Profiling PHP Applications

ZendCon 2012 - Santa Clara, US - Oct 23rd, 2012

Derick Rethans - derick@derickrethans.nl - twitter: @derickr

http://joind.in/6871

About Me

Derick Rethans

- Dutchman living in London
- One of the PHP MongoDB driver maintainers
- Author of the PHP debugger tool Xdebug, PHP's Date/Time/Timezone support, and a bunch of other PHP extensions
- I ♥ maps

10gen | the MongoDB company

mongoDB

The Internet is Full of Nonsense

A small selection of PHP optimisation tips I found on the internet

echo is faster than print

Use pre-incrementing where possible as it is 10% faster

```
++$i;
//is faster than
$i++;
```

Methods in derived classes run faster than ones defined in the base class.

A function call with one parameter and an empty function body takes about the same time as doing 7-8 $localvar++ operations. A similar method call is of course about 15 $localvar++ operations.

Not everything has to be OOP, often it is just overhead, each method and object call consumes a lot of memory.

Avoid overusing function calls. Calling a function in PHP is very expensive, so avoid it whenever you can.

The Internet is Full of Nonsense

I can only say one thing:

Do I need to optimise my code?

Before you can optimise anything:

- Find out if things are running slow
- Find out whether it is the code
- Understand your code, application and execution paths
- Find out which parts of the code are slow
- Find out what you can optimise

How does the application perform?

Benchmark the application: siege (httperf)

- check ~/.siegerc and set the logfile setting
- Create a file with urls:http://derickrethans.nl/ http://derickrethans.nl/spatial-indexes-data-sqlite.html http://derickrethans.nl/talks.html http://derickrethans.nl/who.html
- run against your code: siege -c 4 -r 10 -f /tmp/urls.txt

```
Transactions:                     40 hits
Availability:                 100.00 %
Elapsed time:                   9.76 secs
Data transferred:               1.34 MB
Response time:                  0.18 secs
Transaction rate:               4.10 trans/sec
Throughput:                   0.14 MB/sec
Concurrency:                   0.73
Successful transactions:             40
Failed transactions:                  0
Longest transaction:               0.39
Shortest transaction:              0.08
```

Is it my code that is slow?

Multiple possible reasons:

- The database is slow
- There is lots of IO
- Your code is slow
- The system is busy with other things
- And don't forget front-end performance

$ vmstat 1

```
 procs           memory              swap        io      system       cpu
 r  b   swpd   free   buff   cache   si   so    bi    bo   in   cs *|dd1111|us sy|* id wa
 5  0 121248 367520 767824 3080668    0    0 11272     0 1593 5540 *|dd1111|74 22|*  3  0
 5  0 121248 290784 767960 3082980    0    0   268     0 1555 5381 *|dd1111|77 20|*  3  0
 5  0 121248 238340 768132 3084336    0    0  1364 21160 2263 6815 *|dd1111|70 21|*  3  7
 6  0 121248 170772 768300 3087100    0    0  1652     0 1802 8540 *|dd1111|71 25|*  4  0
```

CPU is (close to) fully in use: 74 + 22 ➔ your code is slow.

$ vmstat 1

```
 procs           memory              swap        io      system       cpu
 r  b   swpd   free   buff   cache   si   so    bi    bo   in   cs us sy id *|dd1111|wa|*
 0  1 121248  50776 314796 4156000    0    0 245800     0 3018 5394  1 11 71 *|dd1111|17|*
 1  0 121248  52112 315108 4236692    0    0 243672     0 3107 5552  1 10 71 *|dd1111|18|*
 1  1 121248  50148 315256 4318268    0    0 243096    44 3146 5463  1 11 72 *|dd1111|17|*
 1  1 121248  52120 315484 4396356    0    0 243784     0 3006 5419  1 10 72 *|dd1111|18|*
```

wait is > 10 ➔ IO is the bottleneck.

