STUDIA INFORMATICA 2009
Volume 30 Number 4 (87)

Hafed ZGHIDI, Adam ŚWITOŃSKI, Błażej ADAMCZYK
Politechnika Śląska, Instytut Informatyki

INTEGRACJA WIELU PLATFORM I SYSTEMÓW OPERACYJNYCH, ZA POMOCĄ MECHANIZMÓW RPC DLA ZRÓWNOLEGLENIA ROZWIĄZANIA ZŁOŻONYCH ALGORYTMÓW

Streszczenie. Celem niniejszej pracy jest wykorzystanie mechanizmów zdalnego wywołania procedur do integracji wielu platform komputerowych do rozwiązania złożonych algorytmów obliczeniowych. W tym celu wykorzystano heterogeniczną sieć komputerową, składającą się z komputerów o różnej architekturze i pracujących pod kontrolą różnych systemów operacyjnych.

Słowa kluczowe: rpc, integracja, algorytmy równoległe

INTEGRATION OF MULTIPLE PLATFORMS AND OPERATING SYSTEMS, BY MEANS OF RPC, TO PARALLELIZE EXECUTION OF COMPLICATED ALGORITHMS

Summary. The aim of this work is to use remote procedure call mechanisms to integrate multiple computer platforms in order to solve complicated computation algorithms. A heterogeneous computer network consisting of computers with different architectures and different operating systems.

Keywords: rpc, integration, parallel algorithms

1. Wstęp

Celem niniejszej pracy jest wykorzystanie mechanizmów zdalnego wywołania procedur do rozwiązania złożonego algorytmu obliczeniowego w heterogenicznej sieci komputerowej. Ma to na celu zbadanie możliwości integracji różnych platform programowych i sprzętowych

do realizacji pojedynczego zadania. Na szczególną uwagę zasługuje możliwość wykorzystania ww. mechanizmów na platformie IBM iSeries.

2. Protokół zdalnego wywołania procedur (RPC)

2.1. Mechanizmy zdalnego wywołania procedur [1 i 5]

Mechanizmy RPC to wysokopoziomowe protokoły komunikacyjne oparte na przesyle komunikatów, pozwalające na tworzenie aplikacji sieciowych przez specjalnie przygotowane procedury. Procedury te zasłaniają niskopoziomowe sieciowe mechanizmy, ułatwiając tym samym zadanie programistom.

Mechanizmy RPC implementują logiczny model klient/serwer. Za ich pomocą klienci wywołują procedury, które zgłaszają żądania wykonania usługi do serwera. Gdy na komputerze serwera pojawia się żądanie, serwer przyjmuje je, wykonuje zadanie i wysyła odpowiedź. Głównym celem programowania za pomocą RPC jest możliwość wykonania aplikacji w sieciowym systemie komputerowym, gdzie dowolny komputer może grać rolę serwera, klienta lub równocześnie serwera i klienta. Aplikacje te korzystają z mechanizmów RPC, by uniknąć szczegółów związanych z programowaniem w sieci, umożliwiając tym samym klientom usługi sieciowe, bez świadomości istnienia i funkcjonowania sieci. Programy napisane z wykorzystaniem RPC mogą być implementowane w różnych językach programowania (C, Pascal, ...), uruchamiane na takich samych komputerach bądź na różnych komputerach o odmiennej architekturze i mogą zapewniać komunikację między procesami na tym samym komputerze lub na różnych komputerach.

Zaletą wywołania procedur zdalnych jest to, że procedury te są wykonywane w przestrzeni adresowej komputera obsługującego (serwera).

Jak już wspomniano, RPC korzysta z architektury klient/serwer. Jest to logiczny model, w którym serwer jest komputerem oferującym usługi sieciowe. Usługa sieciowa jest utożsamiona ze zbiorem składającym się z jednego lub kilku programów zdalnych. Zdalny program implementuje jedną lub kilka zdalnych procedur. Procedury te, ich parametry i wyniki są definiowane w specyficznym protokole. W momencie uruchamiania programu serwera na zdalnych komputerach następuje rejestracja oferowanych przez niego usług u demona *Portmap*. Chcąc korzystać z wybranej usługi (zdalnej procedury), użytkownik sieci (klient) musi ją zainicjować. Informację o dostępnych usługach uzyskuje u demona *Portmap*.

Sposób wywołania procedur zdalnych jest podobny do sposobu wywołania procedur lokalnych. Dla procedur lokalnych argumenty wywoływanej procedury są trzymane w ustalonym miejscu (rejestry, zmienne...). Po wykonywaniu procedury wyniki są pobierane z tego

samego miejsca. W przypadku procedur odległych argumenty i wyniki wywoływanych procedur wymieniane są pomiędzy dwoma procesami, pierwszy jest procesem wywołującym (klient), a drugi jest procesem wywoływanym (serwer). Ze strony klienta odbywa się to w następujący sposób: wysyła on żądanie do serwera i czeka na odpowiedź. Żądanie to zawiera parametry wywoływanej procedury i inne informacje. Odpowiedź zawiera wynik procedury i inne informacje. Po otrzymaniu odpowiedzi klient odbiera wyniki wywoływanej procedury. Z drugiej strony zaś serwer ciągle czeka na żądanie. Po pojawieniu się żądania serwer wczytuje parametry procedury, wykonuje procedurę, wysyła odpowiedź, po czym czeka na kolejne żądanie.

