Demystifying Page Load Performance with WProf

Xiao (Sophia) Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall

University of Washington
Web is the critical part of the Internet
Page load is critical

- Amazon can increase 1% revenue by decreasing page load time by 0.1s.
Page load is critical but slow

- Amazon can increase 1% revenue by decreasing page load time by 0.1s.
- Page load is slow even on top 200 websites
Page load is critical but slow

- Amazon can increase 1% revenue by decreasing page load time by 0.1s.
- Page load is slow even on top 200 websites

Median page load time is 3 seconds.
Page load is critical but slow

- Amazon can increase 1% revenue by decreasing page load time by 0.1s.
- Page load is slow even on top 200 websites

A few top pages take more than 10 seconds to load.
Many techniques aim to optimize page load time

- **Optimization techniques**
  - Server placement: CDNs
  - Web pages and cache: mod_pagespeed, Silo
  - Application level: SPDY
  - TCP/DNS: TCP fast open, ASAP, DNS pre-resolution, TCP pre-connect

- **Problem**
  - Unclear whether they help or hurt page loads*

*http://www.stevesouders.com/
Many techniques aim to optimize page load time

- **Optimization techniques**
  - Server placement: CDNs
  - Web pages and cache: mod_pagespeed, Silo
  - Application level: SPDY
  - TCP/DNS: TCP fast open, ASAP, DNS pre-resolution, TCP pre-connect

- **Problem**
  - Unclear whether they help or hurt page loads*

*http://www.stevesouders.com/

Page load process is poorly understood.
Difficult to understand page load

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

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

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

Understanding dependencies is the key to understand page load.
Overview of our work

- Model the page load process
- Build the WProf tool
- Study page load on real pages
Overview of our work

- Model the page load process
  - How a page is loaded?
  - How to infer dependencies?
- Build the WProf tool
- Study page load on real pages
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
How a page is loaded

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.
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

`index.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

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

Rendering Engine progressively renders the page (i.e., layout and painting).
How to infer dependencies

● **Goal**
  ○ Extract as many dependencies as possible across browsers

● **Methodology**
  ○ Design test pages
  ○ Examine documents
  ○ Inspect browser code
How to infer dependencies

● Goal
  ○ Extract as many dependencies as possible across browsers

● Methodology
  ○ Design test pages
  ○ Examine documents
  ○ Inspect browser code
Reverse engineer page loads with test pages

- Design test pages
Reverse engineer page loads with test pages

- Design test pages

An example Web page
Reverse engineer page loads with test pages

- Design test pages
  - An object follows another

An example Web page
Reverse engineer page loads with test pages

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

An example Web page
Reverse engineer page loads with test pages

- Design test pages
- Observe timings from DevTools
Reverse engineer page loads with test pages

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

● Flow dependency
  ○ Natural order that activities occur
Dependency policy categories

- Flow dependency
- Output dependency
  - Correctness of execution when multiple processes access to the same resource
Dependency policy categories

- Flow dependency
- Output dependency
- Lazy/Eager binding
  - Tradeoffs between data downloads and page load latencies
Dependency policy categories

- Flow dependency
- Output dependency
- Lazy/Eager binding
- 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"/>
    ...
```

Elapsed time

Network

Object Loading

HTML Parsing

Evaluation

DOM

Rendering

UI

Cache, cookie, localStorage
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

HTML Parsing

Evaluation

DOM

UI

Cache, cookie, localStorage

Elapsed time

html

c.css
Output dependency

