Zend_Cache: how to improve the performance of PHP applications

by Enrico Zimuel (enrico.z@zend.com)
Senior Consultant - Zend Technologies

October 31th, PHP Barcelona Conference - 2009

The PHP Company

About me

Enrico Zimuel

- I come from Turin (Italy).
- Senior Consultant at Zend Technologies in Italy.
- Software Engineer since 1996.
- Open source contributor: XCheck, Core XPath
- I was a Research Programmer at the Informatics Institute of the University of Amsterdam.
- ZCE, ZFCE also proficiency in C/C++, Java, Perl, Javascript.
- B.Sc. in Computer Science and Economics

My website: http://www.zimuel.it - **Blog:** http://www.zimuel.it/blog

Summary

- **Keys of performance of a PHP application**
- **The caching mechanism**
- **Zend Framework and Zend_Cache**
- **Using the Zend_Cache to cache data and pages**
- **Benchmarking different caching:** Files, APC, Memcache, Zend Server CE

Performance

- **How to improve the performance of a PHP application or more in general a piece of software?**
  - **Optimize the code** (make the code faster)
  - **Using a caching mechanism** (reduce the amount of code to execute)

What's a cache?

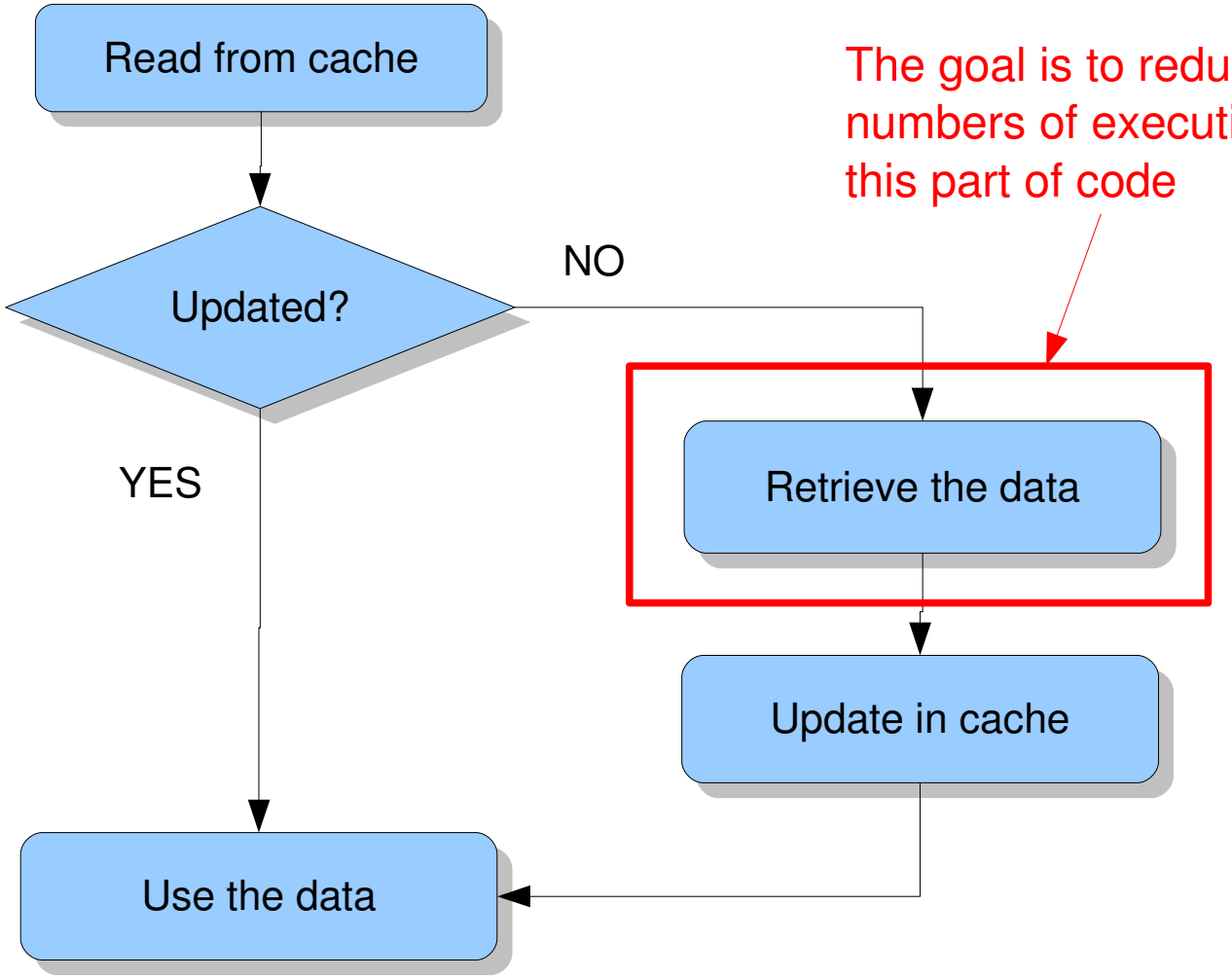
- *In computer science, a **cache** is a **collection of data** duplicating original values, stored elsewhere or computed earlier, where the original data is expensive to fetch (owing to longer access time) or to compute, compared to the cost of reading the cache*

