STUDIA INFORMATICA 2002
Volume 23 Number 4 (51)

Katarzyna HARĘŻLAK, Henryk JOSIŃSKI
Politechnika Śląska, Instytut Informatyki

DOSTĘP DO ROZPROSZONYCH BAZ DANYCH Z WYKORZYSTANIEM AGENTÓW PRZENOŚNYCH

Streszczenie. Artykuł definiuje pojęcie agenta przenośnego, charakteryzuje jego cechy, mechanizmy działania i obszary zastosowań. Zawiera ponadto przegląd środowisk do tworzenia agentów przenośnych. Skoncentrowano się na zastosowaniu technologii agentów przenośnych do realizacji zapytań i transakcji w rozproszonej bazie danych. Zaproponowano możliwości wykorzystania przenośnych agentów w eksperymentalnym systemie zarządzania rozproszoną bazą danych.

ACCESS TO DISTRIBUTED DATABASES APPLYING MOBILE AGENTS

Summary. The paper presents a definition of the mobile agent, its features, mechanisms, and application areas. It contains also an overview of environments for creation and management of mobile agents. The authors discuss the processing of queries and transactions in distributed databases using the mobile agent technology. The possibilities of its application in the experimental distributed database management system are also considered in the paper.

1. Wstęp

Technologia przenośnych agentów stanowi alternatywę dla innych sposobów programowania rozproszonego. Przenośni agenci są programami komputerowymi, które mogą być rozesłane razem z ich danymi z węzła lokalnego do zdalnego wykonania w tym węźle, który gwarantuje najlepsze parametry wykonania. Technologia przenośnych agentów wydaje się być obiecująca, jeśli chodzi o wykorzystanie systemów rozproszonych w heterogenicznych sieciach otwartych.

Niniejsze opracowanie zawiera przegląd środowisk do tworzenia agentów przenośnych oraz omówienie ich przykładowych zastosowań. Skoncentrowano się na aplikacjach związanych z dostępem do rozproszonych baz danych, do których należą między innymi system MDBAS, zrealizowany za pomocą narzędzi Javy, oraz system AQuES, wykorzystujący wirtualnie współdzieloną pamięć do komunikacji między agentami.

Kolejne rozdziały opracowania obejmują charakterystykę agenta przenośnego, wyszczególnienie zalet tej koncepcji oraz propozycję zastosowania jej w tworzonym przez autorów eksperymentalnym systemie zarządzania rozproszoną bazą danych.

2. Definicja przenośnego agenta

Technologia przenośnych agentów jest oparta na idei migracji procesów (wątków) do różnych węzłów, w czasie ich wykonywania, razem z kodem programu oraz ich aktualnym stanem. Przenaszalny fragment zwany jest przenośnym agentem (ang. *mobile agent*). Użytkownik może stworzyć agenta na maszynie lokalnej i wysłać go do jednego lub kilku węzłów w celu wykonania zadań z wykorzystaniem lokalnego dostępu do źródeł na zdalnych komputerach.

Agent przenośny może być opisany jako obiekt programowy, który jest usytuowany w środowisku wykonawczym i posiada następujące własności [8, 15, 19]:

- ma własny mechanizm decyzyjny, może więc sprawować kontrolę nad swoimi działaniami,
- posiada zdolność rozpoznawania środowiska oraz zachodzących w nim zmian i może się do nich dostosować,
- ma cel do zrealizowania – wykonanie obliczeń na rzecz użytkownika lub udostępnienie mu usług,
- może działać bez ciągłego kontaktu z węzłem, z którego został wysłany (asynchronizm połączenia),
- może zadecydować o przerwaniu obliczeń – jeśli wymaga tego zadanie lub zasoby aktualnie odwiedzonego węzła okazują się niewystarczające bądź niedostępne – i wznowieniu ich na innej platformie po przeniesieniu agenta,
- może komunikować się i współpracować bezpośrednio z innymi agentami,
- może być zdolny do generacji agentów potomnych,
- potrafi wykorzystywać zasoby zdalnego węzła (CPU, pamięć, oprogramowanie, przepustowość wyjścia).

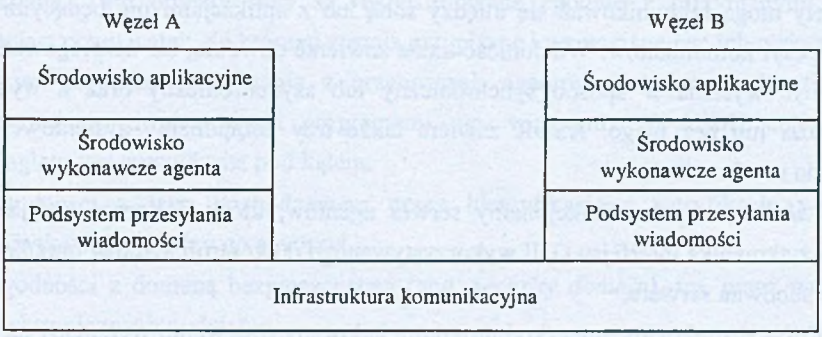
Wśród etapów działania agenta przenośnego można wymienić:

- otwarcie połączenia,

- transfer agenta (kod + dane + stan) z węzła macierzystego,
- wykonanie zadania,
- transfer powrotny (kod + dane + stan + wynik realizacji zadania),
- końcowe przetwarzanie wyniku w węźle macierzystym.

Omawiana technologia zachowuje i rozszerza cechy klasycznego modelu klient – serwer. Zasadniczo agent może być wykorzystywany albo jako klient, albo jako serwer, albo w obu tych rolach. Do modelu klient – serwer, w którym ma miejsce „zdalne wykonanie zdalnego kodu", przenośny agent dodaje możliwość przeniesienia i wykonania kodu w innym węźle, co daje „zdalne wykonanie lokalnego kodu". Co więcej, agent może przenosić się wiele razy, zarówno ze swoim kodem, jak i z danymi zebranymi w poprzednim odwiedzonym węźle.

Węzły, w których działają agenci, muszą dysponować złożonym środowiskiem wykonawczym (środowiskiem działań), umożliwiającym tworzenie, uruchamianie, przenoszenie i zatrzymywanie agentów (rys. 1). Środowisko to odpowiedzialne jest także za identyfikację nadchodzących agentów i udostępnianie im lokalnych zasobów. W jego obrębie komunikacja jest tańsza od komunikacji między komponentami rezydującymi na różnych węzłach [15]. Musi ono też zapewniać zdalną komunikację między agentami (zwykle przez wiadomości).



