ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1996
Seria: INFORMATYKA z. 30 Nr kol. 1315

Henryk KRAWCZYK
Marcin NEYMAN

PODSTAWOWE MECHANIZMY PONAWIANIA OBLICZEŃ W SYSTEMACH ROZPROSZONYCH*

Streszczenie. Zaproponowano dwa programowe mechanizmy dla zapewnienia wiarygodnych obliczeń w systemach rozproszonych. Pierwszy z mechanizmów (rerun), wykorzystujący procedury odtwarzania stanu procesów, opiera się na powtórnym wykonaniu fragmentu obliczeń programu. Drugi z nich (replay) jedynie symuluje wykonane poprzednio obliczenia korzystając z logu rejestującego zdarzenia podczas pierwotnego wykonania. Mechanizmy te mogą być wykorzystane w środowisku PVM.

RERUN AND REPLAY AS MAIN RECOMPUTATION MECHANISMS IN DISTRIBUTED SYSTEMS

Summary. Two software recovery mechanisms for dependable computations in distributed sytems were proposed. One of the mechanisms (rerun) uses process state recovery procedures to rerun a fragment of a program computations. The other one (replay) only simulates computations recovering events recoded during their primary execution. The mechanisms can be utilized in PVM environment.

GRUNDMETHODEN DER WIEDERDURCHFÜHRUNG VON BERECHNUNGEN IN VERTEILTEN SYSTEMEN

Zusammenfassung. In der Arbeit werden zwei Methoden zur Rückgewinnung von Berechnungen in verteilten Systemen vorgeschlagen. Die erste Methode, die

* Praca finansowana przez grant KBN nr 8 S503 017 07

Prozeduren der Wiedererzeugung von Pozeßzustand verwendet, ist gründet auf der wiederdurchführung des Programmfragmentes. Die zweite Methode simuliert lediglich früher durchgeführte Berechnungen, die im Vezeichnis von Ereignisses registriert wurden. Die Methoden können in PVM Environment verwendet werden.

1. Wprowadzenie

Z wielu względów pewne obliczenia w systemach rozproszonych powinny być ponawiane. Wymaga to na ogół odtwarzania pierwotnego stanu systemu, jaki istniał przed ich realizacją i następnie ponowne wykonanie tych, bądź zmodyfikowanych obliczeń. W systemach sekwencyjnych, gdzie poszczególne operacje wykonuje się w z góry określonym porządku, ponowienie obliczeń jest sprawą prostą [4]. Często, zwłaszcza na poziomie wykonujących się instrukcji, do tego celu wykorzystuje się odpowiednie funkcje standardowe. W przypadku stystemów równoległych bądź rozproszonych procedury ponawiania obliczeń są znacznie trudniejsze [6]. Po pierwsze, dotyczą one wszystkich węzłów realizujących jednocześnie wiele różnych obliczeń. Po drugie, trudno często ustalić kolejność zachodzenia pewnych operacji, np. zajścia zdarzeń komunikacyjnych między grupą kooperujących procesów [3].

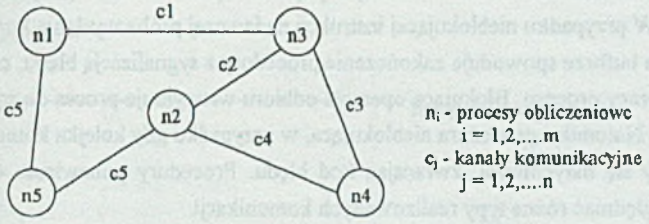
W pracy przedstawiono model systemu rozproszonego na poziomie kooperujących procesów i na podstawie tego modelu zdefiniowano stan systemu rozproszonego oraz podstawowe strategie odtwarzania stanu i obliczeń. Przyjmując za wzorcowe środowisko PVM zaimplementowano dwie funkcje ponawiania obliczeń. Pierwsza z nich (rzeczywiste ponowienie - rerun) zawiera w sobie odtwarzanie poprzedniego stanu systemu i ponowne wykonanie obliczeń od tego stanu. Druga zaś (symulowane ponowienie - replay) polega na symulowanym wykonaniu fragmentu obliczeń, który został zapamiętany w logu podczas pierwotnego wykonania.

Funkcje te są wykorzystywane przy konstrukcji testera programów równoległych, realizowanego w ramach programu Copernicus SEPP - No CP 93:5383 - Software Engineering for Parallel Processing.

