STUDIA INFORMATICA 2001
Volume 22 Number 3 (45)

Katarzyna HARĘŻLAK
Politechnika Śląska, Instytut Informatyki

REALIZACJA TRANSAKCJI NA KOPIACH DANYCH W ŚRODOWISKU WIRTUALNIE WSPÓŁDZIELONEJ PAMIĘCI

Streszczenie. W artykule omówione zostały różne sposoby tworzenia kopii danych oraz różne metody ich aktualizacji. Przedstawiono możliwości zastosowania wybranych metod w środowisku wirtualnie współdzielonej pamięci. Zaproponowano protokoły realizacji transakcji dla wybranych metod i poddano je wstępnej ocenie. Prowadzone badania są kontynuacją prac dotyczących budowy eksperymentalnego systemu zarządzania rozproszoną bazą danych.

TRANSACTION MANAGEMENT ON REPLICATED DATA IN VIRTUAL SHARED MEMORY ENVIRONMENT

Summary. Various update methods of replicated data have been presented. This paper also presents the possibilities of using virtual shared memory in transaction management on replicated data. The chosen update methods have been implemented. The research is the continuation of the studies referring to the already created experimental distributed database management system.

1. Wstęp

Rozwój sieci komputerowych, a co za tym idzie, rozwój systemów rozproszonych baz danych zapewnia szeroki dostęp do informacji poprzez integrację wielu baz danych, niezależnie od ich położenia i środowiska ich funkcjonowania. Rozproszona baza danych, na którą składają się fragmenty pionowe, poziome i kopie danych, z punktu widzenia użytkownika jest jedną scentralizowaną bazą danych. Jednak korzystając z takich danych, należy mieć na uwadze fakt, że pojedyncza transakcja może wymagać dostępu do kilku lokalnych baz danych, co wprowadza pewne opóźnienia lub w przypadku awarii sieci

komputerowej wręcz uniemożliwia realizację tej transakcji. Wymagania dotyczące efektywności wykorzystania rozproszonych baz danych spowodowały wzrost zainteresowania użytkowaniem kopii danych. Korzystanie z takich danych ma wiele zalet. Należą do nich między innymi: przyspieszenie dostępu do danych, ograniczenie ruchu w sieci oraz większa odporność danych na awarie. Niesie jednak też ze sobą utrudnienia związane z wykonywaniem aktualizacji danych, np.: spójność danych we wszystkich węzłach wymaga, by były one aktualizowane w jednej transakcji, co znacznie opóźnia jej realizację, natomiast niezależna modyfikacja dwóch kopii danych przez różnych użytkowników może doprowadzić do rozbieżności w kopiach danych.

Rozmieszczenie kopii danych w wielu węzłach sieci stawia problem uaktualniania danych tak, by transakcje realizowane w różnych węzłach sieci operowały na tym samym zestawie rekordów. Dobór odpowiedniego sposobu powielenia danych i przyjęcie właściwej strategii ich aktualizacji zależne jest od zadań stawianych systemowi przez jego użytkowników. Efektem tego jest pojawienie się różnych sposobów replikowania danych oraz różnych strategii ich modyfikacji. Jednak problem aktualizacji powielonych danych wydaje się być wciąż aktualny, gdyż jego rozwiązanie ściśle związane jest z klasą zadań, której dotyczy, jak również dostępnym środowiskiem pracy. W związku z tym interesująca wydaje się analiza możliwości wykorzystania istniejących algorytmów w środowisku wirtualnie współdzielonej pamięci (model Lindy), co stanowi przedmiot rozważań w niniejszym opracowaniu. Rozważania te są kontynuacją prac dotyczących budowy eksperymentalnego systemu zarządzania bazą danych, opisanego w dalszej części artykułu (rozdział 3).

2. Sposoby replikowania danych i strategie ich uaktualniania

Powielanie danych w różnych węzłach sieci może odbywać się na kilka sposobów [3]:

- kopie identyczne – prawa modyfikacji danych posiadają użytkownicy wszystkich kopii powielonych danych,
- kopie Master/Slave – możliwość zmiany, usuwania lub dodawania rekordów przysługuje użytkownikom jednej kopii (zwanej master), pozostałe natomiast służą tylko do odczytu,
- kopie w postaci migawek – stanowiących widok tabeli bazowej, odświeżany w określonych okresach czasu.

Aktualizacja kopii może być realizowana z wykorzystaniem różnych strategii i jest zależna od przyjętego sposobu powielania danych. Strategie te mogą być sklasyfikowane w następujący sposób [3]:

- aktualizacja synchroniczna (*synchronous all*), w której wszystkie kopie modyfikowane są jednocześnie,
- aktualizacja synchroniczna dla dostępnych węzłów (*synchronous available*) – w transakcji biorą udział aktualnie aktywne węzły; pozostałe kopie aktualizowane są w sposób asynchroniczny,
- aktualizacja bazująca na kworum (*quorum–based*) – w ramach transakcji aktualizowana jest grupa kopii stanowiących kworum; pozostałe kopie aktualizowane są w sposób asynchroniczny,
- aktualizacja niezależna (*independent*) – kopie aktualizowane są w sposób niezależny; może to doprowadzić do kolizji, tzn. do sytuacji, w której dwie różne aplikacje modyfikują dwie różne kopie w tym samym przedziale czasu,
- Master/Slave (*Primary/Secondary*) – istnieje możliwość aktualizowania tylko jednej kopii synchronicznie – kopii Master i asynchronicznie kopii typu Slave,
- Master/Backup (*Primary/Backup*) – aktualizowana jest kopia Master. Jedna z kopii Slave stanowi kopię rezerwową dla kopii Master, wykorzystywaną w celu szybkiego odzyskania danych aktualnych w przypadku awarii kopii głównej. Kopia rezerwowa aktualizowana jest synchronicznie z kopią główną albo w pierwszej kolejności po zakończeniu transakcji dotyczącej kopii Master. Pozostałe kopie uaktualniane są asynchronicznie.

