

TRIAL BY FIRE

REAL WORLD PERFORMANCE
OPTIMIZATION

Who are we?

Some dudes from Imaginary Landscape.

Dan Johnson

Joe Jasinski

What this talk is...

- High level performance overview
- Quickly address major areas of concern
- A talk for those looking for improving performance but have no idea where to start

What this talk isn't...

- A debate on which wsgi http server to use
- A “this is the way and the only way” talk
- An in-depth guide to configuration settings
- The only performance resource you'll ever need

Why Improve Performance?

Delay	User reaction
0 - 100 ms	Instant
100 - 300 ms	<i>Feels sluggish</i>
300 - 1000 ms	Machine is working...
1 s+	Mental context switch
10 s+	I'll come back later...

Source: Building Faster Websites, Ilya Grigorik - bit.ly/12GFKDE

Overall Goals

- Improve user experience via:
 - Quickly displaying response to user
 - Reducing the number of requests
 - Reducing the duration of requests
 - Reducing size of responses
- Improve performance of server by:
 - Offloading assets to 3rd party
 - Deferring non-essential tasks
 - Reducing time in request/response cycle

My site is slow... Where do I start?


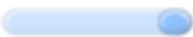














- This is always the hardest part.
- What is slow?
- Be patient
- Be suspicious
- Go see what is happening

Finding Problems on the Frontend

Common Problem Areas

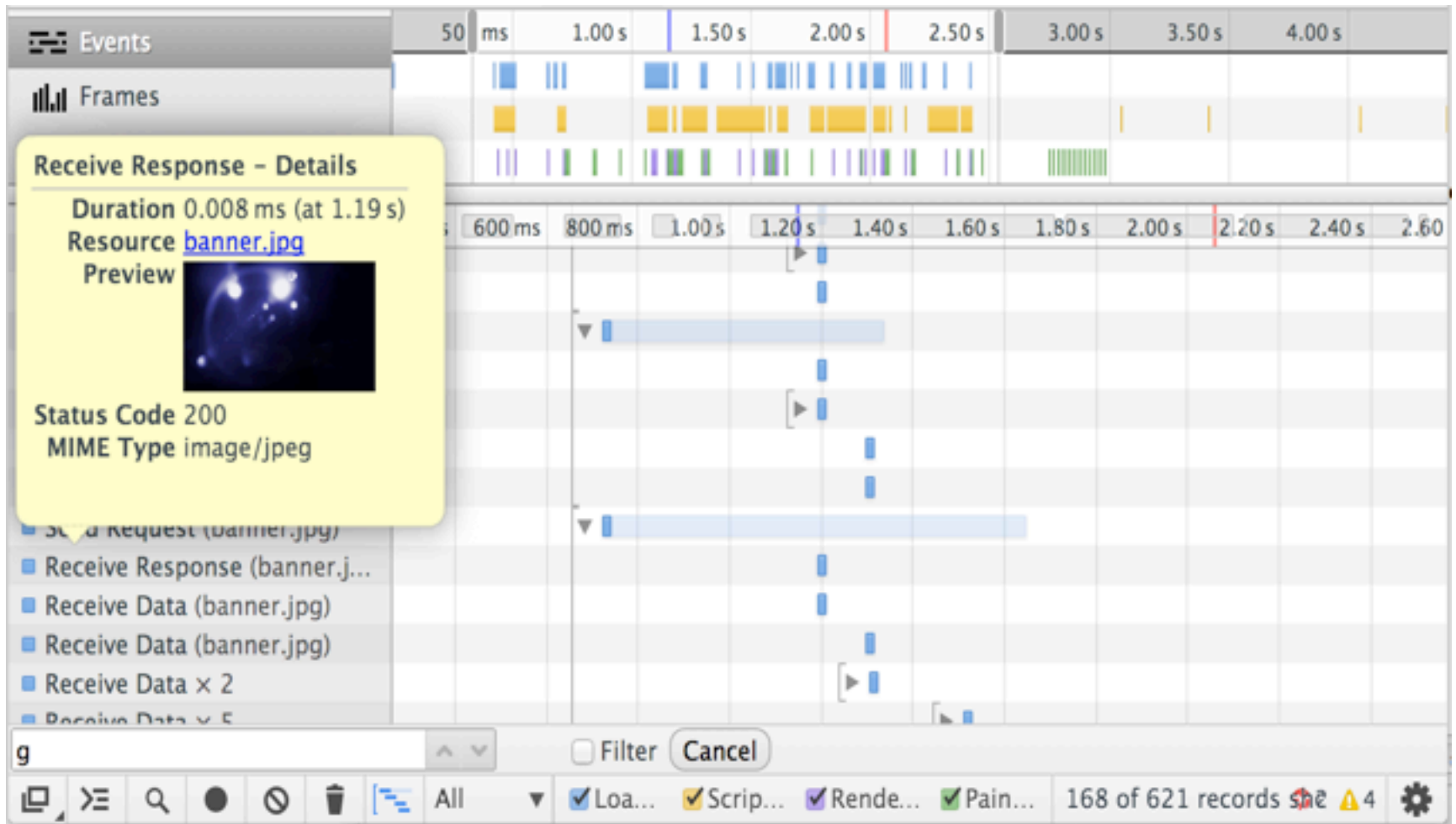
- Large and Unoptimized payloads
- Large quantity of blocking requests
- Slow responses from 3rd party resources

