ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1993

Seria: INFORMATYKA z. 24 Nr kol. 1222

Leszek BORZEMSKI
Sławomir GŁOWACZ
Krzysztof WOJTCZAK

SYSTEM RÓWNOWAŻENIA OBCIĄŻENIA W SIECI KOMPUTERÓW UNIX*

Streszczenie. W pracy opisano rozproszony komputerowy system równoważenia obciążenia w lokalnej sieci komputerowej - Unix. System został początkowo przeznaczony do efektywnego wspomagania wieloetapowego podejmowania decyzji, ale może też być wykorzystany przy wsadowym przetwarzaniu dowolnych zadań.

A LOAD BALANCING SYSTEM FOR UNIX-BASED LOCAL AREA NETWORKS

Summary. This paper presents a distributed load balancing computer system developed for Unix-based local area networks. The system has been initially developed for supporting efficient distributed processing of knowledge-based multistage decision-making but it can also support load balancing in a general environment.

GLEICHGEWICHTSSYSTEM FÜR AUF UNIX BASIERENDE LOKALE NETZWERKE

Zusammenfassung. Dieser Artikel beschreibt ein Computersystem, das sich im Gleichgewicht befindet und für auf Unix basierende Local Area Networks entwickelt wurde. Ursprünglich wurde das System entwickelt, um die effiziente Verteilung von, auf multiplexen Kenntnissen beruhenden Prozessentscheidungen zu unterstützen. Es dient aber auch dazu, allgemeine Systemumgebungen im Gleichgewicht zu halten. Die Grundzüge dieses Systems wurden dem SCO-Unix Betriebssystem ohne Modifikation seines Kerns beigefügt.

* Pracę zrealizowano w ramach projektu Nr 3 0762 91 01 finansowanego w latach 1991-1994 przez Komitet Badań Naukowych.

1. Wstęp

Równoważenie obciążenia komputerów w lokalnej sieci komputerowej jest jednym z najważniejszych celów, do którego dążą projektanci rozproszonych systemów komputerowych. W pracy przedstawiony jest system równoważenia obciążenia dla komputerów z systemami operacyjnymi Unix w lokalnej sieci Ethernet z protokołem TCP/IP. System znalazł zastosowanie przy implementacji rozproszonego systemu rozpoznawania wieloetapowego [1].

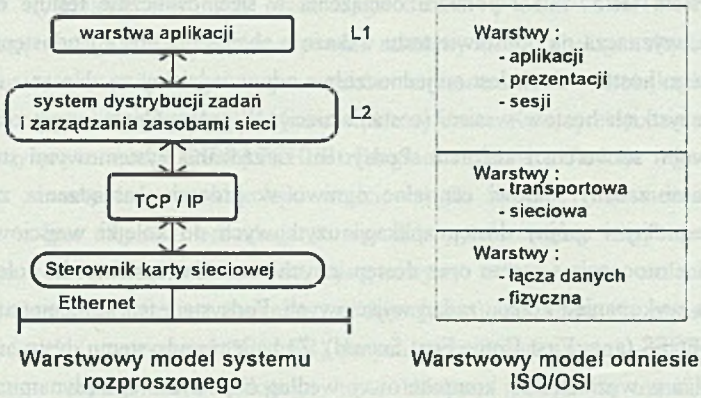
2. Struktura i funkcjonowanie systemu

Prezentowany w pracy system rozproszonego przetwarzania, zorientowany na równoważenie obciążenia i ukierunkowany na wspomaganie wieloetapowego podejmowania decyzji, został skonstruowany w lokalnej sieci komputerowej ze stacjami roboczymi Unix w Instytucie Sterowania i Techniki Systemów Politechniki Wrocławskiej. Przyjęto następujące założenia projektowe:

a) środowisko rozproszone składające się ze zbioru serwerów zadań obliczeniowych (hostów), połączonych wspólnym szybkim medium transmisji danych (Ethernet, 10 Mb/s),
b) jednorodne architektury hostów ze względu na formaty reprezentacji danych (architektury na bazie procesorów Intel 80386 i i486),
c) zgodność systemowa hostów - system operacyjny SCO Unix System V ver.3.2,
d) komunikacja sieciowa oparta na protokole TCP/IP,
e) własny system plików każdego hosta.

