ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1995

Seria: INFORMATYKA z. 29 Nr kol. 1306

Piotr BAJERSKI
Mirosław CHŁOPEK
Henryk JOSIŃSKI
Stanisław KOZIELSKI
Krzysztof LACH

WYKORZYSTANIE SYSTEMU LINDA DO ROZWIĄZYWANIA ZADAŃ WYSZUKIWANIA WEDŁUG MODELU STEROWANIA PRZEPŁYWEM ARGUMENTÓW

Streszczenie. W pracy przedstawiono system umożliwiający równoległe wykonywanie zadań wyszukiwania. Całość zrealizowano według modelu sterowania przepływem argumentów przy wykorzystaniu systemu LINDA. Przedstawiono wyniki przeprowadzonych eksperymentów.

APPLYING THE LINDA SYSTEM TO SOLVING DATA RETRIEVAL PROBLEMS BASED ON THE ARGUMENT FLOW DRIVEN MODEL

Summary. The paper presents a description of a parallel data retrieval system. The system implementation follows the Argument Flow Driven Model and is created using the Linda System. The results of performed experiments are shown.

Pracę wykonano w ramach grantu KBN nr 3 P406 011 04.

AUSNUTZUNG DES LINDA-SYSTEMS ZUR LÖSUNG DER AUSSUCHAUFGABEN NACH DEM MODELL DER STEUERUNG DURCH DEN ARGUMENTENFLUß

Zusammenfassung. Im Artikel wurde ein System dargestellt, das eine parallele Ausführung der Aussuchaufgaben ermöglicht. Das Ganze wurde nach dem Modell der Steuerung durch den Argumentenfluß und mit Ausnutzung des LINDA-Systems realisiert. Der Artikel beinhaltet auch Ergebnisse durchgeführter Experimente.

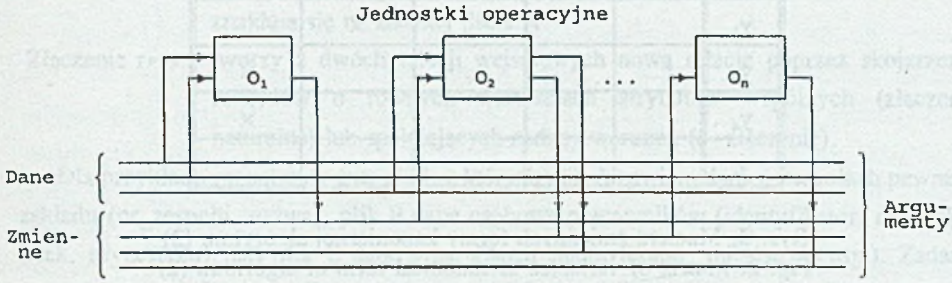
1. Wstęp

Sterowanie przepływem argumentów jest koncepcją organizowania obliczeń rozproszonych w zbiorze jednostek przetwarzających (procesorów, stacji sieci komputerowej). System LINDA tworzy w lokalnej sieci komputerowej środowisko do przetwarzania równoległego i rozproszonego. W pracy zaproponowano wykorzystanie tych mechanizmów do wykonywania zadań wyszukiwania w bazach danych. Przyjęto, że zadania wyszukiwania będą zapisywane za pomocą operatorów algebry relacji. Rozpatrzono kilka wariantów realizacji modelu sterowania przepływem argumentów w systemie LINDA. Przedstawiono organizację programów realizujących wybrany model w sieci stacji roboczych Sun. Przeprowadzono szereg eksperymentów umożliwiających ocenę wpływu na przyspieszenie obliczeń takich czynników, jak: liczba i rodzaj wykorzystywanych stacji sieci, rozmieszczenie plików bazy czy sposób rozpraszania obliczeń. Analizowano też przyspieszenia uzyskiwane przy rozpraszaniu ciągu zadań wyszukiwania.

2. Sterowanie przepływem argumentów w rozwiązywaniu zadań wyszukiwania

Idea sterowania przepływem argumentów została zaproponowana w pracach S. Węgrzyna [6, 7]. Zakłada się w niej, że rozpatrywane zadanie obliczeniowe może zostać zapisane w postaci zbioru operacji (O_1, O_2.. O_n), które mogą być realizowane w niezależnych jednostkach operacyjnych (procesory, stacje sieci). Układ takich jednostek powiązanych

liniami umożliwiającymi przesyłanie argumentów obejmujących wartości wejściowe (dane) oraz wartości pośrednie i wyjściowe (zmienne) przedstawia rys. 1.



Rys. 1. Ogólna struktura systemu sterowanego przepływem argumentów (według [6])
Fig. 1. General structure of an argument flow driven system

Przyjmuje się, że każda jednostka może rozpocząć wykonywanie zleconej jej operacji po pojawieniu się na jej wejściach wszystkich potrzebnych argumentów. Synteza struktury systemu z rys. 1 musi uwzględniać wzajemne powiązania między operacjami wyróżnionymi w rozważanym zadaniu obliczeniowym. W pracy [6] rozpatrzono klasę algorytmów w postaci kanonicznej, tj. sekwencji instrukcji podstawienia. Na przykład zadanie wyznaczenia wartości wyrażenia

$$z = (f_1(x_1) - f_2(x_2)) * (x_1 + x_2) \tag{1}$$

może zostać przedstawione następująco:

$$\begin{aligned} y_1 &= f_1(x_1) \\ y_2 &= f_2(x_2) \\ y_3 &= x_1 + x_2 \\ y_4 &= y_1 - y_2 \\ z &= y_4 * y_3 \end{aligned} \tag{2}$$

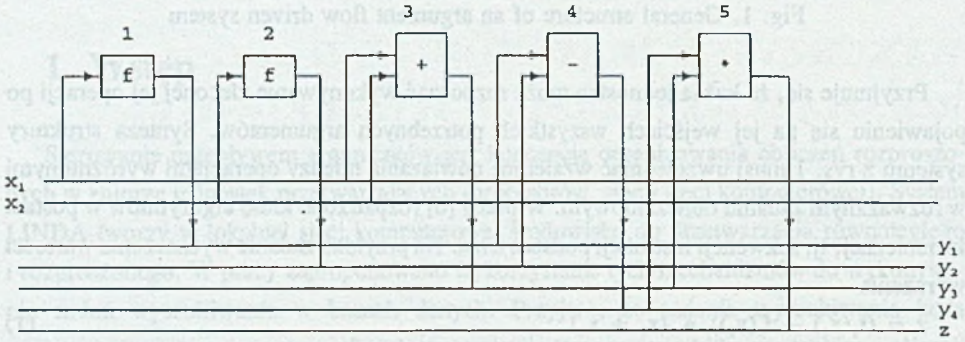
Powyższy zapis przedstawia kanoniczną postać algorytmu wyznaczającego wartość z. Zależności pomiędzy rezultatami otrzymanego ciągu operacji można przedstawić w postaci tzw. macierzy zmiennych formy kanonicznej [6]. Dla rozważanego przykładu macierz tę przedstawia rys. 2.

W pracy [6] zauważono, że jeśli macierz zmiennych formy kanonicznej jest macierzą trójkątną górną bez elementów na przekątnej głównej, to określa ona bezpośrednio strukturę powiązań jednostek operacyjnych systemu sterowanego przepływem argumentów, który realizuje dany algorytm. Dla rozważanego przypadku system ten przedstawiono na rys. 3.

Jedną z dziedzin, w której można zastosować sterowanie przepływem argumentów, jest organizowanie obliczeń w procesach wyszukiwania danych. Złożone zadania wyszukiwania,

	y_1	y_2	y_3	y_4	z
y_1				×	
y_2				×	
y_3					×
y_4					×
z					

Rys. 2. Macierz zmiennych formy kanonicznej algorytmu (2)
Fig. 2. Matrix of variables of canonical form of algorithm (2)



Rys. 3. Struktura systemu sterowanego przepływem argumentów dla wyrażenia (2)
Fig. 3. Structure of argument flow driven system for expression (2)

dotyczące kilku plików danych, są zwykle bardzo czasochłonne. Stąd też zainteresowanie przyspieszaniem tych operacji poprzez zrównoleglanie obliczeń. Głównym problemem, który należy w tym celu rozwiązać, jest wskazanie metody wyróżniania w programach wyszukiwania takich operacji (lub fragmentów programu), które mogą być realizowane równolegle. Rozwiązanie tego problemu zależy przede wszystkim od rodzaju języka zastosowanego do zapisu zadań wyszukiwania. W systemach zarządzania bazami danych stosuje się wiele języków [5], tworzących klasy o odrębnych własnościach. Wybór języka musi uwzględniać typowe czasy wykonywania elementarnych operacji wyszukiwania danych. Dla prostych operacji, dotyczących pojedynczych rekordów, zwykle realizowanych w krótkim czasie, sens ich zrównoleglania jest wątpliwy, bowiem nakład czasu traconego na rozpraszanie tych operacji przewyższy zyski powstałe w trakcie ich równoległej realizacji.