Chrome Developer Tools (waterfall)

× Elements Resources Network Sources Timeline Profiles Audits Console PageSpeed							
Name Path	Me...	Status Text	Type	Size Conten	Time Latency	Timeline	4.00 s
 www.kumbuya.com	GET	200 OK	text/...	6.1 KB 18.4 KE	1.80 s 1.76 s		
 1e56339c6103.css /static/css	GET	200 OK	text/...	27.9 ... 133 KB	219 ms 59 ms		
 jquery.min.js ajax.googleapis.com/ajax/libs/jqu	GET	200 OK	text/...	33.1 ... 91.2 KE	181 ms 138 ms		
 css?family=Lato:400,300,400ital... fonts.googleapis.com	GET	200 OK	text/...	756 B 1.3 KB	151 ms 150 ms		
 955e434524321f4c6feebd34a3c... d2u87r63gui73e.cloudfront.net/th	GET	200 OK	imag...	1.4 KB 963 B	41 ms 40 ms		
 26edeee1940a.js /static/js	GET	200 OK	appli...	72.3 ... 219 KB	260 ms 56 ms		
 socket.io.js /static/js	GET	200 OK	appli...	22.2 ... 72.9 KE	158 ms 71 ms		
 61cdf30117f7711b5c81f9522d... d1aoux765h41ox.cloudfront.net/tf	GET	200 OK	imag...	12.6 ... 12.2 KE	233 ms 225 ms		

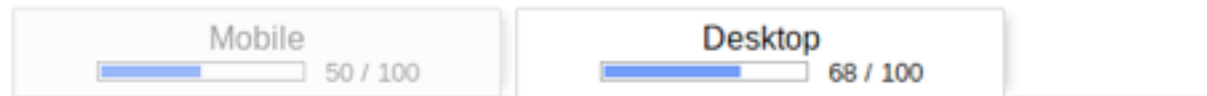
Documents Stylesheets Images Scripts XHR Fonts We 1 25

Events Timeline



External Analyzers

Google PageSpeed Insights



Suggestions Summary

! ▶ Leverage browser caching

Setting an expiry date or a maximum age in the HTTP headers for static resources instructs the browser to load previously downloaded resources from local disk rather than over the network.

! ▶ Eliminate render-blocking JavaScript and CSS in above-the-fold content

Your page has 2 blocking script resources and 2 blocking CSS resources. This causes a delay in rendering your page.

! ▶ Reduce server response time

! ▶ Minify JavaScript

Compacting JavaScript code can save many bytes of data and speed up downloading, parsing, and execution time.

✓ ▶ Enable compression

Compressing resources with gzip or deflate can reduce the number of bytes sent over the network.

