MAGENTO HOSTING

Progressive Server Performance Improvements

Simple Helix, LLC
4092 Memorial Parkway Ste 202
Huntsville, AL 35802

sales@simplehelix.com
1.866.963.0424

www.simplehelix.com

SIMPLE HELIX
WORLD CLASS HOSTING SOLUTIONS

Table of Contents

Introduction

Magento is a popular and feature rich eCommerce platform that gives merchants full control over the look, content, and functionality of their online presence in an intuitive and easy to use format. While there are many benefits to using Magento, the primary handicap is that an out-of-the-box Magento installation can be very resource-intensive.

The main goal of this white paper is to ensure that you have the software infrastructure and configuration needed to provide the optimal Magento online experience for your customers. To this end, we will show the benchmark results of a basic configuration and then demonstrate how various progressive improvements can dramatically increase performance.

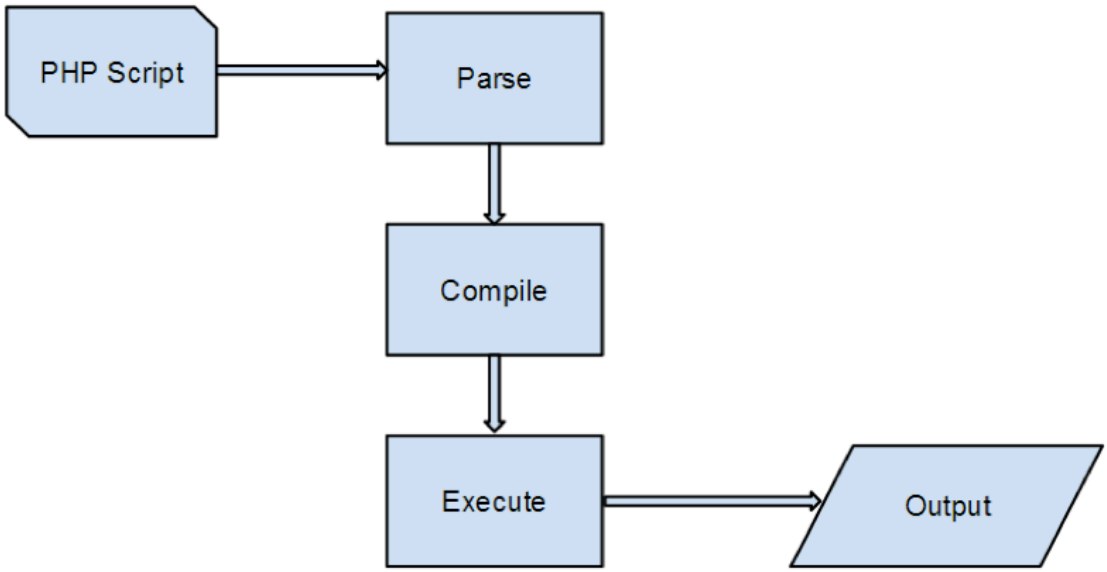
While there are a multitude of ways to tweak and speed up Magento stores, we have limited the scope of this paper to cover four key areas: Magento's internal cache and session storage, the back-end database, HTTP requests and PHP processing, and fullpage caching. Our final test utilizes the Nginx web server, Percona database server, APC opcode caching, PHP's FastCGI Process Manager (PHP-fpm), and the Varnish web accelerator in a loadbalanced configuration to achieve approximately 634.75 sustained transactions per second.