System wymaga specyfikacji (w odpowiednim formacie) zadań generowanych przez oprogramowanie aplikacyjne, które są następnie wykonywane na jednym z procesorów systemu rozproszonego, przy czym aplikacja nie ingeruje bezpośrednio w dystrybucję zadań. Po przetworzeniu zadania wyniki są zwracane do komputera - nadawcy. Z punktu widzenia aplikacji użytkowej system ten pozwala traktować sieć komputerową jako jeden "wirtualny" procesor, przy czym takie cechy jego funkcjonowania, jak mechanizm równoważenia obciążenia procesorów, są dla aplikacji transparentne. Przy projektowaniu systemu oparto się na modelu ISO/OSI. Podejście warstwowe wymagało zdefiniowania usług warstwy "systemowej" (L2) świadczonych warstwie "aplikacyjnej" (L1), a także interfejsu między tymi

warstwami oraz między warstwą L2 a warstwą transportową (TCP/IP). Na potrzeby interfejsu między warstwą L1 a L2 zdefiniowano rozszerzone pojęcie zadania, przez które w dalszej części pracy będzie się rozumieć pewien zbiór danych (reprezentowany np. jako plik dyskowy) o ustalonej strukturze. Z punktu widzenia systemu dystrybucji i transferu zadań struktura ta jest dwudzielna. Pierwsza część zadania stanowi informację inicjującą przetwarzanie i umożliwiającą zwrot wyników przetwarzania. Na drugą część składają się tzw. dane specyficzne zadania (dane wejściowe dla zdalnego programu).

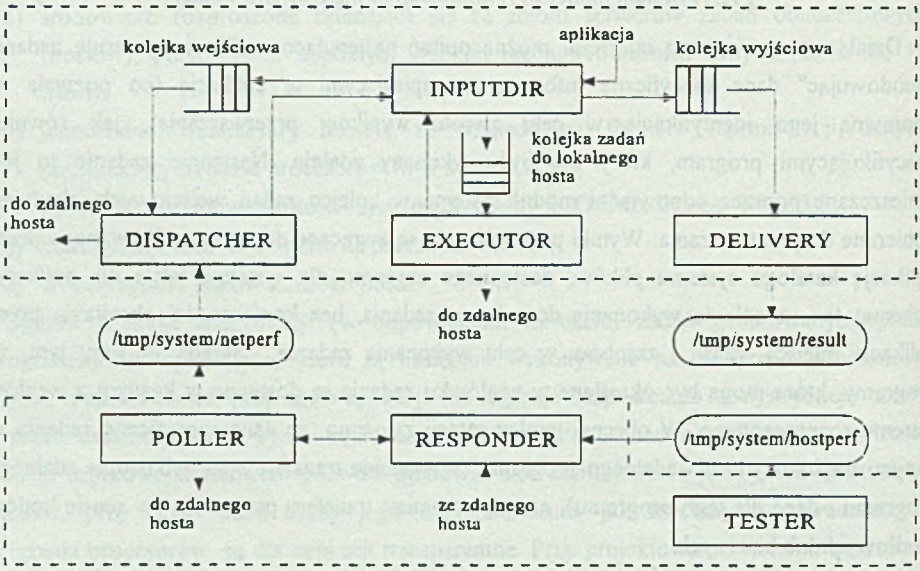


Rys. 1. Model systemu na tle modelu ISO/OSI
Fig. 1. System model in relationship to the ISO/OSI model