Pingdom's Speed Tools



Yslow Browser Plugin

F	Make fewer HTTP requests
F	Use a Content Delivery Network (CDN)
A	Avoid empty src or href
F	Add Expires headers
B	Compress components with gzip
A	Put CSS at top
A	Put JavaScript at bottom
A	Avoid CSS expressions
n/a	Make JavaScript and CSS external
C	Reduce DNS lookups
F	Minify JavaScript and CSS
A	Avoid URL redirects
A	Remove duplicate JavaScript and CSS
A	Configure entity tags (ETags)

Grade F on Make fewer HTTP requests

This page has 27 external Javascript scripts. Try combining them into one.
This page has 3 external stylesheets. Try combining them into one.

Decreasing the number of components on a page reduces the number of HTTP
multiple CSS files into one style sheet, and use CSS Sprites and image maps.

[»Read More](#)

Copyright © 2013 Yahoo! Inc. All rights reserved.

Finding problems on the Backend

Typical Problem Areas

- Large quantity of SQL queries
- Overly complex SQL queries
- Performing unnecessary actions in the request/response cycle
- Complex template actions

Debug Toolbar

The image shows a screenshot of the Django Debug Toolbar (DDT) overlaid on a web application. The toolbar consists of several panels, each with a 'Hide' button and a 'Versions' dropdown menu. The panels are:

- Request Vars:** Shows the view function and its arguments. View Function: `theserver.views.home`, args: `None`, kwargs: `None`.
- View information:** Shows the view function and its arguments.
- COOKIES Variables:** Shows a table of cookies with columns 'Variable' and 'Value'. One variable is visible: `user` with value `0699203111371736535111d3m9n0f0eart0ubpoc...`.
- HTTP Headers:** Shows a table of headers with columns 'Key' and 'Value'. Headers include: `HTTP_ACCEPT` (text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8), `Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/28.0.1500.71 Chrome/28.0.1500.71 Safari/537.36`, `keep-alive` (8000), `localhost` (127.0.0.1), and `WSGI Server/0.1 Python/2.7.3`.
- SQL Queries from 1 connection:** Shows a table of SQL queries. One query is visible: `SELECT "django_session"."session_key", "django_session"."session_data", "django_session"."expire_date" FROM "django_session" WHERE ("django_session"."session_key" = 'mwsjb26iv4v2qg1xmy66g9y9s9kah6' AND "django_session"."expire_date" > '2013-08-08 17:37:55.357685')`. The query took 0.60ms.
- Templates (2 rendered):** Shows the template path and the templates rendered. The path is `/home/djohnson/projects/debug_toolbar_demo/proj/theserver/theserver/templates`. The templates are `home.html` and `base.html`.
- Context processors:** Shows the context processors used. One processor is visible: `django.core.context_processors.i18n`.

Each panel has a 'Hide' button and a 'Versions' dropdown menu showing the current version (Django 1.5.1) and the CPU usage (28.00ms / 26.26ms).

Cache Panel & Template Timings

The image shows two panels from Django's debug toolbar. The top panel is 'Template Timings' and the bottom panel is 'Cache Calls'. Both panels have a 'Hide' button in the top right corner.

Template Timings

Templates

Name	Average Time	Times	Total time	Min. time	Max. time	Queries	Query Duration
home.html	13.2 ms	1	13.2 ms	ms	13.2 ms	1	5.254 ms (39.91%)

Blocks

Name	Average Time	Times	Total time	Min. time	Max. time	Queries	Query Duration
content	0.1 ms	1	0.1 ms	ms	0.1 ms	0	0 ms

