STUDIA INFORMATICA 2010
Volume 31 Number 2B (90)

Paulina BRZOZOWSKA, Miłosz GÓRALCZYK, Łukasz JESIONEK,
Marcin KARPIŃSKI, Grzegorz KĘDZIERSKI, Piotr KĘDZIERSKI,
Jarosław KOSZELA, Emil WRÓBEL
Wojskowa Akademia Techniczna, Instytut Systemów Informatycznych

SYSTEM OBIEKTOWY = OBIEKTOWA BAZA DANYCH + OBIEKTOWA APLIKACJA

Streszczenie. Praca prezentuje koncepcje i wykonane elementy rozproszonego obiektowego systemu baz danych (DOOD System[1]), który stanowi podstawę do badań związanych z unifikacją procesu projektowania i implementacji rozproszonych systemów obiektowych, wraz z opracowaniem języka programowania łączącego w sobie własności języka deklaratywnego i imperatywnego. Język tego typu pozwoli na zautomatyzowanie procesu rozproszonego przetwarzania danych przy wykorzystaniu statycznych i dynamicznych mechanizmów szacowania kosztów przetwarzania. Prezentowana jest również koncepcja architektury DOOD wraz z podstawowymi elementami, np. mechanizm zarządzania, skład obiektów, mechanizm indeksowania, analizy języka, generator planów i środowisko wykonawcze.

Słowa kluczowe: system baz danych, rozproszenie danych, rozproszone przetwarzanie, obiektowe bazy danych, obiektowe języki zapytań

OBJECT SYSTEM = OBJECT DATABASE + OBJECT APPLICATION

Summary. This article presents main ideas and implemented elements of distributed object-oriented database system (DOOD). Military University of Technology researches on unification of project and development of object systems with dedicated programming language is based on DOOD system. Provided language, that supports both imperative and declarative features, allows automatic distribution of tasks, mainly because of implemented static and dynamic cost counting algorithms. This article also introduces some key elements like indexes, management console, language analysis, query planner or execution.

Keywords: database system, distribute data, distributed computing, object database, object query language

