ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1996

Seria: INFORMATYKA z. 30 Nr kol. 1315

Henryk KRAWCZYK
Marek TARGOWSKI
Piotr BRUDŁO

PAKIET PROGRAMOWY *NETMON* MONITORUJĄCY OBCIĄŻENIE WĘZŁÓW KOMPUTEROWEJ SIECI LOKALNEJ *

Streszczenie. Realizacja rozproszonego przetwarzania zadań w sieci lokalnej wymaga informacji o bieżącym obciążeniu węzłów sieci. W pracy przedstawiono metodę pomiaru takiego obciążenia. Zaproponowano również pakiet monitorowania uśrednionego obciążenia węzłów sieci. Pakiet ten o nazwie NetMon składa się z dwóch głównych warstw: testowania i propagacji wyników pomiarów. Implementacji pakietu dokonano w języku C na stacjach roboczych Sun Classic z systemem SunOS 5.x i Open Windows 3.x. Wykorzystano sockety jako mechanizmy komunikacyjne. NetMon wyposażono w przyjazny interfejs zgodny ze standardem Open Look.

NETMON AS SOFTWARE PACKAGE FOR MONITORING CURRENT PROCESSING LOAD OF LAN NODES

Summary. Realisation of distributed processing of tasks in local area network needs information about current processing load of LAN nodes. The method of the processing load determination is presented. Software package for monitoring of an average processing load of LAN nodes is also proposed. NetMon consists of two main layers: a load testing layer and a results propagation layer. The implementation of the packet is done in the C language for Sun Classic workstations with the SunOS 5.x operating system and the Open Windows 3.x graphical environment. The sockets are

* praca w ramach grantu KBN nr: 8 S503 017 07

used as communication mechanism. This software packet contains a user-friendly graphical interface compatible with the Open Look standard.

PAKETPROGRAMM NETMON ZUR ÜBERWACHUNG DER LAST VON KNOTEN IM LAN-NETZE

Zusammenfassung. Die Verwirklichung der verteilten Verarbeitung im lokalen Netz bedarf Kenntnisse über aktuelle Lasten von Knoten der Netze. Im Aufsatz wird eine Methode der Messung solchen Lasten dargestellt. Man schlägt auch die Verwendung des Programmpakets NetMon zur Überwachung der Mittellast eines Knotens vor. Das Paket besteht aus zwei Hauptebenen: Überwachung und Ausbreitung von Meßergebnisse. Das Paket wurde unter SunOS 5.x und Open Windows 3.x betriebene Workstation Sun Classic realisiert: bei der Kommunikation von Rechnern werden Sockets verwendet. NetMon hat ein userfreundliches Interface, dem Standard Open Look gemäß.

1. Wprowadzenie

W dobie daleko zaawansowanych technologii przetwarzania zaobserwować można rosnące zainteresowanie systemami rozproszonymi [10]. Jest ono spowodowane głównie potrzebami:

- efektywnego wykorzystania posiadanych zasobów komputerowych,
- zwiększenia szybkości pracy poprzez równoległe wykonywanie obliczeń,
- zapewnienia niezawodności przetwarzania danych poprzez np. wykonywanie obliczeń na kilku maszynach i porównywaniu otrzymanych wyników,
- minimalizacji kosztów,
- elastyczności architektury umożliwiającej rozbudowę bądź zmianę istniejącej konfiguracji.

Lokalne sieci komputerowe, oprócz tradycyjnych zastosowań, coraz częściej są adaptowane do wykonywania obliczeń równoległych i rozproszonych [1]. Naturalną tendencją jest integracja logiczna fizycznie rozproszonych zasobów sieci w celu uzyskania środowisk o dużych mocach obliczeniowych przy wielokrotnie niższych nakładach [5]. Tym samym sieć komputerowa może realizować swoje tradycyjne funkcje bądź wspomagać przetwarzanie wymagające dużych mocy obliczeniowych.

Przy logicznej integracji rozproszonych zasobów sieci lokalnej istotne staje się monitorowanie rozproszonego stanu węzłów. Może ono dotyczyć zarówno bieżącej wydajności węzła, jak i jego aktualnej sprawności funkcjonowania. Na podstawie znajomości stanów węzłów sieci można wprowadzić mechanizmy zapewniające równomierny rozkład obciążenia na poszczególne węzły, decydujące o sposobie ich wykorzystania, kontrolowania konfiguracji, itp. Istnieje kilka znanych, ogólnie dostępnych implementacji systemów rozproszonych: PVM (*Parallel Virtual Machine*) [5] wraz z HeNCE (*Heterogeneous Network Computing Environment*) [6], Linda, Strand, HP-Task Broker, Amoeba. Tworzą one warstwę oprogramowania nałożoną na środowisko systemowe i na warstwę komunikacji sieciowej. Spełniają rolę integrującą zasoby sieci poprzez utworzenie zintegrowanej maszyny wirtualnej. Podstawową wadą tych systemów jest jednak to, że nie uwzględniają w pełni dynamiki rzeczywistych obciążeń stacji roboczych, co jest jedną z przyczyn nie zawsze efektywnego wykorzystania mocy obliczeniowych.

