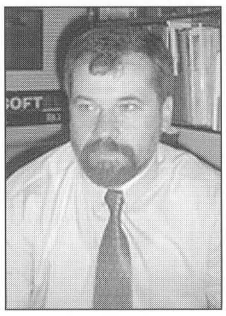
Kazimierz LAL, Tomasz RAK
POLITECHNIKA RZESZOWSKA, KATEDRA INFORMATYKI I AUTOMATYKI

Budowa i strojenie klastra komputerowego Mosix-Linux

Dr inż. Kazimierz LAL

Ukończył studia na Wydziale Elektroniki Politechniki Wrocławskiej w 1981 r. W 1989 uzyskał stopień doktora nauk technicznych na Wydziale Elektroniki Politechniki Gdańskiej. Obecnie pracuje w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Jego praca i zainteresowania naukowe związane są z architekturami sieciowymi i protokołami komunikacyjnymi.

e-mail: k.lal@softel.pl



Mgr inż. Tomasz RAK

Absolwent Wydziału Elektrycznego Politechniki Rzeszowskiej (1999, specjalność: sieci komputerowe). Obecnie pracuje jako asystent w Katedrze Informatyki i Automatyki PRz. Jego zainteresowania naukowe dotyczą modelowania i zastosowań sieciowych systemów komputerowych.

e-mail: trak@prz-rzeszow.pl

Streszczenie

Trwający od wielu lat rozwój architektur procesorów przeznaczonych dla komputerów personalnych (PC) oraz znaczny wzrost wydajności interfejsów lokalnych sieci komputerowych doprowadził do stanu, gdy możliwe jest budowanie tanich i jednocześnie wydajnych systemów wielokomputerowych - klastrów.
Praca omawia problematykę konfigurowania i strojenia (tzw. tuningu) klastrów komputerowych, bazujących na komputerach PC, sieci Ethernet i oprogramowaniu Mosix-Linux. Uzyskane rezultaty dowodzą, że testowane systemy dojrzały do szerszego stosowania w dydaktyce i laboratoriach naukowych.

Abstract

The development of processor architectures designated for personal computers (PC) and also the considerable increase in the efficiency of local computer network interfaces resulted in the possibility of constructing cheap and efficient multicomputer systems - clusters.
The paper discusses a problem of configuring and tuning computer clusters based at PCs, Ethernet, and Mosix Linux software. Final results prove that tested systems are matured to broad using in teaching and in research laboratories.

Słowa kluczowe: klaster, obliczenia równoległe, strojenie, MPI, PVM.

Keywords: cluster, parallel computation, tuning, MPI, PVM.

1. Wstęp

Dynamiczny rozwój architektur oraz wzrost wydajności procesów przeznaczonych do zastosowań w komputerach klasy PC przy jednocześnie ich niskiej cenie spowodowały, że projektanci najwydajniejszych maszyn cyfrowych zaczęli wykorzystywać popularne procesory w swoich superkomputerach [1]. Obecnie, również rozwój lokalnych sieci komputerowych, a w szczególności gigabitowego Ethernetu, staje się zaczątkiem, pozwalającym na budowę relatywnie tanich oraz wydajnych klastrów komputerowych, stanowiących dobrą alternatywę dla klasycznych superkomputerów. Klastry znajdują zastosowanie w średnich i dużych przedsiębiorstwach oraz instytucjach naukowych - wszędzie tam, gdzie finanse są parametrem krytycznym [2].