Korzystając z tych strategii rozwinięto wiele metod uaktualniania danych. Są to między innymi [3,4,5]:

- synchronous all – metoda ROWA (*Read One Write All*),
- synchronous all available – metody: ROWAA (*Read One Write All Available*), DOAC (*Directory–Oriented Available Copies*),
- quorum–based – metody: QC (*Quorum Consensus*), TQP (*Tree Quorum Protocol*), MW (*Missing Writes*), VP (*Virtual Partition*), TGP (*Triangular Grid Protocol*),
- independent – metody: MAR *(Multi–Airline Reservation)* oraz IND *(Independent)*,
- oparte na modyfikacji jednej kopii, Master – metody: RDF *(Remote Backup Database Facility)*, RBP *(Remote Backup Procedure)*, ASAP *(As Soon As Possible)*.

Jako przedmiot dalszej analizy wybrano cztery metody, reprezentujące wybrane strategie. Są to metody: ROWA, TGP, IND oraz ASAP.

3. Środowisko badań

Do analizy i implementacji wybranych metod wykorzystany został eksperymentalny system zarządzania rozproszoną bazą danych, pracujący w lokalnej sieci Ethernet, łączącej

grupę stacji roboczych Sun [7,8,9]. Stacje te posiadają różną architekturę i różnią się mocą obliczeniową. W systemie zaimplementowano procesor aplikacji z wykorzystaniem wybranych systemów zarządzania bazami danych (Ingres, Postgres95) jako procesorów danych. Do realizacji zadań procesora aplikacji zastosowano model wieloprocesorowego komputera z wirtualnie współdzieloną pamięcią (model Lindy). Implementacji dokonano bazując na, wykorzystującym ten model, środowisku programowania rozproszonego i równoległego Paradise [10]. Do zalet tego środowiska należy przede wszystkim zaliczyć:

- 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 (takie jak: wprowadzenie krotki, odczyt niszczący jej zawartość, usunięcie wszystkich krotek),
- operację nieniszczącego odczytu zawartości krotki.

Dostęp procesora aplikacji do zawartości lokalnych baz danych za pośrednictwem odpowiedniego procesora danych uzyskano stosując technikę zanurzania instrukcji języka SQL w języku C.

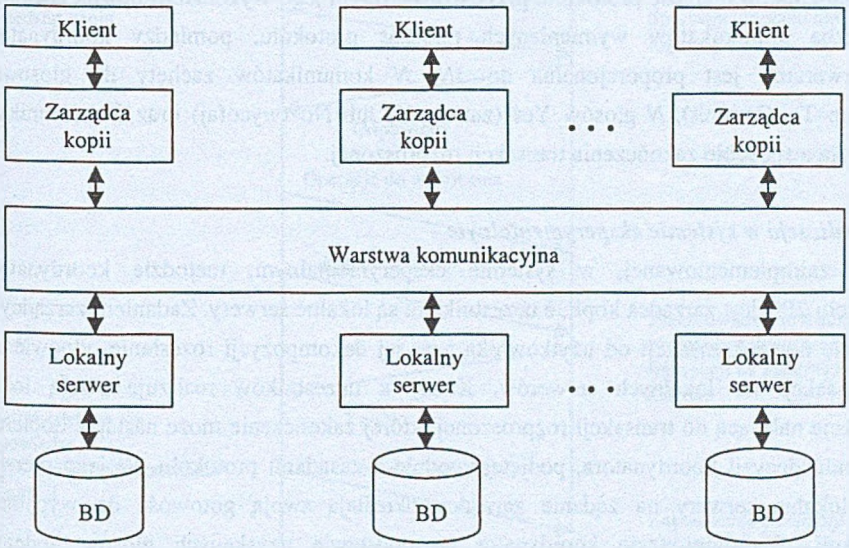
W skład procesora aplikacji wchodzą moduły *klienta* i *lokalnego serwera* [2]. W systemie może równocześnie funkcjonować dowolna liczba programów klienta oraz lokalne serwery w liczbie odpowiadającej liczbie procesorów danych. Poszczególne moduły uruchamiane są niezależnie na różnych komputerach. Komunikują się one poprzez wirtualnie współdzieloną pamięć – *przestrzeń krotek* – za pośrednictwem jednostek nazywanych *krotkami*, stanowiących sekwencje pól. Program klienta jest uruchamiany na lokalnym komputerze użytkownika. Stanowi on interfejs między użytkownikiem i modułem lokalnego serwera. Do jego zadań zalicza się: pobranie od użytkownika ciągu zapytań, jego dekompozycję na podzapytania operujące na danych w różnych węzłach sieci, wysłanie ich do lokalnych serwerów oraz prezentacja wyników [7,8,9]. Klient odpowiedzialny jest również za zarządzanie swoją aktualnie wykonywaną transakcją rozproszoną oraz kontrolę współbieżnego dostępu do danych z wykorzystaniem mechanizmu blokad logicznych, zrealizowanego poprzez wprowadzenie dodatkowego mechanizmu blokowania opartego na protokole Two-Phase Locking [1,2]. Mechanizm ten, pozwalający między innymi na wykrycie zakleszczenia, stosowany jest jako uzupełnienie blokad nakładanych przez procesory danych.

Wymiana informacji pomiędzy modułem klienta a procesorami danych realizowana jest przez lokalne serwery. Specyfika realizacji funkcji lokalnego serwera związana jest z właściwościami systemu stanowiącego procesor danych. Należy do nich między innymi:

ewentualna translacja zapytań w języku SQL na język właściwy dla procesora danych lokalnej bazy danych (np. w systemie Ingres może to być język QUEL), zarządzanie lokalnymi blokadami, o ile tej funkcji nie wykonuje sam procesor danych oraz zarządzanie lokalnymi transakcjami.

4. Zarządzanie kopiami danych w eksperymentalnym systemie zarządzania rozproszoną bazą danych

W tak przygotowanym środowisku pracy przebadane zostały możliwości wykorzystania dwóch modeli powielonych danych: kopie identyczne oraz kopie typu Master/Slave. W tym celu system eksperymentalny został wzbogacony o moduł *replikatora* — zarządcy powielonymi danymi na poziomie całego systemu oraz moduł *zarządcy kopii*, odpowiedzialnego za koordynowanie lokalnych transakcji operujących na lokalnych kopiach danych. Komunikacja pomiędzy tymi modułami odbywa się za pomocą mechanizmów aktywnych baz danych – zdarzeń, wyzwalaczy, procedur baz danych oraz poprzez przestrzeń krotek.