Rys. 1. Konceptualny model środowiska działań z wykorzystaniem agentów przenośnych [5]
Fig. 1. Conceptual model for mobile agent computing [5]

3. Środowiska agentów

Obecnie istnieje wiele systemów do tworzenia agentów, opartych na różnych architekturach. W dalszej części opracowania zostaną przedstawione dwa z nich.

3.1. Aglety – przenośne obiekty Javy

Jednym z systemów, który pozwala użytkownikowi na utworzenie i zarządzanie jego własnymi agentami, jest Aglets Software Development Kit (ASDK) firmy IBM [2].

Aglet jest przenośnym obiektem Javy, który może być wykonywany w dowolnym węźle sieci komputerowej, zapewniającym mu odpowiednie środowisko pracy. Każdy aglet jest jednoznacznie określony przez unikalny identyfikator, który jest niezmienny przez czas życia agletu. Aglet może robić wszystko to, co leży w zakresie możliwości programu w języku Java, włączając prezentację GUI oraz lokalny i zdalny dostęp do zasobów. Aglet jest dostępny przez obiekt *AgletProxy* spełniający rolę jego pełnomocnika. Jego zadaniem jest umożliwienie dostępu do aktualnego miejsca pobytu agletu, dzięki czemu rzeczywiste położenie agletu jest przeźroczyste dla innych obiektów systemu. AgletProxy służy również jako interfejs bezpośredniego dostępu do publicznych metod agletu, które nie powinny być osiągalne dla innych agletów.

Kontekst jest obszarem roboczym agenta, który rezyduje wewnątrz serwera agletów. Jest to obiekt stacjonarny, który zapewnia środki do utrzymywania i zarządzania pracującymi agletami w jednorodnym środowisku wykonawczym oraz do ochrony węzła przed agletami o nieznanym pochodzeniu.

Aglety mogą komunikować się między sobą lub z aplikacjami nie będącymi agletami przez przesył komunikatów. Wiadomość może zawierać dowolne, ale uszeregowane obiekty i może być wysłana w sposób synchroniczny lub asynchroniczny oraz z wymaganiem odpowiedzi lub bez niego. ASDK zawiera także trzy komponenty systemowe, które są gotowe do uruchomienia:

- *Tahiti* – w pełni funkcjonalny serwer agentów, który dostarcza przyjaznego dla użytkownika interfejsu GUI wykorzystywanego do kontroli agletów oraz konfiguracji zachowań serwera,
- *Fiji* – abstrakcyjny aplet (kod programu ładowany z serwera i wykonany na maszynie klienta, np. przez przeglądarkę wyposażoną w maszynę wirtualną Javy), który integruje usługi pojedynczego serwera agentów,
- *Aglets* – klasa pozwalająca na kontrolowanie agletów rezydujących na odległym serwerze.

System ASDK posiada także kilka rozszerzalnych i konfigurowalnych klas, dzięki czemu twórcy aplikacji mogą projektować i implementować ich własne systemy agentów.

Komunikacja sieciowa między systemami agentów odbywa się przez autonomiczny poziom komunikacyjny, co umożliwia prostą adaptację systemu agentów do różnych typów protokołów sieciowych (TCP/IP, CORBA IIOP i wiele innych sposobów udostępnianych w określonej sieci). W przypadku serwera Tahiti do komunikacji wykorzystywana jest prosta

wersja protokołu TCP/IP o nazwie ATP (Agent Transfer Protocol) – podobna do protokołu HTTP i pozwalająca na przeprowadzanie następujących operacji podstawowych:

- *Dispatch* – wysyłanie pracujących agletów do innego węzła, gdzie ich wykonanie jest kontynuowane,
- *Retract* – przywoływanie agletów ze zdalnego węzła,
- *Fetch* – pobranie danych z dowolnego zdalnego źródła,
- wysłanie wiadomości do lokalnych i zdalnych agentów oraz odbiór odpowiedzi.

Aglet może być utworzony na dwa sposoby:

- ustanawiany przez wywołanie metody *AgletContext.CreateAglet()* jako nowy aglet określonej klasy wewnątrz specyficznego kontekstu,
- tworzony jako kopia istniejącego agletu z wykorzystaniem metody *Aglet.Clone()*; taki aglet zachowuje stan oryginalnego agletu, ale różni się od niego identyfikatorem.

Raz utworzony aglet może być zarówno wysyłany (przez siebie samego, innego agenta lub aplikację) do zdalnego serwera, jak i odwoływany z niego, dezaktywowany do osobnej przestrzeni i uaktywniany później.

Rozsyłanie agletów powoduje zawieszenie ich wykonania, porządkuje ich stan wewnętrzny i przekształca kod do postaci bajtowej, w której jest on przenoszony do miejsca przeznaczenia. Tam obiekty Javy są rekonstruowane zgodnie z otrzymanymi danymi, tworzony jest nowy wątek, do którego zostają przypisane i wznawiane jest ich wykonanie.

Ważnym aspektem korzystania z przenośnych agentów jest ochrona węzłów sieci komputerowej przed nieznanymi programami, np. wirusami. W ASDK autentyczność każdego agletu jest sprawdzana pod kątem:

- zgodności z jego pochodzeniem przez identyfikację i weryfikację właściciela (użytkownika), który go stworzył,
- zgodności z domeną bezpieczeństwa (ang. *security domain*), np. przez weryfikację elektronicznych podpisów,
- poprawności i spójności kodu bajtowego.

Po przeprowadzonej weryfikacji administrator może kontrolować poczynania agletu oraz przyznawać mu lub odbierać prawa do różnych operacji dotyczących lokalnych zasobów lub znaczących akcji zmieniających kontekst agletu.

3.2. Agent Tcl

Agent Tcl [9, 10, 14] jest systemem do tworzenia agentów na podstawie skryptowego języka Tcl (Tool Command Language). Stanowi rozszerzenie języka Tcl dające możliwość tworzenia własnych agentów w formie skryptów wykonywanych przez zmodyfikowany interpreter języka Tcl i wykorzystujących zdefiniowane w rozszerzeniu funkcje realizujące

migrację agenta w sieci oraz komunikację z innymi agentami. Obsługą i monitorowaniem agentów przychodzących bądź opuszczających dany węzeł zajmuje się serwer agentów zainstalowany w tym węźle.

