ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1996

Seria: INFORMATYKA z. 30 Nr kol. 1315

Piotr BAJERSKI
Henryk JOSIŃSKI
Katarzyna STĄPOR

ZASTOSOWANIE SYSTEMU PARADISE DO TWORZENIA SYSTEMU ZARZĄDZANIA ROZPROSZONĄ BAZĄ DANYCH

Streszczenie. W pracy przedstawiono prototypowy system zarządzania rozproszoną bazą danych - moduły koordynatora transakcji rozproszonej, klienta oraz odpowiednich sprzęgów. Architektura systemu została oparta na systemie Paradise - modelu wieloprocesorowego komputera z wirtualnie współdzieloną pamięcią. Jako lokalne systemy bazy danych zostały wykorzystane serwery Ingres. Przedstawiono wyniki przeprowadzonych eksperymentów.

USAGE OF THE PARADISE-SYSTEM FOR BUILDING OF DISTRIBUTED DATABASE MANAGEMENT SYSTEM

Summary. The paper presents prototype distributed database management system (DDBMS) - coordinator of global distributed transaction, client and interface modules. These modules constitute application processor. Architecture of the system is based on Paradise system - the multiprocessor computer model with virtual shared memory. As local database management systems Ingres servers are used. These are data processors.

VERWENDUNG DES PARADISE-SYSTEMS ZUR BILDUNG EINES MANAGEMENTSYSTEMS VERTEILTER DATENBANK

Zusammenfassung. Im Artikel wurden Elemente eines prototypischen Managementsystems verteilter Datenbank: Modul des Koordinators, des Clients und der entsprechenden Schnittstellen dargestellt. Die Systemarchitektur ist auf dem Paradise-System - dem Modell eines Multiprozessor-Computers mit einem virtuell gemeinsamen Speicher gegründet worden. Als lokale Datenbanksysteme wurden die Ingres-Server verwendet. Der Artikel beinhaltet auch Ergebnisse durchgeführter Experimente.

1. Wstęp

Obecnie na rynku pojawiają się pierwsze komercyjne systemy rozproszonych baz danych. Są one odpowiedzią na różnorodne zapotrzebowania użytkowników. Szczególnie dotyczy to przypadków, w których występuje naturalne rozproszenie danych i aplikacji. Przez system rozproszonej bazy danych w opracowaniu będzie rozumiany zbiór węzłów połączonych siecią komunikacyjną. W węzłach tych znajdują się scentralizowane systemy baz danych lub interfejsy użytkownika, lub jedno i drugie.

Systemy rozproszonych baz danych są odpowiedzią na potrzeby klientów i wynikiem prowadzonych w wiodących ośrodkach naukowych prac badawczych. Zastosowanie systemu rozproszonej bazy danych w miejsce systemu scentralizowanego niesie następujące korzyści [1]:

- zmniejszenie czasu odpowiedzi systemu przez umieszczenie danych bliżej użytkownika,
- zwiększenie wydajności systemu,
- zwiększenie niezawodności (ang. reliability) i dostępności (ang. availability),
- integracja systemów baz danych pochodzących od różnych dostawców i o różnych modelach danych,
- umożliwienie dostępu do danych na wielu serwerach z zachowaniem w pewnym zakresie kontroli integralności danych na lokalnym serwerze,
- skalowalność systemu.

Należy jednak pamiętać o dodatkowych problemach, wiążących się ze zrealizowaniem bazy rozproszonej, jakie nie występują w bazie scentralizowanej. Należą do nich:

- bardziej złożona modyfikacja danych,
- większa możliwość wystąpienia niespójności danych,

- większe prawdopodobieństwo wystąpienia zakleszczeń (ang. deadlocks),
- dodatkowe narzuty związane z zarządzaniem bazą danych,
- większa złożoność odtwarzania stanu zgodnego bazy danych,
- większa złożoność kontroli dostępu do danych.

W systemie zarządzania rozproszoną bazą danych wyróżnia się dwie warstwy: sprzętową i programową. Warstwę sprzętową tworzy wiele komputerów, zwanych węzłami oraz sieć komunikacyjna lokalna lub rozległa. Różne topologie sieci wyznaczają różne drogi komunikacyjne między węzłami. W warstwie programowej można wyróżnić trzy moduły:

- procesor danych (ang. data processor), który jest odpowiedzialny za zarządzanie danymi znajdującymi się w danym węźle,
- procesor aplikacji (ang. application processor), który odpowiada za koordynację dostępu do danych w różnych węzłach przy użyciu słownika rozproszonej bazy danych (w tym m.in. za zapewnienie spójności replikacji, wygenerowanie planu realizacji zapytania w sposób rozproszony oraz nadzorowanie jego wykonania),
- oprogramowanie komunikacyjne (ang. communication software), które dostarcza podstawowe usługi do realizacji procesora aplikacji.

Prowadzi się również prace nad zwiększeniem wydajności serwerów baz danych poprzez powiązanie architektur wieloprocesorowych z rozproszeniem bazy danych.

Autorzy opracowania mieli na uwadze skonstruowanie oprogramowania tworzącego procesor aplikacji przy maksymalnym wykorzystaniu narzędzi programowych dostępnych aktualnie w Zakładzie Teorii Informatyki. Tak więc w roli procesora danych wykorzystano lokalne serwery systemu Ingres, zaś do komunikacji pomiędzy procesami w różnych węzłach użyto systemu Paradise, który umożliwia między innymi dostęp do relacyjnej bazy danych za pomocą instrukcji języka SQL zanurzonych w programie w języku C. Zaprojektowany procesor aplikacji powinien spełniać następujące funkcje:

- implementacja i eksperymentalne badanie protokołów zarządzania replikacjami,
- implementacja i eksperymentalne badanie protokołów wypełniania transakcji w rozproszonych systemach baz danych,
- dynamiczna rekonfiguracja systemu,
- badanie zakleszczeń w rozproszonych systemach baz danych,
- integracja różnych systemów zarządzania bazami danych.

