Universitatea Babeş-Bolyai Cluj-Napoca
Facultatea de Matematică şi Informatică

RELOCAREA DINAMICĂ A FRAGMENTELOR ÎNTR-O BAZĂ DE DATE DISTRIBUITĂ

Teză de doctorat - rezumat

Doctorand:
Manuela (Horvat) Petrescu

Profesor îndrumător:
Prof. Univ. Dr. Leon Ţâmbulea

Cluj-Napoca
2010

Abstract

Atât în activitatea de cercetare, cât şi în cea de producţie, tehnologiile distribuite câstigă teren în fata celor centralizate. Dacă proiectarea bazelor de date centralizate se face urmând un set de reguli, lucrurile stau diferit în cazul bazelor de date distribuite. Proiectarea bazelor de date distribuite este un proces complex datorită faptului ca fragmentarea datelor şi procedurile de alocare ale acestora trebuie sa folosească funcţii de evaluare a costului şi de analizare a datelor. Aceste procese sunt în general prea complexe pentru a fi rezolvate manual, iar rezultatele găsite nu ar reflecta efortul depus în obţinerea lor. În această teză se propun algoritmi euristici pentru obţinerea unei distribuţii dinamice optime a fragmentelor într-o bază de date distribuită. Printre aceşti algoritmi se număra algoritmi de realocare, de rezolvare a interogărilor, de recuperare, dar şi de curăţare: pe un nod şi centralizat în toata baza de date.

Şabloanele de acces la date într-o bază de date distribuită se modifică în timp, şi ideal ar fi ca distribuţia datelor să se modifice în concordanţă cu aceste şabloane pentru a rămâne optimă. Reproiectarea unei baze de date este costisitoare şi este la fel de predictibilă ca şi modificările interogărilor utilizatorilor. Sub acest aspect, modelul propus este inovativ prin caracteristica sa dinamică: alocarea datelor se face în mod dinamic în funcţie de necesităţile utilizatorului. Datele statistice propuse în model ajută la determinarea şabloanelor de acces dar şi în luarea deciziei privind oportunitatea relocării unui fragment.

Sunt introduse şi definite noi metrici soft bazate pe statisticile sistemului pentru a eficientiza relocarea datelor (paragrafele- Relocarea în model – cazul A: un sistem echilibrat şi cazul B- un sistem neechilibrat).

Datorită algoritmilor de curăţare propuşi, spaţiul de stocare folosit este minimal, întrucât aceşti algoritmi şterg datele care nu sunt folosite sau sunt folosite rar. Algoritmul de curăţare este adaptat necesităţilor şi capacităţilor de stocare din fiecare nod – executându-se când spaţiul disponibil scade sub o anumită limită sau când sunt îndeplinite alte condiţii specifice nodului. Algoritmul de curăţare central execută şi operaţiuni de relocare a fragmentelor şi de schimbare a drepturilor acestora. Algoritmul de recuperare se ocupă atât de recuperarea nodurilor căzute cât şi de erori de partiţionare a reţelei.

Cuvinte cheie: dinamic, baze de date distribuite, optimal, model

Introducere

Motivare

În ciuda complexității şi a costului mai ridicat (comparativ cu bazele de date centralizate), bazele de date distribuite sunt soluția la multe dintre noile cerințe: acces rapid, volum mare de date. Marile companii au creat şi implementat baze de date distribuite: Oracle RAC, IBM DB2, SQL Server2008 Federated DB (FederatedDB). La o privire mai atentă se observă însă faptul că unele sunt doar parțial distribuite - mediul de stocare este comun, şi doar serverele sunt distribuite - e cazul Oracle RAC şi IBM DB2 PureScale. Aceste baze de date reprezintă o soluție bună în cazul căderii serverelor, dar ele însele nu oferă protecție împotriva erorilor apărute în mediul de stocare.

În această teză se propune o metodă pentru distribuirea dinamică a fragmentelor unei baze de date distribuite precum şi o metodă de alocare dinamică a drepturilor de acces la aceste fragmente în funcție de şabloanele de acces la date. În modelul propus, protecția datelor va fi asigurată de faptul că datele vor avea cel puțin două replici.

Noutatea acestui model este reprezentată de caracteristica sa dinamică: distribuția datelor se schimbă în funcție de şabloanele de acces oferind un grad înalt de disponibilitate cu un cel mai mic cost. Sub acest aspect, un întreg model este propus: metode de procesare a tranzacțiilor, metode de curățare şi relocare, metode de prevenire şi rezolvare a interblocării precum şi metode de recuperare. Costul, văzut din perspectiva costului de transmitere în rețea, a costului de relocare şi de execuție a fost o preocupare continuă, deoarece modelul se doreşte a fi nu numai inovativ ci şi eficient. Obiectivele avute în vedere în elaborarea modelului şi a algoritmilor pot fi prezentate pe scurt:

- Relocarea dinamică a datelor
- Disponibilitate şi exactitate
- Îmbunătățirea performanței
- Capacitate de stocare minimă
- Cost minim de comunicație
- Administrare dinamică a mediului de stocare

Caracteristicile modelului propus

Scalabilitate

Metoda tradițională de a mări capacitatea unui server era schimbarea acestuia cu un alt server cu capacitate mai mare (dar şi prețul creşte pe măsura capacității). Exista însă o altă soluție pentru aplicațiile care vor rula pe modelul propus: adăugarea unui alt server la cluster, aplicația beneficiind de el din momentul în care noua instanță/server este pornită.

Administrarea mediilor de stocare

Administratorii bazelor de date întâmpină provocări datorate creşterilor rapide în volum ale acestora, dar şi în găsirea unei ferestre de timp pentru efectuarea operațiilor de întreținere (schimbarea configurărilor). Metodele de recuperare prezentate în model vor oferi soluții la aceste provocări prin automatizarea întregului proces: tot ce va trebui să facă un administrator este să aloce dispozitivele de stocare la o instanță a bazei de date, şi modelul se va ocupă de restul procesului. În model se propune ca distribuția şi relocarea fişierelor bazei de date să fie

automatizată: datele se vor distribui automat către resursele de stocare pentru a optimiza performanța.

Administrarea proactivă a spațiului de stocare

Modelul propune o administrare proactivă a spațiului de stocare prin monitorizarea constantă a spațiului disponibil şi prin procese de curățare şi relocare a datelor. Dacă spațiul disponibil scade sub o anumită limită, se propune obținerea spațiului necesar prin luarea unor măsuri corective: rularea algoritmului de curățare şi ştergerea datelor accesate mai rar (Cu mențiunea că cel puțin două replici ale unei date trebuie să existe la un moment dat în baza de date şi că volumul de date şters se va specifica de administrator).

Distribuirea încărcării

Modelul propus va oferi facilități de distribuire a încărcării, prin re-trimiterea unei interogări către alt nod sau prin operații ca inserarea în masă a datelor. În re-trimiterea unei interogări, se propune alegerea nodului cu cel mai bun timp de răspuns – în acest mod se va ține cont de traficul din rețea, de încărcarea nodurilor dar şi de capacitățile de procesare ale acestora.

Asigurarea performanței

În fiecare nod din rețea vor fi stocate datele cele mai des accesate – deci pentru majoritatea interogărilor datele au un grad mare de disponibilitate şi încărcarea rețelei este minimă. Pentru a optimiza şi mai mult, vor trebui analizați algoritmii de optimizare specifici bazelor de date distribuite, şi modul în care aceştia vor interacționa şi vor fi implementați alături de restul algoritmilor în modelul propus.

Protecția datelor

Protecția datelor va fi asigurată în model prin replicare - se propune ca fiecare dată să aibă cel puțin două replici. Modelul se vrea a fi extrem de configurabil, permițând schimbarea numărului minim de replici. La căderea unui nod sistemul verifică şi măreşte numărul de replici ale unei date pentru a asigura existența numărului minim de replici.

Recuperare

Procedeele de recuperare pentru un nod oferă o recuperare rapidă şi completă, actualizând structura bazei de date, datele (valori şi drepturi), precum şi cataloagele bazei de date fără a necesita intervenția umană.

Politici de configurare

Modelul se vrea a fi extrem de configurabil şi este parametrizat pentru a îndeplini cât mai multe cerințe.

Concluzie

În această teză se propune un model inovativ, care poate fi şi rapid, configurabil, uşor de administrat şi eficient din punct de vedere al costurilor. În implementarea modelului se vor găsi şi problemele/caracteristicile menționate anterior.

Cuprins

1 Baze de date distribuite

O bază de date distribuită poate fi definită [Ta03, Öz99] ca şi o colecție de baze de date interconectate care asigură transparența utilizatorului; datele sunt stocate în fragmente. Fragmentarea reprezintă partiționarea unei relații globale R în fragmentele R_1, R_2, ... R_n, care conțin destulă informație pentru a reconstitui relația inițială R [Ta03, Da04].

Bazele de date distribuite oferă o serie de avantaje dintre care se menționează: acces şi procesare rapidă a datelor, independența arhitecturii, etc, dar şi o mulțime de dezavantaje: complexitatea administrării, lipsa standardelor, etc. [Öz99, Ro09, Ku09].

2 Prezentarea modelului

2.1 Introducere – considerente generale

Modelul propus în această teză se bazează pe următoarele idei şi considerente: o bază de date distribuită, o colecție de fragmente (notată cu C), fiecare fragment este replicat în două sau mai multe noduri, având drepturi diferite (de scriere sau de citire) în funcție de nod. Ideea care stă la baza modelului este: replicile unor fragmente îşi pot schimba drepturile, iar distribuția lor în nodurile rețelei se modifică dinamic în funcție de şabloanele de acces la date ale utilizatorilor [TaHo08]. Se notează cu:

- D - un fragment din colecția C
- $C(N) \subseteq C$ – submulțimea fragmentelor din colecția C care sunt stocate pe nodul N.

În model se consideră:

- Atât mulțimea $C(N)$ se modifică dinamic pentru orice nod N
- Fragmentul D are sau drept de scriere, sau drept de citire în nodul N; dar poate avea alt drept în alt nod.

Datorită costului de propagare şi de actualizare implicat de modificarea datelor replicate, administrarea fragmentelor cu drept de scriere este mai costisitoare decât a celor cu drept de citire, deci obiectivul este de a avea cât mai puține replici cu drept de scriere, dar suficiente pentru a asigura corectitudinea datelor în caz de evenimente neprevăzute (căderea unor noduri, erori ...). Altă problemă importantă: modelul nu propune replicarea tuturor datelor în toate nodurile – chiar dacă au doar drept de citire – deoarece ar fi necesare noduri cu capacități extrem de mari, ar scădea eficiența sistemului, s-ar pierde unele dintre avantajele bazei de date distribuite.

2.2 Descriere – imaginea modelului

Modelul poate fi descris ca o extensie a bazei de date care foloseşte şi îi îmbunătățeşte caracteristicile. Modelul propus implică schimbări atât în algoritmii centrali de administrare ai bazei de date distribuite, cât şi în procesele care se execută în fiecare nod, dar nu interferează cu

protocoalele folosite pentru efectuarea blocărilor sau a tranzacțiilor. Imaginea următoare prezintă vizualizarea modulelor:

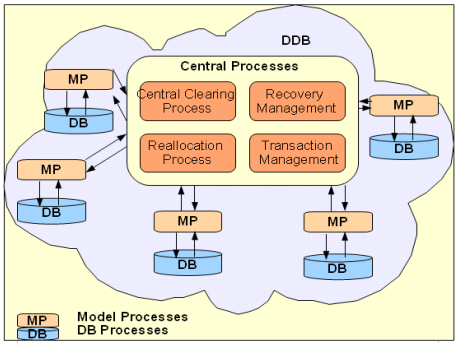

Fig.1 Reprezentarea bazei de date din model

În fiecare nod, MP (procese din model) sunt responsabile de rularea unor procese: de curățare, de recuperare, de actualizare, dar şi de relocare (în cadrul execuției algoritmului pentru rezolvarea interogărilor de actualizare). Exceptând procesul central de curățare şi relocare, nici un alt proces nu este legat de un nod, ca atare, baza de date va continua sa ruleze, chiar dacă a apărut o eroare pe unul dintre noduri. Acest fapt nu ridică o problemă deoarece execuția procesului central poate fi preluată de alt nod în cazul apariției unei erori.

