ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1993

Seria: INFORMATYKA z. 24 Nr kol. 1222

Stanisław KOZIELSKI
Robert TUTAJEWICZ
Rafał FAGAS
Jacek PIEC

ANALIZA MOŻLIWOŚCI I CELOWOŚCI WYKORZYSTANIA SIECI KOMPUTEROWEJ DO RÓWNOLEGŁEJ REALIZACJI ZADAŃ WYSZUKIWANIA DANYCH

Streszczenie. W artykule rozpatrzono możliwości rozpraszania obliczeń w sieci komputerowej w celu uzyskania przyspieszenia realizacji zadań wyszukiwania danych. Przeanalizowano przydatność typowych języków zapytań do formułowania rozpraszanych zadań. Omówiono wyniki przeprowadzonych eksperymentów. Przedstawiono algorytmy optymalizujące proces rozpraszania.

ANALYSIS OF POSSIBILITY AND EXPEDIENCY OF UTILIZING COMPUTER NETWORK IN PARALLEL SOLVING OF DATA RETRIEVING PROBLEMS

Summary. Possibility of computation distribution in a computer network for obtaining data retrieving acceleration is considered in the paper. Usefulness of query languages to formulation of tasks being distributed is analized. Results of experiments are discussed. Algorithms of distribution optimization are presented.

Pracę wykonano w ramach grantu KBN nr 8 0269 91 01.

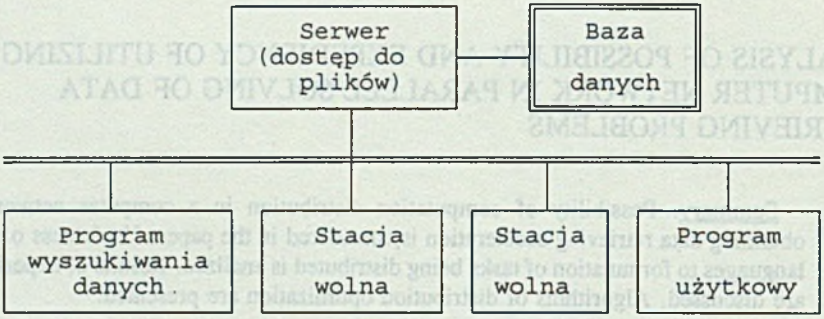
DIE ANALYSE DER MÖGLICHKEITEN UND ZWECKMÄßIGKEIT DER AUSNUTZUNG DES COMPUTER-NETZWERKES ZUR PARALLELEN AUSFÜHRUNG DER DATENAUSSUCHAUFGABEN

Zusammenfassung. Im Artikel wurden die Möglichkeiten der Verteilung der Berechnungen im Computer-Netzwerk erörtert, die zum Erlangen der Beschleunigung der Ausführung der Datenaussuchaufgaben dienen. Die Verwendbarkeit der typischen Abfragensprachen zur Formulierung der verteilten Aufgaben wurde analysiert. Die Ergebnisse der durchgeführten Versuche wurden ausdiskutiert. Die Algorithmen zur Optimierung des Verteilungsprozesses wurden vorgestellt.

1. Wstęp

Celem pracy jest przedyskutowanie możliwości i uwarunkowań rozpraszania zadań obliczeniowych w lokalnej sieci komputerowej w celu uzyskania przyspieszenia realizacji złożonych programów. Jako przykład czasochłonnych zadań obliczeniowych wybrano zagadnienia wyszukiwania w bazach danych.

Załóżmy w tym celu, że baza danych została zorganizowana na dysku serwera takiej sieci (rys. 1). W komputerach - stacjach sieci realizowane są programy użytkowe odwołujące się do bazy poprzez serwer, który między innymi pełni rolę zarządcy plików, umożliwiając dostęp do danych.



Rys. 1. Ogólna ilustracja rozważanej sieci komputerowej
Fig. 1. Outline view of considered computer network

Załóżmy dalej, że ogólną postać programu wyszukiwania danych można przedstawić jako ciąg operacji $O_1, O_2, O_3, \ldots, O_n$. Zwróćmy uwagę, że jeśli dla kolejnych operacji O_{k-1}, O_k wykonanie operacji O_k nie zależy od rezultatów i zakończenia operacji O_{k-1}, to istnieje możliwość równoległej realizacji tych operacji. Jednym ze sposobów takiego wykonania mogłoby być umieszczenie fragmentu programu realizującego operację O_k w wolnej w danej chwili stacji sieci. Oczywiście, przejście do wykonania kolejnej operacji O_{k+1} musiałoby być uwarunkowane sprawdzeniem zakończenia operacji O_k. Gdyby jednak wykonanie O_{k+1} nie zależało od rezultatów O_{k-1}, ani O_k, to również operacja O_{k+1} mogłaby być realizowana równolegle do O_{k-1} oraz O_k w trzeciej stacji sieci.

Zrealizowanie powyższej koncepcji powiązane jest między innymi z następującymi problemami:

- Wstępnym warunkiem realizowalności takiego podejścia jest dostępność narzędzi programowych umożliwiających przekazywanie komunikatów i parametrów do innych stacji, zdalne uruchamianie programu w takich stacjach, a także synchronizowanie realizacji programów w kilku stacjach. Do zrealizowania tych operacji wykorzystywano w trakcie eksperymentów prezentowanych w dalszej części niniejszej pracy bibliotekę prostych programów [5], których sposób użycia omówiono w punkcie 2.2.2.
- Podstawowym zagadnieniem dla zadań wyszukiwania jest wyjaśnienie, jakiego rodzaju operacje kryją się za symbolami $O_1, O_2, O_3, \ldots, O_n$. Zauważmy, że jeśli operacje O_{k-1}, O_k lub O_{k+1} będą operacjami bardzo prostymi, to nakład pracy na rozpraszanie obliczeń przewyższy korzyści z równoległej ich realizacji. Dlatego konieczne jest rozpatrzenie właściwej dla rozważanego zagadnienia formy zapisu zadań wyszukiwania. W tym celu w punkcie 2 przedstawiona zostanie przeglądowa charakterystyka języków zapytań, wraz z oceną ich przydatności w rozważanym podejściu.
- Ostatnim z dyskutowanych zagadnień jest niebezpieczeństwo przeciążenia sieci. Jest to związane ze specyfiką realizacji rozproszonych zadań wyszukiwania. Zauważmy bowiem, że realizacja tych zadań będzie wymagała w każdym przypadku transmisji danych z serwera do stacji realizujących te zadania. Może w związku z tym powstać obawa, że zatłoczenie sieci zniweluje wszelkie ewentualne korzyści wynikające z równoległej realizacji rozproszonych zadań. W pracy przedstawiono rezultaty niektórych przeprowadzonych eksperymentów, uzasadniające celowość wykorzystania sieci komputerowej do równoległej realizacji zadań wyszukiwania danych.

2. Analiza własności języków zapytań pod kątem możliwości rozpraszania obliczeń w zadaniach wyszukiwania danych

Dla relacyjnego modelu danych opracowano szereg języków manipulowania danymi oraz języków zapytań. Tworzą one kilka poziomów abstrakcji. Poniżej przedstawiona zostanie charakterystyka tych języków, ograniczona do zagadnień wyszukiwania danych (języki zapytań). Jej podstawą jest sposób formułowania zadań wyszukiwania danych w przypadkach, kiedy pytania użytkownika dotyczą kilku plików danych. W celu zilustrowania rozważań rozpatrzymy przykładową bazę danych.

Baza danych: Pracownicy(nazwisko, imie, data, nrz, nrp)
Dochody(nrp, nrt, kwota)
Tematy(nrt, nazwa, kierownik)

Niektóre z nazw atrybutów (pól) oznaczają: data - datę urodzenia pracownika, nrz - numer zespołu, w którym pracuje pracownik, nrp - numer (identyfikator) pracownika, nrt - numer (identyfikator) tematu, nazwa - nazwę tematu. Dane w tych relacjach (plikach) dotyczą pracowników pewnej instytucji, tematów realizowanych w niej prac oraz sumarycznych dochodów uzyskanych przez pracowników za realizację określonych tematów. Zakładać będziemy, że dany pracownik może brać udział w realizacji kilku tematów, zaś dany temat może być realizowany przez wielu pracowników.

Rozpatrzmy z kolei dwa pytania kierowane do tej bazy i sposób ich zapisu przy użyciu różnych języków zapytań.

Przykłady zadań wyszukiwania (pytań do bazy):

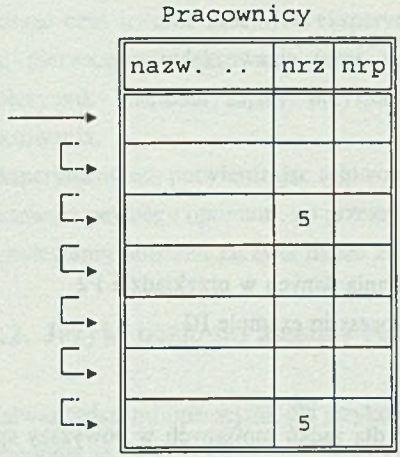
P1) Znajdź nazwiska pracowników zespołu nr 5.

P2) Znajdź nazwiska pracowników zespołu nr 5, których kwoty wypłat przekraczały 1000000. Podaj numery tematów oraz wielkości tych kwot.

2.1. Języki operujące na pojedynczych rekordach

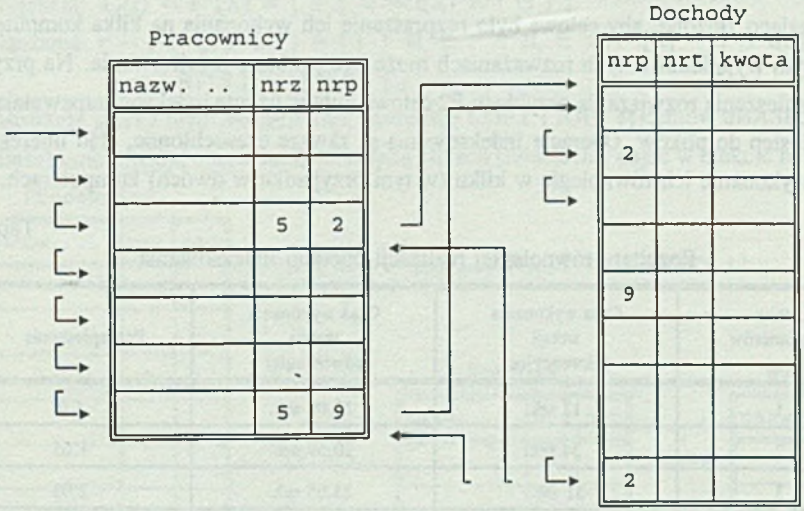
Wykorzystując język tego poziomu (np. dBASE, Clipper, FoxBase, Btrieve), programista musi "nawigować" wśród rekordów pliku, przeglądając je kolejno przy dostępie sekwencyjnym lub wybierając bezpośrednio za pomocą indeksu. Przy operacjach na kilku plikach programista musi indywidualnie kojarzyć rekordy różnych plików dobierając je przez porównanie określonych pól.

