STUDIA INFORMATICA 2002
Volume 23 Number 3 (50)

Tomasz Jordan KRUK, Marcin MAZUROWSKI
Politechnika Warszawska, Instytut Automatyki i Informatyki Stosowanej

PRAKTYCZNE ASPEKTY RÓWNOWAŻENIA OBCIĄŻENIA W ROZPROSZONYCH SYSTEMACH OPERACYJNYCH

Streszczenie. W artykule zaprezentowano dwa podejścia do wykorzystania rozproszonych zasobów obliczeniowych. Jako przeciwstawne do scentralizowanego, zaimplementowanego w środowisku rozproszonego systemu operacyjnego Amoeba, zaproponowano i zrealizowano, w tym samym środowisku, podejście alternatywne, rozproszone. Opisano algorytm i przedstawiono wyniki testów porównawczych.

PRACTICAL ASPECTS OF LOAD BALANCING IN DISTRIBUTED OPERATING SYSTEMS

Summary. Two approaches to the utilization of distributed computing environments have been presented. As opposite to the standard centralized one, implemented in Amoeba distributed operating system, a distributed approach has been proposed and implemented in the same environment. Some practical issues have been mentioned. The algorithm and comparison tests' results have been described.

1. Wstęp

Z wykonywaniem zadań współbieżnie, na wielu procesorach, w sposób przezroczysty dla użytkownika, nierozerwalnie związane jest zagadnienie równoważenia obciążenia, czyli przydziału procesów do procesorów w taki sposób, aby zadanie zostało wykonane jak najszybciej. Gdyby problem ten był trywialny, już teraz, pracując przykładowo na uczelni, moglibyśmy w łatwy i efektywny sposób wykorzystywać jej całe dostępne w danej chwili moce obliczeniowe.

2. Dwa podejścia do wykorzystania rozproszonych zasobów obliczeniowych

We współczesnych systemach obserwujemy dwa główne trendy w reprezentowaniu rozproszonych zasobów. Każdy z nich podyktowany jest zarówno filozofią dotyczącą przeznaczenia systemów komputerowych, jak i przekonaniami związanymi z obserwacją trendów na rynku komputerowym. Oba zaprezentowane podejścia w specyficznych warunkach bronią doskonale swoich racji. Jednak w praktyce spotykamy zazwyczaj mniej lub bardziej pośrednie rozwiązania.

2.1. Model stacji roboczych

Czysty model stacji roboczych nie jest właściwie nawet modelem rozproszonym. Zakłada on izolację użytkowników, ich procesów i wykonywanych przez nich aplikacji. Model ten jest w praktyce z powodzeniem stosowany w sieciowych systemach operacyjnych, gdzie użytkownicy współdzielą rozproszony system plików.

Po wprowadzeniu niewielkich modyfikacji system pracujący w oparciu o ten model może jednak awansować do miana rozwiązania rozproszonego. W tych rozproszonych systemach operacyjnych, w których zastosowano to podejście, implementowany jest zwykle mechanizm dynamicznego równoważenia obciążenia. Proces zaraz po stworzeniu lub w trakcie działania może zmienić środowisko wykonania. Zostaje przeniesiony na inny, zwykle bezczynny, procesor.

Zwolennicy tego typu rozwiązań argumentują swój wybór przede wszystkim twierdzeniem, że użytkownik systemu nie powinien doświadczać strat wydajności tylko i wyłącznie z powodu ogólnego przeciążenia całego systemu, czy intensywnej pracy innego użytkownika. Drugi argument dotyczy interfejsów użytkownika. Procesory i pamięć operacyjna są coraz tańsze, więc niskim kosztem możemy zwiększać moc obliczeniową stacji roboczych. Ponadto, przy pracy przeciętnego użytkownika większość mocy obliczeniowej zostanie zużyta na zapewnienie działania coraz to bardziej wyrafinowanym interfejsom graficznym użytkownika. Najlepiej, by wykorzystywany procesor znajdował się możliwie blisko konsoli użytkownika.

