STUDIA INFORMATICA 2003
Volume 24 Number 3 (55)

Maciej KOCOŃ
Politechnika Śląska, Instytut Informatyki

ŚRODOWISKO PRZETWARZANIA ROZPROSZONEGO W SIECI INTERNET

Streszczenie. Dzięki wykorzystaniu wielu komputerów połączonych ze sobą w sieć możliwe jest uzyskanie potencjalnej mocy obliczeniowej wielokrotnie przekraczającej możliwości pojedynczych stacji roboczych. Niestety stworzenie aplikacji przeprowadzającej takie obliczenia jest zadaniem trudnym i pracochłonnym. Zaprezentowane w niniejszym artykule oprogramowanie zostało stworzone w ramach realizacji pracy dyplomowej na Politechnice Śląskiej. Celem było opracowanie i dostarczenie narzędzia ułatwiającego programiście tworzenie oprogramowania realizującego wybrany problem w architekturze rozproszonej.

Słowa kluczowe: przetwarzanie rozproszone, klient – serwer, COM.

DISTRIBUTED COMPUTING ENVIRONMENT THROUGH INTERNET USE

Summary. By means of use computers connected with each other in net it is possible to obtain computational power repeatedly exceeding abilities of single workstations. Unfortunately the creation of application effecting such processing is hard and laborious task. Software presented in this article has been made within the scope of M.A thesis execution in Silesian Technical University. The goal of this research was to work out and deliver a tool simplifying programmer to create the software executing some problem in distributed architecture.

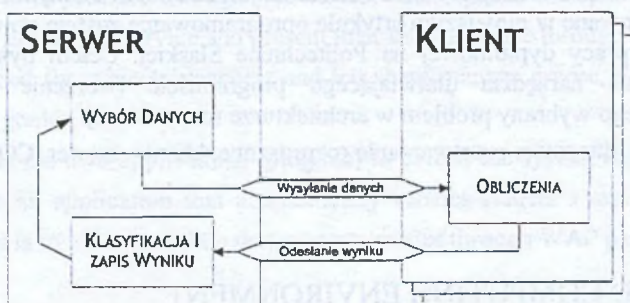
Keywords: distributed computing, klient – server, COM.

1. Wstęp

Od momentu powstania komputera bezustannie prowadzone są prace nad zwiększeniem jego wydajności tak, aby mógł sprostać stale rosnącym wymaganiom. Ale jakie wyjście pozostaje, kiedy prawo Moore'a, dość trafnie jak do tej pory opisujące dokonujący się

technologiczny postęp, okazuje się niewystarczające? Wykraczając daleko poza domowe czy biurowe zastosowanie komputera, pojawia się klasa problemów wymagających użycia wręcz gigantycznej mocy obliczeniowej, gdzie konieczne może okazać się dopiero wykorzystanie superkomputera z setkami pracujących równolegle procesorów. Pociąga to za sobą ogromne koszty, przy czym cena takich rozwiązań rośnie nieporównywalnie w stosunku do uzyskanej wydajności.

Odmienne możliwości otwierają się, kiedy duże zadanie obliczeniowe rozumiane jako całość da się podzielić na podzadania dające się wykonać w sposób nie wymagający ciągłej komunikacji i synchronizacji procesorów w trakcie obliczeń. Właśnie wtedy zastosować można strategię przetwarzania rozproszonego, gdzie jednostkom wykonawczym, stanowiącym osobne maszyny, przydzielane są podzadania, po wykonaniu których uzyskane wyniki odsyłane są do komputera zarządzającego całością procesu (rys. 1). Pozwala to bardzo wydajnie zwielokrotnić potencjalną moc obliczeniową przy wykorzystaniu sieci Internet do łączenia komputerów o mocy stosunkowo niewielkiej.



Rys. 1. Podział ról w przetwarzaniu pomiędzy serwer a procesy klienckie
Fig. 1. Role division in processing among server and client processes

Głównym argumentem przemawiającym za stosowaniem takiego rozwiązania jest jego niewspółmiernie niska cena, gdyż zamiast superkomputera wystarczą przeciętne komputery połączone w sieć. Istotnie, sumaryczna cena komputerów biorących udział w projekcie obliczeniowym SETI [1] jest nieporównywalnie większa od ceny odpowiedniego superkomputera. Jednak przy założeniu, że użytkownicy przyłączają się za pośrednictwem Internetu do systemu obliczeń dobrowolnie, uruchamiając jedynie niezbędne oprogramowanie, jest ona praktycznie zerowa! Dzieje się tak, ponieważ system taki nie wymaga żadnych inwestycji w nowe stacje robocze, a wręcz zakłada wykorzystanie już istniejących.