2.3 Informațiile specifice modelului şi Catalogul bazei de date

Pentru a se obține un model optim se determină şabloanele de acces ale utilizatorilor, care date sunt mai des accesate, de pe ce noduri, etc. Pentru a obține aceste informații se folosesc parametri statistici care monitorizează în fiecare nod accesul şi datele accesate, urmând ca algoritmii de curățare şi relocare (de exemplu) să le folosească în relocarea datelor. Valorile acestor parametri sunt stocate în catalogul bazei de date [TaHo08, TaLHo08]:

Informațiile Statistice din model

R(N, D), W(N, D)

R(N, D), W(N, D) reprezintă numărul de cereri de citire/scriere primite de un nod *N*, pentru accesarea fragmentului *D* în citire/scriere (nu are nici o importanță dacă fragmentul *D* este stocat sau nu în nodul *N*).

W1(N, D)

W1(N, D) reprezintă numărul de cereri de scriere primite de nodul *N* pentru accesarea fragmentului *D* care nu este stocat în acest nod.

Parametrii specifici modelului

W_Max, W_Min

W_Max şi *W_Min* sunt parametri globali ai bazei de date şi reprezintă numărul maxim/minim de noduri în care un fragment poate fi salvat cu drept de scriere.

W(D)

W(D) este numărul de noduri unde un fragment D este salvat cu drept de scriere.

P(N)

P(N) este un parametru specific fiecărui nod şi reprezintă perioada maximă de timp care trebuie să treacă între două rulări consecutive ale algoritmului de curățare.

Functia cond(N, D, Pa)

Cond (N, D, Pa) este o funcție/condiție specifică fiecărui nod, folosită pentru a decide drepturile de acces *Pa(read, write)* ale unui fragment *D* stocat pe nodul *N*. Funcția se va determina în raport cu o mulțime de factori printre care: memoria disponibilă pe nod, încărcarea curentă, etc..

Localizarea Replicilor- *RL(N, D)*

Fiecare nod ar trebui să poată localiza replicile unei date. În acest scop, în fiecare nod, în catalogul bazei de date se va memora pentru fiecare fragment *D* localizarea nodurilor pe care sunt stocate celelalte replici ale fragmentului, precum şi drepturile acelor fragmente în nodurile respective (scriere/citire).

Timpul maxim de actualizare a unei replici de citire- MaxUT

MaxUT este un parametru global care asigură faptul că ultima versiune a anumitor date critice nu este mai veche de o perioadă specificată de timp. Se propune folosirea parametrului numai pentru datele critice deoarece utilizarea lui exagerată contribuie la mărirea traficului în rețea şi la scăderea eficienței.

Numărul *k* de fragmente şterse în procesul central de curățare

Acest parametru stabileşte o limită pentru numărul de fragmente care vor fi şterse în cadrul procesului central de curățare şi relocare. De exemplu, pentru un fragment replicat în 10 noduri, se vor şterse doar *k* replici- cele mai puțin accesate.

Numărul de fragmente *x* şterse în procesul de curățare de pe un nod

Parametrul *x* este generic pentru întreaga bază de date şi este folosit pentru a determina câte fragmente se şterg în procesul de curățare ale unui nod *N*. Condiția de ştergere ar trebui să țină cont de o limită *x* şi să şteargă fragmentele pentru care numărul de citiri/scriere este sub aceasta limită:

$$R(N,d_i) < x \text{ sau } W(N,d_i) < x,$$

Valori inițiale propuse

W_Min (numărul minim de drepturi de scriere) - *W_Min* poate avea o valoare inițială egală cu 1, dar nu este recomandabil din motive de siguranță; ar trebui să fie mai mare sau egal cu 2 pentru asigura datele în cazuri neprevăzute- căderea unui nod. Valoarea trebuie corelată însă şi cu caracteristicile bazei de date: în bazele de date cu puține noduri, două replici sunt suficiente, dar în cazul bazelor de date cu multe noduri sau cu incidență crescută de erori, s-ar putea ca doar două replici să nu fie de ajuns [Gh03,Va07].

W_Max (numărul maxim de drepturi de scriere) - Numărul maxim de drepturi de scriere trebuie corelat cu caracteristicile bazei de date, şi trebuie avut în vedere că numărul de replici de scriere este proporțional cu costul actualizărilor (cu cât există mai multe replici, cu atât mai mare este costul). Bazat pe recomandările [We02, GH03] se propune o valoare inițială 3.

P(N) (perioada de timp specifică nodului N) - Perioada de timp ar trebui specificată în funcție de şabloanele de acces la date, şi se determină în funcție de următorii factori: aria geografică unde sunt localizate nodurile, încărcarea de pe fiecare nod, timpul de răspuns al fiecărui nod.

Parametrul k - Parametrul *k* specifică numărul de replici care se vor şterge în cadrul procesului global de curățare. De exemplu, o valoare de 20-25% ar oferi o rată bună de curățare, căci după 2-3 execuții ale algoritmului, numărul replicilor nefolosite acestora scade aproape la jumătate din numărul inițial.

2.4 Comportamentul modelului

Scopul fiecărei baze de date este rezolvarea interogărilor: de citire şi de scriere. Pentru a profita de caracteristicile interogărilor de citire, se folosesc algoritmi diferiți pentru rezolvarea acestora.

Într-o bază de date centralizată, rezolvarea unei cereri *q* se face găsind setul de fragmente de date necesar, acesta se prelucrează, iar rezultatul este trimis utilizatorului. Într-o bază de date distribuită, fragmentele se găsesc pe noduri diferite, deci în rezolvarea interogării se va ține cont şi de localizarea fragmentelor. În modelul propus, replicile de citire sunt actualizate asincron, o soluție cu multe beneficii dar şi cu inconveniențe [Yo08, Ed97, CaDo09, St09].

Actualizarea asincronă a datelor de citire ridică probleme legate de corectitudinea datelor: toate datele eligibile din baza de date trebuie luate în considerare (mai ales în cazurile adăugare/ştergere)? O soluție ar fi existența a două tabele master (două din rațiuni de siguranță), dar ele ar limita performanța, plus că ar contribui la încărcarea rețelei cu mesaje. O soluție mai bună [Pu91] ar fi trimiterea de mesaje „dirty": în caz de în adăugare/ştergere se va marca întregul tabel, altfel, se vor marca doar fragmentele afectate de actualizări. Marcajul de „dirty" este şters în momentul în care se primesc (după finalizarea tranzacției de scriere) datele actualizate, sau când se face revenirea la starea precedentă începerii tranzacției. În cazul special – tabele cu informații critice – se poate folosi şi parametrul *MaxUT* care garantează că valoarea replicilor nu este mai veche de o anumită perioadă de timp (specificată de valoarea parametrului).

În funcție de localizarea fragmentelor necesare rezolvării unei interogări de citire *q* primită de nodul *N* se întâlnesc două situații [TaHo08]:

Cazul 1. Toate fragmentele $\{d_i \mid d_i \rightarrow q,\ i=1..\ k\}$, sunt salvate pe nodul *N*.

Cazul 2. Unul sau mai multe fragmente nu sunt salvate pe nodul *N*.

Analog cu cererile de citire, în cazul rezolvării unei interogări de actualizare, în funcție de localizarea fragmentelor se disting două cazuri:

Cazul 1. (cu incidența mai mică): Toate fragmentele $\{d_i \mid d_i \rightarrow q,\ i=1..\ k\}$, sunt salvate pe nodul *N* şi au drepturi de scriere

Cazul 2. Unul sau mai multe fragmente nu sunt salvate pe nodul *N,* sau nu au dreptul de scriere necesar.

- Fragmentele cu drept de citire stocate în nodul *N* îşi pot modifica dreptul: dacă anumite condiții sunt îndeplinite, pot primi drept de scriere.
- Cererea poate fi retrimisă altui nod (*N1*) în care fragmentele au drept de scriere.

- Fragmentele care nu pot fi actualizate în nodul *N* sau *N1* – vor fi actualizate în cadrul tranzacției pe alt nod care stochează copia primară a acelor fragmente.

2.5 Relocarea fragmentelor

Relocarea în model – Cazul A – un sistem echilibrat

Relocarea unui fragment sau transferul dreptului de scriere între replicile unui fragment între două noduri va ține cont de cost: diferența dintre costul de rezolvare a interogărilor pe cele doua noduri şi costul de relocare trebuie să fie pozitivă [Ho08]. Se poate vorbi de transferul dreptului de scriere în cazul în care o replică are drept de citire şi una de scriere.

În prima versiune, se consideră un sistem relativ echilibrat în care toate nodurile necesită acelaşi timp pentru scrierea/citirea aceluiaşi set de date. Se presupune de asemenea că latența între două noduri este constantă şi egală cu Δ. Alte notații necesare:

- *Tu* – timpul necesar actualizării unui fragment *d* pe nodul *N*
- *n* – numărul total de noduri din baza de date

Relocarea unui fragment din nodul *N* în nodul *N1* este necesară când:

- Numărul de cereri de actualizare primite de nodul *N1* este mai mare decât numărul de cereri primite direct de nodul *N* (adică $W(N1, d) > W(N, d)$) şi
- Costul de transfer este mai mic decât diferența dintre costul interogărilor primite în nodul *N1* şi retrimise nodului *N* şi costul cererilor care vor fi primite de nodul *N* şi trimise nodului *N1*.

Aceste condiții pot fi scrise matematic, obținându-se următoarele rezultate (timpul de actualizare pe un nod al unui fragment este extrem de mic, astfel se va considera că $Tu \rightarrow 0$):

Costul total (Tc) pentru relocare unui fragment este:

$$Tc = 2 * \Delta * n + 2 * \Delta * W(d) - 4 * \Delta$$

Diferența de cost (Dc):

$$Dc = 2 * \Delta * (W(N1, d) - W(N,d))$$

A doua condiție pentru ca relocarea să fie optimă se poate scrie:

$$Tc < Dc <=>$$
$$W(N1,d) > W(N,d) + n + W(d) - 2$$

În concluzie, transferul replicii/ al dreptului de scriere este recomandat şi este optim în cazul în care diferența dintre numărul de cereri de actualizare dintre nodurile *N* şi *N1* este mai mare decât numărul total de noduri plus numărul de replici de scriere minus 2 [Ho08].

Relocarea în model – Case B – un sistem neechilibrat

În a doua abordare, se presupune ca sistemul nu este echilibrat: nodurile execută aceeaşi operaţie pe acelaşi set de date în timpi diferiţi, latenţa între două noduri depinde de reţea, iar încărcarea pe un nod nu este constantă. Pe moment se va folosi o matrice a costului de transfer şi de latenţă, notată cu *C:*

- $C = \{ c_{P,Q} \mid P,Q$ *noduri în baza de date*$\}$, unde $c_{P,Q}$ *reprezintă* costul de transfer şi latenţa între nodurile *P* şi *Q* din baza de date.

Observaţie:

$$C_{P,Q} = c_{Q,P} \; ; \; P,Q \text{ sunt noduri în baza de date}$$

Costul notificării replicilor cu drept de scriere şi costul obţinerii unui răspuns este:

WriteReplicaCost:

$RW(D) = \{Q$; *Q conţine o replică de scriere a fragmentului d)*

$Wrc = 2 \; (\sum_{Q \in RW(D)} c_{N,Q}$; *Q conţine o replică de scriere a fragmentului d*)

Costul notificării replicilor cu drept de citire şi costul actualizării lor este:

ReadReplicaCost:

$RR(D) = \{P$; *P conţine o replică de citire a fragmentului d)*

$Rrc = (\sum_{P \in RR(D)}^{P} c_{N,P}$; *P conţine o replică de citire a fragmentului d*)

Folosind condiţiile de relocare optimă enunţate în cazul A, se obţine următorul rezultat matematic (*P/ Q* noduri care conţin o replică citire/scriere a fragmentului *d*):

$$2 \left(\sum_{Q \in RW(D)} c_{N,Q} + \sum_{Q \in RW(D)} c_{N1,Q}\right) + \sum_{P \in RR(D)} c_{N,P} + \sum_{P \in RR(D)} c_{N1,P} <$$

$$2 * \left(\sum^{Q} c_{N1,Q} + \sum^{Q} c_{N,Q}\right) * (W(N1,d) - W(N,d)) ;$$

3 Algoritmii din model

3.1 Algoritmi de rezolvare a interogărilor

Algoritmii de rezolvare a interogărilor rezolvă solicitările în funcţie de o mulţime de parametri: tipul interogării (citire sau scriere), locaţiile şi drepturile fragmentelor necesare soluţionării acesteia, etc.

