STUDIA INFORMATICA 2001
Volume 22 Number 1 (43)

Krzysztof ZATWARNICKI
Politechnika Opolska, Katedra Automatyki, Elektroniki i Informatyki

ALGORYTMY RÓWNOWAŻENIA OBCIĄŻEŃ LOKALNIE ROZMIESZCZONYCH SERWERÓW WWW

Streszczenie. W artykule zostały opisane algorytmy równoważenia obciążeń lokalnie rozmieszczonych serwerów WWW. Do algorytmów tych należą: *Session identyfiers*, *Content partiton*, *MultiClass Round Robin*, LARD. W drugiej części zostały przedstawione wyniki przeprowadzonych testów. Na końcu opisano zarys nowego proponowanego algorytmu równoważenia obciążeń.

LOAD BALANCING ALGORITHMS IN LOCALY DISTRIBUTED WEB SYSTEMS

Summary. This paper describes content aware algorithms used in layer-7 web switches. We discuss the following algorithms: Session identifiers, Content partition, MultiClass Round Robin and LARD. In the second part of the paper we describe results of the realised tests. Finally we present a quite new algorithm.

1. Wstęp

Obecnie globalna sieć komputerowa - Internet rozwija się bardzo dynamicznie. Ten dynamiczny rozwój stwarza jednak szereg problemów związanych z niezawodnym i szybkim dostarczaniem poszczególnych usług przez serwery *web*. Im więcej użytkowników korzysta z danej witryny, tym bardziej wydłuża się czas i zmniejsza jakość dostarczanych przez tę witrynę usług. W skrajnym przypadku może dojść nawet do sytuacji, w której witryna zostanie tak przeciążona żądaniami wywołania usług, iż przestanie całkowicie funkcjonować. Stan, w którym witryna traci swoją wydajność lub całkowicie przestaje działać, jest niedopuszczalny w społeczeństwie internetowym, w którym szybkość i niezawodność dostarczanych usług jest

kluczowym warunkiem popularności, a co za tym idzie - rentowności danej witryny. Dlatego powstało i nadal powstaje coraz więcej rozwiązań mających zwiększać jakość i szybkość usług.

Jednym z rozwiązań jest modernizacja serwera *web* i zwiększanie przepustowości sieci poprzez dodawanie nowych coraz wydajniejszych urządzeń sieciowych.

Administrator witryny może rozbudowywać swój serwer o nowsze i coraz bardziej wydajne podzespoły. Może np. zwiększyć szybkość i liczbę procesorów, zwiększyć wielkość i szybkość dostępnej pamięci RAM, zastosować szybsze i wydajniejsze dyski twarde, może również wymienić interfejsy sieciowe na szybsze i bardziej niezawodne. Jednakże takie rozwiązanie jest kosztowne i mało elastyczne, wiąże się z ciągłą wymianą podzespołów lub całej maszyny serwera.

Inną możliwością byłoby zastosowanie rozwiązań równoważących obciążenia sieci serwerów *web*. W przypadku takiego rozwiązania nie mamy już do czynienia z jednym potężnym serwerem, lecz z grupą serwerów pracujących jako jedna witryna. Takie podejście do problemu daje dużą elastyczność jak również możliwość dowolnego zwiększania wydajności poprzez dodawanie kolejnych serwerów.

Obecnie są stosowane dwie grupy rozwiązań umożliwiające równoważenie obciążeń serwerów *web*. Pierwsza z nich polega na łączeniu się klienta z odpowiednim serwerem, tzw. *web switch*, mającym za zadanie przełączenie połączenia w sposób niewidoczny dla klienta na odpowiedni serwer *web*. Przy czym *web switch* nie posiada informacji, jakie dane klient chce pobrać z serwera *web*. Takie podejście nazywane jest przełączaniem połączeń dla 4 warstwy sieciowej (w siedmio warstwowym modelu ISO-OSI) ang. *Level 4 Web switching*. Jest kilka algorytmów, jakie *web switch* może zastosować do doboru serwera WWW. Jedne z nich przydzielają serwer w sposób statyczny, np. *Round Robin* (przy wykorzystaniu tego algorytmu, ze statycznie utworzonej listy wybierane są kolejne serwery, do których trasowane będą pakiety), inne w sposób dynamiczny np. *Least Connections* (algorytm ten stara się określić obciążenia serwerów na podstawie liczby bieżących połączeń na danym serwerze, nowe połączenia wysłane będą do serwera o najmniejszej ich liczbie).

Druga grupa rozwiązań nazywana jest przełączaniem połączeń dla 7 warstwy sieciowej (w siedmio warstwowym modelu ISO-OSI) ang. *Level 7 web switching*. W rozwiązaniu tym klient nawiązuje połączenie z *web switch*, ten pobiera od klienta informacje, co ów klient chciałby pobrać z serwera *web*, następnie łączy klienta w sposób dla niego niewidoczny z odpowiednim serwerem.