[1] DOOD System– ang. Distrubuted Object-Oriented Database System

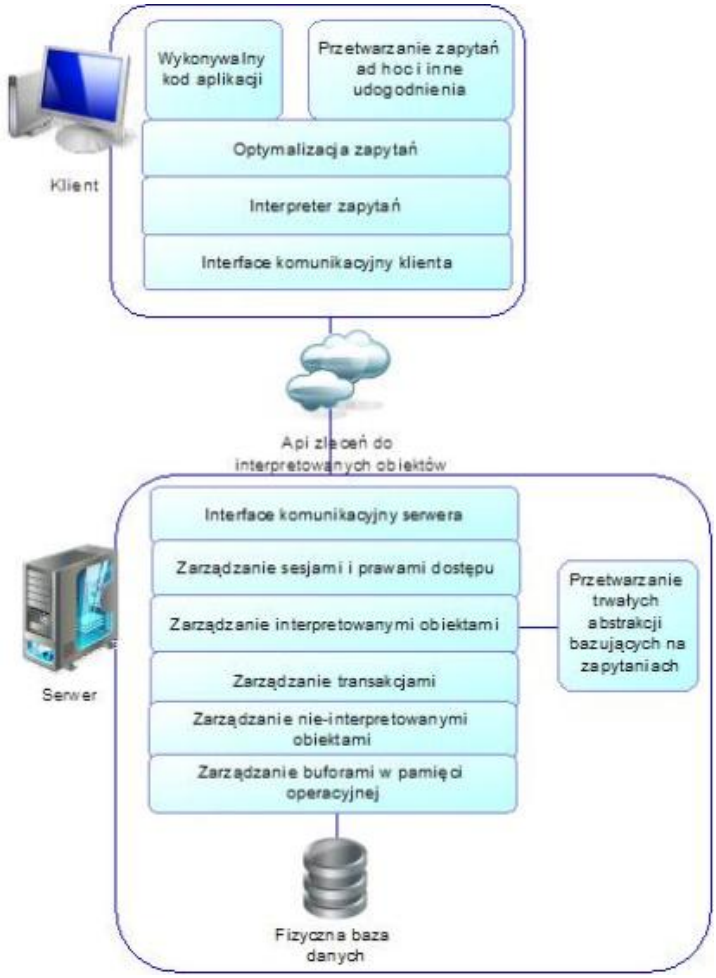
1. Wstęp

Jednym z głównym obszarów informatyki od czasu jej powstania było i jest przetwarzanie i gromadzenie danych. Początkowo aplikacje wykorzystywały swoje własne źródła danych zlokalizowane w pamięci komputera, co było wygodne, jednak z racji małej efektywności i reużywalności szybko wyewoluowało w koncepcję odseparowania danych od aplikacji, a w konsekwencji do wydzielenia osobnej warstwy – baz danych. Rozwój baz danych i współczesnych języków programowania w odmiennych kierunkach i z różną prędkością spowodował jednak problem ze zgodnością technologiczną obu rozwiązań. Podstawowym celem stało się więc znalezienie rozwiązania, które łączyłoby zalety zewnętrznego źródła danych z wygodą korzystania zbliżoną do operowania na danych lokalnych. Zwiększone zapotrzebowanie na przetwarzanie dużych woluminów danych sprawiło, że kolejnym celem stało się szybkie i efektywne przetwarzanie danych, na co sposób upatruje się w systemach rozproszonych, a więc zestawach maszyn zdolnych wspólnie ominąć ograniczenia sprzętowe. Niestety, zmusiło to także programistów do odgórnego planowania, co i jak będzie przetwarzane w środowisku rozproszonym. Dlatego podstawowym założeniem systemu DOOD stało się uwolnienie programistów od konieczności przejmowania się sposobem dostępu do danych, zestawiania połączeń z bazą czy definiowania sposobu rozproszonego przetwarzania jego żądań. System DOOD ma przede wszystkim automatyzować oba procesy. Element łączący bazę i aplikację kliencką ma być do tego stopnia rozbudowany, aby programista nie odczuwał różnicy pomiędzy pracą na obiektach lokalnych i zdalnych. To system ma zatroszczyć się o to, by jeśli to możliwe, metody wywoływane na obiektach zdalnych były wykonywane przez wiele jednostek, a więc w środowisku rozproszonym. To system powinien martwić się, skąd należy pobrać potrzebne dane. Programista nie powinien być świadom tego, czy jego żądanie zostało wykonane w środowisku jednomaszynowym, czy rozproszonym przez jego lokalną maszynę lub zdalną bazę.

2. Mechanizmy integracji programowego dostępu do danych

Wygodnym narzędziem programistycznym można nazwać takie, które spełnia warunek – działanie tego rozwiązania powinno być jak najmniej widoczne bądź odczuwalne dla użytkownika. Nie może on być świadom, że obiekt, na którym właśnie pracuje, jest elementem pobranym ze zdalnego źródła, a nie z zasobów jego maszyny, czy też, że metoda countSallary obiektu pracownik została wykonana przez 3 maszyny o różnych lokalizacjach. Do osiągnięcia tego celu prowadzić może wiele dróg, jednak do momentu, kiedy programista, na poziomie pisania kodu, nie będzie świadom, że wykorzystał tak zaawansowane rozwiązanie,

nie będzie można mówić o pełnym sukcesie. Jak jednak pogodzić imperatywny obiektowy język programowania z deklaratywnymi żądaniami do bazy, które z jednej strony zamykają funkcjonalność bazy danych do skończonego zbioru operacji, a więc idealnej sytuacji do automatyzacji, z drugiej zaś, jako niekompletne obliczeniowo, blokują możliwość wykonania niektórych operacji. Jedynym rozwiązaniem jest synteza obu charakterystyk w jednym języku. Osiągnąć to można zarówno poprzez wytworzenie całkowicie nowego języka łączącego cechy obu typów, jak i poprzez rozbudowę natywnych konstrukcji już istniejącego języka o elementy pozwalające na operacje na danych. System DOOD wyposażony został w oba rozwiązania. W systemie tym podjęto działania, aby zintegrować go z językiem platformy.NET – C# 4.0. W celu zrealizowania tej koncepcji wykorzystano mechanizm budowania obiektów w trakcie działania programu z wykorzystaniem typu *dynamic*.



Rys. 1. Pożądany podział wykonania zapytania w DOOD
Fig. 1. Desired query's execution partition in DOOD

Zrealizowane zostały elementy pozwalające na przenoszenie obiektów bazodanowych do aplikacji klienckiej oraz translacje typów prostych. Rozwiązanie to wymagało jednak jawnego wyspecyfikowania treści zapytania w języku SBQL. Dalsze działania w tym kierunku będą zmierzały do całkowitego ukrycia tego zagadnienia przed użytkownikiem końcowym,

w taki sposób, aby znając składnie języka C# mógł w prosty sposób odwoływać się do obiektów zgromadzonych w bazie danych. Ważnym elementem było wspieranie specyficznych cech języka, przy ukryciu jednocześnie różnic metamodelu bazy danych i wybranego języka programowania, np. kwestii wielodziedziczenia. Docelowy podział etapów wykonania zadania, który ma być osiągnięty w aplikacjach wykorzystujących DOOD System zawiera pozycja [3].