În funcţie de locaţia fragmentelor şi a drepturilor lor, algoritmii se împart în doar cazuri:

- Toate fragmentele necesare sunt stocare în nodul în care s-a primit cererea şi au drepturile necesare (de ex. pentru o cerere de scriere există dreptul de scriere)
- Unul sau mai multe fragmente sunt stocate în alte noduri sau nu au drepturile necesare.

Toate fragmentele necesare sunt stocate în acelaşi nod

Primul şi cel mai simplu caz este cel în care mulțimea de fragmente necesară rezolvării interogării *q* se găseşte în acelaşi nod, şi au drepturile necesare, iar parametrul *MaxUT* nu a fost inițializat. Rezolvarea unei cereri *q* primită în nodul *N* necesită accesarea unei mulțimi de fragmente:

$$r(q) = \{d_i \ , \ i \in I_q\} \subset C(N)$$

În acest caz algoritmul poate fi formalizat astfel:

```
If r(q)= {di | i∈ Iq} ⊂ C(N)
          and r(q) has the permission required by q Then

    If q is read request Then
          //check for dirty status
          Boolean dataIsDirty = true;
         While (dataIsDirty) Do
                dataIsDirty = false;
               For each di; di∈ N, i=1.. k, Do
                         dataIsDirty = dataIsDirty or (di is dirty)
                End For
          End while
    Else
          For each di; i∈ Iq do
                For each Nk; Nk ∈ RL(N,di) and
                            (di has read permission în Nk ) Do
                      send "dirty" message
                End for
          End for
    End If

    Execute q;

    //update system parameters, send last values for write requests
    If q is read request Then
          For each di; i∈ Iq do
                R(N, di) = R(N,di) + 1;
          End for
    Else
          For each di; i∈ Iq do
                W(N, di) = W(N, di) + 1; // q is write request
                //send last value to read replicas
                For each Nk; Nk ∈ RL(N,di) and
                            (di has read permission în Nk ) Do
                      send di
                End for
          End for
    End If
End If
```

Fragmente nu au drepturile necesare sau sunt salvate în locații diferite

Dacă în mulțimea fragmentelor necesare rezolvării interogării *q*, cel puțin un fragment nu are permisiunea necesară sau nu se găsește în nodul care trebuie să rezolve cererea atunci algoritmul devine ceva mai complex.

Dacă există cel puțin un fragment d_j; $0<j<=i$ și mulțimea $r(q)=\{d_i \mid i \in I_q\}$; *not* $d_j \in C(N)$; sau există cel puțin un fragment care nu are permisiunea necesară rezolvării interogării, se va folosi următorul algoritm:

```
//Read Requests:
If q is read request Then

    //transfer data fragments to the node N
    For each dj; 0 < j <= i, r(q) = { di | i∈ Iq }; dj⊄C(N); do
          Select N1 using RL(dj);
          Transfer dj to N

          If cond(N, dj, read) Then      //there is empty storage, ...
                store dj on N
                R(N, dj) =  R(N, dj) + 1;
                update RL(dj);
          End If
    End For

    If (MaxUT isValid) Then
          For each di; i∈ Iq} do
                If (currentTime – lastUpdateTime(di) > MaxUT ) then
                      Update di;
                      lastUpdateTime(di) = currentTime;
                End If
          End for
    End If

    //check for dirty status – data being updated
    Boolean dataIsDirty = true;
    While (dataIsDirty) Do
       dataIsDirty = false;
       For each di; di ∈ C(N), i=1.. k, Do
          dataIsDirty = dataIsDirty or (di is dirty)
       End For
    End while

    Execute q;
End If


//Write Requests
If q is write request Then
    For each dj; 0 < j <= i, r(q) = { di | i∈ Iq}; dj⊄C(N);
          // dj should have write permission în the node N
                                                                  1
          Select N1 using RL(dj);

          // the write request number on N is greater than on N1
          If  W(N, dj) >  W(N1, dj) Then
```

```
                  If W(dj) < W_Max and cond(N, dj, write)Then
                        //increase the number of write permission replica
                        store  dj on N;
                        W(N, dj) =  W(N, dj) +1;
                        update RL(dj);

                  Else If (reallocationIsOptimal
                              and cond(N, dj, write)) Then
                        //move dj with write permission from N1 to N
                        store  dj on N;
                        remove  dj on N1
                        W(N, dj) =  W(N, dj) +1;
                        update RL(dj);
                  End If
            End If
      End For

      For each di; i∈Iq do
            For each Nk; Nk ∈ RL(N,di)and(di has read permission în Nk} Do
                  send "dirty" message
            End for
      End for

      Execute q;  // using Regular algorithm for updating în DDB

      For each di; i∈Iq do
            // q is write request; number of write operations is updated
            W(N,di) = W(N, di) + 1;
            //send last value to read replicas
            For each Nk;Nk ∈ RL(N,di)and(di has read permission în Nk) Do
                  send di
            End for
      End for
End If
```

Observații:

- În selecția nodului *N1,* se va lua în considerare cel mai bun timp de răspuns: astfel se va ține cont de performanța nodului *N1*, de încărcarea sa, dar şi de performanța şi încărcarea rețelei.
- Mesajele „dirty" pot să implice un fragment dintr-un tabel sau întregul tabel (cereri de adăugare, ştergere)
- Replicile cu drept de scriere sunt tot timpul actualizate, iar verificarea parametrului *MaxUT* se face doar în cererile de citire.
- În luarea deciziei de a efectua o realocare se pot folosi rezultatele matematice de mai sus.

Optimizări: găsirea nodului optim pentru retrimiterea interogării

În algoritmii detaliați în paragrafele anterioare, cererile de citire sunt rezolvate în nodul *N* în care au fost primite. Acest fapt implică transferul prin rețea a fragmentelor necesare rezolvării cererilor, fragmente care nu au fost salvate în nodul *N.* Pe de altă parte, păstrarea tuturor datelor în toate nodurile bazei de date necesită existența unor medii de stocare extrem de mari, ineficiente din punct de vedere al costului. Soluția ar fi găsirea un nod mai eficient din punct de vedere al costului căruia să i se trimită interogarea; în acest scop se va folosi algoritmul descris în secțiunea „*Redistribuirea fragmentelor în model"*.

Optimizări: Folosirea statisticilor de citire

O soluție de optimizare se bazează pe analizarea interogărilor de citire şi pe determinarea şabloanelor repetitive. În funcție de şabloane se proiectează fragmentele. De exemplu, dacă majoritatea interogărilor la tabelul *student* se referă la acele înregistrări în care valoare câmpului *avgMark* este mai mică decât *6*,

```
select * from student where avgMark < 6
```

atunci aceste fragmente ar trebui stocate în nod, iar fragmentarea datelor în cazul tabelului *student* va lua în considerare valoarea din câmpul *avgMark*.

Optimizări: Folosirea algoritmului Round Robin pentru inserarea în masă

În cadrul procesului de optimizare a interogărilor într-o bază de date (centralizată sau nu), fiecare operație de I/O este importantă deoarece ține de caracteristici fizice şi nu poate fi îmbunătățită decât prin modificări ale componentei hard. În cazul actualizărilor unei date, numărul de operații de I/O este în general fix, excepție fiind cazul adăugărilor: adăugarea în masă este posibilă şi utilizarea ei reduce timpul necesar scrierii în baza de date.

De asemenea, în cazul tranzacțiilor de scriere mai poate fi aplicat şi alt algoritm în alegerea nodului care să facă inserarea: Round Robin [Fl03, VM07] – este uşor de implementat şi ar asigura o încărcare distribuită.

3.2 Algoritmul de curățare şi relocare

Algoritmul central de curățare verifică valorile parametrilor statistici pentru fiecare fragment, şi în funcție de ei ia decizia de schimbare a drepturilor, de ştergere sau de relocare.

Algoritmul de curățare pe fiecare nod are atribuții mai puține: el decide doar ştergerea unor date în funcție de numărul de citiri. Algoritmul de curățare este inițiat în momentul în care au loc o serie de evenimente în nodul *N*, precum expirarea perioadei de timp *P(N)*, scăderea spațiului disponibil sub o anumită limită, sau lansarea în execuție de către DBA a procesului. Algoritmul de curățare al unui nod poate fi executat şi ca parte a unui proces centralizat în care inițial sunt curățate nodurile şi apoi este rulat procesul central de curățare şi relocare. Trebuie făcută distincția între cei doi algoritmi de curățare: algoritmul de curățare într-un nod nu efectuează nici o relocare şi nici nu schimbă drepturile, are rolul de a obține spațiu şi de a şterge datele nefolosite ajutând la creşterea performanței algoritmului de curățare central care va avea mai puține fragmente de verificat.

Procesul central de curățare şi relocare este format din trei paşi mari:

- Şterge datele nefolosite de pe fiecare nod *N* din baza de date folosind algoritmul local de ştergere dintr-un nod
- Calculează datele statistice pentru relocare
- Execută relocarea dacă decide că relocarea este optimă

Algoritmul de curățare local verifică parametrii statistici şi ia decizia de ştergere care poate fi -

- riguroasă – şterge datele care nu au fost folosite deloc (în funcție de algoritmul folosit de DBMS în tranzacțiile de actualizare, condiția poate să nu se îndeplinească niciodată)

$R(N,d_i) = 1$ sau $W(N, d_i) = 1$

- permisă – şterge datele dacă numărul cererilor de acces în citire/scriere este sub o anumită limită *x* specificate de DBA:

$R(N, d_i) < x$ sau $W(N, d_i) < x$; x este un număr

Decizia de a şterge un fragment *D* care are drept de scriere trebuie să ţină cont atât de numărul total de drepturi de scriere ale acelui fragment cât şi de numărul minim de drepturi de scriere care trebuie să existe în baza de date.

Algoritmul de curăţare într-un nod

Decizia de a şterge un fragment dintr-un nod al bazei de date ar trebui să se ţină cont de parametrul *x* – (poate fi setat de DBA la 1 dacă se doreşte ştergere strictă). Valoarea lui trebuie aleasă cu grijă întrucât alegerea unei valori prea mici implică faptul că nu se vor şterge destule fragmente, iar o valoare prea mare va avea drept consecinţă ştergerea unei părţi mari (dacă nu a majorităţii) datelor din nod.

După ce un fragment a fost şters dintr-un nod, trebuie trimis un mesaj către restul nodurilor pentru a-şi actualiza catalogul bazei de date. Mesajul pentru o data d_i ar trebui trimis numai acelor noduri care se regăsesc în $RL(d_i,N)$: N_k ; $N_k \in RL(d_i,N)$, deoarece restul nodurilor (care nu se regăsesc în $RL(d_i,N)$) nu au o replică a acelui fragment şi nu sunt interesate de acesta.