2. Model systemu rozproszonego

W pracy przyjęto prosty model systemu rozproszonego. Zakłada się, że w systemie wydzielone są niezależne zadania (programy), składające się ze skończonego zbioru procesów i skoń-

czonego zbioru kanałów komunikacyjnych, łączących procesy. Taki program można przedstawić za pomocą grafu, którego węzły reprezentują procesy, a krawędzie kanały komunikacyjne (patrz rys. 1). W tym modelu abstrahuje się od fizycznego rozmieszczenia procesów na stacjach roboczych. Mogą one pracować na pojedynczej stacji bądź mogą być ulokowane na wielu maszynach połączonych siecią komunikacyjną. Odwzorowanie grafu zadania (programu) na sprzętową architekturę systemu nie jest przedmiotem niniejszej pracy. Zakłada się więc, że procesy nie dysponują obszarami wspólnej pamięci. Przykładem środowiska, odpowiadającego takiemu modelowi, może być PVM (Parallel Virtual Machine) i ono będzie stanowić bazę przeprowdzanych rozważań [2].



Rys. 1. Przykładowy graf programu w systemie rozproszonym
Fig. 1. Example of a program graph in a distributed system

PVM jest środowiskiem organizującym pracę zbioru heterogenicznych komputerów, pracujących pod kontrolą różnych odmian sytemu Unix i połączonych siecią komunikacyjną, jako dużego wieloprocesorowego komputera zwanego *maszyną wirtualną*. Oferuje zbiór funkcji pozwalających na konfigurację maszyny wirtualnej (dodawanie i usuwanie jej składników), na kreowanie nowych procesów i rozmieszczanie ich na wybranych elementach maszyny wirtualnej oraz na efektywną komunikację międzyprocesową opartą na schemacie przesyłania komunikatów. Nie posiada jednak mechanizmów ponawiania realizowanych obliczeń.

Model obliczeń rozproszonych zakłada wykonanie się wielu procesów na różnych węzłach sieci współpracujących ze sobą poprzez wymianę informacji. Przekazywanie informacji odbywa się przy wykorzystaniu mechanizmów komunikacyjnych. Akcje komunikujących się procesów muszą być częściowo uporządkowane w czasie, ponieważ informacja musi najpierw zostać uworzona (wysłana), nim zostanie użyta (odebrana). Takie porządkowanie w czasie akcji różnych procesów nazywa się synchronizacją. Jest ona konieczna również przy współzawodnictwie procesów o zasoby. Jeżeli wymagany zasób jest zajęty w danej chwili, to proces potrzebujący tego zasobu powinien być wstrzymany do chwili jego uzyskania. Jeżeli założymy, że sygnał synchronizujący jest pustym komunikatem, to zbiór wszystkich mechanizmów synchronizacyjnych można uznać za podklasę mechanizmów komunikayjnych.

Wśród mechanizmów komunikacji i synchronizacji [3] wyróżniamy mechanizmy typu "one to one" (jeden nadawca, jeden odbiorca), "many to one" (wielu nadawców, jeden odbiorca) oraz "one to many" (typowy broadcasting) i "many to many" (jako kompozycję poprzednich przypadków). Poza tym wymiana informacji może być synchroniczna albo asynchroniczna, a mechanizmy komunikacyjne blokujące i nieblokujące. Komunikacja **synchroniczna** ma miejsce, gdy nadawca czeka, aż adresat odbierze wysłany komunikat. Dzieje się tak w przypadku komunikacji niebuforowanej (kolejka komunikatów o zerowej długości). W komunikacji **asynchronicznej** (buforowanej) instrukcja nadawcza może być blokująca i wtedy proces jest wstrzymywany przy próbie zapisu do pełnego bufora. W takim przypadku proces nadający musi zaczekać aż odbiorca odczyta poprzedni pakiet i zwolni miejsce w kolejce komunikatów. W przypadku nieblokującej instrukcji nadawczej próba wysłania komunikatu przy zapełnionym buforze spowoduje zakończenie procedury z sygnalizacją błędu, co umożliwia kontynuację pracy procesu. Blokująca operacja odbioru wstrzymuje proces do czasu nadejścia komunikatu. Natomiast procedura nieblokująca, w przypadku gdy kolejka komunikatów jest pusta, kończy się natychmiast, zwracając kod błędu. Procedury ponawiania obliczeń powinny więc uwzględniać różne typy realizowanych komunikacji.

