ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: INFORMATYKA z. 34 Nr kol. 1381

Henryk KRAWCZYK, Bartosz KRYSZTOP
Politechnika Gdańska, Katedra Architektur Systemów Komputerowych

ALGORYTMY ODWZOROWANIA UMOŻLIWIAJĄCE ZWIĘKSZENIE TESTOWALNOŚCI APLIKACJI ROZPROSZONYCH

Streszczenie. Zaproponowano nowe kryterium odwzorowania modułów aplikacji na system rozproszony (sieć LAN z systemem PVM) mające na celu zapewnienie nie tyle minimalnego czasu przetwarzania, co skrajnych warunków wykrywalności błędów uwarunkowanych czasowo. W oparciu o to kryterium sformułowano dwa heurystyczne algorytmy odwzorowania i przeprowadzono krótką ocenę ich przydatności dla celów testowania aplikacji.

ASSIGNMENT ALGORITHMS FOR INCREASING DISTRIBUTED APPLICATIONS TESTABILITY

Summary. The paper describes a new criteria for assigning modules of an application to a distributed system (LAN network with PVM system) which is designed not to minimize execution time, but to increase testability of the application for time depended errors. Basing on this criteria we present two heuristic algorithms and provide brief evaluation of their usability for application testing purpose.

1. Wprowadzenie

Testowanie aplikacji rozproszonych jest rzeczą trudniejszą i bardziej złożoną niż testowanie aplikacji sekwencyjnych. Przyczyn jest wiele, jedna z nich to nowa klasa błędów, których występowanie jest uwarunkowane czasowo. Dodatkowym problem jest trudność w ustaleniu i odtworzeniu sytuacji, która prowadzi do takiego błędu. W pracy zaproponowano oryginalną metodę odwzorowania aplikacji umożliwiającą wykrycie tego typu błędów.

Przez *aplikację rozproszoną* rozumiemy program składający się z *modułów*, które komunikują się ze sobą za pomocą *wiadomości*. Aplikacja taka jest opisana jako graf

$G_A(M,C)$, gdzie $M=\{1,2,\ldots,m\}$ to zbiór wierzchołków oznaczających moduły aplikacji oraz C to zbiór krawędzi łączących wierzchołki. Istnienie krawędzi $c_{ij} \in C$ oznacza, że moduły i oraz j komunikują się ze sobą w trakcie wykonywania aplikacji (tzn. i wysyła wiadomości do j i/lub j wysyła wiadomości do i).

Przez *system rozproszony* rozumiemy zbiór procesorów (komputerów). Procesory są ze sobą połączone za pomocą kanałów komunikacyjnych, które wprowadzają pewne opóźnienie pomiędzy czasem wysłania a czasem odebrania przesyłanej wiadomości. System rozproszony jest opisany za pomocą grafu pełnego $G_S=(P,D)$, gdzie $P=\{1,2,\ldots,p\}$ to zbiór wierzchołków oznaczających procesory oraz D to macierz o rozmiarze $p \times p$, która zawiera wagi d_{ij} przypisane krawędziom grafu oznaczające opóźnienie, jakie dany kanał wprowadza pomiędzy procesorami i oraz j. Pola d_{ii} oznaczają opóźnienie, jakie można zaobserwować przy przesyłaniu wiadomości pomiędzy dwoma modułami znajdującymi się na tym samym procesorze i. Macierz D jest symetryczna, tzn. $d_{ij}=d_{ji}$. Dla systemu z rys. 1 macierz przyjmuje następującą wartość:

$$D=\begin{bmatrix} 1 & 4 & 3 \\ 4 & 1 & 3{,}5 \\ 3 & 3{,}5 & 1 \end{bmatrix} \tag{1}$$