Analiza spotykanych języków pokazuje, że najbardziej interesującą pod tym względem klasą są języki korzystające z operatorów algebry relacji [3]. Do podstawowych operatorów należą:

Selekcja $\sigma_w(r)$: tworzy z relacji r nową relację o takiej samej strukturze, składającą się z tych rekordów relacji r, które spełniają zadany warunek w.

Projekcja $\pi_X(r)$: tworzy nową relację, składającą się z tych atrybutów relacji r, które znajdują się na zadanej liście X.

Złączenie $r \bowtie s$: tworzy z dwóch relacji wejściowych nową relację poprzez skojarzenie rekordów o równych wartościach atrybutów wspólnych (złączenie naturalne) lub spełniających zadany warunek (θ -złączenie).

Dla przykładu rozpatrzmy trzy pliki, z których plik A zawiera dane o zespołach pewnego zakładu (nr_zespołu, nazwa), plik B dane osobowe pracowników (identyfikator, nazwisko, wiek, nr_zespołu), zaś plik C dane o zarobkach (identyfikator, miesiąc, pensja). Zadanie wyszukania nazwisk i pensji w miesiącu wrześniu pracowników zespołu Analiz, którzy nie przekroczyli 35 lat, możemy zapisać następująco:

$$r = \pi_{nazwisko,\ pensja}(((\sigma_{nazwa='Analizy'}(\text{plik A})) \bowtie (\sigma_{wiek<=35}(\text{plik B})) \bowtie (\sigma_{miesiac=9}(\text{plik C}))) \quad (3)$$

Zadanie to może również być zapisane jako ciąg następujących instrukcji podstawienia:

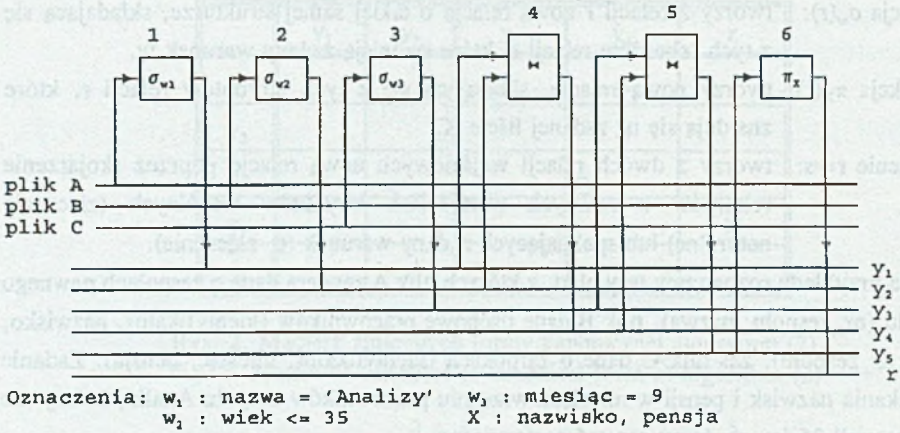
$$\begin{aligned} y_1 &= \sigma_{nazwa='Analizy'}(\text{plik A}) \\ y_2 &= \sigma_{wiek<=35}(\text{plik B}) \\ y_3 &= \sigma_{miesiac=9}(\text{plik C}) \\ y_4 &= y_1 \bowtie y_2 \\ y_5 &= y_3 \bowtie y_4 \\ r &= \pi_{nazwisko,\ pensja}(y_5) \end{aligned} \quad (4)$$

Powyższy zapis przedstawia kanoniczną formę algorytmu rozwiązującego rozpatrywane zadanie. Macierz zmiennych formy kanonicznej dla tego zadania przedstawiono na rys. 4.

	y_1	y_2	y_3	y_4	y_5	r
y_1				×		
y_2				×		
y_3					×	
y_4					×	
y_5						×
r						

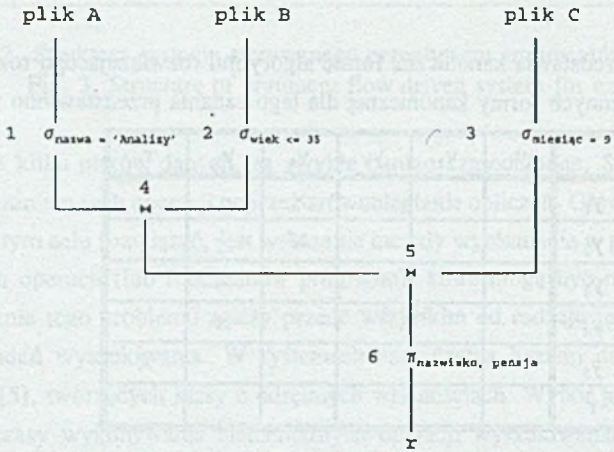
Rys. 4. Macierz zmiennych formy kanonicznej zadania wyszukiwania (4)
Fig. 4. Matrix of variables of canonical form of data retrieving problem (4)

Macierz ta spełnia warunki określone w pracy [6] (macierz trójkątna górna, bez przekątnej głównej), co pozwala bezpośrednio zaprojektować strukturę systemu sterowanego przepływem argumentów, zapewniającą realizację powyższego algorytmu. Przedstawia ją rys. 5.



Rys. 5. System sterowany przepływem argumentów, rozwiązujący zadanie (4)
Fig. 5. The argument flow driven system solving the problem (4)

Dla łatwego wyróżnienia operacji, które w procesie wyszukiwania mogą być wykonywane równolegle, wyrażenie zapisane za pomocą algebry relacji można przedstawić w postaci drzewa obliczeń (rys. 6).



Rys. 6. Drzewo obliczeń do zadania wyszukiwania (4)
Fig. 6. The computational tree for data retrieval problem (4)

Struktura drzewa obliczeń odpowiada dokładnie strukturze powiązań jednostek operacyjnych systemu sterowanego przepływem argumentów.

3. Krótka charakterystyka LINDY

System Linda, wykorzystywany do zaimplementowania systemu sterowanego przepływem argumentów (SSPA), jest realizacją modelu Lindy - modelu współdzielonej pamięci asocjacyjnej, nazywanej przestrzenią krotek. Linda udostępnia operacje umożliwiające wstawianie i pobieranie danych z tej przestrzeni. Dane te, nazywane krotkami, składają się z ciągów pól. Wyróżnia się dwa rodzaje krotek: bierne i aktywne. Bierne są ciągami wartości. Aktywne krotki zawierają wywołania funkcji, które są realizowane jako osobne procesy, równoległe względem siebie i innych procesów realizowanych w środowisku Lindy.

Do zrealizowania obliczeń równoległych potrzebne są dwa języki: język pozwalający zapisać rozwiązywany problem i język umożliwiający tworzenie nowych procesów i ich koordynację. W niniejszej pracy pierwszym językiem jest język C, drugim język C-Linda, będący implementacją modelu Lindy. Można powiedzieć, że język C-Linda jest zanurzony w języku C przez wprowadzenie do niego operacji wstawiania i pobierania krotek do/z przestrzeni krotek.

Operacja **in** próbuje pobrać z przestrzeni krotek krotkę, która pasuje do wzorca, podanego jako argument tej operacji. Dopasowanie zachodzi, gdy: liczba pól w krotce i we wzorcu jest taka sama, zachodzi zgodność typów odpowiadających sobie pól i wartości znajdujące się we wzorcu równają się wartościom w krotce. Jeżeli nie znaleziono pasującej krotki, wykonanie operacji jest blokowane do momentu pojawienia się takowej. Gdy wzorzec można dopasować do kilku krotek, jedna z nich jest wybierana niedeterministycznie. Pasująca krotka jest usuwana z przestrzeni krotek i za zmienne we wzorcu są podstawiane wartości pól krotki. Operacja **rd** działa analogicznie do operacji **in**, z tym wyjątkiem, że odczytana krotka nie jest usuwana z przestrzeni krotek. Operacja **out** umieszcza krotkę podaną jako argument w przestrzeni krotek. Jeżeli którekolwiek pole w operacji **out** zawiera wyrażenie, jest ono obliczane przed wstawieniem krotki do przestrzeni krotek.

Operacja **eval** tworzy aktywną krotkę. Dla każdego pola zawierającego wywołanie funkcji tworzony jest na jednym z komputerów biorących udział w obliczeniach proces obliczania wartości tej funkcji. Procesy te są niezależne i wykonują się równolegle. Operacja **eval** kończy się po utworzeniu tych procesów.

Przykłady operacji języka C-Linda:

int i;

eval("funct", f()) - utworzenie nowego procesu, wykonującego funcję *f()*,

out("p", 2) - wstawienie krotki do przestrzeni krotek,

in("p", 2) - pobranie krotki z przestrzeni krotek,

rd("p", 2) – odczytanie krotki z przestrzeni krotek,

in("p", ?i) – pobranie krotki i podstawienie pod zmienną *i* wartości 2 (dla krotki utworzonej powyższą operacją **out**).

W niniejszej pracy wykorzystano system Linda i język C-Linda w wersji v 2.5.2 firmy SCIENTIFIC COMPUTING ASSOCIATES INC.

4. Organizacja programu interpretera zapytań

4.1. Struktura ogólna interpretera zapytań

W projekcie interpretera przyjęto następujące założenia. Zapis każdego z pytań za pomocą operatorów algebry relacji umieszczony jest w osobnym pliku. Ich nazwy wypisane są w pliku będącym argumentem wywołania programu tak, że w jednej linii znajduje się nazwa jednego pliku.