Rozwiązania powyższych zadań bez użycia indeksów zilustrowano na rys. 2 i 3 (programy zapisano w języku typu dBASE):




```
use pracowni
locate for nrz = 5
  do while .not.eof()
  ? nazwisko
  continue
  enddo
close all
```

Rys. 2. Ilustracja procesu wyszukiwania danych w przykładzie P1
Fig. 2. Data retrieving process in example P1




```
select 2
use dochody
select 1
use pracowni
locate  for nrz = 5
do while .not.eof()
```

```
    select 2
    locate  for nrp = pracowni->nrp  .and. kwota > 1000000
      do while .not.eof()
          ? pracowni->nazwisko, nrt, kwota
          continue
      enddo
    select 1
    continue
enddo
close all
```

Rys. 3. Ilustracja procesu wyszukiwania danych w przykładzie P2

Fig. 3. Data retrieving process in example P2

2.1.1. Analiza możliwości rozpraszania

Rozpatrując możliwość rozpraszania obliczeń dla zadań zapisanych w powyższy sposób można zauważyć, że większość operacji w bardziej złożonych zadaniach, to wzajemnie powiązane odwołania do wielu plików. Rzadkością jest wystąpienie ciągu takich operacji, że wykonanie następnej nie zależy od wykonania poprzedniej. Operacje te nie są przy tym wystarczająco złożone, aby celowe było rozpraszanie ich wykonania na kilka komputerów.

Istotnym wyjątkiem w tych rozważaniach może być problem indeksowania. Na przykład dla przyspieszenia rozwiązania przykładu P2 celowe byłoby użycie indeksów zapewniających szybki dostęp do plików. Operacje indeksowania są zawsze czasochłonne, stąd interesujące byłoby wykonanie ich równolegle w kilku (w tym przypadku w dwóch) komputerach.

Tabela 1

Rezultaty równoległej realizacji operacji indeksowania

Liczba komputerów	Czas wykonania wersji sekwencyjnej	Czas wykonania wersji równoległej	Przyspieszenie
1	17 sek.	17.00 sek.	1.0
2	34 sek.	20.59 sek.	1.65
3	51 sek.	25.05 sek.	2.03
4	68 sek.	28.35 sek.	2.40
5	85 sek.	36.91 sek.	2.30
6	102 sek.	52.45 sek.	1.94

W celu sprawdzenia możliwości uzyskania praktycznych rezultatów przeprowadzono następujące badania. Na dysku sieciowym utworzono 6 kopii pliku danych. Następnie

sześciokrotnie powtórzono operację indeksowania (język Clipper) tych kopii, zwiększając kolejno od 1 do 6 liczbę stacji sieci, które równolegle realizowały operację indeksowania. Odnosząc czas trwania kolejnych eksperymentów do zwielokrotnianej od 1 do 6 długości czasu pierwszego indeksowania (czas sekwencyjnej realizacji indeksowania) określono współczynnik charakteryzujący przyspieszenie uzyskane dzięki równoległej realizacji indeksowania.

Eksperyment ten, potwierdzając celowość rozpraszania indeksowania, pokazał równocześnie istnienie pewnego optimum, po przekroczeniu którego przyspieszenie uzyskiwane dzięki zrównolegleniu obliczeń zaczyna maleć z uwagi na zatłoczenie sieci.

2.2. Języki poziomu algebry relacji

Najważniejszymi operacjami dla języków tego poziomu są selekcja, projekcja i złączenie. Definiujemy je następująco:

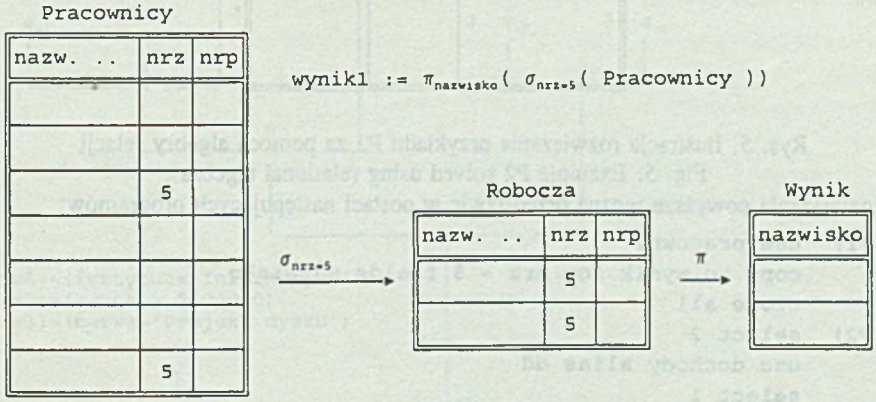
Niech będą dane relacje: r(R), s(S), $A \in R$, $X \subseteq R$

Selekcja: $\sigma_{A=a}(r) = r'(R) = \{ t \mid t \in r \wedge t[A] = a \}$

Projekcja: $\pi_X(r) = r''(X) = \{ u \mid u = t[X] \wedge t \in r \}$

Złączenie: $r \bowtie s = q(R \cup S) = \{ t \mid \exists t_r \in r \wedge \exists t_s \in s \wedge t[R] = t_r \wedge t[S] = t_s \wedge t_r[R \cap S] = t_s[R \cap S] \}$

Instrukcje języka tego poziomu (np. instrukcje COPY i JOIN systemów dBASE, Clipper, FoxBase) umożliwiają operowanie na całych plikach (relacjach), dając w efekcie nowe pliki.

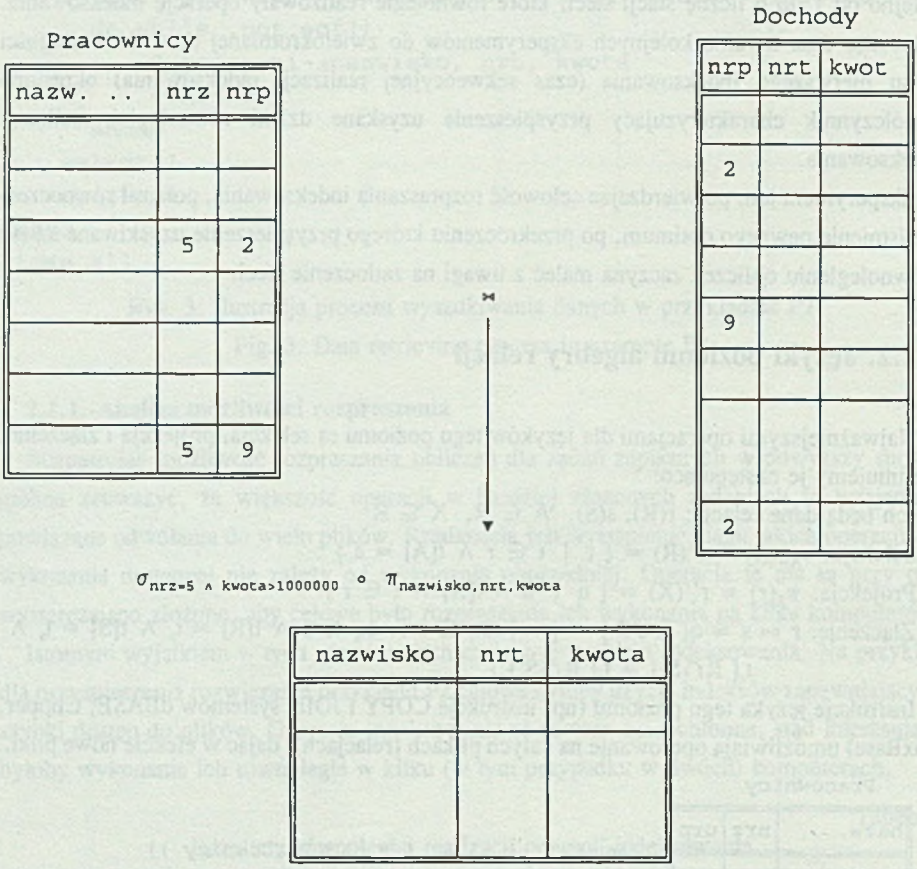


Rys. 4. Ilustracja rozwiązania przykładu P1 za pomocą algebry relacji
Fig. 4. Example P1 solved using relational algebra

Kluczową dla operacji na kilku plikach jest instrukcja złączenia, pozwalająca na automatyczne skojarzenie wszystkich odpowiadających sobie rekordów dwóch plików.

Rozwiązania rozważanych uprzednio przykładów przedstawiają rysunki 4 i 5.

$$\text{wynik2} := \pi_{\text{nazwisko,nrt,kwota}}(\sigma_{\text{nrz=5} \wedge \text{kwota>1000000}}(\text{Pracownicy} \bowtie \text{Dochody}))$$



Rys. 5. Ilustracja rozwiązania przykładu P2 za pomocą algebry relacji
Fig. 5. Example P2 solved using relational algebra

Rozwiązania powyższe można przedstawić w postaci następujących programów:

```
ad P1)  use pracowni
        copy to wynik for nrz = 5 fields nazwisko
        close all
ad P2)  select 2
        use dochody alias dd
        select 1
        use pracowni
        join with dd to wynik for nrp=dd->nrt .and. dd->kwota>1000000;
                  .and. nrz=5 fields nazwisko, nrt, kwota
        close all
```

2.2.1. Analiza możliwości rozpraszania

Podstawowe operacje tego języka są wykonywane na całych relacjach (plikach danych) i ich wynikiem są nowe relacje (pliki).

Złożoność tych operacji jest O(n) dla selekcji i projekcji oraz O(m x n) dla złączenia, gdzie n i m są liczbami rekordów w relacjach biorących udział w tych operacjach.

Przy takiej złożoności uzasadnione jest więc równolegle wykonywanie obliczeń, o ile pozwoli na to brak wzajemnych uwarunkowań kolejnych operacji.