Trudno odmówić racji zwolennikom powyższego rozwiązania (np. twórcom systemu Sprite), jednak podejście to ma wiele wad. Implementując model stacji roboczych trudno spełnić założenie przezroczystości. A jest to przecież podstawowe wymaganie stawiane rozproszonym systemom operacyjnym. System zbudowany w omawianej konwencji nie może również stanowić dobrego środowiska dla typowych aplikacji rozproszonych i równoległych. Architektura tej klasy systemów nie jest w gruncie rzeczy zaprojektowana jako typowe

rozwiązanie rozproszone. Brak niskopoziomowej infrastruktury typowej dla systemu rozproszonego potęguje implementacyjne problemy dotyczące na przykład komunikacji procesów w nietypowych warunkach (po migracji).

2.2. Model puli procesorów

Konkurencyjny do poprzedniego modelu jest model puli procesorów. Zwolennicy tego rozwiązania wychodzą z podobnych założeń co twórcy poprzedniego, dochodzą jednak do innych wniosków. Ich zdaniem spadająca cena procesorów i pamięci zaowocuje w znaczną liczbę procesorów w systemach. Zakładają, że pewnego dnia liczba procesorów w systemie znacznie (wielokrotnie) przewyższy liczbę ich użytkowników. Naturalną więc rzeczą okaże się problem zaszeregowania procesu użytkownika na którymś z nich. W poprzednim rozdziale problem przydziału procesora dla procesu był sprawą wtórną. Nowy proces zawsze mógł znaleźć się na jednostce lokalnej. Przy podejściu puli procesorów nie jest oczywiste, gdzie powinien zostać uruchomiony nowy proces. Bardzo często zdarza się, że uruchomienie procesu zawsze będzie wiązało się z uruchomieniem na maszynie odległej (Amoeba).

Projekt obejmujący model puli roboczej z definicji będzie typowym mechanizmem rozproszonym. Nie ma problemów z osiągnięciem przezroczystości, nawet na poziomie niskich wywołań systemowych. W praktyce systemy o modelu puli procesorów znakomicie nadają się do systemów zorientowanych na przeprowadzanie obliczeń numerycznych. W oparciu o model puli procesorów działa system Amoeba. Natomiast model stacji roboczych lepiej nadaje się, z powodów czysto praktycznych, dla klasycznych instalacji, w których użytkownicy nie przeprowadzają skomplikowanych kalkulacji.

3. Amoeba i run serwer

W systemie Amoeba jest uruchamianych kilka serwerów systemowych realizujących różne elementy rozproszonego systemu operacyjnego. Jednym z nich jest centralny run serwer, którego zadaniem jest równoważenie obciążenia w systemie. W praktyce run serwer zbiera informacje o stanie i obciążeniach maszyn oraz realizuje zlecenia przydziału procesora dla nowego procesu. Mechanizm zawiaduje zbiorami maszyn (procesorów), stanowiącymi logiczne pule procesorów. Dane dotyczące nowego procesu są ograniczone, dlatego należy jak najlepiej wykorzystać informacje o środowisku jego wykonania. W systemie Amoeba nie ma oddzielnego mechanizmu do zbierania informacji na temat obciążenia maszyn. Dlatego run serwer wykorzystuje niezależny moduł, którego zadaniem jest utrzymywanie danych dla modułu właściwego - implementującego algorytm przydziału procesora.

3.1. Cechy algorytmu run serwera

Algorytm serwera run jest **scentralizowany**. Program działa w jednej kopii, na centralnej maszynie. Dojście do run serwera opublikowane jest w rozproszonym systemie plików. Mechanizm jest **globalny**. W jednym miejscu zgromadzone są dane o całym systemie, o wszystkich pulach i rozważanych jednostkach obliczeniowych. Za każdym razem rozważane są wszystkie maszyny o określonych architekturach. Rzadko zdarza się, że dwa kolejne procesy znajdą się na tej samej maszynie. W systemie nie istnieje nawet pojęcie maszyny macierzystej procesu. Użytkownik rejestruje się do systemu jako do całości, a nie na określonej maszynie.

Zaimplementowany został **heurystyczny** algorytm wyboru najlepszej maszyny. Polityka przeniesień jest **statyczna**. Raz zaszeregowany proces nie zmienia miejsca swojego wykonania i kończy się na maszynie, na której się rozpoczął.

