Caching Best Practices

By: Eli White

CTO & Founding Partner:
musketeers.me

Managing Editor & Conference Chair:
php[architect] - phparch.com

eliw.com - @eliw

php[architect]

Caching

…why bother?

Why Caching?

Simply put, it's necessary to relieve the load from more expensive tasks, such as database and API access.

No popular application can survive serving completely dynamic data upon every request, so they all cache:

Facebook
Twitter
Flickr
eBay

WordPress.com
Wikipedia
Craigslist
Youtube

It's often said that there are two hard things in computer programming: cache invalidation, naming things, and off-by-one errors.

But my Data Changes!

The usual response to why someone can't cache.

Yet, Facebook can do caching. So how do they do it?

The trick is in caching small pieces of data that change less often, invalidating when it changes, and/or living with stale data for short periods of time.

Opcode Caches

…let's clear the air here first.

Not this talk

Not actually what this talk is focused on however…

PHP recompiles your program every time that it is
run into a machine readable language, called opcodes.

An opcode cache stores the compilation in memory and
just re-executes it when called a second time.

Make sure you are running one!

Opcode Cache Options

APC (Alternative PHP Cache) - http://pecl.php.net/package/APC

- Works on PHP 4.0+ – Windows, Linux, Mac
- Also includes a user cache (more on this later)

OPcache - http://php.net/opcache

- Works on PHP 5.2+ – Windows, Linux, Mac
- Bundled with PHP as of 5.5

WinCache - http://www.iis.net/expand/WinCacheForPHP

- Works on PHP 5.2+ – IIS/Windows only

Caching Methodologies

…so how we do this?

Caching Methodologies

Whole Page

- Literally caching the entire generated HTML page.
- Best left to well configured proxies, not flexible.

Partial Page

- Caching snippets of your HTML (widgets) that are static.
- Allows a broader cache, with a little more flexibility.

Database Queries

- A simple method is just to wrap all DB queries in cache.
- Simpler, more re-usable, but not as flexible as...

Biggest-Smallest Reusable Object

- Caching processed data in it's most re-usable form.
- More complicated to code, but most flexible.

Briefly: MySQL Query Cache

Yes this cache exists and can give you some gains.

Do not rely upon it, as it's very specific to the query cached, and each database slave has its own.

Also any update invalidates all caches for that table. Update one user and invalidate all user caches.

Briefly: Nginx, PHP-FPM & Varnish

Nginx is a very fast webserver, and can act as a whole page cache & proxy. Often configured on the same server to serve static files, while passing on PHP requests to Apache.

Through the use of PHP-FPM (a standalone process running PHP), you can even configure Nginx to host PHP directly.

Varnish is a dedicated caching proxy, often used instead of Nginx for that purpose as well.

Cache Invalidation Strategy

time expiration

- Easiest & most commonly used.
- Simply set a time limit for how long the cache will last.
- Next request for data after expiration, will reload cache.

NOTE: Can cause cache hammering, where multiple rapid requests cause duplicate reload requests. You can attempt to fix this via using locking or other tricks, but 99% of the time, it's OK to just ignore this problem.

Cache Invalidation Strategy

update invalidation

- Proactive approach.
- Any time that you know you have updated the underlying data, expire any cache that included it.
- Alternatively, don't just expire but update the cache.

Cache Invalidation Strategy

pre-generation

- Used by some of the heaviest traffic companies.
- Usually used in along one of the other strategies.
- Have scripts that pre-generate your cache objects before they are even requested.
- Can happen upon code push, on a set schedule, or based upon site actions (such as a user logging in).

Caching Code Solutions

…what are our options?

Common Cache Solutions

APC (or APCu)

- APC provides an in-memory user variable cache
- APCu provides just that feature, if running OPcache
- Stores data local to that webserver only, not shared

Memcached

- A service that handles in-memory cache storage & retrieval
- Is designed to run as a cluster to distribute load & failures

Filesystem

- Store output as text files onto your local filesystem
- Please Please Please NO

APC

…understand some code samples

APC Common Functions

Full Documentation: http://php.net/apc

- Store data in APC under a key:
```
bool apc_store(string $key, mixed $var [, int $ttl = 0])
```
- Similar, but fail if key already existed:
```
bool apc_add(string $key, mixed $var [, int $ttl = 0])
```
- Fetch a value from the cache:
```
mixed apc_fetch(mixed $key [, bool &$success])
```
- Delete an existing key (invalidate the cache):
```
mixed apc_delete(string $key)
```