Cykl życia agenta przenośnego rozpoczyna się od jego rejestracji w wybranym węźle za pomocą funkcji *agent_begin*. Od tej pory agent przenośny posiada swój identyfikator składający się z następujących elementów:

- nazwa symboliczna węzła odpowiadająca jego adresowi IP,
- adres IP węzła,
- dowolnie wybrana symboliczna nazwa agenta,
- kod numeryczny przypisany agentowi przez serwera agentów w danym węźle.

Migracja agenta odbywa się w następujący sposób: agent kończący realizację swojego zadania w danym węźle wykonuje instrukcję *agent_jump* zdefiniowaną w pakiecie Agent Tcl, która odczytuje i zapamiętuje wewnętrzny stan agenta (kod, dane, zawartość stosu, wartości zmiennych) i przesyła ten zbiór informacji do węzła, w którym agent ma kontynuować działanie. Serwer agentów funkcjonujący w tym węźle odbiera i odczytuje wewnętrzny stan agenta i na tej podstawie wznawia jego pracę. Komunikacja między agentami może odbywać się za pośrednictwem serwerów agentów lub przez połączenie bezpośrednie i realizowana jest między innymi za pomocą funkcji *agent_send* oraz *agent_receive*.

Tworząc pakiet Agent Tcl zwrócono również szczególną uwagę na zagadnienie bezpieczeństwa, wprowadzając mechanizm uwierzytelnienia opartego na systemie szyfrowania PGP z kluczem publicznym [10].

4. Zalety wykorzystywania przenośnych agentów

Agenci przenośni nie rozszerzają zbioru usług dla systemów rozproszonych, jednakże mogą znacząco uprościć złożoność aplikacji i jej rozwój w rozproszonym środowisku. Do głównych zalet agentów należą:

Autonomia. Agent jest traktowany jako jednostka kodu z jej prywatnymi danymi. Podczas przenoszenia agent jest niezależny od jego macierzystego węzła. Nie ma potrzeby utrzymywania ciągłego połączenia między agentem i jakimkolwiek węzłem. Agent decyduje sam, co robić w węźle, jakie lokalne źródła wykorzystać i dokąd przenieść się później. Nie jest wymagane, by ktokolwiek dostarczał mu informacje i odbierał wyniki przeprowadzonych przez niego działań.

Asynchroniczna komunikacja. Ta cecha może być rozumiana dwojako. Po pierwsze, agenci mogą wykorzystywać asynchroniczne komunikaty do porozumiewania się ze sobą.

Natomiast z punktu widzenia użytkownika agenci mogą być pośrednikami w asynchronicznej komunikacji między użytkownikiem i zdalną usługą.

Zdalne wykonanie lokalnego kodu. Jest to jedna z najużyteczniejszych cech przenośnych agentów. W modelu klient – serwer klient zdalnie wykonuje zdalny kod. Klient nie ma zwykle możliwości modyfikowania zachowań serwera do realizacji celu klienta. Zatem, bardziej skomplikowane zadania wymagają dużego natężenia ruchu między klientem i serwerem. Kiedy wykorzystywany jest agent, logika aplikacji klienta może być umieszczona w agencie i wysłana do serwera. Wtedy agent współdziała z serwerem lokalnie i tylko wyniki tego współdziałania są wysyłane z powrotem do klienta. Przede wszystkim zdalne wykonanie lokalnego kodu może znacząco zmniejszyć komunikację między klientem i serwerem. Całkowity czas obliczeń może być obniżony przez możliwość wykorzystania innych zdalnych cech, np. jednostki CPU, pamięci, wielu procesów, wielowątkowości, szybszej podsieci itp. Co więcej, przez zdalne wykonanie agenci mogą być zdolni do realizacji specyficznych dla klienta obliczeń, które nie są możliwe do wykonania w źródle.

Mobilność. Kiedy realizacja zadania agenta dobiega końca w danym węźle, agent może przenieść się do innego węzła i tam kontynuować swoje działania. Prywatne dane agenta są przenoszone razem z jego kodem. Żądanie przeniesienia agenta może być wystosowane przez niego samego lub przez innego agenta bądź aplikację. Kiedy agent podejmuje decyzję dotyczącą zmiany węzła, może posłużyć się przygotowanym planem marszruty lub generować go dynamicznie wykorzystując informacje pochodzące z aktualnie odwiedzonego węzła lub od innych agentów. Plan marszruty opisuje, dokąd należy się udać i jakie zadanie wykonać. Dynamiczny sposób podejmowania decyzji jest używany przez aplikacje, które znają dla agentów tylko punkt startowy. Jeśli agent ma kilka możliwości przeniesienia się, powinien wybrać optymalną spośród nich.

W przypadku agentów potrzebujących wymiany pewnej ilości informacji, mobilność może być wykorzystywana w celu aranżacji spotkań na dowolnym komputerze dla lokalnego współdziałania.

Łatwość projektowania i implementacji aplikacji. Architektura niektórych aplikacji opiera się na idei zbliżonej do stosowanej dla przenośnych agentów – aplikacje takie wykorzystują niezależne, współpracujące ze sobą rozproszone programy. Ponieważ systemy agentów rozwiązują podstawowe problemy dotyczące współpracy agentów w środowisku wykonawczym, administrują nimi i zapewniają komunikację, wykorzystywanie agentów chroni twórców aplikacji przed rozwiązywaniem takich marginalnych problemów i pozwala im skoncentrować się na głównych zagadnieniach.

Prosty system administracyjny. Jednym z krytycznych problemów dotyczących administracji systemami rozproszonymi jest instalowanie oprogramowania na rozproszonych węzłach. Problem jest większy, jeżeli aplikacja powinna być dynamicznie rekonfigurowana

przez dodanie, zmianę funkcjonalności lub usunięcie węzłów. Taka dynamiczna instalacja może zostać zrealizowana na dwa sposoby. Pierwszy polega na wykorzystaniu agentów przenośnych – instalatorów, którzy migrują do odpowiedniego węzła instalując tam oprogramowanie. W drugim przypadku agent przenośny może sam w sobie stanowić niezbędne oprogramowanie.