3.2. Badanie obciążenia procesorów

Realizacja badania maszyn powinna zużywać jak najmniej zasobów systemu. W tym celu zaimplementowano w systemie asynchroniczny mechanizm z kolejkami zadań. Zlecenie badania maszyny umieszczane jest w kolejce w ustalonych odstępach czasu. Co 5 sekund odpytywane są maszyny działające. Co 10 sekund maszyny, które przestały odpowiadać. Każda z takich maszyn do pewnego momentu uznawana jest jeszcze za działającą. Co 60 sekund następuje próba odpytania maszyn uznanych za wyłączone. Wartości te są na stale ustalone w kodzie i nie można ich zmieniać z poziomu poleceń systemowych.

3.3. Wybór procesora

Algorytm wyboru najlepszego procesora dla procesu bierze pod uwagę szereg parametrów zarówno statycznych - ustalanych przez administratora systemu – jak i dynamicznych - ustalanych przez serwer podczas jego wykonywania.

Wybór maszyny dla procesu związany jest z wyliczeniem dla każdej maszyny wartości współczynnika przydatności maszyny i wybrania tej, dla której jego wartość jest największa. Geneza ostatecznej postaci wzoru na współczynnik ma zapewne charakter eksperymentalny, nie ma jednak, jak przyznają sami autorzy, żadnego uzasadnienia teoretycznego.

3.4. Problemy z run serwerem

Mechanizm run serwera niewątpliwie sprawdza się w konfiguracji z pulą procesorów w sytuacji, kiedy procesorów jest o wiele więcej niż użytkowników. Konfiguracja laboratorium,

w którym zainstalowano system, jest skrajnie inna - dwa przeciążone serwery systemowe plus dwanaście stacji roboczych. W takiej konfiguracji standardowe rozwiązanie z run serwerem zawodzi.

Przede wszystkim zawodzi wydajność run serwera. Implementuje on wprawdzie rozproszony dostęp do zasobów, jednak stanowi składową centralną. W rozważanej konfiguracji działa na tej samej maszynie co serwery implementujące system plików. Każde wywołanie to kilka zdalnych wywołań procedury.

Niezawodność rozwiązania również pozostawia wiele do życzenia. Mechanizm zbierania danych spełnia swoje zadanie w warunkach niedociążenia. Jednak w sytuacji intensywnej pracy wszystkich stacji roboczych zawodzi. Po pierwsze, zbierane dane znacznie tracą swoją aktualność (mechanizm asynchroniczny). Dochodzi często do najgorszego z możliwych przypadków, kiedy run serwer kieruje proces na maszynę, która uległa awarii (czy została wyłączona).

Kolejny problem to globalne podejście do wszystkich procesorów w systemie. Procesy użytkownika, często proces sesji czy powłoki, uruchamiane są na zdalnych maszynach. Użytkownik nie kontroluje w żaden sposób maszyny zdalnej. W praktyce jest to maszyna, przy której zasiada inny użytkownik. Jeżeli ten zakończy pracę i nieopatrznie wyłączy maszynę, pierwszy użytkownik nie może kontynuować pracy. W najlepszym przypadku kończy się zablokowaniem terminalu pechowego użytkownika.

4. Konkurencyjne rozwiązanie równoważenia obciążenia w systemie Amoeba

Przed przystąpieniem do projektowania nowego rozwiązania i wyborem algorytmu sformułowane zostały założenia, które powinien nowy mechanizm spełniać. Wszystkie założenia podporządkowane zostały nadrzędnemu celowi, którym było uzyskanie mechanizmu wydajniejszego i mniej zawodnego niż run serwer.

4.1. Cechy nowego rozwiązania

Proponowane rozwiązanie jest **rozproszone**. Na każdej maszynie pracuje lokalny scheduler (nazwiemy go agentem danej stacji roboczej), który decyduje, gdzie wysłać dany proces. Mechanizm jest **lokalny** - jeżeli maszyna lokalna jest niedociążona, proces zostaje uruchomiony lokalnie.Proces wysyłany jest na inną maszynę tylko w przypadku przeciążenia maszyny lokalnej. Projektowane rozwiązanie musi być **wydajne**. Oczywiście możliwości

przewidywania wydajności są bardzo ograniczone, przy wyborze algorytmu uwzględniono jednak postulaty, które popierają prostotę w systemach operacyjnych.

Rozwiązanie jest **niezawodne**. Nacisk został położony na zniwelowanie problemów związanych z działaniem run serwera. Nowy mechanizm powinien być **samodzielny**, tzn. może działać i w pełni realizować swoje zadania po wyłączeniu run serwera. Jednocześnie obecność obu mechanizmów w systemie nie powinna powodować żadnych konfliktów. Zakładamy, że run serwer może działać i realizować żądania maszyn, na których nie wdrożono nowego rozwiązania.

