Hajautettujen tietokantojen ongelmia

Jan Lindström

Hajautetut sovellukset -seminaari

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

Sisältö

1 Johdanto

Tietokanta (TK) on kokoelma tiettyä kohdetta kuvaavia tietoja, joita yksi tai useampi tietosysteemi käyttää tai päivittää. *Tietokannanhallintajärjestelmä* (TKHJ) on tietokannan hallintaan tarkoitettu yleisohjelmisto, jonka tavoitteena on tietojen hallinnan yhteinäistäminen: tietojen saanti perustuu niiden loogiseen rakenteeseen ei talletusrakenteeseen.

Hajautetulla tietojenkäsittelyllä tarkoitetaan tietojenkäsittelyä, joka jaetaan tietokoneverkon eri osissa tapahtuvaksi. Tietokoneverkko on solmuista ja niiden välisistä tietolinkeistä koostuva, yleensä alueellisesti hajallaan sijaitseva järjestelmä. Solmulla tarkoitetaan tietokonejärjestelmää, joka pystyy tietolinkin välityksellä kommunikoimaan muiden vastaavien järjestelmien kanssa. Tavoitteena on mahdollistaa ohjelmien ja tietojen hajauttaminen, so. jakaminen useiden toisiinsa liitettyjen järjestelmien kesken.

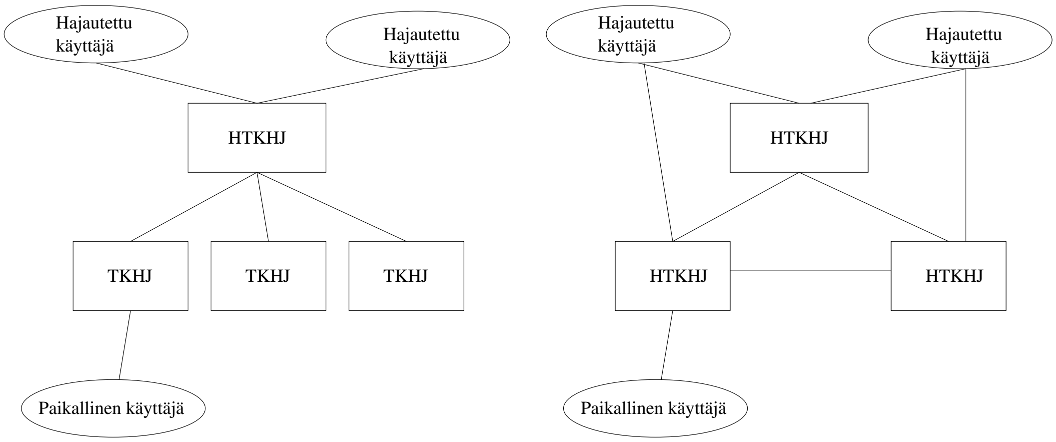
Hajautettu tietokanta (HTK) on kokoelma loogisesti yhteenliittyviä tietokantoja, jotka ovat hajautettu yli tietoverkon. Hajautetut tietokannat pyrkivät yhdistämään näiden tavallaan vastakkaisten lähestymistapojen edut ja tarjoamaan hajautettujen tietojen yhteinäisen käsittelymahdollisuuden. *Hajautettu tietokannanhallintajärjestelmä* (HTKHJ) on ohjelmisto, joka hallinnoi hajautettua tietokantaa ja mahdollistaa käyttäjälle hajautusriippumattoman rajapinnan. Hajautetussa tietokannassa tiedot on sijoiteltu tietokoneverkon eri solmuihin.

Tavoitteena hajautetuissa tietokannanhallintajärjestelmissä on *hajautuksen tuntumattomuus* (data transparency) eli mahdollisuus käyttää hajautettua tietokantaa tarvitsematta välittää tiedon fyysisestä sijainnista verkossa. Tiedon siirto eri solmujen välillä on kallista. Siksi tiedon siirtoa tietokoneverkossa solmujen välillä pyritään minimoimaan.

Hajautetut tietokannat ovat hyödyllisiä, koska ne tarjoavat tehokkuutta, käyttäjäläheisyyttä, luotettavuutta, joustavuutta ja modulaarisuutta [ÖV99]. Tehokkuus parantuu, koska tiedot talletetaan niiden synty- tai käyttöpaikalle jolloin tiedonsiirtokustannuksen pienenevät. Hajautuksen avulla saadaan käyttöön myös useita prosessoreja, jolloin rinnakkaisuusaste kasvaa ja vuorokäyttöisyyden aiheuttama ylimäärä pienee.

Käyttäjäläheisyys paranee, koska paikallisten käsittelymahdollisuuksien lisäys pienentää muutosvastarintaa. Luotettavuus kasvaa, koska yhden solmun kaatuminen ei kaada koko järjestelmää. Myös varajärjestelyjen mahdollisuus tietolinkeissä ja tietojenkäsittelylaitteissa lisää luotettavuutta. Hajautettuun järjestelmään on helppo lisätä uusia solmuja ja solmu voidaan korvata toisella, jolloin järjestelmän joustavuus kasvaa. Näin järjestelmästä muodostuu myös modulaarinen.

