Realizarea serializabilității în prelucrarea bazelor de date din sisteme paralele distribuite cu schedulere fără blocare

Ing. Georgios KAPENEKAS
Technical Education Institute, Halkida, Greece

Teoria serializabilității se bazează pe o serie de reguli matematice care permit să se verifice dacă un scheduler lucrează corect. Necesitatea acestei verificări se datorează posibilității ca două sau mai multe procese care se execută simultan pe un sistem de calcul cu arhitectură paralelă să interacționeze. Execuția paralelă a unei mulțimi de tranzacții se reprezintă printr-o structură numită istoric. El indică ordinea în care operațiile tranzacțiilor din procese se execută. Scopul algoritmilor prezentați este obținerea unor istorice serializabile, care să reprezinte o execuție serializabilă. Tipul de schedulere prezentate nu utilizează metoda de blocare la baza de date. Ele permit dezvoltarea atât a schedulerelor de tip conservativ cât și a celor de tip agresiv. Utilizarea acestor schedulere permite calcularea în paralel a expresiilor relaționale pentru bazele de date, ceea ce conduce la optimizarea execuției.
Cuvinte cheie: *bază de date, manager, scheduler, mărci de timp, graf de serializare, garant*

1. Modele ale bazei de date

Controlul concurenței este activitatea de coordonare a acțiunilor unor procese care lucrează în paralel, cu acces la date comune, caz în care există posibilitatea interferenței proceselor. Pentru a studia paralelismul în calculul relațional se folosește un model al bazei de date. Componenta principală a acestui model este *tranzacția*. În mod neformal, o tranzacție este o execuție a unui program care accesează o bază de date comună. Scopul controlului concurenței este de a asigura ca tranzacțiile să se execute *atomic*, sau altfel spus:

1. Fiecare tranzacție accesează date comune fără a interfera cu alte tranzacții.
2. Dacă o tranzacție se termină normal, atunci efectele ei sunt permanente sau nu au nici un efect. Sunt două motive principale care pot determina ca tranzacțiile să nu fie atomice:

1. Într-un sistem cu partajare temporară activitățile asociate cu două sau mai multe tranzacții se pot executa simultan. Timpul alocat pentru o tranzacție T se poate încheia în timpul procesării și activitățile din alte tranzacții să se execute înainte ca T să se încheie.
2. O tranzacție s-ar putea să nu se încheie deloc. Ea se poate abandona datorită unei erori (un calcul ilegal) sau deoarece are nevoie de anumite date pentru care nu are privilegiul de acces necesar. Sistemul de gestiune însuși poate forța încheierea anormală a unei tranzacții.

Pentru a gestiona paralelismul, baza de date trebuie împărțită în **articole**, care sunt unitățile de date la care accesul este controlat. Modul cel mai utilizat pentru controlul accesului la un articol este "lock-ul". Un lock este un privilegiu de acces la un singur articol. Accesul poate fi acceptat sau anulat pentru o tranzacție. Funcție de modelul bazei de date se pot întâlni diferite tipuri de lock-uri. În cadrul gestionării bazei de date se păstrează și un tabel care constă din înregistrări de tipul: (<articol>, <tip lock>, <tranzacții>).

Pentru o astfel de înregistrare (I,L,T) semnificația este că tranzacția T are un lock de tip L pentru un articol I. Este posibil, funcție de tipul lock-ului, să existe una sau mai multe înregistrări pentru un articol, existând una sau mai multe tranzacții ce operează asupra aceluiași articol.

Pentru un sistem care permite sau blochează accesul la articole poate apare un fenomen inacceptabil. Să presupunem că T1 are acces la un articol A și o tranzacție T2 cere permisiunea de-a accesa acest articol. Cum accesul la A este blocat datorită lui T1, tranzacția T2 intră în stare de așteptare. După un timp tranziția T1 eliberează accesul la articol, dar între timp o altă tranziție T3 cere permisiunea de-a accesa articolul A. Primind dreptul de acces T3, tranzacția T2

rămâne în stare de aşteptare. În anumite condiţii fenomenul se poate repeta pentru noi tranzacţii, astfel că T2 să rămână în stare de aşteptare continuă. Fenomenul se numeşte **livelock** şi trebuie evitat prin sistemul de gestiune al bazei de date.

