Optimizing Drupal Performance

Benchmark Results

February 2010

Table of Contents

Overview	3
Test Environment	3
Results Summary	4
Configurations and Test Details	8
Bytecode Caching	12
Improving Drupal Code with Partial Caching	13
Full Page Caching	14
Super-fast Full Page Caching	15
Performance Optimizations Not Tested	16
About Zend	17
About Acquia	17

Overview

This paper presents the results of performance benchmarks for comparing several common PHP runtime environments and configurations. The application used for benchmarking is Drupal, the open source social publishing platform.

Tested runtime environments included plain PHP, Zend Server (www.zend.com/products/server), and open-source optimizers, such as APC and WinCache.

For the purpose of this benchmark, a simple Drupal-based website was set up and tested in various commonly used Windows and Linux configurations.

Test Environment

The tests were run on dual 2GHz AMD machines (3988.97 bogomips) with 2GB of memory.

Linux Setup

For testing performance on Linux, Fedora Linux 11 running Apache version 2.2.11 was used with the following configuration:

StartServers	50
MinSpareServers	50
MaxSpareServers	100
ServerLimit	256
MaxClients	256

The following versions were used:

PHP version 5.2.10 (from the Zend Server installation)

APC version 3.1.3p1

WinCache version 1.0.1012.0

Windows Setup

For testing performance on Windows, Windows Server 2008 R2 Enterprise running IIS was used. As in the Linux test, PHP version 5.2.10 (from the Zend Server installation) was used.

Test Setup

The test load was generated by 2 client machines running wcat over 1GB Ethernet, and simulating 25 client sessions each (50 parallel connections overall). The test was performed with a 3-minute warm-up (no data collection), a 3-minute duration (data is collected) and a 1-minute cooldown.

A test workload script was used to ensure the results were uniform for different Drupal parts.

Script URLs:

/drupal/ (home page)

/drupal/?q=content/hello-blog-entry (blog entry with 2 comments)

/drupal/?q=node/1 (top node)

/drupal/?q=node/2 (child node)

/drupal/?q=archive/200803 (archive containing links to above 3 nodes)

/drupal/?q=comments/recent (showing 4 comments)

All of the nodes/comments had about 1-1.5Kb of text.

Another script for Drupal forum URLs was used:

/drupal/?q=forum/1

/drupal/?q=content/forum-topic-1

/drupal/?q=content/forum-topic-2

These URLs gave results that were very close to those of running the first script. Therefore, they will not be addressed separately.

Results Summary

Performance improvements for the Drupal setup were tested using the following technologies:

1. Plain PHP 5.2.10
2. APC bytecode caching
3. APC bytecode caching + shared memory caching
4. Zend Server bytecode caching (Optimizer+)
5. Zend Server bytecode caching + shared memory/disk caching
6. Zend Server full-page caching

Drupal was also tested with "Aggressive" and “Normal” caching settings. Aggressive caching yielded better performance results but because some Drupal modules were not compatible with it, both modes were tested.