Rys. 1. Architektura systemu zarządzania kopiami danych
Fig. 1. The architecture of replicated data management system

4.1. Aktualizacja synchroniczna

Jedną ze strategii aktualizacji identycznych kopii danych jest strategia synchroniczna, która omówiona zostanie na przykładzie metody ROWA (ang. *Read One Write All*). W metodzie tej operacja Read(x) odczytu danych x realizowana jest jako Read(x_A), gdzie x_A oznacza dowolną kopię tych danych znajdującą się w węźle A. Natomiast operacja Write(x) zamieniana jest na zestaw operacji {Write(x_{A1}), . . ., Write(x_{An})}, gdzie {x_{A1}, . . ., x_{An}} są kopiami x w węzłach od 1 do n. Do realizacji transakcji zgodnie z tą metodą wykorzystuje się protokół dwufazowego zatwierdzania transakcji 2PC (ang. *The Two Phase Commit Protocol*). Protokół ten składa się z fazy głosowania i fazy kończącej [1]. W pierwszej części protokołu każdy serwer, realizujący operacje Write(x_{Ai}), głosuje za zatwierdzeniem lub wycofaniem transakcji. Z chwilą zagłosowania za zatwierdzeniem transakcji nie wolno serwerowi już jej wycofać. W drugiej fazie protokołu każdy serwer biorący udział w transakcji realizuje wspólną decyzję. Jeżeli którykolwiek z serwerów głosował za wycofaniem transakcji, to musi zostać podjęta decyzja o jej wycofaniu. Problem więc polega na tym, by w głosowaniu wzięły udział wszystkie serwery i żeby podjęły tę samą decyzję, gdyż wskutek awarii przynajmniej jednego z nich lub z powodu przerwy w komunikacji transakcja kończy się niepowodzeniem.

Analizując wydajność protokołu, przy bezawaryjnym jego wykonaniu, można stwierdzić, że liczba komunikatów wymienianych podczas protokołu, pomiędzy koordynatorem a N serwerami, jest proporcjonalna do *3N: N* komunikatów zachęty do głosowania (Prepare–To–Commit), *N* głosów Yes (zatwierdź) lub No (wycofaj) oraz *N* komunikatów z decyzją o sposobie zakończenia transakcji rozproszonej.

Realizacja w systemie eksperymentalnym

W zaimplementowanej, w systemie eksperymentalnym, metodzie koordynatorem protokołu 2PC jest zarządca kopii, a uczestnikami są lokalne serwery. Zadaniem zarządcy jest przyjęcie nowej transakcji od użytkownika i po jej dekompozycji rozesłanie odpowiednich podtransakcji do lokalnych serwerów. Każdy z uczestników realizuje swoją lokalną transakcję należącą do transakcji rozproszonej, której zakończenie może nastąpić dopiero po uzyskaniu decyzji koordynatora, podjętej zgodnie z zasadami protokołu. W jego pierwszej fazie lokalne serwery na żądanie zarządcy określają swoją gotowość do wypełnienia transakcji. W drugiej fazie koordynator na podstawie uzyskanych głosów podejmuje ostateczną decyzję, realizacja której zamyka protokół 2PC. Jeżeli przynajmniej jeden z uczestników transakcji jest niedostępny lub zgłosi brak takiej gotowości, transakcja musi zostać wycofana we wszystkich węzłach.

Analiza protokołu 2PC, w kontekście użytych narzędzi wykorzystanych do komunikacji pomiędzy modułami (odbywa się ona poprzez przestrzeń krotek), wykazała możliwość

ograniczenia liczby przesyłanych komunikatów. Zamiast wysyłania wezwania do głosowania oraz komunikatów z decyzją dotyczącą sposobu zakończenia transakcji do wszystkich lokalnych serwerów, do przestrzeni krotek wstawiana jest jedna krotka danego typu, skąd może być odczytana przez wszystkie procesy biorące udział w rozproszonej transakcji. Stąd liczba przesyłanych komunikatów w postaci krotek zmniejsza się do $N+2$. Poniżej przedstawiono postać komunikatów (krotek) wymienianych przez moduły systemu podczas realizacji protokołu 2PC:

(„Prepare–To–Commit", Id_zarządcy, Id_transakcji) – wezwanie do głosowania,

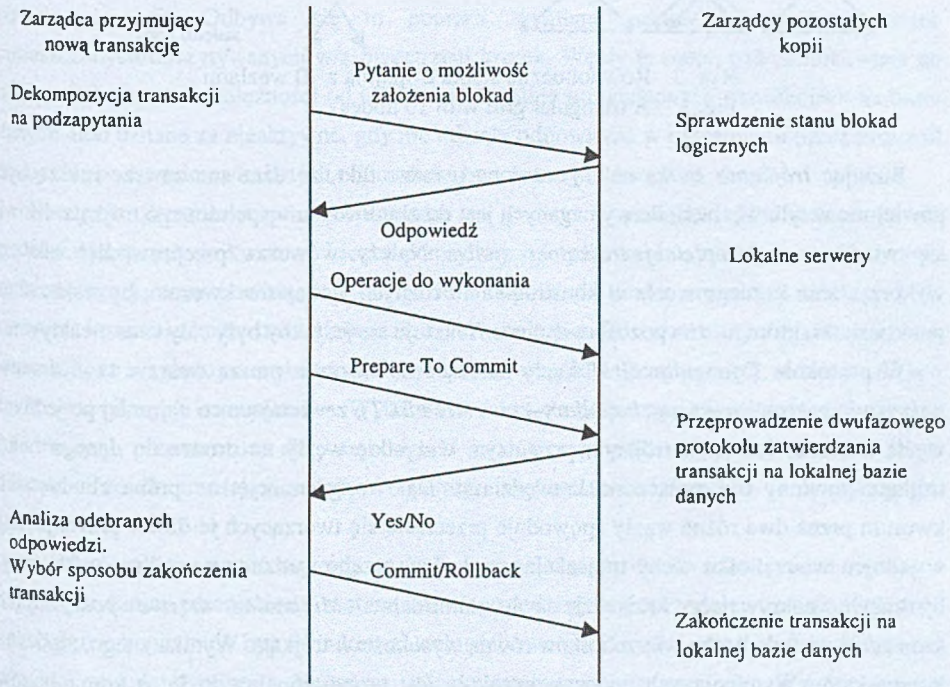
(„Odpowiedź", Id_zarządcy, Id_transakcji, Id_serwera, odpowiedź) – odpowiedź serwera,

(„Decyzja", Id_zarządcy, Id_transakcji, decyzja) – decyzja koordynatora.

