Wydajność aplikacji webowych (1)

Systemy Rozproszone

Paweł Bedyński

Matematyka Informatyka i Mechanika
Uniwersytet Warszawski

19/11/2009

Zawężanie tematu

Aplikacje webowe

- Strony internetowe

Wydajne aplikacje

- szybkość
- meta treść
- treść

Zawężanie tematu

Aplikacje webowe

- Strony internetowe

Wydajne aplikacje

- szybkość
- meta treść
- treść

Zawężanie tematu

Aplikacje webowe

- Strony internetowe

Wydajne aplikacje

- szybkość
- meta treść
- treść

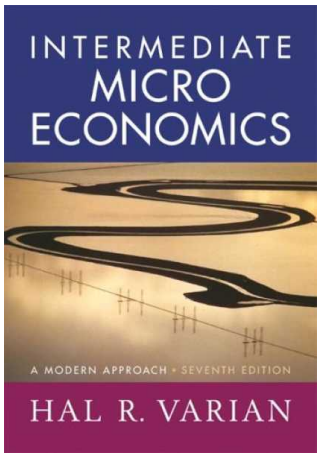
Zawężanie tematu

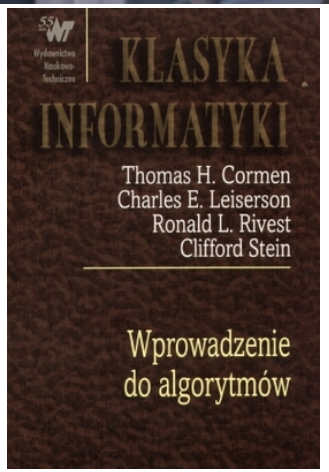
Aplikacje webowe

- Strony internetowe

Wydajne aplikacje

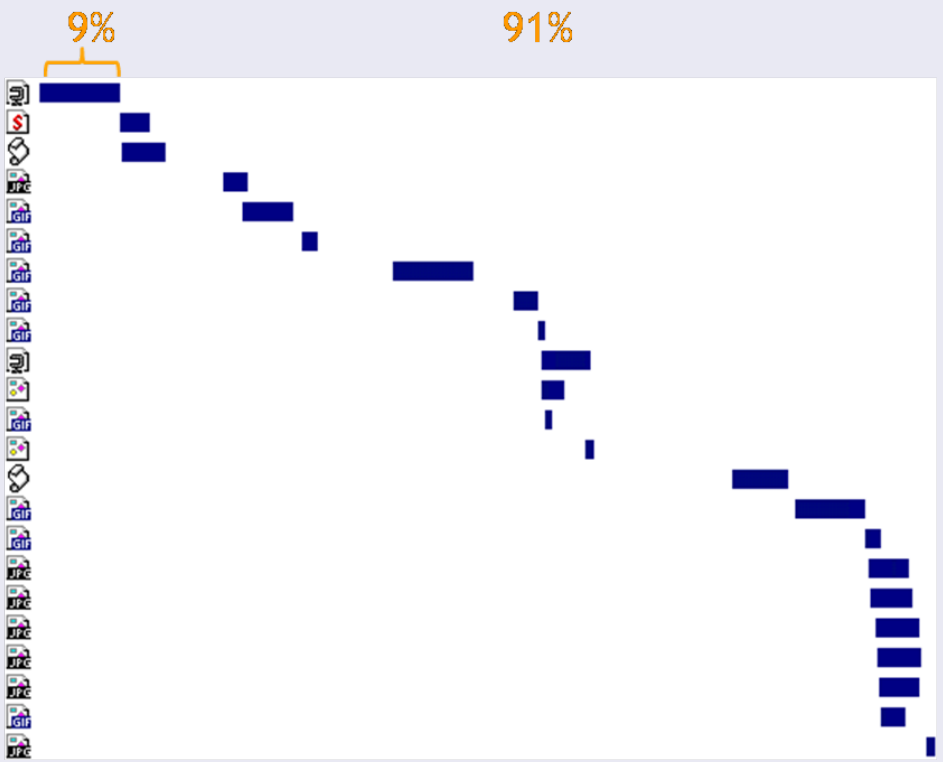
- szybkość
- meta treść
- treść





Frontend vs. Backend - sens życia według sponsora odcinka

iGoogle - pusty cache

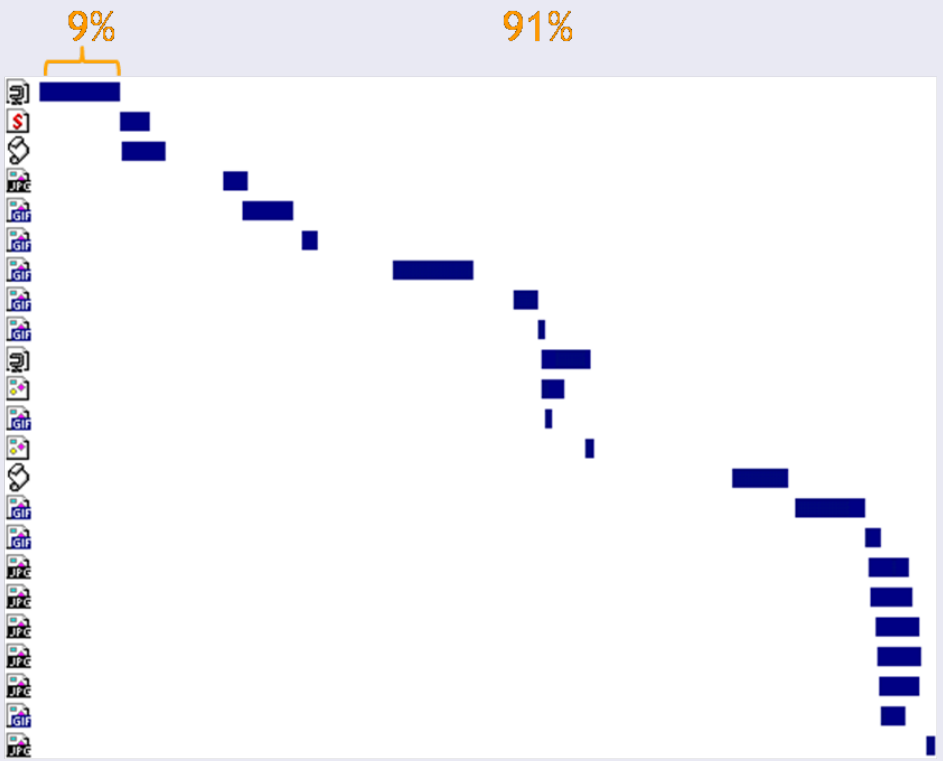


Sponsor Odcinka

Steve Souders
http://stevesouders.com/

Frontend vs. Backend - sens życia według sponsora odcinka

iGoogle - pusty cache



Sponsor Odcinka

Steve Souders
http://stevesouders.com/

iGoogle - primed cache

relacja to 17% na backend i 83% na frontend.

Frontend vs Backend - w rzeczywistości

	Empty Cache	Primed Cache
www.aol.com	97%	97%
www.ebay.com	95%	81%
www.facebook.com	95%	81%
www.google.com/search	47%	0%
search.live.com/results	67%	0%
www.msn.com	98%	94%
www.myspace.com	98%	98%
en.wikipedia.org/wiki	94%	91%
www.yahoo.com	97%	96%
www.youtube.com	98%	97%

Frontend vs Backend

Empty cache i primed cache

- Testy S.Soudersa na Yahoo
- sztuczne ”zdjecie” z *Expires* w przeszłości i *Last-Modified* stałym i ... też w przeszłości
- dwa możliwe kody
  200 pusty cache
  304 ”mam to zdjecie w tej wersji” - ok ta wersja jest dobra.
- (logi z serwera) po wyrównaniu ok 20% zapytań pochodzilo od przegladarek z pustym cachem

Frontend vs Backend

Jednak frontend

80-90% czasu który użytkownik czeka na start strony to *frontend*. Zacznijmy od niego.

- większy **potencjał** usprawnień
- **prostota**
  dbanie o backend jest droższe, to są duże projekty
  a i tak z perspektywy odbiorcy jest to ”prawie” bez znaczenia
- **potwierdzone działanie**

Dlaczego dbanie o wydajność jest istotne

Google

+500ms skutkuje -20% ruchu

Sponsor Odcinka

Steve Souders
http://stevesouders.com/

Dlaczego dbanie o wydajność jest istotne

