High Performance Web Sites

14 rules for faster-loading pages

Steve Souders
souders@yahoo-inc.com

Tenni Theurer
tenni@yahoo-inc.com
Introduction
Exceptional Performance

started in 2004

quantify and improve the performance of all Yahoo! products worldwide

center of expertise

build tools, analyze data

gather, research, and evangelize best practices
Scope

Performance breaks into two categories:
- response time
- efficiency

Current focus is response time of web products
The Importance of Front-End Performance
## Back-end vs. Front-end

<table>
<thead>
<tr>
<th>Website</th>
<th>Empty Cache</th>
<th>Full Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>amazon.com</td>
<td>82%</td>
<td>86%</td>
</tr>
<tr>
<td>aol.com</td>
<td>94%</td>
<td>86%</td>
</tr>
<tr>
<td>cnn.com</td>
<td>81%</td>
<td>92%</td>
</tr>
<tr>
<td>ebay.com</td>
<td>98%</td>
<td>92%</td>
</tr>
<tr>
<td>google.com</td>
<td>86%</td>
<td>64%</td>
</tr>
<tr>
<td>msn.com</td>
<td>97%</td>
<td>95%</td>
</tr>
<tr>
<td>myspace.com</td>
<td>96%</td>
<td>86%</td>
</tr>
<tr>
<td>wikipedia.org</td>
<td>80%</td>
<td>88%</td>
</tr>
<tr>
<td>yahoo.com</td>
<td>95%</td>
<td>88%</td>
</tr>
<tr>
<td>youtube.com</td>
<td>97%</td>
<td>95%</td>
</tr>
</tbody>
</table>

Percentage of time spent on the front-end
The Performance Golden Rule

80-90% of the end-user response time is spent on the front-end. Start there.

- Greater potential for improvement
- Simpler
- Proven to work
Schedule

Performance Research

break

14 Rules

break

Case Studies

Live Analysis
Performance Research
perceived response time

what is the end user’s experience?
Usability and perception are important for performance.

The user’s perception is more relevant than actual unload-to-onload response time.

Definition of "user onload" is undefined or varies from one web page to the next.
Performance Research, Part 1: What the 80/20 Rule Tells Us about Reducing HTTP Requests

November 28, 2006 at 12:56 pm by Tanni Theurer | In Development |

This is the first in a series of articles describing experiments conducted to learn more about optimizing web page performance. You may be wondering why you’re reading a performance article on the YUI Blog. It turns out that most of web page performance is affected by front-end engineering, that is, the user interface design and development.

It’s no secret that users prefer faster web sites. I work in a dedicated team focused on quantifying and improving the performance of Yahoo! products worldwide. As part of our work, we conduct experiments related to web page performance. We are sharing our findings so that other front-end engineers join us in accelerating the user experience on the web.

The 80/20 Performance Rule

Vilfredo Pareto, an economist in the early 1900s, made a famous observation where 80% of the nation’s wealth belonged to 20% of the population. This was later generalized into what’s commonly referred to as the Pareto principle (also known as the 80-20 rule), which states for any phenomenon, 80% of the consequences come from 20% of the causes. We see this phenomenon in software engineering where 80% of the time is spent in only 20% of the code. When we optimize our applications, we know to focus on that 20% of the code. This same technique should also be applied when optimizing web pages. Most performance optimization today are made on the parts that represent the HTML document (e.g., flex, database access) but three other parts...
80/20 Performance Rule

Vilfredo Pareto:
80% of consequences come from 20% of causes

Focus on the 20% that affects 80% of the end-user response time.

Start at the front-end.
Empty vs. Full Cache

1. User requests www.yahoo.com
2. User requests other web pages
3. User re-requests www.yahoo.com
Empty vs. Full Cache

User requests www.yahoo.com

With an empty cache
Empty vs. Full Cache

1. User requests www.yahoo.com
2. User requests other web pages
3. User re-requests www.yahoo.com
Empty vs. Full Cache

1. User requests www.yahoo.com
2. User requests other web pages
3. User re-requests www.yahoo.com

Expires header with a full cache
## Empty vs. Full Cache

<table>
<thead>
<tr>
<th>Empty Cache</th>
<th>Full Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>28.0K 1 HTML document</td>
<td>28.1K 1 HTML document</td>
</tr>
<tr>
<td>1.9K 1 Style Sheet File</td>
<td>0.1K 2 Images</td>
</tr>
<tr>
<td>58.5K 1 JavaScript Files</td>
<td>168.1K Total size</td>
</tr>
<tr>
<td>78.7K 24 Images</td>
<td>30 HTTP requests</td>
</tr>
<tr>
<td><strong>168.1K Total size</strong></td>
<td><strong>28.1K Total size</strong></td>
</tr>
<tr>
<td><strong>30 HTTP requests</strong></td>
<td><strong>3 HTTP requests</strong></td>
</tr>
<tr>
<td><strong>2.4s Response time</strong></td>
<td><strong>0.9s Response time</strong></td>
</tr>
</tbody>
</table>

