
Google™
09
I/O

Designing OpenSocial Apps for Speed and Scale

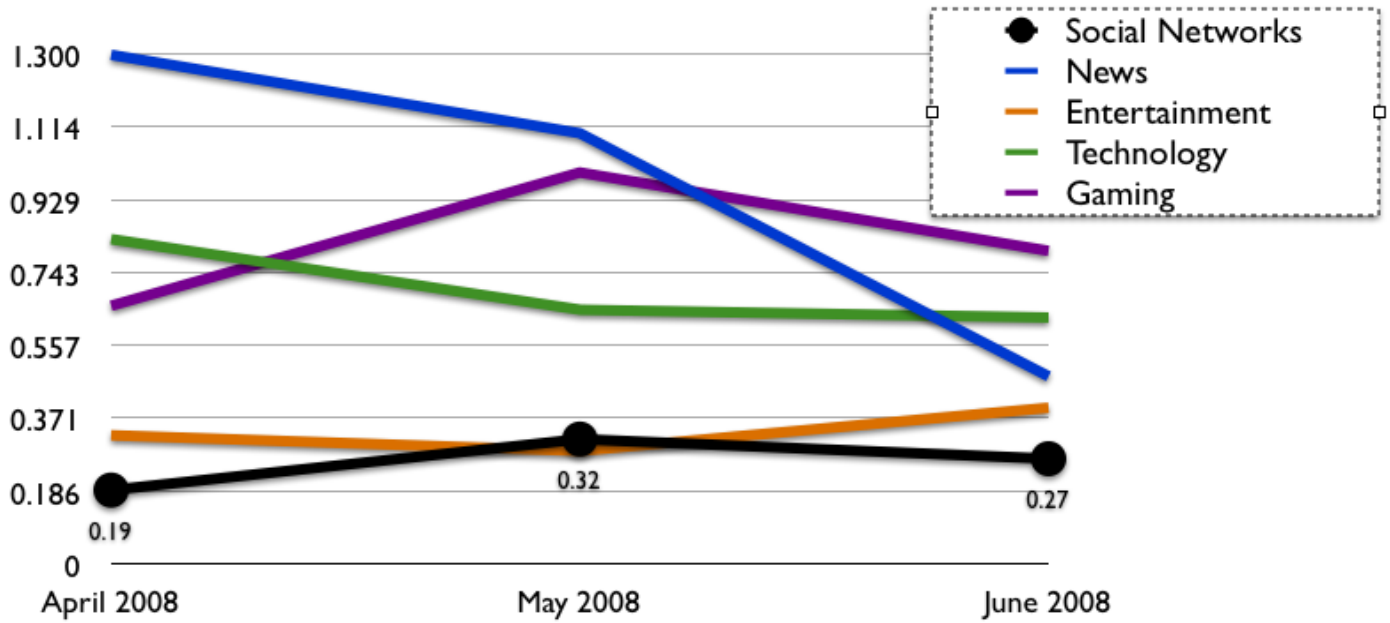
Arne Roomann-Kurrik & Chris Chabot
5/27/2009

Post your questions for this talk on Google Moderator:
http://code.google.com/events/io/questions
Direct link:
http://bit.ly/opensocialspeedscale-questions

Google 09 I/O

Success In A Social Market

- Social application eCPM is lower than average



- Direct access to millions of signed in users
  - Growth functionality built into social platforms
- Small margins x Lots of users = **Small Tweaks Matter**

Google 09

Chart from Pubmatic AdPrice Index 2008: http://pubmatic.com/adpriceindex/index.html

"Improving our latency is really, really important"

+0.5 seconds costs Google 20% search traffic

Marissa Mayer, Google

http://bit.ly/idItc ~12:50

"Even very small delays would result in substantial and costly drops in revenue"

+0.1 seconds costs Amazon 1% of sales

Greg Linden, Amazon

http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html

http://home.blarg.net/~glinden/StanfordDataMining.2006-11-29.ppt

"If you make a product faster, you get that back in terms of increased usage"

-30KB gave Google Maps 30% growth in 3 weeks

Marissa Mayer, Google

http://www.youtube.com/watch?v=6x0cAzQ7PVs ~18:10

This Presentation

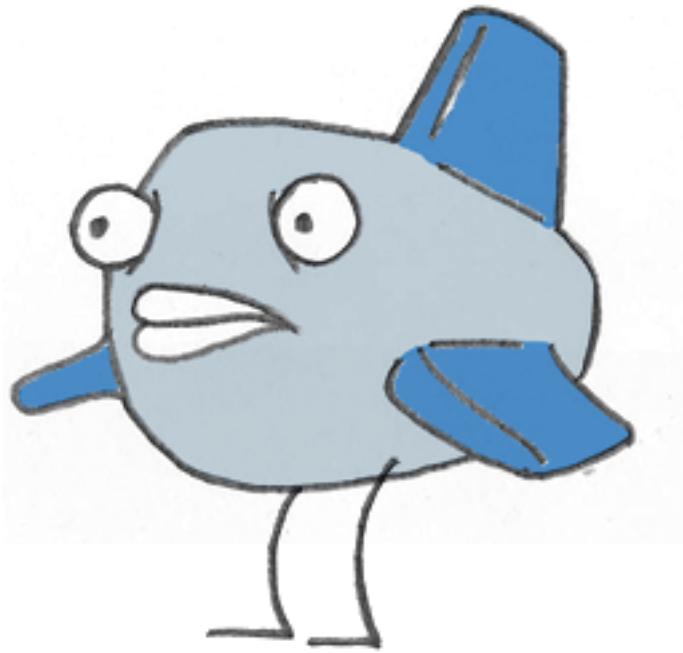
- Measure impact of changes on large scale apps
- Sample OpenSocial application
  - Different strategies
  - Real performance numbers
- Goals
  - Deliver a fast user experience
  - Minimize costs of running a large social app
  - Highlight new OpenSocial features

Introducing Quartermile

Photo by Phil McElhinney http://www.flickr.com/photos/philmcelhinney/1000986005/


quarter mile

Building Quartermile



- Backend:
  - Built on Google App Engine
  - Remote Procedure Calls
  - JSON
- Frontend:
  - Gadgets on many containers
  - JavaScript library for RPC calls
- View the source: http://bit.ly/quartermile-src
- XML spec: http://bit.ly/quartermile-app

Measuring Quartermile

- Request types
  - Quartermile API
  - OpenSocial API
  - Assets (images, JS, CSS)
- Metrics
  - Bandwidth / PV (KB)
  - Requests / PV (#)
  - Latency / PV (ms)
  - CPU time (megacycles)
- Measuring each individually is important
- Have more control over some than others

Quartermile 'Naive' Implementation Metrics

	Bandwidth / PV(kb)	Requests / PV	Latency (ms)	CPU time (megacycles)
Quartermile API calls	4.21	4	6490	2078
Social requests	1.45	2	804	0
Assets	135.6	22	3152	0

Quartermile 'Naive' Implementation Costs

SPEED	NAIVE
LATENCY (Gadget)	2730 MS
LATENCY (Page)	3536 MS
REQUESTS	26
YSlow! SCORE	72

COST	NUMBER	COST	TOTAL
BANDWIDTH	5582 GB	$0.12	$669.84
CPU TIME	1859 HR	$0.10	$185.90
$ / MONTH (11 QPS)			$855.74

Web Development Best Practices*

* Gadgets are web pages too!