From Wikipedia, the free encyclopedia

Caching: key concepts

- **Unique identifier** (a string), usually named ***key***, used to identify cache records.
- **Lifetime**, how long the cached resource is considered updated.
- **Conditional execution,** that parts of your code can be skipped entirely, boosting performance.

Conditional execution



Caching in PHP

- **In PHP we can use different kinds of cache systems:**
  - **File,** by hand using the fwrite, fread, readfile, etc
  - **Pear::Cache_Lite,** using the files as cache storage
  - **APC,** using the apc extension (apc_add, apc_fetch, etc)
  - **Xcache**, from the lighttpd web server project
  - **Memcache**, using the memcache extension (Memcache::add, Memcache::get, etc)
  - **Zend Server/CE**, using Zend Server API (zend_shm_cache_fetch, zend_shm_cache_store, etc)

Zend_Cache

- **Zend_Cache** is a general class of the **Zend Framework** to cache any kind of data: object, string, array, function, class, page, etc
- **Zend_Cache** uses a *Front class* to access the data and a *Backend class* to manage the data
  - Same methods to access the cache for differents backend
  - You can switch from a backend to another without modify your code
- **It can be used WITHOUT the MVC part of the Zend Framework!**

Zend_Cache_Frontend

- **The core of the module (Zend_Cache_Core) is generic, flexible and configurable**
- **Yet, for your specific needs there are cache frontends that extend Zend_Cache_Core for convenience:**
  - Output
  - File
  - Function
  - Class
  - Pages

Zend_Cache_Backend

- **Zend_Cache has different backends to store the caching data:**
  - Zend_Cache_Backend_File
  - Zend_Cache_Backend_Sqlite
  - Zend_Cache_Backend_Memcached
  - Zend_Cache_Backend_Apc
  - Zend_Cache_Backend_Xcache
  - Zend_Cache_Backend_ZendPlatform
  - Zend_Cache_Backend_ZendServer
  - Zend_Cache_Backend_TwoLevels

A first example: caching a SQL query

```
require_once 'Zend/Cache.php';
$frontendOptions = array(
   'lifetime' => 7200, // cache lifetime of 2 hours
   'automatic_serialization' => true
);
$backendOptions = array(
    'cache_dir' => '/tmp/cache'
);
$cache = Zend_Cache::factory('Core',
                             'File',
                             $frontendOptions,
                             $backendOptions);
...
```

A first example (2): caching a SQL query

```
...
// check if the 'myresult' key is present into the cache
if (!$result = $cache->load('myresult')) {
    // cache miss; connect to the database
    require_once 'Zend/Db.php';
    $db = Zend_Db::factory([...]);
    $result = $db->fetchAll('SELECT * FROM table');
    // update the cache
    $cache->save($result, 'myresult');
}
// use the $result
print_r($result);
```

Caching the output with Zend_Cache

- **Zend_Cache_Frontend_Output** is an *output-capturing* frontend.
- It utilizes output buffering in PHP to capture everything between its **start()** and **end()** methods.
- We can use this cache to store single parts of a page
- Using an MVC architecture we use this caching into the View

Output caching: example

```
require_once 'Zend/Cache.php';
$frontendOptions = array(
   'lifetime' => 30, // cache lifetime of 30 seconds
   'automatic_serialization' => false
);
$backendOptions = array(
    'cache_dir' => '/tmp/cache'
);
$cache = Zend_Cache::factory('Output',
                             'File',
                             $frontendOptions,
                             $backendOptions);
...
```