2.2. Komunikacja klient/serwer w RPC

Programy klienta i serwera komunikują się za pośrednictwem interfejsów, które zapewniają przekazywanie parametrów wywoływanych procedur i ich wyników. W systemie klienta istnieje procedura lokalna zwana łącznikiem (pieńkiem, ang. *stub*) klienta, a w systemie serwera istnieje odpowiednio łącznik (pieniek) serwera. Rola łącznika klienta polega na odebraniu sterowania od programu klienta, przyjęciu parametrów wywoływanej procedury, ich przekształceniu do odpowiedniej postaci standardowej (XDR), utworzeniu (w zależności od rozmiaru komunikatu) odpowiedniej liczby komunikatów sieciowych i ich wysłaniu przez sieć do komputera serwera. Po nadejściu odpowiedzi łącznik klienta odbiera komunikaty, przekształca je do postaci komputera lokalnego, przekazuje je do programu klienta i kończy działanie przekazując sterowanie do programu klienta. Rola łącznika serwera jest podobna do roli łącznika klienta. Odbiera on komunikaty wysyłane przez łącznika klienta, przekształca je do postaci komputera serwera i przekazuje je do odpowiedniej procedury, przekazując tym samym do niej sterowanie. Po zakończeniu działania wywołana procedura przekazuje sterowanie i wyniki do pieńka serwera, który znowu je przekształca do postaci standardowej, tworzy odpowiednią liczbę komunikatów sieciowych i wysyła je przez sieć do pieńka klienta kończąc tym samym działanie.

2.3. Reprezentacja danych

Aplikacje napisane z wykorzystaniem RPC są przeznaczone do działania w sieci. Sieć może być heterogeniczna, tzn. że składa się z komputerów o różnej architekturze, komunikujących się za pomocą różnego rodzaju protokołów i pracujących pod różnymi systemami operacyjnymi, lub też jednolita, tzn. że składa się z komputerów o tej samej architekturze z wykorzystaniem jednego systemu operacyjnego. Dla zapewnienia działania mechanizmów RPC w obu przypadkach należy przekształcić parametry wejściowe i wyj-

ściowe wywoływanych procedur do jednolitej postaci, gdyż proces wywołujący i proces wywoływany mogą być wykonywane na różnych platformach sprzętowych i programowych. Takim standardem jest format sieciowy XDR (ang. *eXternal Data Representation*). Mechanizmy RPC przekształcają parametry i wyniki wywoływanych procedur przed ich wysyłaniem do sieci. Czas konwersji danych do postaci XDR oraz z postaci XDR do postaci docelowego komputera jest niewielki i jest krótszy od czasu ich transmisji. Czynność ta jest wykonywana nawet w przypadku, gdy procesy są wykonywane na takich samych komputerach z jednym systemem operacyjnym, zapewniając tym samym jednolity, przenośny mechanizm. Proces konwersji danych z jednej postaci na inną nazywany jest szeregowaniem danych (ang. serializing/deserializing). Funkcje wykonujące tę czynność nazywane są filtrami. Ten sam filtr dokonuje zarówno przekształcenia do postaci XDR, jak i przekształcenie odwrotne, rozpoznając kierunek przekształcenia. Dla podstawowych typów zmiennych języka C istnieją gotowe filtry XDR. Możliwe jest również przekształcenie złożonych typów danych.

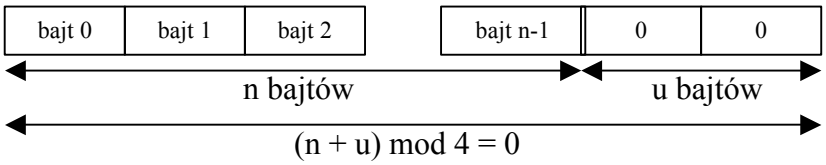
2.3.1. Struktura bloku XDR

Długość bloku danych w formacie XDR jest zawsze wielokrotnością 4 bajtów (32 bitów). Dane są ustawione w takiej kolejności, że bajt m wyprzedza bajt m+1 (ang. *"big-endian"*) i numerowane od 0 do n-1. Jeśli długość bloku nie jest wielokrotnością 4, to jest on uzupełniony odpowiednią liczbą (od 1 do 3) zerowych bajtów (rys. 3).

Przekształcenie danych za pomocą filtru XDR wymaga umieszczenia danych w potoku XDR. Potok to strumień danych (ciąg bajtów). Komputer nadawca przekształca dane do postaci XDR i umieszcza je w potoku. Odbiorca natomiast czyta dane z potoku i przekształca je do formatu komputera.

