AUTOMATYKA • 2008 • Tom 12 • Zeszyt 2

Mirosław Kasper*

Replikacja w systemach baz danych zaprojektowanych w architekturze wielowarstwowej

1. Wprowadzenie

Większe firmy oraz duże korporacje używają różnorodnych aplikacji do zarządzania danymi, do których zaliczyć można systemy CRM służące do zarządzania relacjami z klientami (*Customer Relationship Management Systems*), systemy SCM umożliwiające zarządzanie dostawami towarów (*Supply Chain Management Systems*), systemy wspomagające podejmowanie decyzji i raportowanie (*Business Intelligence Systems*), systemy do zarządzania zasobami ludzkimi (*Human Resource Management Systems*) i wiele innych. Komunikacja pomiędzy takimi systemami jest bardzo często utrudniona ponieważ wykorzystują one zwykle różne oprogramowanie oraz inne systemy do przechowywania danych, a także mogą pracować na różnych platformach sprzętowych. Prowadzi to do zmniejszenia wydajności takich systemów ze względu na konieczność przechowywania tych samych danych w wielu różnych lokalizacjach.

Rozwiązaniem tego problemu mogą być systemy klasy EAI (*Enterprise Application Integration*) integrujące aplikacje i dane w jeden spójny zestaw procesów biznesowych. Systemy takie pozwalają współdzielić dane przez wiele systemów informatycznych oraz umożliwiają automatyzację procesów biznesowych w ramach firmy. Największym wyzwaniem dla systemów EAI jest utworzenie wydajnych połączeń pomiędzy różnymi systemami operacyjnymi, różnymi bazami danych, a także pomiędzy aplikacjami pisanymi w różnych językach programowania.

W celu synchronizacji danych pomiędzy różnymi aplikacjami czy lokalizacjami w zastosowaniach praktycznych wykorzystuje się sprzętową replikację danych (*hard-ware replication*) lub replikację na poziomie oprogramowania (*software replication*). Systemy wykorzystujące replikację na poziomie sprzętu, umożliwiają przenoszenie pomiędzy lokalizacjami poszczególnych operacji na poziomie fizycznych nośników danych (np. fizycznych zapisów bloków danych na dysk). Jednak replikacja taka wykorzystywana jest zwykle w przypadku niedużych odległości pomiędzy kopiami danych i najczęściej

* Katedra Automatyki, Akademia Górniczo-Hutnicza w Krakowie

służy tylko do zapewnienia dostępu do danych na wypadek awarii głównego centrum. Podczas normalnej pracy systemu dane w lokalizacji zapasowej nie są dostępne dla jego użytkowników.

Sytuacja wygląda inaczej w przypadku systemów transakcyjnych OLTP (*On-line Transaction Processing Systems*). W ostatnim czasie w większości systemów informatycznych współpracujących z transakcyjnymi bazami danych, realizowanych w architekturze klastrowej, czy też opartych na współpracy z rozproszonymi systemami webowymi, bardzo ważnym mechanizmem do zarządzania danymi stała się replikacja danych. Takie systemy replikacji danych tworzone są obecnie najczęściej w architekturze wielowarstwowej, wykorzystując zastosowanie warstwy pośredniej nazywanej warstwą middleware. Zarządzanie procesem replikacji odbywa się w nich poprzez specjalne oprogramowanie pracujące w warstwie middleware. Replikacja danych wykorzystująca warstwę pośredniczącą middleware (*middleware-based database replication*) może być wykorzystywana w wielu aplikacjach przetwarzających duże ilości transakcji, w celu zapewnienia skalowalności systemu (*scalability*) oraz zwiększenia jego odporności na awarie pojedynczych węzłów lub całych lokalizacji (*fault-tolerance*), zapewniając jednocześnie dostęp do identycznych danych w wielu odległych lokalizacjach.

