STUDIA INFORMATICA 2002
Volume 23 Number 2B (49)

Henryk KRAWCZYK, Jan ZIELIŃSKI
Politechnika Gdańska, Wydział ETI, Katedra Architektury Systemów Komputerowych

STRATEGIE REPLIKACJI W CELU ZWIĘKSZANIA WYDAJNOŚCI I WIARYGODNOŚCI APLIKACJI SIECIOWYCH

Streszczenie. Przedstawiono kategorie replikacji danych i serwerów stosowanych w środowiskach rozproszonych. Przeanalizowano parametry jakościowe podstawowych strategii i przedyskutowano sposoby ich implementacji w środowisku klient - serwer i równy z równym. Pokazano zależności między wydajnością, dostępnością i tolerancją w celu porównania różnego typu rozwiązań.

REPLICATION STRATEGIES FOR INCREASING PERFORMANCE AND DEPENDABILITY OF NETWORK APPLICATIONS

Summary. Categories of data and server replications used in distributed environments are determined. Quality parameters of basic replication strategies are analyzed and their implementation for client-server and peer to peer environments are discussed. Existing relationships between efficiency, availability and fault-tolerance for comparison of different solutions are shown.

1. Wstęp

Pod pojęciem replikacji, czyli zwielokrotniania, kryje się wiele zagadnień pochodzących z bardzo różnorodnych obszarów badań nad systemami informatycznymi. Najbardziej spektakularną i stanowiącą obecnie największe wyzwanie formą replikacji jest zwielokrotnianie całych serwerów lub nawet usług (mogących składać się z grup serwerów), jednak do technik replikacji zalicza się takie tak rozpowszechnione metody, jak przechowywanie kopii pewnej części danych na dyskach lokalnych maszyn pracujących w sieci w formie pamięci podręcznej (pamięci cache). Nie można zapominać także o sprzętowych realizacjach replikacji, np. w postaci macierzy dyskowych RAID.

Głównymi obszarami zastosowań technik replikacji są systemy rozproszone i bazy danych [1], jest to jednak stosunkowo kosztowne, zarówno pod względem sprzętowym, jak i wykorzystania zasobów takich, jak przepustowość łączy sieciowych. Stopień skomplikowania budowy aplikacji zwielokrotnionej, jak i metod jej zarządzania stanowi poważne ograniczenie. Niemniej wyróżnia się trzy podstawowe korzyści płynące z utworzenia systemu replikowanego [2]:

- zwiększenie wydajności przetwarzania,
- zwiększenie dostępności dla użytkowników,
- poprawienie tolerancji uszkodzeń.

Zwiększenie wydajności uzyskuje się dzięki rozdzieleniu zadań obliczeniowych lub pominięciu powolnych łącz. Istnienie wielu kopii tych samych danych oraz tych samych programów na niezależnych hostach pozwala odwoływać się do dowolnego z takich serwerów i wykonać usługę mimo zaistniałych w systemie uszkodzeń. Taka wysoka dostępność nie gwarantuje poprawnego wykonywania usług. Konieczna jest możliwość tolerowania błędów generowanych przez uszkodzony serwer (fałszywe lub nieaktualne dane), co np. osiąga się poprzez ustalenie poprawnego stanu systemu w drodze głosowania.