Najbardziej rozpowszechnionym systemem jest częściowo niezależny od sprzętu PVM. System ten dostarcza biblioteki funkcji umożliwiające wykorzystanie sieci lokalnej stacji roboczych z systemem Unix jako scentralizowanej maszyny wirtualnej. Komunikację i synchronizację zadań w ramach całego systemu uzyskuje się włączając wywołania funkcji PVM do kodu źródłowego programu. Zarządzaniem maszyną wirtualną zajmują się demony PVM, po jednym w każdej stacji roboczej, wchodzącej w skład maszyny wirtualnej. Dzięki systemowi PVM możliwe jest między innymi wykonywanie programu równoległego na wielu procesorach równocześnie. Sam PVM częściowo wyręcza użytkownika przy dzieleniu programu na procesy oraz przy ich przypisywaniu do konkretnych stacji. Celem projektowym PVM było zapewnienie środowiska do przetwarzania rozproszonego w sieci lokalnej i cel ten został w pełni osiągnięty, natomiast niedogodnościami tego systemu w jego obecnej wersji są:

- brak kontroli rzeczywistego zrównoważenia obciążenia wszystkich stacji składowych maszyny wirtualnej,
- brak automatyzacji procesu rozdziału programów na procesy i ich efektywnej alokacji.

Na bazie systemu PVM zbudowany został system HeNCE. Wprowadza on interfejs graficzny, ułatwiający definiowanie równoległości i przepływu sterowania pomiędzy procesami. W efekcie powstaje skierowany graf obliczeń, który następnie może być wykonany przez HeNCE. System HeNCE dziedziczy wszystkie funkcje komunikacyjne i synchronizacyjne dostępne w PVM. HeNCE nie uwzględnia jednak rzeczywistego obciążenia stacji w sieci lokalnej. W fazie definiowania grafu obliczeń użytkownik w sposób jawny podaje koszty wykonania danego modułu na poszczególnych maszynach i koszty te są jedyną podstawą do *mapping*'u. Przy czym koszty te powinny uwzględniać zarówno możliwości obliczeniowe stacji (w przypadku środowisk heterogenicznych), jak i zawierać przewidywane obciążenia stacji.

Amoeba to system rozproszony powstały na Uniwersytecie *Vrije* w Amsterdamie. Integruje stacje robocze sieci lokalnej Ethernet. Z punktu widzenia użytkownika sprawia wrażenie systemu scentralizowanego, pracującego z podziałem czasu. Użytkownik rejestruje się w systemie jako całości zamiast (jak to ma miejsce w systemie Unix) uzyskania dostępu tylko do konkretnej maszyny. W systemie tym zastosowano podejście obiektowe. Operacje na obiektach są zaimplementowane przy wykorzystaniu zdalnych wywołań procedur wykorzystujących protokół sieciowy FLIP. Jądro systemu, którego kopia znajduje się na każdej maszynie, dostarcza środowiska do wykonania procesów użytkownika oraz procesów systemowych świadczących usługi procesom użytkownika zgodnie z podejściem klient - serwer. Amoeba umożliwia ponadto częściową emulację systemu Unix, zapewnia możliwość połączenia z sieciami rozległymi i dostęp do maszyny wirtualnej z zewnątrz.

HP-Task Broker realizuje przetwarzanie rozproszone w konwencji klient - serwer na poziomie systemowym. Stacja robocza może być skonfigurowana jako stacja klienta, serwera lub pełnić obie funkcje równocześnie. Stacja klienta w przypadku otrzymania zadania do wykonania wysyła zapytania do stacji serwerów w celu zorientowania się o ich aktualnych możliwościach obliczeniowych. Spośród odpowiedzi wybiera stację najmniej obciążoną i jej przydziela zadanie. Powyższy mechanizm jest całkowicie niewidoczny z punktu widzenia użytkownika. Komunikacja opiera się na rozgłaszaniu. Stąd obciążenie komunikacyjne jest silnie zależne od ilości zadań pojawiających się w systemie. Stacje serwerów określają swoje obciążenia obliczeniowe na podstawie inicjalnie zadanych parametrów, uwzględniając tylko obciążenia wprowadzane przez procesy Task Brokera. Całość systemu została zaimplementowana w formie demonów współpracujących z systemem operacyjnym Unix.

W sytemie Linda procesy wykonujące się równolegle wymieniają dane poprzez generowanie, odczyt i konsumowanie podstawowych krotek w skali maszyny wirtualnej.

W systemie Strand procesy komunikują się ze sobą poprzez współdzielone zmienne i są synchronizowane przy dostępie do danych.

Aktualnie istniejące systemy realizujące przetwarzanie rozproszone możemy więc podzielić na dwie grupy. Kryterium podziału jest stosunek danego systemu do monitorowania:

1. Systemy przetwarzania rozproszonego, posiadające tylko warstwę dystrybuującą zadania do wykonania. O dystrybucji danego zadania decydują wartości (wagi) parametrów ustalonych na stałe [5], [6] bądź wykonanie alokacji procesów przez użytkownika. Do tej grupy zaliczyć można: PVM wraz z HeNCE.
2. Systemy przetwarzania rozproszonego posiadające mechanizmy umożliwiające równoważenie obciążenia w obrębie maszyny wirtualnej. O dystrybucji danego zadania decyduje aktualny stan poszczególnych stacji roboczych. Do tej grupy zaliczyć można systemy Linda, Strand, Amoeba, oraz HP Task Broker.