Działanie omawianego interfejsu można opisać następująco: aplikacja generuje zadanie "obudowując" dane specyficzne informacjami opisującymi tę aplikację (co pozwala na poprawną jego identyfikację w celu zwrotu wyników przetwarzania), jak również specyfikującymi program, który ma być wykonany zdalnie. Następnie zadanie to jest umieszczane poprzez odpowiedni moduł systemu w kolejce zadań wejściowych, skąd jest pobierane do przetwarzania. Wyniki przetwarzania są zwracane do wyspecyfikowanego przez aplikację katalogu systemu plików, dostępnego zarówno dla systemu, jak i dla aplikacji. Schemat ten umożliwia wykonanie dowolnego zadania, bez konieczności określania przez aplikację miejsca, czasu i zasobów w celu wykonania zadania. Zakłada się przy tym, że programy, które mogą być określane w nagłówku zadania są dostępne w każdym z węzłów systemu rozproszonego. W obecnej implementacji założono, że dane specyficzne zadania są danymi wejściowymi dla zdalnego programu (w systemie transferuje się tylko opis zdalnego programu i dane dla tego programu), nie ma natomiast transferu programów w sensie kodów źródłowych lub binarnych.

System składa się z czterech podsystemów: komunikacji międzyprocesowej, testowania i pomiaru obciążenia, dystrybucji zadań oraz zarządzania systemowymi kolejkami zadań.

Zadaniem podsystemu komunikacji jest dostarczenie pozostałym elementom systemu mechanizmów komunikacji między sobą w obrębie jednego komputera, jak również w obrębie sieci. Korzysta on z usług TCP/IP. Ze względu na chęć zachowania jednolitego podejścia do projektowania systemu nie wyróżniono komunikacji w obrębie lokalnego hosta (w której można wykorzystać bezpośrednio system operacyjny w dostępie do lokalnych zasobów) i komunikacji ze zdalnymi hostami, w której korzysta się z TCP/IP. Podejście to powoduje dodatkowe narzuty w komunikacji lokalnej, pozwala jednak uprościć proces projektowania systemu. Podsystem testowania i pomiaru obciążenia w sieci cyklicznie testuje obciążenie lokalnego hosta, wyznacza na podstawie testu wskaźnik obciążenia hosta i udostępnia go na żądanie dowolnego hosta w sieci. Jest on jednocześnie odpowiedzialny za zbieranie informacji o obciążeniu wszystkich hostów w sieci (o stanie sieci). W szczególnym przypadku jest on jednocześnie swoim serwerem i klientem. Podsystem zarządzania systemowymi strukturami danych (kolejkami zadań) stanowi centralne ogniwo w procesie zarządzania zadaniami. Zapewnia on jednolity i spójny dostęp aplikacji użytkowych do kolejki wejściowej zadań przychodzących z otoczenia systemu oraz dostęp innych modułów systemu do kolejki zadań oczekujących na wykonanie i kolejki zadań wyjściowych. Podsystem ten implementuje sposób obsługi kolejek FCFS (ang. First Come First Served). Zadaniem podsystemu dystrybucji zadań jest alokacja zadań w węzłach sieci komputerowej, według odpowiedniego (dynamicznego lub innego) algorytmu alokacji zadań, zorientowanego na problem równoważenia obciążenia.