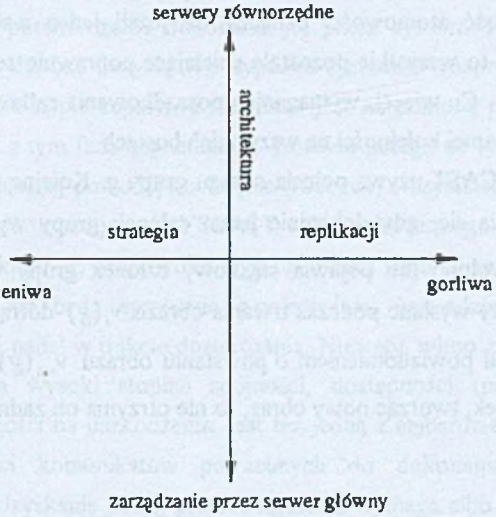
Każda realizacja (sprzętowa lub programowa) replikacji powinna spełniać dwa podstawowe wymagania. Pierwszym jest *przezroczystość* replikacji. Oznacza ona, że klient nie jest świadom zachodzenia procesu zwielokrotniania — z jego punktu widzenia grupa replikowanych serwerów udostępnia spójną usługę. Rolę bufora oddzielającego klienta od szczegółów implementacji może pełnić tzw. osłona — obiekt (proces) przyjmujący zapytania od klienta (często także udostępniający komunikację między klientami) i kierujący je do odpowiednich serwerów, a następnie przekazujący odpowiedź (dokładnie jedną, nawet jeśli w przetwarzaniu bierze udział więcej hostów). Drugim z wymagań jest *spójność*. Problemy ze spójnością występują tylko w przypadku, gdy przetwarzane są zapytania związane z modyfikacją danych przez klienta. Możliwe jest przyjmowanie zapytań klienta i następnie rozsyłanie uaktualnień do innych serwerów (podejście *asynchroniczne*) lub powstrzymanie się z potwierdzeniem przyjęcia danych, dopóki nie będzie pewności, że cała transakcja została dokonana na wszystkich hostach (podejście *synchroniczne*).

2. Kategorie replikacji

Przedmiotem replikacji jest zbiór obiektów nazywanych *replikami*, *zarządcami kopii* lub *serwerami*. Są to obiekty dostarczające pewnych usług (np. udostępnianie danych). Mogą

mieć postać pojedynczych procesów lub (częściej) fizycznych serwerów. Repliki odpowiadają na *zapytania klientów*.

Możliwych jest wiele rozwiązań replikacji serwerów, w zależności od planowanego zastosowania i dostępnych środków komunikacji oraz wymagań nakładanych na system. Podstawowe kategorie architektur replikowanych przedstawia rysunek 1.



Rys. 1. Kategorie replikacji
Fig. 1. Categories of replication

Istnieją dwa skrajne podejścia do komunikacji klienta z replikowaną usługą. W przypadku architektury równy-z-równym, klient może komunikować się z dowolnym z serwerów, czyli pełnią one role równorzędne. W modelu hierarchicznym klienci kierują wszystkie zapytania dotyczące aktualizacji danych do serwera-zarządcy. Odczyt danych nadal może odbywać się z dowolnej repliki.

Istnieją również dwie metody aktualizacji danych. Pierwsza dotyczy tzw. *replikacji leniwej*, w której operacja aktualizacji nie jest blokująca dla klientów — a wybrany serwer natychmiast potwierdza wykonanie operacji. Dane o uaktualnieniu rozsyłane są za pomocą tzw. plotek — komunikatów (tzw. gossip architecture). Metoda nie zapewnia pełnej spójności, bo informacja o aktualizacji nie dociera do wszystkich serwerów natychmiastowo. Innym rozwiązaniem jest tzw. replikacja gorliwa (eager replication), w której klient nie otrzymuje potwierdzenia wykonania operacji aktualizacji, dopóki wszystkie serwery jej nie zaakceptują. Istnieją, oczywiście, liczne rozwiązania pośrednie, w sposób rozmaity traktujące wymagania powyższych podstawowych modeli replikacji.

Często w implementacjach replikacji wykorzystywane są mechanizmy komunikacji grupowej, które pozwalają ukryć wymagane rozwiązania w warstwach protokołów. Do nich

należą problemy detekcji uszkodzeń, dołączania nowych replik i kasowania istniejących. W przypadku rozsyłania danych typu jeden-do-wielu (broadcast) wyróżniamy:

- rozsyłanie atomowe (ABCAST),
- rozsyłanie obrazowe (VSCAST).

