

High Performance Plone

Naples Conference 2007

Author: Joel Burton <joel@joelburton.com>
Copyright: Copyright 2007 Joel Burton. All rights reserved.
Covering: Plone 2.5 or newer
Notice: Distribution outside of presentation prohibited.
Company: PloneBootcamps: www.plonebootcamps.com

Contents

1	High Performance Plone	2
1.1	Introduction	3
1.1.1	About Me	3
1.1.2	Plone Does A Lot	3
1.1.3	Plone Does A Lot II	3
1.1.4	This Is Good	3
1.1.5	This Can Be Bad	3
1.1.6	Rephrasing	4
1.1.7	Benchmarking In 1 Minute	4
1.1.8	Using ab	4
1.1.9	ab Results	4
1.1.10	ab Shortcomings	4
1.1.11	Holistic Benchmarking	4
1.2	Bank of Plone	5
1.2.1	Bank of Plone	5
1.2.2	Bank of Plone	5
1.2.3	Diagnosing Problem	5
1.3	ZEO	5
1.3.1	ZEO Overview	6
1.3.2	ZEO Speed Improvements	6
1.3.3	Creating a ZEO Server	6
1.3.4	Using New ZEO Server	6
1.3.5	Multiple Zopes	7
1.3.6	Load Balancing	7
1.4	Caching	7
1.4.1	Where Things Get Cached	7
1.4.2	Where Things Get Cached	7
1.4.3	Where Things Get Cached	8
1.4.4	Where Things Get Cached	8
1.4.5	Disk Caching by HD	8
1.4.6	ZODB Caching	8
1.4.7	ZEO Caching	8
1.4.8	RAMCache Manager	8

1.4.9	RAMCache Unconditionally	9
1.4.10	RAMCache For Limited Time	9
1.4.11	RAMCache On User + Time	9
1.4.12	ZSQL Method Caching	9
1.4.13	Memoize	9
1.4.14	View Caching	10
1.4.15	Strategies for Caching	10
1.4.16	Caching Risks	10
1.4.17	Proxy Server	10
1.4.18	Apache	10
1.4.19	Squid	11
1.4.20	Varnish	11
1.4.21	Apache -> Squid -> Zope	11
1.4.22	Squid Is Still New To Us	11
1.4.23	CacheFu	11
1.4.24	Getting CacheFu	12
1.4.25	CacheFu Concepts	12
1.4.26	ETags	12
1.4.27	CacheFu Settings	12
1.4.28	Other Tricks CacheFu Does	12
1.4.29	Header Policies	13
1.4.30	Rulesets	13
1.4.31	Default CacheFu Rulesets	13
1.4.32	The Sibling Case	13
1.4.33	Rule: HTTPCache	13
1.4.34	To Do: HTTPCache	14
1.4.35	Rule: ContentCache	14
1.4.36	To Do: ContentCache	14
1.4.37	Rule: Containers	14
1.4.38	To Do: ContentCache	14
1.4.39	Rule: Templates	15
1.4.40	To Do: Templates	15
1.4.41	Rule: CSS/JS	15
1.4.42	To Do: CSS/JS	15
1.4.43	Rule: Files/Images	15
1.4.44	To Do: Files/Images	15
1.4.45	Rule: DTML CSS	16
1.4.46	To Do: DTML CSS	16
1.4.47	Personalization Vs. Security	16
1.4.48	Speed of Caches	16
1.4.49	Strategies	16
1.4.50	Strategies II	16
1.4.51	Strategies III	17
1.4.52	Strategies IV	17
1.4.53	Strategies V	17
1.4.54	Strategies VI	17
1.4.55	Getting More Information	17
1.4.56	How You Can Help	17
1.4.57	Thanks!	17

1 High Performance Plone

1.1 Introduction

1.1.1 About Me

- Site builder / Plone integrator
- Plone trainer
 - Trainings at plonebootcamps.com
- Volunteer on Plone project

1.1.2 Plone Does A Lot

- Checks for authentication
 - HTTP is stateless
- Find the object in the DB
- Check that user has rights to object
 - And to everything (skins, etc.)