Să presupunem două tranzacţii T1 şi T2:

T1: LOCK A; LOCK B; UNLOCK A; UNLOCK B;

T2: LOCK B; LOCK A; UNLOCK B; UNLOCK A;

Dacă la un moment dat cele două tranzacţii reuşesc să execute prima instrucţiune de blocare a articolelor A şi B în pasul al doilea, nici una din ele nu va primi dreptul de acces la următorul articol de care are nevoie. O situaţie în care fiecare membru al unei mulţimi S, de două sau mai multe tranzacţii, aşteaptă eliberarea unui articol care este blocat de o altă tranzacţie din aceeaşi mulţime S, se numeşte **deadlock**. Cum fiecare tranzacţie din S aşteaptă, ea nu va putea elibera articolul de care are nevoie o alta. Pentru a evita deadlockul există soluţiile:

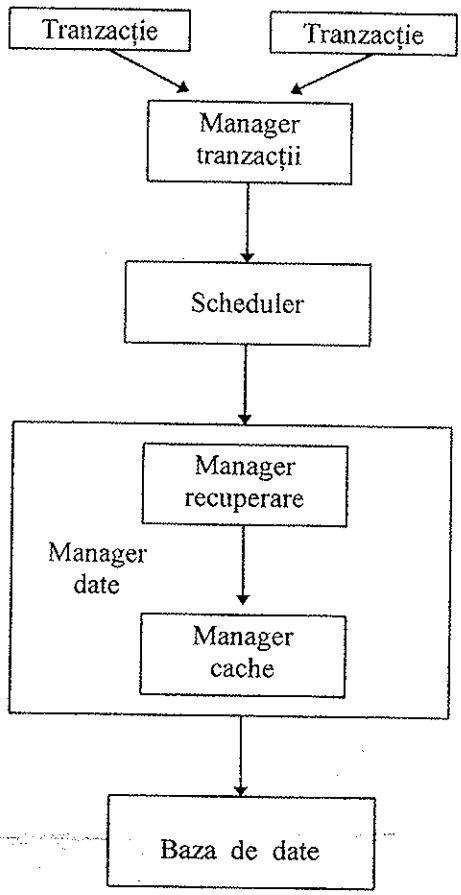
1. Se impune tranzacţiilor să ceară toate lock-urile odată. Sistemul, dacă este posibil, le acceptă pe toate, iar în caz contrar nu se ia în seamă nici unul, procesul fiind trecut în stare de aşteptare.
2. Se determină o ordine liniară a articolelor şi se cere tranzacţiilor ca cererile de acces pentru articole să aibă loc în această ordine.

Modelul sistemului bază de date

În studiul controlului paralelismului se foloseşte următorul model al structurii interne a sistemului bazei de date, format din modulele: *manager tranzacţie*, care execută orice cerere de procesare a bazei de date şi operaţiile tranzacţionale pe care le primeşte de la tranzacţii; *scheduler*, care controlează ordinea relativă de execuţie a operaţiilor asupra bazei de date şi a operaţiilor tranzacţionale; *manager cache*, care operează direct pe baza de date.

Operaţiile pe baza de date şi de tranzacţii transmise de o tranzacţie către sistemul bazei de date este recepţionat de managerul de tranzacţii. Operaţiile parcurg apoi schedulerul, managerul de recuperare şi managerul cache. Fiecare nivel transmite cereri şi primeşte răspunsuri de la următorul nivel.

Managerul cache. Un sistem este compus din două tipuri de memorii: volatile şi stabile.



Datorită limitării dimensionale a memoriei volatile sistemul bazei de date păstrează doar o parte din date în memorie, porţi-une numită *cache*. Conducerea lucrului în cache este asigurată de managerul de cache. Operaţiile specifice pe care le execută sunt: Fetch(x) şi Flush(x) şi ele sunt comunicate de celelalte nivele din sistemul bazei de date. În plus, dacă la un moment dat memoria volatilă s-a completat se poate lua iniţiativa proprie de-a executa operaţii Flush pentru a elibera memoria.

Managerul de recuperare asigură ca baza de date să conţină toate efectele tranzacţiilor realizate şi nici un efect al celor abandonate. Operaţiile pe care le înţelege sunt: Start, Commit, Abort, Read, Write, pe care le execută prin utilizarea de operaţii Fetch şi Flush. Managerul de recuperare este proiectat să reziste deranjamentelor în care întreaga memorie volatilă s-a pierdut. După restabilirea funcţionării sistemului el trebuie să

asigure consistența bazei de date, efectele tranzacțiilor realizate și nici un efect al celor abandonate sau a celor active la momentul deranjamentului. Trebuie înlăturat și efectul tranzacțiilor active deoarece prin pierderea memoriei principale s-au pierdut și stările interne ale tranzacțiilor. Singura informație disponibilă este cea existentă în memoria stabilă. Deoarece momentul deranjamentului nu se poate prevedea trebuie asigurată mutarea datelor din memoria volatilă în cea stabilă astfel încât să nu apară situații în care: 1) memoria stabilă să nu conțină o actualizare a unei tranzacții încheiate, 2) memoria stabilă să conțină valori scrise de o tranzacție neîncheiată, dar să nu conțină ultima valoare scrisă de o tranzacție realizată. Pentru aceasta managerul de recuperare va trebui să restricționeze inițiativele managerului de cache. Trebuie asigurat prin proiectare și o rezistență la deranjamente a memoriei stabile prin păstrarea de copii redundante ale datei în cel puțin două locuri distincte care nu se pot defecte simultan. Se observă că este utilă unirea celor două module într-unul singur - managerul datei.

