COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

PERFORMANCE AND SPEED OPTIMIZATIONS OF WORDPRESS-BASED WEB APPLICATION AND ITS UNDERLYING SERVER STACK

Bachelor's thesis

2015 **Rastislav Lamoš**

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

PERFORMANCE AND SPEED OPTIMIZATIONS OF WORDPRESS-BASED WEB APPLICATION AND ITS UNDERLYING SERVER STACK

Bachelor's thesis

Study program: Applied computer science
Supervisor: Mgr. Kamil Maráz.

Bratislava 2015 **Rastislav Lamoš**

Declaration of authorship

I hereby declare and confirm that this thesis is entirely the result of my own work except where otherwise indicated.

Bratislava, 31. May 2015 .....

Acknowledgement

To be written...

Abstract

In our work, we will be dealing with the performance and speed optimisations of an underlying server and a WordPress-based web application running on it. After a thoughtful study of our work, an ordinary web programmer or administrator will be able to install and configure his web server and refactor the source code of his PHP (WordPress) web application, without having to research what to modify or avoid from other sources, thus saving his time and resources. Product of our work will be a highly optimised server with proper caching and a WordPress application developed to be as efficient as possible.

Keywords: *page loading speed, WordPress, optimisation, PHP, server*

Abstrakt

V našej práci sa budeme zaoberať optimalizáciou výkonu a rýchlosti serveru a web aplikácií založenej na systéme WordPress, ktorá na tomto serveri beží. Po pozornom prečítaní a naštudovaní našej práce si bude bežný web programátor či administrátor vedieť nainštalovať a nakonfigurovať svoj web server a refaktorovať zdrojový kód svojej PHP (WordPress) web aplikácie bez toho, aby musel z iných zdrojov zisťovať čo zmeniť a čomu sa vyvarovať, čím sa ušetrí jeho čas a zdroje. Výsledkom našej práce bude vysoko optimalizovaný server s vhodným caching a WordPress aplikácia vyvinutá tak, aby bola čo najefektívnejšia.

Kľúčové slová: *rýchlosť načítavania stránky, WordPress, optimalizácie, PHP, server*

Glossary

caching test 15

Contents

List of Figures

List of Tables

1. Introduction

1.1 Work Structure

Our work is divided into three main parts.

Firstly, we delve into the problematics, describing what is WordPress and why its performance matters. We present past experiments and their results, pointing out their achievements as well as their deficiencies. We also specify and describe all the necessary tools, software and skills needed to understand and be able to reproduce the work in this part.

Secondly, we present our solutions to the problem(s) with thorough explanations. The author of the work keeps a technical blog [15] where a step by step guide to achieve the results shown in this document is located. We comment and describe only the most important sections of the guide in this document.

Lastly, we recapitulate and select the most efficient and optimized configurations for both server software and WordPress. A command line application (script), which installs all the required software and configures it is also included.

1.2 Problem Definition

Website loading and rendering speed has a critical impact on page abandonment rate among its visitors. According to a Google experiment, web page loading time increased only by **half of a second**, had a 20% drop in its visitor's traffic [13]. On figure 1.1 [31], we can see a chart of page abandonment relative to a web page loading time in seconds. By the time a ten-seconds loading web page is finished rendering to the users, more than 35% of them will close the page, never seeing what is on it.

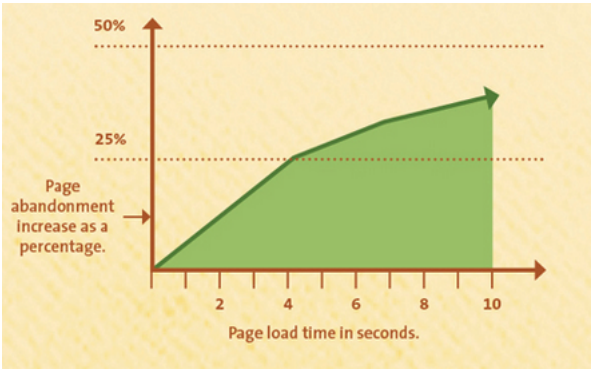

Figure 1.1: Slower page response time results in an increase in page abandonment

If we want to highlight the seriousness of this issue, imagine a situation in which an e-commerce site owner is making $100,000 per day. One second page delay could potentially cost him or her $2.5 million in lost sales every year [31]. See figure 1.2 for more detail.

Figure 1.2: What can a one-second page delay cause to your e-commerce site?

What is more, Google incorporated site speed in search rankings in 2010 [6], meaning that the slower a website loading is, the lower rank in Google search results it will receive. The author of this work believes he has shown the reader enough evidence that optimizing the performance and speed of a website is crucial for its success, especially in today's fast-paced world.

1.3 About WordPress

1.3.1 General Introduction

To cite WordPress.org, "WordPress is web software you can use to create a beautiful website or blog."[30] Simply put, WordPress is a very powerful, open source web publishing software, content management system and a web platform for building rich web applications. People and companies are using WordPress for various purposes and reasons, most notably for their blogs, websites, e-commerce solutions and large portals. At the time of writing (January 2015), it is estimated that about 23% [27] of all websites, whose content management system is known, is being run on WordPress. This number is quite

astonishing because it means that visiting five random websites, one of them will be a WordPress-powered one.

WordPress, in comparison to other web softwares and frameworks, is a full-featured, stand alone web publishing software and content management system. At its core, it consists of a request-response routing subsystem, classes for managing database and content handling, security features and others. WordPress was initially started as an open source hobby project of Matt Mullenweg in 2003 [10]. Since then, web programmers from all around the world have contributed to it, making it robust, secure and fast. However, as there are several bottlenecks in the system, the author of this work, himself a web developer, decided to analyze and improve them. His findings and results are summarized in this thesis.

1.3.2 WordPress-Powered Website

In order to customize the look and feel of user's instance of the website, there exists a mechanism called the WordPress **Theme**. A WordPress theme is simply a collection of scripts, stylesheets and images which get combined, processed and the generated content is sent back to the client. A user can install any theme which is compatible with his or her WordPress version. There is no central body governing the quality and correctness of a theme. As the themes developers are free to construct them almost arbitrarily, numerous security and performance flaws and issues can occur. What is more, core WordPress developers introduced a handy feature called the **Plugin**. WordPress plugin is a pluggable piece of software which enhances the basic WordPress functionality, thus enabling its users to heavily modify their WordPress-based web applications and sites. Plugins are also open source and without any quality guarantees, thus they are predated with the same problems as the aforementioned themes.

Moreover, if the user has a high-traffic website, his web delivering server is not able to keep up with all the requests resulting in a slow, unresponsive website. The reader might assume that the problems would be solved by increasing plugins quality. While the previous statement is true, it is usually not viable, mainly due to a reason that **work of an experienced web developer is relatively expensive**. In many cases, upgrading the server and/or getting additional ones is the preferred way. In our work, we are concerned with optimizing the server software first and only then examining the best practices, tips and tricks of a plugin or a theme development.

1.4 General Measures and Techniques for Website Speed Improvement

Before we can delve into the actual solutions of our problems, we need to list and describe several general techniques and measures we can use to decrease not only the website loading times, but also the server resources usage and load.

1.4.1 Web-Serving Software Efficiency

Using the most performant and the least resource-hungry web-serving and accompanying software is usually the most effective way of decreasing web page loading speed. Some studies have found that doing even small changes to configuration files can make quite a difference. In our work, we are making comparisons between the two most popular web-serving softwares, namely Apache HTTP and Nginx as well as comparisons between different PHP interpreters. Jump to section **??** for more details and statistics.

