Podstawowe mechanizmy systemu Linux w przetwarzaniu rozproszonym

Boiński T.

5 marca 2006

Streszczenie. Linux to otwarty, ciągle rozwijany system operacyjny. Dostępność kodu źródłowego oraz pełnej dokumentacji zaowocowała rozwojem wielu mechanizmów wspierających przetwarzanie rozproszone. W systemie Linux zaimplementowane zostały proste mechanizmy typu gniazda czy potoki jak i bardziej zaawansowane typu współdzielenie systemu plików czy możliwość tworzenia klastrów opartych o migrację procesów czy przekazywanie komunikatów. Narzędzia te, również będąc rozwiązaniami otwartymi, doczekały się wielu wersji i rozszerzeń gwarantując dalszy rozwój wykorzystujących je aplikacji.

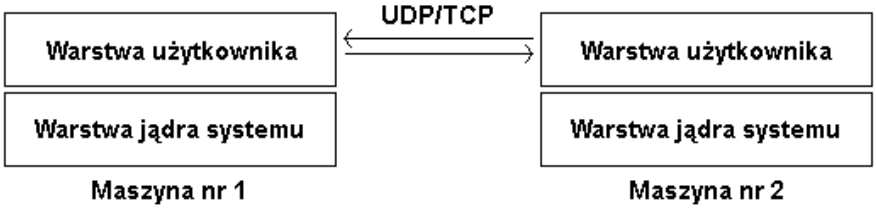
Słowa kluczowe. Linux, NFS, MPI, PVM, openMOSIX, mechanizmy, przetwarzanie, klaster

Linux jest w pełni otwartym systemem[1] ciągle rozwijanym przez szerokie rzesze użytkowników na całym świecie. Dzięki swojej otwartości system ten znalazł bardzo wiele zastosowań w różnych dziedzinach. Całkowita dostępność kodu źródłowego umożliwia o wiele większe dostosowanie do własnych potrzeb niż jakikolwiek inny system operacyjny, stąd nie jest dziwne, iż powstało wiele mechanizmów i rozszerzeń wspierających w różny sposób przetwarzanie rozproszone w tym systemie. Oprócz standardowych mechanizmów, takich jak gniazda czy potoki, dostępnych praktycznie w każdym systemie operacyjnym, oferowane są narzędzia umożliwiające budowanie wysocewydajnych systemów klastrowych, czyli grupy komputerów połączonych siecią, wykonujących obliczenia mające na celu rozwiązanie wspólnego problemu, będących głównym polem zastosowań systemów typu Linux w przetwarzaniu rozproszonym. Klastry budowane są między innymi w oparciu o mechanizmy przekazywania komunikatów czy też system openMOSIX pozwalający na traktowanie grupy niezależnych maszyn jak jedną platformę wieloprocesorową z niezależną pamięcią.[2] W celu uproszczenia użytkowania takich systemów i dodatkowe zwiększenie wrażenia korzystania z jednej, silnej maszyny często dodatkowo stosuje się współdzielony system plików poprzez Network File System (NFS).

1 Network File System (NFS)

Mechanim ten umożliwia współdzielenie systemu plików między wieloma dowolnie od siebie oddalonymi komputerami połączonymi dowolnym rodzajem

sieci.[7][8] Oryginalnie NFS ten został zaprojektowany do współpracy z systemem Linux, jednak powstało wiele implementacji na inne systemy spełniające wszystkie założenia standardu. Dzięki temu możliwe jest bezproblemowe współdzielenie zasobów w środowiskach heterogenicznych. Obecnie protokół NFS jest ciągle rozwijany. Wraz z jego ostatnią, czwartą wersją wprowadzone zostało wiele rozszerzeń takich jak blokowanie plików przez klienta, cache'owanie klientów pozwalające na korzystanie z pliku przez dwóch klientów bez pośrednictwa serwera oraz wysoki poziom zabezpieczeń i wielonarodowość.