Schedulerul este un program sau o colecție de programe care controlează paralelismul execuției tranzacțiilor, prin restricționarea ordinii în care managerul datelor execută operațiile Read, Write, Commit și Abort ale diferitelor tranzacții. Scopul său este ordonarea operațiilor astfel ca rezultatul execuției să fie serializabil și recuperabil. El va asigura evitarea cascadării abandonărilor. Acțiunile desfășurate de scheduler, după recepționarea operațiilor de la tranzacții sunt:

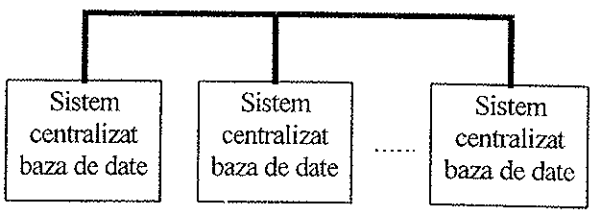
1. Execuția: Operația este transmisă spre managerul datelor, care la sfârșitul execuției va informa schedulerul. Dacă operația este Read, la răspuns se adaugă și valoarea citită pe care schedulerul o transmite tranzacției.
2. Reject: Se refuză execuția operației, caz în care se comunică tranzacției anularea execuției, ceea ce va determina abandonarea tranzacției. Instrucțiunea Abort poate fi produsă fie de tranzacție fie de managerul tranzacțiilor.
3. Delay: Se întârzie execuția operației prin plasarea într-o coadă de așteptare internă. Mai târziu această operație va fi extrasă și luată în considerare. În acest interval se pot primi și alte operații.

Managerul de tranzacții. Tranzacțiile interacționează cu sistemul bazei de date prin managerul tranzacțiilor. El recepționează operațiile emise de tranzacții și le transmite către scheduler. Funcție de tipul de control al paralelismului se pot executa și alte funcții. În cazul sistemelor distribuite, trebuie precizat locul în care se poate executa fiecare operație cerută de o tranzacție.

Modelul sistemului bază de date distribuit

Un sistem distribuit de baze de date este o colecție de locuri conectate printr-o rețea de comunicație. Două procese pot schimba mesaje oriunde sunt localizate. Fiecare loc este un sistem de bază de date centralizat care păstrează o porțiune a bazei de date. Fiecare tranzacție constă din unul sau mai multe procese care se execută în unul sau mai multe locuri. Tranzacțiile comunică operațiile pe care vor să le execute celui mai apropiat manager de tranzacții.

Dacă operația nu poate fi executată în locul respectiv va fi transmisă către schedulerul aflat în locul în care operația poate fi procesată. Comunicația se face cu mesaje prin rețea.



2.Schedulere fără blocare

2.1. Ordonarea după marca de timp

În ordonarea după marca de timp managerul tranzacțiilor atribuie o marcă unică $t_s(T_i)$ pentru fiecare tranzacție T_i. În continuare managerul MT va atașa marca de timp a tranzacției fiecărei operații propusă de tranzacție. Se poate vorbi de marcă de timp a unei operații $o_i(x)$, care este marca de timp a tranzacției care a solicitat execuția operației respective. Un scheduler cu ordonarea mărcilor de timp rezolvă operațiile conflictuale

în concordanță cu mărcile lor. El forțează respectarea următoarei reguli:

Regulă TO - Dacă $p_i(x)$ şi $q_j(x)$ sunt operații conflictuale atunci managerul de date va procesa operația $p_i(x)$ înaintea lui $q_j(x)$ dacă şi numai dacă $t_s(T_i) < t_s(T_j)$. Respectând această regulă execuțiile care se obțin sunt serializabile.

- **Varianta de bază**

Varianta de bază a ordonării după marca de timp realizează o implementare simplă şi agresivă a regulii enunțate. Operațiile sunt acceptate de la managerul de tranzacții şi sunt transmise imediat către managerul bazei de date în ordinea FIFO. Pentru a asigura că ordinea de execuție nu încalcă regula TO, schedulerul va elimina operațiile care vin prea târziu. O operație $p_i(x)$ este întârziată dacă a sosit după ce schedulerul a transmis deja o operație conflictuală $q_j(x)$ cu $t_s(T_j)< t_s(T_i)$. Dacă $p_i(x)$ este întârziată ea nu se poate executa şi deci T_i va fi abandonată. Când tranzacția este din nou inițiată, va primi o nouă marcă de la managerul MT.