Application specific tools

mongostat:

```
insert  query update delete getmore command flushes mapped  vsize    res faults  locked db idx miss %     qr|qw   ar|aw
netIn netOut  conn       time
    0   2260      0       0       0       2       0  3.44g  7.19g    73m      0 phpunit:0.0%          0       0|0     1|0
201k   478m     2   08:46:23
    22   1071      0       0       0      43       0  3.44g  7.19g    83m      0 phpunit:1.8%          0       0|0     0|0
3m   233m     2   08:46:24
    0       0      0       0       0       1       0  3.44g  7.19g    73m      0 test2:0.0%          0       0|0     0|0
62b     3k     2   08:46:25
    0       0      0       0       0       1       0  3.44g  7.19g    73m      0 test2:0.0%          0       0|0     0|0
62b     3k     2   08:46:26
    42       0      0       0       0      42       0  3.44g  7.19g    85m      9 phpunit:2.1%          0       0|0     0|1
10m     6k     2   08:46:27
  120      77      0       0       0     122       0  3.44g  7.19g   148m    139 phpunit:70.2%          0       0|0     0|0
31m    19m     2   08:46:28
    34     117      0       0       0      47       0  3.44g  7.19g   129m      0 phpunit:1.2%          0       0|0     0|0
8m    30m     2   08:46:29
    0       0      0       0       0       1       0  3.44g  7.19g   113m      0     .:0.2%          0       0|0     0|0
62b     3k     2   08:46:30
    0       0      0       0       0       1       0  3.44g  7.19g   105m      0     .:0.2%          0       0|0     0|0
62b     3k     2   08:46:31
    0       0      0       0       0       1       0  3.44g  7.19g   105m      0 test2:0.0%          0       0|0     0|0
62b     3k     2   08:46:32
```

Understand your code, application and execution paths

In-code tools:

- Add timing points

External tools:

- Basic overview: inclued
- Deep details: xdebug

Timing Points

Code with analysis in mind:

- Add timing points around specific events
- Check changes over time

eZ Publish:

Symfony:

Timing points:

Checkpoint	Begins at (sec)	Lasts (sec)	Memory at start (KB)	Memory used (KB)
Module start 'setup'	1.5722	1.7874	6,439.2578	5,780.6953
Module end 'setup'	3.3596	2.6235	12,219.9531	8,173.0625
End	5.9831		20,393.0156	

Resources:

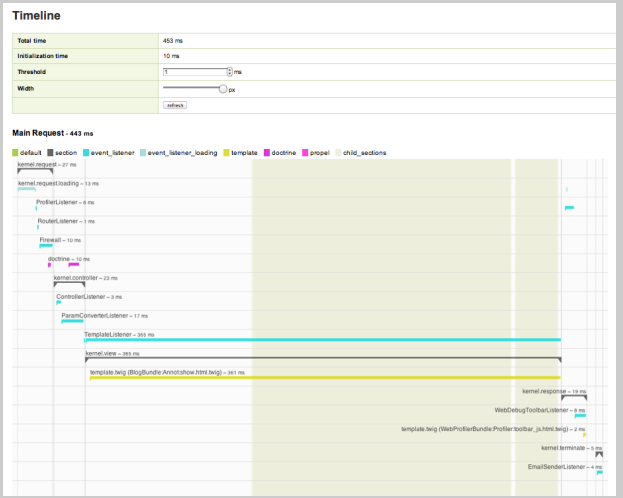
Total runtime (sec): 6.0161
Peak memory usage (KB): 22,098.9375

Time accumulators:

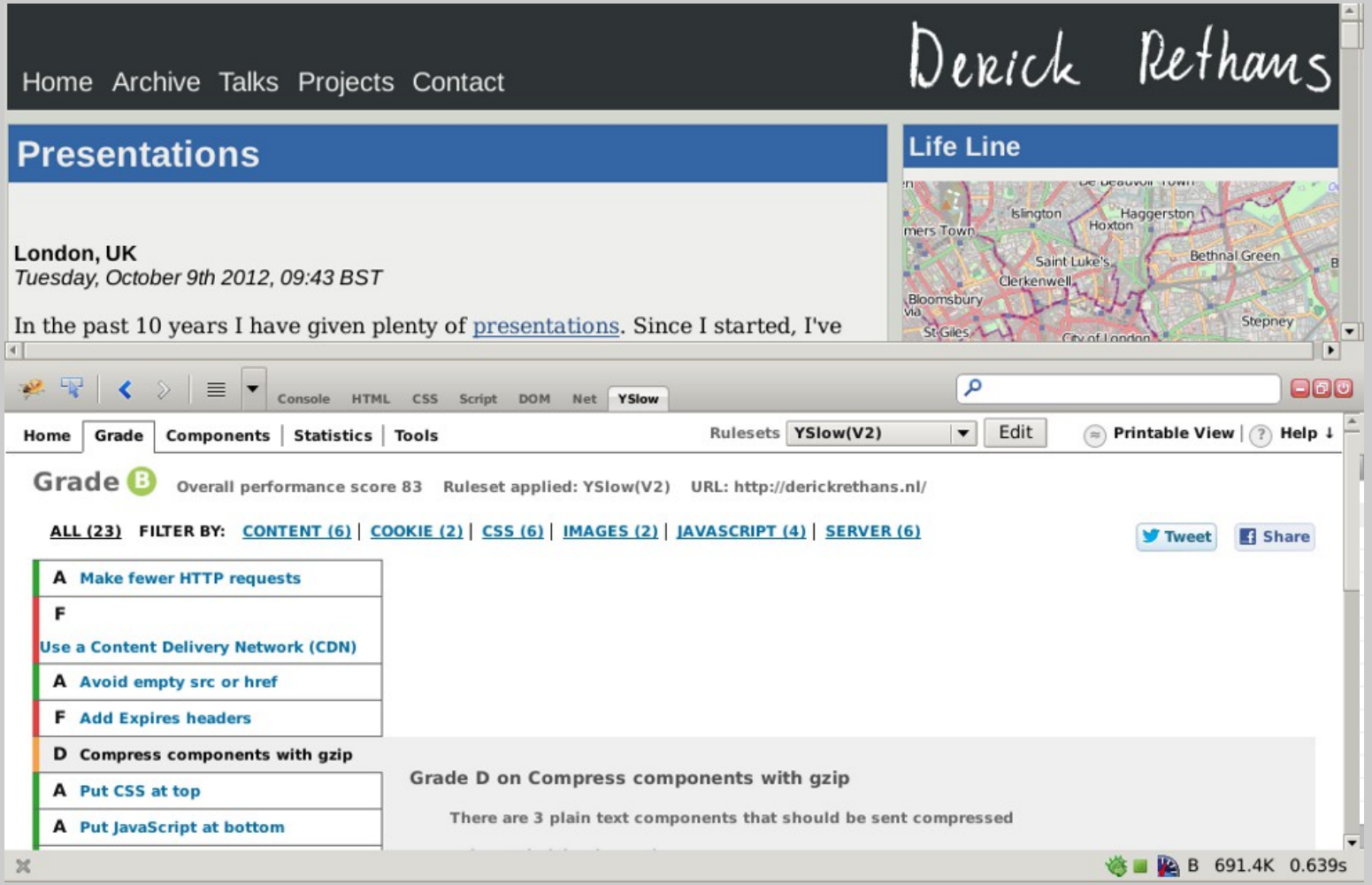
Accumulator	Lasts (sec)	Lasts (%)	Count	Average (sec)
ini_load				
Load cache	0.5950	9.8895	29	0.0205
Find INI Files	0.4026	6.6922	18	0.0224
Parse	0.0981	1.6302	18	0.0054
Save Cache	0.0699	1.1613	18	0.0039
Check MTime	0.0102	0.1703	11	0.0009
Mysql Total				
Mysqli_queries	0.0285	0.4745	18	0.0016
Looping result	0.0005	0.0090	14	0.0000
Template Total	*4.4332*	*73.7*	*3*	*1.4777*
Template load	1.8871	31.3673	3	0.6290
String conversion in template resource	0.0066	0.1105	75	0.0001
Template parser: create text elements	0.0978	1.6264	42	0.0023
Template parser: remove whitespace	0.0264	0.4382	42	0.0006
Template parser: construct tree	0.3477	5.7793	42	0.0083
Template load and register function	0.0264	0.4394	11	0.0024
Template processing	2.5457	42.3150	3	0.8486
override				
Cache load	1.3062	21.7117	15	0.0871
General				
INI string conversion	0.0065	0.1075	74	0.0001
String conversion	0.0041	0.0683	149	0.0000
String conversion w/ mbstring	0.0004	0.0072	8	0.0001
dbfile	0.0581	0.9665	47	0.0012