1.1.3 Plone Does A Lot II

- “Skin” object
- Perform any logic (portlets, etc.)
- Add custom info about user
- Decided which JS/CSS to show
- Create dynamic navigation elements

1.1.4 This Is Good

- Everything is fresh
- Everything is security-aware
- High degree of personalization

1.1.5 This Can Be Bad

- High stress on server

1.1.6 Rephrasing

- It's not true that "Plone is slow"
 - It's quite efficient
- Fair to say "It's doing more than I need"
 - We can trade off freshness for speed

1.1.7 Benchmarking In 1 Minute

- ab, Apache Benchmark
 - From Apache Foundation
 - Included in most Linux distributions

1.1.8 Using ab

```
ab -n 50 -c 2 http://yoursite/
```

- n: number of requests
- c: concurrency
- Don't forget the trailing slash!

1.1.9 ab Results

```
HTML transferred:      152650 bytes
Requests per second:   64.35 [#/sec] (mean)
Time per request:     15.540 [ms] (mean)
Transfer rate:        203.35 [Kbytes/sec] received
```

1.1.10 ab Shortcomings

- Doesn't act like a real user
 - Requesting the same page 50 times?!
 - Measures just HTTP page
 - * Not CSS/javascript

1.1.11 **Holistic Benchmarking**

- Specialized tools & web applications
- In a pinch, Selenium can work fine

1.2 **Bank of Plone**

1.2.1 **Bank of Plone**

- Biking to the bank
- Waiting in line
- Pulling your records
- Time with your teller
- Taking your money

1.2.2 **Bank of Plone**

- Biking to the bank: Network connection speed
- Waiting in line: Waiting for worker
- Pulling your records: Getting object
- Time with your teller: Speed/caching
- Taking your money: Streaming results

1.2.3 **Diagnosing Problem**

- *CallProfiler*
 - Skin objects (templates, scripts)
- *PTProfiler*
 - Page template TALEX expressions
- Zope Profiler
 - Low level: all methods called

1.3 **ZEO**

1.3.1 ZEO Overview

- ZEO
 - Object server (ORB)
 - Zope server is ZEO client
 - Allows multiple Zope clients

- Use ZEO all the time

Even though ZEO was invented for scaling Zope, it's recommended you use it for your development environment. This has several advantages:

Testing for deployment Though the cases are rare, there are differences in running under ZEO and not. It's better to discover this on your development server than your production server.

Faster restarts Since the ZODB isn't restarted when Zope is restarted

Debugging a live server You can debug using a second ZEO client without touching the main ZEO client.

1.3.2 ZEO Speed Improvements

- Page production takes same time
 - But often starts sooner
- Also provides failover capability

1.3.3 Creating a ZEO Server

- Script to create new ZEO server:

```
$SOFTWARE_HOME/bin/mkzeoinstance.py home [port]
```

- *port* is port ZEO server runs on

- Move existing *Data.fs* from *var* to *zoe/var*

1.3.4 Using New ZEO Server

- Edit `$INSTANCE_HOME/etc/zope.conf`
 - Comment out first `zope_db main`; uncomment second
 - Port should be set to ZEO server

1.3.5 Multiple Zopes

- Can run on separate physical servers
- Or more than once on a single box
 - Run at least 1 Zope/CPU
- Python, Zope, and all Products **must** be same
 - Or very bad things can happen
- Products that store files on disk require shared disk access
 - eg LocalFolderNG, ExternalFile
 - NFS, Samba, or other solutions fine

1.3.6 Load Balancing

- Handled by proxy server
 - Best choice for small/medium setups
- DNS round-robin
 - “zope.mynetwork” goes to different boxes
- Dedicated hardware
 - Same IP goes to different boxes

1.4 Caching

1.4.1 Where Things Get Cached

- Low-level than Zope
 - Completely transparent
 - * ie, no side effects, Zope is unaware of

1.4.2 Where Things Get Cached

- Inside Zope
 - Least dramatic benefits
 - Easy to recognize changes + update

1.4.3 Where Things Get Cached

- On a proxy server
 - Excellent benefits
 - Changes can be often be handled
 - Great for lots of different users

1.4.4 Where Things Get Cached

- In browser
 - Can be best benefits
 - Hard to notify changes
 - Tremendous benefit for same users coming back

1.4.5 Disk Caching by HD

- Completely transparent to Zope

1.4.6 ZODB Caching

- Retrieval of raw object is cached
 - Prior to skinning / work
- Completely transparent to your app
- Settings in `zope.conf`
 - May be worthwhile to change

