ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1999
Seria: INFORMATYKA z. 36 Nr kol. 1414

Krzysztof PIERZCHAŁA
Politechnika Śląska, Instytut Informatyki

MOŻLIWOŚĆ IMPLEMENTACJI MECHANIZMÓW TOLEROWANIA AWARII W ŚRODOWISKACH PRZETWARZANIA ROZPROSZONEGO

Streszczenie. W artykule zaprezentowano pożądane cechy środowisk przetwarzania rozproszonego ułatwiające implementację w nich mechanizmów tolerowania awarii. Przeanalizowano popularne środowiska przetwarzania rozproszonego pod kątem posiadania tych cech. Wybrano najlepsze środowisko, w którym można implementować mechanizmy tolerowania awarii.

POSIBILITY OF IMPLEMENTING FAULT-TOLLERANT MECHANISMS IN DISTRIBUTED COMPUTING SYSTEMS

Summary. Desirable features of distributed computational environments enabling the implementation of fault-tolerant mechanizms are presented in the paper. Popular distributed computational environments were analyzed from the point of view of having that features. The best environment giving the possibility of impementation of fault-tolerant mechanisms was chosen.

1. Wstęp

Systemy tolerujące awarie (*fault tolerant*) to zgodnie z definicją systemy mogące kontynuować swą pracę pomimo awarii niektórych elementów systemu. Jest to szczególnie ważne w systemach czasu rzeczywistego oraz w przypadku obliczeń rozproszonych. W systemach czasu rzeczywistego jest to oczywiste – systemy takie powiny działać bez przerw. Niejasna może się wydać konieczność stosowania metod pozwalających na tolerowanie awarii w przypadku systemów obliczeń rozproszonych. Wystarczy jednak zauważyć, że konieczność przeprowadzania obliczeń w sposób rozproszony pojawia się

głównie wtedy, gdy obliczenia są złożone i ich wykonanie na jednym komputerze mogłoby spowodować nieaktualność wyników wynikającą z czasu trwania obliczeń (np. w prognozie pogody). Mechanizmy tolerowania awarii w przetwarzaniu rozproszonym powinny być używane właśnie ze względu na konieczność uzyskania wyników w zadanym czasie. Umożliwiają one bowiem kontynuację obliczeń w przypadku awarii jednego (lub większej liczby) komputerów biorących udział w przetwarzaniu. Poza tym, w systemach pozbawionych mechanizmów tolerowania awarii awaria jednego komputera nawet przy 99% zaawansowaniu obliczeń powoduje konieczność obliczania wszystkiego od początku, co jest dużą stratą czasu oraz mocy obliczeniowej.

Obecnie większość systemów pozwalających na obliczenia rozproszone nie ma jawnych mechanizmów tolerowania awarii, część posiada jednak cechy powodujące, że takie mechanizmy jest łatwiej implementować. Oczywiście, istnieją też gotowe systemy pozwalające na przetwarzanie rozproszone z tolerowaniem awarii – są one jednak zamknięte (przeznaczone tylko do bardzo konkretnych obliczeń) oraz przeważnie nie dysponują gotowymi biblioteczkami rozproszonych obliczeń numerycznych (np. SCALAPACK).

Chcąc stosować w obliczeniach rozproszonych mechanizmy tolerowania awarii, najczęściej wybiera się jedną z poniższych ścieżek:

1. Przepisać całość programu oraz wszystkie biblioteczki z normalnego systemu obliczeń rozproszonych do systemu umożliwiającego tolerowanie awarii.
2. Próbować zaimplementować mechanizmy tolerowania awarii w swoim programie.

Istnieje jednak trzecia możliwość – można zmodyfikować środowisko rozpraszania obliczeń tak, aby zapewniało mechanizmy tolerowania awarii w sposób niezauważalny lub wymagało niewielkich zmian w kodzie już istniejących programów.

Modyfikacja środowiska rozpraszania obliczeń jest korzystna – użytkownicy nie muszą diametralnie zmieniać swoich przyzwyczajeń (wystarczy, że zapoznają się z kilkoma dodatkowymi funkcjami), programy już istniejące będą nadal działać (można będzie je spokojnie przerabiać, aby wprowadzić w nich ulepszenia wynikające z modyfikacji systemu rozpraszania obliczeń), można nadal wykorzystywać biblioteczki funkcji, do których nie posiadamy kodu źródłowego.

