Under The Hood

Ilia Alshanetsky
@iliaa

http://joind.in/3789

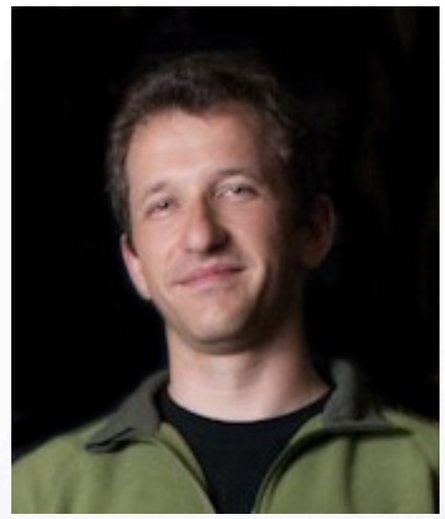
Ilia Alshanetsky

PHP Core Developer

CIO @ Centah Inc.

Occasional Photographer

I like to make things “better”



Before starting on any optimizations, you first MUST understand where your bottleneck is.

The best means of identifying bottlenecks, is via a use of a profiler!

Enter XDebug

Xdebug claim to fame is that it is a PHP debugger, however it also has an awesome built-in profiler, that generates Kcachegrind readable output!

http://xdebug.org/

Xdebug Config

; Enable the profiler

xdebug.profiler_enable="On"

