Make Plone Fast!

Use CacheFu to Make Your Site Fly

Geoff Davis

geoff@geoffdavis.net

Plone Symposium, 2006

Overview

- Big Picture
  - The Problem: Plone is slow
  - The Solution: CacheFu
- How does CacheFu work?
  - Key concepts
  - Gory details
- Squid

How fast is your site?

- Simplest measurement: Apache benchmark (ab)
  - comes with Apache 2.0 distribution
  - simulates lots of users hitting a single page sequentially and / or simultaneously
  - measures pages served / second
- Limitations of ab
  - doesn't load associated images, CSS, JS
    - JS and CSS matter a lot! ~50% of your bandwidth
  - doesn't know about browser caching, etc

- Better benchmarks feasible with Selenium??

How fast is Plone out of the box?

- ab = Apache benchmark
  - part of the Apache 2.0 distribution
- ab -n 50 http://localhost:8080/myplonesite/
  - 50 requests for front page
  - Key number to look for is “Requests per second:” (average; median is better)

Using ab

- Tips:
  - Make sure you add the trailing “/” to the URL
  - Be sure your site has “warmed up” before running
    - Lots of one-time startup expenses
      - ZODB needs to load objects into memory
      - pages templates need to be parsed, etc
    - Run twice and look only at second result
  - Make sure Zope is not in debug mode

Results

- ~3.5 requests/sec on my laptop
  - SLOW!
- Front page is only part of the problem:
  - also have ~200K of CSS / JS / images!
- Quick tip: If you have an English-only site, delete PlacelessTranslationService
  - Boosts speed to 4 req/sec (~15%)

CacheFu

- Download CacheFu
  - Copy 4 packages to my Products directory:
    - CacheSetup
    - PageCacheManager
    - PolicyHTTPCacheManager
    - CMFSquidTool
  - Install CacheSetup with QuickInstaller
- Repeat the ab test:
  - Get ~35 req/second
  - ~10x faster; also improves JS, CSS, and images

CacheFu + squid

- Set up squid
  - Install squid
  - Set up the squid.conf that ships with CacheFu
  - Adjust squid settings in the cache settings portlet
- Run ab again
  - Get 150 req/sec
  - ~40x faster

Transparency

- CacheFu is almost completely transparent
  - CacheFu caches content views (not everything)
  - Big problem: cache needs to be purged when content changes
  - CacheFu takes care of this for you
- When you update your content, changes will appear on your site immediately!
  - A few exceptions; we will discuss these
  - Need to understand how things work to make this work for you
  - Very straightforward in most cases

How does CacheFu work?

- CacheFu is pretty complicated
  - Ideas are straightforward
  - Infrastructure in Zope for implementing them is not
  - Lots of partial solutions that step on each other
  - Biggest value-add: (relatively) seamless integration

- Not a perfect solution
- Hopefully will provide a better way to think about the problem in Zope 3

Why is Plone slow?

- Multiple sources
- In order of decreasing importance:
  - Page rendering
  - ZServer
  - Network latency
  - Connection setup times
- We will attack each problem separately
  - Multiple approaches to some problems

Speeding things up

- Page Rendering
  - Lots of benchmarking
  - Biggest time sink is TAL rendering
  - Not much we can do about it
  - EXCEPT not render
- Cache pages to reduce rendering time
  - Several different ways

Speeding things up

- ZServer sluggishness
  - Don't use ZServer when we don't have to
- ZServer is smart
  - Don't need brains to serve up static content

- Set up fast proxy cache (squid)
  - Proxy cache handles static stuff
  - ZServer handles things that require some smarts

Speeding things up

- Network latency
  - Tell browsers not to ask for things they don't need
    - Caching!
  - Don't re-send pages when you don't have to
    - More caching!
  - Compress content
    - gzip HTML pages
    - JS / CSS whitespace removal related tricks

Speeding things up

- Connection setup times
  - Combine multiple CSS files into one
  - Combine multiple JS files into one
  - Prevent unnecessary requests
    - Cache as much as possible (but no more) in the client

Caching, Caching, and more Caching

- Common theme in all approaches: Cache!