3. Odtwarzanie stanu a ponawianie obliczeń

Kluczowym problemem jest definicja stanu systemu rozproszonego. Stanowi ona iloczyn kartezjański stanów procesów realizujących daną aplikację. Stan pojedynczego procesu można określić przez podanie aktualnej zawartości jego obszaru adresowego i rejestrów sprzętowych oraz struktur danych jądra systemu z nim powiązanych. Można również ustalić go przez wyspecyfikowanie stanu inicjalnego oraz ciągu zdarzeń, ktore zaszły w czasie wykonania procesu. Zdarzenie e w procesie p jest niepodzielną akcją, która może zmienić stan samego procesu i co najwyżej jednego kanału komunikacyjnego c związanego z nim. Stan kanału może zostać zmieniony poprzez wysłanie albo odebranie komunikatu. Zatem zdarzenie e jest zdefiniowane przez:

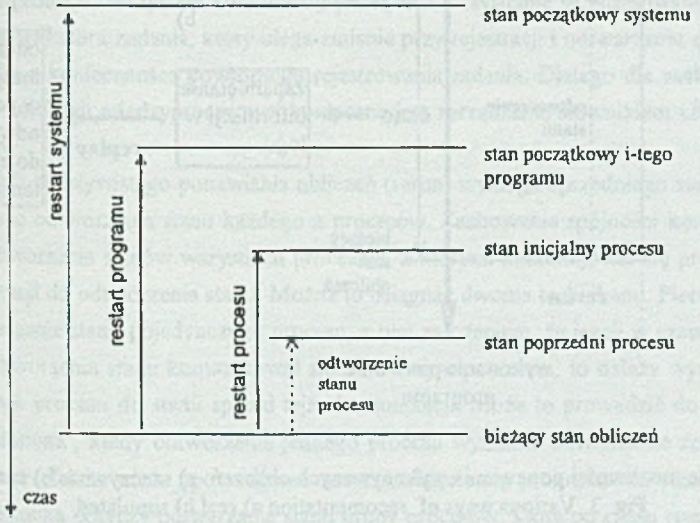
- proces p, w którym zdarzenie zachodzi,
- stan s procesu p bezpośrednio przed zajściem zdarzenia,
- stan s' procesu p natychmiast po zdarzeniu,
- kanał c, którego stan został zmieniony w wyniku zdarzenia,
- komunikat M wysłany albo odebrany z kanału c.

Zdarzenie e można zdefiniować jako 5-elementowy wektor $<p,s,s',M,c>$, gdzie M i c przyjmują wartość *null*, jeśli zdarzenie nie jest związane z wysłaniem albo odebraniem komunikatu

W konsekwencji stan programu jest sumą stanów wszystkich procesów wchodzących w jego skład i stanów łączących je kanałów komukacyjnych. Można przedstawić go jako sumę wektorów: $< s_{p1}, s_{p2}, ..., s_{pj} > \cup < s_{c1}, s_{c2}, ..., s_{ck} >$ [7]. W pracy będziemy utożsamiać stan systemu ze stanem wykonywanego przez niego programu rozproszonego.

Odtwarzanie stanu można określić jako powrót systemu, programu lub procesu do pewnej wyspecyfikowanej sytuacji, która wystąpiła w czasie poprzedzającym odtworzenie. Wymaga to zwykle uprzedniej rejestracji stanu, przy użyciu techniki pozwalającej tenże stan przywrócić. Odtwarzanie stanu można rozpatrywać na różnych poziomach:

- całego systemu,
- wybranego programu,
- pojedynczego procesu.

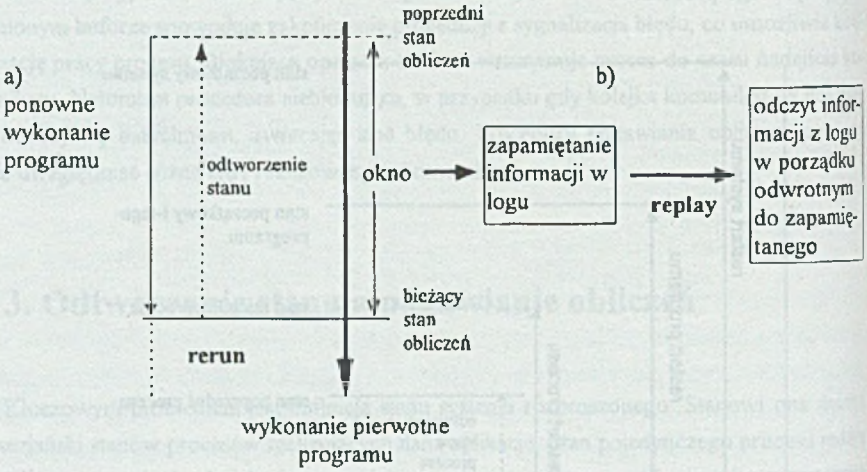


Rys. 2. Różne poziomy odtwarzania stanu
Fig. 2. Different levels of state recovery