Plik zawierający pytanie dzieli się na trzy części:
- zapis pytania w algebrze relacji,
- nazwa pliku z wynikami,
- nazwy plików z danymi.

W systemie rozróżniane są następujące operacje (w nawiasach podano etykiety użyte do ich oznaczania w programie):
- złączenie (*zlacz*),
- selekcja (*sel*),
- projekcja (*proj*),
- złączenie połączone z projekcją (*pzlacz*),
- selekcja połączona z projekcją (*psel*).

Pojedyncza operacja algebry relacji jest zapisywana w jednej linii, w następującym formacie:

zlacz <plik_wejściowy_1> <plik_wejściowy_2> <plik_wyjściowy> <warunek_złączenia>
<metoda_złączenia>

sel <plik_wejściowy> <plik_wyjściowy> <warunek_selekcji>

proj <plik_wejściowy> <plik_wyjściowy> <lista_atrybutów>

pzlacz <plik_wejściowy_1> <plik_wejściowy_2> <plik_wyjściowy> <warunek_złączenia>
<lista_atrybutów> <metoda_złączenia>

psel <plik_wejściowy> <plik_wyjściowy> <warunek_selekcji> <lista_atrybutów>

Kolejność zapisu operacji nie ma wpływu na ich następstwo w czasie wykonania. Jest ono wyznaczane przez zależności między wynikami pośrednimi a argumentami operacji.

Nazwa pliku wynikowego znajduje się w osobnej linii i jest poprzedzona znakiem '#'. Pliki dane na wejściu są wypisane w kolejnych liniach. Zapisy użytych w eksperymentach pytań zostały przedstawione w rozdziale 5.2.

Dla rozpraszania operacji algebry relacji dokonano trzech różnych implementacji SSPA:

1 - uruchomienie wszystkich gotowych operacji,

2 - uruchomienie gotowych operacji w liczbie nie większej od liczby użytych procesorów,

3 - symulacja komputera przepływowego.

W każdej z implementacji na początku działania programu nadzorca (proces utworzony z funkcji *real_main()* na lokalnym komputerze) wczytuje z plików zadania (pytania). Każde z pytań jest przechowywane w dynamicznie tworzonym rekordzie. Rekordy te są zorganizowane w listę. Rekord zawiera tablicę, której wiersze opisują operacje algebry relacji. Poza argumentami operacji, zapisanymi w pytaniu, przechowywane są informacje o przebiegu wykonania zadania, takie jak dostępność argumentów i flaga wykonania operacji. Definicje tych struktur znajdują się na rys. 7.

We wszystkich użytych rozwiązaniach proces nadzorcy pobiera rezultaty z przestrzeni krotek i dopasowuje je do operacji. Realizacja tych kroków ma miejsce w funkcji *Engine()*, gdzie w pętli nadzorca przegląda wszystkie operacje algebry relacji według listy rekordów, sprawdzając każdą z pozycji tablic opisujących zapytania. Jeżeli znajdzie operację, której flagi dostępności argumentów zostały ustawione, a flaga wykonania - nie, przystępuje do jej uruchomienia. W komputerze przepływowym tworzona jest krotka opisująca tę operację, a w dwóch pozostałych rozwiązaniach uruchamiany jest nowy proces.

Wykonawca w komputerze przepływowym jest uniwersalny, tzn. może wykonać dowolną z operacji zaimplementowanego SSPA. Pobiera on w pętli krotki opisujące operacje gotowe do wykonania, w instrukcji wyboru wywołuje lokalnie funkcję realizującą zadaną operację SSPA, a po jej zakończeniu wyprowadza wynik do przestrzeni krotek. Procesy wykonawców są tworzone na początku działania programu i pozostają aktywne aż do wykonania wszystkich operacji SSPA. Ich liczba odpowiada liczbie dostępnych procesorów. Uproszczony kod nadzorcy dla tego rozwiązania przedstawiono na rys. 7, a kod wykonawcy na rys. 8.

W rozwiązaniu 1 nadzorca tworzy nowy proces dla każdej operacji SSPA natychmiast po skompletowaniu jej argumentów. W rozwiązaniu 2 utworzenie nowego procesu jest uzależnione od dostępności wolnego procesora. Wymaga to zliczania przez proces nadzorcy uruchomionych i zakończonych procesów. Ponieważ do funkcji, będącej argumentem operacji eval, nie można przekazać w wywołaniu argumentów będących tablicami, funkcje realizujące operacje algebry relacji obudowano funkcjami pośredniczącymi. Nazwy tych funkcji tworzy się z nazwy operacji algebry relacji z przedrostkiem *"P_"*.

```
typedef struct {     /* struktura przechowująca opis operacji algebry relacji */
  int  typOper;              /* kod operacji */
  char plwe1[], plwe2[];          /* argumenty wejściowe */
  short  dostplwe1, dostplwe2;     /* flagi dostępności argumentów wejściowych */
  char plwy[];               /* nazwa pliku zawierającego rezultat */
  char argop[4][];                /* argumenty operacji */
  short  wysloper;                /* flaga wysłania operacji do wykonania */
} toperPA;

struct tzapyt {              /* struktura opisująca pojedyncze zapytanie */
  toperPA  *operacje;             /* tablica opisująca operacje algebry relacji */
  int  il_oper;              /* ilość pozycji w tablicy */
  char plwy[];               /* nazwa pliku zawierającego rezultat */
  struct tzapyt *next;
};

real_main(ilarg, arg) {
  struct tzapyt  *head = NULL;

  nworkers = atoi(*(arg + 1));     /* utworzenie wykonawców */
  for (i = 0; i < nworkers; i++) eval("worker", worker());

  head = CzytPytania(*(arg + 2));      /* odczytanie zadań (pytań) z plików */
  start_timer();             /* wystartowanie pomiaru czasu */
  Engine(head);              /* uruchomienie funkcji zarządzającej */
  print_times();             /* wydrukowanie uzyskanych czasów wykonania zadań */

  for (i = 0; i < nworkers; i++) /* wysłanie "trujących pigułek" */
    out("oper", -1, "":, "":, "":, "":, "":, "":, "":);
}

Engine(struct tzapyt  *head) {
  toperPA  *wskoper;

  while (/* są operacje algebry relacji do wykonania */) {

    /* iteruj po wszystkich operacjach, podstawiając ich wskaźnik za wskoper  */
      if (wskoper->wysloper == FALSE && wskoper->dostplwe1 == TRUE &&
        wskoper->dostplwe2 == TRUE) {
        out("oper", wskoper->typOper, wskoper->plwe1:, wskoper->plwe2:, wskoper->plwy:,
            wskoper->argop[0]:, wskoper->argop[1]:, wskoper->argop[2]:, wskoper->argop[3]:);
        wskoper->wysloper = TRUE;
      };

    in("result", ?namepl:);  /* pobranie wyniku */

    /* ustaw flagę dostępności we wszystkich operacjach wykorzystujących namepl */
  }
}
```

Rys. 7. Kod procesu nadzorcy dla implementacji SSPA przy użyciu symulacji komputera przepływowego

Fig. 7. The master process code for AFDS implementation based on the idea of the dataflow computer

```
worker() {

  do {
    in("oper", ?oper, ? plwe1:dlwe1, ? plwe2:dlwe2, ? plwy:dlwy, ? argop[0]:dlarg0, ? argop[1]:dlarg1,
      ? argop[2]:dlarg2, ? argop[3]:dlarg3);

    if (oper < 0) break;        /* pobrano "trującą pigułkę" */

    switch (oper) {         /* wywołanie funkcji realizującej podaną operację */
      case PSEL:
      case SEL:   sel(plwe1, plwy, argop[0], argop[1]); break;
      case PROJ:  proj(plwe1, plwe2, plwy, argop[0]); break;
      case ZLACZ: zlacz(plwe1, plwe2, plwy, argop[0], argop[1]); break;
      case PZLACZ: pzlacz(plwe1, plwe2, plwy, argop[0], argop[1], argop[2]); break;
    };
    out("result", plwy:);       /* wyprowadzenie rezultatu */
  } while (1);
}
```

Rys. 8. Kod procesu wykonawcy dla implementacji SSPA przy użyciu symulacji komputera przepływowego

Fig. 8. The worker process code for AFDS implementation based on the idea of the dataflow computer

Przykład takiej funkcji, o nazwie *P_proj()*, przedstawiono na rys. 9. Funkcje pośredniczące na podstawie identyfikatora operacji przekazanego jako parametr ich wywołania pobierają krotkę opisującą jej argumenty i uruchamiają zadaną operację poprzez lokalne wywołanie funkcji. Po zakończeniu operacji wyprowadzają wynik do przestrzeni krotek.

