Understanding Web Performance

Day 12
HTTP/1.1: The standard to load Web pages

HTTP/1.1 becomes slow for rich, modern pages

Google developed SPDY to make the Web faster

- Starting to be deployed
- Standardized HTTP/2.0

Google developed QUIC
Difficult to understand page load and its Impact

- Factors that affect page load
  - Page structure
  - Inter-dependencies between network and computation activities
  - **Browser implementations**

- Factors that determine page load impact
  - User satisfaction (Click thru)
  - User experience (pagination/query refinement)
Agenda

• Understanding Implication of page load times

• Challenges quantifying Page load times

• WPROF: Capturing Dependencies

• SPDY v HTTP1.1
Understanding Impact of Page load Time

• Why understand this?
  – Motivates performance discussions and analysis
  – Informs engineering efforts and investments

• Interesting Questions:
  – If the whole page is slow what happens?
  – If page results are slow but header is fast, what happens?
  – If ads are slow what happens?
New York City - Wikipedia, the free encyclopedia

The City of New York, commonly called New York City and New York, has been the most populous city in the United States since 1790. [2] while the New York metropolitan area ranks...

History · Geography · Cityscape · Culture and ... · Economy · Demographics
en.wikipedia.org/wiki/New_York_City · enhanced view

Las Vegas Hotel - New York New York Hotel & Casino
Make reservations at New York - New York Hotel & Casino in Las Vegas.
www.nynyhotelcasino.com · cached page
ALL RESULTS
1-20 of 465,000,000 results · advanced

New York City - Wikipedia, the free encyclopedia
The City of New York, commonly called New York City and New York, has been the most populous city in the United States since 1790. [2] while the New York metropolitan area ranks ... History · Geography · Cityscape · Culture and ... · Economy · Demographics
en.wikipedia.org/wiki/New_York_City · enhanced view

Las Vegas Hotel - New York New York Hotel & Casino
Make reservations at New York - New York Hotel & Casino in Las Vegas.
www.nynyhotelcasino.com · cached page
Google Web Search Delay Experiments

• A series of experiments on a small % search traffic to measure the impact of latency on user behavior

• Randomly assign users to the experiment and control groups (A/B testing)

• Server-side delay:
  – Emulates additional server processing time
  – May be partially masked by network connection

• Varied type of delay, magnitude (in ms), and duration (number of weeks)
Server Delays Experiment: Results

- Results from Delay EVERYTHING!!!!

<table>
<thead>
<tr>
<th>Time to Click</th>
<th>Revenue/user</th>
<th>Any Clicks on page</th>
</tr>
</thead>
<tbody>
<tr>
<td>50ms N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>200ms 500</td>
<td>N/A</td>
<td>-0.3%</td>
</tr>
<tr>
<td>500ms 1200</td>
<td>-1.2%</td>
<td>-1%</td>
</tr>
<tr>
<td>1000ms 1900</td>
<td>-2.8%</td>
<td>-1.9%</td>
</tr>
<tr>
<td>2000ms 3100</td>
<td>-4.3%</td>
<td>-4.4%</td>
</tr>
</tbody>
</table>

- Strong negative impacts
- Roughly linear changes with increasing delay
- Time to Click changed by roughly double the delay
## Search Traffic Impact

<table>
<thead>
<tr>
<th>Type of Delay</th>
<th>Delay (ms)</th>
<th>Experiment Duration (weeks)</th>
<th>Impact on Average Daily Searches Per User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-header</td>
<td>50</td>
<td>4</td>
<td>Not measurable</td>
</tr>
<tr>
<td><strong>Pre-header</strong></td>
<td><strong>100</strong></td>
<td>4</td>
<td>-0.20%</td>
</tr>
<tr>
<td>Post-header</td>
<td>200</td>
<td>6</td>
<td>-0.29%</td>
</tr>
<tr>
<td>Post-header</td>
<td>400</td>
<td>6</td>
<td>-0.59%</td>
</tr>
<tr>
<td>Post-ads</td>
<td>200</td>
<td>4</td>
<td>-0.30%</td>
</tr>
</tbody>
</table>

- Increase in abandonment heuristic = less satisfaction
  - Abandonment heuristic measures if a user stops interacting with search engine before they find what they are looking for
- Active users (users that search more often a priori) are more sensitive
Progressive Rendering Experiment

- **Goal**
  - Determine impact sending visual header before results.