Hajautetulla tietokannanhallintajärjestelmällä on kaksi arkkitehtuurillista perusratkaisua: *hierarkinen hajautus* ja *täydellinen hajautus* [ÖV99]. Hierarkisessa hajautuksessa vain tiedot ovat hajautettu. Järjestelmä sisältää keskussolmun ja useita paikallisia tietokannanhallintajärjestemiä (Kuva 1(a)). Keskussolmusta saattaa tässä mallissa muodostua pullonkaula ja keskussolmun kaatuminen johtaa koko järjestelmän kaatumiseen. Hierarkinen hajautus on kuitenkin helpompi toteuttaa kuin täysin hajautettu järjestelmä. Täysin hajautetussa arkkitehtuurissa kaikki solmut ovat samanarvoisia (Kuva 1(b)). Täysin hajautetussa arkkitehtuurissa saadaan täysi hyöty hajautuksesta, mutta toteutusongelmat ovat suuria. Suuntaus on kohti tämäntyyppisiä järjestelmiä.



(a) Hierarkinen arkkitehtuuri (b) Hajautettu arkkitehtuuri

Kuva 1: Hajautettut tietokanta-arkkitehtuurit.

Tässä esityksessä esitellään millaisia uusia ongelmia syntyy tietokannan hajauttamisesta. Esityksen rakenne on seuraava. Kappaleessa 2 esitellään hajautettujen tietokantojen ongelmista: hajautettujen tietokantojen suunnittelu, kyselynkäsittely, samanaikaisuudenhallinta, luotettavuus, käyttöjärjestelmätuki ja heterogeeniset tietokannat. Tietoturva jätetään käsittelemättä, koska siitä on ollut jo esitelmä [Lai01]. Tarkempaan käsittelyyn näistä aiheista on valittu hajautettu kyselynkäsittely (Kappale 3), koska siinä on useita mielenkiintoisia ja tutkimuksellisestikkin avoimia ongelmia. Kappaleessa 4 tutkitaan mihin tilaan tietokannanhallintajärjestelmät ovat kehittyneet ja mitä ongelmia tästä kehityksesta on seurannut. Lisäksi esitellään aivan uusi arkkitehtuurimalli tietokantoihin. Lopuksi esitetään yhteenveto kappaleessa 5.

2 Hajautettujen tietokantojen ongelmia

Tässä kappaleessa esitellään joukko tutkimuksessa esitettyjä hajautettujen tietokantojen ongelma-alueita, joita ovat hajautettujen tietokantojen suunnittely, kyselynkäsittely, samanaikaisuudenhallinta, luotettavuus, käyttöjärjestelmätuki ja heterogeeniset tietokannat [ÖV99].

Tavoitteena on esitellä ongelmien perusperiaatteet ja perusteluita, miksi esitetty ongelma on relevantti hajautettujen tietokantojen kannalta. Tiettyihin ongelmiin on esitetty myös yleisimmin käytössä oleva ratkaisu.

2.1 Hajautetun tietokannan suunnittelu

Hajautetun tietokannan suunnittelun peruskysymyksenä on kuinka tietokanta ja tietokantaa käsittelevät ohjelmistot sijoitetaan hajautetussa järjestelmässä [ÖV99]. Tiedon sijoitteluun on olemassa kaksi perusvaihtoehtoa: *ositettu* (partioned) ja *toisinnettu* (replicated). Osittamismenetelmässä tietokanta jaetaan toisistaan erillisiin osiin ja nämä osat sijoitetaan eri solmuihin. Toisinnettu tietokanta voi ollla joko *täysin toisinnettu* jolloin koko tietokanta on talletettuna jokaiseen solmuun tai *osittain toisinnettu* jolloin jokainen tietokannan osa on talletettuna useampaan kuin yhteen solmuun.

Suunnittelun peruskysymyksiä ovat: *fragmentoiminen* ja *hajauttaminen* [ÖV99, Kappale 5]. Fragmentoinnissa tietokanta jaetaan osiin, joita kutsutaan fragmenteiksi. Hajauttamisessa pyritään etsimään optimaalinen fragmenttien sijoittelu. Fragmentointiin on kaksi perusmenetelmää: *horisontaalinen fragmentointi* ja *vertikaalinen fragmentointi*. Horisontaalisessa fragmentoinnissa relaatio jaetaan jonkin attribuuttiarvon tai joidenkin attribuuttiarvojen perusteella horisontaalisesti fragmentteihin (Kuva 2(b)). Vertikaalisessa fragmentoinnissa relaation attribuutit jaetaan fragmentteihin kuitenkin siten että relaation avainattribuutit ovat läsnä jokaisessa fragmentissa (Kuva 2(c)). Alueen tutkimus koostuu pääosin matemaattisesta ohjelmoinnista, jonka tavoitteena on minimoida tietokannan talletuksen, tapahtumakäsittelyn ja tietoliikenteen yhteenlaskettua kustannusta. Yleisessä tapauksessa ongelma on NP-täydellinen. Siksi esitetyt ratkaisut perustuvat heuristiikkaan.