Photo by rudolf_schuba http://www.flickr.com/photos/rudolf_schuba/153225000/

Web Development Best Practices

Gadgets are web pages too!

- We're interested in the following metrics:
  - Application size
  - Request count
  - Latency of initial load
- Measuring
  - Safari: Web Inspector
  - Firefox: Firebug & YSlow
  - IE: HttpWatch

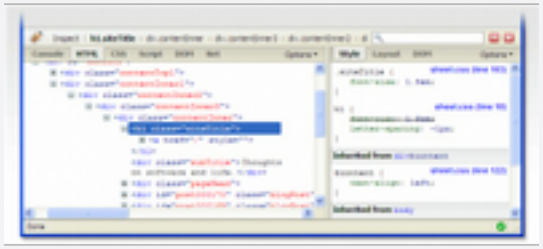


Photo by theeerin http://www.flickr.com/photos/theeerin/3306640432/

Web Development Best Practices

Minify JavaScript & CSS

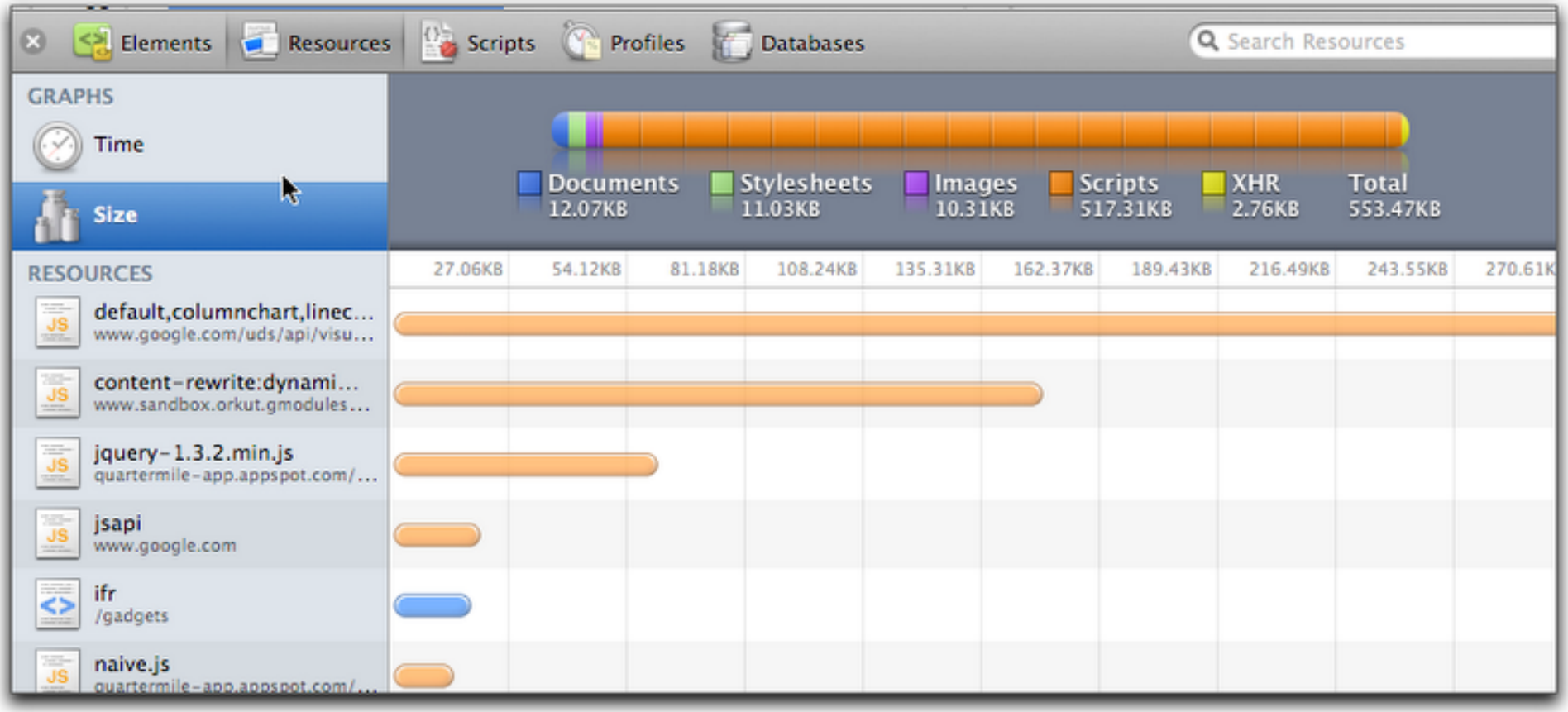
- Concatenate JavaScript & CSS files
  - Reduces latency
  - Reduces HTTP Requests
- Compress JavaScript and CSS
  - Reduces download size

```
var foo = ['1','2','3','4'];
for (var i = 0 ; i < 4 ; i++) {
   alert(foo[i]);
}
```

```
var foo=["1","2","3","4"];for(var i=0;i<4;i++){alert(foo[i])}
```

Web Development Best Practices

Determining application size



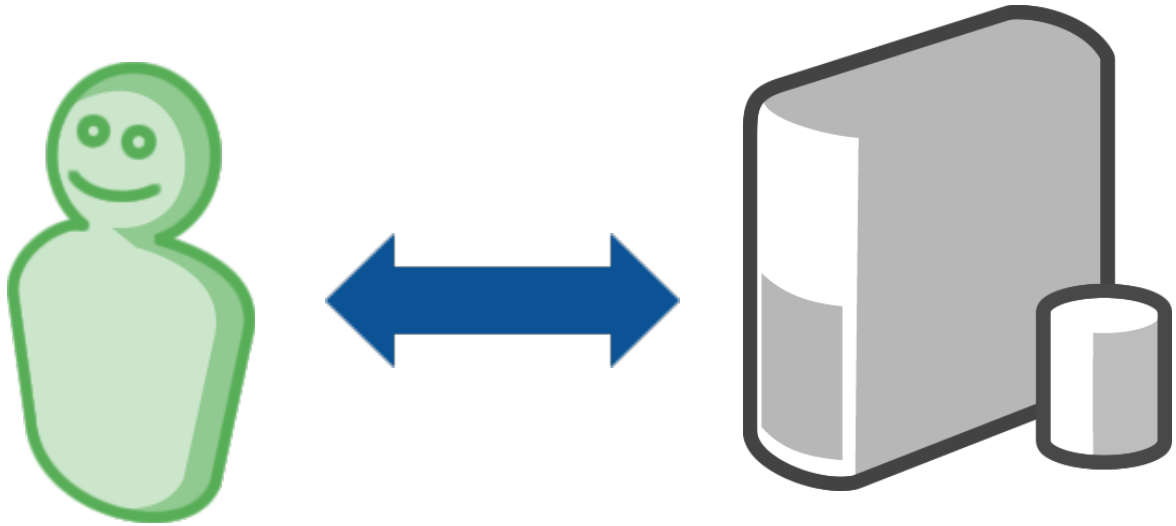
Quartermile Improvement

After JavaScript + CSS Minification (YUI Compressor)