Rys. 2. Schemat strukturalny systemu
Fig.2. Diagram of the system structure

Poszczególne podsystemy składają się z modułów (rys.2). Każdy z modułów jest implementowany w postaci oddzielnego programu (procesu). Moduły świadczące usługi (serwery) są tzw. demonami systemowymi i oferują swoje usługi poprzez określone porty i zainstalowane na tych portach gniazdka. Moduł INPUTDIR oczekuje na gniazdku na żądanie połączenia. W momencie, gdy inny moduł systemu lub zewnętrzny wobec systemu proces aplikacyjny żąda połączenia, program rozwidla się, przy czym proces główny powraca do trybu oczekiwania, natomiast proces potomny jest odpowiedzialny za komunikację z klientem modułu i wykonanie zleconych przez niego działań. Moduł INPUTDIR umożliwiając jednoczesną obsługę wielu klientów zapewnia założony sposób obsługi kolejek poprzez spójny sposób tworzenia unikatowych nazw plików umieszczanych w kolejkach systemowych, którymi są odpowiednie katalogi w systemie plików Unixa. Kolejny etap działania systemu jest realizowany przez moduł DISPATCHER. Program realizujący funkcje tego modułu testuje cyklicznie zawartość kartoteki-kolejki wejściowej dla zadań. W momencie wykrycia, że kartoteka ta jest niepusta, odszukuje w niej plik o najmniejszej leksykograficznie nazwie (nazwa ta jest związana z czasem systemowym, wobec czego zadanie o mniejszej leksykograficznie nazwie jest zadaniem wcześniejszym), po czym na podstawie wcześniej zgromadzonych (przez moduł POLLER) danych o obciążeniu systemu, podejmuje decyzję o alokacji tego zadania do określonego hosta. Alokacja polega na transferze zadania za pośrednictwem modułu INPUTDIR na wybranym hoście do kolejki zadań do lokalnego wykonania. W systemie zaimplementowano klasę algorytmów alokacji zadań inicjowanych przez nadawcę zadania (ang. sender-initiative algorithms). Poza tym rozpatrywany sposób alokacji zakłada, że alokowane zadanie nie może zostać przez wybrany host odrzucone, nie ma więc w systemie migracji zadań, mogącej potencjalnie prowadzić do "martwego" krążenia zadania w sieci. Integralną częścią omawianego modułu, a zarazem elementem decydującym o zachowaniu się systemu i jego efektywności ze względu na warunek równoważenia obciążenia, jest procedura podejmowania decyzji o alokacji zadań. Zadaniem modułu EXECUTOR jest wykonywanie zleconych zadań i zwracanie rezultatów przetwarzania do komputera - nadawcy zadania. W tym celu testuje on cyklicznie zawartość kolejki wejściowej i w momencie wykrycia obecności w niej zadania podejmuje się jego wykonania poprzez inicjację odpowiedniego programu. O samych programach zakłada się, że dane wejściowe pobierają ze standardowego wejścia, a wyniki zwracają na standardowe wyjście. Po zakończeniu przetwarzania proces uruchomiony przez moduł EXECUTOR odbiera od wykonanego programu wyniki przetwarzania i poprzedzając je odpowiednimi informacjami organizacyjnymi, pozwalającymi zidentyfikować nadawcę zadania w hoście-nadawcy oraz same wyniki przetwarzania, przekazuje je poprzez moduł INPUTDIR hosta-nadawcy do kolejki wyjściowej. Kolejkę wyjściową testuje cyklicznie moduł DELIVERY, który na

podstawie informacji z nagłówka pliku przepisuje go pod określoną nazwą do odpowiedniej kartoteki. Aplikacja, która wygenerowała zadanie, może następnie odczytać wyniki przetwarzania w tym pliku. Zadaniem modułu TESTER jest cykliczne (z okresem ok. 0,5 sek.) testowanie obciążenia komputera. Moduł RESPONDER służy do udostępniania danych o obciążeniu lokalnego hosta wszystkim pozostałym hostom w sieci. Klientem modułu RESPONDER jest moduł POLLER. Jego zadaniem jest gromadzenie danych o obciążeniu wszystkich hostów w sieci. Hosty w sieci są testowane co 2 sekundy (jest to wartość wybrana eksperymentalnie i stanowi kompromis między częstym testowaniem hostów, co zwiększa adekwatność danych o obciążeniu, ale powoduje duże narzuty czasowe, a zbyt rzadkim testowaniem, mogącym prowadzić do dezaktualizacji danych).