Pentru a determina dacă o operație a sosit întârziat, schedulerul păstrează pentru fiecare articol x marcile maxime pentru operațiile Read şi Write cu parametru x care au fost transmise către managerul DM notate max.r.scheduled(x) şi max.w.scheduled(x). Când schedulerul primeşte $p_i(x)$ compară $t_s(T_i)$ cu max.q.scheduled(x) pentru toate operațiile de tip q cu care intră în conflict p. Dacă $t_s(T_i)$ este mai mic atunci $p_i(x)$ va fi rejectată, altfel va fi acceptată, şi dacă $t_s(T_i)>$ max.p.scheduled(x) va actualiza marca de timp. Între schedulerul şi managerul bazei de date trebuie să existe o comunicație care să garanteze că operațiile sunt procesate de acesta din urmă în ordinea în care sunt transmise. Chiar dacă schedulerul decide ca $p_i(x)$ poate fi executată, ea nu poate fi transmisă până când toate operațiile conflictuale $q_j(x)$ transmise înainte nu au fost efectiv executate. Pentru a forța aceasta, schedulerul va păstra pentru fiecare articol x numărul de operații Read şi Write care au fost transmise, dar nu au primit confirmarea de la managerul bazei de date, notate r.in.tranzit(x) şi w.in.tranzit(x). Pentru fiecare articol x schedulerul va păstra o coadă în care sunt depuse operațiile care se pot executa, dar aşteaptă confirmarea de la managerul bazei de date că operațiile conflictuale anterioare s-au executat.

- **Varianta strictă**

Reamintim că w.in.tranzit(x) semnifică numărul de operații w(x) pe care schedulerul le-a trimis către managerul bazei de date şi aceasta nu le-a validat. Cum două operații conflictuale nu pot fi în tranzit odată, rezultă că w.in.tranzit(x) este fie 0, fie 1. Varianta strictă a schedulerului lucrează la fel ca şi varianta de bază cu excepția faptului că trecerea lui w.in.tranzit(x) pe 0 nu se face la primirea confirmării execuției lui $w_i(x)$ de la managerul bazei de date. Se aşteaptă până când se primeşte confirmarea execuției lui a_i sau c_i şi apoi se setează w.in.tranzit(x) pe 0 pentru fiecare x pentru care a existat o operație de scriere $w_i(x)$ din partea tranziției T_i. Aceasta întârzie toate operațiile $r_i(x)$ şi $w_i(x)$ pentru care $t_s(T_j)<t_s(T_i)$ până când tranziția T_i fie este abandonată fie se realizează, ceea ce conduce la o execuție strictă. De notat că aşteaptă după tranziția T_i doar acele tranziții T_j pentru care $t_s(T_j)> t_s(T_i)$, situație care nu poate conduce la deadlock.

- **Gestionarea mărcilor de timp**

Se presupune că mărcile de timp se păstrează într-un tabel în care fiecare intrare este de forma (x, max.r.scheduled(x), max.w.scheduled(x)). În cazul în care datele sunt mici, informațiile asociate privind mărcile de timp pot ocupa un spațiu comparabil cu baza de date. Pentru a rezolva această problemă se poate utiliza următoarea observație. Se presupune că managerul MT foloseşte pentru generarea mărcilor un ceas real, iar tranzacțiile se execută într-un interval relativ scurt de timp. În aceste condiții, la un moment t, schedulerul va fi aproape sigur că nu va mai primi noi operații cu marca de timp mai mică decât t-δ, unde δ este suficient de mare comparativ cu timpul de execuție a tranzacției. Singurul motiv pentru care schedulerul are nevoie de marca de-tip din max.r.scheduled(x) şi max.w.scheduled(x), fie t_{s_r} şi t_{s_w}, este pentru a rejecta operațiile Read şi Write care au marca mai

mică decât t_{s_r} şi t_{s_w}. Odată ce t_{s_r} şi t_{s_w} sunt mai mici ca t-δ ele devin prea mici pentru scheduler, nemaifiind probabilă apariţia operaţiilor cu o marcă mai mică decât t_{s_r} şi t_{s_w}.