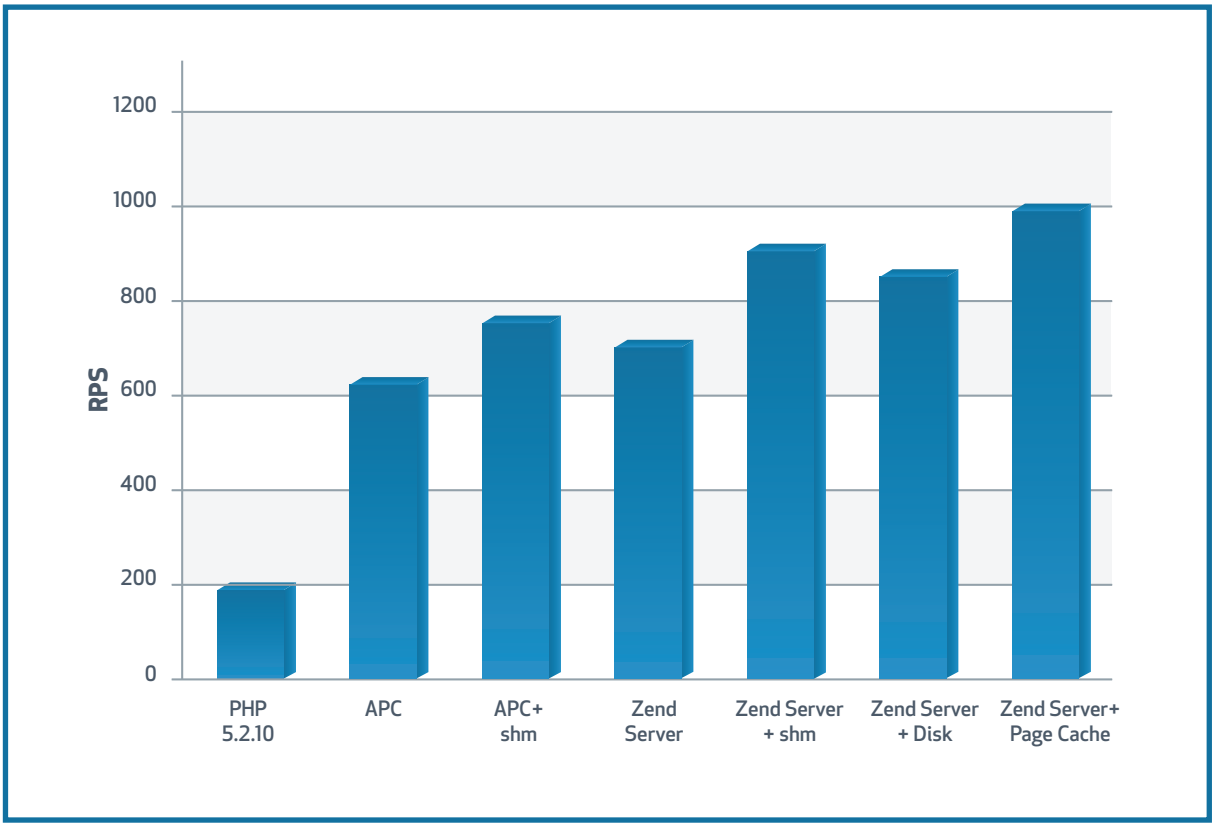
On Windows WinCache was used instead of APC due to the lack of a stable APC version for Windows. Since WinCache does not have a shared memory cache module, these tests were not performed on Windows.

Linux & Aggressive Caching

The following results show the number of Requests Per Second (RPS) for each of the Linux configurations, with Drupal caching set to "Aggressive".

Note: The higher the number of requests, the better the performance.

	RPS	%PHP	%APC	%Zend Server
PHP 5.2.10	192	100%		
APC	629	328%	100%	
APC+shm	741	386%	118%	
Zend Server	706	368%	112%	100%
Zend Server+shm	891	464%	142%	126%
Zend Server+disk	855	445%	136%	121%
Zend Server+page cache	988	515%	157%	140%