- **empty cache**: 2.4 seconds
- **full cache**: 0.9 seconds

- 83% fewer bytes
- 90% fewer HTTP requests
How much does this benefit our users?

It depends on how many users have components in cache.

- What percentage of users view a page with an empty cache?*

  *“Empty cache” means the browser has to request the components instead of pulling them from the browser disk cache.

- What percentage of page views are done with an empty cache?*
Performance Research, Part 2: Browser Cache Usage - Exposed!
January 4, 2007 at 12:24 pm by Tenni Theurer | In Development |

This is the second in a series of articles describing experiments conducted to learn more about optimizing web page performance. You may be wondering why you’re reading a performance article on the YUI Blog. It turns out that most of web page performance is affected by front-end engineering, that is, the user interface design and development.

In an earlier post, I described What the 80/20 Rule Tells Us about Reducing HTTP Requests. Since browsers spend 80% of the time fetching external components including scripts, stylesheets and images, reducing the number of HTTP requests has the biggest impact on reducing response time. But shouldn’t everything be saved in the browser’s cache anyway?

Why does cache matter?

It’s important to differentiate between end user experiences for an empty versus a full cache page view. An “empty cache” means the browser bypasses the disk cache and has to request all the components to load the page. A “full cache” means all (or at least most) of the components are found in the disk cache and the corresponding HTTP requests are avoided.

The main reason for an empty cache page view is because the user is visiting the page for the first time and the browser has to download all the components to load the page. Other reasons include:

- The user visited the page previously but cleared the browser cache.
- The browser cache was automatically cleared, based on the browser’s settings.

Browser Cache Experiment

Add a new image to your page

```
<img src="image/blank.gif" height="1" width="1"/>
```

with the following response headers:

```
Expires: Thu, 15 Apr 2004 20:00:00 GMT
```
Requests from the browser will have one of these response status codes:

200 - The browser does not have the image in its cache.
304 - The browser has the image in its cache, but needs to verify the last modified date.
Browser Cache Experiment

What percentage of users view with an empty cache?

# unique users with at least one 200 response
---
total # unique users

What percentage of page views are done with an empty cache?

total # of 200 responses
---
# of 200 + # of 304 responses

![Browser Cache Experiment](image)
Surprising Results

- **40-60%** users with empty cache
- **~20%** page views with empty cache
Experiment Takeaways

Keep in mind the empty cache user experience. It might be more prevalent than you think!

Use different techniques to optimize full versus empty cache experience.
Performance Research, Part 3: When the Cookie Crumbles
March 1, 2007 at 4:41 pm by Tenni Theurer | In Development |

This article, co-written by Patty Chi, is the third in a series of articles describing experiments conducted to learn more about optimizing web page performance (Part 1, Part 2). You may be wondering why you're reading a performance article on the YUI Blog. It turns out that most of web page performance is affected by front-end engineering — that is, the user interface design and development.

HTTP cookies are used for a variety of reasons such as authentication and personalization. Information about cookies is exchanged in the HTTP headers between web servers and browsers. This article discusses the impact of cookies on the overall user response time.

HTTP Quick Review

Cookies originate from web servers when browsers request a page. Here is a sample HTTP header sent by the web server after a request for www.yahoo.com:

   HTTP/1.1 200 OK
   Content-Type: text/html; charset=utf-8
   Set-Cookie: C=axbode; path=/; domain=.yahoo.com

The header includes information about the response such as the protocol version, status code, and content-type. The Set-Cookie is also included in the response and in this example the name of the cookie is “C” and the value of the cookie is “abcde”. Note: The maximum size of a cookie is
HTTP response header sent by the web server:

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Set-Cookie: C=abcdefghijklmnopqrstuvwxyz; domain=.yahoo.com
HTTP request header sent by the browser:
GET / HTTP/1.1
Host: finance.yahoo.com
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; ...)
Cookie: C=abcdefghijklmnopqrstuvwxyz;
HTTP Quick Review