2. Definicje podstawowych pojęć

Przez system rozproszonej bazy danych w opracowaniu będzie rozumiany zbiór węzłów połączonych siecią komunikacyjną [1, 2, 3]. Transakcją globalną będzie nazywana transakcja, w ramach której może być wykonanych wiele operacji w różnych węzłach sieci. Operacje w jednym węźle, w ramach jednej transakcji globalnej, będą nazywane podtransakcją lub transakcją lokalną. Relacje mogą ulec procesowi fragmentacji: poziomej, pionowej lub mieszanej. Poziomą fragmentację relacji tworzy podzbiór jej rekordów. Pionową fragmentację relacji stanowią jej wybrane kolumny, przy czym fragmentacje nie muszą być rozłączne. Fragmentacja mieszana jest kombinacją wyżej wymienionych. Określona jednostka danych może również być zapamiętana w wielu węzłach sieci (mogą wystąpić jej replikacje). Takie konkretne wystąpienie będzie nazywane jednostką fizyczną. Wszystkie kopie danej jednostki będą określane mianem jednostki logicznej. Rozróżnienie to pociąga za sobą rozróżnienie blokad. Blokada jednostki fizycznej będzie nazywana blokadą fizyczną, a blokada jednostki logicznej będzie nazywana blokadą logiczną.

3. Narzędzia programowe

Architektura procesora aplikacji została oparta na systemie Paradise - modelu wieloprocesorowego komputera z wirtualnie współdzieloną pamięcią. W roli procesorów danych wykorzystano scentralizowane systemy baz danych Ingres.

3.1. Ogólna charakterystyka systemu Paradise

System Paradise stanowi środowisko równoległego i rozproszonego programowania, które wykorzystuje model wirtualnej pamięci współdzielonej (ang. VSM - Virtual Shared Memory). Model ten tworzy wiele przestrzeni danych (krotek), które stanowią wspólną pamięć dla wielu niezależnych programów. Programy takie mają dostęp do danych (współdzielą je) w tych przestrzeniach krotek. Wszystkie programy "połączone" poprzez taką wspólną pamięć dzieloną tworzą jedną, wspólną aplikację. Dzięki temu, że każda z takich przestrzeni krotek może być trwała (tzn. zdeponowana w pamięci zewnętrznej), poszczególne programy korzystające z takiej przestrzeni krotek mogą być wykonywane w różnym czasie i w tym sensie są od siebie niezależne. Programy te mogą być uruchamiane w komputerach o różnej architekturze dzięki odpowiednim procedurom konwersji (własność pełnej przeźroczystości). Aplikacje w systemie Paradise mogą posiadać struk-

rurę komunikacji typu klient-serwer, gdzie jeden z programów stanowi proces serwera, zaś wiele procesów (programów) klienckich posyła żądania do procesu serwera poprzez wspólną pamięć.

System Paradise jest kontrolowany przez proces serwera, dostarczający usług procesom klienckim, które mogą być uruchomione na komputerach innych niż komputer z procesem serwera. Komenda *paradise*, która może mieć wiele opcji, startuje proces serwera, co z kolei powoduje wykonanie następujących zadań:

- utworzenie przestrzeni krotek *rootts* i tymczasowej *tmpts*,
- umieszczenie krotek w przestrzeni głównej *rootts* (włączając w to krotkę z deskryptorem do przestrzeni tymczasowej, która jest przestrzenią ogólnie dostępną),
- wystartowanie serwera deskryptorów i zarejestrowanie deskryptorów o ograniczonych przywilejach do przestrzeni *rootts* i *tmpts*.

W systemie Paradise istnieje możliwość tworzenia kopii (ang. checkpoint) przestrzeni krotek do zbiorów dyskowych. Głównym jego celem jest zapewnienie możliwości odtworzenia danych z przestrzeni krotek w przypadku awarii serwera, nienormalnego zakończenia transakcji, itp. Kontroluje tę usługę zasób *chkpntinterval*. Domyślnie serwer testuje wszystkie przestrzenie krotek, którymi zarządza: usuwa nietrwałe przestrzenie krotek, dla których brak otwartych deskryptorów, zapisuje do pliku przestrzenie krotek, które nie wykazują aktywności przez dłuższy okres czasu. Odnosi się to do przestrzeni trwałych oraz nietrwałych, dla których nie upłynął jeszcze czas ustalony dla procesu automatycznego usuwania nietrwałych przestrzeni krotek (*purgeinterval*). Zasoby *swapinterval* oraz *purgeinterval* specyfikują przedział czasu, przez jaki przestrzeń krotek musi pozostawać nieaktywna, aby mógł rozpocząć się proces tzw. *swapingu* (automatycznego zapisu do pamięci zewnętrznej przestrzeni krotek) oraz proces *purgingu* (automatyczne usunięcie nietrwałych przestrzeni krotek). Domyślne parametry dla zasobów *swapinterval* i *purgeinterval* wynoszą odpowiednio 60 i 1440 min. Parametr *xactiontimeout* umożliwia określenie czasu, po którym nastąpi automatyczne wycofanie transakcji (domyślnie 5 min).