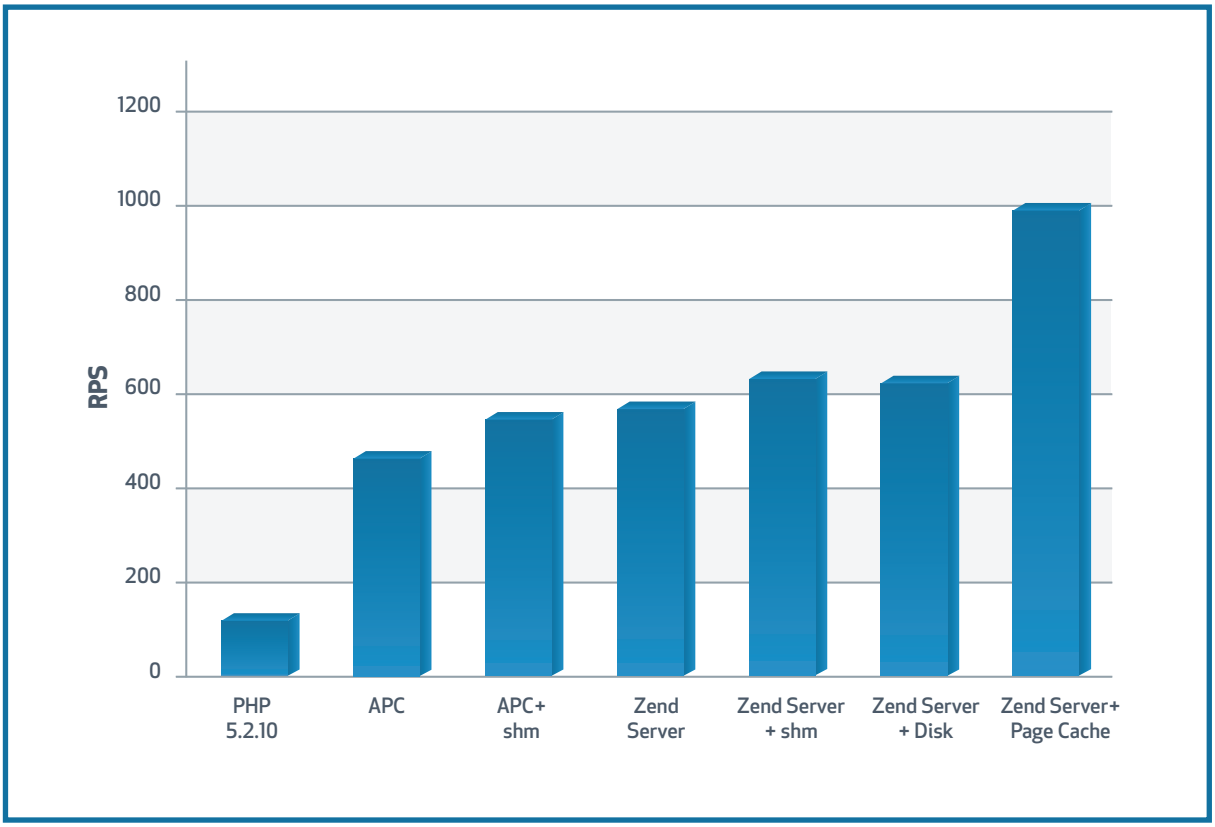


Linux & Normal Caching

Following are the Linux test results with Drupal caching set to "Normal".

Here too, the higher the number of requests, the better the performance.