	NAIVE	AFTER MINIFICATION	SAVINGS
SIZE / REQUEST	33 KB	15 KB	-54%
SIZE / MONTH	930 GB	420 GB	
$	$111.60	$50.40	-$61.20

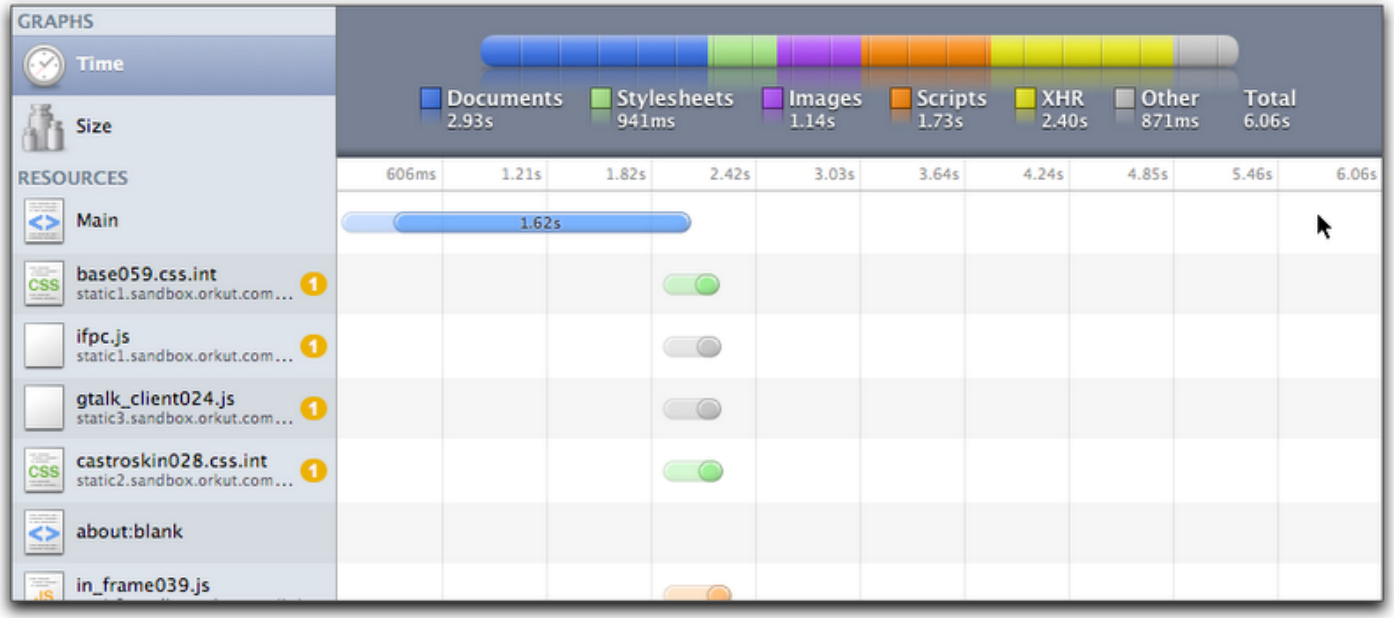
Web Development Best Practices

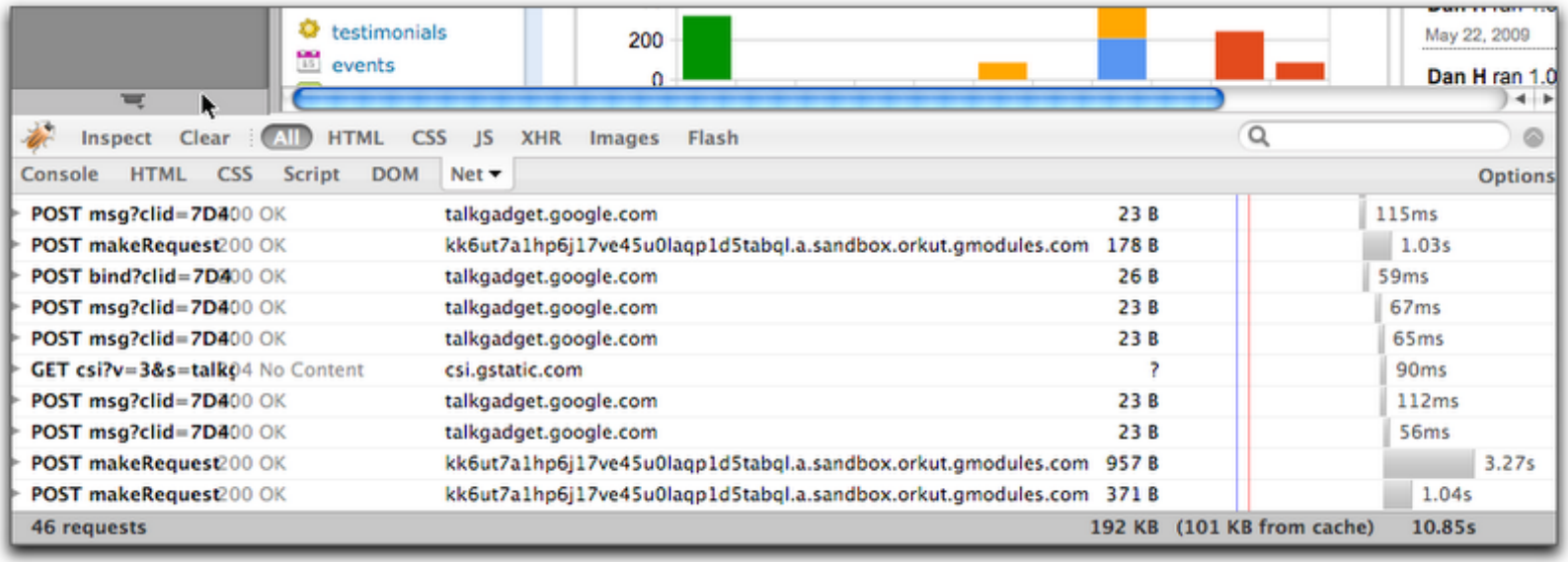
Measuring Latency



Web Development Best Practices

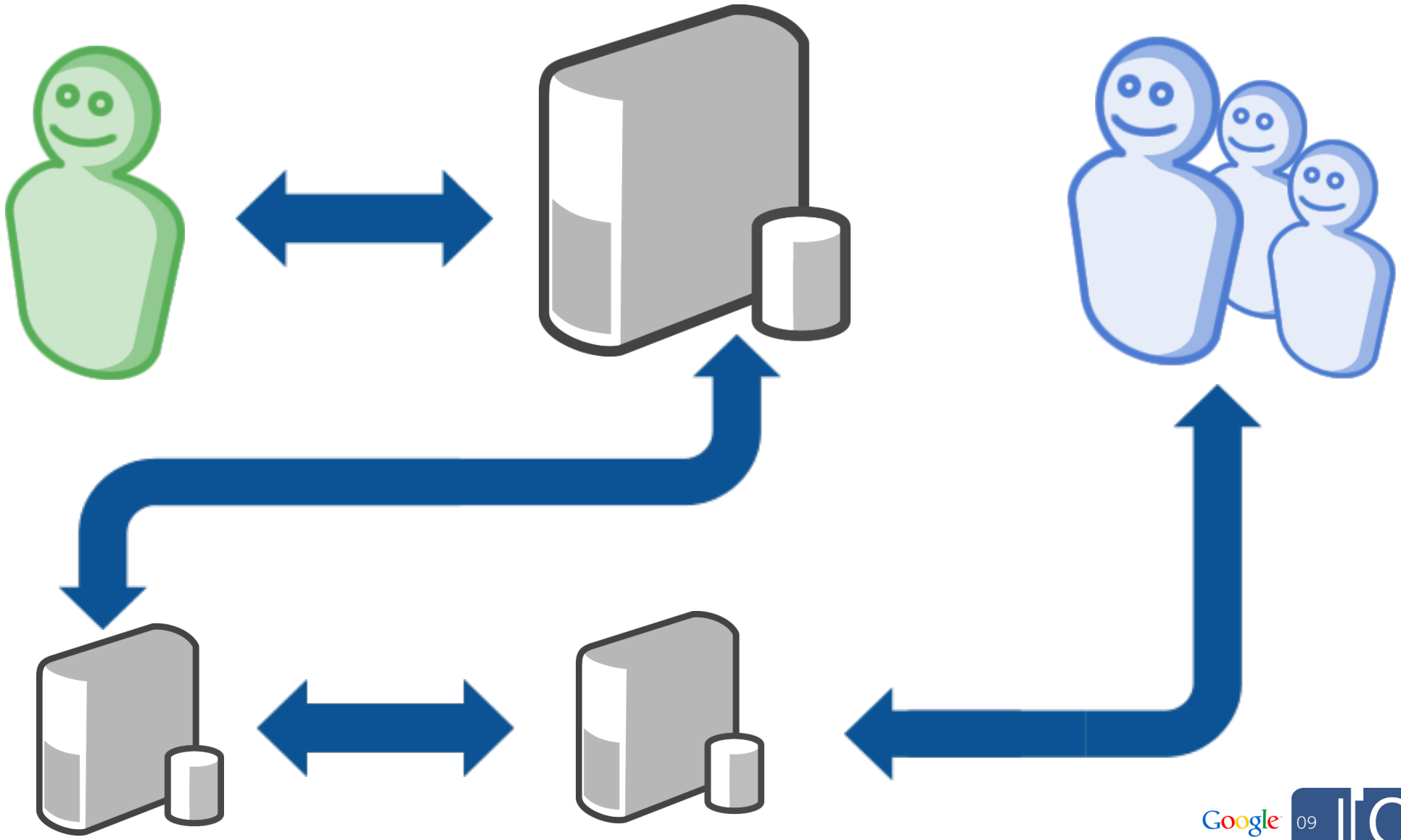
Measuring latency





Web Development Best Practices

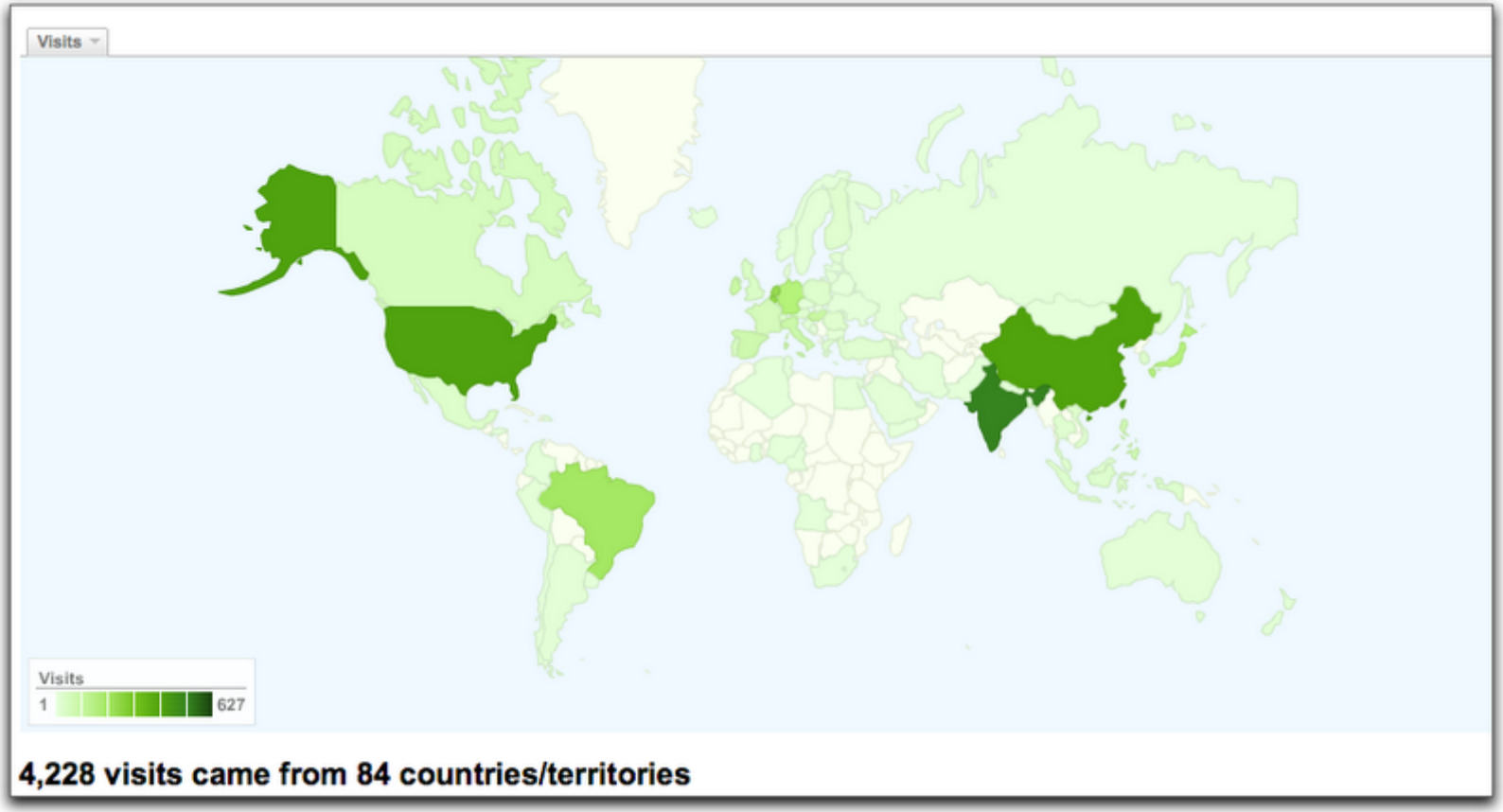
Latency is not a static number