Klaster komputerowy to nie tylko połączone ze sobą fizycznie komputery, ale przede wszystkim odpowiednie oprogramowanie systemowe i użytkowe. Jednym z ciekawych projektów, wspierających budowę omawianych klastrów jest Mosix [3]. Cechą charakterystyczną wspomnianego systemu - swoistej „łaty" na jądro - są adaptacyjne (zdecentralizowane) algorytmy monitorujące i „wyrównujące" wykorzystanie zasobów poprzez migrację procesów do węzłów mniej obciążonych i bardziej predestynowanych do obsługi konkretnych procesów. Tak więc Mosix upodabnia klaster do maszyny SMP (ang. Symmetric Multi Processing), ale do - podkreślmy - maszyny relatywnie łatwo skalowalnej. Każdy węzeł klastra jest niezależny w sensie fizycznym, poprawne (nie awaryjnie) wyłączenie węzła z zasady nie prowadzi do awarii innych węzłów i nie zakłóca pracy przetwarzanych zadań. Po prostu zadania w takim przypadku są przenoszone na inne węzły - gdzie są kontynuowane. Ważną cechą systemu Mosix jest udostępnianie zasobów w trybie SSI (ang. Single System Image). Aplikacje pracujące pod kontrolą Mosixa nie muszą być „świadome", że są przetwarzane w wielokomputerowym środowisku równoległym [4]. W szczególności proces aplikacji przeniesiony na inny węzeł, pracuje bez wiedzy o tym fakcie. Co najważniejsze, nie stwarza to w zasadzie żadnych zakłóceń w jego pracy. Cecha taka powoduje, że nawet uruchomienie aplikacji sekwencyjnej w systemie Mosix może przynieść przyspieszenie jej pracy, jeśli dostępne są wydajniejsze węzły niż węzeł domowy. Omawiany system nie narzuca praktycznie żadnych wymagań na proces tworzenia aplikacji. Twórca nie musi „wiązać" swojego programu z żadnymi dodatkowymi bibliotekami, nie ma konieczności zmian w strukturze programu. System Mosix umożliwia dodatkowo uruchamianie programów dla środowisk PVM (ang. Parallel Virtual Machine) i MPI (ang. Message Passing Interface) (po zainstalowaniu tych środowisk przynajmniej na maszynie, na której program będzie uruchamiany). W trakcie pracy Mosix zbiera informacje o stopniu wykorzystywania zasobów i o aktywnych procesach. Na podstawie takich informacji podejmuje decyzje o ewentualnym przeniesieniu konkretnego procesu na konkretny węzeł. Dla aplikacji wieloprocesowych omawiany system zapewnia poprawną pracę nawet jeśli wykorzystują one mechanizmy IPC (ang. Inter Process Communication), ale z wyłączeniem komunikacji za pośrednictwem pamięci wspólnej. Dostępne są wersje Mosixa, pracujące wraz z systemem operacyjnym Linux, różnymi wersjami Unixa oraz BSD. Akceptuje on sprzęt, bazujący na platformach x86. Mogą to być jednoprocesorowe stacje robocze, ale również maszyny SMP. Typowe wykorzystywane połączenia to sieci LAN. Klaster Mosix może być „budowany" zarówno jako klaster komputerów PC, połączonych siecią Ethernet 100Mb/s, jak i klaster o wysokich osiągach złożony z serwerów SMP klasy Pentium-Xeon, połączonych wysokoprzepustową siecią gigabitową np. Myrinet, Gigabit Ethernet.

2. Strojenie klastra Mosix-Linux

Po zainstalowaniu i skonfigurowaniu klastra Mosix konieczne jest jego dostrojenie (ang. tuning). Operacja taka jest zalecana, ponieważ Mosix po instalacji działa ze standardowymi parametrami, które nie zawsze odzwierciedlają aktualną wydajność komponentów zbudowanego systemu. Optymalną

wydajność można osiągnąć jedynie poprzez dokładne określenie parametrów poszczególnych węzłów. Do tego celu służą polecenia tune i mtune.

Program tune mierzy czasy operacji jądra systemu w Mosixie, natomiast program mtune mierzy czasy operacji wykonywanych na systemie plików MFS (ang. Mosix File System). Na parametry strojenia wpływają następujące elementy:
- typ procesora,
- typ i szybkość pamięci RAM,
- liczba stanów oczekiwania pamięci RAM (oddzielnie dla odczytu i zapisu),
- użyty sprzęt sieciowy (rodzaj i szybkość karty sieciowej, itp.),
- rodzaj użytego jądra - standardowy lub skonfigurowany z opcją SMP.

Program tune jako wynik swojego działania tworzy dwa pliki:
- dscost.h - plik ten może zostać włączony do źródeł jądra. Parametry w nim zawarte zostaną użyte podczas kompilacji jądra systemu jako domyślne wartości dla danego węzła klastra.
- mosix.costs - plik zawiera 14 liczb całkowitych, które są parametrami służącymi do zoptymalizowania pracy jądra systemu. Jego zawartość należy przenieść do pliku /proc/mosix/admin/overheads, aby nadpisać domyślne ustawienia Mosixa. Jeśli przy następnym uruchomieniu węzła mają być użyte parametry uzyskane w wyniku strojenia, wówczas plik mosix.costs należy przekopiować do katalogu /etc ze zmienioną nazwą na overheads (/etc/overheads).

