Ajax Performance Analysis
Ryan Breen
▪ **Who**
  - Ryan Breen: VP Technology at Gomez and blogger at ajaxperformance.com

▪ **Goals**
  - Survey tools available to developers
  - Understand how to approach performance optimization
  - Real world examples of how to improve the end user experience (with exciting data!)
Understand the tools available

- **Network visualization**
  - Firebug
  - WebKit Web Inspector
  - IBM Page Detailer

- **Client side profiling**
  - Firebug and Firebug Lite
  - Dojo.Profile
Network Visualization

- **Firebug**
  - Network visualization since version 1.0
  - Useful information, but not overwhelming in detail
  - A great addition to a tool you are already (or should be) using
- **Web Inspector**
  - Currently available in WebKit nightlies
  - Beautiful presentation
  - Part of a comprehensive toolkit for Safari, a worthy companion to Firebug
IBM Page Detailer
- Basic version is free
- OS-level network profiling, so supports any browser on Windows
- Very focused, unitasker
- Much more granular network analysis
- Really, really nerdy
- **Firebug**
  - JS function call profiler
  - Firefox only
Client side profiling

- **Pure JavaScript solutions**
  - Dojo.Profile
  - Firebug Lite

- **More limited functionality, but available cross browser**
How to approach performance

- Avoid Premature Optimization
- Understand your users
  - Connection profile
  - Usage model: how do they interact with your application?
- Measure, then optimize along two axes
  1. Reduce overall latency
  2. Hide the remaining latency from user perception
- Define performance targets
The goal of performance optimization is to hide latency from the end user

In high speed networks, latency dominates

- The speed of establishing connections and issuing requests has not scaled in pace with the speed of transmitting data
- Round trip times are slow, transfer completion times are fast
How do we reduce latency?

- Fewer requests means better performance
  - No matter the connection profile, there is no better way to improve performance
  - Every other trick is a way to hide the cost of downloading objects
- So, pack more information into each request
- Reuse connections
- Increase connection parallelism
- Minimize bandwidth used where possible
- Respect caching, but don’t put too much faith in it
Packing more into a request

- **Image concatenation**
  - Replace multiple requests for small images with one request
  - Use CSS background-position to select
  - Further optimization: use CSS transparency to represent disabled images
Packing more into a request

- **JS and CSS bundling**
  - JavaScript and CSS are frequently in HEAD
  - JavaScript is downloaded and executed serially, exacerbating the latency experienced by the end user
  - Combine scripts and CSS into one file to avoid round trip costs
  - Many frameworks or web toolkits now do this on the fly

<table>
<thead>
<tr>
<th>Data Summary</th>
<th>Avg Response Time (sec)</th>
<th>Availability (%)</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musicstore combined</td>
<td>4.174</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>Musicstore default</td>
<td>4.986</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>
Connection persistence is a huge win

- Creating each connection adds latency
- Without connection persistence, that latency hit is incurred for each object on the page
- Caution: Persistence is frequently broken due to firewall configuration issues or other operational mishaps
Increase connection parallelism

- **Most browsers allow 2 connections per host**
  - But that host check is a simple string match against hostname
  - We can trick the browser into send 6 requests at a time with a DNS wildcard
  - Potential savings of 40%
  - This trick is used by every RIA map application
Optimizing against Latency

- **Latency is a focus, but also reduce bandwidth**
  - HTTP Compression for text artifacts
  - JS obfuscation – Potential side benefit of reducing readability

- **Don’t overlook the simple things**
  - When redirecting to a directory, remember the trailing slash
Caching effectively reduces requests on subsequent visits
  - So avoid breaking it with performance tweaks

But you can’t trust caching to hide performance crimes
  - YUI Blog numbers: 40-60% of users and 20% of page hits had empty cache
  - All new users have an empty cache, so avoid making a bad first impression
Where possible, avoid freshness checks

- Getting that 304 is still a round trip
- The latency for the request is often indistinguishable from a 200
- Freshness typically involves a tradeoff between useless refreshes and stale data
- **An elegant solution from Google GWT**
  - The file name for ImageBundles includes an MD5 sum of the contents, so cache headers can be set for infinite cacheability
  - Users are guaranteed to have the newest copy

- **Solution could easily be extended to JS / CSS bundles**
Perceived Performance

- Despite our best efforts, some latency remains
- How can we hide it?
  - Understand the role of the user’s perception
  - Defer anything that isn’t critical to that perception
Postponing non-critical latency

- **Avoid the defer attribute**
  - Only supported in IE
  - Only defers execution, not network overhead

- **Alternative mechanisms**
  - DOM scripting
  - Dojo’s XHR approach
Postponing events

- **onload vs. ondomready**
  - ondomready lets us schedule events sooner for more timely behavioral or presentational modifications
  - But ondomready may occur before the document is interactive

- **Is there room for a 3rd?**
  - How about onperceivedrender
  - How would that be calculated
Questions?

- **Contact Information**
  
  rbreen@gomez.com
  
  http://ajaxperformance.com