Web Development Best Practices

Decreasing latency increases user happiness and engagement

- Measure from different locations
- Measure often



Web Development Best Practices

Automatically collecting latency measurements

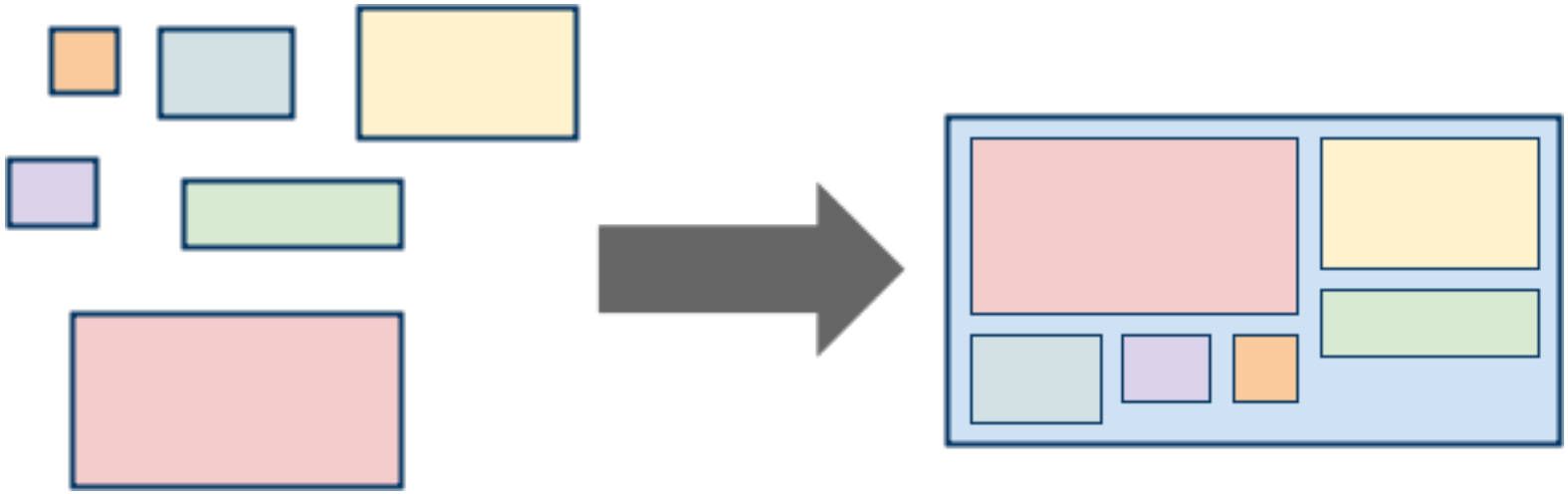
- JavaScript code for recording gadget latency:

```
var startTime = new Date();
var imgElement = new Image()
imgElement.onload = function() {
  var endTime = new Date();