- Several different types of caching
  - Cache in server memory
  - Cache in proxy cache
  - Cache in client's browser
    - “Unconditional” client-side caching
      - Browser always uses local file
    - “Conditional” client-side caching (**NEW!**)
      - Browser checks with server before using local file

More techniques

Will touch on a few more approaches, but not in depth

- Tune the ZODB/ZEO object caches
  - speeds up Zserver
- Load balancing
  - reduces page rendering times under load
- Optimize your code
  - reduces page rendering time
- Cache intermediate code results
  - reduces page rendering time

Strategy 1: Cache static content in browser

- When user visits site, content stored in their browser's cache
  - HTTP headers tell how long to cache
- Subsequent requests pulled from local cache rather than server

- Most useful for *static content* that is *viewed frequently*
  - *Images, CSS, JS*

HTTP headers

- Understand HTTP headers to do caching right
- Good tutorial at http://www.web-caching.com/mnot_tutorial/

HTTP header basics

- Will use both HTTP 1.0 and 1.1 headers in case ancient clients visit

- HTTP 1.0 headers
  - Expires: [date and time]
    - Browser will cache if date is in the future
  - Last-Modified: [date and time]
    - Complicated heuristics for image caching based on Last-Modified header absent more explicit info
    - The longer your image has been unchanged, the longer the browser will cache it
  - Headache: both require correct client clock

HTTP header basics

- HTTP 1.1: much more fine-grained control
  - Cache-Control: [tons of options]
    - Most important for our purposes:
      - max-age=N
        - browser will cache your content for N seconds
        - preferable to Expires because makes no assumptions about client clock
      - public
        - tells browser OK to cache even when it might not otherwise
    - Cache-Control options not to include (for now):
      - no-cache, no-store, must-revalidate, private

Setting HTTP headers

- AcceleratedHTTPCacheManager
  - Part of CMF - sets cache headers for skin elements
  - Used by Plone OOTB to set headers for static stuff
  - HTTPCache
  - Associate template / image / file with HTTPCache using metadata
  - cache=HTTPCache

- One of the 10 places that headers get tweaked

CacheFu and headers

- CacheFu consolidates header-setting
  - Most headers set in CachingPolicyManager
  - Allows for much finer-grained control
    - We will need it!

- CacheFu replaces HTTPCache with a PolicyHTTPCacheManager
  - Farms HTTPCache's old job out to CachingPolicyManager
- Sets better default cache timeout
  - 24 hours instead of 1 hour

CachingPolicyManager

- Take a look in ZMI: caching_policy_manager
  - Details: Definitive Guide to Plone, Chapter 14
  - http://docs.neuroinf.de/PloneBook/ch14.rst
- Container full of header setting policies
  - Each policy has a predicate
  - Pages to be rendered walk through policies until they hit a true predicate, then headers are set
- You will not need to look in here much
  - Most of policy-choosing logic is elsewhere

Caching Policy

- CacheFu assigns the cache_in_browser policy to items associated with HTTPCache

- cache_in_browser policy:
  - key items:
    - last-modified = python:object.modified()
    - max-age = 86400
      - 86400 secs = 24 hours
    - s-max-age = 86400
      - instructions to squid
    - public
      - Use cached items even in situations when maybe not OK (e.g. when authorized, possibly with https connections, etc)

Caching in Browser

- cache_in_browser policy gives us the least control
  - Once something is in the browser, it is stuck there
  - Browser won't check for anything newer for 24 hours
- Takes a big load off server, though
  - Safe to use this policy for things that rarely change
  - If you plan to change stuff, consider:
    - lower max-age time limit the day before
    - increase again when you are done

Testing the headers

- LiveHTTPHeaders plug-in for FireFox
  - Your new best friend

- Invaluable for testing caching
- Shows all request and response headers

- Tip: clear your browser cache manually before starting a session

- Let's take a look

ResourceRegistries

- Most of the content associated with HTTPCache is images
- JS and CSS used to be, but no more

- ResourceRegistries are the new way to go
  - In the ZMI:
    - portal_css
    - portal_javascripts
  - Let's take a look

ResourceRegistries

- Look at portal_css
- Lots of CSS files registered
- Line in main_template pulls in all registered CSS in the page <head> section
- Options:
  - Enabled: lets you turn on/off file inclusion
  - TAL condition: lets you conditionally include
  - Merging allowed: can file be merged?
  - Caching allowed: used for RR's internal caching (which we bypass)

ResourceRegistries

- RR serves up a set of merged CSS files with URLs like this:
  - portal_css/Default%20Skin/ploneStyles1234.css
  - Skin name is in the URL so that different skins have distinct URLS
    - Avoids user retrieving cached css file for one skin when viewing a different skin
  - Number in filename is version number
    - every time you hit Save button, version number changes