W systemie Paradise operacje *xaction* oraz *commit* definiują transakcję, tzn. taki zbiór operacji, z których wszystkie albo żadna z nich muszą zostać wykonane. W przypadku niezdefiniowania takiej transakcji mogłoby dojść do niespójności w przestrzeni krotek. (np. w przypadku awarii komputera - pobranie z przestrzeni krotki, która stanowi część opisu pewnego obiektu i niezapisanie tej krotki z nowymi wartościami, podczas gdy inne krotki opisujące ten obiekt zostają zapisane). Transakcje w systemie Paradise dotyczą instrukcji zmieniających zawartość przestrzeni krotek: *in, out, flush, inp*. Instrukcje te służą do wprowadzania bądź wyprowadzania pojedynczych krotek lub ich zbioru do przestrzeni krotek. Wycofanie rozpoczętej transakcji może również nastąpić, jeśli nie

zostanie ona zatwierdzona w określonym przedziale czasu (domyślnie 300 s). Możliwe jest ustawienie okresu, po którym transakcja zostanie automatycznie wycofana za pomocą operacji *xaction @ ts_handle (PARADISE_TIMEOUT(n))*, gdzie *ts_handle* stanowi deskryptor otwartej przestrzeni krotek, zaś n - pożądany czas oczekiwania w sekundach. Wycofanie transakcji powoduje, że wyniki działania wszystkich następnych operacji (aż do operacji *commit* lub *cancel*) związanych z tym samym deskryptorem przestrzeni krotek również zostają unieważnione. W przypadku wycofania transakcji automatycznie wskutek upłynięcia określonego czasu, jeśli operacja *in* nie może zostać z pewnych względów wykonana (np. brak odpowiedniej krotki w przestrzeni krotek), następujące po niej operacje w obrębie danej biegnącej transakcji również skończą się niepowodzeniem.

4. Zanurzony język SQL

W celu uzyskanie dostępu do zawartości rozproszonej bazy danych posłużono się instrukcjami języka SQL (ang. Structured Query Language), zanurzonymi w programie w języku C (ang. Embedded SQL/C, ESQL/C). Poniżej omówiono zasady stosowania konstrukcji języka ESQL/C.

4.1. Wprowadzenie

System Ingres umożliwia zanurzenie instrukcji języka SQL w programach pisanych w języku trzeciej generacji (3GL) [4, 5, 6, 7]. Każda zanurzona instrukcja SQL musi być w programie poprzedzona frazą EXEC SQL. Wymiana danych pomiędzy programem a bazą danych w ogólnym przypadku odbywa się za pomocą predefiniowanych struktur danych, współpracujących ze zdefiniowanym przez programistę buforem danych lub zmiennymi programowymi. Do nawiązania połączenia z bazą służy instrukcja CONNECT, natomiast zamknięcie połączenia ma miejsce po wykonaniu instrukcji DISCONNECT. Istnieje możliwość nawiązania połączeń z wieloma bazami danych równocześnie. Pojedyncze połączenie nosi nazwę *sesji*. Wyboru bieżącej sesji dokonuje się za pomocą instrukcji SET_SQL (SESSION = <numer sesji>).

Jeśli dana instrukcja SQL będzie wielokrotnie wykonywana w programie, to w celu uniknięcia powtórzenia realizacji operacji wstępnych, związanych z wykonaniem instrukcji (analiza składniowa, optymalizacja), należy poprzedzić ją frazą REPEATED.

Instrukcja SQL może zostać umieszczona bezpośrednio w kodzie programu (statycznie formułowana instrukcja SQL), może również zostać sformułowana przez użytkownika dopiero w trakcie pracy programu (dynamicznie formułowana instrukcja SQL).

Generacja kodu wynikowego programu odbywa się w trzech etapach:

1) prekompilacja (zastąpienie instrukcji SQL przez konstrukcje języka 3GL),
2) kompilacja,
3) konsolidacja.

4.2. Instrukcja SELECT w ESQL/C - poziom zbioru rekordów

Instrukcja SELECT pozwala na wyszukanie zbioru rekordów spełniających określone kryteria. Ponieważ instrukcje języków 3GL nie są w stanie równocześnie przetworzyć całego zbioru, konieczne jest sekwencyjne przetwarzanie kolejnych rekordów. Istnieją dwie metody, pozwalające na pracę z pojedynczym rekordem zbioru:

- kursor,
- pętla SELECT.

Kursor jest wskaźnikiem na pojedynczy rekord zbioru wyszukanego przez instrukcję SELECT i umieszczonego w buforze.

Etapy wykorzystania kursora:

1) deklaracja kursora (DECLARE CURSOR),
2) otwarcie bufora związanego z kursorem przez wykonanie instrukcji SELECT i wypełnienie bufora wynikowym zbiorem rekordów (OPEN),
3) ustawienie kursora na pierwszym rekordzie w buforze (FETCH),
4) wykonanie operacji należącej do grupy DML (języka manipulowania danymi, ang. Data Manipulation Language) na pojedynczym rekordzie,
5) przemieszczenie kursora do następnego rekordu w buforze (FETCH),
6) powtórzenie etapów 4 i 5 dla każdego rekordu w buforze,
7) zamknięcie bufora (CLOSE).

Aby umożliwić modyfikację wybranych kolumn rekordów znajdujących się w buforze kursora, należy w deklaracji kursora użyć frazy FOR UPDATE OF. Modyfikacja pojedynczego rekordu następuje w etapie 4. Zapisanie dokonanych zmian w bazie danych może nastąpić natychmiast (tryb pracy DIRECT) lub dopiero po zamknięciu bufora (tryb pracy DEFERRED). Wybór trybu pracy ma miejsce przy deklaracji kursora.

Pętla SELECT umożliwia wykonanie instrukcji języka 3GL, zawartych w jej wnętrzu, dla każdego rekordu należącego do zbioru.