1.4.2 Server-Side Caching

Let us divide server-side caching into three main categories:

1. **Caching intermediate PHP code**
2. **Caching the output of PHP interpreter (so called "page cache")**
3. **Static resources (files) caching**
4. **Database caching**

Caching intermediate PHP code

PHP source code is interpreted on each run, thus the PHP interpreter has to read and collect all required files each time a new request is sent to our WordPress-based website. With PHP opcode caching, an intermediate source code is generated on the first run. It is stored temporarily and when a new request comes, instead of going through all the files, PHP loads cached opcode and executes it. We can observe the process from the figure 1.3 [20].

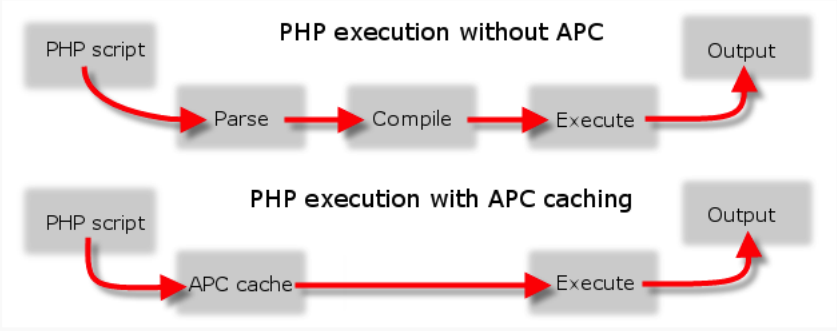

Figure 1.3: PHP execution diagram with and without APC opcode cache.

Caching the output of PHP interpreter (so called "page cache")

While a website based on WordPress is dynamic, in some cases it is possible to store the constructed web page for subsequent usage. This process is usually called *page caching* and it is suitable for websites with static blocks of content. However, problems arise when we have to handle special cases such as when a visitor is logged into our website. In that case we are not able to cache the request because other visitors would see the logged-in visitor's cached content instead of general one.

Static resources (files) caching

If our website consists of a large number of (smaller) files such as JavaScript scripts, CSS stylesheets, images and others, caching these resources is an idea worth mentioning. They are constructed and loaded on the initial request, stored in a local web server cache and retrieved from the cache on subsequent requests.

The largest disadvantage of static file caching is usually a drop in free memory available to different needs of the web server, especially when files are cached in the RAM memory.

Database caching

When multiple requests to our web server trigger the same database queries, it is reasonable to store the retrieved data for later use. When a subsequent request is made, querying into the database is omitted, thus preserving valuable server resources and outputting the resulting web page faster.

1.4.3 Client-Side Caching

What makes client-side caching different from the server-side caching is that data is stored locally in the visitor's browser, not on the server to which the requests are made. [21]

The whole hypertext document with all its resources including JavaScript scripts and CSS stylesheets can get cached in local storage and loaded and executed from it on the consecutive requests. Naturally, the most notable advantage of client-side caching is the fact that the user's browser does not need to download the locally cached resources.

On the other hand, a mechanism handling resource changes has to be implemented on the server-side. If it is not done properly, some of the visitors might see outdated content due to the reason that the visitor's browser has not been instructed to revalidate its cache.

1.4.4 JavaScript and CSS Resources Minification and Combination

At the time of writing (early 2015), a large number of WordPress themes and plugins contain tens or more JavaScript scripts and CSS styles, especially the more professional ones. It is caused by the fact that users like having amazingly-looking, feature-rich websites. The problem with this fact is twofold:

- **There is a limit on the number of concurrent downloads of resources from the same domain. Both Internet Explorer 8 and Google Chrome allow six concurrent downloads, while Firefox eight [32].**
- **Each request carry an overhead of constructing a packet, sending it to the web server and waiting for the reply.**

There are two well-known methods of solving this issue:

1. **Resources combination and/or**
2. **resources minification.**

Resources combination

Resources combination is a process in which resources on the requested web page are collected into (preferably) single file which is then sent back to the visitor's browser. There exist two main downsides of this approach. The first is that collecting the resources consumes additional CPU cycles on the server-side. Another disadvantage happens when the owner of the website modifies source code of any of the grouped resource. The whole group has to be gathered together again, including revalidating browser cache if present.

Resources minification

JavaScript scripts and CSS stylesheets can be minified before being inserted into the response body. Minification is a procedure in which parts of a resource source code are reduced or completely removed, thus reducing its size and length. Advanced minification tools are capable of refactoring the source code in a manner that it becomes even more compact.

1.4.5 Compression of Images, HTML and Other Resources

Image compression

Images, particularly those produced by the JPEG lossy compression mechanism, can be compressed further, thus reducing their size while keeping tolerable quality of picture details. At the time of writing this work, the cost of satisfying web service [34] for image compression is zero.

Hypertext documents and other text-based assets compression

Before the data is sent back in a response to visitor's request, the size of some of them can be reduced further with a process called data compression [8]. Most modern browsers [22] support *GZIP* compression of textual data. One of the downsides of performing this process is that additional CPU cycles on both server and client sides are expended in order to compress and uncompress the data.

1.4.6 Optimizing Images — Image Sprites

Images used for our website's user interface as well as other images can become more optimized for use in the web environment. A mechanism called image spriting [24] is a procedure during which multiple images, sometimes even all of them, get collected and combined into a single larger image — sprite. When a web page is requested, instead of loading tens of user interface icons and images, only this single one is sent back to the user, thus saving additional HTTP requests and bandwidth. When rendering the user interface, icons and images are taken from that single image.

1.4.7 CDN and Resource Distribution

In some cases our website is accessed from many different countries, even continents. If we have web servers located only locally, additional milliseconds start to congregate in these situations. To solve this problem, web administrators usually distribute the website's data

across multiple servers positioned in multiple places around the world. CDN services greatly reduce amount of work needed to accomplish the goal by doing it automatically for us. Another quite useful concept is called **load balancing**. It is a method of distributing requests to a website across multiple web servers decreasing the overall load on each machine.

1.5 Previous Similar Studies and Work

In this section, we will be analyzing and discussing similar studies done by other web developers and programmers. Before we look at their results, we need to have a basic understanding of the web serving software they were comparing.

Apache HTTP, also called Apache, is a free and open source world's most widely used (57.9%[29] of all websites whose web server we know) web server software. **Nginx**, released nearly 10 years later than Apache, was designed with a high concurrency, high performance and low memory usage in mind, specifically to solve the C10K problem [14].

Both **PHP** and **HHVM** are PHP script interpreters. HipHop Virtual Machine (HHVM), released by Facebook in 2011, was developed to increase the performance of PHP script execution on the Facebook site. Both of them are open source and free to use.

1.5.1 Using Nginx, Apache, APC and Varnish in Different Scenarios — Garron.me

Guillermo Garron, the author of the work [7], has performed several comparisons between the most popular web-serving and caching softwares. He used a relatively weak web server with highly limited computing power — 512MB RAM, shared CPU and shared disk. However, the result of his experiment is clear — caching the output of the PHP interpreter has a considerable impact on loading times of WordPress-powered web page.

From the figure 1.4, we observe that if the number of simultaneous visitors exceeds ten, response times of the web server start to dramatically decrease in a linear fashion. On the other hand, response times start to worsen only after thirty concurrent visitors, as we can see from the figure 1.5.