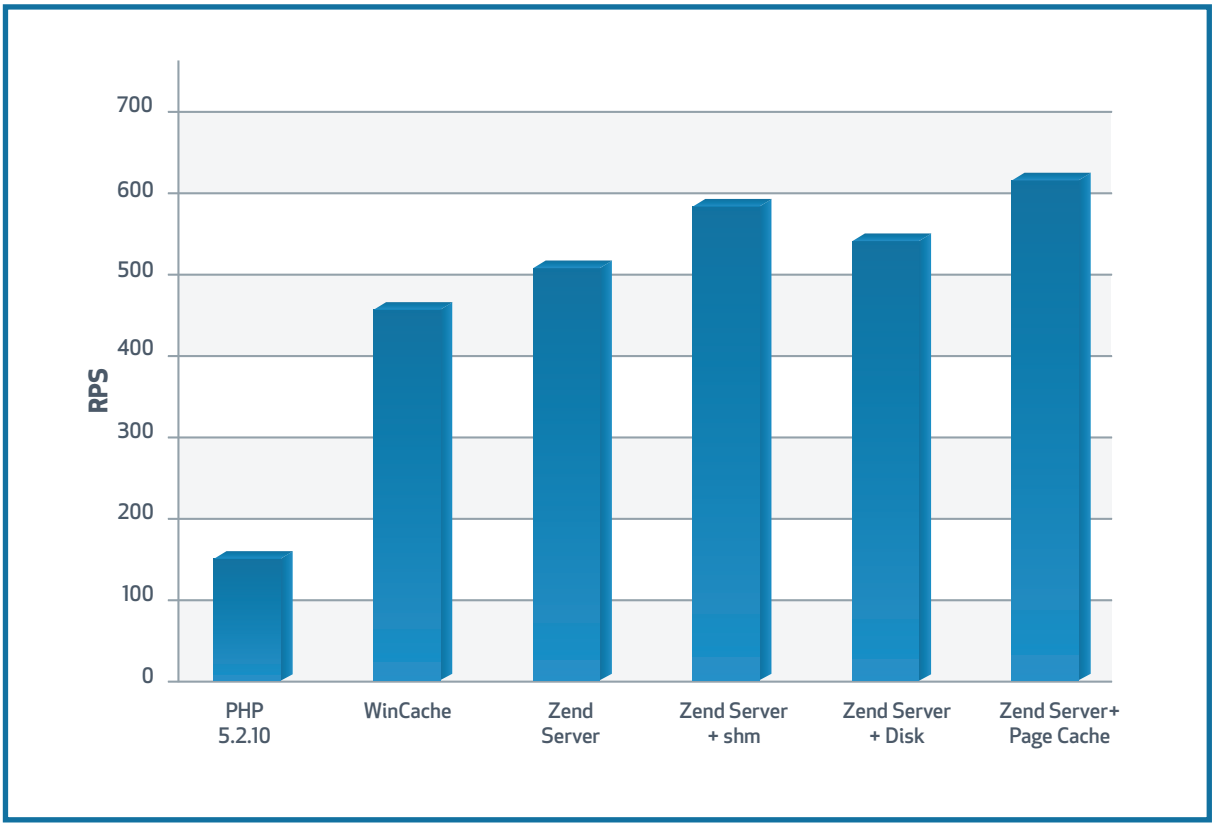
	RPS	%PHP	%APC	%Zend Server
PHP 5.2.10	126	100%		
APC	475	377%	100%	
APC+shm	534	424%	112%	
Zend Server	564	448%	119%	100%
Zend Server+shm	647	513%	136%	115%
Zend Server+disk	629	499%	132%	112%
Zend Server+page cache	987	783%	208%	175%



Windows & Aggressive Caching