Protokół ABCAST pozwala na zachowanie wymogów *atomowości* oraz *uporządkowania całkowitego*. Własność atomowości oznacza, że jeżeli jeden z serwerów dostarcza dwie wiadomości: m i m', to wszystkie pozostałe działające poprawnie serwery otrzymają również te dwie wiadomości. Co więcej, wymaganie uporządkowania całkowitego zapewnia, że będą one odebrane w tej samej kolejności na wszystkich hostach.

Protokół VSCAST używa pojęcia obrazu grupy g. Kolejne obrazy grupy oznacza się $v_i(g)$. Obraz zmienia się, gdy dokładnie jeden członek grupy występuje z niej (w wyniku awarii bądź dobrowolnie) lub pojawia się nowy członek grupy. Protokół gwarantuje, że wszystkie komunikaty wysłane podczas trwania obrazu $v_i(g)$ dotrą do wszystkich członków grupy (obrazu) przed powiadomieniem o powstaniu obrazu $v_{i+1}(g)$. Z kolei jeżeli do grupy wchodzi nowy członek, tworząc nowy obraz, to nie otrzyma on żadnego komunikatu z obrazu poprzedniego.

3. Opis protokołów replikacji

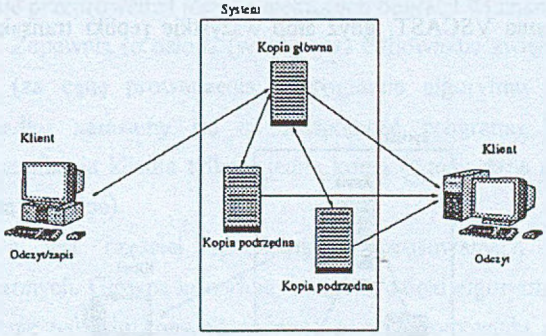
Poniżej przyjęto założenie (spełnione w systemach rozproszonych), że wszystkie transakcje składają się z pojedynczej operacji. Nie jest ono prawdziwe w przypadku replikacji baz danych, jednak wszystkie protokoły dają się łatwo uogólnić do tego przypadku ([2]).

3.1. Replikacja gorliwa

Replikacja gorliwa zakłada, że modyfikacja (aktualizacja) danych jest dla klienta blokująca. Replikacja gorliwa zapewnia największy stopień spójności — jeśli klient otrzymał potwierdzenie transakcji, ma pewność, że została ona dokonana na wszystkich serwerach w systemie, zatem nie ma znaczenia, do którego z nich będzie się odwoływał. Daje to zwiększoną odporność na uszkodzenia oraz zwiększa dostępność. Klient (lub osłona, przez którą komunikuje się on z systemem) może wybrać dowolny z serwerów, np. ten, który aktualnie jest najmniej obciążony, co prowadzi również do wzrostu wydajności.

W przypadku architektury nadrzędnej, wszystkie aktualizacje są kierowane przez klientów do kopii głównej (master copy) przechowywanej przez serwer zarządzający. Operacje odczytu mogą być prowadzone na dowolnej z replik i nie wymagają żadnych

operacji dodatkowych. Wykonanie aktualizacji rozpoczyna się od wysłania przez klienta odpowiedniego zapytania do serwera głównego. Ten wykonuje odpowiednią operację i przesyła polecenia dokonania aktualizacji do wszystkich kopii podrzędnych (slaves). Następuje faza uzgodnienia za pomocą protokołu 2PC (Two Phase Commit). Jeżeli wszystkie serwery dokonają potwierdzenia transakcji (commit), wtedy transakcja jest pomyślnie kończona i klient dostaje potwierdzenie dokonania jej przez system. Takie podejście jest bardziej charakterystyczne dla baz danych, w przypadku systemów rozproszonych polecenie dokonania aktualizacji przez kopie zapasowe rozsyłana jest za pomocą protokołu VSCAST i nie występuje w związku z tym faza uzgadniania. Różnica polega na tym, że w przypadku baz danych awaria kopii głównej powoduje cofnięcie (rollback) wszystkich transakcji w toku, natomiast w systemach wykorzystujących VSCAST rolę uszkodzonego serwera głównego może przejąć jedna z jego kopii. Z właściwości tego protokołu wynika, że polecenia wysyłane przez nowego zarządcę nie będą sprzeczne z poleceniami uszkodzonej kopii głównej, prawdopodobnie będącymi nadal w trakcie dostarczania. Niestety, mimo że gorliwa replikacja z kopią główną zapewnia wysoki stopień spójności, dostępności (przy wielu kopiach podrzędnych) oraz odporności na uszkodzenia, jest też jedną z najbardziej kosztownych pod względem zarówno ilości komunikatów potrzebnych do dokonania aktualizacji, jak i opóźnienia odpowiedzi. Uzyskanie pełnej przezroczystości wymaga albo użycia osłony, albo algorytmów automatycznej elekcji serwera zarządzającego. Obecnie architektura ta jest wykorzystywana głównie w prostych konfiguracjach. Przykładową architekturę systemu z kopią główną (niezależnie od tego, czy jest to replikacja gorliwa czy leniwa) przedstawia rysunek 2.

