

## Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Tuesday, February 9, 2010

### A quick look at XHP

Facebook released a new PHP extension today that supports inlining XML. This is a feature known as XML Literals in Visual Basic. Go read their description here:  
<http://www.facebook.com/notes/facebook-engineering/xhp-a-new-way-to-write-php/294003943919>

It adds an extra parsing step which maps inlined XML elements to PHP classes. These classes are core.php and html.php which covers all the main HTML elements. The syntax of those class definitions is a bit odd. That oddness is explained in the How It Works document.

Essentially, it lets you turn:

```
renderBaseAttrs() . ' />';  
}  
}
```

which extends html-element which in turn extends primitive. You can go read all the code for those yourself.

Note that to build XHP you will need flex 2.5.35 which most distros won't have installed by default. Grab the flex tarball and ./configure && make install it. Then you are ready to go.

I pointed Siege at my rather underpowered AS1410 SU2300 with the above trivial form examples. The plain PHP one and the XHP version. Ran each one 5 times benchmarking for 30s each time. The plain PHP one averaged around 1300 requests/sec. Here is a representative sample:

```
acer:~> siege -c 3 -b -t30s http://xhp.localhost/1.php  
** SIEGE 2.68  
** Preparing 3 concurrent users for battle.  
The server is now under siege...  
Lifting the server siege... done.  
Transactions:      38239 hits  
Availability:     100.00 %  
Elapsed time:     29.60 secs  
Data transferred:  3.97 MB  
Response time:    0.00 secs  
Transaction rate: 1291.86 trans/sec  
Throughput:       0.13 MB/sec  
Concurrency:      2.93  
Successful transactions: 38239  
Failed transactions: 0  
Longest transaction: 0.05  
Shortest transaction: 0.00
```

And the XHP version:

```
Transactions:      868 hits  
Availability:     100.00 %  
Elapsed time:     29.28 secs  
Data transferred:  0.08 MB  
Response time:    0.10 secs  
Transaction rate: 29.64 trans/sec
```

## Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Throughput: 0.00 MB/sec  
Concurrency: 2.99  
Successful transactions: 868  
Failed transactions: 0  
Longest transaction: 0.21  
Shortest transaction: 0.05

So, a drop from 1300 to around 30 requests per second and latency from less than 10ms to 100ms. Running XHP on plain PHP is definitely out of the question. But, knowing that Facebook uses APC heavily and looking through the code (see the MINIT function in ext.cpp) we can see that it should play nicely with APC. So, re-running our PHP version of the form, now with APC enabled, that goes from 1300 to around 1460 requests per second, and no measurable latency:

Transactions: 43773 hits  
Availability: 100.00 %  
Elapsed time: 29.88 secs  
Data transferred: 4.55 MB  
Response time: 0.00 secs  
Transaction rate: 1464.96 trans/sec  
Throughput: 0.15 MB/sec  
Concurrency: 2.93  
Successful transactions: 43773  
Failed transactions: 0  
Longest transaction: 0.07  
Shortest transaction: 0.00

The XHP version of the form now with APC enabled:

Transactions: 9707 hits  
Availability: 100.00 %  
Elapsed time: 29.45 secs  
Data transferred: 0.94 MB  
Response time: 0.01 secs  
Transaction rate: 329.61 trans/sec  
Throughput: 0.03 MB/sec  
Concurrency: 2.97  
Successful transactions: 9707  
Failed transactions: 0  
Longest transaction: 0.21  
Shortest transaction: 0.00

Much better. But it is still around a 75% performance drop from 1460 to 330 and a ~10ms latency penalty. And yes, I did have a default filter enabled for these tests, so there was basic XSS filtering in place for the naked `$_POST['name']` variable in the plain PHP version. Of course, the default filtering would likely fail if the user data was used in a different context. And this 75% is obviously going to depend on what else is going on during the request. If you are spending most of your time calculating a fractal or waiting on MySQL, you may not notice XHP very much at all.

The bulk of the time is spent in all the tag to class interaction. If the core.php and html.php code was all baked into the XHP extension, it would be a lot quicker, of course. So, when you combine XHP with HipHop PHP you can start to imagine that the performance penalty would be a lot less than 75% and it becomes a viable approach. Of course, this also means that if you are unable to run HipHop you probably want to think a bit and run some tests before adopting this. If you are already doing some sort of external templating, XHP could very well be a faster approach.

Update: Here are the callgraphs.

The first is the plain PHP+APC version without XHP. And the second is the PHP+APC+XHP version. In the first you see all sorts of bits and pieces all across the stack getting cpu time.

## Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

In the second graph we see the effect of needing to copy and instantiate those core and html classes on every request. They are cached in APC, of course, but because of PHP's perfect sandbox, they cannot persist.

So we went from spending around 1% of our time in the executor to over 80%.

This isn't an entirely fair comparison, of course, since the plain version has close to no PHP to execute while the XHP version has 93 userspace classes to deal with. I would guess that XHP could get quite a boost if at least the primitives in core.php could be baked into the extension. Ideally all 93 basic html classes would be in C++ in the extension itself, but that would be a bit of a tedious undertaking.

Posted by Rasmus at 21:22

Thursday, February 4, 2010

### HipHop PHP - Nifty Trick?

In a response to a question from ReadWriteWeb, among other things, I wrote:

My main worry here is that people think this is some kind of magic bullet that will solve their site performance problems. Generating C++ code from PHP code is a nifty trick and people seem to have gotten quite excited about it. I'd love to see those same people get excited about basic profiling and identifying the most costly areas of an application. Speeding up one of the faster parts of your system isn't going to give you anywhere near as much of a benefit as speeding up, or eliminating, one of the slower parts of your overall system.

The "nifty trick" part of that seems to have become the story, and them injecting a "just" in front of it makes it sound more derogatory. Anyone who knows me knows that I am a big fan of nifty tricks that solve the problem. When I first heard about the Facebook effort I was assuming they were writing a JIT based on LLVM V8 or something along those lines. Writing a good JIT is hard. Doing static code analysis and generating compilable C++ from it is indeed a nifty trick. It's not "just" a nifty trick, it is a cool trick that takes advantage of a number of characteristics of PHP. The main one being that you can't overload PHP functions. `strlen()` is always `strlen`, for example. In Python, this would be harder because you can overload everything.

I also noted that most sites on the Web have a lot of lower hanging fruit that would provide a much bigger performance improvement, if fixed, than doubling the speed of the PHP execution phase. The ReadWriteWeb site, for example, needs 160 separate HTTP requests and 41 distinct DNS lookups to load the front page. And once you get beyond the frontend inefficiencies you usually find Database issues, inefficient system call issues and general architecture problems that again aren't solved by speeding up PHP execution.

If you have done your homework and find that your web servers are cpu-bound, you are already using an opcode cache like APC and your Callgrind callgraph shows you that the PHP executor is a significant bottleneck, then HipHop PHP is definitely something you should be looking at.

Posted by Rasmus at 10:50