Output caching (2): example

```
...
// we pass a unique identifier to the start() method
if (!$cache->start('mypage')) {
    // output as usual:
    echo '<h1>Hello world!</h1> ';
    echo '<h3>This is cached ('.date ('H:m:s').')</h3>';
    $cache->end(); // the output is saved and sent to the browser
}

echo '<h3>This is never cached ('.date ('H:m:s').').</h3>';
```

Full page caching with Zend_Cache

- **Zend_Cache_Frontend_Page** is designed to cache a complete page.
- The key of the cache value is **calculated automatically** with $_SERVER['REQUEST_URI'] and (depending on options) $_GET, $_POST, $_SESSION, $_COOKIE, $_FILES.
- Moreover, you have only one method to call, **start()**, because the end() call is fully automatic when the page is ended.

Full page caching with Zend_Cache (2)

- Using an MVC architecture with a front end controller you can build a **centralized cache management** in the bootstrap file
- The Zend_Cache_Frontend_Page comes with a **regexps** param to cache only specific pages:
  - an associative array to set options only for some REQUEST_URI, keys are (PCRE) regexps, values are associative arrays with specific options to set if the regexp matchs on $_SERVER['REQUEST_URI']

Tagging Records

- On the main problem of the cache mechanism is the **invalidation** of cache values
- To update or delete a value in the cache you have to know the **unique id** of the value.
- Build a good key generation system is a challenge.
- The **Zend_Cache** uses a tag system to group together cache values. In this way you can invalidate a set of cache values using tags.

Tagging Records (2)

- When you save a cache with the **save()** method, you can set an **array of tags** to apply for this record:

```
$cache->save($data, 'myKey', array('tagA', 'tagB'));
```

- Then you will be able to clean all cache records tagged with a given tag (or tags).

- At the moment the only backends that support the tag system are: **File, Sqlite, ZendPlatform.** Anyway you can always use tags with TwoLevels backend, we will see it.

Cleaning the cache

- To remove or invalidate in particular cache id, you can use the **remove()** method: $cache->remove('myKey');
- To remove or invalidate several cache ids in one operation, you can use the **clean()** method. For example to remove all cache records :

```
// clean all records
$cache->clean(Zend_Cache::CLEANING_MODE_ALL);

// clean only outdated
$cache->clean(Zend_Cache::CLEANING_MODE_OLD);
```

Cleaning the cache with tags

- If you want to remove cache entries matching the tags **'tagA'** and **'tagB'**:

```
$cache->clean(
    Zend_Cache::CLEANING_MODE_MATCHING_TAG,
    array('tagA', 'tagB')
);
```

- You can clean using different boolean conditions:
  OR = Zend_Cache::CLEANING_MODE_MATCHING_ANY_TAG
  NOT = Zend_Cache::CLEANING_MODE_NOT_MATCHING_TAG

A special backend: TwoLevels

- This backend is an hybrid one. **It stores cache records in two other backends**: a fast one like APC and a "slow" one like File.
- This backend will use the **priority parameter** (given at the frontend level when storing a record) and the **remaining space** in the fast backend to optimize the usage of these two backends.