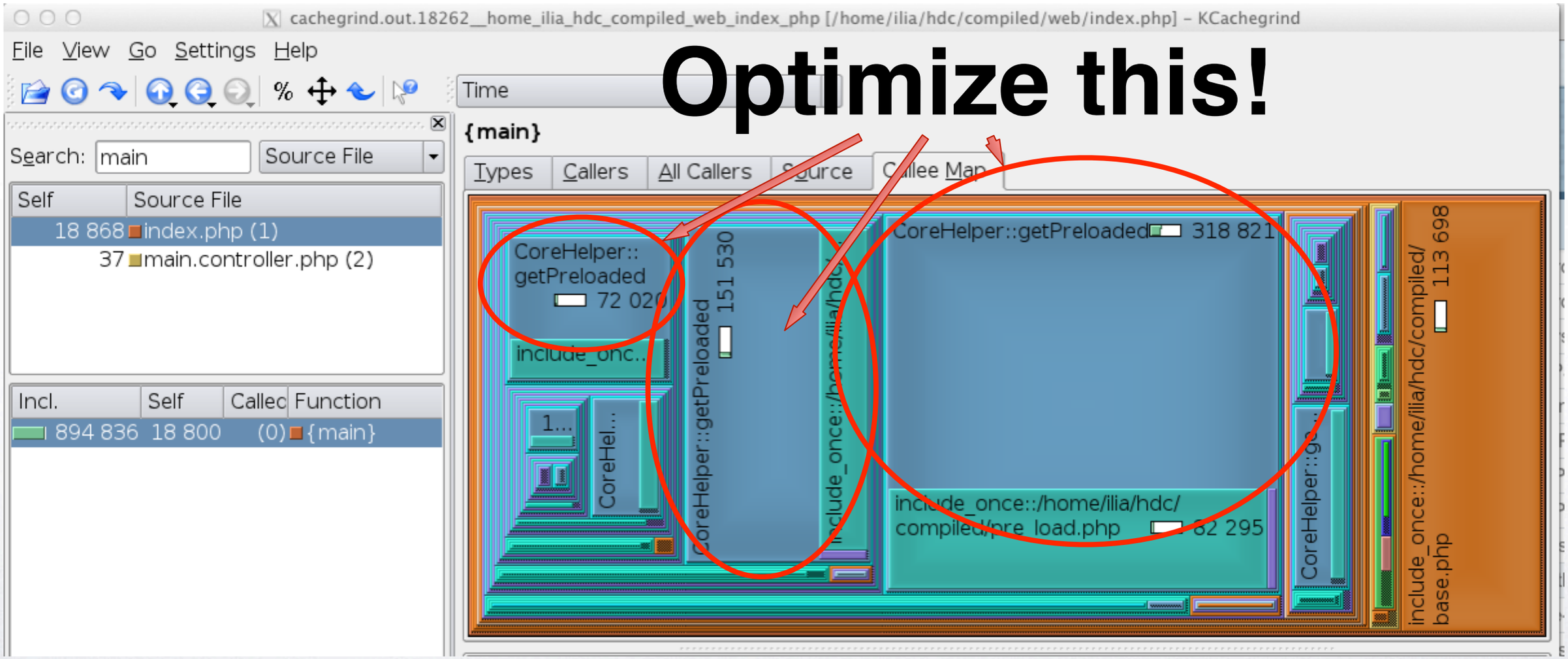
;Write profile data here

xdebug.profiler_output_dir="/tmp"

; Individual profile files (script name, pid, high-res timestamp)

xdebug.profiler_output_name="cachegrind.out.%s.%p.%u"

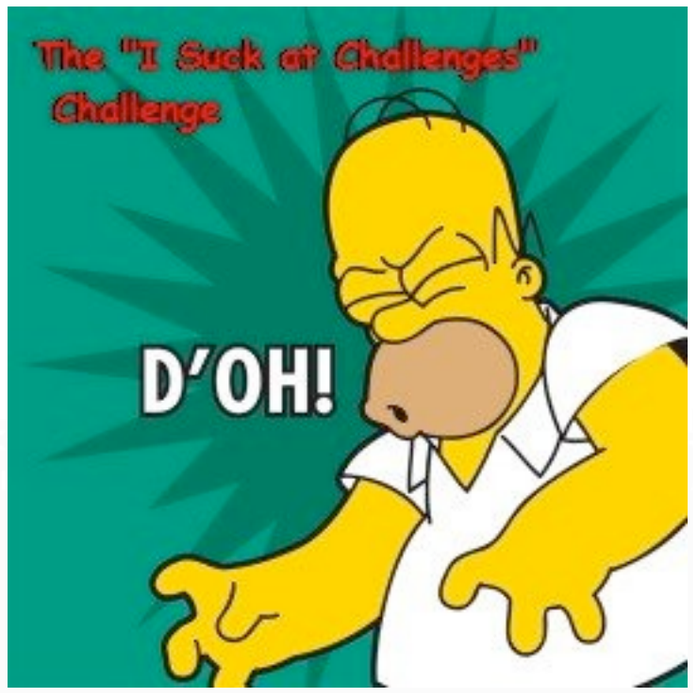
kcachegrind [profile-output]



Often however, your performance issues are of the “Heisenbug” variety.

Wikipedia:

A **heisenbug** (named after the Heisenberg Uncertainty Principle) is a computer bug that disappears or alters its characteristics when an attempt is made to study it.



The challenge is that Xdebug tends to kill performance, making it virtually unusable in production environment.

xhprof

Light weight PHP profiler designed for use in production environment.

- Aggregate run data
- Web interface
- In-Production “sampling” mode

http://pecl.php.net/package/xhprof

http://github.com/preinheimer/xhprof

Profiling

;; Pre-pended to every PHP script (init)

```
auto_prepend_file = /xhprof/external/header.php
```

```
include_once __DIR__ . '/xhprof_lib/config.php');
include_once __DIR__ . '/xhprof_lib/utils/xhprof_lib.php';
include_once __DIR__ . '/xhprof_lib/utils/xhprof_runs.php';
xhprof_enable(XHPROF_FLAGS_CPU + XHPROF_FLAGS_MEMORY);
```