5. Zalety przenośnych agentów wykorzystywane w aplikacjach rozproszonych baz danych

Po wskazaniu zalet przenośnych agentów, należy zastanowić się, jakie cechy agentów mogą pomóc w rozwiązywaniu problemów odnoszących się do baz danych i systemów pozyskiwania informacji.

Wstępne przetwarzanie danych. Dobrym przykładem zagadnienia tej klasy jest kompresja danych. Zastosowanie właściwej kompresji do transmitowanych danych może znacząco zwiększyć szybkość transmisji. Ponieważ istnieją różne metody kompresji dla źródeł danych różnego typu, agent może nie tylko dokonać samej kompresji danych przed ich transmisją, ale także przeanalizować kontekst danych i wybrać właściwą metodę kompresji.

Rozszerzenie funkcji zdalnego komputera. Obok umiejętności przetwarzania danych agent przenośny może być wyposażony w inne funkcje, które rozszerzą zestaw usług dostarczanych przez zdalny węzeł, kiedy agent dotrze do tego węzła.

Integracja rozproszonych źródeł. Budowanie systemu opartego na rozproszonych źródłach natrafia na problem dotyczący różnic pojawiających się w następujących zagadnieniach: typach danych, zakresie zdalnych usług, protokołach komunikacyjnych itp. Zwykle system taki specyfikuje swój wewnętrzny sposób dla jednolitego dostępu do zdalnych źródeł. Kiedy jego architektura jest prosta, np. klient obsługuje wiele baz danych, logika aplikacji może być zainstalowana tylko po stronie klienta. Administracja takim systemem jest łatwa nawet w przypadku rekonfiguracji lub zmian funkcji systemu, które wymuszają modyfikacje tylko na komputerze klienta (instalację nowych driverów lub programów). W przypadku kiedy architektura jest bardziej złożona i system wymaga instalacji komponentów w węźle źródła, niektóre zmiany funkcji systemu lub jego rekonfiguracja mogą być wysoce skomplikowane.

Agent przenośny może być wykorzystany jako komponent na zdalnym komputerze, którego zadaniem jest integracja źródeł. W takiej sytuacji zmiana funkcjonalności systemu oznacza zamianę aktualnego agenta przez innego.

Efektywne przetwarzanie dotyczące wielu baz danych. Przeszukiwanie rozproszonych baz danych, optymalizacja zapytania i jego efektywne wykonanie są bardzo ważne z punktu widzenia skrócenia czasu odpowiedzi na zapytanie. Mając zainstalowany system agentów w każdym węźle, gdzie umieszczona jest baza danych, można użyć przenośnych agentów w celu optymalizacji wykonania pojedynczego zapytania lub grupy zapytań. Rozważa się trzy scenariusze dla tego obszaru zastosowań agentów przenośnych.

Pierwszy scenariusz wykorzystuje pojedynczego agenta przenośnego do realizacji pojedynczego zapytania. Agent ten posiada listę węzłów, które powinien odwiedzić, zestaw operacji do wykonania w poszczególnych węzłach oraz informacje, jakie dane zebrać i przechować na potrzeby dalszego przetwarzania.

Drugi scenariusz rozprasza zadanie do wykonania między wielu współpracujących ze sobą agentów przenośnych. Agenci oferują możliwość łatwej implementacji efektywnych algorytmów realizacji rozproszonych zapytań także wtedy, gdy oparta jest ona na dynamicznej optymalizacji zapytań.

W trzecim scenariuszu cała grupa zapytań może być zakodowana jako agent przenośny. Wykorzystując swoją mobilność, agent może różne fragmenty zadania wykonywać na różnych komputerach, wybierając te węzły, które w danym momencie gwarantują najlepsze rezultaty przetwarzania.

6. Aplikacje przenośnych agentów dla rozproszonych baz danych

Prowadzone rozważania zostaną zilustrowane przykładami kilku projektów, które wykorzystują przenośnych agentów do przetwarzania w rozproszonych bazach danych.

6.1. Ruchomi agenci dla dostępu do rozproszonych baz danych w WWW

Projekt dotyczy problemu dostępu do rozproszonych baz danych z przeglądarki sieci Web, wykorzystującej technologię Javy. Zaproponowany został nowy model *DBMS-Aglet Framework* [13], oparty na przenośnych agentach Javy. Po zaimplementowaniu model ten został porównany pod względem efektywności z podejściem wykorzystującym obiekty typu aplet oraz JDBC, w którym aplet bazy danych wraz z odpowiednimi driverami JDBC na żądanie pobierany jest z serwera Web do przeglądarki klienta. Nawet, jeżeli niektóre drivery są optymalizowane pod kątem wykorzystania ich do takiego celu, operacja ta pociąga za sobą znaczące koszty.

Natomiast DBMS-Aglet Framework opiera się na standardowej infrastrukturze obejmującej przeglądarki obsługujące język Java, serwer Web i serwer SQL, poszerzonej o serwer agletów włączony w przeglądarkę, router agletów instalowany na serwerze

sieciowym oraz serwer agletów Tahiti umieszczony w węźle, na którym znajduje się serwer SQL.

6.2. Wykorzystanie systemu Agent Tcl

Wśród projektów zrealizowanych z użyciem tego systemu jest między innymi agent sprawdzający dostępność zdalnego węzła w sieci, przewidujący czas transferu w sieci oraz czas przetwarzania w węźle. Umiejętność uzyskania tych informacji pozwala agentowi na skonstruowanie marszruty unikającej węzłów nieosiągalnych lub przeciążonych. Takie rozwiązanie zostało wykorzystane do budowy systemu, w którym agenci przeszukują rozproszone w sieci zbiory dokumentów tekstowych [14].

