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]

Optimize this!
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.
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);
Optimize this!
Code Timeline

Made it “better”
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?
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();
$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
$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?
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)

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>mean[+/-sd]</th>
<th>median</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>5</td>
<td>7</td>
<td>0.9</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processing</td>
<td>76</td>
<td>174</td>
<td>38.0</td>
<td>173</td>
</tr>
<tr>
<td></td>
<td></td>
<td>320</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Waiting</td>
<td>76</td>
<td>173</td>
<td>38.0</td>
<td>173</td>
</tr>
<tr>
<td></td>
<td></td>
<td>320</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>82</td>
<td>180</td>
<td>38.1</td>
<td>179</td>
</tr>
<tr>
<td></td>
<td></td>
<td>327</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Percentage of the requests served within a certain time (ms)

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>179</td>
</tr>
<tr>
<td>66%</td>
<td>194</td>
</tr>
<tr>
<td>75%</td>
<td>202</td>
</tr>
<tr>
<td>80%</td>
<td>208</td>
</tr>
<tr>
<td>90%</td>
<td>224</td>
</tr>
<tr>
<td>95%</td>
<td>244</td>
</tr>
<tr>
<td>98%</td>
<td>271</td>
</tr>
<tr>
<td>99%</td>
<td>291</td>
</tr>
<tr>
<td><strong>100%</strong></td>
<td><strong>327</strong>   <em>(longest request)</em></td>
</tr>
</tbody>
</table>
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 @iliao