W celu skasowania istniejącego potoku należy korzystać z funkcji xdr_destroy().

Biblioteka XDR posiada wiele funkcji operujących na potokach. Umożliwiają one między innymi: określenie pozycji w potoku (xdr_getpos()), ustawienie pozycji w potoku (xdr_setpos()), sprawdzenie, czy w potoku są dane do czytania (xdrrec_eof()).



Rys. 1. Struktura bloku XDR
Fig. 2. XDR block structure

3. Grupowanie danych Fuzzy isoData

W celu oszacowania mocy obliczeniowej systemu rozproszonego wykorzystano iteracyjny algorytm grupowania danych *Fuzzy ISODATA* [3], dokonujący podziału *n*-elementowego zbioru wektorów na c grup. Algorytm ten stosowany jest w zadaniach eksploracji danych, gdzie stanowi jedną z wiodących metod wyznaczania grup obiektów podobnych. Charakteryzuje się on dużą złożonością obliczeniową, tak więc równoległa jego implementacja ma istotne praktyczne znaczenie. Patrząc natomiast z perspektywy programowania równoległego każdy z kroków iteracji może być podzielony na dowolną liczbę niezależnych podzadań, przy którym teoretycznie możliwe jest uzyskanie przyspieszenia równego liczbie niezależnych jednostek obliczeniowych.

W algorytmie Fuzzy ISODATA podział wektorów na grupy reprezentowany jest przez rozmytą macierzą podziału B:

$$B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1c} \\ b_{21} & b_{22} & \cdots & b_{2c} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nk} \end{bmatrix} \tag{1}$$

Element b_{ij} podaje przynależność *i*-tego wektora do grupy o identyfikatorze *j*.

Algorytm ma za zadanie wyszukanie takiego podziału, dla którego funkcja kryterialna J osiąga minimum:

$$J(v,B) = \sum_{i=1}^{c}\sum_{k=1}^{n}(b_{ik})^q \cdot d^2(v_i, x_k) \tag{2}$$

gdzie $d(v_i,x_k)$ to odległość wektora xx od prototypu grupy v_i.

We wzorze (2) zastosowano odległość Euklidesową i prototyp punktowy wyliczany jako środek ciężkości wektorów przypisanych do danej grupy.

Działanie algorytmu można opisać w następujących krokach:

- Ustalenie liczby grup, stopnia rozmytości *q*, minimalnej zmiany *e*, inicjalizacja licznika iteracji *t*=0.
- Inicjalizacja macierzy podziału B(t) (losowa).
- Aktualizacja wektora parametrów:

$$m_i(t) = \frac{\sum_{k=1}^{n} b_{ik}(t) \cdot x_x}{\sum_{k=1}^{n}(b_{ik}(t))^q} \tag{3}$$

- Aktualizacja macierzy podziału

$$b_{ik}(t) = \frac{\left(\frac{1}{d(v_i, x_k)}\right)^{\frac{2}{q-1}}}{\sum_{j=1}^{c}\left(\frac{1}{d(v_j, x_k)}\right)^{\frac{2}{q-1}}} \tag{4}$$

- Jeśli $|B(t)-B(t-1)|<e$, to $t=t+1$ oraz idź do 3, w przeciwnym przypadku zakończ działanie.

Dodatkowo, w ramach kroku 5. na konsoli systemowej wypisywana jest statystyka szybkości wykonania, co ma wpływ na działanie podsystemu QINTER.

W kolejnych chwilach czasu zliczana jest liczba wykonanych iteracji, gdzie właściwa iteracja obejmuje kroki o numerach 3, 4 i 5. Można zauważyć, że wszystkie iteracje mają jednakową złożoność obliczeniową, tak więc liczba ich wykonań przypadająca na jednostkę czasu będzie zależna jedynie od dostępnej mocy obliczeniowej.

4. Zrównoleglenie algorytmu „Fuzzy ISO Data Clustering" za pomocą RPC

Problem: Zadaniem algorytmu jest podzielenie kolejnych grup wektorów na mniejsze klastery za pomocą iteracyjnego algorytmu Fuzzy ISO Data Clustering. Zakładamy, że wielkość problemu definiowana będzie za pomocą ilości grup wektorów przeznaczonych do grupowania.

Inicjalizacja i podział problemu odbywa się na jednym, z góry wybranym, komputerze, zwanym klientem. Wszystkie inne komputery w sieci są serwerami oczekującymi na kolejne zadania do wykonania. Klient po rozesłaniu zadań, co pewien okres czasu, pyta każdy z serwerów, czy obliczenia zostały zakończone. Serwery dzięki implementacji wielowątkowej, za pomocą biblioteki *pthreads*, są w stanie podczas obliczeń odpowiadać klientowi.