Ponadto w przestrzeni krotek umieszczane są komunikaty z operacjami należącymi do transakcji o postaci:

(„Transakcja", Id_zarządcy, Id_transakcji, liczba_operacji)

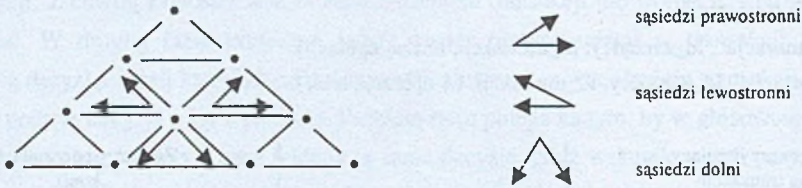
(„Operacja", Id_zarządcy, Id_transakcji, Id_operacji, treść).



Rys. 2. Protokół realizacji transakcji oparty na strategii synchronicznej
Fig. 2. The protocol of transaction realization based on the synchronous strategy

4.2. Aktualizacja bazująca na kworum

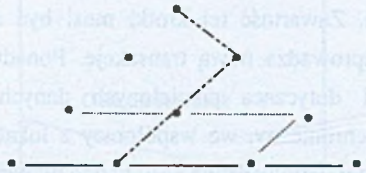
Spośród grupy metod określanych jako *Quorum–based* przeanalizowano protokół *Triangular Grid* [5], w którym wszystkie wierzchołki zawierające kopie danych odwzorowane są w logiczną strukturę *trójkątnej siatki*, niezależnie od ich rzeczywistego rozłożenia w sieci. Trójkąt siatki jest równoboczny. Punkty przecięcia zawierają węzły z powielonymi danymi, z których każdy zaznajomiony jest z fizycznym położeniem pozostałych kopii danych. *Wysokość h* trójkąta odpowiada liczbie poziomów siatki, natomiast bok trójkąta zwany jest jego *linią graniczną*. Węzły umieszczone w trójkącie, w zależności od ich miejsca w siatce, mają *sąsiadów* lewostronnych, prawostronnych lub dolnych. Schemat takiego trójkąta pokazano na rys. 3.



Rys. 3. Równoboczna siatka trójkątna z 10 węzłami
Fig. 3. A triangular grid with 10 nodes

Budując *trójkątną siatkę* należy mieć na uwadze fakt, że dane nie zawsze muszą być powielone w tylu węzłach, ile wymaganych jest do zbudowania wypełnionego trójkąta. Mówi się wtedy o *niekompletnej trójkątnej siatce*. Należy wówczas przeprowadzić analizę wykorzystania każdego węzła w konstruowaniu różnych wariantów kworum, by wskazać te punkty siatki, które można pozostawić puste. Traktuje się je, jakby były cały czas nieaktywne.

W protokole *Triangular Grid* węzły należące do kworum muszą tworzyć tzw. *drzewo pokrywające krawędzie (*ang. *boundary–cover–tree BCT)*, zawierające co najmniej po jednym węźle z każdej krawędzi trójkąta, przy czym wszystkie węzły na drodze do danego boku trójkąta powinny być połączone krawędziami. Daje to gwarancję, że próba zbudowania kworum przez dwa różne węzły spowoduje przecięcie się tworzących je drzew przynajmniej w jednym wierzchołku. Żeby transakcja mogła być przeprowadzona pomyślnie, wystarczy, by drzewo pokrywające krawędzie było minimalne. *Minimalne drzewo pokrywające krawędzie* posiada liczbę wierzchołków równą wysokości *h* trójkąta. Wynika z tego, że liczba komunikatów wymienianych podczas protokołu jest proporcjonalna do *3h*: *h* komunikatów zachęty do głosowania, *h* głosów Yes lub No oraz *h* komunikatów z decyzją o sposobie zakończenia transakcji rozproszonej. Przykłady minimalnych drzew pokrywających krawędzie przedstawia rys. 4.



Rys. 4. Przykłady minimalnych drzew pokrywających krawędzie
Fig. 4. Examples of minimal boundary–cover–trees

Realizacja w systemie eksperymentalnym

W systemie eksperymentalnym protokół *Triangular Grid* inicjowany jest przez jednego z zarządców kopii, który zainteresowany jest przeprowadzeniem transakcji. Ponieważ proces budowy siatki trójkątnej odbywa się w momencie tworzenia systemu lub jego ewentualnej przebudowy (dodawanie lub usuwanie węzłów z kopiami), każdy zarządca kopii zna zarówno swoje położenie w tej strukturze, jak i różne warianty kworum pozwalające na realizację jego transakcji. W pierwszym kroku protokołu sprawdzana jest gotowość węzłów do realizacji nowej transakcji. Odbywa się to poprzez wymianę odpowiedniego typu krotek umieszczanych/odczytywanych w/z przestrzeni krotek. Węzły te mogą być zablokowane do odczytu lub zapisu w zależności od udziału w aktualnie prowadzonych transakcjach na bazie danych albo uznane za nieaktywne, gdy nie udzielą odpowiedzi w określonym przez protokół czasie. Na podstawie nadesłanych odpowiedzi budowane jest kworum.