Bardzo wygórowane wymagania związane z krótkim czasem dostępu do danych, czy też z koniecznością ciągłego do nich dostępu, określane w umowach gwarantujących odpowiednią jakość świadczonych usług, np. w SLA (Service Level Agreement), powodują że systemy muszą przetwarzać transakcje aplikacyjne przez 7 dni w tygodniu przez cały rok, praktycznie bez przerwy. Dzięki zastosowaniu architektury wielowarstwowej możliwe jest m.in. osiągnięcie wymaganego poziomu dostępności danych. Jeśli któryś z replikowanych węzłów (replik) ulegnie awarii, pozostałe dostępne węzły są w stanie przejąć zadania uszkodzonego węzła. Jeśli system działa w architekturze wielowarstwowej, z replikacją danych realizowaną w warstwie middleware, rozbudowa systemu (osiągana przez dodawanie kolejnych replik) i zwiększanie odporności na wypadek awarii (poprzez umieszczanie węzłów systemu w odległych lokalizacjach) stają się nie tylko możliwe, ale są bardzo często operacją niezbyt skomplikowaną. W przypadku zwiększenia wymagań odnośnie do wydajności systemu skonfigurowanego w architekturze wielowarstwowej, nowe węzły dodawane są do niego w taki sposób, aby pomimo zwiększonego obciążenia w systemie spełniać nowe wymagania, zapewniając skrócenie czasu dostępu do żądanych danych na odpowiednim poziomie, i w związku z tym pozwalając na zmniejszenie czasu koniecznego do wykonania całej usługi korzystającej z replikowanych danych.

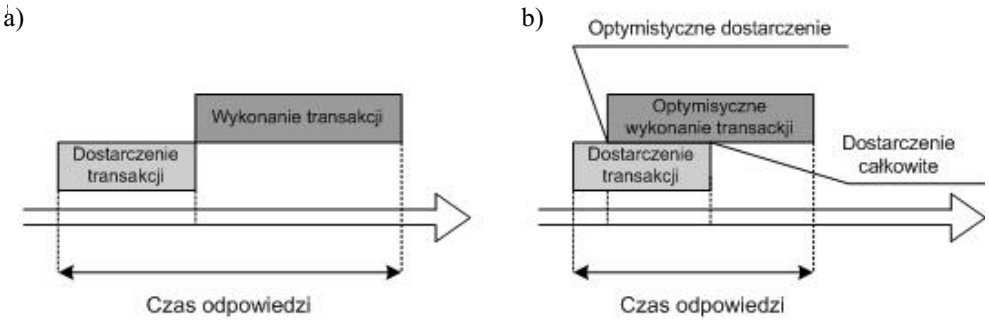
W zastosowaniach praktycznych zarządzanie procesem replikacji polega na utrzymaniu spójnych kopii danych we wszystkich węzłach systemu, co jest bardzo złożonym procesem i wymaga dużej wiedzy, zarówno pod kątem architektury systemu, jak też specyfiki konkretnej aplikacji. Typowymi problemami występującymi podczas replikacji danych w systemach baz danych są zarówno trudności związane ze skalowalnością systemu, długie czasy wykonywania operacji, częste występowanie zakleszczeń (*deadlocks*), oraz opóźnienia wynikające z prędkości przesyłania komunikatów pomiędzy węzłami w sieci, jak i sama

wielkość tych komunikatów [1]. W artykule przedstawione zostały zagadnienia dotyczące replikacji danych w systemach transakcyjnych pracujących przy użyciu architektury middleware.

2. Rozwój technik replikacji

Replikacja danych w początkowej fazie rozwoju realizowana była poprzez modyfikację kodu motoru systemu zarządzania bazą danych (SZBD). Modyfikacje takie wykonywane były np. w module obsługi logów transakcyjnych zawierających wszystkie modyfikacje w systemie, czy też poprzez wykorzystanie modułów umożliwiających komunikację grupową (*group communication*). Przykładem takiej realizacji replikacji jest implementacja systemu Postgres-R [2], który wykazuje dużą wydajność w związku z niewielkim narzutem związanym z procesem replikacji w takim systemie. W związku z koniecznością modyfikacji kodu źródłowego centralnej części SZBD, rozwiązania wymagające zmian w kodzie są bardzo trudne do przeniesienia pomiędzy systemami. Trudności takie pojawiają się nie tylko w przypadku systemów różnych pod względem stosowanego systemu zarządzania bazą danych, czy systemu operacyjnego, ale bardzo często występują również w systemach od tych samych dostawców, nawet tych, które tylko nieznacznie się różnią, np. kolejnych wersji tych samych systemów.