W zależności od systemu operacyjnego implementacje RPC i kompilatory C lekko różnią się. W związku z tym przygotowane zostały trzy główne wersje algorytmu:

1. Wersja dla systemów Microsoft Windows:
   - biblioteka RPC – Netbula ONCRPC,
   - środowisko Microsoft Visual Studio.
2. Wersja dla systemów Unix/Linux:
   - standardowa biblioteka RPC,
   - kompilator g++.

3. Wersja dla systemów i5/OS:
   - standardowa biblioteka RPC,
   - środowisko PDM.

Wszystkie implementacje są napisane w języku C i część obliczeniowa kodu jest identyczna. W celu przeprowadzenia testów i porównań algorytmu sekwencyjnego i równoległego dodatkowo wymagane również były trzy wersje algorytmu sekwencyjnego.

5. Wyniki

W przypadku testowanego algorytmu pożądanym efektem jest wzrost szybkości wykonywania algorytmu. Warto jednak zauważyć, że mechanizm RPC integrujący tak różne platformy i systemy operacyjne daje również wiele innych możliwości. Dla przykładu, istnieją takie algorytmy, które wymagają zbyt dużo pamięci operacyjnej, by zostać uruchomione na jednym komputerze. Dzięki RPC możemy, relatywnie łatwo, rozproszyć wykonywanie takich algorytmów na wiele maszyn, poszerzając tym samym pamięć operacyjną. Jeszcze innym przykładem zastosowania takiego rozwiązania jest podział algorytmu na części, które wykorzystują konkretną funkcjonalność lub zasób systemu operacyjnego. Dzięki takiemu podziałowi możemy zdecydować, który komputer ma wykonywać konkretne zadanie. I tak na przykład, operacje dyskowe i związane z bazą danych moglibyśmy wykonywać na systemie i5, który jest z założenia przystosowany do takich zadań, natomiast operacje obliczeniowe przekazać innemu systemowi.

Wracając jednak do algorytmu ISO Data Clustering, wszystkie pomiary czasu obliczeń zostały wykonane dla trzech różnych wielkości problemu. Inicjalizacja nie jest brana pod uwagę. W implementacjach RPC rozsyłanie i kolekcjonowanie danych jest oczywiście również uwzględnione. Wszystkie pomiary pobierane były trzykrotnie, aby mieć pewność, że są dokładne i nie zostały zakłócone.

Komputery wykorzystane podczas testów:

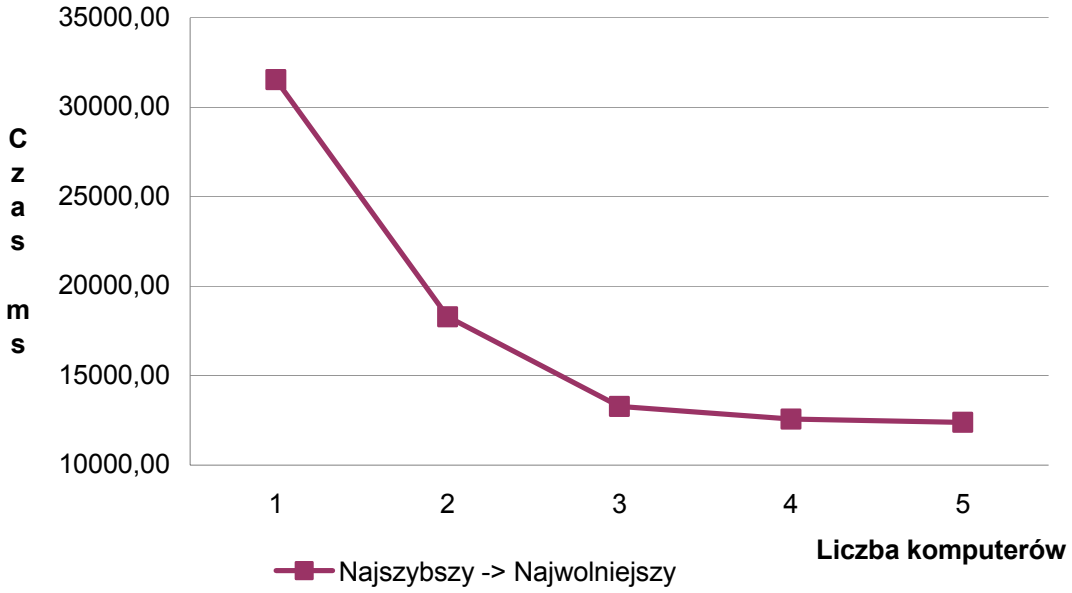
1. **AMD Sampron 2600+**
   448 MB RAM
   OS: Microsoft Windows XP
2. **Pentium mobile 1.6 GHz**
   512 MB RAM
   OS: Microsoft Windows XP
3. **Intel Core Duo 1.66 GHz**
   2 GB RAM
   OS: Microsoft Windows XP

4. **Pentium III 800MHz**
   256 MB RAM
   OS: Linux Ubuntu
5. **AS/400 750MHz**
   2 GB RAM
   OS: i5/OS

Testy wykonane zostały tak, aby można było nie tylko porównać wydajność algorytmów sekwencyjnego i równoległego, ale również sprawdzić, jaki wpływ na wydajność ma zwiększająca się liczba komputerów w sieci RPC.