Ogólna konstrukcja pętli:

```
EXEC SQL BEGIN;
   <instrukcje języka 3GL>
   [EXEC SQL ENDSELECT;]
   [<instrukcje języka 3GL>]
EXEC SQL END;
```

Instrukcja ENDSELECT powoduje natychmiastowe opuszczenie pętli. Jest jedyną konstrukcją języka SQL dopuszczalną wewnątrz pętli. Pętla jest związana bezpośrednio z poprzedzającą ją instrukcją SQL i nie może być od niej oddzielona innymi instrukcjami. Aby w trakcie przetwarzania zbioru rekordów móc sięgnąć do bazy danych (usuwając lub modyfikując rekord bądź używając wielu tabel równocześnie), należy zastosować kursor.

4.3. Wymiana danych pomiędzy programem a bazą danych - poziom pojedynczego rekordu

Wymiana danych pomiędzy programem a bazą danych na poziomie pojedynczego rekordu bazy może odbywać się jednym z dwóch sposobów:

- za pomocą zmiennych programowych,
- przez bufor danych.

Zmienne programowe użyte w instrukcji SQL muszą zostać zadeklarowane. Część deklaracyjna programu znajduje się pomiędzy następującymi instrukcjami:

```
EXEC SQL BEGIN DECLARE SECTION;

EXEC SQL END DECLARE SECTION;
```

Każda zmienna użyta wewnątrz instrukcji SQL musi być poprzedzona znakiem ':', aby mogła zostać odróżniona od obiektów bazy danych. Z pojedynczą kolumną relacji mogą zostać związane dwie zmienne: zmienna główna (ang. host variable) i zmienna pomocnicza (ang. indicator variable). Zmienna główna reprezentuje wartość w kolumnie lub wartość porównywaną z wartością w kolumnie. Zmienna pomocnicza spełnia jedną z trzech funkcji:

1) Sprawdzenie, czy zmiennej głównej przypisano wartość pustą NULL. Zmienna pomocnicza zawiera wtedy wartość -1. Z wartości pustej korzysta się na przykład przy tworzeniu nowego rekordu, kiedy nie wypełnia się jawnie wszystkich jego kolumn.

2) Wstawienie wartości pustej NULL do kolumny tabeli. Zmiennej pomocniczej należy przed wykonaniem odpowiedniej instrukcji SQL przypisać wartość -1.
3) Sprawdzenie, czy wartość przypisana zmiennej głównej została zniekształcona wskutek różnicy długości zmiennej głównej i długości wartości przypisywanej. Zmienna pomocnicza zawiera wtedy rzeczywistą długość przypisywanej wartości.

Sposób zapisu obu rodzajów zmiennych jest następujący:

:<zmienna główna>:<zmienna pomocnicza>

Przykład:

```
EXEC SQL BEGIN DECLARE SECTION;
   int v1, v2, v3;
   short null1, null2, null3;
EXEC SQL END DECLARE SECTION;

main()
{
  EXEC SQL INSERT INTO tab1
     VALUES (:v1:null1, :v2:null2, :v3:null3);
  /* jeśli zmiennym pomocniczym null1, null2, null3 zostanie wpierw
     przypisana wartość -1, to do kolumn im odpowiadających zostanie
     wprowadzona wartość NULL (o ile jest dopuszczalna) */

  EXEC SQL SELECT * FROM tab1
     INTO :v1:null1, :v2:null2, :v3:null3);
  /* jeśli kolumny wyszukanego rekordu zawierają wartości NULL, to
     odpowiednim zmiennym pomocniczym zostanie przypisana wartość -1 */
}
```

Zamiast zmiennych programowych można użyć bufora danych o wielkości i zawartości modyfikowalnej w trakcie pracy programu. Bufor obejmuje pojedynczy rekord bazy danych. Jest bardziej uniwersalny od zmiennych - może zawierać wartości dowolnych typów, co ma szczególne znaczenie dla dynamicznie formułowanej instrukcji SQL, gdyż zarówno liczba, jak i typ jej kolumn wynikowych są na ogół dla programisty nieznane.

4.4. Dynamicznie formułowana instrukcja SQL

Instrukcja SQL formułowana przez użytkownika w trakcie pracy programu (wprowadzana z klawiatury lub pliku tekstowego) zostanie wykonana w trzech etapach:

1) analiza tekstu instrukcji,
2) jeśli jest to tekst instrukcji SELECT, to przypisanie predefiniowanej struktury *sqlda* do kolumn wynikowych i wypełnienie jej informacjami o tych kolumnach,
3) wykonanie kodu instrukcji.

Istnieją dwa sposoby wykonania instrukcji: CREATE, INSERT, MODIFY, UPDATE, DROP, DELETE:

1) Za pomocą instrukcji PREPARE przetwarza wprowadzony tekst instrukcji na kod przypisywany zmiennej programowej; kod ten zostanie następnie wykonany przez instrukcję EXECUTE:

```
EXEC SQL PREPARE kod FROM :instr;
EXEC SQL EXECUTE kod;
/* instr - tekst wprowadzonej instrukcji SQL,
   kod - zmienna programowa zawierająca przetworzony kod instrukcji */
```

Ten sposób jest polecany, gdy wprowadzona instrukcja ma wielokrotnie zostać wykonana w programie, gdyż tekst instrukcji podlega tylko jednokrotnemu przetwarzaniu.