W artykule tym zajmiemy się algorytmami wyboru odpowiedniego serwera WWW, czyli przełączaniem połączeń dla 7 warstwy w odniesieniu do protokołu HTTP dla lokalnie rozmieszczonych serwerów. Na początku zostaną omówione korzyści płynące ze stosowania rozwiązań opartych na przełączaniu połączeń 7 warstwy, następnie zostanie pokazana w skrócie

architektura technicznych rozwiązań dla tego typu podejścia, potem będą omówione algorytmy wyboru odpowiedniego serwera WWW. Na końcu zostaną przedstawione wyniki badań własnych oraz zarys proponowanego algorytmu.

2. Korzyści płynące ze stosowania rozwiązań opartych na przełączaniu połączeń warstwy siódmej

W obecnych czasach trudno wyobrazić sobie życie i pracę pewnych grup społeczeństwa bez możliwości korzystania z sieci WWW. Aby praca ta była możliwa, efektywna i przyjemna, serwisy WWW powinny posiadać pewne cechy:

- wysoką wydajność,
- bezpieczeństwo,
- łatwą osiągalność dla klienta,
- skalowalność.

By wymienione cechy były osiągalne, konieczne jest zastosowanie rozwiązań opartych na przełączaniu połączeń warstwy 7, ponieważ mają one następujące zalety [1]:

1. Elastyczne lokowanie zawartości serwerów WWW. Poprzez zapoznanie się z zapytaniem klienta *web switch* może wysłać zapytanie na odpowiedni serwer WWW, dzięki czemu nie jest konieczne wykonywanie całej kopii zawartości serwera WWW na wszystkich serwerach w serwisie.
2. Wspomaganie stałego połączenia. Jak wiadomo, protokół HTTP jest bezpołączeniowy, są jednak takie zadania, które wymagają stałego połączenia, np. transakcja płacenia poprzez sieć, złożone wyszukiwanie w sieci, wykonywanie zakupów itp. Przełączanie połączeń warstwy 7 umożliwia stałe połączenia.
3. Podniesiona wydajność grupy serwerów. Poprzez zapoznanie się z zapytaniem klienta *web switch* może wysłać określone zapytania na określone serwery. Dzięki temu w pamięci operacyjnej *cache* wybranych serwerów będą znajdować się wybrane pliki. Przy następnych odwołaniach o dane pliki do serwera zapytania kierowane są do tych samych serwerów co poprzednio, dzięki czemu odpowiedzi będą dużo szybsze.
4. Poprawione działanie grupy serwerów HTTP. HTTP wersja 1.1 pozwala na kilkukrotne pobieranie różnych zasobów w czasie jednego połączenia. Dzięki zastosowaniu przełączania połączeń warstwy 7 możliwe jest rozbicie jednej długiej transakcji pobierania kilku zasobów z jednego serwera na kilka krótkich zapytań, które można wysłać na różne serwery. Umożliwia to lepsze zbalansowanie obciążeń.

5. Identyfikacja użytkownika; dzięki możliwości poznania nagłówka HTTP *web switch* może identyfikować jednoznacznie użytkowników, a co za tym idzie - podzielić ich na klasy, np. na tych, którzy będą szybciej obsługiwani, bo częściej robią zakupy w danym serwisie.

Zastosowanie algorytmów przełączania połączeń warstwy 4 podnosi wydajność serwisów internetowych, jednakże serwisy takie nie posiadają cech wymienionych powyżej, które są niezbędne w obecnych czasach do poprawnego funkcjonowania.

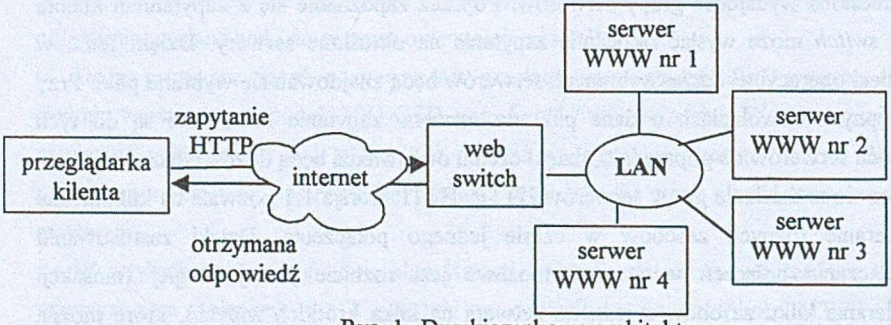
3. Architektury połączeń grupy serwerów dla zastosowania przełączania połączeń warstwy siódmej

Istnieje wiele rozwiązań technicznych umożliwiających grupowanie serwerów WWW w celu obsługi jednego lub kilku witryn WWW (grupa taka nazywana jest web cluster). Podziału rozwiązań można dokonać według następujących kryteriów [6]:

- mechanizmy użyte przez *web switch* do przekierowania przychodzących wiadomości do serwera WWW,
- drogę powrotną pakietów z serwera WWW do *web switch* lub klienta.