Folosind această observaţie periodic se pot elimina din tabelul cu mărci intrările care devin inutile. Fiecare operaţie Purge foloseşte un t_{smin}, echivalentul lui t-δ, care va elimina fiecare intrare cu o marcă de timp mai mică decât t_{smin}, iar în plus va eticheta tabelul cu t_{smin}, indicând faptul că operaţia Purge a avut loc cu acea valoare.

În acest caz trebuie modificat modul în care se determină dacă o operaţie $o_i(x)$ este întârziată. Prima dată se caută intrări pentru x în tabel. Dacă se găseşte una se compară $t_s(T_i)$ cu max.r.scheduled(x) sau max.w.scheduled(x), în maniera cunoscută. Dacă nu există intrări, atunci trebuie compa-rat cu t_{smin}. Dacă $t_s(T_i) \geq t_{smin}$, atunci eventualele intrări eliminate erau fără importanţă, iar dacă $t_s(T_i) < t_{smin}$ atunci s-a eliminat o intrare a unei operaţii cu care se intră în conflict. Pentru siguranţă, schedulerul va trebui să elimine $o_i(x)$.

- **Varianta distribuită**

Fiecare loc are un scheduler care ordonează operaţiile ce accesează datele din propriul loc. Decizia de execuţie, întârziere sau anulare a unei operaţii $o_i(x)$ depinde doar de alte operaţiile care accesează articolul x. Fiecare scheduler poate menţine toate informaţiile legate de operaţiile de accesare a datelor pe care le gestionează putând lua decizia independent de alte schedulere, proprietate care conduce la obsenţa necesităţii comunicaţiei.

- **Varianta conservativă**

Dacă se recepţionează operaţii într-o ordine foarte diferită de ordinea normală a mărcilor de timp se poate ajunge la situaţia în care se elimină prea multe operaţii, ce determină prea multe tranzacţii să abandoneze. Problema se poate rezolva prin proiectarea unui scheduler conservativ. Pentru aceasta o primă cerere este de a întârzia fiecare operaţie care se primeşte o perioadă de timp ceea ce creşte şansa ca operaţii conflictuale să poată sosi la timp pentru a fi luate în discuţie. Desigur, întârzierea execuţiei prea mult creează probleme prin încetinirea procesării tranzacţiilor, fiind necesară păstrarea unui echilibru între întârzierea pentru a evita prea multe eliminări de operaţii şi încetinirea tranzacţiilor.

Un scheduler conservativ se poate realiza prin utilizarea unei cozi de aşteptare ce va conţine operaţiile recepţionate de la managerii de tranzacţii şi care nu au fost luate în considerare. Operaţiile din această coadă sunt păstrate în ordinea mărcilor de timp, operaţiile cu mărci mai mici fiind în vârful cozii. Operaţiile cu aceeaşi marcă de timp se depun în coadă în ordinea în care au sosit, cele mai recente fiind mai aproape de vârf. La primirea unei operaţii $p_i(x)$ de la managerul de tranzacţii, schedulerul o inserează în locul corespunzător din coadă prin menţinerea ordinii stabilite anterior. Schedulerul va verifica apoi dacă prima operaţie din coada de aşteptare este pregătită pentru a fi transmisă către managerul bazei de date. O operaţie $q_i(x)$ este pregătită dacă:

1. coada de aşteptare conţine cel puţin o operaţie de la fiecare manager de tranzacţii;
2. toate operaţiile ce intră în conflict cu $q_i(x)$ ce au fost transmise anterior către managerul bazei de date au fost validate.

Dacă prima operaţie este pregătită, schedulerul o elimină din coada de aşteptare şi o va transmite către managerul MD, operaţie care se repetă până când se regăseşte o operaţie care nu este pregătită.

Pentru a şti dacă există operaţii de la toţi managerii de tranzacţii este necesară memorarea numărului de operaţii în contori asociaţi tranzacţiilor notaţi op.counter(x). Pentru a realiza decrementarea în coada de aşteptare operaţiile se vor păstra sub formă de perechi (contor, operaţie).

O problemă cu acest scheduler poate apărea dacă un manager de tranzacţii nu mai transmite operaţii pentru o perioadă, ceea ce ar bloca execuţia. Pentru a evita această situaţie, dacă un manager MT nu are de transmis operaţii, el va transmite operaţia Null, care are la rândul ei asociată o marcă de timp. Aceasta va fi procesată de scheduler în mod normal, dar în final ea nu va mai fi transmisă către managerul bazei de date.