Cache Calls

Total Calls: 6 Total Time: 0.442ms Hits: 0 Misses: 3 Sets: 3 Gets: 3

Duration	Call	Key	Args	Result
0.107 ms	get	var1	()	None
Stacktrace: /home/tjohnson/projects/debug_toolbar_demo/local/lib/python2.7/site-packages/django/contrib/staticfiles/handlers.py in <code>__call__</code> (72) return self.application(enviro, start_response) /home/tjohnson/projects/debug_toolbar_demo/proj/theserver/theserver/views.py in <code>home</code> (10) var1 = cache.get('var1')				
0.082 ms	get	var2	()	None
0.076 ms	set	var2	(1235677345345345, 500)	None
0.067 ms	get	var3	()	None
0.064 ms	set	var1	({'something': 'else'}, 500)	None
0.046 ms	set	var3	('Some generic data', 500)	None

Cache Panel Summary:

- Hide »
- Versions: Django 1.5.1
- Time: CPU: 28.00ms (26.26ms)
- Settings
- HTTP Headers
- Request Vars
- Templates
- SQL: 1 query in 0.60ms
- Signals: 6 receivers from 12 signals
- Logging: 0 messages
- Cache: 6 calls, 0.44ms
- Template Timings:** 13 ms over 1 queries (39.91% SQL)

Profiling Middleware

Get contextual information on all calls.

If something looks strange, its probably worth investigating.

```
Sun Sep 1 12:00:31 2013 /tmp/tmpSbcmii
```

```
72804 function calls (70095 primitive calls) in 0.065 seconds
```

```
Ordered by: cumulative time
```

ncalls	totttime	percall	cumtime	percall	filename:lineno(function)
1	0.000	0.000	0.065	0.065	views.py:249(landing)
1	0.000	0.000	0.059	0.059	base.py:44(view)
1	0.000	0.000	0.059	0.059	base.py:58(dispatch)
1	0.000	0.000	0.059	0.059	base.py:122(get)
1	0.000	0.000	0.059	0.059	views.py:166(get_context_data)
6	0.000	0.000	0.045	0.008	query.py:108(_result_iter)
4	0.000	0.000	0.045	0.011	query.py:867(_fill_cache)
7	0.000	0.000	0.045	0.006	query.py:231(iterator)
7	0.000	0.000	0.042	0.006	compiler.py:751(results_iter)

Improving Front End Performance

HTML Minification

- Remove non-essential whitespace in html
 - Results in fewer bytes going over the wire
- Django module: django-htmlmin

```
<!DOCTYPE html><html lang="en"><head><link href="/static/images/favicon.ico" rel="SHORTCUT
ICON"/> <title>Jaz Studios Home</title><meta content="text/html; charset=utf-8" http-
equiv="Content-Type"/><meta content="JAZ Studios, jaz studios, video, editing, web design,
web development, programing, programming, Joe Jasinski, Joe, Jaz, Jazz, Jaz Studios"
name="keywords"/><meta content="Personal web development, video editing, photography, and
computer programming site" name="description"/><meta content="Joe J. Jasinski"
name="author"/><meta content="true" name="HandheldFriendly"/><meta content="width=device-
width; initial-scale=0.666667; maximum-scale=0.666667; user-scalable=0" name="viewport"/>
<meta content="width=device-width" name="viewport"/><!--[if ie]><meta http-equiv="X-UA-
Compatible" content="IE=edge" /><![endif]--><link href="/static/CACHE/css/2e8410b45f53.css"
rel="stylesheet" type="text/css"/><script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"
type="text/javascript"></script></head><body><div itemscope=""
itemtype="http://schema.org/Person"><span content="Joe Jasinski" itemprop="name"><span
content="Web Application Architect" itemprop="jobTitle"><div itemprop="address"
itemscope="" itemtype="http://schema.org/PostalAddress"><span content="Chicago"
itemprop="addressLocality"><span content="IL" itemprop="addressRegion"></span></span></div>
</span></span></div><div class="pagebody"> <div class="header"> <div class="banner"></div>
<div class="top content_width"><h1 class="brand">Jaz Studios</h1><div class="menu_top"><ul>
```

JS and CSS Compression

- Combine all JS and all CSS for fewer requests
- Compress combined files
- Create unique urls for compressed objects
 - **i.e.** /static/CACHE/css/2e8410b45f53.css