Przez *odwzorowanie* α rozumiemy funkcję

$$\alpha: M \rightarrow P \tag{2}$$

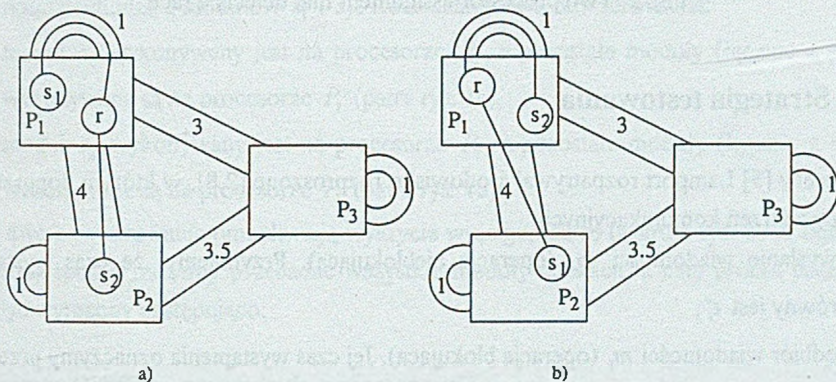
która przyporządkowuje każdemu modułowi i ze zbioru M procesor $k=\alpha(i)$ ze zbioru P. Kiedy odwzorowanie ma miejsce, krawędzie c_{ij} grafu G_A otrzymują wagi, które są równe opóźnieniu wprowadzanemu przez kanał, za pośrednictwem którego moduły i oraz j komunikują się ze sobą: $c_{ij}(\alpha)=d_{\alpha(i)\alpha(j)}$. Przykładowe odwzorowanie $\alpha_1=\{s_1 \rightarrow P_1, r \rightarrow P_2, s_2 \rightarrow P_2\}$ przedstawione jest na rys. 1a, zaś $\alpha_2=\{s_1 \rightarrow P_2, r \rightarrow P_2, s_2 \rightarrow P_1\}$ na rys. 1b.

Odwzorowania aplikacji na system rozproszony dokonuje się najczęściej w celu zwiększenia wydajności przetwarzania tej aplikacji [1,4,6,7]. Kryterium takiego odwzorowania zwane *kryterium wydajnościowym* można sformułować jako zadanie minimalizacji funkcji kosztu przetwarzania PC, której postać w szczególności zależy od przyjętego modelu, lecz na ogół ma postać zbliżoną do:

$$PC(\alpha)=\sum_{i\in M} e_{i\alpha(i)} \tag{3}$$

gdzie $e_{i\alpha(i)}$ jest kosztem przetwarzania modułu i na procesorze $\alpha(i)$.

Drugim najczęściej spotykanym kryterium odwzorowania jest wiarygodność danej aplikacji [3,8]. W tej sytuacji ma miejsce replikacja modułów aplikacji, aby była ona bardziej odporna na błędy pojawiające się w systemie, przy czym wiarygodność aplikacji szacuje się poprzez dopuszczalną liczbę uszkodzonych procesorów, które nie zakłócą pracy całej aplikacji. Kryterium doboru odwzorowania jest następujące: znaleźć odwzorowanie α takie, że $PC(\alpha)$ jest minimalne, oraz aplikacja jest k-wiarygodna (tzn. jest odporna na uszkodzenie maksymalnie k procesorów spośród wszystkich wykonujących tę aplikację).



Rys. 1. Przykładowe odwzorowania aplikacji na system
Fig. 1. An example assignments of an application to a system

W pracy sformułowano nowe kryterium doboru odwzorowania, którym jest maksymalizacja testowalności aplikacji. Testowalność jest wielkością, którą szacuje się za pomocą dwóch parametrów:

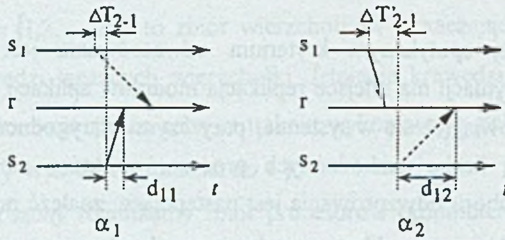
1. współczynnik *wydajności* testowania (e), tj.:

e=średnia ilość cykli testowania koniecznych do wykrycia pierwszego błędu (4)

2. współczynnik *pokrycia testowania* (c) dla zadanej liczby cykli testowych, tj.:

c=stosunek średniej ilości wykrytych błędów do ilości wszystkich błędów w aplikacji (5)

Pojawia się więc problem znalezienia takiej strategii odwzorowania aplikacji na system rozproszony, aby spełnić jedno z powyższych kryteriów testowalności.