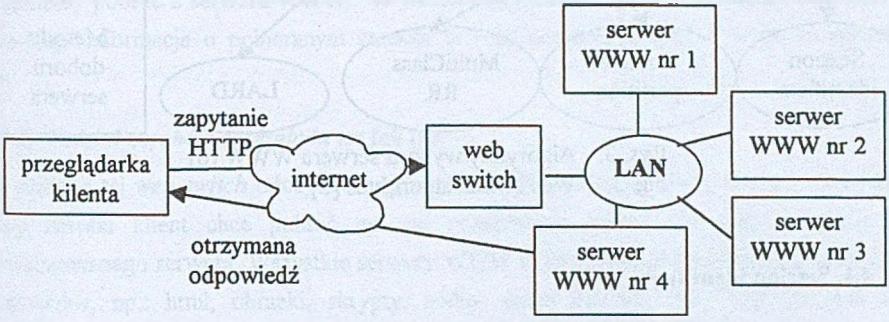
Biorąc pod uwagę opisane kryteria otrzymujemy następujące grupy rozwiązań:

1) dwukierunkowa architektura
   - TCP *gateway*
   - TCP *slicing*
2) jednokierunkowa architektura
   - TCP *handoff*
   - TCP *conection hop*.



Rys. 1. Dwukierunkowa architektura
Fig. 1. Two-ways architecture

Na rysunku 1 przedstawiony został schemat architektury dwukierunkowej. W rozwiązaniu tym klient nawiązuje połączenie z *web switch*, następnie przekazuje mu nagłówek zapytania HTTP. *Web switch* wybiera odpowiedni serwer WWW, nawiązuje z nim połączenie i pobiera z niego odpowiednie dane, które są przekazywane klientowi. W zależności od sposobu przekazania danych klientowi wyróżniamy dwie architektury: TCP *gateway*, gdzie odpowiedź z serwera WWW jest przesyłana do *web switch*, przechodzi przez cały stos protokołowy do warstwy aplikacji i przechodząc znowu przez cały stos idzie do klienta; TCP *splicing*, gdzie odpowiedź z serwera WWW jest dostarczana do *web switch*, przechodzi przez stos protokołowy do warstwy połączeń sieciowych, a stamtąd idzie z powrotem do klienta.

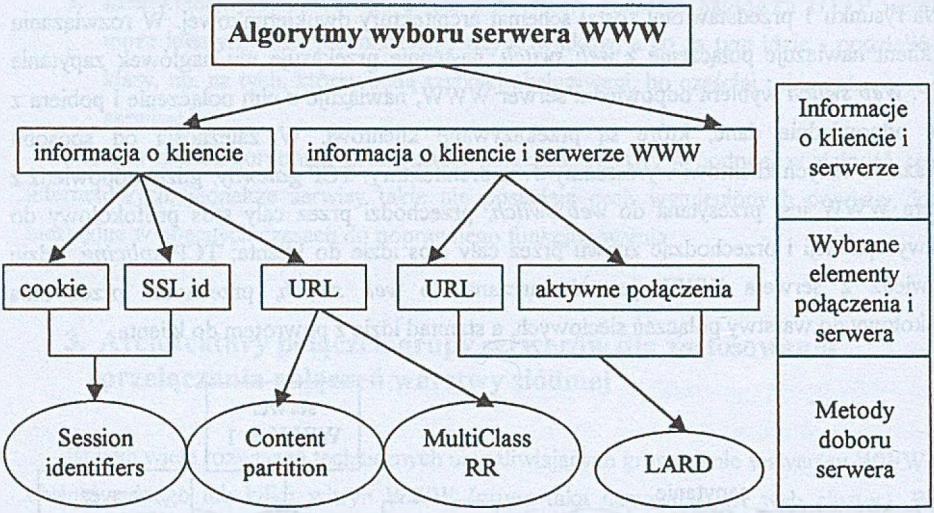


Rys. 2. Jednokierunkowa architektura
Fig. 2. One-way architecture

Na rysunku 2 została przedstawiona architektura jednokierunkowa, gdzie klient łączy się z *web switch*, wysyła mu nagłówek HTTP, *web switch* wybiera odpowiedni serwer WWW i przesyła mu nagłówek HTTP. Serwer WWW wysyła odpowiedź bezpośrednio do klienta z ominięciem *web switch*.

4. Algorytmy wyboru odpowiedniego serwera WWW

Algorytmy wyboru odpowiedniego serwera WWW można podzielić na takie, które decyzje o przydzieleniu do serwera podejmują na podstawie informacji o kliencie (przesłanej w nagłówku HTTP) lub też połączonych informacji o kliencie i stanie lub zawartości serwerów WWW [6]. Na rysunku 3 przedstawiona jest klasyfikacja algorytmów doboru serwera.



Rys. 3. Algorytmy wyboru serwera WWW [6]
Fig. 3. Web switch algorithms [6]

4.1. Session identifiers

Poprzez identyfikację sesji użytkownika *web switch* może połączyć użytkownika z odpowiednim serwerem WWW. Od przyjętej strategii zależy, jaki ma to być serwer. Najczęściej mówi się o dwóch strategiach: *Cookie-Based Sheduling* i *Persistance*.

