STUDIA INFORMATICA 2002
Volume 23 Number 2B (49)

Krzysztof ZATWARNICKI
Politechnika Opolska, Katedra Automatyki,Elektroniki i Informatyki

ADAPTACYJNY ALGORYTM RÓWNOWAŻENIA OBCIĄŻEŃ LOKALNIE ROZMIESZCZONYCH SERWERÓW WWW

Streszczenie. W artykule został opisany nowy adaptacyjny algorytm równoważenia obciążeń lokalnie rozmieszczonych serwerów WWW. Na wstępie przedstawione są sposoby równoważenia obciążeń dla klastra serwerów. Następnie opisany jest nowy algorytm równoważenia obciążeń oraz wyniki przeprowadzonych eksperymentów. Na końcu przedstawione są wnioski wynikające z eksperymentów.

ADAPTIVE LOAD BALANCING ALGORITHM FOR LOCALLY DISTRIBUTED WEB SYSTEMS

Summary. This paper describes new adaptive load balancing algorithm for locally distributed web systems. At first we discuss methods, which can be used in order to balance load in cluster of web servers. Next we characterize a new algorithm and preset results of curried out experiments. Finally we discuss results of experiments.

1. Wstęp

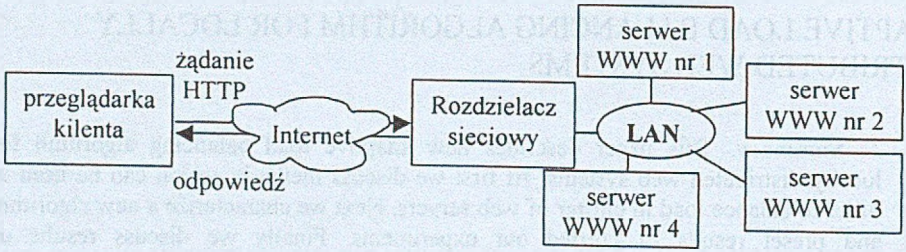
Obecnie globalna sieć komputerowa - Internet, rozwija się bardzo dynamicznie. Użytkownicy korzystają z Internetu dla rozrywki, w celach biznesowych, aby pozyskać niezbędne informacje do nauki lub badań. Niestety, nieraz zdarza się, że internauci muszą dość długo oczekiwać na pozyskanie pożądanych informacji, niejednokrotnie czas oczekiwania jest tak długi, że rezygnują z tych informacji. Aby zmniejszyć czas obsługi żądań klienta można zmodernizować serwer WWW i zwiększyć przepustowość sieci. Inną możliwością byłoby zastosowanie grupy serwerów WWW pracujących jako jedna witryna. Obecnie są stosowane dwie grupy rozwiązań umożliwiające równoważenie obciążeń serwerów WWW. Pierwsza z nich to przełączanie połączeń dla 4 warstwy sieciowej (ang.

Level 4 Web switching). W rozwiązaniu tym użytkownik łączy się z urządzeniem nazywanym rozdzielaczem sieciowym. Rozdzielacz podejmuje decyzje o przekierowaniu zapytania użytkownika nie wiedząc, jakie dane klient chce pobrać z serwera WWW, a następnie wysyła żądanie użytkownika do wybranego serwera. Jest kilka algorytmów, które rozdzielacz sieciowy może zastosować do doboru serwera WWW, jednym z najbardziej popularnych jest Weighted *Round Robin*.

Druga grupa rozwiązań nazywana jest przełączaniem połączeń dla 7 warstwy sieciowej (ang. *Level 7 web switching*). W rozwiązaniu tym rozdzielacz sieciowy podejmuje decyzje o przekierowaniu żądania użytkownika na podstawie nagłówka HTTP. Jednym z najlepszych algorytmów tej grupy jest algorytm LARD.

W artykule tym zostanie zaprezentowany nowy adaptacyjny algorytm przełączania połączeń dla 7 warstwy sieciowej. Algorytm ten został nazwany Tablice Obciążeń (ang. Load Tables) i w skrócie będzie oznaczony TO (ang. LT). Powinien on być stosowany w rozdzielaczach wykorzystujących architekturę dwukierunkową.