ResourceRegistries

- Version number is VERY IMPORTANT
  - Means you can cache stuff forever in browser
  - When you change your CSS, hit Save
    - Merged filename changes
    - Pages now point to new CSS file; user won't see the old one
- CSS and JS are ~1/2 of bandwidth on a typical site
  - If you have repeat visitors, long-time caching is great

ResourceRegistries

- Added bonus:
  - RR 1.3 does safe CSS and JS compression
  - (Plone 2.1.2 ships with RR 1.2)

- Ideal solution: serve gzipped CSS / JS
  - Buggy in many browsers, unfortunately
  - RR instead strips whitespace, other tricks
    - “Safe” compression cuts CSS and JS by about 25% each
    - More aggressive compression cuts JS by ~50%
  - RR does this on the fly each request
    - CacheFu caches the results so RR only compresses once

ResourceRegistries

- CacheFu bypasses RR's caching machinery
  - Routes JS and CSS through caching_policy_manager
- Policy used is cache_file_forever
  - CSS and JS can live on the browser for a year
  - Really important to remember to Save!

ResourceRegistries

- Tips:
  - Files have to be mergeable for renaming to work
  - Use debug mode for development and debugging
    - Files don't get merged or cached
  - Pages cached in squid may refer to the old CSS / JS files
    - If you make big CSS/JS changes and want them to appear immediately, you will also have to purge squid
    - purging script (purgesquid) is supplied

Quick Recap

- Step 1: Cache your static content in the browser
  - Associate files and images in your skins with HTTPCache
    - Use cache=HTTPCache in the .metadata file
    - CacheFu will do the rest
  - Register your CSS and JS with portal_css/portal_js
    - Make them mergeable
    - Save when css/js change
    - CacheFu will take care of caching

Quick Recap

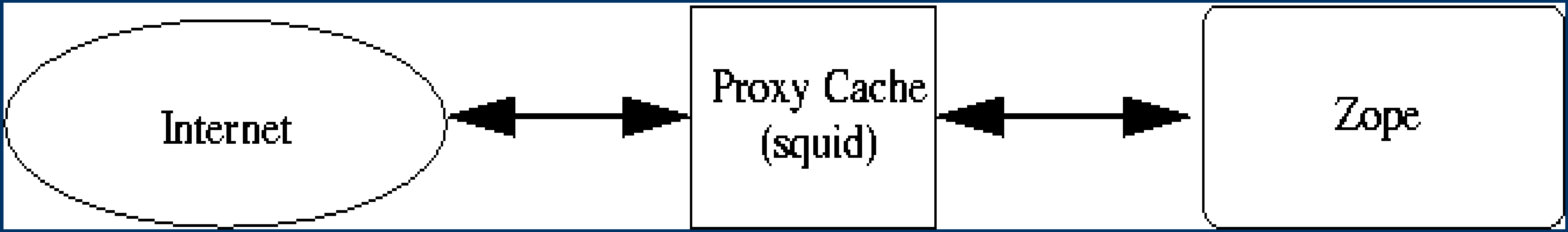
- Keep limitations in mind
  - Only helps if people load the URL more than once!
    - Great for CSS / JS / images that appear on all pages
  - Once it's on the browser, can't change until it expires
    - Unless you are using something cool like RR

Proxy cache

- Benefit of browser cache:
  - Every request served by cache is one less request served by ZServer
- Drawback of browser cache:
  - Can't invalidate stale content
- Alternative for content that changes more frequently: use a **proxy cache**

Strategy 2: Proxy Caching

- Idea: put a fast but dumb proxy cache in front of Zope
- Proxy cache serves up (static) content, keeps load off Zope
- Zope can tell proxy cache when content expires so you don't serve up stale content



Proxy cache

- Because it is server side, cached content is shared
  - Browser cache only helps if 1 client requests same resource twice
  - Proxy cache helps if 2 (anonymous) people request same thing even if they are different people
  - Much less help when content is personalized, though
    - Our strategy: cache anonymous content
    - Possible to expand if content is personalized based on, say, roles instead of username
    - Will talk more about personalized content later

Plone and content caching

- By default, Plone sends no Cache-Control header, which means that pages won't be cached in general