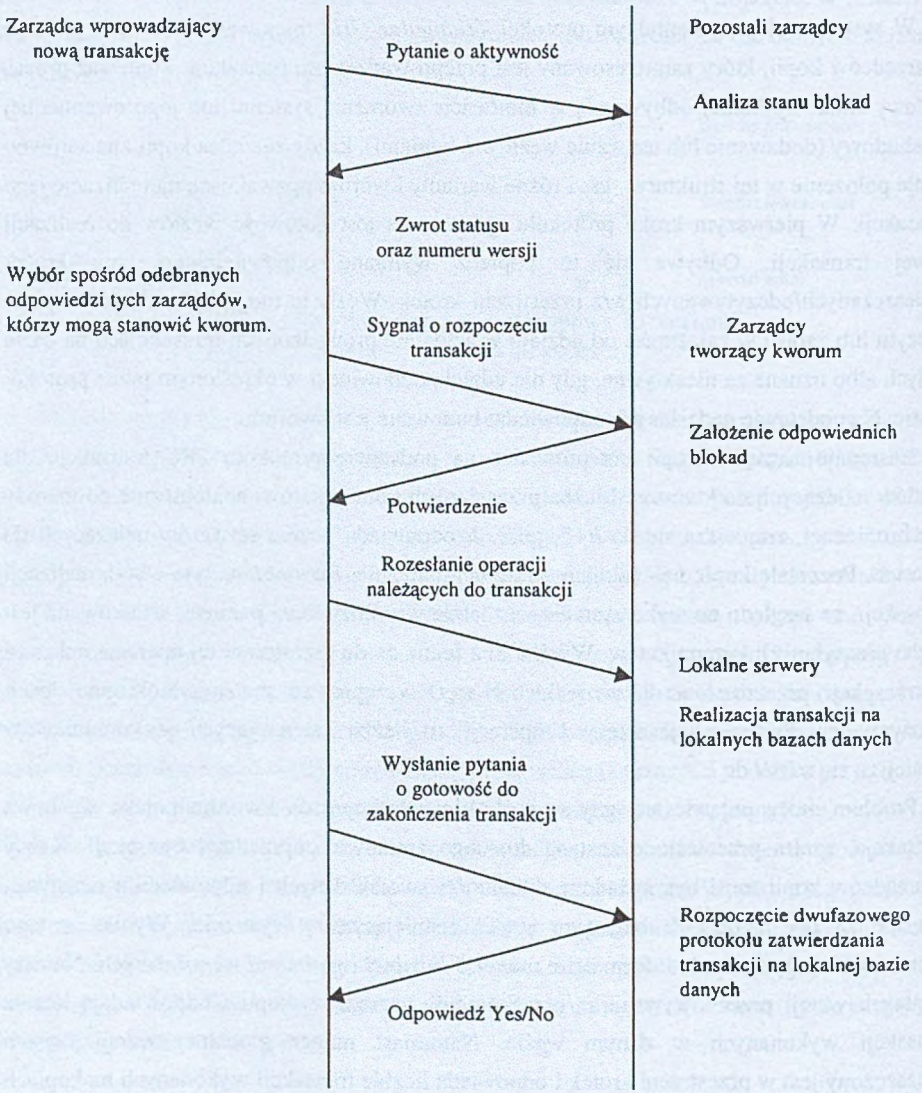
Następnie zarządca kopii przeprowadza na podstawie protokołu 2PC transakcję dla węzłów należących do kworum. Liczba przesyłanych komunikatów, analogicznie do metody synchronicznej, zmniejsza się do $h+2$, gdzie h odpowiada liczbie serwerów należących do kworum. Pozostałe kopie uaktualniane są asynchronicznie. Również na tym etapie realizacji transakcji, ze względu na wykorzystanie wirtualnie współdzielonej pamięci, redukowana jest liczba przesyłanych komunikatów. Wynika to z faktu, że do przestrzeni tej operacje należące do transakcji, przeznaczone dla wszystkich N węzłów, wprowadzane są jednokrotnie. Jeżeli założymy, że do transakcji należy k operacji, to liczba zawierających je komunikatów zmniejsza się z $k*N$ do k.

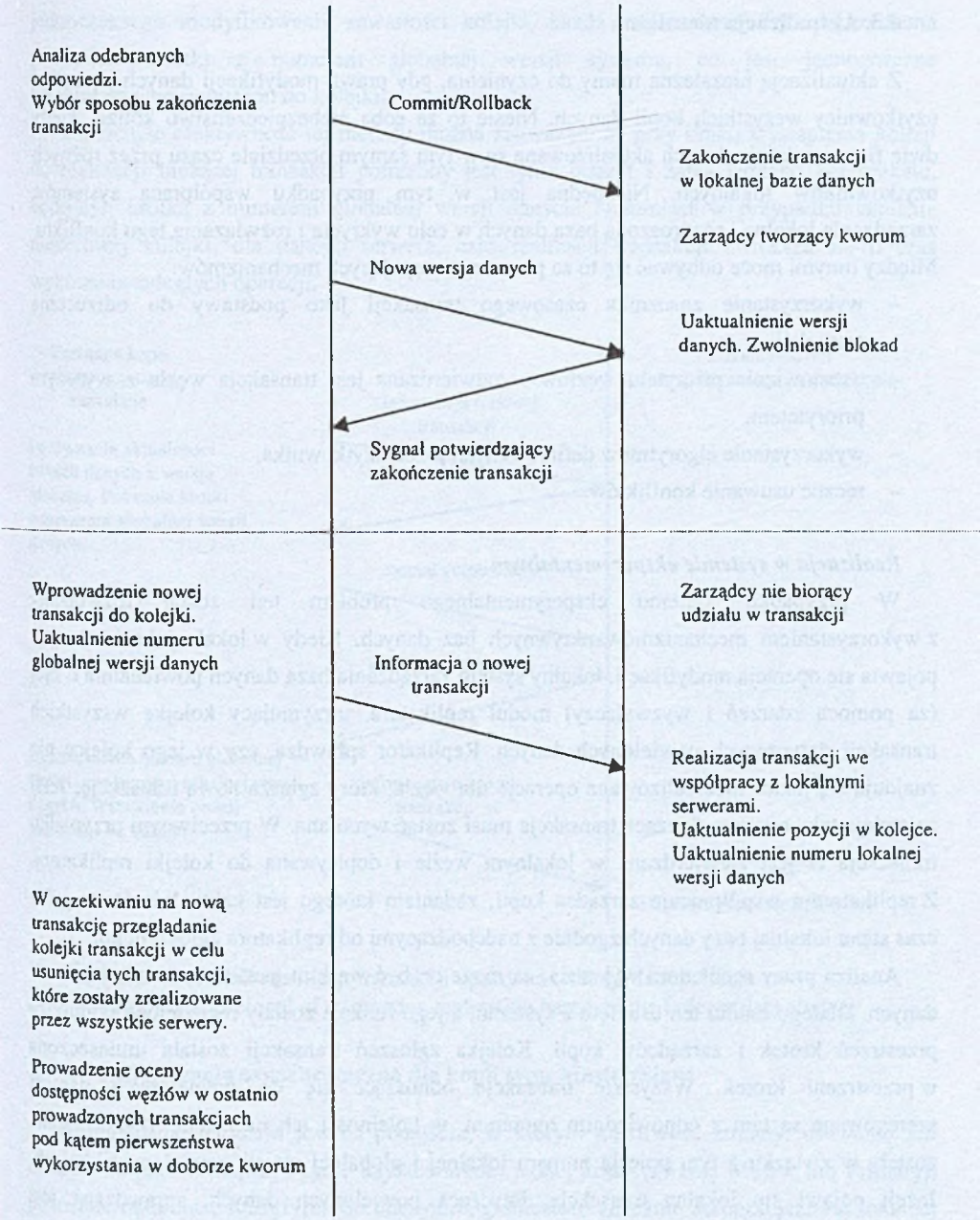
Problem może pojawić się, gdy w węźle nie należącym do kworum pojawi się nowa transakcja, zanim przeniesione zostaną do niego zmiany z poprzedniej transakcji. Każdy z zarządców kopii musi być świadom aktualności swoich danych i odpowiednio reagować, gdy nie są one zgodne z bieżącym stanem istniejącym w systemie. Wynika z tego konieczność utrzymywania informacji o *numerze lokalnej* i *globalnej wersji* danych. Numery lokalnych wersji przechowywane są przez moduły zarządców kopii i odpowiadają liczbie transakcji wykonanych w danym węźle. Natomiast numer globalnej wersji danych umieszczony jest w przestrzeni krotek i odpowiada liczbie transakcji wykonanych na kopiach