Following are the results of the test conducted on Windows Server 2008 with Drupal caching set to "Aggressive".

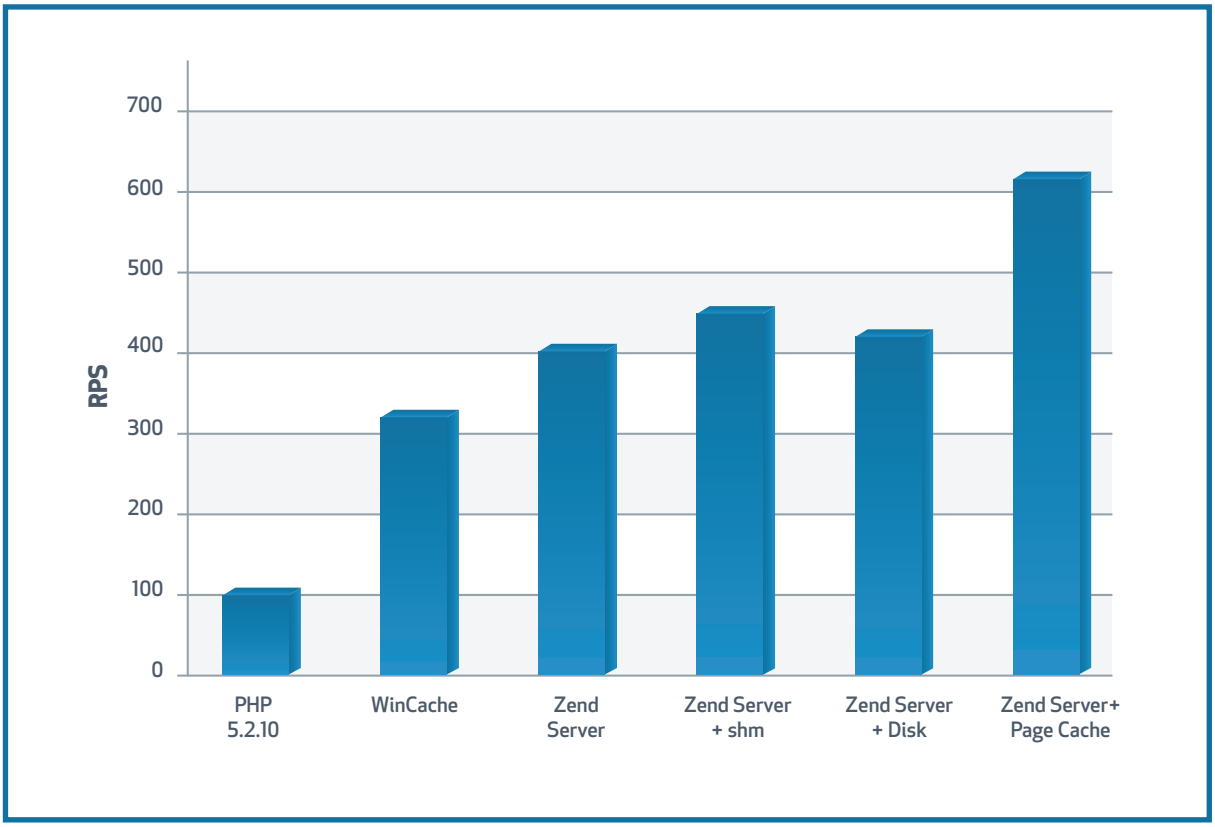
	RPS	%PHP	%WC	%Zend Server
PHP 5.2.10	159	100%		
WinCache	465	292%	100%	
Zend Server	512	322%	110%	100%
Zend Server+shm	584	367%	126%	114%
Zend Server+disk	545	343%	117%	106%
Zend Server+page cache	624	392%	134%	122%