Optimization Options

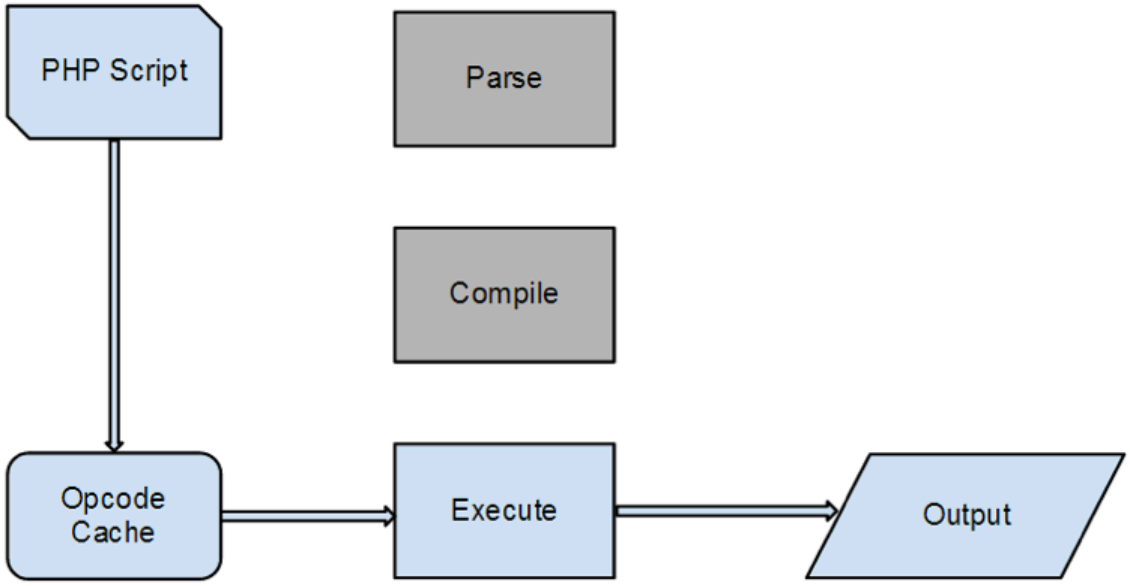
APC Opcode Caching

The PHP language is used to power Magento as well as a large number of other commonly used web platforms. Unlike a compiled language like C or C++, PHP code is not compiled directly to machine code. Instead, PHP code is compiled down to an intermediary bytecode that is interpreted by the PHP runtime engine. This allows PHP to be easily ported to a wide variety of platforms quickly and efficiently.

Although the interpreted nature of PHP has many positive aspects, the primary drawback is that it can make heavy PHP pages run slowly since each page request must be compiled into the intermediary bytecode before it can be interpreted. A visual breakdown of this process is listed below:



As you may imagine, this process is slow and can cause high server load given the right conditions. To help resolve this potential bottleneck, a number of different compilation optimizations have evolved over the years. One of the most common optimizations is to cache the opcode created during the compilation process, which will eliminate the need to continually parse and re-compile the PHP code. A visual representation of opcode caching is listed below:



Skipping the parse and compile each time a PHP Script is executed will not only speed up page loading, but can also reduce server load since valuable CPU cycles will not be wasted re-compiling the same code over and over.

Although there are several popular cache engines for accelerating PHP execution, we recommend Alternative PHP Cache (APC) because it is stable, easy to configure, works well with Zend Optimizer, and it will be included in PHP 6. In addition to caching and speeding up PHP execution, APC can also be used to speed up Magento's internal cache.

Percona Database Server

Magento makes use of the InnoDB engine for the backend database. From a developer standpoint, this is very important since InnoDB implements locking on the row level. Most Magento stores have a high transaction rate, so InnoDB's row-locking allows only the row being modified to lock, leaving the rest of the table open to modification if needed. While this allows many simultaneous users to perform transactions at the same time, it can cause significant I/O delays which result in your site running slowly.

Since most Magento installations use the MySQL relational database management system for the backend database, these I/O delays in the InnoDB engine are usually dealt with by tuning MySQL settings until the optimal balance of response time and I/O load is found. While this can work, a more efficient and stable solution would be to replace MySQL with Percona Server.