Najprostszą techniką odtwarzania (nie wymagającą rejestracji stanu) jest restart, czyli powrót do stanu inicjalnego. Restart całego systemu zwykle nie przedstawia większych problemów, również restart programu nie niesie ze sobą dużych trudności, przy założeniu o niezależności poszczególnych programów. Jednak już restart pojedynczego procesu wiąże się z poważnymi problemami wynikającymi z zależności komunikacyjnych między procesami. O ile

restart jest najprostszą techniką odtwarzania stanu, o tyle jej użycie jest ograniczone do niewielu klas zastosowań. Hierarchię poziomów odtwarzania stanu ilustruje rys. 2.

Ponowienie obliczeń polega na powtórnym wykonaniu tego samego fragmentu programu, pojedynczego procesu lub grupy procesów (patrz rys. 3). W pierwszym przypadku wymaga zarejestrowania stanu, a następnie odtworzenia go w celu umożliwienia powtórnego wykonania części obliczeń. Drugi sposób, nie wymagający rzeczywistego wykonania, wymaga zapamiętania zdarzeń modyfikujących stan procesu. Ta metoda pozwala na ustalenie szczegółów odtworzenia poprzez wybór klas rejestrowanych zdarzeń. Wspomniane wyżej funkcje ponawiania obliczeń nazywane są odpowiedio **rerun** i **replay**. Ich analizie oraz opisowi implementacji w środowisku PVM poświęcony jest rozdział 4.



Rys. 3. Różne możliwości ponawiania wykonywanych obliczeń: a) rzeczywiste b) symulowane
Fig. 3. Various ways of recomputation a) real b) simulated

4. Możliwości ponawiania obliczeń w środowisku PVM

Metodą zarejestrowania stanu procesu w systemie Unix, pozwalającą na późniejsze jego odtworzenie [6][8], jest utworzenie kopii procesu poprzez funkcję systemową *fork*, a następnie, po zdefiniowaniu obsługi odpowiednich sygnałów funkcją *signal*, natychmiastowe jego uśpienie funkcją *pause*. Konieczna jest zatem zmiana standardowej obsługi dwóch sygnałów. Pierwszy jest wysyłany przez jądro systemu operacyjnego, które okresowo budzi uśpione pro-

cesy. Procedura obsługi tego sygnału musi zapewnić natychmiastowe uśpienie procesu obudzonego w niewłaściwym momencie. Drugi z sygnałów wysyłany jest przez funkcję odtwarzania stanu. Zmiana jego standardowej obsługi ma na celu zapobieżenie zabiciu procesu po odebraniu sygnału. Odtworzenie tak zapamiętanego stanu polega na wysłaniu funkcją *kill* odpowiedniego sygnału do uśpionego procesu. Proces wołający procedurę odtwarzania stanu kończy życie, wykonując funkcję *exit*.

Z konstrukcji środowiska PVM wynika, że utworzenie funkcją *fork* kopii procesu, będącego zadaniem należącym do maszyny wirtualnej, doprowadzi do niespójności. Dlatego rejestracja stanu procesu poprzez utworzenie jego kopii musi być poprzedzona wyrejestrowaniem zadania z maszyny wirtualnej. Po wykonaniu funkcji *fork* proces może ponownie zarejestrować się w maszynie wirtualnej. Również proces, który zostanie obudzony, musi zarejestrować się w maszynie wirtualnej, zanim przejdzie do realizacji swoich obliczeń. Z powyższym wiąże się poważny problem. Wszystkie komunikaty przesyłane w systemie są adresowane poprzez podanie identyfikatora zadania, który ulega zmianie przy rejestracji i odtwarzaniu stanu procesu, w związku z koniecznością powtórnego rejestrowania zadania. Dlatego dla zachowania spójności komunikacji miedzyprocesowej konieczne jest zarządzanie słownikiem identyfikatorów procesów.