System Agent Tcl został zastosowany również do badania różnych strategii realizacji zapytania w rozproszonej bazie danych [12]. Wynik końcowy takiego zapytania otrzymywany jest po przeprowadzeniu ciągu operacji scalających na zbiorach rekordów (wynikach cząstkowych) otrzymanych z lokalnych baz danych. Ustalenie kolejności wykonania tych operacji stanowi przedmiot optymalizacji, której rezultatem jest plan realizacji zapytania. Wykonanie zapytania według wyznaczonego planu może przebiegać zgodnie ze strategią statyczną lub dynamiczną. Strategia *statyczna* zakłada realizację operacji scalających według planu opracowanego *a priori*, przede wszystkim na podstawie szacunkowych informacji o profilu wyników cząstkowych. Plan nie podlega żadnym modyfikacjom, mimo że stan rzeczywisty może znacząco odbiegać od oszacowań. Strategia *dynamiczna* eliminuje tę rozbieżność dopuszczając wielokrotną modyfikację planu podczas jego realizacji. Istnieje wiele odmian strategii dynamicznej różniących się następującymi elementami:

- techniką udostępniania wyników monitorowania przebiegu procesu realizacji zapytania, prowadzonego w celu zbierania informacji o generowanych wynikach cząstkowych; technika *scentralizowana* zakłada, że wszystkie informacje spływają do węzła centralnego; *rozproszona* – polega na dostarczeniu kompletu informacji do każdego węzła; *niezależna* – ogranicza zakres informacji pozostających do dyspozycji danego węzła tylko do tych dostępnych lokalnie – bez wymiany informacji z innymi węzłami,
- okolicznościami podejmowania decyzji o modyfikacji realizowanego planu; poszukiwanie lepszej strategii wykonania odbywa się po każdym wygenerowaniu pojedynczego wyniku cząstkowego, wykorzystując informacje dotyczące jego rzeczywistego profilu lub ma miejsce jedynie wobec przekroczenia wartości progowej kosztu wygenerowania danego wyniku cząstkowego,
- sposobem podejmowania decyzji o korekcie planu; w podejściu *scentralizowanym* decyzja należy do węzła centralnego; w *rozproszonym* – następuje uzgodnienie

decyzji przez wszystkie węzły; sposób *niezależny* pozostawia decyzję w gestii każdego węzła nakładając na niego obowiązek powiadomienia innych węzłów o podejmowanym działaniu korygującym.

Wykorzystując różne warianty poszczególnych elementów strategii dynamicznej zaproponowano w pracy [12] trzy algorytmy – scentralizowany, rozproszony i migracyjny – i porównano je z algorytmem opartym na statycznej metodzie realizacji zapytań. W algorytmie *scentralizowanym* wybrany agent przekazuje wyniki monitorowania i podejmuje decyzję o korekcie planu. Wspólne działania wszystkich agentów w zakresie monitorowania i decydowania o podjęciu działań korygujących są założeniem algorytmu *rozproszonego*. Algorytm *migracyjny* również opiera się na współpracy agentów, uwzględniając ponadto możliwość migracji agenta z węzła, w którym realizowanych jest kilka operacji scalających po to, by zleconą mu operację wykonać w mniej obciążonym węźle.

Zastosowany w procesie optymalizacji model kosztów operuje wartościami w dziedzinie czasu i uwzględnia następujące składniki:

- koszt przetwarzania lokalnego w węźle,
- koszt migracji agenta między węzłami,
- koszt transferu wyników cząstkowych między węzłami,
- koszt przesyłu komunikatów sterujących między agentami operującymi na różnych węzłach.

6.3. Wykorzystanie technologii przenośnych agentów w budowie systemu wielobazowego

System MDBAS (Mobile Database Agent System) [18, 19, 20, 21] integruje autonomiczne bazy danych w sieci, pozwala użytkownikom na utworzenie schematu globalnego, zarządza efektywną a zarazem przeźroczystą dla użytkownika, rozproszoną realizacją jego zapytań oraz transakcji rozproszonych, wypełniając je zgodnie z protokołem 2PC [3]. W schemacie globalnym znajdują się dwa rodzaje tabel: fizyczne tabele lokalne oraz tabele logiczne, definiowane na potrzeby użytkownika przez instrukcje SELECT i UNION, realizowane w lokalnych bazach danych. Operacje modyfikacji danych mogą dotyczyć wyłącznie tych tabel logicznych, które reprezentują pojedyncze tabele fizyczne. Chociaż MDBAS integruje lokalne bazy danych w schemat globalny, lokalne bazy danych zachowują pełną niezależność i praca lokalnych aplikacji z lokalnymi danymi nie jest w żaden sposób ograniczona. Istotną cechą systemu jest zdolność do przeźroczystego zarządzania zdalnym wykonaniem operacji na zbiorze autonomicznych baz danych rozmieszczonych w wielu węzłach sieci. Operacje te wykonują agenci przenośni, uzyskujący dostęp do baz danych przez JDBC API. W mechanizmie zdalnej realizacji zapytań wykorzystuje się migrację

agentów, redukującą wielkość transmitowanych danych, oraz buforowanie danych, pochodzących ze zdalnych węzłów, w innych, najbardziej opłacalnych węzłach. Dane w buforze są dostępne na zasadach analogicznych do bazy danych.

Rozproszona realizacja zapytań bazuje na algorytmie optymalizacji dynamicznej. Model kosztów obejmuje przesyłanie wiadomości i migrację agentów, zakładając stałą wielkość agenta w trakcie działania i jednakowy koszt przetwarzania lokalnego w każdym węźle.

W architekturze systemu MDBAS wyróżnia się moduł centralny (*Central*), do którego docierają wszystkie żądania użytkowników, oraz moduły robocze (*Workplace*), odpowiadające za realizację operacji składowych zapytania. Przyjęto założenie, że system działa w sieci w pełni połączonej.

Agenci komunikują się ze sobą przesyłając wiadomości. Środowisko wykonawcze w danym węźle stanowi serwer agentów. W tej roli użyto serwera Tahiti, wchodzącego w skład pakietu ASDK. W systemie MDBAS współpracują ze sobą agenci różnych typów, realizujący następujące funkcje:

- *Master* – kontroluje przepływ informacji w całym systemie,
- *Manager* – koordynuje transakcję rozproszoną, zarządza sesją użytkownika,
- *Porter* – kontroluje dostęp do bazy danych, odpowiada za udostępnienie agentowi *Workerowi* połączenia z bazą danych, samodzielnie wykonuje proste modyfikacje na pojedynczej tabeli fizycznej w lokalnej bazie danych (INSERT, UPDATE, DELETE), jest uczestnikiem w głosowaniu podczas realizacji protokołu wypełnienia transakcji rozproszonej,
- *Worker* – wykonuje operacje scalające wyniki cząstkowe,
- *Runner* – wykonuje grupę zapytań, dysponuje informacją o aktualnym stanie sieci,
- *Buffer* – tworzony przez agenta *Runnera*, współpracuje z nim ładując zdalne dane, wymagane przez grupę zapytań, do bufora w najodpowiedniejszym węźle,
- *Permeability agent* – kontroluje stan sieci w celu określenia efektywnej marszruty dla agenta *Runnera*.

