ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: INFORMATYKA z. 34 Nr kol. 1381

Mirosław CHŁOPEK, Katarzyna HARĘŻLAK, Henryk JOSIŃSKI
Politechnika Śląska, Instytut Informatyki

STEROWANIE WSPÓŁBIEŻNYM DOSTĘPEM DO DANYCH PODCZAS REALIZACJI TRANSAKCJI ROZPROSZONYCH - ROZWÓJ EKSPERYMENTALNEGO SYSTEMU ZARZĄDZANIA ROZPROSZONĄ BAZĄ DANYCH

Streszczenie. Współpraca z bazami danych znajdującymi się w różnych węzłach sieci wymaga od systemu zarządzania rozproszoną bazą danych sprawowania kontroli nad współbieżnym dostępem do danych. W artykule przedstawiono mechanizmy, w które powinien zostać wyposażony taki system w celu zapewnienia poprawnej realizacji transakcji rozproszonej.

CONCURRENCY CONTROL DURING EXECUTION OF GLOBAL TRANSACTIONS - DEVELOPMENT OF THE EXPERIMENTAL DISTRIBUTED DATABASE MANAGEMENT SYSTEM

Summary. Cooperation between the distributed database management system (DDBMS) and local databases located on different network nodes requires implementation of concurrency control mechanism. The article presents the DDBMS mechanisms proposed for ensuring the correct execution of global transactions and implemented in the experimental DDBMS.

1. Wstęp

Rozwój sieci komputerowych otwiera coraz większe możliwości dostępu do różnorodnych informacji, a w szczególności do tych, które przechowywane są w bazach danych znajdujących się w różnych węzłach sieci. Uzyskanie informacji z pojedynczego odległego węzła wymaga jedynie zdalnego połączenia się z nim i posiadania odpowiednich praw dostępu. Nie pozwala jednak na równoczesne operowanie na danych z kilku wierzchołków sieci lub bardzo je utrudnia. Problem ten rozwiązywany jest przy użyciu systemów zarządzania rozproszonymi bazami danych (SZRBD), które ułatwiają dostęp do danych, oraz zwalniają użytkownika z konieczności poznawania schematów i lokalizacji baz danych wchodzących w skład bazy rozproszonej. Użytkownik powinien widzieć wszystkie udostępnione mu dane tak, jak gdyby były one scentralizowane w lokalnej bazie danych. Systemy zarządzania rozproszonymi bazami danych stają się odpowiedzialne za nawiązanie kontaktu z odpowiednimi bazami danych, zapewnienie bezpieczeństwa i spójności zawartych w nich informacji oraz efektywne ich przetwarzanie. Realizacja tych zadań odbywa się poprzez przygotowanie między innymi:

- odpowiedniego mechanizmu synchronizacji współbieżnego dostępu do danych,
- algorytmu wykrywania i usuwania zakleszczeń albo zapobiegania im,
- w przypadku posiadania danych rozproszonych lub powielonych (replikacji) – modułu koordynującego manipulowanie nimi.

W niniejszym artykule omówione zostaną wyżej wspomniane mechanizmy wraz ze sposobem ich implementacji w eksperymentalnym systemie zarządzania rozproszoną bazą danych.

2. Analiza wybranych mechanizmów współbieżnego dostępu do danych w systemie zarządzania rozproszoną bazą danych

2.1. Lokalna aktualizacja danych

Włączenie lokalnej bazy danych do bazy rozproszonej zwiększa zasięg dostępności danych, nie powinno mieć jednak wpływu na jej funkcjonowanie jako scentralizowanej bazy danych obsługiwanej przez lokalny serwer. Jednak, z drugiej strony, manipulowanie danymi w lokalnej bazie danych powinno znaleźć odzwierciedlenie w rozproszonej bazie danych tak, by dane widziane z pozycji globalnego schematu bazy danych były poprawne i integralne.

Stosowanie replikacji przyspiesza dostęp do danych, zmniejsza natężenie ruchu w sieci oraz zwiększa niezawodność systemu. Układ taki wymaga jednak stworzenia mechanizmu

zapewniającego przeniesienie lokalnych zmian dokonywanych na danych na wszystkie istniejące kopie tak, by ich użytkownicy, niezależnie od miejsca pobytu, uzyskiwali na to samo zapytanie taką samą odpowiedź. Utrzymywanie kilku kopii tej samej tabeli niesie ze sobą ponadto niebezpieczeństwo tak zwanej *kolizji*, kiedy dwie fizyczne kopie danych aktualizowane są w tym samym przedziale czasu przez różnych użytkowników lokalnych. Niezbędna jest w tym przypadku współpraca systemów zarządzania lokalną i rozproszoną bazą danych w celu wykrycia i rozwiązania tego konfliktu. Wymaga to, ze strony administratora lokalnej bazy danych, zaangażowania mechanizmów aktywnej bazy danych: reguł, zdarzeń oraz procedur bazy danych [16]. Natomiast system zarządzania rozproszoną bazą danych powinien zawierać moduł zwany *replikatorem*, który, po uzyskaniu informacji od lokalnego systemu, przeniesie zmiany do odpowiednich węzłów [16].