3. Integracja języka deklaratywnego i imperatywnego

Przykładem języka, który posłużył także za pierwowzór elementów języka wykorzystywanego w DOOD, jest Stack-Based Query Language, czyli język oparty na podejściu stosowym, które zakłada, że języki zapytań są szczególnym przypadkiem języków programowania. Do głównych cech języka SBQL zaliczyć możemy m. in.:

- zapytania przedstawiane są za pomocą wyrażeń;
- możliwość stosowania zapytań również do danych nietrwałych;
- założenie, iż język nie posiada innych wyrażeń niż zapytania;
- wykonywanie zapytań w trybie iteracyjnym, dynamiczne wiązanie, brak fazy kompilacji i konsolidacji zapytań z całością aplikacji.

```
SELECT Nazwisko          Osoba(WHERE
FROM Osoba               Wiek>18).Nazwisko
WHERE Wiek > 18
        SQL                      SBQL
```

Rys. 2. Przykładowe zapytanie w SQL i SBQL
Fig. 2. Queries example

Ponadto użycie SBQL otwiera drogę do wykorzystania innej architektury połączenia aplikacja-baza danych. Zaproponowana architektura przedstawiona została na rysunku 1. Porównanie zapytań (pobierających listę osób pełnoletnich) na przykładzie języków SQL i SBQL przedstawiono na rysunku 2.

W ramach projektowanego systemu dokonano częściowej specyfikacji obiektowego języka zapytań opartej o funkcjonalności prezentowane przez język SBQL. Opracowana gramatyka pozostawia miejsce na rozszerzenie elementów języka w oparciu o wybrany obiektowy język programowania. Dotychczas zaimplementowano części systemu, które dokonują analizy syntaktycznej i semantycznej zapytań.

4. Architektura systemu zarządzania bazą danych

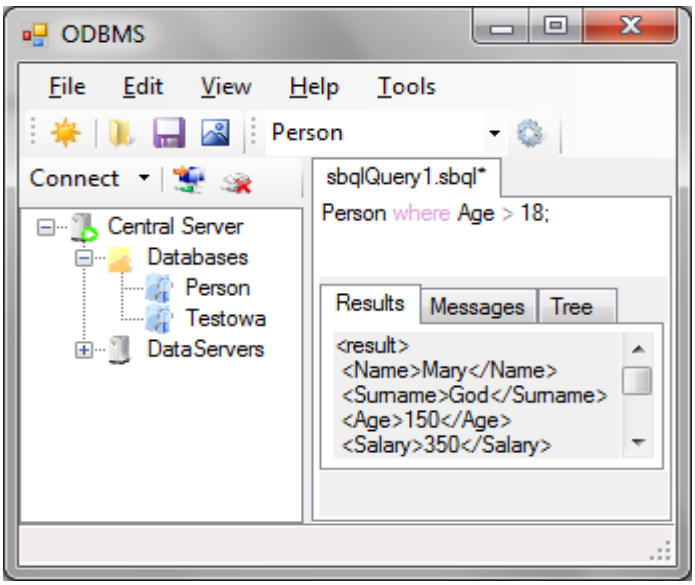
Pełna integracja języka zapytań i języka programowania pozwala na zastosowanie nowego ciekawego podejścia do architektury Systemu Zarządzania Bazą Danych. Tradycyjne rozwiązania oparte są głównie na architekturze klient-serwer, w której klient wykorzystywany jest w tak zwanym wariancie cienkiego klienta. Oznacza to, że w tradycyjnych rozwiązaniach to serwer wykonuje większość operacji procesu wykonania zapytania. Zintegrowany język zapytań taki, jak stosowany będzie w DOOD, umożliwia wykorzystanie klienta w dużo szerszym spectrum. Zastosowanie podejścia grubego klienta pozwala na odciążenie serwera z części operacji przeprowadzanych przed samym wykonania zapytania i przeniesienie ich na maszynę klienta. Przykładowy podział etapów wykonania zapytania został przedstawiony na rysunku 1.