  var latency = endTime. getTime() - startTime.getTime();
  // report latency back to your server
}
imgElement.src = "http://your.server.com/ping.gif";
```

Web Development Best Practices

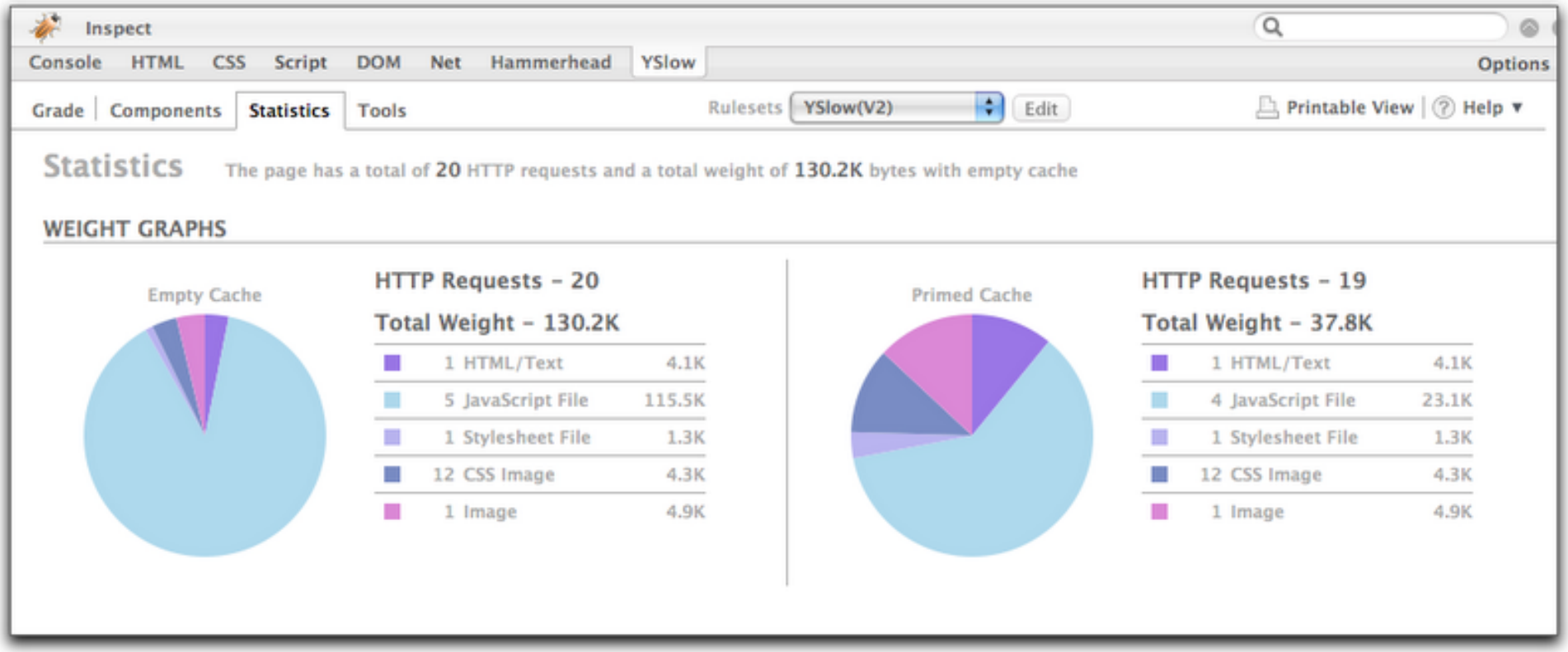
Spriting Images

- Concatenate images into a single file (sprite) and use CSS to selectively display portions of the sprite
  - Reduce # of requests
  - Reduce latency



Web Development Best Practices

Determining Request Count With YSlow!



