Integroitu ja Adaptiivinen Optimistinen Samanaikaisuudenhallintamenetelmä

Tosiaikatietokantoihin

Jan Lindström

Tosiaikajärjestelmät-seminaari

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

Sisältö

1 Johdanto

Useimmat kaupalliset tietokannanhallintajärjestelmät käyttävät lukitusta samanaikaisuudenhallintamenetelmänä. Tästä huolimatta optimistiset menetelmät ovat herättäneet kiinnostusta viimeisen kymmenen vuoden aikana, koska optimistiset menetelmät ovat tehokkaita paljon tietoa sisältävissä järjestelmissä. Tällaisia järjestelmiä ovat esimerkiksi puhelinkeskus, tietoliikenteen hallinta, navikointi ja pörssi.

Perinteiset tietokannat käsittelevät pysyvää tietoa. Transaktiot käsittelevät tätä pysyvää tietoa tavoitteena säilyttää tiedon eheys. Tavoitteena on hyvä vasteaika tai suoritusteho. Tosiaikatietokantojen tavoitteet ovat vastakkaisia. Tosiaikatietokannoissa voidaan säilyttää myös temporaalista tietoa eli tietoa, joka vanhenee tietyn ajan jälkeen. Tiedon temporaalisuuden ja ympäristön tosiaikavaatimusten takia transaktioilla tosiaikatietokannoissa on aikarajat. Tärkeä ero perinteisiin tietokantoihin on tarve pystyä vastaamaan tietokantaoperaatioihin annetun aikarajan puitteissa [SSH99].

Samanaikaisuudenhallinta on yksi keskeisimmistä tutkimuksenaloista tosiaikatietokannoissa. Useimmat esitetyistä menetelmistä pohjautuvat kaksivaiheiseen lukitukseen (2PL) [EGLT76]. 2PL-menetelmää on tutkittu paljon ja menetelmä on laajasti käytössä kaupallisissa tietokannanhallintajärjestelmissä. Kuitenkin 2PL-menetelmällä on luontaisesti lukkiutumisen ja pitkien ja ennalta määrittelemättömien odotusten ongelmat. Nämä ongelmat saattavat muodostua vakaviksi tosiaikatietokannoissa, koska tosiaikatransaktioiden tulee pysyä määritellyissä aikarajoissa [Ram93].

Optimistiset samanaikaisuudenhallintamenetelmät [Här84, KR81] ovat erityisen houkuttelevia tosiaikatietokannoissa, koska ne eivät sisällä odotusta ja ovat lukkiutumattomia. Siksi viime vuosikymmenen aikana on ehdotettu useita optimistisia menetelmiä tosiaikatietokantoihin [DS96, DVSK97, HSRT91, LLH95a, LLH95b, LS96, Lin00].

Optimistiset menetelmät on osoitettu tehokkaammiksi kuin lukitusmenetelmät tosiaikatietokannoissa [HCL90a, HCL90b], mutta optimististen menetelmien ongelmana on turhat uudelleenkäynnistykset ja raskas peruutus. Konfliktit havaitaan vasta transaktion lopussa, joka johtaa lähes valmiin transaktion peruutukseen ja uudelleenkäynnistykseen.

Prioriteettitietoisia optimistiseen menetelmään pohjautuvia menetelmiä on tutkittu vähän. Ainoat poikkeikset ovat OCC-APR [DS96, DVSK97] ja OCC-TI variaatio [DSK00]. Tutkimuksissa ei kuitenkaan prioriteettien käytöstä havaittu mitään hyötyä transaktioiden läpimenon kannalta.

Kyseiset menetelmät käyttivät prioriteettiä konfliktien hallinnassa, mikä ei välttämättä ole hyvä valinta dynaamisissa järjestelmissä. Transaktiot erittäin lyhyillä aikarajoilla eivät välttämättä ole kriittisiä ja päinvastoin. Siksi, tässä työssä ehdotetaan transaktioiden kriittisyyden käyttöä prioriteetin sijasta konfliktien hallinnassa. Ehdotettu menetelmä perustuu integroituun samanaikaisuudenhallintamenetelmään [BHG87, HCJ96, SPL92], joka laajentaa optimistisia menetelmiä transaktioiden attribuuttitiedolla.

Loppuesitys on organisoitu seuraavasti. Kappale 2 esittelee ehdotetun menetelmän perusideat. Kappale 3 esittelee testiympäristön ja testitulokset. Lopuksi esityksen yhteenveto esitetään Kappaleessa 3.

2 Tosiaikapiirteitä optimistiseen menetelmään

Monet aikaisemmin ehdotettu menetelmät eivät sisällä mitään tosiaikapiirteitä. Siksi, nämä menetelmät ovat liian ’reiluja’. Useimmat tosiaikajärjestelmien skedulointimenetelmät käyttävät prioriteettipohjaista skedulointia [AGM88], missä transaktioille asetetaan prioriteetti. Tämä prioriteetti on implisiittinen tai eksplisiittinen fuktio transaktion prioriteetista tai tärkeydestä tai molemmista. Kriittisuus kuvaa transaktion tärkeyttä järjestelmän kannalta. Nämä kaksi ominaisuutta kuitenkin voivat olla ristiriitaisia.