Wyniki kolejnych testów przedstawione zostały w tabeli 1.

Jak łatwo można zaobserwować, komputery z systemami Microsoft Windows są znacznie szybsze w porównaniu do systemów Linux i i5/os. System operacyjny nie ma tutaj jednak dużego wpływu. Zarówno maszyna z systemem Linux, jak i i5/os są generacyjnie dużo starsze niż pozostałe użyte komputery. Dodatkowo serwer as/400 posiada jeden procesor i tylko 75% jego pracy jest przydzielone do partycji logicznej, na której uruchomiony był nasz program. System i5/os pracuje również jako serwer, więc jest również znacznie obciążony różnego rodzaju usługami.



Rys. 2. Czas wykonywania problemu pierwszego w miarę dodawania coraz wolniejszych maszyn
Fig. 2. First problem execution time each time adding a slower machine

Dla problemu pierwszego (50×20 wektorów, 5 klasterów, 100000 iteracji) test był wykonany dla każdego komputera lokalnie, a następnie zwiększano liczbę komputerów. Chcąc określić, jaki wpływ na wydajność ma zwiększanie sieci komputerów, należy zwrócić uwagę na wydajność kolejno dodawanych maszyn. Po wykonaniu kilku próbnych pomiarów potwierdziło się, że dodając kolejne komputery przyspieszenie ciągle zmniejsza się.

Tabela 1

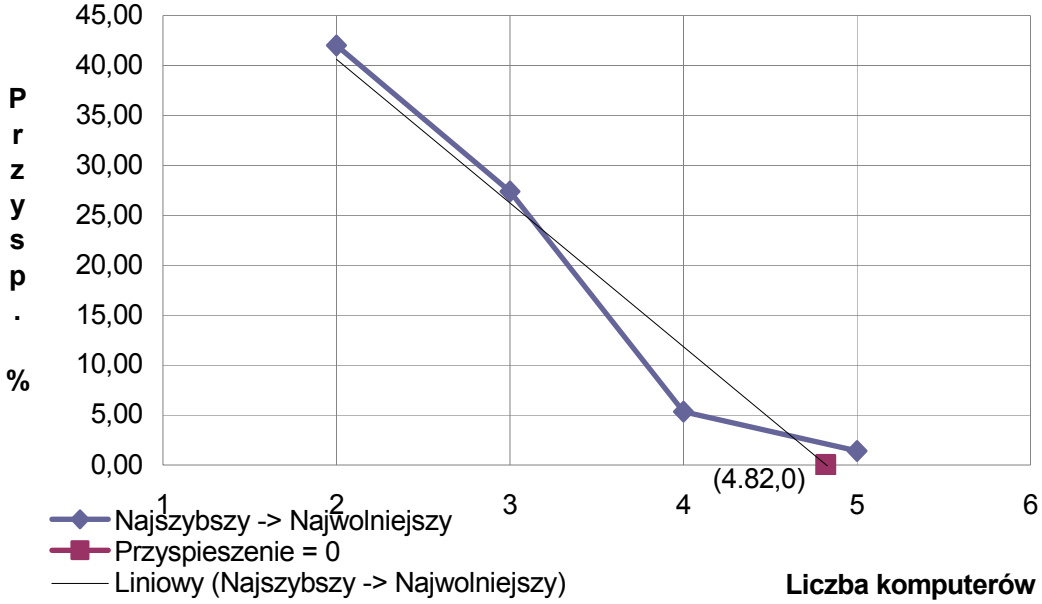
Porównanie wydajności algorytmu RPC w zależności od problemu i liczby komputerów

Komputer / Problem	XP AMD Sempron 448 RAM	XP P1.6mobile 512 RAM	XP Core Duo 1.66 2048 RAM	Linux Ubuntu P3 800MHz 256 RAM	as400
50 x 20V 5C 100000I	31547	50332	51297	255196	434320
	31531	49310	51484	255247	434128
	31546	49270	50828	255203	434369
	18343		-		
	18281				
	18234				
	13266			-	
	13281				
	13282				
	12532				-
	12641				
	12532				
	-			161788	
				162047	
				161984	
	-		34266		
			34094		
			34266		
	-	18953			
		18953			
		18906			
	12468				
	12344				
	12363				
250 x 20V 5C 100000I	157609	246124	251734	1274540	2171567
	157610	246144	253125	1276320	2171582
	157625	246204	252031	1276132	2171624
	97375		-		
	97079				
	97078				
	68953			-	
	69188				
	69000				
	64296				-
	63984				
	63407				
	60156				
	63766				
	60297				
3000 x 20V 5C 100000I	712219				
	1893188	-	-	-	-