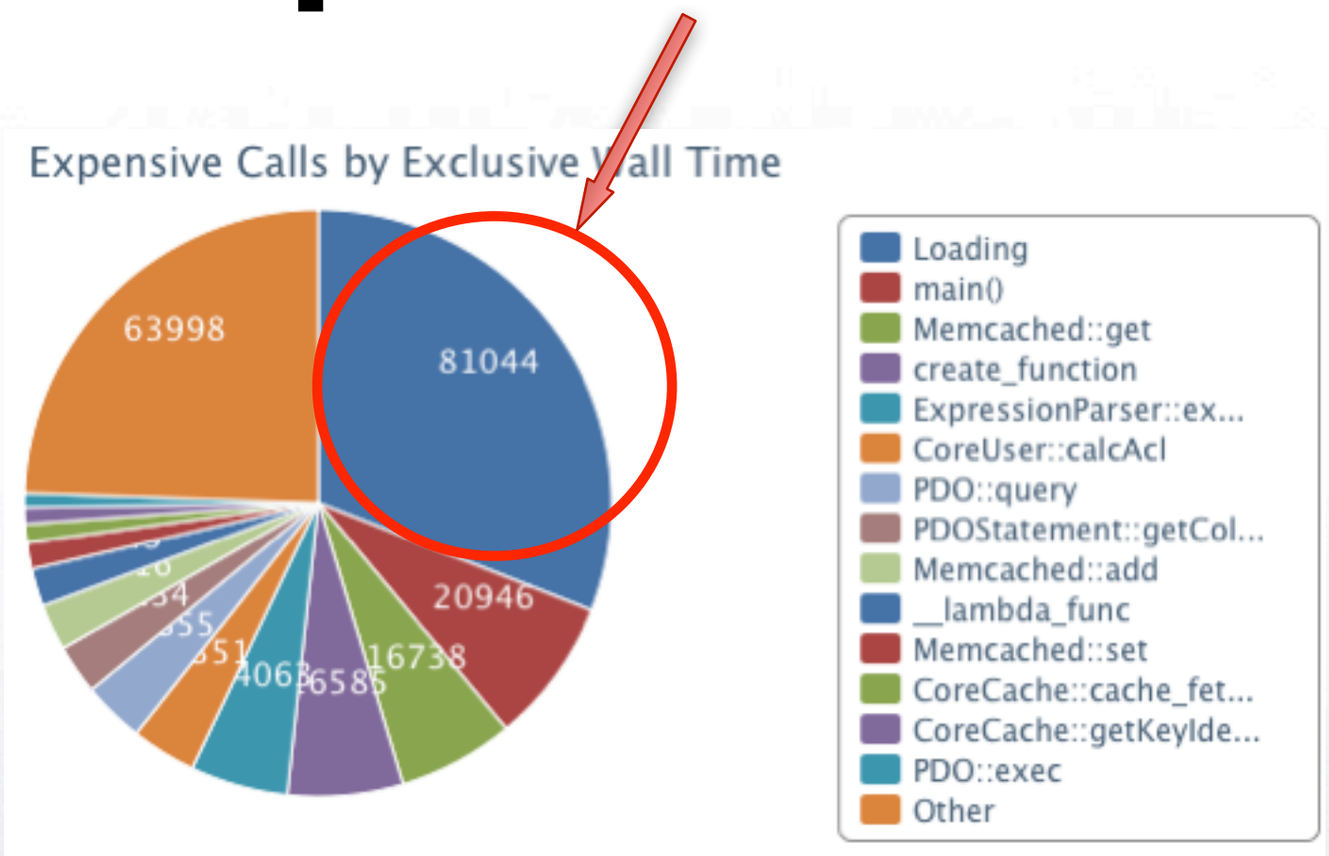
;;Appended to every PHP script (store)

```
auto_append_file = /xhprof/external/footer.php
```