Google

+500ms skutkuje -20% ruchu

Yahoo

+400ms skutkuje -5%-9% ruchu

Sponsor Odcinka

Steve Souders
http://stevesouders.com/

Dlaczego dbanie o wydajność jest istotne

Google

+500ms skutkuje -20% ruchu

Yahoo

+400ms skutkuje -5%-9% ruchu

Amazon

+100ms skutkuje -1% sprzedaży
to około **30.000.000$** rocznie (zysku a nie obrotu)

Sponsor Odcinka

Steve Souders
http://stevesouders.com/

Dlaczego dbanie o wydajność jest istotne

Google

+500ms skutkuje -20% ruchu

Yahoo

+400ms skutkuje -5%-9% ruchu

Amazon

+100ms skutkuje -1% sprzedaży
to około **30.000.000$** rocznie (zysku a nie obrotu)

troche angielszczyzny

carrying coal to Newcastle

Sponsor Odcinka

Steve Souders
http://stevesouders.com/

Standardowe dwie drogi

Droga pierwsza

APTIMIZE i podobne

Standardowe dwie drogi

Droga pierwsza

APTIMIZE i podobne

Droga druga

zrób to samemu

Narzędzia rozwijane przez Google

Web Page Analysis

- **Page Speed -** Open source Firefox/Firebug Add-on that evaluates the performance of web pages and gives suggestions for improvement.
- **Chrome Developer Tools -** Tools included in Google Chrome that let you edit, debug, and monitor CSS, HTML, and JavaScript live in any web page. You can also use them to optimize web page performance by profiling CPU and memory usage.

Resource Optimization

- **Closure Compiler -** Optimize the speed and size of your JavaScript.

Development tools

- **Closure Tools -** Use the Closure Compiler, Closure Library, and Closure Templates to build rich web applications with JavaScript that is faster, more powerful, and more optimized. .

Narzędzia rozwijane przez innych producentów

Development

- **Cuzillion**
- **Hammerhead**
- **OOCSS**

Web debugging

- **Fiddler 2**
- **Firebug**
- **HttpWatch**

Web page analysis

- **AOL Page Test**
- **IBM Page Detailer**
- **Microsoft VRTA**
- **MySpace Performance Tracker**
- **Yahoo! YSlow**

PHP profiling

- **Xdebug**
- **XHProf by Facebook**

Performance benchmarking

- **httperf**
- **mon.itor.us**
- **Pylot**

Resource optimization

- **CSS Sprite Generator**
- **JSLint**
- **JSMinr**
- **Smush It**
- **SpriteMe!**
- **YUI Compressor**

funkcja flush()

brak flush()

- To typowa sytuacja

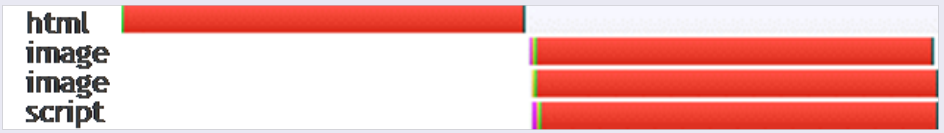
funkcja flush()

brak flush()