Realizacja takiego programowego zwiększania wydajności jest jednak o wiele trudniejsza w porównaniu z analogicznym rozwiązaniem bazującym na algorytmie sekwencyjnym. Chcąc nie chcąc programista-projektant zmuszony jest do stworzenia pewnych podstawowych procedur komunikacji i synchronizacji obliczeń. Gotowa biblioteka funkcji z

pewnością okaże się przydatna, jednak nie zwalnia programisty od przemyślenia całego przedsięwzięcia, jakim jest rozproszona aplikacja działająca w sieci oraz zaprojektowania z dostępnych elementów motoru działania takiego mechanizmu sieciowego.

Celem podjętych prac badawczych było zaprojektowanie i stworzenie narzędzia pozwalającego programiście możliwie jak najmniejszym nakładem pracy otrzymać aplikację działającą w trybie rozproszonym. Środowisko przetwarzania *DiCE* jest wymiernym efektem podjętych prac badawczych i właśnie jemu poświęcony jest niniejszy artykuł.

2. Przetwarzanie rozproszone

Realizując wybrane zagadnienie w strategii rozproszonej można oprzeć rozwiązanie na jednym z dostępnych środowisk przetwarzania rozproszonego. Chyba najbardziej znane i powszechnie używane są systemy klastrowe.

Mianem klastra określa się system kilku (minimum dwóch) do kilkuset maszyn, które współpracują ze sobą i są „widziane" przez zewnętrzne oprogramowanie, a co za tym idzie przez użytkownika jako jeden system komputerowy. Środowisko klastrowe tworzy się instalując i konfigurując specjalne pakiety oprogramowania, umożliwiające stworzenie wirtualnej maszyny, jak np. PVM i MPI, chociaż mogą być też inne, jak np. MOSIX [2]. W praktyce jest to łatwy sposób na zbudowanie niewielkim kosztem tzw. wielokomputera z reguły pracującego pod kontrolą systemu z rodziny UNIX/Linux.

Bardzo interesującym rozwiązaniem jest technologia JavaSpaces, będąca częścią platformy J2EE firmy Sun [3]. Wykorzystywany tu mechanizm współdzielonej pamięci, zwanej przestrzenią krotek (ang. *tuple space*), pozwala aplikacjom poprzez wymianę obiektów koordynować swoje działanie. Dane umieszczone w przestrzeni są dostępne na zasadzie asocjacji – wyszukiwanie odbywa się na podstawie ich zawartości a nie położenia w pamięci czy też identyfikatora, co ułatwia wyrażanie zapytań w naturalny sposób, np.: "Czy są jeszcze jakieś zadania do przetworzenia?". Przechowywanie obiektów Javy umożliwia również wymianę kodu wykonywalnego (a dokładniej ByteCode Javy) na równi z wymianą danych. Pozwala to na zmianę funkcjonalności tak zbudowanego systemu bez zmiany i ponownej kompilacji jego kodu, a nawet w trakcie działania.

Wymienione techniki nie są oczywiście jedynymi, jednak pozostałe stanowią pewne ich odmiany lub są dość mało powszechne. Działające projekty obliczeniowe typu klient-serwer, w rodzaju Seti@Home czy GIMPS są autonomicznymi produktami z zaszytą wewnątrz logiką przetwarzania i nie przedstawiają cech narzędzi programistycznych. Fakt, że chyba najpopularniejszy z nich, SETI@Home, dysponuje średnią mocą rzędu 65 TeraFLOPów, bardzo dobrze jednak obrazuje siłę rozwiązań opartych na przetwarzaniu rozproszonym [1].