3. Problem pomiaru obciążenia

Zasadniczym problemem przy opracowywaniu podsystemu badania obciążenia jest pomiar obciążenia hosta. Pomiar ten powinien umożliwić wyznaczenie w postaci liczbowej wskaźnika obciążenia hosta do oceny "wirtualnej mocy obliczeniowej" z punktu widzenia nowego procesu użytkowego, które miałby być uruchomiony na danym hoście. W systemie wielozadaniowym Unix sprawność systemu widziana od strony nowo uruchamianego procesu zależy nie tylko od fizycznej mocy obliczeniowej danej maszyny, ale również od chwilowej liczby procesów zarządzanych przez system. Można więc powiedzieć, że sprawność systemu fluktuuje w miarę jak zmienia się liczba zadań (procesów) w systemie. Jeśli ponadto w systemie rozproszonym występują hosty o niehomogenicznych architekturach, to przy pomiarze obciążenia całego systemu należy brać pod uwagę nie tylko liczby procesów w poszczególnych hostach oraz długości kolejek zadań, ale również fizyczną moc poszczególnych maszyn. Ponieważ ocena chwilowej sprawności danego hosta jest dokonywana z punktu widzenia pojedynczego procesu, za miarę tej sprawności został przyjęty czas przetworzenia przez system pewnego uogólnionego, testowego procesu. Załóżmy bowiem, że w danej chwili jest obecnych w pewnym węźle sieci komputerowej K procesów (aktywnych). System operacyjny przydziela cyklicznie każdemu procesowi aktywnemu kwant czasu zgodnie z pewną strategią szeregowania, mającą na celu maksymalizację wykorzystania zasobów systemu. Oznaczając ten kwant czasu przez Δt oraz przez T, wymaganie całkowitego czasu procesora danego procesu, można napisać $T = l * \Delta t$, gdzie l - oznacza liczbę kwantów czasu procesora potrzebną na skompletowanie zadania (procesu). Stałą l można też wyrazić wzorem, $l=k/W$, gdzie k - stała określająca rozmiar zadania, W - bezwzględna moc procesora (systemu). Wówczas czas T można zapisać następująco: $T=k* \Delta t/W$. Rzeczywisty czas wykonania

zadania T_{real} jest większy od czasu T, gdyż o dostęp do procesora ubiega się K procesów, które są równoprawnie traktowane przez system. Dany proces uzyskuje więc kwant czasu średnio tylko raz na $K*\Delta t$ jednostek czasu. Czas T_{real} można wyrazić wobec tego jako: $T_{real}=k*K*\Delta t/W$. Powyższą sytuację można traktować w ten sposób, jakby dany proces był uruchamiany pojedynczo na procesorze o chwilowej mocy wirtualnej: $W^{vir}=W/K$. Wartość W^{vir} może stanowić podstawę do oceny chwilowej wydajności systemu, uwzględnia bowiem fizyczną moc procesora, a także liczbę procesów aktywnych.

4. Testowanie stacji roboczych Unix i badanie obciążenia hostów

Do oceny całościowej sprawności systemu komputerowego najlepiej nadają się testy syntetyczne, pozwalające wyznaczyć tzw. ogólny wskaźnik sprawności systemu. Do wyznaczenia wskaźnika obciążenia hostów napisano program testowy realizujący wybrane zadania (m.in. kompilacja programu w języku C, obliczanie pierwiastka kwadratowego metodą Newtona, rozwiązanie zadania "Wieże Hanoi"). Zastosowaną metodę pomiaru obciążenia można przedstawić następująco: ponieważ na sprawność konkretnego systemu wpływa liczba procesów aktywnych obecnych w systemie, za miarę obciążenia przyjęto liczbę procesów aktywnych. Liczba ta nie jest bezpośrednio dostępna (przynajmniej na poziomie aplikacji, bez wnikania w jądro systemu), postanowiono więc szacować ją na podstawie czasu wykonania programu testowego, przy zadanej liczbie procesów aktywnych. W tym celu wyznaczono (w warunkach quasi-statycznych, w tzw. trybie jednoużytkownikowym systemu) charakterystykę wyrażającą zależność tego czasu od liczby procesów aktywnych. Czas ten posiada trzy składniki: czas procesora zużyty na wykonanie tego programu, czas systemu zużyty na jego szeregowanie oraz narzut czasowy, wynikający z szeregowania innych procesów. Z uzyskanych danych wynika, że wraz ze wzrostem liczby procesów aktywnych w systemie wzrasta czas wykonania zadania, natomiast prawie niezmienny pozostaje łączny czas procesora (czas czystych obliczeń procesora na rzecz programu plus czas systemu zużyty na zarządzanie procesem). Czas ten można uważać za bezwzględny wskaźnik mocy komputera. Wskaźnik ten może być wykorzystany przy normalizacji globalnego wskaźnika chwilowej mocy systemu. Moc tę można określać wyznaczając na drodze eksperymentu w trakcie działania systemu stosunek rzeczywistego czasu wykonania pewnej krótkiej procedury testowej do łącznego czasu systemowego tej procedury. Iloraz ten, odzwierciedlający liczbę procesów aktywnych w danym (krótkim) przedziale czasu, należy przemnożyć przez wyznaczony uprzednio bezwzględny wskaźnik mocy komputera. Otrzymany wynikowy globalny wskaźnik może