ESIMERKKI 2.1 *Olkoon relaatio $R = (Hetu, Nimi, Osoite, Puhelin)$, jonka sisällöstä on otos Kuvassa 2(a). Tällöin relaation R horisontaalinen fragmentointi relaatioihin R_1 ja R_2 ehdoilla $R_1 = \sigma_{Hetu>0 \wedge Hetu<200000}(R)$ ja $R_2 = \sigma_{Hetu>200000 \wedge Hetu<300000}(R)$ on Kuvan 2(b) mukainen. Vastaavasti relaation R vertikaalinen fragmentointi relaatioihin R_1 ja R_2 ehdoilla $R_1 = \pi_{Hetu,Nimi}(R)$ ja $R_2 = \pi_{Hetu,Osoite,Puhelin}(R)$ on Kuvan 2(c) mukainen.*

Hetu	Nimi	Osoite	Puhelinnumero
123456	Pekka	Kallio	123456
123661	Tiina	Kivelä	653211
125749	Sari	Tornio	9765433
214641	Matti	Muonio	1256767
215741	Pirjo	Mattila	2157656

(a) Esimerkkirelaatio R

Hetu	Nimi	Osoite	Puhelinnumero
123456	Pekka	Kallio	123456
123661	Tiina	Kivelä	653211
125749	Sari	Tornio	9765433

Hetu	Nimi	Osoite	Puhelinnumero
214641	Matti	Muonio	1256767
215741	Pirjo	Mattila	2157656

(b) Horisontaalinen fragmentointi

Hetu	Nimi
123456	Pekka
123661	Tiina
125749	Sari
214641	Matti
215741	Pirjo

Hetu	Osoite	Puhelinnumero
123456	Kallio	123456
123661	Kivelä	653211
125749	Tornio	9765433
214641	Muonio	1256767
215741	Mattila	2157656

(c) Vertikaalinen fragmentointi

Kuva 2: Esimerkin relaatio ja sen horisontaalinen ja vertikaalinen fragmentointi.

2.2 Hajautettu kyselynkäsittely

Kyselynkäsittely (query processing) tarkoittaa transaktiota generoivaan tietokantaohjelmaan sisältyvän kyselyn tai päivitysoperaation laskentastrategian valintaa ja valitun strategian mukaista suoritusta [ÖV99, Kappale7]. Laskentastrategiat riippuvat tietokannanhallintajärjestelmän tarjoamista yksittäisten tietokantaoperaatioiden laskentamenetelmistä sekä fyysisen tietokannan rakenteesta.

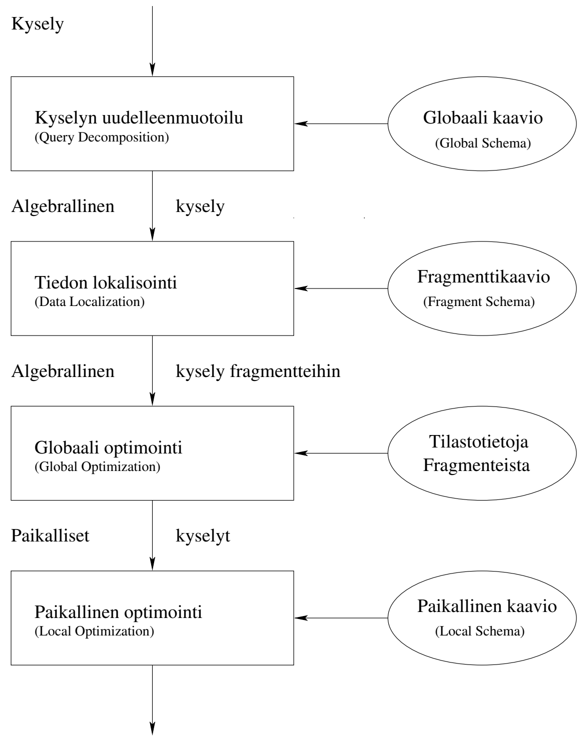
Mitä monipuolisempi rakenne fyysisellä tietokannalla on, so. mitä useampia saantipolkuja siihen sisältyy, sitä useampia vaihtoehtoisia laskentastrategioita kyselyille löytyy. Korkeatasoisella kyselykielellä (esimerkiksi SQL) ilmaistulle kyselylle on yleensä useita eri toteutusvaihtoehtoja jo siitäkin syystä, ettei kyselyn semantiikka aina edes kiinnitä tarkasti sitä transaktiomallin mukaista monikkokohtaisten operaatioiden jonoa, joka kyselystä generoituu.

Kyselyn optimoinnin (query optimization) keskeisenä periaatteena on pyrkiä lukemaan kyselyn argumenttirelaatioista vain ne monikot, jotka jollain tavoin myötävaikuttavat (contribute) kyselyn tulokseen, so. joista jokin kyselyn vastausmonikko saadaan kyselyn operaatioilla.