HTTP request header sent by the browser:

GET / HTTP/1.1
Host: autos.yahoo.com
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; ...)
Cookie: C=abcdefghijkmnopqrstuvwxyz;
HTTP request header sent by the browser:
GET / HTTP/1.1
Host: mail.yahoo.com
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; ...)
Cookie: C=abcdefghijklmnopqrstuvwxyz;
HTTP request header sent by the browser:

GET / HTTP/1.1
Host: tech.yahoo.com
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; ...)
Cookie: C=abcdefghijklmnopqrstuvwxyz;
# Impact of Cookies on Response Time

<table>
<thead>
<tr>
<th>Cookie Size</th>
<th>Time</th>
<th>Delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 bytes</td>
<td>78 ms</td>
<td>0 ms</td>
</tr>
<tr>
<td>500 bytes</td>
<td>79 ms</td>
<td>+1 ms</td>
</tr>
<tr>
<td>1000 bytes</td>
<td>94 ms</td>
<td>+16 ms</td>
</tr>
<tr>
<td>1500 bytes</td>
<td>109 ms</td>
<td>+31 ms</td>
</tr>
<tr>
<td>2000 bytes</td>
<td>125 ms</td>
<td>+47 ms</td>
</tr>
<tr>
<td>2500 bytes</td>
<td>141 ms</td>
<td>+63 ms</td>
</tr>
<tr>
<td>3000 bytes</td>
<td>156 ms</td>
<td>+78 ms</td>
</tr>
</tbody>
</table>

- keep sizes low
- 80 ms delay
- dialup users
.yahoo.com cookie sizes

- 1.55% of page views are over 1501 bytes
- 51.80% of page views are 1001-1500 bytes
- 28.86% of page views are 1-500 bytes
## Analysis of Cookie Sizes across the Web

<table>
<thead>
<tr>
<th>Website</th>
<th>Total Cookie Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon</td>
<td>60 bytes</td>
</tr>
<tr>
<td>Google</td>
<td>72 bytes</td>
</tr>
<tr>
<td>Yahoo</td>
<td>122 bytes</td>
</tr>
<tr>
<td>CNN</td>
<td>184 bytes</td>
</tr>
<tr>
<td>YouTube</td>
<td>218 bytes</td>
</tr>
<tr>
<td>MSN</td>
<td>268 bytes</td>
</tr>
<tr>
<td>eBay</td>
<td>331 bytes</td>
</tr>
<tr>
<td>MySpace</td>
<td>500 bytes</td>
</tr>
</tbody>
</table>
Experiment Takeaways

eliminate unnecessary cookies
keep cookie sizes low
set cookies at appropriate domain level
set Expires date appropriately
  - earlier date or none removes cookie sooner
Performance Research, Part 4: Maximizing Parallel Downloads in the Carpool Lane
April 11, 2007 at 11:47 am by Tonni Thurer | in Development |

This article, co-written by Steve Souders, is the fourth in a series of articles describing experiments conducted to learn more about optimizing web page performance (Part 1, Part 2, Part 3). You may be wondering why you’re reading a performance article on the YUI Blog. It turns out that most of web page performance is affected by front-end engineering, that is, the user interface design and development.

Parallel Downloads

The biggest impact on end-user response times is the number of components in the page. Each component requires an extra HTTP request, perhaps not when the cache is full, but definitely when the cache is empty. Knowing that the browser performs HTTP requests in parallel, you may ask why the number of HTTP requests affects response time. Can’t the browser download them all at once?

The explanation goes back to the HTTP/1.1 spec, which suggests that browsers download two components in parallel per hostname. Many web pages download all their components from a single hostname. Viewing these HTTP requests reveals a stair-step pattern, as shown
Parallel Downloads

<table>
<thead>
<tr>
<th>Two components</th>
<th>in parallel</th>
<th>per hostname</th>
</tr>
</thead>
</table>

HTTP/1.1
Parallel Downloads

Two in parallel
Four in parallel
Eight in parallel
Maximizing Parallel Downloads

response time (seconds)

aliases
Maximizing Parallel Downloads

- Response time (seconds)
- Aliases

Graph showing the relationship between response time and number of aliases for different image sizes:
- 36 x 36 px (0.9 Kb)
- 116 x 61 px (3.4 Kb)
Maximizing Parallel Downloads

- Average response time (seconds)
- 36 x 36 px (0.9 Kb)
- 116 x 61 px (3.4 Kb)

Aliases

- 1
- 2
- 4
- 5
- 10
Maximizing Parallel Downloads

rule of thumb: use at least two but no more than four aliases

response time (seconds)
Experiment Takeaways

consider the effects of CPU thrashing
DNS lookup times vary across ISPs and geographic locations
domain names may not be cached
Summary

What the 80/20 Rule Tells Us about Reducing HTTP Requests

Browser Cache Usage - Exposed!