$\alpha_1 = \{s_1 \to P_1, r \to P_2, s_2 \to P_2\}$ $\alpha_2 = \{s_1 \to P_2, r \to P_2, s_2 \to P_1\}$ (patrz rys. 1a,b)

Rys. 2. Wykrycie wyścigu przez odwzorowanie dwuprocesorowe
Fig. 2. Two processor assignment that detects a race

2. Strategia testowania

W pracy [5] Lamport rozpatrywał środowisko rozproszone [2,8], w którym dopuszcza się dwa typy zdarzeń komunikacyjnych:

- wysłanie wiadomości m_i (operacja nieblokująca). Przyjmiemy, że czas wystąpienia równy jest t_i^s,
- odbiór wiadomości m_i (operacja blokująca). Jej czas wystąpienia oznaczymy przez t_i^d.

W praktyce, aby określić relację porządkującą wiadomości dla wykonującej się aplikacji (nazywaną *scenariuszem wykonania*), konieczne jest zarejestrowanie czasów t_i^s oraz t_i^d dla wszystkich zdarzeń komunikacyjnych. Przez *wystąpienie wyścigu* (który jest objawem błędu uwarunkowanego czasowo) rozumiemy dwukrotne wykonanie aplikacji w taki sposób, że jej scenariusze wykonania się różnią. Oznacza to, że istnieje taki moduł, w którym kolejność odbioru wiadomości różni się w tych obu wykonaniach. Przykład takiej sytuacji dla systemu z rys. 1 przedstawia rys. 2.

Zgodnie powyższymi założeniami zaproponowano strategię testowania aplikacji polegającą na wielokrotnym uruchomieniu aplikacji dla tych samych danych wejściowych i zapisu ich scenariuszy wykonania. Następnie przez porównywanie uzyskanych zapisów można zlokalizować wystąpienie ewentualnego wyścigu. Aby testowanie było efektywne, należy w odpowiedni sposób „potrząsnąć" aplikacją, tak aby potencjalny błąd czasowy ujawnił się. To uzasadnia wykonywanie aplikacji dla różnych odwzorowań. Poniżej przedstawiono dwie takie heurystyczne strategie odwzorowania i porównano ich skuteczność właśnie dla celów testowania aplikacji.

3. Odwzorowanie dwuprocesorowe

Odwzorowanie dwuprocesorowe jest jednym z najprostszych odwzorowań z punktu widzenia testowalności i zakłada rozmieszczenie modułów aplikacji na dwóch procesorach: P_1 i P_2. Łącze pomiędzy procesorami charakteryzuje się opóźnieniem d_{12}. Moduły znajdujące się na tym samym procesorze P_1 komunikują się z opóźnieniem d_{11}, które jest znacznie mniejsze od d_{12}, tj.: $d_{11} < d_{12}$.

Zgodnie z rys. 2 szukamy wystąpienia wyścigu w module r, gdzie dwie „ścigające się" wiadomości pochodzą z modułów s_1 i s_2. Rozważamy dwa odwzorowania:

- moduł s_1 wykonywany jest na procesorze P_2, a pozostałe moduły (łącznie z r i s_2) wykonywane są na procesorze P_1 (patrz rys. 1a),
- moduł s_2 wykonywany jest na procesorze P_2, a pozostałe moduły (łącznie z r i s_1) wykonywane są na procesorze P_1 (patrz rys. 1b).

Te dwa odwzorowania umożliwiają wykrycie wyścigu, jak to ilustruje rys. 2. W ogólności w podobny sposób możemy przebadać wszystkie moduły aplikacji, a cały proces testowania może być wyrażony następująco:

Algorytm OD (Odwzorowanie dwuprocesorowe)

1: **for** i=1 **to** m **do**
2: Przyporządkuj moduł i do procesora P_2.
3: Przyporządkuj moduły 1,2,...,i-1,i+1,...,m do procesora P_1.
4: Wykonaj aplikację i zarejestruj jej zachowanie.
5: **end for**
6: Porównaj otrzymane zapisy i zgłoś ewentualne wystąpienie wyścigu.