- To typowa sytuacja
- Przegladarka jest bezczynna przez długi okres

funkcja flush()

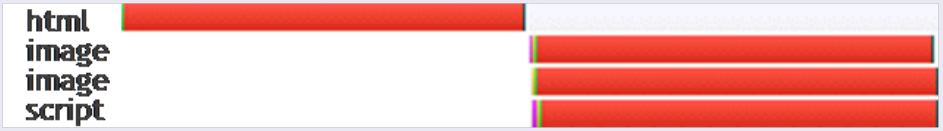
brak flush()



- To typowa sytuacja
- Przegladarka jest bezczynna przez długi okres
- ...Może można by to zmienić

funkcja flush()

brak flush()

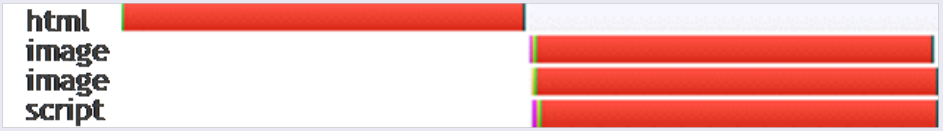


stosując flush()

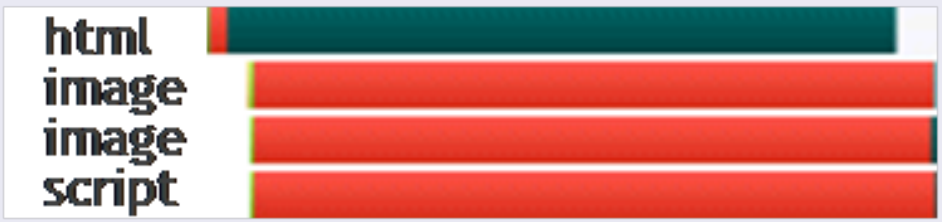
- Funkcja `flush()` (PHP)

funkcja flush()

brak flush()



stosując flush()



- Funkcja `flush()` (PHP)
- powoduje wysłanie już przetworzonych danych do przeglądarki

problemy z flush()

flush() domain blocking- IE7

problemy z flush()

flush() domain blocking- IE7

- *ob_flush()* w PHP output buffering
- **Transfer-Encoding: chunked** - powinien być włączony
- Gzip – 8K jako default (Apache DeflateBufferSize wersje przed 2.2.8)
- Poxy i antivirusy mogą blokować flushing
- Przegladarki mają minimalne limity na aktywowanie flush()

problemy z flush()

flush() domain blocking- IE7

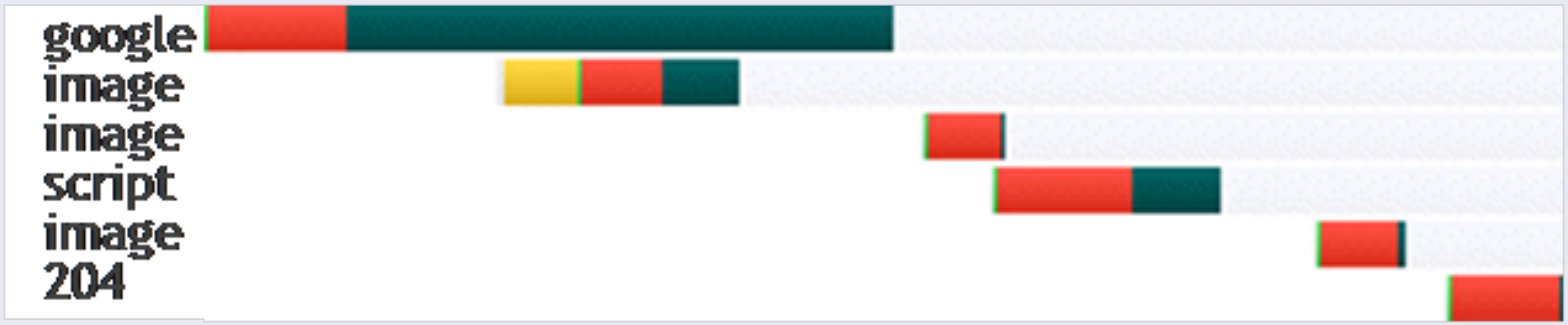
- *ob_flush()* w PHP output buffering
- **Transfer-Encoding: chunked** - powinien być włączony
- Gzip – 8K jako default (Apache DeflateBufferSize wersje przed 2.2.8)
- Poxy i antivirusy mogą blokować flushing
- Przegladarki mają minimalne limity na aktywowanie flush()

czy pakowanie danych zawsze jest dobre?

Safari(2K), Chrome(2K), IE(255B)

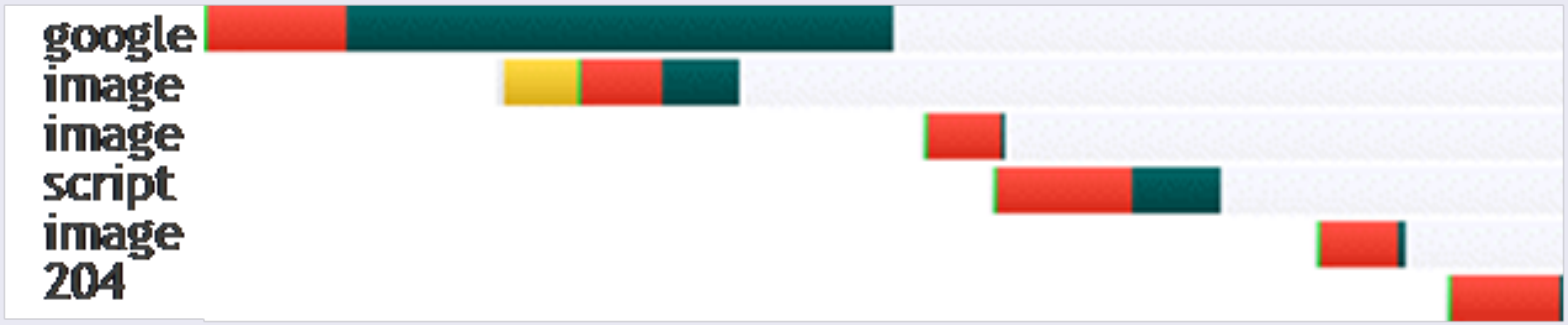
jeśli nie wiesz jak to zrobić...