```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"/>

...
# Dependency policies

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Flow</strong></td>
<td>F1</td>
<td>Loading an object $\rightarrow$ Parsing the tag that references the object</td>
</tr>
<tr>
<td></td>
<td>F2</td>
<td>Evaluating an object $\rightarrow$ Loading the object</td>
</tr>
<tr>
<td></td>
<td>F3</td>
<td>Parsing the HTML page $\rightarrow$ Loading the first block of the HTML page*</td>
</tr>
<tr>
<td></td>
<td>F4</td>
<td>Rendering the DOM tree $\rightarrow$ Updating the DOM</td>
</tr>
<tr>
<td></td>
<td>F5</td>
<td>Loading an object referenced by a JavaScript or CSS $\rightarrow$ Evaluating the JavaScript or CSS*</td>
</tr>
<tr>
<td></td>
<td>F6</td>
<td>Downloading/ Evaluating an object $\rightarrow$ Listener triggers or timers</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>O1</td>
<td>Parsing the next tag $\rightarrow$ Completion of a previous JavaScript download and evaluation</td>
</tr>
<tr>
<td></td>
<td>O2</td>
<td>JavaScript evaluation $\rightarrow$ Completion of a previous CSS evaluation</td>
</tr>
<tr>
<td></td>
<td>O3</td>
<td>Parsing the next tag $\rightarrow$ Completion of a previous CSS download and evaluation</td>
</tr>
<tr>
<td><strong>Lazy/Eager binding</strong></td>
<td>B1</td>
<td>[Lazy] Loading an image appeared in a CSS $\rightarrow$ Parsing the tag decorated by the image</td>
</tr>
<tr>
<td></td>
<td>B2</td>
<td>[Lazy] Loading an image appeared in a CSS $\rightarrow$ Evaluation of any CSS that appears in front of the tag decorated by the image</td>
</tr>
<tr>
<td></td>
<td>B3</td>
<td>[Eager] Preloading embedded objects does not depend on the status of HTML parsing. (breaks F1)</td>
</tr>
<tr>
<td><strong>Resource constraint</strong></td>
<td>R1</td>
<td>Number of objects fetched from different servers $\rightarrow$ Number of TCP connections allowed per domain</td>
</tr>
<tr>
<td></td>
<td>R2</td>
<td>Browsers may execute key computational activities on the same thread, creating dependencies among the activities. This dependency is determined by the scheduling policy.</td>
</tr>
</tbody>
</table>

* An activity depends on *partial* completion of another activity.
## Dependency policies

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow</td>
<td>F1</td>
<td>Loading an object → Parsing the tag that references the object</td>
</tr>
<tr>
<td></td>
<td>F2</td>
<td>Evaluating an object → Loading the object</td>
</tr>
<tr>
<td></td>
<td>F3</td>
<td>Parsing the HTML page → Loading the first block of the HTML page*</td>
</tr>
<tr>
<td></td>
<td>F4</td>
<td>Rendering the DOM tree → Updating the DOM</td>
</tr>
<tr>
<td>Output</td>
<td>F5</td>
<td>Loading an object referenced by a JavaScript or CSS → Evaluating the JavaScript or CSS*</td>
</tr>
<tr>
<td></td>
<td>F6</td>
<td>Downloading/Evaluating an object → Listener triggers or timers</td>
</tr>
<tr>
<td></td>
<td>O1</td>
<td>Parsing the next tag → Completion of a previous JavaScript download and evaluation</td>
</tr>
<tr>
<td></td>
<td>O2</td>
<td>JavaScript evaluation → Completion of a previous CSS evaluation</td>
</tr>
<tr>
<td>Lazy/Eager binding</td>
<td>B1</td>
<td>[Lazy] Loading an image appeared in a CSS → Parsing the tag decorated by the image</td>
</tr>
<tr>
<td></td>
<td>B2</td>
<td>[Lazy] Loading an image appeared in a CSS → Evaluation of any CSS that appears in front of the tag decorated by the image</td>
</tr>
<tr>
<td></td>
<td>B3</td>
<td>[Eager] Preloading embedded objects does not depend on the status of HTML parsing. (breaks F1)</td>
</tr>
<tr>
<td>Resource constraint</td>
<td>R1</td>
<td>Number of objects fetched from different servers → Number of TCP connections allowed per domain</td>
</tr>
<tr>
<td></td>
<td>R2</td>
<td>Browsers may execute key computational activities on the same thread, creating dependencies among the activities. This dependency is determined by the scheduling policy.</td>
</tr>
</tbody>
</table>

* An activity depends on partial completion of another activity.
## Dependency policies across browsers

<table>
<thead>
<tr>
<th>Dependency</th>
<th>IE</th>
<th>Firefox</th>
<th>WebKit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>all</td>
<td>no O3</td>
<td>no O3</td>
</tr>
<tr>
<td>Late binding</td>
<td>all</td>
<td>all</td>
<td>all</td>
</tr>
<tr>
<td>Eager binding</td>
<td>Preloads img, js, css</td>
<td>Preloads img, js, css</td>
<td>Preloads css, js</td>
</tr>
<tr>
<td>Net resource</td>
<td>6 conn.</td>
<td>6 conn.</td>
<td>6 conn.</td>
</tr>
</tbody>
</table>
## Dependency policies across browsers

<table>
<thead>
<tr>
<th>Dependency</th>
<th>IE</th>
<th>Firefox</th>
<th>WebKit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>all</td>
<td>no O3</td>
<td>no O3</td>
</tr>
<tr>
<td>Late binding</td>
<td>all</td>
<td>all</td>
<td>all</td>
</tr>
<tr>
<td>Eager binding</td>
<td>Preloads img, js, css</td>
<td>Preloads img, js, css</td>
<td>Preloads css, js</td>
</tr>
<tr>
<td>Net resource</td>
<td>6 conn.</td>
<td>6 conn.</td>
<td>6 conn.</td>
</tr>
</tbody>
</table>

