STUDIA INFORMATICA 2014
Volume 35 Number 2 (116)

Andrzej BARCZAK, Dariusz ZACHARCZUK
Uniwersytet Przyrodniczo-Humanistyczny, Instytut Informatyki

TECHNIKI WPŁYWANIA NA WYDAJNOŚĆ APLIKACJI WEBOWYCH W WARSTWIE PREZENTACJI DANYCH

Streszczenie. Optymalizacja działania aplikacji to nie tylko efektywne zapytania SQL i odpowiednia konfiguracja sprzętu. Mimo stale rosnących przepustowości łącz internetowych nadal dużą rolę w tym procesie odgrywa warstwa prezentacji i sposobu przechowywania oraz dostarczania danych w tej warstwie. Artykuł omawia techniki wpływające na szybszą reakcję aplikacji WWW po stronie klienta, co minimalizuje obciążenie serwera.

Słowa kluczowe: apache, mysql, blob, wydajność, WWW, prezentacja danych, optymalizacja, warstwa prezentacji

TECHNIQUES TO INFLUENCE THE PERFORMANCE OF WEB APPLICATIONS IN THE PRESENTATION LAYER

Summary. Efficient SQL queries and corresponding hardware configuration are not only one methotd to optimize performance of web applications. Despite the ever-increasing bandwidth of Internet connections, still a large role in this process has presentation layer and how to store and deliver data in this layer. This article discusses techniques that affect faster response of web applications on the client side to minimize server load.

Keywords: apache, mysql, blob, efficiency, web, presentation of data, optimization, presentation layer

1. Wstęp

Wraz ze wzrostem technicznych możliwości komputerów, które stają się szybsze i bardziej pojemne, rosną także oczekiwania użytkowników wobec aplikacji, których używają.

Użytkownik wymaga natychmiastowych odpowiedzi na jego reakcje. Sprawa komplikuje się, gdy nie jest to w pełni uzależnione tylko od programu. Taka sytuacja ma miejsce w przypadku aplikacji WWW. Bez względu na to, czy jest to zwykła strona WWW, portal społecznościowy, czy gra przeglądarkowa, na szybkość reakcji składa się kilka głównych czynników:

- wydajny serwer,
- optymalny kod źródłowy,
- dobre połączenie internetowe.

Na prędkość połączenia internetowego żaden programista nie ma wpływu. Na wydajność serwera, co prawda, można wpływać pośrednio, np. wybierając z oferty hostingowej mocny, dedykowany VPS. Jednak nawet najmocniejszy serwer nie sprawdzi się, jeżeli kod źródłowy i sposoby przetwarzania w nim danych będą dalekie od optymalnych. Jakie zabiegi stosować, aby nie obciążać niepotrzebnie serwera i łącza, a jednocześnie sprawić, aby użytkownik był zadowolony z prędkości działania aplikacji? Temat jest bardzo rozległy, począwszy od zapytań SQL, poprzez kod źródłowy, przechowywanie, na prezentowaniu danych przez protokół HTTP skończywszy. W tym artykule zajmiemy się optymalizacją obszaru przechowywania i prezentowania danych.

2. Metody przechowywania danych i ich podział

Przede wszystkim należy ustalić, jakie konkretnie dane bierzemy pod uwagę. Warstwa prezentacji danych po stronie klienta to:

- pliki ściśle powiązane z layoutem, np. banery, ikony, obrazy teł, ale również pliki arkuszy stylów czy skryptów,
- pliki multimedialne audio/wideo – w tym konkretnym przypadku najlepszym i w zasadzie jedynym rozsądnym wyjściem jest skorzystanie z dedykowanych serwerów streamingowych jako oddzielnych zewnętrznych serwerów aplikacji lub skorzystanie z usług monopolisty w tej dziedzinie, czyli YouTube, ewentualnie Vimeo,
- typowe pliki do pobrania na dysk użytkownika, np. dokumenty tekstowe, arkusze, prezentacje itd.

Dane, o których będzie traktował ten artykuł, to dane z punktów pierwszego i ostatniego.

2.1. Baza danych kontra dysk twardy

Pierwszą grupą metod, która prawdopodobnie nasuwa się każdemu, jest podział ze względu na miejsce przechowywania danych:

1. przechowywanie plików fizycznie na dysku serwera,
2. przechowywanie ich jako obiektów w bazie danych.

Oba rozwiązania mają swoje zalety oraz wady. Stawiając za cel efektywność prezentowania danych, musimy zignorować inne kwestie, wynikające z korzystania z jednej lub drugiej metody, np.:

- Przechowywanie danych w bazie pozwala na tworzenie backupu w prosty sposób, natomiast przy plikach fizycznych operację tę należy wykonywać oddzielnie.
- Przy pierwszym rozwiązaniu transakcje muszą być obsługiwane oddzielnie; prawdopodobnie należy oprogramować własny system obsługi transakcyjności dla plików w systemie operacyjnym. Takich problemów nie ma przy drugiej opcji.
- Dla plików zewnętrznych konieczne są mechanizmy sprawdzania integralności danych i spójności informacji zawartych w bazie z danymi przechowywanymi na dysku. Problem ten nie występuje, jeżeli zaprzęgniemy bazę danych do przechowywania danych binarnych.
- Występuje brak kaskadowego kasowania danych dla fizycznych plików – w bazie danych taki mechanizm istnieje.
- Dostęp do plików na dysku dla osób administrujących jest łatwiejszy, bardziej naturalny i nie wymaga w zasadzie żadnego dodatkowego oprogramowania. W przypadku bazy danych niezbędne jest dodatkowe narzędzie.
- Przy plikach zewnętrznych może być także konieczny odczyt danych z bazy, aby pobrać informacje o położeniu pliku, a dopiero później następuje wskazanie na plik.