Windows & Normal Caching

Following are the results of the test conducted on Windows Server 2008 with Drupal caching set to "Normal".

	RPS	%PHP	%WC	%Zend Server
PHP 5.2.10	105	100%		
WinCache	323	308%	100%	
Zend Server	402	383%	124%	100%
Zend Server+shm	442	421%	137%	110%
Zend Server+disk	421	401%	130%	105%
Zend Server+page cache	628	598%	194%	156%

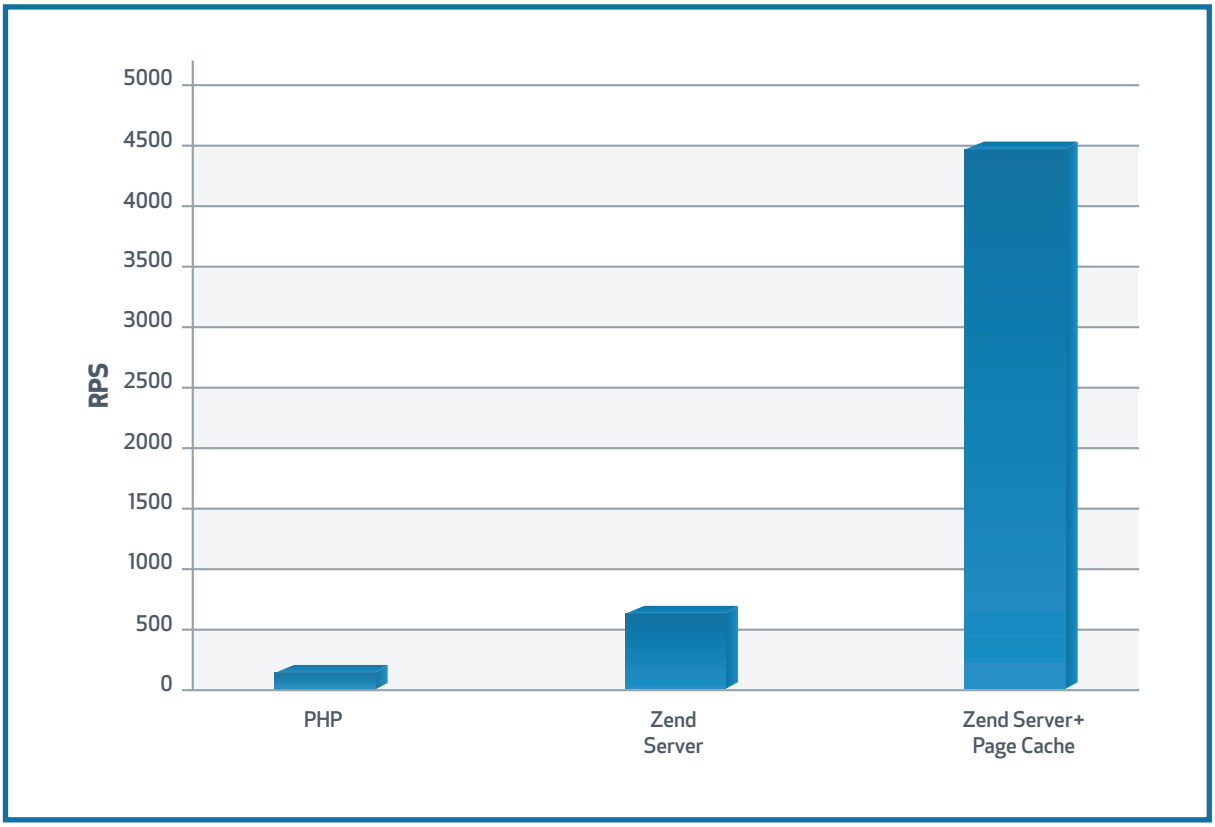


Super-fast Page Caching

Following are the results of a test conducted with Pressflow, which includes backported functionality from Drupal 7 that allows for faster full page caching (see a detailed explanation in the following section). The scenario has only been tested only on Linux.

Using full page caching resulted in performance equivalent to that of static HTML – an astounding 26(!) times faster than plain PHP.

	Linux	%PHP	%Zend Server
PHP	171		
Zend Server	758	443%	
Zend Server + page cache	4456	2605%	587%



Configurations and Test Details

The following is a detailed explanation of configurations and code changes that were done for each test.

Plain PHP Setup

The PHP settings used were mostly defaults with the following changes:

realpath_cache_size=256k

cgi.check_shebang_line=0

display_errors=0

realpath_cache_ttl=12000

Drupal Modules

Drupal 6.13 was used with the following modules installed and enabled:

Ajax, Ajax plugin – comment, Ajax plugin - disable_redirect, Ajax plugin – thickbox, Ajax plugin – wysiwyg, Ajax ui, Content, Content Copy, Content Permissions, Fieldgroup, ImageField Tokens, Location CCK, Node Reference, Number, Option Widgets, Text, User Reference, Aggregator, Blog, Blog API, Book, Color, Comment, Contact, Forum, Help, Menu, OpenID, Path, Ping, Poll, Profile, Search, Statistics, Taxonomy, Tracker, Trigger, Update status, Block, Filter, Node, System, User, Nodequeue generate, Google Analytics API, ImageAPI, ImageAPI GD2, GMap, GMap Location, GMap Macro Builder, GMap Taxonomy Markers, Location, Location Add Another, Location Fax, Location Phone, Location Search, Node Locations, User Locations, Nodequeue, Smartqueue taxonomy, Advanced help, Automatic Nodetitles, Global Redirect, Integrated Metatags, Integrated Metatags - CCK, Meta tags, Moderation, Nice Menus, Page Title, Path redirect, Pathauto, Public Preview, Secure Pages, Taxonomy Manager, Token, Token actions, TokenSTARTER, Transliteration, Vocabulary Index, SEO Friend, Wysiwyg, Views, Views exporter, Views UI, XML sitemap, XML sitemap engines, XML sitemap node, XML sitemap taxonomy, XML sitemap user.