Wszystkie wyżej zaprezentowane systemy, z wyjątkiem HP Task Broker, posiadają następującą wadę: wymagają zmodyfikowania programów, które mają być wykonane w obrębie systemu. Użytkownik musi najpierw w odpowiednich miejscach programu umieścić odpowiednie wywołania funkcji systemowych lub bibliotecznych (Linda, Strand, PVM), bądź "tylko" spróbować skompilować i zlinkować program (Amoeba). W tym drugim przypadku użytkownik nie ma pewności, że zakończy się to sukcesem ze względu na niepełną kompatybilność z systemem Unix.

W przypadku HP Task Brokera podstawową wadą jest wprowadzanie dość dużego obciążenia komunikacyjnego. Stacja której zlecono program do wykonania, wysyła do wszystkich stacji pakiet rozgłoszeniowy (ang. *broadcast*), zawierający zapytanie o bieżące obciążenie. Wówczas wszystkie stacje serwerów dokonują sprawdzenia, ile zadań uprzednio przesłanych im do wykonania jeszcze się nie zakończyło, po czym wysyłają odpowiedzi do stacji klienta. Za każdym razem, kiedy stacja klienta ma do wykonania zadanie, do sieci wprowadzanych jest tyle pakietów, ile jest stacji serwerów. Istotną wadą jest także nieuwzględnianie rzeczywistego obciążenia poszczególnych stacji, a jedynie obciążenia zadaniami rozdystrybuowanymi przez Task Broker.

Prócz warstw monitorowania wbudowanych w systemy przetwarzania rozproszonego istnieją aplikacje badające stan aktualnego obciążenia pojedyńczej stacji roboczej. Przykładami takich aplikacji są Performance Meter i Xload. Tego typu aplikacja wykonuje się na odległej stacji roboczej, przesyłając okresowo stacji użytkownika informacje o aktualnym obciążeniu badanej stacji. Następnie stacja użytkownika wykonuje wizualizację uzyskanych wyników w postaci wykresu wartości obciążenia względem czasu. Jak widać, monitorowanie ograniczone jest do jednej tylko stacji. Oczywiście można sobie wyobrazić taką sytuację, że użytkownik uruchamia wiele aplikacji - tyle, ile stacji roboczych chce monitorować. Wówczas:

- wzrasta obciążenie łączy komunikacyjnych - jest wprost proporcjonalne do liczby uruchomionych aplikacji,
- wzrasta obciążenie stacji użytkownika, gdzie skupiają się wszystkie logiczne "nitki" komunikacji - liczba dodatkowych procesów obciążających system (procesor, pamięć) jest równa liczbie kopii tej części aplikacji monitorującej, która odbiera dane od stacji odległych i wykonuje wizualizację.

Co więcej, jest to podejście scentralizowane, ponieważ jedynie wyróżniona stacja użytkownika (klient) otrzymuje informacje o obciążeniach pozostałych stacji roboczych (serwery). Celem niniejszej pracy jest przedstawienie opracowanego narzędzia o nazwie NetMon, umożliwiającego na bieżąco monitorowanie całkowitego obciążenia stacji roboczych pracujących w systemie rozproszonym.

2. Architektura systemu NetMon

Na podstawie opracowanego [1,3] modelu przetwarzania rozproszonego skonstruowany został system rozproszonego monitorowania [2,7]. W systemie tym na bieżąco monitorowany jest stan sieci, gdzie sieć stanowi grupa hostów pracujących pod systemem Unix, połączonych siecią LAN.

W opracowanym systemie monitorowania NetMon wyróżnić można dwie warstwy:

- testującą - odpowiadającą za określanie stanu obciążenia lokalnego jednostki,
- komunikacyjną - informującą pozostałe jednostki w sieci o aktualnych obciążeniach obliczeniowych węzłów.

W celu określenia lokalnego obciążenia stacji roboczej wykorzystano metodę aktywnego testowania węzła [2]. Testowanie co prawda obciąża węzeł (aktywny proces testujący), ale metoda ta pozwala wiernie ocenić aktualne możliwości obliczeniowe węzła.

2.1. Konstrukcja testu wydajnościowego

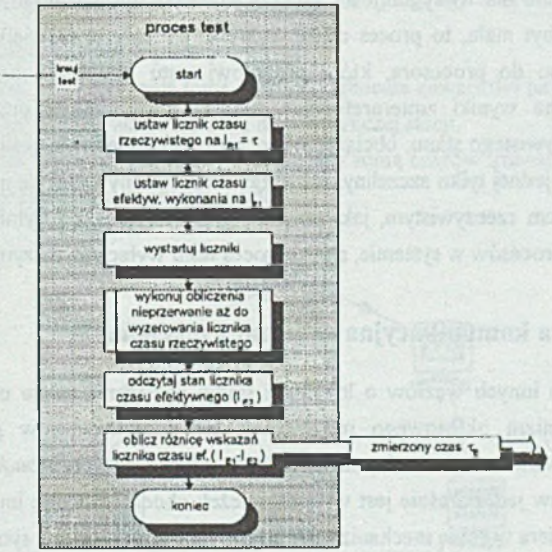
Warstwa testująca jest w zasadzie oddzielnym procesem-testem oznaczanym symbolem *T*. Aby zbadać obciążenie węzła, musi ona określić, jaki procent czasu procesora zostaje przydzielony procesowi testu. Na tej podstawie wyznacza się liczbę procesów aktualnie obciążających procesor. Jeśli procesowi przydzielono np. 33% czasu procesora, oznacza to, że w systemie w momencie wykonywania testu działały jeszcze dwa inne aktywne procesy. W rzeczywistości może to być jeden proces o wyższym priorytecie, lecz nie ma to znaczenia z punktu widzenia obciążenia stacji.