...zrób to jak Google



jeśli nie wiesz jak to zrobić...

...zrób to jak Google



firmujemy wyniki w ciemno

http://www.google.com/images/nav_logo4.png

- zasoby ściagane są wcześnie
- przeglądarka szybciej renderuje grafike
- **faster user experience**

co to jest selektor

```
#object1 > LI { text-align: center; }
```

- ID selektor

kod CSS

```
#toc { margin-left: 20px; }
```

wyjaśnienie

wybiera element, którego atrybut ID ma wartość toc

co to jest selektor

```
#object1 > LI { text-align: center; }
```

- ID selektor
- selektor klasy

kod CSS

```
.chapter { font-weight: bold; }
```

wyjaśnienie

wybiera elementy z class=chapter

co to jest selektor

```
#object1 > LI { text-align: center; }
```

- ID selektor
- selektor klasy
- selektor typu

kod CSS

```
A { text-decoration: none; }
```

wyjaśnienie

wybiera wszystkie elementy A w drzewie dokumentu

co to jest selektor

```
#object1 > LI { text-align: center; }
```

- ID selektor
- selektor klasy
- selektor typu
- selektor ”następnika”

kod CSS

```
H1 + #toc { margin-top: 40px; }
```

wyjaśnienie

wybiera element z ID=toc występujący bezpośrenio po H1

co to jest selektor

```
#object1 > LI { text-align: center; }
```

- ID selektor
- selektor klasy
- selektor typu
- selektor ”następnika”
- sektor dziecka

kod CSS

```
#toc > LI { font-weight: bold; }
```

wyjaśnienie

wybiera wszystkie elemtny LI, których rodzice mają ID=toc

co to jest selektor

```
#object1 > LI { text-align: center; }
```

- ID selektor
- selektor klasy
- selektor typu
- selektor ”następnika”
- sektor dziecka
- selektor potomka

kod CSS

```
#toc A { color: #444; }
```

wyjaśnienie

wybiera wszystkie elementy A, których przodek ma ID=toc

co to jest selektor

```
#object1 > LI { text-align: center; }
```

- ID selektor
- selektor klasy
- selektor typu
- selektor ”następnika”
- sektor dziecka
- selektor potomka
- selektor uniwersalny

kod CSS

```
* { font-family: Arial; }
```

wyjaśnienie

wybiera wszystkie elementy

co to jest selektor

```
#object1 > LI { text-align: center; }
```

- ID selektor
- selektor klasy
- selektor typu
- selektor ”następnika”
- sektor dziecka
- selektor potomka
- selektor uniwersalny
- selektor po atrybutach

kod CSS

```
[href="#index"] { font-style: italic; }
```

wyjaśnienie

wybiera elementy, których atrybuty pasują do zadanego wzorca

co to jest selektor

```
#object1 > LI { text-align: center; }
```

- ID selektor
- selektor klasy
- selektor typu
- selektor ”następnika”
- sektor dziecka
- selektor potomka
- selektor uniwersalny
- selektor po atrybutach
- psudo klasy i elementy

kod CSS

```
A:hover { text-decoration: underline; }
```

wyjaśnienie

wybiera elementy w zależności od innej akcji (np. pozycji myszki)

pisanie wydajnych CSSów - przyczyna

gdzie jest haczyk?

pisanie wydajnych CSSów - przyczyna

gdzie jest haczyk?

The style system matches a rule by starting with the **rightmost** selector and **moving to the left** through the rule's selectors. As long as your little subtree continues to check out, the style system will continue moving to the left until it either matches the rule or bails out because of a mismatch.

https://developer.mozilla.org/en/Writing_Efficient_CSS

pisanie wydajnych CSSów - przyczyna

gdzie jest haczyk?

The style system matches a rule by starting with the **rightmost** selector and **moving to the left** through the rule's selectors. As long as your little subtree continues to check out, the style system will continue moving to the left until it either matches the rule or bails out because of a mismatch.
https://developer.mozilla.org/en/Writing_Efficient_CSS

przykład 1

```
#toc > LI { font-weight: bold; }
```

szuka wszystkich elementów *LI* w treści dokumentu i sprawdza czy ich rodzic ma ID="toc"

pisanie wydajnych CSSów - przyczyna

gdzie jest haczyk?