Zdecydowana większość algorytmów równoważenia obciążenia w systemach to algorytmy heurystyczne, działające w oparciu o schematy często intuicyjne, opracowane i przetestowane, jednak bez matematycznych twierdzeń podstaw. Algorytm, który zaproponowano, również jest **heurystyczny**. Nowe rozwiązanie zostało jednak oparte na pewnych podstawach teoretycznych.

Zaimplementowano algorytm wyboru maszyny **inicjowany przez nadawcę**. Maszyna przeciążona decyduje o pozbyciu się procesu i poszukuje maszyny zdalnej. Podobnie jak run serwer nowy mechanizm uruchamiany jest w **przestrzeni użytkownika**. Zachowany zostaje główny **postulat przezroczystości**. Użytkownik ma wrażenie, że rejestruje się do monolitycznego systemu. Jego procesy uruchamiane są na różnych maszynach bez jego ingerencji.

4.2. Opis nowego rozwiązania

Centralny run serwer zastąpiony został grupą rozproszonych agentów. Na każdej maszynie działa bliźniacza kopia agenta. Agentów rozróżniamy jedynie po portach nasłuchiwania i maszynach, na których są uruchomieni. Adres portu jest wyliczany zawsze na podstawie nazwy komputera, która musi być unikatowa w całym systemie. Jest to wymaganie samego systemu, nie wiążą się więc z tym wymaganiem żadne dodatkowe konsekwencje. Uruchomienie kopii agenta na zadanej maszynie, zwykle nowej, która została włączona i zarejestrowana w systemie, odbywać się może na dwa sposoby.

Agenci mają zdolność rozmnażania się. Monitorują stan systemu i odpytują innych agentów o ich stan. W przypadku stwierdzenia braku programu na zadanej maszynie agent spróbuje uruchomić na niej swoją kopię. Używa do tego specjalnego wywołania systemowego lub bezpośrednio kopiuje swój kod na zdalny procesor, uniezależniając się w ten sposób od systemu plików.

Druga możliwość - to wykorzystanie zmodyfikowanych procesów systemowych login. Obecność agenta na maszynie nie jest niezbędna do czasu, gdy na stacji roboczej nie zarejestruje się użytkownik. Do tej chwili brak agenta powoduje tylko ogólne zmniejszenie

mocy obliczeniowej całego systemu. Prawdopodobnie wkrótce któryś z odległych agentów powieli się na taką maszynę. Jednak w przypadku, gdy przy stanowisku rejestruje się użytkownik, musimy mieć pewność, że zostanie uruchomiony agent, który będzie obsługiwał jego zlecenia – stąd uruchamianie agentów po pomyślnej autoryzacji użytkownika.

Na algorytm składają się dwie fazy. Podczas fazy lokalnej maszyna sprawdza lokalne obciążenie. Decyduje, czy jest przeciążona czy niedociążona. W przypadku niedociążenia maszyny lokalnej agent natychmiast wskazuje lokalny procesor jako miejsce wykonania procesu. W przeciwnym przypadku przechodzi do drugiej fazy, w której rozważane jest uruchomienie procesu na jednej z pozostałych maszyn całego systemu. Stosowany jest właściwy algorytm równoważenia obciążenia opisany poniżej.

4.2.1. Zasada działania właściwego algorytmu

Wybrany algorytm jest oparty na teorii stochastycznych automatów uczących. Algorytm ten nie wymaga wyrafinowanych obliczeń i kalkulacji, za to mocno osadzony jest w realiach matematycznych. Prezentacja stosownej teorii oraz dowodów poprawności i stabilności samego algorytmu znajduje się w pozycji [4] spisu literatury.

Zasada działania rozwiązania opiera się na wyborze optymalnego rozwiązania ze zbioru dopuszczalnych akcji. Agent może podjąć akcję postaci: wybierz maszynę x, gdzie x jest maszyną zarejestrowaną w systemie. Automat stochastyczny rozwiązuje zadany problem w następujący sposób. Do każdej możliwej akcji przypisany jest współczynnik prawdopodobieństwa z nią związany. Na początku wszystkie współczynniki są równe, ponieważ nie wiadomo nic o systemie. W oparciu o te współczynniki losowana jest maszyna zdalna. Za każdym razem obserwowana jest odpowiedź maszyny, która relacjonuje trafność wyboru. Na podstawie tej odpowiedzi współczynniki wyboru zmieniają się zgodnie z przyjętą heurystyką. Przy kolejnym wyborze algorytm używa nowych współczynników i cała procedura powtarza się.