Zastosowanie algorytmu OD umożliwia wykrycie wyścigów w aplikacji rozproszonej, o ile spełnione są odpowiednie zależności czasowe. Szczegółową analizę testowalności przedstawiono w rozdziale 5.

4. Odwzorowanie wieloprocesorowe

W celu przedstawienia odwzorowania wieloprocesorowego wprowadzamy kilka nowych pojęć. O kolejności zaistnienia w czasie zdarzeń komunikacyjnych decydują opóźnienia komunikacyjne pomiędzy modułami c_{ij}, a te zależą od przyjętego odwzorowania α oraz

macierzy D. Pojedynczą kombinację wag $c_{ij}(\alpha)$ nazywamy *stanem komunikacyjnym* danego odwzorowania α i oznaczamy przez $C(\alpha)$. Często się zdarza, że różne odwzorowania $\alpha \neq \beta$ mają ten sam stan komunikacyjny $C(\alpha) = C(\beta)$, co oznacza, że scenariusz wykonania danej aplikacji będzie w obu przypadkach ten sam. Zestawienie wszystkich możliwych odwzorowań dla aplikacji i systemu z rys. 1 znajduje się w tabelce na rys. 3a. Natomiast rys. 3b przedstawia zbiór wszystkich, ponumerowanych stanów komunikacyjnych odpowiadających podanym odwzorowaniom.

Ideą odwzorowania wieloprocesorowego (OW) jest takie wygenerowanie kolejnych odwzorowań, aby szanse na wczesne wykrycie wyścigu były jak największe (patrz kryterium 4). W tym celu wprowadza się funkcję $distance(\alpha, \beta)$, która definiuje „odległość" pomiędzy dwoma odwzorowaniami α i β jako odległość pomiędzy ich stanami komunikacyjnymi. Miarą tej odległości może być wybrana metryka, na przykład:

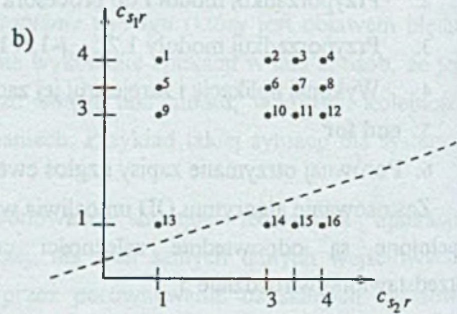
$$|C(\alpha) - C(\beta)| = \sqrt{\sum_{c_{ij} \in C} \left(c_{ij}(\alpha) - c_{ij}(\beta)\right)^2} \qquad (6)$$

Wówczas odległość odwzorowania α od zbioru odwzorowań B określona jest wzorem:

$$dist(\alpha, B) = \min_{\beta \in B} distance(\alpha, \beta) \qquad (7)$$

a)

s_1	s_2	r	poł.	s_1	s_2	r	poł.
P_1	P_1	P_1	13	P_2	P_2	P_3	7
P_1	P_1	P_2	4	P_2	P_3	P_1	2
P_1	P_1	P_3	10	P_2	P_3	P_2	15
P_1	P_2	P_1	16	P_2	P_3	P_3	5
P_1	P_2	P_2	1	P_3	P_1	P_1	9
P_1	P_2	P_3	11	P_3	P_1	P_2	8
P_1	P_3	P_1	14	P_3	P_1	P_3	14
P_1	P_3	P_2	3	P_3	P_2	P_1	12
P_1	P_3	P_3	9	P_3	P_2	P_2	5
P_2	P_1	P_1	1	P_3	P_2	P_3	15
P_2	P_1	P_2	16	P_3	P_3	P_1	10
P_2	P_1	P_3	6	P_3	P_3	P_2	7
P_2	P_2	P_1	4	P_3	P_3	P_3	13
P_2	P_2	P_2	13				



Rys. 3. Możliwe odwzorowania przykładowej aplikacji (a) oraz rozkład ich stanów komunikacyjnych (b)

Fig. 3. Possible assignments of an example application (a) and distribution of their communication states (b)

Bazując na (6) i (7) możemy podać następujący algorytm odwzorowania:

Algorytm OW (Odwzorowanie wieloprocesorowe)

1: Wykonaj aplikację z zadanym odwzorowaniem początkowym α i zarejestruj jej wykonanie na poziomie zdarzeń komunikacyjnych.