- **Methodology**
  - Build page in phases
  - Send using HTTP 1.1 Chunked Transfer Encoding
Progressive Rendering Experiment: Results

- **Goal**
  - Determine impact sending visual header before results.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Faster across all latency percentiles&lt;br&gt;4-18% faster to download all HTML&lt;br&gt;Roughly halved time to see visible page change</td>
</tr>
<tr>
<td>Time to Click</td>
<td>~9% faster</td>
</tr>
<tr>
<td>Query refinement</td>
<td>+2.2%</td>
</tr>
<tr>
<td>Clicks overall</td>
<td>+0.7%</td>
</tr>
<tr>
<td>Pagination</td>
<td>+2.3%</td>
</tr>
</tbody>
</table>
Conclusion

- "Speed matters" is not just lip service
- Delays under half a second impact business metrics
- The cost of delay increases over time and persists
- Number of bytes in response is less important than what they are and when they are sent
Agenda

• Understanding Implication of page load times
• Challenges quantifying Page load times
• WPROF: Capturing Dependencies
• SPDY v HTTP1.1
Difficult to understand page load

<html>
  <script src="b.js"></script>
  <img src="c.png"/>
</html>

<html>
  <img src="c.png"/>
  <script src="b.js"></script>
  <script src="b.js"></script>
</html>
Difficult to understand page load

```html
<html>
  <img src="c.png"/>
  <script src="b.js"></script>
</html>
```

```html
<html>
  <script src="b.js"></script>
  <img src="c.png"/>
</html>
```
Difficult to understand page load

```html
<html>
  <img src="c.png"/>
  <script src="b.js"></script>
</html>

html
b.js<html>
  <script src="b.js"></script>
  <img src="c.png"/>
</html>
```

```html
<html>
  <script src="b.js"></script>
  <img src="c.png"/>
</html>
```
Difficult to understand page load

Understanding dependencies is the key to understand page load.
How a page is loaded
How a page is loaded

Concurrencies among the four components
How a page is loaded

Dependencies: one component can block others
A page load starts with a user-initiated request.
How a page is loaded

index.html
1  <html>
2       <script src="main.js"/>
3  </html>

Object Loader downloads the corresponding Web page.
How a page is loaded

index.html
1  <html>
2   <script src="main.js"/>
3  </html>

Upon receiving the first chunk of the root page, the HTML Parser starts to parse the page.
How a page is loaded

index.html
1  <html>
2  <script src="main.js"/>
3  </html>

HTML Parser requests embedded objects, i.e., JavaScript.
How a page is loaded

index.html
1  <html>
2  <script src="main.js"/>
3  </html>

main.js
...

Object Loader requests the inlined JS and sends it for evaluation.
How a page is loaded

```html
1  <html>
2  <script src="main.js"/>
3  </html>
```

main.js
...

JS evaluation can modify the DOM and its completion resumes HTML parsing.
How a page is loaded

`index.html`
1  `<html>`
2  `<script src="main.js"/>`
3  `</html>`

HTML continues being parsed and added to the DOM.
How a page is loaded

index.html

1  <html>
2   <script src="main.js"/>
3  </html>

Rendering Engine progressively renders the page (i.e., layout and painting).

http://www.example.com/
Agenda

- Understanding Implication of page load times
- Challenges quantifying Page load times
- WPROF: Capturing Dependencies
- SPDY vs HTTP1.1
Challenges in Capturing Dependencies

• Many factors that affect page load
  0 Page structure
  0 Inter-dependencies between network and computation activities
  0 Browser implementations

• Different browsers have different policies on how to parse and load webpages

• Only Way to Understand is to Model Dependencies
  – Reverse engineer page loads with test pages
Reverse engineer page loads with test pages

- Design test pages
What Makes for good Test pages?