4.1.1. Cookie-Based Sheduling

Poprzez stosowanie informacji, która może być dodana do nagłówka HTTP w *cookie*, możliwa jest jednoznaczna identyfikacja użytkowników, a co za tym idzie podzielenie ich na klasy [1][3]. Taki podział na klasy może mieć na celu rozróżnienie na przykład użytkowników bardziej uprzywilejowanych od tych mniej uprzywilejowanych. Użytkownicy uprzywilejowani mogą być przydzielani do lepszych serwerów lub grup serwerów, gdzie ich obsługa będzie szybsza i bardziej niezawodna lub też będą mieli możliwość wykorzystania zasobów, z których pozostali klienci nie będą mogli korzystać.

4.1.2. Persistance

Jak wiadomo, HTTP jest protokołem bezpołączeniowym. Poprzez stosowanie *cookie* lub niezaszyfrowanego identyfikatora SSL (tylko w wersji 3.0) możliwa jest jednoznaczna identyfikacja użytkownika [3]. Identyfikacja ta może być użyteczna, gdy klient wykonuje bardziej złożone operacje w serwisie WWW i konieczne jest, by łączył się zawsze z tym samym

serwerem, ponieważ ten przechowuje pewne informacje na temat użytkownika. Dobrymi przykładami wykorzystania stałych połączeń są:

- wykonywanie zakupów w serwisie internetowym, gdzie serwer powinien pamiętać, co dotychczas kupił klient,
- wykorzystywanie aplikacji *e-business*, gdzie wiele operacji jest transakcyjnych i wymaga wielokrotnego łączenia się tego samego klienta.

4.2. Content partition

Dzięki informacji zawartej w ramce HTTP klienta *web switch* dowiaduje się, jakie zasoby klient chciałby pobrać z serwera WWW. W trzech algorytmach opisanych poniżej *web switch* wykorzystuje informacje o pobieranym zasobie w celu podjęcia decyzji o wyborze serwera WWW.

4.2.1. Podział zasobów ze względu na ich typ

W strategii tej *web switch* odczytuje nagłówek HTTP, jaki otrzymał od klienta, sprawdza, jaki typ zasobu klient chce pobrać np. po rozszerzeniu pliku i wysyła zapytanie do wyspecjalizowanego serwera. Wszystkie serwery WWW w tym algorytmie obsługują konkretne typy zasobów, np.: html, obrazki, skrypty, audio, video itp. [8]. Aby algorytm ten był efektywny, poszczególne serwery obsługujące konkretne typy zasobów powinny być w swej budowie przystosowane do najlepszej ich obsługi.

4.2.2. Podział zasobów ze względu na ich wielkość

W algorytmie tym zakłada się podział wszystkich zasobów (głównie plików html i obrazków) ze względu na ich wielkość. *Web switch* musi posiadać informacje o wielkościach poszczególnych zasobów i biorąc to pod uwagę powinien według ustalonych progów wielkości plików wysyłać zapytania klientów do serwerów [7]. Przyjęto, że progi mogą być wybierane dynamicznie.

4.2.3. Podział zasobów według funkcji mieszających

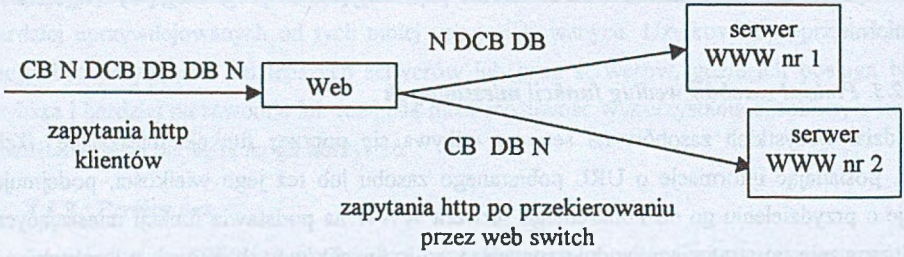
Podział wszystkich zasobów na serwery odbywa się poprzez funkcje mieszające. *Web switch*, posiadając informacje o URL pobieranego zasobu lub też jego wielkości, podejmuje decyzje o przydzieleniu go do konkretnego serwera WWW na podstawie funkcji mieszających [9]. Stosowanie tej strategii powoduje zmniejszenie liczby plików pobieranych z konkretnego serwera, a co za tym idzie, zwiększenie trafialności występowania danego zasobu w pamięci *cache* serwera.

4.3. Multi-Class Round Robin (MC-RR)

W tej strategii przyjmuje się, że zasoby oferowane przez serwery WWW można podzielić na następujące klasy [4]:

- publikacje WWW – są to strony, na których znajdują się zasoby statyczne, np. strony html wraz z obrazkami oraz strony dynamiczne, np. CGI, które w małym stopniu obciążają serwer,
- transakcje WWW – są to strony dynamiczne, najczęściej skrypty CGI obsługujące np. bazy danych, które w dużym stopniu obciążają dysk i procesor serwera,
- multimedia – strony obsługujące sewisy audio i video.