Zaproponowane później w [3] rozwiązanie opiera się na propagacji wszystkich wykonywanych lokalnie operacji do wszystkich zdalnych węzłów systemu. Niestety propagowanie każdej operacji do pozostałych węzłów doprowadziło do częstego występowania rozproszonych zakleszczeń pomiędzy tymi operacjami (*distributed deadlocks*). Opracowana została zatem nowa metoda zapewniająca spójność replikacji, ROWAA (*Read One Write All Available*). W metodzie tej operacje wchodzące w skład transakcji przetwarzane są najpierw tylko w jednym węźle, a następnie do pozostałych węzłów systemu przesyłane są tylko modyfikacje na danych bez konieczności przesyłania dodatkowych komunikatów. Przesyłanie mniejszej liczby komunikatów przez sieć oczywiście znacznie obniża czas potrzebny do wykonania transakcji, zwiększając wydajność procesu replikacji. Przykładem realizacji tej metody jest algorytm optymistycznego blokowania dwufazowego O2PL (*Optimistic 2 Phase-Locking*) zaprezentowany w [4], będący jednym z pierwszych algorytmów mających na celu zapewnienie spójności replikowanych danych przy użyciu metody ROWAA. Algorytm ten wykorzystuje modyfikację algorytmu blokowania dwufazowego (2PL), w której rozróżniane są transakcje wykonywane w węźle lokalnym od transakcji zdalnych (rys. 1). Dzięki takiemu podejściu możliwe jest przewidywanie zakleszczeń i zmniejszanie ich ilości. Rozpoczęcie transakcji w momencie, gdy dostarczona jest tylko jej część, pozwala na skrócenie całkowitego czasu potrzebnego do zatwierdzenia transakcji. Po dostarczeniu całej transakcji określana jest jej właściwa kolejność i jeśli transakcja nie jest w konflikcie z poprzedzającymi ją transakcjami, jest ona zatwierdzana. W przeciwnym razie cała transakcja jest wykonywana jeszcze raz od początku.



Rys. 1. Skrócenie czasu potrzebnego do zatwierdzenia transakcji w przypadku zastosowania protokołu optymistycznego blokowania dwufazowego O2PL: a) czas wykonania transakcji w przypadku standardowego protokołu; b) czas wykonania transakcji dla protokołu O2PL

W celu zmniejszenia wpływu zakleszczeń na proces replikacji danych opracowane zostały również metody oparte na komunikacji grupowej GCS (*Group Communication Systems*). Metody oparte na GCS, np. opisane w [5–8], zapewniają mechanizm gwarantujący określoną kolejność dostarczanych komunikatów w sieci, jak również umożliwiają wykrywanie awarii w węzłach całego systemu. Najbardziej restrykcyjna kolejność dostarczania komunikatów wymaga, aby były one dostarczane w identycznej kolejności do wszystkich węzłów systemu, dzięki czemu możliwe jest zapobieganie występowania rozproszonych zakleszczeń. Przykładami praktycznych realizacji replikacji przy użyciu GCS może być algorytm BRP (*Basic Replication Protocol*) [5], czy też wspomniana już metoda opracowana w projekcie o nazwie Postgres-R [2, 9].

Wykorzystywanie metod GCS do każdego zastosowania czy też każdego typu transakcji niestety nie jest najskuteczniejszą techniką mającą na celu zapobieganie występowania zakleszczeń. Dzieje się tak, ponieważ w celu zapewnienia odpowiedniej kolejności dostarczania transakcji konieczna jest wymiana komunikatów pomiędzy węzłami. Taki nakład komunikacji sieciowej może znacznie obniżyć wydajność metody [10]. Dalsze prace nad rozwojem technik GCS doprowadziły do zmniejszenia wpływu opóźnień sieci na całkowity czas potrzebny do dostarczenia transakcji do węzłów w odpowiedniej kolejności. Przykładem takiego rozwiązania jest metoda Generic Broadcast, opisana w [11, 12], w której kolejność dostarczania ważna jest wyłącznie dla komunikatów związanych z transakcjami będącymi ze sobą w konflikcie. Pozostałe transakcje mogą oczywiście być dostarczane w dowolnej kolejności.