Wymienione przykłady to sprawdzone i stabilne narzędzia, których celem jest dostarczenie mechanizmów wspomagających tworzenie i działanie końcowej aplikacji. Stąd właśnie mowa o środowisku – aplikacja budowana jest z uwzględnieniem specyfiki danego środowiska, a następnie gotowy już program wykonuje się w nim. Każde z tych rozwiązań posiada cechy dla siebie charakterystyczne, stanowiące o swojej sile i słabości. Żadne z nich zatem nie jest pozbawione wad i ograniczeń. Może to być przywiązanie do konkretnego języka programowania bądź systemu operacyjnego lub na przykład większa elastyczność okupiona dłuższym czasem wykonania, charakterystycznym dla interpreterów. Ulepszanie i poszukiwanie alternatywnych narzędzi wspomagających przetwarzanie rozproszone ma w związku z tym jak najbardziej rację bytu.

3. Architektura środowiska

Podstawowe wymogi funkcjonalne, jakie musi spełniać środowisko, to praca w sieci oraz elastyczność. Dla programisty, mającego zaimplementować konkretne rozwiązanie, środowisko *DiCE* stanowi bibliotekę gotowych usług, z których będzie on korzystał. W tym celu powinien stworzyć moduł, realizujący właściwą logikę przetwarzania, a następnie osadzić go w środowisku. Usprawnienie polega na tym, że nie musi się on zajmować projektowaniem i implementacją pewnych podstawowych procedur, wspólnych dla programów komunikujących się między sobą poprzez sieć Internet. Pozwoli mu to skoncentrować swą pracę na tworzeniu logiki samego algorytmu rozwiązującego konkretne zadanie, stosując się jedynie do narzuconej wcześniej specyfikacji budowy modułu oraz korzystania z usług oferowanych przez środowisko.

Ta druga cecha decyduje o zakresie zadań, do których może być ono zastosowane, a zatem o jego użyteczności. Środowisko *DiCE* zapewnia elastyczność dzięki zastosowaniu dynamicznie ładowanych bibliotek dll w roli wymiennych modułów, z którymi współpracuje. W module takim programista implementuje istotę procesu przetwarzania, co pozwala na oddzielenie zarówno logiczne, jak i fizyczne samego algorytmu od środowiska.

Kolejny podział wynika już z samej architektury typu klient-serwer, w jakiej zostało zrealizowane oprogramowanie. Funkcjonalnie środowisko *DiCE* to dwa odrębne programy:

KO – klient środowiska, przeznaczony do uruchamiania na komputerach, których moc obliczeniowa ma być wykorzystana do przetwarzania danych. Po nawiązaniu połączenia z serwerem, automatycznie włączany jest on w proces obliczeń.

SK – serwer koordynujący przebieg procesu obliczeniowego. Zapewnia usługi, m.in. przezroczystą bezkolizyjną transmisję danych z odległymi procesami

wykonującymi obliczenia, a także zajmuje się dystrybucją modułu do części klienckiej środowiska.

Schemat pracy utworzonego systemu jest prosty. Administrujący procesem obliczeniowym za pomocą interfejsu programu SK wybiera i ładuje odpowiedni moduł, a następnie nakazuje rozpoczęcie przetwarzania. Sam proces jednak rozpocznie się dopiero z chwilą, kiedy do systemu zgłosi się przynajmniej jeden program KO. Poprzez zgłoszenie rozumie się nawiązanie połączenia sieciowego z serwerem wykonane z komputera, na którym został uruchomiony program KO.

Po przyjęciu zgłoszenia pomiędzy serwerem a klientem następuje wymiana informacji mająca na celu ustalenie wersji modułu po stronie klienta. W przypadku niezgodności lub braku modułu inicjowany jest transfer odpowiedniej biblioteki. Środowisko zapewnia więc przezroczystą dystrybucję wykonywanego kodu algorytmu pomiędzy wszystkie stacje robocze biorące udział w przetwarzaniu.

Przy tego typu architekturze sieciowej, gdzie obecny jest serwer i wielu klientów, naturalna i najbardziej użyteczna wydaje się być organizacja przetwarzania typu „farming" [4]. Polega ona na prostym podziale na jednego zarządcę, rozdzielającego zadania i wielu robotników, którzy je wykonują.

W miarę jak do systemu zgłaszają się nowe jednostki wykonawcze, należy im przydzielić odpowiednią porcję danych do przetworzenia. Adaptację algorytmu należy więc zacząć od rozpatrzenia możliwości podziału problemu w ujęciu całościowym na mniejsze podzadania. Bardzo ważne jest, aby specyfika przyjętego zadania stwarzała taką możliwość. Brak tej cechy z oczywistych powodów dyskwalifikuje dane zadanie, problem, jako taki, gdzie można by zastosować tryb przetwarzania rozproszonego.