Pobieranie z serwera WWW zasobu z każdej z wymienionych klas powoduje różne obciążenie dysku i procesora serwera. Bazując na powyższej klasyfikacji można podzielić zapytania HTTP na cztery klasy: statyczne i lekko dynamiczne publikacje WWW (N), serwisy obciążające dysk (DB), serwisy obciążające procesor (CB) oraz serwisy obciążające dysk i procesor (DCB). W algorytmie MC-RR *web switch* przełącza cyklicznie zapytanie z każdej z klas do innego serwera WWW tak, aby żaden serwer nie był przeciążony jednym rodzajem zapytań, a co za tym idzie, nie miał przeciążonego żadnego ze swoich zasobów (procesora i dysku). W strategii tej każdy z serwerów WWW może obsłużyć każde z prawidłowych zapytań oraz wszystkie serwery otrzymują podobną liczbę zapytań należących do tej samej klasy. Opiszmy to następującym przykładem. Załóżmy, że serwer A ma w swojej kolejce trzy rodzaje zapytań typu N jedno typu DB i dwa typu CD, serwer B natomiast otrzymał poprzednio dwa zapytania typu N, i po jednym z zapytań typu DB, CB, DCB. Następnie do serwisu WWW przyszły kolejno następujące zapytania : CB, N, DCB, DB, DB, N. Przy użyciu MC-RR *web switch* przydzielił pierwsze zapytania serwerowi B, następne trzy serwerowi A i dwa ostatnie ponownie do B. Po takim przydziale serwery A i B posiadają podobną liczbę zapytań każdej klasy. Na rysunku 4 przedstawione zostało przekazywanie zapytań na poszczególne serwery.



Rys. 4. Multi-Class Round Robin
Fig. 4. Multi-Class Round Robin

4.4. LARD

4.4.1. Basic Locality-Aware Request Distribution

W algorytmie tym *web switch* podejmuje decyzje o przekierowaniu zapytania na podstawie informacji od klienta o potrzebnych zasobach oraz informacji o obciążeniu serwerów WWW. W tym przypadku jako obciążenie serwera brana jest pod uwagę liczba aktywnych połączeń na danym serwerze. Algorytm można opisać w sposób następujący [9]:

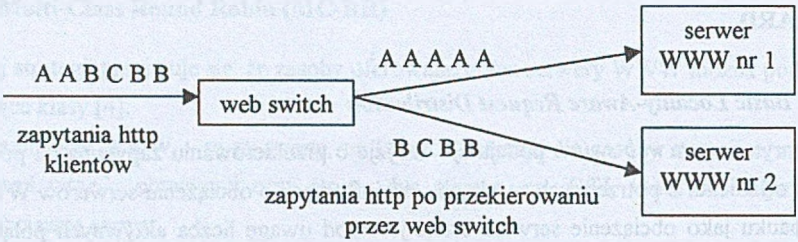
1. Jeśli zasób pobierany jest po raz pierwszy, przekieruj go na najmniej obciążony serwer.
2. Jeśli zasób pobierany jest po raz kolejny, przekieruj go na serwer, z którego był już pobierany, o ile serwer nie jest przeciążony, tzn. nie ma dużej różnicy między obciążeniem tego serwera a innymi.
3. Jeśli zasób pobierany jest po raz kolejny i serwer, z którego był już pobierany, jest przeciążony, przekieruj go na najmniej obciążony.

Pseudokod dla tego algorytmu został przedstawiony poniżej.

```
while (prawda)
  pobierz następne zapytanie r;
  if serwer[r.docelowy]=null then
     n:=najmniej obciążony serwer; serwer[r.docelowy]= najmniej obciążony serwer;
  else
     n:= serwer[r.docelowy];
     if (n.obciążenie > T_max  &&  ∃ serwer z obciążeniem < T_min) ||
           n.obciążenie > 2*T_max  then
              n:=najmniej obciążony serwer;
              serwer[r.docelowy]:= najmniej obciążony serwer;
  wyślij r do n;
```

Stosowanie tej strategii powoduje poprawne zbalansowanie dociążeń serwerów oraz zmniejszenie liczby plików pobieranych z konkretnego serwera, a co za tym idzie, zwiększenie trafialności występowania danego zasobu w pamięci *cache* serwera.

Poniżej na rysunku 5 przedstawiony został schemat podziału zadań na serwery WWW.



Rys. 5. Locality-Aware Request Distribution
Fig. 5. Locality-Aware Request Distribution

4.4.2. Locality -Aware Request Distribution with Replication

Algorytm ten podobny jest do poprzedniego z tą różnicą, że jeden zasób może być pobierany z kilku serwerów naraz, jeśli pojawią się żądania o niego od kilku klientów [9]. W rozwiązaniu tym kolejny serwer do obsługi danego zasobu włączany jest do grupy serwerów tego zasobu, jeśli poprzednie serwery zostały przeciążone w czasie, gdy obsługiwały zapytania dotyczące stałej liczby zasobów.