Program mtune także zwraca wynik w postaci dwóch plików:
- mfscosts.h - plik ten podobnie jak dhosts.h może zostać włączony do źródeł jądra. Parametry w nim zawarte zostaną użyte podczas kompilacji jądra systemu jako domyślne wartości dla danego węzła klastra.
- mfs.costs - plik zawiera 6 liczb całkowitych. Wartości te służą do zoptymalizowania pracy systemu plików MFS. Zawartość omawianego pliku należy przenieść do /proc/mosix/admin/mfscosts, aby nadpisać domyślne ustawienia Mosixa. Jeśli przy kolejnym uruchomieniu węzła mają być użyte parametry uzyskane w wyniku strojenia, wówczas mfs.costs należy skopiować do katalogu /etc ze zmianą nazwy na mfscosts (/etc/mfscosts).

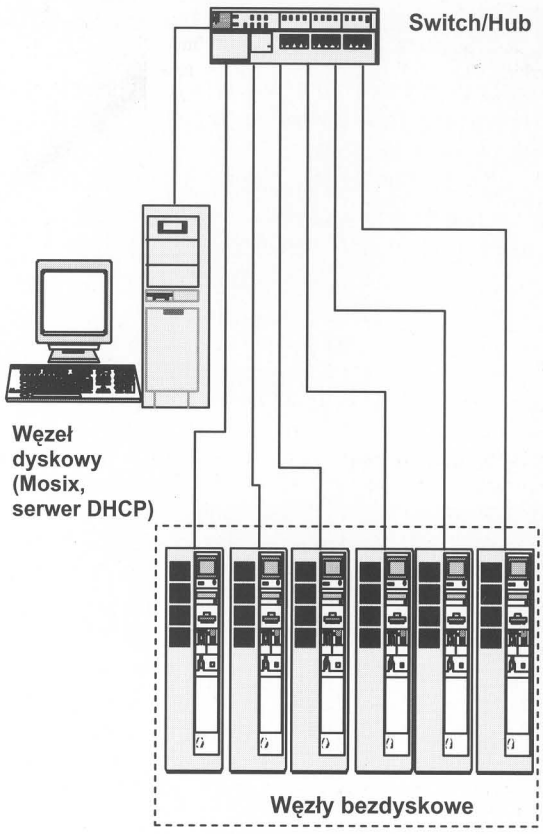
W praktyce do przeprowadzenia operacji strojenia służy skrypt tune_kernel.

3. Budowa, konfiguracja i strojenia klastra

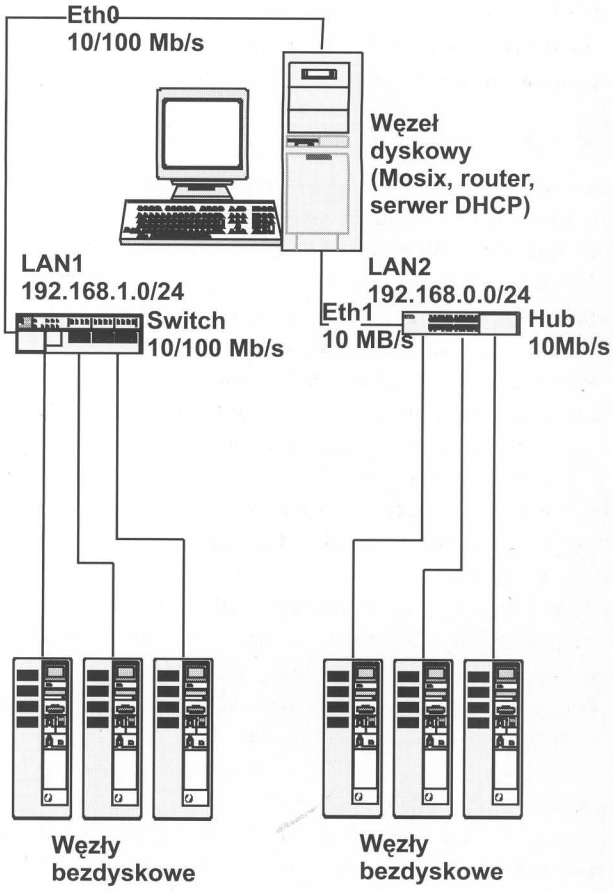
Do budowy testowego klastra wykorzystano komputery z procesorami Pentium IV 1,7 GHz 256 kB cache, 256 MB pamięci RAM, kartami sieciowymi firmy 3Com 10/100 Mb/s (w tym jeden z dyskiem twardym, na którym była utworzona partycja Swap Linuxa) [5]. Korzystano z następujących urządzeń sieciowych:
- Hub Longshine 10Mb/s 8 portów,
- Hub Compex TP1008C 10Mb/s 8 portów,
- Switch Compex PS2216 10/100 Mb/s 16 portów,
- Switch Canyon 10/100 Mb/s 5 portów.