Istnieją dwie koncepcje rozwiązania problemu aktualizowania zreplikowanych danych. Pierwsza z nich oparta jest na podejściu *Master/Slave*, w którym możliwość zmiany, usuwania lub dodawania rekordów przysługuje użytkownikom jednej kopii (Master), pozostałe natomiast służą tylko do odczytu [15]. Ogranicza to autonomiczność lokalnej bazy danych, w zamian za co problem kolizji przestaje istnieć. Każda zmiana w kopii Master, za pomocą mechanizmów aktywnej bazy danych, przekazywana jest do replikatora, który, w sposób asynchroniczny, uaktualnia pozostałe kopie. Inaczej wygląda sytuacja w przypadku systemów opartych na modelu *Peer-To-Peer* ("równy z równym") [15]. Prawa modyfikacji danych posiadają użytkownicy wszystkich kopii replikowanej tabeli. Propagacja zmian odbywa się, podobnie jak w pierwszej koncepcji, poprzez moduł replikatora, który dodatkowo wykrywa zaistniałe konflikty. Usuwanie ich realizowane jest za pomocą wbudowanych algorytmów rozwiązywania kolizji, zaproponowanych przez projektanta systemu na podstawie wiedzy o jego przeznaczeniu [16]. Istnieją systemy, w których użytkownik sam przygotowuje odpowiedni algorytm [15, 16].

2.2. Kontrola współbieżnego dostępu do danych

Utrzymywanie spójności bazy danych wymaga od SZRBD umiejętności synchronizacji współbieżnego dostępu do danych. Oznacza to, że żaden z użytkowników bazy danych nie powinien odczytywać lub aktualizować rekordów, których zawartość jest właśnie zmieniana przez innego użytkownika. W konsekwencji zezwolenia na taką sytuację może dojść do udostępnienia nieaktualnych danych. Niezbędne jest więc użycie odpowiednich środków kontroli współbieżnego dostępu do danych. Wśród klasycznych metod synchronizacji można wyodrębnić [4]:

- grupę metod pesymistycznych – opartych na blokadzie dostępu do danych,

- grupę metod optymistycznych - polegających na niezależnym wykonaniu transakcji i późniejszej detekcji ewentualnego konfliktu.

Niepożądanymi zjawiskami, mogącymi wystąpić przy próbie współbieżnego dostępu do danych, są: *zakleszczenie* (impas) oraz *uwięzienie* (zagłodzenie) [4].

Zakleszczenie

Sytuacja, w której każdy element zbioru S dwóch lub większej liczby transakcji czeka na blokadę jednostki właśnie zablokowanej przez pewną inną transakcję ze zbioru S, jest nazywana zakleszczeniem [4]. Sytuacji tej można zapobiegać stosując strategie blokowania nie dopuszczające do niego. Jedną z nich jest stosowanie konserwatywnego protokołu 2PL (ang. Two-Phase Locking) [4]. Polega on na tym, że każda transakcja zgłasza od razu wszystkie swoje żądania blokowania. System albo realizuje je wszystkie, albo, w razie konfliktu z inną transakcją, nie realizuje żadnego i powoduje, że proces czeka. Alternatywnym rozwiązaniem jest wykorzystywanie metod pozwalających zlikwidować zakleszczenie przez wycofanie jednej z transakcji, co pozwala na kontynuowanie pozostałych. Sprawdzenia wystąpienia zakleszczenia można dokonać konstruując graf oczekiwania (ang. Wait-For-Graph). Jego wierzchołkami są transakcje, a krawędzie $T_i \rightarrow T_j$ oznaczają, że transakcja T_i czeka na zablokowanie jednostki, którą blokuje T_j. Każdy cykl w grafie sygnalizuje wystąpienie impasu [9].

Uwięzienie

Sytuację, w której znajduje się transakcja czekająca na zablokowanie jednostki, która jest ciągle przydzielana innym transakcjom, nazywa się uwięzieniem. Prosta metoda jego uniknięcia polega na tym, że system przyznający blokady umieszcza wszystkie żądania, które nie zostały natychmiast spełnione, w kolejce o organizacji FIFO, i z chwilą odblokowania jednostki umożliwia jej blokadę przez tę transakcję, spośród wszystkich czekających, która zażądała jej najwcześniej [4].