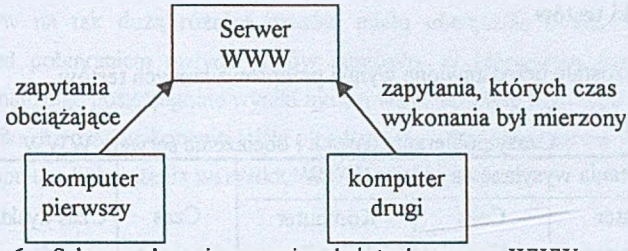
5. Test określający czas pobierania wybranych zasobów z serwera WWW przy zróżnicowanym obciążeniu serwera

W celu dokładniejszego poznania zachowania się serwera WWW przy różnorodnych obciążeniach został przeprowadzony zestaw testów. W testach określane są czasy pobierania konkretnych danych z serwera WWW przy różnych obciążeniach elementów komputera, na którym działał serwer.

W teście zostały wykorzystane trzy komputery:

- pierwszy z nich generował i wysyłał zapytania http do serwera WWW w celu obciążenia serwera (tak by obciążenie było stałe w czasie),
- drugi wysyłał zapytania do serwera WWW i mierzył czas wykonania przez serwer zadania,
- na trzecim zainstalowano serwer WWW,

Opisana sytuacja przedstawiona jest na rysunku 6.



Rys. 6. Schemat obrazujący pomiar obciążenia serwera WWW
Fig. 6. Measurement of web server's load

Jako serwer WWW posłużył *Internet Information Server*, działający pod kontrolą systemu operacyjnego *Windows 2000 Profesional* na komputerze Pentium III 450 MHz z dyskiem 13 GB Seagate, 128 MB pamięci operacyjnej, kartą sieciową 10 Mb/s. Za klienta pobierającego dane, których czas wykonania był mierzony, posłużył komputer z procesorem Pentium MMX 200 MHz, na którym działała aplikacja pobierająca kolejne zasoby jeden po drugim (nie równolegle). Jako klient obciążający serwer posłużył komputer z procesorem Pentium 100 MHz, na którym działała aplikacja wysyłająca do serwera kolejne zapytania obciążające go.

Z serwera WWW były pobierane następujące zasoby:

1. Duże pliki, których wielkość przekracza 12 MB.
2. Małe pliki, których wielkość nie przekracza 20 KB.
3. Dane udostępniane przez serwer po wykonaniu skryptów *Active Server Pages*. Skrypty te nie pobierały danych z baz danych czy też systemu plików serwera — działanie ich polegało na wykonywaniu funkcji matematycznych lub operacji na łańcuchach znaków.
4. Dane pochodzące z bazy danych Access'a udostępnionej poprzez interfejs ODBC oraz pliki typu IDC i HTX. Baza danych składała się z dwóch tabel powiązanych ze sobą relacją (tabele nie zawierają indeksów). Dane pobierane przez serwer WWW pochodzących z obu tabel. Każda z tabel zawierała 10 000 rekordów.

Z komputera z serwerem WWW były pobierane następujące dane dotyczące obciążenia wybranych zasobów:

1. Czas dysku (%) - jest wyrażoną w procentach ilością czasu, spędzoną przez wybrany dysk na obsłudze żądań odczytu i zapisu.
2. Czas procesora (%) - jest wyrażoną w procentach ilością czasu procesora, spędzoną przez wszystkie wątki i procesy na wykonywaniu kolejnych instrukcji.
3. Obciążenie karty sieciowej (%) - jest wskaźnikiem szybkości wyrażonym w procentach, z jaką bajty są wysyłane i odbierane przez kartę sieciową.

5.1. Wyniki testów

W tabeli 1 zostały przedstawione wyniki przeprowadzonych testów.

Tabela 1

Czasy pobierania danych i obciążenie serwera WWW

Zapytania wysyłane na serwer WWW			Czas procesora %	Czas dysku %	Obciążenie karty sieciowej %
Komputer mierzący czas	Czas pobierania (s)	Komputer obciążający serwer			
małe pliki	0,040	-	9,5	14	8,56
2*małe pliki	0,050	-	15,6	28	14,88
duże pliki	12,819	-	62,7	19,3	82,4
baza danych	0,100	-	21,2	1,2	2,96
asp	0,120	-	53	1	4,8
małe pliki	0,042	asp	55	15	12
małe pliki	0,043	baza danych	28	14,5	16
małe pliki	0,104	duże pliki	62	30	76,8
małe pliki	0,055	2*małe pliki	22	42	22,4
baza danych	0,111	asp	65	1,4	6,64
baza danych	0,229	duże pliki	74	23	83,2
baza danych	0,114	2*małe pliki	34	29	18,96
asp	0,165	asp	83	1,3	5,76
asp	0,142	baza danych	65	1,4	6,64
asp	0,308	duże pliki	88	20,7	81,6
asp	0,138	2*małe pliki	62	29	20,4
duże pliki	13,400	asp	88	20,7	81,6
duże pliki	13,000	baza danych	74	23	83,2
duże pliki	14,531	małe pliki	59	30	76,8

Jak wykazały testy, ważnym czynnikiem decydującym o wysyłaniu zadań na serwery WWW jest wiedza o obciążeniu kilku różnych elementów serwera (nie tylko jednego) oraz rodzaj zasobu, jaki ma być z serwera pobierany. Z tabeli 1 wynika, że gdy procesor serwera był początkowo obciążony w 53% udostępnianiem zasobów typu *asp*, to średni czas pobierania małych plików wynosił 55 ms. Natomiast czas pobierania małych plików, gdy procesor serwera był początkowo obciążony w 62% (poprzez udostępnianie dużych plików), wynosił aż 104 ms, czyli był dwa razy dłuższy przy niewielkiej początkowej różnicy obciążenia procesora serwera.