Percona Server is a fork of the MySQL relational database management system. While maintaining 100% compatibility with MySQL, the Percona Server includes enhancements and self-tuning algorithms, created by industry leading database experts, that can yield up to 40% faster overall performance. Percona also offers many features that MySQL lacks for monitoring the health and statistics of the InnoDB engine.

Nginx Web Server
Apache is the most common web server choice for Magento sites due to its long history and generally good reputation as a solid process-based web server. In recent years, however, Nginx has received praise and attention for being a relatively lightweight and very fast alternative.

Unlike Apache which is process-based, Nginx is an asynchronous server. This primary difference is why Apache uses considerable server resources under heavy loads that can degrade site performance. Due to its asynchronous nature, Nginx handles server requests via event-driven threads which uses much fewer resources under heavy load and results in better performance.

Nginx provides a unique combination of a web server, a caching proxy with a low memory footprint and a load balancing solution. For serving large amounts of static traffic, Nginx maintains a small memory footprint and generally requires fewer system resources than the Apache web server. This allows it to perform better than Apache when the server is under heavy load, which makes it ideal for hosting a Magento store that expects a large amount of user traffic.

Varnish HTTP Accelerator
Varnish is a Web Application Accelerator that can be used to enhance website performance. It works by caching page requests and serving future requests for that page from cache. If your site makes many database queries, such as is common with any Magento site, then Varnish can give your site a performance boost and reduce the total load on your server by reducing the number of database queries that must be made. Serving content from cache results in page loading that is almost as fast as static HTML pages.

Benchmark Setup and Configuration

Tools Used

In order to determine if new software or new configurations yields better performance, it is necessary to measure the performance before and after the change. Accurately measuring the performance of a Magento installation requires testing many facets including, but not limited to, the initial HTTP request, database query time, sustained server load, and data throughput. Two tools were used to conduct benchmarking for each test: Apache Bench and Siege.

Apache Bench, developed by the Apache Software Foundation, is a single-threaded tool designed to test the performance of an HTTP server. It is especially useful for determining network latency and the maximum number of requests per second that a particular HTTP server is capable of.

Siege is a multi-threaded tool designed to simulate the load of many concurrent users all visiting different pages on a site. It provides a number of measurements including total data transferred, server response time, transaction rate, throughput, concurrency and the number of times a page failed to load.

Software Versions

The following software versions were used during the testing process:

Name	Version
CentOS (x86_64)	6.4
Apache	2.2.23
Nginx	1.0.15
PHP	5.3.18
PHP-fpm	5.3.18
MySQL	5.5.30
Percona	5.5.30
APC	3.1.9
Siege	2.72
Magento Enterprise	1.13

Note that control panel software such as CPanel or Direct Admin was not used during our testing. The reason for this decision is that control panels introduce extra server overhead and, since they perform various server administration and maintenance, may introduce unknown variables that would skew test results.

Hardware Architecture

The hardware used in all tests consisted of one database server, two web servers, and a load-balancer that was used to evenly distribute traffic to each web server.

Database Server
HP ProLiant BL460c G6
Intel Xeon 4-core L5520 @ 2.27GHz
16GB ECC RAM
500GB SATA hard drive

Web Server (2x)
HP ProLiant BL460c G7
Intel Xeon 4-core X5667 @ 3.07GHz
8GB ECC RAM
146GB 15.5K SAS hard drive

Load-Balancer
F5 Viprion 2400

The testing servers were placed on a gigabit redundant network within the Simple Helix data center in Atlanta, GA.

Pre-test Normalization

To avoid anomalous data from skewing test results, all tests were ran seven times and then the average of all results was taken. Before each test, all levels of cache were cleared, machine load averages settled back to an idle state, and all network connections on the load balancer were reset. This procedure ensured that all tests would encounter the same system state upon beginning.