3. Elementy systemu eksperymentalnego i jego środowisko pracy

Ideą autorów opracowania był rozwój eksperymentalnego systemu zarządzania rozproszoną bazą danych, opisanego w pracach [12, 13], w oparciu o model wieloprocesorowego komputera z wirtualnie współdzieloną pamięcią w środowisku programowania rozproszonego i równoległego Paradise [14, 17]. Wirtualnie współdzielona pamięć jest w tym ujęciu nazywana *przestrzenią krotek*.

Do zalet środowiska Paradise należy zaliczyć przede wszystkim:

- możliwość współistnienia wielu przestrzeni krotek,

- tworzenie kopii przestrzeni krotek w plikach dyskowych,
- atrybut trwałości przestrzeni krotek, dzięki któremu istnieje ona nadal mimo zakończenia aplikacji, która ją stworzyła,
- definiowanie transakcji obejmujących operacje modyfikujące zawartość przestrzeni krotek, np. wprowadzenie krotki, odczyt niszczący jej zawartości, usunięcie wszystkich krotek,
- operację nieniszczącego odczytu zawartości krotki.

4. Rozproszona architektura procesora aplikacji

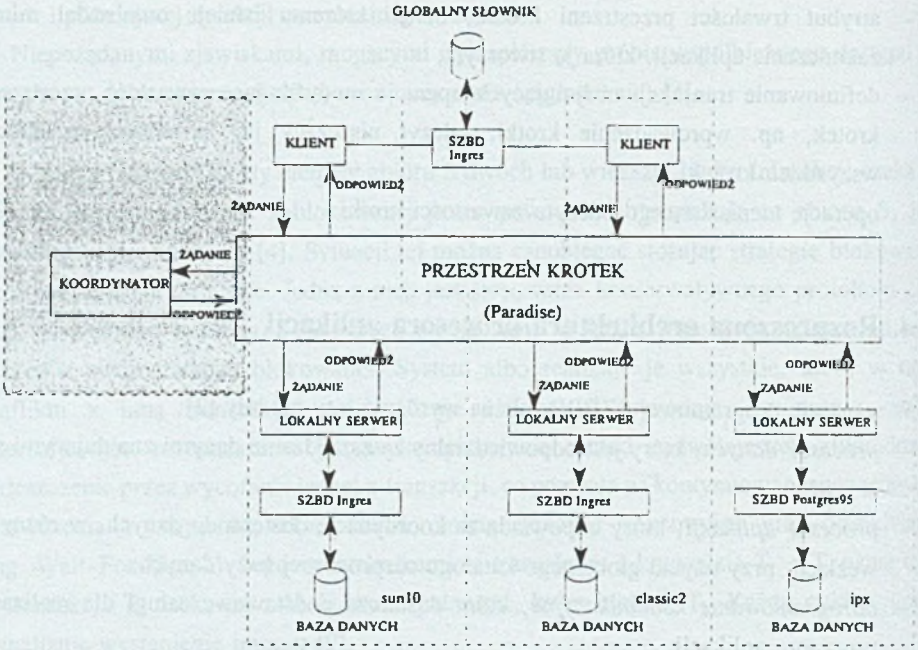
W warstwie programowej SZRBD można wyróżnić trzy moduły [2]:

- *procesor danych*, który jest odpowiedzialny za zarządzanie danymi znajdującymi się w węźle;
- *procesor aplikacji*, który odpowiada za koordynację dostępu do danych, w różnych węzłach, przy użyciu globalnego katalogu rozproszonej bazy danych;
- *oprogramowanie komunikacyjne*, które dostarcza podstawowe usługi dla realizacji procesora aplikacji.

Procesor aplikacji, w pierwszej wersji eksperymentalnego systemu zarządzania rozproszoną bazą danych, składał się z modułów: klienta, koordynatora i lokalnego serwera (rys. 1). Jego dokładny opis został zawarty w pracy [13]. Badania nad efektywnością i bezpieczeństwem pracy tego elementu wykazały, że przy intensywnym wykorzystaniu systemu koordynator może okazać się "wąskim gardłem" opóźniającym pracę całego systemu. Natomiast awaria tego modułu rodzi konieczność powołania koordynatora zastępczego, przejmującego zadania swego poprzednika. W związku z tym zaproponowano modyfikację architektury procesora aplikacji poprzez wyeliminowanie modułu koordynatora, rozdzielając jego funkcje pomiędzy pozostałe elementy systemu. Klient, poza zadaniami wymienionymi w pracy [13], jest więc również odpowiedzialny za:

- zarządzanie swoją aktualnie wykonywaną transakcją rozproszoną (rozpoczynanie, monitorowanie, wypełnianie bądź wycofywanie),
- kontrolę współbieżnego dostępu do danych z wykorzystaniem mechanizmu blokad logicznych we współpracy z lokalnym serwerem.

Dodatkowo wyeliminowanie modułu koordynatora powoduje, że to moduł klienta rozsyła zapytania pomiędzy lokalne serwery.