Performance Settings

The following settings were used for APC:

apc.shm_size=80

apc.include_once_override=1

apc.lazy_classes=1

apc.lazy_functions=1

The following settings were used for WinCache:

wincache.fcenabled=1

wincache.ocenabled=1

wincache.ocachesize=80

wincache.chkinterval=300

The following settings were used for Zend Server (Zend Optimizer+):

zend_optimizerplus.revalidate_freq=300

zend_optimizerplus.memory_consumption=80

zend_optimizerplus.consistency_checks=0

zend_optimizerplus.fast_shutdown=1

The remaining settings were left at their default values.

Zend Server modules that were not needed for particular tests were not used (e.g. Zend Server application monitoring was not activated).

Drupal was modified to use a persistent MySQL connection (mysql_pconnect). This was done so as to avoid spending an excessive amount of time establishing connections to the database.

The Drupal cache lifetime was set to 10 minutes.

Bytecode Caching

APC (WinCache on Windows) and Zend Server bytecode caching were tested with 80MB of memory (sufficient to handle Drupal data).

Tests showed that on Linux, Zend Server provided 3.7 times the performance improvement, compared to 3.3 times the improvement when using APC, i.e. Zend Server performed 12% better than APC.

On Windows, Zend Server provided better performance in the same order of magnitude:

Results with Drupal "Aggressive" caching (the higher the number, the better):

	Linux	%PHP	%APC	Windows	%PHP	%APC
PHP	192			159		
APC/WC	629	328%		456	287%	
Zend Server	706	112%	112%	512	322%	112%

Results with Drupal "Normal" caching (the higher the number, the better):

	Linux	%PHP	%APC	Windows	%PHP	%APC
PHP	126			105		
APC/WC	534	424%		323	308%	
Zend Server	564	448%	106%	402	383%	124%

Note that performance gains from caching were higher when Drupal was set to "Normal" caching. When this mode was used, parts of the page were cached, thus requiring a larger number of modules for forming the page on each request. This means that more files were being loaded from the disk, hence more time was saved through bytecode caching.

This test did not require any changes to the code.

Improving Drupal Code with Partial Caching

Our next objective was to modify the Drupal code to take advantage of the shared memory caching capability of both APC and Zend Server.

Two files were changed; bootstrap.inc and cache.inc . This was done in order to change the DB-based Drupal cache into memory or a disk-based cache. The scenarios tested involved virtually only reads from the caches, as it is the most frequent mode of operation for the web site. There was only one request per cache expiry time per page that involved writing to the cache.

An attempt was made to preserve the original semantics of the DB cache. However, thorough testing was not done to determine if the semantics of the code matched the semantics of the original Drupal code with regards to expire times, partial cleanups, etc. It is recommended to test the full set of scenarios before using such modifications in production. Only content-generation functionality was tested in this particular case.

Testing was performed with bytecode caching enabled. On Windows, WinCache has no shared memory caching module so it was not tested.

Following are the results yielded for "Aggressive" caching:

	Linux	%PHP	%APC	%Zend Server	Windows	%PHP	%Zend Server
PHP	192				159		
APC/WC	629	328%			456	287%	
Zend Server	706	368%			512	322%	
APC+shm	741	386%	118%				
Zend Server+disk	855	445%		121%	579	364%	113%
Zend Server+shm	891	464%		126%	584	367%	114%

Test results showed that the use of more efficient caching provided the expected performance gains:

- 18% improvement for APC cache (3.9x overall)
- 21% improvement for Zend Server disk cache (4.5x overall)
- 26% improvement for Zend Server memory cache (4.6x overall)

Here again, performance gains on Windows were slightly lower than on Linux, especially for a disk-related cache. It is assumed that this is due to a slower file system.

Following are the results yielded for "Normal" caching:

PHP	126				105		
APC/WC	475	377%			323	308%	
Zend Server	564	448%			402	383%	
APC+shm	534	424%	112%				
Zend Server+disk	629	499%		112%	442	421%	110%
Zend Server+shm	647	513%		115%	421	401%	105%

As expected, with "Normal" caching selected, performance gains from caching were lower, since multiple elements were cached separately, and more code was being executed. On Windows, since the file system is slower, disk cache appeared to provide the smallest performance improvement compared to the default Drupal DB cache.