Rysunek 1: Architektura NFS

NFS charakteryzuje się dwuwarstwową architekturą (Rysunek 1). Warstwa jądra systemu zaimplementowana jest bezpośrednio w jądrze Linuxa. Zajmuje się wykonywaniem bezpośrednich operacji dyskowych oraz zarządzaniem dostępem. Od niej zależy dbanie o bezpieczeństwo danych oraz na niej spoczywa zachowanie spójności zapisywanego systemu plików w przypadku restartu serwera. W warstwie tej zaimplementowane są wszystkie mechanizmy kontroli dostępu wielu klientów do tego samego zasobu jak i śledzenia zmian w położeniu plików. Warstwa użytkownika składa się z narzędzi uruchomionych poza jądrem. Zadaniem warstwy jest zarządzanie komunikacją z klientami i serwerami w sieci. Narzędzia tej warstwy uruchamiane są w postaci demonów realizujących komunikację zarówno w trybie połączeniowym (poprzez protokół TCP) jak i bezpołączeniowym (poprzez protokół UDP).

Wraz z pojawieniem się wersji 4 protokołu, NFS oferuje możliwość uwierzytelniania i kontroli dostępu zgodnie ze standardem Kerberos.[7] Autoryzacja i autentykacja opiera się o interfejs RPCSEC GSSAPI zdefiniowany przez firmę Sun. W założeniach interfejs ten pozwoli na stosowanie kontroli dostępu we wszystkich wersjach protokołu NFS, jednakże obecnie jest on w dość wczesnej fazie implementacyjnej więc oferowana jest jedynie autentykacja w NFSv4.

Używanie NFS jest bardzo proste. Wymagane jest jedynie jądro z obsługą nfs (obsługa ta jest dostępna w czystym jądrze linuxowym jakie można pobrać z http://www.kernel.org a także praktycznie każdym jądrze dołączanym do jakiejkolwiek dystrybucji) oraz pakiet nfs-tools zawierajacy demony serwera i klienta NFS. W celu konfiguracji serwera NFS należy w pliku /etc/exports wylistować wszystkie udostępniane katalogi oraz podaż prawa dostępu do nich. Następnie wystarczy uruchomić usługi 'portmap', 'statd' oraz 'nfsd' w sposób właściwy dla danej dystrybucji. W nowych dystrybucjach wystarczy uruchomić usługę 'nfsd' a wszystkie pozostałe zostaną automatycznie uruchomione jako zależności. Po stronie klienta należy uruchomić usługi 'portmap', 'lockd' oraz 'statd'. Po wykonaniu tej czynności użytkownik powinien mieć możliwość montowania udostępnionych przez serwer katalogów za pomocą polecenia 'mount ad-

res.serwera:/katalog_na_serwerze katalog_lokalny'. Możliwe jest także automatyczne montowanie katalogów poprzez standardowy wpis w pliku /etc/fstab.

2 Konstrukcja klastrów oparta o przekazywanie komunikatów

Najbardziej znanymi rodzajami klastrów są maszyny typu Beowulf.[3] Klastry te wymagają specjalistycznego oprogramowania zdolnego wykorzystać topologię sieci łączącej poszczególne węzły w celu uzyskania jak najlepszego efektu pracy. Każdy z węzłów dysponuje swoją własną, niedostępną dla innych przestrzenią adresową, stąd konieczne jest zastosowanie mechanizmów umożliwiających wydajną komunikację między procesami wykonywanymi na różnych maszynach. Dla takich zastosowań storzono odpowiednie biblioteki takie jak MPI czy PVM. Zarówno MPI jak i PVM są bardzo dojrzałymi projektami szeroko stosowanymi w praktyce.

2.1 PVM

PVM[4] został stworzony w 1989 roku w Oak Ridge National Laboratory. Począwszy od wersji 2.0 jest on dostępny publicznie i zaczął być wykorzystywany przez coraz większą liczbę aplikacji naukowych na całym świecie. The Parallel Virtual Machine (PVM) jest zbiorem narzędzi wykorzystujących model przekazywania komunikatów pozwalających na efektywne wytwarzanie aplikacji działających na dostępnym sprzęcie. Głównym celem PVM jest umożliwienie wykorzystania dowolnie dużej i dowolnie zróżnicowanej, zarówno pod względem architektury jak i systemu operacyjnego, grupy komputerów jako jednej wirtualnej maszyny. PVM w sposób niewidoczny da użytkownika zajmuje się routowaniem wiadomości, konwersją danych między różnymi systemami (np. little endian -> big endian i odwrotnie) oraz planowaniem zadań w sieci niekompatybilnych architektur sprzętowo/programowych.