Największy wpływ na tak dużą różnicę czasów miało obciążenie dysku twardego, które początkowo przed pobieraniem małych plików wynosiło: w pierwszym przypadku 1%, w drugim 19,3%. Analizując poszczególne wyniki można wskazać wiele tego typu przypadków.

Z testów wynika, że czas wykonania ściśle określonych zadań przez serwer WWW zależy w bardzo dużym stopniu od obciążenia wszystkich elementów serwera.

6. Koncepcja nowego algorytmu równoważenia obciążeń

Jak wynika z przedstawionych testów, warto byłoby wprowadzić algorytm biorący pod uwagę obciążenie kilku elementów serwera oraz rodzaj i cechy zasobu, który ma być z serwera pobrany. W rozdziale tym przedstawiona zostanie wstępna koncepcja konstrukcji takiego algorytmu.

Jak wykazały testy przy pobieraniu plików z serwera, duże znaczenie ma obciążenie wszystkich trzech wymienionych wcześniej elementów komputera: obciążenie procesora, pamięci operacyjnej i karty sieciowej. Dla wielu rodzajów plików można ustalić do jakiej klasy plików należą (np. jaką mają wielkość) oraz dość dokładnie określić czas pobierania pliku z konkretnego serwera przy danym jego obciążeniu. Proponowane jest stworzenie tabeli obciążeń, w której zapisane będą czasy pobierania plików (należących do pewnych klas wielkości plików) dla poszczególnych obciążeń serwerów WWW, pracujących w grupie. W celu zmniejszenia wielkości tabeli zakłada się, że obciążenie konkretnego elementu serwera może przyjąć jedynie 4 wartości. Tabela będzie dotyczyła obciążeń procesorów, dysków i kart sieciowych poszczególnych serwerów. W każdym rzędzie tabeli będą czasy pobierania zasobów tej samej klasy dla różnych obciążeń i serwerów. Każda kolumna będzie oznaczała inne obciążenie serwera. Przy takich założeniach możliwe jest zapisanie informacji o czasie pobierania pliku z pojedynczego serwera, pliku należącego do konkretnej klasy wielkości plików, w obszarze pamięci 64 B. Liczbę taką otrzymujemy, gdyż istnieją 64 wartości, których każda cyfra może przyjąć wartości od 0 do 3 (bo takie są obciążenia serwera) oraz są trzy elementy komputera, których obciążenie będzie mierzone, co daje 4^3. Poniżej w tabeli 2 został przedstawiony przykład tabeli obciążeń. Obciążenia poszczególnych elementów serwera zostały przedstawione jako liczby trzycyfrowe, każda cyfra oznacza wielkość obciążenia wybranego elementu serwera.

Tabela 2

Tabela obciążeń

Rozmiar pliku	Serwer 1				Serwer 2			
	Obciążenia serwerów							
	000	100	...	333	000	100	...	333
	Czasy pobierania dla poszczególnych obciążeń							
2-4 kB	15	20	...	80	5	8	...	50
4-8 kB	20	30	...	120	8	10	...	60

Podział na klasy wielkości pobieranych plików może dotyczyć jedynie plików html oraz obrazków i statycznych zasobów. Jeśli zaś chodzi o zasoby dynamiczne, nie da się ich w prosty sposób zakwalifikować, ponieważ dla przykładu skrypty CGI mogą nie tylko obciążać procesor, ale również dysk i kartę sieciową (w przypadku gdy obsługują bazę danych). Dlatego proponuje się utworzenie osobnej tabeli obciążeń, w której znajdą się informacje dotyczące każdego obiektu dynamicznego osobno.

Trzecia tabela obciążeń przeznaczona będzie dla plików, do których wymagany jest szybki dostęp i które są najczęściej pobieranymi plikami. Tabela ta będzie zawierała podobne dane dotyczące wielkości i czasu obsługi plików, lecz będzie dotyczyła plików, do których częstość zapytań przekracza pewien próg. Zastosowanie osobnej tabeli dla często pobieranych plików jest konieczne, ponieważ pliki te znajdują się w pamięci *cache* serwerów, z których są pobierane i ich obsługa jest dużo szybsza od pozostałych plików.

W algorytmie przyjmuje się, że będzie istniała możliwość dynamicznej modyfikacji czasu pobierania zasobów w tabelach obciążeń, jeśli faktyczne czasy pobierania będą się znacznie różniły od opisanych w tabeli. Dzięki temu algorytm będzie adaptował się do zmiennych środowiska pracy.