3.1. Klaster jednosegmentowy

Klaster Mosix zbudowany został z 7 komputerów PC klasy Intel Pentium IV, połączonych w sieć komputerową o przepustowości 100Mb/s (do testów zestawiono również sieć 10Mb/s) (rys. 1). Na jednym z komputerów została zainstalowana pełna wersja Linuxa Aurox 9.2 wraz z odpowiednimi narzędziami (w tym serwerem DHCP (ang. Dynamic Host Configuration Protocol)), natomiast na pozostałych węzłach system był uruchamiany z płyty CD za pomocą przygotowanej przez autorów minidystrybucji Linuxa. Na każdym węźle umieszczono taki sam plik mosix.map. Podczas przeprowadzania strojenia klastra skorzystano z opcji optymalizacji dla złożonej topologii-sieci.



Rys. 1. Jednosegmentowy klaster testowy
Fig. 1. One-segment test cluster



Rys. 2. Dwusegmentowy klaster testowy
Fig. 2. Two-segment test cluster

3.2. Klaster dwusegmentowy z routerem

W celu przeprowadzenia testów klastra podzielonego na dwa segmenty, zbudowano sieć składającą się z dwóch podsieci LAN1 i LAN2, połączonych między sobą routerem, będącym jednocześnie węzłem klastra (rys. 2). Dodatkowo na routerze zainstalowany i skonfigurowany został serwer DHCP dla dwóch oddzielnych podsieci.

Podczas przeprowadzania strojenia klastra skorzystano z opcji optymalizacji dla złożonej topologii-sieci. Wykonano następującą serię pomiarów:
- router względem grupy komputerów z sieci LAN1,
- router względem grupy komputerów z sieci LAN2,
- komputery z sieci LAN1 względem siebie,
- komputery z sieci LAN1 względem routera,
- komputery z sieci LAN2 względem siebie,
- komputery z sieci LAN2 względem routera,
- komputery z sieci LAN1 względem komputerów z sieci LAN2.

4. Testy klastra Mosix

Badania przeprowadzone w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej (Laboratorium Sieci Komputerowych) miały na celu sprawdzenie:
- poprawności pracy klastra Mosix - podstawowe testy dla wieloprocesowej aplikacji mające uwidocznić przyspieszenie wykonywania takich aplikacji,
- wydajności aplikacji, pracujących w środowiskach PVM i MPI (nabudowanych na klaster Mosix),
- wydajności klastra w przypadku wykorzystania sieci bazujących na koncentratorach oraz przełącznikach,
- wpływu strojenia na wydajność klastra jednosegmentowego oraz dwusegmentowego.

4.1. Test programem forkbomb

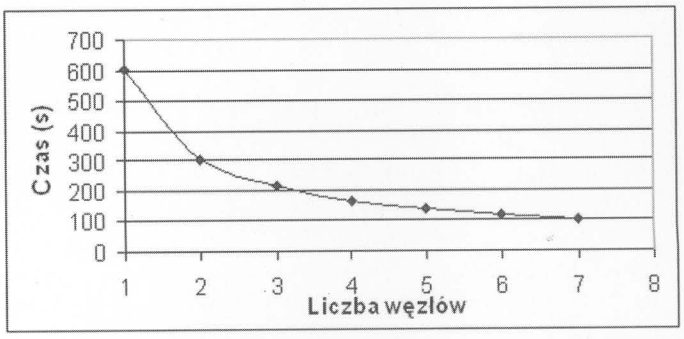
Celem testu było sprawdzenie poprawności funkcjonowania klastra [6]. Pomiary wykonano zarówno dla klastra dostrojonego, jak i dla klastra przy ustawieniach domyślnych (bez strojenia).