Prosty a zarazem bardzo ogólny model przetwarzania dostarczany przez PVM pozwala na proste tworzenie aplikacji, w których procesy komunikują się za pomocą ustalonego interfejsu uzyskując tym samym pełną możliwość wymiany danych, synchronizacji czy dostępu do wspólnych zasobów. PVM dostarcza funkcji pozowalających na szeroki wachlarz komunikacji punkt-punkt czy też wysokopoziomowe metody typu broadcast czy bariera. Każdy proces w dowolnym momencie może zainicjalizować dowolny rodzaj komunikacji. Możliwe jest również dynamiczne zarządzanie strukturą klastra, tzn. w dowolnym momencie dowolny proces może zatrzymywać oraz uruchamiać procesy, dołączać lub odłączać komputery z maszyny wirtualnej.

Obecna, trzecia wersja zwana też PVM3, wspiera języki C, C++ oraz Fortran. Użytkownicy tworzący aplikację dla PVM mogą korzystać z dowolnego z tych języków a nawet na poszczególnych węzłach znajdować się mogą aplikacje napisane za pomocą różnych języków. Więcej na temat programowania z wykorzystaniem PVM znajduje się w dokumentacji oraz na stronie WWW http://www.csm.ornl.gov/pvm/.

PVM jest narzędziem ciągle rozwijanym. Doczekał sie wielu rozszerzeń i modyfikacji takich jak DAMPVM[5], pozwalający na automatyczne alokowanie

zasobów do procesów czy migrację procesów w zależności od bieżącego obciążenia poszczególnych węzłów.

PVM został tak zaprojektowany by jego instalacja i uruchomienie nie wymagałopraw administratora. Konfiguracja PVM składa się z nastepujących kroków, które należy wykonać na każdej z maszyn wchodzących w skład klastra:

1. ustawienie zmiennych $PVM_ROOT (katalog gdzie zostanie zainstalowany PVM) oraz $PVM_ARCH (architektura bieżącej maszyny, dla Linuxa jest to LINUX, pełna lista wartości znajduje się w dokumentacji).

2. rozpakowanie i zbudowanie za pomocą komendy 'make' właściwego narzędzia w katalogu wskazanego w poprzednim kroku za pomocą zmiennej systemowej $PVM_ROOT

3. zapewnienie każdemu węzłowi możliwość zalogowania się na każdy inny węzeł jako użytkownik uruchamiający maszynę wirtualną

Wykonawszy powyższe kroki użytkownik z poziomu dowolnej maszyny wchodzącej w skład klastra może uruchomić wirtualną maszynę za pomocą polecenia 'pvm'. Uruchomiona zostanie konsola zarządzająca maszyną wirtualną z poziomu której możliwe jest dodawanie i usuwanie maszyn, uruchamianie procesów i ich unicestwianie.

2.2 MPI

W przeciwieństwie do PVM, który jest narzędziem powstałym przy okazji badań, MPI jest standardem opracowanym przez Message Passing Interface Forum początkowo dla języków C oraz Fortran 77.[9] Pierwsza wersja specyfikacji zakończona została w kwietniu 1994 roku. W drugiej wersji specyfikacji standard rozszerzony został na języki C++ oraz Fortran 90. Standard MPI znacząco różni się od narzędzia PVM zakresem jaki pokrywa. Brakuje tutaj zarządzania procesami, pojęcia maszyny wirtualnej czy wsparcia dla interakcji z użytkownikiem. Standard definiuje składnię i semantykę samego jądra funkcji zapewniających przekazywanie komunikatów i jako taki może zostać z punktu widzenia użytkownika uznany za leżący w warstwie niższej niż np. PVM. Stąd możliwe jest zaimplementowanie PVM w oparciu o MPI. W skład standardu wchodzi między innymi komunikacja punkt-punkt, komunikacja grupowa oraz tworzenie grup procesów.

MPI jako standard został pozytywnie przyjęty, powstało wiele jego implementacji, zarówno komercyjnych jak i typu Open Source. Najbardziej znane otwarte implementacje to MPICH[10] oraz LAM-MPI[11][12].