2) Za pomocą instrukcji EXECUTE IMMEDIATE, która wykorzystuje bezpośrednio wprowadzony tekst instrukcji, wykonując ją:

```
EXEC SQL EXECUTE IMMEDIATE :instr;
```

Dla instrukcji SELECT etapy przetwarzania jej tekstu i wypełniania struktury *sqlda* mogą zostać zrealizowane za pomocą instrukcji PREPARE i DESCRIBE:

```
EXEC SQL PREPARE kod FROM :instr;
EXEC SQL DESCRIBE kod INTO :deskr;
/* deskr - wskaźnik do struktury sqlda */
```

lub łącząc oba etapy w instrukcji PREPARE z użyciem wskaźnika do struktury *sqlda*:

```
EXEC SQL PREPARE INTO :deskr FROM :instr;
/* lub: EXEC SQL PREPARE USING DESCRIPTOR :deskr FROM :instr; */
```

Istnieją dwa sposoby wykonania przetworzonego kodu instrukcji: przez zastosowanie kursora lub za pomocą instrukcji EXECUTE IMMEDIATE z użyciem wskaźnika do struktury *sqlda* i pętli SELECT:

```
EXEC SQL EXECUTE IMMEDIATE :instr USING DESCRIPTOR :deskr;
   EXEC SQL BEGIN;
      <instrukcje języka 3GL>
      [EXEC SQL ENDSELECT;]
      [<instrukcje języka 3GL>]
   EXEC SQL END;
```

Dla rozstrzygnięcia, czy wprowadzona instrukcja jest instrukcją SELECT, należy przeanalizować wprowadzony tekst lub przetestować wartość pola *sqld* struktury *sqlda* po jego wypełnieniu: sqld > 0 ⇒ SELECT.

4.4.1. Struktura *sqlda*

Struktura *sqlda* (*SQL Description Area*) znajduje zastosowanie, jeśli nieznana jest liczba i typ kolumn rekordu zbioru wynikowego dynamicznie formułowanej instrukcji SELECT. Program zapisuje w strukturze informacje o nazwach i typach danych poszczególnych kolumn, aby za jej pośrednictwem umieścić zawartość pojedynczego rekordu w przewidzianych przez programistę obiektach programu (buforze danych lub zmiennych programowych).

Deklaracja struktury *sqlda*:

```
#define IISQ_MAX_COLS 300
/* maksymalna liczba rekordów w zbiorze wynikowym */
typedef struct sqlvar_ {
   short sqltype;
   /* kod typu danych w kolumnie */
   short sqllen;
   /* długość kolumny; 0 dla typu DATE */
   char *sqldata;
   /* wskaźnik na obiekt programu, któremu zostanie przypisana
      wartość w kolumnie */
   short *sqlind;
   /* wskaźnik na obiekt programu, któremu zostanie przypisany
      ekwiwalent wartości NULL */
   struct {
      short sqlnamel;
      char sqlnamec[ 34 ];
   } sqlname;
   /* nazwa kolumny */
} IISQLVAR;
/* struktura opisująca pojedynczą kolumnę rekordu w zbiorze wynikowym */

typedef struct sqda_ {
   char sqldaid[ 8 ];
   /* "SQLDA   " */
   long sqldabc;
   /* rozmiar struktury sqlda */
   short sqln;
   /* maksymalna liczba kolumn objętych przez strukturę */
   short sqld;
   /* liczba kolumn rekordu zbioru wynikowego danej instrukcji SELECT */
   IISQLVAR sqlvar[ IISQ_MAX_COLS ];
} IISQLDA;
/* struktura sqlda */
```

```
/* kody typów wstawiane przez program do pól struktury sqlvar;
   dla kolumny dopuszczającej wystąpienie wartości NULL kod typu będzie
   liczbą przeciwną do odpowiedniej wartości podanej poniżej */
#define IISQ_DTE_TYPE  3 /* DATE */
#define IISQ_MNY_TYPE  5 /* MONEY */
#define IISQ_CHA_TYPE 20 /* CHAR */
#define IISQ_VCH_TYPE 21 /* VARCHAR */
#define IISQ_INT_TYPE 30 /* INTEGER */
#define IISQ_FLT_TYPE 31 /* FLOAT */
#define IISQ_TBL_TYPE 52 /* typ danych używany w konstrukcjach
                            wykorzystujących formularze ekranowe */
#define IISQ_DTE_LEN  25
/* długość wartości typu DATE */

/* rozmiary przydzielonych obszarów pamięci */
#define IISQDA_HEAD_SIZE 16
/* nagłówek struktury sqlda */
#define IISQDA_VAR_SIZE  sizeof (IISQLVAR)
/* obszar pamięci dla pojedynczej struktury sqlvar */
```

Powyższa deklaracja znajdzie się w kodzie programu po wykonaniu instrukcji:

```
EXEC SQL INCLUDE sqlda;
```

Obszar pamięci dla struktury *sqlda* można przydzielić statycznie lub dynamicznie.

Przykład: `/* statyczny przydział pamięci */`

```
EXEC SQL INCLUDE sqlda;
IISQLDA _deskr;
IISQLDA *deskr = &_deskr;

deskr->sqln = max_lb_kol;

EXEC SQL DESCRIBE kod INTO :deskr;
```

Przykład: `/* dynamiczny przydział pamięci */`

```
EXEC SQL INCLUDE sqlda;
IISQLDA *deskr;

deskr = ((IISQLDA *) calloc (1, IISQDA_HEAD_SIZE +
            lb_kol * IISQDA_VAR_SIZE);
if (deskr == (IISQLDA *) 0)
   /* reakcja na błąd zarządzania pamięcią */

deskr->sqln = max_lb_kol;

EXEC SQL DESCRIBE kod INTO :deskr;
```

```
<instrukcje języków 3GL lub SQL>

free ((char *) deskr);
/* zwolnienie obszaru pamięci */
```

Instrukcja DESCRIBE lub PREPARE (→ punkt 4.4) wypełnia pole *sqld* oraz pola: *sqltype*, *sqllen* (zawartość obu pól może wymagać modyfikacji (→ punkt 4.4.2)) i *sqlname* w szeregu struktur *sqlvar*, z których każda jest przyporządkowana pojedynczej kolumnie. Programista musi określić wartość dla pola *sqln* oraz dla każdej kolumny przypisać polom *sqldata*, *sqlind* adresy odpowiednich obiektów programu.