APC Example

A common use of APC is to store parsed configuration files from your codebase that change rarely.

```
class Config
{
    private $_config;

    public function __construct() {
        if (!($this->_config = apc_fetch('config.parsed'))) {
            $parsed = simplexml_load_file('/path/to/config.xml');
            $this->_config = $parsed;
            apc_store('config.parsed', $parsed);
        }
    }
}
```


Memcached

…understand some code samples

Memcached Libraries

There are two libraries in PHP to access Memcached.

memcache - http://php.net/memcache

- The original library, commonly available
- Limited in its functionality, especially in cluster work

memcached - http://php.net/memcached

- Updated library written to overcome some needs
- Much more complete feature set
- Highly recommended at this point over the original

Configuring Memcached

To start using it, you need to add your server configuration (one at a time, or in bulk) via:

```
public bool Memcached::addServer(string $host, int $port [, int $weight=0])
public bool Memcached::addServers(array $servers)
```

The `$weight` parameter can be used to unevenly distribute the keys across your server pool:

```
$cache = new Memcached();
$cache->addServers([
    ['cache1.example.com', 11211, 50],
    ['192.168.3.1', 11212, 100]
]);
```


Managing Memcached

Expirations can be seconds into the future or a unix timestamp

- Store data under a key:
```
public bool set(string $key, mixed $value [, int $expiration])
```
- Similar, but fail if key already existed:
```
public bool add(string $key, mixed $value [, int $expiration])
```
- Similar, but fail if key doesn't already exist:
```
public bool replace(string $key, mixed $value [, int $expiration])
```
- Delete an existing key (invalidate the cache):
```
public bool delete(string $key)
```
- Fetch a value from the cache:
```
public mixed get(string $key)
```


Simple Example

```php
$value = "I am caching this\n";
$cache->set('my.key', $value, 60*5); // Save this for 5 minutes
$result = $cache->get('my.key');
echo $result; // Works, as long as Memcached was configured

sleep(60*5 + 1); // Past the expiration
if ($result = $cache->get('my.key')) {
    // This won't happen, result will be FALSE as it fails
} else {
    echo "Failed, no cached data\n";
}
```


Manipulating Key Values

If your value is a string, you can prepend/append to it:

```
public bool append ( string $key , string $value )
public bool prepend ( string $key , string $value )
```

If an integer, you can increment or decrement it:

($initial_value will be set if the key doesn't exist yet)

```
public int increment ( string $key [, int $offset = 1
     [, int $initial_value = 0 [, int $expiry = 0 ]]] )

public int decrement ( string $key [, int $offset = 1
     [, int $initial_value = 0 [, int $expiry = 0 ]]] )
```


Read Through Cache Callback

A 2nd parameter to `get()` can be passed in: a callback that will be used if the key cannot be found and will be used to generate/store/return the value atomically.

```
$user = $cache->get('user.1234', function ($memcache, $key, &$value) {
    $userid = substr($key, 5);
    $value = new User($userid); // Assuming a User class
    return true;
});
```

If you return `true`, the `$value` is `set()` for you with unlimited expiration. If you don't want this, you need to manually make the `set()` call yourself, and return false.

Check and Set Operations

It's possible to only set a key's value, if it hasn't changed since reading it, allowing cache protection.

```
public mixed get(string $key [, callable $cache_cb [, float &$cas_token]])
public bool cas(float $cas_token , string $key , mixed $value [, int $expiration])
```

Provide a variable to be filled with an access token, then pass the token into `cas()` when ready to update:

```
$token = null;
$user = $cache->get('user.1234', null, $token);
/* Perform a bunch of tasks, decide to resave the user cache */
$cache->cas($token, 'user.1234', $user, 60*5);
/* 'user.1234' will only be updated if no other process changed it */
```