Analysis of Testing Methods

Apache Bench - Latency and Responsiveness Testing
Running a single trial of Apache Bench will not yield true results because there are many factors such as server load and network speed which could skew the results of a single request. For this reason, much more accurate results can be obtained by running N trials and then aggregating the results. All of our tests consisted of 1,000 trials with 200 concurrent connections and a time limit of 60 seconds. This test was repeated every minute on the minute for 30 minutes. The data collected for each test was then averaged for the end test results.
Siege - HTTP/HTTPS Stress Test
Our Siege test consisted of 200 concurrent connections for a sustained 60 seconds and were repeated every minute on the minute for a total of 30 minutes. The data collected for each test was then averaged for the end test results.

To make the most of our test cases, we configured Magento Enterprise with 100,000 SKUs of test products. Using Magento's sitemap, we then created a text file containing the URL for all of these products as well as standard Magento URLs. Siege was configured to hit random URLs from this output file which simulated the type of random traffic typically generated by users.

Test Cases

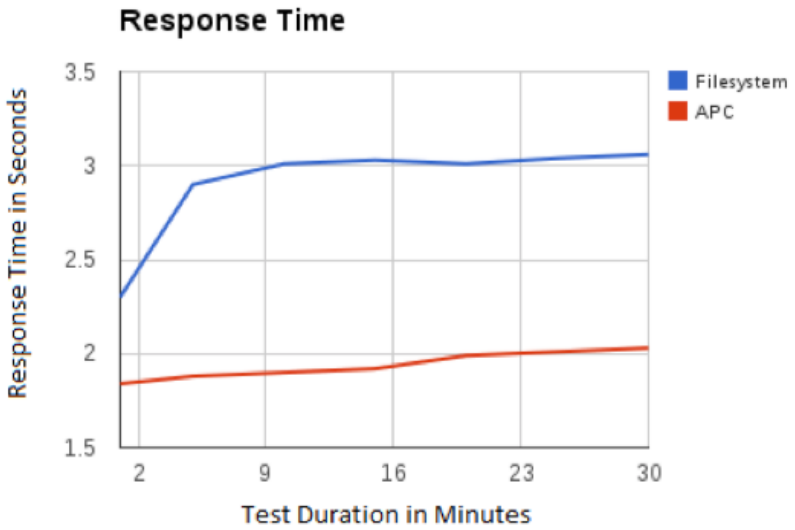
Test 1: Filesystem vs. APC Cache Storage

Magento stores cache entries in the filesystem by default. This is fine for smaller sites with relatively small cache sizes, but as traffic and the number of products increase the seek time will get exponentially slower. Additionally, the disk I/O required to read and write cache data to the disk can cause excessive server load which diminishes the server's ability to process client requests. An alternative to filesystem cache is to store the cache in APC, a PHP opcode caching solution.

In this test, we configured a basic Apache and MySQL configuration to serve our Magento Enterprise store. We then proceeded to run tests against the stock configuration that stored cache in the filesystem. Once these tests were completed, we installed APC and configured Magento to store its cache in APC. The tests were then ran again.