When the Cookie Crumbles

Maximizing Parallel Downloads in the Carpool Lane
14 Rules
14 Rules

1. Make fewer HTTP requests
2. Use a CDN
3. Add an Expires header
4. Gzip components
5. Put CSS at the top
6. Move JS to the bottom
7. Avoid CSS expressions
8. Make JS and CSS external
9. Reduce DNS lookups
10. Minify JS
11. Avoid redirects
12. Remove duplicate scripts
13. Turn off ETags
14. Make AJAX cacheable and small
Rule 1: Make fewer HTTP requests

- image maps
- CSS sprites
- inline images
- combined scripts, combined stylesheets
Image maps

server-side

\[
\text{<a href="navbar.cgi">\text{<img ismap src="imagemap.gif">}\text{/a>}
}\rightarrow \text{http://.../navbar.cgi?127,13}
\]

client-side - preferred

\[
\text{<img usemap="#map1" border=0 src="/images/imagemap.gif">}
\text{<map name="map1"}\
\text{  \text{<area shape="rect" coords="0,0,31,31" href="home.html" title="Home">}}
\text{...}
\text{</map>}
\]

drawbacks:
- must be contiguous
- defining area coordinates - tedious, errors

http://www.w3.org/TR/html401/struct/objects.html#h-13.6
CSS Sprites - Preferred

size of combined image is less
not supported in Opera 6

http://alistapart.com/articles/sprites
Inline Images

**data**: URL scheme

```javascript
data:[<mediatype>][;base64],<data>
```

```html
<IMG ALT="Red Star" ⭐
SRC="data:image/gif;base64,R0lGODlhDAAMALMLAPN8ffBiYvWWlvrKy/FvcPewsO9VVfa
ajo+w6O/z15estLv/8/AAAAAAAAAAAAAACH5BAEAAAAsALAAAAAMAAwAAAQzcElZyryTE
HyTUGknHd9xGV+qKsYirKkwDYiKDBiatt2H1KBLQRFIJAIKywRgmhwAIlEEADs=">
```

**not supported in IE**

**avoid increasing size of HTML pages:**

put inline images in cached stylesheets

## Combined Scripts, Combined Stylesheets

<table>
<thead>
<tr>
<th>Domain</th>
<th>Scripts</th>
<th>Stylesheets</th>
</tr>
</thead>
<tbody>
<tr>
<td>amazon.com</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>aol.com</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>cnn.com</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>ebay.com</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>froogle.google.com</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>msn.com</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>myspace.com</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>wikipedia.org</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>yahoo.com</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>youtube.com</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>6.5</strong></td>
<td><strong>1.5</strong></td>
</tr>
</tbody>
</table>
Combined Scripts, Combined Stylesheets

combining six scripts into one eliminates five HTTP requests

challenges:
- develop as separate modules
- number of possible combinations vs. loading more than needed
- maximize browser cache

one solution:
- dynamically combine and cache
**Rule 2: Use a CDN**

<table>
<thead>
<tr>
<th>Domain</th>
<th>Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>amazon.com</td>
<td>Akamai</td>
</tr>
<tr>
<td>aol.com</td>
<td>Akamai</td>
</tr>
<tr>
<td>cnn.com</td>
<td></td>
</tr>
<tr>
<td>ebay.com</td>
<td>Akamai, Mirror Image</td>
</tr>
<tr>
<td>google.com</td>
<td></td>
</tr>
<tr>
<td>msn.com</td>
<td>SAVVIS</td>
</tr>
<tr>
<td>myspace.com</td>
<td>Akamai, Limelight</td>
</tr>
<tr>
<td>wikipedia.org</td>
<td></td>
</tr>
<tr>
<td>yahoo.com</td>
<td>Akamai</td>
</tr>
<tr>
<td>youtube.com</td>
<td></td>
</tr>
</tbody>
</table>

distribute your static content before distributing your dynamic content
Rule 3: Add an Expires header
not just for images