2: Określ (na podstawie zapisu - logu) graf aplikacji niezbędny do określenia wszystkich stanów komunikacyjnych.

3: $B \leftarrow \{\alpha\}$

4: **repeat**

5: Znajdź w dziedzinie odwzorowań takie odwzorowanie α, dla którego funkcja $dist(\alpha, B)$ (równanie 7) jest maksymalna.

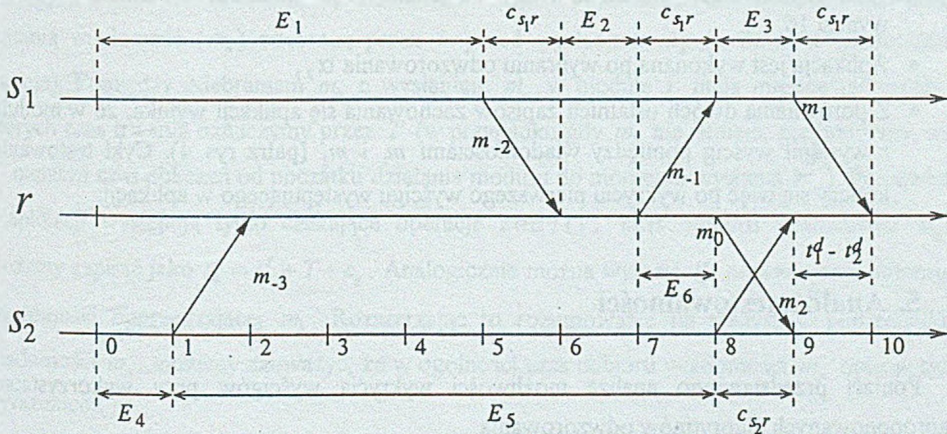
6: Wykonaj aplikację z odwzorowaniem α i zarejestruj jej wykonanie.

7: Porównaj otrzymany zapis z już uzyskanymi i zgłoś ewentualne wystąpienie wyścigu.

8: $B \leftarrow B \cup \{\alpha\}$

9: **until** spełnione kryterium zakończenia.

Wadą tego rozwiązania jest wykładnicza złożoność operacji wykonywanych w punkcie 5 algorytmu OW. W tym miejscu można wywołać dowolny heurystyczny algorytm szukający kolejnego odwzorowania. Kryterium zakończenia (punkt 9) może być różne w zależności od aktualnych preferencji. Najczęściej testowanie zostaje zakończone w momencie, gdy zostanie wykryty pierwszy błąd lub osiągnięta została przyjęta z góry ilość cykli testowania.



Rys. 4. Czasowy zapis zachowania się aplikacji
Fig. 4. Space-time log of an application behavior

Rozpatrzmy przykład zachowania się algorytmu OW dla aplikacji i systemu przedstawionych na rys. 1.