Aby przeanalizować ten aspekt, zwróćmy uwagę na możliwe formy zapisu zadania wyszukiwania. Rozważmy w tym celu przykład następującego zadania:

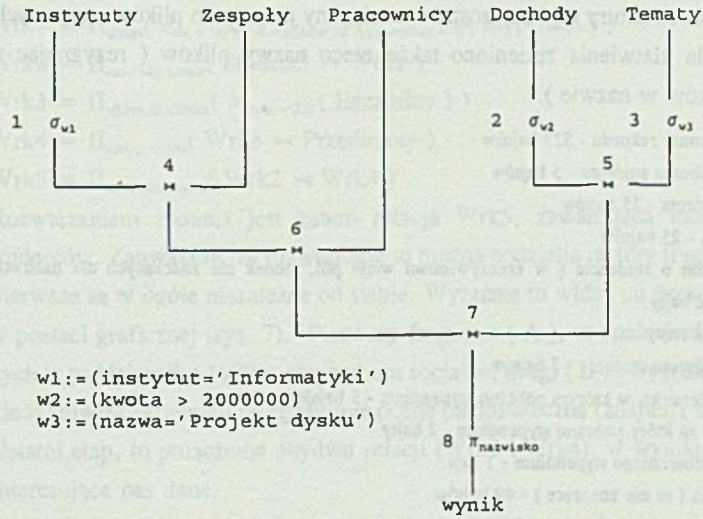
"Podać nazwiska pracowników Instytutu Informatyki, którzy w ramach tematu "Projekt dysku" otrzymali kwoty wypłat większe od 2 000 000",

realizowanego w poprzednio zdefiniowanej bazie danych rozbudowanej o dwa pliki:

Instytuty(instytut, dyrektor)

Zespoły(nazwa_zesp, nrz, kierownik_zesp).

Na początkowym etapie analizy zadania tworzone może być drzewo o postaci przedstawionej na rys. 6. Drzewo to przedstawia jedną z kilku możliwych dróg otrzymania rozwiązania.



Rys. 6. Drzewo obliczeń dla złożonego zadania wyszukiwania
Fig. 6. Computation tree for complex data retrieving problem

Pobieżna ocena przedstawionego rysunku pozwala przedstawić następującą propozycję rozproszenia obliczeń:

Etap 1: równoległa realizacja operacji 1, 2, 3 na trzech różnych stacjach sieci.
Etap 2: równoległa realizacja 4 i 5 (lub 4 + 6 i 5).
Etap 3: sekwencyjna realizacja 6, 7, 8, przy czym 7 i 8 będą realizowane wspólnie.
Z rozważań tych wynika wniosek, że analiza zadania wyszukiwania na odpowiednio wczesnym etapie pozwala określić możliwość rozproszenia obliczeń i wykonanie ich równolegle.

Poniżej przedstawiono eksperyment, który uzasadnia celowość prezentowanych rozważań. Odrębny punkt pracy (punkt 3) poświęcono natomiast analizie różnych możliwości rozproszonej realizacji samej operacji złączenia. Jest to uzasadnione dużą złożonością tej operacji.

2.2.2. Opis przeprowadzonego eksperymentu

W badaniach wykorzystywano dane rzeczywiste z bazy danych dziekanatu. Na wstępie zostanie więc krótko przedstawiona organizacja samej bazy.

Struktura bazy

Baza ta składała się z kilkunastu plików, z których jednak tylko część była wykorzystywana. Dlatego też opis struktury tej bazy został ograniczony jedynie do plików używanych w eksperymentach. Dla ułatwienia zmieniono także nieco nazwy plików (rezygnując z ograniczenia do 8 znaków w nazwie).

a) plik STUDENCI: rozmiar rekordu - 527 bajtów
album - numer albumu studenta - 5 bajtów
nazwisko - nazwisko studenta - 25 bajtów
imię - imię studenta - 25 bajtów
pozostałe - pozostałe dane o studencie (w rzeczywistości wiele pól, jednak nie znaczących dla dalszych rozważań) 472 bajty.

b) plik STYPENDIA: 62 bajty
album - numer albumu studenta - 5 bajtów
rok - rok akademicki, w którym pobrano stypendium - 5 bajtów
miesiąc - miesiąc, za który pobrano stypendium - 2 bajty
stypendium - rodzaj pobieranego stypendium - 1 bajt
pozostałe - pozostałe pola (tu nie znaczące) - 49 bajtów

c) plik PRZEDMIOTY: rozmiar rekordu - 55 bajtów
przedmiot - skrót nazwy przedmiotu - 5 bajtów
nazwa - pełna nazwa przedmiotu - 50 bajtów

d) plik EGZAMINY: rozmiar rekordu - 37 bajtów
album - numer albumu studenta - 5 bajtów
rok - rok akademicki, w którym student zdawał egzamin - 5 bajtów

przedmiot - skrót nazwy przedmiotu - 5 bajtów
ocena - uzyskana ocena - 3 bajty
pozostałe - pozostałe pola - 19 bajtów

e) plik ZALICZENIA: rozmiar rekordu - 35 bajtów

W pliku tym występują podobne pola jak w pliku EGZAMINY, z tą jedynie różnicą, że pola *pozostałe* zajmują 17 bajtów.

W badaniach używano dwu wersji opisywanej bazy, różniących się ilością rekordów w poszczególnych plikach.

Sformułowanie zadania wyszukiwania

Załóżmy, że naszym zadaniem będzie wyszukanie nazwisk wszystkich studentów pobierających w październiku 1990 r. stypendium socjalne, którzy podczas jakiegokolwiek egzaminu uzyskali ocenę niedostateczną. W pliku wynikowym powinny się znaleźć obok nazwiska studenta także nazwy przedmiotów, z których uzyskał on ocenę negatywną na egzaminie.

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

$Wrk1 = \Pi_{album}(\sigma_{rok='90/91' \wedge miesiac='10' \wedge stypendium='S'}(Stypendia))$

$Wrk2 = \Pi_{nazwisko,album}(Studenci \bowtie Wrk1)$

$Wrk3 = \Pi_{album,przedmiot}(\sigma_{ocena='2.0'}(Egzaminy))$

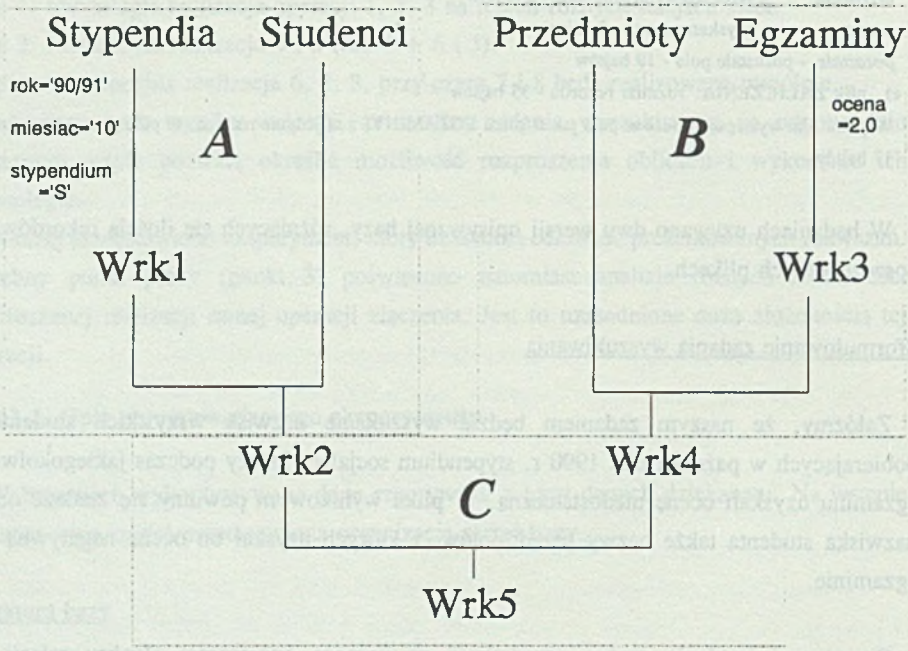
$Wrk4 = \Pi_{album,nazwa}(Wrk3 \bowtie Przedmioty)$

$Wrk5 = \Pi_{nazwisko,nazwa}(Wrk2 \bowtie Wrk4)$

Rozwiązaniem zadania jest zatem relacja Wrk5, zawierająca nazwiska poszukiwanych studentów. Zauważmy, że rozwiązanie to można podzielić na trzy fragmenty, przy czym dwa pierwsze są w ogóle niezależne od siebie. Wyraźnie to widać po przedstawieniu rozwiązania w postaci graficznej (rys. 7). Pierwszy fragment (A), to znalezienie studentów pobierających w październiku 1990 r. stypendium socjalne, drugi (B) - wyszukanie studentów, którzy kiedykolwiek otrzymali na egzaminie ocenę niedostateczną (znajdują się oni w relacji Wrk4). Ostatni etap, to połączenie obydwu relacji (Wrk2 i Wrk4), w wyniku którego otrzymujemy interesujące nas dane.

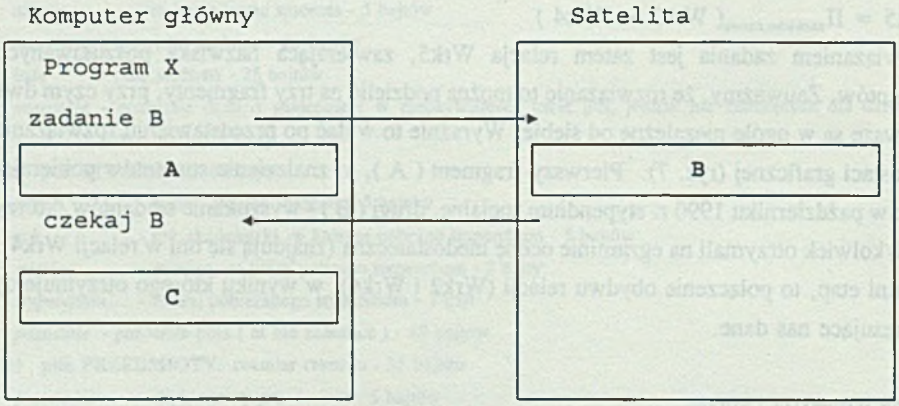
Użyte narzędzia i sprzęt

W celu realizacji eksperymentów posłużono się kompilatorem Clipper (wersja Summer'87). Przeprowadzono je w sieci Novell, przy średnim obciążeniu sieci, na komputerach



Rys. 7. Drzewo obliczeń dla przeprowadzonego eksperymentu
Fig. 7. Computation tree in executed experiment

klasy PC z procesorami 80386SX. W większości eksperymentów używano dwóch komputerów, różniących się częstotliwością zegara. Pierwszy, zwany dalej K1/16, był taktowany zegarem o częstotliwości 16 MHz, drugi (K2/18) nieco szybszy - 18 MHz.