Komunikacja pomiędzy procesami zapewnia koordynację procesu obliczeniowego, a czas takiej synchronizacji ma krytyczny wpływ na wydajność całego systemu. Konkluzja jest prosta – im większa gruboziarnistość podzadań, tym lepszą wydajność taki system jest w stanie osiągnąć. Jest to klucz do skalowalności i stanowi siłę tego typu rozwiązań. Należy o tym pamiętać obierając strategię wymiany danych i w miarę możliwości podział dopasować jak najlepiej do indywidualnego przypadku.

W związku z tym, że jedynie serwer ma kontrolę nad całością procesu przetwarzania, tutaj dokonuje się przydzielanie podzadań. Do klienta należy wykonywanie obliczeń na podstawie otrzymanych porcji danych. Wyniki obliczeń częściowych odsyłane są z powrotem do serwera i tam są one sprawdzane i zapamiętywane. Te trzy rodzaje działań, przedstawione na rys.1, opisują w sumie algorytm przetwarzania rozproszonego i są w całości definiowane przez użytkownika środowiska.

W tym celu środowisko *DiCE* wykorzystuje moduł zawierający opis takiego algorytmu, będący w istocie składnikiem COM implementującym interfejs *IRun* [5]. Interfejs ten zawiera

dwie funkcje: *Run* oraz *Work*. Pierwsza z nich docelowo definiować ma sposób przydzielania danych do przetworzenia oraz gromadzenia wyników częściowych i jest uruchamiana po stronie serwera. *Work* z kolei implementuje właściwą logikę obliczeń na podstawie otrzymanej paczki danych.

Po wywołaniu odpowiednich funkcji interfejsu *IRun* komunikacja pomiędzy środowiskiem a modułem przebiega za pomocą mechanizmu wymiany komunikatów i sygnalizowania zdarzeń [6]. Zapewnia to m.in. większą szybkość działania w porównaniu z wariantem wywoływania odpowiednich funkcji za każdym razem, kiedy zajdzie odpowiednie zdarzenie [5,7].

W celu usprawnienia odbierania i nadawania komunikatów przez moduł zaprojektowane zostały dwie klasy do obsługi takich operacji na wspólnej pamięci. Programista ma do dyspozycji obiekty *CRunComm* oraz *CWorkComm*, w zależności od tego, którą z funkcji interfejsu implementuje. Klasy te realizują wszystkie możliwe operacje zgodnie ze zdefiniowanym protokołem zewnętrznym wymiany komunikatów *DiCE*, dotyczącym współpracy środowiska z modułem. Dzięki temu programista za pomocą prostego wywołania funkcji skorzystać może z dowolnej z usług oferowanych przez środowisko, takich jak przesłanie pliku do stacji roboczej lub np. sprawdzenie czasu jej odpowiedzi.

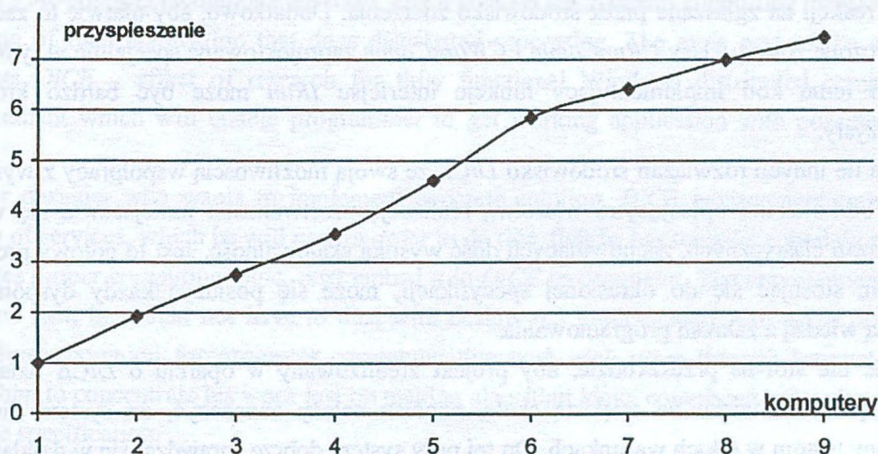
Przy takim podejściu do zadań programisty będzie należało praktycznie napisanie dwóch funkcji interfejsu opisujących wykonywany proces przetwarzania. Obie te funkcje (przy czym jedna z założenia będzie uruchomiona w wielu kopiach) prowadzić będą dialog między sobą. Protokół wewnętrzny wymiany komunikatów środowiska *DiCE* definiuje sieciową wymianę pakietów danych pomiędzy klientem KO a serwerem SK. Właśnie on zapewnia przezroczystość komunikacyjną oraz enkapsulację pakietów przekazywanych pomiędzy modułami działającymi na różnych stacjach roboczych.