Tosiaikajärjestelmissä transaktiot skeduloidaan prioriteettien perusteella. Siksi, korkean prioriteetin transaktiot suoritetaan ennen alemman prioriteetin transaktioita. Tämä tapahtuu vain jos korkeamman prioriteetin transaktio on valmiina suoritukseen. Muuten alemman prioriteetin transaktiota suoritetaan. Tästä syystä korkeamman prioriteetin transaktio saatta konfliktoida alemman prioriteetin transaktion kanssa.

2.1 Integroitu optimistinen samanaikaisuudenhallintamenetelmä

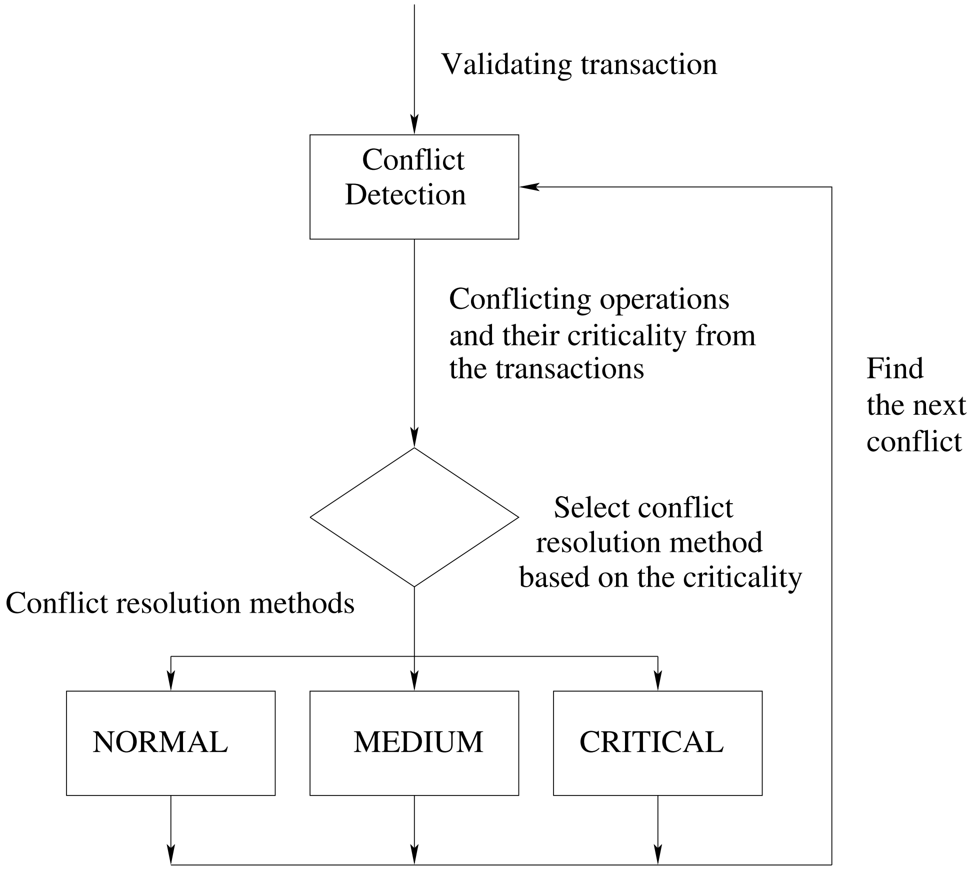
Jokaisella transaktio-olion ilmentymällä on attribuutit: *prioriteetti, aikaraja ja kriittisyys*. Järjestelmän toteuttaja asettaa transaktiolle aikarajan ja kriittisyyden. Prioriteetin laskee skeduloija käyttäen muita attribuutteja. Siksi prioriteetti riippuu transaktion saapumisajasta järjestelmään ja on siis dynaaminen. Kriittisyys on jokaisella saman transaktio-olion ilmentymällä sama ja on siis staattinen ominaisuus kuten aikarajakin. Kriittisyydestä käytetään seuraavia arvoja koodattuina numeroiksi:

- **Normal**: Transaktio ei ole kriittinen, mutta tulisi saada suoritetuksi jos historia on sarjallistuva. Tälle transaktioluokalle käytetään perusmenetelmää.

- **Medium**: Transaktio on tärkeä ja sitä ei tulisi peruuttaa jos historiassa on konflikti vähemmän tärkeän transaktion kanssa. Tälle transaktioluokalle käytetään menetelmää, missä vähemmän tärkeä transaktio peruutetaan jos aktiivinen tärkeämpi transaktio peruutettaisiin konfliktin takia.