Hajautetussa kyselynkäsittelyssä perusongelmana on kuinka valitaan jokaiselle kyselyn osalle kyselynsuoritusstategia, jolla on mahdollisimman pieni kustannus riippumatta kustannuksen määritelmästä. Kustannukseen vaikuttavat tiedon hajautus, tietoliikennekustannus ja riittävän paikallisen tiedon puute. Ongelma on NP-täydellinen ja siksi ratkaisut perustuvat heuristiikkaan.

Hajautetun tietokannan käyttäjä käyttää näennäisiä eli loogisia relaatioita, jotka voivat olla ositettuja eri pisteissä sijaitseviin todellisiin eli fyysisiin relaatioihin. Kyselyn laskemiseksi alkuperäisen kyselylausekkeen loogiset relaatiot korvataan määrittelyillään, kunnes kaikki lausekkeen argumentit ovat fyysisiä relaatioita.

Kyselynkäsittely muodostuu useammasta osaongelmasta (Kuva 3), joista muodostuu nelikerroksinen rakenne [ÖV99, Kappale 7]. Nämä kerrokset ovat *kyselyn uudelleenmuotoilu, tiedon lokalisointi, globaali kyselynoptimointi* ja *paikallinen kyselynoptimointi*.



Kuva 3: Kerrosmallinen kyselynkäsittely.

Kyselyn uudelleenmuotoilussa hajautettu kysely muotoillaan algebralliseksi kyselyksi globaaleihin relaatioihin. Kyselyn uudelleenmuotoilu koostuu neljästä peräkkäisestä operaatiosta [ÖV99, Kappale 8]:

1. Kyselyn uudelleenkirjoitus normalisoituun muotoon. Normalisoinnissa kyselyn osia manipuloidaan käyttäen loogisten operaattoreiden prioriteetteja.

2. Normalisoitu kysely analysoidaan semanttisesti, jolloin virheelliset kyselyt voidaan havaita mahdollisimman aikaisessa vaiheessa.

3. Kyselyn yksinkertaistus.

4. Kyselyn uudelleenkirjoitus algebralliseen muotoon.

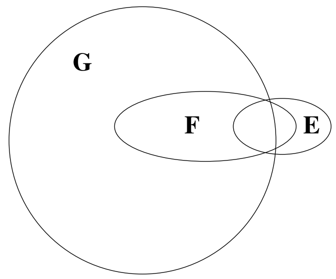
Tiedon lokalisoinnin syötteenä on algebrallinen kysely globaaleihin relaatioihin. Tiedon lokalisoinnin päätehtävänä on lokalisoida kyselyn data käyttäen tiedon hajautusinformaatiota.

Lokalisointi tehdään selvittämällä, mitkä fragmentit tarvitaan kyselyn laskemiseen ja muuntamalla kysely näihin fragmentteihin.

Globaalin optimoinnin syötteenä on kysely fragmentteihin. Globaalin optimoinnin tavoitteena on löytää kyselylle suoritusstrategia, joka olisi mahdollisimman optimaalinen [ÖV99, Kappale 9].

Paikallinen optimointi saa syötteenä paikallisen kyselyn, jolle suoritetaan paikallisen kyselyn optimointi.

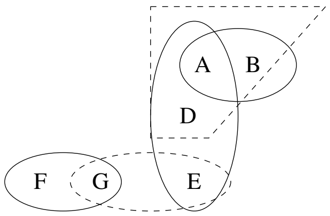
Hajautettu kysely voidaan mallintaa hyperverkolla. *Hyperverkko* on verkko, joka voi koostua aliverkoista. Esimerkiksi olkoot E ja F hyperverkon G särmiä siten, että joukon $E \setminus F$ solmut eivät esiinny muusa G:n särmässä kuin E:ssä (Kuva 4).



Kuva 4: Esimerkin hyperverkko.

Silloin E on G:n *korva* (ear) särmän F suhteen ja E:n poisto G:stä on *korvan poisto* (ear removal) G:stä F:n hyväksi. Yksisärmäisen verkon ainoan särmän poisto on myös korvan poisto. Hyperverkon *korvanpoistoalgoritmi* tai GYO-sievennys (GYO reduction; Graham, Yu & Ozsoyoglu) [Gra79a, YÖ79] saadaan poistamalla G:stä korvia, kunnes jäljelle jää korvaton verkko. Hyperverkko on syklitön, jos sen GYO-sievennys on tyhjä, muutoin syklinen. Voidaan osoittaa, että syklisyyden määritelmä on riippumaton korvien poistojärjestyksestä. GYO-sievennys on yksi tunnetuista heuristisista hajautettujen kyselyiden optimointimenetelmistä.

ESIMERKKI 2.2 *Kysely* $\pi_{A,D,E,G}(\sigma_{A=C \wedge (B<C \vee B<D) \wedge G<E \wedge F=f \wedge B=b}(AB \times CDE \times FG))$ *muodostaa kuvan 5 mukaisen hyperverkon.*



Kuva 5: Esimerkkikyselyn hyperverkko.