Asynchronous Fetching

For extra performance, you can issue a request for multiple keys via `getDelayed()`, then read the values, one at a time with `fetch()`, or all at once with `fetchAll()`, when ready.

```
public bool getDelayed (array $keys [, bool $with_cas [, callable $value_cb ] ])
public array fetch ( void )
public array fetchAll ( void )
```

```
$cache->getDelayed(['user.1234', 'user.42', 'user.314']);
/* Do a bunch of other tasks while this runs */
while ($cache->getResultCode() != Memcached::RES_END) {
    if ($user = $cache->fetch()) {
        /* We got the result, process as desired */
    } else {
        usleep(10); // Don't tight loop
    }
}
```


Other Useful Features

Ability to set/get/delete multiple keys at once:

```
public mixed getMulti(array $keys [, array &$cas_tokens [, int $flags]])
public bool setMulti(array $items [, int $expiration])
public bool deleteMulti( array $keys [, int $time = 0])
```

Reset a cache expiry without changing the data:

```
public bool touch(string $key, int $expiration)
```

Invalidate the entire cache at once:

```
public bool flush([int $delay = 0])
```

Still not everything: http://php.net/memcached

Combination Techniques

…getting more advanced now

Database Caching Example

As mentioned, an easy strategy is to just cache the output directly of a database query. For example:

```php
function getUser($userid, PDO $db, Memcached $cache) {
    $key = "user.{$userid}";
    if (!($user = $cache->get($key))) {
        $result = $db->prepare('SELECT * FROM users WHERE id = ?')
                     ->execute([$userid]);
        $user = $result->fetchObject();
        $cache->set($key, $user, 60*5);
    }
    return $user;
}
```


More Generic Example

A function to automatically cache any database result:

(as long as the query given to it only returned 1 row)

```
function cachedSingleRowQuery($query, Array $bind, $expiry,
                              PDO $db, Memcached $cache) {
    $key = 'SingleQuery' . md5($query) . md5(implode(',', $bind);
    if (!($obj = $cache->get($key))) {
        $result = $db->prepare($query)->execute($bind);
        $obj = $result->fetchObject();
        $cache->set($key, $obj, $expiry);
    }
    return $obj;
}
```


Concept: Multi-Layer Cache

Some large sites rely heavily on the idea of a multiple layer cache, where items are cached in multiple spots, each successively faster, but more restricted in use.

A common 3 layer cache technique:

- Store a copy in PHP instance
- Store a copy in APC (quick expiration)
- Store a copy in Memcached (longer expiration)
- Worst case default back to database.

Multi-Layer Cache Example

```
class MultiCache
{
    private $this->_store = [];
    private $this->_db;
    private $this->_memcache;

    public function __construct(PDO $db, Memcached $memcache) {
        $this->_db = $db;
        $this->_memcache = $memcache;
    }

    public function query($query, Array $bind) {
        $key = 'MultiCache' . md5($query);
        if (empty($this->_store[$key])) {
            if (!($this->_store[$key] = apc_fetch($key))) {
                if (!($this->_store[$key] = $this->_memcache->get($key))) {
                    $result = $this->_db->prepare($query)->execute($bind);
                    $this->_store[$key] = $result->fetchObject();
                    apc_store($key, $this->_store[$key], 60*5); // 5 minutes
                    $this->_memcache->set($key, $obj, 60*60); // 1 hour
                }
            }
        }
        return $this->_store[$key];
    }
}
```


Parting Thoughts

…some things to keep you thinking

Caching Discussions

Do not store something in cache that you cannot recreate from another source.

Cache is transient.

Caching Discussions

Beware `FALSE`.

You cannot check for a false value in memcached because it returns false when it can't find the key. If you need to store `NULL` instead.

Caching Discussions

Consider caching more aggressively for anonymous users, than when a user in logged in.

Caching Discussions

Consider having cache expirations be configurable.

Allowing you in times of higher load, to simply serve more stale data, versus going down.

This is a strategy that Twitter uses.

Caching Discussions

Consider adding a configurable prefix to all your keys, so that you can easily invalidate data without a flush.

Biggest-Smallest Reusable

Ensure you cache large chunks of data, so you make fewer lookups, but data is discrete enough that they can be reused.

On a blog, you might cache an array of all comments to a single post, since you will always be displaying them all.

But there are cases (social media) where more queries, for smaller data, is more efficient due to constant re-use.

Questions?

For this presentation & more:
http://eliw.com/

Twitter: **@EliW**

php[architect]: www.phparch.com
musketeers: musketeers.me

Rate this talk!
https://joind.in/16685

php[architect]