Rys. 1. Struktura systemu
Fig. 1. The structure of the system

W systemie może równocześnie funkcjonować dowolna liczba programów klienta oraz lokalne serwery w liczbie odpowiadającej liczbie procesorów danych, które komunikują się ze sobą poprzez wirtualnie współdzieloną pamięć – przestrzeń krotek.

4.1. Ogólny algorytm realizacji transakcji rozproszonej w wersji *klient–lokalny serwer*

W procesie realizacji transakcji rozproszonej, złożonej z zadanego przez użytkownika ciągu instrukcji, można wyróżnić następujące fazy (częściowo nakładające się na siebie):

(1) *Klient*:

- dekompozycja zapytania na ciąg podzapytań operujących na jednostkach fizycznych;
- założenie blokad logicznych pozwalających na wycofanie transakcji, której wykonanie jest w danej chwili niemożliwe, i dających możliwość wykrywania zakleszczeń;

- wysłanie krotek z podzapytaniami do lokalnych serwerów;

(2) *Lokalne serwery, procesory danych*:
- wykonanie operacji na lokalnych bazach danych za pomocą mechanizmów procesora danych;

(3) *Lokalne serwery*:
- umieszczenie krotek z danymi w przestrzeni krotek oraz wysłanie potwierdzenia wykonania podzapytania do klienta;

(4) *Klient, lokalne serwery*:
- realizacja protokołu wypełnienia transakcji rozproszonej;

(5) *Lokalne serwery*:
- zdjęcie blokad logicznych;

(6) *Klient*:
- pobranie krotek z danymi oraz prezentacja wyników.

5. Implementacja badanych cech oraz mechanizmów systemu zarządzania rozproszoną bazą danych

Charakterystykę najistotniejszych cech SZRBD oraz sposób ich implementacji w wersji *klient-koordynator-lokalny serwer* zamieszczono w pracy [13]. Natomiast w rozdziale 2 niniejszego opracowania skupiono uwagę na tych cechach SZRBD, których implementacja stanowiła cel prac na obecnym etapie rozwoju systemu eksperymentalnego. Rozwiązania opisane poniżej dotyczą wersji *klient-lokalny serwer*.

5.1. Lokalna aktualizacja danych

Badania nad możliwością wykorzystywania wirtualnie współdzielonej pamięci jako platformy do budowy SZRBD wymagają włączenia w ich zakres zarządzania replikacjami. Jak wspomniano w rozdziale 2, z zarządzaniem tym wiążą się dwa podstawowe problemy: rozpowszechnianie zmian dokonywanych na danych oraz kolizje wynikające z równoczesnej modyfikacji różnych kopii przez niezależnych użytkowników.

W celu rozwiązania tych zagadnień system eksperymentalny wzbogacony został o moduł replikatora, który odpowiedzialny jest za koordynację wszystkich zadań operujących na powielonych danych. Ponadto w węźle zawierającym kopię tabeli uprawnioną do modyfikacji pracuje *lokalny klient*, który przejmuje wszystkie instrukcje do niej kierowane. Komunikacja tego modułu z replikatorem odbywa się za pomocą mechanizmów aktywnych baz danych: wyzwalaczy (triggerów), procedur bazy danych oraz zdarzeń [18]. Trigger zostaje

uaktywniony w momencie wystąpienia jednej z operacji: UPDATE, INSERT, DELETE. Jego zadaniem jest uruchomienie procedury bazy danych, która, za pomocą mechanizmu zdarzeń lub poprzez zaktualizowanie przestrzeni krotek, powiadamia replikator o lokalnie dokonanej zmianie danych. Zarządca replikacji umieszcza wszystkie operacje w kolejce, przekazuje je do odpowiednich serwerów i zbiera nadesłane odpowiedzi. Jeśli liczba informacji odebranych od lokalnych serwerów równa jest liczbie istniejących kopii tabeli, to następuje usunięcie zadania z kolejki. Ogniwem łączącym replikator z lokalnymi serwerami jest wirtualnie współdzielona pamięć, do której replikator składa krotki z operacjami, a lokalny serwer - wynik ich wykonania.

Replikator, jak każdy element systemu narażony jest na wystąpienie awarii, co może spowodować utratę integralności danych. Wynika stąd konieczność prowadzenia dziennika, w którym zapisywane są, na bieżąco, informacje o stanie prowadzonych transakcji.