- **Critical**: Transaktio on kriittinen ja sita ei peruuteta jos historiassa on konflikti vähemmän tärkeän transaktion kanssa. Tälle transaktioluokalle käytetään menetelmää, missä vähemmän tärkeä transaktio peruutetaan aina jos se konfliktoi kriittisen transaktion kanssa.

Oletetaan, että konfliktien etsintäalgoritmi (Kuva 2) on havainnut validoivan transaktion T_v ja aktiivisen transaktion T_1 olevan konfliktissa. Validoiva transaktio on lukenut tietoalkion X ja aktiivinen transaktion on esikirjoittanut (prewrite) saman tietoalkion X. Lisäksi oletetaan, että $criticality(T_v) = criticality(T_1) = normal$. Tässä tapauksessa konflikti selvitetään käyttämällä normaalia eteenpäinsiirtoa.



Kuva 1: Integrated Optimistic Concurrency Control Method.

Oletetaan, että konfliktien etsintäalgoritmi (Kuva 2) on havainnut validoivan transaktion T_v ja aktiivisen transaktion T_2 olevan konfliktissa. Validoiva transaktio on lukenut tietoalkion X ja aktiivinen transaktion on esikirjoittanut (prewrite) saman tietoalkion X. Lisäksi oletetaan, että $criticality(T_v) < criticality(T_2) = medium$. Tässä tapauksessa konflikti selvitetään käyttämällä medium menetelmän eteenpäinsiirtoa (Kuva 1).

Oletetaan, että konfliktien etsintäalgoritmi (Kuva 2) on havainnut validoivan transaktion T_v ja aktiivisen transaktion T_3 olevan konfliktissa. Validoiva transaktio on lukenut tietoalkion X ja aktiivinen transaktion on esikirjoittanut (prewrite) saman tietoalkion X. Lisäksi oletetaan, että $criticality(T_v) < criticality(T_3) = critical$. Tässä tapauksessa konflikti selvitetään käyttämällä critical menetelmän eteenpäinsiirtoa (Kuva 1).

Käytännössä jokaisen konfliktin kohdalla tarkistetaan konfliktiin osallistuvien transaktioiden kriittisyys ja valitaan niistä suurempi. Tämän jälkeen valitaan kyseisen kriittisyyden menetelmä. Tämä tehdään jokaisen konfliktin yhteydessä. Menetelmässä siis saatetaan vaihtaa konfliktien hallintamenetelmää jokaisen konfliktin yhteydessä. Menetelmä siis adaptoituu transaktioiden kriittisyyden mukaan.

2.2 Konfliktien etsintä

Tässä kappaleessa esitetään konfliktien etsintämenetelmä. Konfliktien etsintä perustuu eteenpäin validointiin (forward validation) [Här84]. Eteenpäin validoinnissa validoiva transaktiota verrataan muihin konfliktoiviin aktiivisiin transaktioihin.

Menetelmässä jokaisella transaktiolla on aikaväli TI (timestamp interval) [LS93]. Samantapaista menetelmää käytetään myös OCC-TI menetelmässä [LS96]. Transaktion alussa aikaväli alustetaan koko väliksi eli $[0, \infty[$. Jokainen transaktio tulee suorittaa aikavälinsä puitteissa. Konfliktitilanteessa transaktion aikaväliä muutetaan kuvaamaan transaktioiden välisiä riippuvuuksia eli muodostetaan aikaväleistä osittaisjärjestys. Jos transaktion aikaväli tulee tyhjäksi, transaktio täytyy peruuttaa. Tyhjä aikaväli kuvaa epäsarjallistuvaa transaktiota.

Validoinnin aluksi (Kuva 2) validoivalle trasaktiolle T_v valitaan sitoutumisaikaleima. Sitoutumisaikaleimaksi $TS(T_v)$ valitaan validointiaika jos se kuuluu validoivan transaktion aikaväliin tai validoivan transaktion aikavälin maksimiarvo.

Tämän jälkeen läpikäydään jokainen validoivan transaktion lukemat $RS(T_v)$ ja kirjoittamat $WS(T_v)$ tietoalkiot. Luetuista tietoalkioista tarkisttaan että lukuoperaatio on tapahtunut sitoutuneelta transaktiolta. Kirjoitetuista tietoalkioista tarkistetaan että tietoalkiota ei ole kirjoitettu transaktion suorituksen aikana.

Jokaiselle tietoalkiolle etsitään konfliktoivat aktiiviset transaktiot jos sellaisia löytyy. Riippuen konfliktin muodosta tehdään seuvat dynaamiset siirrot:

- Jos validoiva transaktio T_v on lukenut tietoalkion D_i eli $D_i \in RS(T_v)$ ja aktiivinen transaktio T_a on kirjoittanut tietoalkion D_i eli $D_i \in WS(T_a)$, aktiivista transaktiota T_a täytyy siirtää eteenpäin. Eli $TI(T_a) = TI(T_a) \cup [TS(T_v) + 1, \infty[$.
-
- Jos validoiva transaktio T_v on kirjoittanut tietoalkion D_i eli $D_i \in WS(T_v)$ ja aktiivinen transaktio T_a on lukenut tietoalkion D_i eli $D_i \in RS(T_a)$, aktiivista transaktiota T_a täytyy siirtää taaksepäin. Eli $TI(T_a) = TI(T_a) \cup [0, TS(T_v) - 1]$.