<table>
<thead>
<tr>
<th>URL</th>
<th>Images</th>
<th>Stylesheets</th>
<th>Scripts</th>
<th>%</th>
<th>Median Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>amazon.com</td>
<td>0/62</td>
<td>0/1</td>
<td>0/3</td>
<td>0%</td>
<td>114 days</td>
</tr>
<tr>
<td>aol.com</td>
<td>23/43</td>
<td>1/1</td>
<td>6/18</td>
<td>48%</td>
<td>217 days</td>
</tr>
<tr>
<td>cnn.com</td>
<td>0/138</td>
<td>0/2</td>
<td>2/11</td>
<td>1%</td>
<td>227 days</td>
</tr>
<tr>
<td>ebay.com</td>
<td>16/20</td>
<td>0/2</td>
<td>0/7</td>
<td>55%</td>
<td>140 days</td>
</tr>
<tr>
<td>froogle.google.com</td>
<td>1/23</td>
<td>0/1</td>
<td>0/1</td>
<td>4%</td>
<td>454 days</td>
</tr>
<tr>
<td>msn.com</td>
<td>32/35</td>
<td>1/1</td>
<td>3/9</td>
<td>80%</td>
<td>34 days</td>
</tr>
<tr>
<td>myspace.com</td>
<td>0/18</td>
<td>0/2</td>
<td>0/2</td>
<td>0%</td>
<td>1 day</td>
</tr>
<tr>
<td>wikipedia.org</td>
<td>6/8</td>
<td>1/1</td>
<td>2/3</td>
<td>75%</td>
<td>1 day</td>
</tr>
<tr>
<td>yahoo.com</td>
<td>23/23</td>
<td>1/1</td>
<td>4/4</td>
<td>100%</td>
<td>n/a</td>
</tr>
<tr>
<td>youtube.com</td>
<td>0/32</td>
<td>0/3</td>
<td>0/7</td>
<td>0%</td>
<td>26 days</td>
</tr>
</tbody>
</table>
Rule 4: Gzip components

you can affect users' download times
90%+ of browsers support compression
## Gzip vs. Deflate

<table>
<thead>
<tr>
<th></th>
<th>Gzip</th>
<th></th>
<th>Deflate</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size</td>
<td>Size</td>
<td>Savings</td>
<td>Size</td>
</tr>
<tr>
<td>Script</td>
<td>3.3K</td>
<td>1.1K</td>
<td>67%</td>
<td>1.1K</td>
</tr>
<tr>
<td>Script</td>
<td>39.7K</td>
<td>14.5K</td>
<td>64%</td>
<td>16.6K</td>
</tr>
<tr>
<td>Stylesheet</td>
<td>1.0K</td>
<td>0.4K</td>
<td>56%</td>
<td>0.5K</td>
</tr>
<tr>
<td>Stylesheet</td>
<td>14.1K</td>
<td>3.7K</td>
<td>73%</td>
<td>4.7K</td>
</tr>
</tbody>
</table>

Gzip compresses more

Gzip supported in more browsers
## Gzip: not just for HTML

<table>
<thead>
<tr>
<th>Domain</th>
<th>HTML</th>
<th>Scripts</th>
<th>Stylesheets</th>
</tr>
</thead>
<tbody>
<tr>
<td>amazon.com</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>aol.com</td>
<td>x</td>
<td>some</td>
<td>some</td>
</tr>
<tr>
<td>cnn.com</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ebay.com</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>froogle.google.com</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>msn.com</td>
<td>x</td>
<td>deflate</td>
<td>deflate</td>
</tr>
<tr>
<td>myspace.com</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>wikipedia.org</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>yahoo.com</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>youtube.com</td>
<td>x</td>
<td>some</td>
<td>some</td>
</tr>
</tbody>
</table>

Gzip scripts, stylesheets, XML, JSON (not images, PDF)
Gzip Configuration

Apache 2.x: mod_deflate
AddOutputFilterByType DEFLATE text/html text/css application/x-javascript