Dla rozpraszania całych pytań zaimplementowano SSPA w postaci komputera przepływowego. Implementacja komputera przepływowego zapewnia dynamiczne równoważenie obciążenia w sytuacji zmiennego obciążenia procesorów i różnych ich architektur. Składają się na to głównie trzy czynniki: w jednym procesorze jest realizowany jeden proces wykonawcy, każdy z wykonawców może pobrać i wykonać dowolne zadanie, jest więcej zadań niż wykonawców. Tak więc szybsze i mniej obciążone komputery wykonają więcej zadań, co prowadzi do skrócenia sumarycznego czasu obliczeń. Analogicznie działa równoważenie przy uruchamianiu gotowych operacji na nie zajętych procesorach (wersja 2).

W przypadku uruchamiania wszystkich gotowych operacji równoważenie obciążenia zasadza się na mechanizmach systemu Linda, przydzielającego tworzony proces na najszybszy i najmniej obciążony procesor. System kieruje się przy tym mocą obliczeniową procesora i jego aktualnym obciążeniem.

Wykorzystane metody dynamicznego równoważenia obciążenia dają dobre wyniki, jeśli pojedyncze podzadania mają zbliżoną złożoność i żadne z nich nie jest porównywalne z całym zadaniem. Jeżeli ten warunek nie jest spełniony, zachodzi niebezpieczeństwo, że najbardziej złożone zadanie może zostać pobrane do wykonania jako ostatnie. Wtedy wszystkie procesory oprócz tego, który pobrał ostatnie zadanie, będą czekać bezczynnie na

```
real_main(ilarg, arg) {
  struct tzapyt  *head = NULL;

  head = CzytPytania(*(arg + 2));      /* odczytanie zadań (pytań) z plików */
  start_timer();                       /* wystartowanie pomiaru czasu */
  Engine(head);                        /* uruchomienie funkcji zarządzającej */
  print_times();                       /* wydrukowanie uzyskanych czasów wykonania zadań */
}

Engine(struct tzapyt  *head) {
  toperPA  *wskoper;

  op_num = 0;
  while (/* są operacje algebry relacji do wykonania */) {
    /* iteruj po wszystkich operacjach, podstawiając ich wskaźnik za wskoper  */
    if (wskoper->wysloper == FALSE && wskoper->dostplwe1 == TRUE &&
       wskoper->dostplwe2 == TRUE) {
      out("oper", op_num, wskoper->plwe1:, wskoper->plwe2:, wskoper->plwy:, wskoper->argop[0]:,
          wskoper->argop[1]:, wskoper->argop[2]:, wskoper->argop[3]:);

        switch (wskoper->typOper) {    /* utworzenie procesu operacji */
          case PSEL:
          case SEL: eval("wop", P_sel(op_num++)); break;
          case PROJ: eval("wop", P_proj(op_num++)); break;
          case ZLACZ: eval("wop", P_zlacz(op_num++)); break;
          case PZLACZ: eval("wop", P_pzlacz(op_num++)); break;
        };
      wskoper->wysloper = TRUE;
      };
    };


    in("result", ?namepl:);   /* pobranie wyniku */

    /* ustaw flagę dostępności we wszystkich operacjach wykorzystujących namepl */
  }

  return 0;
}

P_proj(int op_num) { /* przyklad funkcji realizującej operacje */

  in("oper", op_num, ? plwe1:dlwe1, ? plwe2:dlwe2, ? plwy:dlwy, ? argop[0]:dlarg0, ? argop[1]:dlarg1,
     ? argop[2]:dlarg2, ? argop[3]:dlarg3);
  proj(plwe1, plwe2, plwy, argop[0]);
  out("result", plwy:);
}
```

Rys. 9. Kod procesu nadzorcy dla implementacji SSPA z uruchamianiem wszystkich gotowych operacji

Fig. 9. The master process code for AFDS implementation starting all ready operations

zakończenie obliczeń. Prowadzi to do zmniejszenia wartości współczynnika przyśpieszenia i efektywności wykorzystania komputerów.

4.2. Organizacja modułu realizującego selekcję i projekcję

Zadaniem modułu selekcji i projekcji jest pobieranie danych (rekordów) ze zbioru, przetwarzanie danych, tj. wykonywanie operacji selekcji i projekcji na rekordach odczytywanych do bufora i zapisywanie wyników przetwarzania danych do zbioru.

Moduł selekcji i projekcji w zależności od doboru parametrów wejściowych realizuje operacje selekcji, projekcji lub selekcji z projekcją. Parametrami wejściowymi modułu są:

- nazwa zbioru zawierającego dane (rekordy),
- nazwa zbioru wynikowego powstającego w wyniku przetwarzania rekordów,
- zapis warunku selekcji w konwencji nawiasowej,
- zapis zadania projekcji.

Rodzaj operacji do wykonania określany jest na podstawie zawartości dwóch ostatnich parametrów wejściowych modułu, tj. warunku selekcji i zadania projekcji.

Działanie modułu projekcji i selekcji można podzielić na trzy fazy:

1. Przygotowanie struktur danych.
2. Przekształcenie zapisu warunku selekcji na zapis beznawiasowy.
3. Wykonanie operacji selekcji i projekcji.

Przekształcenie zapisu warunku selekcji na zapis beznawiasowy zgodny z zasadami Odwrotnej Notacji Polskiej umożliwia wyznaczenie wartości logicznej wyrażenia przy wykorzystaniu stosu wyliczeniowego.

Operacja selekcji i projekcji wykonywana jest na danych odczytywanych do bufora ze zbioru. Z buforem związane są dwa wskaźniki: wskaźnik odczytu i zapisu. Wskaźnik odczytu wskazuje na rekord aktualnie przetwarzany, a wskaźnik zapisu na rekord ostatnio zapisany do bufora. Rozmiar bufora jest stały, przez co operacja odczytu danych do bufora jest kilkakrotnie powtarzana dla dużych zbiorów. Liczba odczytów danych do bufora zależy od wielkości zbioru, tzn. od liczby rekordów w zbiorze. Liczba zapisów wyników do zbioru jest o jeden większa od liczby odczytów. Wynika to z konieczności aktualizacji danych zawartych w nagłówku zbioru.

Dla każdego rekordu znajdującego się w buforze budowany jest stos wyliczeniowy, którego elementami są argumenty wyrażenia określającego warunek selekcji. W trakcie budowy stosu wyliczeniowego, zgodnie z zasadami Odwrotnej Notacji Polskiej, są wykonywane operacje na argumentach (elementach stosu). Wynikiem przetwarzania na stosie wyliczeniowym jest wyznaczenie wartości logicznej wyrażenia (warunku selekcji). W zależności od wartości wyrażenia logicznego rekord jest odrzucany lub zachowywany. Przed zachowaniem rekordu wykonywana jest operacja projekcji, po czym wynik projekcji umieszczany jest w buforze. Pojedyncze przetwarzanie rekordów oraz wykorzystanie

wskaźnika odczytu i zapisu umożliwia wykorzystanie bufora odczytu jako bufora zapisu rekordów. Po zakończeniu operacji selekcji i projekcji dla rekordów znajdujących się w buforze następuje ich zapis do zbioru. Proces przetwarzania rekordów powtarzany jest do wyczerpania rekordów w zbiorze.

4.3. Organizacja modułu realizującego złączenie i projekcję

Operacja złączenia realizowana w opisywanym module wraz z opcjonalną projekcją jest operacją *równozłączenia*, ponieważ o sklejeniu (konkatenacji) dwóch rekordów decyduje równość wartości obydwu atrybutów występujących w kryterium złączenia. W module zaimplementowano następujące algorytmy złączenia (wybór metody należy do użytkownika formułującego zadanie wyszukiwania):

1) algorytm przeglądania zagnieżdżonego <ang. *nested loops*, *brute force*>;
2) algorytm wykorzystujący indeks zbudowany na jednej z tabel uczestniczących w złączeniu.

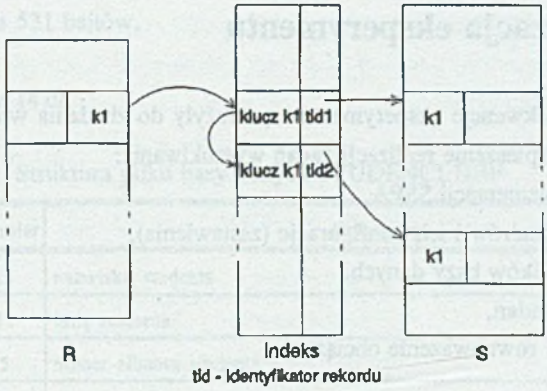
Złączeniu podlegają dwa zbiory rekordów (mogą nimi być tabele lub ich wyselekcjonowane fragmenty) określane jako *zewnętrzny* i *wewnętrzny zbiór danych*. Aktualnie wybrany rekord zbioru zewnętrznego sklejany jest z rekordami zbioru wewnętrznego, spełniającymi kryterium złączenia. Operację tę powtarza się dla każdego rekordu zbioru zewnętrznego.

Algorytm przeglądania zagnieżdżonego

Działając zgodnie z algorytmem przeglądania zagnieżdżonego analizuje się kolejno rekordy zewnętrznego zbioru danych. Dla każdego z nich przegląda się sekwencyjnie wewnętrzny zbiór danych, poszukując krotek spełniających kryterium operacji złączenia z aktualnie wybranym rekordem zbioru zewnętrznego.