Rysunek 1 przedstawia schemat architektury dwukierunkowej. W rozwiązaniu tym klient nawiązuje połączenie z rozdzielaczem sieciowym, następnie przekazuje mu nagłówek żądania HTTP. Rozdzielacz sieciowy wybiera odpowiedni serwer WWW, nawiązuje z nim połączenie i pobiera z niego odpowiednie dane, które są przekazywane klientowi.

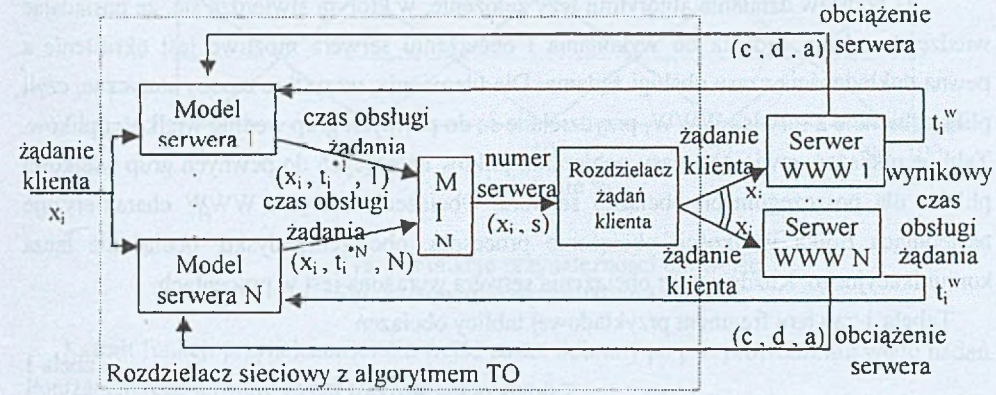


Rys. 1. Architektura dwukierunkowa
Fig. 1. Two-ways architecture

2. Opis algorytmu TO

Celem działania proponowanego algorytmu jest minimalizacja czasu obsługi pojedynczego żądania przez serwis WWW. Jako czas obsługi żądania klienta przyjmuje się czas od momentu wysłania przez rozdzielacz sieciowy pierwszej ramki TCP nawiązującej połączenie z serwerem WWW do momentu wysłania ostatniej ramki TCP zamykającej połączenie. Schemat ideowy rozdzielacza sieciowego z zaimplementowanym algorytmem TO przedstawia rysunek 3. Jak wynika z przedstawionego schematu, w serwisie pracuje N serwerów WWW. Rozdzielacz sieciowy składa się z wielu modułów. Pierwsza grupa

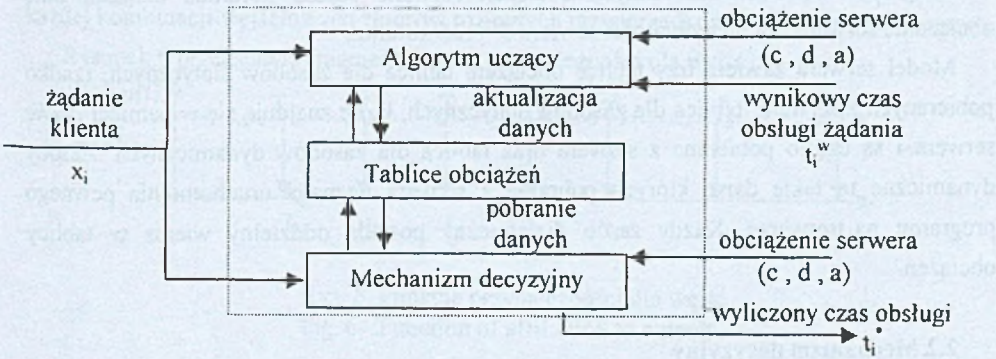
modułów to modele serwerów, jest ich tyle, ile prawdziwych serwerów pracujących w grupie. Każdy model przyporządkowany jest do pewnego serwera.