Rys. 8. Sposób synchronizacji między komputerem głównym a satelitą
Fig. 8. Synchronization between main computer and satellite

W eksperymentach wykorzystano pakiet kilku programów, umożliwiający zdalne uruchamianie programów na komputerach połączonych w sieć. Idea działania tego pakietu opiera się na wyróżnieniu jednego komputera w sieci jako tzw. komputera głównego, rozdzielającego zadania między pozostałe komputery, zwane dalej satelitami. Komputer główny wysyła opis zadania (w postaci nazwy zadania oraz nazwy programu do uruchomienia wraz z ewentualnymi parametrami), które jest następnie wykonywane przez któryś z satelitów. To, który z satelitów wykona dane zadanie, jest dla użytkownika zupełnie przypadkowe. Nie ma możliwości jawnego wskazania danego satelity.

Rysunek 8 ilustruje sposób synchronizacji między komputerem głównym a satelitą. Na każdym komputerze-satelicie uruchamiany jest program, który oczekuje na zlecenia przekazywane w postaci opisu zadania z komputera głównego i wykonuje je. Opis ten jest przesyłany do satelity za pomocą programu ZADANIE. Następnie satelita analizuje przekazany mu opis i wykonuje zlecone zadanie. W tym czasie komputer główny może wykonywać inne czynności i po pewnym czasie sprawdzić, czy wykonywanie zleconego zadania zostało zakończone. Sprawdzanie to realizuje program CZEKAJ, wstrzymujący pracę na komputerze głównym aż do momentu zakończenia wykonywania wszystkich, wskazanych poprzez nazwy, zadań.

W większości eksperymentów jeden z komputerów pełnił rolę komputera głównego zarządzającego całością i wykonującego niektóre zadania, drugi natomiast był satelitą. Takie rozwiązanie pozwoliło wymusić wykonanie danego zadania na wskazanej maszynie.

Wyniki eksperymentów

Przeprowadzono całą serię eksperymentów różniących się rozmieszczeniem plików (na dysku lokalnym bądź dysku serwera) oraz rozmieszczeniem podzadań na komputerach. Ich wyniki zamieszczono w poniższych tabelkach. Pierwsza z nich obrazuje wpływ różnego rozmieszczenia plików z danymi na czas wykonania poszczególnych części zadania, druga natomiast wpływ rozmieszczenia zadań na poszczególnych komputerach na tenże czas.

W wyniku przeprowadzonych eksperymentów okazało się, że możliwe jest dość znaczne zmniejszenie czasu wykonania zadań wyszukiwania poprzez zrównoleglenie poszczególnych części tych zadań. Przy czym nie ma znaczenia lokalizacja plików na dysku lokalnym czy też na sieciowym (oczywiście przy założeniu, że parametry obydwu dysków są do siebie zbliżone i dysk sieciowy nie jest nadmiernie obciążony przez inne zadania; porównaj wyniki dla eksperymentów 1, 2 i 3, 4). Duży wpływ na czas wykonania całego zadania może mieć natomiast umiejętne rozdzielenie podzadań pomiędzy komputery o różnych mocach

Tabela 2

Analiza wpływu rozmieszczenia danych na czas równoległej realizacji zadań

Operacja	Mniejsza baza				Większa baza	
	1	2	3	4	1	2
A	213	213	212	212	769	779
B	129	149	128	150	372	435
A i B (razem)	340 (100%)	213 (62.65%)	339 (100%)	212 (62.54%)	1143 (100%)	780 (68.24%)
C	171	203	171	202	527	588
Całe zadanie	513 (100%)	419 (81.68%)	512 (100%)	419 (81.84%)	1670 (100%)	1368 (81.92%)

1 - sekwencyjna wersja rozwiązania, dane na dysku serwera,

2 - wersja równoległa rozwiązania, dane na dysku serwera,

3 - wersja sekwencyjna, dane na dysku lokalnym

4 - wersja równoległa, dane na dyskach lokalnych

Tabela 3

Analiza wpływu rozmieszczenia zadań na czas ich równoległej realizacji

Operacja	Mniejsza baza				Większa baza		
	A1	A2	A3	A4	A1	A2	A4
A	320	323	385	392	1267	1289	1524
B	394	456	408	391	1417	1667	1421
A i B (razem)	714 (100%)	462 (64.7%)	408 (57.1%)	402 (56.3%)	2687 (100%)	1667 (62.04%)	1524 (56.72%)
C	285	334	335	287	1424	1459	1344
Całe zadanie	995 (100%)	857 (86.13%)	742 (74.57%)	679 (68.24%)	4034 (100%)	3128 (77.54%)	2867 (71.07%)

A1 - wersja sekwencyjna rozwiązania, wszytkie pliki z danymi umieszczone na dysku serwera (podobnie jak i w pozostałych przypadkach).

A2 - wersja równoległa, K2/18 (satelita) wykonał część A, części B i C wykonał K1/16.

A3 - wersja równoległa, K2 wykonał część B, K1/16 - części A i C.

A4 - wersja równoległa, zadanie A wykonał K1/16, pozostałe (B i C) - K2/18.

obliczeniowych. Zaleca się dążyć do tego, aby czasy wykonania poszczególnych podzadań, wykonywanych jednocześnie, jak najmniej różniły się pomiędzy sobą.

2.3. Języki poziomu rachunku relacji

Podstawy tych języków wywodzą się z rachunku zdań. Ogólną postać wyrażeń w tych językach zapisać można jako:

$\{ t \mid \phi(t) \}$

i zapis ten oznacza zbiór krotek t spełniających formułę ϕ.

Formuły i poprzedzające je atomy definiujemy następująco:

Atomy: r(R), s(S) - relacje

- r(t), s(u) - t, u są zmiennymi krotkowymi w relacjach r i s,
- $t.A\ \theta\ u.B$ - $A \in R$, $B \in S$, θ ma postać =, >, <, >=, <=,
- $t.A\ \theta\ a$ - a - stała.

Formuły:

- każdy atom jest formułą,
- ϕ_1, ϕ_2 - formuły $\Rightarrow$ $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\neg\phi_1$ - też formuły,
- ϕ - formuła $\Rightarrow$ $(\exists t)(\phi)$, $(\forall t)(\phi)$ - też formuły.

Zapis zdefiniowanych uprzednio pytań przedstawia się w tej konwencji następująco:

ad P1)

$\{t(nazwisko) \mid (\exists p)(Pracownicy(p)) \wedge p.nrz=5 \wedge t.nazwisko=p.nazwisko\}$

ad P2)

$\{x(nazwisko,nrt,kwota) \mid (\exists p)(Pracownicy(p)) \wedge p.nrz=5 \wedge x.nazwisko=$
$=p.nazwisko \wedge (\exists d)(Dochody(d)) \wedge p.nrp=d.nrp \wedge x.nrt=d.nrt \wedge$
$\wedge\ x.kwota=d.kwota \wedge d.kwota>1000000\}$

Zapis pytania w językach tego poziomu (np. SQL) sprowadza się (zazwyczaj) do wskazania potrzebnych plików, nazw wyszukiwanych danych, określenia wymaganych cech tych danych oraz wskazania powiązań między wykorzystywanymi plikami. Są to języki nieproceduralne w stosunku do języków algebry relacji, nie wymagają bowiem określania rodzaju ani kolejności operacji wykonywanych na plikach danych.

Zdefiniowane uprzednio zadania zapiszemy w języku SQL następująco:

```
ad P1)  select nazwisko
        from Pracownicy
        where nrz=5;

ad P2)  select nazwisko, nrt, kwota
        from Pracownicy p, Dochody d
        where p.nrz=5 and d.kwota>1000000 and p.nrp=d.nrp;
lub
        select nazwisko
        from Pracownicy p
        where nrz=5 and nrp in
```

```
(select nrp
from Dochody
where kwota>1000000);
```

2.3.1. Analiza możliwości rozpraszania

W tym przypadku całe zadanie wyszukiwania zapisane jest zazwyczaj jako jedna operacja (instrukcja) języka zapytań. Na poziomie samego języka (np. SQL) brak więc podstaw do rozpraszania obliczeń.

Pozostaje natomiast problem interpretacji (lub kompilacji) wyrażeń języka rachunku relacji. Jednym ze stosowanych rozwiązań jest bowiem kompilacja do zapisu w języku algebry relacji i wtedy należy wrócić do rozważań z punktu 2.2.1.

2.4. Język poziomu relacji uniwersalnej

Przy tym podejściu cała baza danych traktowana jest jako jeden plik (relacja) obejmujący wszystkie pola (atrybuty) danych. Użytkownik nie musi więc znać struktury poszczególnych plików ani powiązań między nimi - są one ustalane automatycznie według identycznych nazw atrybutów z poszczególnych plików. Sformułowanie pytania ogranicza się wtedy do wskazania nazw wyszukiwanych danych (atrybutów) oraz określenia cech, jakie powinny spełniać. Zapis pytania jest więc uproszczony w stosunku do języków rachunku relacji, nie ma bowiem potrzeby wskazywania wymaganych w rozwiązaniu plików danych, ani też warunków wiążących te pliki.

Dla rozważanych przykładów otrzymamy:

U(nazwisko, imie, data, nrz, nrp, nrt, kwota, nazwa, kierownik)

ad P1) retrieve (nazwisko) where nrz = 5;

ad P2) retrieve (nazwisko, nrt, kwota) where nrz = 5 and kwota > 1000000;

Cechą charakterystyczną takiego języka jest fakt, że użytkownik przy zadawaniu pytań do bazy używa jedynie nazw atrybutów, zaś system bazy danych automatycznie określa relacje, w których te atrybuty występują oraz powiązania pomiędzy relacjami. Tym samym, z punktu widzenia użytkownika baza danych widziana jest poprzez schemat relacji uniwersalnej, który w istocie stanowi sumę logiczną schematów wszystkich relacji występujących w bazie danych (tzn. zbiór wszystkich atrybutów występujących w bazie).