The style system matches a rule by starting with the **rightmost** selector and **moving to the left** through the rule's selectors. As long as your little subtree continues to check out, the style system will continue moving to the left until it either matches the rule or bails out because of a mismatch.
https://developer.mozilla.org/en/Writing_Efficient_CSS

przykład 1

```
#toc > LI { font-weight: bold; }
```

szuka wszystkich elementów *LI* w treści dokumentu i sprawdza czy ich rodzic ma ID="toc"

przykład 2

```
#toc A { color: #444; }
```

szuka wszystkich elementów *A* w treści dokumentu i sprawdza czy którykolwiek przodek ma ID="toc"

pisanie wydajnych CSSów - wnioski

- unikaj uniwersalnych selektorów

pisanie wydajnych CSSów - wnioski

- unikaj uniwersalnych selektorów
- unikaj kombinacji z selektorami ID

źle

```
DIV #navbar {}
```

dobrze

```
#navbar {}
```

pisanie wydajnych CSSów - wnioski

- unikaj uniwersalnych selektorów
- unikaj kombinacji z selektorami ID

źle

```
DIV #navbar {}
```

dobrze

```
#navbar {}
```

- unikaj ”nieokreśloności”

źle

```
UL LI A {}
```

lepiej

```
UL > LI > A {}
```

pisanie wydajnych CSSów - wnioski

- unikaj uniwersalnych selektorów
- unikaj kombinacji z selektorami ID

źle

```
DIV #navbar {}
```

dobrze

```
#navbar {}
```

- unikaj ”nieokreśloności”

źle

```
UL LI A {}
```

lepiej

```
UL > LI > A {}
```

- unikaj relacji dziecko/potomek

lepiej

```
UL > LI > A {}
```

najlepiej

```
.li-anchor {}
```

po co to komu ... testowanie ekstremalne

Jon Sykes
http://jon.sykes.me/153/more-css-performance-testing-pt-3

po co to komu ... testowanie ekstremalne

Jon Sykes
http://jon.sykes.me/153/more-css-performance-testing-pt-3

20.000 elementow **A**

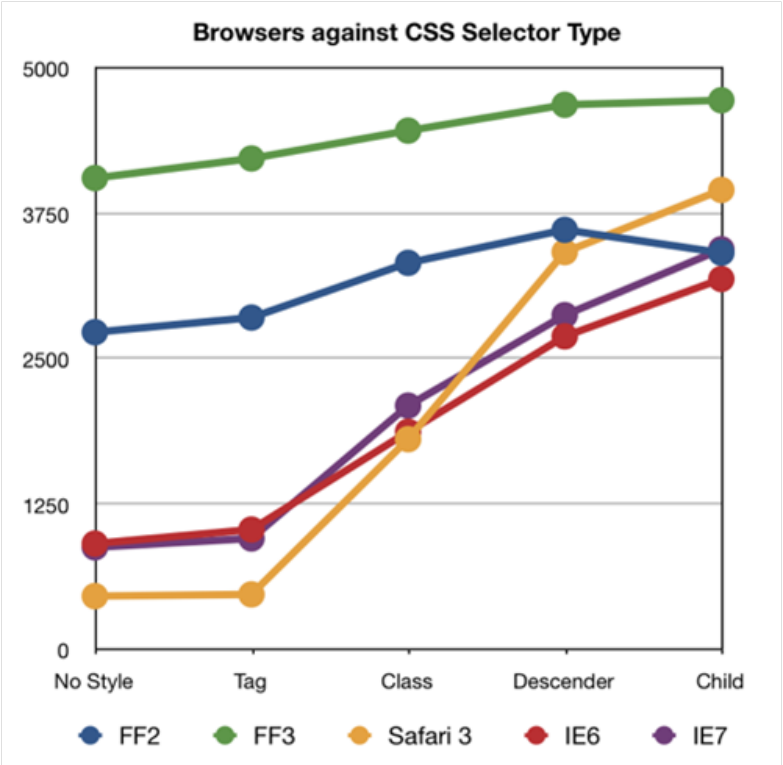
1. brak arkusza stylów
2. tylko tag: **A** {}
3. class:
   .a00001 {}
   .a00002 {}
4. potomek:
   DIV DIV DIV P A.00001 {}
5. dziecko:
   DIV > DIV > DIV > P >
   A.a00001 {}

po co to komu ... testowanie ekstremalne

Jon Sykes
http://jon.sykes.me/153/more-css-performance-testing-pt-3

20.000 elementow **A**

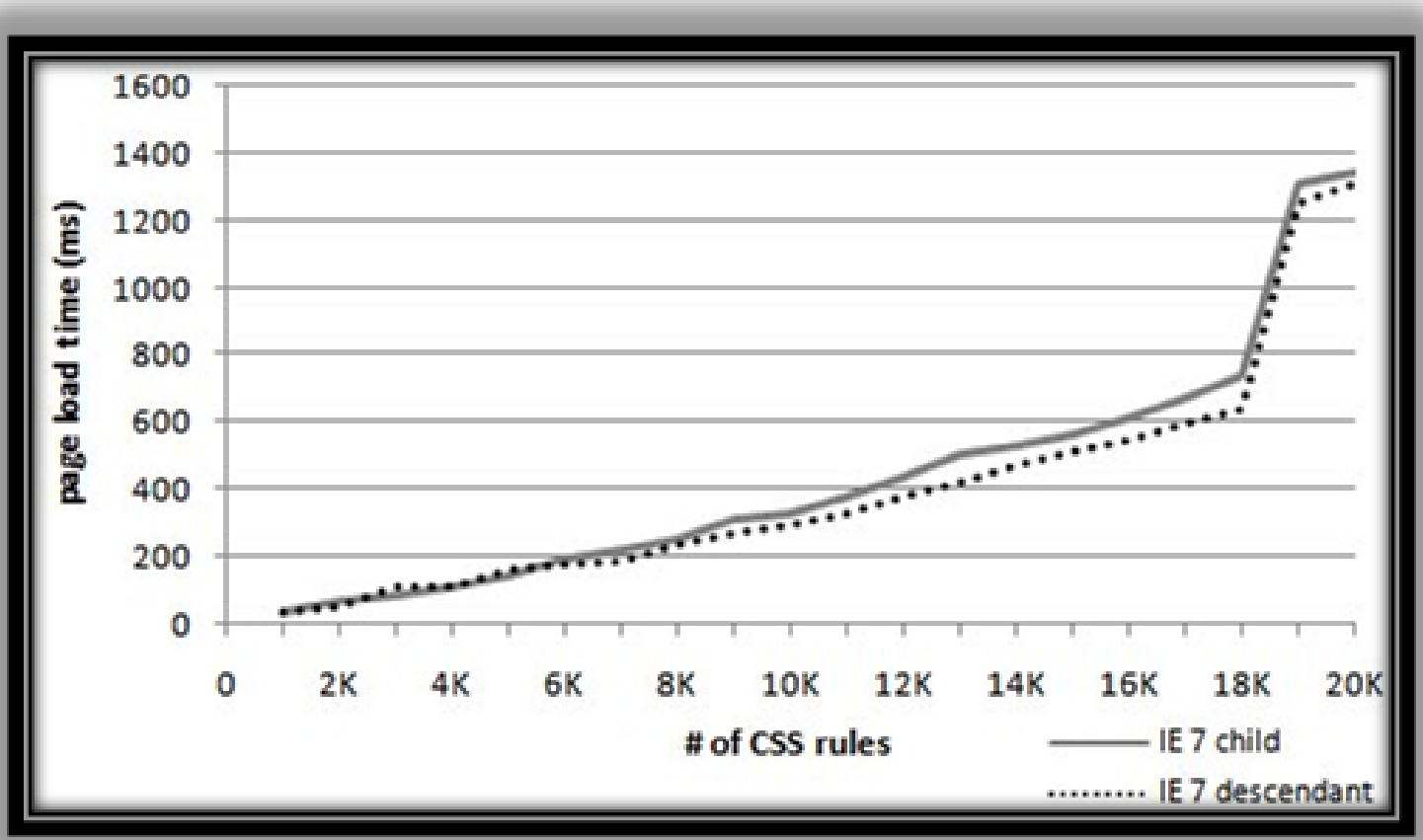
1. brak arkusza stylów
2. tylko tag: **A** {}
3. class:
   .a00001 {}
   .a00002 {}
4. potomek:
   DIV DIV DIV P A.00001 {}
5. dziecko:
   DIV > DIV > DIV > P > A.a00001 {}