- Jos validoiva transaktio T_v on kirjoittanut tietoalkion D_i eli $D_i \in WS(T_v)$ ja aktiivinen transaktio T_a on kirjoittanut tietoalkion D_i eli $D_i \in WS(T_a)$, aktiivista transaktiota T_a täytyy siirtää eteenpäin. Eli $TI(T_a) = TI(T_a) \cup [TS(T_v) + 1, \infty[$.

Kaikki aikavälimuunnokset tehdään väliaikaisiin muuttujiin, jotka talletetaan *adjusted* tietorakenteeseen. Jos validoivan transaktion tarkistus onnistuu, tehdään aktiivisten transaktioiden aikavälien muutokset pysyviksi eli siirretään *adjusted* tietorakenteeseen talletetut aikavälit transaktioiden attribuutteihin. Jos aktiivisen transaktion aivaväli on tyhjä, peruutetaan ja uudelleenkäynnistetään kyseinen transaktio.

Validoivan transaktion käsittelemiin tietoalkioihin päivitetään lopuksi luku- ja kirjoitusajankohdat ja kirjoitetut alkiot kirjoitetaan fyysisesti tietokantaan.

```
conflict_detection(T_v)
{
  // Select final timestamp for the transaction
  TS(T_v) = min(validation_time, max(TI(T_v)));
  // Iterate for all objects read/written
  for ( ∀ D_i ∈ (RS(T_v) ∪ WS(T_v) ))
  {
    if (D_i ∈ RS(T_v))
      TI(T_v) = TI(T_v) ∩ [WTS(D_i), ∞[ ;
    if (D_i ∈ WS(T_v))
      TI(T_v) = TI(T_v) ∩ [WTS(D_i), ∞[ ∩ [RTS(D_i), ∞[ ;
    if (TI(T_v) == []) restart(T_v);

    // Conflict checking and TI calculation
    for ( ∀ T_a ∈ active_conflicting_transactions() )
    {
      if (D_i ∈ (RS(T_v) ∩ WS(T_a)))
        forward_adjustment(T_a, T_v, adjusted);

      if (D_i ∈ (WS(T_v) ∩ RS(T_a)))
        backward_adjustment(T_a, T_v, adjusted);

      if (D_i ∈ (WS(T_v) ∩ WS(T_a)))
        forward_adjustment(T_a, T_v, adjusted);
    }
  }

  // Adjust conflicting transactions
  for ( ∀ T_a ∈ adjusted)
  {
    TI(T_a) = adjusted.pop(T_a);

    if (TI(T_a) == []) restart(T_a);
  }

  // Update object timestamps
  for ( ∀ D_i ∈ (RS(T_v) ∪ WS(T_v) ))
  {
    if (D_i ∈ RS(T_v))
      RTS(D_i) = max(RTS(D_i), TS(T_v));
    if (D_i ∈ WS(T_v))
      WTS(D_i) = max(WTS(D_i), TS(T_v));
  }

  commit T_v to database;
}
```

Kuva 2: Konfliktien etsintäalgoritmi.

2.3 Konfliktien hallinta

Konfliktien hallinnassa sekä eteenpäinsiirrosa (forward adjustment) että taaksepäinsiirrossa (backward adjustment) määritellään ensin kriittisimmän transaktion kriittisyystaso. Tämän jälkeen selvitetään aktiivisen transaktion aikaväli, joko *adjusted* tietorakenteesta tai suoraan transaktion attribuuteista.

Kriittisyystason perusteella sitten tarkistetaan joudutaanko joko validoiva transaktio tai aktiivinen transaktio peruuttamaan konfliktin takia. Jos ei tarvitse, tehdään aikavälin muutos apumuuttujaan ja lisätään tämä tieto *adjusted* tietorakenteeseen (Kuva 3).

```
forward_adjustment(Ta, Tv, adjusted)
{
  criticality = max(Ta.criticality, Tv.criticality);

  if (Ta ∈ adjusted)
    TI = adjusted.pop(Ta);
  else
    TI = TI(Ta);

  if ( criticality >= NORMAL_CRIT_MIN && criticality <= NORMAL_CRIT_MAX )
      continue;
  if ( criticality >= MEDIUM_CRIT_MIN && criticality <= MEDIUM_CRIT_MAX ) {
      if ( Tv.criticality < Ta.criticality )
        if ( TI == ∅ )
          restart(Tv); /* Validation ends here /*
  }
  if ( criticality >= CRITICAL_CRIT_MIN && criticality <= CRITICAL_CRIT_MAX ) {
      if ( Tv.criticality < Ta.criticality )
          restart(Tv); /* Validation ends here /*
  }

  TI = TI ∩ [TS(Tv) + 1, ∞[ ;
  adjusted.push({(Ta, TI)});
}

backward_adjustment(Ta, Tv, adjusted)
{
  criticality = max(Ta.criticality, Tv.criticality);

  if (Ta ∈ adjusted)
    TI = adjusted.pop(Ta);
  else
    TI = TI(Ta);

  if ( criticality >= NORMAL_CRIT_MIN && criticality <= NORMAL_CRIT_MAX )
      continue;
  if ( criticality >= MEDIUM_CRIT_MIN && criticality <= MEDIUM_CRIT_MAX ) {
      if ( Tv.criticality < Ta.criticality )
          restart(Tv); /* Validation ends here /*
  }
  if ( criticality >= CRITICAL_CRIT_MIN && criticality <= CRITICAL_CRIT_MAX ) {
      if ( Tv.criticality < Ta.criticality )
          restart(Tv); /* Validation ends here /*
      if ( Tv.criticality > Ta.criticality )
          restart(Ta); return;
  }

  TI = TI ∩ [0, TS(Tv) − 1] ;
  adjusted.push({(Ta, TI)});
}
```