4.4.2. Interpretacja wartości różnych typów danych

Informacje o typie i długości danych każdej kolumny umożliwiają prawidłowe umieszczenie wartości wynikowych w buforze danych lub innych obiektach programu. Aby te wartości mogły być właściwie interpretowane przez instrukcje języka C, należy dla każdego typu danych systemu Ingres określić odpowiedni typ danych języka C. Zależności pomiędzy typami numerycznymi są trywialne. Typy MONEY i DATE nie posiadają swoich bezpośrednich odpowiedników w języku C i dlatego zapisy w strukturze *sqlda*, dokonane przez instrukcje DESCRIBE lub PREPARE i dotyczące tych typów, powinny zostać w programie zmodyfikowane. Wartość typu MONEY powinna być interpretowana jako *double*:

```
sqltype ← IISQ_FLT_TYPE
sqllen  ← sizeof (double)
```

Wartość typu DATE jest łańcuchem, którego długość jest określona przez stałą IISQ_DTE_LEN, stąd:

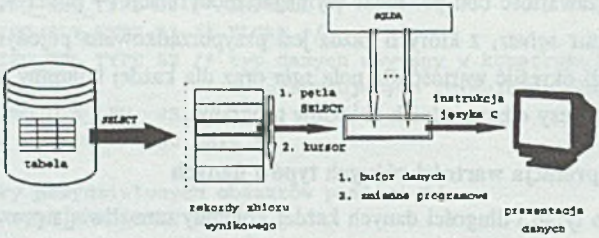
```
sqltype ← IISQ_CHA_TYPE
sqllen  ← IISQ_DTE_LEN
(instrukcja DESCRIBE umieszcza dla typu DATE w polu sqllen wartość 0)
```

Instrukcja DESCRIBE umieszcza następujące wartości dla typów alfanumerycznych:

```
char(n):        sqltype = ±IISQ_CHA_TYPE
                sqllen  = n
varchar(n):     sqltype = ±IISQ_VCH_TYPE
                sqllen  = n
```

4.4.3. Struktura programu z dynamicznie formułowaną instrukcją SELECT

Ogólny schemat realizacji dynamicznie formułowanej instrukcji SELECT pokazano na rysunku 1.



Rys. 1. Schemat realizacji dynamicznie formułowanej instrukcji SELECT
Fig. 1. Schema of execution of dynamically formulated SQL-command

Zarys algorytmu realizacji instrukcji SELECT z zastosowaniem kursora:

```
EXEC SQL CONNECT testdb;
/* nawiązanie połączenia z bazą danych testdb */
EXEC SQL PREPARE kod FROM :instr;
/* analiza tekstu instrukcji */
EXEC SQL DESCRIBE kod INTO :deskr;
/* wypełnienie struktury sqlda */
if (deskr->sqld > 0)
{
  /* wystąpiła instrukcja SELECT */
  EXEC SQL DECLARE kursor CURSOR FOR kod;
  /* deklaracja kursora */
  EXEC SQL OPEN kursor;
  /* wykonanie instrukcji SELECT skojarzonej z kursorem */
  /* korekta typu i długości danych */
  /* powiązanie kolumn z obiektami programu przez strukturę sqlda */
  lb_rek = 0;
  /* lb_rek - liczba rekordów analizowanych w buforze kursora */
  do
  {
    EXEC SQL FETCH kursor USING DESCRIPTOR :deskr;
    /* przesunięcie kursora do następnego rekordu */
    lb_rek += sqlca.sqlerrd[ 2 ];
    /* sqlca.sqlerrd[ 2 ] - liczba rekordów objętych zasięgiem FETCH */
```

```
    if (sqlca.sqlcode == 0)
    {
     /* instrukcja FETCH wykonana pomyślnie;
        wyprowadzenie danych */
    }
  } while (sqlca.sqlcode == 0);
  EXEC SQL CLOSE kursor;
  /* zamknięcie bufora kursora */
  /* sqlca.sqlcode == 100 - przeanalizowano cały bufor kursora */
}
EXEC SQL DISCONNECT;
/* zamknięcie połączenia z bazą danych */
```

4.5. Transakcje w ESQL/C

Instrukcja SET AUTOCOMMIT określa zasięg pojedynczej transakcji dla danej sesji. Transakcja rozpoczyna się w chwili nawiązania połączenia z bazą danych. Użycie frazy OFF w instrukcji SET AUTOCOMMIT (fraza domyślna) powoduje, że transakcja zostaje uznana za zakończoną:

- po wykonaniu instrukcji COMMIT,
- po zamknięciu połączenia z bazą danych (instrukcja DISCONNECT).

Natomiast użycie frazy ON ogranicza zasięg transakcji do pojedynczej instrukcji SQL. Dla instrukcji dynamicznie formułowanej transakcja zostanie uznana za zakończoną po wykonaniu instrukcji EXECUTE.

Do programowego wycofania transakcji służy instrukcja ROLLBACK.