danych w całym systemie. Zawartość tej krotki musi być aktualizowana przez każdego zarządcę kopii, który przeprowadza nową transakcję. Ponadto w przestrzeni krotek musi istnieć historia transakcji dotycząca powielonych danych, by umożliwić wszystkim zarządcom w sposób asynchroniczny, we współpracy z lokalnym serwerem, uaktualnianie swoich kopii. W metodzie tej wprowadzone zostały dodatkowo następujące rodzaje krotek:

(„Globalna wersja", nr_globalnej_wersji_danych),

(„Historia", Id_zarządcy, Id_transakcji, liczba_operacji, liczba_uaktualnień)





Rys. 5. Protokół realizacji transakcji oparty na strategii Quorum-based
Fig. 5. The protocol of transaction realization based on the Quorum-based strategy

4.3. Aktualizacja niezależna

Z aktualizacją niezależną mamy do czynienia, gdy prawa modyfikacji danych posiadają użytkownicy wszystkich kopii danych. Niesie to ze sobą niebezpieczeństwo 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. Między innymi może odbywać się to za pomocą następujących mechanizmów:

- wykorzystanie znacznika czasowego transakcji jako podstawy do odrzucenia transakcji,
- ustanowienie priorytetu węzłów – zatwierdzana jest transakcja węzła z wyższym priorytetem,
- wykorzystanie algorytmów definiowanych przez użytkownika,
- ręczne usuwanie konfliktów.

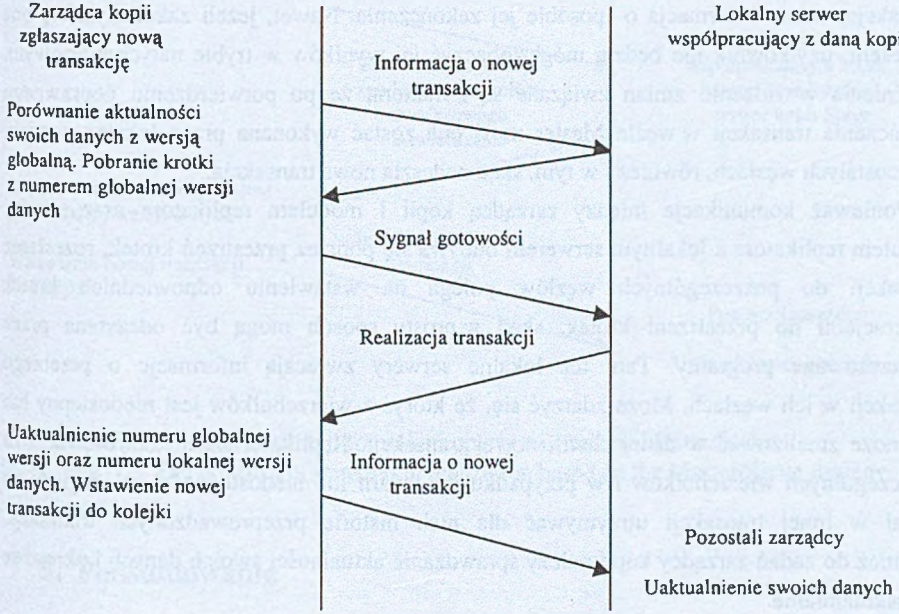
Realizacja w systemie eksperymentalnym

W przypadku systemu eksperymentalnego problem ten został rozwiązany z wykorzystaniem mechanizmów aktywnych baz danych. Kiedy w lokalnej bazie danych pojawia się operacja modyfikacji, lokalny system zarządzania bazą danych powiadamia o niej (za pomocą zdarzeń i wyzwalaczy) moduł replikatora, utrzymujący kolejkę wszystkich transakcji dotyczących powielonych danych. Replikator sprawdza, czy w jego kolejce nie znajdują się jakieś niezrealizowane operacje dla węzła, który zgłasza nową transakcję. Jeśli zaistnieje taka sytuacja, bieżąca transakcja musi zostać wycofana. W przeciwnym przypadku transakcja ta jest zatwierdzana w lokalnym węźle i dopisywana do kolejki replikatora. Z replikatorem współpracuje zarządca kopii, zadaniem którego jest uaktualnianie co jakiś czas stanu lokalnej bazy danych zgodnie z nadchodzącymi od replikatora zgłoszeniami.