Quartermile Improvement

After Image Spriting

	NAIVE	AFTER SPRITING	AFTER PALETTE FIX	SAVINGS
LATENCY (Gadget)	592 MS	378 MS	325 MS	-45%
SIZE	9.59 KB	11.15 KB	5.82 KB	-39%
REQUESTS	15	1	1	-93%
$	$27.30	$38.28	$16.80	-$10.5

Web Development Best Practices

Adjusting Cache Headers

- Use the browser's cache for static data
  - Reduces total bandwidth & requests
  - Reduces latency
- Apache configuration example:

```
<FilesMatch "\.(css|js|gif|jpe?g|png)$">
  Header set Cache-Control "max-age=290304000, public"
</FilesMatch>
```

- Use Cache-busting to force refresh

```
http://example.org/css/style.css?v=3
```

Server Assisted Optimizations

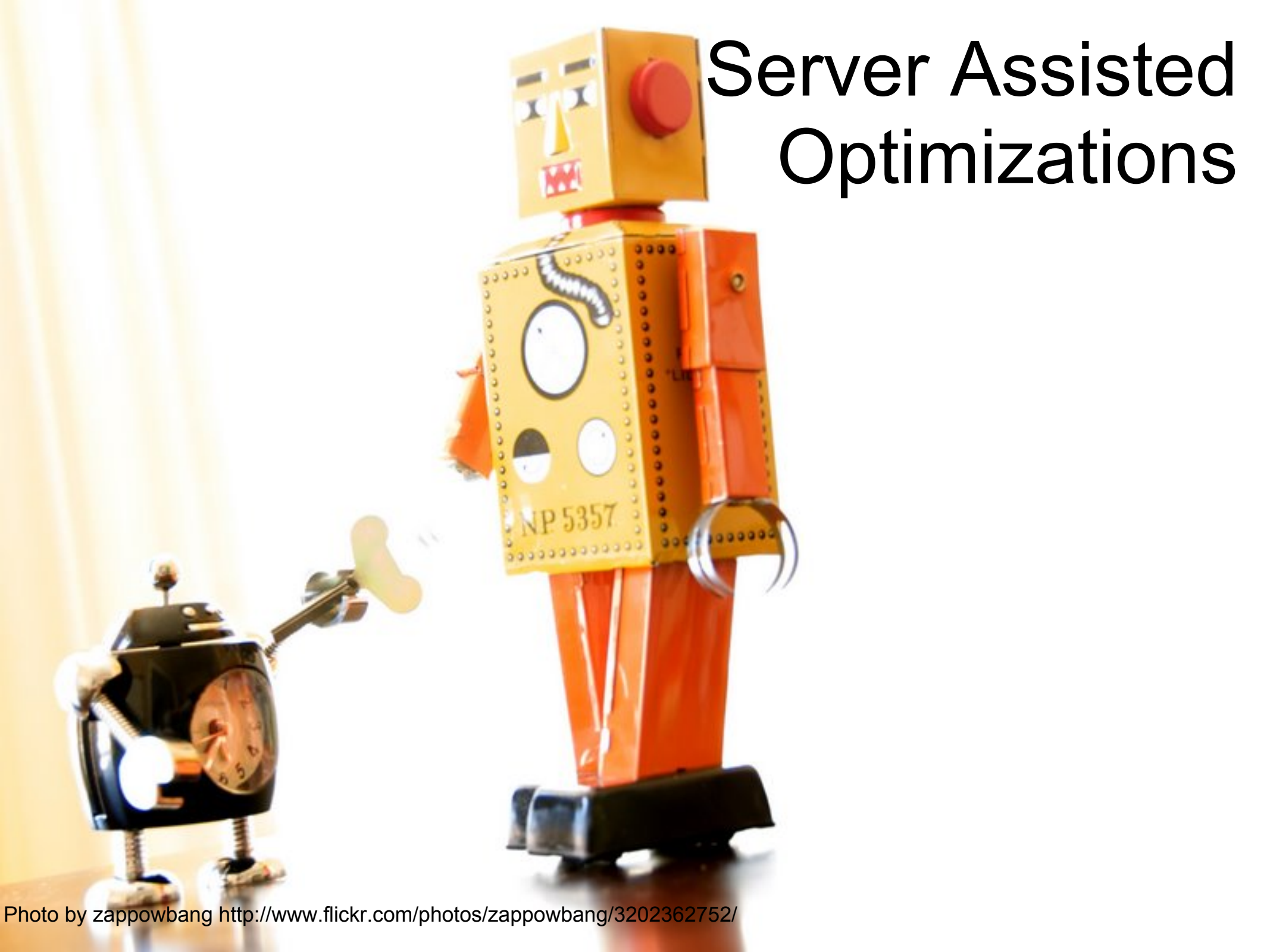


Photo by zappowbang http://www.flickr.com/photos/zappowbang/3202362752/

Server Assisted Optimizations

- Social gadget:
  - Small tweak == big gain
- Social network
  - Many small tweaks == very big gain

- Social network advantages:
  - Control over the HTML they output
  - Better network infrastructure

Static Content Proxies

- Social network willing to absorb some traffic for you
  - Could use content delivery networks (CDNs)
  - Distributed network, great for serving static content all over the world
- Important for clients further away from your servers!

```
var div = $("#flashcontainer");
var url = gadgets.io.getProxyUrl(
          "http://me.com/flash.swf");
gadgets.flash.embedFlash(url, div, 10);
```

Content Rewriting

- Social network has control over its own output
  - CSS first, JS last
  - Concatenate, Minify
  - Rewrite URLs for static proxy
- Caching controls in your gadget spec:

```
<Module>
  <ModulePrefs>
    <Optional feature="content-rewrite">
      <Param name="include-urls"></Param>
      <Param name="exclude-urls">.*</Param>
      <Param name="include-tags"></Param>
    </Optional>
  ...
```

OpenSocial Best Practices

Image by Paul Downey http://www.flickr.com/photos/psd/2841928867/in/datetaken

OpenSocial: Designed For Social Apps

- Some optimizations only make sense in a social application context
- OpenSocial offers conveniences to social app developers

- Learn from the OpenSocial API when designing your own application interfaces

Batching

- One API call == 1 HTTP request:

```
osapi.people.getViewer().execute(onVwr);
osapi.people.getOwner().execute(onOwnr);
osapi.people.getViewerFriends().execute(onFrnd);
osapi.people.getOwnerFriends().execute(onOFrnd);
```

- Do as much as you can in a single trip:

```
var batch = osapi.newBatch()
  .add("vwr", osapi.people.getViewer())
  .add("vfd", osapi.people.getViewerFriends())
  .add("owr", osapi.people.getOwner())
  .add("ofd", osapi.people.getOwnerFriends())
  .execute(onData);
```