Algoritmul de curăţare într-un nod *N* se poate formaliza astfel:

```
For each di; i∈ I, di ∈ C(N) Do

      // remove unread data
      If R(N, di) <= x and di does not have write permission Then
            remove di;
            K = { Nk | Nk ∈ RL(di,N)  }
            For each Nk ∈ K
                  update RL(di, Nk)
            End for;
      End If

      // remove unwritten data
      If W(N, di) <=x and W(di) > W_Min Then
            remove di;
            K = { Nk | Nk∈ RL(di,N)  }
            For each Nk ∈ K
                  update RL(Nk, di)
            End for;
      End If
End For
```

Algoritmul central de curăţare şi relocare

Algoritmul de relocare ar trebui să parcurgă fiecare nod şi să verifice pentru fiecare fragment dacă relocarea este oportună. Datorită volumului mare de muncă, a costului deloc neglijabil, se impun unele optimizări care au un impact major: fiecărui fragment i se ataşează o

marcă de timp; astfel, un fragment şi replicile sale vor fi verificate o singură dată în cadrul procesului de relocare şi de curățare globală [Ho09].

```
mark = getMark( currentTime);
For each Nj; j =1,n;

      For each di; i∈ I, di∈ C(N) Do

            //optimize parsing and continue if the data fragment was checked
            If (di has mark; i∈ I) then
                  Continue;
            End If

            //compute statistical data for reallocation -number of reads
            //and writes for a data fragment în the whole db
            compute { Nj, R(di), W(di)| i∈ I ..j=1,n}

            //number of nodes having data di with read permission
            noNodes = |{Nj | j=1,n}|

            //remove read fragments with small number of read requests
            //k' is the number of nodes where the deleting will occur
            k' = k;
            If k not percent Then
                  k' = min(k, noNodes);
            Else
                  k' = [k * noNodes ]  // full part
            End if
            Remove first k' fragments with the smallest Rk

            If (di has write permission în Ni) Then
                  //remove with small number of write requests
                  If (W(di) = W_Max and
                      W(Ni, di) < avg{ W(Nj, di)|
                              di has write permission în Nj}) Then
                        remove di from Ni
                  End If
            End If

            //set write permission for large number of write requests
            If (W(di) < W_Max and cond(N, di, write) and
                W(Ni, di)> avg { W(Nj, di)|
                        di has write permission în Nj })  Then
                        add write permission for di în Ni
            End If

            //reallocate if necessary
            If W(di) = W_Max and cond(N, di, write) and
               W(Ni,di)> avg{W(Nj,di)|di has write permission în Nj} Then

                  //check if reallocation between node N and node Nk
                  //with the smaller number of write request is optimal

                  minWriteReq= min {W(Nj, di)| di has write perm. în Nj};

                  Nk = Nj; W(Nj, di)= minWriteReq,

                  If (reallocationIsOptimal between Nk, Ni) Then
```

```
                        reallocate d_i from N_k
                  End If
            End If

            //add mark to data fragment and its replicas
            For each N_i; N_i ∈ RL(N, d_i)
                  Add mark to d_i from N_i
                  R (N_i, d_i) = 1
                  W (N_i, d_i) = 1
            End for

            Add mark to d_i from N
            R (N, d_i) = 1
            W (N, d_i) = 1
      End for
End For
```

Explicații şi alte comentarii:

- *first d_k fragments*: *k* este un parametru in model, poate fi configurat ca număr sau ca procent
- de ce se foloseste *k'? k'* este un parametru folosit pentru a stabili o valoare fixă a lui *k* dacă *k* a fost configurat ca procent.
- se pot folosi rezultatele matematice obținute anterior pentru a determina dacă relocarea unui fragment este sau nu optimă.
- Marca de timp trebuie să fie corelată cu algoritmul central, căci compararea timpului de pe două noduri diferite nu este relevantă.

Optimizări în algoritmul de relocare

Algoritmul central de curățare şi relocare reprezintă o parte importantă a modelului propus şi este important să fie optim şi să genereze cât mai puțin trafic. Ca atare, în procesul de execuție, un fragment şi replicile acestuia vor trebui analizate o singura dată. Dacă procesului de relocare s-ar executa în cadrul procesului de curățare local de pe fiecare nod s-ar ridica probleme mari legate de consumul de resurse precum şi de sincronizarea replicilor unei date utilizate de două procese de relocare inițiate în noduri diferite.

Optimizările incluse în algoritm sunt:

- Abordarea centralizată,
- Folosirea unei mărci de timp pentru marcarea replicilor verificate, astfel ele fac obiectul unei singure verificări,
- Folosirea relației matematice obținută anterior în procesul de decizie a primalității realocării.

Dacă se marchează fragmentul de date d_{qp} localizat în nodul N_q, înseamnă că s-a verificat (atât fragmentul d_{qp} cât şi replicile sale în cadrul execuției algoritmului de relocare). Ca atare, restul replicilor nu vor fi verificate la rularea algoritmului pe nodurile unde sunt stocate, costul verificării lor în aceste noduri fiind aici egal cu zero. Astfel se reduce substanțial costul execuției algoritmului de curățare.

```
If cost(d_kp)<>0, k =1,n, n-număr de noduri
        → cost (d_fp ) = 0 , f<>k →
```

$$C = cost(d_{11}) + \ldots + cost(d_{1p}) + cost(d_{21}) + \ldots + cost(d_{2p}) + \ldots + cost(d_{q1}) + cost(d_{qp}) + \ldots + cost(d_{n1}) + cost(d_{np}) <=>$$

$$C = cost(d_{11}) + \ldots cost(d_{1p}) + 0 + \ldots 0 + \ldots 0 + 0 + \ldots 0 + 0 <=>$$

$$C = |p| * c$$

Conform rezultatelor, costul total este influențat de numărul de fragmente şi de costul *c* necesar relocării unui fragment. Deşi numărul total de noduri nu apare în formula finală, el este inclus în costul *c* datorat optimizării unui fragment: un număr mai mare de noduri duce automat la un cost mai mare de actualizare a unui fragment de date.

Constrângeri în algoritmul de relocare

Unele constrângeri din algoritmul de curățare şi relocare sunt datorate constrângerilor sistemului, altele sunt necesare pentru a asigura funcționarea la capacitate maximă cu cel mai mic cost:

- Un proces nou (de curățare şi relocare) nu poate fi început până când nu este finalizat procesul curent- cu alte cuvinte nu pot fi rulate în paralel două procese centrale de curățare.
- Procesul de curățare de pe un nod trebuie finalizat înainte de începerea procesului central de curățare
- Este recomandabil să fie cât mai multe noduri care efectuează procesul de curățare local înaintea celui global.
- Marca ataşată fiecărui fragment trebuie să fie unică în raport cu procesul central de curățare – şi trebuie să depindă de un parametru unic: timpul de începere a procesului central de curățare.

Procesul de relocare este un proces complex datorită faptului că implică toate nodurile din rețea: li se cer informații legate de numărul de citiri şi scrieri, iar aceste mesaje generează trafic. Reducerea volumului de date prin ştergerea datelor fără o analiza atentă (se poate compara gradul de utilizare) de pe noduri ar fi eronată, deoarece exista o probabilitate mare să se şteargă date des folosite şi care ar trebui păstrate în acele noduri.

Relația dintre algoritmul de curățare şi interogări

Procesele de rezolvare a interogărilor se pot executa în paralel cu procesul central de curățare, dar timpul de răspuns ar putea fi influențat datorită faptului că algoritmul de curățare este un proces care implică toate nodurile din baza de date şi care consumă resurse. Chiar dacă optimizările făcute ajută la scăderea traficului şi a costului total, este recomandabil ca rularea algoritmului de curățare să se facă în momentele în care baza de date nu este solicitată la maxim. Aceste ipoteze logice vor trebui demonstrate în practică după ce modelul va fi implementat.

4 Administrarea tranzacțiilor

Modelul propus este un model orientat pe tranzacții, care sunt folosite pentru a asigura integritatea datelor [Gr93, Pi00, Co02]. O tranzacție este o serie de una sau mai multe instrucțiuni

SQL, tratate ca şi o unitate, definite cu scopul de a îndeplini anumite însărcinări [Gr93]. O tranzacţie distribuită este o tranzacţie care include una sau mai multe instrucţiuni ce vizează date din două sau mai multe noduri distincte ale unei baze de date distribuite. În cazul în are operaţiile se execută în moduri multiple este greu de specificat şi de impus argumentul serializabilităţii. Complicaţiile rezultă din faptul că aceleaşi mulţime de tranzacţii poate avea o ordine diferită la noduri diferite [Oz08].

O operaţie de citire nu implică schimbări, pe când o operaţie de scriere trebuie propagată la toate replicile în toate nodurile;serializabilitatea globală poate fi obţinută prin algoritmi cu blocare centralizată, cu blocare a copiei primare sau cu blocare distribuită. Un efect secundar al utilizării acestor algoritmi îl constituie apariţia inter-blocărilor. Cu toate astea, relativa lor simplitate, precum şi performanţa bună îi fac să fie mai des folosiţi decât alternativele: algoritmi bazaţi de mărci de timp sau pe protocoale optimistice. Algoritmii centralizaţi cu blocare pot fi extinşi cu uşurinţă şi fără schimbări majore pentru a fi aplicaţi în baze de date distribuite [Ha90, Ba91].

4.1 Blocarea în model

Actualizarea structurii bazei de date

Modelul nu oferă facilităţi speciale care să permită actualizarea bazei de date în timp ce aceasta lucrează şi răspunde interogărilor. Ca atare, pentru a executa o tranzacţie care modifică structura bazei de date, se restricţionează accesul utilizatorilor până când se finalizează tranzacţia. Dacă un nod este deconectat de la reţea şi nu poate fi actualizat, două opţiuni sunt valide:

- Nu se execută tranzacţia până când toate nodurile sunt disponibile
- Se execută tranzacţia şi apoi se foloseşte pentru actualizare algoritmul UpdateStructureAlgorithm prezentat în cele ce urmează.

Prima soluţie este uşor de implementat dar este greu de utilizat deoarece restaurarea unei conexiuni poate dura (de exemplu, în cazul apariţiei unor erori care necesită înlocuirea componentei hard, timpul de livrare a acesteia poate sa fie de ordinul orelor, sau chiar a zilelor); între timp modelul nu poate fi actualizat deşi restul bazei de date funcţionează la parametri normali. Această situaţie poate fi evitată dacă se optează pentru a doua variantă- folosirea algoritmului de recuperare pentru actualizarea structurii bazei de date a unui nod deconectat.

În următorul algoritm se presupune că s-a determinat un nod *N1* care va fi folosit pentru a actualiza structura bazei de date de pe nodul *N* (tabele, proceduri stocate, triggere, etc).

```
For each table T from N Do
   Bring T from N1
End for
For each stored procedure St, view V Do
   Bring St,V from N1
End for
For each trigger t, cursor c from N Do
   Bring t, c from N1
End for
```

Algoritmul poate optimizat: se vor actualiza doar obiectele modificate, iar verificarea se face folosind o suma de control pe acelaşi set de date. De asemenea, în cazul existenţei view-urilor,

este mai rentabilă refacerea acestora prin execuția instrucțiunilor SQL decât transferul părților/datelor modificate. În algoritmul următor nu se exemplifică decât optimizarea pentru view-uri şi pentru tabele (procedurile stocate, trigger-ele sau cursoarele se comportă la fel):

```
For each table T from N Do
   If (checksum (T,N) <> checksum(T,N1)) Then
         Bring T from N1
   End if
End for
For each view V Do
   //Qv-query corresponding to the view V
   If (checksum (Qv,N) <> checksum(Qv,N1)) Then
         Bring Qv from N1
         Run Qv
   End if
End for
```

Algoritmul poate fi mai departe optimizat: dacă un tabel a fost modificat (diferă suma de control), tabelul se poate diviza în părți mai mici (pagini) care să fie la rândul lor verificate – şi astfel să se actualizeze doar datele modificate. Astfel se evită traficul în rețea generat de transferul întregului tabel.

4.2 Tranzacții de citire în model

O tranzacție de citire este o tranzacție care accesează valoarea unei date [GM82]. Se pot folosi algoritmi generali de procesare a tranzacțiilor, dar în general este mai eficientă folosirea unor algoritmi speciali care să țină cont de faptul că aceste tranzacții nu modifică datele [Sa93]. Ca urmare, protocoalele pentru rezolvarea interogărilor de citire sunt mai simple şi mai puțin costisitoare, iar tranzacțiile de citire nu pot genera inter- blocări. Pentru tranzacțiile de citire se propune în model folosirea blocării de tip partajat.