- Anything using main_template has headers set in global_cache_headers.pt
  - In CMFPlone/skins/plone_templates
    - contains Cache-Control: no-cache
  - CacheFu overrides, uses caching_policy_manager instead

Plone and content caching

- Want to override default headers for a single page?
  - Simplest way: call request.RESPONSE.setHeader in body of template.
    - Overrides previous header, affects only template in question.
    - May get stomped by caching_policy_manager
  - Harder way: create a caching_policy_manager policy
- (You won't need to do this in general)

Content cache headers

- Goal is to cache anonymous content views
- Not much point caching personalized views
  - Not enough hits per cached page to justify
  - Fills up the cache

- How do we control content cache headers?
  - With a caching policy, of course
  - Content views will use 2 different policies
    - cache_in_squid if you are anonymous
    - cache_in_memory if you are authenticated

Content cache policies

- Leave content in squid; purge as needed
- cache_in_squid
  - max-age = 0
    - Don't cache in the browser!
  - s-max-age = 86400
    - Cache in squid for up to 24 hours

- Keep out of squid
- cache_in_memory
  - Don't cache in browser or squid
  - max-age = 0, s-max-age = 0

How policies are assigned

- How does Zope know what caching policies to apply?
  - Cache setup tool controls everything: The One Ring
    - Integrates the 7 different products
  - Nice portlet – let's look
    - Site setup -> Cache Configuration Tool
  - Main tab controls relationship with squid
    - Talk about that later
  - Next tabs control policy assignments

Cache configuration tool

- When an object looks for headers, it gets sent to CacheSetup
- CacheSetup walks through its own policies to figure out what the appropriate caching policy is

- HTTPCache content
  - Assigns all content associated with HTTPCache
  - Both anonymous and authenticated users get “Cache in browser” policy
  - Hopefully reasonably self-explanatory

Cache configuration tool

- Next tab: Plone content types
- Have an object + template in hand. Does the policy apply?
  - Look at content type – is it in the list?
  - Look at template
    - Is it a default view for the object?
    - Is it on the list of templates?
  - Look at request – is there anything that should stop caching?

Cache configuration tool

- Ok, so the configuration policy applies, now what?
- Need to figure out a caching policy
- 2 methods:
  - Use policy specified for anonymous or authenticated users
  - Get the policy ID from an external script
- For default views of main Plone content objects:
  - cache in squid for anonymous users
  - cache in memory for authenticated users

Cache configuration tool

- For default views of main Plone container objects:
  - cache in memory for anonymous and authenticated users
- Reason:
  - Can purge content objects when they change, BUT
  - Container views change when any of their contents change
    - So either all content has to purge parent OR
    - Just cache in RAM and work out purging another way (will discuss later)

Caching Your Views

- Recommended method:
  - Add a new assignment policy
- In portal_cache_settings, add a new content policy
  - Select your content types
  - Indicate that default views should be cached
  - Choose type of caching policy for anonymous and authenticated
  - Configure ETags (will discuss later – default Plone Etags are good starting point)

Purging

- What happens when content changes?
  - CMFSquidTool purges the object
    - CacheSetup configures squidtool so you don't have to
  - Monkey patches index, unindex, reindex, etc
  - When an object is created / modified / deleted, cache is purged
- Cache configuration tool figures out the right pages to purge
  - Typically just the views and templates specified
  - If you want extras, you can add a script

Purging

- Plone content types
  - uses script to purge extra pages
- Why?
  - If you modify the file “myfile”, need to purge:
    - default views: myfile, myfile/, myfile/view
    - also myfile/download
  - If you modify the image “myimg”, need to purge
    - default views: myimg, myimg/, myimg/view
    - also myimg/image_thumbnail, etc
- Script supplies the extra /download, image_thumbnail, etc