Pracochłonność czy też problemy przedstawione wyżej zostaną w tym artykule pominięte i programista sam powinien zdecydować, czy ewentualne bardziej kłopotliwe wdrożenie efektywniejszego rozwiązania zrekompensuje straty związane z większym nakładem pracy. Po omówieniu metod dostarczania danych do warstwy prezentacji zostanie porównana wydajność przechowywania danych dla obu przypadków. Wyniki planowanych testów pokażą, które podejście jest rekomendowane dla zastosowania w aplikacjach webowych w celu uzyskania szybszej reakcji na działania użytkownika.

2.2. Techniki dostarczania danych

Drugi podział możemy przeprowadzić ze względu na techniki dostarczania danych do warstwy prezentacji.

2.2.1. Złączenia danych o tym samym typie

Złączenia danych są jednym z najprostszych zabiegów poprawiania wydajności stron WWW. Działanie polega na zapisaniu w jednym pliku danych o tym samym typie. Najczęściej są to pliki CSS, JavaScript oraz dane graficzne. Zabieg wydaje się banalny, ale jego skuteczność jest bardzo duża. Wpływa on bardzo pozytywnie na RTT[1], radykalnie zmniejszając ich ilość. Dla grafiki ten proces nosi specjalną nazwę: *image sprites*[2]. Przykład takiego sprajtu przedstawia rysunek 1. Ten konkretny prosty przykład powoduje aż trzykrotną redukcję połączeń HTTP, gdyż zamiast trzech plików pobieramy tylko jeden.

Rys. 1. Przykład złączenia trzech obrazków 44x44 px w jeden sprajt 134x44 px
Fig. 1. Example of three 44x44 px images join into one 134x44 px image sprite

Sprajty są umieszczane jako tło obiektów. Stosując odpowiednie instrukcje CSS, można wyłuskać odpowiednią sekcję obrazka. Przykładowy kod, dzięki któremu w tle pojawi się strzałka w lewo, to:

```
/* 44px szerokość domku + 1px czarna linia = przesunięcie o 45px */
img.lewo { background:url(sprites.gif) -45px 0; width: 44px; height: 44px; }
```

Sprajty mogą zawierać nie tylko ikonki, lecz także powtarzalne tła – wszystko zależy od tego, jak się zaprojektuje ich użycie. Przyjęto jednak zasadę grupowania obrazów o podobnych cechach, np. o jednakowych wielkościach lub tylko o jednakowej składowej, np. szerokości. Wprowadza to pewien porządek, jednak do programisty należy decyzja, co jest ważniejsze w danym przypadku: porządek czy wydajność – nic nie stoi na przeszkodzie, aby sprajty były tworzone z innych sprajtów.

2.2.2. Równoległe pobieranie i asynchroniczne wstawianie

Równoległe pobieranie, czyli wskazanie dwóch lub więcej różnych hostów źródeł, to kolejny sposób na poprawienie szybkości działania. Specyfikacja HTTP1.1 (sekcja 8.1.4[3]) mówi, że przeglądarki powinny pozwalać na maksymalnie dwa jednoczesne połączenia do danego hosta (w praktyce jednak są to trochę większe wartości[4]). Na każde takie dwa połącze-

[1] Round Trip Time (lub Round Trip Delay Time) – jest to czas wymagany do przesłania sygnału w obu kierunkach: klient – serwer – klient, bez uwzględniania czasu przesyłania danych. Czas ten jest także popularnie nazywany pingiem. Na przykład dla przeglądarki inicjującej połączenie z serwerem WWW niezbędne są minimum trzy RTT-esy: dla DNS, TCP i dla żądania HTTP.

[2] Image Sprites (w wolnym tłumaczeniu: krasnale obrazkowe, popularnie nazywane po prostu sprajtami) – kolekcja obrazków umieszczona w jednym obrazie/pliku. Pojedyncze obrazki są wyłuskiwane dzięki właściwościom stylów CSS.

[3] http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html#sec8.1.4

[4] http://www.browserscope.org/ – serwis poświęcony testowaniu właściwości przeglądarek internetowych.

nia przypada jeden RTT. Różnicując źródła, powodujemy, że przeglądarka może pobrać ich więcej równolegle. Okupione jest to niestety większą liczbą RTT oraz większym użyciem CPU przez program klienta, dlatego nie zaleca się korzystania z więcej niż pięciu różnych hostów. Dodatkowo rozwiązanie to jest nieefektywne, jeżeli z jednego hosta pobieramy mniej niż 10 plików. Rozkład plików powinien być uśredniony, czyli przy 100 zasobach i czterech hostach każdy powinien mieć po 25 odwołań.

Kolejnym ograniczeniem mogą być pliki JavaScript – przeglądarki nie pobierają ich równolegle, więc rozkład źródeł między serwerami nic nie da, a wręcz może zaszkodzić. Większość przeglądarek blokuje pobieranie **wszystkich innych** plików podczas ściągania plików JS, uniemożliwiając równoległe pobieranie. Należy minimalizować liczbę odwołań do zewnętrznych skryptów JS oraz pamiętać o poprawnej kolejności – pliki skryptów ładujemy na końcu. Tabela 1 ilustruje opisywaną blokadę.