Rys. 3. Schemat ideowy rozdzielacza sieciowego z algorytmem TO
Fig. 3. Diagram of web switch with LT algorithm

Modele serwerów wyliczają czas realizacji ti* żądania klienta x_i dla poszczególnych serwerów na podstawie obciążeń (c,d,a) tych serwerów. Następnie moduł MIN wybiera serwer, którego wyliczony czas realizacji żądania jest minimalny. Rozdzielacz żądań klienta wysyła żądanie do wybranego serwera. Wynikowy czas obsługi żądania trafia do modelu serwera przyporządkowanego do wybranego serwera.

Rysunek 4 prezentuje schemat ideowy modelu serwera.



Rys. 4. Schemat ideowy modelu serwera
Fig. 4. Diagram of server model

Tablice obciążeń będące częścią modelu serwera są bazą danych przechowującą informacje o modelowanym serwerze. Mechanizm decyzyjny wylicza przewidywany czas obsługi żądania na podstawie żądania użytkownika, aktualnego obciążenia serwera oraz informacji zgromadzonych w tablicach obciążeń. Algorytm uczący jest odpowiedzialny za aktualizacje informacji w tablicach obciążeń.

2.1. Tablice obciążeń

U podstaw działania algorytmu leży założenie, w którym stwierdza się, że posiadając wiedzę o rodzaju żądania do wykonania i obciążeniu serwera możliwe jest określenie z pewną dokładnością czasu obsługi żądania. Dla ułatwienia, wszystkie zasoby statyczne, czyli pliki pobierane z serwisu WWW, przydzielane są do pewnych grup według wielkości plików. Tablice obciążeń zawierają czasy pobierania plików należących do pewnych grup wielkości plików dla poszczególnych obciążeń serwera. Obciążenie serwera WWW charakteryzuje następująca trójka wielkości: obciążenie procesora, obciążenie dysku, obciążenie łącza komunikacyjnego. Każda z miar obciążenia serwera wyrażona jest w procentach.

Tabela 1 zawiera fragment przykładowej tablicy obciążeń

Tabela 1

Tablica obciążeń

Rozmiar pliku	Serwer 1							
	Obciążenie serwera							
	0,0,0	1,0,0	...	2,1,1	2,2,1	2,2,2	...	4,4,4
	Czasy obsługi żądania							
0-4 kB	15	20	...	40	50	80	...	150
4-8 kB	20	30	...	120	125	140	...	160

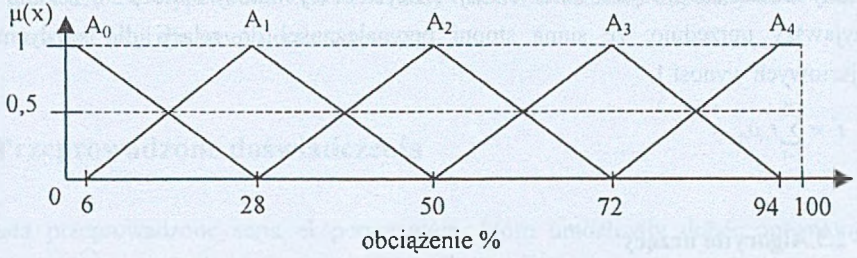
Obciążenie konkretnego elementu serwera może przyjąć w tablicy obciążeń jedynie 5 wartości od 0 do 4. Każdy wiersz tablicy zawiera czasy obsługi zasobów należących do tej samej grupy wielkości dla różnych obciążeń serwera. Każda kolumna oznacza inne obciążenie serwera. Tablica obciążeń zawiera 5^3=125 kolumn.

Model serwera zawiera trzy tablice obciążeń: tablica dla zasobów statycznych, rzadko pobieranych z serwera; tablica dla zasobów statycznych, które znajdują się w pamięci *cache* serwera i są często pobierane z serwera oraz tablica dla zasobów dynamicznych. Zasoby dynamiczne to takie dane, których pobranie z serwera wymaga uruchomienia pewnego programu na serwerze. Każdy zasób dynamiczny posiada oddzielny wiersz w tablicy obciążeń.

