Scaling PHP in the real world: a performance optimization checklist

PHP is slower than Java, C++, Erlang and Go, but that doesn't mean you have to resign yourself to having a slow app. On the contrary – there is plenty you can do to make your app scale for the masses. After all, the likes of Facebook, Yahoo, Zynga, Tumblr, Etsy and Wikipedia all use PHP. In this checklist you'll find some quick-and-dirty tricks as well as some bigger, ongoing projects to help improve the performance of your PHP app, whether you have one server or a thousand.

Going beyond a single server

If your PHP application runs on a single server and you're dealing with significant load, then the first thing you might want to think about is adding some infrastructure. Assuming that a single server is all you need for the purposes of your application, there are a few easy things you can do to get a performance boost. If you have a larger application, don't skip this section – make sure you've got all the low-hanging fruit before you embark on anything more ambitious.

Upgrade PHP. One of the easiest optimizations you can make to improve performance and stability is to upgrade your version of PHP. PHP 5.3.x was released in 2009. If you haven't migrated to PHP 5.5, now is the time! Not only do you benefit from bug fixes and new features, but you will also see faster response times immediately.1 Go to PHP.net to get started.

Try: Nginx + PHP-FPM. Nginx is a free open source HTTP server and proxy server known for its high performance and low memory footprint. Its scalable, event-driven architecture make it perform better in high-volume environments than other servers that use threads to service requests.

PHP-FPM is a FastCGI process manager for PHP that's both easy to tune and fast, since it runs independent of the web server process.

Use an opcode cache. PHP is an interpreted language, which means that every time a PHP page is requested, the server will interpret the PHP file and compile it into something the machine can understand (opcode). Opcode caches preserve this generated code in a cache so that it will only need to be interpreted on the first request. If you aren't using an opcode cache you're missing out on a very easy performance gain. Pick your flavor: APC, Zend Optimizer, or XCache.

Try: APC (Alternative PHP Cache) is a free and open cache written by the creator of PHP, Rasmus Lerdorf. APC works well for small and frequently accessed objects on a single server.

Use autoloading. Many developers writing object-oriented applications create one PHP source file per class definition. One of the biggest annoyances in writing PHP is having to write a long list of needed includes at the beginning of each script (one for each class). PHP re-evaluates these require/include expressions over and over during the evaluation period each time a file containing one or more of these expressions is loaded into the runtime. Using an autoloader will enable you to remove all of your require/include statements and benefit from a performance improvement. You can even cache the class map of your autoloader in APC for a small performance improvement, too.

Try: Symfony2 ClassLoader Component.

Small-to-medium sized app

Optimize your sessions. While HTTP is stateless, most real life web applications require a way to manage user data. In PHP, application state is managed via sessions. The default configuration for PHP is to persist session data to disk. This is extremely slow and not scalable beyond a single server. A better solution is to store your session data in a database and front it with an LRU (Least Recently Used) cache with Memcached or Redis. If you are super smart you will realize you should limit your session data size (4096 bytes) and store all session data in a signed or encrypted cookie.

Try: Storing sessions in Memcached, a popular in-memory key-value store.

Further reading:
www.php.net/manual/en/book.session.php
http://php.net/manual/en/function.session-set-save-handler.php
http://php.net/manual/en/memcached.sessions.php
www.hardened-php.net/suhosin/configuration.html#suhosin.session.encrypt

Leverage an in-memory data cache. Applications usually require data. Data is usually structured and organized in a database. Depending on the data set and how it is accessed it can be expensive to query. An easy solution is to cache the result of the first query in a data cache like Memcached or Redis.

If the data changes, you invalidate the cache and make another SQL query to get the updated result set from the database. There are also many use cases for a distributed data cache from caching web service responses and app configurations to entire rendered pages.

Try: Doctrine Object Relational Mapper (ORM) for PHP, which has built-in caching support for Memcached or Redis. Doctrine is a popular choice because it doesn't require much configuration to set up and offers developers the ability to use an object oriented SQL dialect (DQL) inspired by Hibernate's HQL.