Tabela 1

Przykład blokady spowodowanej złą kolejnością załączania zasobów

Zła kolejność zasobów	Pliki JS blokują pobieranie			Poprawna kolejność	Pobieranie w ciągu dwóch jednostek czasu	
Styl_1.css	T1			Styl_1.css	T1	
skrypt_1.js	T1			Styl_2.css	T1	
Skrypt_2.js		T2		Styl_3.css	T1	
Styl_2.css			T3	skrypt_1.js	T1	
Styl_3.css			T3	Skrypt_2.js		T2

W przypadku skryptów z pomocą przychodzi nam jeszcze asynchroniczne ładowanie. Zasoby JavaScript, które nie są niezbędne przy inicjowaniu wyglądu strony, np. statystyki, śledzenie, ale także mniej istotne skrypty, powinny być ładowane asynchronicznie. Nowe kody śledzenia Google Analytics opierają się na tym mechanizmie. Zaletą takiego pobierania jest unikanie blokad przy ładowaniu i renderowaniu strony. Skrypty ładowane asynchronicznie wykonują się na samym końcu, np.:

```
<script>
var node = document.createElement('script');
node.type = 'text/javascript';
node.async = true;
node.src = 'skrypt.js';
//Teraz wstawiamy węzeł do struktury DOM za pomocą insertBefore()
var s = document.getElementsByTagName('script')[0];
s.parentNode.insertBefore(node, s);
</script>
```

2.2.3. Opóźnione wstawianie

W przypadku opóźnionego wstawiania mamy do czynienia raczej z wypracowanym pewnym rozwiązaniem niż z metodą. Można je porównać z programowym asynchronicznym ładowaniem. W sytuacji gdy prezentujemy dużą liczbę (10, 50, 300... więcej) indywidualnych

obiektów, np. obrazków, do których odwołania są bezpośrednio w kodzie strony – przeglądarka stara się je ładować tak szybko, jak to możliwe. Uniemożliwia to renderowanie, niekiedy poprawne, wyświetlenie, a także korzystanie ze strony do momentu załadowania danych. W takim wypadku jedynym rozsądnym rozwiązaniem jest załadowanie danych dopiero po wyrenderowaniu strony. Dzięki temu użytkownik może zacząć z niej korzystać od razu, a zasoby pobiorą się po chwili – zazwyczaj nie są one potrzebne wszystkie naraz i od razu przy starcie strony. Do tego celu można użyć popularnej technologii Ajax i pobrać dane po załadowaniu się strony lub po prostu zmodyfikować bezpośrednio model DOM. Przykłady pokazuje drugi, prostszy sposób, opierający się na automatycznym pobieraniu danych, kiedy zmienimy adres do obiektu, oraz na bibliotece jQuery[5].

```
<!-- kod HTML: kolekcja obiektów – obrazków -->
<img src="blank.gif" data-src="/img1.jpg" alt="" class="kolekcja" />
...
<img src="blank.gif" data-src="/imgN.jpg" alt="" class="kolekcja" />

//kod JavaScript: Aktywacja kolekcji obrazów po załadowaniu strony
$(window).load(function(){
  $('.kolekcja').each(function(){
    $(this).attr('src', $(this).attr('data-src'));
     });
});
```

Powyższe rozwiązanie spowoduje, że bez względu na liczbę obiektów w kolekcji, użytkownik będzie w stanie rozpocząć korzystanie ze strony natychmiast po jej pobraniu, a zawartość kolekcji będzie się pojawiać sukcesywnie w miarę pobierania danych.

2.2.4. Kompresja danych

Pomimo że funkcjonalność ta jest standardowo wspierana przez wszystkie nowoczesne przeglądarki, Google szacuje, że każdego dnia ponad 99 lat jest marnowanych z uwagi na brak stosowania kompresji danych. Kompresja odbywa się przy wykorzystaniu algorytmu gzip/deflate[6]. Jeżeli serwer jest skonfigurowany w sposób umożliwiający kompresję, to wszystkie tekstowe dane, tj. HTML, CSS, JS, XML, zostaną przed wysyłką skompresowane. W przypadku braku włączonej kompresji[7] programista sam może o nią zadbać, włączając ją manualnie:

```
#kod pliku .htaccess
<ifModule mod_gzip.c>
  mod_gzip_on Yes
```

[5] JQuery (jquery.com) – lekka biblioteka programistyczna dla języka JavaScript, ułatwiająca korzystanie z JavaScript (w tym manipulację drzewem DOM). Źródło: Wikipedia.

[6] Format kompresji danych zlib. Algorytm deflate oraz format plików gzip zostały ustandaryzowane w następujących dokumentach: RFC 1950, RFC 1951 i RFC 1952. Standardy te wykorzystuje m.in. format graficzny PNG.

[7] http://checkgzipcompression.com/ – jedno z wielu narzędzi online do sprawdzania, czy serwer wysyła dane w postaci GZIP.

```
    mod_gzip_dechunk Yes
    mod_gzip_item_include file \.(html?|txt|css|js|php|pl)$
    mod_gzip_item_include handler ^cgi-script$
    mod_gzip_item_include mime ^text/.*
    mod_gzip_item_include mime ^application/x-javascript.*
    mod_gzip_item_exclude mime ^image/.*
    mod_gzip_item_exclude rspheader ^Content-Encoding:.*gzip.*
  </ifModule>
```

Poza tym można także użyć instrukcji języka programowania do wymuszenia kompresji; przykład dla PHP:

```
ob_start('ob_gzhandler'); /* kod PHP */ ob_end_clean();
```

2.2.5. Optymalizacja i kompilacja kodu źródłowego

Optymalizacja kodu źródłowego polega na oczyszczeniu go ze zbędnych białych znaków, komentarzy czy tagów. One oczywiście pomagają programiście zapanować nad kodem, ale dla analizatora składniowego nic nie znaczą. Parser musi wykonać dodatkową pracę, czyli poświęcić dodatkowy czas na analizę tych zbędnych danych i ich pominięcie. Do usuwania zbędnych znaków z kodu HTML[8], CSS[9] i JS[10] istnieje dużo narzędzi dostępnych online za darmo. Często taka operacja jest też nazywana zaciemnianiem kodu i stosowana, aby utrudnić osobom trzecim skopiowanie rozwiązań używanych w aplikacji. To jednak nie wszystko, jeśli chodzi o optymalizowanie kodu źródłowego. HTML5[11] pozwala na pójście krok dalej – w tej wersji HTML można wręcz pomijać opcjonalne tagi, np.:

```
<li>punkt listy</li>    <!-- stara wersja -->
<li>punkt listy         <!-- wersja HTML5 -->
```