Innym przykładem jest metoda VOAB (*Optimistic Atomic Broadcast*) zaprezentowana w [13, 14]. W tej metodzie komunikaty dostarczane są tak, jak zostały odebrane, co umożliwia szybką aplikację zestawu danych (*writeset*) w zdalnych węzłach, pomimo oczekiwania na końcową kolejność dostarczonych transakcji przed ich zatwierdzeniem. W tym przypadku wycofywane są tylko te transakcje, które nie zostały zaaplikowane we właściwej kolejności, po czym są one aplikowane ponownie, już w odpowiedniej kolejności.

Techniki komunikacji grupowej wykorzystane zostały również w algorytmach epidemicznych (*epidemic algorithms*). Przykład algorytmu epidemicznego został przedstawiony w [15].

3. Konflikty

W przypadku realizacji asynchronicznej replikacji danych (*lazy repliaction*), w której zmiany dokonane w węźle lokalnym propagowane są z opóźnieniami, bardzo często występują konflikty. Konflikty występują, gdy co najmniej dwa węzły dokonują niespójnej modyfikacji tego samego rekordu.

Wyróżnia się trzy rodzaje konfliktów:

1) konflikt pomiędzy dwoma transakcjami dodającymi nowe dane (*insert – insert*) polegający na dodaniu rekordu o tym samym identyfikatorze (kluczu głównym) w różnych węzłach, co narusza ograniczenie unikalności i nie zezwala na realizację tych modyfikacji;
2) konflikt pomiędzy modyfikacjami (*update – update*) zachodzący, gdy jednocześnie w wielu różnych węzłach następuje próba modyfikacji tego samego rekordu;
3) konflikt pomiędzy operacjami zapisu i usuwania (*update – delete*) występujący podczas, gdy ten sam rekord jest w jednym węźle modyfikowany, a w innym usuwany.

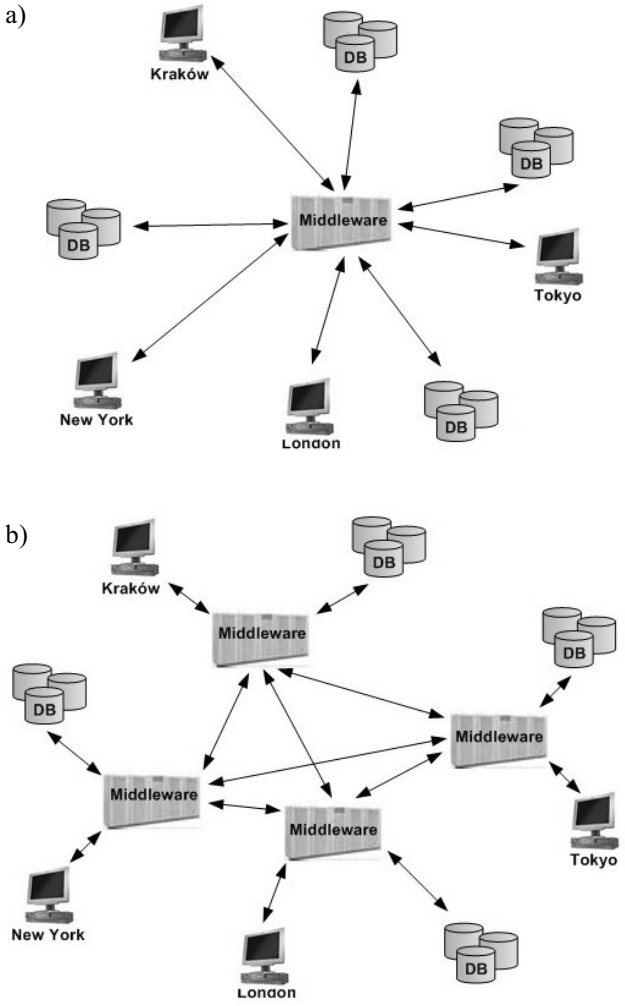
W celu minimalizacji wpływu konfliktów na proces replikacji stosuje się różne metody zapobiegania konfliktom replikacji (*conflict resolution protocols*). Jedną z nich jest generowanie globalnie unikalnych wartości klucza głównego rekordu. Może to być realizowane poprzez składanie identyfikatora wiersza z dwóch części, jednej będącej identyfikatorem bazy w której zachodzi operacja, oraz drugiej będącej wartością pocho-dzącą np. z sekwencji. Unikalność klucza głównego jest realizowana również poprzez utworzenie sekwencji o rozłącznych przedziałach generowanych wartości we wszystkich węzłach systemu, np.: {1,…,1000000} w węźle pierwszym, {1000001,…,2000000} w węźle drugim, w kolejnym {2000001,…,3000000}, itd. Stosowane mogą być również globalne identyfikatory GUI (*Globally Unique Identifier*), jednak metoda ta charakteryzuje się małą wydajnością i nie jest często stosowana.