Większość procesów, które można zaobserwować w klasycznych stacjach roboczych (np. SUN czy HP), a są to głównie edytory tekstu, procesy obsługi terminali, procesy typu *shell*, czy programy komunikacyjne, takie jak: poczta elektroniczna, *telnet* czy *ftp*, nie powoduje znaczącego obciążenie procesora. Procesy te są najczęściej uśpione i czekają na wystąpienie określonego zdarzenia (np. naciśnięce przycisku myszy), aby podjąć odpowiednią akcję, wówczas stają się aktywne. Istnienie kilkunastu czy kilkudziesięciu tego typu procesów nie ma większego znaczenia z punktu widzenia obciążenia procesora - w tym przypadku punkt krytyczny stanowi ilość pamięci operacyjnej, jaką dysponuje host. Widoczne obciążenie wprowadzają do systemu jedynie programy aktywne wykonujące obliczenia (do takich zalicza się proces-test), czyli między innymi różnego rodzaju symulatory, jak również kompilatory i programy łączące (ang. *linkers*), także proces-test. Zatem można przyjąć, że **miernikiem obciążenia stacji jest liczba, działających na niej, aktywnych procesów**. Wówczas algorytm testu przyjmuje postać przedstawioną na rys.1. Tuż po kreacji proces zadaje liczniki czasów rzeczywistego i efektywnego, następnie uruchamia liczniki które mierzą czasy:

- czas wykonywania procesu τ_E, tzn. czas, kiedy proces korzysta z procesora,
- czas rzeczywisty τ, jaki upływa, bez względu na to, czy proces korzysta z procesora, czy też oczekuje np. w kolejce procesów gotowych,

po czym rozpoczyna się właściwy test. Wykonywana jest procedura testująca, mająca na celu obciążenie procesora, na którą są nałożone następujące ograniczenia:

- procedura obciążająca wykonuje się przez zadany z góry przedział czasu τ,
- procedura testująca obciążenie nie może być zawieszona z powodu oczekiwania na komunikację z urządzeniami wejścia/wyjścia bądź komunikację z innym procesem.



Rys.1. Algorytm testowania obciążenia stacji roboczej
Fig.1. The algorithm for testing of workstation processing load

Po upływie czasu testowania τ procedura test zostaje przerwana, a następnie wykonywany jest odczyt licznika czasu efektywnego Obliczona zostaje różnica (tzw *delta*) pomiędzy ustawieniem początkowym i końcowym tego licznika. Iloraz rzeczywistego czasu trwania testu i delty czasu wykonywania procesu jest równy liczbie procesów obecnych w systemie (licząc także proces testu), tzn.:

$$n = \frac{\tau}{I_{E1} - I_{E2}} = \frac{\tau}{\tau_E}, \qquad (1)$$

gdzie: n - liczba procesów w systemie,

τ - czas obserwacji,

I_{E1} - wskazanie początkowe licznika czasu wykonywania,

I_{E2} - wskazanie końcowe licznika czasu wykonywania,

τ_E - czas efektywnego wykonywania testu.

Ponieważ wyniki testowania obejmują również proces testujący, liczba aktywnych procesów w systemie powinna być zmiejszona o jeden:

$$n = \frac{\tau}{\tau_E} - 1 \tag{2}$$

Istotne znaczenie dla wiarygodności uzyskanych wyników ma długość czasu testowania [4]. Jeśli będzie zbyt mała, to proces może wykonać się cały w niewielkiej liczbie szczelin czasowych dostępu do procesora, które procesowi testu przydzielił proces zarządzający. Wówczas uzyskane wyniki zinterpretowane zostaną niezgodnie z prawdą zaniżając, w stosunku do rzeczywistego stanu, obciążenie procesora. Przykładowo, jeśli proces wykonałby się cały w obrębie jednej tylko szczeliny, wówczas czas wykonywania się na procesorze byłby identyczny z czasem rzeczywistym, jaki upłynął od startu testu, co byłoby zinterpretowane jako brak innych procesów w systemie, skoro proces testu wyłącznie otrzymał procesor.

2.2. Warstwa komunikacyjna -wariant z tokenem

Powiadamianie innych węzłów o lokalnym stanie obciążenia może odbywać się np. w oparciu o mechanizm okresowego rozgłaszania lub rozgłaszania w przypadku zmiany obciążenia. Stosowanie rozgłaszania, czyli wysłanie informacji typu *broadcast* do wszystkich pozostałych węzłów jednocześnie jest wygodne, jeżeli chodzi o stronę implementacyjną [4]. Natomiast nie zawiera w sobie mechanizmów umożliwiających detekcję sytuacji wyjątkowych, np. awarii lub odłączenia danej jednostki. Przy rozgłaszaniu okresowym można wykrywać ogólną niesprawność (bez możliwości określania przyczyny) jedynie implementując mechanizm *time-out* dla rugowania z systemu tych węzłów, od których inne węzły w określonym limicie czasu nie otrzymały informacji. Alternatywnym rozwiązaniem dla warstwy monitorującej jest proponowana komunikacja pierścieniowa [3], [7]. Stacja po określeniu własnego stanu przesyła o nim informacje do stacji będącej logicznie następną w pierścieniu. W efekcie w pierścieniu krąży okresowo pakiet zawierający stany obciążeń wszystkich stacji. Ten rodzaj komunikacji przedstawia rys.2.