Zaproponowana architektura SZBD przenosi dwa etapy procesu wykonania, tj. interpretację zapytań oraz ich optymalizację na maszynę kliencką. Taki zabieg przełoży się na zwiększenie liczby jednocześnie obsługiwanych zapytań przez serwer baz danych. Niestety, tego typu architektura niesie za sobą konieczność wyposażenia bazy czy też systemu bazodanowego, jakim jest DOOD w mechanizmy dostarczające klientowi metainformacji. Chcąc wykorzystać maszynę kliencką do przygotowania zapytania, a więc do tego, by była zdolna przeanalizować zapytanie i wygenerować instrukcje interpretowalne przez silnik bazy, musi ona mieć dostęp do takich danych przechowywanych na serwerze, jak: schemat bazy, metody oraz indeksy, informacje o aktualnym stanie elementów środowiska bazy danych, inne informacje z metamodelu – tj. klasy, typy, które mogą zostać wykorzystane w programie, a także pozwolą kontrolować poprawność zapytań.

Niezależnie od stopnia zaawansowania systemu czy bazy danych musi istnieć element umożliwiający sprawowanie kontroli nad jego działaniem. Tradycyjnym rozwiązaniem jest kontrola systemu bazodanowego poprzez język zapytań. Język taki musi więc być wyposażony w odpowiednie słowa kluczowe. W obecnej wersji tworzonego systemu bazodanowego zaimplementowane zostały mechanizmy odpowiedzialne za przesyłanie oraz przechowywanie ustawień (serwerów). Obsługiwane są zarówno parametry trwałe (np. nazwa serwera), jak i tymczasowe – ważne w trakcie działania jednej sesji użytkownika, np. określające formę zwracania wyników.

W celu ułatwienia współpracy z systemem została również zaprojektowana i zaimplementowana dedykowana aplikacja zarządzająca. Pozwala ona na kontrolę i modyfikację stanu elementów systemu. Ponadto aplikacja ta umożliwia bezpośrednie zadawanie, konstruowanie i wykonywanie zapytań. Oprogramowanie zarządzające systemem DOOD oparte jest na mechanizmie wtyczek, co umożliwia rozbudowywanie aplikacji o nowe funkcjonalności

w prosty, dynamiczny sposób. Zrzut ekranu z omawianej aplikacji został przedstawiony na rys. 3.



Rys. 3. Aplikacja zarządzająca DOOD
Fig. 3. DOOD's managment studio

5. Elementy wykonania zapytania w rozproszonym obiektowym środowisku

Dwa podstawowe aspekty – rozproszenie i obiektowość, które są nierozerwalnie związane z systemem DOOD, narzucają znaczną zmianę architektury całego systemu. Dotyczy to również mechanizmów, które odpowiadają za przetwarzanie zapytań. Przedstawione poniżej mechanizmy uwzględniają również fakt, iż obiekty przechowywane w bazie danych powinny mieć taką samą strukturę, jak obiekty przechowywane w pamięci operacyjnej aplikacji.

5.1. Skład obiektów oraz wykonywanie metod

Składowanie obiektów wiąże się nie tylko z zapisaniem wartości poszczególnych atrybutów – takie podejście byłoby właściwe tylko dla obiektów prostych. Wymóg ujednolicenia struktury obiektów wymusza również umożliwienie przechowywania referencji na inne obiekty (np. referencja na producenta w obiekcie produktu), co więcej, powinno być możliwe zapisanie listy referencji do większej liczby obiektów pewnego typu (lista referencji na produkty danego producenta). Przechowywanie referencji prowadzi również do przechowywania złożonych struktur danych. Należy tu również zwrócić uwagę, iż popularne typy proste (np. int, char) też są obiektami, więc tak naprawdę każdy obiekt utworzony przez programistę

będzie zawierał inne obiekty – będzie obiektem złożonym. Podobna sytuacja będzie w przypadku dziedziczenia obiektów – wewnątrz potomnego obiektu przechowywany jest (niejawnie) obiekt klasy macierzystej. Zachowanie struktury obiektów przechowywanych w pamięci operacyjnej wymaga utworzenia własnego sposobu odwzorowania referencji – niemożliwe jest przechowanie "wprost" fizycznego adresu obiektów w pamięci operacyjnej. System DOOD tworzy metaelement OID, czyli unikalny identyfikator, który nadawany jest każdemu obiektowi przechowywanemu w bazie danych. Zatem przechowanie referencji na inny obiekt sprowadza się do przechowania OID-u zależnej klasy. Dopiero przy odtwarzaniu (obiektów) odwołania do konkretnych OID-ów będą zamieniane na fizyczny adres poszczególnych obiektów w pamięci operacyjnej. Konstrukcja referencji oparta na OID-ach umożliwia również podział (poziomy i pionowy) obiektów oraz rozmieszczenie poszczególnych elementów na innych maszynach systemu niż obiekt główny.

