Community Experience Distilled

Magento Site Performance Optimization

Leverage the power of Magento to speed up your website

Mathieu Nayrolles

[PACKT] open source*
PUBLISHING
community experience distilled

Magento Site Performance Optimization

Leverage the power of Magento to speed up your website

Mathieu Nayrolles

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Magento Site Performance Optimization

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: May 2014

Production Reference: 1140514

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78328-705-5

www.packtpub.com

Cover Image by Jarek Blaminsky (milak6@wp.pl)

Credits

Author
Mathieu Nayrolles

Reviewers
Bartosz Górski
Eugene Ivashin
Jaspal Singh

Commissioning Editor
Ashwin Nair

Acquisition Editor
Mohammad Rizvi

Content Development Editor
Ruchita Bhansali

Technical Editor
Tanvi Bhatt

Copy Editor
Roshni Banerjee

Project Coordinator
Puja Shukla

Proofreaders
Simran Bhogal
Linda Morris

Indexer
Hemangini Bari

Graphics
Ronak Dhruv
Valentina Dsilva
Disha Haria

Production Coordinator
Alwin Roy

Cover Work
Alwin Roy

About the Author

Mathieu Nayrolles was born in France and lived in a small village in Côte d'Azur for almost 15 years. He started his computer science studies in France and continued them in Montréal, Canada, where he now lives with his wife. He holds two master degrees from eXia.Cesi (Software Engineering) and UQAM (Computer Science). He is now a PhD. student at Concordia University (Electrical and Computer Engineering), Montréal, Canada, under the supervision of Dr. Abdlewahab Hamou-Lhadj, where he tries to improve the bug fixing process.

Along with his academic journey, Mathieu has been consulting as a Magento Performance Specialist since the development of Magento 1.6 (August 2011) and has also worked for worldwide companies such as Eurocopter or Saint-Gobain, where he learned how important good technical resources are.

You can discover more about his work by referring to other texts he has written, such as *Instant Magento Performance Optimization How-to, Packt Publishing* (February 2013), *Mastering Apache Solr,* or its latest realization at `http://caramboles.fr`.

You can find even more information on his personal website: `http://mathieu-nayrolles.com`.

About the Reviewers

Bartosz Górski is a Magento Certified Developer Plus and a Magento Certified Frontend Developer. He's been working in the web development/programming field for over six years, and has over three years of experience in developing only for the Magento e-commerce platform.

Bartosz is a big fan of doing things the right way, so he always aims to write as clean and efficient code as possible. He's always happy to give and receive feedback on how a given piece of code can be improved.

When he's not at work, he's probably playing pool somewhere, or he's sitting at home, browsing camera lenses on eBay and complaining how little time he has to actually go outside and take some photos himself.

> I'd like to thank my wife for her love and support.

Eugene Ivashin was born in Russia and currently resides in Kiev, Ukraine. He graduated from Dnepr State Academy of Building and Architecture in Dnepropetrovsk, Ukraine, and he got a diploma with distinction as a Construction Industry Automation Engineer in 1997.

Eugene worked at the South Ukrainian Nuclear Power Plant (SUNPP) as a repair engineer for the next six and half years and got interested in web development at that time. By the end of 2003, Eugene left SUNPP, headed to Kiev, and became a web programmer in a small private web design agency. There, he spent more than two years building websites for various customers and growing into a project manager in the process. At the same time, he continued freelancing for various customers and participating in a few sole proprietorship companies in the area of web development and services. From April 2008, for the next one and half years, Eugene worked as a web developer and technical support at ExpoPromoter, a company that leads in the trade show industry and provides a large catalog of trade show organizers across the world.

In January 2010, Eugene entered Varien Inc., which was rebranded as Magento Inc. afterwards, as a software engineer, but later became a technical trainer. When Magento was acquired by X.Commerce, an eBay company, in August 2011, Eugene became a training manager responsible for providing technical knowledge to all X.Commerce employees. Since then, Eugene decided to return back to software development and is now a Senior Backend Engineer at Vaimo, Magento Gold Partner.

Eugene speaks Russian, Ukrainian, and English. He likes to read science fiction, admires the fine arts, and loves to draw and sketch in his free time.