Pakiet składa się z nagłówka zawierającego między innymi informację o długości pakietu oraz rekordów zawierających np. adres IP stacji oraz wartość jej aktualnego obciążenia. Każda stacja P_i (i=1..m, gdzie m jest liczbą wszystkich stacji pierścienia) znajdująca się w pierścieniu opisywana jest przez jeden rekord. Pakiet jest wprowadzany do obiegu w fazie inicjalizacji pierścienia tworzonego na podstawie pliku tekstowego, zawierającego listę

potencjalnie dostępnych węzłów sieci. Oprogramowanie warstwy monitorującej danej stacji roboczej oczekuje na przyjście pakietu. W momencie jego otrzymania oprogramowanie stacji wykonuje lokalnie test T, po czym uaktualnia pole swojego stanu w pakiecie i przesyła pakiet do stacji będącej logicznie następną (P_{i+1}). W tym wariancie test T jest więc wykonywany w danym momencie czasu co najwyżej przez jedną stację. Łatwo wywnioskować, że wprowadzane obciążenie Q_1 do sieci nie jest większe od obciążenia wnoszonego przez pojedyncze zadanie. Ponadto wprowadzane obciążenie jest niezależne od czasu testowania τ, którego wartość jest stała. Zwiększenie (zmniejszenie) czasu τ powoduje szybszy (wolniejszy) okres obiegu pakietu w pierścieniu:

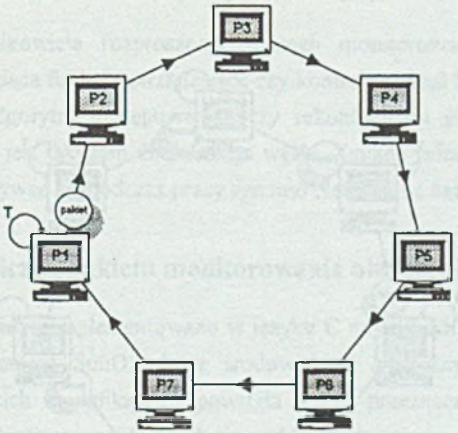
$$\text{czas pojedynczego obiegu pakietu} = m \bullet t + m \bullet \tau + T_N = m \bullet (t + \tau) + T_N \,, \qquad (3)$$

gdzie: m - liczba węzłów w pierścieniu,

t - czas przygotowania testu oraz uaktualnienia zawartości pakietu,

τ - czas wykonywania się testu na pojedynczej stacji,

T_N - czas transmisji pakietu w sieci będący sumą czasów transmisji pomiędzy poszczególnymi parami węzłów.



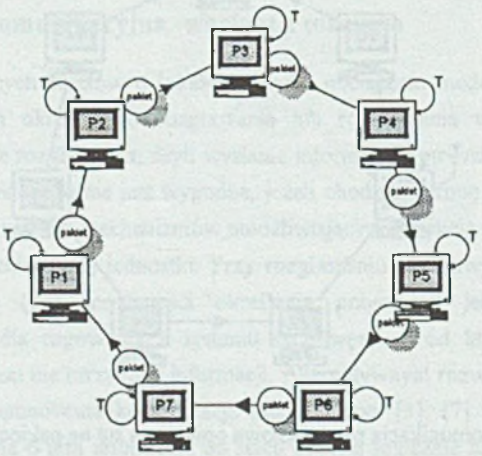
Rys.2. Komunikacja pierścieniowa opierająca sie na pakiecie-tokenie
Fig.2. The communication with a packet-token passing in the ring

Jeśli przyjmiemy, że czasy przygotowania testu oraz transmisji pakietu w sieci są pomijalnie małe (rzędu milisekund), wówczas czas pojedynczego obiegu pakietu w pierścieniu będzie równy iloczynowi czasu wykonywania się testu (τ) i liczbie węzłów (m). Oczywiście szybszy czas obiegu pakietu powoduje, że posiadane informacje o węzłach w systemie są bardziej aktualne. Czas τ jest stały, ale jako parametr może być zmieniany przez użytkownika

z poziomu interfejsu. Wielkość czasu τ wynika z konkretnego kryterium, pod kątem którego testowany jest węzeł i jest związana z konkretną implementacją. Ze względu jednak na aktualność informacji pożądane jest, aby czas ten był możliwie najkrótszy. Zależność okresu aktualizacji (okresu obiegu pakietu) zależy liniowo od rozmiaru sieci, przy czym czas ten nie może być niższy niż iloczyn $m \bullet \tau$. Dalsze zmniejszenie obciążenia wprowadzanego do systemu może być osiągnięte poprzez wprowadzenie przerw w okresie testowania.

2.3. Warstwa komunikacyjna -wariant okresowej generacji pakietu

Innym wariantem komunikacji, jaki opracowano i zaimplementowano w systemie NetMon, jest wariant, w którym testowanie stanów oddzielono od obiegu pakietu w pierścieniu. Każda stacja periodycznie testuje się sama. Niezależnie od wykonywania się testu co określony czas generuje pakiet zawierający informacje o swoim ostatnio zbadanym stanie oraz aktualne informacje, jakie posiada na temat stanów innych stacji. Każda stacja po odebraniu pakietu uaktualnia swoją lokalną informację o innych stacjach i kolejny pakiet przez nią wysłany zawiera już zmodyfikowaną zawartość. Informacja, zawarta w części pakietu dotyczącej danego węzła, obiega pierścień logiczny. Ten wariant komunikacji zobrazowano na rys.3.



