudelleen tosiaikaisuuden näkökulmasta. Esimerkiksi turvallisuus (engl.*security*) ja aikarajat eivät aina ole täysin yhteensopivia. Nämä ongelmat pitää ratkaista ennen kuin tosiaikatietokannat voivat yleistyä myös alueille, joissa niistä on hyötyä, mutta jossa niitä ei voida vielä käyttää. Esimerkiksi teletoiminnassa käytetään paljon tavanomaisia tietokantoja tai sovelluskohtaisia erikoiskantoja, koska tosiaikatietokannat eivät vielä sovellu kaikkiin kohteisiin.

Viitteet

[AGM90] Robert K. Abbott and Hector Garcia-Molina. Scheduling I/O requests with deadlines: A performance evaluation. In *Proceedings of Real-Time Systems Symposium*, pages 113–124, Lake Buena Vista, Florida, December 1990. IEEE.

[AGN+96] Rohan Aranha, Venkatesh Ganti, Srinivasa Narayanan, C.R. Muthukrishnan, S.T.S. Prasad, and Krithi Ramamritham. Implementation of real-time database system. *Information Systems*, 21(1):55–74, 1996.

[BDH+93] Anupam Bhide, Daniel Dias, Nagui Halim, Basil Smith, and Francis Parr. A case for fault-tolerant memory for transaction processing. In *the 23rd International Symposium on Fault-Tolerant Computing*, pages 451–460, 1993.

[BFW97] Azer Bestavros and Victor Fay-Wolfe, editors. *Real-Time Database and Information Systems*. Kluwer Academic Publishers, London, 1997.

[BLS97] Azer Bestavros, Kwei-Jay Lin, and Sang H. Son, editors. *Real-Time Database Systems: Issues and Applications*. Kluwer Academic Publishers, 1997.

[CJL89] Michael J. Carey, Rajiv Jauhari, and Miron Livny. Priority in DBMS resource scheduling. In P. M. G. Apers and G. Wiederhold, editors, *Proceedings of the 15th VLDB Conference*, pages 397–410, San Mateo, Calif., 1989. Morgan Kaufmann.

[CSKT91] Shenze Chen, John A. Stankovic, James F. Kurose, and Don Towsley. Performance evaluation of two new disk scheduling algorithms for real-time systems. *The Journal of Real-Time Systems*, 3(3):307–336, September 1991.

[DMK+96] A. Datta, S. Mukherjee, P. Konana, I. Viguier, and A. Bajaj. Multiclass transaction scheduling and overload management in firm real-time database systems. *Information Systems*, 21(1):29–54, March 1996.

[KLS96] Young-Kuk Kim, Matthew R. Lehr, and Sang H. Son. Software architecture for a firm real-time database system. *Journal of System Architecture*, 42(6):547–562, December 1996.

[LL73] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[LS93] J. Lee and S. H. Son. Using dynamic adjustment of serialization order for real-time database systems. In *Proceedings of the 14th IEEE Real-Time Systems Symposium*, pages 66–75, Raleigh-Durham, NC, USA, 1993. IEEE Computer Society Press.

[LSK00] Victor B. Lortz, Kang G. Shin, and Jinho Kim. MDARTS: A multiprocessor database architecture for hard real-time systems. *IEEE Transaction on Knowledge and Data Engineering*, 12(4):621–644, July 2000.

[PLN97] Ching-Shan Peng, Kwei-Jay Lin, and Tony P. Ng. A performance study of the semantic-based concurrency control protocol in air traffic control systems. In A. Bestavros and V. Fay-Wolfe, editors, *Real-Time Database and Information Systems*, pages 181–205, London, 1997. Kluwer Academic Publishers.

[PSRS95] Bhaskar Purimetla, Rajendran M. Sivasankaran, Krithi Ramamritham, and John A. Stankovic. *Real-Time Databases: Issues and Applications*, pages 487–507. 1995.

[Ram93] Krithi Ramamritham. Real-time databases. *Distributed and Parallel Databases*, 1:199–226, April 1993.

[Son95] Sang Son, editor. *Advances in Real-Time Systems*. Prentice Hall, USA, 1995.

[SSH99] John A. Stankovic, Sang Son, and Jorgen Hansson. Misconceptions about real-time databases. *IEEE Computer*, pages 29–36, June 1999.

[SSS02] LihChyun Shu, John A. Stankovic, and Sang H. Son. Real-time logging and failure recovery. Submitted to *Euromicro Real-Time Systems Conference*, 2002.

[THK96] Øystein Torbjørnsen, Svein-Olaf Hvasshovd, and Young-Kuk Kim. Towards real-time performance in a scalable continuously available telecom dbms. In *Proceedings of the RTDB 1996 Conference*, 1996.

[VT99] Susan Vrbsky and Sasa Tomic. Satisfying temporal consistency constraints of real-time databases. *Journal of Systems and Software*, 45:45–60, 1999.

[YWLS94] Philip S. Yu, Kun-Lung Wu, Kwei-Jay Lin, and Sang H. Son. On real-time databases: Concurrency control and scheduling. *Proceedings of the IEEE*, 82(1):140–157, January 1994.

[ZJ99] Hengming Zou and Farnam Jahanian. A real-time primary-backup replication service. *IEEE Transaction on Parallel and Distributed Systems*, 10(6):533–548, June 1999.