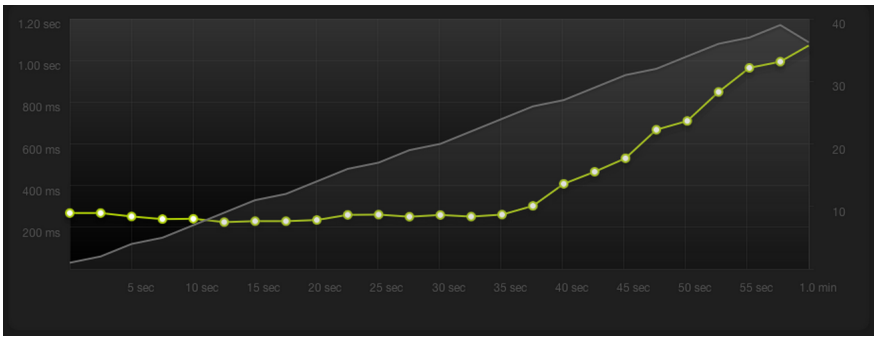

Figure 1.4: Apache HTTP + PHP, no opcode caching

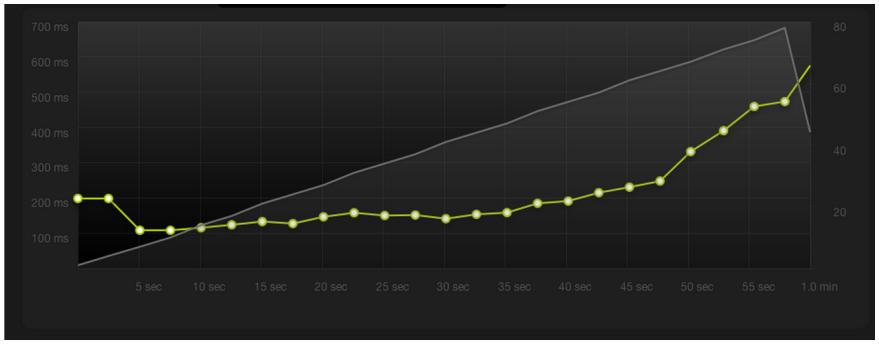

Figure 1.5: Apache HTTP + PHP, APC opcode caching

Although using Nginx instead of Apache HTTP yields additional performance gains, they are less notable. Nginx strengths are demonstrated when a website is composed of many resources. However, Garron used a standard WordPress installation, with no extra plugins or complex custom themes.

1.5.2 WordPress on HHVM vs WordPress on PHP-FPM — WPengine.com

WPengine.com is a company specializing in offering WordPress web hosting services. They have introduced a new hosting plan which differentiates itself from others by using HHVM PHP interpreter instead of the standard PHP. Before releasing the hosting plan, a study in which the performance of HHVM vs PHP was compared was undertaken. The study concluded with a fact that HHVM increased the speed of their servers by 600% [23]. This fact is displayed on figure 1.6.

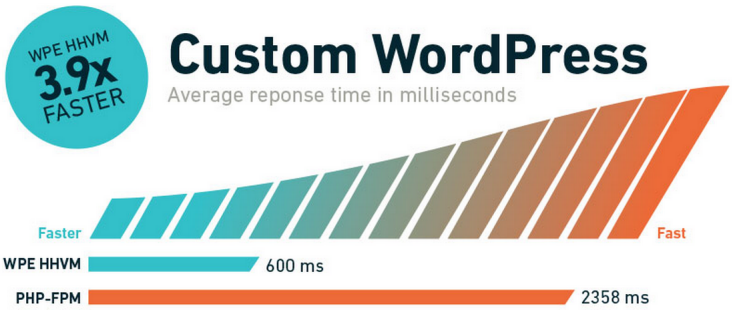

Figure 1.6: WordPress on PHP vs WordPress on HHVM response times

In the study, they found out that HHVM is not 100% stable yet and occasionally stops working. Their solution to this problem was to redirect the incoming requests to a fallback PHP interpreter while HHVM gets restarted and functioning.

1.5.3 WordPress HHVM vs PHP — xyu.io

Xiao Yu, a web developer, benchmarked [35] WordPress running on PHP vs WordPress running on HHVM. His findings show a similar pattern to the findings of WPengine.com in section 1.5.2. Outcome of his experimentation can be observed in table 1.1.

	Response Time	*Ok Responses*	*Errors / Timeouts*
Anon PHP	4.091	8,939	0.94%
Anon HHVM	2.122	18,308	0.00%
Change	48.1%	**2.05X**	
Auth PHP	20.688	457	74.17%
Auth HHVM	14.359	1,242	43.45%
Change	30.6%	**2.72X**	

Table 1.1: WordPress on HHVM vs WordPress on PHP — xyu.io

”In the numbers above anonymous requests represents hits to various pages without a WordPress logged in cookie which are eligible for Batcache caching whereas authorized requests are hits to the same pages with a login cookie thus bypassing page caching.”[35]

2. Configuring testing environment

2.1 Server parameters

The benchmarking and WordPress optimizations will be done on a Ubuntu-based VPS server. Ubuntu comes with a prebuilt software packages (through apt-get package handling utility), therefore it is quick and quite easy to install and set up applications we will use.
We are going to use vpsfree.cz [28] as the server provider. However, content of this work can be reproduced on any kind of modern Ubuntu hosting platforms such as digitalocean.com [25].
Specification of the testing server are as follows:

- 3x CPU
- 4GB RAM
- 300Mbps connectivity

We will be using Ubuntu version 14.04.

2.2 Ansible automation

Before benchmarking the performance of various web serving software, we have to install and configure the server. However, if done manually, it is a tedious task and takes a lot of time. One of the solutions to this problem is to use an automation tool.

Ansible is a modern IT automation tool. It is a command line application which runs from a host computer, executing commands on remote servers. The main parts of Ansible are tasks and modules.

Ansible Task has a name describing, in short, what it does. The task runs single Ansible module which performs some operations on the server. These modules are subprograms that can usually take several arguments, altering its behavior.

For better organization, tasks can be grouped into roles and playbooks. Ansible Role is a logical container for multiple related tasks aiming to accomplish a single goal, like installing and setting up Nginx. An Ansible Playbook is a file specifying roles which should be run on specific servers. These servers are listed in a hosts file located within the Ansible project directory.

Ansible files (except for the hosts) are written in the YAML syntax. It is straightforward to read and understand. We have prepared several playbooks which will provision the whole server so it can be benchmarked. They are located in the appendix of the work.

2.3 Installing required software on the server

We are going to install all the required software on our testing server. Copy the attached wordpress-ansible folder or clone its GitHub repository to your local computer. In the folder, we can find a few playbooks, hosts file, variables and roles. We will be referring to this folder by the name ”wordpress-ansible” in the work.

The first step is editing the hosts file. Put the IP address or hostname of your testing server in the file and assign it to the webservers group.

Secondly, we need to install Ansible on our local computer. Official Ansible documentation [12] explains how to do it.

The next step is to edit the group variables for our testing server. Open the group_vars/webservers file and modify it to your needs. If you are satisfied with the default values, there is no need to change it.

After completing all the previous steps, navigate to the wordpress-ansible folder in your command line and execute following command:
ansible-playbook -i hosts install-all-software.yml
During the installation process, Ansible will inform you about the status of executed tasks, whether they went successfully or not. ¡information about success¿. We are now ready to benchmark web serving software.

2.4 Using loader.io for load testing

Apart from having a server running WordPress-based site, we need a tool for load testing the server. Initially, we used a popular command line utility called ”ab” (standing for Apache Bench). However, this approach also required a second server from which the ab utility would be run. What is more, storing the tests results and plotting them on charts with ab is not trivial.

We found a simple cloud-based load testing service called ”loader.io” [3]. In a subscription, free of charge, we get 10,000 clients (requests) per test and one target host. For our purposes, these parameters are enough.

After creating an account at loader.io and adding a target host (our testing VPS server), we are presented with a dialog to verify the ownership of the server. While putting the token manually in a text file to our WordPress site directory will allow loader.io to verify the server, it is better to come up with an Ansible playbook which would automate this task for us. Below is the YAML definition file of the playbook.

```
---
- name: Loader.io host verification >

  hosts: webservers
  remote_user: root
  vars:
    token: <YOUR-TOKEN-HERE>

  tasks:
  - name: Install loader.io verification token
    shell: echo {{ token }} > {{ token }}.txt
    args:
      chdir: "{{ remote_wordpress_dir }}"
      creates: "{{ remote_wordpress_dir}}/{{ token }}.txt"
```

Replace ¡YOUR-TOKEN-HERE¿ placeholder with your verification token and save the playbook as ”loader.io-verification.yml”. Whenever we provision a new testing WordPress site, we only need to run the loader.io-verification.yml playbook to have the token inserted to the site.