mało wiarygodne testy

po co to komu ... testowanie ekstremalne (2)

Ciekawy przypadek IE7

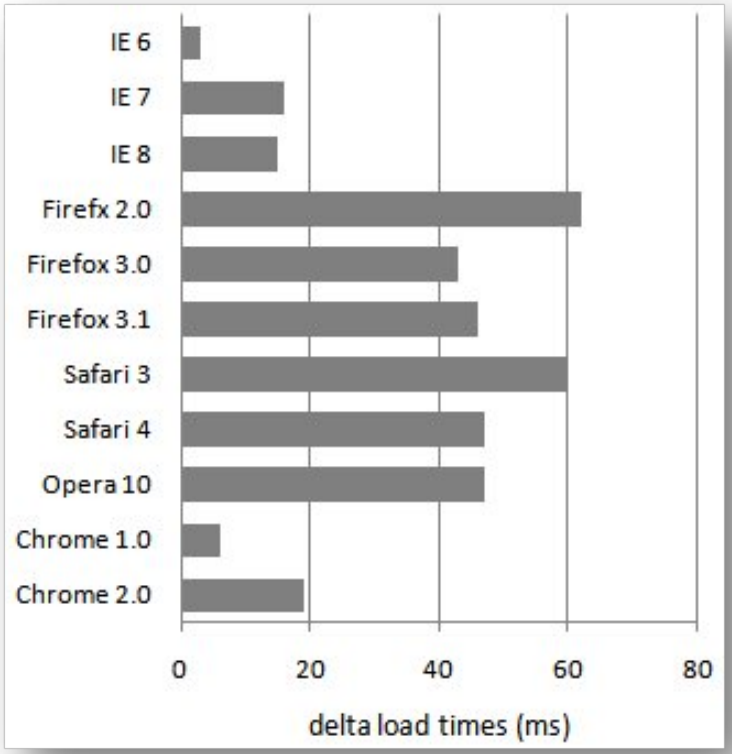


rzeczywiste przypadki

	# Rules	# elements	Avg Depth
AOL	2289	1628	13
eBay	305	588	14
Facebook	2882	1966	17
Google Search	92	552	8
Live Search	376	449	12
MSN.com	1038	886	11
MySpace	932	444	9
Wikipedia	795	1333	10
Yahoo!	800	564	13
YouTube	821	817	9
average	1033	923	12

Rzeczywiste przypadki (1)

Test 1.000 reguł VS 20.000 reguł



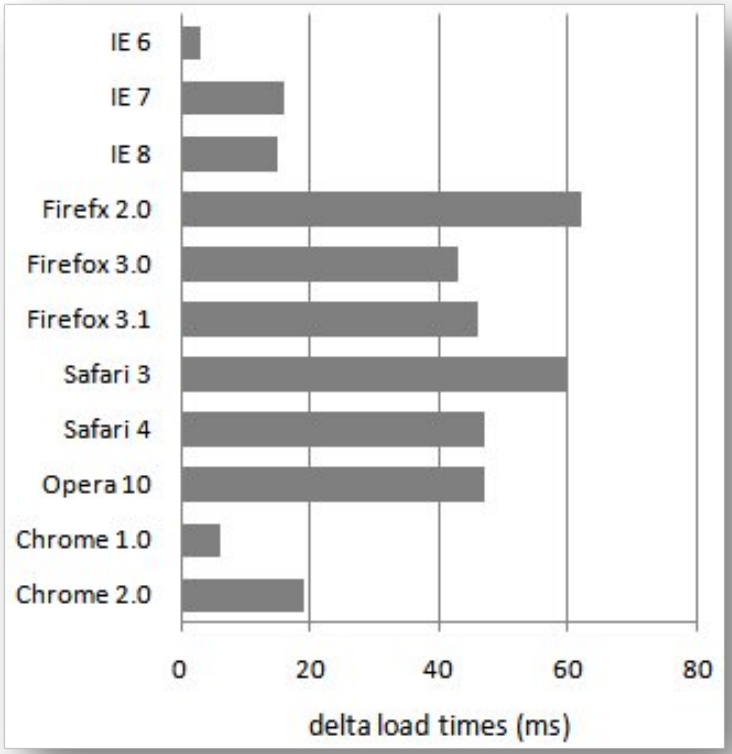
pytanie

czy zatem ”kosztowne” selektory na typowym (realistycznym) poziomie nie są tak kosztowne, jak by się to mogło wywadać