W rozwiązaniu zastosowano wariant rozszerzony, który wykorzystuje wektor stanów systemu. W każdym stanie zdefiniowany jest oddzielny wektor współczynników prawdopodobieństwa. Stan systemu determinuje wektor, który zostanie użyty. Stan systemu jest znany dzięki przekazywaniu komunikatów o treści: host x jest przeciążony/niedociążony. Agent przeszukuje system w poszukiwaniu niedociążonego procesora. Jeżeli go znajdzie, dokonuje odwzorowania między kolejnym numerem maszyny a wektorem stanu systemu.

4.2.2. Wektory stanów

Algorytmy często wykorzystują element losowości. Jednak prawdopodobieństwa wybrania poszczególnych maszyn są zwykle takie same. W najgorszym razie zadanie będzie przerzucane z maszyny na maszynę do wyczerpania maksymalnej liczby prób. W pewnych

przypadkach, jak w klasycznym algorytmie inicjalizowanym przez nadawcę, może to prowadzić do niekontrolowanego ruchu w sieci. Opisywany tutaj algorytm wykorzystuje wprawdzie element losowości, ale przypisuje każdej maszynie odpowiednie wagi, tzn. maszyna A zostaje wybrana z prawdopodobieństwem X. Prawdopodobieństwo to jest przechowywane we wspomnianych wcześniej wektorach opisujących stany systemu. Prawdopodobieństwo to jest również na bieżąco aktualizowane z uwzględnieniem odpowiedzi wybranej maszyny i zastosowanej heurystyki.

4.2.3. Proces uczenia się

Każdy agent przechowuje kilka wektorów prawdopodobieństw. Każdy z nich odzwierciedla bliżej nieokreślony stan systemu. Agent wybiera stan systemu na podstawie zapytań do zdalnych agentów. Losuje stację roboczą używając współczynników wybranego wektora stanu. Odpytuje agenta zdalnego o trafność wyboru. Na podstawie odpowiedzi zmieniany jest wybrany wektor współczynników. Niech p_j (n) oznacza prawdopodobieństwo wybrania maszyny j w czasie n, wtedy p_j (n + 1) oznacza to samo prawdopodobieństwa w czasie n + 1 . Załóżmy, że w chwili n została wybrana maszyna i. Współczynniki prawdopodobieństwa są zmieniane w następujący sposób:

- gdy wybór nie przeciążył maszyny zdalnej (nagroda):

$$p_j(n+1) = p_j(n) - f[p_j(n)] \text{dla } j \neq i$$

$$p_i(n+1) = p_j(n) + \sum f[p_j(n)]$$

- gdy wybór doprowadził do przeciążenia maszyny zdalnej (kara):

$$p_j(n+1) = p_j(n) + g[p_j(n)] \text{dla } j \neq i$$

$$p_i(n+1) = p_j(n) - \sum g[p_j(n)]$$

gdzie f i g są odpowiednio funkcjami kary i nagrody. Stosujemy heurystykę liniową:

$$f(x) = A * x, \text{gdzie } 0 < A < 1$$

$$g(x) = \frac{B * x}{r-2}, \text{gdzie } 0 < B < 1$$

gdzie r jest liczbą maszyn zarejestrowanych w systemie, a A i B stałymi współczynnikami algorytmu.

5. Wyniki testów porównawczych

Wszystkie testy zostały zrealizowane w możliwie identycznym środowisku. W eksperymencie wykorzystano komputery zainstalowane w laboratorium zorganizowanym w sali numer 25 gmachu WEiTI Politechniki Warszawskiej. Środowisko objęło 12 jednakowych pod względem parametrów maszyn. Są to komputery klasy PC połączone w jedną sieć. Na maszynach tych zainstalowano system Amoeba w wersji 5.3.