2.3 Hajautettu samanaikaisuudenhallinta

Hajautettu tietokanta koostuu joukosta tietokoneverkon toisiinsa kytkemiä solmuja, joista kukin ylläpitää omaa tietokantajärjestelmäänsä [ÖV99, Kappale 10]. Jokainen solmu pystyy käsittelemään paikallisia transaktioita, jotka operoivat ainoastaan kyseisessä solmussa säilytettäviin tietoalkioihin. Solmu voi myös osallistua *globaaleihin transaktioihin*, jotka operoivat useampien eri solmujen tietoihin.

Globaali transaktio, joka käsittelee solmuissa $s_0, s_1, ..., s_n$ säilytettäviin tietoalkioita, koostuu *alitransaktioista* $T_=, T_1, ..., T_n$, missä kukin T_i käsittelee ainoastaan solmun s_i tietoalkioita. Globaali transaktio alitransaktioidensa joukkona on itse asiassa joukko paikallisia transaktioita, joiden suoritusta *koordinoidaan*.

Jokaisessa solmussa on oma paikallinen transaktiohallitsin, jonka tehtävänä on hallita solmun paikallisia transaktioita sekä solmun tietoihin operoivia globaalien transaktioiden alitransaktioita. Solmun *transaktiokoordinoijan* tehtävänä on koordinoida niiden globaalien transaktioiden suoritusta, jotka on käynnistetty kyseisessä solmussa.

Solmun transaktiohallitsin ylläpitää normaaliin tapaan lokia solmussa toimivien transktioiden solmun tietoihin kohdistamista päivityksistä sekä hallitsee transaktioiden samanaikaisuutta. Sen lisäksi solmun transaktiohallitsin osallistuu kyseisessä solmussa olevien, globaalien transaktioiden suorituksen koordinoitiin.

Solmun transaktiokoordinoijan tehtävänä on käynnistetyn globaalin transaktion pilkkominen alitransaktioiksi niihin solmuihin, joiden tietoihin transaktiossa operoidaan, sekä koordinoida näiden alitransaktioiden suorituksen päättymistä. Transaktioiden päättymisen koordinoinnin tulee johtaa joko kaikkien alitransaktioiden sitoutumiseen tai kaikkien alitransaktioiden keskeytymiseen ja peruuntumiseen.

Yleisin käytössä oleva globaalien transaktioiden sitoutumiskäytäntö on nimeltään *kaksivaiheinen sitoutumiskäytäntö* (two-phase commit protocol, 2PC) [ÖV99, Kappale 11]. Kaksivaiheisessa sitoutumiskäytännössä transaktion suoritusta koordinoidaan transaktion aloitussolmussa. Transaktion sitoutumisoperaation suoritus sisältää kaksi vaihetta:

1. Koordinaatiosolmu tiedustelee kaikilta muilta suoritussolmuilta ovatko ne valmiit sitouttamaan kyseisen tapahtuman (prepare).

   Mikäli kaikki solmut ovat yksimielisiä siitä, että tapahtuma voi sitoutua, siirrytään vaiheeseen 2.

2. Koordinaatiosolmu päättää sitouttaa kyseisen tapahtuman suorittamalla sitoutumisen vaatimat toimenpiteet paikalliseen tietokantaan ja viestittää sitoumispäätöksen kaikkiin suoritussolmuihin.

   Kukin solmu on nyt velvolinen lupauksensa mukaisesti suorittamaan kyseisen tapahtuman sitoutumisen vaatimat toimenpiteet paikallisiin tietokantoihinsa.

Samanaikaisuudenhallinta hajautetussa ympäristössä on siis erilaista kuin keskitetyssä ympäristössä. Ei riitä yksittäisen tietokannan eheyden varmistaminen vaan vaaditaan useamman tietokannan keskinäistä eheyttä. Ehto, joka vaatii usean saman tietoalkion arvon eheyttä kutsutaan *keskinäiseksi eheydeksi* (mutual consistency).

Samanaikaisuuden paikallisesti voi taata useilla eri menetelmillä. Perusmenetelmiä on kaksi: *pessimistinen* ja *optimistinen*. Pessimistisessä menetelmässä transaktion tulee pyytää ennen tietoalkion käsittelyä synkronointia. Optimistisessa menetelmässä ensin suoritetaan tietokantaoperaatiot ja sen jälkeen tarkistetaan säilyykö tietokannan eheys kyseisessä suorituksessa. Molemmissa menetelmissä voidaan käyttää lukkoja primitiiveinä, joilla synkronointi taataan. Lukitusmenetelmä perustuu poissulkemiseen. Toinen perustemenetelmä on aikaleimamenetelmä, jossa tapahtumat suoritetaan tietyssä järjestyksessä.

Lukitusmenetelmään hajautetussa ympäristössä liittyy riski lukkiumiin. Lukkiumaongelma on samantyyppinen kuin käyttöjärjestelmistä tavattava. Käyttäjien välien kilpailu pääsystä saman resurssiin voi johtaa lukkiumaan jos synkronointiprimitiivinä käytetään lukkoja. Tietokannoissakin voidaan käyttää käyttöjärjestelmistä tuttuja ratkaisuvaihtoehtoja: *ehkäiseminen* (prevention), *välttäminen* (avoidance) ja *toipuva havaitseminen*.