Algorytm równozłączenia przez dostęp wykorzystujący indeks dodatkowy

Algorytm zakłada istnienie indeksu zbudowanego na występującym w kryterium złączenia atrybucie jednej z tabel. Indeks jest uporządkowany według wartości tego atrybutu i gęsty, to znaczy zawiera wszystkie wartości atrybutu, jakie występują w tabeli, wraz z ich powtórzeniami. Pojedynczy zapis w indeksie odpowiada więc pojedynczemu rekordowi tabeli. Tabela, na której zbudowano indeks, pełni rolę wewnętrznego zbioru danych. Działając według algorytmu przegląda się sekwencyjnie rekordy zbioru zewnętrznego. Dla każdego z nich przeszukuje się indeks, uzyskując za jego pomocą dostęp do rekordów tabeli bazowej, które charakteryzują się tą samą wartością klucza, co aktualnie wybrany rekord zbioru zewnętrznego (rys. 10).



Rys. 10. Znajdowanie rekordów w wewnętrznym zbiorze S za pomocą indeksu dodatkowego

Fig. 10. Finding records in an internal set S using a secondary index

Operację indeksowania zrealizowano również w opisywanym module. W trakcie równozłączenia utworzony indeks przeglądany jest metodą przeszukiwania dychotomicznego.

Ciąg operacji bezpośrednio związanych z równozłączeniem poprzedza faza przygotowawcza, w trakcie której następuje sprawdzenie poprawności sformułowanych kryteriów: złączenia oraz, opcjonalnie, projekcji. Badaniu podlega: istnienie tabel oraz atrybutów wymienionych w kryteriach, przyporządkowanie atrybutów tabelom, jednoznaczność nazw atrybutów, a także zgodność typów atrybutów, według których nastąpi równozłączenie.

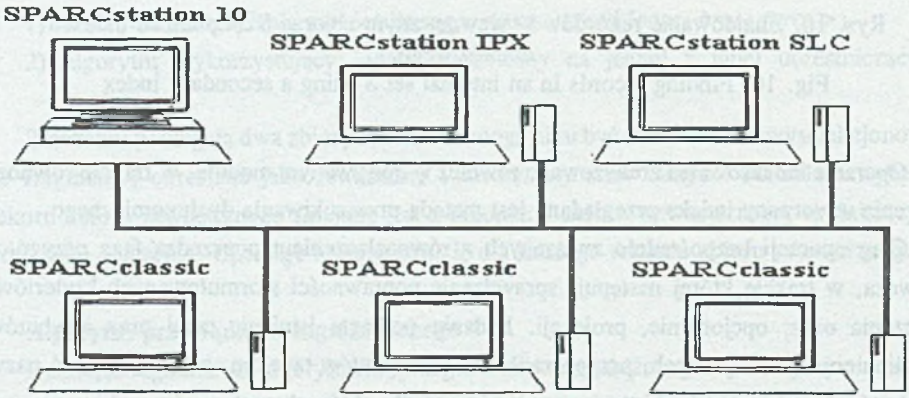
Jeśli nie zostanie sformułowane kryterium projekcji, to tabela utworzona w wyniku operacji równozłączenia zawierać będzie wszystkie atrybuty z obydwu użytych tabel. Jeśli natomiast operacja projekcji ma zostać wykonana, to będzie realizowana na pojedynczym rekordzie sklejonym z dwóch rekordów składowych, tuż przed jego zapisem do tabeli wynikowej. Wszystkie ewentualne kolizje nazw atrybutów tabeli wynikowej zostaną usunięte przez modyfikację nazw.

Przedstawione w opracowaniu wyniki pochodzą z eksperymentów, w których zastosowano metodę przeglądania zagnieżdżonego do realizacji złączenia. Niewielka liczba rekordów uzyskanych w wyniku przeprowadzenia selekcji powoduje, że użycie algorytmu wykorzystującego indeks w złączeniu nie przyspiesza znacząco jego realizacji. Jednakże zapytania, w których operacja złączenia wykonywana jest na dużych zbiorach rekordów, wykazują lepszą efektywność algorytmu z indeksowaniem.

5. Organizacja eksperymentu

Zrealizowane sekwencje eksperymentów posłużyły do zbadania wpływu następujących czynników na przyspieszenie realizacji zadań wyszukiwania:

- metoda implementacji SSPA,
- liczba komputerów i ich konfiguracje (zestawienia),
- replikacje plików bazy danych,
- granulacja zadań,
- dynamiczne równoważenie obciążenia.



Rys. 11. Konfiguracja sieci lokalnej
Fig. 11. The local network configuration

Badania przeprowadzono w sieci stacji roboczych Sun (rys. 11). Obejmowała ona 6 komputerów, wśród których *sun10* reprezentował największą moc obliczeniową, *ipx* oraz *classic* - pośrednią, zaś *slc* - najmniejszą.

5.1. Struktura plików bazy danych

W badaniach wykorzystano część plików bazy danych zawierającej dane dotyczące studentów wyższej uczelni. Baza ta składa się z kilkunastu plików, z których cztery wykorzystano w eksperymentach. Struktura tych plików jest następująca (wszystkie pola w plikach są typu CHARACTER):

a) STUDENCI.DBF - opis pól istotnych dla eksperymentów zawiera tabela 1
- rozmiar rekordu 531 bajtów,
- liczba pól 35,
- liczba rekordów 1479.

Tabela 1

Struktura pliku bazy danych STUDENCI.DBF

Nazwa pola	Rozmiar	Opis pola
NAZWISKO	25	nazwisko studenta
IMIE	25	imię studenta
ALBUM	5	numer albumu studenta
DATAURODZ	8	data urodzenia studenta
KRAJ	10	kraj pochodzenia studenta
PLEC	1	jednoliterowy skrót ('K' lub 'M') oznaczający płeć studenta
AKADEMIK	14	nazwa akademika, w którym student zamieszkuje
WYDZIAL	3	skrót nazwy wydziału
KIERUNEK	3	skrót nazwy kierunku studiów
pozostałe	445	pozostałe dane o studencie (wiele pól nie mających znaczenia w eksperymentach)

b) ZALICZEN.DBF - opis pól istotnych dla eksperymentów zawiera tabela 2
- rozmiar rekordu 35 bajtów,
- liczba pól 9,
- liczba rekordów 42749.

Tabela 2

Struktura pliku bazy danych ZALICZEN.DBF

Nazwa pola	Rozmiar	Opis pola
ALBUM	5	numer albumu studenta
SEMESTR	2	numer semestru
PRZEDMIOT	5	skrót nazwy przedmiotu
OCENA	3	ocena z zaliczenia
pozostałe	20	pozostałe pola (tu nieistotne)

c) EGZAMINY.DBF - opis pól istotnych dla eksperymentów zawiera tabela 3
- rozmiar rekordu 37 bajtów,
- liczba pól 3,
- liczba rekordów 11731.

Tabela 3

Struktura pliku bazy danych EGZAMINY.DBF

Nazwa pola	Rozmiar	Opis pola
ALBUM	5	numer albumu studenta
SEMESTR	2	numer semestru
TERMIN	1	numer terminu egzaminu
PRZEDMIOT	5	skrót nazwy przedmiotu
OCENA	3	ocena z egzaminu
pozostałe	21	pozostałe pola (tu nieistotne)

d) SEMESTRY.DBF - opis pól istotnych dla eksperymentów zawiera tabela 4
- rozmiar rekordu 114 bajtów,
- liczba pól 14,
- liczba rekordów 3635.

Tabela 4

Struktura bazy danych SEMESTRY.DBF

Nazwa pola	Rozmiar	Opis pola
ALBUM	5	numer albumu studenta
SEMESTR	2	numer semestru
SEMESTRZAL	1	czy semestr zaliczony ('T' - tak, ' ' - nie)
ZALICZENIA	4	średnia ocen z zaliczeń
EGZAMINY	4	średnia ocen z egzaminów
pozostałe	98	pozostałe pola (tu nieistotne)

5.2. Zadania wyszukiwania

W pojedynczym eksperymencie wykonywano kilkanaście zapytań o zróżnicowanym stopniu skomplikowania. Poniżej przedstawiono trzy z nich, które są reprezentatywne dla całej serii pytań.

5.2.1. Pytanie 1

Treść pierwszego z pytań wykorzystywanego w eksperymentach jest następująca: "Spośród studentów kierunku Informatyka na wydziale RAU wybrać osoby, które nie zaliczyły semestru pierwszego. Podać nazwy przedmiotów, z których nie uzyskały jeszcze zaliczenia oraz nazwy przedmiotów egzaminacyjnych, z których zostały skierowane na egzamin komisyjny (otrzymały ocenę niedostateczną na trzecim terminie)."