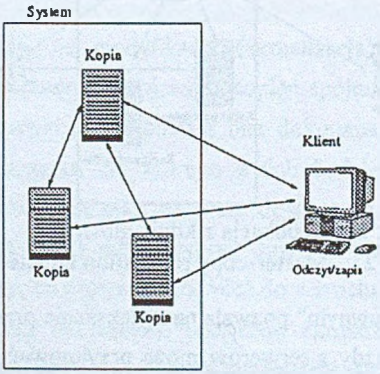


Rys. 2. Replikacja z kopią główną
Fig. 2. Master-copy replication architecture

Architektura „równy z równym" pozwala na zwiększenie przezroczystości replikacji nawet bez użycia osłon. Tutaj każdy z serwerów może przyjmować oba typy zapytań: zarówno polecenia odczytu, jak i modyfikacji danych. Pojawia się zatem problem zapewnienia spójności danych. Istnieją dwie zasadnicze metody rozwiązania tego zagadnienia. Pierwsze

określa się mianem *rozproszonego blokowania*. Po otrzymaniu zapytania (aktualizacji) od klienta, serwer (dowolna z replik) wysyła do pozostałych hostów żądanie zablokowania jednostek danych, których dotyczy aktualizacja. Każda z kopii może takiego pozwolenia udzielić lub nie. Serwer rozpocznie modyfikację danych tylko wtedy, gdy uzyska blokadę odpowiedniej jednostki danych na wszystkich serwerach. W przypadku nieotrzymania takiej zgody, transakcję opóźnia się lub odrzuca. Komunikaty pozyskiwania blokady zawierają informacje o przewidywanej operacji, zatem po pomyślnym zakończeniu procesu blokowania, transakcje są wykonywane we wszystkich jednostkach systemu, po czym następuje faza uzgodnienia transakcji za pomocą protokołu 2PC. Jeżeli i on przebiegnie pomyślnie, klient otrzymuje potwierdzenie dokonania operacji. Jest to spełnione dla baz danych. Systemy rozproszone radzą sobie inaczej, używając komunikacji grupowej w procesie zwanym replikacją pół-aktywną. Zamiast uzyskiwania rozproszonej blokady, zapytanie jest rozsyłane przez klienta do wszystkich serwerów przy użyciu protokołu ABCAST. Dzięki własnościom protokołu jest to równoważne z uzyskaniem globalnej blokady. Z kolei faza 2PC zastępowana jest rozsyłaniem decyzji za pomocą VSCAST. Rozesłanie to jest konieczne, ponieważ w przypadku replikacji pół-aktywnej zezwala się na niedeterministyczne (względem danych) zachowanie zarządców kopii.

Wprowadzenie determinizmu oraz komunikacji grupowej w bazach danych prowadzi do wspólnego modelu replikacji opartego na *rozsyłaniu atomowym*. Przyjmuje się, że klient wysyła zapytanie do grupy za pomocą ABCAST bądź do jednego z serwerów, a ten z kolei przekazuje je do pozostałych również za pomocą ABCAST. Ponieważ wszystkie serwery są deterministyczne (tzn. reagują na dane w ten sam sposób), nie ma potrzeby uzgadniania transakcji ani stosowania VSCAST, gdyż albo wszystkie repliki transakcję wykonają, albo odrzucą ją.