Hajautetussa sitoutumiskäytännössä perusoletuksena on tietoliikenteen luotettavuus. Jos tietoliikenteen luotettavuudesta ei voida tehdä oletusta, muodostuu sitoutumisongelma hyvin vaikeaksi. Ongelmaa kutsutaan *yksimielisyysongelmaksi* (consensus problem). Voidaankin osoittaa, että ei ole olemassa algoritmiä joka ratkaisee hajautetun sitoutumisongelman yleisessä tapauksessa [Lyn96]. Kyseinen mahdottomuustulos osoittaa perustavalaatuisen rajoituksen hajautetuissa järjestelmissä. On hyvin vähän tehtävissä hajautetun sitoutumisongelman ratkaisemiseksi jos tietoliikenne on epäluotettavaa.

2.4 Hajautettujen tietokantojen luotettavuus

Yksi hajautettujen tietokantojen potenttiaalisista eduista on parantunut luotettavuus ja saatavuus. Hajautettu järjestelmä voi kuitenkin kärsiä samantyyppisistä häiriöistä kuin keskitetty järjestelmä. Näiden lisäksi hajautetussa ympäristössä on varauduttava seuraaviin häiriöihin [ÖV99, Kappale 12]:

- yksittäisen pisteen häiriö,
- pisteiden välisen linkin häiriö,
- viestien menetys ja
- verkon osittuminen.

Vikasietoisen järjestelmän täytyy havaita kaikki mahdolliset häiriöt, konfiguroida järjestelmä uudelleen toiminnan jatkamiseksi ja elpyä. Järjestelmä pystyy yleensä havaitsemaan häiriön sattuneen, muttei pysty päättelemään sen tyyppiä. Esimerkiksi havaitaan, ettei piste s_1 pysty kommunikoimaan pisteen s_2 kanssa. Häiriö voi olla pisteessä s_2 tai pisteiden välisessä linkissä.

2.5 Käyttöjärjestelmätuki

Monet hajautetut tietokannanhallintajärjestelmät on toteutettu kaupallisen käyttöjärjestelmän päälle (esimerkiksi Oracle). Ongelmana on tehokkuus, sillä käyttöjärjestelmien tuki tietokantaoperaatioille ei ole riittävää hajautettuille tietokannanhallintajärjestelmille. Käyttöjärjestelmien tuki on suunnattu yleensä tavanomaisille sovelluksille.

Perusongelmana hajautettujen tietokannanhallintajärjestelmien kannalta on muistinhallinta, tietostojärjestelmä ja saantimenetelmät, vioista toipuminen ja prosessienhallinta. Hajautetussa ympäristössä lisäongelmia tuo tietoliikenneohjelmiston eri kerrokset, joita pitäisi pystyä erikseen hallitsemaan. Tavoitteena tämän alueen tutkimuksessa on löytää perustuki, joka soveltuisi paremmin tietokannanhallintajärjestelmille samalla kun tuetaan tavanomaisia sovelluksia [Gra79b].

2.6 Heterogeeniset tietokannat

Monitietokantajärjestelmä (multidatabase system, MDBS) on ohjelmistokerros, jonka välityksellä on mahdollista käsitellä erilaisissa olemassa olevissa tietokannoissa säilytettävää tietoa [ÖV99, Kappale 15]. Järjestelmä on yleensä heterogeeninen: paikalliset tietokannat käyttävät eri tietomalleja ja kyselykieliä, ja myös samanaikaisuudenhallintakäytännöissä ja elpymiskäytännöissä voi olla eroja.

Jotkin paikalliset tietokannat voivat olla relaatiotietokantoja, jotkin oliotietokantoja, jotkin verkkotietokantoja tai jopa hierarkisia tietokantoja. Joissakin paikallisissa tietokannoissa transaktioiden hallinta saattaa perustua lokiin ja kaksivaiheeseen lukitukseen, joissakin taas varjosivutukseen ja aikaleimoihin.

Monitietokannan tehtävänä on luoda käyttäjälle illuusio loogisesti yhdistetystä tietokannasta. Tietojen fyysiseen organisointiin tai säilytyspaikkaan järjestely ei aiheuta mitään muutosta. Kunkin paikallisen tietokannan autonomia säilyy ja tietokantojen ohjelmistoon ei aiheudu mitään muutoksia. Olemassa olevat sovellusohjelmat säilyvät toimivina.

Tietojen välitykseen erityyppisten tietokantojen välillä on perinteisesti käytetty *yhdyskäytäviä* (gateway). Monitietokantajärjestelmä on yhdyskäytävää yleisempi järjestelmä: siinä on täydellinen transaktioiden hallinta. Monitietokannan tietomalliksi soveltyy hyvin *oliorelaatio-naalinen* (object-relational) malli, jollaista käytetään mm. UniSQL- ja Illustra-järjestelmissä (nykyään myös Oracle 8 -järjestelmässä).