În termeni de performanță, modificările algoritmilor generali propuse în model ar trebui sa aibă un impact minor deoarece implică în general doar calcule suplimentare. Aceste calcule sunt executate în serverul care a primit interogarea. Algoritmii propuşi nu modifică protocoale legate de transmiterea în rețea, procesul de obținere al blocărilor sau agregarea rezultatelor unei interogări.

Problema care apare în tranzacțiile de citire este următoarea: cum se determină toate datele necesare în rezolvarea unei interogări când nu toate nodurile din baza de date au fost interogate. Interogarea tuturor nodurilor ar avea un impact major asupra costurilor şi performanței; iar şansele apariției unor noduri cu probleme în a răspunde se majorează simțitor [Al05, Al03] – fiecare nod ar trebui să răspundă fiecărei tranzacții din model. De aceea, nodul care a primit interogarea trebuie să ştie care sunt nodurile care conțin informație relevantă (se foloseşte Replica Location). De asemenea, se propune ca în momentul începerii unei tranzacții de actualizare a unor date, trimiterea unor mesaje de atenționare „dirty” către nodurile care conțin replici de citire ale datelor care urmează să fie actualizate. Valoarea actualizată va fi trimisă asincron după finalizarea tranzacției de actualizare, când se va şterge şi marcajul de „dirty”.

Algoritmul generic se modifică prin adăugarea unor paşi:

- verificarea marcajului „dirty”
- aşteptarea primirea valorii actualizate a datelor (dacă datele au fost marcate ca „dirty”)
- ştergerea marcajului.

O altă problemă care trebuie rezolvată: ce se întîmplă în cazul inserării sau ştergerii unor date? Rezolvarea poate veni prin folosirea marcajului de „dirty” pentru întregul tabel care se actualizează. Dacă actualizarea s-a efectuat pe o bază de date partiționată, nu se mai poate face nimic, iar interogarea pur şi simplu nu va ține seama de actualizată decât după ce conexiunea a fost refăcută.

4.3 Tranzacții de scriere

O tranzacție de scriere este o tranzacție care modifică datele şi este definită ca o unitate atomică de operații care este finalizată dacă şi numai dacă toate operațiile sunt finalizate [Gr93].

În modelul propus, modificarea algoritmilor prin adăugarea de caracteristici noi au impact numai în nodul care procesează cererea, algoritmul principal (partea de implementare a tranzacțiilor distribuite) rămâne neschimbat. În termeni de performanță, impactul ar trebui sa fie mic, căci modificările de procesare sunt mai puțin costisitoare decât ar fi (de exemplu) transmiterea unor mesaje prin rețea. Din punct de vedere al utilizatorului, replicile unei date trebuie tratate ca o singură entitate, iar operația de actualizare (executată pe replicile cu drept de scriere şi folosind protocoale distribuite precum 2PL), trebuie propagată şi la replicile cu drept de citire. Propagarea actualizărilor se face asincron, iar marcajele de „dirty” ajută la păstratea acurateții datelor.

Algorimul pentru gestionarea tranzacțiilor de actualizare se modifică astfel:
Versiunea veche:

- Trimite la replicile cu drept de scriere a mesajelor de pregătite
- ...../// alți paşi necesari
- Finalizarea tranzacției şi eliberarea blocărilor

Versiunea propusă:

- Verifică dacă relocarea fragmentelor este optimă, şi eventual realocă
- Trimite replicilor de citire mesaje de „dirty”
- Trimite replicilor cu drept de citire mesaje de pregătire
- ...../// alți paşi necesari
- Finalizarea tranzacției şi eliberarea blocărilor
- Trimite asincron replicilor de citire mesaje conținând valoarea actualizată, care determina ştergerea marcajului de „dirty”

Revenirea la starea initiala în the model: Metodele de revenire ar trebui să țină cont de existența procesului de relocare şi de faptul că fragmentele îşi pot schimba drepturile; ca atare, trebuie restaurate atât drepturile fragmentelor, localizarea lor cât şi datele din catalogul bazei de date. Modificările sunt însă mici, impactul acestora ar trebui sa fie neînsemnat în termeni de performanță.

4.4 Consistența în model

În model, fiecare nod vede operațiile de actualizare în aceeaşi ordine (consistența secvențială), chiar dacă această ordine poate fi diferită de ordinea reală. Chiar dacă actualizarea replicilor de citire se face asincron, administratorul bazei de date are posibilitatea ca în cazul tabelelor critice să folosească parametrul *MaxUT*, garantând în acest fel că replica a fost actualizată în acel interval de timp.

4.4.1 Demonstrarea corectitudinii modelului

Corectitudinea în tranzacțiile de scriere

Modelul propus nu schimbă radical conceptele şi modalitatea de execuție a tranzacțiilor de scriere într-o bază distribuită, doar le extinde prin adăugarea unor paşi: găsirea unui nod cu drept de scriere a datei respective, trimiterea cererii de actualizare la acel nod, etc..Ca atare, modelul propus oferă acelaşi grad de consistentă ca şi protocolul de actualizare din baza de date.

În cazul apariției unei erori în rezolvare cererii de actualizare, corectitudinea este asigurată de metodele de recuperare şi de revenire la starea inițială. În conformitate cu modelul, eroarea poate să apară în protocolul bazei de date (şi atunci revenirea la starea inițială este executat in baza de date) sau în paşii introduşi de model (revenirea la starea inițială şi procesul de recuperare este executat de procesul central din model)- metoda este detaliată în lucrare în capitolul „*Tratarea evenimentelor neprevăzute”.*

Corectitudinea în tranzacțiile de citire

Datorită actualizării asincrone a replicilor de citire, valoarea găsită într-un anumit moment poate să nu fie ultima valoare. Pentru a remedia aceasta problemă, se folosesc metode de marcare a replicilor de citire cu „dirty”; şi dacă şi această metodă nu este suficientă, pentru tabele critice se poate folosi parametrul *MaxUT.*

4.5 Interblocarea

Descoperirea interblocarii este mai greu de realizat într-o bază de date distribuită decât într-una centralizată [Hoxx, Ka01, Kn87], şi datorită acestui fapt s-au creat algoritmi speciali ca “edge-chasing” sau crearea unui graf global folosind grafuri locale „wait-for”. De exemplu, în tranzacțiile distribuite, interblocarea locală este detectată de *Oracle* prin analizarea grafului „waits for”, iar interblocarea globală prin time-out [Na02]. Ambele tipuri de interblocări sunt tratate în aceeaşi maniera. În *SQL Server*, interblocarea este detectată de thread-ul Lock Monitor prin inspectarea proceselor la fiecare 5 secunde şi căutarea unui ciclu de interblocare.

4.5.1 Interblocarea în modelul propus

Interblocarea nu poate fi complet prevenită, dar există anumiți factori (de exemplu: procese de extinderea a blocării sau dimensiunea fragmentelor blocate) care favorizează apariția

ei.. Blocarea în model se face la nivel de fragment, şi în funcţie de aplicaţie poate ajunge la nivel de înregistrare (ca şi în Oracle) sau la nivel de pagină (în funcţie de numărul de actualizări), minimizându-se astfel şansele de apariţie a interblocării.

Modificările care ar trebui efectuate în politica de găsire şi tratare a interblocării se referă în special la determinarea procesului care va fi anulat (în procesul *Lock Monitor Thread*). Dacă în mod tradiţional procesul se alegea pe baza criteriilor de prioritate şi de cost, în acest model se propune adăugarea un alt criteriu: tipul blocării – citire sau scriere. În general, se recomandă ca procesul de citire sa fie anulat deoarece este mai puţin costisitor decât un proces de scriere. Totuşi un proces de citire nu poate aştepta indefinit; prioritatea îi va creşte cu fiecare anulare sau îi va creşte proporţional cu timpul petrecut în sistem; la un moment dat de timp va fi mai mare decât a unui proces de scriere – şi astfel se va executa.

Ca atare, criteriile în model sunt:

- Tipul de blocare al procesului
- Prioritatea procesului şi
- Costul procesului.

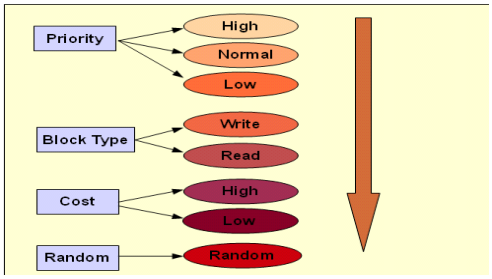

Fig.15 Vizualizarea criteriilor

5 Tratarea evenimentelor neprevăzute

În acest capitol sunt analizate diferite tipuri de erori care pot apărea după ce modelul va fi implementat. Sunt erori generale, dar soluţia dată precum şi procedeele de recuperare sunt modificate pentru a fi adaptate modelului propus. Tipurile de erori sunt organizate în două mari categorii în funcţie de factorul declanşator: erori umane sau erori de sistem. Recuperarea se face în funcţie de tipul erorii, iar erorile se pot grupa astfel:

- Erori datorate enunţurilor şi erori în procesul utilizatorului (întrerupere anormală a procesului, a sesiunii, excepţii în proces, etc.)
- Erori datorate greşelilor utilizatorului în adăugarea/ştergerea/actualizarea informaţiilor din baza de date.
- Erori cauzate de terminarea anormală a unei instanţe (pană de curent, erori în O/S, etc.)
- Erori cauzate de o problemă fizică care are loc când calculatorul încearcă să citească sau să scrie un fişier necesar operării cu baza de date (erori în capul de citire de pe disk, fişiere şterge, suprascrise sau corupte, etc.).
- Erori cauzate de partiţionarea sistemului (căderea conexiunii,) – recuperarea şi actualizarea datelor trebuie să folosească conexiunea căzută. În cazul special în care se debranşează un subsistem –
    - Trebuie avut în vedere ca numărul de replici pentru efectuarea unei actualizări să fie mai mare decât numărul jumătate plus unu din total de replici cu drept de scriere (se evită astfel posibilitatea actualizării unei date în ambele subsisteme).
    - Algoritmul de recuperare se modifică astfel:

```
Select the subsystem S1 with the smallest number of nodes.
For each node N în S1 Do
```

```
            Resynchronize node N using the restored connection
        End for
```

Algoritmul de recuperare pentru un nod poate fi detaliat astfel, cu mențiunea că „cel mai apropiat nod" e nodul cu cel mai bun timp de răspuns, iar actualizarea fiecărui tabel, view, se face prin suma de control(pentru a eficientiza procesul se foloseşte suma de control):

- *Rollback tranzacțiile începute şi nefinalizate*
- *Şterge blocarile pe fragmentele din nodul N*
- *Alege „cel mai apropiat" node N1*
- *Foloseste algoritmul optimizat „updateStructureAlgorithm" pentru a actualiza nodul N folosind nodul N1*
- *Dacă numărul de drepturi de scriere ptr o data D este mai mare decât W_Max, şterge dreptul de scriere*
- *Actualizeaza catalogul bazei de date în nodul N*
- *Actualizeaza catalogul bazei de date din restul nodurilor*

În cazul special în care este afectată întreaga bază de date, se face revenirea la starea dinaintea tranzacției şi baza de date este restaurată folosindu-se un fişier de log sau ultima versiune a bazei de date.

6 Eficiența în sistemul propus

Numărul de tranzacții efectuate de un sistem variază în funcție de anumite condiții: numărul de fragmente, dimensiunea acestora, etc; şi ca atare nu poate fi considerat ca o metrică în determinarea eficienței unui sistem. O metrică mai bună ar fi numărul de bytes modificați [Ha05]. Într-un model în care datele sunt replicate, eficiența este influențată şi de numărul replicilor şi de localizarea acestora. Conform [Va07], cel mai bun timp de răspuns se obține când sunt două copii ale datelor - de observat că s-a sugerat o valoare inițială de 2 pentru numărul minim de drepturi de scriere *W_Min. Numărul* de replici de citire şi modul în care sunt actualizate are un impact asupra eficienței, fapt luat în considerare când s-a propus ca actualizarea replicilor de citire să se facă asincron. Chiar dacă toti aceşti factori au fost luați in considerare, eficiența modelului poate fi doar dedusă şi nu demonstrată. Dovada eficienței se va face după ce modelul ca fi implementat.