Benchmarking Zend_Cache backends

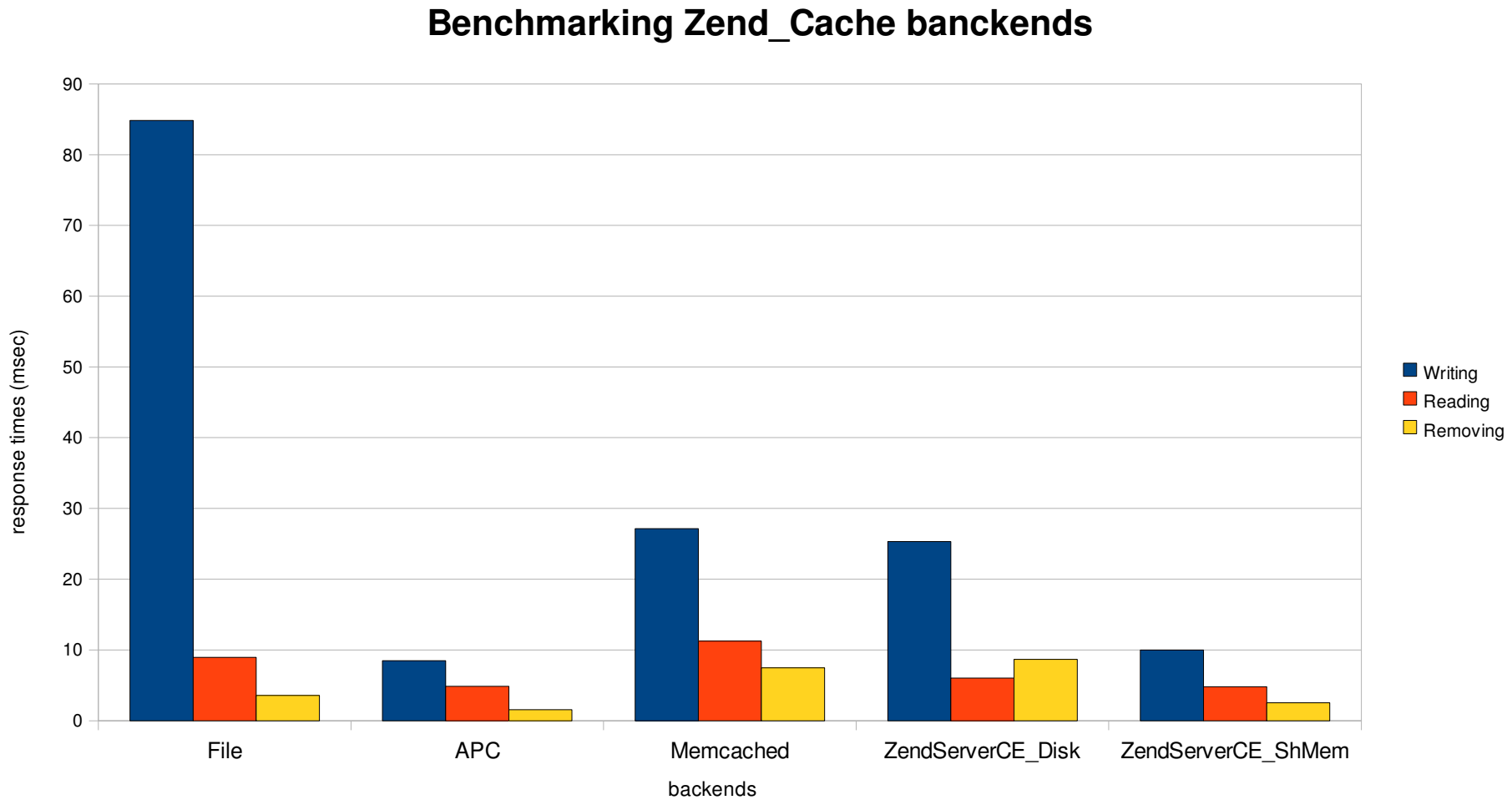
- I provided an experiment to benchmark the **Zend_Cache** using differents backends: File, APC, Memcached, ZendServerCE (disk/ram)
- **The Experiment:**
  I tested the execution times of **writing**, **reading** and **deleting** 100 values from the cache (each value was an array of 100 elements). I run the experiment 10 times and I elaborate the averages of the results.
- **I run the experiment on my laptop:**
  Gnu/Linux Ubuntu 9.04, Kernel 2.6.28, CPU Intel Centrino vPro 2.01Ghz, RAM 2 Gb, HD 250 Gb, Zend Server CE 4.0.5, Apache 2.2.12, PHP 5.3, ZendFramework 1.9.4, Memcached 1.2.2

Benchmarking Zend_Cache: results

Backends	Writing	Reading	Removing
File	84,850	8,964	3,562
APC	8,471	4,846	1,548
Memcached	27,129	11,259	7,467
ZendServerCE_Disk	25,323	6,005	8,681
ZendServerCE_ShMem	9,996	4,785	2,532

All the times are execution times in milliseconds

Benchmarking Zend_Cache: graphic



Caching with Zend_Cache: best practices

- To cache SQL result use the **MD5 of the SQL string** as *key* for the cache value
- Estimate the **lifetime** of the cache in a real environment (it depends on traffic!)
- Always use the **tag** system to cache data
- **Don't delete** a single cache value. It's better to clean using the tag system or clean all the cache!
- On a single server the faster cache systems are **APC** and **ZendServer/CE** using the memory (ShMem)

Questions?

Thank you!

For more info: http://www.zend.com
http://framework.zend.com