Rys.3. Komunikacja pierścieniowa; wariant okresowej generacji pakietu
Fig.3. The communication with the periodical packet generation by every workstation

Czas t_p propagacji informacji o zmianie stanu stacji jest dla tego wariantu następujący:

$$(m-1) \bullet t_2 \le t_p \le (m-1) \bullet (t_r + t_2) , \qquad (4)$$

gdzie: t_r -czas pomiędzy wygenerowaniem kolejnego ogłoszenia o stanie stacji,
t_2 -czas zbudowania i wysłania ogłoszenia,
t_{w2} -okres pomiędzy wywołaniem kolejnego testu.

Wprowadzane obciążenie Q_2 do sieci jest większe od obciążenia Q_1 wprowadzanego w wariancie pierwszym komunikacji. Testowanie odbywa się równocześnie w każdym z węzłów, stąd obciążenie Q_2 jest liniowo zależne od liczby wszystkich węzłów. Zatem opisywany wariant komunikacji powinien być stosowany w przypadkach, gdy dokładna znajomość aktualnego stanu zasobów systemu w poszczególnych węzłach jest decydująca. Ma to miejsce, gdy czas testowania τ jest długi, a ilość węzłów w systemie znaczna.

W celu zapobieżenia dwukrotnemu otrzymaniu tej samej informacji o obciążeniu danej stacji roboczej wprowadzono ograniczenie na czas generacji pakietu; nie może on być krótszy niż czas przygotowania i wykonania testu. Podstawowe parametry, takie jak czas wykonywania testu, czas odstępu między zakończeniem wykonywania się ostatniego testu a rozpoczęciem kolejnego testowania oraz czas oczekiwania między kolejnymi wysłaniami pakietu do pierścienia (czas generacji) są parametrami, które mogą być zmieniane z poziomu interfejsu systemu NetMon. Wartości tych parametrów są identyczne dla wszystkich węzłów pierścienia.

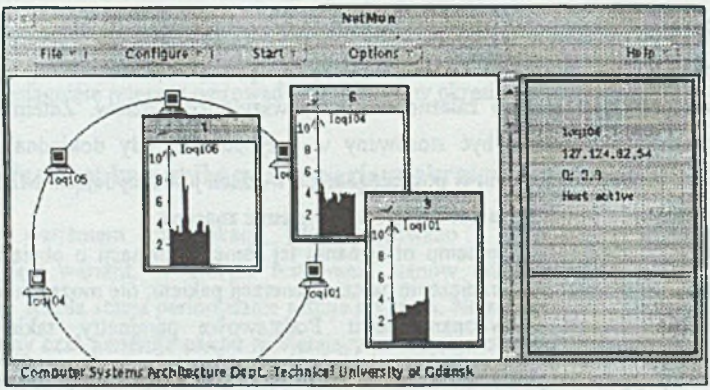
Dla założonej całkowicie rozproszonej strategii monitorowania nie istnieje żadna wyróżniona stacja pełniąca funkcje zarządzające czy kontrolne. Stąd NetMon jest odporny na pojedyncze błędy i algorytm postępowania przy rekonfiguracji pierścienia w przypadku wykrycia awarii węzła jest taki sam dla każdego węzła. Zmiana jednego z dwóch wariantów komunikacji może odbywać się podczas pracy systemu NetMon na żądanie użytkownika.

2.4. Dane techniczne pakietu monitorowania obciążenia węzłów

Przedstawiony pakiet zaimplementowano w języku C na stacjach roboczych Sun Classic, pracujących pod systemem SunOS 5 ze środowiskiem graficznym Open Windows 3. Następnie po niewielkich modyfikacjach powstała wersja przeznaczona do monitorowania komputerów PC 486/Pentium, znajdujących się pod kontrolą systemu Linux ze środowiskiem graficznym XFree86. Wykorzystano sockety [7] jako mechanizmy komunikacyjne z uwagi na zachowanie standardu i zapewnienie przenośności narzędzia. Dzięki temu pakiet może być wykorzystywany do monitorowania sieci LAN dowolnego standardu: Ethernet/802.3, Token Ring, Token Bus. Pakiet zawiera przyjazny interfejs (rys.4) z użytkownikiem, bazujący na standardzie Open Look.

Zainstalowanie pakietu wymaga około 200 kB pamięci dyskowej, 1.2 MB pamięci RAM/swap. Zaś jego uruchomienie obciąża węzeł w stopniu porównywalnym do mocy obliczeniowej wykorzystywanej przez około 1.7-1.8 procesu aktywnego. Oznacza to, że przy

np. 15-16 innych procesach aktywnych w węźle uruchomienie systemu NetMon wykorzystuje ok. 10% mocy obliczeniowej węzła.