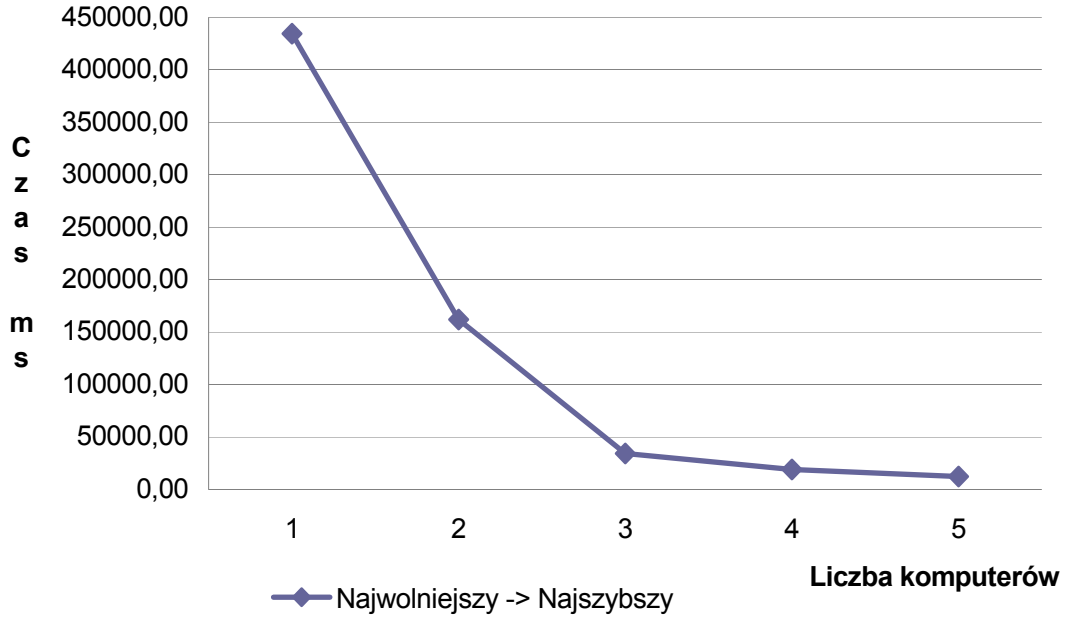
Można było przypuszczać, że efekt ten będzie potęgowany, jeżeli będziemy dodawać coraz wolniejsze komputery, i odwrotnie, w miarę dodawania szybszych jednostek przyspieszenie będzie wolniej malało. Rysunki od 2 do 7 przedstawiają, jak maleje czas wykonywania problemu pierwszego oraz jak zachowuje się przyspieszenie w miarę powiększania się sieci:

Testując te dwa skrajne przypadki (dodawanie coraz wolniejszych i coraz szybszych jednostek), można spróbować „przewidzieć", dla jakiej ilości komputerów podobnej klasy dodawanie kolejnych nie będzie już się opłacać. Prowadząc linię trendu dla obu wykresów przyspieszenia aż do punktu przecięcia z osią poziomą, następnie obliczając średnią z argumentów obu punktów przecięcia możemy stwierdzić, iż przypuszczalnie, dla problemu pierwszego, dodając komputery podobnej klasy osiągniemy maksimum wydajności po dodaniu 7 komputera. Powyższe obliczenia to oczywiście bardzo proste przypuszczenia, ale pokazują one, w jaki sposób można sprawdzić, czy realne jest osiągnięcie oczekiwanej wydajności.

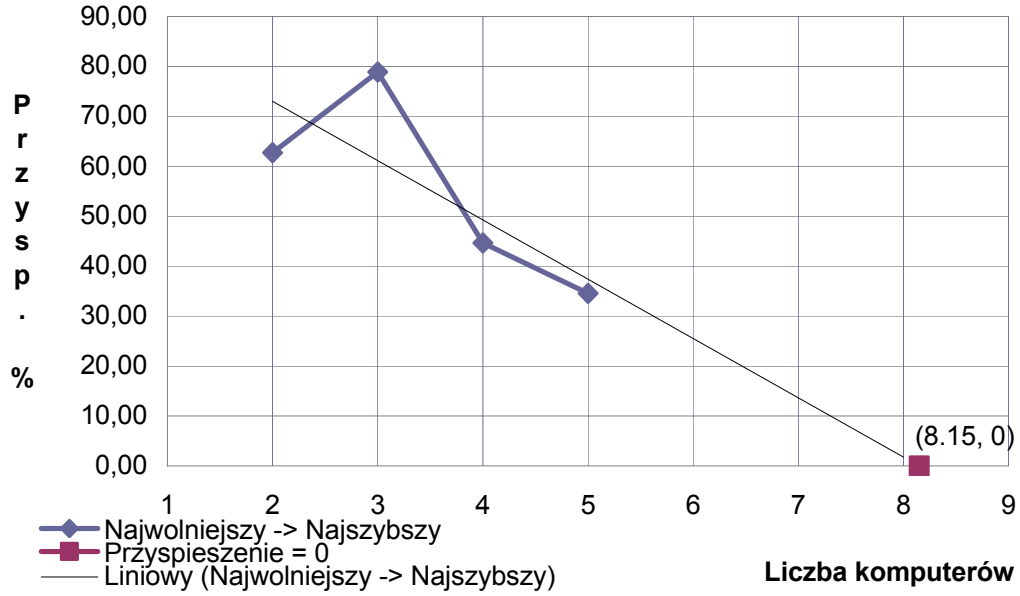
Warto również zauważyć, że gdyby dodać do sieci o wiele szybszą maszynę od pozostałych komputerów w sieci, możemy nawet osiągnąć wzrost przyspieszenia, tak jak można to zaobserwować na rys. 4, gdzie do sieci dwóch komputerów dodano trzeci, który jest znacznie szybszy, przez co przyspieszenie tymczasowo wzrasta.