Wyniki dla klastra nie dostrojonego są niemal identyczne. Wstępnie otrzymane rezultaty można tłumaczyć:
- domyślną optymalizacją testowej wersji klastra Mosix dla sieci Fast Ethernet (podstawowa przyczyna),
- brakiem alokacji pamięci,
- prostotą programu (brak tablic i struktur),
- brakiem dodatkowej komunikacji miedzy procesami,
- homogenicznością klastra.

4.2. Aplikacje środowisk PVM i MPI a klaster Mosix

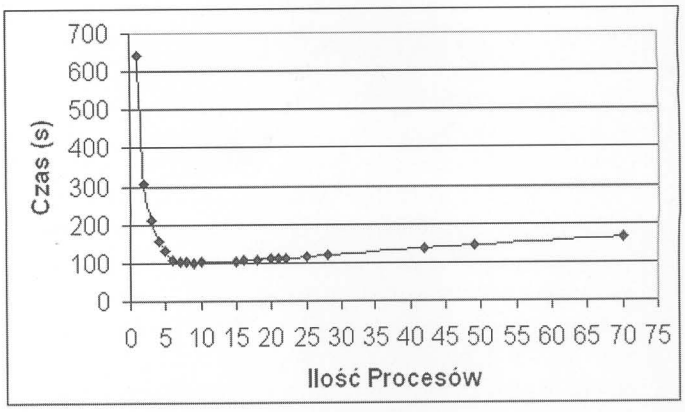
Celem testów było sprawdzenie wydajności klastra Mosix w przypadku wykonywania aplikacji napisanych dla środowisk PVM i MPI. Podkreślmy - zarówno dla PVM jak i MPI testy były identyczne.

Badany system zawierał jeden węzeł centralny i sześć bezdyskowych węzłów dodatkowych. Nie wykonano strojenia klastra. Instalacja PVM w środowisku Mosix polega na instalacji i konfiguracji jednego demona. Jego zadaniem jest uruchamianie programów i tworzenie procesów. Wszystkie działania związanie z migracją procesów na inne maszyny i z wyrównywaniem obciążeń wykonuje Mosix. Pierwszy z pomiarów w środowisku PVM polegał na ocenie zmiany wydajności wraz ze wzrostem liczby węzłów klastra (rys. 3). Użyty do testów program POVRay PVM tworzy 10 procesów potomnych. Jego zadaniem jest rendering pliku obraz.pov.



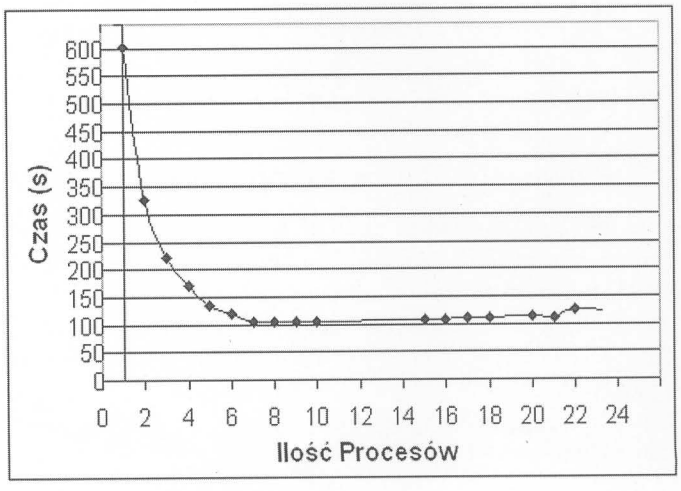
Rys. 3. Czas wykonania programu dla różnej liczby węzłów
Fig. 3. Processing program time for different nodes number

Kolejny test dla środowiska PVM polegał na wykonaniu renderingu obrazu (rys. 4). Badano czas wykonania zadania w zależności od liczby procesów przypadających na zadanie. W klastrze pracowało 7 węzłów (1 dyskowy i 6 uruchamianych z CD).