Rys. 3. Replikowana architektura typu równy-z-równym
Fig. 3. Master-copy replication architecture

Głównym zastosowaniem replikacji tego typu są rozwiązania wymagające maksymalnej odporności na błędy, ponieważ awaria kopii jest całkowicie przezroczysta dla systemu — nie ma nawet potrzeby wybierania nowej kopii głównej, co przy wykorzystaniu osłony daje pełną przezroczystość. Niestety, jest to również rozwiązanie niezwykle kosztowne pod względem kosztów komunikacji, choć przy wykorzystaniu środków sprzętowych do realizacji protokołu ABCAST staję się ona interesującą propozycją. Rysunek 3 przedstawia architekturę replikacji równy-z-równym.

3.2. Replikacja leniwa

W replikacji leniwej operacja odczytu przebiega analogicznie do przypadku zwielokrotniania gorliwego. Natomiast aktualizacja jest nieblokująca — serwer po akceptacji zapytania modyfikującego dane zwraca sterowanie klientowi. Dane o dokonanej operacji są zapisywane wraz z odpowiednimi znacznikami czasu (wprowadzającymi uporządkowanie przyczynowe). Kiedy zostanie dokonana odpowiednia ilość modyfikacji lub upłynie pewien czas, dane o wszystkich dokonanych ostatnio aktualizacjach zbierane są w jeden komunikat, tzw. „plotkę" i przesyłane do pozostałych serwerów. Cechę spójności danych zapewniają algorytmy porządkowania oparte na wspomnianych znacznikach czasu. Nie jest to jednak spójność całkowita — pewne serwery mogą być opóźnione w stosunku do kopii głównej. Klient komunikuje się z usługą poprzez osłonę lub bezpośrednio. W obu jednak przypadkach nie jest w pełni zachowana własność przezroczystości — należy odrzucić dane zwrócone przez serwer, który nie przeprowadził jeszcze niektórych operacji widzianych już przez klienta na innych serwerach. Zapewnia to osłona (wtedy czas odpowiedzi zwiększa się) lub musi to uczynić sam klient (za cenę prowadzenia w programie algorytmu znaczników czasu). W przeciwnym wypadku narażamy się na niespójność programu klienta. Wyjściem ostatecznym jest komunikacja klienta tylko z jedną kopią (wtedy dane są na pewno spójne, choć, być może, nie są aktualne).

Replikacja leniwa jest częściej spotykana w zastosowaniach bazodanowych niż w systemach rozproszonych. Główną korzyścią, osiąganą dzięki algorytmom zwielokrotniania leniwego, jest obniżenie nakładu komunikacyjnego przy dokonywaniu aktualizacji, poprzez rozsyłanie informacji o wielu zmianach w jednym komunikacie. Wymóg spójności danych jest zachowany w sposób nie całkowicie przezroczysty dla użytkownika, zwłaszcza jeśli nie wykorzystuje on osłony. Replikacja leniwa ma obniżoną zdolność do tolerowania awarii, ponieważ uszkodzenie hosta w każdym innym stanie niż po wysłaniu „plotki" a przed przyjściem kolejnych modyfikacji, może być związane z nieodwracalną utratą danych. Zalety tego podejścia dotyczą dostępności — kopia może szybciej niż w przypadku replikacji

gorliwej przejąć rolę uszkodzonego zarządcy (w modelu kopii głównej) i wydajności — dostępnych jest wiele hostów, a komunikacja między nimi nie zakłóca ruchu w sieci (jest niewielka).

4. Próba analizy jakościowej

4.1. Wydajność

Najczęściej stosowaną miarą wydajności w systemach klient-serwer jest czas odpowiedzi serwera na żądania klienta. Uwzględnimy to przy analizie czasu odpowiedzi serwera podlegającego replikacji według podanych wyżej schematów zwielokrotniania. Co gorsza, na wydajność ma wpływ również architektura systemu z serwerem zarządzającym lub serwery równoległe.