Rys.4. Graficzny interfejs systemu NetMon
Fig.4. The graphical interface of NetMon

3. Eksperymenty praktyczne

Tabela I

Pomiar obciążenia wprowadzanego przez system NetMon

Czas testowania τ [s]	Czas rzeczywisty wykorzystania procesora przez proces testu		
	bez systemu NetMon [s]	z systemem NetMon tryb okresowej generacji pakietu [s]	z systemem NetMon tryb komunikacji z tokenem [s]
10	średnio: 9.68	średnio: 5.09	średnio: 5.11
	obciążenie stacji** [l. procesów] : 1.03	obciążenie stacji [l. procesów] : 1.96	obciążenie stacji [l. procesów] : 1.96
20	średnio: 19.65	średnio: 5.12	średnio: 5.06
	obciążenie stacji [l. procesów] : 1.02	obciążenie stacji [l. procesów] : 1.95	obciążenie stacji [l. procesów] : 1.98

Przeprowadzone eksperymenty miały na celu określenie przydatności systemu NetMon do dynamicznego określania stanu obciążeń stacji roboczych. Poniżej przedstawiono wyniki dwóch testów. Test pierwszy miał za zadanie określenie obciążenia (rozumianego jako liczba

**liczba procesów aktywnych = (czas testowania) / (czas wykorzystania procesora przez test)

aktywnych procesów definiowana wzorem (2)) wprowadzanego przez system NetMon dla stacji roboczej poprzez pomiar efektywnego czasu przetwarzania testu (wykorzystywanego w systemie NetMon) na stacji roboczej typu Sparc Classic, którą obciążało:

- 0 zadań użytkowych (aktywnych procesów),
- aktywna instancja systemu NetMon pracująca w trybie komunikacji z okresowym generowaniem pakietu (pierścień 2 stacji, brak odstępu pomiędzy kolejnymi testami),
- aktywna instancja systemu NetMon, pracująca w trybie komunikacji z tokenem (pierścień 2 stacji roboczych).

Badania przeprowadzono dla czasów testowania wynoszących 10 i 20 sekund. Uzyskane wyniki przedstawiono w tabeli 1.

Z porównania rezultatów, uzyskanych dla różnych czasów trwania testu, wynika, że dokładność testu (a zatem i całego systemu monitoringu) nie zależy od czasu trwania testu w zakresie <10,20> sekund. Z przeprowadzonych badań (omówionych w pracy [2]) wynika, że dla $\tau \in <1,5>$ dokładność testowania jest niezadowalająca dla dokładnego określenia obciążenia i wynosi 1-5 procesów.

Test drugi miał za zadanie pomiar obciążenia stacji roboczej z wykorzystaniem systemu NetMon. Wykonano pomiar obciążenia Q stacji roboczej Sparc Classic:

- nie obciążonej dodatkowymi aktywnymi procesami,
- dociążonej trzema dodatkowymi aktywnymi procesami,

w różnych konfiguracjach pierścienia. Czas testowania ustalono na 10 sekund.

Na podstawie otrzymanych wyników drugiego testu można sformułować następujące wnioski:

1. Wielkość pierścienia nie ma wpływu na dokładność pomiarów obciążenia.
2. Wyniki uzyskiwane przez system NetMon są zgodne z oczekiwaniami, tzn. liczba procesów aktywnych wykrytych przez system monitorujący jest określana z **dokładnością do 0.1 procesu**.

Tabela 2

Obciążenie stacji roboczej

Liczba węzłów w pierścieniu	Q (obciążenie) przy braku dodatkowych procesów w węźle	Q (obciążenie) przy 3 dodatkowych procesach w węźle
3	Q śr.: 1.82	Q śr.: 4.78
5	Q śr.: 1.69	Q śr.: 4.63
7	Q śr.: 1.73	Q śr.: 4.62

4. Wnioski końcowe

Opracowany pakiet rozproszonego monitorowania stanu węzłów sieci lokalnej stanowić może warstwę systemu przetwarzania rozproszonego [1], bowiem udostępnia interfejs do gromadzonej informacji o obciążeniu. Te dane mogą być wykorzystywane przez warstwę dystrybuującą (pomiędzy poszczególne węzły) zadania do wykonania. Sam pakiet monitorowania zapewnia efektywne powiadamianie zbioru stacji o stanach stacji pozostałych, co powoduje, że system przetwarzania rozproszonego, stworzony na bazie systemu NetMon, nie musi być systemem sterowanym centralnie.

Zaproponowany pakiet może być wykorzystany również do zadań diagnostycznych heterogenicznych stacji roboczych pracujących pod kontrolą systemu Unix i połączonych siecią lokalną. Dotychczas zaiplementowane zostały wersje dla stacji Sparc Classic (SunOS/OpenWindows) oraz PC 586/Pentium (Linux/XFree86). Jednakże istnieje również wersja nie zawierająca interfejsu graficznego dla stacji roboczych HP 9000, zaś interfejs zgodny ze standardem Motif jest w fazie opracowywania. W bliskiej przyszłości system NetMon będzie można wykorzystywać na większości głównych platform unixowych.