Rys. 4. Czas wykonania zadania w funkcji liczby procesów, przypadających na zadanie
Fig. 4. Processing job time in function of processes number corresponded to job

Kolejny test przeprowadzono w środowisku MPI. Program POVRay wykonywał rendering pliku obraz.pov. Zmianie ulegała liczba procesów, przypadających na zadanie (rys. 5).



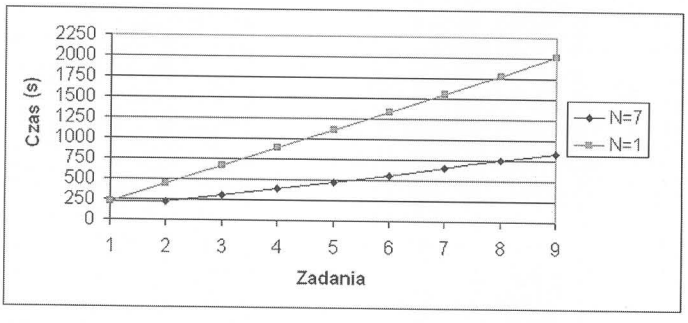
Rys. 5. Czas wykonania zadania w funkcji liczby procesów przypadających na zadanie (środowisko MPI)
Fig. 5. Processing job time in function of processes number corresponded to job (MPI environment)

Wykonując jedno zadanie najlepszy wynik uzyskano przy rozbiciu na 7 lub 8 procesów (rys. 5). Przy większej liczbie procesów szybkość wykonania zadania ulega stopniowemu pogorszeniu. Należy zwrócić uwagę, że niemal identyczne charakterystyki występują w przypadku analogicznego testu w środowisku PVM (rys. 4).

Wpływ na pogorszenie skuteczności wiąże się m.in. z:
- Stopniowym podziałem zadania na coraz mniejsze bloki - co do pewnego momentu przynosi dobre efekty, ułatwiając zrównoleglenie prac nad zadaniem, lecz w pewnym momencie dochodzi do zachwiania właściwych proporcji pomiędzy czasem wykonywania obliczeń a czasem poświęconym na wywołanie procesu potomnego oraz jego przesłaniem na inny węzeł (i późniejszym odbiorem).
- Wtórną migracją procesów pomiędzy węzłami - po opuszczeniu węzła domowego i przeniesieniu na inne węzły dochodzi do dalszych przypadków przeniesienia. Części danych dla różnych procesów nie mają identycznej złożoności. W przypadkach, gdy jest wiele zadań do wykonania, migracja taka jest pożyteczna. Odbija się to niewielkimi stratami na czasie wykonania.
- Opóźnieniami czasowymi, wynikającymi z faktu migracji zadań.

Celem kolejnego testu było zbadanie charakterystyki pracy klastra w taki sposób, aby na wyniki nie miał wpływu podział zadania na mniejsze części.

W eksperymencie wzrasta nie liczba procesów w zadaniu, lecz sama liczba zadań. Każde zadanie jest dodatkowo podzielone na 3 procesy (liczba ta nie ulega zmianie w przeprowadzonym teście). Klaster pracuje bez strojenia. Liczba węzłów wynosi 7. Sieć pracuje na bazie przełącznika Ethernet 100TX. Test polega na wykonaniu skryptu uruchamiającego od 1 do 9 zadań jednocześnie w tle. Wskazywany w niej czas pracy jest więc czasem całkowitego wykonania zadań zleconych w skrypcie (start następuje w niemal tym samym momencie). Skrypt dodatkowo ustawia parametry pracy programu POVRay (rozdzielczość obrazu wyjściowego itp.). W taki sposób każde zadanie pracuje nad innym plikiem (lecz w praktyce, dla uproszczenia, każdy z tych plików zawiera identyczną scenę do renderingu) (rys. 6).