O a doua problemă care apare este eceea că tipul conservativ în acest caz este mult prea

restrictiv, execuţiile rezultate putând fi seriale. Pentru a îmbunătăţi concurenţa se pot folosi clase de tranzacţii. O clasă de tranzacţii este definită de o mulţime de citire şi o mulţime de scriere. O tranzacţie este membră a unei clase dacă mulţimile de citire şi de scriere sunt submulţimi ale clasei. Schedulerul nu va mai testa primirea de operaţii de la toţi managerii MT, ci este necesar să primească doar de la aceia care sunt în aceeaşi clasă de tranzacţii. Fiecare tranzacţie va trebui să-şi declare la început mulţimea de citire şi cea de scriere. Definirea claselor şi asocierea cu managerii tranzacţiilor trebuie să rămână statică pe durata operării sistemului daza de date.

2.2. Testarea grafului de serializare

Un scheduler fără blocare păstrează un graf de serializare care reprezintă execuţia pe care o controlează. În acest graf vor fi incluse noduri pentru toate tranzacţiile active şi periodic pot fi eliminate noduri care corespund tranzacţiilor realizate.

- **Varianta de bază**

În acest caz, când schedulerul primeşte o operaţie $p_i(x)$, mai întâi adaugă un nod pentru T_i în graf, dacă acesta nu există. Adaugă apoi un arc de la T_j la T_i pentru fiecare operaţie $q_j(x)$ anterioară care intră în conflict cu $p_i(x)$. Apar două cazuri:

1. Graful rezultat conţine un ciclu şi deci dacă $p_i(x)$ ar fi executată acum ar rezulta o execuţie neserializabilă, caz în care $p_i(x)$ trebuie rejectată. Pentru aceasta se transmite a_i către managerul bazei de date, iar la primirea confirmării se elimină toate cele care au legătură cu T_i. Eliminând acest nod, graful redevine serializabil.
2. Graful rezultat nu conţine nici un ciclu, $p_i(x)$ este acceptat şi poate fi transmis imediat dacă s-au primit toate confirmările de execuţie pentru operaţiile conflictuale transmise spre MD. În caz contrar, se va întârzia $p_i(x)$ până se primesc aceste confirmări. Implementarea se face tot printr-o coadă de aşteptare de tip FIFO, în care sunt păstrate operaţiile, precum şi doi contori r.in.transit(x) şi w.in.transit(x).

Câteva consideraţii trebuie făcute în ceea ce priveşte momentul în care schedulerul poate renunţa la informaţiile colectate despre o tranzacţie. Pentru a detecta conflictele trebuie păstrate mulţimile de citire şi de scriere a fiecei tranzacţii, ceea ce consumă un spaţiu important. Schedulerul poate şterge informaţii despre tranzacţii încheiate dacă şi numai dacă T_i nu va ma fi implicat în nici un ciclu. Pentru ca un nod să participe la un ciclu trebuie să existe cel puţin un arc ce intră în nod şi unul care iese din nod. Odată tranzacţia încheiată nu mai pot apărea noi arce direcţionate spre nod. O regulă sigură de eliminare a nodurilor este aceea că informaţiile despre tranzacţii pot fi eliberate imediat ce tranzacţia s-a terminat şi nodul este sursă în graf.

- **Varianta conservativă**

Un scheduler cu testarea grafului de serializabilitate nu va elimina operaţii, dar le va întârzia. Pentru aceasta trebuie să se realizeze o predeclarare a mulţimilor de scriere şi de citire notate r.set(T_i) şi w.set(T_i), care se poate face în operaţia Start. Îndată ce schedulerul primeşte operaţia Start de la T_i, va salva cele două mulţimi r.set(T_i) şi w.set(T_i) va crea un nou nod pentru T_i şi va adăuga arce $T_j \rightarrow T_i$ pentru fiecare tranzacţie T_j din graf pentru care p.set(T_i)$\cap$q.set(T_j) $\neq(\varnothing)$ pentru toate perechile de operaţii conflictuale p şi q.

Pentru fiecare articol x, schedulerul păstrează coada de aşteptare a operaţiilor care acce-seză x. Operaţiile conflictuale dintr-o coadă de aşteptare sunt păstrate în ordinea arcelor din graf. Altfel, dacă există $T_j \rightarrow T_i$, $q_j(x)$ va fi mai aproape de începutul cozii decât $p_i(x)$, iar ordinea operaţiilor neconflictuale nu are importanţă. O operaţie din coada de aştepta-re este transmisă către managerul bazei de date dacă este pregătită:

1. toate operaţiile ce intră în conflict cu $p_i(x)$ au fost transmise anterior şi au fost confirmate de managerul bazei de date;
2. pentru fiecare T_j care precede direct T_i în graf şi pentru fiecare operaţie a cărui tip q intră în conflict cu p sau x nu aparţine lui q.set(T_j) sau $q_j(x)$ deja a fost primită de scheduler (există în q.scheduled(T_j)).