HTTP request
Accept-Encoding: gzip, deflate

HTTP response
Content-Encoding: gzip
Vary: Accept-Encoding

needed for proxies
Gzip Edge Cases

<1% of browsers have problems with gzip
- IE 5.5:
  http://support.microsoft.com/default.aspx?scid=kb;en-us;Q313712
- IE 6.0:
  http://support.microsoft.com/default.aspx?scid=kb;en-us;Q31249
- Netscape 3.x, 4.x
  http://www.schroepl.net/projekte/mod_gzip/browser.htm

consider adding Cache-Control: Private

remove ETags (Rule 13)

hard to diagnose; problem getting smaller
Rule 5: Put CSS at the top

stylesheets block rendering in IE
http://stevesouders.com/examples/css-bottom.php

solution: put stylesheets in HEAD (per spec)

avoids Flash of Unstyled Content

use LINK (not @import)
Slowest is Fastest

css-bottom.php
image 1
image 2
image 3
image 4
image 5
image 6
stylesheet
6.3 seconds

css-top.php
stylesheet
image 1
image 2
image 3
image 4
image 5
image 6
7.3 seconds

css-top-import.php
image 1
image 2
image 3
image 4
image 5
image 6
stylesheet
6.3 seconds
Rule 6: Move scripts to the bottom

scripts block parallel downloads across all hostnames
scripts block rendering of everything below them in the page

IE and FF
http://stevesouders.com/examples/js-middle.php
**Rule 6: Move scripts to the bottom**

script defer attribute is not a solution
- blocks rendering and downloads in FF
- slight blocking in IE

solution: move them as low in the page as possible
Rule 7: Avoid CSS expressions

used to set CSS properties dynamically in IE

width: expression(
   document.body.clientWidth < 600 ?
   "600px" : "auto" );

problem: expressions execute many times
- mouse move, key press, resize, scroll, etc.

http://stevesouders.com/examples/expression-counter.php
One-Time Expressions

expression overwrites itself

```html
<style>
P {
    background-color: expression(altBgcolor(this));
}
</style>

<script>
function altBgcolor(elem) {
    elem.style.backgroundColor = (new Date()).getHours()%2 ? "#F08A00" : "#B8D4FF";
}
</script>
```
Event Handlers

tie behavior to (fewer) specific events

window.onresize = setMinWidth;
function setMinWidth() {
    var aElements =
        document.getElementsByTagName("p");
    for ( var i = 0; i < aElements.length; i++ ) {
        aElements[i].runtimeStyle.width = (document.body.clientWidth<600 ? 
            "600px" : "auto" );
    }
}
**Rule 8: Make JS and CSS external**

inline: HTML document is bigger
external: more HTTP requests, but cached
variables
- page views per user (per session)
- empty vs. full cache stats
- component re-use

external is typically better
- home pages may be an exception
Post-Onload Download

inline in front page

download external files after onload

```javascript
window.onload = downloadComponents;
function downloadComponents() {
  var elem = document.createElement("script");
  elem.src = "http://.../file1.js";
  document.body.appendChild(elem);
  ...
}
```

speeds up secondary pages
Dynamic Inlining

start with post-onload download
set cookie after components downloaded
server-side:
  - if cookie, use external
  - else, do inline with post-onload download
cookie expiration date is key
speeds up all pages
Rule 9: Reduce DNS lookups

typically 20-120 ms
block parallel downloads
OS and browser both have DNS caches
## TTL (Time To Live)

<table>
<thead>
<tr>
<th>Website</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>1 minute</td>
</tr>
<tr>
<td><a href="http://www.aol.com">www.aol.com</a></td>
<td>1 minute</td>
</tr>
<tr>
<td><a href="http://www.cnn.com">www.cnn.com</a></td>
<td>10 minutes</td>
</tr>
<tr>
<td><a href="http://www.ebay.com">www.ebay.com</a></td>
<td>1 hour</td>
</tr>
<tr>
<td><a href="http://www.google.com">www.google.com</a></td>
<td>5 minutes</td>
</tr>
<tr>
<td><a href="http://www.msn.com">www.msn.com</a></td>
<td>5 minutes</td>
</tr>
<tr>
<td><a href="http://www.myspace.com">www.myspace.com</a></td>
<td>1 hour</td>
</tr>
<tr>
<td><a href="http://www.wikipedia.org">www.wikipedia.org</a></td>
<td>1 hour</td>
</tr>
<tr>
<td><a href="http://www.yahoo.com">www.yahoo.com</a></td>
<td>1 minute</td>
</tr>
<tr>
<td><a href="http://www.youtube.com">www.youtube.com</a></td>
<td>5 minutes</td>
</tr>
</tbody>
</table>

**TTL** - how long record can be cached. Browser settings override TTL.
Browser DNS Cache

IE
- DnsCacheTimeout: 30 minutes
- KeepAliveTimeout: 1 minute
- ServerInfoTimeout: 2 minutes