Note: percentages do not add up to 100% because some accumulators overlap

Total runtime - incl. debug printing (sec): 6.3711



Firebug/YSlow



Debugger and Profiler

- Xdebug: An Open Source debugging tool
- About 10 years old
- Works on "every" operating system
- PHP 5.1, 5.2, 5.3, 5.4 and trunk
- Version 2.2.2 is the latest

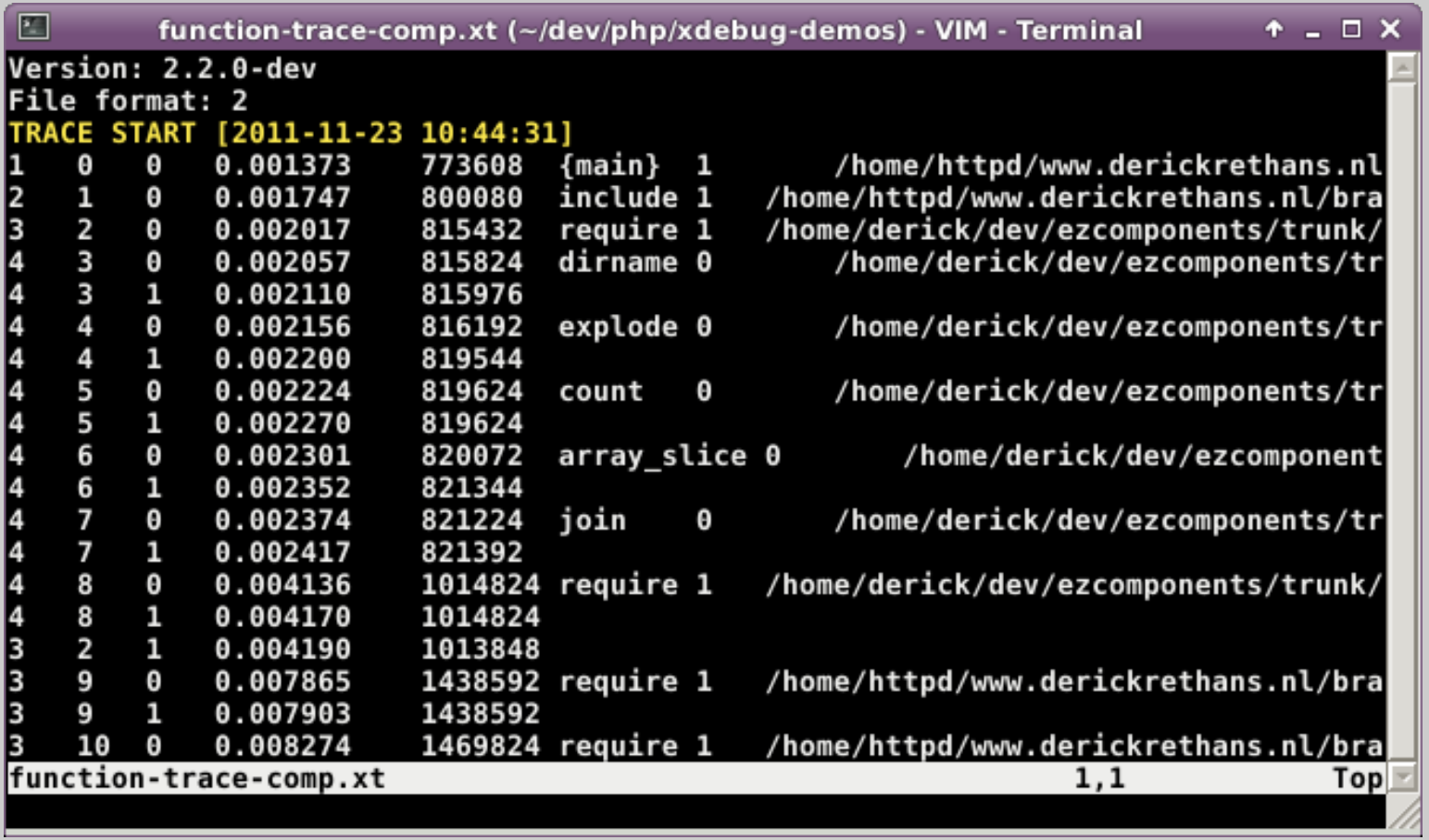
Function trace

Some settings:

```
xdebug.auto_trace=1
xdebug.trace_output_dir=/tmp
xdebug.collect_params=1
xdebug.collect_return=1
xdebug.collect_includes=1
xdebug.collect_assignments=1
```

Function trace to file

Automatic readable format




```
xdebug.auto_trace=1      ; enable tracing
xdebug.trace_format=1    ; selects computerized format
xdebug.trace_options=0   ; sets extra option (1 = append)
```

Trace file analysers

One bundled with Xdebug:

```
php ~/dev/php/xdebug/trunk/contrib/tracefile-analyser.php time-own 5
...
Showing the 5 most costly calls sorted by 'time-own'.

                                          Inclusive        Own
function                            #calls  time     memory  time     memory
-------------------------------------------------------------------------------
array_pop                              715  1.0252  -139880  1.0252  -139880
preg_match                            2986  0.3718  1016336  0.3718  1016336