- Design test pages
  - An object follows another

An example Web page

An example Web page
What Makes for good Test pages?

- Design test pages
  - An object follows another
  - An object embeds another

An example Web page
What Makes for good Test pages?

- Design test pages
- Observe timings from DevTools

Injecting arbitrary delays for an object should change things in a predictable manner:
- How will things change for embedded objects?
- How will things change for objects that follow each other?
What Makes for good Test pages?

- Design test pages
- Observe timings from DevTools
What Makes for good Test pages?

- Design test pages
- Observe timings from DevTools
Dependency policy categories

- Flow dependency
- Output dependency
- Lazy/Eager binding
  - Lazy = load when user gets there
  - Eager = preloading
- Resource constraints
  - Limited computing power or network resources (# TCP conn.)
Output dependency

```
index.html
1  <html>
2   <link rel="stylesheet" href="c.css">
3   <script src="f.js"/>
   ...
```
Output dependency

`index.html`
1 `<html>`
2 `  <link rel="stylesheet" href="c.css">`
3 `  <script src="f.js"/>
...`
**Output dependency**

```html
1  <html>
2      <link rel="stylesheet" href="c.css">
3      <script src="f.js"/>
... 
```

- **Network**
  - Object Loading
  - Cache, cookie, localStorage

- **UI**
  - HTML Parsing
  - Evaluation
  - DOM

- **Rendering**

*Elapsed time*

html

c.css
Output dependency

index.html
1  <html>
2   <link rel="stylesheet" href="c.css">
3   <script src="f.js"/>
4   ...

Elapsed time

html

c.css
**Output dependency**

```
index.html
1  <html>
2     <link rel="stylesheet" href="c.css">
3     <script src="f.js"/>
...  
```
Output dependency

```
index.html
1  <html>
2   <link rel="stylesheet" href="c.css"/>
3   <script src="f.js"/>
   ...
```
Output dependency

index.html
1  <html>
2   <link rel="stylesheet" href="c.css"/>
3  <script src="f.js"/>
...
Output dependency

index.html
1  <html>
2   <link rel="stylesheet" href="c.css"/>
3   <script src="f.js"/>
   ...

html
- Elapsed time
  - c.css
  - f.js

Network
- Object Loading
  - Cache, cookie, localStorage
- HTML Parsing
- Rendering
- DOM

UI
Agenda

• Understanding Implication of page load times

• Challenges quantifying Page load times

• WPROF: Capturing Dependencies

• SPDY v HTTP1.1
HTTP/1.1 problems
HTTP/1.1 problems

- Opens too many TCP connections
HTTP/1.1 problems

- Opens too many TCP connections
- Initiates object transfers strictly by the client
HTTP/1.1 problems

- Opens too many TCP connections
- Initiates object transfers strictly by the client
- Compresses only HTTP payloads, not headers
HTTP/1.1 problems

- Opens too many TCP connections
- Initiates object transfers strictly by the client
- Compresses only HTTP payloads, not headers

SPDY is proposed to address these issues
How well does SPDY perform?

SPDY helps 27% to 60%
How well does SPDY perform?

SPDY helps 27% to 60% according to Google.

SPDY sometimes helps and sometimes hurts. Overall, SPDY helps less than 10% according to Microsoft.

Measurement results conflict.
Goals

• A systematic study of SPDY that
  – Extensively sweeps the parameter space
  – Links SPDY performance to underlying factors
  – Identifies the dominant factors
Many factors external to SPDY affect SPDY.