2.2. Mechanizm decyzyjny

Mechanizm decyzyjny zawiera algorytm z logiką rozmytą. Do obliczania czasu obsługi żądania wykorzystany został klasyczny model Mandaniego.

Wejściami w opisywanym modelu są obciążenia procesora, dysku i łącza komunikacyjnego serwera. Wartości wejść są podane w procentach. Istnieje pięć zbiorów rozmytych każdego z trzech wejść i są one oznaczane A0 .. A4. Kształt funkcji przynależności poszczególnych wejść do zbiorów rozmytych jest identyczny dla każdego wejścia. Rysunek 5 przedstawia kształt funkcji przynależności dla wejść.



Rys. 5. Funkcje przynależności dla wejść
Fig. 5. Function of affiliation to inputs

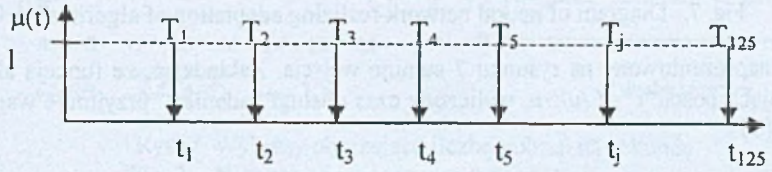
Kształt funkcji przynależności dla wejść został dobrany po przeprowadzeniu wielu badań. Poniżej zaprezentowany został fragment bazy reguł.

R1: JEŚLI (c=Ac0) I (d=Ad0) I (a=Aa0) TO (t=T1)
R2: JEŚLI (c=Ac0) I (d=Ad0) I (a=Aa1) TO (t=T2)
R3: JEŚLI (c=Ac0) I (d=Ad1) I (a=Aa1) TO (t=T3)
...
Rj: JEŚLI (c=Acl) I (d=Adm) I (a=Aan) TO (t=Tj)
...
R125: JEŚLI (c=Ac4) I (d=Ad4) I (a=Aa4)TO(t=T125)

c – obciążenie procesora; d – obciążenie dysku; a – obciążenie łącza komunikacyjnego t – czas obsługi żądania; Ac0,...,Ac4 – zbiory rozmyte obciążenia procesora; Ad0,...,Ad4 – zbiory rozmyte obciążenia dysku Aa0,...,Aa4 – zbiory rozmyte obciążenia łącza komunikacyjnego; T1,...,T125 – zbiory wyjścia t

Baza reguł zawiera 125 reguł i jest zarówno lingwistycznie, jak i numerycznie kompletna, co oznacza, że dla każdego stanu wejść przyporządkowany jest jeden stan wyjść oraz do każdej kombinacji wejściowych zbiorów rozmytych przyporządkowany jest zbiór wyjściowy.

Rysunek 6 przedstawia fragment funkcji przynależności dla wyjść.



Rys. 6. Funkcje przynależności dla wejść
Fig. 6. Function of affiliation to outputs

Zbiorami wyjść są singletony, czyli jednoelementowe zbiory rozmyte, dla których stopień przynależności wynosi 1. Poszczególne wartości $t_1...t_{125}$ wskazywane przez zbiory $T_1...T_{125}$ są czasami zapisywanymi w tablicach obciążeń.

Dla przedstawionych reguł i przykładowych argumentów wejściowych c=c*, d=d*, a=a* stopień przynależności do relacji Rj obliczyć można ze wzoru (1).

$$\mu_{Rj}(c^*,d^*,a^*) = \mu_{Acl}(c^*)*\mu_{Adm}(d^*)*\mu_{Aan}(a^*) \quad (1)$$