służyć do oceny chwilowego obciążenia hosta, gdyż uwzględnia zarówno liczbę procesów w systemie, jak i fizyczną moc komputerów (dla tej samej liczby procesów w systemie komputer szybszy jest mniej obciążony, podobnie dla identycznych komputerów - im więcej procesów w systemie, tym większe jest jego obciążenie). W celu weryfikacji tej metody przeprowadzono badania (w identycznych warunkach co poprzednio, tzn. w trybie jednoużytkownikowym), w których dla ustalonej liczby procesów aktywnych wyznaczano iloraz czasu rzeczywistego t_{real} (czas ten był ustalony na 0.160 sek.) do łącznego czasu systemowego $t_{user} + t_{sys}$ pewnej pętli programu testującego. Na podstawie testów okazało się, że wartość tego ilorazu jest dla tej samej liczby procesów aktywnych taka sama w przypadku niejednakowo wydajnych komputerów - można go więc wykorzystać do oceny liczby procesów aktywnych wykonywanych w danej chwili przez system, a co za tym idzie - do oceny obciążenia. Otrzymany w powyższy sposób globalny wskaźnik chwilowej wydajności systemu stanowi estymatę czasu rzeczywistego (czasu wykonania), jaki uzyskano by, gdyby wykonać pomiar z użyciem programu testowego w tych samych warunkach. Estymacja nie wymaga jednak przeprowadzania długiego (a co za tym idzie, nieodpowiedniego w rzeczywistym działaniu systemu, w którym założenie o quasi-stacjonarności stanu obciążenia w dłuższym odcinku czasu jest niesłuszne) testu. Poza tym nie ma dodatkowych obciążeń systemu przez program testowy. Estymowany czas rzeczywisty stanowi syntetyczny wskaźnik chwilowego obciążenia systemu (im większe jest obciążenie systemu, tym dłuższy jest estymowany czas).

Problem badania obciążenia hostów sieci jest związany z wyborem algorytmu zbierania informacji o aktualnym obciążeniu tych hostów. W prezentowanym rozwiązaniu zdecydowano się na podejście, w którym każdy host testuje obciążenie wszystkich węzłów sieci. Testowanie odbywa się cyklicznie z arbitralnie wybranym okresem T=2 sek. Okres ten nie może być zbyt mały, gdyż narzuty czasowe spowodowane badaniem kolejnych hostów sumują się, więc efektywny okres i tak się zwiększa (zwłaszcza przy większej liczbie hostów w sieci). Osobnym zagadnieniem jest problem wyznaczania wskaźnika obciążenia hosta, udostępnianego innym hostom, na podstawie lokalnego pomiaru chwilowego obciążenia. Jak pokazały obserwacje działania systemu operacyjnego w normalnych warunkach pracy, wartość obciążenia uzyskiwana w kolejnych taktach testowania podlegała dużym szybkozmiennym fluktuacjom. Ponieważ mogłoby to potencjalnie powodować niestabilność systemu, zaproponowano następujący model:

$$L_i(0) = \overline{U}_i(t_0)|_{t_0=0},$$

$$L_i(n) = (1-k) * L_i(n-1) + k * U_i(n),\ n = 1,2,\ldots,$$

gdzie

$L_i(j)$ - wskaźnik obciążenia i-tego hosta w j-tym takcie testowania,

$\overline{U}_i(t_0)$ - wartość początkowa (np. średnie obciążenie i-tego hosta w pewnym odcinku czasu),