Miten globaalien transaktioiden ajoitusten sarjallistuvuus taataan monitietokannassa ? Tämä kysymys on relevantti, koska paikallinen sarjallistuvuus ei takaa globaalia sarjallistuvuutta. Eräs ratkaisu on kieltää päivitykset globaaleita transaktioilta. Toinen ratkaisu on vaatia transaktioiden sarjallistuvuusjärjestys samaksi kaikissa paikallisissa ajoituksissa. Mutta miten tämä taataan loukkaamatta paikallisten tietokantojen autonomiaa ?

3 Tietokanta-arkkitehtuuri

Tietokannanhallintajärjestelmä on monimutkainen monoliittinen sovellus. Järjestelmiin on tullut paljon uusia toimintoja ja osia. Tästä syystä tietokannanhallintajärjestelmien käytön opiskelu ja osaamisen ylläpito on muodostunut entistä vaikeammaksi [CW00]. Tässä kappaleessa esitetään muutamia näkökulmia, kuinka hyvin tietokannanhallintajärjestelmät oikeasti palvelevat käyttäjiä ja tutkijoita. Analyysi alkaa esittämällä muutamia kriisin merkkejä. Tämän jälkeen esitellään aivan uusi arkkitehtuurimalli, nk. RISC-tietokantamalli, joka on esitelty lähteessä [CW00].

3.1 Kriisin merkit

Havainto 1: Ominaisuuksien paljous ajaa ohi käsiteltävyyden [CW00]. Tietokannanhallintajärjestelmät tarjoavat entistä enemmän ominaisuuksia, jonka takia järjestelmistä muodostuu erittäin laajoja ja siksi monimutkaisia. Usein tarjotut ominaisuudet ovat enemmänkin markkinoinnin kannalta tarpeellisia. Uusi liitostapa tai aika-avaruudellinen hakemisto voi olla esimerkkejä uusista ominaisuuksista. Tuloksena tietokannanhallintajärjestelmän toiminnallisuus muodostuu ylikuormitetuksi johtaen järjestelmän asennuksen ja ylläpidon vaikeutumiseen. Usein käyttäjät käyttävät vain hyvin pientä osaa tarjotuista ominaisuuksista.

Havainto 2: SQL on piinallista. SQL on yhdiste kaikista mahdollisista ominaisuuksista, joista monien käyttö on vähintääkin epäsuotavaa. SQL on myös turhan monimutkainen tyypilliselle sovellusohjelmoijalle. SQL:n ydin eli valinta-projektio-yhdiste -kyselyt ja aggregaatiot ovat hyödyllisiä, mutta monista erikoisominaisuuksista on hyvin harvoin hyötyä [CW00].

Havainto 3: Tehokkuus on arvaamatonta. Kaupalliset tietokannanhallintajärjestelmät ovat kehittyneimpiä ohjelmistoja, joita on koskaan tietojenkäsittelyn historiassa rakennettu [CW00]. Tuloksena tietokannanhallintajärjestelmät ovat luonnostaa arvaamattomia täsmälliseltä toiminnaltaan erityisesti tehokkuuden osalta. Yksittäiset osat kuten kyselynoptimointi ovat jo ylittäneet kriittisen monimutkaisuusrajan.

Havainto 4: Tehokkuuden viritys on painajaista ja automaattinen viritys on tällä hetkellä toiveajattelua [CW00]. Laajat käyttöprofiilit muodostavat automaatiisen virityksen mahdottomaksi. Päinvastoin kaupalliset tietokannanhallintajärjestelmät tarjoavat suuren joukon viritystyökaluja, joiden vaikuksesta käyttäjällä ei pysty saamaan hyvää kuvaa ilman testausta.

Havainto 5: Emme ole yksin maailmankaikkeudessa [CW00]. Tietokannanhallintajärjestelmä ei ole sovellusohjelmistojen ytimessä vaan osana järjestelmää. Tietokannanhallintajärjestelmän onkin pystyttävä keskustelemaan sähköpostijärjestelmien, dokumenttihallintajärjestelmien ja monien muiden järjestelmien kanssa. Tästä syystä tietokannahallintajärjestelmien täytyy tarjota entistä enemmän uusia rajapintoja eri järjestelmien käyttöön.

3.2 RISC-tietokantamalli

RISC-tietokantamallin intuitiota kannattaa aloittaa esittelemällä esimerkkiä yhdestä mahdollisesta komponentista. Valitaan komponentiksi kyselynkäsittely [CW00]. Yksinkertaisin mahdollinen kyselykomponentti on yhden taulun valintaprosessori, jonka tehtävänä on tarjota yhteen tauluun kodistuvan kyselyn prosessoinnin ja yksinkertaiset päivitykset kyseiseen tauluun muodostettuun B+-hakemistoon. Tällaista voitaisiin käyttää useammassa sovellusohjelmassa jos tarjotaan vielä transaktiopalvelu. Aivan viime aikoihin asti SAP R/3 tietovarasto käytti tällaista menetelmää [CW00].