Kuva 3: Konfliktien hallinta-algoritmi.

2.4 Oikeellisuustodistus

Tässä kappaleessa todistetaan että OCC-IDATI menetelmä muodostaa sarjallistuvia historioita. Jotta OCC-IDATI menetelmän muodostama historia H olisi sarjallistuva, täytyy todistaa että historian H muodostama sarjallistuvuusverkko $SG(H)$ on syklitön [BHG87]. Siksi seuraava lause osoittaa että jos validoivan transaktion ja aktiivisen transaktion välillä on konflikti, on niiden välillä myös täydellinen järjestys aikavälien muodossa.

LEMMA 2.1 *Olkoot T_1 ja T_2 transaktioita OCC-IDATI menetelmän muodostamassa historiassa H ja $SG(H)$ sen sarjallistuvuusverkko. Jos sarjallistuvuusverkossa $SG(H)$ on särmä $T_1 \rightarrow T_2$, niin tällöin $TS(T_1) < TS(T_2)$.*

Todistus: *Jos sarjallistuvuusverkossa $SG(H)$ on särmä $T_1 \rightarrow T_2$, täytyy olla olemassa yksi tai useampia konfliktoivia operaatioita joiden tyyppi on yksi seuraavista kolmesta:*

1. *$r_1[x] \rightarrow w_2[x]$: Tällöin T_1 sitoutuu ennenkuin T_2 saavuttaa validointivaiheensa, koska operaatioon $r_1[x]$ ei vaikuta operaatio $w_2[x]$. Kirjoitusoperaatiolle $w_2[x]$, OCC-IDATI siirtää $TI(T_2)$:n seuraamaan arvoa $RTS(x)$, joka on suurempi tai yhtäsuuri kuin $TS(T_1)$. Siis, $TS(T_1) \leq RTS(x) < TS(T_2)$. Siksi, $TS(T_1) < TS(T_2)$.*

2. *$w_1[x] \rightarrow r_2[x]$: Tällöin transaktion T_1:n kirjoitusvaihe päättyy ennenkuin T_2:n lukuvaihe suorittaa operaation $r_2[x]$. Operaatiossa $r_2[x]$, OCC-IDATI siirtää $TI(T_2)$:n seuraamaan arvoa $WTS(x)$, joka on suurempi tai yhtäsuuri kuin $TS(T_1)$. Siis, $TS(T_1) \leq WTS(x) < TS(T_2)$. Siksi, $TS(T_1) < TS(T_2)$.*

3. *$w_1[x] \rightarrow w_2[x]$: Tällöin transaktion T_1:n kirjoitusvaihe päättyy ennen operaation $w_2[x]$ suoritusta. Operaatiolle $w_2[x]$, OCC-IDATI siirtää $TI(T_2)$:n seuraamaan arvoa $WTS(x)$, joka on suurempi tai yhtäsuuri kuin $TS(T_1)$. Siis, $TS(T_1) \leq WTS(x) < TS(T_2)$. Siksi, $TS(T_1) < TS(T_2)$. $\square$*

Osoittaakseen että OCC-IDATI menetelmän tuottaa vain sarjallistuvia historioita, oletetaan että menetelmä muodostaa syklin sarjallistuvuusverkkoon ja osoitetaan että tämä oletus johtaa ristiriitaan.