Django Compressor

Before Compression

```
{% load compress %}
{% compress css %}<link rel="stylesheet" href="/static/css/one.css"
type="text/css" charset="utf-8"><style type="text/css">p { border:5px solid
green;}</style><link rel="stylesheet" href="/static/css/two.css" type="text/
css" charset="utf-8">
{% endcompress %}
```

After Compression

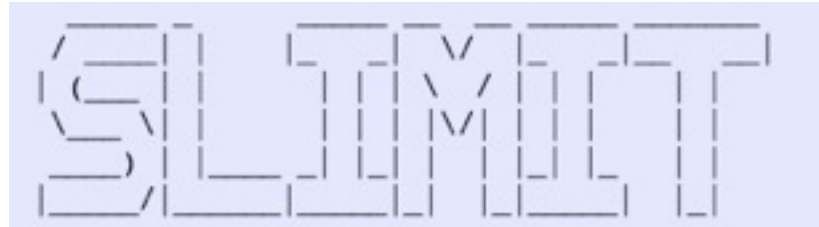
```
<link rel="stylesheet" href="/static/CACHE/css/f7c661b7a124.css" type="text/
css" charset="utf-8">
```


Django Compressor Supported Engines

CSSTidy

JSMIn

cssmin



Closure Compiler

Image Optimizations

- Remove unneeded image meta-data
- Compress Images when you can
 - Suggestion: size \leq 70 kb
 - Offer WebP to supported browsers
 - JPEG? PNG?

Django tools for auto-thumbnailing:

- sori-thumbnails
- easy-thumbnails

Image Sprites to Reduce # Requests

```
<style>
div { float: left; border: solid black 1px;} #add, #edit, #remove, #join
{ width:20px; height:20px;}#remove { background:url("remove.png") } #add
{ background:url("add.png") }#edit { background:url("edit.png") } #join { background:url("join.png") }
</style><div id="remove"></div><div id="add"></div><div id="edit"></div><div id="join"></div>
```