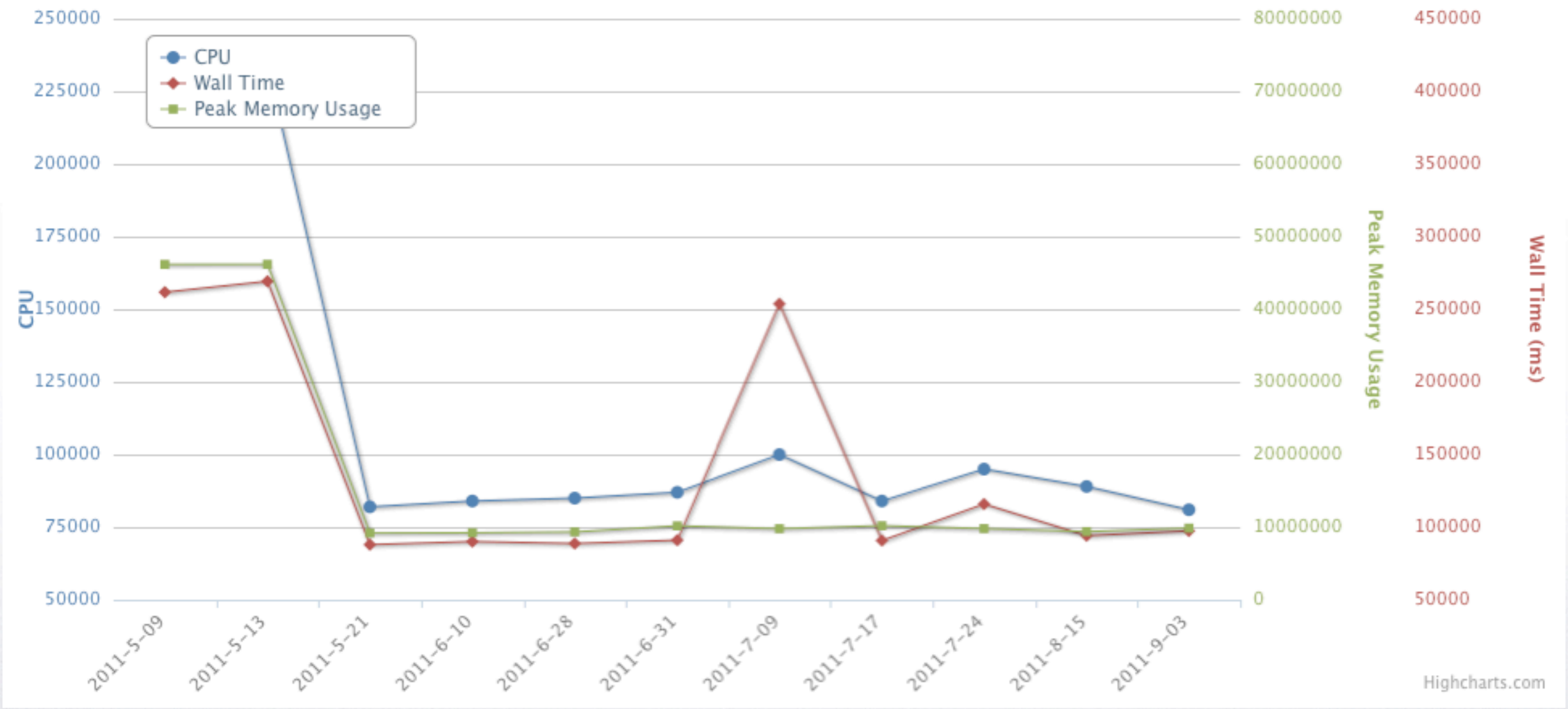
```
$xhprof_data = xhprof_disable();
$xhprof_runs = new XHProfRuns_Default();
$xhprof_runs->save_run($xhprof_data, 'AppName', null, $_xhprof);
```