Problem konfliktów jest również rozwiązywany przez zastosowanie metody przynależności danych (*data ownership*). W metodzie tej, opisanej w [17], każdy węzeł posiada możliwość modyfikacji tylko tych danych, których jest właścicielem. Pomimo dużej skuteczności i wydajności tej metody, jej zastosowanie nie jest duże, gdyż w wielu systemach wymagana jest duża konkurencyjność w dostępie do tych samych danych, co uniemożliwia jej zastosowanie.

Konflikty, których nie uda się wyeliminować, muszą być rozwiązane w odpowiedni sposób. W tym celu stosuje się różne metody, takie jak generacja unikalnych kluczy, znaczniki czasowe, priorytety węzłów czy funkcje matematyczne. Rozwiązywanie konfliktów bardzo często realizowane jest w warstwie middleware systemu, dzięki czemu nie ma konieczności dokonywania modyfikacji w samej bazie danych. Pozwala to na zwiększenie skalowalności (*scalability*) i przenośności (*portability*) systemu.

4. Replikacja z wykorzystaniem warstwy middleware

Głównym celem replikacji z zastosowaniem architektury wielowarstwowej jest zapewnienie spójności replikowanych danych (rys. 2). W przeciwieństwie do systemów ze zmodyfikowanym kodem motoru bazy danych, systemy replikacji budowane w oparciu o architekturę z warstwą pośrednią middleware, zapewniają przeźroczystość replikacji zarówno dla użytkowników tych systemów, jak i aplikacji. Replikacja danych w systemach wykorzystujących warstwę middleware nie wymaga żadnych modyfikacji w kodzie motoru bazy danych.



Rys. 2. Systemy replikacji danych w architekturze wielowarstwowej: a) z wykorzystaniem pojedynczego węzła w warstwie middleware; b) z warstwą middleware rozproszoną pomiędzy poszczególne lokalizacje

Przykładem replikacji z wykorzystaniem warstwy pośredniej jest zaprezentowany w [16] algorytm, w którym realizacja współbieżnego wykonywania transakcji oparta została na klasach konfliktów (*conflict classes*). W metodzie tej podstawowa klasa reprezentuje zestaw danych. Podział danych na klasy zależy od specyfiki aplikacji i prostym przykładem może być utworzenie klasy dla każdej tabeli. Rozwiązywanie konfliktów w przypadku takich prostych klas może odbywać się na poziomie wierszy w tabelach, przy czym konflikty mogą występować podczas odwoływania się do tych samych klas (tabel).