W obu trybach komunikacji zaimplementowana została automatyczna rekonfiguracja pierścienia w przypadku awarii dowolnego węzła [8]. Natomiast odłączanie na żądanie węzła z pierścienia (wykonywane z poziomu interfejsu) w trybie komunikacji z jednym pakietem-tokenem nie jest jeszcze w pełni wykonane. Proponuje się następującą procedurę: informacja o usuwaniu żądanego węzła P_i rozesłana przez węzeł użytkownika-administratora dociera do wszystkich węzłów mniej więcej w tym samym czasie. Wszystkie węzły dokonują modyfikacji własnych zbiorów informacji, tzn. usuwają stamtąd informację o odejmowanym węźle. Następnie węzeł, który był poprzednikiem usuwanego węzła P_i, sprawdza czy token ostatnio wysłany przez niego do węzła P_i dotarł do P_i w chwili jego usuwania z pierścienia. Jeśli tak, wówczas węzeł P_{i-1} ponownie generuje token i wysyła go do węzła P_{i+1} (który jest teraz jego następnikiem). Sprawdzenie utraty tokena odbywa się na podstawie zależności czasowych. Jeśli czas, który upływa od momentu wysłania tokena z węzła P_{i-1} do węzła P_i, jest mniejszy niż suma czasu testowania (τ) i czasu transmisji tokena pomiędzy węzłami P_{i-1} oraz P_i, to wówczas węzeł P_i został usunięty wraz z tokenem. Niestety, czasu transmisji nie można dokładnie określić (jest zmienny) i można go jedynie oszacować. To może spowodować (w przypadku gdy rzeczywisty czas transmisji jest większy niż szacowany), że utrata tokena nie będzie zauważona. W przypadku przeciwnym zostanie niepotrzebnie wygenerowany taki sam token. Tymczasowo w pakiecie NetMon wykorzystano wariant drugi. Do implementacji pozostaje więc algorytm usuwania dodatkowego tokena oraz dołączanie (na żądanie) węzła do pierścienia logicznego. Możliwe są do wykorzystania algorytmy zaproponowane w pracy [7].

LITERATURA

[1] Brudło P., Krawczyk H.: *A Concept of a Fully Distributed Processing Strategy oriented for Unix Local Area Networks.* Microprocessing and Microprogramming The Euromocro Journal, vol. 40 (1994), pp. 723-726.

[2] Brudło P., Krawczyk H.: *Wyznaczanie bieżących obciążeń węzłów lokalnej sieci komputerowej.* Materiały seminarium sieci Komputerowe II, Gliwice, luty 1995, Zeszyty Naukowe Politechniki Śląskiej, S. Informatyka, ss. 177-186.

[3] Brudło P., Krawczyk H.: *Dwuwarstwowa strategia rozproszonego przetwarzania zadań w lokalnej sieci komputerowej.* Materiały konferencji "Informatyka na wyższych uczelniach dla gospodarki narodowej", Gdańsk, listopad 1994, tom I, ss. 135-139.

[4] Comer Douglas E., Stevens David L.: *Internetworking with TCP/IP Volume III,* Prentince Hall, Inc. 1993.

[5] Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R., Sunderam V.: *PVM 3 User's Guide and Reference Manual.* Oak Ridge National Lab., Univ. of Tennessee, maj 1993.

[6] Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R., Sunderam V.: *HeNCE: A User's Guide Version 1.2,* Oak Ridge National Lab., Univ. of Tennessee, December 1992.

[7] Mamczyński M., Targowski M.: *Badanie obciążenia i konfiguracji sieci lokalnej Unix.* Praca dyplomowa obroniona na Politechnice Gdańskiej, październik 1995 .

[8] Silva L., Silva J.: *DIP: Distributed Diagnosis Protocol.* Microprocessing and Microprogramming, vol. 38, 1993, pp. 171-178.

[9] Stevens W. Richard: *UNIX Network Programing.* Prentince Hall, Inc. 1990.

[10] Umar A.. *Distributed Computing. A Practical Synthesis.* Prentice-Hall, New Jersey, 1993

Recenzent: Prof. dr hab. inż. Andrzej Grzywak

Wpłynęło do Redakcji 21 grudnia 1995 r.

Abstract

Local area network consists of independently working stations connected by communication links. Users can work independently and according to their needs they can

communicate with other users or with common resources (files, I/O devices). Network software such as PVM, Linda or Amoeba (see Section 1) allows users to create special environments for parallel/distributed processing of tasks . Efficient realisation of such task processing strategy requires information about current workload of each network station. However, there is no acceptable solution for determining current station workload in an automatic way; i.e. without user intervention. The paper shows a new approach based on the previously defined original model of distribute processing described in papers [1,3].

In section 2 the workload test is proposed. This test allows users to estimate average workstation workload as the number of active processes. The flowchart of the test is given in Fig.1. The number of active processes in the station can be determined as follows:

$$n = \frac{\tau}{\tau_E} - 1 \tag{2}$$

where τ is the time of testing, given in advance, and τ_E is the real time of test execution (having a processor resource). We assume that the test cannot be suspended.

The proposed test should be performed occasionally by each network station. The information packet with the number of active processes executing by every station should be delivered to all stations. Two communication strategies based on logical ring communication scheme are proposed in this paper. In the first variant (see Section 2.2) the station waits for the packet, then determines its local workload states, updates the packet and sends the packet to its neighbour. In the second variant (see Section 2.3) circulation of the packet is independent of the test application. Each station performs its test periodically and then updates the information about its current workload state. All stations send packets with current information in the same moment. These periods can differ and the packet is sent independently of the test starting point. To evaluate suitability of the presented variants the software package named NetMon was designed and implemented in the C language (see Section 2.4). It runs on Sun Classic stations with operating systems SunOS 5.x and Open Windows 3.x. Also another version for PC with Linux system is prepared. The communication is based on sockets available in Unix system. NetMon has a user-friendly interface compatible with the Open Look standard.

To illustrate suitability of the tool, two simple experiments are described (see Section 3). It was shown that for test execution time balancing between 10 and 20 seconds, the results are acceptable and the overload of the test does not affect their exactness. Moreover the influence of the communication ring size is also negligible. In conclusion, suggestions about the package improvement are given. Main problems connected with ring reconfiguration controlled by users are discussed and possible solutions are given.