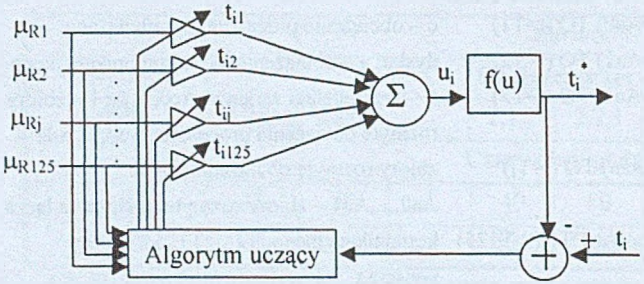
Ponieważ funkcja przynależności wyjść jest w postaci singletonów, dlatego do obliczenia wyjścia t^* została wykorzystana metoda wysokości, która jest uproszczoną dyskretną wersją

metody środka sum. Wynik defazyfikacji z użyciem tej metody oblicza się według wzoru 2 przyjąwszy uprzednio, że suma stopni przynależności do relacji dla każdych danych wejściowych wynosi 1.

$$t^* = \sum_{j=1}^{125} t_j \mu_{Rj} \tag{2}$$

2.3. Algorytm uczący

Zadaniem modułu Algorytmu Uczącego jest modyfikacja Tablic Obciążeń tak, by w tablicach tych znalazły się aktualne czasy obsługi żądania klienta. Powyżej mechanizm decyzyjny przedstawiony został dla ułatwienia jako algorytm z logiką rozmytą, teraz mechanizm decyzyjny zaprezentowany zostanie jako sieć neuronowa z jednym neuronem.



i – indeks oznaczający numer kolejnego żądania użytkownika

μ_{Rj} – stopień przynależności do relacji Rj dla argumentów wejściowych $c=c_i^*$, $d=d_i^*$, $a=a_i^*$

t_i^* – czas obsługi żądania wyliczony w algorytmie

t_i^w – wynikowy czas obsługi żądania zrealizowanego przez dany serwer

t_{ij} – waga wejścia μ_{Rj} w kroku „i", czas obsługi zadania dla konkretnego obciążenia serwera zapisany w tablicy obciążeń

Rys. 7. Schemat sieci neuronowej realizującej adaptacje algorytmu
Fig. 7. Diagram of neural network realizing adaptation of algorithm

Neuron zaprezentowany na rysunku 7 sumuje wejścia. Zakładając, że funkcja aktywacji ma następującą postać $t^*=f(u)=u$, wyliczony czas obsługi zadania t^* przyjmuje wartość jak we wzorze (3)

$$t^* = \sum_{j=1}^{125} t_j \mu_{Rj} \tag{3}$$

Do uczenia sieci zastosowana została tradycyjna metoda nauczania z nauczycielem – metoda propagacji wstecznej błędu. Wzór (4) przedstawia sposób obliczania nowej wartości $t_{j(i+1)}$ zmodyfikowanej wagi.

$$t_{j_{(i+1)}} = \begin{cases} t_{j_i} + \eta \mu_{Rj} (t_i^* - t_i^w) \text{ gdy } \left| t_i^* - t_i^w \right| \le p * t_i^w \\ t_{j_i} \text{ gdy } \left| t_i^* - t_i^w \right| > p * t_i^w \end{cases} \tag{4}$$

We wzorze (4) p jest pewną stałą i $p<1$. Współczynnik uczenia η przyjmuje wartość 0,1.

Przedstawiona sieć neuronowa uczy się przez cały czas działania algorytmu TO. Nie ma typowej dla sieci neuronowych fazy uczenia się i pracy. Na początku działania algorytmu

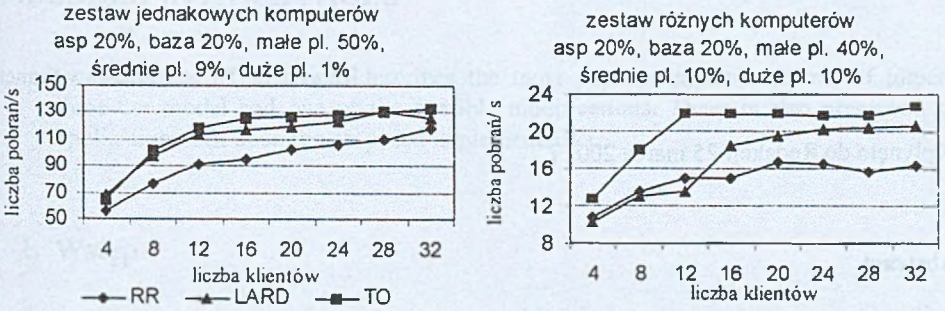
Tablice Obciążeń są wypełnione zerami. Każdy wiersz z czasami zapisanymi w tablicy obciążeń zawiera wagi dla innej sieci neuronowej.