Analiza pracy replikatora wykazała, że może on być wąskim gardłem w obsłudze kopii danych. Dlatego moduł ten usunięto z systemu, a jego funkcje zostały rozdzielone pomiędzy przestrzeń krotek i zarządców kopii. Kolejka zgłoszeń transakcji została umieszczona w przestrzeni krotek. Wszystkie transakcje odnoszące się do powielonych danych szeregowane są tam z odpowiednim numerem, w kolejności ich nadejścia. Wykorzystane zostały w związku z tym pojęcia numeru lokalnej i globalnej wersji powielonych danych. Jeżeli pojawi się lokalna transakcja dotycząca powielonych danych, sprawdzana jest zgodność numeru lokalnej wersji danego wierzchołka sieci z numerem globalnej wersji w całym systemie. W przypadku gdy nie są one zgodne, zarządca kopii musi najpierw zrealizować wszystkie transakcje znajdujące się w kolejce, które powodują różnice w wersjach danych. Na koniec sam wstawia swoją transakcję do kolejki. Aby uniknąć

jednoczesnego modyfikowania zawartości kolejki, każda zmiana musi być poprzedzona pobraniem krotki z numerem globalnej wersji systemu, co jest jednoznaczne z zablokowaniem dostępu do kolejki.

Analizując efektywność tej metody można zauważyć, że przy braku wystąpienia kolizji do realizacji bieżącej transakcji potrzebny jest tylko odczyt i zapis jednego komunikatu, będącego krotką z numerem globalnej wersji danych. Natomiast w przypadku istnienia niezerowej kolejki, dla danego serwera, czas realizacji transakcji zwiększa się o czas wykonania zaległych operacji.



Rys. 6. Protokół realizacji transakcji oparty na strategii Independent
Fig. 6. The protocol of transaction realization based on the Independent strategy

4.4. Aktualizacja asynchroniczna dla kopii typu Master/Slave

Aktualizacja ta oparta jest na podejściu, w którym możliwość zmiany, usuwania lub dodawania rekordów przysługuje użytkownikom jednej kopii (zwanej Master lub Primary), pozostałe natomiast służą tylko do odczytu. Ogranicza to znacznie autonomiczność lokalnej bazy danych, ale daje pewność utrzymania spójności wszystkich kopii danych. Potencjalnie każda kopia danych może być kopią Master przy założeniu, że w danej chwili w systemie jest tylko jedna kopia tego typu.

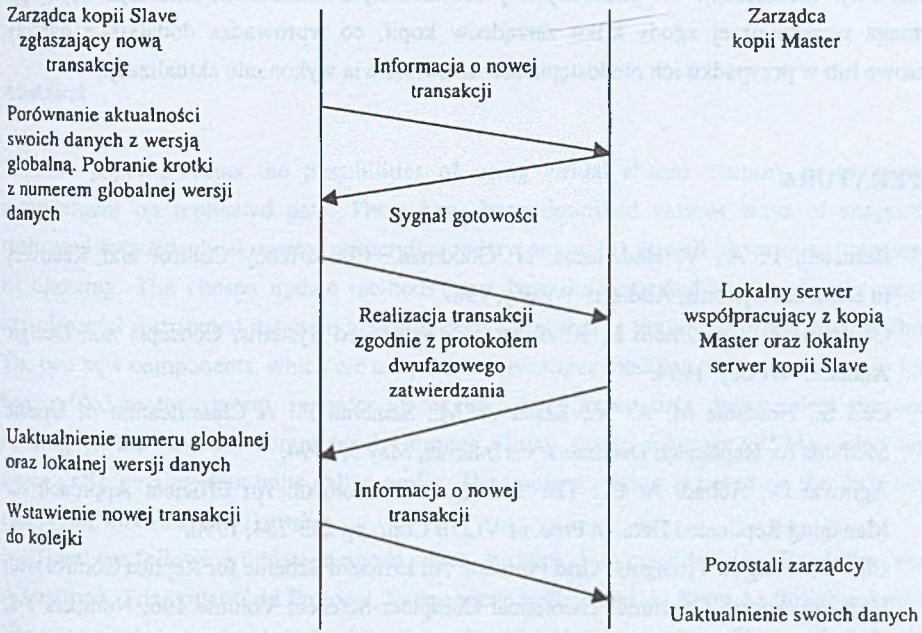
Implementacja w systemie eksperymentalnym

W celu realizacji tego sposobu prowadzenia zmian wykorzystano metodę ASAP (ang. *As Soon As Possible*). Przy realizacji transakcji zgodnie z tą metodą współpracują trzy moduły: zarządca kopii, lokalny serwer oraz replikator. Zarządca przyjmuje lokalną transakcję w węźle, w którym został uruchomiony i przekazuje ją za pośrednictwem replikatora do lokalnego serwera pracującego w węźle z kopią Master. Lokalny serwer realizuje otrzymaną transakcję i wynik jej wykonania zwraca replikatorowi. Jeżeli transakcja została zrealizowana pomyślnie, zmiany przenoszone są do pozostałych węzłów w czasie możliwie jak najkrótszym. Równocześnie do zarządcy kopii, od którego nadeszła nowa transakcja, wraca informacja o sposobie jej zakończenia. Nawet, jeżeli zakończyła się ona sukcesem, użytkownik nie będzie mógł zobaczyć jej wyników w trybie natychmiastowym. Opóźnienia w widzeniu zmian związane są z faktem, że po potwierdzeniu poprawnego zakończenia transakcji w węźle Master musi ona zostać wykonana przez lokalne serwery w pozostałych węzłach, również i w tym, skąd nadeszła nowa transakcja.

Ponieważ komunikacja między zarządcą kopii i modułem replikatora oraz między modułem replikatora a lokalnym serwerem odbywa się poprzez przestrzeń krotek, rozesłanie transakcji do poszczególnych węzłów polega na wstawieniu odpowiednich krotek z operacjami do przestrzeni krotek, skąd w prosty sposób mogą być odczytane przez zainteresowane programy. Tam też lokalne serwery zwracają informacje o przebiegu transakcji w ich węzłach. Może zdarzyć się, że któryś z wierzchołków jest niedostępny lub nie może zrealizować w danej chwili nowej transakcji. Replikator musi kontrolować stan poszczególnych wierzchołków i w przypadku ich awarii lub niedostępności ze względu na udział w innej transakcji utrzymywać dla nich historię przeprowadzonych transakcji. Również do zadań zarządcy kopii należy sprawdzanie aktualności swoich danych i okresowe ich uaktualnianie.