{main}                                   1  6.4562  7335704  0.3476 -15198832
next                                   432  0.2386        0  0.2386        0
count                                 3302  0.2132        0  0.2123        0
```

ValaXdebugTools (http://tinyurl.com/vxdebugtools):

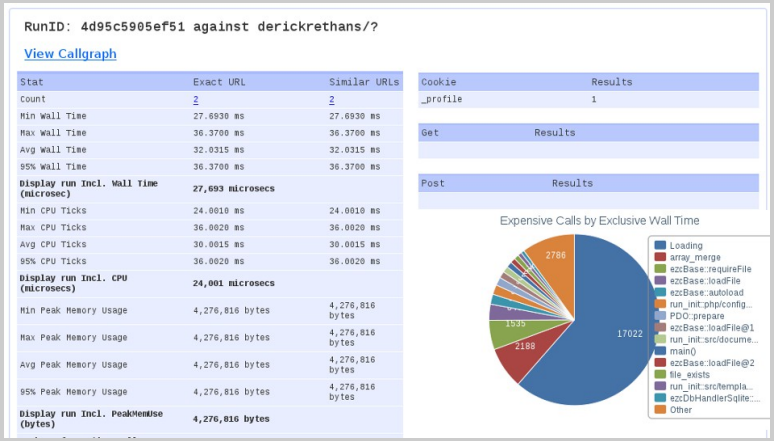
- trace4func: Give a function name, get a stack trace for each time it is called.
- evaled: Prints an ordered list of evals, together with what file is responsible for this evil.
- whocalls: Find out what files contain calls to the given function.
- whatisincluded: Find out what files are included and where they are included from.

demo

Profiling

XHProf

- Was original developed by Facebook
- XHProf and XHGui (https://github.com/preinheimer/xhprof)
- or XHProf.io (http://xhprof.io/) ?
- Meant to be run in a production environment



Profiling

Valgrind / Cachegrind

- Written for C-applications
- Xdebug uses the same format
- KCacheGrind is an awesome viewer
- Profile your app with both CacheGrind and Xdebug
- Alternative viewers: WinGrind, Macgrind, Webgrind, PHP Storm

demo

Resources

- Slides: http://derickrethans.nl/talks/profiling-zendcon12
- Contact me: Derick Rethans: @derickr, derick@derickrethans.nl
- Feedback: http://joind.in/6871