1.4.7 ZEO Caching

- Retrieval of object over ZEO is cached
 - Prior to skinning / work
- Completely transparent to app
- Settings in `zeo.conf`

1.4.8 RAMCache Manager

- Generalized cache for result-of-rendering
 - Commonly PythonScripts
 - * Also could be PageTemplates / DTML / etc

1.4.9 RAMCache Unconditionally

```
# Script (Python) "mult.py"  
## parameters = x, y  
  
return x * y
```

- Can cache forever on the parameters
 - Parameters always considered in cache

1.4.10 RAMCache For Limited Time

```
# Script (Python) "getWeather.py"  
## parameters = zipcode  
  
return context.getWeatherFromWebService(zipcode)
```

- Results should be stale after some period

1.4.11 RAMCache On User + Time

```
# Script (Python) "getArticles.py"  
  
return [ article.Title for article in  
         context.portal_catalog(Type='Article') ]
```

- Need to separate my results from yours
- Cache using AUTHENTICATED_USER as key

1.4.12 ZSQL Method Caching

- For each ZSQL Method, can cache
 - Cached for search keys
- DBs are often slow, can be helpful
- Don't cache INSERT/UPDATE, etc.
 - Though this might be useful for logging apps

1.4.13 Memoize

- New caching system for Zope3-style code and views
 - Very flexible in how it can vary
 - No exposed UI for configurability

1.4.14 View Caching

```
>>> from Products.Five import BrowserView
>>> from plone.memoize import ram

>>> def _render_details_cachekey(method, self, brain):
...     return hash((brain.getPath(), brain.modified))

>>> class View(BrowserView):
...     @ram.cache(_render_details_cachekey)
...     def render_details(self, brain):
...         obj = brain.getObject()
...         view = obj.restrictedTraverse('@@obj-view')
...         return view.render()
```

1.4.15 Strategies for Caching

- Microcaching is very helpful
 - 5 sec - 5 mins not likely to be noticed
 - Can reduce work by 500x
- Name caches and associate NOW
 - Even if you turn on later!

1.4.16 Caching Risks

- Staleness
 - Content has changed / been deleted
- Inappropriate delivery
 - ie, showing something to user they shouldn't see

1.4.17 Proxy Server

- Technically, this is “reverse proxy”
- Squid, Apache, other caching apps
 - Can only do invalidation in Squid

1.4.18 Apache

- Excellent, world-class web server
- Weak proxy server
- Since we can't invalidate, we don't emit headers for things that might need it

1.4.19 Squid

- Excellent proxy server
- Not as featureful for generic web serving
- Can invalidate, so can cache and handle stale cases

1.4.20 Varnish

- Excellent proxy server
 - Though not quite feature-complete to Squid
 - Nor as documented / well-known
- Very promising for non-risk-averse

1.4.21 Apache -> Squid -> Zope

- Apache handles PHP, other stuff
- Squid caches Zope output
- Everyone wins!
 - More complex to set up

1.4.22 Squid Is Still New To Us

- Very stable product, used in serious deployments
 - Not widely used among small/medium Plone people
- Sample squid configs and (new!) config wizard
- Good (longish) book by O'Reilly
 - Only have to read the reverse-caching parts!

1.4.23 CacheFu

- Combination of useful caching strategies
 - Merged and enhanced by Geoff Davis
- Maximum Caching with Minimum Side Effects
 - But lots of knobs for making more tradeoffs

1.4.24 Getting CacheFu

- Download from Plone Software Center at plone.org
 - Included (but not installed!) with Plone 2.5 and 3.0
 - Might need to grab products CacheFu/ directory
- Several products, install them all

1.4.25 CacheFu Concepts

- Caches some things in Zope
 - When we need fast invalidation, user-specific cache
- Caches some stuff in proxy server
 - When we need invalidation but not user-specific
- Caches some stuff in browser
 - When it can live forever and a day

1.4.26 ETags

- Stuff we use to determine “state” of something
 - Date it changed
 - User (or user role or permission or ...)
 - Date of any catalog change
- Used to “tell” when we can re-present

1.4.27 CacheFu Settings

- Zope, Squid, Apache, Varnish or Combos
- Site domains
- Squid:
 - Address of your Squid

1.4.28 Other Tricks CacheFu Does

- Can compress (gzip) output
- Can remove extraneous whitespace
- Takes less time to travel over the wire
- Keep these on safe
 - I worry about buggy browsers

1.4.29 Header Policies

- Controls actual HTTP cache settings emitted
- Geek-land
 - Few moving parts for mortals
- Have names like “Cache in Squid for 24h”, etc.