3. Benchmarking server software

The first step is to compare the performance of Apache and Nginx web serving software. However, it is not necessary to compare these software in their speed of serving static content (for example HTML files) because it has been proven and because of Nginx architecture that Nginx is more performant in this manner.
What is interesting to compare, though, is running of PHP under Apache and Nginx.

3.1 Apache with mod_php

”Apache supports a variety of features, many implemented as compiled modules which extend the core functionality.” ”Instead of implementing a single architecture, Apache provides a variety of MultiProcessing Modules (MPMs), which allow Apache to run in a process-based, hybrid (process and thread) or event-hybrid mode, to better match the demands of each particular infrastructure.” [4]

In a default configuration, when HTTP requests are received, Apache starts to process them one by one. It spawns multiple child processes to handle the load. However, these proccesses are standalone, meaning they initialize all modules (including mod_php) even for static file requests. This behavior results in high RAM usage as seen on figure 3.1.

mod_php is a PHP interpreter module for Apache. It is the most common way of executing PHP scripts when using Apache HTTP. The module is loaded in each Apache process, thus a request for a PHP script is executed directly within the process, not being relayed to another server with the ability to run the script. The main disadvantage to embedding PHP inside Apache is that if a user is requesting a static resource such as CSS, JS or an image, it still has to go through Apache process with PHP module, therefore increasing RAM and CPU usage.

3.1.1 Load testing Apache with mod_php

We are going to load test Apache with mod_php running a simple WordPress-powered site. Within your command line, navigate to the wordpress-ansible directory and run the ”apache_mod_php.yml” Ansible playbook:

```
ansible-playbook -i hosts apache_mod_php.yml
```

The playbook will configure Apache and create a virtual host for the WordPress site. We are using a default Apache HTTP configuration file without any performance modifications. [2]

To deploy the simple WordPress site, run the ”wordpress_basic.yml” Ansible playbook. It will download the latest version of WordPress with the default theme and installs the database tables. We are now ready to carry out the testing.

At the loader.io load testing service, we have created a new test [5], configuring it to send from 0 to 200 simultaneous client requests to the index of our testing server for a duration of 30 seconds. Figure 3.1 depicts the final chart.

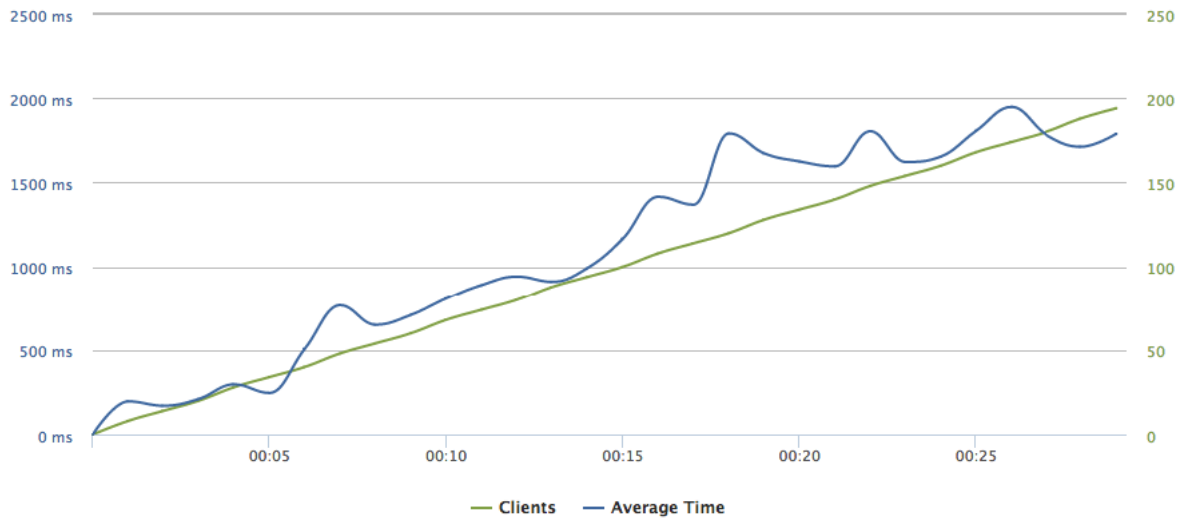

Figure 3.1: Apache HTTP with mod_php: clients versus average response time

This chart shows the average server response times when a number of simultaneous client requests are sent to it. We can observe that even under a high load of 200 concurrent requests the average response time is under 2 seconds. It grows approximately linearly.

However, to truly see how the server copes with the load, the figures 3.2 and 3.4 come in handy. They represent screenshots of Htop interactive process viewer [11] during 2 and 25 seconds into the load testing, respectively.

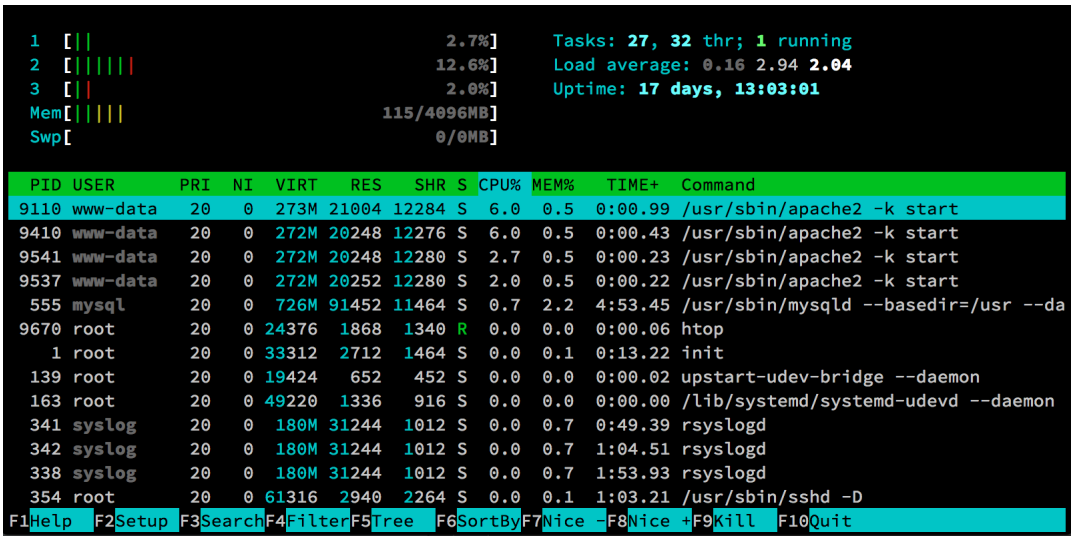

Figure 3.2: Apache HTTP with mod_php: Htop process viewer 2 seconds into test

On the screenshot, we can see a table of currently running processes. The columns RES, CPU% and MEM% show the RAM usage in kilobytes, CPU usage in percentage and RAM usage in percentage of a process, respectively. What is more, there are several gauges, visually depicting the usage or particular CPU core (or thread), RAM and Linux Swap. Opening the Htop's help screen (F1 in Htop), the gauges are explained as follows:

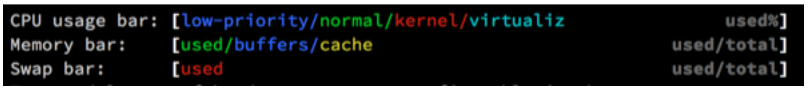

Figure 3.3: Screenshot of Htop's help screen explaining the main gauges

It is crucial to notice that the yellow bars in the Memory bar section show cached memory, which is not counted towards the used memory since it can be misleading at times (will discuss this a bit later).