- **2** -> **1** OpenSocial requests
- **4** -> **1** Quartermile API requests

Quartermile Improvement

After Request Batching

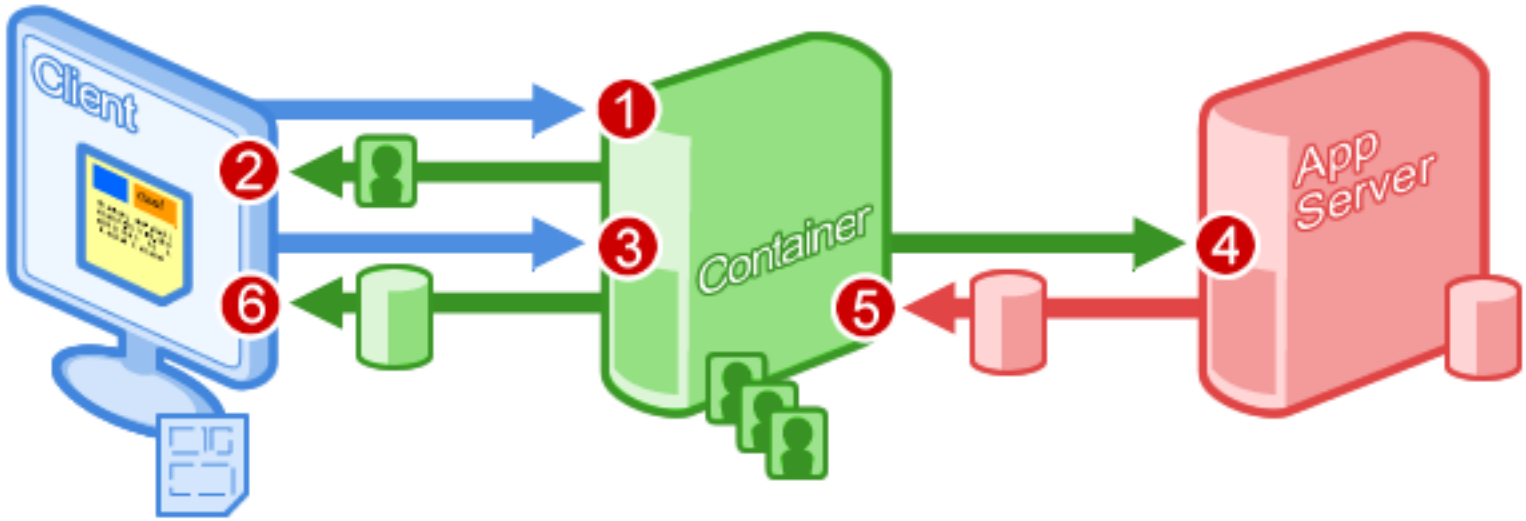
SPEED	NAIVE	BATCHED	DELTA
LATENCY (Gadget)	2730 MS	2743 MS	+13 MS
LATENCY (Page)	3536 MS	3504 MS	-32 MS
REQUESTS	26	24	-2
YSlow! SCORE	72	74	+2

COST	NUMBER	COST	TOTAL	CHANGE
BANDWIDTH	5576 GB	$0.12	$669.12	
CPU TIME	1499 HR	$0.10	$149.90	
$ / MONTH (11 QPS)			$819.02	-$36.72

OpenSocial Best Practices

Data Pipelining + Proxied Content

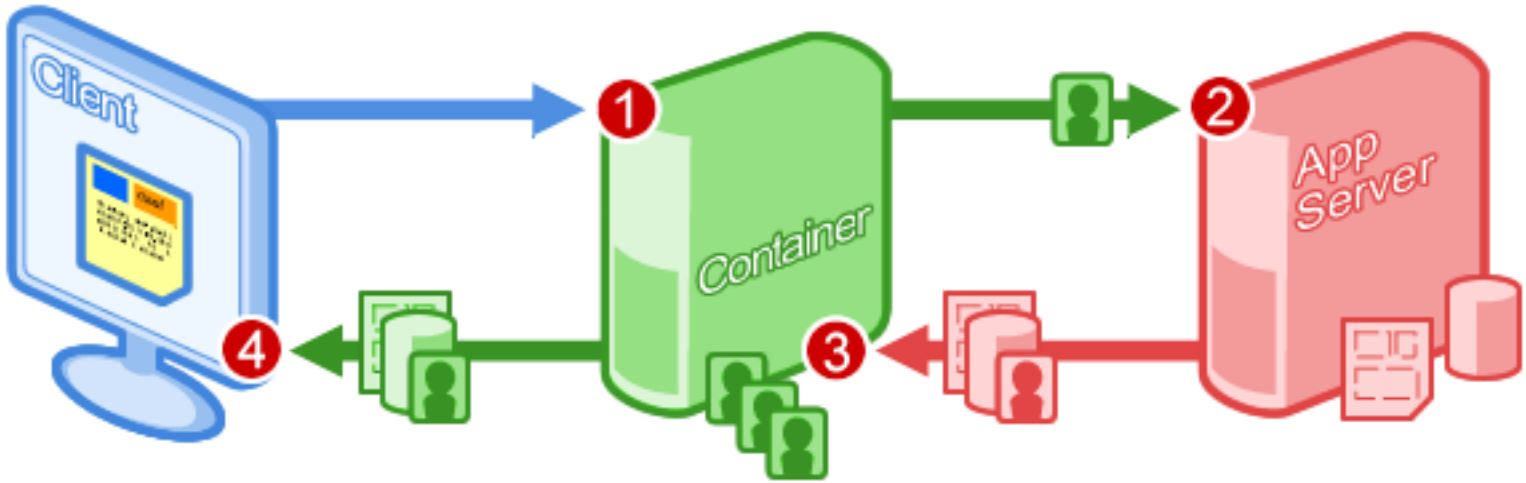
- The Naive implementation makes a lot of requests
- How can we improve on that?



OpenSocial Best Practices

Data Pipelining + Proxied Content

- Using OpenSocial 0.9's Data-Pipelining, we can declare which social data to POST to your server
- Your server operates on the data and returns the HTML to display
- Available in iGoogle & Orkut sandboxes, coming to a container near you soon(tm)



OpenSocial Best Practices

Data Pipelining + Proxied Content

```
<Module>
  <ModulePrefs ... etc .../>
  <Content type="html" view="profile"
                href="http://yoursite.com/proxied.php"
                authz="signed">
    <os:ViewerRequest key="vwrData" fields="id,displayName"
/>
    <os:OwnerRequest key="ownData"/>
    <os:PeopleRequest key="ownFriends"
                 userId="@owner" groupId="@self"/>
  </Content>
 </Module>
```

OpenSocial Best Practices

Data Pipelining + Proxied Content

```
<?php

$postData = json_decode(file_get_contents("php://input"));

echo "<h1>Hello {$postData['vwrData']['name']}</h1>";
echo "These are {$postData['ownData']['name']}'s friends:";
echo "<br/>";

foreach ($postData['ownFriends'] as $friend) {
   echo "{$friend['name']}<br/>";
}
```

Quartermile Improvement

After Data Pipelining

SPEED	NAIVE	PROXIED	DELTA
LATENCY (Gadget)	2730 MS	1094 MS	-1636 MS
LATENCY (Page)	3536 MS	2861 MS	-675 MS
REQUESTS	26	20	-6
YSlow! SCORE	72	76	+4

COST	NUMBER	COST	TOTAL	CHANGE
BANDWIDTH	3705 GB	$0.12	$444.60	
CPU TIME	902 HR	$0.10	$90.20	
$ / MONTH (11 QPS)			$534.80	-$320.94

OpenSocial Best Practices

Invalidation Pattern

- New application design pattern available with the features introduced in 0.9
- When your internal state changes on the application server, use REST/RPC calls to invalidate data:
  - Per user or url
  - Application ID is determined by 2 Legged OAuth call

OpenSocial Best Practices

Invalidation Pattern

- Calling the invalidation API:

```
POST /api/rest/cache/invalidate
HOST opensocial.example.org
Content-Type: application/json
{
    invalidationKeys : [
     "http://www.myapp.com/gadgetspec.xml",
     "http://yoursite.com/proxied.php"
     "user:123"]
}
```

Optimizing Your Data Store

Image by euthman http://www.flickr.com/photos/euthman/1846038389/

OpenSocial Best Practices

Data Store Structuring

- Database joins against friend lists are generally very expensive
- Plan ahead if you're not using App Engine
  - Master / Slave architecture
  - Database partitioning
- Use Memcache to cache data (in App Engine too!)
  - Filter results in software, make the most of cache
- Consider storing frequently used data in JSON Blobs instead traditional relational storage

OpenSocial Best Practices

Data Store Structuring

- Consider using background processing
  - Updates are slightly delayed
  - Doesn't block user interaction
  - Great for "What are your friends doing" result sets
- Use a off-the-shelf / open source Queue system or
- App Engine, use cron.yaml:

```
cron:
- description: process activities entries
  url: /tasks/processActivities
  schedule: every 1 minutes
```

Designing Quartermile's Data Model

- How to plan a scalable social application?
- **Prefer to enforce hard limits up front, than deliver a poor user experience**
- Decided friend queries were too expensive:
  - orkut lets you have 1000 friends
  - MySpace lets you have 100,000s+ of friends
- Do all 100,000 friends need to see your exercises?
  - Created artificial idea of "teams"
  - Choose a subset of friends to invite to your team
  - Side effect: Drive adoption!