Program SK spełnia bardzo ważną funkcję pomostu pomiędzy tymi dwoma protokołami. Zapewnia on bezkolizyjność wymiany informacji z modułem. Tutaj konieczna jest ochrona dostępu, ponieważ w danym momencie tylko jeden proces klienta może komunikować się z zarządzającą całością obliczeń częścią modułu.

4. Przykłady zastosowania

Dla celów prezentacji utworzonego środowiska zaimplementowane zostały w postaci modułów dwa przykładowe algorytmy: algorytm poszukujący liczb pierwszych oraz algorytm poszukujący możliwie najkrótszego połączenia pomiędzy punktami, rozwiązujący tzw. problem komiwojażera [8].

Działanie przykładowych algorytmów *Primes* i *Traveller* w środowisku *DiCE* zostało przetestowane w sieci lokalnej łączącej dziewięć komputerów. Na pojedynczym komputerze algorytm *Primes* zadany przedział 200 000 liczb przetwarzał w poszukiwaniu liczb pierwszych w czasie około 95 sekund. Uruchomienie systemu w konfiguracji dziewięciu komputerów pozwoliło skrócić ten czas do ok. 12 sekund, co oznacza około 7,5-krotne przyspieszenie procesu obliczeń w porównaniu do czasu uzyskanego na pojedynczej stacji. Rysunek 2 przedstawia przyspieszenie uzyskane podczas dołączania kolejnych stacji do systemu.



Rys. 2. Przyspieszenie obliczeń uzyskane dla różnej liczby komputerów
Fig. 2. Acceleration of computations reached for variant number of computers

Moduł *Traveller* został skonfigurowany do poszukiwań dla trzynastu punktów pomiędzy punktem startowym a końcowym. Obliczenia należało zatem wykonać dla 13! kombinacji, czyli dla ok. 6 mld możliwych ścieżek. Dziewięć jednostek wykonawczych rozwiązało zadanie w 9 minut i 46 sekund, co oznaczało ponad 8.5-krotne przyspieszenie procesu obliczeń.

Uzyskane, dość wysokie współczynniki przyspieszenia zawdzięczane są w głównej mierze faktowi, że procesy obliczeniowe działające na stacjach roboczych były luźno powiązane ze sobą. Synchronizacja odbywała się w zasadzie jedynie przy starcie i zakończeniu procesu obliczeń, nie było zatem sytuacji, w których jakiś procesor byłby bezczynny, czekając aż inna stacja robocza dokończy swoje obliczenia.

5. Podsumowanie

Środowisko *DiCE* stworzone zostało tak, aby implementacja lub adaptacja już gotowego algorytmu do przetwarzania w trybie rozproszonym odbywała się jak najmniejszym nakładem pracy ze strony programisty. Do stworzenia biblioteki wymagana jest jedynie

elementarna znajomość technologii COM. Praca ze środowiskiem DiCE pozwala w efekcie zaoszczędzić żmudnego tworzenia części oprogramowania realizującej transmisję danych przez sieć, do minimum ogranicza czynności związane z dystrybucją i instalacją modułów na stacjach roboczych. Środowisko realizuje przezroczysty transport danych pomiędzy komplementarnymi funkcjami modułu, zapewnia również bezkolizyjny dostęp do wątku funkcji zarządzającej *Run* modułu.