Oprócz atrybutów w obiektowej bazie danych muszą być składowane również metody klas. Warto podkreślić, iż wykorzystanie metod obiektów, które mogą być współdzielone przez kilka aplikacji, będzie tak naprawdę realizacją architektury zorientowanej na usługi (SOA). Należy również rozważyć problem, czy metody wykonywane będą po stronie serwera, czy ich wykonanie (wraz z wymaganymi danymi) przesłane będzie do klienta, co zapewne zredukuje obciążenie serwera.

Obecny poziom rozwoju systemu DOOD pozwala na składowanie obiektów wraz z referencjami na inne (pojedyncze) obiekty (w tym realizację dziedziczenia). Następnym etapem rozwoju składu danych będzie przechowywanie obiektów zawierających więcej niż jeden obiekt. Ponadto dodana powinna być funkcjonalność przechowywania i przetwarzania metod obiektów wraz ze sposobem określenia (przez użytkownika) maszyny, na której wykonywane będą obliczenia.

5.2. Rozproszone indeksy

Z założenia każdy byt utworzony w DOOD jest obiektem – prawidłowość ta dotyczy również klas. Dlatego wszystkie obiekty, które są utworzone na podstawie definicji danej klasy, tworzą jej ekstensje. Obiekty utworzone bezpośrednio na podstawie definicji danej klasy należą do kategorii ekstensji bezpośredniej, a te, które wykorzystują operacje dziedziczenia, należą do klasy ekstensji pośrednich. Przy próbie nałożenia indeksu na atrybut klasy pochodnej możliwe są następujące scenariusze działania systemu, polegające na utworzeniu struktury indeksującej: łącznej dla obiektów klasy macierzystej i dziedziczącej; wyłącznej dla obiektów klasy dziedziczącej; zarówno dla obiektów klasy macierzystej i dziedziczącej, jak i oddzielnej dla obiektów klasy dziedziczącej.

Z uwagi na obiektowe zorientowanie bazy danych, każda struktura indeksowa nałożona jest na dany atrybut należący do określonej klasy. Atrybut ten (będący obiektem) posiada własny identyfikator (OID) oraz wskaźnik do odpowiedniej struktury organizacji danych związanej z ekstensjami klasy, do której należy zaindeksowany atrybut. Struktury danych zawierają niepowtarzające się elementy, a z każdym z nich powiązana jest lista OID określających obiekty, których wartość zaindeksowanego atrybutu odpowiada wartości klucza w węźle struktury.

Rozproszenie systemu bazodanowego ma na celu przyśpieszenie operacji wykonywanych na nim. Pociąga ono za sobą jednak problemy związane z zachowaniem spójności i aktualności zapisanych w nim danych. Problem ten dotyczy w dużej mierze mechanizmu indeksowania. Opisane wcześniej struktury mogą znajdować się na oddzielnych serwerach – serwer główny zawiera wszystkie struktury związane z mechanizmem indeksowania i bezpośrednio realizuje operacje wyszukania zaindeksowanego obiektu. Rozwiązanie to gwarantuje minimalną ilość obciążenia sieci i nie wymaga synchronizacji, gdyż mechanizm indeksowania jest przechowywany wyłącznie na pojedynczej maszynie. Wadą tego rozwiązania może być jednak zagrożenie zbytniego obciążenia serwera głównego i w efekcie powstanie „wąskiego gardła".

Warto zwrócić uwagę, iż serwer główny przechowuje wyłącznie strukturę zawierającą OID zaindeksowanych obiektów oraz powiązanych z nimi wskaźników na struktury danych, które przechowywane są na serwerach danych. Rozwiązanie to znacząco zmniejsza obciążenie serwera głównego, poprzez wykonywanie operacji wyszukania elementu w strukturze na serwerach danych, które te struktury przechowują. Wadą takiego rozwiązania jest zwiększenie ruchu sieciowego i konieczność częstych aktualizacji tych struktur, w przypadku usuwania, dodawania lub modyfikacji obiektów, których atrybuty są zaindeksowane.

Zaimplementowane rozwiązanie opiera się na zastosowaniu jednego składu indeksów przechowywanych na serwerze centralnym. Jako struktura organizacji danych wykorzystane zostało drzewo poszukiwań binarnych. Rozwiązanie to przyniosło spodziewany efekt znaczącego przyśpieszenia wykonywania zapytań w bazie danych, jednocześnie zastosowanie bardziej odpowiednich do indeksowania struktur może jeszcze dodatkowo przyśpieszyć wyszukiwanie kilkakrotnie. W przyszłych etapach rozwoju systemu DOOD zaimplementowane zostaną rozwiązania opierające się na idei indeksów lokalnych (na serwerach danych), a jako strukturę organizacji danych wykorzystane zostaną B-drzewa. W efekcie powinno to wpłynąć na wzrost wydajności całego systemu.