Czas odpowiedzi systemu oznacza czas, który upływa od momentu połączenia się klienta z serwerem do momentu powiadomienia klienta o dokonaniu transakcji. Niech czas T_u oznacza czas dokonania operacji modyfikacji danych (*Update*), a T_r czas trwania odczytu danych (*Read*). Zazwyczaj $T_u > T_r$, jeśli nie wliczamy czasu transmisji danych. Innymi słowy, interesuje nas czas przetwarzania transakcji (zapytania) przez serwer. Oznaczmy czasy przetwarzania danych przez system nie podlegający replikacji jako odpowiednio T_{unr} i T_{rnr}. Przyjmijmy, że wynoszą one: $T_{unr} = T_u$ i $T_{rnr} = T_r$.

Rozpatrzmy najpierw przypadek replikacji gorliwej opartej na architekturze serwera zarządzającego. Wtedy czas modyfikacji danych wynosi:

$$T_{uemc} = T_u + T_k + T_u + T_{2PC}, \tag{1}$$

gdzie: T_u — czas przetwarzania zapytania przez serwer główny, a następnie przez serwery potomne (stąd drugi czynnik T_u we wzorze), T_k — czas komunikacji, tzn. przekazania transakcji serwerom podrzędnym przez serwer główny, T_{2PC} — czas fazy ostatecznego zatwierdzenia transakcji za pomocą protokołu 2PC. Przy założeniu, że kopia główna najpierw rozsyła informacje o transakcji do kopii podrzędnych, a następnie wszystkie serwery wykonują transakcję równolegle, wzór (1) można sprowadzić do postaci:

$$T_{uemc} = T_k + T_u + T_{2PC}. \tag{2}$$

Czas odczytu jest taki sam niezależnie od tego, z której kopii odbywa się transfer danych i wynosi $T_{remc} = T_r$. Dla architektury o równorzędnych serwerach mamy:

$$T_{uepp} = T_b + T_u + T_{2PC}. \tag{3}$$

Czas T_b oznacza czas uzyskania rozproszonej blokady danych potrzebnej do dokonania

transakcji i wynosi: $T_b = 2T_k + T_i$, czyli podwojony czas komunikacji (zapytanie i odpowiedź) zwiększony o pewien czas przetwarzania T_i, podczas którego serwery podejmują decyzję, czy udzielić blokady, czy też nie. Po uzyskaniu blokady następują fazy przetwarzania transakcji i wykonania protokołu 2PC. Ponieważ $T_b > 2T_k$, uzyskujemy: $T_{uemc} > T_{uepp}$, a to oznacza formalne potwierdzenie, że replikacja oparta na architekturze serwera nadrzędnego może szybciej przeprowadzać operacje aktualizacji danych. Oczywiście, czas dokonania odczytu jest taki sam w obu przypadkach (wynosi T_r).

Replikacja leniwa jest znacznie korzystniejsza pod względem wydajności rozpatrywanej z punktu widzenia klienta. Czasy odpowiedzi na dwa podstawowe typy żądań dla usługi z kopią główną można zapisać jako:

$$T_{ulmc} = T_u + \Delta T_{umc} \text{ i } T_{rlmc} = T_r + \Delta T_{rmc}, \tag{4}$$

gdzie ΔT_{umc} i ΔT_{rmc} oznaczają narzut czasowy powstały w wyniku przetwarzania przez serwery danych o transakcjach dokonanych na pozostałych kopiach. Kopia główna tworzy „plotki" i rozsyła je (w tle), kopie podrzędne przyjmują „plotki" oraz dokonują transakcji w nich opisanych. W przypadku architektury równorzędnych serwerów otrzymujemy analogicznie: $T_{ulpp} = T_u + \Delta T_{upp}$ i $T_{rlpp} = T_r + \Delta T_{rpp}$, jednak tutaj narzut może być większy, ponieważ wszystkie serwery tworzą i wysyłają komunikaty o transakcjach oraz je wykonują. Czasy opóźnień wymykają się modelowaniu analitycznemu ze względu na duży stopień komplikacji i niedeterminizmu. Niemniej jednak można założyć, że są one znacznie mniejsze niż narzut powstający w przypadku replikacji gorliwej, czyli $\Delta T_{umc} < T_k + T_u + T_{2PC}$ i $\Delta T_{upp} < T_b + T_{2PC}$.