lub zastępować je bardziej optymalnymi, skróconymi:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!DOCTYPE html>         <!-- krótsza wersja HTML5 -->
```

co także redukuje czas potrzebny na parsowanie i zmniejsza objętość dokumentu. HTML5 wprowadził też atrybut *async* znacznika *script*, wskazujący przeglądarce pliki, które mają być ładowane asynchronicznie.

W większość aplikacji kod HTML jest jednak kodem wynikowym, generowanym np. przez jeden z najpopularniejszych języków programowania w Internecie[12]: PHP. Od ok. 2002 roku dla źródeł w języku PHP jest dostępna jeszcze jedna, dużo bardziej wydajniejsza opcja optymalizacji kodu niż w przypadku czystego HTML – kompilacja. Kompilacja kodu do byte-kodu przynosi zyski, zwłaszcza przy aplikacjach potrzebujących dużo zasobów

[8] HTML optimizer: http://www.webtoolhub.com/tn561361-html-optimizer.aspx
[9] Online JavaScript/CSS Compression: http://refresh-sf.com/yui/
[10] Closure compiler: http://closure-compiler.appspot.com/home
[11] http://pl.wikipedia.org/wiki/HTML_5, http://www.whatwg.org/specs/web-apps/current-work/multipage/
[12] http://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites

procesora. Do tego celu można użyć IonCube[13] lub ZendGuard[14]. Oprogramowanie kompiluje skrypty do postaci binarnej i później dodatkowo koduje binaria. Pliki wynikowe nie zawierają w sobie żadnej pierwotnej wersji kodu źródłowego. Proces ten chroni także kod przed złośliwym oprogramowaniem, ale to już kwestia drugorzędna. Przykładowe linie kodu PHP generowanego przez ionCube to:

```
HR+cP//l/hmhgaC53ATFAGY3IZSLSnpFXT74o8+iuh0UxAnjB9gpQvxpkoHvV2jX1CCVPicB+Iby
o9HSVrT96XOE1T4/6bAe3XoBZ48bmCmZThijwLt6yAzLp6azV8lzvcqTB0mMh0lPfYZPuKlECpw4
qmnDAhppYHJjgRoxm4sbXAUYuv0zDERTj8xquKibJzP87pWSIbBayzRmHWuHDji7Stdr8/vBJuOU
...
```

Hosting oczywiście musi mieć zainstalowane odpowiednie biblioteki do obsługi tego typu plików.

2.2.6. Renderowanie

W kwestii renderowania główną uwagę powinno się skupić na kodzie CSS, który jest odpowiedzialny za wygląd aplikacji. Kluczem do poprawnego i szybkiego działania jest:

- kasowanie nieużywanych stylów, umieszczanie bloków CSS i tagów *link* w nagłówku dokumentu HTML, dzięki czemu renderowanie może się rozpocząć wcześniej i trwa krócej,
- używanie efektywnych selektorów CSS: preferowane są nazwy klas i identyfikatory zamiast tagów,
- unikanie wyrażeń/formuł CSS głównie z uwagi na użytkowników IE, który dopiero w wersji 8 i tylko w trybie zgodności w pełni obsługuje standard CSS,
- unikanie instrukcji @import,
- określanie rozmiarów obrazów: eliminuje niepotrzebne przeliczenia i odświeżanie widoku strony,
- poprawne specyfikowanie kodowania i zawartości dokumentu, co umożliwi natychmiastowe parsowanie i uruchomienie skryptów.

  Są to drobiazgi, ale diabeł tkwi w szczegółach.

2.2.7. Buforowanie

Jest to ostatnia opisywana metoda, ale z całą odpowiedzialnością można powiedzieć, że jest najważniejsza i przynosi największe oszczędności i najlepsze rezultaty. Buforowanie nie jest niczym niezwykłym czy nowatorskim. Każdy dobry programista jest świadomy tej techniki wpływania na efektywność. Warto jednak poświęcić temu zagadnieniu kilka zdań mimo wszystko.

[13] IonCube (http://www.ioncube.com/).
[14] ZensGuard (http://www.zend.com/en/products/guard/).

Każda aplikacja składa się z plików, które muszą zostać pobrane z Internetu. Pobieranie tych plików to nic innego jak czas, z którym walczymy. Dzięki buforowaniu wygrywamy potrójnie: redukujemy RTT przez eliminację żądań HTTP, zmniejszamy ilość danych do pobrania, a także oszczędzamy przepustowość serwera. Możemy buforować, wykorzystując pamięć podręczną przeglądarki, jak również serwery proxy. W pierwszym przypadku należy skorzystać z odpowiednich instrukcji w nagłówkach[15] HTTP, tj.: Expires i Cache-Control[16] albo Last-Modified i ETag[17]. Buforowanie oparte na dwóch ostatnich instrukcjach jest słabym buforowaniem, gdyż wymusza pobranie nagłówka HTTP dla zasobu, aby uzyskać niezbędne informacje o pliku. Zdecydowanie lepiej jest używać daty ważności (*expires*) oraz znacznika czasu (ang. *fingerprint*). Aktualne przeglądarki mają wbudowane heurystyczne metody decydowania o tym, czy użyć pliku z buforu, czy nie, a decyzje są podejmowane głównie na podstawie adresów url. Jeśli adres się powtórzy, najprawdopodobniej zostanie użyty bufor, nawet jeśli serwer nie wysłał informacji determinującej takie działanie. Mimo to powinno się włączyć buforowanie dla wszystkich plików statycznych nawet na rok do przodu, a przy pomocy znacznika czasu – dynamicznie kontrolować buforowanie.

Wykorzystanie buforowania opartego na proxy pozwala natomiast na pobranie plików z bliższego serwera zamiast z oryginalnego, znajdującego się dalej. Oznacza to korzyści już przy pierwszym wejściu na stronę. Kiedy statyczny zasób zostanie pobrany przez jednego użytkownika przez proxy, inny użytkownicy korzystający z tego samego proxy zostaną już obsłużeni przez bufor tegoż serwera proxy. (Ważna informacja: przy korzystaniu z bufora proxy nie należy korzystać z adresów zawierających „?”. Większość proxy nie buforuje wtedy plików). Zastosowanie obu metod wygląda następująco:

```
#plik .htaccess
#włączamy moduł buforowania
<IfModule mod_expires.c>
ExpiresActive On
# ustawiamy czas wygaśnięcia domyślnie na 1 miesiąc
ExpiresDefault                          "access plus 1 month"

#a czas dla plików statycznych na 1 rok
ExpiresByType image/gif                 "access plus 1 years"
ExpiresByType text/css                "access plus 1 years"
ExpiresByType text/javascript         "access plus 1 years"
ExpiresByType application/javascript    "access plus 1 years"

#Favicon
ExpiresByType image/x-icon              "access plus 1 years"

#dyrektywa 'public' włącza buforowanie przy użyciu proxy
```

[15] us3.php.net/manual/en/function.header.php, httpd.apache.org/docs/2.0/mod/mod_expires.html

[16] Expires and Cache-Control: instrukcja określa, przez jaki czas przeglądarka ma wykorzystywać dany zasób bez sprawdzania, czy nowsza wersja jest dostępna. Jest to po prostu data ważności pliku.

[17] Last-Modified and ETag: wartości podane w tych nagłówkach mówią przeglądarce, czy plik jest taki sam jak przechowywany w buforze. Wartość dla Last-Modified jest zawsze datą, natomiast ETag może być dowolną wartością determinującą zmiany w pliku, np. wersja pliku, hasz.

```
<IfModule mod_headers.c>
    Header append Cache-Control "public"
</IfModule>
</IfModule>
```

Znacznik czasu wymuszający pobranie nowej wersji pliku można ustawić np. tak:

```
/style.css?t=1387850446
/42b6bc450914a01097b96b4bcf64babd_style.css
```

2.3. Uwagi optymalizacyjne dla urządzeń mobilnych

Urządzenia mobilne nadal mają dużo mniejsze możliwości przetwarzania danych niż komputery stacjonarne. Z tego względu optymalizacja dla tych urządzeń ma dużo większe znaczenie. Wszystkie opisane techniki dostarczania danych znajdują zastosowanie także i w tym przypadku, możemy jednak wykorzystać jeszcze kilka operacji poprawiających wydajność, szczególnie na stronach mobilnych.

Przede wszystkim nic nie zastąpi dedykowanych stron mobilnych. Nawet cieszące się ostatnio dużą popularnością strony responsywne[18], z uwagi na nadmiarowość kodu związaną z funkcjonowaniem tego typu rozwiązania, nie dorównają wydajnością typowym stronom mobilnym.

Druga kwestia jest taka, że z testów przeprowadzonych przez developerów z Google w 2011 roku wynika, że każdy dodatkowy KB kodu JavaScript odpowiada ok. +1 ms czasu parsera, czyli 100 KB kodu JS to dodatkowe 100 ms generowania strony. Ten dodatkowy czas występuje zawsze, gdyż JS musi być parsowana przy każdej wizycie na stronie bez względu na to, czy dane są ładowane z sieci, z bufora, czy w trybie offline HTML5. Jeśli dodamy do tego fakt, że użytkownicy muszą czekać np. na możliwość powiększenia widoku do momentu zakończenia ładowania strony, to każda część sekundy jest na wagę złota. Ważne jest więc, aby minimalizować liczbę plików JS potrzebnych do renderowania strony i korzystać z opóźnionego ich parsowania, jeśli nie są niezbędne od razu.

W niektórych przypadkach (np. praca offline) konieczne jest pobranie od razu całego kodu JS w celu poprawnego działania aplikacji. Można tego dokonać w dość sprytny sposób, niepowodujący niepotrzebnych opóźnień, umieszczając kod JS jako komentarz i wykorzystując funkcję eval()[19] do jego parsowania, w momencie kiedy będzie potrzebny. Przykład takiego zastosowania możemy obserwować w GMailu[20].

[18] Strona responsywna (ang. *responsive*) – strona internetowa dostosowująca się do rozdzielczości urządzenia, na jakim jest oglądana (dosłownie: wrażliwa na jego rozdzielczość). Źródło: http://pl.wikipedia.org/wiki/Strona_responsywna

[19] Eval() – funkcja JavaScript, która w zależności od argumentu oblicza wyrażenie lub wykonuje instrukcje.

[20] http://googlecode.blogspot.com/2009/09/gmail-for-mobile-html5-series-reducing.html

Chcąc maksymalizować zyski wynikające z zabiegów optymalizacyjnych i mając na uwadze fakt, że jeden duży obiekt pobiera się szybciej niż kilka małych, można rozważyć także użycie schematu danych URI[21]. Dzięki temu, zamiast generować dodatkowe wywołania HTTP:

```
<img src=”/1px.gif” alt=”” />
```

możemy umieścić obrazek () bezpośrednio w kodzie HTML czy CSS:

```
<img src="
AAAAABAAEAAAICVAEAOw==" alt=”” />
```

Dane przekazywane w ten sposób (ciąg base64[22]) mogą powodować utratę kompresji niektórych formatów obrazów (możliwa jedynie kompresja na podstawie gzip), ale czynnik redukujący RTT jest ważniejszy w przypadku urządzeń mobilnych. Oczywiście nic nie stoi na przeszkodzie, aby takie rozwiązanie stosować także na zwykłych stronach WWW.

3. Nowe standardy i protokoły

Twórcom oprogramowania przychodzi z pomocą, jak zawsze, przedsiębiorstwo Google. Google jest kluczowym członkiem grypy W3C Web Performance Working Group, której zadaniem jest dostarczanie metod do pomiaru aspektów wydajności aplikacji. Opracowali oni nowe rozwiązania, które mimo że na chwilę obecną są wspierane głównie przez Google Chrome, to warto się z nimi zapoznać. Inni producenci przeglądarek nie będą na pewno długo zwlekać z wprowadzeniem obsługi tych mechanizmów.

3.1. Protokół SPDY

SPDY[23] jest oparty na TCP, dzięki czemu nie wymaga zmian w obecnej organizacji w infrastrukturze sieciowej. Jego celem jest skrócenie czasu ładowania stron głównie dzięki kompresji, co ma obrazować także nazwa, która jest akronimem słowa SPeeDY. Od protokołu HTTP odróżnia go głównie:

- multipleksacja stumieni: SPDY pozwala na nielimitowaną ilość jedneczesnych stumieni podczas jednego połączenia TCP,

[21] Data URI scheme – metoda pozwalająca na załączanie danych binarnych bezpośrednio w kodzie stron WWW lub plików CSS.

[22] Base64 – rodzaj kodowania, używany do prezentacji danych binarnych w formie ciągu znaków.

[23] SPDY (czyt. spidi) – eksperymentalny protokół przyspieszający działanie sieci WWW. Oficjalny dokument informacyjny dostępny pod adresem https://sites.google.com/a/chromium.org/dev/spdy/spdy-whitepaper

- priorytetyzacja: możliwość określania priorytetów dla pobieranych zasobów, dzięki czemu przy ograniczonym paśmie internetowym serwer może zdecydować, które pliki przesłać jako pierwsze,
- kompresja wszystkich nagłówków HTTP.

SPDY zawiera także zaawansowaną funkcję inicjowania połączenia przez serwer, dzięki czemu dane mogą zostać przesłane do klienta bez proszenia o nie. Opcja ta jest konfigurowalna przez dewelopera na dwa sposoby:

- nacisk serwera (*server push*): serwer wysyła dane do klienta za pośrednictwem nagłówka X-Associated-content. Nagłówek ten informuje klienta, że serwer wysyła mu dane, zanim on ich zażąda;
- wskazówki serwera (*server hint*): przy użyciu hagłówka X-Subresources serwer informuje, o które dane klient powinien poprosić, jednak wstrzymuje się z ich wysłaniem do momentu faktycznej prośby o nie.

SPDY jest produktem open-source i instaluje się jako moduł Apache[24]. Jest już używany przez Google Chrome do komunikacji z usługami Google. W Firefox od wersji 11 jest możliwość włączenia obsługi SPDY (przełączając wartość ustawienia network.http.spdy.enabled na true w about:config), a od wersji 13 jest ona włączona domyślnie. Opera obsługuje SPDY od wersji 12.10. IE11 także obsługuje, ale tylko pod systemem Windows 8.1, który udostępnia ten protokół dla wszystkich przeglądarek.

Okazuje się, że większość przedstawionych w tym artykule technik optymalizacji jest zbędna lub wręcz negatywnie wpływa przy korzystaniu z protokołu SPDY. Równoległe pobieranie z użyciem kilku hostów jest złym rozwiązaniem, gdyż SPDY jest wydajniejszy, jeśli używamy minimalnej liczby połączeń przez protokół TCP. Używanie opcji *serwer push*, zamiast umieszczania danych w treści za pomocą kodowania base64, umożliwia buforowanie. Natomiast redukcja połączeń HTTP za pomocą sprajtów jest niepotrzebna przy SPDY z uwagi na multipleksowość strumieni. W takim razie, czy przedstawione rady należy zignorować? Jeszcze nie. Protokół nie jest jeszcze standardem, więc nadal wszystkie opisane zabiegi należy stosować w celu zwiększania wydajności aplikacji. Nawet narzędzia Google do audytu stron WWW nadal przeprowadzają analizę i prezentują wyniki bez uwzględniania protokołu SPDY. Przykładem może być demo oparte na mod_spdy pod adresem https://www.modspdy.com/world-flags/, gdzie PageSpeed[25] proponuje m.in. użycie sprajtów CSS w celu zwiększenia wydajności. Szanse na zdewaluowanie się tych metod w przyszłości są jednak dość duże. Z przeprowadzonych przez Google symulacji na 25 czołowych stronach

[24] Instrukcja instalacji SPDY dla Apache: https://developers.google.com/speed/spdy/mod_spdy/

[25] PageSpeed – darmowe narzędzie dostarczane bezpośrednio przez Google do audytu i analizy stron WWW pod względem ich wydajności, https://developers.google.com/speed/pagespeed/

WWW (symulacja została przeprowadzona po pobraniu i uruchomieniu stron w symulowanym środowisku przy użyciu prototypu klienta Google Chrome) wynika, że używając SPDY, można uzyskać od 27% do 60% szybszy czas ładowania strony dla zwykłego TCP oraz od 39% do 55% przy SSL.

3.2. Standard WebP

WebP[26] to nowy graficzny format danych dla statycznych i animowanych obrazów rastrowych, który umożliwia dwa rodzaje kompresji: stratną i bezstratną. Testy Google potwierdzają o 26% mniejszy rozmiar pliku przy kompresji bezstratnej w stosunku do odpowiednika w formacie PNG oraz zysk na poziomie 25-34% w odniesieniu do formatu JPEG (opierając się na indeksie podobieństwa SSIM[27]) dla kompresji stratnej. Co ciekawe – kompresja stratna obsługuje kanał alfa, co powoduje, że taki plik jest nawet trzy razy lżejszy niż odpowiednik PNG. Do tej pory użycie przeźroczystości determinowało korzystanie z formatu PNG (ewentualnie GIF, jeśli paleta kolorów zawierała ich nie więcej niż 256). Mniejsze pliki oczywiście przekładają się na szybsze pobieranie ich z serwerów, a w rezultacie użytkownik może wcześniej oglądać daną stronę.