Dzięki zastosowaniu architektury wielowarstwowej i przeniesieniu do warstwy middleware części zadań realizowanych wcześniej przez motor bazy danych, istnienie wielu replik w systemie nie wpływa na konieczność wprowadzania zmian zarówno po stronie użytkowników, jak i kodu aplikacji. W warstwie middleware mogą być wykonywane operacje związane z ustalaniem kolejności wykonywania transakcji oraz rozwiązywaniem konfliktów dla wykonywanych transakcji, jeszcze przed przesłaniem ich do systemu baz danych. Do połączeń z replikowaną bazą używane są zwykle API programistyczne, udostępniające takie same funkcje, jak w przypadku obsługi pojedynczej instancji bazy danych.

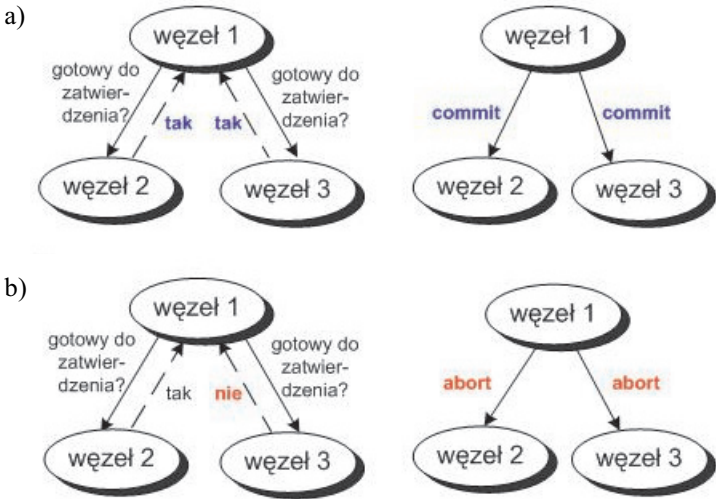
Zastosowanie warstwy pośredniej w systemach replikacji danych wpływa również na zwiększenie skalowalności całego systemu. W przypadku rozproszenia zarówno replik danych, jak i warstwy pośredniej, dodanie nowej lokalizacji sprowadza się praktycznie do dodania nowej repliki i nowego serwera middleware, co jest zwykle operacją dość prostą. W takim przypadku zwiększa się także znacznie odporność systemu na awarie pojedynczych węzłów czy elementów warstwy pośredniej (*fault-tolerance*). Uszkodzony pojedynczy element całego systemu może być z łatwością zastąpiony nowym, przy nieznacznym zmniejszeniu jego całkowitej wydajności, co może być zupełnie niezauważalne dla jego użytkowników.

5. Protokół dwufazowego zatwierdzania

W popularnych systemach zarządzania danymi, jak np. Oracle, IBM DB2, czy MS SQL Server, zaimplementowane zostały funkcjonalności umożliwiające wykonywanie transakcji rozproszonych (obejmujących wiele baz) z wykorzystaniem protokołu dwufazowego zatwierdzania (*Two-phase commit*). Protokół ten, przedstawiony schematycznie na rysunku 3, może być realizowany przy użyciu standardowych bibliotek stosowanych do współpracy aplikacji z bazami danych, np. JDBC (*Java Database Connectivity*). Został on również zaimplementowany w różnorodnych, stosowanych w praktyce oprogramowaniach typu middleware jak np. MQseries w IBM WebSphere, Bea Tuxedo czy Bea Weblogic.

W pierwszej fazie zatwierdzania transakcji w protokole dwufazowego zatwierdzania węzeł koordynujący wykonanie transakcji rozsyła ją do wszystkich pozostałych węzłów, a następnie na podstawie odpowiedzi ze wszystkich węzłów, podejmowana jest decyzja o jej zatwierdzeniu lub wycofaniu. W drugiej fazie następuje wykonanie transakcji, przy czym transakcja jest zatwierdzana tylko w przypadku, gdy wszystkie węzły potwierdzą gotowość do jej zatwierdzenia. Jeśli chociaż jeden z węzłów biorących udział w transakcji nie odpowie pozytywnie w wymaganym czasie, wówczas cała transakcja jest wycofywana.