5.3. Przetwarzanie zapytań

Charakterystyczną cechą baz danych, wynikającą także z zastosowanych języków programowania, jest ich deklaratywna natura. Stosując takie podejście, to nie użytkownicy czy operatorzy muszą się martwić, jak wydobyć z bazy potrzebne im dane. Zadanie wydobycia danych to zadanie silnika bazy. Operacje na danych muszą przebiegać możliwie sprawnie i wydajnie. Dlatego mechanizm planowania zapytań (Query Planner – QP) musi generować optymalne plany wykonania zapytania.

Rola query plannera w rozproszonych bazach danych ulega rozszerzeniu. Ważną kwestią, o którą query planner musi zadbać w takiej bazie danych, jest wykorzystanie mocy obliczeniowej węzłów wchodzących w skład takiej bazy. Ilość i różnorodność planów zapytań w takiej sytuacji wzrasta gwałtownie, ponieważ dochodzi jeszcze jedna ważna zmienna – maszyna wykonująca zapytanie (podzapytanie). Ponadto musi również zostać rozwiązana kwestia zapytań kaskadowych. Każde wywołane zapytanie może wywołać kolejne, co może prowadzić do nieskończonych pętli. Należy ustrzec system przed błędami wynikającymi z cyklicznych wywołań oraz wywołań rekurencyjnych.

Generując plany zapytań, QP musi wziąć pod uwagę dodatkowe dane związane z rozmieszczeniem danych na wielu maszynach. Przykładem niech będzie tutaj żądanie wydobycia wszystkich właścicieli pojazdów konkretnej marki. Przede wszystkim wiele maszyn może posiadać dane o pojazdach, jak również wiele maszyn może dysponować zapisami o właścicielach. W generowanych planach zapytania musi się więc znaleźć informacja o tym, na którym węźle operacja powinna być wykonana, np. dlatego, iż posiada najwięcej obiektów danego typu bądź nie jest (w danym momencie) obciążony.

Ważnym aspektem towarzyszącym rozproszonemu (z racji operowania na rozproszonej bazie danych) planiście zapytań są podzapytania. Stanowią one podstawowe punkty rozdziału zadań na poszczególne węzły (również uwzględniając rozmieszenie obiektów). Niezależnie od tego, czy zapytanie posiada, podzapytania, czy nie, istnieje kilka technik podziału zapytań na mniejsze elementy.

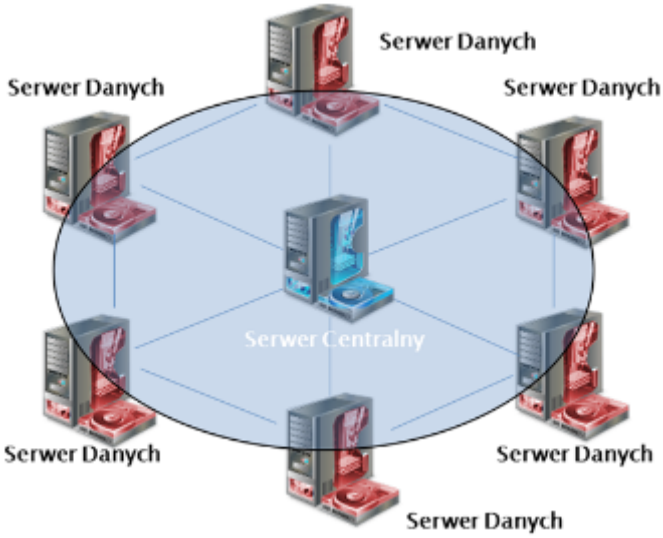
Pierwszą z nich jest statyczna dekompozycja, która generuje rozłączne zadania dla każdego z węzłów, które te po prostu wykonują. W opozycji do tej metody jest tak zwana dekompozycja dynamiczna. Zadania dla poszczególnych węzłów nie są generowane równocześnie, lecz sekwencyjnie – tj. na bazie zapytania wejściowego tworzone jest podzapytanie, które wysyłane jest do jednego z węzłów, kolejne podzapytania tworzone są na bazie już posiadanych danych oraz danych zwróconych przez ostatnie podzapytanie. Można sądzić, że tracona jest tutaj równoległość przetwarzania, co jednak okazuje się złudzeniem. Faktycznie, to podejście uniemożliwia wykonanie jednego zapytania równolegle przez wiele maszyn, jednak dzieląc duże zadania na podzadania dajemy węzłom możliwość równoległego przetwarzania wielu mniejszych elementów. Zamiast obciążać każdy z węzłów dużymi, a wła-

ściwie długimi zadaniami, możemy prosić o wykonanie tylko jakiejś części dużego zadania, dzięki czemu taki węzeł po krótkim czasie może przystąpić do pracy nad częścią innego zadania.