Profile Output

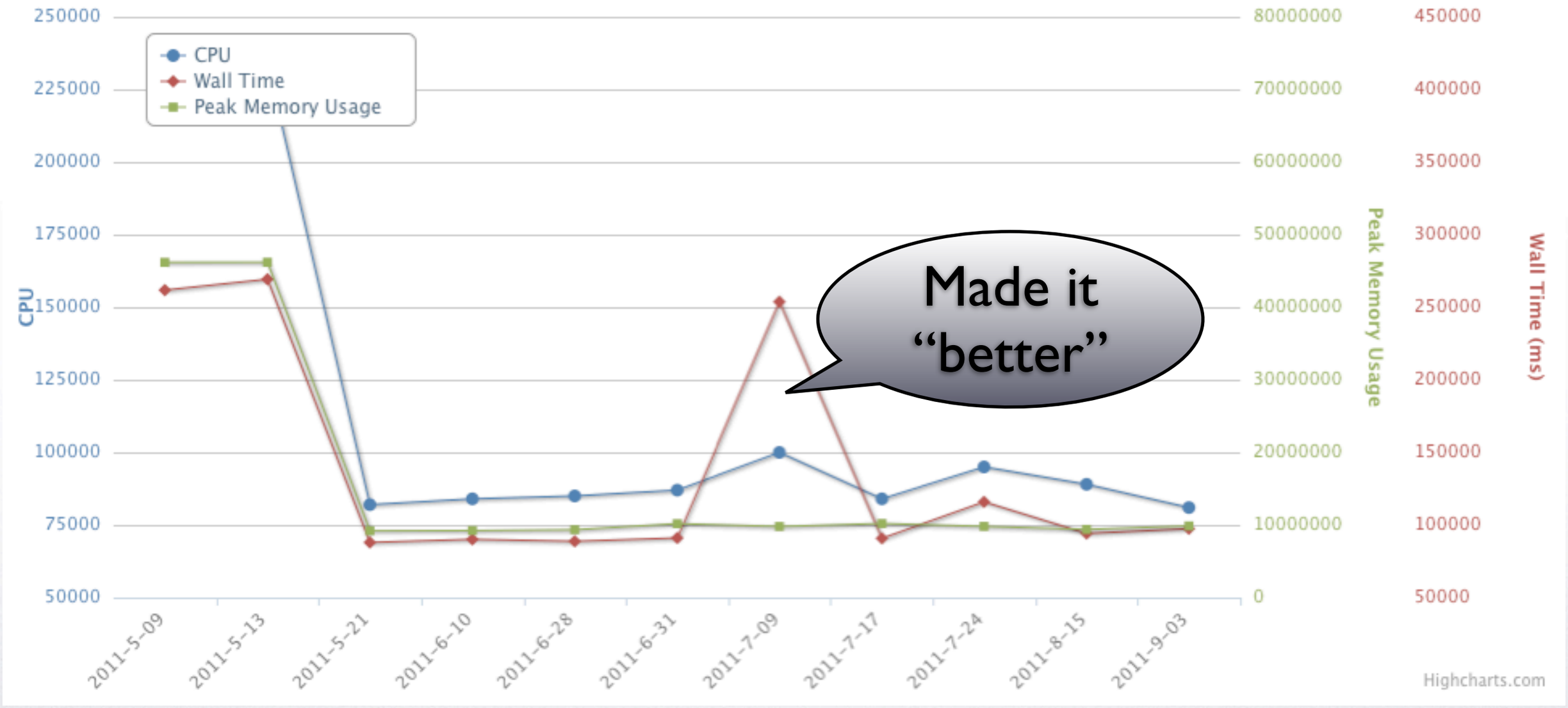
Optimize this!



Code Timeline



Code Timeline



While Xhprof is “nicer” on the system, it will still slow you down by 10-15%.

Which is far from ideal...

But, what if you could determine what to profile first?

```php
class SlowPageDetector {
    const ALERT_THRESHOLD = 0.25; // 1/4 second
    private $start;

    function __construct() { $this->start = microtime(1); }
    function __destruct() {
        $time = microtime(1) - $this->start;
        if ($time > self::ALERT_THRESHOLD) {
            // slower (option A)
            file_put_contents(
"/tmp/slow_pages",
$_SERVER["PHP_SELF"] . " " . $time . "\n",
FILE_APPEND|LOCK_EX);

            // faster (option B)
            apc_add("profile_" . $_SERVER["PHP_SELF"] . "_" .
              $this->start, array(
                "page" => $_SERVER["PHP_SELF"],
                "time" => $time,
                "input" => array($_GET, $_POST)
            ));
        }
    }
}
$a = new SlowPageDetector();
```

File-System Solution

```
$slow = array();
foreach (file("/tmp/slow_pages") as $v) {
    list($page,$speed) = explode(" ", $v);
    if (empty($slow[$page])) {
        $slow[$page] = array(1, (double)$speed);
    } else {
        ++$slow[$page][0];
        $slow[$page][1] += (double) $speed;
    }
}

foreach ($slow as &$v) {
    $v = $v[1] / $v[0];
}

arsort($slow);
print_r(array_slice($slow, 0, 5)); // 5 slowest
```

APC Solution

```
$slow = array();
foreach (new APCIterator("user", "!^profile_!") as $v) {
    if (!isset($slow[$v['value']['page']])) {
        $slow[$v['value']['page']] = array(1, $v['value']['time']);
    } else {
        ++$slow[$v['value']['page']][0];
        $slow[$v['value']['page']][1] += $v['value']['time'];
    }
}

foreach ($slow as &$v) {
    $v = $v[1] / $v[0];
}

arsort($slow);
print_r(array_slice($slow, 0, 5)); // 5 slowest
```

What load can your application handle?