Rys. 3. Przyspieszenie problemu pierwszego w miarę dodawania coraz wolniejszych maszyn
Fig. 3. First problem acceleration each time adding a slower machine

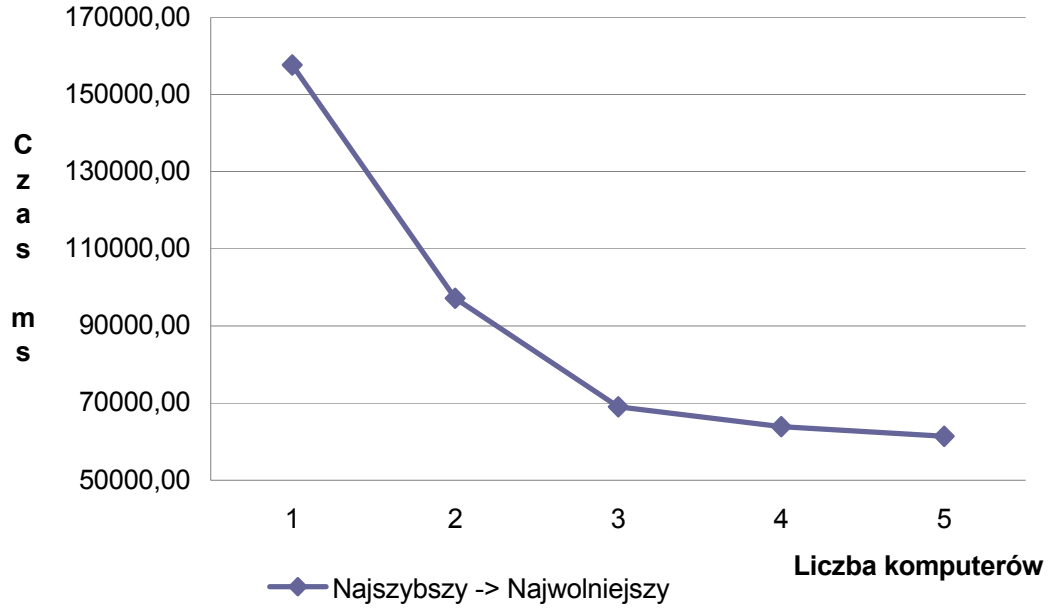


Rys. 4. Czas wykonywania problemu pierwszego w miarę dodawania coraz szybszych maszyn
Fig. 4. First problem execution time each time adding a faster machine

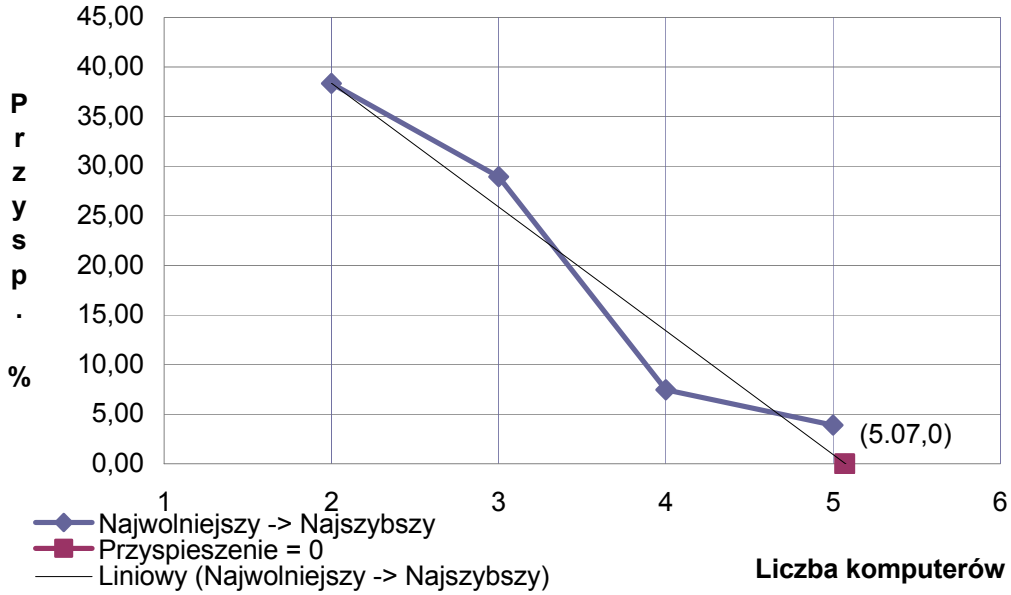


Rys. 5. Przyspieszenie problemu pierwszego w miarę dodawania coraz szybszych maszyn
Fig. 5. First problem acceleration each time adding a faster machine.