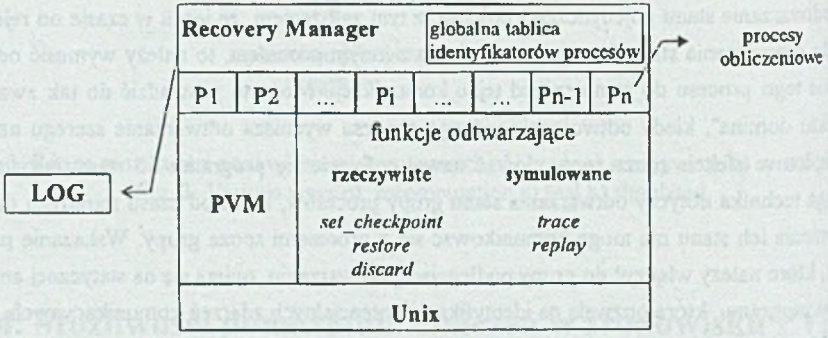
Metoda rzeczywistego ponawiania obliczeń (rerun) wymaga uprzedniego zarejestrowania, a następnie odtworzenia stanu każdego z procesów. Zachowanie spójności komunikacji wymaga odtworzenia stanów wszystkich procesów, z którymi komunikował się proces w czasie od rejestracji do odtworzenia stanu. Można to osiągnąć dwoma technikami. Pierwsza pozwala na odtwarzanie stanu pojedynczego procesu, z tym założeniem, że jeżeli w czasie od rejestracji do odtworzenia stanu komunikował się on z innym procesem, to należy wymusić odtworzenie tego procesu do stanu sprzed tejże komunikacji. Może to prowadzić do tak zwanego "efektu domina", kiedy odtworzenie jednego procesu wymusza odtwarzanie szeregu następnych, co w efekcie może spowodować nawet cofnięcie się programu do stanu inicjalnego. Druga technika dotyczy odtwarzania stanu grupy procesów, które od czasu rejestracji do odtworzenia ich stanu nie mogą komunikować się z procesami spoza grupy. Wskazanie procesów, które należy włączyć do grupy podlegającej odtwarzaniu, opiera się na statycznej analizie kodu programu, która pozwala na identyfikację potencjalnych zdarzeń komunikacyjnych.

Metoda symulowanego ponawiania obliczeń (replay), w odróżnieniu od rerun, polega na odtworzeniu obliczeń bez faktycznego wykonania programu. W tym celu zapisuje się w specjalnym pliku (logu) informacje na temat zdarzeń modyfikujących stan procesu. Jest to odmiana techniki znanej jako debugowanie pośmiertne (post-mortem debugging). Zaleta tej metody polega na tym, że nie wymaga się tu żadnych ingerencji w działanie programu oraz nie nakłada się ograniczeń na komunikację jak w metodzie rzeczywistego ponawiania. Wada tej metody polega na tym, że pozwala ona jedynie na powtórzenie sekwencji obliczeń bez możliwości

wybrania ścieżki alternatywnej. Kluczowym zagadnieniem w tej metodzie odtwarzania jest rejestracja zdarzeń. Wybór klas rejestrowanych zdarzeń pozwala na ustalenie poziomu szczegółowości obserwacji. W najprostszym przypadku można rejestrować jedynie zdarzenia komunikacyjne. Dokładne odtworzenie obliczeń wymaga jednak rejestracji instrukcji sekwencyjnych, a szczególnie instrukcji warunkowych, co pozwala na prześledzenie wykonanej ścieżki.

Opisane wyżej mechanizmy odtwarzające zostały zaimplementowane w module o nazwie TESEM (Test Scenario Execution Manager), będącym częścią narzędzia do systematycznego testowania programów rozproszonych. Schemat tego modułu przedstawia rys. 4. Funkcje ponawiania obliczeń mogą być realizowane w z góry określonym fragmencie programu, zwanym oknem wykonania programu.

Analiza wykonania procesów w oknie wspomagana jest funkcjami odtwarzenia obliczeń. Wszystkie procesy osiągając górną krawędź okna dokonują rejestracji ich stanu, wykorzystując opisaną wyżej metodę. W celu zachowania spójności systemu, po zarejestrowaniu stanu, procesy są wstrzymywane do chwili, gdy ostatni z nich dokona tej czynności. Na określanie rozmiarów okna (wybór procesów do niego należących i wskazanie fragmentów tych procesów) nakłada się dwa ograniczenia. Zgodnie z pierwszym procesy wewnątrz okna nie mogą komunikować się z tymi spoza niego. Drugie wymaga, aby stan procesu rejestrowany był, gdy kolejka komunikatów oczekujących na odbiór jest pusta. Odtworzenie stanu może być zrealizowane w momencie, gdy procesy osiągną dolną krawędź okna testowego.