Omawiany rodzaj aktualizacji może odbywać się również bez udziału replikatora. Zarządca kopii, chcący zrealizować nową transakcję, może komunikować się bezpośrednio z zarządcą kopii Master. Po sprawdzeniu aktualności swoich danych, poprzez porównanie numeru lokalnej wersji swoich danych z numerem globalnej wersji danych, wykonuje on transakcję zgodnie z zasadami protokołu 2PC na dwóch kopiach – znajdującej się w jego węźle oraz na kopii Master. Pozostałe kopie są aktualizowane asynchronicznie przez swoich zarządców na podstawie historii znajdującej się w przestrzeni krotek. W tym przypadku liczba komunikatów niezbędnych do wykonania transakcji jest związana z ustaleniem aktualności danych lokalnej bazy danych (odczyt i zapis numeru globalnej wersji danych) oraz z przeprowadzeniem protokołu 2PC z lokalnym serwerem kopii Master (4 komunikaty – Prepare–To–Commit, odpowiedź Yes/No od dwóch serwerów i decyzja o sposobie

zakończenia transakcji). Rysunek 7 przedstawia protokół realizacji transakcji bez udziału replikatora.



Rys. 7. Protokół realizacji transakcji oparty na strategii Master/Slave
Fig. 7. The protocol of transaction realization based on the Master/Slave strategy

5. Podsumowanie

W pracy przeanalizowano właściwości wybranych algorytmów uaktualniania kopii danych w rozproszonej bazie danych przy zastosowaniu modelu Lindy (współdzielona pamięć i krotki) do komunikacji pomiędzy węzłami. Zastosowanie wirtualnie współdzielonej pamięci pozwoliło na redukcję liczby wymienianych komunikatów (w tym środowisku są to krotki wstawiane do przestrzeni krotek) pomiędzy N serwerami biorącymi udział w transakcji rozproszonej. Ograniczenia te dotyczą przede wszystkim liczby komunikatów z k operacjami należącymi do transakcji, która z $k*N$ zmniejsza się do k. Poza tym dla metod synchronicznej i bazującej na kworum – redukcje te obejmują również liczbę komunikatów związanych z przeprowadzeniem protokołu 2PC odpowiednio: z wartości $3*N(h)$ do $N(h)+2$.

Końcowym etapem analizy tych metod będzie seria testów, mająca na celu ocenę ich efektywności w środowisku wirtualnie współdzielonej pamięci. Wstępna ocena omawianych

metod wskazuje, że biorąc pod uwagę liczbę wymienianych informacji podczas realizacji transakcji rozproszonej najefektywniejsza wydaje się być metoda oparta na strategii aktualizacji niezależnej. W pozostałych przedstawionych metodach realizacja transakcji wymaga wcześniejszej zgody kilku zarządców kopii, co wprowadza dodatkowe narzuty czasowe lub w przypadku ich niedostępności uniemożliwia wykonanie aktualizacji.

LITERATURA

1. Bernstein P. A., V. Hadzilacos, N. Goodman.: Concurrency Control and Recovery in Database Systems, Addison–Wesley, 1987.
2. Coulouris G., Dollimore J., Kindberg T.: Distributed Systems, Concepts and Design, Addison–Wesley, 1994.
3. Ceri S., Houtsma M. A. W., Keller A. M., Samarati P.: A Classification of Update Methods for Replicated Databases. via Internet, May 5, 1994.
4. Agrawal D., Abbadi A. El.: The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. in Proc. of VLDB Conf. pp 243-254, 1990.
5. Cho C., Wang J.: Triangular Grid Protocol: An Efficient Scheme for Replica Control with Uniform Access Quorums. Theoretical Computer Science, Volume 196, Numbers 1-2, Apr 6,1998.
6. Schussel G.: Replication, The Next Generation of Distributed Database Technology, via Internet: http://www.dci.com/speakers/archive/replica.html
7. Bajerski P., Chłopek M., Harężlak K., Josiński H.: Environment for Distributed Database Management Algorithms Evaluation Based on Virtual Shared Memory. Proceedings of the 16th IASTED International Conference APPLIED INFORMATICS, held February 23–25, 1998, Garmisch–Partenkirchen, Germany.
8. Chłopek M., Harężlak K., Josiński H.: Implementacja podstawowych mechanizmów zarządzania rozproszoną bazą danych w eksperymentalnym systemie wykorzystującym model współdzielonej pamięci. IV Seminarium "Sieci Komputerowe". Zeszyty Naukowe Pol. Śl. s. Informatyka, z.32, Gliwice 1997.
9. Chłopek M., Harężlak K., Josiński H.: Sterowanie współbieżnym dostępem do danych podczas realizacji transakcji rozproszonych – rozwój eksperymentalnego systemu zarządzania rozproszoną bazą danych. V Seminarium "Sieci Komputerowe". Zeszyty Naukowe Pol. Śl., s. Informatyka, z.34, Gliwice 1998.
10. Paradise User's Guide and Reference Manual, Scientific Computing Associates, 1995
11. Virtual Shared Memory and the Paradise system for Distributed Computing – A technical White Paper for Scientific, April 20, 1999.

Recenzent: Dr inż. Arkadiusz Sochan

Wpłynęło do Redakcji 6 kwietnia 2001 r.

Abstract

This paper presents the possibilities of using virtual shared memory in transaction management on replicated data. There have been described various ways of supporting replicated data (identical copies, primary/secondary, snapshot) as well as various strategies of its updating. The chosen update methods have been implemented in the already created experimental distributed database management system that is presented in the chapter three. The two new components, which are a *replication manager* module and a *copy manager* have been added to the system, in order to maintain a replicated data. Independent processes running on the network communicate through virtual shared memory (VSM) called *tuple space* (TS), passing data units called *tuples*. The implementation is based on the distributed computing environment *Paradise*.

There are following update methods which are taken into consideration: Read One Write All method, Triangular Grid Protocol, Independent method and As Soon As Possible method. The ways of their implementation have been described in chapter four. Figures 2, 5, 6 and 7 show protocols of transaction realization based on methods mentioned above. The analysis of these protocols in the context of the tools used made it possible to reduce the number of messages exchanged between system modules.