Podejście takie, pomijając wszystkie opisane w literaturze [2, 4] problemy związane z realizacją modelu, stanowi kapitalne ułatwienie dla użytkownika, który formułuje pytanie w sposób bardzo prosty i (z punktu widzenia informacji podawanych w trakcie formułowania pytania) bliski językowi naturalnemu.

U podstaw modelu relacji uniwersalnej leży założenie, że system automatycznie określa

związki występujące pomiędzy relacjami, wyszukując pary atrybutów o identycznej nazwie, należące do poszczególnych relacji. Daje to podstawę do wyboru relacji niezbędnych do rozwiązania zadania wyszukiwania. Na relacjach tych wykonywana jest operacja złączenia według warunku równości wspólnych atrybutów (równołączenie). Tak utworzone wyrażenie poddawane jest selekcji z warunkami określonymi w pytaniu oraz projekcji na atrybuty wyszukiwane.

Kluczowym problemem, ze względu na używane algorytmy realizacji zadań wyszukiwania, jest znalezienie minimalnego zbioru relacji bazy danych, który zawierałby wszystkie atrybuty użyte w pytaniu i umożliwiałby wykonanie złączenia relacji poprzez atrybuty wspólne.

Tym samym model relacji uniwersalnej, ze swej natury, gwarantuje, że zadanie wyszukiwania sprowadza się zawsze do realizacji sekwencji równołączeń uzupełnionych o ewentualne selekcje oraz rzutowania.

2.4.1. Analiza możliwości rozpraszania

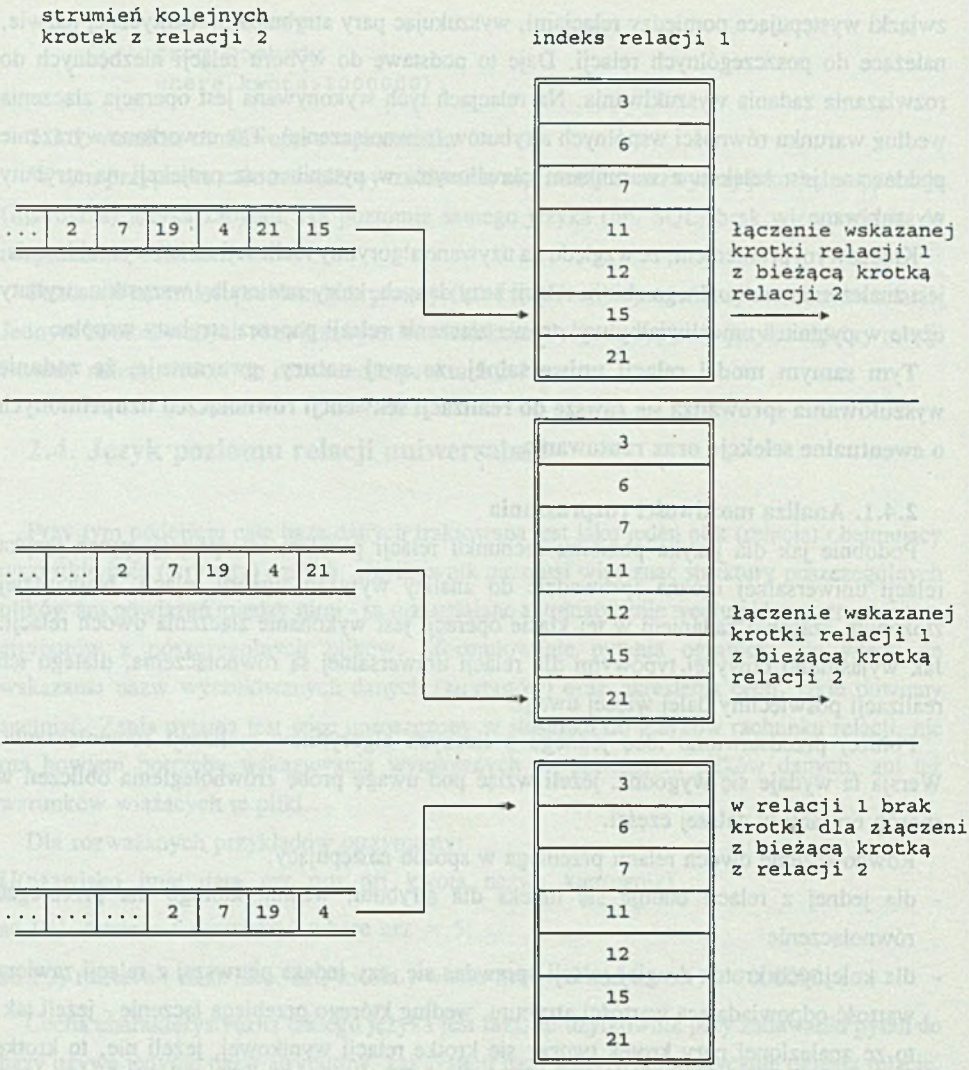
Podobnie jak dla języka poziomu rachunku relacji problem interpretacji pytań języka relacji uniwersalnej można sprowadzić do analizy wyrażeń algebry relacji. Najbardziej złożonym czasowo zadaniem w tej klasie operacji jest wykonanie złączenia dwóch relacji. Jak wyjaśniono powyżej, typowymi dla relacji uniwersalnej są równołączenia, dlatego ich realizacji poświęcimy dalej więcej uwagi.

Poniżej przedstawiono ideę jednego z licznych algorytmów realizacji równołączenia. Wersja ta wydaje się wygodna, jeżeli wziąć pod uwagę próbę zrównoleglenia obliczeń w sposób opisany w dalszej części.

Równołączenie dwóch relacji przebiega w sposób następujący:

- dla jednej z relacji buduje się indeks dla atrybutu, według którego ma przebiegać równołączenie
- dla kolejnych krotek drugiej relacji sprawdza się, czy indeks pierwszej z relacji zawiera wartość odpowiadającą wartości atrybutu, według którego przebiega łączenie - jeżeli tak, to ze znalezionej pary krotek tworzy się krotkę relacji wynikowej, jeżeli nie, to krotkę relacji drugiej się pomija.

Powstające w ten sposób krotki relacji wynikowej są gotowe do łączenia z kolejną relacją użytą w danym pytaniu (rys. 9).



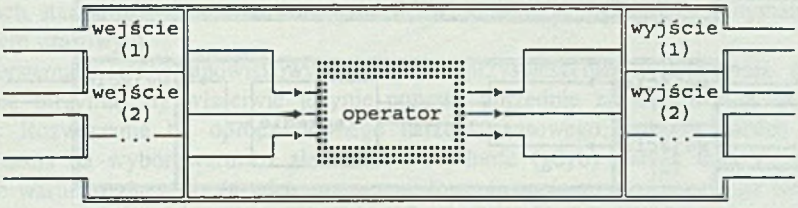
Rys 9. Idea proponowanego algorytmu równołączenia (dla uproszczenia pokazano tylko wartości atrybutu, według którego następuje łączenie)

Fig. 9. Idea of considered equijoin algorithm (only attributes involved in joining are shown)

W proponowanym rozwiązaniu zakłada się, że poszczególne równołączenia realizowane są przez oddzielne procesory (w naszym przypadku stacje sieci lokalnej), a kolejne krotki relacji, powstające w danej stacji wskutek łączenia, przekazywane są do kolejnej stacji niezwłocznie po ich "wyprodukowaniu". W ten sposób zrównolegleniu podlega przede wszystkim kosztowny proces tworzenia plików indeksowych, zaś przetwarzanie kolejnych krotek nabiera charakteru przetwarzania potokowego. Poszczególne stacje sieci powinny,

oczywiście, mieć również zdolność realizacji innych operacji relacyjnych, takich jak selekcja, projekcja, sumowanie, sortowanie, itd.

W ogólnym przypadku zakłada się, że poszczególne stacje przygotowane są do implementacji złożonych operatorów, obsługujących wiele strumieni wejściowych i generujących wiele strumieni wyjściowych krotek.

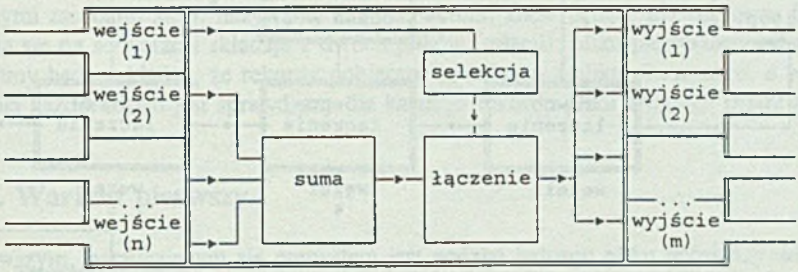


Rys. 10. Ogólna postać węzła przetwarzania
Fig. 10. General view of processing node

W ogólnym przypadku, w celu rozproszenia obliczeń przy rozwiązywaniu zadania wyszukiwania wyrażonego w języku opartym na pojęciu relacji uniwersalnej, wygodnie jest zdefiniować specjalizowany węzeł przetwarzania (rys. 11), który wykonuje złączenie krotek pochodzących z:

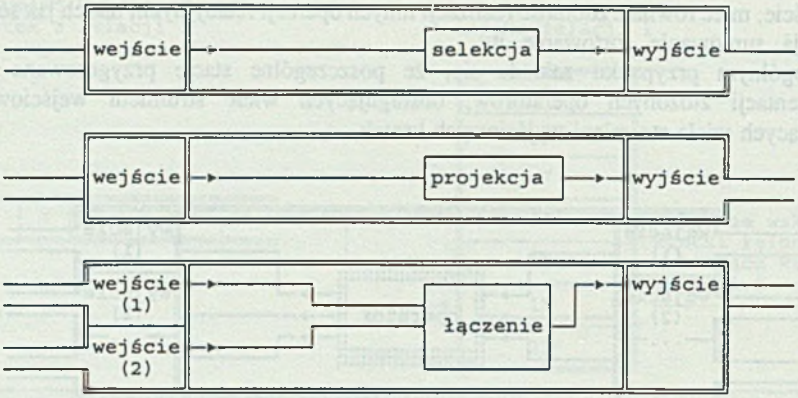
- sumarycznego strumienia krotek napływających do wejść o numerach 2 do n,
- strumienia krotek napływających do wejścia 1 poddanego wstępnej selekcji.

Powstające w wyniku złączenia krotki rozsyłane są do wyjść węzła o numerach od 1 do m.