- Pierwsze wykonanie ma miejsce na procesorze P_1 (wszystkie moduły na tym samym procesorze). Stan komunikacyjny tego odwzorowania $C(\alpha_0)$ jest pokazany na rys. 3b jako punkt z numerem 13,
- Z zapisu zdarzeń komunikacyjnych (patrz rys. 4) wynika, że zbiór krawędzi grafu aplikacji wynosi $C = \{c_{s_1 r}, c_{s_2 r}\}$,
- Zbiór B zawiera jedynie początkowe odwzorowanie α_0,
- Nowe odwzorowanie (z maksymalną odległością w stosunku do α_0) jest obliczone jako $\alpha_1 = \{s_1 \to P_1, r \to P_2, s_2 \to P_1\}$, dla którego $dist(\alpha_1, B) = 4{,}24$. Stan komunikacyjny $C(\alpha_1)$ posiada numer 4 (patrz rys. 3b),
- Aplikacja jest wykonana po wybraniu odwzorowania α_1,
- Porównanie uzyskanych zapisów wykonania aplikacji nie przynosi informacji o zaistnieniu wyścigu,
- Zbiór B zawiera teraz α_0 i α_1,
- Obliczane jest kolejne odwzorowanie: $\alpha_2 = \{s_1 \to P_1, r \to P_1, s_2 \to P_2\}$ $dist(\alpha_2, B) = 3$. Odwzorowanie jest przedstawione na rys. 1a. Numer stanu $C(\alpha_2)$ wynosi 16,
- Aplikacja jest wykonana po wybraniu odwzorowania α_2,
- Z porównania dwóch ostatnich zapisów zachowania się aplikacji wynika, że w module r wystąpił wyścig pomiędzy wiadomościami m_1 i m_2 (patrz rys. 4). Cykl testowania kończy się więc po wykryciu pierwszego wyścigu występującego w aplikacji.

5. Analiza testowalności

Poniżej przedstawiono analizę możliwości wykrycia wyścigów przy wykorzystaniu zaproponowanych algorytmów odwzorowania.

5.1. Odwzorowanie dwuprocesorowe

Testowalność odwzorowania dwuprocesorowego zależy od opóźnień komunikacyjnych wprowadzonych przez system komputerowy opisany macierzą D oraz od struktury aplikacji.

Zgodnie z rys. 2 wyścig ujawnia się w momencie, gdy spełnione są następujące relacje czasowe:

$$\Delta T_{2-1} > d_{12} - d_{11} \text{ i } \Delta T'_{2-1} < d_{12} - d_{11} \tag{8}$$

Z równania (8) wynika, że testowalność algorytmu może być większa, gdy różnica $d_{12} - d_{11}$ jest możliwie największa (czyli $d_{11} << d_{12}$). Wadą algorytmu jest to, że w praktyce wartości ΔT_{2-1} i $\Delta T'_{2-1}$ często zależą od wartości d_{11} i d_{12}, co oznacza, że równanie (8) może nigdy nie być spełnione, niezależnie od różnicy $d_{12} - d_{11}$. Taka sytuacja wynika z braku informacji o „historii" ścigających się wiadomości. Ten problem jest rozwiązany dopiero przez odwzorowanie wieloprocesorowe.

5.2. Odwzorowanie wieloprocesorowe

Zgodnie z przyjętym modelem operacja odbioru wiadomości `recv()` jest blokująca. Przez t_i^r oznaczmy czas wywołania przez moduł operacji `recv()`, a przez t_i^a czas, gdy operacja `recv()` przekazuje do modułu odebraną wiadomość m_i. W takiej sytuacji możemy rozróżnić operacje `recv()` ze względu na jej zachowanie:

- natychmiastowa operacja `recv()` ma miejsce, gdy $t_i^d < t_i^r = t_i^a$,
- czekająca operacja `recv()` ma miejsce, gdy $t_i^r < t_i^d = t_i^a$.

Poprzednikiem wiadomości m_a wysłanej z i-tego modułu do j-tego modułu nazywamy ostatnią wiadomość (m_b) odebraną przez i przed wysłaniem m_a (o ile taka wiadomość istnieje). Pomiędzy odebraniem m_b a wysłaniem m_a w module i mają miejsce obliczenia, których czas trwania oznaczymy przez T (w przypadku gdy m_b nie istnieje, przyjmujemy, że T oznacza czas obliczeń od początku działania modułu do momentu wysłania m_a). Ponieważ w aplikacji występują tylko czekające operacje `recv()`, czas odbioru wiadomości m_a możemy zapisać jako $t_a^d = t_b^d + T + c_{ij}$. Analogicznie można wyrazić t_b^d poprzez czas odbioru wiadomości poprzedzającej m_b. Rozszerzając to rozumowanie na wszystkie poprzedniki wiadomości m_a, możemy zauważyć, że w ogólności czas odbioru wiadomości m_a opisuje się wyrażeniem:

$$t_a^d = \sum_{c_{ij} \in C} n_{ij}^a c_{ij} + T^a \tag{9}$$

gdzie n_{ij}^a to ilość wiadomości przesyłanych pomiędzy modułami i oraz j, T^a to całkowity czas obliczeń zawartych pomiędzy poprzednikami wiadomości m_a. Dla przykładu

przedstawionego na rys. 4 otrzymujemy: $t_0^d = c_{s_1 r} + c_{s_1 r} + (E_1 + E_2 + E_6)$, $t_2^d = 3c_{s_1 r} + (E_1 + E_2 + E_3)$

Wyścig ujawnia się w momencie, gdy dwa wykonania tej samej aplikacji zawierają odmienną kolejność odbioru wiadomości m_1 i m_2 przez jeden moduł, czyli:

$$t_1^d < t_2^d \text{ (pierwszy scenariusz wykonania - z odwzorowaniem } \alpha \text{)} \tag{10}$$

$$t_1^{d'} > t_2^{d'} \text{ (drugi scenariusz wykonania - z odwzorowaniem } \beta \text{)} \tag{11}$$

Zestawiając wzory (4),(5) i (6) otrzymujemy ostateczną postać równania, które warunkuje zaobserwowanie wyścigu pomiędzy m_1 i m_2:

$$\sum_{c_{ij} \in C} \left(n_{ij}^1 - n_{ij}^2\right) c_{ij} + \left(T^1 - T^2\right) < 0 \text{ (pierwszy scenariusz wykonania)} \tag{12}$$

$$\sum_{c_{ij} \in C} \left(n_{ij}^1 - n_{ij}^2\right) c_{ij}' + \left(T^1 - T^2\right) > 0 \text{ (drugi scenariusz wykonania)} \tag{13}$$

gdzie $c_{ij} = d_{\alpha(i)\alpha(j)}$, $c_{ij}' = d_{\beta(i)\beta(j)}$. Jak widać, równania (12) i (13) są liniowe względem zmiennych c_{ij} i c_{ij}'. Oznacza to, że dziedzina stanów komunikacyjnych, dla których otrzymujemy pierwszy scenariusz wykonania, jest oddzielona od stanów, dla których otrzymujemy drugi scenariusz wykonania przez określoną hiperpłaszczyznę. Zatem najszybciej można znaleźć dwa różne scenariusze wykonania, badając „narożniki" dziedziny stanów komunikacyjnych. Właśnie tę własność wykorzystuje odwzorowanie wieloprocesorowe. Dla rozpatrywanej aplikacji równania (12),(13) mają postać

$$3c_{s_1 r} - c_{s_2 r} - 1 < 0 \text{ i } 3c_{s_1 r} - c_{s_2 r} - 1 > 0 \tag{14}$$

Na rys. 3b ilustruje je linia przerywana. Położenie tej linii wyjaśnia, dlaczego w przykładowym testowaniu wystarczyły 3 odwzorowania do wykrycia wyścigu.

6. Wnioski końcowe

Proponowane algorytmy odwzorowania zaimplementowano w środowisku PVM ([2]). Ich skuteczność przedstawiono na przykładzie prostej aplikacji składającą się z 10 modułów, zawierającej 5 wyścigów. Testowanie odbywało się w heterogenicznej sieci LAN składającej się z różnych stacji roboczych Sun. Macierz D, opisująca system została wyznaczona za pomocą prostego programu monitorującego szybkość przesyłania komunikatów PVM pomiędzy różnymi stacjami roboczymi. Jako kryterium oceny przyjęto kryteria (4) i (5) zdefiniowane w rozdziale 1, przy czym w przypadku drugiego z nich przyjęto stałą liczbę cykli

testowania równą 10. Eksperyment został wielokrotnie powtórzony. W tabeli 1 przestawiono wyniki uśrednione dla architektury złożonej z różnej liczby stacji roboczych.