$\overline{U}_i(j)$ - zmierzona wartość obciążenia hosta i-tego w j-tym takcie testowania.

W powyższym modelu dla k=0 otrzymujemy model bazujący całkowicie na uśrednionych pomiarach, natomiast dla k=1 model uwzględnia jedynie bieżący pomiar obciążenia. W przyjętym rozwiązaniu zdecydowano się na wybór k=0.1, co umożliwia adaptację systemu do wolnych zmian obciążenia przy zmniejszonej wrażliwości na krótkotrwałe "skoki" obciążenia, wynikające ze specyfiki działania systemu operacyjnego.

5. Algorytm alokacji zadań i liczba jednocześnie uruchamianych zadań

W systemie zaimplementowano dynamiczny algorytm alokacji zadań, zorientowany na przetwarzanie wsadowe z uwzględnieniem warunku równoważenia obciążenia. Algorytm ten alokuje zadanie do hosta o najmniejszym wskaźniku bieżącego obciążenia (opiera się on na informacji o stanie sieci gromadzonej przez podsystem testowania i pomiaru obciążenia). Algorytm ten adaptuje się do bieżącego stanu w sieci w ten sposób, że w chwili początkowej przybycia wsadu zadań do systemu, do każdego węzła sieci jest alokowane jedno zadanie, które ma umożliwić ocenę czułości wskaźników obciążenia poszczególnych hostów. Stanowi to podstawę do predykcji wskaźników obciążenia (na podstawie bieżącej informacji o stanie sieci) przed podjęciem decyzji o alokacji zadania. Tak więc wybór hosta o minimalnym wskaźniku opiera się na podstawie przewidywanych wartości wszystkich wskaźników. Ponieważ algorytm stosuje strategię niewywłaszczeniową, zabezpieczeniem przed seryjnym alokowaniem zadań do hosta wybranego w danej chwili jako najmniej obciążony jest zastosowanie limitu zadań, które mogą być w danym odcinku czasu (między kolejnymi taktami w procesie gromadzenia danych o stanie sieci) zaalokowane do jednego hosta.

W klasycznych systemach kolejkowych, używanych najczęściej do teoretycznej analizy systemów rozproszonego przetwarzania, przyjmuje się, że każdy z procesorów systemu rozproszonego, w warunkach dostępności co najmniej jednego zadania, wykonuje tylko jedno zadanie. Upraszcza to teoretyczne modele (w szczególności obciążenie, przy dodatkowym założeniu o identyczności procesorów, można wyrażać przez długości kolejek wejściowych), lecz w przypadku implementacji systemu rozproszonego przy oparciu się na systemie wielozadaniowm, założenie to byłoby sztuczne, a poza tym i tak byłoby nie do spełnienia ze względu na samą naturę systemu operacyjnego. Na podstawie rezultatów przedstawionych w poprzednim punkcie widać, że wraz ze wzrostem liczby procesów aktywnych w systemie czas odpowiedzi na kolejne zadania zwiększa się proporcjonalnie do tej liczby, przy czym