W celu zapewnienia konkurencyjności dostępu do danych protokół ten najczęściej używa protokołu dwufazowego blokowania. Wadą zarówno protokołu dwufazowego zatwierdzania, jak i protokołów tworzonych na jego podstawie jest duży narzut czasowy, który w znacznym stopniu obniża wydajność protokołu.



Rys. 3. Protokół dwufazowego zatwierdzania: a) zatwierdzania transakcji (commit), gdy wszystkie węzły gotowe do jej zatwierdzenia; b) wycofanie transakcji (abort), przy braku możliwości zatwierdzenia transakcji w przynajmniej jednym węźle

Protokół dwufazowego zatwierdzania może być wykorzystywany również do realizacji replikacji, umożliwiając natychmiastową propagację modyfikacji dokonanych w lokalnej bazie danych do wszystkich pozostałych baz biorących udział w replikacji. Największą zaletą tego protokołu jest stałe utrzymywanie spójnych i integralnych danych we wszystkich węzłach systemu. Protokół ten, dzięki istnieniu interfejsów programistycznych dla wielu stosowanych w praktyce systemów baz danych, może być również wykorzystywany w środowiskach heterogenicznych. Niestety, niska wydajność spowodowana narzutem protokołu dwufazowego zatwierdzania oraz występowanie zakleszczeń związanych z dwufazowym blokowaniem wpływają na obniżenie całkowitej wydajności systemu replikacji zbudowanego z wykorzystaniem tego protokołu.

6. Podsumowanie

W systemach transakcyjnych w ostatnim czasie bardzo popularna stała się replikacja danych z wykorzystaniem architektury wielowarstwowej, w której zarządzanie wykonywaniem transakcji rozproszonych przeniesione zostało z motoru bazy danych do warstwy pośredniej middleware. Replikacja danych z wykorzystaniem warstw middleware jest obecnie

w fazie ciągłego rozwoju i stale pojawiają się nowe propozycje algorytmów, zarówno w literaturze, jak i w praktycznych implementacjach. Architektura z użyciem warstwy pośredniej umożliwia stosunkowo proste tworzenie rozwiązań o wysokiej dostępności (*high availability*).

Obecnie powstaje wiele różnych rozwiązań umożliwiających realizację replikacji w systemach transakcyjnych, np. [5, 14, 16, 18], w których zmniejszenie wydajności procesu replikacji w warstwie middleware może być minimalne lub całkowicie zredukowane, w porównaniu z wydajnością tego procesu w przypadku implementacji replikacji bezpośrednio w systemie baz danych. Dzieje się tak dlatego, że część operacji wykonywanych w motorze bazy danych zostaje przeniesiona do dodatkowej warstwy, tym samym zmniejszając obciążenie systemu bazy danych. Dodatkowo w warstwie middleware mogą być wykonywane różne operacje logiczne na danych, redukując tym samym nadmierne obciążenie po stronie aplikacji klienta oraz systemu bazy danych.

W przypadku tworzenia systemu replikacji z wykorzystaniem middleware zawsze występuje konieczność tworzenia kodu aplikacji. W praktyce wiąże się to z dodatkowym czasem i kosztami związanymi z implementacją aplikacji klienta współpracującej z warstwą middleware, oraz koszt samego rozwiązania middleware. Niestety, w przypadku niektórych systemów koszt ten może być bardzo duży, porównywalny z kosztem realizacji replikacji wykorzystującej modyfikacje jądra bazy danych.

Dzięki modularnemu podejściu do budowania systemu replikacji i rozkładaniu zadań pomiędzy większą liczbą aplikacji i serwerów uzyskuje się dużą skalowalność całego systemu, a sama rozbudowa jest stosunkowo prosta. Zastosowanie architektury wielowarstwowej pozwala również na zwiększenie odporności systemu na awarie poszczególnych węzłów, czy nawet całych lokalizacji, jednocześnie umożliwiając użytkownikom systemu dostęp do tych samych danych w wielu miejscach znajdujących się w dużych odległościach od siebie.