Rozwiązanie zadania można przedstawić za pomocą operatorów algebry relacji w następujący sposób:

$w11 = \pi_{album,\ nazwisko,\ imie}(\sigma_{wydzial='RAU'\ \wedge\ RTRIM(kierunek)='I'}(Studenci))$

$w12 = \pi_{album}(\sigma_{semestr='01'\ \wedge\ semestrzal='\ '}(Semestry))$

$w13 = \pi_{album,\ przedmiot}(\sigma_{semestr='01'\ \wedge\ termin='3'\ \wedge\ ocena='2.0'}(Egzaminy))$

$w14 = \pi_{album,\ przedmiot}(\sigma_{semestr='01'\ \wedge\ ocena='\ '}(Zaliczen))$

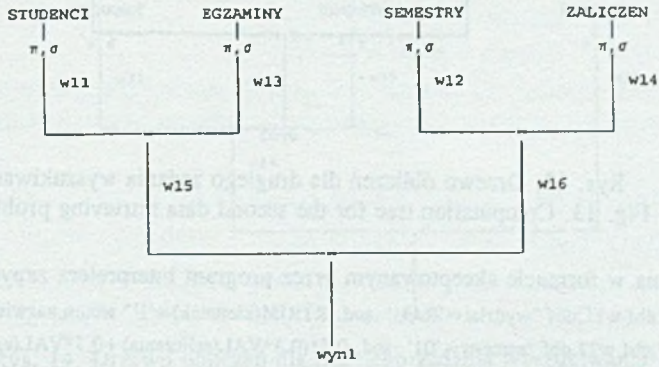
$w15 = w11 \bowtie w13$

$w16 = w12 \bowtie w14$

$wyn1 = w16 \bowtie w15$

Zapis pytania w formacie akceptowanym przez program interpretera zapytań:

```
psel studenci.dbf w11.dbf "wydzial='RAU' .and. RTRIM(kierunek)='I'" album,nazwisko,imie
psel semestry.dbf w12.dbf "semestr='01' .and. semestrzal=' '" album
psel egzaminy.dbf w13.dbf "semestr='01' .and. termin='3' .and. ocena='2.0'" album,przedmiot
psel zaliczen.dbf w14.dbf "semestr='01' .and. ocena=' '" album,przedmiot
zlacz w11.dbf w13.dbf w15.dbf w11.album=w13.album 1
zlacz w12.dbf w14.dbf w16.dbf w12.album=w14.album 1
zlacz w15.dbf w16.dbf wyn1.dbf w15.album=w16.album 1
# wyn1.dbf
studenci.dbf
semestry.dbf
egzaminy.dbf
zaliczen.dbf
```



Rys. 12. Drzewo obliczeń dla pierwszego zadania wyszukiwania
Fig. 12. Computation tree for the first data retrieving problem

Rozwiązaniem zadania jest zatem relacja WYN1, zawierająca nazwiska i imiona poszukiwanych studentów, ich numery albumów oraz wykaz nie zaliczonych przedmiotów. Rozwiązanie w postaci graficznej zaprezentowano na rysunku 12.

5.2.2. Pytanie 2

Kolejne pytanie wykorzystywane w eksperymentach ma następującą treść:
"Spośród studentów Informatyki na wydziale RAU wybrać osoby, które na pierwszym semestrze studiów:

- uzyskały średnią ważoną ocen z zaliczeń i egzaminów co najmniej 3,
- uzyskały z matematyki zaliczenie na co najmniej 4,
- egzamin z matematyki zdały w pierwszym terminie z oceną co najmniej 4."

Rozwiązanie zadania można przedstawić za pomocą operatorów algebry relacji w następujący sposób:

$$w21 = \pi_{album,\ nazwisko,\ imie}(\sigma_{wydzial='RAU'\ \wedge\ RTRIM(kierunek)\ =\ 'I'}(Studenci))$$

$$w22 = \pi_{album}(\sigma_{0.5*(0.3*VAL(zaliczenia)+0.7*VAL(egzaminy))>=3}(Semestry))$$

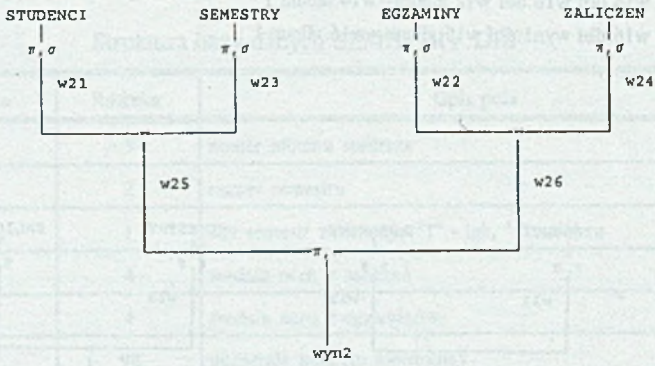
$$w23 = \pi_{album,\ ocena}(\sigma_{semestr='01'\ \wedge\ termin='I'\ \wedge\ VAL(ocena)>=4\ \wedge\ przedmiot='MAT'}(Egzaminy))$$

$$w24 = \pi_{album,\ ocena}(\sigma_{semestr='01'\ \wedge\ VAL(ocena)>=4\ \wedge\ przedmiot='MAT'}(Zaliczen))$$

$$w25 = w21 \bowtie w22$$

$$w26 = w22 \bowtie w24$$

$$wyn2 = \pi_{nazwisko,\ imie}(w26 \bowtie w25)$$



Rys. 13. Drzewo obliczeń dla drugiego zadania wyszukiwania
Fig. 13. Computation tree for the second data retrieving problem

Zapis pytania w formacie akceptowanym przez program interpretera zapytań:

```
psel studenci.dbf w21.dbf "wydzial='RAU' .and. RTRIM(kierunek)='I'" album,nazwisko,imie,kraj
psel semestry.dbf w22.dbf "semestr='01' .and. 0.5*(0.3*VAL(zaliczenia)+0.7*VAL(egzaminy))>=3"
    album
```

```
psel egzaminy.dbf w23.dbf "przedmiot='MAT' .and. VAL(ocena)>=4 .and. termin='1' .and.
    semestr='01'" album,ocena
psel zaliczen.dbf w24.dbf "przedmiot='MAT' .and. VAL(ocena)>=4 .and. semestr='01'" album,ocena
zlacz w21.dbf w22.dbf w25.dbf w21.album=w22.album 1
zlacz w23.dbf w24.dbf w26.dbf w23.album=w24.album 1
pzlacz w25.dbf w26.dbf wyn2.dbf w25.album=w26.album nazwisko,imie,kraj 1
# wyn2.dbf
studenci.dbf
semestry.dbf
egzaminy.dbf
zaliczen.dbf
```

Rozwiązaniem zadania jest relacja WYN2, zawierająca nazwiska i imiona poszukiwanych studentów. Rozwiązanie w postaci graficznej zaprezentowano na rysunku 13.

5.2.3. Pytanie 3

Treść kolejnego z pytań wykorzystywanego w eksperymentach jest następująca: "Spośród mężczyzn w wieku do 24 lat, mieszkających w akademiku ONDRASZEK, wybrać tych, którzy nie zaliczyli dowolnego semestru i co najmniej raz oblali pierwszy termin dowolnego egzaminu"

Rozwiązanie zadania można przedstawić za pomocą operatorów algebry relacji w następujący sposób:

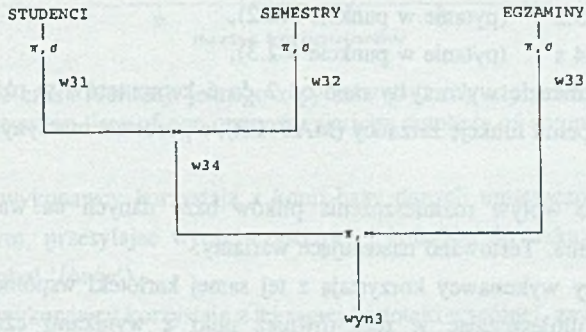
$w31 = \pi_{album,\ nazwisko,\ imie}(\sigma_{plec='M' \wedge akademik='ONDRASZEK' \wedge 94-YEAR(CTOD(dataurodz))<=24}(\text{Studenci}))$

$w32 = \pi_{album}(\sigma_{semestrzal='0'}(\text{Semestry}))$

$w33 = \pi_{album}(\sigma_{termin='1' \wedge ocena='2.0'}(\text{Egzaminy}))$

$w34 = w31 \bowtie w32$

$wyn3 = \pi_{nazwisko,\ imie}(w34 \bowtie w33)$



Rys. 14. Drzewo obliczeń dla trzeciego zadania wyszukiwania
Fig. 14. Computation tree for the third data retrieving problem

Zapis pytania w formacie akceptowanym przez program interpretera zapytań:

```
psel studenci.dbf w31.dbf "plec='M' .and. akademik='ONDRASZEK' .and.
    94-YEAR(CTOD(dataurodz))<=24" album,nazwisko,imie
psel semestry.dbf w32.dbf "semestrzal=' '" album
psel egzaminy.dbf w33.dbf "ocena='2.0' .and. termin='1'" album
zlacz w31.dbf w32.dbf w34.dbf w31.album=w32.album 1
pzlacz w33.dbf w34.dbf wyn3.dbf w33.album=w34.album nazwisko,imie 1
# wyn3.dbf
studenci.dbf
semestry.dbf
egzaminy.dbf
```

Rozwiązaniem zadania jest zatem relacja WYN3, zawierająca nazwiska i imiona poszukiwanych studentów. Rozwiązanie w postaci graficznej zaprezentowano na rysunku 14.