Toinen mahdollinen komponentti on valinta-projektio-yhdiste prosessori, jota voitaisiin hyödyntää esimerkiksi tietovarastoissa ja kaupallishallinnollisissa järjestelmissä. Tällainen RISC-komponentti voidaan toteuttaa yhden taulun valintaprosessorin päälle samaan tapaan kuin

RDS oli kerrostettu RSS:n päälle System-R järjestelmässä [CW00]. Kyseisen osuuden teoria ja tehokkuus on paremmin ymmärretty kuin täydellisen SQL-kyselynkäsittelijän.

Samaan tapaan tallenepalvelimet (storage managers) voitaisiin uudelleensuunnitella RISC-tyylisesti. Mahdollisia erillisiä komponentteja olisivat levyhallinta, välimuistinhallinta, lokinkäsittely, toipuminen ja samanaikaisuudenhallinta. Erilaiset tieto- ja sovellusluokat tarvitsisivat räätälöityjä ratkaisuja, jotka olisivat helpommin ylläpidettäviä ja viritettäviä kuin kaikkiin sovelluksiin yhteisesti soveltuvat menetelmät.

Tavallaan koko tietokannanhallintajärjestelmä uudelleensuunniteltaisiin komponenttipohjaiseksi arkkitehtuuriksi, jossa jokaisella komponentilla on tarkasti rajattu tehtävä. Tärkein tavoite tässä mallissa on mahdollisuus vaihtaa tai lisätä järjestelmään mikä tahansa komponentti tarpeiden mukaan.

4 Yhteenveto

Viime vuosikymmenen aikana on kehitty useita uusia järjestelmiä, jotka ovat luonnollisesti hajautettuja. Esimerkkejä tällaisista järjestelmistä löytyy niin WWW:n (eKauppa) kuin tietoliikenteestä, pörssikaupasta tai teollisuudesta. Siksi tarve myös kehittyneille hajautetuille tietokannoille on tullut entistä suuremmaksi. Kaupalliset tietokannanhallintajärjestelmät ovatkin kehittyneet samana aikana tukemaan entistä enemmän erilaisia sovellustarpeita. Monet kaupalliset tietokannanhallintajärjestelmät perustuvat edelleen asiakas-palvelin malliin, mutta ovat pitkälle kehittyneitä. Tämän hetken kaupalliset tietokannanhallintajärjestelmät tarjoavat paremmin tiedon tuntumattomuutta tarjoten mahdollisuuden esittää kyselyitä loogisiin relaatioihin ja suorittavat kyselyitä horisontaalisesti fragmentoituun tietoon, jotka ovat sijoitettu useisiin palvelimiin. Kaupalliset järjestelmät tukevat myös hajautettuja tapahtumia, joiden avulla voidaan toteuttaa entistä monimutkaisempia järjestelmiä sekä synkroniseen että asynkroniseen toimintaan.

Moniin tässä esityksessä esitettyihin hajautettujen tietokantojen ongelmiin on esitetty vain heuristisia ratkaisuja. Siksi hajautettujen tietokantojen alueella on useita tutkimuksellisestikin kiinnostavia haasteita varsinkin uusien sovellusalueiden kannalta. Tällaisia ovat esimerkiksi rinnakkaistietokannat ja liikkuvat tietokannat (mobile databases). Monet ongelmat ovat osoittautuneet NP-täydellisiksi, jolloin täysin optimaalinen ratkaisu on mahdollinen vain jos pystytään osoittamaan että P=NP. Koska tähän kysymykseen ei tiedetä positiivista tai negatiivista vastausta, jäävät ratkaisutkin heuristisiksi.

Viitteet

[CW00] S. Chaudhuri and G. Weikum. Rethinking database system architecture: Towards a self-tuning rics-style database system. In *Proceedings of the 26th International Conference on Very Large Databases*, Cairo, Egypt, 2000. Morgan Kaufmann Publishers.

[Gra79a] M. H. Graham. On the universal relation. Tech. report, University of Toronto, Toronto, Ont., Canada, 1979.

[Gra79b] J. N. Gray. Notes on data base operating systems. In R. Bayer, R. M. Graham, and G. Seegmuller, editors, *Operating Systems: An Advanced Course*, pages 393–481. Springer-Verlag, New York, 1979.

[Lai01] R. Laiho. Hajautetun järjestelmän tietoturva. Hajautetut sovellukset -seminaari, Helsingin yliopisto, tietojenkäsittelytieteen laitos, 2001.

[Lyn96] N. A. Lynch. *Distributed Algorithms*. Prentice-Hall, 1996.

[ÖV99] M. T. Özsu and P. Valduriez. *Principles of Distributed Database System*. Prentice Hall, second edition, 1999.

[YÖ79] C. T. Yu and M. Z. Özsoyoglu. An algorithm for tree-query membership of a distributed query. In *Proceedings of the IEEE COMPSAC*, pages 306–312. IEEE Computer Society, 1979.