6.1.1 Eficiența

Un model bun este mai presus de toate eficient. Modelul propus vrea să rezolve o problemă curentă: descrierea unui model dinamic în care timpul cel mai bun de răspuns este obținut cu un cost minim în termeni de transfer şi stocare (subcapitol 3.1). Acest obiectiv poate fi obținut prin mai multe metode: distribuirea încărcării, replicarea şi stocarea datelor în nodurile în care sunt cel mai des accesate, folosirea statisticilor de citire, folosirea unor algoritmi diferiți pentru interogările de citire/scriere, etc. Comparand ideea de la baza modelului cu cea enunțată de Yahiro M. (şi colegii) în patentul 5379424 - „*Distributed database management system for retrieving data files from databases selected based upon retrieval time*" – modelul are următoarele avantaje:

- Nu trebuie ca toate datele dintr-un tabel să fie stocate în acelaşi nod
- Dacă un nod este prea ocupat şi nu poate executa o interogare, ea va fi trimisă altui nod, dar nu se vor trimite fragmentele necesare rezolvării ei. Pentru a minimiza traficul,

transmiterea fragmentelor prin rețea se face după o analiză atentă (se ține cont de numărul de fragmente de pe un nod, de dimensiunea lor – subcapitolul 2.6.1).

Optimizarea interogărilor

Optimizarea interogărilor se referă la acel proces care găseşte cea mai bună strategie de execuție a unei interogări dintr-o mulțime de alternative [Va00]. O strategie de execuție pentru o interogare distribuită poate fi descrisă pe operații relaționale algebrice şi primitive de comunicare (operații de trimitere/primire) necesare transferului de date. Într-un model distribuit, între descompunerea interogărilor şi optimizarea lor intervin alți doi paşi: localizarea datelor şi optimizarea globală a interogării. Localizarea datelor se face uşor în model – folosind informațiile din catalogul bazei de date: *Replica Location*. Optimizarea interogării foloseşte din trei componente: un spațiu de căutare, un model de cost (definit în termeni de unități de timp se referă în general la spațiul de pe disk, la numărul de operații de I/O, la spațiul din buffer, cost CPU, costul de comunicație, etc.[Oz08]) şi o strategie de căutare. Va trebui văzut cum se pot aplica algoritmii de optimizare în cazul modelului propus.

6.2 Comparație cu BDD: Oracle RAC, Federated DB şi IBM DB2 PureScale

Arhitectura

Oracle RAC şi IBM DB2 PureScale s-au dezvoltat pe acelaşi tip de arhitectură: servere multiple (maxim 128 în cazul DB2) care rezolvă interogările venite şi care partajează un singur mediu de stocare – oferind astfel un grad mare de disponibilitate şi de scalabilitate. Acelaşi grad de scalabilitate va fi oferit în modelul propus- serverele pot fi adăugate şi îndepărtate din sistem, dar sistemul va oferi şi algoritmi pentru aceste cazuri (algoritmul de recuperare poate fi folosit la adăugarea unui nod în rețea). Această facilitate apare în Oracle abia de la versiunea 10g [Mi08], însă numai la nivelul mediului de stocare, nu şi al serverelor. Doar SQL Federated Database se poate lăuda cu o arhitectură distribuită şi la nivel de server şi la nivel de medii de stocare.

Noduri supraîncărcate

În cazul Oracle RAC şi DB2 PureScale, conexiunile sunt rutate la nodul cel mai puțin încărcat, iar în cazul FederatedDB, administratorul trebuie să partiționeze datele pentru a evita supraîncărcarea unui nod. În modelul propus, o interogare este retrimită altui nod numai dacă acesta are un cost de rezolvare mai mic. Optimizările prin care se folosesc statisticile de citire ajută la ştergerea datelor nefolosite şi implicit la creşterea eficienței şi la minimizarea şanselor de apariție a nodurilor supraîncărcate.

Backup-ul bazei de date

Nici una dintre bazele de date (Oracle RAC, DB2 sau FederateDB) nu oferă de una singură soluții pentru asigurarea datelor. Datorită faptului că în modelul propus datele sunt replicate (având cel puțin două replici de scriere), nu este necesară asigurarea datelor. Procesele de curățare şi realocare, precum şi cele de rezolvare a interogărilor de scriere monitorizeaza numărul de replici de scriere pentru a nu scădea sub valoarea *W_Min.*

Căderea unui nod

Datorită arhitecturii distribuite (pe partea de servere), căderea unui nod nu are repercursiuni nici în Oracle RAC, nici în DB2. Lucrurile nu stau la fel în cazul FederatedDB unde o parte din date nu mai sunt disponibile; puține aplicații pot tolera aşa ceva. Modelul propus nu

întâmpină probleme în cazul căderii unui nod – datele sunt replicate deci pot fi tot timpul accesate, iar cererile viitoare vor fi preluate de alte noduri. Momentan nu s-a luat în calcul (va trebui studiată) o modalitate prin care conexiunile curente la nodul căzut să fie transferate altui nod.

Eficiența

Oracle RAC lucrează cu un singur mediu de stocare, care nu face decât să servească cererile de date prin intermediul operațiilor I/O. Ținând cont că durata unei operații I/O este fixă (depinde de tipul de hard disk), mediul de stocare poate introduce întîrzieri în executarea interogărilor. DB2 PureScale introduce CF (acceleration facility) şi RDMA (Remote direct Memory Access) pentru a reduce comunicațiile dintre noduri referitoare la administrarea blocărilor şi la serviciile globale de caching. Dintre cele trei, FederatedDB se confruntă cu probleme legate de eficiența: aceeaşi interogare diferă în funcție de serverul căruia i se adresează. In model se propune a alta abordare: interogările se pot trimite la alte noduri, datele sunt replicate iar dacă un nod este supraîncărcat, datele necesare pot fi obținute de la alt nod (folosind o altă replică). Se poate asigura astfel un răspuns rapid şi eficient. O comparație exactă între eficiența sistemelor menționate şi a modelului propus este greu de făcut deoarece ar trebui analizată şi folosirea algoritmilor de optimizare a planurilor de execuție a interogărilor într-un context dinamic distribuit.

Administrarea spațiului

În momentul în care spațiul disponibil este plin în proporție de 85% / 97%, Oracle trimite un mesaj de atenționare / critic (de regula administratorului) pentru a lua măsuri corective. SQL Server a implementat un mecanism de monitorizare şi alertă bazat pe meta date extrase din baza de date, proces destul de scump şi ineficient [Ko08]. În modelul propus, intervenția umană nu ar mai este necesară, căci se va rula automat procesul de curățare a bazei de date când spațiul disponibil scade sub o anumită limită specificată de administrator.

Uşurința în folosire

Oracle RAC, DB2 şi modelul propus oferă acelaşi grad de uşurință în folosire: o abordare transparentă în care utilizatorul lucrează la fel ca într-o bază de date centralizată. Problema cea mai mare în cazul FederatedDB constă modalitatea de folosire: fiecare interogare trebuie pe personalizată pentru serverul căruia i se adresează caci fiecare modificare are implicații majore- trebuind modificate şi toate interogările care folosesc acele date- un proces lent şi extrem de scump.

7 Concluzie

În această teză se propune o metodă pentru distribuirea dinamică a fragmentelor unei baze de date distribuite precum şi o metodă de alocare dinamică a drepturilor de acces la aceste fragmente în funcție de şabloanele de acces la date. Noutatea acestui model este reprezentată de caracteristica sa dinamică: distribuția datelor se schimbă în funcție de şabloanele de acces pentru a oferi un grad înalt de disponibilitate cu un cel mai mic cost. Sub acest aspect, un întreg model este propus: metode de procesare a tranzacțiilor, metode de curățare şi relocare, metode de prevenire şi rezolvare a interblocărilor precum şi metode de recuperare. Costul, văzut din perspectiva costului de transmitere în rețea, a costului de relocare şi de execuție a fost o preocupare continuă, deoarece modelul se dorește a fi nu numai inovativ ci şi eficient.

Modelul ar oferi un grad mare de disponibilitate datorat arhitecturii distribuite şi numărului variabil de servere. Modelul este scalabil, oferind alternative în cazul în care serverele

rămân fără memorie. În fiecare nod sunt stocate cele mai des aceste fragmente – deci pentru majoritatea interogărilor datele au un grad mare de disponibilitate – contribuind la creşterea performanţei modelului. Pentru a optimiza şi mai mult, se vor folosi algoritmi de optimizare în baze de date distribuite. Distribuirea încărcării se face în cadrul procesului de rezolvare a interogărilor (retrimiterea interogării la alt nod sau în operaţii sql de tipul inserare în masă).

Administratorii bazelor de date întâlnesc dificultăţi în administrarea acestora; dificultăţi datorate creşterii rapide şi cererii de conectare non-stop, astfel lucrările de întreţinere care presupun deconectarea mediilor de stocare sunt tot mai greu de programat. Metodele de recuperare prezentate în model ar putea oferi soluţii pentru automatizarea anumitor procese – în cazul în care se adaugă un mediu de stocare nou, modelul ar putea sa actualizeze structura bazei de date, sa distribuie fragmentele în reţea, etc. De asemenea modelul propune administrarea pro-activă a spaţiului de stocare: când spaţiul disponibil scade sub un anumit nivel sa se ia măsuri corective prin rularea algoritmului de curăţare şi prin ştergerea datelor accesate mai rar.

Ar trebui sa fie uşor pentru un DBA să modifice comportamentul modelului – s-a propus un model puternic configurabil şi parametrizat pentru a putea veni în întâmpinarea oricăror cerinţe. Replicarea datelor şi metodelor de recuperare descrise vor ajuta modelul sa ofere protecţie împotriva dezastrelor şi a căderilor hard. Metodele de recuperare pentru un nod vor putea oferi o recuperare rapidă şi completă, actualizând structura bazei de date, fragmentele, precum şi informaţiile din catalogul bazei de date fără a necesita intervenţia umană.

O simplă propoziţie poate concluziona: modelul propus nu este numai inovativ ci şi performant, configurabil, uşor de administrat. Datele vor fi replicate, distribuind încărcarea şi oferind toleranţă în caz de erori.

Planurile viitoare vizează o implementare sumară a modelului dar şi executarea de teste de eficienţă şi performanţă, teste de simulare pentru a determina valorile iniţiale ale parametrilor în funcţie de şabloanele de acces.