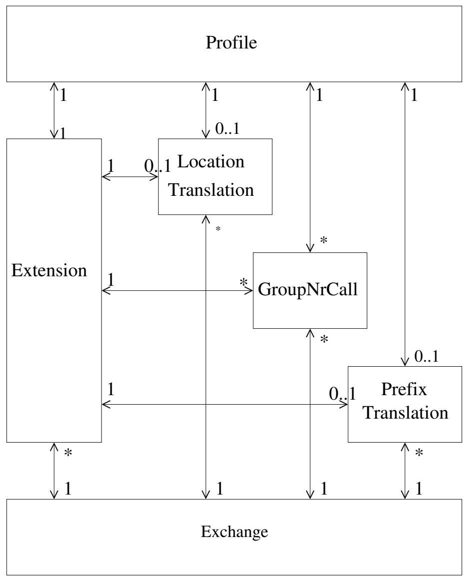
TEOREEMA 2.1 *Jokainen OCC-IDATi menetelmän muodostaqma historia on sarjallistuva.*

Todistus: *Olkoon H OCC-IDATi menetelmällä muodostettu historia ja $SG(H)$ sen sarjallistuvuusverkko. Tehdään vastaoletus että $SG(H)$ sisältää syklin $T_1 \rightarrow T_2 \rightarrow ... \rightarrow T_n \rightarrow T_1$, where $n > 1$. Lemman 2.1 perusteella siis $TS(T_1) < TS(T_2) < ... < TS(T_n) < TS(T_1)$.*

Helpolla induktiolla nähdään, että $TS(T_1) < TS(T_1)$. Tämä on ristiriita. Siksi sykliä ei voi esiintyä $SG(H)$:ssa ja OCC-IDATI tuottaa vain sarjallistuvia historioita. $\square$

3 Testiympäristö ja testitulokset

Testitietokanta mallintaa tyypillista älyverkkopalvelua (Intelligent Network service, IN) [ITU93]. Tietokanta mallintaa virtuaalista paikallista verkkoa (Virtual Private Network). Tietokanta sisältää $30\,000$ oliota. Luokat on kuvattu Kuvassa 4.



Kuva 4: Tietokannan kaavio.

Seuraavassa lyhyesti esitellään jokaisen luokan sisältö. Kuusi luokkaa on määritelty:

1. Luokka *Profile* sisältää kaikki tarvittavat tiedot käyttäjien hallintaan.
2. Luokka *Exchange* sisältää tarvittavan informaation järjestelmän vaihteista.
3. Luokkaa *PrefixTranslation* käytetään antamaan puhelinnumeroille lyhytnumero.
4. Luokka *Extension* määrittelee yhden liittymän tiedot.
5. Luokan *LocationTranslation* avulla käyttäjä voi määritellä mihin numeroon hän haluaa puhelut siirrettäviksi.

6. Luokan *GroupNrCall* avulla voidaan useita liittymiä yhdistää yhden uniikin hakemistonumeron numeroon. Esimerkiksi 112.

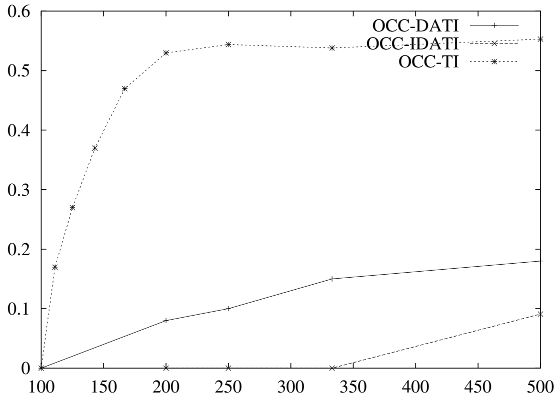
Seuraavia transaktioita on käytetty testeissä:

- **Find Subscriber** transaktiolla etsitään tietty käyttäjä tietokannasta. Transaktio palauttaa käyttäjän puhelinnumeron. Tämä transaktio on kriittinen.
- **Get New Destination Number** transaktion avulla selvitetään annetun numeron avulla se puhelinnumero, johon puhelu tulisi yhdistää. Syötteenä voi olla lyhytnumero, siirretty numero tai ryhmänumero. Transaktio palauttaa numeron, johon puhely yhdistetään. Tämän transaktion kriittisyys on medium.
- **Get Subscribers Basic Data** transaktion avulla haetaan käyttäjän perustiedot. Tämän transaktion kriittisyys on medium.
- **Update Subscriber Data** transaktiolla päivitetään käyttäjän perustietoja. Tämän transaktion kriittisyys on normal.

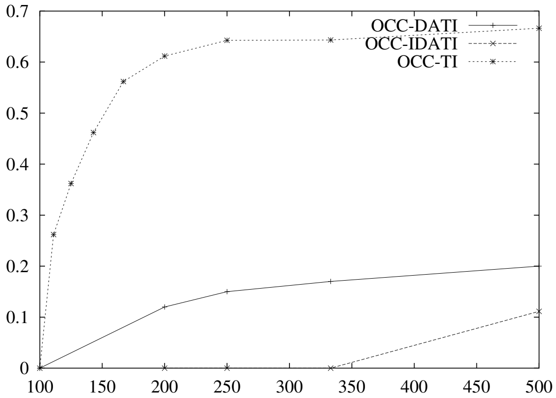
Käytännön testauksessa on ajettu joukko testejä, joiden avulla pyritään selvittämään kuinka ehdotettu OCC-IDATI menetelmä toimii käytännössä. Testialustana on käytetty tosiaikatietokantaprototyyppiä RODAIN *Real-Time Object-Oriented Database Architecture for Intelligent Networks* (RODAIN) [KNPR97, LNPR99, NKR97], joka on tosiaikainen, oliopohjainen ja vikasietoinen tietokannanhallintajärjestelmä. RODAIN prototyyppi on keskusmuistitietokanta, joka käyttää prioriteettipohjaista skedulointia ja optimistista samanaikaisuudenhallintamenetelmää. Kaikki testit on suoritettu käyttäen Pentium Pro 200MHz tietokonetta 64 MB:n keskusmuistilla ja käyttäen Chorus/ClassiX tosiaikakäyttöjärjestelmää [Pou94].