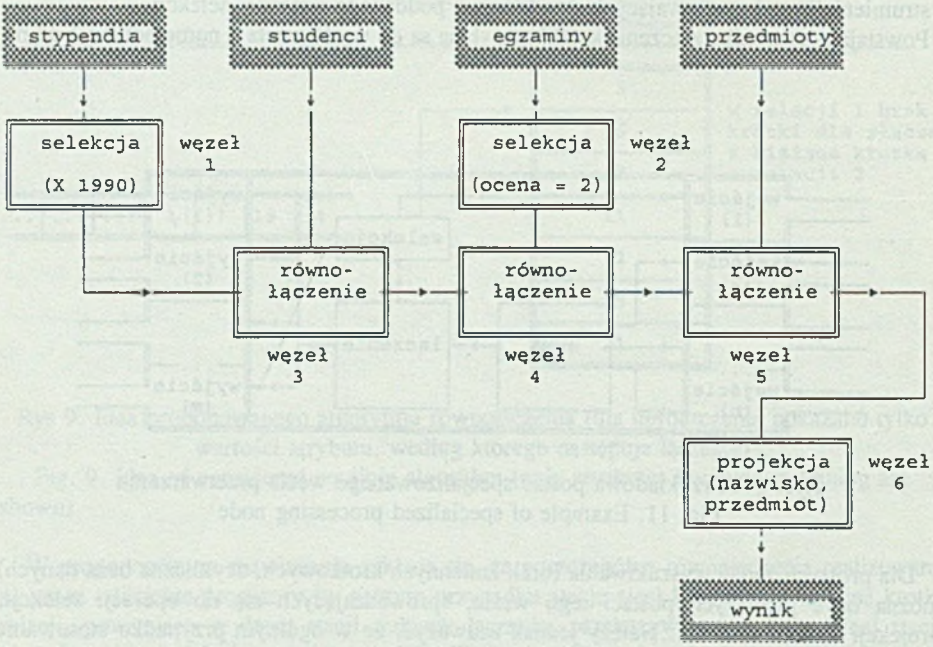
Rys. 11. Przykładowa postać specjalizowanego węzła przetwarzania
Fig. 11. Example of specialized processing node

Dla prostych zadań wyszukiwania (brak zmiennych krotkowych, acykliczna baza danych) można użyć prostszych postaci tego węzła, sprowadzających się do operacji selekcji, projekcji i równołączenia. Należy jednak zauważyć, że w ogólnym przypadku stosowanie postaci węzła przedstawionej powyżej ma uzasadnienie.



Rys. 12. Proste warianty węzła przetwarzania
Fig. 12. Simple variants of processin node

W przyjętym rozwiązaniu proponuje się tak zdekomponować zadanie wyszukiwania w języku opartym na relacji uniwersalnej, by w każdym wykonywanym równolączeniu jeden ze strumieni krotek wejściowych pochodził z relacji źródłowej (ew. poddanej selekcji). Przy takim podejściu realizacja opisanego wcześniej (punkt 2.2.2), przykładowego zadania mogłaby przebiegać tak, jak to przedstawia rys. 13.



Rys. 13. Nowy wariant realizacji zadania wyszukiwania z punktu 2.2.2
Fig. 13. New variant of data retrieving realization (p. 2.2.2)

3. Realizacja rozproszonego łączenia plików baz danych w środowisku sieci lokalnej

Pośród operatorów algebry relacji, złączenie jest z pewnością najbardziej czasochłonne, a co za tym idzie, kłopotliwe w realizacji. Mniej lub bardziej efektywny sposób implementacji złączenia decyduje o sprawności całego systemu zapytań. Dlatego też należy dołożyć wszelkich starań, aby złączenie odbywało się w sposób jak najbardziej optymalny pod względem czasowym.

W systemach jednostanowiskowych znaczące przyspieszenie wykonywania operacji złączenia otrzymujemy właściwie jedynie poprzez uprzednie zaindeksowanie łączonych plików. Rozwiązanie to, oprócz pewnego narzutu czasowego, nakłada bardzo istotne ograniczenia na wybór warunku złączenia. Dowolność (gdyby jednak taką założyć) w doborze warunku złączenia (a więc zarazem w formułowaniu zapytań) wymaga tworzenia odpowiedniego, charakterystycznego dla zapytania indeksu bezpośrednio przed każdym zapytaniem, co wprowadza bardzo znaczne opóźnienie czasowe lub utrzymywanie osobnego indeksu dla każdego atrybutu we wszystkich plikach, co z kolei oprócz ogromnego narzutu czasowego przy dopisywaniu rekordu (gdy konieczne jest uaktualnienie wszystkich indeksów), wymaga bardzo dużych przestrzeni pamięci dyskowej. Dlatego też systemy jednostanowiskowe (jak np. dBase, Xtrieve czy XQL) przy wykonywaniu operacji złączenia wykorzystują zazwyczaj tylko już istniejące w bazie danych indeksy i to tylko wtedy, gdy indeksowany atrybut pełni w zapytaniu znaczącą rolę. Gdy indeksu nie ma, systemy te stosują tradycyjny, skrajnie nieefektywny czasowo algorytm przeglądania całego pliku bazy danych w poszukiwaniu rekordów do połączenia.

Pewną nadzieję na przyspieszenie operacji złączenia bez nakładania znaczących ograniczeń na formułowanie zapytań stwarzają systemy przetwarzania rozproszonego. Poniższe rozważania stanowią próbę stworzenia optymalnego algorytmu rozproszonego łączenia dwóch plików w systemie opartym na klasycznej sieci lokalnej (np. Novell NetWare). Sieć ta składa się ze stacji połączonych ze sobą kablem sieciowym oraz ze stacji wyróżnionej, zarządzającej wspólnymi zasobami sieci, nazywanej dalej serwerem. Dane będące przedmiotem łączenia znajdują się na serwerze i składają z dwóch plików (relacji): pliku pierwszego i drugiego. Algorytmy będą zakładać, że rekordy pobierane są kolejno z pliku pierwszego, a warunek złączenia każdorazowo jest sprawdzany dla każdego rekordu pliku drugiego (patrz rysunek 14).

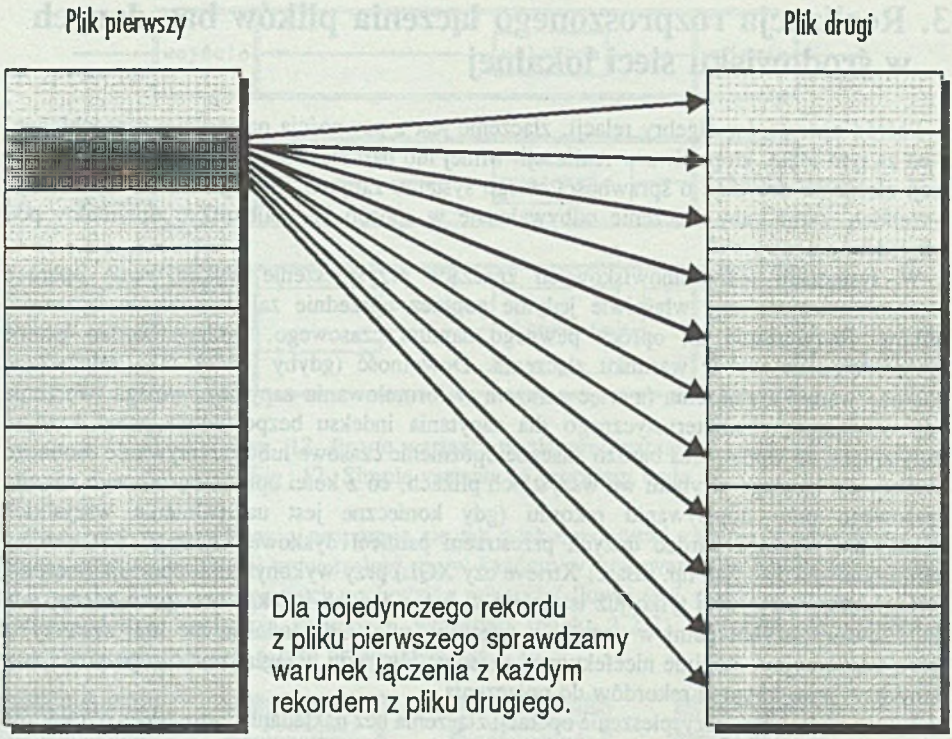
3.1. Wariant pierwszy

Pierwszym, narzucającym się pomysłem jest podział jednego pliku pomiędzy wszystkie stacje w sieci, tak aby każda z nich zajmowała się łączeniem tylko pewnej jego części (przy N stacjach, na każdą stację przypada 1/N pliku). Konfigurację tą ilustruje rysunek 15.

A oto algorytm działania programu łączącego w postaci słownej:

Algorytm łączenia plików PLIK1 i PLIK2 z warunkiem WARUNEK, wykonywany na M-tej z N stacji systemu rozproszonego.

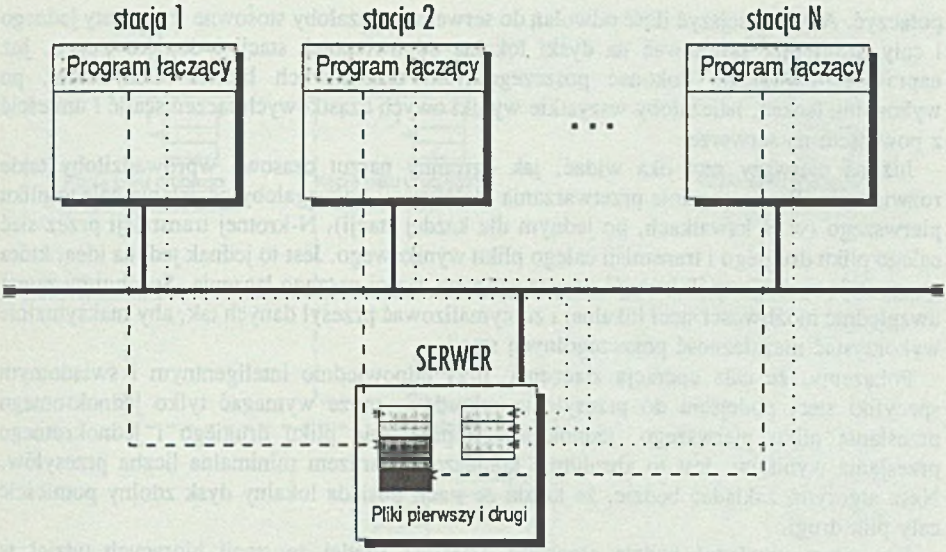
Zmienna L przechowuje liczbę rekordów, jaką stacja ma połączyć
Zmienna Nr przechowuje numer aktualnie łączonego rekordu pliku PLIK1