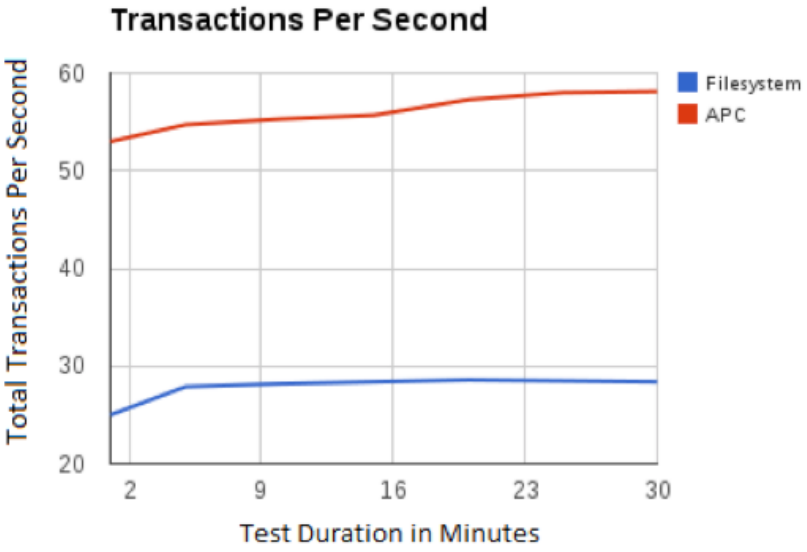
Test 1: Results

The following graph demonstrates the response time difference between each test case. This metric was gathered using Apache Bench to measure the average time, in seconds, that the server required to respond to client requests. Lower numbers are better in this test since they indicate a faster response time.



While both tests started out with relatively fast response times, the Filesystem test quickly fell almost a full second behind the APC test. This drop can be attributed to the extra I/O required for the Filesystem test that APC allows us to bypass.

The following graph demonstrates the total transactions per second that were possible in each test case. This metric was gathered using Siege to measure the total complete transactions processed every second. Higher numbers are better in this case since they indicate more total transactions per second.



Since APC allows cache to be stored and retrieved directly from ram, eliminating disk I/O, we are able to perform approximately 3x more transactions per second.

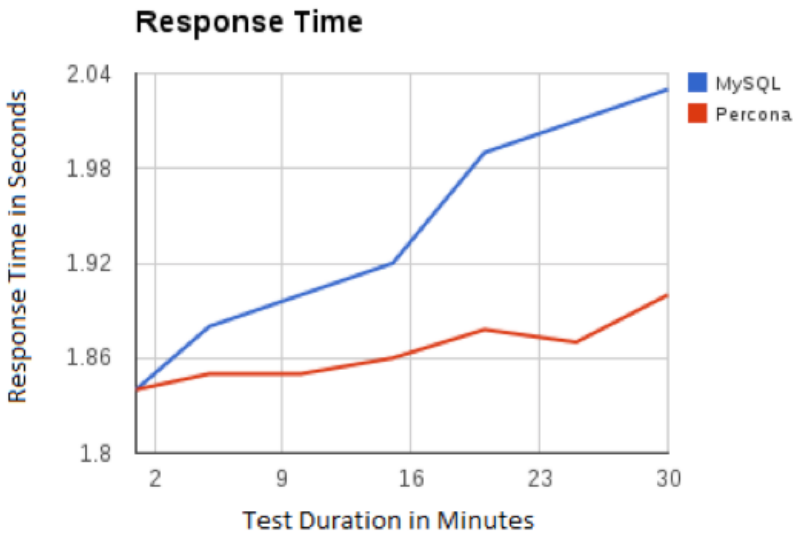
Test 2: MySQL vs. Percona

Both MySQL and Percona can be used interchangeably as the database back-end for Magento with relatively little setup work. Although MySQL can be tuned for performance, Percona has been designed by database professionals and boasts up to 40% better performance.

In this test, we ran the first set of benchmarks against the Magento store using Apache, APC opcode caching, and MySQL. We then replaced MySQL with Percona and ran the second set of benchmarks.