Najpierw porównano wydajność standardowego i nowego rozwiązania, zakładając bezawaryjność maszyn. Poza przypadkiem obciążenia systemu plików rozproszony agent nie ustępuje rozwiązaniu run serwera. Oferuje mniejszą wydajność tylko w przypadku obciążenia pamięci przy 5 równolegle pracujących użytkownikach. Różnica jest mimo wszystko bardzo mała, wręcz pomijalna. Rozproszony mechanizm równoważenia obciążenia wypada bardzo dobrze przy większym obciążeniu systemu (8 równoległych użytkowników). Pozostaje interpretacja znacznej różnicy wydajności przy obciążeniu systemu plików. Serwer plików (tzw. bullet server), choć implementuje rozproszony system plików, z punktu widzenia wydajności systemu jest składową centralną. Jeżeli te same żądania napływają do niego stosunkowo rzadko, potrafi on zrealizować je szybciej, niż gdyby napłynęły w stosunkowo krótkim czasie. Takie zachowanie systemu plików wyjaśnia spadek wydajności mechanizmu wykorzystującego rozproszonego agenta, które de facto jest rozwiązaniem szybszym. Szybciej wybiera maszyny dla procesów, w związku z czym następuje nagromadzenie dużej liczby żądań do systemu plików w krótkim czasie.

Następną grupę testów przeprowadzono w środowisku, w którym maszyny często były wyłączane. Rozproszony agent radzi sobie podczas awarii 2 razy lepiej. Związane jest to przede wszystkim z lokalnym charakterem nowego rozwiązania. W przypadku run serwera ginie dużo więcej procesów testowych, nie mówiąc już o przypadkach przedwczesnego zakończenia całego testu, czy wręcz zakończeniu się sesji użytkownika usytuowanej na komputerze, który uległ awarii. Z sytuacją taką spotykaliśmy się w praktyce wielokrotnie. Kilkuletnie doświadczenia w laboratorium pokazały, że opisana właściwość systemu była wyjątkowo uciążliwa. Mechanizm run serwera nie uwzględnia w swoich kalkulacjach informacji o tym, z której maszyny nadeszło żądanie. W związku z tym procesy rozmieszczane były równomiernie w całym systemie, jednak w taki sposób, że procesy użytkownika, w znakomitej większości, mogły zostać uruchomione na odległej maszynie. Użytkownik nie może kontrolować maszyn odległych. W przypadku awarii takiej maszyny, lub po jej wyłączeniu przez innego użytkownika, możliwość dalszej pracy staje pod znakiem zapytania. W praktyce sytuacje takie kończyły się zwykle zablokowaniem terminalu lub wręcz zakończeniem sesji użytkownika, co związane było z restartem jego systemu.

LITERATURA

1. Tanenbaum A. S.: Rozproszone systemy operacyjne. PWN, Warszawa 1997.
2. Gościński A.: Distributed Operating Systems - The Logical Design. Addison-Wesley Publishers 1991.
3. Chow R., Johnson T.: Distributed Operating Systems & Algorithms. Addison-Wesley 1997.
4. Mirchandaney R., Stankovic J. A.: Using Stochastic Learning Automata for Job Scheduling in Distributed Processing Systems. Journal of Parallel and Distributed Computing, 1986.
5. Kunz T.: The Learning Behaviour of Scheduler using Stochastic Learning Automaton. 1991.
6. Babicz A.: Migracja procesów w systemie rozproszonym Amoeba. Praca magisterska, Politechnika Warszawska 2000.
7. Mazurowski M.: Równoważenie obciążenia w rozproszonych systemach operacyjnych. Praca magisterska, Politechnika Warszawska 2002.

Recenzent: Dr inż. Rafał Cupek

Wpłynęło do Redakcji 25 marca 2002 r.

Abstract

Two approaches to the utilization of distributed computing environments have been presented. Pros and cons of workstation model and processor pool model are considered. Next, the run server implemented in Amoeba distributed operating system has been described. Run server as a centralized solution does not behave correctly when the load of user processes is high. Moreover, it's resistance to hardware turn-off or failures is too weak.

Therefore, a new approach to load sharing has been proposed and implemented. The new solution is distributed and faster (better) reacts to hardware failures. There is some theoretical background (stochastic learning automates) behind the proposed algorithm, although in some aspects it is still heuristic. The algorithm, which has been implemented under Amoeba, is thoroughly described. Finally, comparison test results of the standard run server and the new approach under Amoeba prove some advantages of the new solution.