W eksperymentalnym systemie zarządzania rozproszoną bazą danych moduł koordynujący pracę replikacji przygotowano w dwóch wersjach. Pierwsza z nich - prostsza - oparta została na koncepcji Master/Slave. W kolejnej wersji zastosowano podejście Peer-To-Peer jako odpowiedni punkt wyjścia do poszukiwań algorytmów rozwiązywania kolizji. Algorytmy te są ściśle związane z wycinkiem rzeczywistości, który opisują dane przechowywane w bazie. W miarę rozszerzania zasięgu systemu eksperymentalnego będą powstawały nowe obszary do analizy sposobu rozwiązywania konfliktu równoczesnej aktualizacji. W obecnej sytuacji podstawą do wycofania operacji jest czas, w którym informacja o niej dociera do replikatora. Instrukcja nadchodząca, od jednego z serwerów danych, może być zaakceptowana, gdy serwer ten wykonał każdą operację umieszczoną w kolejce replikatora. W przeciwnym przypadku jest ona wycofywana. Wykorzystanie tego modelu do budowy replikatora w połączeniu z zastosowaniem mechanizmów aktywnej bazy danych wymaga uruchomienia lokalnego klienta we wszystkich węzłach posiadających kopie replikowanej tabeli. Procedura bazy danych (wywołana w efekcie dokonanych lokalnie zmian) może spowodować lawinowe uruchamianie podobnych procedur w wyniku propagacji zmian na inne wierzchołki sieci. Jednym z rozwiązań tego problemu jest rozszerzenie tabeli o dodatkową kolumnę niewidoczną dla użytkownika (dostęp do tabeli odbywa się np. przez perspektywę). Wartości w niej umieszczane pozwalają na warunkowe wykonanie procedury bazy danych w przypadku pierwotnej modyfikacji.

5.2. Kontrola współbieżnego dostępu do danych

W systemie eksperymentalnym przyjęto dwustopniowy sposób blokowania (blokady logiczne klienta oraz blokady fizyczne procesorów danych). Dzięki blokadom logicznym klient może wycofać transakcję, której kontynuacja, w danej chwili, jest niemożliwa, nie

angażując w ten proces lokalnych serwerów ani procesorów danych. Blokady logiczne przydzielane są etapami przed każdym kolejnym podzapytaniem.

Mechanizm blokad logicznych został oparty na tzw. *dryfujących krotkach*, które odzwierciedlają stan blokad w całym systemie rozproszonym. Każdy z klientów informacje o blokadach związanych z aktualnie wykonywaną transakcją umieszcza w przestrzeni krotek. Jednostką granulacji blokady logicznej jest tabela fizyczna, której identyfikator składa się z następujących elementów:

- numeru lokalnego serwera, na którym znajduje się tabela,
- nazwy lokalnej bazy danych zawierającej tabelę,
- nazwy tabeli.

Krotka z blokadą zawiera następujące informacje:

- identyfikator tabeli fizycznej, której dotyczy blokada,
- identyfikator transakcji, która ją nałożyła,
- rodzaj blokady (zapisu lub całkowita [4]).

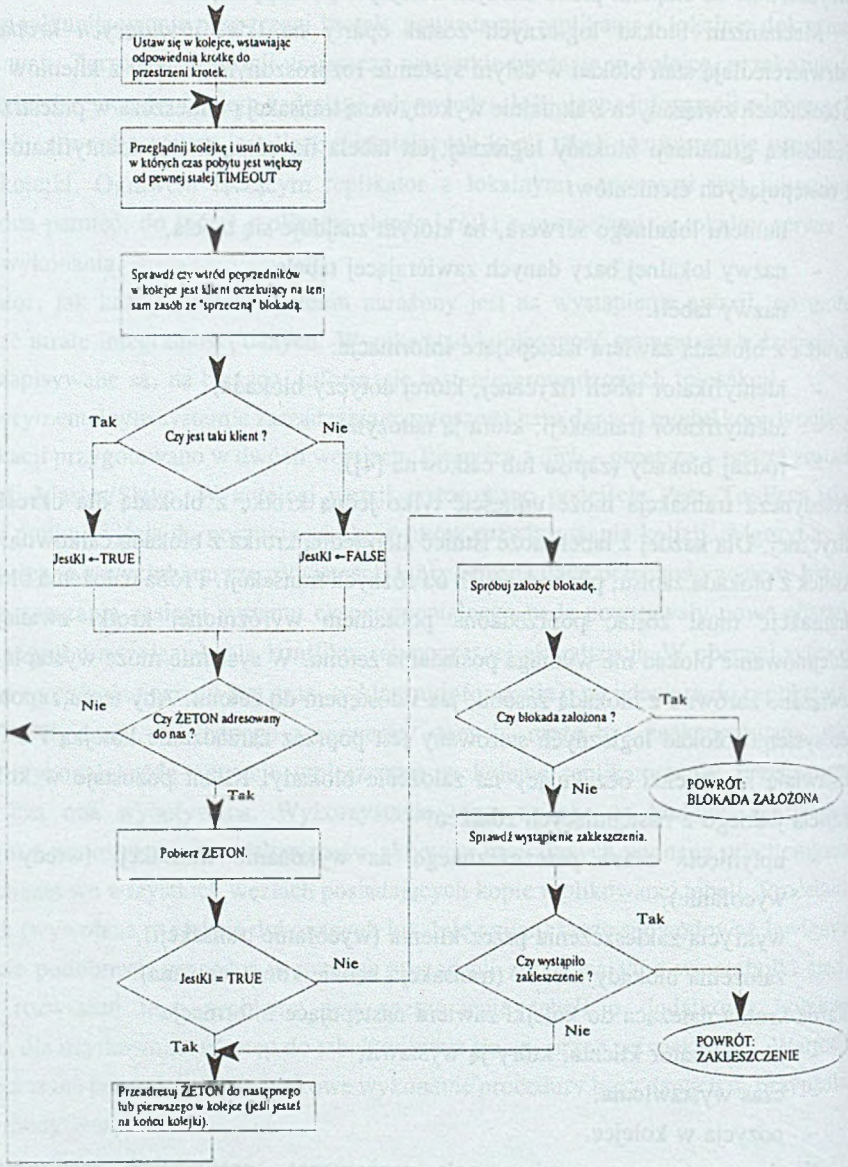
Pojedyncza transakcja może umieścić tylko jedną krotkę z blokadą dla określonej tabeli fizycznej. Dla każdej z tabel może istnieć albo jedna krotka z blokadą całkowitą, albo wiele krotek z blokadą zapisu, pochodzących od różnych transakcji. Próba nałożenia blokady przez transakcję musi zostać poprzedzona pobraniem wyróżnionej krotki zwanej *żetonem*. Zdejmowanie blokad nie wymaga posiadania żetonu. W systemie może wystąpić uwięzienie związane zarówno z blokadą zasobu, jak i dostępem do żetonu. Aby temu zapobiec, dostęp do systemu blokad logicznych sterowany jest poprzez zarządzanie kolejką FIFO, w której ustawiają się klienci oczekujący na założenie blokady. Klient pozostaje w kolejce aż do zajścia jednego z następujących zdarzeń:

- upłynięcia czasu przeznaczonego na wykonanie transakcji (wtedy nastąpi jej wycofanie),
- wykrycia zakleszczenia przez klienta (wycofanie transakcji),
- założenia blokady zasobu (transakcja będzie kontynuowana).

Każda krotka należąca do kolejki zawiera następujące informacje:

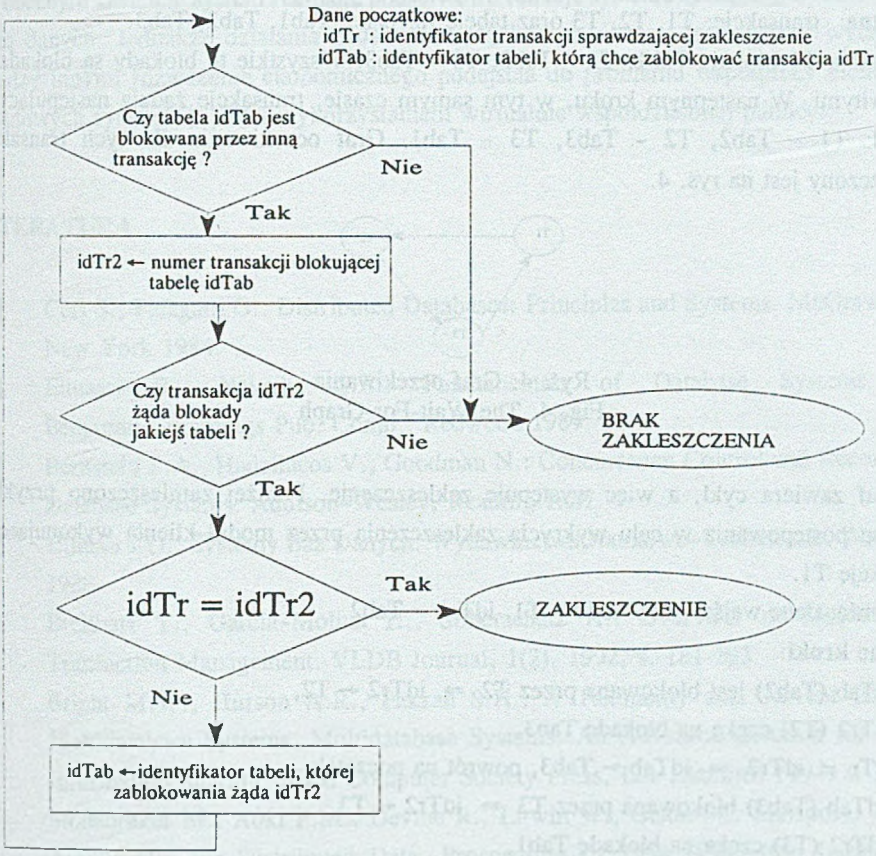
- identyfikator klienta, który ją wystawił,
- czas wystawienia,
- pozycja w kolejce.