Rys. 14. Ilustracja idei łączenia bez wykorzystania indeksów
Fig. 14. Idea of join without indices

```
POCZĄTEK
  L := Liczba rekordów w pliku PLIK1 / N
  Nr := M * L
  POWTARZAJ
    Pobierz rekord o numerze Nr z pliku PLIK1
    DLA WSZYSTKICH REKORDÓW z pliku PLIK2 WYKONAJ
      POCZĄTEK
        Gdy rekord o numerze Nr i rekord z pliku PLIK2 spełniają WARUNEK, wykonaj łączenie tych rekordów
        i dopisz wynik do pliku wynikowego
      KONIEC
    Nr := Nr + 1
    L := L - 1
  DOPÓKI L>0
KONIEC
```

Teoretycznie taka realizacja pozwoliłaby na N-krotne zmniejszenie czasu wykonywania złączenia, jako że N komputerów pracowałoby równolegle nad tym problemem. Niestety, jest to tylko teoria. W praktyce wyżej wspomniany algorytm w środowisku sieci lokalnej nie tylko nie daje oczekiwanej N-krotnej poprawy efektywności, ale wręcz w pewnych sytuacjach pogarsza tą efektywność. Jest to wynik działania specyfiki środowiska sieci lokalnej. Jak już wspomnieliśmy, pliki będące przedmiotem łączenia znajdują się na serwerze. Dlatego też wszystkie żądania dostępu do tych plików muszą być przez niego obsłużone. Serwer zaś jest



Rys. 15. Ilustracja pierwszego algorytmu łączenia
Fig. 15. First distributed join algorithm

komputerem jednoprocesorowym i w efekcie wszystkie operacje dostępu do plików, dominujące w naszym algorytmie, wykonywane są przez niego sekwencyjnie. Równolegle wykonywane są tylko bardzo krótkie instrukcje inkrementacji, dekrementacji i sprawdzenia warunku złączenia.

Ponadto, jak się okazuje, serwer nie jest jedynym wąskim gardłem takiej realizacji. Drugim jest kabel sieciowy i protokół dostępu do niego. Czy będzie to sieć z protokołem rywalizacyjnym (Ethernet) czy z protokołem opartym na przekazywaniu uprawnienia, zawsze tylko jedna stacja będzie mogła zajmować łącze. Efektem tego będzie jeszcze mniejszy stopień równoległości wykonywanych operacji. Dodatkowo, przy protokołach rywalizacyjnych, powstanie pewien narzut czasowy związany z dużą ilością kolizji, jaką taka ilość stacji, intensywnie korzystając z łącza, z pewnością wywoła.

Wypada zatem odrzucić przedstawiony wyżej algorytm i zaprojektować inny, biorący pod uwagę specyfikę środowiska.

3.2. Wariant drugi

Wąskimi gardłami w wykonaniu przedstawionego powyżej algorytmu były serwer i łącza sieci, a więc właściwie wszystko, co stanowi sieć lokalną. To sama sieć, jej konstrukcja i zasady działania uniemożliwiły szybszą realizację algorytmu łączenia. Nic w tym dziwnego. Sieć lokalna nie została skonstruowana po to, aby ułatwiać przetwarzanie rozproszone, ale aby umożliwić wielu komputerom dostęp do dzielonych zasobów. Wynika z tego nieco paradoksalny wniosek, że aby uzyskać efektywne przetwarzanie rozproszone w sieci lokalnej, należy jak najmniej z niej korzystać. Minimalizacji muszą podlegać przede wszystkim odwołania do serwera.

Kłopot w tym, że właśnie na serwerze przechowywane są pliki, które nasz algorytm ma

połączyć. Aby zmniejszyć ilość odwołań do serwera, należałoby stosowne fragmenty jednego i cały drugi plik skopiować na dyski lokalne odpowiednich stacji i dopiero potem, już naprawdę równolegle dokonać poszczególnych, cząstkowych łączeń. Oczywiście, po wykonaniu łączeń, należałoby wszystkie wyniki owych cząstkowych łączeń scalić i umieścić z powrotem na serwerze.

Już na pierwszy rzut oka widać, jak ogromny narzut czasowy wprowadziłoby takie rozwiązanie. Przygotowanie przetwarzania dla N stacji wymagałoby przesłania całego pliku pierwszego (w N kawałkach, po jednym dla każdej stacji), N-krotnej transmisji przez sieć całego pliku drugiego i transmisji całego pliku wynikowego. Jest to jednak jedyna idea, która gwarantuje realną współbieżność poszczególnych części naszego łączenia. Spróbujmy zatem uwzględnić możliwości sieci lokalnej i zoptymalizować przesył danych tak, aby maksymalnie wykorzystać niezależność poszczególnych stacji.

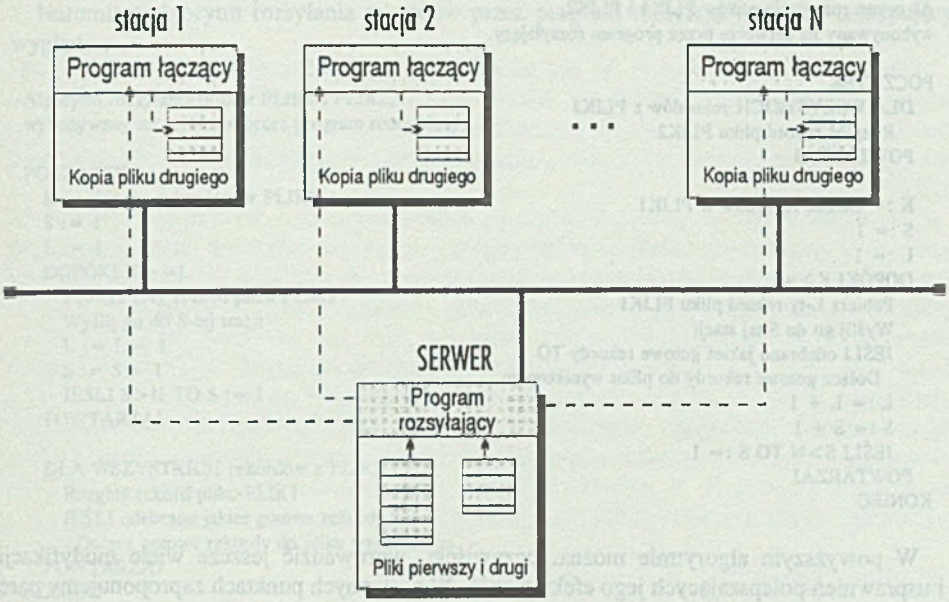
Pokażemy, że cała operacja złączenia, przy odpowiednio inteligentnym i świadomym specyfiki sieci podejściu do przesyłania rekordów, może wymagać tylko jednokrotnego przesłania pliku pierwszego, jednokrotnego przesłania pliku drugiego i jednokrotnego przesłania wyników. Jest to absolutnie konieczna i zarazem minimalna liczba przesyłów. Nasz algorytm zakładać będzie, że każda ze stacji posiada lokalny dysk zdolny pomieścić cały plik drugi.

Jak zatem wyglądać będzie algorytm działania każdej ze stacji biorących udział w rozproszonym łączeniu? Idea łączenia pozostaje taka, jak w pomyśle pierwszym, tzn. każda z N stacji łączy 1/N rekordów pliku pierwszego z wszystkimi rekordami pliku drugiego. Zmienia się natomiast sposób dostępu do tych plików, czyli to, co zadecydowało o odrzuceniu przez nas poprzedniego algorytmu.

W algorytmie pierwszym serwer spełniał rolę pasywną. Jego zadaniem było tylko udostępnianie plików żądającym stacjom. Teraz serwerowi przydzielimy rolę aktywną - będzie on dystrybuował odpowiednie rekordy do odpowiednich stacji oraz zbierał rekordy wynikowe i dołączał je do pliku wynikowego. Aby to uzyskać, konieczne jest uruchomienie na serwerze stosownego procesu. Sieciowy system operacyjny Novell NetWare umożliwia nam zainstalowanie własnego oprogramowania na serwerze w postaci tzw. NetWare Loadable Module (NLM).

Rysunek 16 przedstawia ideę naszego pomysłu. Jak widać, każda ze stacji posiada na dysku lokalnym kopię pliku drugiego, dzięki czemu sprawdzanie warunku złączenia z każdym jego rekordem nie wymaga żadnych transmisji przez łącza sieci. Program rozsyłający, działający na serwerze, pobiera kolejne rekordy pliku pierwszego i rozsyła je do oczekujących na nie stacji. Pierwszy rekord trafia do stacji pierwszej, drugi do drugiej, trzeci do trzeciej, N-ty do N-tej, a rekord N+1 trafia znowu do stacji pierwszej i tak w kółko aż do wyczerpania wszystkich rekordów. Każda ze stacji, po otrzymaniu rekordu, sprawdza dla niego warunek łączenia z każdym rekordem lokalnej kopii pliku drugiego. Jeżeli warunek jest spełniony, stacja łączy te rekordy i wysyła rezultaty z powrotem do serwera, gdzie tworzony jest z nich plik wynikowy.