Proxy Caches

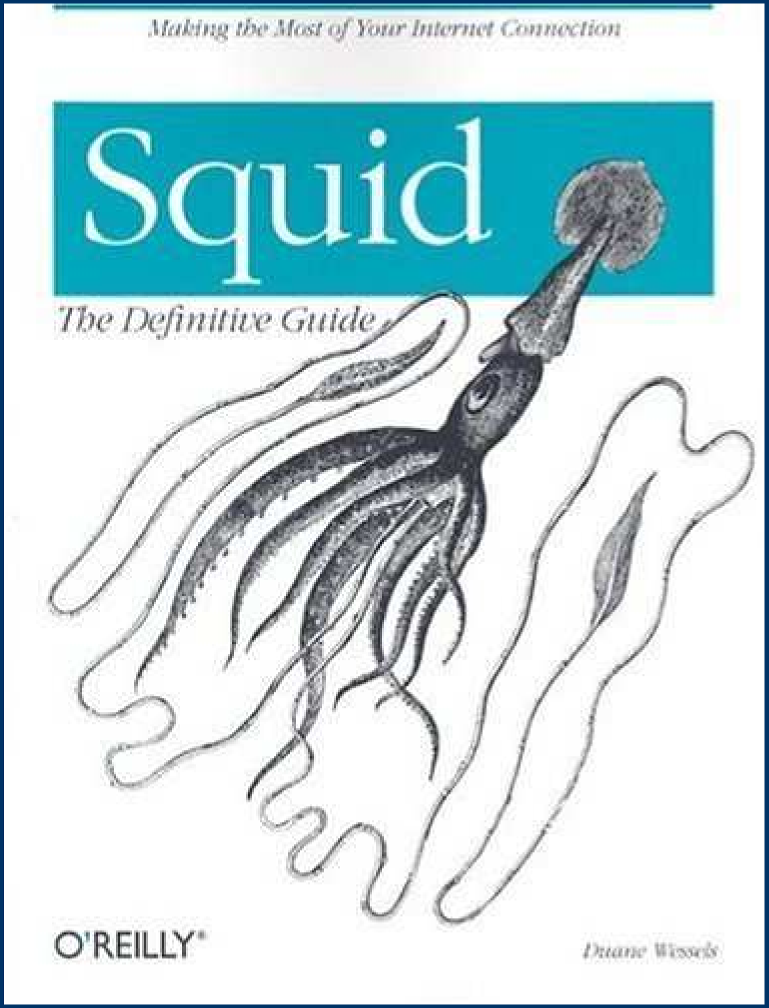
- Squid
  - free, open source; runs on Linux, Windows, OSX
  - http://www.squid-cache.org
  - Super fast (~150 requests/second on mid-range box)
- Some (but probably not all) of CacheFu strategy should work with IIS + Enfold Enterprise Proxy
  - http://www.enfoldsystems.com/Products/EEP

Why not Apache?

- Apache + mod_cache
  - Lots of documentation about using Apache for caching
- Problem: mod_cache doesn’t support purging
  - No easy way to delete stale pages from cache
- Should be possible to modify CacheFu to get some (but not full) benefit from Apache
  - 1-2 days work
  - Sponsorship welcome!

Using Squid

- Excellent documentation available
- (Only need to read a few chapters, not whole book)



Using Squid

- Squid has a reputation of being complex
- Problem is that default squid.conf is 3500 lines
  - 99% documentation
  - most options don't apply
- CacheFu contains sample squid.conf
  - 137 lines (including comments)
  - straightforward to configure

- CacheFu has sample configurations for
  - squid by itself
  - squid behind Apache
    - useful if you need to wire together different web apps and want to use mod_rewrite, etc
  - setup is similar
- Pick the appropriate setup

Configuring squid

- Go to the directory for the configuration you have chosen
  - squid_direct or squid_behind_apache

- Edit squid.conf and follow the instructions
  - Walkthrough
- Edit redirector_class.py and set up the redirection rules
  - Syntax is like mod_rewrite for Apache
  - Walkthrough

Setting up squid

- Copy everything (squid.conf, all .py files) to /etc/squid
- Fire up squid!

Setting up squid

- Tips:
  - Check file permissions
    - squid must have read access to squid.conf, iRedirector.py, squidAcl.py, and redirector_class.py
    - squid must have execute access to iRedirectory.py and squidAcl.py
  - squidAcl.py and iRedirectory.py get called directly
    - First line is #!/usr/local/bin/python -Ou
    - If your python is not at /usr/local/bin/python, change the path to python in the first lines of these files
    - Make sure you can run both of these from the command line without getting an exception

Setting up squid

- More tips:
  - While debugging your squid configuration, run squid from the command line and echo errors to the console:
    - /usr/sbin/squid -d1
  - To stop squid from the command line, use
    - /usr/sbin/squid -k kill
  - To reconfigure squid after modifying squid.conf, use:
    - /usr/sbin/squid -k reconfigure