6.4. AQuES (Agent-based Query Evaluation System) – system wykorzystujący agentów w realizacji zapytań w rozproszonej bazie danych z zastosowaniem pionowej równoległości (pipelining) i intraparallelizmu

Wielofunkcyjni agenci systemu AQuES [16] współpracują ze sobą, zmierzając do równoważenia obciążenia, a komunikacja między nimi opiera się na wirtualnie współdzielonej pamięci i odbywa się z wykorzystaniem przestrzeni krotek zarządzanej przez system Linda [4].

Do części składowych agenta należą:

- *interfejs użytkownika*, *parser* – umożliwiają wprowadzenie i analizę polecenia użytkownika,
- *koordynator* – interpretuje nadchodzące wiadomości i zarządza wewnętrzną komunikacją między komponentami agenta,
- *optymalizator statyczny* – opracowuje plan realizacji zapytania złożony z planów realizacji poszczególnych zadań,
- *zarządca zadań* – odpowiada za realizację zadań, decyduje o przeprowadzeniu optymalizacji dynamicznej po stwierdzeniu odchyleń wartości rzeczywistych parametrów wejściowych w porównaniu z ich wartościami przewidywanymi, dostarcza informacji zwrotnych modułowi uczącemu się,
- *optymalizator dynamiczny* – koncentruje się na adaptacji systemu do zmian w czasie rzeczywistym – podejmuje decyzję o tym, czy zastosować inny algorytm realizacji zadania, pobrać od innego agenta dodatkowe zadanie do wykonania, oddać zadanie innemu agentowi, rozdzielić zadanie między siebie i innego agenta stosując intraparallelizm, opracować od podstaw nowy plan realizacji pojedynczego zadania,
- *moduł uczący się* – na podstawie informacji zwrotnej po zakończeniu zadania aktualizuje metadane – bazę wiedzy o środowisku pracy,
- *moduł komunikacyjny* – implementuje asynchroniczny protokół przekazywania komunikatów między agentami.

W systemie AQuES nie istnieje żaden centralny element, który koordynowałby rozpoczętą realizację danego zapytania.

6.5. Zasilanie hurtowni danych zewnętrznymi informacjami filtrowanymi z zaufanych źródeł informacji

Na potrzeby obsługi hurtowni danych zbudowano eksperymentalny system eDW (enhanced Data Warehouse [1]), w którym stosuje się agentów (programy napisane w języku Java) do poszukiwania informacji w heterogenicznych źródłach znajdujących się w Internecie. Aplikacje systemu eDW współpracują z hurtownią danych stworzoną za pomocą narzędzi pakietu SAS. Aplikacja *profilująca* tworzy dla każdego obiektu hurtowni danych zdefiniowanego w metadanych (np. tabel, raportów) profil potrzeb informacyjnych, wykorzystywany przez agenta do filtrowania informacji ze wskazanych mu źródeł. Każdy zaakceptowany dokument jest również badany pod kątem przedziału czasu, jakiego dotyczy jego treść. Aplikacja *raportująca* tworzy raporty hurtowni danych rozszerzone o listę rankingową dokumentów dopasowanych zarówno czasowo, jak i tematycznie do danego raportu.

7. Wykorzystanie przenośnych agentów do realizacji wybranych funkcji eksperymentalnego systemu zarządzania rozproszoną bazą danych

Analiza cech technologii przenośnych agentów oraz obszaru jej zastosowań, przeprowadzona w poprzednich rozdziałach niniejszego opracowania, wykazała, że może ona być z powodzeniem wykorzystywana do rozwiązywania problemów odnoszących się do rozproszonych baz danych. Skłoniło to autorów do rozważenia możliwości zastosowania przenośnych agentów w celu rozszerzenia funkcjonalności tworzonego przez nich eksperymentalnego systemu zarządzania rozproszoną bazą danych. System ten, opisany w [6, 7], w obecnym kształcie realizuje podstawowe zadania dotyczące rozproszonej bazy danych. Obejmują one:

- przyjęcie i analizę składniową poleceń języka SQL,
- ich dekompozycję zgodnie ze schematem rozproszonej bazy danych zawartym w globalnym słowniku danych; schemat ten dopuszcza istnienie takich typów rozproszeń, jak: fragmentacja pionowa, fragmentacja pozioma oraz replikacja,
- obsługę blokad logicznych [7],
- realizację dwufazowego protokołu wypełnienia transakcji 2PC,
- prezentację wyników.

Komunikacja między modułami systemu realizującymi powyższe funkcje odbywa się przez wirtualnie współdzieloną pamięć opartą na modelu Lindy [4, 17].

Obecnie prowadzone są badania dotyczące metod aktualizacji kopii danych oraz realizacji operacji wyszukiwania w rozproszonej bazie danych z wykorzystaniem dynamicznej optymalizacji zapytań.

7.1. Zastosowanie agentów do realizacji zapytań

Wśród zadań, do których realizacji możliwe jest użycie agentów, należy wymienić:

- nadzorowanie i koordynowanie realizacji planu wykonania zapytania, wydanie decyzji o jego korekcie w przypadku strategii dynamicznych,
- wykonywanie elementów składowych zapytania: podzapytań i operacji scalających, informowanie o ich rezultatach,
- nadzorowanie i raportowanie stanu sieci i węzłów.