Firefox
- network.dnsCacheExpiration: 1 minute
- network.dnsCacheEntries: 20
- network.http.keep-alive.timeout: 5 minutes
- Fasterfox: 1 hour, 512 entries, 30 seconds
Reducing DNS Lookups

fewer hostnames - 2-4
keep-alive
# Rule 10: Minify JavaScript

<table>
<thead>
<tr>
<th></th>
<th>Minify External?</th>
<th>Minify Inline?</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><a href="http://www.aol.com">www.aol.com</a></td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><a href="http://www.cnn.com">www.cnn.com</a></td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><a href="http://www.ebay.com">www.ebay.com</a></td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>froogle.google.com</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td><a href="http://www.msn.com">www.msn.com</a></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td><a href="http://www.myspace.com">www.myspace.com</a></td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><a href="http://www.wikipedia.org">www.wikipedia.org</a></td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><a href="http://www.yahoo.com">www.yahoo.com</a></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td><a href="http://www.youtube.com">www.youtube.com</a></td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

*minify inline scripts, too*
# Minify vs. Obfuscate

<table>
<thead>
<tr>
<th>Domain</th>
<th>Original</th>
<th>JSMin Savings</th>
<th>Dojo Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>204K</td>
<td>31K (15%)</td>
<td>48K (24%)</td>
</tr>
<tr>
<td><a href="http://www.aol.com">www.aol.com</a></td>
<td>44K</td>
<td>4K (10%)</td>
<td>4K (10%)</td>
</tr>
<tr>
<td><a href="http://www.cnn.com">www.cnn.com</a></td>
<td>98K</td>
<td>19K (20%)</td>
<td>24K (25%)</td>
</tr>
<tr>
<td><a href="http://www.myspace.com">www.myspace.com</a></td>
<td>88K</td>
<td>23K (27%)</td>
<td>24K (28%)</td>
</tr>
<tr>
<td><a href="http://www.wikipedia.org">www.wikipedia.org</a></td>
<td>42K</td>
<td>14K (34%)</td>
<td>16K (38%)</td>
</tr>
<tr>
<td><a href="http://www.youtube.com">www.youtube.com</a></td>
<td>34K</td>
<td>8K (22%)</td>
<td>10K (29%)</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>85K</strong></td>
<td><strong>17K (21%)</strong></td>
<td><strong>21K (25%)</strong></td>
</tr>
</tbody>
</table>

- **minify** - it's safer

http://crockford.com/javascript/jsmin
http://dojotoolkit.org/docs/shrinksafe
Rule 11: Avoid redirects

3xx status codes - mostly 301 and 302
HTTP/1.1 301 Moved Permanently
Location: http://stevesouders.com/newuri

add Expires headers to cache redirects
worst form of blocking

http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
## Redirects

<table>
<thead>
<tr>
<th>URL</th>
<th>Redirects</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td>no</td>
</tr>
<tr>
<td><a href="http://www.aol.com">www.aol.com</a></td>
<td>yes - secondary page</td>
</tr>
<tr>
<td><a href="http://www.cnn.com">www.cnn.com</a></td>
<td>yes - initial page</td>
</tr>
<tr>
<td><a href="http://www.ebay.com">www.ebay.com</a></td>
<td>yes - secondary page</td>
</tr>
<tr>
<td>froogle.google.com</td>
<td>no</td>
</tr>
<tr>
<td><a href="http://www.msn.com">www.msn.com</a></td>
<td>yes - initial page</td>
</tr>
<tr>
<td><a href="http://www.myspace.com">www.myspace.com</a></td>
<td>yes - secondary page</td>
</tr>
<tr>
<td><a href="http://www.wikipedia.org">www.wikipedia.org</a></td>
<td>yes - secondary page</td>
</tr>
<tr>
<td><a href="http://www.yahoo.com">www.yahoo.com</a></td>
<td>yes - secondary page</td>
</tr>
<tr>
<td><a href="http://www.youtube.com">www.youtube.com</a></td>
<td>no</td>
</tr>
</tbody>
</table>
Avoid Redirects

missing trailing slash
- http://astrology.yahoo.com/astrology
- use Alias or DirectorySlash

mod_rewrite

CNAMEs

log referer - track internal links

outbound links - harder
- beacons - beware of race condition
- XHR - bail at readyState 2
Rule 12: Remove duplicate scripts