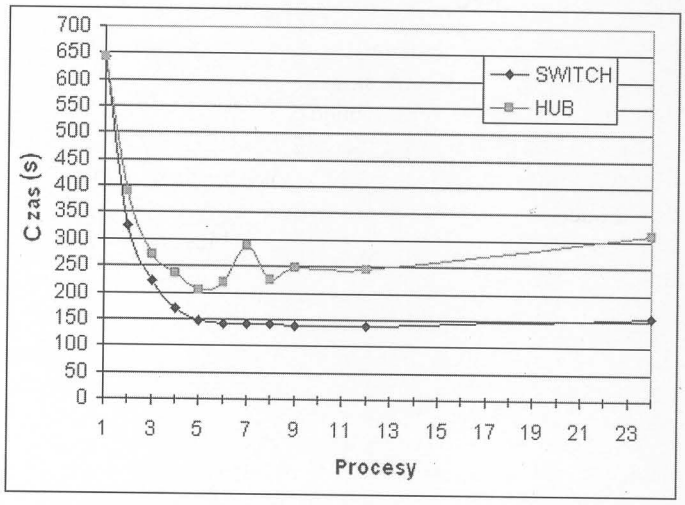
Rys. 6. Czas wykonania w zależności od liczby zadań (N liczba węzłów)
Fig. 6. Processing time depending on jobs number (N nodes number)

4.3. Test wydajności sieci dla koncentratorów i przełączników

Test polegał na wykonaniu renderingu w programie POVRay MPI (zmienna ilość procesów).

Po wykonaniu kilku serii pomiarów przełącznika wykonano takie same pomiary dla koncentratora. Nie napotkano problemu z przepustowością sieci 10 Mb (natężenie ruchu nie przekraczało tej granicy). Testy wykazały zupełną nieprzydatność koncentratorów jako elementów struktury połączeń pomiędzy węzłami klastra, spowodowaną kolizjami (rys. 7).

Straty przy zastosowaniu koncentratora wahały się od 52% (4 procesy) do 99% (24 procesy). Spadek efektywności dla dużej liczby procesów wiąże się m.in. z podziałem zadania na zbyt małe części (rys. 7).



Rys. 7. Czas wykonania wieloprocesowego zadania w przypadku zastosowania przełącznika i koncentratora
Fig. 7. Processing time of multiprocess job in case of using switch and hub

4.4. Podsumowanie

Podsumowując - wzrostowi liczby węzłów oraz liczby procesów do wykonania na systemie klastrowym musi towarzyszyć przejście na urządzenia sieciowe, które tworzą domeny bezkolizyjne. Przełączniki pracujące w standardzie Fast Ethernet są dobrym rozwiązaniem dla niewielkich klastrów również dlatego, że często oprogramowanie zarządzające jest wstępnie skonfigurowane dla takiej sieci [7]. W sytuacji wykonywania zadań polegających na przetwarzaniu dużej ilości danych zgromadzonych w dużych plikach, opłacalne jest zastosowanie Gigabit Ethernetu i strojenia klastra.

5. Zakończenie

W pracy omówiono wyniki badań nad wykorzystaniem ogólnie dostępnych popularnych komputerów PC do budowy klastrów obliczeniowych. W pierwszym etapie przetestowano różne konfiguracje systemowe klastra Mosix-Linux. Później omówiono integrację Mosixa ze środowiskami PVM oraz MPI i podkreślono zalety takiej konfiguracji. Na koniec przeprowadzono testy wydajności klastrów po wykonaniu tzw. strojenia (tuningu). Obecnie realizowane działania zmierzają do wdrożenia różnych algorytmów kolejkowania zadań i mechanizmów zwiększania odporności na awarie.

6. Literatura

[1] Informacje internetowe o superkomputerach: www.idg.pl/news/68760.html
[2] McClure S., Wheeler R.: Mosix-How Linux Clusters Solve Real Word Problems, Proc. 2000 USENIX Annual Tech. Conf., pp. 49-56, San Diego, 2000.
[3] Barak A., Guday S., Wheeler R.: The Mosix Distributed Operating System, Load Balancing for UNIX, Lecture Notes in Computer Science, vol. 672, Springer-Verlag, 1993.
[4] Amar L., Barak A., Shiloh A.: The Mosix Parallel I/O System for Scalable I/O Performance, Proc. 14th IASTED, pp. 495-500, 2002.
[5] Kozioł M., Kudłacz M.: Instalacja konfiguracja klastra Mosix, KIA, PRz, 2004.
[6] Strona domowa programu forkbomb: home.tiscali.cz:8080/~cz210552/forkbomb.html
[7] Korespondencja z prof. Amnonem Barakiem z The Hebrew University of Jerusalem - koordynatorem projektu Mosix.

Title: Structure and Tuning of Mosix-Linux Computer Cluster

Artykuł recenzowany