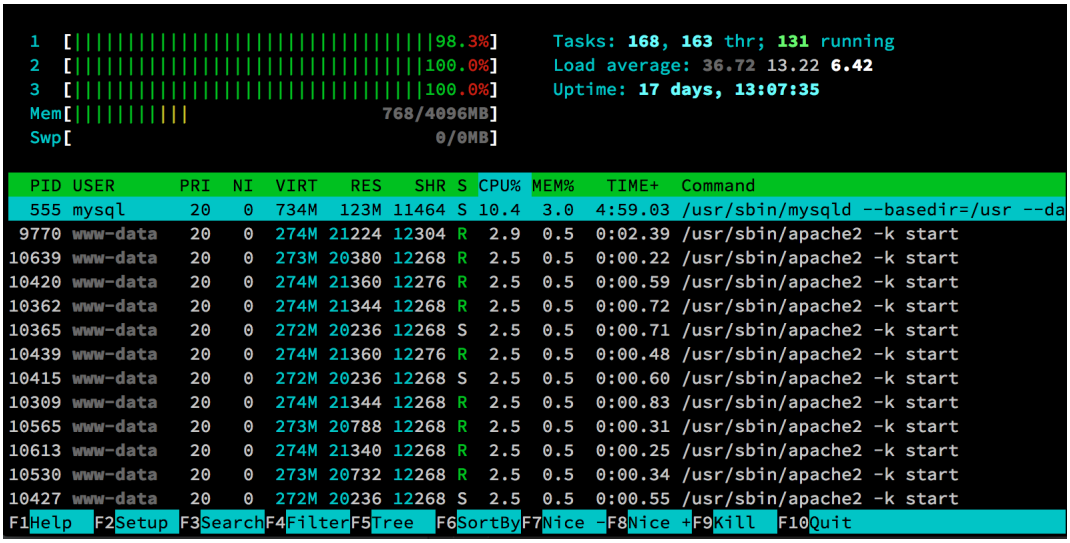

Figure 3.4: Apache HTTP with mod_php: Htop process viewer 25 seconds into test

From both screenshots, we can observe that Apache HTTP, represented by the process /usr/sbin/apache2 is, at first, just warming up, spawning three child processes and under heavy load, consuming the CPU completely and taking about 770 megabytes of memory. It does not tell us much without comparing it to the results of benchmarking different server stacks, what we are going to do shortly.

Loader.io testing results page [5] also shows us more statistics about the test:

- **Average response time:** 1229 ms
- **Min/Max response times:** 144 / 4660 ms
- **Count of successful responses:** 2178

3.2 Nginx with PHP-FPM

”Nginx (pronounced ”engine-x”) is an open source [...] web server. The nginx project started with a strong focus on high concurrency, high performance and low memory usage. [...] Nginx uses an asynchronous event-driven approach to handling requests, instead of the Apache HTTP Server model that defaults to a threaded or process-oriented approach, where the Event MPM is required for asynchronous processing. Nginx’s modular event-driven architecture can provide more predictable performance under high loads.” [19]

Due to Nginx’s asynchronous event-driven nature, its primary use case scenario is to relay the incoming requests to another application for processing, such as PHP-FPM, as quickly as possible, thus being able to handle large amounts of concurrent connections.

PHP-FPM is a FastCGI server bound to a TCP port or socket, bundled with the official PHP interpreter. It listens for PHP requests (request for a PHP script), responding with the resulting content (usually HTML). Comparing Apache’s mod_php with PHP-FPM, while a PHP request is processed by Apache directly, in case of Nginx, it has to be relayed to PHP-FPM, which is able to process it. We do not have to know how it works in a greater depth to understand that when a request is flowing through multiple applications, more server resources have to be consumed. On the other hand, as Nginx and PHP-FPM are specialized applications, they are more efficient when put together. It can be observed on the figures below.

3.2.1 Load testing Nginx with PHP-FPM

Within your command line, navigate to the wordpress-ansible directory and run the ”nginx_php-fpm.yml” Ansible playbook:

```
ansible-playbook -i hosts nginx_php-fpm.yml
```

Analogous to the load testing Apache with mod_php, we create a new test on Loader.io with the same configuration. [18]

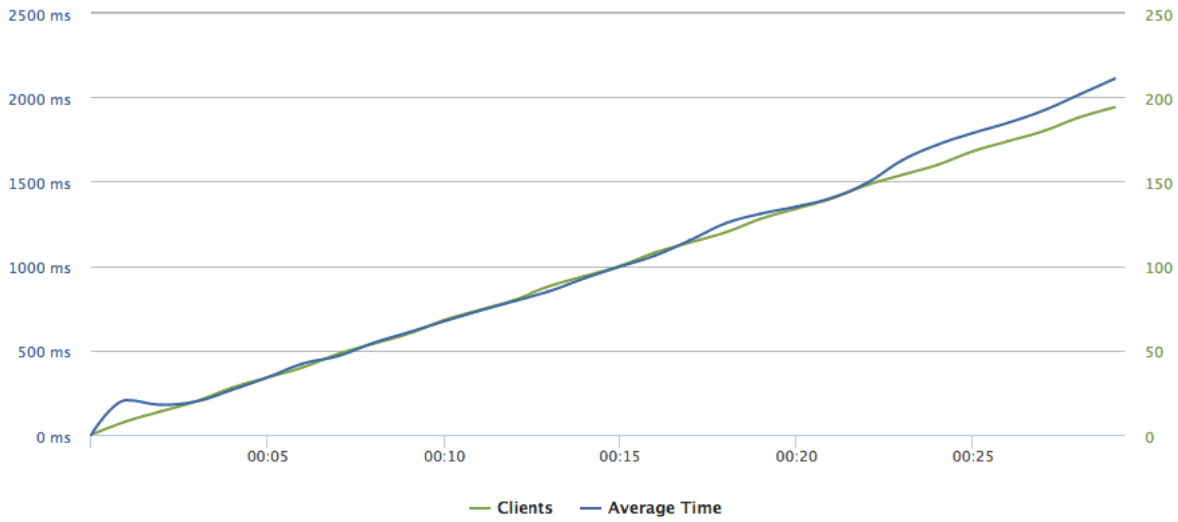

Figure 3.5: Nginx with PHP-FPM: clients versus average response time

Figure 3.5 shows us similar average response times to the figure 3.1 (Apache with mod_php), even slightly worse. However, screenshots of Htop during 1 and 22 seconds into the test depict a different situation. As we can see, even under 150 simultaneous client requests, server RAM usage is only at 46 MB. Compared with Apache’s 770 MB, by installing Nginx with PHP-FPM we optimized the server markedly. We can also see that Nginx is not using more than 3% of the server’s CPU for the reason that it was programmed to manage thousands of concurrent connections and we sent only 200 of them. Better benchmark for Nginx performance is described in the 4.2 section.

Although we could tweak Apache with mod_php to have a better performance, it has a steep learning curve, requiring a considerable amount of time. The objective of this work is to provide web administrators and WordPress developers with solutions they can easily implement, therefore we will not elaborate on this topic in greater detail.