4.6. Diagnostyka błędów w ESQL/C

Do najważniejszych sposobów diagnostyki błędów należy analiza struktury *sqlca* oraz stosowanie instrukcji WHENEVER. W strukturze *sqlca* (*SQL Communication Area*) program zapisuje informacje o przebiegu wykonania bieżącej instrukcji SQL. Wybrane pola struktury:

sqlcabc – rozmiar struktury;

sqlcode – wartość przypisywana po wykonaniu instrukcji; jej interpretacja:

- 0 – pomyślne wykonanie instrukcji;
- <0 – wystąpił błąd o kodzie *sqlcode*;
- >0 – instrukcja została wykonana pomyślnie, ale wystąpiła dodatkowa okoliczność, np. *sqlcode* = 100 - jedna z poniższych instrukcji: DELETE, FETCH, INSERT, SELECT, UPDATE, MODIFY, COPY, CREATE INDEX, CREATE AS SELECT nie objęła żadnego rekordu;

sqlerrm – komunikat o błędzie odpowiadającym ujemnej wartości *sqlcode*;

sqlerrd – wektor, w którym między innymi: *sqlerrd*[2] - liczba rekordów objętych przez jedną z poniższych instrukcji: DELETE, FETCH, INSERT, SELECT, UPDATE, MODIFY, COPY, CREATE INDEX, CREATE AS SELECT;

sqlwarn – struktura sygnalizująca wystąpienie ostrzeżenia przez umieszczenie symbolu 'W' w odpowiednim polu:
sqlwarn0 = 'W' - wystąpiło ostrzeżenie; odpowiednie pole $sqlwarn_i$ zawiera 'W' (np. *sqlwarn2* = 'W' - przed obliczeniem jednej z funkcji agregujących usunięto z kolumny wszystkie wartości NULL).

Deklaracja struktury *sqlca* znajdzie się w programie po wykonaniu instrukcji:

```
EXEC SQL INCLUDE sqlca;
```

Instrukcja WHENEVER określa sposób reakcji programu na wystąpienie określonego błędu podczas realizacji instrukcji SQL. Program taki musi zawierać deklarację struktury *sqlca*. Ogólna postać instrukcji WHENEVER:

```
EXEC SQL WHENEVER <warunek> <reakcja>;
```

gdzie <warunek>:

sqlerror ≡ *sqlcode* < 0;
sqlwarning ≡ *sqlwarn0* = 'W';
not found ≡ *sqlcode* = 100;
sqlmessage ≡ *sqlcode* = 700;

<reakcja>:

goto <znacznik> – realizacja programu będzie kontynuowana od miejsca wskazanego przez <znacznik>;
stop – realizacja programu zostanie natychmiast przerwana, a połączenie z bazą danych zakończone;
continue – realizacja programu będzie kontynuowana bez konsekwencji;
call <procedura języka 3GL> – zostanie wywołana <procedura języka 3GL>.

Instrukcja WHENEVER <warunek> jest aktywna od miejsca swojego wystąpienia w kodzie programu do kolejnej instrukcji WHENEVER z tym samym <warunkiem> lub do końca programu, jeśli jest ostatnią instrukcją WHENEVER <warunek>.

5. Opis systemu

Skontruowane przez autorów opracowania oprogramowanie pracuje na grupie stacji roboczych Sun połączonych siecią lokalną Ethernet (rys. 2). Stacje te posiadają różną architekturę i różnią się mocą obliczeniową.

Rys. 2. Konfiguracja sieci lokalnej
Fig. 2. The configuration of the local area network

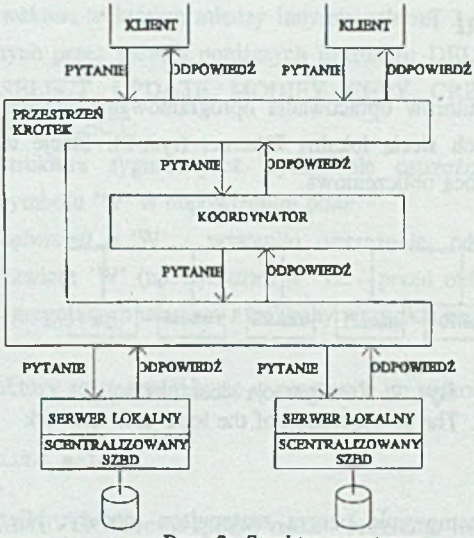
Skonstruowane oprogramowanie tworzą następujące moduły: klienta, koordynatora i lokalnego serwera (rys. 3). W systemie może być dowolna liczba programów klienta, tylko jeden koordynator i po jednym lokalnym serwerze na każdą instalację scentralizowanego systemu bazy danych. Poszczególne moduły są uruchamiane jako niezależne programy z powłoki. Wykonujące się na różnych komputerach programy komunikują się za pomocą przestrzeni krotek. Przyjęto, że w środowisku rozproszonym jednostką granulacji przy blokowaniu jest plik. Koordynator jest scentralizowany i może być uruchomiony na dowolnym komputerze w sieci. Program klienta jest uruchamiany na lokalnym komputerze użytkownika.

5.1. Moduł klienta

Moduł klienta stanowi pośrednictwo między użytkownikiem a koordynatorem.

Funkcje modułu klienta:

- pobieranie od użytkownika polecenia w języku SQL,
- prezentacja wyników wykonanych operacji,
- żądanie inicjacji globalnej transakcji,
- żądanie wypełnienia lub wycofania globalnej transakcji,
- opcjonalnie jawne żądanie logicznej blokady,



Rys. 3. Struktura systemu
Fig. 3. The structure of the system

- wysłanie operacji do wykonania do koordynatora,
- odebranie wyników wykonania operacji od koordynatora.

5.2. Moduł koordynatora

Koordynator zarządza wykonaniem operacji zleconych przez użytkownika w środowisku rozproszonym. Założono, że koordynator zna położenie wszystkich plików, co jest wykorzystywane przy przydzielaniu operacji do lokalnych serwerów.

Funkcje modułu koordynatora:

- kierowanie odebranych od klientów operacji do wykonania do odpowiednich lokalnych serwerów,
- odbieranie od lokalnych serwerów wyników wykonania operacji i odsyłanie ich do klientów,
- zarządzanie globalnymi transakcjami,
- zarządzanie logicznymi blokadami,
- rozpoczynanie, wypełnianie i wycofywanie globalnych transakcji.

5.3. Moduł lokalnego serwera

Lokalny serwer stanowi sprzęg między koordynatorem a scentralizowanym (lokalnym) systemem bazy danych. Program lokalnego serwera musi być uruchomiony na komputerze, na którym znajduje się lokalny serwer bazy danych, z którym on współpracuje.