2. Cechy środowiska rozpraszania obliczeń, w którym można zaimplementować system tolerowania awarii

Środowisko rozpraszania obliczeń, aby można w nim było zaimplementować system tolerowania awarii, musi posiadać kilka cech. Podstawową cechą musi być możliwość kontynuowania obliczeń pomimo awarii jednego z elementów składowych środowiska

w makroskali (komputer, sieć połączeń) lub mikroskali (pojedyncze zadanie). Należy tu zwrócić uwagę, że nie chodzi w tym przypadku o automatyczne kończenie wszystkich zadań aktualnie wykonywanych w środowisku, tylko o pozostawienie zadań pracujących, z którymi istnieje łączność. W przypadku braku tej cechy środowisko podczas wzbogacania o funkcje tolerowania awarii wymaga przeróbki na niskim poziomie, co poddaje w wątpliwość sensowność takiej przeróbki. Zwykle prościej jest wówczas napisać środowisko od nowa.

Inną ważną cechą jest informowanie przez środowisko wszystkich pozostałych zadań w przypadku wykrycia awarii. Można tu także wyróżnić makroskalę (środowisko informuje o awarii komputerów) i mikroskalę (środowisko informuje o awarii zadań). Brak tej cechy można zrekompensować, wysyłając co pewien czas sygnały informujące o pracy zadań. Wiąże się to jednak ze stosunkowo dużym narzutem na zajętość medium komunikacyjnego oraz zajmuje pewien czas, który mógłby być wykorzystany na obliczenia. Innym problemem mogącym występować przy samodzielnym uzupełnianiu tej cechy jest konieczność działania asynchronicznego w stosunku do prowadzonych obliczeń. Asynchroniczność wynika z faktu, że takie sygnały powinny być wysyłane i sprawdzane nie rzadziej niż co pewien zadany czas, niezależnie od stopnia obciążenia komputerów i sieci połączeń.

Pozytywną cechą, jaką może posiadać środowisko, jest również możliwość dynamicznej zmiany konfiguracji. Umożliwia to wyłączanie z obliczeń komputerów przeciążonych lub uszkodzonych i włączanie do obliczeń komputerów rezerwowych w miejsce uszkodzonych. W przypadku braku możliwości dynamicznej zmiany konfiguracji zmuszeni jesteśmy włączyć w środowisko wszystkie potencjalnie dostępne komputery – może to spowodować zbędne obciążenie niektórych z nich oraz zaburzać pracę mechanizmów rozkładania obciążeń na komputerach.

Mile widzianą cechą jest też możliwość niezależnego obsługiwania zdarzeń. Pozwala to logicznie oddzielić obsługę zdarzeń związanych z awariami od reszty kodu programu. Można stworzyć biblioteczkę zawierającą funkcje obsługujące zdarzenia związane z awariami, które to funkcje wystarczy zainicjalizować na początku programu, a nie zmuszać twórcę oprogramowania do wywoływania ich co pewien czas (ściśle zadany – limity czasowe na uznanie zadania za nieodpowiadające).

Następną cechą, którą warto aby środowisko miało zaimplementowaną, są konteksty komunikacyjne. Pozwala to oddzielić informacje pochodzące z normalnych obliczeń od informacji, które muszą być przesyłane w celu zapewnienia odporności na awarie (np. zapis stanu). Poza tym istnienie kontekstów komunikacyjnych pozwala na zmniejszenie ograniczeń stawianych przed użytkownikiem środowiska – nie musi on unikać pewnych typów komunikatów lub krotek w swoich programach.

Inne cechy, które mogą być przydatne, są to uruchamianie zadania obliczeniowego na ściśle określonym komputerze z wnętrza programu uruchomionego w środowisku oraz

możliwość migrowania zadań między komputerami. Uruchamianie zadania na określonym komputerze pozwala uruchomić obliczenia utracone na uszkodzonym komputerze na innym (zapasowym) komputerze. Możliwość dokonania tego z wnętrza programu pozwala na dokonanie tego bez ingerencji operatora. Migrowanie zadań wydaje się cechą podstawową przy wzbogacaniu środowiska obliczeń rozproszonych w mechanizmy tolerowania awarii, niestety, systemy posiadające taką możliwość wymagają, aby zadanie mające przenieść się na inny komputer było sprawne – w przypadku awarii komputera wchodzącego w skład środowiska wszystkie programy są również utracone. Migrowanie zadań może być natomiast przydatne do wyrównywania obciążeń komputerów biorących udział w obliczeniach po awarii.

3. Przegląd popularnych systemów rozpraszania obliczeń

Najpopularnieszymi w tej chwili środowiskami rozpraszania wydają się być systemy PVM, MPI oraz Linda. PVM i MPI są darmowymi środowiskami z dostępną dużą liczbą bibliotek i narzędzi wspomagających obliczenia rozproszone (łącznie z profilerami i debugerami). Oba środowiska bazują na modelu obliczeń rozproszonych z przesyłem komunikatów. System Linda jest systemem komercyjnym bazującym na modelu obliczeń rozproszonych ze wspólną pamięcią ("przestrzeń krotek").