Rys. 4. Struktura modułu ponawiania obliczeń
Fig. 4.Recovery module structure

Funkcje ponawiania obliczeń są włączane do testowanego programu poprzez instrumentację kodu źródłowego na poziomie instrukcji PVM. Zarządzanie mechanizmami odtwarzający-

mi oraz słownikiem synonimów identyfikatorów procesów jest realizowane przez proces o nazwie Recovery Manager.

Oprócz funkcji pozwalających na rzeczywiste wykonanie procesów w oknie moduł TESEM udostępnia również mechanizmy symulowanego ponawiania obliczeń. Funkcja ta pozwala na odtwarzanie w dowolnej chwili dowolnie wybranego fragmentu dowolnych procesów. Uwaga koncentruje się tu głównie na analizie zdarzeń komunikacyjnych, które rejestrowane są w logu na dysku. Funkcje odtwarzające sprowadzają się w tym przypadku do analizy i interpretacji danych zgromadzonych w logu.

4. Uwagi końcowe

Implementacja funkcji rerun i replay w środowisku PVM wymagała rozwiązania następujących problemów:

- użycie funkcji fork w środowisku PVM;
- utożsamianie różnych identyfikatorów przydzielanych procesom w wyniku wykonywania funkcji rerun;
- zapewnienie zgodności rozproszonej informacji o identyfikatorach procesów.

Opracowane mechanizmy ponawiania obliczeń są obecnie intensywnie testowane. Koncentruje się przy tym na zapewnieniu efektywności ich działania, jak i na ich wykorzystaniu przy realizacji różnych scenariuszy testowych umożliwiających systematyczne sprawdzanie różnego typu programów równoległych.

LITERATURA

[1] Chandy K. M., Lamport L.,: Distributed Snapshots: Determining Global States of Distributed Systems. ACM Transactions on Computer Systems, 1985, t. 3, nr 1, s. 63-75.

[2] Geist A., Beguelin A., Dongarra J., Jiang W., Mancheck R., Sunderam V.: PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing. The MIT Press, Cambridge, Massachusetts, 1994.

[3] Krawczyk H.: Dependable Processing. Skrypt Politechniki Gdańskiej, Gdańsk 1995.

[4] Krawczyk H., Brudło P., Neyman M.: Modele komunikacji i synchronizacji dla obliczeń rozproszonych. Materiały konferencji POLMAN' 95, Poznań, 1985, ss. 172-184.

[5] Krawczyk H., Wiszniewski B.: Interactive Tesing Tool for Parallel Programs. Referat przyjęty na konferencję Software Enginering for Parallel and Distributed Systems, 1996.

[6] Neyman M., Krawczyk H.: Modelowanie wstecznej starategii odtwarzania poprawnego stanu procesów. Materiały konferencji "Informatyka na wyższych uczelniach dla gospodarki narodowej", 1994, t. 2, ss. 183-186.

[7] Spezialetti M., Kearns P.: Efficient Distributed Snapshots. Proceedings of the 6th International Conference on Distributed Computing Systems, 1986, ss. 382-388.

[8] Taylor D. J., Wright M. L.: Bakward Error Recovery in a Unix Environment. Prof. of FTCS-16, 1986.

Recenzent: Prof. dr hab. inż. Andrzej Grzywak

Wpłynęło do Redakcji 21 grudnia 1995 r.

Abstract

The paper presents some proposals of state recovery functions in order to achieve repetition of execution of a given fragment (window) of a distributed program. We assume that the program is reperesented as a graph, where vertices correspond to computation processes and edges reperesent communicatin channels between two processes (see fig. 1). For the model we define state recovery functions (see fig. 2) and couple of functions providing repetition of a portion of program computations (see fig. 3).

In the paper we have concentrated on the two functions: rerun and replay which can be used in PVM environment. To implement such functions we had to solve the following problems:

- utilisation of fork function in PVM
- equivalence of different identifiers assigned o processes during execution of rerun function
- integrity of distributed information on processes' identifiers

Actually the functions are heavily tested and being improved to achieve highier efficiency. We utilise them in the TESEM module in order to generate testing scenarios for parallel programs.