3. Przeprowadzone doświadczenia

Została przeprowadzone seria eksperymentów, które umożliwiły dobór optymalnych parametrów pracy algorytmu TO, jak również porównanie tego algorytmu z innymi. W eksperymentach wykorzystane zostały następujące urządzenia: rozdzielacz sieciowy, switch, zestaw 4 jednakowych serwerów WWW, zestaw 4 różnych serwerów WWW.

Badania prowadzone w celu dobrania najlepszych parametrów pracy algorytmu TO dotyczyły: kształtu funkcji przynależności wejść, liczby zbiorów rozmytych dla wejść, współczynnika uczenia η, miar obciążenia serwera WWW.

Aby możliwe było określenie skuteczności pracy algorytmu TO, przeprowadzono eksperymenty porównawcze algorytmów TO, LARD, WRR. Zostało przeprowadzonych 26 eksperymentów dla dwóch różnych zestawów serwerów i witryn WWW zawierających różny procentowy udział plików dużych, małych, średnich, skryptów ASP, odwołań do bazy danych. Rysunek 7 przedstawia wybrane wyniki badań.



Rys. 7. Wykresy obrazujące liczbę pobrań na sekundę
Fig. 7. Number of requests per second

Oba wykresy dotyczą średniej liczby obsłużonych przez serwis żądań w funkcji liczby klientów pobierających. Jak wynika z przeprowadzonych eksperymentów, algorytm TO ma w większości przypadków największą liczbę obsłużonych żądań w jednostce czasu.

4. Podsumowanie

Algorytm TO dla większości przeprowadzonych eksperymentów porównawczych ma najkrótszy czas obsługi żądań klienta przy większej liczbie klientów (powyżej 8). Badania pokazują, że algorytm najlepiej pracuje dla żądań krótkich, których czas realizacji nie

przekracza 0.7 s. Algorytm TO lepiej od pozostałych algorytmów pracuje dla zestawu różnych komputerów.

Jak wynika z przeprowadzonych badań algorytm TO najlepiej realizuje postawione założenia o minimalizacji czasu obsługi żądań użytkownika.

LITERATURA

1. Casalicchio E., Colajanni M.: Scalable Web Cluster with Static and Dynamic Contents. Tech. Rep. DISP-2000-05, University of Roma Tor Vegata, Feb. 2000.
2. Colajanni M., Yu P.S., Cardellini V.: Scalable Web-Server Systems:Architectures, Models and Load Balancing Algorithms. Sigmetrics 2000.
3. Korbicz J.,Obuchowicz A.,Uciński D.: Sztuczne sieci neuronowe, podstawy i zastosowania. Akademicka Oficyna Wydawnicza PLJ, Warszawa 1994.
4. Pai V.S., Aron M.: Localy-aware request distribution in cluster-based network servers. Proc. 8th Int'l Conf. ASPLOS, San Jose, CA, Oct. 1998.
5. Piegat A.: Modelowanie i sterowanie rozmyte. Akademicka Oficyna wydawnicza EXIT, Warszawa 1999.

Recenzent: Dr inż. Jarosław Francik

Wpłynęło do Redakcji 25 marca 2002 r.

Abstract

This paper describes new adaptive load balancing algorithm for locally distributed web systems. At first we discuss methods, which can be used in order to balance load in cluster of web servers.

Next we present a quite new algorithm, which in an effective way balances the load of cluster of servers. This algorithm takes in account features of the content and locality in server cache. It takes advantage of quasi logic and neural networks in order to predict time of realization users request. New algorithm can adapt to conditions of work.

Finally we show results of curried out experiments and discuss results of experiments.