Setting up squid

- More tips:
  - Look at squid's logs if you have problems
    - /var/log/squid/cache.log – squid messages about its internal state
      - If you notice all squid's external processes are dying, it probably means that you have a problem with your python path in iRedirector.py or squidAcl.py
      - Try running these python files from the command line to see what's going on. Use "./iRedirector.py", NOT "python iRedirector.py"
    - /var/log/squid/access.log – squid messages about cache hits and misses

Setting up squid

- Tips:
  - iRedirector.py does URL rewriting
  - Uses redirector_class.py as a helper
    - Both iRedirector.py and redirector.py do debug logging
    - Edit them and replace “debug = 0” with “debug = 1” if you have problems

Setting up squid

- Once you have squid working, It Just Works
- Setup can be a headache the first time
  - Tips should help a lot

Configuring CacheFu for Squid

- Once squid runs, tell Zope about it
- Go to first pane of Cache configuration tool
  - Indicate URLs of your site
    - include *all* URLs, e.g. http://www.mysite.com, https://www.mysite.com, http://mysite.com, etc
  - If squid behind apache, URL of squid (typically http://localhost:3128)

Vary header and gzipping

- Set the Vary header (default should be OK)
  - Vary header tells squid to store different versions of content depending on the values of the headers specified
  - Vary: Accept-Encoding for gzip
    - One version for browsers that accept gzipped content
    - One version for those that don't
- Select gzipping method (default is recommended)
  - Gzipping cuts down network latency
  - Content cached in gzipped form so only gzip once

Demo

- Let's try it out!

- Tips:
  - Use LiveHTTPHeaders to see if getting cache hits
  - Look at headers:
    - X-Cache: HIT or X-Cache: MISS
  - If you don't see any HITs, clear your browser cache manually and try again
  - If that fails, something may be wrong

Strategy 3: Load Balancing

- Zope Enterprise Objects let you do load balancing
  - ZEO server = essentially an object database
  - ZEO client executes your python scripts, serves up your content, etc
  - ZEO comes with Zope
- Set up multiple ZEO clients on multiple machines or multiple processors (single instance of Zope won't take much advantage of multiple processors)

Setting up ZEO

- You can transform a Zope site into a ZEO site using the mkzeoinstance.py script in ~Zope/bin
- Change a few lines in ~instance/etc/zope.conf and ~instance/etc/zeo.conf and you are good to go
- See Definitive Guide to Plone, Chapter 14
  - http://docs.neuroinf.de/PloneBook/ch14.rst

Squid + ZEO

- Main idea: give your proxy cache lots of places from which to get content it can't serve
- Squid can in theory take care of load balancing
- I would use pound instead
  - pound = load-balancing proxy designed for Zope
  - http://www.apsis.ch/pound/
  - Put pound between squid and ZEO clients
  - Big advantage if you use sessions – pound keeps client talking to same back-end server

Resource requirements

- My site: 20K page views/day
  - 1 squid instance, 1 ZEO client
  - 2.4 GHz P4 + 1G RAM
- plone.org:
  - 1 squid instance + 2 ZEO clients
  - 2x 3GHz Xeon box with 2 GB of RAM
- Bulk of load is from authenticated clients
- Don't need that much power, especially if most clients are anonymous
- squid is *very* efficient
- Main requirement is lots of memory for Zope

Strategy 4: Use Entity Tags

- ETags let us do smart browser caching
- The idea:
  - ETag = arbitrary string, should have the property:
    - If I have 2 files with same ETag, files should be the same
  - Send an ETag to browser with a page
  - Browser caches the page
  - Before rendering from cache, browser sends ETag of cached page to server
  - Server responds with Status 304 + no page (meaning cached stuff OK) or Status 200 + new page

ETags

- What are good ETags?
  - Depends on what we are serving up

- Example: Images
  - 2 images with same URL and same modification time are probably the same
  - ETag for images, files can just be last modified time
  - ETags not really useful for files and images, since we can do a conditional request based on modification time

ETags

- Example: document
  - ETag for document should include modification time
    - That lets us distinguish different versions of the doc
  - Should depend on authenticated member
    - Since we have personalization in document view
  - Should depend on state of the navtree, other portlets

Setting ETags

- CacheFu provides an easy way to generate ETags
- Go to policy for Plone content in Cache configuration portlet
  - Look at ETag section
  - Ingredients for building an Etag
    - Use member ID (personalization)
    - Time of last catalog modification (covers age of document + navtree state)
    - REQUEST vars: month, year, orig_query (covers state of calendar portlet)
    - Time out after 3600 secs