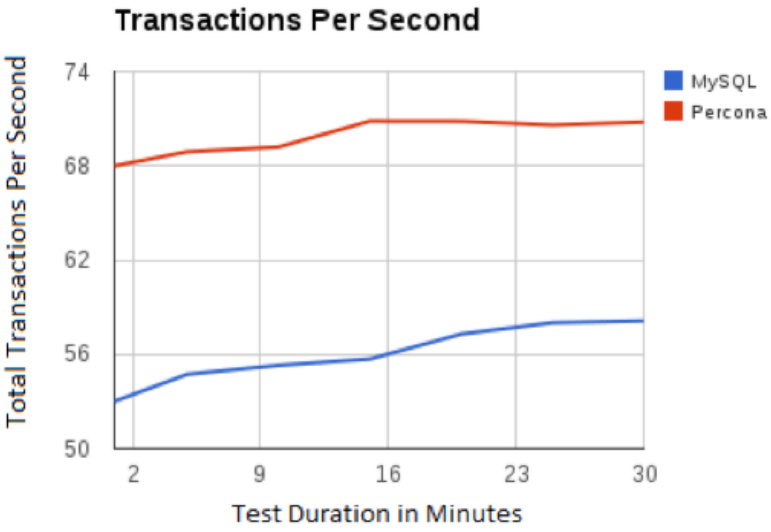
Test 2: Results

The following graph demonstrates the response time difference between each test case. This metric was gathered using Apache Bench to measure the average time, in seconds, that the server required to respond to client requests. Lower numbers are better in this test since they indicate a faster response time.



As you can see in the test above, the response time of the MySQL test quickly increased under sustained load. Percona exhibited more stable behavior and was able to handle sustained connections without sacrificing response time.

The following graph demonstrates the total transactions per second that were possible in each test case. This metric was gathered using Siege to measure the total complete transactions processed every second. Higher numbers are better in this case since they indicate more total transactions per second.



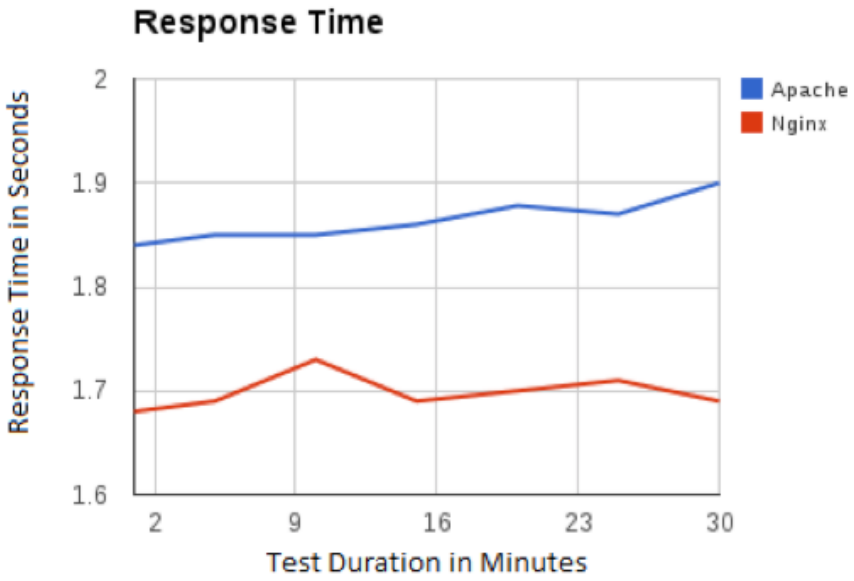
The reduced server load and faster query times offered by the Percona Server results in approximately 21.48% more transactions per second. This means that by using Percona Server, we can process 387 more transactions in a 30-minute period than would be possible by using MySQL.

Test 3: Apache vs. Nginx

In this test, we ran the first set of benchmarks against a Magento store running on Apache, APC opcode caching, and the Percona database. We then replaced the Apache web server with Nginx and PHP-fpm and ran the second set of benchmarks.