**Approach:**
Isolate factors, sweep the parameter space.

- **Network parameters:**
  - RTT
  - Bandwidth

- **TCP settings:**
  - Loss rate
  - TCP initial congestion window

- **Web page effects:**
  - Synthetic objects
  - Real objects
  - Real pages

**Challenge 1**
Page load time has high variance

Variance: 0.5 second
Difference: 0.02 second

Approach to source of variability by:
- Experimenting in a controlled network
- Using our emulator instead of browsers
Dependencies between network and browser computation affect page loads.

No browser

Browser computation

Elapsed time

Object

Preserve dependencies.
Extensively sweep parameter space

<table>
<thead>
<tr>
<th>Factors</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTT</td>
<td>20ms, 100ms, 200ms</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>1Mbps, 10Mbps</td>
</tr>
<tr>
<td>Loss rate</td>
<td>0, .5%, 1%, 2%</td>
</tr>
<tr>
<td>TCP IW</td>
<td>3, 10, 21, 32</td>
</tr>
<tr>
<td>Web obj. size</td>
<td>100B, 1K, 10K, 100K, 1M</td>
</tr>
<tr>
<td># of objects</td>
<td>2, 8, 16, 32, 64, 128, 512</td>
</tr>
</tbody>
</table>
Link SPDY performance to factors

Decision tree analysis

Six factors, Thousands of data points
SPDY helps on small objects

Unlike in HTTP, a TCP segment can carry multiple Web objects in SPDY.

Why SPDY helps

Explanations
SPDY helps on large objects, low loss

In HTTP, Multiple connections compete with each other

→ More retransmissions
SPDY hurts on large objects, high loss

Most performance impact of SPDY comes from a single TCP connection.
Identify dominant factors

# obj

BW

RTT

obj

size

Importance?

loss

IW
Identify dominant factors

- obj size
- loss
- # obj

more important than

- RTT
- BW
- IW

# obj shows a trend

TCP IW

RTT: 200ms
BW: 10Mbps
Loss: 0
IW: 3
obj size: 10KB
# obj: 8

IW doesn’t show a trend
Synthetic objects → Real objects

**Network parameters**
- RTT: 20ms, 100ms, 200ms
- Bandwidth: 1Mbps, 10Mbps
- Loss rate: 0, 0.5%, 1%, 2%
- TCP IW: 3, 10, 21, 32
- Web obj. size: 100B, 1K, 10K, 100K, 1M
- # of objects: 2, 8, 16, 32, 64, 128, 512

**TCP settings**
- Web objects: Top 200 Alexa pages

**Make HTTP requests**
SPDY helps 60% in the median case because it largely reduces retransmissions.
Assumption that objects are fetched at the same time does not hold.
Epload captures browser effects

- Recorder: capture the dependency graph
- Replayer: make network requests while simulating the computation portions

Epload makes experiments reproducible
Real objects → Real pages

<table>
<thead>
<tr>
<th>Factors</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTT</td>
<td>20ms, 100ms, 200ms</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>1Mbps, 10Mbps</td>
</tr>
<tr>
<td>Loss rate</td>
<td>0, .5%, 1%, 2%</td>
</tr>
<tr>
<td>TCP IW</td>
<td>3, 10, 21, 32</td>
</tr>
<tr>
<td>Web obj. size</td>
<td>100B, 1K, 10K, 100K, 1M</td>
</tr>
</tbody>
</table>

# of objects

Top 200 Alexa pages

Emulate page loads with Epload
SPDY helps marginally because

- Computation and dependencies increase PLT of both SPDY and HTTP.

Dependencies and computation in real page loads reduce the impact of SPDY.
Enhanced policies for SPDY
Improving SPDY with server push

Mod_spdy’s: one level of HTML embedding
Our policy: one level of the dependency graph
Improving SPDY with server push

• Leverage information from dependency graphs
  – Web objects that are closer to the root should be pushed earlier
Improving SPDY with server push

- ‘Server push’ with our policy and mod_spdy’s both helps page load time by 10%~30%
- ‘Server push’ breaks dependency and sends objects before their time

Improving page load performance requires restructuring the page load process, e.g. server push.
Concluding Remarks