O3: CSS downloads and evaluation block HTML parsing.
Overview of our work

● Model the page load process
● Build the WProf tool
  ○ Profiling in browsers
  ○ Generating dependency graphs
  ○ Analyzing critical paths
● Study page load on real pages
WProf architecture

Browser Stack

- Web page instances
- Browser extension/plug-in framework
- Native browser
WProf architecture

Browser Stack

- Web page instances
- Browser extension/plug-in framework
- Native browser

WProf profiler

| Object Loader | HTML Parser | CSS Engine | JavaScript Engine | Rendering Engine |
WProf architecture

Browser Stack

Web page instances

Browser extension/plug-in framework

Native browser

WProf profiler

Object Loader
HTML Parser
CSS Engine
JavaScript Engine
Rendering Engine

Activity timing

Dependencies
WProf architecture

Browser Stack

- Web page instances
- Browser extension/plug-in framework
- Native browser
  - WProf profiler
  - Object Loader
  - HTML Parser
  - CSS Engine
  - JavaScript Engine
  - Rendering Engine

- Log activity timings
- Track dependencies by using HTML tags under parsing when an activity occurs
WProf architecture

Browser Stack

- Web page instances
- Browser extension/plug-in framework
- Native browser

WProf profiler

- Object Loader
- HTML Parser
- CSS Engine
- JavaScript Engine
- Rendering Engine

Lightweight
Our evaluation suggests negligible performance overhead.
WProf architecture

Browser Stack

Web page instances

Browser extension/plug-in framework

Native browser

WProf profiler

Object Loader
HTML Parser
CSS Engine
JavaScript Engine
Rendering Engine

Implementation
- Built on WebKit
- Extended in Chrome and Safari
- Written in C++

Activity timing
Dependences
WProf architecture

Browser Stack

Web page instances

Browser extension/plug-in framework

Native browser

WProf profiler

- Object Loader
- HTML Parser
- CSS Engine
- JavaScript Engine
- Rendering Engine

Critical paths

Dependency graphs

Activity timing

Dependencies
WProf architecture

Browser Stack

Web page instances

Browser extension/plug-in framework

Native browser

WProf profiler

- Object Loader
- HTML Parser
- CSS Engine
- JavaScript Engine
- Rendering Engine

Critical paths

Dependency graphs

Activity timing

Dependencies
Dependency graph

```html
<html>
  <head>
    <link rel="stylesheet" src="./main.css">
    <script src="./main.js" />
  </head>
  <!--request a JS-->
  <body onload="...">
    <img src="test.png" />
  </body>
</html>
```
Critical path analysis

Critical path: the longest bottleneck path.
Critical path analysis

Critical path: the longest bottleneck path.
Critical path analysis

Critical path: the longest bottleneck path.
Critical path analysis

Critical path: the longest bottleneck path.
Critical path analysis

Critical path: the longest bottleneck path.
Critical path analysis

Critical path: the longest bottleneck path.
Critical path analysis

Critical path: the longest bottleneck path.
Critical path analysis

Critical path: the longest bottleneck path.
Critical path analysis

Critical path: the longest bottleneck path.
Critical path analysis

Critical path: the longest bottleneck path.

Improving activities off the critical path doesn't help page load.
Overview of our work

● Model the page load process
● Build the WProf tool
● Study page load with real pages
Experimental setup

- **Location**
  - UW Seattle campus network
- **Browser**
  - WProf-instrumented Chrome
- **Web pages**
  - 150 out of top 200 Alexa pages
- **Page load time**
  - Minimum out of 5 repeats
How much does computation contribute to page load time?
Computation is significant

Network/Computation as a fraction of page load time
Computation is significant

Computation is $\sim 35\%$ of page load time (median) on the critical path.
How much does caching help page load performance?
How much does caching help?

- Caching eliminates 80% Web object loads
- It doesn't reduce page load time as much
How much does caching help?

- Caching eliminates 80% Web object loads
- It doesn't reduce page load time as much
- Caching only eliminates 40% Web object loads on the critical path
Summary of other results

- Most object downloads are not critical
- JS blocks parsing on 60% top pages
- SPDY doesn't help much as expected
- Minification with mod_pagespeed doesn't reduce received bytes on the critical path
Related work

● Industry tools
  ○ DevTools, Pagespeed Insights

● Academic
  ○ WebProphet [NSDI'2010]
    ■ Only consider network time
Conclusion

- Model page load process
- WProf automatically extracts dependencies and analyzes critical paths
- WProf can be used to
  - Understand performance of any page load
  - Explain behaviors of current optimizations
  - Perform what-if analysis

Project website: wprof.cs.washington.edu