wymagania zadania pozostają niezmienne. W związku z tym powstał problem optymalnej liczby zadań wykonywanych jednocześnie jako procesy aktywne. Przeprowadzono serię testów pomiarowych, w normalnych warunkach funkcjonowania systemu operacyjnego, dla jednej wybranej stacji Unix (wyniki są uzależnione od systemu operacyjnego, a nie od platformy sprzętowej). Jak wynika z uzyskanych rezultatów, istnieje zależność między całkowitym czasem wykonania wsadu zadań a liczbą jednocześnie uruchamianych zadań, przy czym wraz ze wzrostem tej liczby maleje czas wykonania wsadu. Tłumaczyć to można tym, że w przypadku uruchamiania zadań pojedynczo (tzn. kolejne zadanie jest uruchamiane po zakończeniu poprzedniego), narzuty czasowe związane z uruchomieniem kolejnego procesu przez system operacyjny sumują się i dodają do łącznego czasu czystych obliczeń związanych z każdym procesem. Wraz ze zwiększaniem liczby procesów jednocześnie uruchamianych w systemie narzuty te pokrywają się na osi czasu z czasem wykonywania poprzedniego procesu, wobec czego łączny czas wykonywania wsadu ulega zmniejszeniu. Czas wykonywania wsadu "nasyca" się przy pewnej liczbie procesów (im mniejszy czas wykonywania pojedynczego zadania, tym szybciej następuje nasycenie). Tłumaczy się to tym, że ponieważ uruchomienie każdego kolejnego procesu wymaga pewnego czasu (opóźnienia, które jest wprost proporcjonalne do liczby procesów aktywnych obecnych w danej chwili w systemie), więc efektywna liczba procesów aktywnych stabilizuje się na pewnym poziomie, pomimo zwiększania maksymalnej dopuszczalnej liczby procesów. Ponadto im krótszy czas wykonywania pojedynczego zadania w porównaniu z opóźnieniem uruchamiania kolejnego zadania, tym bardziej ta efektywna liczba się zmniejsza. W praktyce więc maksymalna liczba zadań powinna być wybrana tak, aby uzyskać minimum czasu (nasycenie) dla każdego typu zadania (w ustalonych granicach rozmiaru zadań wsadowych), a więc jako graniczna wartość dla największego w sensie wymagań czasowych zadania. W rozpatrywanym systemie przyjęto graniczną liczbę zadań jednocześnie uruchamianych równą 5.

6. Uwagi końcowe

Prezentowany system wdrożono i przetestowano w eksperymentach obliczeniowych dla krotnego rozproszonego rozpoznawania wieloetapowego. W pracy [1] przedstawiono wyniki badań i pomiarów systemu w realnych warunkach sieciowych. Przeprowadzono tam również badania porównawcze z wersją systemu dysponującą losowym algorytmem alokacji, wykazujące wyższość podejścia z dynamicznym algorytmem alokacji zadań.

LITERATURA

[1] Borzemski L., Błażków R., Wronka J., Wieloetapowe podejmowanie decyzji w sieci komputerowej z równoważeniem obciążenia, Zeszyty Naukowe Politechniki Śląskiej, Seria: Informatyka z. 24, 1993, 205-216.

Recenzent: Dr inż. Krzysztof Nałęcki

Wpłynęło do redakcji 20 września 1993 r.

Abstract

The system presented in this paper has been developed with the aim of good load balancing (LB). Such a criterion is one of the most important goals from the point of view of the efficient utilisation of distributed systems. The load balancing algorithms can distribute computational tasks to processors such that the processor utilization and the throughput of the system are maximised. Our load balancing system was implemented on a network of several 386/486 PC computers connected via a 10 Mbits/s Ethernet and under control of SCO Unix. The system is able to distribute only the tasks that are loaded under its control - all other tasks are treated as the background load without any moving possibilities. Each host has own LB subsystem working in co-operation with each other. Each host is configured of several deamons supporting the user interface, initial workload input, load balancing control, estimation of processors' loads, task distribution, local task execution, gathering of final results - all actions are performed automatically and in a fully distributed way. The LB mechanism used is decentralized, nonpreemptive and sender-initiative. Each host periodically updates the information about current loads of all hosts by sending requests for running test processes on all hosts and stores the results in a form of specifically calculated performance index vector. The results are normalized in the relative scale for all processors to include different computer power. The system has been used as the vehicle in implementation of the distributed multistage decision-making system.