- **Varianta strictă**

Ambele vriante pot fi modificate pentru a produce doar istorice stricte. Schedulerul setează w.in.tranzit(x) pe 1 când transmite $w_i(x)$ la managerul bazei de date. Decrementarea la 0 nu se va realiza când se primeşte confirmarea lui $w_i(x)$, ci atunci când se confirmă execuţia lui a_i sau c_i.

Pentru a evita abandonările în cascadă este necesară doar asigurarea că înainte de execuţia lui $r_i(x)$, tranzacţia de la care T_i îl va citi pe x s-a încheiat. Pentru aceasta de fiecare dată când schedulerul primeşte o confirmare de la o operaţie Commit c_j va marca nodul T_j din graf. Se presupune că schedulerul primeşte $r_i(x)$ de la managerul TM. Fie T_j o tranzacţie pentru care:

1. x există în w.scheduled(T_j) şi
2. pentru oricare $T_k \neq T_j$ pentru care x există în w.scheduled(T_k) există arcul $T_k \rightarrow T_j$ în graf. În acest caz schedulerul poate transmite $r_i(x)$ doar dacă T_j este marcat ca "realizată" sau nu există T_j.

3. Garanţi

Abordarea anterioară a presupus că de fiecare dată când primea o operaţie, schedulerul decidea dacă să o accepte, să o respingă sau să o întârzie. O abordare diferită este aceea de a accepta imediat operaţia primită şi doar din când în când să se verifice ce s-a efectuat. Dacă nu a apărut nici un conflict, se continuă în mod asemănător, iar dacă se determină că s-au efectuat operaţii conflictuale într-o ordine necorespunzătoare se abandonează tranzacţiile. Când se primeşte o operaţie Commit de la o tranzacţie T_i se verifică dacă execuţia acestuia conduce la o execuţie serializabilă. Dacă nu, ea este eliminată şi deci T_i este forţată să abandoneze. Un astfel de scheduler se numeşte *garant*, iar procesul de testare a posibilităţii de execuţie a lui Commit se numeşte *atestare*. Garanţii mai sunt denumiţi şi *scheduleri optimişti* deoarece prelucrează operaţiile agresiv, cu presupunerea că nimic rău nu se înpâmplă.

- **Garant cu blocare în două faze**

Când un astfel de garant primeşte o operaţie, notează articolul accesat de operaţie şi o transmite imediat către managerul bazei de date. Când recepţionează Commit verifică dacă există o operaţie $p_i(x)$ din T_i care să intre în conflict cu operaţia $q_i(x)$ din alte tranzacţii active T_i. Dacă aceasta se întâmplă, se elimină c_i şi abandonează T_i, altfel va garanta T_i prin transmiterea lui c_i către managerul MD, permiţând ca tranzacţia T_i să se încheie cu succes. Un astfel de garant foloseşte diferite structuri de date: o mulţime conţinând numele tranzacţiilor active şi două mulţimi r.scheduled(T_i) şi w.scheduled(T_i) pentru fiecare tranzacţie activă T_i, care conţine articolele citite şi scrise de tranzacţie T_i până atunci. Când se primeşte o operaţie $r_i(x)$ (sau $w_i(x)$) se adaugă x la r.scheduled(T_i) (sau la w.scheduled(T_i)). Când se primeşte c_i, T_i a terminat execuţia şi deci r.scheduled(T_i) şi w.scheduled(T_i) conţin mulţimea de citire şi de scriere. Testul de conflict se poate realiza prin examinarea intersecţiilor mulţimilor r.scheduled(T_i) şi w.scheduled(T_i). Pentru a procesa c_i garantul verifică fiecare tranzacţie T_j activă pentru a determina dacă r.scheduled(T_i)$\cap$w.scheduled(T_j), w.scheduled(T_i) $\cap$ r.scheduled(T_j) sau w.scheduled(T_i) $\cap$ w.scheduled(T_j) este diferită de mulţimea vidă pentru a elimina c_j.

Denumirea de garant cu blocare în două faze nu provine atât de la utilizarea lock-urilor, care după cum se vede lipsesc, ci datorită comportamentului său de a elimina una din operaţiile care produc un conflict.

Algoritmul prezentat asigură serializabilitatea. Dacă se doreşte ca execuţia să fie strictă este nevoie ca atunci când este abandonată o tranzacţie T_i să fie abandonate şi alte tranzacţii active T_j, pentru care w.scheduled(T_i) $\cap$ r.scheduled(T_j) nu este vidă.