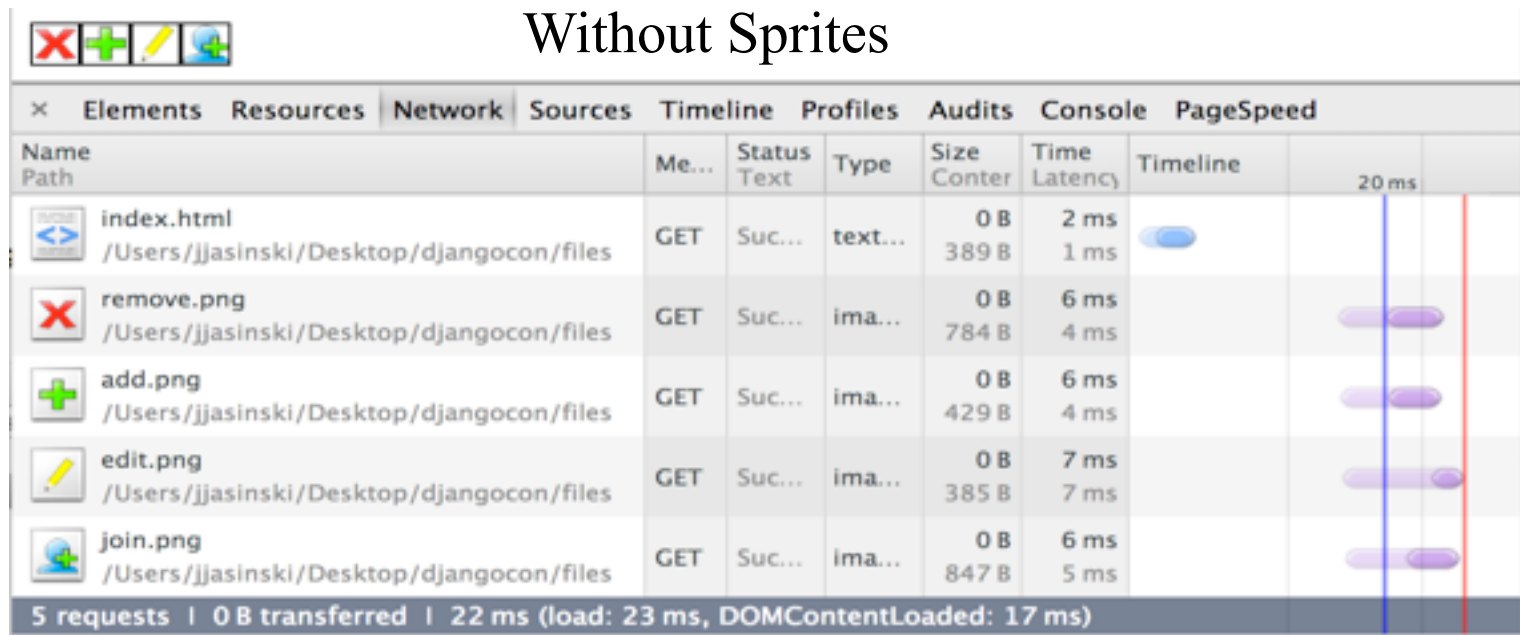






Image Sprites to Reduce # Requests

```
<style>
div { float: left; border: solid black 1px;} #add, #edit, #remove, #join, #leave
{ width:20px; height:20px; }#remove { background:url("sprites.png") 0 0; }#add
{ background:url("sprites.png") 20 0; }#edit { background:url("sprites.png") 36 0; width:18px; }#join
{ background:url("sprites.png") 56 0; }
</style><div id="remove"></div><div id="add"></div><div id="edit"></div><div id="join"></div>
```

With Sprites



× Elements Resources Network Sources Timeline Profiles Audits Console PageSpeed							
Name Path	Me...	Status Text	Type	Size Conter	Time Latency	Timeline	10 ms
 index_sprite.html /Users/jjasinski/Desktop/djangocon/files	GET	Suc...	text...	0 B 441 B	1 ms 1 ms		
 sprites.png /Users/jjasinski/Desktop/djangocon/files	GET	Suc...	ima...	0 B 2.4 KB	2 ms 2 ms		

2 requests | 0 B transferred | 12 ms (load: 13 ms, DOMContentLoaded: 12 ms)

Resource Order

- Load first styles in the critical path
 - Consider inlining a few critical styles
- Place javascript after other resources
 - ideally at end of the html document
- Markup after script tags is not processed until after the script has been downloaded and executed

Lazy Loading

- Don't bother loading images far down on a page until a user gets there.
 - If they never scroll down that far, why bother right?
- Can be done via JS or Nginx module
- Particularly useful for pages with long lists of images
- Reduce traffic to server/CDN

Asset CDNS

Host assets on a content delivery network.

- Worldwide edge nodes
- Serve assets from closest server
- Have high availability of up time
- Reduce load from server
- Improve site load time

Each alternate domain used requires a DNS lookup and a round-trip.

Improving Back End Performance

SQL Queries

- Use “values_list”
- Don't be afraid to use “raw”
 - But be skeptical if you really need to
- Verify you are not repeating the same query multiple times.
- Use select_related and prefetch_related

Select Related

```
users = User.objects.all()[:10]
for user in users: print user.userprofile.slug
```

11 SQL queries!

```
users = User.objects.select_related(
    'userprofile')[:10]
```

1 SQL query!

Prefetch Related

```
content = Content.objects.all()[:10]
for c in content: print c.want_it_users.all()
```

11 SQL queries!

```
c = Content.objects.all().prefetch_related(
    'want_it_users')[:10]
```

2 SQL queries!

Cache?

- “Just cache it.”
- Which cache should I use?
 - Redis?
 - Memcached?
 - Local Mem?
 - Database?
- How should I cache it?