8 Referințe

[Ab05] Abbasi A.,*Oracle Database Administration Concepts & Implementation Made Simple,* ISBN: 0-9770739-0-4, 2005, pp.78-80

[Ab88] Abbot R., Garcia-Molina H., *Scheduling Real Time Transactions: A Performance Evaluation,*Proceedings of the 14th International Conference on Very Large Data Bases, ISBN:0-934613-75-3,1988, pp.1-12

[Al03] Alkhatib G., Labban R.S., *Transaction Management în Distributed Database Systems: the Case of Oracle's Two-Phase Commit*, Journal of Information Systems Education, Vol. 13(2)*,* http://www.jise.appstate.edu/Issues/13/095.pdf*,* 2003 pp. 95-103

[Al05] Alapati S.R., *chapter Oracle Transaction Management*, *Expert Oracle Database 10g Administration*, ISBN: 978-1-59059-451-3 (Print) 978-1-4302-0066-6 (Online), 2005, pp. 225-275

[Ala07] Alapati S.R., Kim C., *Oracle Database 11g NewFeatures for DBAs and Developers,* ISBN-13: 978-1-59059-910-5, ISBN-10: 1-59059-910-1, 2007, pp. 128-130

[Au04] Mike Ault, Madhu Tumma, *Oracle 10g Grid & Real Application Clusters Oracle10g Grid Computing with RAC*, ISBN 0-9744355-4-6, ISBN13 978-0974435541, 2004, pp. 581-584

[Au08] Auvray S., *StrokeDB, Just Another Distributed Database? Not Really.,* http://www.infoq.com/news/2008/04/distributed-db-strokedb , Apr. 2008

[Ba01] Baker M., *Cluster Computing White Paper* , http://arxiv.org/ftp/cs/papers/0004/0004014.pdf, Jan 2001

[Ba91] Barghouti N.S., Kaiser E., *Concurrency control în advanced database applications,* ACM Computing Surveys, Vol, 23, No 3, ISSN 0360-0300, Sep 1991, pp 269 – 317

[Ba92] Badrinath B.R., Ramamritham K.,*Semantics Based Concurrency Control: Beyond Commutativity,* ACM Transactions on Database Systems, Vol.17, Issue 1, ISSN:0362-5915, 1992, pp.163-199

[Ba95] Balanescu T., *Corectitudinea algoritmilor,* Eds. Tehnica, ISBN 911-T-T, 1995

[Be81] Bernstein P.A., Goodman N., *Concurency Control în Distribuited Database Systems,* ACM Computing Surveys (CSUR), Vol 13. , No2., ACM 0010-4892/81/0600-0185, 1981 pp. 185 – 221

[Bi09] Birman K., *A History of the Virtual Synchrony Replication Model,* http://www.cs.cornell.edu/ken/History.pdf, 2009

[Bi87] Birman K.P., Joseph T., *Exploiting virtual synchrony în distributed systems,* Proceedings of the 11th ACM Symposium on Operating systems principles, Austin Texas, Vol. 21, Issue 5, ISSN:0163-5980, Nov. 1987, pp.123 – 138

[CaDo09] Castro P., Dockter B., Divringi L., Sundaresan S., *US Patent 7503052 - Asynchronous database API,* Mar 2009

[CaJ09] Callison J., *Database Locking: What it is, Why it Matters and What to do About it,* http://www.peakperformancetechnologies.com/; Issue of Methods & Tools, 2009

[Ca09] Callagham M., *The futures of replication în MySQL,* http://www.facebook.com/note.php?note_id=126049465932, Aug, 2009

[Ch02] Cheng C.H., Lee W.K., "*A Genetic Algorithm-Based Clustering Approach for Database Partitioning"*, IEEE Transactions on Systems Man and Cybernetics Part C-Applications and Reviews*,* 32: 215-230, 2002.

[Chxx] Chapple M., *Sql Server Replication,* http://databases.about.com/cs/sqlserver/a/aa041303a.htm

[Co02] Connolly T., Begg C., *Database Systems*. ISBN-10: 0321210255, ISBN-13: 978-0321210258, 2002, pp. 572-629, 780-800

[Co07] Cox K., Marinucci T., Bevilacqua S., *Distributed Partitioned Views / Federated Databases: Lessons Learned,* http://sqlcat.com/technicalnotes/archive/2007/09/11/distributed-partitioned-views-federated-databases-lessons-learned.aspx, 2007

[Co71] Coffman E.G., Elpnick M., Shoshani A., *System Deadlocks***,** ACM Computing Surveys (CSUR)Vol. 3 ,Issue 2, ISSN:0360-0300, June 1971, pp. 67-78

[Da04] Darabant A, *Specificare şi modelare obiectuală în baze de date distribuite*, PhD Thesis, Library of Babes Bolyai University, Cluj Napoca, 2004

[Da09] Darabant A., *Proiectarea bazelor de date distribuite,* ISBN 9789731336057, Eds. Casa cartii de Stiinta, Cluj-Napoca, 2009

[De09] Delaney K., *Lock modes: Microsoft SQL Server 2008 Internals,* Print ISBN-13: 978-0-735-62624-9, Mar. 2009, pp. 598 - 618

[Di00] Diestel R., *"Graph Theory"*, *Springer-Verlag,* Heidelberg 2000, Electronic Edition.

[Di77] Dijkstra E.W., *Selected Writings on Computing: A Personal Perspective,* Springer-Verlag, ISBN 0-387-90652-5, 1982, pp.308-312

[Do99] Donovan R., Martin J., *High Performance Legacy Data Propagation*, DM Review Magazine, March 1999

[Dy99] Dye C,. *Oracle Distributed Systems*, ISBN 10:1-56592-432-0, ISBN 13: 9781565924321, Apr 1999

[Ed97] Edwards W.M., Hidalgo D. S., Williamson, L. A., *US Patent 5689697 - System and method for asynchronous database command processing,* 1997

[En03] Engler D., Ashcraft K. *RacerX: Effective, static detection of race conditions and deadlocks,* 19th ACM Symposiom on Operating Systems Principles, 2003.

[Es07] Esakkirajan S., Sumathi S., *Fundamentals of Relational Database Management Systems*, ISSN print edition:1860-949X, ISSN electronic edition: 1860-9503, ISBN-10: 3-540-48397-7, ISBN-13: 978-3-540-48397-7, 2007, pp.568-572, 578-586

[Fl03] Flickenger R., *Linux Server Hacks,* Print ISBN: 978-0-596-00461-3, ISBN10: 0-596-00461-3, Jan. 2003, hack 79

[Fr09] Fritcey G., Dam S., *chapter. Deadlock Analysis, SQL Server 2008 Query Performance Tuning Distilled,* ISBN-13: 978-1-4302-1902-6, ISBN-13(online): 978-1-4302-1903-3, 2009, pp. 401- 414

[Ga03] Garmany J., Freeman R.G., Burleson D.,*Oracle Replication Snapshot, Multi-Master and Materialized Views Scripts*, ISBN 0972751335, 9780972751339, 2003, pp. 16-20

[Gh03] Ghemawat S., Gobioff H., Leung ST., *The Google File System,* labs.google.com/papers/gfs-sosp2003.pdf, 2003

[GM82] Garcia-Molina H., Wiederhold G., *Read-Only Transactions în a Distributed Database,* ACM Transactions on Database Systems (TODS) Vol. 7 , Issue 2,ISSN:0362-5915, 1982, pp. 209 – 234

[Gr03] Graham J., "*Efficient Allocation în Distributed Object Oriented Databases"*, Proceedings of the ISCA 16th International Conference on parallel and Distributed Computing Systems , Reno Nevada, August 2003, pp150-154.

[Gr05] Gray J., van Ingen C., *Empirical Measurements of Disk Failure Rates and Error Rates,* Microsoft Research Technical Report MSR-TR-2005-166, http://research.microsoft.com/pubs/64599/tr-2005-166.pdf, Dec. 2005

[Gr93] Gray J., Reuter A., *Transaction Processing:Concepts and Techniques*, Morgan Kaufmann Publishers,Inc. ISSN 1046-1698, 1993, pp. 426-428

[Gr96] Gray J., Helland P., O'Neil P., Shasha D., *The Dangers of Replication and a Solution,* Proceedings of the 1996 ACM SIGMOD international conference on Management of data, ISBN:0-89791-794-4, 1996, pp.173-182

[Go08] Gold B.T., Ailamaki A., Huston L., Falsafi B., *Accelerating Database Operators Using a Network Processor*, White Papers, http://www.silicon.com/white-

papers/components/2008/12/04/accelerating-database-operators-using-a-network-processor-60502340/ , Dec. 2008

[Ha05] Harris, T., Marlow, S., Jones S. P., and Herlihy, M., *Composable Memory Transactions*, ACM Conference on Principles and Practice of Parallel Programming 2005 (PpoPP'05). 2005, pp. 48-60.

[Ha06] Hall T., *Oracle PL/SQL Tuning. Expert Secrets for High Performance Programming,* ISBN: 0-9761573-9-X, ISBN 13: 978-0976157397 , 2006, ch. 3

[Ha90] Haritsa J.R., Carey M.J., Livny M., *Dynamic Real Time Optimistic Concurrency control,* Proceedings of ACM PODS 1990,

[He03] Herlihy M., Luchangco V, Moir M, and Scherer III W., *Software Transactional Memory for Dynamic-Sized Data Structures*. Proceedings of the Twenty-Second Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC) 2003, pp. 92–101.

[He07] Hernandez G., *Locking*, http://www.georgehernandez.com/h/xComputers/Databases/SQLServer/Locking.asp, 2007

[Ho08] Horvat M., *Reallocation în Dynamic Distributed Database Model*, Proceedings of the National Conference ZAC 2008, (Zilele Academice Clujene 2008), ISBN: 978 973 610 730 6, 2008, pag 171-180,

[Ho09] Horvat-Petrescu M., *Optimal dynamic distribution în a distributed database,* Information Technologies' 2009 -15th International Conference on Information and Software Technologies. Research Communications/ Kaunas University of Technology. Kaunas: Technologija, ISSN 2029-0039, 2009, pp. 159 - 165

[HoSm05] Ho A., Smith S., Hand S., *Technical Report Number 633, On deadlock, livelock, and forward progress*, ISSN 1476-2986, May 2005

[Hoxx] Holliday, J.A., Abbadi A., *Distributed Deadlock Detection,* Encyclopedia of Distributed Computing, Kluwer Academic Publishers, accepted for publication.

[Hp03] *TPC Benchmark C Full Disclosure Report for hp Integrity rx5670 Cluster 64P Using Oracle Database 10g Enterprise Edition with Real Application Cluster and Partitioning; and Red Hat Enterprise Linux AS 3,* http://www.tpc.org/results/FDR/TPCC/HP%20Integrity%20rx5670%20Cluster %2064P_FDR.pdf, 2003

[Hu01] Huang Y., Chen J., *"Fragment Allocation în Distributed Database Design"*, Fragment Allocation în Distributed Database Design, Journal Of Information Science And Engineering, 17, 2001, pp. 491-506

[IB09] IBM White Paper, *Transparent Application Scaling with IBM DB2 pureScale*, ftp://ftp.software.ibm.com/software/data/sw-library/db2/papers/db2-pure-scale-wp.pdf, Oct. 2009, pp.4 - 10

[Ja09] Jacobs K., *InnoDB.Innovative Technologiesfor Performance andData Protection,* MySQL Conference, April 2009

[Ju08] Jula H., Tralamazza D., Zamfir C., Candea G., *Deadlock Immunity: Enabling Systems To Defend Against Deadlocks,* Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Dec 2008

[Ju98] Jung I., Lee J., Moon S., *Concurrency control în multidatabase systems: A performance study,*Journal of Systems Architecture: the EUROMICRO Journal, ISSN:1383-7621, Oct. 1998, pp. 97 - 114

[Ka01] Kaveh N., Emmerich W., *Deadlock Detection în Distributed Object Systems,* Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, ISBN: 1-58113-390-1, 2001, pp.44-51

[Ka07] Karanja E., Badamas M.A., *An organizational data replication model for mobile databases.,* Journal of Academy of Business and Economics, ISSN: 1542-8710, Mar 1, 2007, 122-129

[KaSi07] Kaur N., Singh R., Misra M., Sarje A. K., *Secure Transaction Management Protocols for MLS/DDBMS,* Springer Berlin / Heidelberg, ISSN: 0302-9743 (Print) 1611-3349 (Online), ISBN: 978-3-540-77085-5, 2007, pp.219-233

[Ke01] Keating B., *Challenges Involved în Multimaster Replication,* Database Specialists, Inc., http://www.dbspecialists.com, 2001

[Kn87] Knapp Edgar, *Deadlock detection în distributed databases,* ACM Computing Surveys (CSUR) Vol. 19 , Issue 4, ISSN:0360-0300, Dec. 1987, pp.303-328

[Ko08] Sergey Koltakov, Mughees Minhas, *Technical Comparison of Oracle Database 11g and SQL Server 2008: Focus on Manageability,* http://www.oracle.com/technology/products/manageability/database/pdf/competitive/ss_2008_vs_oracle_11g_tech_comparison.pdf, Dec. 2008

[KoHe08] Koskinen E., Herlihy M. *Deadlocks: Efficient deadlock detection.,* Proceedings of the 20th ACM Symposiom on Parallelism în Algorithms and Architectures, 2008.