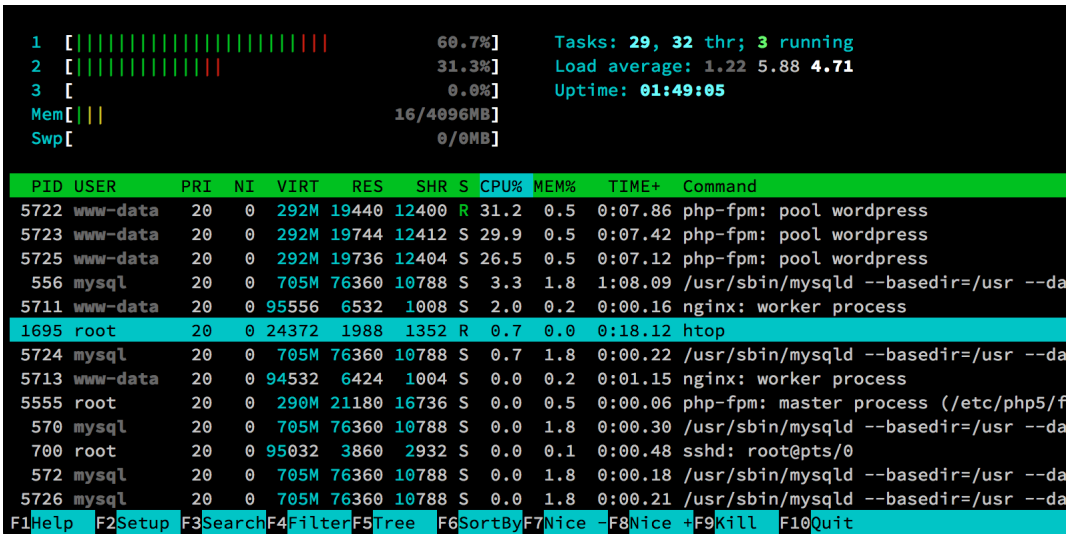

Figure 3.6: Nginx with PHP-FPM: Htop process viewer 1 second into test

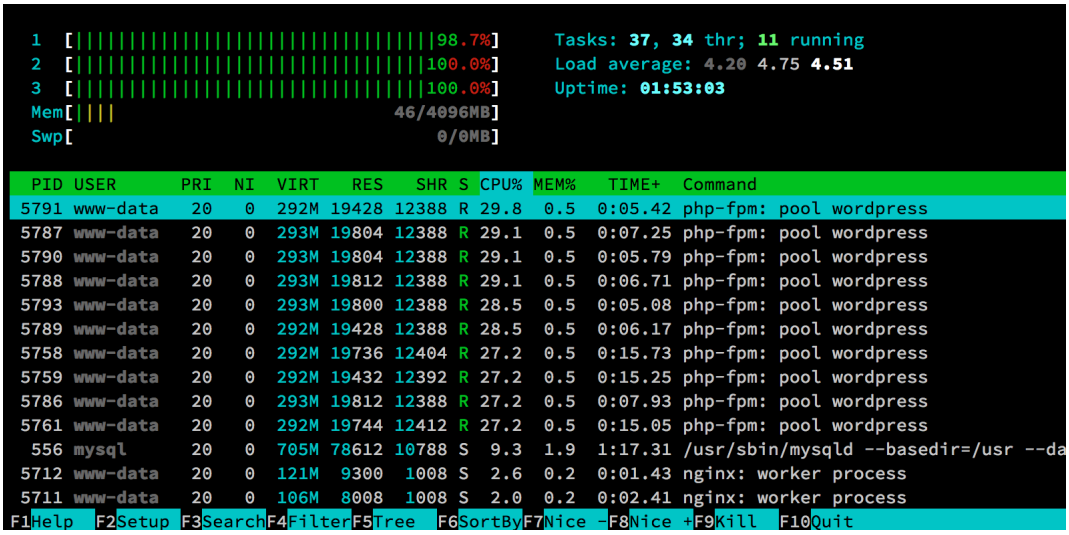

Figure 3.7: Nginx with PHP-FPM: Htop process viewer 22 seconds into test

Astute readers might notice that MariaDB process (/usr/sbin/mysqld) consumes more memory (RES column) than Htop's Mem gauge total used memory is showing (46 MB) alone. As we have explained from the figure 3.3, memory bar is not counting the cached memory to its total used value. The only indication is the yellow bars which follow the green ones.

3.3 Nginx + HHVM

In this section, we are going to substitute PHP-FPM with a different FastCGI PHP interpreter, HHVM and load-test it as we did with the earlier technologies.

”HipHop Virtual Machine (HHVM) is a process virtual machine based on just-in-time (JIT) compilation, serving as an execution engine for PHP. [...] By using the principle of JIT compilation, executed PHP [...] code is first transformed into intermediate HipHop bytecode (HHBC), which is then dynamically translated into the x86-64 machine code, optimized and natively executed. This contrasts to the PHP’s usual interpreted execution, in which the Zend Engine transforms the PHP source code into opcodes as a form of intermediate code, and executes the opcodes directly on the Zend Engine’s virtual CPU.”. [9]

3.3.1 Load testing Nginx with HHVM

Within your command line, navigate to the wordpress-ansible directory and run the ”nginx_hhvm.yml” Ansible playbook:

```
ansible-playbook -i hosts nginx_hhvm.yml
```

Analogous to the load testing Apache with mod_php, we create a new test on Loader.io with the same configuration. [16] As HHVM is transforming and optimizing the PHP code during the initial requests, we have made several ones before running the load testing to get more accurate results (response time for the initial request was more than 10 seconds long).

Figure 3.8: Nginx with HHVM: clients versus average response time

Reviewing the above chart, we can see that average response times have decreased from the 2-second to under 1-second levels. When 200 simultaneous requests are sent to the server, we receive them after 800 ms in average. This means that just by using HHVM instead of PHP-FPM, the performance of our server increases 2,5 times. What is more, the count of successful responses leaps to around 5800, thus nearly tripling the throughput of the server. [16]