Łatwo zauważyć, że wydajność serwerów replikowanych rozpatrywana jedynie z punktu widzenia czasu odpowiedzi w zależności od wykorzystanego algorytmu jest daleko niewystarczająca. Wynika z niej jasno, że wykorzystanie replikacji prowadzi nieodwołalnie do pogorszenia się parametrów serwera. System serwerów replikowanych zanurzony jest w pewnym środowisku, którego istotnym składnikiem jest m.in. sieć komputerowa łącząca kopie. Przy tym istotna jest nie tylko jej ogólna przepustowość, ale także topologia połączeń, ich niezawodność oraz względna prędkość (w przypadku sieci heterogenicznych), a nawet dobowe wahania jej obciążenia. Dodatkowo każdy serwis ma swoją własną charakterystykę ruchu, opisywaną przez liczbę zapytań w jednostce czasu, średni czas trwania transakcji czy stosunek liczby aktualizacji do operacji odczytu. Kiedy uwzględni się te wszystkie parametry, może okazać się (i dzieje się tak w praktyce), że replikacja może jednak przynosić (czasem znaczne) zyski wydajnościowe. Początkowo podejmowano próby tworzenia w pełni

analitycznych modeli replikacji, jednak obecnie wydaje się, że zagadnienia analizy wydajnościowej środowisk replikowanych należy raczej rozwiązywać za pomocą symulacji.

4.2. Dostępność i tolerancja uszkodzeń

Załóżmy, że prawdopodobieństwo awarii pojedynczego elementu, który mógłby podlegać zwielokrotnieniu, wynosi p. Wtedy dostępność definiowana jako prawdopodobieństwo możliwości dostępu do sprawnego systemu wynosi $q = 1 - p$. Należy jeszcze uwzględnić prawdopodobieństwo awarii łącza, które oznaczmy p_l. Wtedy dostępność wynosi: $q = (1-p)(1-p_l)$. Przykładowo, przy prawdopodobieństwie awarii węzła $p = 10\%$ i awarii (niedostępności — np. w wyniku przeciążenia) łącza $p_l = 20\%$ otrzymujemy dostępność $q = 0{,}72 = 72\%$. Ogólnie, prawdopodobieństwo awarii układu serwer-łącze wynosi $p_s = p + p_l - pp_l$, natomiast funkcję dostępności w przypadku obecności n niezależnych kopii (wraz z łączami) można opisać wzorem: $q = 1 - p_s^n$. Funkcję tolerancji błędów można zdefiniować następująco:

$$FT(n) = \begin{cases} \frac{n}{2} - 1; & n = 2k, k \in N - \{0\} \\ \left\lfloor \frac{n}{2} \right\rfloor; & n = 2k+1, k \in N \end{cases} \tag{5}$$