Testiympäristössä transaktiot saapuvat erityisen käyttörajapinnan kautta, joka vastaanottaa transaktiot valmiiksi generoidusta tiedostosta. Jokainen testi sisältää 10 000:ta transaktiota ja testi suoritetaan 20 kertaa. Ilmoitetut arvot ovat suorituskertojen keskiarvoja. Testauksessa ehdotettua menetelmää OCC-IDATI on verrattu OCC-DATI [LR99] ja OCC-TI [LS93] menetelmiin.

Ensimmäisessä testijoukossa on käytetty kiinteätä kirjoitusosuutta ja transaktioiden saapumistiheys on vaihteleva. OCC-IDATI osoittautuu kaikissa testeissa tarjoavan parhaan tehokkuuden (Kuva 5).



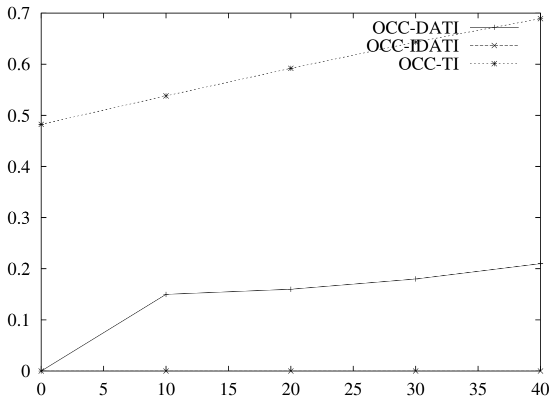
(a) 10% kirjoituksia



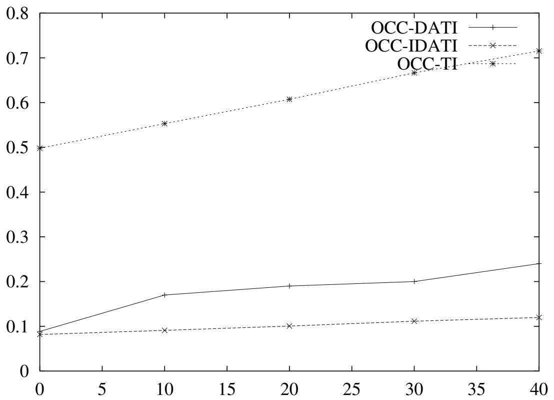
(b) 30% kirjoituksia

Kuva 5: Vertailu vaihtelevalla saapumistiheydellä.

Toisessa testijoukossa saapumistiheys on vakio ja transaktioiden kirjoitustodennäköisyys vaihteleva (Kuva 6). Jälleen OCC-IDATI osoittautuu tehokkaimmaksi. Tämä osoittaa että OCC-IDATI pystyy parhaiten sopeutumaan muuttuvaan työkuormaan ja siten OCC-IDATI soveltuu parhaiten dynaamiseen järjestelmään.



(a) 333 transaktiota/s



(b) 500 transaktiota/s

Kuva 6: Vertailu vaihtelevalla kirjoitustodennäköisyydellä.

4 Yhteenveto

Vaikka optimistiset menetelmät on osoitettu tehokkaimmiksi tosiaikatietokannoissa, niissä ongelmaksi muodostuu turhat uudelleenkäynnistykset ja korkea peruutustyö. Tässä esitelmässä on ehdotettu uutta optimistista samanaikaisuudenhallintamenetelmää, joka sekä vähentää turhia uudelleenkäynnistyksiä että adaptoituu kulloiseenkin työkuormaan. Menetelmä suosii kriittisiä transaktioita ja siten soveltuu paremmin tosiaikatietokantoihin. Käytännön testit osoittavat, että ehdotettu menetelmä on tehokkaampi kuin aikaisemmat tunnetut menetelmät ja siten soveltuu hyvin dynaamisiin tosiaikatietokantoihin.

Viitteet

[AGM88] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions. *ACM SIGMOD Record*, 17(1):71–81, March 1988.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[DS96] A. Datta and S. H. Son. A study of concurrency control in real-time active database systems. Tech. report, Department of MIS, University of Arizona, Tucson, 1996.