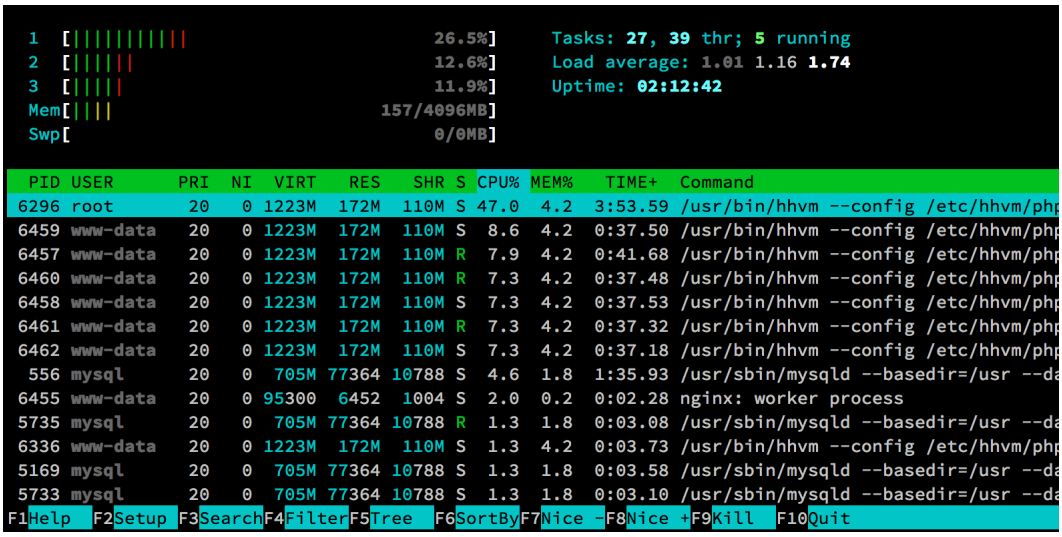

Figure 3.9: Nginx with HHVM: Htop process viewer 1 second into test

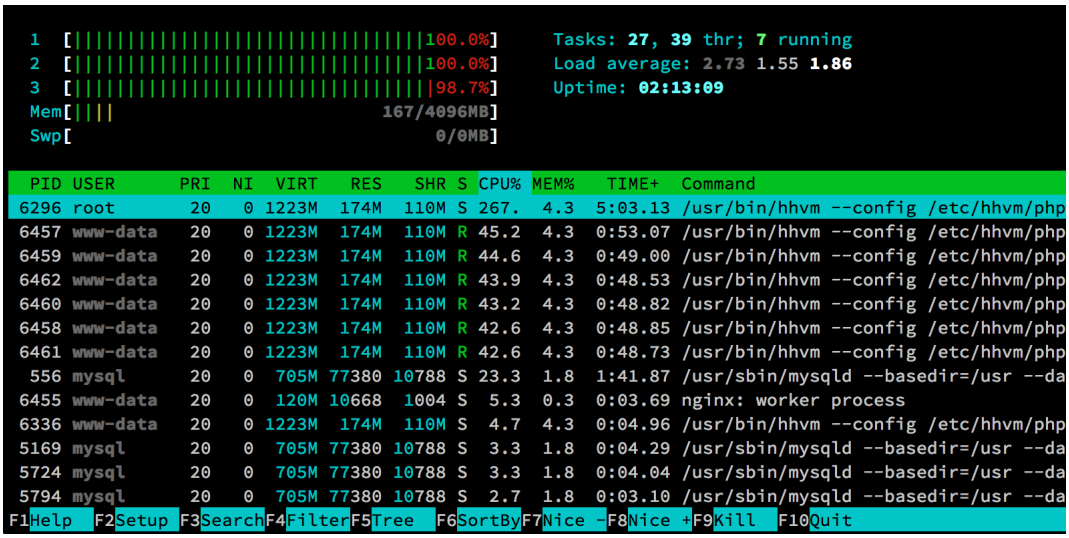

Figure 3.10: Nginx with HHVM: Htop process viewer 22 seconds into test

Looking at the Htop process viewer screenshot, in comparison to using PHP-FPM, RAM usage more than tripled. However, it is still relatively low, especially compared to the memory usage when using Apache with mod_php.

If this was a competition, choosing Nginx with HHVM as your web-serving stack would beat all its contestants, therefore we recommend going with it.

3.3.2 Advanced WordPress and HHVM

Our standard test did not benchmark HHVM's full potential, as we could observe from the figure 3.8. We will make the WordPress-powered site more complex. The new installation will include:

- several plugins, such as WooCommerce, Jetpack, WordPress SEO by Yoast and others [1]
- a free e-commerce theme Storefront [26]
- WooCommerce dummy data
- WP Test dummy data [33]

In order to have the advanced WordPress configured, we run the ”wordpress_advanced.yml” playbook:

```
ansible-playbook -i hosts wordpress_advanced.yml
```

We configured the Loader.io test to send requests to two different URL addresses, the root of the server and to ”?product=woo-ninja-3” (a WooCommerce product page). [17]. The results can be seen on the figure 3.11. Average response times decreased from under 1-second to around 7-second levels, which is also reflected in the count of succcessful responses (1237 versus 5769). Installing complex WordPress plugins and themes has clearly an extensive impact on the site’s performance.

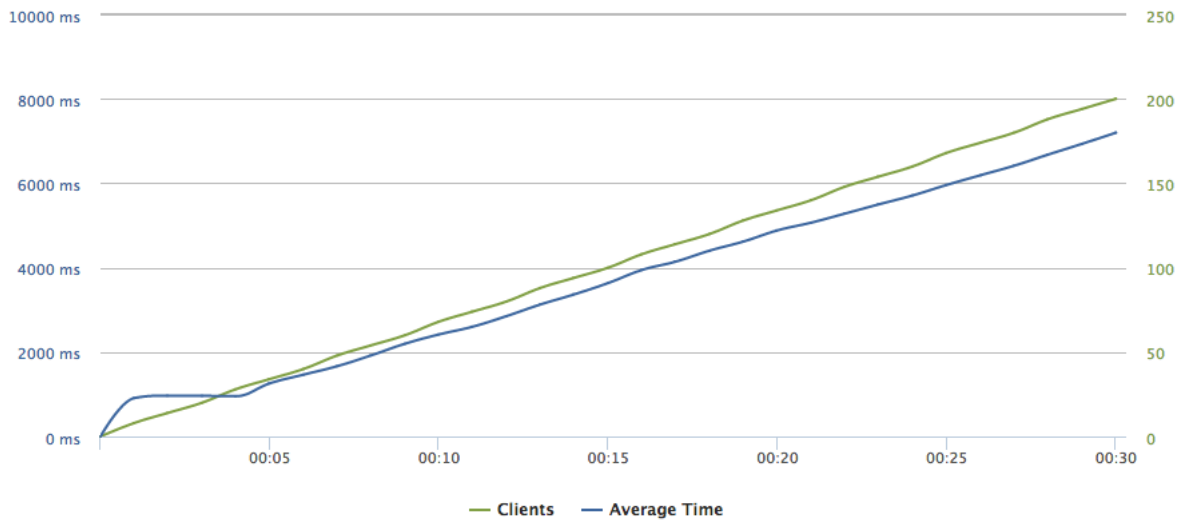

Figure 3.11: Nginx with HHVM and advanced WordPress: clients versus average response time

4. Caching

4.1 Database caching

WordPress-powered sites are doing quite a lot of queries into the database on each load. If an user installs plugins and uses a more complex theme, querying the database starts to become a bottleneck. It can take up to 200-300 milliseconds from each page load. When there is a high traffic coming to a site, database stops being able to process all the queries and PHP processing halts (queries are synchronized).
A solution to this problem is to store (cache) the results of the database queries so when the same query is run again, the stored data, instead of querying the database, is returned.
WordPress comes with a object cache API in its Core. All Core functions and methods querying the database (such as get_option, get_posts, etc) use this object cache API. However, as WordPress supports wide range of hosts and server versions, this API caches the data only during the request procesing. That means that when the data are queried for the first time, they get stored in a PHP array. Next time they are retrieved (during the same request processing) from that array.
Fortunately, native WordPress object caching API can be swapped for more robust caching mechanisms. It is done by writing an object-cache.php file, putting it into the wp-content directory. If WordPress detects this file, it will use the functions defined from it instead of loading its own object cache API code.
There are two main caching mechanisms: WordPress Transient API and in-memory caching. WordPress Transient API works by storing the cached data back into the database as a serialized PHP array. Instead of having to perform a complex database query, the data is simply retrieved from the database and unserialized back into a PHP array. The downside of this approach is that the server database (MySQL for example) is still used to store data, so it's still queried, data inserted and then deleted. This slows down the server and might even hurt the performance in case of a large number of tiny data stored in the database.
Another approach — in-memory caching — is the most optimal one. In this approach, data is saved into the server's RAM memory and loaded to the PHP process back from it. This is the fastest way of caching database data and queries as RAM is the fastest type of memory in your server. The downside is that it uses additional megabytes of RAM. Fortunately, the numbers are somewhere between tens of megabytes, which, in todays' world, is not that

much. On the other hand, this approach saves the trip to the database completely, eliminating the 200-300ms and increasing the performance of your site enormously.
The two most popular in-memory storage applications are memcached and redis. To compare memcached with redis, redis is the newer, more performant and optimized one. That's why we'll be using Redis in our work.
Redis is a simple key-value in-memory storage mechanism. It has an API to operate with the data. To work with Redis from PHP, you need a module. Then, it is enough to simply copy-paste a pre-made PHP Redis api for WordPress and put it into your wp-content directory.
Using Ansible automation, the author of the work has prepared a role, named Redis, to install and configure WordPress and your server to use it. Simply run wp-redis.yml playbook from your command line and your server will be Redis-ready.
To test the performance, we are going to use the loader.io.
From the above graph, we can see that the response time has decreased dramatically initially. However, as more simultaneous requests are made to the WordPress-based testing site, we can see that the performance is worsening. That's because database caching solves only a part of the problem.
Similar results to that of without using Redis for database caching, however. Might be caused because HHVM does some kind of object caching on its own. On the other hand, loading page from user's browser seems much faster, especially many different subpages of a site. HHVM has to cache all of that subpages before it starts to be fast again. Database is not bottleneck, too few requests for it to be. CPU is bottleneck. If you had multiple servers, many users, having a single in-memory database such as Redis would be beneficial.