Krotka z żetonem zawiera pole numeryczne, pozwalające zaadresować ją do odpowiedniego klienta. Może ją pobrać tylko ten klient, którego pozycja jest zgodna z numerem żetonu. Próbę założenia blokady klient podejmuje dopiero wtedy, gdy stwierdzi, że krotka z żetonem jest adresowana do niego oraz że wśród poprzedników w kolejce nie ma klienta uniemożliwiającego mu założenie blokady, tzn. czekającego na ten sam zasób z zamiarem założenia blokady "sprzecznej". Awaria modułu klienta, który wstawił krotkę



Rys. 2. Obsługa kolejki FIFO
Fig. 2. Maintaining the FIFO queue

do kolejki, mogłaby spowodować, że taka krotka przez dłuższy czas nie zostałaby pobrana. Dlatego każdy z klientów kontroluje wszystkie krotki w kolejce, sprawdzając czas wstawienia krotki do kolejki. Gdy czas ten jest większy od pewnej wartości granicznej, ustalonej dla systemu, klient zabiera taką krotkę. Wymaga to od klienta odświeżania czasu wstawienia swojej krotki do kolejki. Algorytm obsługi kolejki FIFO do żetonu został przedstawiony na rys. 2.



Rys. 3. Algorytm wykrywania zakleszczeń
Fig. 3. The deadlock detection algorithm

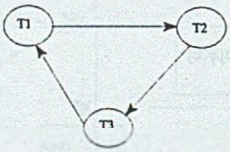
Dzięki zastosowaniu mechanizmu dryfujących krotek można również wykryć fakt wystąpienia zakleszczenia. Jeżeli w trakcie nakładania blokady okaże się, że nie jest to możliwe, klient sprawdza, czy doszło do zakleszczenia. Sprawdzenie to odbywa się za pomocą algorytmu opartego na konstrukcji grafu oczekiwania (rys. 3) [4, 9]. Klient buduje graf i sprawdza, czy istnieje w nim cykl, do którego on sam należy. Jeżeli okaże się, że taki

cykl istnieje, klient podejmuje decyzję o wycofaniu swojej transakcji, likwidując w ten sposób zakleszczenie. Implementacja tego algorytmu wymaga umieszczenia w przestrzeni krotek dodatkowych informacji o zasobach, na blokadę których oczekują poszczególni klienci. Krotka taka zawiera informacje takie jak krotka z blokadą oraz dodatkowo numer identyfikacyjny klienta i jest wstawiana do przestrzeni krotek bezpośrednio po ustawieniu się klienta w kolejce.

Przykład

Dane: transakcje: T1, T2, T3 oraz tabele fizyczne: Tab1, Tab2, Tab3.

Transakcja T1 blokuje Tab1, T2 - Tab2, T3 - Tab3. Wszystkie te blokady są blokadami całkowitymi. W następnym kroku, w tym samym czasie, transakcje żądają następujących blokad: T1 - Tab2, T2 - Tab3, T3 - Tab1. Graf oczekiwania dla tych transakcji umieszczony jest na rys. 4.



Rys. 4. Graf oczekiwania
Fig. 4. The Wait-For-Graph

Graf zawiera cykl, a więc występuje zakleszczenie. Poniżej zamieszczono przykład sposobu postępowania w celu wykrycia zakleszczenia przez moduł klienta wykonującego transakcję T1.

Założenie: dane wejściowe - idTr = T1, idTab = Tab2.

Kolejne kroki:

1) idTab (Tab2) jest blokowana przez T2 $\Rightarrow$ idTr2 $\leftarrow$ T2,
2) idTr2 (T2) czeka na blokadę Tab3,
3) idTr $\neq$ idTr2 $\Rightarrow$ idTab $\leftarrow$ Tab3, powrót na początek,
1') idTab (Tab3) blokowana przez T3 $\Rightarrow$ idTr2 $\leftarrow$ T3,
2') idTr2 (T3) czeka na blokadę Tab1,
3') idTr $\neq$ idTr2 $\Rightarrow$ idTab $\leftarrow$ Tab1, powrót na początek,
1") idTab (Tab1) blokowana przez T1 $\Rightarrow$ idTr2 $\leftarrow$ T1,
2") idTr2 (T1) czeka na blokadę Tab2,
3") idTr = idTr2 $\Rightarrow$ wystąpiło zakleszczenie.

Transakcja T1 zostanie wycofana, co umożliwi kontynuację transakcji T2 oraz T3.