ETags

- ETags useful for 2 things
  - First, allows for smart conditional browser caching
    - If document changes or something in document's containing folder changes or calendar changes or logged in member changes, ETag will change
  - Second, provides a useful cache key for a RAM cache

PageCacheManager

- PageCacheManager stores full pages + headers in a memory
  - Uses ETags as cache key, so ETag is required
  - ETags are set using CachingPolicyManager policy

- If template uses Cache configuration tool to generate an ETag and policy is not “Do not cache”
- CacheFu automatically associates templates that have ETags generated
- Content views automatically cached in memory

PageCacheManager

- Try it out
- Look for X-Pagecache: HIT

Things you should know

- Some things to watch out for when digging deeper
  - If browser has a page in hand, will do a conditional GET
    - GET /foo
    - If-None-Match: ETAG-OF-PAGE-IN-HAND
    - If-Modified-Since: LAST-MOD-OF-PAGE-IN-HAND
  - Squid can handle If-Modified-Since but is too dumb to deal with If-None-Match
  - Any requests with an If-None-Match bypass squid
    - Code in squidAcl.py is used to do this

More things you should know

- Squid is not typically very useful for caching content from authenticated users
  - squidAcl.py causes squid to be bypassed if the user is authenticated
- Squid IS useful for caching images and files even if user is authenticated
  - Code in squid.conf that tells squid to always use the cache for files ending with .js, .css, .jpg, etc

More things you should know

- Images and Files get routed through CachingPolicyManager through a nasty method
  - Monkey patch associates them with DefaultCache
  - DefaultCache is an HTTPPolicyCacheManager
- Existing caching policies assume that images and files do not have any security on them and are the same for authenticated and anonymous users
  - May be be possible to work around but will require some effort

Strategy 5: Optimize Your Code

- Don't guess about what to optimize – use a profiler
- Several available
  - Zope Profiler:
    - http://www.dieter.handshake.de/pyprojects/zope/
  - Call Profiler:
    - http://zope.org/Members/richard/CallProfiler
  - Page Template Profiler:
    - http://zope.org/Members/guido_w/PTProfiler
- Identify and focus on slowest macros / calls

Code Optimization: Example

- Suppose you find that a portlet is your bottleneck
  - Calendar portlet, for example, is pretty expensive
- How to fix?
- Idea: don't update calendar portlet every hit
  - Update, say, every hour
  - Cache the result in memory
  - Serve up the cached result
- Similar idea applies to other possible bottlenecks:
  - *Cache the most expensive pieces of your pages*

RAMCacheManager

- RAMCacheManager is a standard Zope product
- Caches results of associated templates / scripts in memory
- Caveats:
  - Can't cache persistent objects
  - Can't cache macros

- Calendar portlet is a macro – how can we cache it?

Trick: Caching Macro Output

- Idea:
  - create a template that renders the macro
  - output of template is snippet of HTML, i.e. a string
  - cache output of the template

Caching the Calendar

- Step 1: Create a template called cache_calendar.pt:

  <metal:macro use-macro=”here/portlet_calendar/macros/portlet” />

- Step 2: In the ZMI, add a RAMCacheManager to your site root

- Step 3: in the RAMCacheManager, set the REQUEST variables to AUTHENTICATED_USER, leave the others as defaults (this caches one calendar per user)

Caching the Calendar

- Step 4: Associate cache_calendar.pt with your new RAMCacheManager. Output of cache_calendar.pt will now be cached for 1 hour.

- Step 5: In your site's properties tab, replace here/portlet_calendar/macros/portlet with here/cache_calendar
- Voila!

- Use RAMCacheManager to cache output of slow scripts, etc.

Future Directions

- Make CacheFu more Apache-friendly
  - Should be possible to make CacheFu work without squid (currently only provides limited benefits)
- General clean-up and polish
  - Autogenerate squid config files
  - More unit tests
  - Minor refactoring for simplification
  - Let PageCacheManager use memcached
- Even bigger gains to be had...

Future directions

- Poor man's ESI
  - Split out chunks of pages
  - Cache them independently
  - Insert SSI directives in their place
  - Have Apache reassemble chunks
- Header, footer, portlets, personal bar, etc could all be cached and invalidated separately
- CacheFu speeds up views – this could speed up *everything*

- *Sponsorship welcomed! geoff@geoffdavis.net*