[KoMi08] Koltakov S., Minhas M., *Technical Comparison of Oracle Database 11g and Sql Server 2008: Focus on Manageability. An Oracle White Paper.*, http://www.oracle.com/technology/products/manageability/database/pdf/competitive/ss_2008_vs_oracle_11g_tech_comparison.pdf, Dec. 2008

[Kr07] Kroenke D. M., David J. A. *Database Concepts. 3rd ed*. New York: Prentice, ISBN-10: 0131986252, ISBN-13: 978-0131986251, 2007

[Ku09] Kumar A., *Pros and Cons of Distributed Databases,* http://www.articlealley.com/article_911011_11.html, Jun. 2009

[La98] Lamport L., *The part-time parliament.* ACM Transactions on Computing Systems Vol. 16, Issue 2, ISSN:0734-2071, May 1998, pp. 133 – 169

[Le85] Leung C. H. C., Wolfenden K., *Analysis and Optimisation of Data Currency and Consistency în Replicated Distributed Databases,* The Computer Journal, Vol 28, No. 5, 1985, pp. 518-523

[Lu06] Lungu I., Fodor A.G., *"Optimizing Queries în Distributed Systems"*, Revista Informatica Economica nr. 1 (37), 67-72, 2006.

[Ma01] Markus T., Moroşanu C., Varga V.: *Stochastic Query Optimization în Distributed Databases using Semijoins*, Annales Universitatis Scientiarum Budapestinensis de Rolando Eotvos Nominatae Sectio Computatorica 20, 2001, pp. 107-131.

[McPr07] McElroy P., Pratt M., *Oracle Database 11g: Oracle Streams Replication,* http://www.oracle.com/technology/products/dataint/pdf/twp_streams_replication_11gr1.pdf , July 2007

[Mc07] McGehee B., *SQL Server Federated Database Performance Tuning Tips,* http://www.sql-server-performance.com/tips/federated_databases_p1.aspx, Mar. 2007

[McSt07] McRobb S., Stahl B.C., *Privacy as a shared feature of the e-phenomenon: a comparison of privacy policies în e-government, e-commerce and e-teaching,* http://www.ccsr.cse.dmu.ac.uk/jpapers/papers/Mcrobb_2007_ijitm.pdf, International Journal of Information Technology and Management, Vol. 7, 2007, pp. 232-249

[Mc09] Mckoy Distributed Database, *Introduction,* http://www.mckoi.com/index.html, http://www.mckoi.com/Introduction.html, 2009

[Mi08] Michalewicz M., Newlan P., Lundhild B., *Oracle Real Application Clusters 11g, Technical Comparison with Microsoft SQL Server 2008, An Oracle Competitive white Paper*, http://www.oracle.com/technology/products/database/clustering/pdf/twp_racsqlserver_2008.pdf, 2008

[Mo08] Moldovan G., Valeanu M., *The performance optimization for date redistributing system în computernetwork*, International Journal of Computer, Communications & Control, ISSN 1841-9836, E-ISSN1841-9844, Vol III, Supl. issue, 2008, pp. 116-118.

[Mo93] Mosberger D., *Memory Consistecy Models,* ACM Operating Systems Reviews, Vol. 27, Issue 1, 1993, pp.18-26

[MS5.1] *MySQL 5.1 Reference Manual:: 16 Replication,* http://dev.mysql.com/doc/refman/5.1/en/replication.html

[Mu05] Muntean C., *Using the database replication technology for the jet engine,* Revista Informatica Economică, nr. 1(33)/2005 , pp. 108-111

[Na02] Nagarkar N., *Autonomous and Distributed Transactions în Oracle 8i/9i*, Database Journal, http://www.databasejournal.com/features/oracle/article.php/1550951/Autonomous-and-Distributed-Transactions-în-Oracle-8i9i.htm, Dec, 2002

[Ne09] Newman H., *RAID's Days May Be Numbered,* http://www.enterprisestorageforum.com/technology/features/article.php/3839636,Sep. 2009

[OB08] O'Brien J., Marakas G.M., *Management Information Systems,* ISBN: 0073511544, 2008, pp.185-189

[Oi07] Oiaga M., *Microsoft Takes Peer-to-Peer to the Next Level with a New Synchronization Platform,* http://news.softpedia.com/news/Microsoft-Takes-Peer-to-Peer-to-the-Next-Level-with-a-New-Synchronization-Platform-70009.shtml, Nov. 2007

[Or8i01] *Replication Overview,* Oracle8*i* Replication Release 2 (8.1.6)**,** Part Number A76959-01,
http://www.mscd.edu/~ittsdba/oradoc817/server.817/a76959/repover.htm#14080, 2001

[Öz08] Özsu M.T., Valduriez P., *Distributed and Parallel Database Systems,* http://www.silicon.com/white-papers/server-hardware/2008/04/10/distributed-and-parallel-database-systems-60308215/, Apr. 2008

[Öz94] Özsu M.T., Valduriez P., Dayal U., Distributed Object Management, ISBN-13: 9781558602564, ISBN: 1558602569, 1994, pp.212 – 231

[Öz99] Özsu M.T., Valduriez P., *Principles of Distributed Database Systems,* Prentice Hall, ISBN 0-13-659707-6, 1999

[Pa06] Pavlin I. *Transaction and Concurrency în Oracle Server,* http://silicondetector.org/display/ds/Transaction+and+Concurrency+în+Oracle+Server, 2006

[Pa92]Pang H., Livny M., Carey M.J.,*Transaction Scheduling în Multiclass Real-Time Database Systems*, Sept 1992

[Pi00] Piattini M., Diaz O., *Advanced Database Technology and Design*, ArtechHouse Publishing, ISBN-10: 0890063958, ISBN-13: 978-0890063958, ISBN:0890063958, 2000

[Pu91] Pu C., Leff A., *Replica Control în Distributed Systems: An Asynchronous Approach* Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data, ISBN ISBN:0-89791-425-2, 1991, pp 377-386

[Ra04] Ramamritham K.,Sang H.S., DiPippo L.C., *Real-Time Databases and Data Services,* International Journal of ISSN:0926-8782 ,Vol. 28, Issue 2-3 , pp. 179-215

[Ro03] Rothschild M., *US Patent 6567823 - Change propagation method using DBMS log files,* http://www.patentstorm.us/patents/6567823-fulltext.html, May 2003,

[Ro09] Rob P., Coronel C., *Database systems: design, implementation and management*, ISBN-13: 978-1-4239-0202-0, 2009

[Sa93] Satyanarayanan O.T., Agrawal, *Efficient execution of read-only transactions în replicated multiversion databases,* IEEE Transactions on Knowledge and Data Engineering,Vol. 5, Issue 5, ISSN:1041-4347, Oct 1993, pp: 859-871

[ScGi07] Schroeder B., Gibson G.A., *Disk Failures în the Real World: what does an MTTF of 1,000,000 hours Mean to You?* Proceedings of the 5th USENIX Conference on File and Storage Technologies, 2007

[Sc07] Schlichting D.,*What's New în SQL Server 2008*, Database Journal, http://www.databasejournal.com/features/mssql/article.php/3702381/Whats-New-în-SQL-Server-2008-Part-3.htm, Oct.2007

[Sc08] Schwartz B., Zaitsev P., Tkachenko V., Zawodny J.D., Lentz A., Balling D, *High performance MySQL (second edition)*, ISBN: 980-0-596-10171-8, June 2008, pp. 262 – 264

[Sh07] Shi J.Y., *Why Synchronous Parallel Transaction Replication is Hard, But Inevitable?A White Paper,* CTO Parallel Computers Technology Inc. (PCTI), 2007

[Si05] Sitar-Taut D., *Baze de date distribuite*, ISBN 9736510380, 2005, pp210-215

[Sl07] Sleit A., AlMobaideen W., Al-Areqi S., Yahya A., "*A Dynamic Object Fragmentation and Replication Algorithm In Distributed Database Systems",* American Journal of Applied Sciences 4 (8)*,* 2007, pp. 613-618

[St07] Stahl B.C., *Reflective responsibility for risk: a critical view of software and information systems development risk management,* http://www.ccsr.cse.dmu.ac.uk/jpapers/papers/2007_reflective_resp_risk_IJRAM.pdf, International Journal of Risk Assessment and Management, Vol. 7, No. 3, 2007, pp. 312-325

[St09] Steinberg D.H, *Asynchronous and Synchronous,* http://weblogs.java.net/blog/2003/10/24/asynchronous-and-synchronous*,* 2003

[Ta03] Tambulea L., *Baze de Date*, 2003.

[TaHo08] Tambulea L., Horvat M., *Dynamic Distribution Model în Distributed Database*, Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844 Vol. III (2008), Suppl. issue: Proceedings of ICCCC 2008, pp. 512-515

[TaLHo08] Tambulea L., Horvat M., *Redistributing Fragments into a Distributed Database*, Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844, Vol. III (2008), No. 4, pp. 384-394

[Th05] Thomas B., *Solutions for Highly Scalable Database Applications. An analysis of architectures and technologies,* http://download.microsoft.com/download/a/4/7/a47b7b0e-976d-4f49-b15d-f02ade638ebe/OracleRAC.pdf , Performance Tuning Corporation, 2005

[Th09] Thalman L., Kindahl M., *New Replication Features*, http://assets.en.oreilly.com/1/event/21/New%20Replication%20Features %20Presentation.pdf, Apr. 2009

[Tu09] Tuuri H., Sun C., *InnoDB Internals: InnoDB File Formats and Source Code Structure,* MySQL Conference, April 2009

[Ul03] Ulus T., Uysal M., *"Heuristic Approach to Dynamic Data Allocation în Distributed Database Systems",* Pakistan Journal of Information and Technology 2 (3), 2003, pp. 231-239.

[Up08] Upadhyaya S., Lata S., *"Task allocation în Distributed computing VS distributed database systems: A Comparative study"*, IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.3, March 2008.

[Va00] Varga V., *"Optimal Strategies for Query Processing în Distributed Databases*", Teza de Doctorat, Biblioteca Universitatii Babes Bolyai, Cluj Napoca, 2000

[Va06] Varga V., *Interogarea bazelor de date distribuite*, Casa Cartii de Stiinta, Cluj-Napoca, ISBN 973-686-842-7, 2006.

[Va07] Vasileva S., Milev P, Stoyanov B., *Some Models of a Distributed Database Management System with Data Replication,* International Conference on Computer Systems and Technologies – CompSysTech'07, http://ecet.ecs.ru.acad.bg/cst07/Docs/cp/SII/II.12.pdf, ISBN:978-954-9641-50-9 , 2007

[VM07] VMware technical note, *Round-Robin Load Balancing,* http://www.vmware.com/pdf/vi3_35_25_roundrobin.pdf, Dec. 2007

[Wa08] Waldron T., *OS/2's Symmetrical Multiprocessing Demystified*, http://www.edm2.com/index.php/OS/2%27s_Symmetrical_Multiprocessing_Demystified, July 2008

[We02] Weikum G., Vossen G., *Transaction Information Systems Theory, Algorithms, and Practice of Concurrency Control and Recovery*, ISBN 1558605088, 2002

[We10] Werner V., *Choosing Consistency,* http://www.allthingsdistributed.com/, 2010

[Wi05] Williams A., Thies W., Ernst M.D. *Static deadlock detection for Java libraries*. 19th European Conference on Object-Oriented Programming, 2005.

[Wo92] Wolfson O., Jajodia S., *"An Algorithm for Dynamic Data Distribution",* Proceedings of the 2nd Workshop on the Management of Replicated Data (WMRD-II), Monterey, CA, Nov. 1992.

[Yo08] Young S.T., Givens M., Gianninas D., *Adobe® AIR™ Programming Unleashed, chapter Synchronous Versus Asynchronous Database Operations,* ISBN-10: 0-672-32971-9, Print ISBN-13**:** 978-0-672-32971-5, Nov.2008, pp. 145 – 151

[Yo95] Yojiro M., Sekiguchi K., Muranaga M., Kato N., *US Patent 5379424 - Distributed database management system for retrieving data files from databases selected based upon retrieval time,* http://www.patentstorm.us/patents/5379424.html*,* 1995

[Xi04] Xiong M.,Agarwal S.,*SQL Server 2000 Incremental Bulk Load Case Study,* http://technet.microsoft.com/en-us/library/cc917716.aspx*,* Sept 2004