Jaspal Singh is a technology evangelist with more than 15 years of professional experience in the IT sector. He has hands-on as well as strategic-level experience of working on leading-edge technologies, such as PHP, Solr, Redis, Node.js, and MongoDB.

Jaspal has also been a fairly active tech entrepreneur, with engagement in many web applications and web portals. In his spare time, he likes to read and update himself about the latest technologies and trends in the IT space.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

PACKTLiB™

http://PacktLib.PacktPub.com

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

In the open source e-commerce platforms community, Magento has established itself as the most popular e-commerce platform in the market. Indeed, it is supported by a very strong developer community and it fits most of the usual and unusual needs.

The Magento Inc. website claims that Magento powers more than 2,00,000 businesses around the world. That's a 62.5 percent progression compared to December 2012. Even for a great open source and free application, 62.5 percent a year is pretty high. This trust is well-earned, as dozens of worldwide companies use Magento in their day-to-day online sales. Among the most known, we can find Nike or Lenovo.

From the beginning, there are two versions of the Magento platform: the Community version and the Enterprise version. The Enterprise version has annual fees starting from USD 17,000 (there was a hike of 21 percent from November 2013, which is certainly related to the eBay acquisition of Magento in June 2013), while the Community Edition remains free. Recently, a third version came out. This version is named Magento Go and allows you to pay only for what you need. In this version, same as in the Enterprise Edition, Magento Inc. takes care of the hosting fees and bills you something ranging from USD 15 per month for 100 SKUs to USD 125 per month for 10,000 SKUs. While the Magento Go Edition could be a credible alternative to the open source one, considering that you will also have fees for hosting, it is likely that the Community version is the most used.

The only problem with the open source, or Community Edition, is that if you own a successful store, it will grow slowly—very slowly. When we wrote slow, we mean over 3 seconds to add something in the cart and over 2 seconds to display a single category overview. Why so? Magento is an advanced e-commerce platform, and advanced also means a complex platform. Like any complex platform for online businesses, Magento configurations must evolve to fit new user requirements and operational needs. The changes resulting from the increasing number of browsers and buyers may degrade the quality of service and the user experience of any e-commerce website. Of course, all these problems go away if you are ready to pay the hosting proposed by Magento Inc.

The optimization of an attractive commercial website is a complicated task that deserves time and knowledge. Moreover, the optimization is a critical point for all growing businesses, because a misconfiguration could make you lose money, a lot of money. Indeed, if your server is overloaded, even for a short period of time, a browser that wants to turn into a buyer will not be able to do it; and it's good to know that, on average, a dissatisfied customer will talk to 12 people about his bad experience while a satisfied customer will only talk to three. It's also noteworthy that search engines now measure the required loading time as a part of your indexation score.

Magento Site Performance Optimization has been designed to be a quick, handful, and easy-to-understand book for administrators and developers who want to improve the performances of their Magento server. This book will be a shortcut to save you from a huge amount of unsuccessful online researches and testing, by giving you the key to an efficient optimization. We spent hundreds of hours in building simple, step-by-step tutorials that anyone can follow along with their results. Indeed, for each section of this book, we will provide a summary of impacts that sums up how many milliseconds the section makes us win.

Our experimentations will be based on the latest release of the Magento Community Edition 1.8.0.0 (December 2013) with the Sample Data 1.6.1.0 provided by Magento Inc. As the sample data contains only a few SKUs and does not reflect your real store, we will also apply these modifications to our latest realization: caramboles. fr. Various tricks and tips exposed in this book will certainly work with earlier versions, especially Version 1.7, but we can't guarantee it for all of them. Moreover, if your Magento Community version isn't the latest one, the first step towards the optimization is to upgrade your website.

This book will teach you how to improve the performances of your desperately slow Magento Community Edition. Moreover, it will be a shortcut through unsuccessful Internet research and testing. By the end of this book, all the most popular and most effective practices that can be applied on your Magento for speed improvement will be explained to you. Moreover, all tricks and sections are composed of a step-by-step tutorial that we test against different Magento users (expert, advanced, and beginners), therefore, you don't have to be an experienced Magento developer to follow our steps and optimize your website. Also, if you already have been using Magento as a developer for years, you might find the beginning of sections trivial but don't forget that the end of the sections come with measurable optimizations. More specifically, in this book, we will learn how to pick the right hardware and how to configure web servers, PHP, and MySQL caches. To configure your backend correctly, in order to obtain performances, you might have a look at another book by Packt Publishing: *Instant Magento Performances Optimization How-to.*

What this book covers

Chapter 1, *Starting with the Right Hardware*, enables us to understand our needs in terms of horsepower. In addition, this first chapter will show us which criteria should dictate the choice of a trustee company to host your Magento. Finally, we will also learn how to handle more than we can physically handle by taking advantage of the localization and using CDN or external services.