Wybór pomiędzy implementacjami w dużej mierze zależy od zastosowania aplikacji oraz tego jak wydajna jest dana implementacja w przyjętym rozwiązaniu sprzętowym. Dzięki standaryzacji użytkownik ma możliwość bezproblemowego przetestowania swojej aplikacji na różnych implementacjach po prostu przekompilowując kod. Również instalacja samych środowisk jest bardzo prosta, zazwyczaj składa się z czterech kroków:

1. pobranie oraz rozpakowanie kodu źródłowego implementacji standardu MPI, oraz przejście do katalogu gdzie źródła zostały rozpakowane

2. wykonanie polecenia './configure –prefix=/katalog/do/którego/instalujemy'

3. wykonanie polecenia 'make'

4. wykonanie polecenia 'make install'

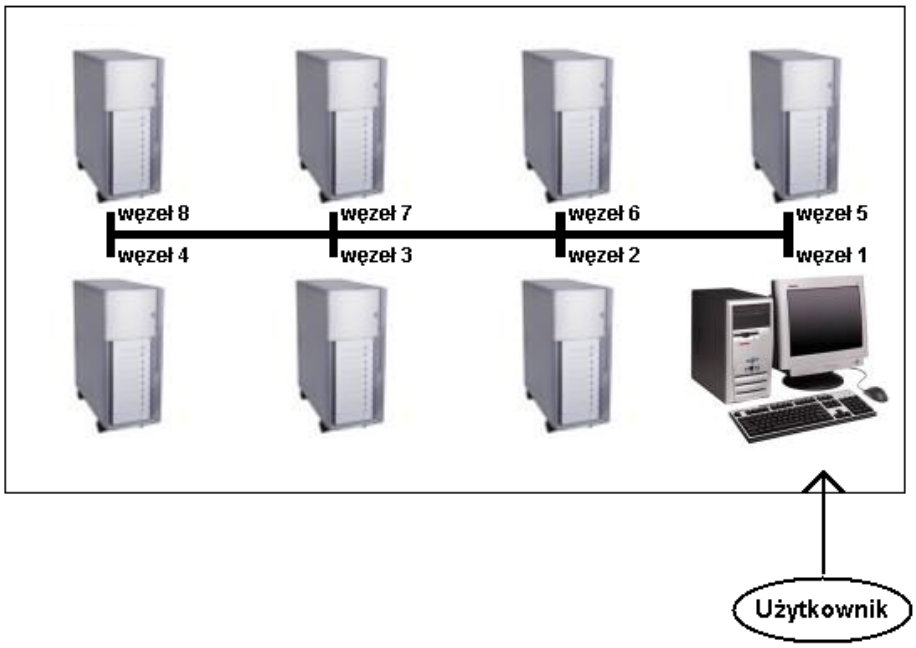
Proces ten należy wykonać na każdej z maszyn wchodzących w skład klastra. Definiowanie maszyn na jakich wykonany ma zostać właściwy kod zależy od implementacji standardu. Po wykonaniu czynności możemy przystąpić do kompilacji aplikacji wybranym kompilatorem i jej uruchomienia za pomocą polecenia 'mpirun'. Polecenie to musi mieć możliwość zalogowania się z dowolnej maszyny wchodzącej w skład klastra na dowolną inną wewnątrz klastra bez konieczności podawania hasła.

Najprostszą, a przy okazji i bezpieczną, metodą jest autoryzacja poprzez klucze publiczne dostępnego w oprogramowaniu openssh. Generacja kluczy odbywa się za pomocą polecenia 'ssh-keygen -t dsa -b 2048'. W trakcie generacji klucza, nie należy podawać 'passphrase' gdy zostaniemy o to zapytani. Spowoduje to wygenerowanie klucza publicznego oraz prywatnego. Klucz prywatny zapisany zostanie w pliku 'id_dsa' a klucz publiczny w pliku 'id_dsa.pub'. Klucz prywatny należy skopiować do katalogu '$HOME/.ssh/' a klucz publiczny skopiować na wszystkie serwery, na które chcemy mieć możliwość logowania bez hasła i wykonać na każdym z tych komputerów polecenie 'cat id_dsa.pub » $HOME/.ssh/authorized_keys'. Następnie należy w pliku '/etc/ssh/sshd_config' umieścić następujące linijki: 'PubkeyAuthentication yes' oraz 'AuthorizedKeysFile .ssh/authorized_keys'. Ostatnim krokiem jest zrestartowanie demona ssh poleceniem '/etc/init.d/sshd restart'. Dwa ostatnie kroki wymagają uprawnień administratora. Wykonując powyższą procedurę na każdym z węzłów zapewniamy bezproblemową i bezpieczną komunikację między procesami w klastrze.