Big app

Do blocking work in the background. Often times web applications have to run tasks that can take a while to complete. In most cases there is no good reason to force the end-user to have to wait for the job to finish. The solution is to queue blocking work to run in background jobs. Background jobs are jobs that are executed outside the main flow of your program, and usually handled by a queue or message system. There are a lot of great solutions that can help solve running background jobs. The benefits come in terms of end-user experience and scaling by writing and processing long-running jobs from a queue. I am a big fan of Resque for PHP that is a simple toolkit for running tasks from queues. There are a variety of tools that provide queuing or messaging systems that work well with PHP: Resque, Gearman, RabbitMQ, Beanstalkd, ZeroMQ, ActiveMQ

Try: Resque, a very simple Redis-backed library for creating background jobs created by GitHub.

Further reading:
https://github.com/chrisboulton/php-resque
https://github.com/kamisama/php-resque-ex
https://github.com/kamisama/ResqueBoard

Leverage HTTP caching. HTTP caching is one of the most misunderstood technologies of the Internet. Go read the HTTP caching specification. Don't worry, I'll wait. Seriously, go do it! They solved all of these caching design problems a few decades ago. It boils down to expiration or invalidation and when used properly can save your app servers a lot of load. Please read the excellent HTTP caching guide from Mark Nottingham. There are several reverse proxy caches available including Varnish, Squid, Nginx Proxy Cache and Apache Proxy Cache.

Try: Varnish, a free reverse proxy cache designed to be very fast and flexible.

Further reading:
www.w3.org/Protocols/rfc2616/rfc2616-sec13.html
www.mnot.net/cache_docs/

Holy sh!t scale

Create your own variant of PHP. When you become too big, PHP simply doesn't scale anymore. But, as we mentioned at the beginning, some of the biggest websites in the world run on PHP – so how do they do it? Some organizations choose to write their more complex functionalities as PHP extensions in C or C++.

- Facebook and the HipHopVM: Facebook built an open source virtual machine (VM) designed to "achieve superior performance while maintaining the flexibility that PHP developers are accustomed to." With the HipHopVM, Facebook's developers can still write in PHP without sacrificing performance.

Of course, these strategies are really only practical for the tech giants that have the scale to require a project like this (and the developer resources to throw at it). Most organizations won't need or want to create their own PHP variant in order to improve performance.

Everyone

Learn to profile PHP code. Caching and other quick performance optimizations are great, but if you have real issues in your PHP application, at some point you're going to have to get down and dirty with the code. Setting up a profiler in a dev or test environment is a great way to find out where exactly your bottlenecks are. Here are a few good profilers to get you started:

- Xdebug is a PHP extension for powerful debugging. It support stack and function traces, profiling information and memory allocation and script execution analysis. It allows developers to easily profile PHP code.
- WebGrind is an Xdebug profiling web frontend in PHPS. It implements a subset of the features of kcachegrind and installs in seconds and works on all platforms. For quick and dirty optimizations it does the job. Here's a screenshot showing the output from profiling (?)
- XHprof is a function-level hierarchical profiler for PHP with a reporting and UI layer. XHProf is capable of reporting function-level inclusive and exclusive wall times, memory usage, CPU times and number of calls for each function. Additionally, it supports the ability to compare two runs (hierarchical DIFF reports) or aggregate results from multiple runs.

Optimize your framework. Deep diving into the specifics of optimizing each framework is outside the scope of this post, but these principles apply to every framework:

- Stay up-to-date with the latest stable version of your favorite framework
- Disable features you are not using (I18N, Security, etc)
- Enable caching features for view and result set caching

Don't forget to optimize the client side! Server side application performance is only part of the battle. Now that you've optimized the server side, you can spend time improving the client side! In modern web applications most of the end user experience time is spent waiting on the client side to render. Google has dedicated many resources to helping developers improve client side performance.

Build for performance. Scalability is about the entire architecture, not some minor code optimizations. If you want a fast app, you need to build for speed – and look at what kinds of major improvements can be made to make your app perform better.

Try: Google PageSpeed is a great tool for finding quick and easy ways to reduce the load time of your web pages. It analyzes the contents of your web pages and then produces some suggestions for how to improve their performance. They even have modules for Apache and Nginx to automatically implement some of the rules.

Conclusion

There's no easy answer to making your app fast, unfortunately. There are some quick optimizations you can implement to get a performance boost, but ultimately performance is an ongoing part of building and maintaining an application. Hopefully this checklist helped you get an idea of where you sit on the performance spectrum, and what you need to do next in order to scale your app.