Kolejnym problemem jest rozruch działania *web switch* z zaimplementowanym algorytmem, gdzie poszczególne tabele obciążeń nie są jeszcze wypełnione. Proponuje się tu przekierowywanie zapytań do serwerów o najmniejszej ilości aktywnych połączeń do momentu aż tabele obciążeń nie zostaną wypełnione do pewnego pułapu.

Można rozważać, ile połączeń na sekundę będzie w stanie ustanowić *web switch* z zaproponowanym algorytmem. Jest faktem, że algorytm ten wymaga przeszukiwania dużych tabel obciążenia oraz tabel zawierających informacje o wielkości plików. Jednak jak wykazały testy, komputer klasy Pentium III 450 MHz, zawierający tabele obciążeń z informacją o 10 000 plików na cztery serwery WWW, może w ciągu 1 ms pobrać informacje o 3 000 plików. Taki wynik wydaje się być zadowalający i dowodzi, że przedstawiony algorytm jest wydajny.

Użycie algorytmu opisanego powyżej wymaga stosowania architektury dwukierunkowej.

7. Podsumowanie

W pracy zostały przedstawione algorytmy przełączania połączeń warstwy 7 stosowane obecnie. W większości algorytmów dąży się do tego, aby klient był niezawodnie obsłużony w jak najkrótszym czasie. Zostały przedstawione następujące algorytmy: *Session identyfiers*, umożliwiający z jednej strony wykonywanie transakcyjnych operacji, z drugiej zaś podział użytkowników na grupy, *Content partiton*, pozwalający na zmniejszenie ilości replik w grupie serwerów, *MultiClass Round Robin*, mający na celu równomierne rozłożenie obciążeń poszczególnych elementów serwerów w grupie oraz LARD, wykorzystujący fakt, że część plików może znajdować się w pamięci *cache* serwerów.

W drugiej części pracy zostały przedstawione wyniki testów pokazujących, że czas wykonania ściśle określonych zadań przez serwer WWW zależy w bardzo dużym stopniu od obciążenia wszystkich elementów serwera.

Przedstawiono również wstępną koncepcję nowego algorytmu, który powinien jeszcze wydajniej wyrównywać obciążenia wszystkich serwerów w grupie. Algorytm ten bierze pod uwagę charakterystyczne cechy poszczególnych zasobów pobieranych z serwerów, jak również możliwość znalezienia się tych zasobów w pamięci *cache* serwerów. Aktualnie prowadzone są dalsze badania nad przedstawionym algorytmem.

LITERATURA

1. Alteon Web Systems, http://www.alteonweb.systems.com.
2. Aron M.,Druschel P., Zwaenepoel W.: Efficient support for P-HTTP in cluster baset Web serwers. Proc. USENIX 1999, Monterey, CA, June 1999.
3. ArrowPoint Communications, http://www.arrowpoint.com.
4. Casalicchio E., Colajanni M.: Scalable Web Cluster with Static and Dynamic Contents. Tech. Rep. DISP-2000-05, University of Roma Tor Vegata, Feb. 2000.
5. Cohen A., Rangarajan S., Slye H.: On the performance of TCP splicing for URL-aware redirection. Proc. 2nd USENIX Symp. On Internet Technologies and Systems. Oct. 1999.
6. Colajanni M., Yu P.S., Cardellini V.: Scalable Web-Server Systems:Architectures, Models and Load Balancing Algorithms. Sigmetrics 2000.
7. Harchol-Balter M., Crovelia M.E., Murta C.D.: On choosing a task assignment policy for a distributed server system. J. of Parallel and Distributed Computing , Vol. 59. Pp. 204-228, 1999.
8. IBM Network Dispatcher, http://www.ibm.com/software/network/dispatcher/

9. Pai V.S., Aron M., Banga G., Svendsen M., Druschel P., Zwaenepoel W., Nahum E.: Localy-aware request distribution in cluster-based network servers. Proc. 8th Int'l Conf. On Architectural Support for Programming Languages and Operating Systems, San Jose, CA, Oct. 1998.
10. Resonate Central Dispatcher, http:// www.resonate.com/products/central_dispatch.

Recenzent: Dr Jarosław Francik

Wpłynęło do Redakcji 11 kwietnia 2001 r.

Abstract

This paper describes content aware algorithms used in layer-7 web switches. Following algorithms are discussed:

1. Session identifiers – avoids multiple client identification for the same session,
2. Content partition – uses specialised servers for different contents,
3. MultiClass Round Robin – augment load sharing of component bound requests among Web servers,
4. LARD – improves locality (cache hit rate) in server cache.

In the second part of the paper results of the realised tests are described. In the conclusion of this test we can say that the time of getting some resources from Web server depends on the load of CPU, hard drive and a network card of the server.

Finally we present a quite new algorithm, which in an effective way balances the load of cluster of servers. This algorithm takes into account features of the content and locality in server cache.