7.2. Zastosowanie agentów do zadań aktualizacji kopii danych

Podstawowy problem takich aktualizacji polega na zapewnieniu spójności danych znajdujących się w różnych węzłach z równoczesnym zapewnieniem powodzenia wykonania transakcji rozproszonej. Transakcja taka realizowana jest w następujących etapach:

1. Pobranie od użytkownika operacji należących do transakcji rozproszonej.
2. Zbadanie stanu blokad logicznych.
3. Rozesłanie transakcji i wykonanie jej w węzłach zawierających kopie.
4. Przeprowadzenie protokołu 2PC w celu ustalenia sposobu zakończenia transakcji.
5. Prezentacja wyników realizacji transakcji.
6. Uaktualnienie stanu węzłów, które z różnych przyczyn nie mogły brać udziału w transakcji. Zastosowanie jednej z asynchronicznych strategii aktualizacji kopii [11] stanowi jedną z przyczyn takiej sytuacji.

Analiza powyższych etapów oraz doświadczenie zdobyte podczas budowy eksperymentalnego systemu zarządzania rozproszoną bazą danych pozwoliły wyodrębnić następujący zestaw przenośnych agentów wraz z ich zadaniami:

- *Koordynator* – odpowiedzialny za zarządzanie transakcją rozproszoną,
- *Zarządca_blokad* – odpowiedzialny za utrzymywanie informacji o stanie blokad logicznych dla poszczególnych węzłów,
- *Wykonawca* – realizujący transakcje na lokalnej bazie danych,
- *Uaktualniający* – zmieniający stan węzłów, które nie brały udziału w transakcji.

Ponieważ zestaw realizowanych zadań może ulegać zmianie w zależności od przyjętej strategii aktualizacji kopii danych, z opracowanych strategii zajmujących się rozwiązaniem tego problemu wybrano cztery, które zostały przeanalizowane pod kątem możliwości wykorzystania przenośnych agentów do realizacji przewidzianych w nich zadań.

Strategia synchroniczna. Strategia ta zakłada, że wszystkie kopie uaktualniane są w ramach jednej transakcji rozproszonej, co wymaga zastosowania protokołu 2PC, którego uczestnikami są wszystkie węzły zawierające kopie danych.

W momencie pojawienia się w węźle nowej transakcji powoływany jest do życia Koordynator, który po przeprowadzeniu analizy składniowej operacji należących do transakcji wzywa Zarządcę_blokad. Jeżeli stan blokad wskazuje na to, że w systemie realizowana jest inna konfliktowa transakcja, wtedy nowa transakcja nie jest dopuszczana do realizacji. W przeciwnym przypadku tworzeni są Wykonawcy, którzy rozsyłani są do odpowiednich węzłów, gdzie realizują swoje zadania – operacje należące do transakcji rozproszonej. Koordynator przeprowadza protokół 2PC, a po jego zakończeniu wzywa Zarządcę_blokad i zmienia stan blokad logicznych.

Ponieważ komunikacja może odbywać się przez przestrzeń krotek, to nie jest istotne, gdzie w danym momencie znajduje się agent zarządzający blokadami. Żądanie przybycia

tego agenta realizowane jest przez umieszczenie w przestrzeni krotek odpowiedniej informacji – identyfikatora węzła oraz czasu wstawienia krotki.

Strategia oparta na kworum. Do zadań Koordynatora działającego zgodnie z tą strategią, poza zarządzaniem transakcją rozproszoną, należy również zadbanie o uzyskanie odpowiedniego kworum do jej realizacji. Ponieważ sposób budowy takiego kworum zależny jest od przyjętej metody, Koordynator posługując się ustalonym kryterium, odwiedza węzły w sieci komputerowej, każdorazowo dynamicznie decydując o miejscu przeniesienia. Do czynników wpływających na tę decyzję należą:

- zdolność węzła do przeprowadzenia transakcji – brak równolegle biegnącej transakcji,
- aktualny stan danych – w węźle tym wykonano wszystkie transakcje zrealizowane do tej chwili w systemie.

W węzłach, które będą brały udział w transakcji, tworzeni są agenci Wykonawcy. Razem z Koordynatorem w sieci migruje agent Zarządca_blokad, niosący ze sobą stan blokad, co pozwala na ich bieżącą aktualizację. Po transakcji zakończonej protokołem 2PC blokady te ulegają zwolnieniu i uruchamiany jest zbiór agentów Uaktualniających. W przypadku niedostępności któregoś z węzłów, jego agent Uaktualniający czeka w węźle macierzystym na zmianę tej sytuacji. Jeżeli niedostępność węzła przedłuża się na czas kilku transakcji, agent Uaktualniający, przeznaczony do działania w tym węźle, gromadzi zestaw takich transakcji, by w odpowiednim czasie przenieść się tam i je zrealizować. W ten sposób każdy z węzłów może mieć swoją własną kolejkę transakcji oczekujących na wykonanie.

Strategia Master/Slave. W strategii tej mamy do czynienia z jednym wyróżnionym węzłem (Master), który uprawniony jest do przyjmowania zadań uaktualniania danych. W związku z tym wykonanie transakcji za pomocą agenta przenośnego sprowadza się, przy braku konfliktu blokad, do powołania do życia w węźle Master agenta Wykonawcy, który realizuje tam swoje zadanie. Następnie, podobnie jak w strategii opartej na kworum, tworzona jest grupa agentów z zadaniem modyfikacji pozostałych kopii danych lub zmienia się stan istniejących agentów Uaktualniających, rozszerzając kolejkę transakcji oczekujących na wykonanie.

Strategia Independent. Cechą charakterystyczną tej strategii jest fakt, że wykonanie nowej transakcji ma miejsce w jednym, dowolnym węźle i nie jest uzależnione od ogólnej sytuacji panującej w systemie. Zatem do realizacji nowej transakcji wystarczy powołanie jednego agenta Wykonawcy, który zrealizuje zadanie w odpowiednim węźle. Taki sposób postępowania może zrodzić niebezpieczeństwo równoległego wykonania kilku różnych transakcji na różnych kopiach danych. Wymaga to wyposażenia powoływanych agentów Uaktualniających w wiedzę dotyczącą sposobu rozwiązywania ewentualnych napotkanych konfliktów. Wykorzystanie technologii przenośnych agentów do usuwania takich problemów daje łatwość implementacji algorytmów rozwiązywania konfliktów i ich modyfikacji

w sytuacji, gdy w trakcie istnienia systemu pojawią się nowe okoliczności decydujące o zmianie sposobu postępowania w momencie pojawienia się niespójności w kopiach danych.