ApacheBench

Usage: /usr/sbin/ab [options] [http[s]://]hostname[:port]/path
Options are:

-n Number of requests to perform
-c Number of multiple requests to make
-C Add cookie, eg. 'Apache=1234. (repeatable)
-t Seconds to max. wait for responses
-A Add Basic WWW Authentication, the attributes are a colon separated username and password.
-i Use HEAD instead of GET

Concurrency Level: 10
Time taken for tests: 18.054 seconds
Complete requests: 1000
Failed requests: 0
Write errors: 0
Total transferred: 494000 bytes
HTML transferred: 0 bytes
Requests per second: 55.39 [#/sec] (mean)
Time per request: 180.543 [ms] (mean)
Time per request: 18.054 [ms]
(mean, across all concurrent requests)
Transfer rate: 26.72 [Kbytes/sec] received

Connection Times (ms)

	min	mean[+/-sd]		median	max
Connect	5	7	0.9	6	12
Processing	76	174	38.0	173	320
Waiting	76	173	38.0	173	320
Total	82	180	38.1	179	327

Percentage of the requests served within a certain time (ms)

50% 179
66% 194
75% 202
80% 208
90% 224
95% 244
98% 271
99% 291
100% 327 (longest request)

While ApacheBench is very convenient, by a virtue of being found on almost any Linux machine, in many instances it is too limited for a “true” test.

For a bit more sophisticated load testing or user activity simulation you may want to consider using Siege.

http://www.joedog.org/index/siege-home

```
CURRENT  SIEGE  CONFIGURATION
------------------------------------
connection:                 close
concurrent users:           25
time to run:                n/a
repetitions:                n/a
socket timeout:             30
delay:                      1 sec
internet simulation:        true
benchmark mode:             false
named URL:                  none
URLs file:                  /tmp/siege_urls.txt
logging:                    true
log file:                   /tmp/siege_ilia.log
resource file:              /home/ilia/.siegerc
```

```
Lifting the server siege...      done.
Transactions:                    98 hits
Availability:                    97.03 %
Elapsed time:                    4.65 secs
Data transferred:                0.20 MB
Response time:                   0.43 secs
Transaction rate:                21.08 trans/sec
Throughput:                      0.04 MB/sec
Concurrency:                     9.17
Successful transactions:         98
Failed transactions:             3
Longest transaction:             3.31
Shortest transaction:            0.23
```

Saidar

A "`top`" like utility for the rest of your system, and a user friendlier "`vmstat`".

http://www.i-scream.org/libstatgrab/

Not all performance issues reside on server side. In many cases an improvement on the server, will not make the user's experience any faster!

To determine why does the application appear to be “slow” to the user, you need a way to measure the user’s experience.

```
<html>
<head>
<script type="text/javascript">
var PageTimerStart = new Date().getTime();
window.onload = function() {
    new Image().src = '/pixel.gif?page_load=' +
        (new Date().getTime() - PageTimerStart) +
        '&url=' + escape(window.location.href);
};
</script>
            /* rest of your code ... */
```

```
awk -F = '
BEGIN { sum=0; total=0; }
/page_load=/ { total++; sum += sprintf("%f\n",$2); }
END { print strftime("%Y-%m-%d"),": ",total,sum,sum/total; }
 ' /var/log/apache2/access_log
```

For a more sophisticated JavaScript user experience profiler, take a look at Boomerang.

http://yahoo.github.com/boomerang/doc/

Slides will be available
at **http://ilia.ws**

Please Provide Feedback
http://joind.in/3789

Ilia Alshanetsky
@iliaa