Full Page Caching

To measure the impact of full page caching (available in the commercial version of Zend Server), a scenario was created that caches non-authenticated access to Drupal, while leaving authenticated content uncached (since it might differ from one user to another). User and login pages were also excluded from caching, though they were not featured in the test scenario. The GET query string was chosen as the cache key, and expiry time was set to 30 seconds.

The result showed an overall 5.1 times improvement compared to plain PHP and about a 40% improvement compared to using only bytecode caching:

	Linux	**%PHP**	**%Zend Server**	**Windows**	**%PHP**	**%Zend Server**
PHP	192			159		
Zend Server	706	368%		512	322%	
Zend Server+page cache	988	515%	140%	624	392%	122%

Improvements appeared to be more modest on Windows, probably because page cache uses disk as storage and the Windows file system is slower.

The results of using page caching were very similar for "Normal" and "Aggressive" caching modes, since cached pages were delivered independently of Drupal caching mechanisms.

Super-fast Full Page Caching

In the tested page caching scenario, a PHP session variable was used to check if the page can be cached. Unfortunately, this requires PHP to start up and for parts of the page to potentially execute until the session is started and its variables become available.

Based on recommendation from Dries Buytaert, the original author of Drupal, additional tests were conducted with Pressflow (http://bit.ly/73Acph) – an API-compatible version of Drupal that includes performance backports from the yet-unreleased Drupal 7 (due to be released during 2010). One of the new features added to Pressflow is a COOKIE variable that can be used to determine whether the current page is being rendered for authenticated access or not. This allows for faster full page caching, completely eliminating any overhead associated with executing PHP for non-authenticated pages. The scenario has only been tested only on Linux.

	Linux	%PHP	%Zend Server
PHP	171		
Zend Server	758	443%	
Zend Server+page cache	4456	2605%	587%

Effectively, using full page caching based on a GET variable results in performance equivalent to that of static HTML – an astounding 26(!) times faster than plain PHP. While full page caching cannot always be applied, using it where possible can make a remarkable difference. If this approach is implemented, it is important to configure Zend Server so that it will not cache pages that include user-specific content.

Performance Optimizations Not Tested

Memcached

One of the best ways to improve caching performance, especially in clustered setups, is to use memcached. Performance gains of memcached will probably be lower than that of shared memory caching, but clustered setups can benefit more from it.

Boost

Boost is a Drupal module that provides static page caching through mod_rewrite and cron. Similarly to Zend Server's full page caching, Boost is useful only for pages accessed anonymously. In contrast to Zend Server, Boost does not retain custom HTTP headers that may be set by Drupal modules.

Optimize DB Queries

Drupal code uses the UTF-8 character set for a DB connection on every request. Proper configuration of the database, client, and table data may eliminate such need and thus reduce DB access time.

Optimize Variables

Currently, Drupal variables are being read and written as one block, which can mean slowdown if this is done more than once. With proper individual caching this can be optimized, though the impact of this optimization is not certain.

CAPTCHA

Captcha modules have a very significant performance penalty because they involve time-consuming image generation upon every request, and are not cached. This problem may be addressed by using partial page caching, by pre-generating images, or by using services like reCaptcha or solutions like Anti-Captcha.

About Acquia

Acquia helps organizations of all sizes build social publishing websites quickly, easily and with a lower total cost of ownership by leveraging Drupal, the open source social publishing platform that blends content and community. Our products, services and support enable companies to leverage the power, technical innovation and economic value of Drupal while simplifying the experience, removing the complexity and minimizing the risk. Please visit http://www.acquia.com and download Acquia Drupal, the completely free Drupal distribution, at http://acquia.com/downloads.

About Zend Technologies

Zend Technologies, Inc., the PHP Company, is the leading provider of products and services for developing, deploying, and managing business-critical PHP applications. PHP runs 35 percent of the world's Web sites and has quickly become the most popular language for building dynamic Web applications. Deployed at more than 30,000 companies worldwide, the Zend family of products is a comprehensive platform for supporting the entire lifecycle of PHP applications. Zend is headquartered in Cupertino, California.