Funkcje lokalnego serwera:

- pobieranie operacji od koordynatora,
- wykonywanie pobranych operacji (zapisanych w dynamicznym języku SQL lub wewnętrznych operacji systemu),
- odsyłanie wyników i kodu błędu operacji do koordynatora,
- zarządzanie lokalnymi blokadami,
- rozpoczynanie, wypełnianie i wycofywanie lokalnych transakcji.

5.4. Opis wiadomości przesyłanych pomiędzy modułami systemu

Podstawowy cykl wykonania operacji zadanej przez użytkownika można podzielić na następujące fazy:

Faza 1 - przesłanie od klienta do koordynatora krotki z żądaniem wykonania operacji,

Faza 2 - przesłanie od koordynatora do lokalnego serwera lub lokalnych serwerów krotki z żądaniem wykonania operacji,

Faza 3 - przesłanie od lokalnego serwera lub lokalnych serwerów do koordynatora krotki z wynikiem wykonania operacji,

Faza 4 - przesłanie od koordynatora do klienta krotki z wynikiem wykonania operacji.

Przy użyciu replikacji i dwu- lub trójfazowego wypełniania transakcji pojawiają się dodatkowe fazy.

Postać krotek przesyłanych w fazie 1:

- etykieta identyfikująca klasę krotki,
- numer klienta, który wysłał operację,
- numer operacji w ramach sesji,
- kod operacji - numeryczny identyfikator operacji,
- tekst operacji - pełny tekst operacji w języku SQL podanej przez użytkownika.

Krotki są identyfikowane przez: etykietę, numer klienta i numer operacji.

Postać krotek przesyłanych w fazie 2:

- etykieta identyfikująca klasę krotki,
- numer lokalnego serwera, dla którego jest przeznaczona dana operacja,
- numer klienta, który wysłał operację,
- numer operacji w ramach sesji danego klienta,
- kod operacji - numeryczny identyfikator operacji,
- tekst operacji - pełny tekst operacji w języku SQL podanej przez użytkownika lub argumenty dla wewnętrznych operacji systemu.

Krotki są identyfikowane przez: etykietę, numer lokalnego serwera, numer klienta i numer operacji.

Postać krotek przesyłanych w fazie 3:

- etykieta identyfikująca klasę krotki,
- numer lokalnego serwera, który wykonał daną operację,
- numer klienta, który wysłał operację,
- numer operacji w ramach sesji danego klienta,
- kod błędu wykonanej operacji,
- wyniki wykonania operacji (pola te są buforami, których zawartość jest interpretowana w zależności od operacji; bufory te są głównie wykorzystywane przy operacji SELECT do przesłania pobranych danych).

Krotki są identyfikowane przez: etykietę, numer lokalnego serwera, numer klienta i numer operacji.

Postać krotek przesyłanych w fazie 4:

- etykieta identyfikująca klasę krotki,
- numer klienta, który wysłał operację,
- numer operacji w ramach sesji danego klienta,
- kod błędu wykonanej operacji,
- wyniki wykonania operacji (pola te są buforami, których zawartość jest interpretowana w zależności od operacji; bufory te są głównie wykorzystywane przy operacji SELECT do przesłania pobranych danych).

Krotki są identyfikowane przez: etykietę, numer klienta i numer operacji.

6. Podsumowanie

Do chwili obecnej został napisany i przetestowany kod procesora aplikacji w swojej podstawowej postaci. System realizuje zadane przez użytkownika operacje SQL należące do grupy DML poprzez przygotowanie planu realizacji zapytania oraz nadzorowanie jego wykonania. Stworzona struktura koordynacji zapewnia przepływ komunikatów pomiędzy procesami.

Planuje się wykorzystanie systemu do następujących celów:

- badanie różnych protokołów blokowania jednostek logicznych,
- testowanie różnych wariantów dwu- i trójfazowych protokołów wypełniania transakcji,
- optymalizacja realizacji zapytania w sposób rozproszony.

Prowadzone są również prace nad rozproszeniem samego koordynatora.

Przedstawione opracowanie stanowi podsumowanie prac autorów nad algorytmami wykorzystywanymi w systemach zarządzania rozproszoną bazą danych i ma na celu sformułowanie dalszych zamierzeń badawczych.

LITERATURA

[1] Ceri S., Pelagatti G.: Distributed Databases. Principles and Systems, McGraw Hill, 1986.

[2] Elmasri R., Navathe B.: Fundamentals of Database Systems, Benjamin Cumings, 1989.

[3] Bernstein P. at. al: Concurrency Control and Recovery in Database Systems, Addison-Wesley, 1987.

[4] Petkovič D.: INGRES. Das relationale Datenbanksystem mit Knowledge-Base und Object-Base, Addison-Wesley, 1992.

[5] ASK/INGRES Technical Communications: INGRES/ESQL Companion Guide for C.

[6] ASK/INGRES Technical Communications: INGRES/SQL Reference Manual.

[7] ASK/INGRES Technical Communications: INGRES/Embedded Open SQL Forms Reference Manual.

[8] ASK/INGRES Technical Communications: INGRES Database Administrator's Guide.

Recenzent: Dr hab. inż. Adam Mrózek

Wpłynęło do Redakcji 3 stycznia 1996 r.

Abstract

The paper presents prototype distributed database management system (DDBMS) - coordinator of global distributed transaction, client and interface modules. These modules constitute application processor. Architecture of the system is based on Paradise system - the multiprocessor computer model with virtual shared memory. As local database management systems Ingres servers are used. These are data processors. Several variants of two-phase locking and two-phase commit protocols where implemented in coordinator module.