5.3. Opis eksperymentów

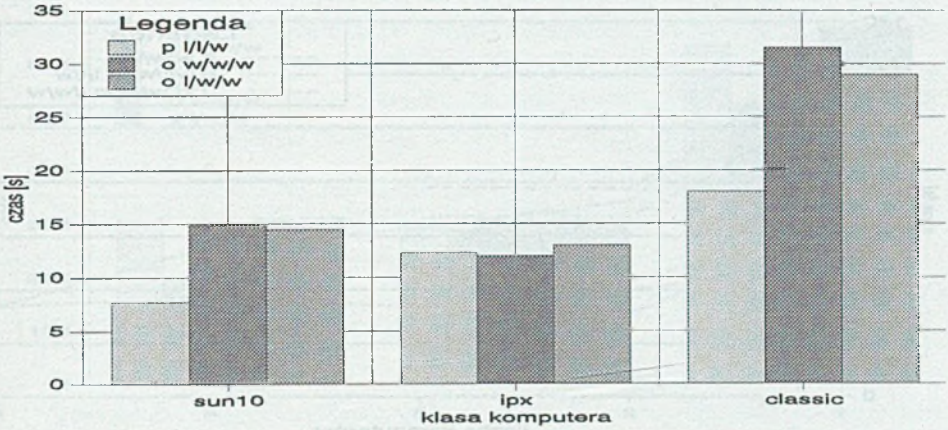
Eksperymenty realizowano w sieci heterogenicznej (rys. 11), która nie była obciążona innymi zadaniami. Każdy eksperyment obejmował wykonanie pojedynczego zadania wyszukiwania (zapytania) lub ciągu niezależnych zapytań w liczbie trzech lub piętnastu. Wykonywano dwa warianty eksperymentów. W pierwszym (wykresy opisane na rysunkach symbolem 'o') pomiędzy stacje sieci komputerowej rozpraszano pojedyncze operacje algebry relacji. W drugim wariancie (rezultaty oznaczone symbolem 'p') rozpraszano całe zapytania. Wykonywane zadania wyszukiwania można podzielić na trzy klasy, biorąc pod uwagę czas wykonania na pojedynczym komputerze. Uśrednione czasy wykonania zapytań danej klasy na komputerze typu *classic* wynoszą:

klasa 1 - 27.5 s (reprezentuje ją pytanie opisane w punkcie 5.2.1),
klasa 2 - 75.5 s (pytanie w punkcie 5.2.2),
klasa 3 - 24 s (pytanie w punkcie 5.2.3).

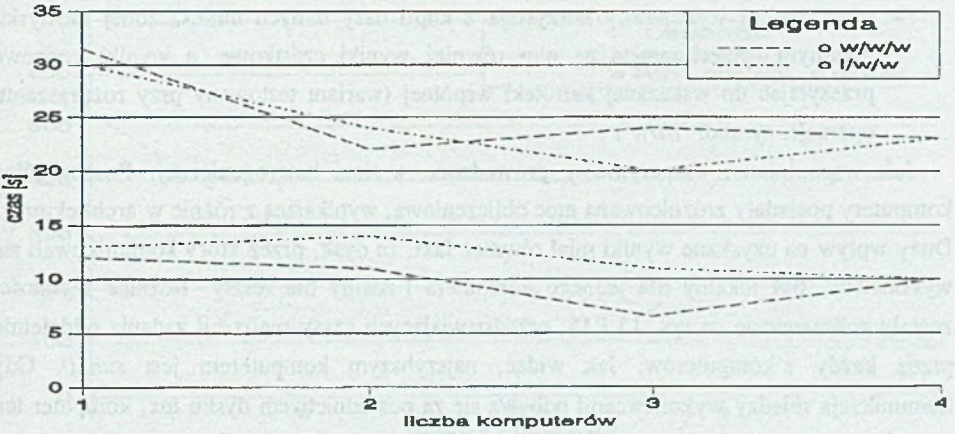
W eksperymencie wykorzystywano od 2 do 6 komputerów w różnych zestawieniach. Komputer *slc* pełnił funkcję zarządcy (MASTER), a pozostałe maszyny realizowały operacje SSPA.

Przebadano wpływ rozmieszczenia plików bazy danych na wartość przyśpieszenia realizacji zadania. Testowano następujące warianty:

— wszyscy wykonawcy korzystają z tej samej kartoteki wspólnej, zawierającej pliki bazy, umieszczając w niej również pliki z wynikami cząstkowymi (wykresy reprezentowane na rysunkach symbolem '*w/w/w*'),

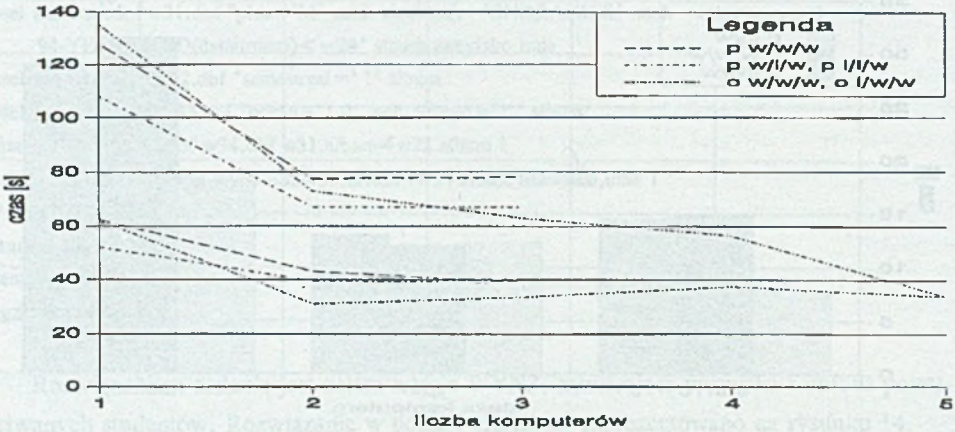


Rys. 15. Czas wykonania pytania klasy 1 na jednym komputerze
Fig. 15. Execution time of a query from class no 1 on one computer



Rys. 16. Zależność czasu realizacji jednego zapytania od liczby użytych komputerów
Fig. 16. Execution time of one query for various numbers of computers

— poszczególni wykonawcy korzystają z kopii bazy danych umieszczonej na swoim dysku lokalnym, przesyłając wyniki cząstkowe i końcowe do wskazanej kartoteki wspólnej (symbol '*l/w/w*'),
— poszczególni wykonawcy korzystają z tej samej kartoteki wspólnej, zawierającej pliki bazy, umieszczając w niej wyniki końcowe, a wyniki cząstkowe przechowując na swoim dysku lokalnym (wariant testowany przy rozpraszaniu zapytań; symbol '*w/l/w*'),

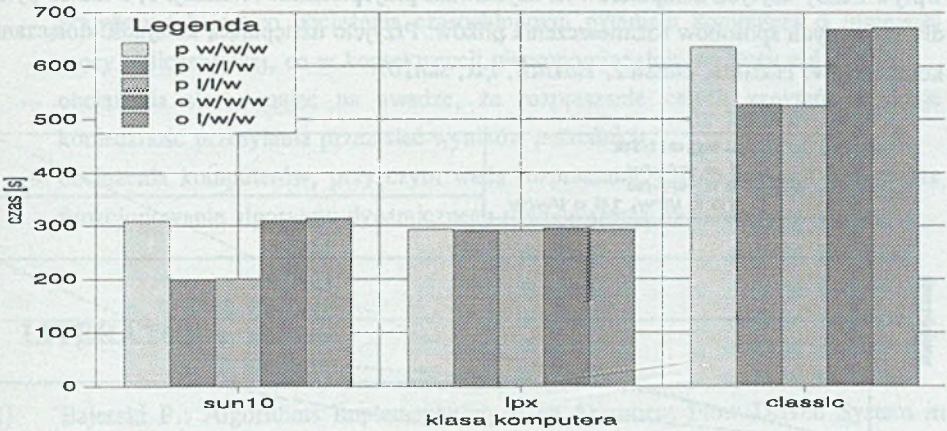


Rys. 17. Zależność czasu realizacji trzech zapytań od liczby użytych komputerów
Fig. 17. Execution time of three queries for various numbers of computers

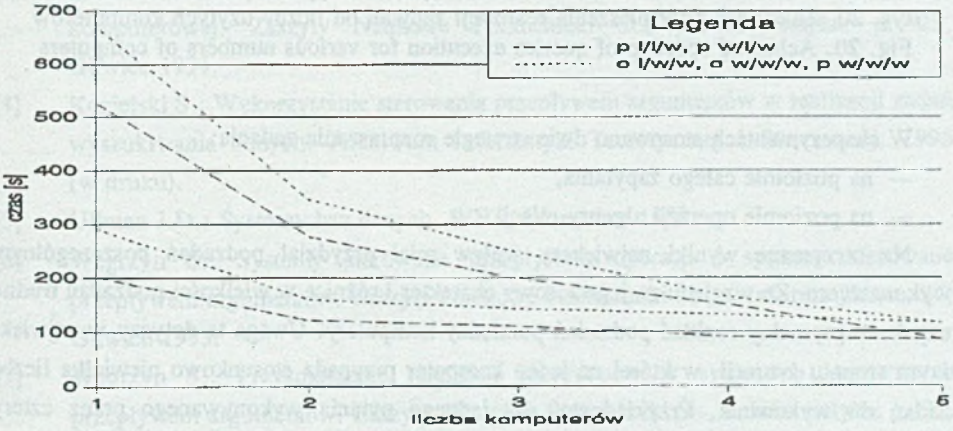
— poszczególni wykonawcy korzystają z kopii bazy danych umieszczonej na dysku lokalnym, przechowując na nim również wyniki cząstkowe, a wyniki końcowe przesyłając do wskazanej kartoteki wspólnej (wariant testowany przy rozpraszaniu operacji; symbol '*l/l/w*') .