Niestety, z uwagi na młody wiek WebP nie jest on jeszcze obsługiwany we wszystkich przeglądarkach. Na chwilę obecną standard ten wspierają oczywiście Google Chrome, Opera 11.10 i Android Ice Cream Sandwich. Dla IE Google zaleca instalację wtyczki Google Chrome Frame.

Proces tworzenia oraz przeglądania grafiki w tym standardzie jest także problematyczny. Desktopowe programy graficzne nie są przystosowane do pracy z tym typem danych i nie uda nam się zapisać obrazu jako WebP w tak popularnych dzisiaj IrfranView, Picassa czy Imagina. Niezbędne będzie pobranie dodatkowych narzędzi[28] do tworzenia grafiki w tym standardzie oraz zainstalowania w systemie operacyjnym kodeka do bezpośredniego przeglądania grafiki.

Warto dodać, że próby upowszechnienia WebP zostały już podjęte. Nowa wersja layoutu dla Google Play korzysta z grafik właśnie w formacie WebP. Także Facebook w 2013 roku przeprowadził akcję propagującą ten format i prezentował losowym użytkownikom dane w formacie WebP. Reakcja użytkowników była jednak bardzo negatywna z uwagi na fakt, że

[26] WebP – nowy rodzaj kompresji obrazów przedstawiony 30 września 2010 roku, oparty na open-source-owym kodeku video VP8, używany przez format WebM, http://blog.chromium.org/2010/09/webp-new-image-format-for-web.html, http://blog.webmproject.org/2010/05/introducing-webm-open-web-media-project.html

[27] The Structural SIMilarity (SSIM) index – strukturalne podobieństwo – metoda pomiaru podobieństwa między dwoma obrazami. Indeks SSIM może być postrzegany jako wskaźnik jakości obrazu drugiego względem pierwszego, pod warunkiem że pierwszy jest postrzegany jako idealny (SSIM=1).

[28] https://developers.google.com/speed/webp/download

poza wyświetleniem obrazów w przeglądarce nie potrafili oni nic więcej z nimi zrobić. Brak wsparcia z poziomu systemu operacyjnego i używanego oprogramowania dał o sobie znać. Mimo wszystko warto się zapoznać z tym standardem, gdyż możliwości tego formatu są bardzo obiecujące, a z uwagi na twórcę – prędzej czy później zacznie on wypierać popularne JPG.

4. Teoria w praktyce – testy i podsumowanie

W celu zobrazowania efektywności przedstawionych technik zostały wykonane dwie strony demonstracyjne. Ponieważ celem jest wskazanie efektów odnoszących się do aktualnych warunków, dla jakich aplikacje są tworzone, nie będą brane pod uwagę protokół SPDY oraz format WebP. Testy zostały przeprowadzone na podstawie wspominanego już wcześniej PageSpeed.

Wykonana strona demonstracyjna istniejąca pod adresem http://dzariusz.pl/bdas/dobrze/ jest banalnie prosta: kilka fraz tekstowych, prezentacja kilkudziesięciu flag 25x17 px, kilkudziesięciu zdjęć 150x150 px oraz kilka funkcji w języku JS z wykorzystaniem jQuery. (Wykonanie – zgodnie ze wszystkimi wskazówkami zawartymi w niniejszym artykule.) Podczas analizy Google PageSpeed pokazuje wynik 100/100 punktów zarówno w wersji dla komórek, jak i dla komputerów. Wystarczy jednak zupełnie zignorować wspominane techniki, aby ocena wydajności tak prostej strony spadła o 35% dla urządzeń mobilnych i o 26% dla stacjonarnych (http://dzariusz.pl/bdas/2014/zle/). Analizator wskazał następujące błędy (cytaty z aplikacji):

- Wyeliminuj blokujący renderowanie kod JavaScript i CSS z części strony widocznej na ekranie. Strona zawiera blokujące skrypty (3) i blokujące zasoby CSS (3). Powoduje to opóźnienia w renderowaniu strony.
- Włącz kompresowanie, aby zredukować rozmiar transferu zasobów o 38,9 KB (97%).
- Zoptymalizuj grafiki, aby zredukować ich rozmiar o 2,1 KB (14%).
- Zmniejsz plik JavaScript zasobów, aby zredukować ich rozmiar o 23,6 KB (98%).
- Zmniejsz plik CSS zasobów, aby zredukować ich rozmiar o 15,5 KB (97%).
- Zmniejsz plik HTML zasobów, aby zredukować ich rozmiar o 745 B (28%).

Straty wynikające z tego typu zaniedbań w przypadku dużego portalu mogą być ogromne.

Kolejna zapowiadana konfrontacja odbyła się na stronie z poprzedniego testu – stronie zero. Zostały utworzone dwie jej kopie: jedna przechowująca w bazie tylko sprajt z flagami, dostępny pod linkiem zewnętrznym z poziomu CSS, druga przechowująca w bazie 87 obra-

zów galerii. Nie potrzeba testów, aby stwierdzić, że 87 dodatkowych zapytań o pliki pobierane z bazy zdeklasuje stronę pod względem prędkości działania. W celu wyrównania szans do strony zero dodano instrukcję symulującą pobranie z bazy informacji o ścieżkach do plików galerii (jedno zapytanie). Natomiast na stronie galerii wszystkie zdjęcia są pobierane za pomocą jednego zapytania i umieszczane bezpośrednio w kodzie przy użyciu base64. Mimo wszystko przechowywanie danych w bazie ma wiele zalet i zgodnie ze stwierdzeniem z rozdziału 2.1 – jeśli różnica nie będzie znaczna, programista może nadać zaletom stojącym za bazą danych wyższy priorytet.

Do analizy wydajności tych trzech stron posłużyło oprogramowanie ApacheBench[29], które mierzy czas ładowania strony. Oprogramowanie to podaje dużo więcej informacji, np. liczbę zakończonych poprawnie żądań, średnią liczbę żądań wykonywanych w ciągu sekundy, czas oczekiwania na nawiązanie połączenia, średni transfer itd. W zależności od badanego problemu należy brać pod uwagę odpowiednie wyniki pomiarów. Test w zasadzie sprowadza się do skonstruowania polecenia testującego z odpowiednio dobranymi do problemu parametrami oraz odczytania wyników. Polecenie wywołujemy z linii komend, a wyniki są prezentowane w formie tekstowej na ekranie. W tym przypadku kluczowe były średni czas wykonania jednego żądania oraz wielkość dokumentu. Polecenie testujące każe pobierać stronę pod wskazanym adresem URL z wykorzystaniem kompresji danych 1000-krotnie przy równoczesnych 10 odwołaniach:

```
ab -k -H "Accept-Encoding: gzip,deflate" -d -n 1000 -c 10 URL
```

Tabela 2

Wyniki testów ApacheBench konfrontujących sposoby przechowywania danych

	Strona zero	Test flag	Test galerii
Wielkość dokumentu [b]	13 253	13 254 + 13 679	735 899
średni czas wykonania jednego żądania [ms]	1030	25 (flagi) + 1042	1099

Test flag musiał odbyć się w dwóch etapach: najpierw, strona później, sam plik flag. ApacheBench nie chciał brać pod uwagę pliku podanego w CSS jako tło (sprajt). W tym przypadku bezpośrednie wywołanie pliku strony inicjowało połączenie z bazą, a bezpośrednie wywołanie sprajtu – pobranie grafiki, dlatego wynik jest sumą. Należy zauważyć, że zdjęcia galerii były pobierane łącznie z dokumentem tylko przy trzecim teście, stąd znaczna różnica w wielkości przesłanych danych. Nie wpłynęło to jednak w dużym stopniu na uzyskany czas – różnice są na poziomie dziesiątek milisekund. Rezultaty są dość zaskakujące, zwłaszcza że wynik nie jest zaciemniony przez prędkość połączenia internetowego, gdyż testy odbyły się w obrębie jednej maszyny.

[29] ApacheBench (ab) – program do wykonywania testów wydajnościowych serwerów HTTP. Wskazówka konfiguracyjna: jeśli wystąpi błąd o kodzie 730047, należy do pliku host w OS dopisać linię „::1 localhost”, http://httpd.apache.org/docs/2.2/programs/ab.html

Z obserwacji wynika, że bardziej krytyczną wartością jest w tym przypadku liczba nawiązywanych połączeń z systemem bazy danych niż ilość pobranych z niej danych. Temat można zgłębić, badając zachowanie i zależności dla większych plików niż użyte w testach. Ciekawym eksperymentem byłby także podobny test na serwerze produkcyjnym, który w odróżnieniu od maszyny z tego testu jest bezustannie odpytywany przez klientów setek czy tysięcy innych stron WWW znajdujących się w jego obrębie. Można także pokusić się o uzyskanie czasów generowania w inny, może nawet ciekawszy sposób, np. za pomocą PhantomJS.

BIBLIOGRAFIA

1. Freeman E., Robson E.: HTML5. Rusz głową! Helion, Gliwice 2011.
2. Pasternak Ł.: CSS3. Tworzenie nowoczesnych stron WWW. Helion, Gliwice 2012.
3. Frederick G. R., Lai R.: Projektowanie witryn internetowych dla urządzeń mobilnych. Helion, Gliwice 2010.
4. Chow S.-W.: PHP Web 2.0. Helion, Gliwice 2009.
5. Vossen G., Hagemann S.: Serwis Web 2.0. Helion, Gliwice 2010.
6. Henderson C.: Skalowalne witryny internetowe. Helion, Gliwice 2007.
7. Allsopp J.: Tworzenie serwisów WWW. Helion, Gliwice 2010.
8. Schwartz B., Zaitsew P., Tkachenko V., Zawodny J. D., Lentz A., Balling D. J.: Wysoko wydajne MySQL. Helion, Gliwice 2009.
9. Wang Z., Bovik A. C., Sheikh H. R., Simoncelli E. P.: Image quality assessment: From error visibility to structural similarity. IEEE Transactions on Image Processing, Vol. 13, No. 4, 2004, s. 600÷612.
10. http://www.w3schools.com/
11. https://developers.google.com
12. http://httpd.apache.org/docs/2.2/mod/
13. http://en.wikipedia.org/wiki/Data_URI_scheme
14. http://pl.wikipedia.org/wiki/Base64,
15. http://tools.ietf.org/html/rfc4648
16. http://dev.chromium.org/spdy
17. https://developers.google.com/speed/webp/

Wpłynęło do Redakcji 2 stycznia 2014 r.

Abstract

Article examines the impact of various factors on the speed of web application. Some delivering and managing data in the presentation layer techniques such as sprites (Fig. 1 Example of three 44x44px images join into one 134x44px image sprite), were discussed and tested. Conducted a brief confrontation showing, that in certain situations storing binary data directly in a database may not be a bad solution (Table 2. Test results ApacheBench confronting ways to store data). Additionally, discusses promising: the new SPDY protocol and the new WebP graphics format, that soon can become a tempting alternative to existing optimization techniques.

Adresy

Andrzej BARCZAK: Uniwersytet Przyrodniczo Humanistyczny, Instytut Informatyki, ul. 3 Maja 54, 08-110 Siedlce, Polska, andrzej.barczak@neostrada.pl.
Dariusz ZACHARCZUK: Uniwersytet Przyrodniczo Humanistyczny, Instytut Informatyki, ul. 3 Maja 54, 08-110 Siedlce, Polska, dzariusz@dzariusz.pl.