Oprócz wymienionych wcześniej zalet mechanizmu replikacji działającego w architekturze wielowarstwowej, replikacja z wykorzystaniem middleware ma jeszcze jedną bardzo istotną cechę – rozdzielenie mechanizmu zarządzania transakcjami rozproszonymi od motoru bazy danych umożliwia uniezależnienie mechanizmu dostępu do replik danych od konkretnej platformy. Dzięki takiemu podejściu możliwa jest realizacja replikacji danych w środowiskach heterogenicznych.

Literatura

[1] Gray J., Helland P., O'Neil P., Shasha D., *The dangers of replication and a solution*. ACM SIGMOD, 1996.

[2] Kemme B., Alonso G., *Don't be lazy, be consistent: Postgres-R, a new way to implement database replication*. VLDB, 2000.

[3] Bernstein P.A., Hadzilacos V., Goodman N., *Concurrency Control and Recovery in Database Systems*. Addison Wesley, 1987.

[4] Carey M.J., Livny M., *Conflict detection tradeoffs for replicated data*. ACM Press, 1991.

[5] Irún-Briz L., Decker H., de Juan-Marín R., Castro-Company F., Armendáriz-Iñigo J.E., Muñoz-Escoí F.D., *Madis: A slim middleware for database replication*. Berlin, Springer 2005.

[6] Armendariz J.E., Juarez J.R., Garitagoitia J.R., González de Mendívil J.R., Muñoz-Escoí F.D., *Implementing database replication protocols based on O2PL in a middleware architecture*. ACTA Press, 2006.
[7] Chockler G., Keidar I., Vitenberg R., *Group communication specifications: a comprehensive study*. ACM Computing Surveys, 2001.
[8] Hadzilacos V., Toueg S., *Related problems. Helion. A modular approach to fault-tolerant broadcasts and related problems*. Department of Computer Science, Cornell University, 1994.
[9] Kemme B., Alonso G., *A new approach to developing and implementing eager database replication protocols*. ACM Transactions on Database Systems, 2000.
[10] Défago X., Schiper A., Urbán P., *Total order broadcast and multicast algorithms: Taxonomy and survey*. ACM Computing Surveys, 2004.
[11] Aguilera M.K., Delporte-Gallet C., Fauconnier H., Toueg S., *Thrifty generic broadcast*. Springer, 2000.
[12] Pedone F., Schiper A., *Generic broadcast*. Springer, 1999.
[13] Kemme B., Pedone F., Alonso G., Schiper A., Wiesmann M., *Using optimistic atomic broadcast in transaction processing systems*. IEEE Transactions on Knowledge and Data Engineering, 2003.
[14] Vicente P., Rodrigues L., *An Indulgent Uniform Total Order Algorithm with Optimistic Delivery*. IEEE Computer Society, 2002.
[15] Holliday J., Steinke R.C., Agrawal D., Abbadi A.E., *Epidemic algorithms for replicated databases*. IEEE Transactions on Knowledge and Data Engineering, 2003.
[16] Patiño-Martinez M., Jiménez-Peris R., Kemme B., Alonso G., *Consistent database replication at the middleware level*. ACM, 2005.
[17] *Replication Strategies: Data Migration, Distribution and Synchronization*. Sybase White Paper, 2003.
[18] Lin Y., Kemme B., Patiño-Martinez M., Jiménez-Peris R., *Middleware based data replication providing snapshot isolation*. ACM Press, 2005.
[19] Khadzynov W., Maksymiuk M., *Realizacja replikacji w systemach heterogenicznych baz danych*. Wydawnictwa Komunikacji i Łączności, 2006.
[20] Thomasian A., *Database Concurrency Control: Methods, Performance, and Analysis*. Springer, 1996.
[21] Kindler E., *Serializability, Concurrency Control, and Replication Control*. Berlin, Springer 2000.
[22] Helal A.A., Heddaya A.A., Bhargava B.K., *Replication Techniques in Distributed Systems*. Kluwer Academic Pub., 1996.
[23] Wiesmann M., *Group communications and database replication: techniques, issues and performance*. Ecole Polytechnique Federale De Lausanne, 2002.
[24] Bartoli A., *Implementing a replicated service with group communication.* Elsevier North-Holland, 2004.