Dreaming Up Reasonable Limits

- **Goal: Fetch all of a team's data for any given week in one database query**

- **How many users can we put on a team?**

- App Engine returns 1000 entries max for any query
- 1 entry / workout
- 3 / day ~ 20 / week

- 1000 / 20 = **50 users**

Limits: Made To Be Broken

Not every app will be able to enforce such restrictions

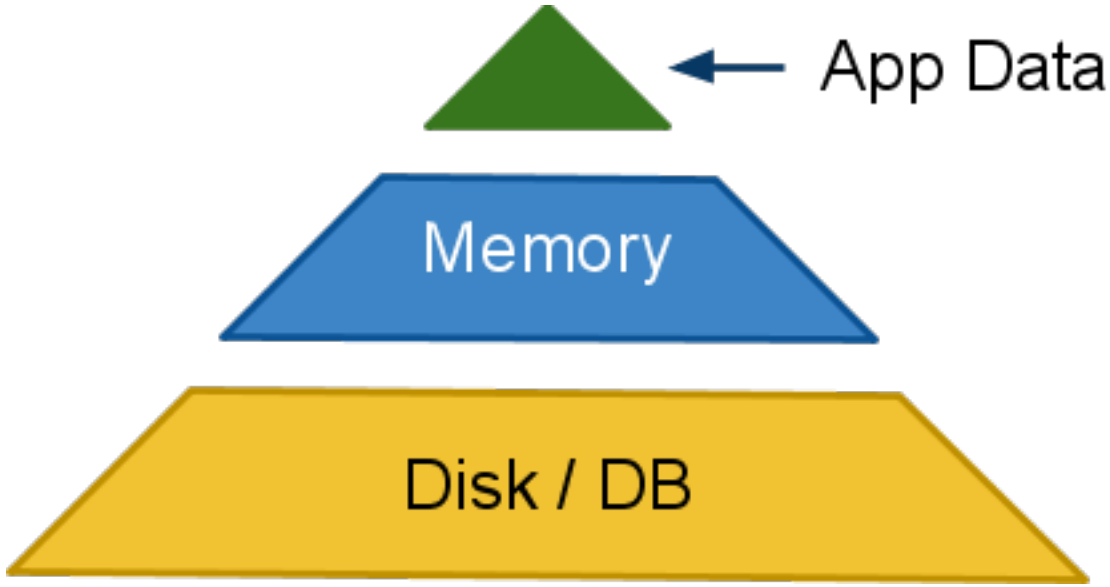
- **Goal: Implement "updates from your friends" inside of the Quartermile app**

- Slow!
  - Fetch all friends
  - See which updated recently
  - Sort

- Friend updates are lower priority than team updates
  - Process in the background
  - Fetch friends using 2-legged OAuth
  - Do "updates from friends" calculation
  - Store result of calculation (can be stale)

Where To Put It?

- Database
- Memcache
- OpenSocial App Data



"App Data is one of the most misunderstood and misused portions of the OpenSocial specification"

App Data is often used incorrectly

Arne Roomann-Kurrik, Google

Right now

App Data

- Data store of keys and values
  - Essentially public, so you can't put secrets here
  - User writable via JS, so you can't trust it
  - But it's fast!
- Perfect place to cache slow data

- Background process does slow calculation
- Pushes result to AppData
- When rendered, injected directly into the gadget
  - No server hit!

Container-Mandated Optimizations

Photo by redjar http://www.flickr.com/photos/redjar/704710355/

Container-Mandated Optimizations

- 'Naive' implementation:
  - Easy to make mistakes
- Container:
  - Keep gadgets fast == keep container fast
  - Userbase affects acceptable latency values
  - Constraints to keep gadget developers honest

iGoogle's Latency Penalty

- Directory will soon be taking latency into account
- Implicit latency penalty, too:

orkut's Template-Only Profiles

- Profiles:
  - The most traffic on orkut
  - Users love gadgets!
- OpenSocial 0.9 templates
  - Can display social data
  - Can display App Data
  - No external fetches or dynamic data
- Produces an extremely fast profile render
- Great use case for AppData cache

orkut's Template-Only Profiles

```
<Module>
  <ModulePrefs title="Template">
    <Require feature="opensocial-data" />
    <Require feature="opensocial-templates">
      <Param name="process-on-server">true</Param>
    </Require>
  </ModulePrefs>
  <Content type="html" view="profile"><![CDATA[
    <script type="text/os-data">
      <os:PeopleRequest key="friends" userId="@viewer"
          groupId="@friends"/>
    </script>
    <script type="text/os-template">
      <div repeat="${friends}">${Cur.name.givenName}</div>
    </script>
  ]]></Content>
</Module>
```

Trips 12 19 May 1827

David A Bisdale

D A B	1 Tat Batter	25	4¼	8½		6	4	
H	1 Ham	25		8½		17	2	
						1	3	4
				Straight		1	11	1 11
					£	1	1	5

Summary

~~Stephen Hoyt~~ John Banks

H	13¾ Bush Potatoes	W		2/4		1	12	1
				Straight			4	7 4 7
					£	1	7	6

James Ryder

H	7¾ Bush Potatoes	J		3/1		1	7	1
				Straight			2	7 2 7
					£	1	4	6

Summary

Comparison of each technique

App Size:

- JS Minification
- Content Rewriting
- Content Proxy
- Pipelining
- Invalidation
- Cache Headers

Requests:

- JS Minification
- Spriting
- Content Rewriting
- Cache Headers

Latency:

- JS Minification
- Spriting
- Content Rewriting
- Content Proxy
- Batching
- Data Store Optimization
- Background
- App Data Cache
- Pipelining
- Invalidation
- Limited Profiles
- Cache Headers

Quartermile 'Most Optmized' Implementation

SPEED	NAIVE	OPTIMAL	DELTA
LATENCY (Gadget)	2730 MS	833 MS	-1636 MS
LATENCY (Page)	3536 MS	2686 MS	-675 MS
REQUESTS	26	6	-20
YSlow! SCORE	72	89	+17

COST	NUMBER	COST	TOTAL	CHANGE
BANDWIDTH	3616 GB	$0.12	$433.92	
CPU TIME	963 HR	$0.10	$96.30	
$ / MONTH (11 QPS)			$530.22	-$325.52

Q & A

Post your questions for this talk on Google Moderator:
http://code.google.com/events/io/questions

Direct link:
http://bit.ly/opensocialspeedscale-questions


Google™
09
I/O