Tabela 1

Parametry e oraz c oszacowane przy testowaniu aplikacji PVM

Algorytm	3 st. robocze		5 st. roboczych		7 st. roboczych	
	e	c	e	c	e	c
Odw. dwuprocesorowe	11,76	72%	6,45	62%	19,23	72%
Odw. wieloprocesorowe	3,68	64%	2,4	62%	8,52	56%

Z tabeli 1 wynika, że odwzorowanie wieloprocesorowe zapewnia lepszy współczynnik wydajności e. Jednak w przypadku parametru pokrycia c odwzorowanie dwuprocesorowe zachowało się nieco lepiej. Oznacza to, że algorytm OD radzi sobie lepiej z większą ilością wyścigów, natomiast algorytm OW może być wykorzystany do detekcji pierwszego wyścigu i po jego skorygowaniu z powodzeniem może szukać dalszych wyścigów w testowanej aplikacji.

Uzyskane rezultaty mogą służyć jako wyjściowy materiał do opracowania bardziej wydajnej metody testowania, która nie musi być obarczona charakterystyką systemu, na którym aplikacja jest testowana. W tym celu należy rozważyć możliwość symulacji systemu rozproszonego o dowolnie dobranej macierzy D. Można to zaimplementować w formie opóźnień komunikacyjnych wprowadzonych do aplikacji. Takie podejście również może korzystać z zapisów zdarzeń komunikacyjnych w celu wyznaczenia optymalnych w sensie testowalności wag c_{ij}.

LITERATURA

[1] Bokhari S.A.: Dual processor scheduling with dynamic reassignment. IEEE Transactions on Software Engineering, 1979, t. SE-5, s. 341-349.

[2] Geist A., Beguelin J.,Dongarra J., Jiang W., Manchek R., Sunderam V.: Parallel Virtual Machine, PVM 3 Users Guide and Reference Manual. Oak Ridge National Laboratory, 1994.

[3] Hayes J.P.: A Graph Model for Fault-Tolerant Computing Systems. IEEE Transactions on Computers, 1976, t. C-25, nr 9, s. 875-883

[4] Krawczyk H., Skakowski A.: Segmentation Algorithm for Programs Assignment to Pipeline Systems. Proceedings of the 21st EUROMICRO Conference on Design of Hardware/ Software Systems, IEEE Computer Society Press, 1995, s. 169-176.

[5] Lamport L.: Time, Clocks and the Ordering of Events in a Distributed System. Communications of ACM, 1978, 21, 7, s. 558-565.

[6] Nicol D.M.,O'Hallaron D.R.: Improved Algorithms for Mapping Pipelined and Parallel Computations, IEEE Transactions on Computers, 1991,t. 40, nr 3, s. 295-306.

[7] Stone H.S.: Multiprocessor scheduling with the aid of network flow algorithms. IEEE Transactions on Software Engineering, 1978, t. SE-4, nr 3, s. 254-258.

[8] Shatz S.M., Wang J.-P.: Models and Algorithms for Reliability-Oriented Task-Allocation in Redundant Distributed-Computer Systems. 1989, t. 38, nr 1, s. 16-26

Recenzent: Prof. dr hab. inż. Andrzej Grzywak

Wpłynęło do Redakcji 15 grudnia 1997 r.

Abstract

The paper describes a criteria for assigning modules of distributed application on a given computer system (LAN network with PVM). We present graph models of program and system (see Fig. 1) and emphasize new requirements referring to improving testability of distributed applications. In other words, we do not want to minimize execution time of the application, but we try to find suitable time conditions stimulating occurrence of some time-dependent errors (races or deadlocks) (see Fig. 2).

We propose two heuristic assignment strategies: OD (two-processor assignment) and OW (multiple-processor assignment) which base on concept of distance for a given set of assignments (see Formulas no. 6,7) used for testing and debugging of distributed application. Moreover testability of the proposed strategies are analyzed and some general race occurrence conditions are formulated (see Formulas no. 8,12,13). Besides, some experiments were done and their results given in table 1 show advantages and disadvantages of the proposed assignment strategies.