System PVM (Parallel Virtual Machine) występuje obecnie w dwu wersjach: 3.3 (stabilnej) i 3.4 (beta). Warto omówić obie wersje, gdyż pierwsza jest standardem, a druga zawiera dużo więcej możliwości implementacji mechanizmów tolerowania awarii.

System MPI (Message Passing Interface) występuje w dużej liczbie wersji (nie ma polityki utrzymywania jednego standardu). Obecnie wszystkie wersje wspierają standard MPI 1.1, niektóre zaczynają być wzbogacane o pojedyncze funkcje ze standardu MPI 2.0. Nie istnieje na razie wersja zgodna ze standardem MPI 2.0. Jako środowisko rozpraszania obliczeń polecana jest implementacja LAM (Local Area Multicomputer), która jest w pełni zgodna ze specyfikacją MPI 1.1 oraz zawiera pewne rozszerzenia zgodne ze standardem MPI 2.0.

Porównanie cech opisanych środowisk przetwarzania rozproszonego zawiera tabela 1.

4. Podsumowanie

Optymalnym środowiskiem do implementacji mechanizmów tolerowania awarii jest system PVM 3.4. Posiada on najwięcej cech ułatwiających implementację. Jest produktem darmowym dostarczanym z pełnymi źródłami – istnieje więc możliwość ingerencji w sam system. Wersja PVM 3.4 ma ciągle status programu rozwijanego (beta), istnieją więc możliwości włączenia

rozszerzeń do systemu. System MPI oraz jego implementacje są na razie nakierowane na przetwarzanie statyczne. Standard MPI 2.0 zapowiada możliwość dynamicznej zmiany konfiguracji środowiska. Na razie jest on standardem martwym – brak jakiejkolwiek implementacji. System Linda oraz jego rozszerzenie, system Paradise, posiadają małą liczbę cech, a ponadto jako rozwiązania komercyjne nie są udostępniane ich źródła.

Tabela 1

Porównanie cech popularnych środowisk przetwarzania rozproszonego

	PVM 3.3	PVM 3.4	MPI 1.1	LAM	Linda
Kontynuacja obliczeń	+	+	–	+	+
Informacja o awarii	+	+	–	+	–
Dynamiczna zmiana konfiguracji	+	+	–	+ (1)	–
Obsługa zdarzeń	–	+	–	–	–
Konteksty komunikacyjne	–	+	– (2)	– (2)	– (3)
Uruchamianie zadania na określonym komputerze	+	+	+ (4)	+ (4)	–
Migracja zadań	–	–	–	–	– (3)

\+ – system posiada cechę

\- – system nie posiada cechy

Uwagi:

1. Semistatyczna – wymaga działania operatora za pomocą programów lamgrow i lamshrink.
2. Można stworzyć komunikatory służące tylko w tym celu.
3. Dostępne w systemie Paradise – wzbogaconej wersji systemu Linda.
4. Statyczne – deklarowane w momencie uruchamiania obliczeń.

LITERATURA

1. Kozielski S. i inni: Systemy umożliwiające realizację algorytmów równoległych w sieciach komputerowych. Skrypt Uczelniany Pol. Śl. nr 1975, Gliwice 1996.
2. Zieliński K. i inni: Środowiska programowania rozproszonego w sieciach komputerowych. Księgarnia Akademicka, Kraków 1994.
3. Foster I.: Designing and building parallel programs. Publikacja elektroniczna, 1995.
4. Praca zbiorowa: MPI: A message passing interface standard v1.1. Publikacja elektroniczna.
5. Praca zbiorowa: MPI: A message passing interface standard v2.0. Publikacja elektroniczna.
6. Dokumentacja elektroniczna pakietu PVM 3.3.
7. Dokumentacja elektroniczna pakietu PVM 3.4.
8. Dokumentacja elektroniczna pakietu LAM 6.1.
9. Dokumentacja elektroniczna systemu Linda.
10. Dokumentacja elektroniczna systemu Paradise.

Recenzent: Doc. dr hab. inż. Tadeusz Czachórski

Wpłynęło do Redakcji 31 marca 1999 r.

Abstract

Fault-tollerance as an aim of distributed computing (espetially in long term computation) was presented in the paper. Desirable features for distributed computational environments enabling fault-tolerant computing implementation were analyzed. Most popular distributed system environments comparision was presented (table 1). The best environment giving the possibility of impementation fault-tolerant mechanisms was chosen.