Średnia 30ms

Rzeczywiste przypadki (1)

Test 1.000 reguł VS 20.000 reguł



pytanie

czy zatem ”kosztowne” selektory na typowym (realistycznym) poziomie nie są tak kosztowne, jak by się to mogło wywadać

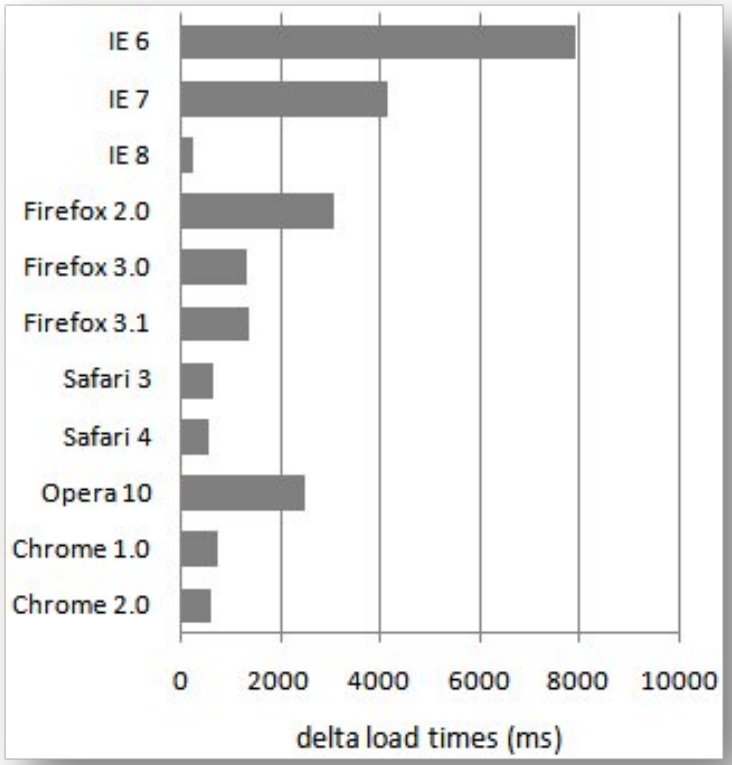
odpowiedź

może selektor DIV DIV DIV P A.class0007 {} wcale nie jest taki kosztowny

Średnia 30ms

Rzeczywiste przypadki (2)

Test 1.000 reguł VS 20.000 reguł



poprzednio badaliśmy selektor

DIV DIV DIV P A.class0007 {}

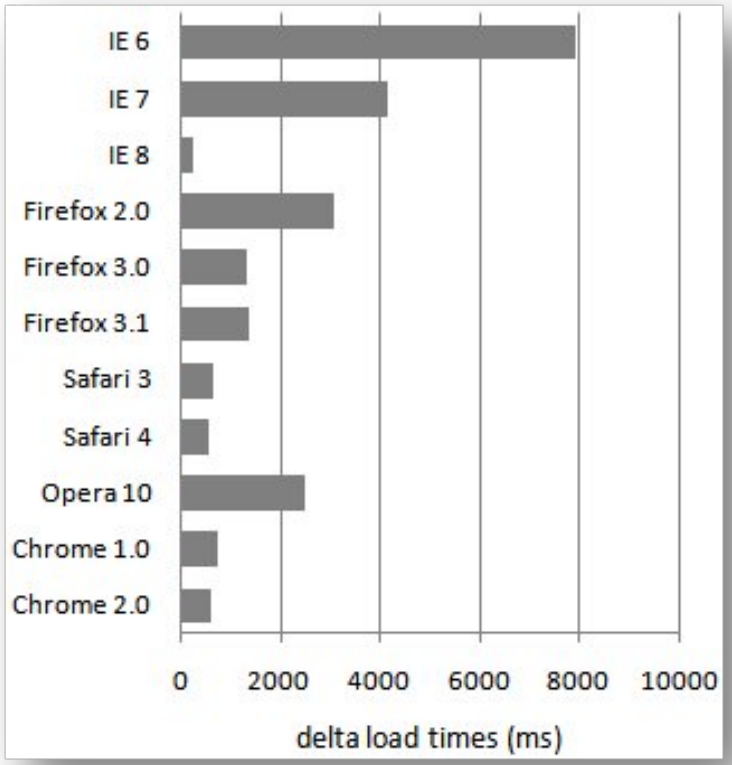
teraz badamy selektor

A.class0007 * {}

Średnia 2126ms

Rzeczywiste przypadki (2)

Test 1.000 reguł VS 20.000 reguł



Średnia 2126ms

poprzednio badaliśmy selektor

DIV DIV DIV P A.class0007 {}

teraz badamy selektor

A.class0007 * {}

wnioski

- kluczowy jest selektor pierwszy z prawej.
- optymalizowanie CSSów pod względem wydajności oprócz oczywistych przypadków, ma sens tylko przy dość dużych projektach (dużych plikach CSS)

włączamy GZIPa

Zasada Google Page Speed

Compressing resources with gzip can reduce the number of bytes sent over the network.