Komunikacja modułu algorytmu bazuje na wymianie komunikatów i sygnalizowaniu zdarzeń. Do programisty należy praktycznie jedynie zaimplementowanie w funkcjach *Run* i *Work* reakcji na zgłaszane przez środowisko zdarzenia. Dodatkowo, aby ułatwić to zadanie, dostarczone zostały klasy *CRunComm* i *CWorkComm* zaprojektowane specjalnie w tym celu. Dzięki temu kod implementujący funkcje interfejsu *IRun* może być bardzo krótki i przejrzysty.

Na tle innych rozwiązań środowisko *DiCE* ze swoją możliwością współpracy z wymienionymi bibliotekami opisującymi właściwą realizację przetwarzania umiejscawia się wśród rozwiązań elastycznych, zachowujących dość wysoką skalowalność. Jest to gotowy produkt, którym, stosując się do określonej specyfikacji, może się posłużyć każdy dysponujący średnią wiedzą z zakresu programowania.

Nic nie stoi na przeszkodzie, aby projekt zrealizowany w oparciu o *DiCE* działał w rozległym środowisku na większą skalę, chociaż należy zaznaczyć, że system nie był poddany testom w takich warunkach. Do tej pory system dobrze sprawdzał się w działaniu w sieciach lokalnych i z powodzeniem może znaleźć zastosowanie na szerokim polu obliczeń inżynierskich. Proponowane rozwiązanie pozwala wydatnie skrócić czas obliczeń i drogę do uzyskania kompletnego rozwiązania programistycznego, co zawsze oszczędza zarówno wysiłek jak i czas, a w konsekwencji nakłady finansowe.

LITERATURA

1. Search for Extraterrestrial Intelligence at home, http://setiathome.ssl.berkeley.edu/
2. Wyrzykowski R., Piech H.: PVM - MPI - teraźniejszość i przyszłość, http://k2.pcz.czest.pl/~roman/pvm_mpi1.html
3. The JavaSpaces™ v1.1 Specification, http://www.sun.com/jini/specs/js1_1.pdf
4. Grids and Grid Technologies for Wide-Area Distributed Computing, http://www.csse.monash.edu.au/~rajkumar/papers/gridtech.pdf
5. Guy i Henry Eddon, „COM+ Programowanie", wyd. 1, wyd. RM, 2001
6. Al Williams: Programowanie Windows 2000. Czarna księga, wyd. 1, wyd. Helion, 2001
7. MSDN library July 2002 – dokumentacja MSDN, Czerwiec 2002
8. Solving Traveling Salesman Problem, http://www.math.princeton.edu/tsp/index.html

Recenzent: Dr inż. Hafedh Zghidi

Wpłynęło do Redakcji: 31 marca 2003 r.

Abstract

Distributed computing environments, such as cluster systems based on PVI/MPI libraries or *Sun's JavaSpaces* technology are tools which provide mechanisms supporting creation and working of final application that does distributed processing. The main part of the article presents *DiCE* – effect of research for fully functional Windows distributed computing environment which will enable programmer to get working application with possible less effort.

For designer who wants to implement concrete solution, *DiCE* environment represent library of services, which he will use. In order to do this, first he has to make a module which executes proper computing logic, next embed it in *DiCE* environment. The improvement will relay on fact, he would not have to deal with design and implementation of some primary procedures, common for programs communicating with each other through Internet. It'll allow him to concentrate his work just on making algorithm logic, complying only with *DiCE* module specification.

DiCE consists of two applications: *SK* server and *KO* client. Replaceable module is in fact COM component implementing interface *IRun* containing two functions *Run* and *Work*. First of them runs on *DiCE's* server side and should define the way of assigning data to compute and storing results (fig. 1). *Work* describes processing of received data subranges. To simplify the task of delivering module a pair of classes: *CRunComm* and *CWorkComm* has been designed to handle communication with both server and client. Preliminary tests using 9 workstations revealed 7,5 times faster completion of *Prime* trial algorithm (fig. 2) and acceleration over 8,5 in *Traveller* module which is good and satisfying result.

DiCE environment with his ability to cooperate with replaceable modules places itself among flexible but also scalable solutions. Only a little knowledge of COM is enough for the programmer to build his own project. *DiCE* allow him to short distinctly the way of achieving complete software solution, what always saves effort as well as time and in consequence financial expenditures.

Adres

Maciej KOCOŃ: SKG S.A., ul. Listopadowa 11, 43-300 Bielsko-Biała, Polska,
maciej_kocon@skg.pl .