8. Wnioski

Rozważając zalety przenośnych agentów i oceniając je w stosunku do alternatywnych metod uzyskiwania tych samych funkcji, należy zaznaczyć, że agenci nie rozszerzają zbioru usług dla systemów rozproszonych. Do zalet tego rozwiązania natomiast można zaliczyć:

- tworzenie przenośnych agentów zdolnych do rozprzestrzeniania się w różnych kierunkach, co zwiększa dostępność dużej liczby usług i aplikacji sieciowych,
- wykorzystanie tej samej technologii do rozwiązania zadań wielu klas.

Przeprowadzona w punkcie 7 analiza użycia przenośnych agentów w eksperymentalnym systemie zarządzania rozproszoną bazą danych ujawniła wiele możliwości zastosowania tej technologii w badanym obszarze. Ponieważ jednak agenci przenośni mają stanowić jedynie rozszerzenie funkcjonującego systemu zarządzania rozproszoną bazą danych, nie wszystkie rodzaje agentów, wyszczególnionych w przeprowadzonej analizie, przewidziane są do zaimplementowania w systemie. Zostanie on rozszerzony o agentów przeznaczonych do realizacji następujących zadań:

- zarządzanie blokadami logicznymi,
- dostarczenie zestawu operacji do wykonania modułowi odpowiedzialnemu za ich realizację na lokalnej bazie danych,
- uaktualnienie kopii danych w węzłach nie uczestniczących w protokole 2PC,
- nadzorowanie i raportowanie stanu sieci i węzłów,
- wykonywanie operacji scalających, informowanie o ich rezultatach.

LITERATURA

1. Abramowicz W., Kalczyński P.J., Węcel K.: Aplikacje profilująca i raportująca w enhanced Data Warehouse. Forum Naukowe „Budowanie pomostu między nauką a biznesem", Kraków, 25.10.2001.
2. ASDK Homepage: http://www.trl.ibm.co.jp.
3. Bernstein P. et al.: Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987.
4. Carriero N., Gelernter D., Mattson T., Sherman A.: The Linda Alternative to Message-Passing Systems. Parallel Computing, Vol. 20, 1994, pp. 633-655.

5. Chess D., Harrison C., Kershenbaum A.: Mobile Agents: Are They a Good Idea? IBM Research Report, IBM Research Division, T.J. Watson Research Center, New York, 1995.
6. Chłopek M., Harężlak K., Josiński H.: Implementacja podstawowych mechanizmów zarządzania rozproszoną bazą danych w eksperymentalnym systemie wykorzystującym model wirtualnie współdzielonej pamięci. ZN Pol. Śl. s. Informatyka, z. 32, Gliwice 1997.
7. Chłopek M., Harężlak K., Josiński H.: Sterowanie współbieżnym dostępem do danych podczas realizacji transakcji rozproszonych – rozwój eksperymentalnego systemu rozproszonej bazy danych. ZN Pol. Śl. s. Informatyka, z. 34, Gliwice 1998.
8. Ferber J.: Multi-Agent Systems. An Introduction to Distributed Artificial Intelligence. Addison-Wesley, 1999.
9. Gray R., Cybenko G., Kotz D., Rus D.: Agent Tcl. In: Itinerant Agents: Explanations and Examples with CDROM. Cockayne W., Zypa M. (ed.), Manning Publishing, 1997.
10. Gray R.: Agent Tcl. Software, nr 5, 1997.
11. Harężlak K.: Realizacja transakcji na kopiach danych w środowisku wirtualnie współdzielonej pamięci. Studia Informatica vol. 22, no 3 (45), Gliwice 2001.
12. Jones R., Brown J.: Distributed Query Processing via Mobile Agents. Via Internet: http://www.cs.umd.edu/~keleher/818s97/ryan/paper.html, 1997.
13. Papastavrou S., Samaras G., Pitoura E.: Mobile Agents for WWW Distributed Database Access. 15th International Conference on Data Engineering (ICDEE99), Sydney, 1999.
14. Rus D., Gray R., Kotz D.: Autonomous and Adaptive Agents that Gather Information. AAAI'96 International Workshop on Intelligent Adaptive Agents, 1996.
15. Sahuguet A.: About agents and databases. CIS-650, 1997. Via Internet: http://www.cis.upenn.edu/~sahuguet/Agents/Agents_DB.pdf.
16. Stillger M., Obermaier J.K., Freytag J.Ch.: AQuES: An Agent-based Query Evaluation System. Proceedings of the 2nd IFCIS International Conference on Cooperative Information Systems (CoopIS '97), Kiawah Island, South Carolina, USA, 1997.
17. Virtual Shared Memory and the Paradise System for Distributed Computing – A technical White Paper. SCIENTIFIC Computing Associates, Inc. Via Internet: http://www.lindaspaces.com/downloads/vsm.pdf, 1999.
18. Vlach R., Lána J., Marek J., Navara D.: MDBAS – A Prototype of a Multidatabase Management System Based on Mobile Agents. Proceedings of SOFSEM'00, 27th Conference on Current Trends in Theory and Practice of Informatics. Lecture Notes in Computer Science Vol. 1963. Springer-Verlag, 2000, pp. 440-449.
19. Vlach R., Marek J., Pokorný J.: Mobile Agents and Databases. Via Internet – MDBAS download page: http://kocour.ms.mff.cuni.cz/mdba/, 1999.
20. Vlach R.: Efficient Execution Strategies for Mobile Procedures Querying Distributed Databases. In: Proceedings of DOA'00, IEEE, 2000.

21. Vlach R.: Mobile Database Procedures in MDBAS. Via Internet – MDBAS download page: http://kocour.ms.mff.cuni.cz/mdba/, 2001.

Recenzent: Prof. dr hab. Mieczysław Muraszkiewicz

Wpłynęło do Redakcji 26 marca 2002 r.

Abstract

Mobile agents are programs that may be dispatched together with their data from a client computer to a remote server for execution. The mobile agent technology offers new approaches in distributed computing especially for efficient exploitation of distributed data sources. This paper presents a definition of the mobile agent, its features, mechanisms, and application areas. The conceptual model for mobile agent computing (Fig. 1) is also included. Besides, the paper contains an overview of environments for creation and management of mobile agents. The authors discuss the processing of queries and transactions in distributed databases using the mobile agent technology. The possibilities of its application in the experimental distributed database management system are also considered in the paper.