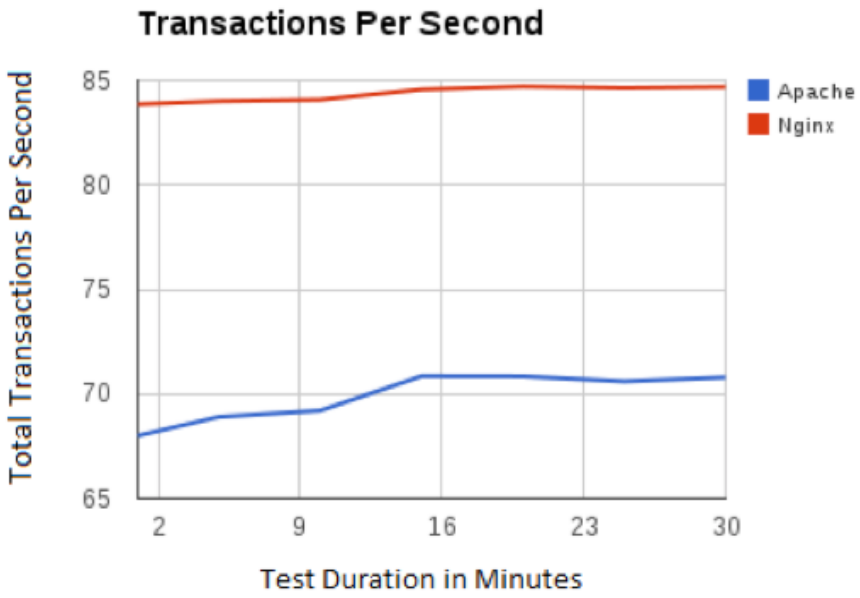
Test 3: Results

The following graph demonstrates the response time difference between each test case. This metric was gathered using Apache Bench to measure the average time, in seconds, that the server required to respond to client requests. Lower numbers are better in this test since they indicate a faster response time.



While the response time of Nginx is marginally faster, both web servers remained relatively fast with response times of less than 2 seconds.

The following graph demonstrates the total transactions per second that were possible in each test case. This metric was gathered using Siege to measure the total complete transactions processed every second. Higher numbers are better in this case since they indicate more total transactions per second.



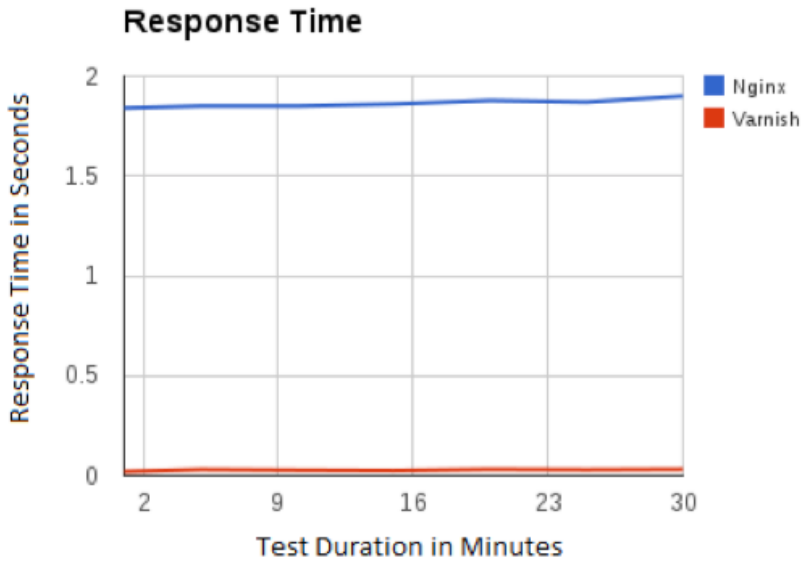
Nginx was able to process 17.9% more transactions per second than Apache.

Test 4: Nginx vs. Nginx + Varnish

In this test, we ran the first set of benchmarks against a Magento store running on Nginx, APC opcode caching, and Percona. We then setup the Varnish cache engine to work with Nginx and ran the second set of benchmarks.