- hurts performance
  - extra HTTP requests (IE only)
  - extra executions

- atypical?
  - 2 of 10 top sites contain duplicate scripts

- team size, # of scripts
<?php
function insertScript($jsfile) {
    if ( alreadyInserted($jsfile) ) { return; }

    pushInserted($jsfile);

    if ( hasDependencies($jsfile) ) {
        $dependencies = getDependencies($jsfile);
        for ( $i = 0; $i < count($dependencies); $i++ ) {
            insertScript($dependencies[$i]);
        }
    }
}

echo '<script type="text/javascript" src="' .
    getVersion($jsfile) . '"
    <![endif]-->
    </script>";
}?>
Rule 13: Turn off ETags

unique identifier returned in response
ETag: "c8897e-ae4-4165acf0"
Last-Modified: Thu, 07 Oct 2004 20:54:08 GMT

used in conditional GET requests
If-None-Match: "c8897e-ae4-4165acf0"
If-Modified-Since: Thu, 07 Oct 2004 20:54:08 GMT

if ETag doesn't match, can't send 304
The Problem with ETags

ETag for a single entity is always different across servers

ETag format
- Apache: inode-size-timestamp
- IIS: Filetimestamp:ChangeNumber

Sites with >1 server return too few 304s
- (n-1)/n

Remove them
- Apache: FileETag none
- IIS: http://support.microsoft.com/kb/922703/
Rule 14: Make AJAX cacheable
and small
XHR, JSON, iframe, dynamic scripts can
still be cached, minified, and gzipped
a personalized response should still be
cacheable by that person
AJAX Example: Yahoo! Mail Beta

address book XML request
→ GET /yab/[...]&r=0.5289571053069156 HTTP/1.1
   Host: us.xxx.mail.yahoo.com
← HTTP/1.1 200 OK
   Date: Thu, 12 Apr 2007 19:39:09 GMT
   Cache-Control: private,max-age=0
   Last-Modified: Sat, 31 Mar 2007 01:17:17 GMT
   Content-Type: text/xml; charset=utf-8
   Content-Encoding: gzip

address book changes infrequently
- cache it; add last-modified-time in URL
IBM Page Detailer

packet sniffer
Windows only
IE, FF, any .exe
  c:\windows\wd_WS2s.ini
  Executable=(NETSCAPE.EXE),(NETSCP6.EXE),(firefox.exe)

free trial, $300 license

http://alphaworks.ibm.com/tech/pagedetailer
http://alphaworks.ibm.com/tech/pagedetailer
Fasterfox

measures load time of pages
alters config settings for faster loading
Firefox extension
free

http://fasterfox.mozdev.org/
LiveHTTPHeaders

view HTTP headers
Firefox extension
free

http://livehttpheaders.mozdev.org/
Firebug

web development evolved
inspect and edit HTML
tweak and visualize CSS
debug and profile JavaScript
monitor network activity (caveat)
Firefox extension
free

http://getfirebug.com/
YSlow

performance lint tool
grades web pages for each rule
Firefox extension
Yahoo! internal tool
Disturbing thoughts

Find out what Yahoo! users think of "Disturbia" – a modern twist on the classic "Rear Window." » Reviews

- Showtimes  Trailers  Photos
- LeSeour to star in 'Indiana Jones 4'

Users review the new thriller 'Disturbia'

Scientists: Scrap the Internet and start over

Performance Grade: A (94)

1. Minimize HTTP requests
   - This page has 4 external JavaScript files.

2. Use edge computing

3. Add an Expires header

4. Gzip components
   - These components are not gzipped:
     - (1.9K) http://i.yimg.com/us.js.yimg.com/lib/bc/bc_2.0.3.js

5. Move CSS to the top
Conclusion
Takeaways

focus on the front-end
harvest the low-hanging fruit
you *do* control user response times
LOFNO - be an advocate for your users
Links

examples: http://stevesouders.com/examples/
image maps: http://www.w3.org/TR/html401/struct/objects.html#h-13.6
CSS sprites: http://alistapart.com/articles/sprites
jsmin: http://crockford.com/javascript/jsmin
dojo compressor: http://dojotoolkit.org/docs/shrinksafe
HTTP status codes: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
IBM Page Detailer: http://alphaworks.ibm.com/tech/pagedetailer
Fasterfox: http://fasterfox.mozdev.org/
LiveHTTPHeaders: http://livehttpheaders.mozdev.org/
Firebug: http://getfirebug.com/
http://yuiblog.com/blog/2007/03/01/performance-research-part-3/
YDN: http://developer.yahoo.net/blog/archives/2007/03/high_performanc.html
http://developer.yahoo.net/blog/archives/2007/04/rule_1_make_few.html