HTTP >=1.1

request: Accept-Encoding : gzip, deflate
response: Content-Encoding : gzip

Apache 2.x

AddOutputFilterByType DEFLATE text/html text/css application/x-javascript
w pliku konfiguracyjnym

korzyści z włączenia GZIPa

co z tego mamy

- do 70% redukcji w transferze

korzyści z włączenia GZIPa

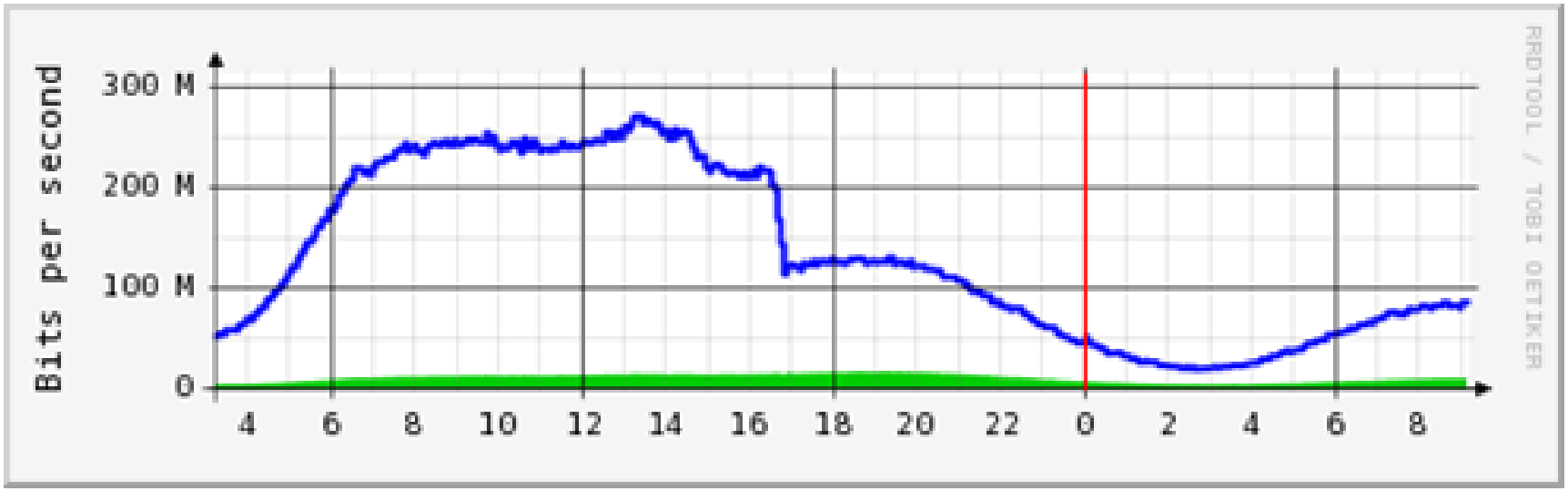
co z tego mamy

- do 70% redukcji w transferze
- i to nie tylko HTML ale również JavaStript, CSS, XML, JSON

korzyści z włączenia GZIPa

co z tego mamy

- do 70% redukcji w transferze
- i to nie tylko HTML ale również JavaStript, CSS, XML, JSON



No to włączyliśmy GZIPa. Problem z głowy?

Ale...

No to włączyliśmy GZIPa. Problem z głowy?

Ale...

15% użytkowników odbiera nieskompresowane dane. Dlaczego?

No to włączyliśmy GZIPa. Problem z głowy?

Ale...

15% użytkowników odbiera nieskompresowane dane. Dlaczego?

stare przegladarki

- Netscape Navigator 3 – 0.0%
- Netscape Communicator 4 – 0.1%
- Opera 3.5 – 0.0%
- IE <3 – 0.01%

No to włączyliśmy GZIPa. Problem z głowy?

Ale...

15% użytkowników odbiera nieskompresowane dane. Dlaczego?

stare przegladarki

- Netscape Navigator 3 – 0.0%
- Netscape Communicator 4 – 0.1%
- Opera 3.5 – 0.0%
- IE <3 – 0.01%

Podpowiedź

większość requestów pochodzi z Bliskiego Wschodu i Watykanu

biedne 15% - przyczyny

przyczyna 1 (ok. 14%)

Brakuje wpisu Accept-Encoding w nagłówku requesta

przyczyna 2 (ok. 1%)

Czasem nagłowek jest celowo zniekształcony

```
 Accept-EncodXng: gzip, deflate
X-cept-Encoding: gzip, deflate
XXXXXXXXXXXXXXX: XXXXXXXXXXXXX
```

przyczyna 3

proxy i antywirusy celowo wyłączają GZIPa aby ”łatwiej” filtrować odpowiedzi

Co można zrobić.

- nie zakładać że kompresja zadziała
- zmniejszyć transferowane dane poprzez:
- ”minimalizacje” HTML, JavaScript, CSS
- używanie CSS, zamiast wpisywania stylów ręcznie (bo to generuje wiecej znaczków)
- używanie aliasów na długie nazwy (JavaScripts)

Thank you!

URLografia

Główne źródło

1. Wykłady Steve'a Soudersa na YouTube
2. Wykorzystano zdjęcia i inne materiały z prezentacji Steve'a Soudersa

Źródła

1. Kanał na YouTube http://www.youtube.com/view_play_list?p=689D6EE903ED5CB6
2. Steve Souders home page http://stevesouders.com/
3. Steve Souders 14 rules http://stevesouders.com/hpws/rules.php
4. Google page speed - Best Practices http://code.google.com/intl/pl/speed/page-speed/docs/rules_intro.html
5. ComparePages http://stevesouders.com/compare.php
6. Aptimize http://www.aptimize.com/
7. SpriteMe http://spriteme.org/
8. Browserscope http://www.browserscope.org/