Django Low Level Caching

1. Low level cache API
 - a. `django.core.cache`
2. Reduce expensive lookups
3. Utilize `get()`, `set()`, and `delete()` methods directly modify individual cache keys
4. Cache invalidation can be a challenge
5. Consider priming the cache

Django Low Level Caching (cont)

Example: lookup a potentially cached object within a manager

```
class MyModelManager(models.Manager):  
    def lookup_variable(self, string):  
        return_value = cache.get(string, [])  
        if not return_value:  
            logger.debug("Miss: not in Cache: %s" % (string))  
            return_value = u"%s" % self.get_query_set().get(slug=string).value  
            cache.set(string, return_value)  
        else:  
            logger.debug("Hit: Value Fetched from Cache: %s" % (string))  
        return return_value
```

Template Fragment Caching

- Renders pieces of template content and caches the HTML

```
{% with role=mymodel|role:request.user %}  
  {% cache 3000 cache_key_name role %}  
    {% include "mymodule/includes/my_cached_include.html" %}  
  {% endcache %}  
{% endwith %}
```

The Per-Site Cache

- Cache the your entire site through middleware
- Configurable to only cache anonymous visits
- Built In and Easily Configurable
- Has some problems with Google Analytics
 - Easily fixed!

Cache Frameworks

There are a variety of cache frameworks:

- Johnny Cache
- Cache Machine

Both provide ORM model caching and automatic invalidation on updates.

Easily drop them into your application with minimal effort.

More on CDNs...

Browsers:

- ~6 connections per hostname
- more maximum connections

This means there are still ways we can fully take advantage of what the browser can do.

We should serve our assets from several CDN subdomains so that we use as many of the available connections as possible.

Hash Ring with CDN Domains

- Consistent hashing

```
MEDIA_CDN_DOMAINS = ['1.foo.net', '2.foo.net',]
```

```
HASH_RING = hash_ring.HashRing(cdn_domains)
```

```
# Overriding url method in my storage backend
```

```
def url(self, name):
```

```
    return "://%s/%s" % (HASH_RING.get_node(name), name)
```

The url returned is the same each time, unless I change the number of cdn domains.

Limit what you do inside a request

Always ask yourself “Does this have to happen right now?”

If the answer is “no” or “maybe”, defer it.

Your goal should be to do the bare minimum inside of a request/response cycle to accomplish the job.

Every Django process is valuable.

Job Queues

Use job queues systems like Celery or RQ (Redis Queue) to run jobs out of band.

Celery is the “go-to” solution and offers some additional features over redis-queue. Typically requires more configuration.

RQ is simple to configure and run “out of the box”, but less featureful.

Sample Job Queue (using RQ)

```
# jobs.py
```

```
def make_a_call(x):
```

```
    # Do a time consuming task
```

```
    api_result = api.send(x)
```

```
    result = models.Result()
```

```
    result.result = api_result
```

```
    result.save()
```

```
# the shell
```

```
from django_rq import *
```

```
queue = get_queue('default')
```

```
import jobs
```

```
j = queue.enqueue(
```

```
    jobs.make_a_call, 1)
```

```
(rq-presentation)djohnson@dan-p7-1380t:~/../rqdemo$ python manage.py rqworker default
```

```
14:43:46 RQ worker started, version 0.3.7
```

```
14:43:46
```

```
14:43:46 *** Listening on default...
```

Server Level Optimizations

Set Expires and Cache Control

- Expires: helps browsers to cache content
 - Download once per cache-period
- Cache-Control: external caches may cache

```
location /static {  
    root /www/site/htdocs;  
    access_log off;  
    add_header Pragma public;  
    add_header Cache-Control "public";  
    location ~* \.(css|js)$ {  
        expires 60d;  
    }  
}
```


Use Gzip Responses

- Most browsers support gzip content encoding
 - Reduces page download size
- Configured at the Application or Server level
 - Django GzipMiddleware
 - Nginx/Apache configuration
- Lowers network delay in exchange for a bit of CPU work
- Do NOT use for HTTPS content
 - Due to recent BREACH attack

Google ModPageSpeed

- Automates a lot of the aforementioned
 - Combine/Compresses css/js automatically
 - Optimizes images; removes metadata
 - Fingerprints assets
 - Adds expires header
 - Serves webp as needed
- Asynchronously optimizes/caches content on first load
- Nginx/Apache Module

Database Connection Poolers

- Sits between the application and the database
- Reuses database connections
 - prevents connection overhead
- Relatively easy to configure for a quick performance boost
- Use IPs for database connections

Use Cached Django Sessions

Don't use database for session storage

Why?