- **Garant cu testarea grafului de serializare**

În acest caz se păstrează dinamic un graf de serializare a execuţiei realizate până atunci. De fiecare dată când se primeşte o operaţie $p_i(x)$ se adaugă un arc $T_j \rightarrow T_i$ în graf pentru fiecare tranzacţie T_j pentru care garantul a trimis către managerul bazei de date o operaţie $q_i(x)$ ce intră în conflict cu $p_i(x)$, după care este transmisă $p_i(x)$. Hands-

hakingul este în continuare necesar între garant şi managerul bazei de date în procesarea operaţiilor conflictuale. La recepţia lui c_i se verifică dacă tranzacţia T_i este implicată într-un ciclu din graful de serializare. Dacă da, este eliminată operaţia c_i şi T_i este abandonată. O variantă strictă a acestui garant se obţine prin adăugarea regulii că dacă a tranzacţie T_i este abandonată, trebuie abandonate şi tranzacţiile active T_j la care, pentru articole $x \in w.scheduled(T_i) \cap r.scheduled(T_j)$, arcul $T_i \rightarrow T_j$ este în graful de serializare şi pentru fiecare T_k pentru care $T_i \rightarrow T_k$ şi $T_k \rightarrow T_j$ sunt de asemenea în graf $x \notin w.scheduled(T_k)$.

- **Garant cu ordonarea mărcilor de timp**

Operaţiile Read şi Write sunt transmise fără nici o întârziere de garant cu excepţia motivelor ce ţin de interblocarea dintre garant şi managerul bazei de date. Când este recepţionat Commit se verifică dacă operaţiile conflictuale a lui T_i sunt în ordinea mărcilor de timp. Dacă aceasta nu se întâmplă tranzacţia T_i este abandonată.

- **Garant distribuit**

Un garant distribuit constă dintr-o colecţie de procese de atestare, câte unul pentru fiecare loc. Se presupune că garantul dintr-un loc este responsabil cu accesul la datele păstrate în acel loc. Fiecare garant transmite operaţii către managerul bazei de date local, independent de ceilalţi garanţi. Pentru a valida o tranzacţie, o decizie care se ia trebuie să implice toţi garanţii care au primit operaţii de la acea tranzacţie. În varianta de testare a grafului de serializare, garanţii trebuie să schimbe grafurile locale pentru a putea testa în graful global dacă nu a apărut un ciclu ce implică tranzacţia care urmează a fi validată. Dacă aceasta nu se întâmplă, tranzacţia este garantată, în caz contrar este abandonată.

În varianta cu blocare în două faze şi în varianta cu ordonarea mărcilor de timp fiecare garant decide local dacă validează tranzacţia, decizie bazată pe informaţiile pe care le deţine. O decizie globală trebuie luată prin consens; dacă toate deciziile locale validează tranzacţia, decizia globală va fi de garantare a tranzacţiei. Dacă toate deciziile locale sunt de abandonare, decizia globală este de abandonare. Soarta tranzacţiei este decisă doar după decizia globală, un garant local nu poate valida tranzacţia doar pe baza informaţiei locale.

Acest mod de lucru poate fi obţinut folosind următorul protocol de comunicaţie între managerul de tranzacţii care supervizează T_i şi garanţii care au procesat operaţiile lui T_i. Managerul MT va transmite Commit către toţi garanţii care au participat la execuţia tranzacţiei T_i. Când fiecare garant primeşte c_i va lua o decizie locală numită *vot* pe care o transmite managerului de tranzacţii. După ce a primit toate voturile de la participanţi, managerul tranzacţiei ia o decizie globală pe care o va transmite garanţilor implicaţi, care o vor executa de îndată ce o recepţionează. În acest caz, pentru un garant care a votat **da**, urmează o perioadă de incertitudine privind viitorul tranzacţiei şi anume perioada între votul său şi primirea deciziei globale. Pentru garantul care a votat abandonarea tranzacţiei, această incertitudine nu există. El ştie că tranzacţia va fi, eventual, abandonată de toţi garanţii.

Bibliografie

C.J. Date *An Introduction to Database Systems* Addison Wesley 1995

E. F. Codd *The Relational Model for Database Management* Addison Wesley 1990

Grady Booch *Object Oriented Design with Application* Benjamin/Cummings 1991

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides *Design Patterns: Elements of Reusibale Object Oriented Software* Addison Wesley 1995

Cannan S., Otten G. *SQL - The Standard Handbook* McGraw-Hill 1992

C.J. Date, H Darwen *A Guide to the SQL Standard* Addison Wesley 1992

Bruce Eckel *C++ Inside & Out* McGraw-Hill 1993

J.K. Groff, N Weinberg *LAN Times Guide to SQL* McGraw-Hill 1994

James A. Freeman, David M. Skapura *Neural networks: algorithms, applications and programming techniques* Addison Wesley 1991