Na obecnym etapie rozwoju w DOOD zaimplementowany jest lokalny QP, tzn. generujący plany zapytań tylko dla obiektów rozmieszczonych na lokalnej maszynie. Następnym etapem będzie rozszerzenie funkcjonalności QP o generowanie planów w rozproszonym środowisku wraz z uwzględnieniem liczności obiektów na poszczególnych maszynach oraz ich aktualnego obciążenia.

6. Rozproszenie i replikacja obiektów

Nieustannie rosnąca ilość gromadzonych danych doprowadziła do sytuacji, w której utrata zdobytych danych wiązałaby się z dużymi stratami, czyli czymś, na co nikt nie może sobie pozwolić. Doprowadziło to do potrzeby ochrony danych, a jedynym rozwiązaniem, które mogło zapobiec utracie cennego materiału, było wykonanie jego kopii, która nie mogła znajdować się w tym samym miejscu, co oryginał, bo narażona byłaby na te same destrukcyjne czynniki, co jej pierwowzór. Powstawały więc nie tylko kopie danych, ale i samych maszyn, na których dane były gromadzone. Szybko jednak zauważono, że takie maszyny nie muszą być biernymi magazynami i także mogą brać udział w przetwarzaniu danych. W ten sposób narodziła się koncepcja rozproszenia. Każda baza danych z racji swojego zadania powinna więc oferować możliwość replikacji swojej zawartości na większą liczbę maszyn, chociażby w celu ich ochrony.



Rys. 4. Rozproszone hosty DOOD
Fig. 4. Distribution of DOOD's hosts

Zaproponowany w DOOD System mechanizm replikacji został zaprojektowany tak, aby nie ograniczał możliwości rozproszenia tylko do ochrony danych, ale także umożliwiał wykorzystanie potencjału obliczeniowego zapasowych maszyn. U podstaw zaproponowanego mechanizmu leży hybrydowa architektura komunikacji międzywęzłowej, która została przedstawiona na rysunku 4. Architektura ta charakteryzuje się istnieniem węzła centralnego, który pełni rolę koordynatora całego systemu. Serwer centralny jest jednostką logiczną zbudowaną w oparciu o usługę sieciową, która w obecnej wersji systemu pracuje na jednej fizycznej maszynie. Docelowo funkcjonalność serwera centralnego ma zostać rozproszona na wszystkie jednostki pracujące w bazie. Brak obecności takiego węzła niósłaby za sobą konsekwencje w postaci trudności z utrzymaniem bazy w spójnym stanie, jak również z utrzymaniem sekwencyjności w dostępie do danych. Kolejnym założeniem towarzyszącym prezentowanemu rozwiązaniu było sprowadzenie działania mechanizmu replikacji i synchronizacji na poziom pojedynczego obiektu, nie zaś, jak to występuje na przykład w bazie w niektórych współczesnych systemach baz danych, pozostawienie go na poziomie zapytań.

W tradycyjnym rozwiązaniu mechanizm replikacji zajmowałby się tylko wykonywaniem zapytań, wykonanych na serwerze głównym, na serwerach pobocznych. Zaproponowane rozwiązanie schodzi o poziom niżej. W obu podejściach elementem replikowanym byłby obiekt. Jednak w odróżnieniu od tradycyjnego podejścia, w zaproponowanym rozwiązaniu podmiotem replikacji byłby pojedynczy obiekt, nie zaś grupa obiektów, których dotyczy zapytanie.

7. Podsumowanie

Niniejsza praca przedstawia koncepcje i wykonane elementy rozproszonego obiektowego systemu baz danych (DOOD System), który stanowi podstawę do badań związanych z unifikacją procesu projektowania i implementacji systemów obiektowych, wraz z opracowaniem języka programowania łączącego w sobie własności języka deklaratywnego i imperatywnego. Język tego typu pozwoli na zautomatyzowanie procesu rozproszonego przetwarzania danych przy wykorzystaniu statycznych i dynamicznych mechanizmów szacowania kosztów przetwarzania. Prezentowana jest również koncepcja architektury DOOD wraz z podstawowymi elementami, np. mechanizm zarządzania, skład obiektów, mechanizm indeksowania, analizy języka, generator planów i środowisko wykonawcze.