4.2 Page caching

To improve the performance even more, we can actually cache the result of the PHP processing of our site. We can save the resulting HTML file on a disk or to a RAM memory and when the next request arrives, just output the cached page. Full page caching can be done on several abstraction levels. The easiest one to set up is to have a WordPress plugin construct a static HTML cache of each requested web page on your site and store it as a flat file on your server disk drive. The problem with this approach is that there still has to be some kind of routing done on the PHP level as Nginx doesn't know which file to load on a request. Hard-drive cna also easily become a bottleneck if a lot of concurrent clients are loading the site, thus reading the file.
Better solution is to have the page cache done on a lower level, the Nginx one. Nginx has a fastcgi cache module for exactly this purpose. We can configure Nginx to store the output

from PHP processor into the RAM (tmpfs file system). When a new request comes, Nginx checks whether there already is a cached page or not. If it is, it will return it back to the user as a response. If it is not, it will forward the request to the PHP listening on the FastCGI server. When the resulting HTML file gets back to Nginx, it will store it in the page cache on the RAM for later usage.

This process is rather fast, as Nginx is able to respond to thousands of concurrent requests, as seen from the chart below. Nginx uses a tree-like structure to store the data with hashing mechanisms, thus increasing the performance.

In order to revalidate and purge the old data, a Nginx location directive can be added. This directive can then be called from within WordPress to purge the cache if new content was added to our site. There is a handy plugin called Nginx Helper from rtcamp which automatically purges the cache on new post or page addition.

Run nginx-page-cache.yaml playbook to have your VPS server fully configured with Nginx FastCGI page caching.

The downsides of page caching are that if a cached page is updated – just a small part of it — you need to purge it from the cache and re-load it. If a person is logged in WordPress, it's not possible to cache most pages because they are customized for the particular user. Solutions such as CacheBuddy are specially made for this purpose.

Better than W3 Total Cache page caching because it goes directly into RAM, can skip caching if cookie is present or specific location, relying on nginx caching mechanisms, fast, can be purged automatically on new post/page, etc.

Figure 4.1: Nginx with FastCGI caching: clients versus average response time

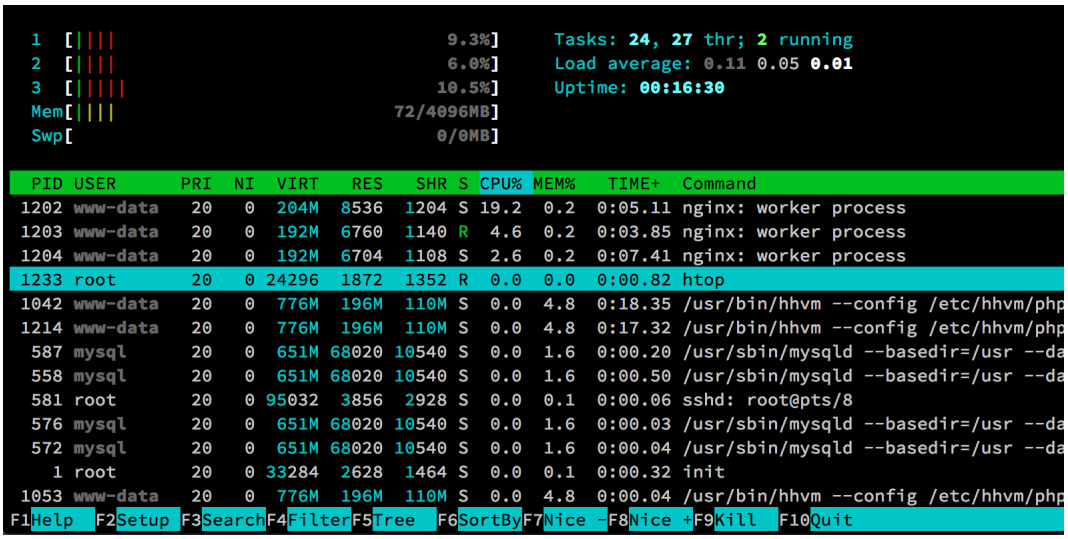

Figure 4.2: Nginx with FastCGI caching: Htop process viewer 9 seconds into test

4.3 Browser caching

Browser caching is the process of storing data in the client's browser memory. If we store the resources (CSS, JS, images and fonts) and HTML pages in the client's browser cache, the browser doesn't have to load the resources from our servers, therefore saving time and bandwidth and server processing power.

The main disadvantage of browser caching is that if addition to the resources were added or content changed, we need to revalidate the cache somehow. As the cached resources are not loaded from our server, we need to do it some other way. The preferred way to purge browser cache is to rename the resources so the browser will see them as new files which it has to load again.

To set caching, we need to specify caching options and expiration date as headers when serving files from Nginx.

Ansible playbook..

5. Client-side performance optimizations

5.1 Measurement tools

To measure how well we do client-side optimizations, we need to be comfortable with using some tools. There are two well-made online apps for this purpose: gtmetrix.com pagespeed.com
GTMetrix measures these things: - ..

5.2 Assets minification and concatenation

WordPress
has an API for working with assets, wp_register_script/style, wp_enqueue_script/style. By using the API, we can collect all the assets and modify them before returning the HTML output. To reduce the size of the assets as well as reduce the roundtrip redundancy (latency), we can also concatenate the assets into fewer files. In order to do this automatically, the best approach is to use a plugin. One of the most used plugins for minification and concatenation is W3 Total Cache.
After installing W3TC, open up the Minification page. As we can see, there are several options: - -
Using some themes, we are able to perform automatic minification and concatenation without any problems. However, sometimes we have to manually select the scripts and styles to minify, trying one by one, seeing if it breaks the site or not.
Lastly, we need to modify the Nginx rules to accomodate the new minified files.

5.3 Assets compression

We can also compress assets with Gzip compression. Most modern browsers support it by default. To have the resources compressed, we need to add these rules into Nginx configuration file.

5.4 CloudFlare Content delivery network

CDN is a great way to save resources of your server, redistribute the assets across the world (lower latencies) and have a DDOS protection. CloudFlare is a solution for this. They offer a free plan with full page caching, images caching and compression, DDOS protection, as well as minification which we turn off.

6. Source code performance optimizations

6.1 WordPress architecture in brief

WordPress is based on event-driven architecture, observer pattern. There are filters and action on which functions can be hooked during the execution time. This way, WordPress core, themes and plugins as well can be altered in their behavior.
To get the most performance of WordPress, we need to utilize the most suitable hooks for the function. For example, if I want to perform an operation on a custom post type when saving it (transitional state), I could hook to a post transitional generic function hook. On the other hand, I can hook into a specific action executed only for the specific post type, thus saving resources and time.

6.2 Profiling web application with xhprof

Xhprof is a PHP-based web application used for profiling your codebase. On each request, xhprof analyzes the callstacks, functions and computes all the time and memory it takes to execute a function.
¡picture¿
¡features¿
¡how to install/use¿

6.3 Using AJAX in plugins and themes

Another useful technique for increasing the performance of your WordPress-powered web app is to let the client perform some computations. We can output a page and compute additional data through ajax dynamically, thus the site will appear faster to the end user. We can also offload some computations to the client-side, such as getting external data, etc.

7. Concluding remarks

8. Future work

8.1 Load balancing

Nginx has a useful module called upstream. It can be used to load-balance servers, redirecting traffic to different VPS servers when needed.

8.2 Better plugin and theme architecture