Problem drugi (250×20 wektorów, 5 klasterów, 100000 iteracji), jak widać w tabeli 1, dał podobne wyniki. Znów RPC przyspiesza znacznie wykonywanie zadania. Na rys. 6 i 7 przedstawiono wykres czasu i przyspieszenia względem liczby komputerów w sieci.



Rys. 6. Czas wykonywania problemu drugiego w miarę dodawania coraz szybszych maszyn
Fig. 6. Second problem execution time each time adding a faster machine



Rys. 7. Przyspieszenie problemu drugiego w miarę dodawania coraz szybszych maszyn
Fig. 7. Second problem acceleration each time adding a faster machine.

Algorytm RPC, dla tego problemu, przy 5 komputerach jest 2.61 razy szybszy od algorytmu sekwencyjnego wykonanego na najszybszym komputerze w całym zestawie. Może się wydawać, że przyspieszenie powinno być większe, jednak trzeba pamiętać o tym, że pozostałe komputery były wolniejsze (czasami dużo wolniejsze) niż najszybsza jednostka.

Ostatni test, jako że był bardzo czasochłonny, został wykonany tylko na najszybszej maszynie oraz na wszystkich komputerach przy użyciu RPC. Porównanie wyników daje bardzo podobny rezultat do wyniku otrzymanego z problemu drugiego. Algorytm RPC, dla

problemu trzeciego, przy 5 komputerach okazał się być 2.65 razy szybszy w porównaniu do algorytmu sekwencyjnego wykonanego na najszybszej maszynie.

BIBLIOGRAFIA

1. Bloomer J.: Power programming with RPC. O'Reilly, 1992.
2. Soktys F. G.: Inside the AS/400 F, Twenty Ninth Street Press 1996.
3. Schimunek G., Dupuche D., Fung T., Kirkdale P., Myhra E., Stein H.: Slicing the AS/400 with Logical Partitioning: A How to Guide, IBM RedBooks, 1999.
4. Stąpor K.: Automatyczna klasyfikacja obiektów. Exit, 2005.
5. Kozielski S.: Systemy umożliwiające realizację algorytmów równoległych w sieciach komputerowych. Praca zbiorowa pod redakcją Stanisława Kozielskiego. Politechnika Śląska, Skrypty uczelniane Nr 1975, Gliwice 1996.
6. Czech Z.: Programowanie współbieżne, wybrane zagadnienia. Praca zbiorowa pod redakcją Zbigniewa Czecha. Politechnika Śląska, Skrypty uczelniane Nr 1931, Gliwice 1995.
7. Gabassi M., Dupouy B.: Przetwarzanie rozproszone w systemie Unix. LUPUS, Warszawa 1995.
8. Stevens W. R.: Programowanie zastosowań sieciowych w systemie Unix. Wydawnictwo Naukowo-Techniczne, Warszawa 1995.
9. Zieliński K.: Środowiska programowania rozproszonego w sieciach komputerowych. Praca zbiorowa pod redakcją Krzysztofa Zielińskiego, Kraków 1994.
10. Wiess Z., Grużlewski T.: Programowanie współbieżne i rozproszone. Wydawnictwo Naukowo-Techniczne, Warszawa 1993.

Recenzent: Dr inż. Joanna Domańska

Wpłynęło do Redakcji 13 sierpnia 2009 r.

Abstract

In the above article it was briefly described how *Remote Procedure Call* (RPC) mechanisms work and how they can be used to solve problems in a parallel manner. The way of communication between server and clients was discussed as well as the XDR block structure presented.

RPC was further used to present means of integration of multiple operating systems together to solve one common problem faster. The problem was Fuzzy isoData clustering. It is an iteration algorithm used to divide large set of vectors into smaller similar groups. The exact algorithm procedure with all formulas was also presented.

Having the distributed, parallel algorithm ready it was possible to compare time complexity of different computer clusters. Different computers with different operating systems were used. The improvement was very high while each time adding a faster computer to the network what seemed to be obvious. More interesting is the fact that addition of much slower node still resulted in a small improvement of performance.

A common observation, no matter what order of computer attachment was used, is that while adding new machines into the network there is always a border when addition of further computers becomes unprofitable.

Adresy

Hafed ZGHIDI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16,
44-100 Gliwice, Polska, hafed.zghidi@polsl.pl.
Adam ŚWITOŃSKI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16,
44-100 Gliwice, Polska, adam.switonski@polsl.pl.
Błażej ADAMCZYK: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16,
44-100 Gliwice, Polska, blazej.adamczyk@gmail.com.