BIBLIOGRAFIA

1. Subieta, K.: Teoria i konstrukcja obiektowych języków zapytań. Wydawnictwo PJWSTK, Warszawa 2004.
2. Lausen G., Vossen G.: Obiektowe bazy danych Modele danych i języki. Warszawa 2000.
3. Góralczyk M.: Projekt oprogramowania zarządzającego obiektową bazą danych. Praca dyplomowa WAT, Warszawa 2010.
4. Karpiński M.: Projekt mechanizmu replikacji i synchronizacji elementów obiektowej bazy danych. Praca dyplomowa WAT, Warszawa 2010.
5. Brzozowska P.: Projekt analizatora syntaktycznego i semantycznego obiektowego języka zapytań. Praca dyplomowa WAT, Warszawa 2010.
6. Jesionek Ł.: Projekt generatora planów wykonania zapytania. Praca dyplomowa WAT, Warszawa 2010.
7. Wróbel E.: Projekt interfejsu programistycznego dostępu do obiektowej bazy danych. Praca dyplomowa WAT, Warszawa 2010.
8. Kędzierski G.: Projekt indeksów w obiektowej bazie danych. Praca dyplomowa WAT, Warszawa 2010.

Recenzenci: Dr hab. inż. Tadeusz Wieczorek, prof. Pol. Śląskiej

Wpłynęło do Redakcji 31 stycznia 2010 r.

Abstract

This article contains the description of the distributed, object database system with the special focus on its main attributes. The mechanism of program integrated data access and the desirable division of the query realization plan has been presented on the picture 2.1. The query language which has been used in this solution is based on the stack approach and the example has been put in the listing 3.1. The distribution mechanism is based on the client-server architecture, with the approach of the thick-client and the concept of using one main mother-server and many data servers. The described situation has been presented on the picture 6.2. The replication mechanism implemented in the system guarantees that the consistency of the stored data. The user can communicate with the system through the management application presented on the picture 4.1. The object character of the solution can be seen in the way data is stored in the system – it has the exact same structure as if it was stored in RAM. The solution also specifies the way of creating complex types object types (i.e. class

inheritance). The optimization mechanism is based mainly on usage of indexing structures (the local ones stored on the client's machine or the data servers and the global index structure stored on the main server) and the realization of the optimal query realization plan (thorough the ones generated in the time-limited generation process). The other important aspects of the optimization process consist of: usage of metadata, distributed Job division and the sub-query planning.

Adresy

Jarosław KOSZELA :Wojskowa Akademia Techniczna im. Jarosława Dąbrowskiego, ul. Gen. Sylwestra Kaliskiego 2, 00-908 Warszawa , Polska, jkoszela@wat.edu.pl.
Piotr KĘDZIERSKI : Wojskowa Akademia Techniczna im. Jarosława Dąbrowskiego, ul. Gen. Sylwestra Kaliskiego 2, 00-908 Warszawa , Polska, pkedzierski@wat.edu.pl.
Miłosz GÓRALCZYK : Wojskowa Akademia Techniczna im. Jarosława Dąbrowskiego, ul. Gen. Sylwestra Kaliskiego 2, 00-908 Warszawa , Polska, milosz.goralczyk@gmail.com.
Marcin KARPIŃSKI :Wojskowa Akademia Techniczna im. Jarosława Dąbrowskiego, ul. Gen. Sylwestra Kaliskiego 2, 00-908 Warszawa , Polska, marcin@karpinski.waw.pl.
Paulina BRZOZOWSKA : Wojskowa Akademia Techniczna im. Jarosława Dąbrowskiego, ul. Gen. Sylwestra Kaliskiego 2, 00-908 Warszawa , Polska, p.j.brzozowska@gmail.com.
Łukasz JESIONEK: Wojskowa Akademia Techniczna im. Jarosława Dąbrowskiego, ul. Gen. Sylwestra Kaliskiego 2, 00-908 Warszawa , Polska, lukas.jesionek@gmail.com.
Emil WRÓBEL :Wojskowa Akademia Techniczna im. Jarosława Dąbrowskiego, ul. Gen. Sylwestra Kaliskiego 2, 00-908 Warszawa , Polska, wrobel.emil@gmail.com.
Grzegorz KĘDZIERSKI :Wojskowa Akademia Techniczna im. Jarosława Dąbrowskiego, ul. Gen. Sylwestra Kaliskiego 2, 00-908 Warszawa , Polska, drkedzierski@gmail.com.