Milcząco założyliśmy, że kopie pliku drugiego znajdują się już na dyskach lokalnych wszystkich stacji, a przecież wykonanie takich kopii musi zająć bardzo dużo czasu i co za tym idzie, znowu podważyć sensowność rozproszonego łączenia. Na szczęście tak nie jest. Wszystkie rekordy pliku drugiego można bowiem rozesłać do stacji, wykorzystując dostępny w sieciach lokalnych tryb transmisji zwany rozgłaszaniem (ang. broadcast), polegający na jednoczesnym wysłaniu informacji do wszystkich stacji. Rozgłaszamy zatem kolejno wszystkie rekordy pliku drugiego, a czuwające stacje równolegle budują z nich jego lokalne kopie. Praktycznie nie tracimy przy tym czasu, jako że czas rozgłoszenia informacji jest



Rys. 16. Algorytm łączenia uwzględniający specyfikę sieci lokalnej
Fig. 16. Join algorithm optimized for local area network

porównywalny z czasem przesyłu informacji. Kompletny algorytm działania pojedynczej stacji wyglądałby zatem następująco:

Drugi algorytm łączenia plików PLIK1 i PLIK2 z warunkiem WARUNEK, wykonywany na każdej z N stacji systemu rozproszonego.

```
POCZĄTEK
  DOPÓKI serwer rozgłasza rekordy z PLIK2
    Odbierz rekord pliku PLIK2
    Dopisz odebrany rekord do lokalnej kopii
      pliku PLIK2
  POWTARZAJ

  DOPÓKI serwer przysyła rekordy z PLIK1
    Obierz rekord pliku PLIK1
    DLA WSZYSTKICH REKORDÓW z lokalnego pliku PLIK2 WYKONAJ
      POCZĄTEK
        Gdy rekord z pliku PLIK1 i rekord z lokalnego
          pliku PLIK2 spełniają WARUNEK, wykonaj
          łączenie tych rekordów, a rekord wynikowy
          wyślij do serwera
      KONIEC
  POWTARZAJ
KONIEC
```

Natomiast algorytm rozsyłania rekordów przez program rozsyłający miałby następujący wygląd:

Algorytm rozsyłania plików PLIK1 i PLIK2, wykonywany na serwerze przez program rozsyłający.

```
POCZĄTEK
  DLA WSZYSTKICH rekordów z PLIK2
    Rozgłoś rekord pliku PLIK2
  POWTARZAJ

  K := Liczba rekordów w PLIK1
  S := 1
  L := 1
  DOPÓKI K>=L
    Pobierz L-ty rekord pliku PLIK1
    Wyślij go do S-tej stacji
    JEŚLI odebrano jakieś gotowe rekordy TO
      Dołącz gotowe rekordy do pliku wynikowego
    L := L + 1
    S := S + 1
    JEŚLI S>N TO S := 1
  POWTARZAJ
KONIEC
```

W powyższym algorytmie można, oczywiście, wprowadzić jeszcze wiele modyfikacji i usprawnień polepszających jego efektywność. W następnych punktach zaproponujemy parę z nich.

3.3. Wariant trzeci

Punktem wyjścia do analizy nowego wariantu jest odwrócenie sposobu widzenia wariantu poprzedniego. Rozważmy w tym celu, co by było, gdyby stacje nie buforowały całego pliku drugiego, tylko odpowiednie, rozłączne jego części, a rekordy pliku pierwszego byłyby rozgłaszane? Algorytm wyglądałby wtedy następująco:

Trzeci algorytm łączenia plików PLIK1 i PLIK2 z warunkiem WARUNEK, wykonywany na każdej z N stacji systemu rozproszonego.

```
POCZĄTEK
  DOPÓKI serwer przysyła rekordy z PLIK2
    Obierz rekord pliku PLIK2
    Dopisz odebrany rekord do lokalnej kopii
      części pliku PLIK2
  POWTARZAJ

  DOPÓKI serwer rozgłasza rekordy z PLIK1
    Obierz rekord pliku PLIK1
    DLA WSZYSTKICH REKORDÓW z lokalnej
      części pliku PLIK2 WYKONAJ
      POCZĄTEK
        Gdy rekord z pliku PLIK1 i rekord z lokalnej
          części pliku PLIK2 spełniają WARUNEK,
          wykonaj łączenie tych rekordów, a rekord
          wynikowy wyślij do serwera
      KONIEC
  POWTARZAJ
KONIEC
```

Natomiast algorytm rozsyłania rekordów przez program rozsyłający miałby następujący wygląd:

```
Algorytm rozsyłania plików PLIK1 i PLIK2,
wykonywany na serwerze przez program rozsyłający.

POCZĄTEK
  K := Liczba rekordów w PLIK2
  S := 1
  L := 1
  DOPÓKI K> =L
    Pobierz L-ty rekord pliku PLIK2
    Wyślij go do S-tej stacji
    L := L + 1
    S := S + 1
    JEŚLI S>N TO S := 1
  POWTARZAJ

  DLA WSZYSTKICH rekordów z PLIK1
    Rozgłoś rekord pliku PLIK1
    JEŚLI odebrano jakieś gotowe rekordy TO
      Dołącz gotowe rekordy do pliku wynikowego
  POWTARZAJ
KONIEC
```

Czy taka modyfikacja ma sens? Otóż tak. To odwrócenie poprzedniego algorytmu przynosi nam, jak się okazuje, wszechstronne korzyści.

Po pierwsze, na dyskach lokalnych nie jest już buforowany cały plik drugi, a tylko jego 1/N-ta. Zmniejszyły się więc wymagania co do pojemności dysków lokalnych we wszystkich stacjach sieci.

Po drugie, łączenie otrzymanego rekordu nie wymaga już przeglądnięcia całego pliku drugiego, a tylko jego 1/N-tej.

Po trzecie, łączenie odbywa się teraz w sposób bardziej współbieżny. W poprzednim algorytmie rekordy pliku pierwszego dostarczane były stacjom sekwencyjnie - rekord pierwszy do stacji pierwszej, rekord drugi do drugiej itd. Stacje, które wykonały już łączenie otrzymanego rekordu, były bezczynne do chwili, w której otrzymały rekord następny. Wywoływało to ciekawy efekt: zwiększanie ilości stacji biorących udział w łączeniu nie poprawiało w sposób znaczący efektywności. Po prostu im więcej stacji, tym dłużej trwały owe okresy bezczynności. W nowym algorytmie zwiększenie ilości stacji zmniejsza rozmiar lokalnej części pliku drugiego w każdej stacji (a co za tym idzie, czas jej przeszukiwania), a rekordy pliku pierwszego są rozgłaszane, więc żadna ze stacji nie stoi bezczynnie.

Pewien problem może stworzyć duża ilość kolizji na łączu, gdy wszystkie stacje równocześnie zechcą wysyłać wyniki swojego działania. Można temu zaradzić wprowadzając nowe mechanizmy, synchronizujące wysyłanie gotowych rekordów do serwera.

3.4. Zmodyfikowany wariant trzeci

We wstępie do tego punktu wspomnieliśmy o zastosowaniu indeksów przy wykonywaniu złączenia, odrzucając tę ideę jako ograniczającą formułowanie warunku złączenia lub wprowadzającą ogromne narzuty czasowe. Zauważmy jednak, że o ile przy złączeniu tradycyjnym, każdorazowe budowanie indeksu jest niczym innym, jak tylko marnotrawstwem

czasu, o tyle przy złączeniu rozproszonym może ono przynieść dodatkowe, pozytywne efekty.

Zwróćmy uwagę na sposób, w jaki w naszym algorytmie rozsyłane są rekordy pliku drugiego. Stacja, która otrzymała taki rekord, musi go tylko dopisać do lokalnej części kopii pliku drugiego. Po tej operacji ma jeszcze sporo wolnego czasu dla siebie, tym więcej, im więcej stacji bierze udział w łączeniu, ponieważ rekordy są wysyłane kolejno do wszystkich stacji.

Ten wolny (czyli w rezultacie stracony) czas stacji można wykorzystać właśnie na budowanie cząstkowych indeksów pliku drugiego na dyskach lokalnych. Indeksy te byłyby następnie wykorzystywane podczas sprawdzania warunku złączenia dla nadsyłanych z serwera rekordów pliku pierwszego. Rozwiązanie takie kapitalnie przyspieszyłoby wyszukiwanie rekordów do łączenia, czyli najbardziej kosztowną, jeżeli chodzi o czas, część łączenia.

Niestety, modyfikacja ta zwiększa zapotrzebowanie na przestrzeń na lokalnych dyskach stacji, jako że oprócz buforowanych rekordów pliku drugiego, przechowywano by na nich również struktury indeksów. Ograniczenie to można, oczywiście, ominąć pozwalając, aby w sytuacji braku miejsca na dysku lokalnym, stacja kontynuowała pracę korzystając z dysku sieciowego. Spowoduje to jednakże obniżenie efektywności całego systemu z przyczyn podanych na początku tego punktu.

4. Zakończenie

W pracy przedstawiono analizę możliwości wykorzystania sieci komputerowej do zrównoleglenia obliczeń w zadaniach wyszukiwania danych.

Przedyskutowano przydatność różnych klas języków zapytań do zapisu zadań wyszukiwania w formie umożliwiającej rozpraszanie. Wskazano na algebrę relacji jako poziom zapisu zadań szczególnie przydatny do tych celów.

Przedstawiono wyniki praktycznych badań dotyczących równoległego wyszukiwania w wybranej bazie danych, przeprowadzonych w sieci Novell NetWare 3.11, potwierdzających możliwość uzyskania wyraźnych oszczędności czasowych.

W pracy przedstawiono również projekt rozproszonej realizacji wyszukiwania dla modelu tzw. relacji uniwersalnej. Szczególną uwagę zwrócono przy tym na analizę możliwości równoległej realizacji operacji złączenia, kluczowej w relacyjnym modelu danych.

LITERATURA

[1] Korth H.F., Kuper G.M., Feigenbaum J., van Gelder A., Ullman J.D.: System/U: A Database System Based on the Universal Relation Assumption. ACM TODS, 9, 3, 1984.

[2] Kozielski S., Piec J., Grzywocz J.: System wyszukiwania danych oparty na modelu relacji uniwersalnej, Archiwum Informatyki Teoretycznej i Stosowanej, 1993, (w druku).

[3] Ullman J.D.: Principles of Database Systems. Computer Science Press, Rockville, 1983. Polskie wydanie: Systemy baz danych. WNT. Warszawa 1988.

[4] Ullman J.D.: Database and Knowledge-Base Systems. Computer Science Press, Rockville 1989.

[5] Wilk A.: Wykorzystanie sieci NetWare firmy Novell do tworzenia oprogramowania współbieżnego na przykładzie implementacji równoległego algorytmu rozwiązania

uproszczonego problemu Hilberta, Gliwice 1990 (opracowanie niepublikowane).

Recenzent: Dr inż. Andrzej Wilk

Wpłynęło do redakcji 27.09.1993 r.

Abstract

Several possibilities of distributed data retrieving in local area networks are presented in the paper.

Section 1 presents problems encountered and conditions that must be met for distributed database processing to make it run significantly faster than in single computer environment.

Section 2 divides existing query languages into several classes and considers their efficiency in distributed processing. Methods of distribution and examples illustrating efficiency are presented for every class.

Model of distributed database processing composed with processing nodes and record streams, connecting these nodes is considered in the next section.

Possibilities of distributed joining in local area networks are given in the last section. Authors present three algorithms evolving from single computer environment to LAN environment to utilize specific network possibilities and to avoid bottelnecks such as server or link overflows.