3 openMOSIX

Podstawową wadą przedstawiona powyżej klastrów typu Beowulf jest konieczność tworzenia specjalnych aplikacji potrafiących wykorzystać możliwości klastra. Zazwyczaj wymaga to implementacji komunikacji za pośrednictwem PVM lub MPI. Konieczność ta mocno zawęża grupę odbiorców do badaczy i naukowców jednak uniemożliwia wykorzystanie klastrów użytkownikom o mniejszych zdolnościach programistycznych. Za pomocą mechanizmu przekazywania komunikatów nie jest możliwe po porstu połączenie kilku komputerów siecią LAN i uzyskanie przez to większej ich łącznej wydajności. Do takich właśnie zastosowań stworzono MOSIX. Jest to system, który pozwala na traktowanie klastra, z punktu widzenia działających na nim aplikacji, jako jednej maszyny wieloprocesorowej (Rysunek 2). Zadania uruchomione na stacji roboczej użytkownika są automatycznie rozsyłane na inne węzły klastra, gdzie zostaną wykonane. Po ich zakończeniu wynik obliczeń zwracany jest na maszynie, z której oryginalnie uruchomiono zadania.[2]

Różnica polega jedynie na tym, że w zwykłej maszynie wieloprocesorowej komunikacja między procesorami jest bardzo szybka, gdyż wszystkie procesory połączone są wspólną szyną. W przypadku MOSIX komunikacja ograniczona jest prędkością sieci więc zakres zastosowania klastra jest ograniczony do zadań nie zawierających dużej ilości komunikacji. Również, w przeciwieństwie do maszyn z technologią Hyper Threading czy Dual Core, openMosix nie potrafi rozpraszać wątków jednej aplikacji - migracji podlegać mogą jedynie całe



Rysunek 2: Z punktu widzenia użytkownika jak i wykonywanych zadań klaster zachowuje się jak maszyna wieloprocesorowa

procesy. Nic nie stoi jednak na przeszkodzie by jako węzły klastra zastosować maszyny wyposażone w odpowiednią technologię.

W 2001 roku projekt MOSIX zmienił licencję na niezgodną z GPL. W odpowiedzi na ten krok jeden z twórców MOSIX'a uruchomił aktywnie obecnie rozwijany projekt openMOSIX mający zapewnić ciągły dostęp użytkowników do wolnej wersji systemu. Wkrótce większość użytkowników przeszła z MOSIX na openMOSIX a sam system doczekał się wielu zaawansowanych funkcji.

Obecnie openMOSIX jest bardzo rozbudowanym systemem oferującym mechanizmy równoważenia obciążeń oraz migracji procesów. Co ważne potrafi migrować na mniej obciążone węzły praktycznie dowolne procesy linuxowe - nie jest wymagana żadna modyfikacja kodu czy ponowna kompilacja aplikacji. Możliwe jest nawet migrowanie procesów potomnych utworzonych instrukcją fork() na różne maszyny. Równocześnie systemy oparte o openMOSIX kontrolują stan poszczególnych węzłów i są w stanie reagować na dołączanie nowych maszyn do klastra czy upadek już do niego podłącznych w sposób nie zagrażający wykonywanej aplikacji. Przez to openMOSIX doskonale nadaje się do pracy wszędzie tam gdzie nie możemy pozwolić sobie na tworzenie specjalistycznego oprogramowania a konieczne jest jak najlepsze wykorzystanie posiadanego sprzętu.