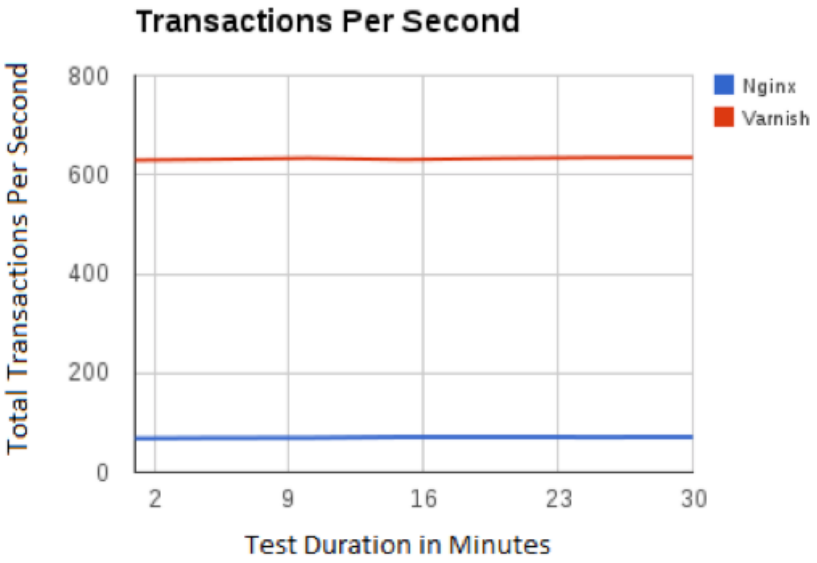
Test 4: Results

The following graph demonstrates the response time difference between each test case. This metric was gathered using Apache Bench to measure the average time, in seconds, that the server required to respond to client requests. Lower numbers are better in this test since they indicate a faster response time.



Although the response time of the Nginx tests remained just under 2 seconds, the response time of the Varnish test was almost instantaneous.

The following graph demonstrates the total transactions per second that were possible in each test case. This metric was gathered using Siege to measure the total complete transactions processed every second. Higher numbers are better in this case since they indicate more total transactions per second.



Using a Varnish in combination with APC, Percona, and Nginx + PHP-fpm, we were able to achieve approximately 634.75 sustained transactions per second.

Summary of Results and Optimizations

All testing was done on a Magento Enterprise installation without any plugins, extensions, or third-party themes installed. Depending on their functionality, such add-ons may have a negative performance impact. It is recommended that benchmarks of your site be taken before installing new add-ons so that you can be aware of any performance differences resulting from the installation.

As you can see from the test results, each optimization option covered in this paper offers ample performance increases. This testing was done in a progressive fashion, where each consecutive test resulted in better efficiency. These tests demonstrate how the performance of your Magento store can be steadily increased without requiring expensive development work or complex and potentially buggy extensions.

The testing outlined in this document revealed how it is possible to optimize a Magento installation capable of processing less than 30 sustained transactions per second to the point where it can process 634.75 sustained transactions per second. To accomplish this transformative performance increase, we implemented the following progressive upgrades:

- Used APC opcode caching to speed up Magento's built-in caching and reduce disk I/O.
- Replaced MySQL with Percona for faster database queries.
- Run Nginx and PHP-fpm instead of Apache to maintain a smaller resource footprint capable of serving more requests.
- Use Varnish to reduce server load and ameliorate server requests by serving cached content.

It is important to note that while APC and Percona can be installed and configured with relative ease, Nginx and Varnish require extensive configuration to work well with Magento.

Appendix A: Benchmarking Configurations

Apache Bench

The following command line options were used for all Apache Bench tests:

/usr/sbin/ab -n 1000 -c 200 -e $ABTESTNUM.log $URL

$ABTESTNUM: This is a shell variable that was incremented with each test case we ran. This variable was used to make cataloging and parsing the results easier.
$URL: This is a shell variable used to test a series of different pages pulled from the Magento sitemap.

Siege

The following command line options were used for all Siege tests:

/usr/local/bin/siege -c 200 -i -b -t60s -f url.txt --log=$SGTESTNUM.log -A “$SGTESTNUM”

url.txt: This file consisted of the complete Magento sitemap parsed into a text file.
$SGTESTNUM: This is a shell variable that was incremented with each test case we ran. This variable was used to make cataloging and parsing the results easier.