1.4.30 Rulesets

- Chain of rules that are attempted
 - In order
 - Try to find right policy (in-Zope or output header)

1.4.31 Default CacheFu Rulesets

- Generally, conservative
- Everything is invalidated on change
- Auth users never use each others cached data

1.4.32 The Sibling Case

- Only affects anonymous users
- When editing something (newsitems/project-a)
 - Display of that is 100% up-to-date
 - However, navigation may show sibling items in folder
 - * These titles/URLs can be stale
- Times out after an hour

1.4.33 Rule: HTTPCache

- Pages that
 - Rarely change
 - Have no security information
 - Are not personalized

1.4.34 To Do: HTTPCache

- Associate pages with *HTTPCache*

1.4.35 Rule: ContentCache

- “Normal” content pages
 - Cached in memory or Squid for anonymous
 - Cached with “ETag” check for members
 - * Any catalog change invalidates

1.4.36 To Do: ContentCache

- Add “never cache” REQUEST things
 - Like `portal_status_message`
- Add “vary” REQUEST things
 - Like `month`, `year` for catalog
- Add non-default templates
 - eg, `shorter_newsitem_view`
- Select for new types of content

1.4.37 Rule: Containers

- “Folderish” things
 - Cached with ETags for everyone
 - (Can’t use Squid because can’t invalidate on child changes)

1.4.38 To Do: ContentCache

- Add “never cache” REQUEST things
 - Like `portal_status_message`
- Add “vary” REQUEST things
 - Like `month`, `year` for catalog
- Add non-default templates
 - eg, `shorter_newsitem_view`
- Select for new types of folders

1.4.39 Rule: Templates

- “Non-content” templates
 - But might show content info, like a catalog query
- Cached with ETags, not Squid

1.4.40 To Do: Templates

- Add any applicable templates

1.4.41 Rule: CSS/JS

- Cached **forever**
 - CSS + JS registries merge and number files
 - As soon as CSS or JS changes, gets new number

1.4.42 To Do: CSS/JS

- Nothing :)

1.4.43 Rule: Files/Images

- The “actual” thing, not the page-about
 - Page-about is cached with content rule
- Files & Image have no personalization
 - So, either it has changed or hasn’t
- Cached in Squid if anon-can-view
- Cached in browser if private

1.4.44 To Do: Files/Images

- Select new content types that act like this
 - Flash
 - Movies
 - etc.

1.4.45 Rule: DTML CSS

- CSS that might be different per user/request/etc.
 - IE-specific CSS is only one for now
- Caches in memory for 24 hours

1.4.46 To Do: DTML CSS

- If needed, add your CSS files here.

1.4.47 Personalization Vs. Security

- ETag on user name
 - “Personalized” for user
- ETag on roles
 - Non-personalized, but same security
 - Can cache much more
 - Take user name off of page!
 - * Could still show role

1.4.48 Speed of Caches

- CacheFu In Memory: ~30-40x
- Squid: ~5-10x CacheFu in Memory
 - Up to ~400x! plain Plone

1.4.49 Strategies

- Got servers?
 - Use ZEO
 - Handle requests sooner

1.4.50 Strategies II

- Got memory?
 - Increase ZODB and/or ZEO disk caches

1.4.51 Strategies III

- Got stable functions?
 - eg, tax calculator
 - Cache them forever with RAMCache

1.4.52 Strategies IV

- Got time-cachable functions?
 - eg, weather retrieval
 - Cache them for a while with RAMCache

1.4.53 Strategies V

- Got security-constrained functions?
 - Cache them with RAMCache by user

1.4.54 Strategies VI

- Use CacheFu to handle the rest

1.4.55 Getting More Information

- CacheFu README.txt
- CacheFu “Audiences” documentation
 - New Caching Manual coming out
- O’Reilly Squid book

1.4.56 How You Can Help

- Help support the product
- Help with documentation
- Blog about its coolness

1.4.57 Thanks!

- Questions?
`joel@joelburton.com`
- Handouts
`http://plonebootcamps.com/resources`