[DSK00] A. Datta, S. H. Son, and V. Kumar. Is a bird in the hand worth more than two in the bush? limitations of priority cognizance in conflict resolution for firm real-time database systems. *IEEE Transactions on Computers*, 49(5):482–502, May 2000.

[DVSK97] A. Datta, I. R. Viguier, S. H Son, and V. Kumar. A study of priority cognizance in conflict resolution for firm real time database systems. In *Proceedings of the Second International Workshop on Real-Time Databases: Issues and Applications*, pages 167–180, Burlington, Vermont, USA, 1997. Kluwer Academic Publishers.

[EGLT76] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The notions of consistency and predicate locks in a database system. *Communications of the ACM*, 19(11):624–633, November 1976.

[Här84] T. Härder. Observations on optimistic concurrency control schemes. *Information Systems*, 9(2):111–120, 1984.

[HCJ96] D. Hong, S. Chakravarthy, and T. Johnson. Locking based concurrency control for integrated real-time database systems. In *Proceedings of the First International Workshop on Real-Time Databases: Issues and Applications*, Newport Beach, California, March 7-8, 1996.

[HCL90a] J. R. Haritsa, M. J. Carey, and M. Livny. Dynamic real-time optimistic concurrency control. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pages 94–103, Lake Buena Vista, Florida, USA, 1990. IEEE Computer Society Press.

[HCL90b] J. R. Haritsa, M. J. Carey, and M. Livny. On being optimistic about real-time constraints. In *Proceedings of the 9th ACM Symposium on Principles of Database Systems*, pages 331–343, Nashville, Tennessee, 1990. ACM Press.

[HSRT91] J. Huang, J. A. Stankovic, K. Ramamritham, and D. Towsley. Experimental evaluation of real-time optimistic concurrency control schemes. In *Proceedings of the 17th VLDB Conference*, pages 35–46, Barcelona, Catalonia, Spain, September 1991. Morgan Kaufmann.

[ITU93] ITU. *Introduction to Intelligent Network Capability Set 1. Recommendation Q.1211*. ITU, International Telecommunications Union, Geneva, Switzerland, 1993.

[KNPR97] J. Kiviniemi, T. Niklander, P. Porkka, and K. Raatikainen. Transaction processing in the RODAIN real-time database system. In A. Bestavros and V. Fay-Wolfe, editors, *Real-Time Database and Information Systems*, pages 355–375, London, 1997. Kluwer Academic Publishers.

[KR81] H. T. Kung and J. T. Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems*, 6(2):213–226, June 1981.

[Lin00] J. Lindström. Extensions to optimistic concurrency control with time intervals. In *Proceedings of 7th International Conference on Real-Time Computing Systems and Applications*, pages 108–115, Cheju Island, South Korea, 2000. IEEE Computer Society Press.

[LLH95a] K.-W. Lam, K.-Y. Lam, and S. Hung. An efficient real-time optimistic concurrency control protocol. In *Proceedings of the First International Workshop on Active and Real-Time Database Systems*, pages 209–225, Skövde, Sweden, 1995. Springer.

[LLH95b] K.-W. Lam, K.-Y. Lam, and S. Hung. Real-time optimistic concurrency control protocol with dynamic adjustment of serialization order. In *Proceedings of the IEEE Real-Time Technology and Application Symposium*, pages 174–179, Chigago, Illinois, 1995. IEEE Computer Society Press.

[LNPR99] J. Lindström, T. Niklander, P. Porkka, and K. Raatikainen. A distributed real-time main-memory database for telecommunication. In *Databases in Telecommunications*, Lecture Notes in Computer Science, vol 1819, pages 158–173, Edinburgh, UK, Co-located with VLDB-99, 1999.

[LR99] J. Lindström and K. Raatikainen. Dynamic adjustment of serialization order using timestamp intervals in real-time databases. In *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications*, pages 13–20, Hong Kong, China, 1999. IEEE Computer Society Press.

[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.

[LS96] J. Lee and S. H. Son. Performance of concurrency control algorithms for real-time database systems. In V. Kumar, editor, *Performance of Concurrency Control Mechanisms in Centralized Database Systems*, pages 429–460. Prentice-Hall, 1996.

[NKR97] T. Niklander, J. Kiviniemi, and K. Raatikainen. A real-time database for future telecommunication services. In D. Gaïti, editor, *Intelligent Networks and Intelligence in Networks*, pages 413–430, Paris, France, 1997. Chapman & Hall.

[Pou94] Dick Pountain. The Chorus microkernel. *Byte*, pages 131–138, January 1994.

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

[SPL92] S. H. Son, S. Park, and Y. Lin. An integrated real-time locking protocol. In *Proceedings of the 8th International Conference on Data Engineering*, pages 527–534, Tempe, Arizona, USA, 1992. IEEE Computer Society Press.

[SSH99] J. A. Stankovic, S. H. Son, and J. Hansson. Misconceptions about real-time databases. *IEEE Computer*, 32(6):29–36, June 1999.