Jej wartość obrazuje maksymalną liczbę serwerów, które mogą ulec awarii (tzn. mogą produkować błędne wyniki) bez zakłócania pracy systemu w zależności od całkowitej liczby serwerów w grupie replikowanej (n). Oczywiste jest, że w przypadku braku replikacji $n = 1$, $FT(1) = 0$, czyli nie występuje tolerancja uszkodzeń. Słabością architektury tego typu jest pojedynczy punkt wykonywania aktualizacji. W związku z tym należy osobno rozpatrzeć dostępność i tolerancję uszkodzeń względem zapisu i względem odczytu. Tak więc dla zapisu liczba replik wynosi $n = 1$, a więc funkcja dostępności przyjmuje postać $q = 1 - p_s$, czyli redukuje się do braku zwielokrotniania. Podobnie, $FT(1) = 0$. Dla odczytu sytuacja jest znacznie bardziej korzystna, ponieważ ilość punktów odczytu wynosi n, przy $n-1$ kopiach potomnych. Funkcje powyższe mogą przyjąć znacznie bardziej skomplikowane postaci w przypadku replikacji leniwej, w której kopie potomne mogą mieć dane z różnych faz działania systemu. Jeśli każda z kopii zawiera inne dane, nie jesteśmy w ogóle w stanie rozstrzygnąć, która z odpowiedzi jest prawidłowa. Dlatego w przypadku replikacji leniwej należy rozpatrywać raczej zagadnienia dostępności niż tolerancji uszkodzeń. Podobnie ma się rzecz w przypadku architektury o równoległych serwerach.

6. Podsumowanie

Problemy replikacji zyskują na znaczeniu wraz ze wzrostem popularności systemów rozproszonych oraz zwiększającymi się względem nich wymaganiami wydajnościowymi i wiarygodnościowymi. Ocena satysfakcji z zastosowania odpowiednich mechanizmów może być dokonana na drodze symulacji. Do stałych poszukiwań zaliczyć należy zwiększanie przezroczystości i wydajności oraz odporności na uszkodzenia systemów replikowanych. Replikację rozważać można nie tylko na poziomie całych serwerów czy usług, ale także na poziomie sprzętowym czy też zwielokrotniania obiektów w systemach takich, jak FTCORBA (Fault Tolerance CORBA – specyfikację podaje dokument [7]). Zmiana poziomu abstrakcji dostarcza kolejnych wyzwań, przy czym opisane powyżej algorytmy i architektury mogą znaleźć zastosowanie również w tym przypadku.

LITERATURA

1. Coulouris G., Dollimore J., Kindberg T.: Systemy rozproszone: podstawy i projektowanie. WNT, Warszawa 1998.
2. Wiesmann M., Pedone F., Schiper A., Kemme B., Alonso G.: Understanding Replication in Databases and Distributed Systems: http://lsewww.epfl.ch/Documents/html/WPS+00.html
3. Ingham D. B., Shrivastava S. K., Panzieri F.: Constructing Dependable Web Services. IEEE Internet Computing, Jan. Feb. 2000, str. 25 - 33
4. Information & Communcations Systems Research Group, ETH Zurich and LSE, EPF Lausanne. *DRAGON: Database Replication Based on Group Communication*, May 1998: http://www.inf.ethz.ch/department/IS/iks/research/dragon.html.
5. Umiński P.: Praca doktorska. Strategia niezawodnego i efektywnego przetwarzania zadań w systemach rozproszonych. Politechnika Gdańska, Gdańsk 1999.
6. Satyanarayanan M., Kistler J. J., Kumar P., Okasaki M. E., Siegel E. H. and Steere D. C.: Coda: A Highly Available File System for a Distributed Workstation Environment, *IEEE Transacions on Computers 39*, 4 (April 1990).
7. OMG Group: Fault-Tolerant CORBA specification v1.0, Grudzień 2000: http://grouppac.sourceforge.net/groupac/00-04-04.pdf

Recenzent: Prof. dr hab. inż. Andrzej Grzywak

Wpłynęło do Redakcji 5 kwietnia 2002 r.

Abstract

The paper presents replication strategies used in distributed systems. General architecture of a replicated system is discussed and such advantages as system efficiency, system availability for user and fault-tolerance are considered. For both two kinds of system architectures (peer-to-peer and master copy ones) and two kinds of replication strategies (see Fig. 1), four categories of replication are defined and basic quality characteristics, such as consistency and transparency are discussed. Besides, communication protocols and data consistency mechanisms for master copy (Fig. 2) and peer-to-peer (Fig. 3) replication architectures are analyzed. To compare different solutions the suitable formulas for expressing efficiency and dependability are given. They allow to estimate impacts of replication mechanisms on quality of distributed systems.