Aby uruchomić klaster oparty o openMOSIX użytkownik musi posiadać:

- dowolną ilość komputerów połączonych siecią. Im bardziej wydajne łącza tym migracja, a tym samym wydajność całegoklastra będzie lepsza.
- na każdym węźle zainstalowane jądro z obsługą openMOSIX
- na każdym węźle zainstalowany pakiet openMOSIX Userland dostarcza-

jący zestaw narzędzi niezbędnych do pracy klastra

- na każdym węźle w pliku /etc/fstab umieszczony wpis 'mymfs /mfs fs dfsa=1 0 0' oraz utworzony katalog /mfs, w którym zamontowany zostanie wspólny system plików wszystkich węzłów

Komplet wymaganego oprogramowania dostępny jest pod adresem WWW http://openmosix.sourceforge.net. Można również zastosować już gotowe jądra systemu linux dostępne w praktycznie każdej większej dystrybucji. Po zainstalowaniu wymaganego oprogramowania użytkownik musi jeszcze tylko na każdym z węzłów w pliku /etc/mosix.map zdefiniować jakie komputery wchodzą w skład klastra. Ostatnim krokiem jest uruchomienie demona obsługującego migrację procesów poleceniem '/etc/init.d/openmosix start'.

Tak utworzony klaster jest gotowy do użycia. Za pomocą narzędzia 'mmon' (w niektórych dystrybucjach systemu Linux narzędzie to nazywa się 'mosmon') użytkownik może śledzić rozkład obciążenia na poszczególne węzły. Wystarczy teraz na jednym z węzłów uruchomić dowolną ilość procesów, których nie trzeba w żadne sposób wcześniej przygotowywać, i już po chwili zostaną one rozesłane na wszystkie węzły klastra w sposób równoważący obciążenie. W ten oto sposób niskim nakładem pracy i przy niskich kosztach można stworzyć wydajny klaster komputerowy ogólnego zastosowania.

4 Podsumowanie

Dzięki swojej otwartości i dostępności Linux stał się podstawowym narzędziem pracy w przypadku przetwarzania rozproszonego, zwłaszcza w badaniach naukowych czy wszelkiego rodzaju symulacjach. Mnogość dostępnych mechanizmów umożliwiających pracę rozproszoną jest ogromna, w tej pracy pobieżnie przedstawiono jedynie najważniejsze z nich. Praktycznie każdego dnia powstają nowe, coraz lepsze implementacje najróżniejszych standardów wzbogacający zestaw już dostępnych narzędzi i powodujących, że każdy potencjalny użytkownik znajdzie rozwiązanie w pełni odpowiadające jego potrzebom.

Literatura

[1] Linux Online, Inc., Linux Online FAQ, http://www.linux.org/lininfo/faq1.html

[2] Wawryniuk, I., 2005, Klastry oparte na systemie operacyjnym Linux jako rozwiązanie zwiększające wydajność i niezawodność systemów informatycznych, http://www.klastry.org.pl/

[3] Brown, R. G., 2004, Engineering a Beowulf-style Compute Cluster, Duke University Physics Department, Durham, http://www.phy.duke.edu/∼rgb/Beowulf/beowulf_book/beowulf_book/index.html

[4] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderam V., 1994, PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing, Massachusetts Institute of Technology, MIT Press, Cambridge, http://www.netlib.org/pvm3/book/pvm-book.html

[5] Czarnul, P., Krawczyk, H., 1999, Dynamic Assignments of Applications in Distributed Environments, Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Poland

[6] Imhof, M., Andrews, M., 2003, Klastry openMosix w Gentoo, Dokumentacja Gentoo Linux, http://www.gentoo.org/doc/pl/openmosix-howto.xml

[7] Lever, Ch., Linux NFS Frequently Asked Questions, http://nfs.sourceforge.net/

[8] Barr, T., Langfeldt, N., Vidal, S., McNeal, T., 2002, Linux NFS-HOWTO revision v3.1, http://nfs.sourceforge.net/nfs-howto/

[9] MPI Forum, The Message Passing Interface Standard, http://www.mpi-forum.org/docs/docs.html

[10] Gropp, W., Lusk, E., Installation and User's Guide to mpich, a Portable Implementation of MPI Version 1.2.5. The chp4 device for Workstation Networks, http://www-unix.mcs.anl.gov/mpi/mpich/docs/mpichman-chp4/mpichman-chp4.htm

[11] The LAM/MPI Team, Open Systems Lab, 2004, LAM/MPI User's Guide, http://www.lam-mpi.org/download/files/7.1.1-user.pdf

[12] The LAM/MPI Team, Open Systems Lab, 2004, LAM/MPI Installation Guide, http://www.lam-mpi.org/download/files/7.1.1-install.pdf

Indeks rzeczowy

Indeks autorów