Jak wspomniano, eksperymenty prowadzono w sieci heterogenicznej. Poszczególne komputery posiadały zróżnicowaną moc obliczeniową, wynikającą z różnic w architekturze. Duży wpływ na uzyskane wyniki miał również fakt, że dysk, przez który komunikowali się wykonawcy, był lokalny dla jednego komputera i zdalny dla reszty. Różnice szybkości zostały zobrazowane na rys. 15 i 18, przedstawiających czasy realizacji zadania oddzielnie przez każdy z komputerów. Jak widać, najszybszym komputerem jest *sun10*. Gdy komunikacja między wykonawcami odbywa się za pośrednictwem dysku *ipx*, komputer ten wykonuje zadania najszybciej. Dla danej liczby komputerów uczestniczących w realizacji zadania można uzyskać różny czas jego wykonania w zależności od doboru użytych komputerów.

W grupie eksperymentów wykonanych przez taką samą liczbę komputerów najkrótszym czasem wykonania charakteryzują się eksperymenty z wykorzystaniem komputerów *ipx* i *sun10*. Na rys. 16, 17 i 19 zobrazowano zależności czasu wykonania zadania od liczby komputerów. Dla różnych sposobów rozmieszczenia plików przedstawiono wyniki dla najszybszej i najwolniejszej konfiguracji. Pozostałe możliwe do uzyskania konfiguracje dają czasy pośrednie. Z uwagi na czytelność wykresów pozycje w legendzie reprezentują sposoby rozmieszczenia plików, charakteryzujące się zbliżonymi czasami wykonania.



Rys. 18. Czas wykonania piętnastu pytań na jednym komputerze
Fig. 18. Execution time of fifteen queries on one computer

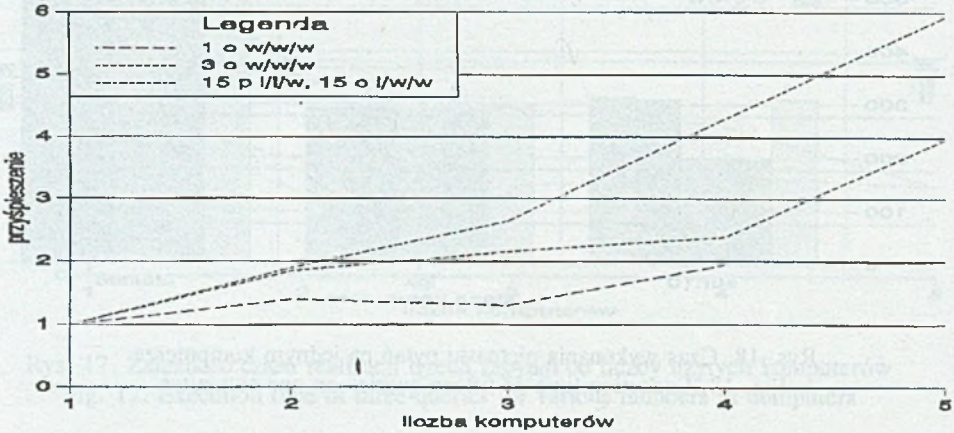


Rys. 19. Zależność czasu realizacji piętnastu zapytań od liczby użytych komputerów
Fig. 19. Execution time of fifteen queries for various numbers of computers

5.4. Podsumowanie

Eksperymenty wykazały, że wykorzystanie systemu LINDA do rozwiązywania zadań wyszukiwania według modelu sterowania przepływem argumentów umożliwia uzyskanie przyspieszenia wykonania zadania przez jego rozproszenie. Na rys. 20 zaprezentowano

wpływ liczby użytych komputerów na uzyskiwane przyspieszenie realizacji 1, 3 lub 15 pytań dla wybranych sposobów rozmieszczenia plików. Przyjęto następującą kolejność dołączania komputerów: *classic1*, *classic2*, *classic3*, *ipx*, *sun10*.



Rys. 20. Zależność przyspieszenia realizacji zapytań od liczby użytych komputerów
Fig. 20. Achieved speedup of queries execution for various numbers of computers

W eksperymentach stosowano dwie strategie rozpraszania zadania:

— na poziomie całego zapytania,

— na poziomie operacji algebry relacji.

Na otrzymane wyniki największy wpływ miał przydział podzadań poszczególnym wykonawcom. Ze względu na jego losowy charakter i różnice w wielkości podzadań trudno uzyskać optymalny rozkład podzadań pomiędzy komputery. Uwaga ta dotyczy w największym stopniu sytuacji, w której na jeden komputer przypada stosunkowo niewielka liczba zadań do wykonania. Przykładowo, dla jednego pytania wykonywanego przez cztery komputery czas realizacji w zależności od rozkładu podzadań wynosił 10.5 s lub 22,4 s. Dokonanie replikacji bazy danych, czyli skopiowanie danych na dyski lokalne komputerów uczestniczących w realizacji zadania wyszukiwania skróciło nieznacznie czas jego wykonania. W nieobciążonej sieci dla długiej serii zapytań bardziej opłacalne jest rozproszenie pytań niż operacji. Żadna z trzech zaimplementowanych koncepcji SSPA nie wyróżniła się swoją efektywnością na tle innych.

Wybór strategii rozpraszania zadania należałoby uzależnić od:

— liczby zapytań, z uwagi na trudności w rozproszeniu ciągu pytań, których liczba jest niewielka w stosunku do liczby komputerów,

— zróżnicowania ich złożoności, gdyż przy rozpraszaniu całych zapytań istnieje większe prawdopodobieństwo obciążenia czasochłonnym pytaniem komputera o mniejszej mocy obliczeniowej, co w konsekwencji nieproporcjonalnie wydłuży obliczenia,
— obciążenia sieci, mając na uwadze, że rozpraszanie całych zapytań eliminuje konieczność przesyłania przez sieć wyników pośrednich,
— obciążenia komputerów, przy czym wadą rozpraszania całych zapytań jest gorsze funkcjonowanie algorytmu dynamicznego równoważenia obciążenia.

LITERATURA

[1] Bajerski P.: Algorithms Implementation in an Argument Flow Driven System in a Local Network, Bull. Pol. Ac., (w druku).

[2] C-Linda User's Guide & Reference Manual, Scientific Computing Associates Inc. 1993.

[3] Kozielski S.: Języki zapytań relacyjnych baz danych a rozpraszanie obliczeń w sieci komputerowej. Zeszyty Naukowe Politechniki Śl., s. Informatyka, z. 25, Gliwice 1994.

[4] Kozielski S.: Wykorzystanie sterowania przepływem argumentów w realizacji zadań wyszukiwania danych. Archiwum Informatyki Teoretycznej i Stosowanej, 1995 (w druku).

[5] Ullman J.D.: Systemy baz danych. WNT, Warszawa 1988.

[6] Węgrzyn S.: Systemy sterowane przepływem operacji i systemy sterowane przepływem argumentów. Zeszyty Naukowe Politechniki Śl., s. Informatyka, z. 24, Gliwice 1993.

[7] Węgrzyn S.: Przyspieszenie realizacji algorytmów w systemach sterowanych przepływem argumentów. Zeszyty Naukowe Politechniki Śl., s. Informatyka, z. 25, Gliwice 1994.

[8] Wyciślik M.: The Necessary and Sufficient Condition of Realizability of an Algorithm in Systems Controlled by a Flow of Arguments, Bull. Pol. Ac., (w druku).

Recenzent: Dr inż. Andrzej Wilk

Wpłynęło do Redakcji 15 marca 1995 r.

Abstract

Presented parallel retrieval system speeds up processing of queries by means of distributed computations in a cluster of workstations connected by a local network. The system follows the Argument Flow Driven Model. A tree of computations consists of relational algebra operators (selection, projection, join), which gives a high degree of parallelism. The experiments were carried out on a cluster consisting of five Sun workstations of different type. The Argument Flow Driven System was written using the Network Linda System in C-Linda language, the relational algebra operations were written in C language.

In our experiments we examined an impact of some factors on speed of query processing: the Argument Flow Driven System implementation method, number of computers used and their configuration, database files location and their replication, task granularity, and dynamic load balancing.

Results achieved show that query processing can be speeded up by means of distributed computations in a cluster of workstations and allow to suggest a way of associating the distribution strategy with the complexity of queries, network parameters and database replication.