6. Podsumowanie

Wpływ na proces rozwoju systemu eksperymentalnego miało zastosowanie środowiska programowania równoległego i rozproszonego Paradise. Wykorzystanie współdzielonej pamięci do komunikacji pomiędzy poszczególnymi elementami systemu znalazło odbicie w sposobie realizacji omówionych mechanizmów współbieżnego dostępu do danych. W obecnym kształcie system realizuje podstawowe funkcje systemu zarządzania rozproszoną bazą danych. Jednakże działania badawcze będą kontynuowane. Interesujące wydaje się między innymi rozważenie ekonomicznego podejścia do problemu współpracy elementów składowych systemu [7, 8] z wykorzystaniem wirtualnie współdzielonej pamięci.

LITERATURA

[1] Ceri S., Pelagatti G.: Distributed Databases: Principles and Systems. McGraw-Hill, New York 1984.

[2] Elmasri R., Navathe S.B.: Fundamentals of Database Systems. The Benjamin/Cummings Pub. Comp., Redwood 1989.

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

[4] Ullman J.D.: Systemy Baz Danych. Wydawnictwa Naukowo-Techniczne, Warszawa 1988.

[5] Breitbart Y., Garcia-Molina H., Silberschatz A.: Overview of Multidatabase Transaction Management. VLDB Journal, 1(2), 1992, s. 181-293.

[6] Bright M.W., Hurson A.R., Pakzad S.H.: A Taxonomy and Current Issues in Multidatabase Systems. Multidatabase Systems: An Advanced Solution for Global Information Sharing. IEEE Computer Society Press, Los Alamitos 1994, s. 3-12.

[7] Stonebraker M., Aoki P.M., Devine R., Litwin W., Olson M.: Mariposa: A New Architecture for Distributed Data. Proceedings of 1994 IEEE 10th International Conference on Data Engineering. Houston, TX, USA, Februar 14-18, 1994, IEEE Computer Society Press, Los Alamitos 1994, s. 54-65.

[8] Stonebraker M., Aoki P.M., Pfeffer A., Sah A., Sidell J., Staelin C., Yu A.: Mariposa: A Wide-Area Distributed Database System. VLDB Journal, 5(1), 1996, s. 48-63.

[9] Singhal M.: Deadlock Detection in Distributed Systems. Readings in Distributed Computing Systems. IEEE Computer Society Press, Los Alamitos 1994, s. 52-71.

[10] Yeung C., Hung S., Lam K.: Performance Evaluation of A New Distributed Deadlock Detection Algorithm. SIGMOD Record 23(3), September 1994, s. 21-26.

[11] Staniszkis S.: Projektowanie rozproszonych systemów informatycznych. Informatyka nr 1, 1995, s. 6-11.

[12] Bajerski P., Josiński H., Stąpor K.: Zastosowanie systemu Paradise do tworzenia systemu zarządzania rozproszoną bazą danych. Zeszyty Naukowe Politechniki Śl., s. Informatyka, z. 30, Gliwice 1996, s. 449–470.

[13] Chłopek M., Harężlak K., Josiński H.: Implementacja podstawowych mechanizmów zarzadzania rozproszoną bazą danych w eksperymentalnym systemie wykorzystującym model wirtualnie współdzielonej pamięci. Zeszyty Naukowe Politechniki Śl., s. Informatyka, z. 32, Gliwice 1997, s. 267-283.

[14] Praca zbiorowa pod redakcją S. Kozielskiego: Systemy umożliwiające realizację algorytmów równoległych w sieciach komputerowych. Skrypt Politechniki Śl. nr 1975, Gliwice 1996.

[15] Schussel G.: Replication, The Next Generation of Distributed Database Technology (via Internet: *http://www.dci.com/speakers/archive/replica.htm*).

[16] Stacey D.: Replication: DB2, Oracle, or Sybase ? Database Programming & Design, December 1994, s. 42-50.

[17] Paradise User's Guide & Reference Manual. Scientific Computing Associates, 1995.

[18] ASK/INGRES Technical Communications: INGRES Database Administrator's Guide. Ingres Corporation, Alameda 1992.

Recenzent: Dr hab. inż. Adam Mrózek, Prof. Politechniki Śląskiej

Wpłynęło do Redakcji 23 grudnia 1997 r.

Abstract

The article presents implementation of distributed database management mechanisms in an experimental system working on a cluster of Sun workstations connected by an Ethernet network. The architecture of the system was based on virtual shared memory model supported by the Paradise system. The functions as well as the tasks performed by application processor have been presented. The application processor consists of the following modules: clients and local servers. The global structure of the application processor has been shown in Figure 1. The authors have focused on the global transaction commitment,

concurrency control and maintaining replicated data items. The distributed transaction is committed or aborted according to the 2PC protocol. The distributed deadlock detection and resolution algorithm in connection with global logical locking mechanism has been proposed. The Replication Manager, responsible for coordination all tasks performed on replicated data items, has been implemented in the experimental system to resolve problems connected with replicas. This module was prepared in two versions: Master/Slave and Peer-To-Peer.