- Page request = Database read
- Session create/modify = Database write

Cached session reduce database load and have a higher performance throughput.

Would you like to know more?

- Django Debug Toolbar:
 - Debug toolbar: <https://github.com/django-debug-toolbar/django-debug-toolbar>
 - Cache Panel: <https://github.com/lincolnloop/django-cache-panel>
 - Template Timings: <https://github.com/orf/django-debug-toolbar-template-timings>
- Hash Ring: <http://amix.dk/blog/viewEntry/19367>
- Browser Concurrent Connections: <http://www.browserscope.org/results?o=xhr&v=top&category=network>
- Lazy Image Loading:
 - PageSpeed Module (Nginx/Apache): <https://developers.google.com/speed/pagespeed/module/filter-lazyload-images>
 - Unveil.js: <http://luis-almeida.github.io/unveil/>
 - lazyload.js: <http://www.appelsiini.net/projects/lazyload>
- Task Queues:
 - Celery: <http://celeryproject.org/>
 - django-celery: <http://docs.celeryproject.org/en/latest/django/index.html>
 - RQ: <http://python-rq.org/>
 - django-rq: <https://github.com/ui/django-rq>
- Caching:
 - Johnny Cache: <http://pythonhosted.org/johnny-cache/>
 - Cache Machine: <https://cache-machine.readthedocs.org/en/latest/>
 - The Per-Site Cache: <https://docs.djangoproject.com/en/dev/topics/cache/#the-per-site-cache>
 - Fixing Site-Wide Caching: <https://www.silviogutierrez.com/blog/fixing-site-wide-caching-django/>
 - Where Django Caching Busts at the Seams: <http://www.slideshare.net/csky/where-django-caching-bust-at-the-seams>
- Minification
 - HTML minification with django-htmlmin <https://github.com/cobrateam/django-htmlmin>
 - Image Thumbnail
 - sorl-thumbnail <http://sorl-thumbnail.readthedocs.org/en/latest/>
 - easy-thumbnail <https://github.com/SmileyChris/easy-thumbnails>



Would you like to know more? (cont)

- Minification (continued)
 - CSS minifiers comparison: <http://www.phpied.com/css-minifiers-comparison/>
 - Slimy Vs YUI: <http://ruslanspivak.com/2012/01/15/slimy-vs-yui-compressor/>
- Profiling Middleware: <http://djangosnippets.org/snippets/1579/>
- Cache Controls & Headers:
 - Google Resource: <https://developers.google.com/speed/docs/best-practices/caching>
 - Cache Control Directives Demystified: <http://palpapers.plynt.com/issues/2008Jul/cache-control-attributes/>
- Minimize round-trip time: <https://developers.google.com/speed/docs/best-practices/rtt>
- Developer Tools:
 - Chrome Network Tab: <https://developers.google.com/chrome-developer-tools/docs/network>
 - Chrome Events Tab: <https://developers.google.com/chrome-developer-tools/docs/timeline>
- Third Parties Speed Analyzers:
 - Google Pagespeed: <https://developers.google.com/speed/pagespeed/insights/>
 - Yahoo!'s Yslow: <http://developer.yahoo.com/yslow/>
 - Pingdom's Speed Tools: <http://tools.pingdom.com/fpt/>
- Cached Session Backends:
 - Redis: <https://pypi.python.org/pypi/django-redis-sessions>
 - Memcached: <https://docs.djangoproject.com/en/dev/topics/http/sessions/?from=olddocs#using-cached-sessions>
- General Information:
 - 1000ms to Glass: <http://alistapart.com/blog/post/breaking-the-1000ms-time-to-glass-mobile-barrier>

Dan Johnson

Joe Jasinski

Imaginary Landscape
www.imagescape.com