Chapter 2, *Choosing the Best Web Server*, guides us through the installation of three major players in the web server market: Apache, lighttpd, and Nginx. We will also learn how to optimize each one of these three web servers and present the clear winner.

Chapter 3, *Tuning, Scaling, and Replicating MySQL*, makes us understand why MySQL is the bottleneck of our Magento applications and how to optimize it. To do so, we will use a set of dedicated scripts which are analyzing the usage of the MySQL database and advice for better configuration.

Chapter 4, *Caching Them All*, explains how to take advantage of different caching mechanisms in order to store—instead of recomputing—the requests of your customers, in terms of HTTP requests, PHP byte code, and random objects using Varnish, APC, and memcache, respectively. We also see how to use FPC and the Facebook HipHop Virtual Machine.

What you need for this book

In order to follow, without any difficulties, the technics and code sample included into this book, the reader should have a little knowledge about Linux commands such as `cd`, `ls`, `wget`, and so on. Also, the reader should have already installed a Magento server by him/herself in order to be familiar with the folder and file architecture of Magento.

Who this book is for

This book is written for Magento administrators who wish to optimize their store to increase the performance without having to spend USD 17,000 a year for the Enterprise Edition of Magento. It should be noted that Magento is a project with a gigantic code base and the tools to optimize it aren't trivial either; the readers must be willing to get their hands dirty and produce some code themselves.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Use the `ps:scale` command to scale up the web processes."

A block of code is set as follows:

```
#! /bin/sh
# /etc/init.d/syncache.sh
#

echo "Synching Magento Cache to  Hard drive"
echo [`date +"%Y-%m-%d %H:%M"`] Magento Cache Synched to Disk >> /var/
log/magento_ram_cache.log
    rsync -av --delete --recursive --force /var/www/YOUR_DOMAIN.
COM/var/cache/ /var/www/YOUR_DOMAIN.
COM/var/cache-backup/


exit 0
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold style:

```
DAEMON_OPTS="-a :80 \
             -T localhost:6082 \

             -f /etc/varnish/default.vcl \

             -S /etc/varnish/secret \

             -s malloc,256m"
```

Any command-line input or output is written as follows:

```
$ ab -n 100 -c 5 http://192.168.0.103/index.php/furniture/living-room.html
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Clicking the **List Action** button moves you to the next screen."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title through the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website, or added to any list of existing errata, under the Errata section of that title.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Starting with the Right Hardware

In this chapter, you will not have classic recipes but experimental experiences on how hardware changes can help your Magento server to run faster. By the end of this chapter, you might be surprised by what you will find out about hardware and performances.

Before upgrading your hardware, consider the following general points that fit any web hosting situation and that you should apply:

- **Get a dedicated server**: Most of the hosting enterprises will gather your website with hundreds of others. If you are on a mutual hosting platform, your performance will depend on the other websites' traffic.
- **Hosting country**: Choose a hosting company that owns servers in the country where your customers are, and not necessarily the best ones in the market.
- **Versions**: Always update your servers with the latest versions. With updated versions, you'll get features, security, and performances.
- **Images**: Always rasterize your images to the size you want them to be displayed and crop all white spaces. Go for PNG or GIF files instead of JPG files.
- **Extensions and modules**: Disable any Magento extensions or modules that you don't use.

Measuring performance

We measure the performance of each topic using various tools. Among them, the main ones are Pingdom tools, Mozilla Firebug, Google Speed Tracker, and Google Page Speed. Also, we stress our server using the small yet powerful Apache Benchmark. **Apache Benchmark** is a simple load testing tool that allows fake HTTPS requests to be generated. We will come back to that later, but as an example, the following command will test how fast `google.com` can handle 100 requests. Among these 100 requests, 10 are concurrent:

```
ab -n 100 -c 10 http://www.google.com/
```

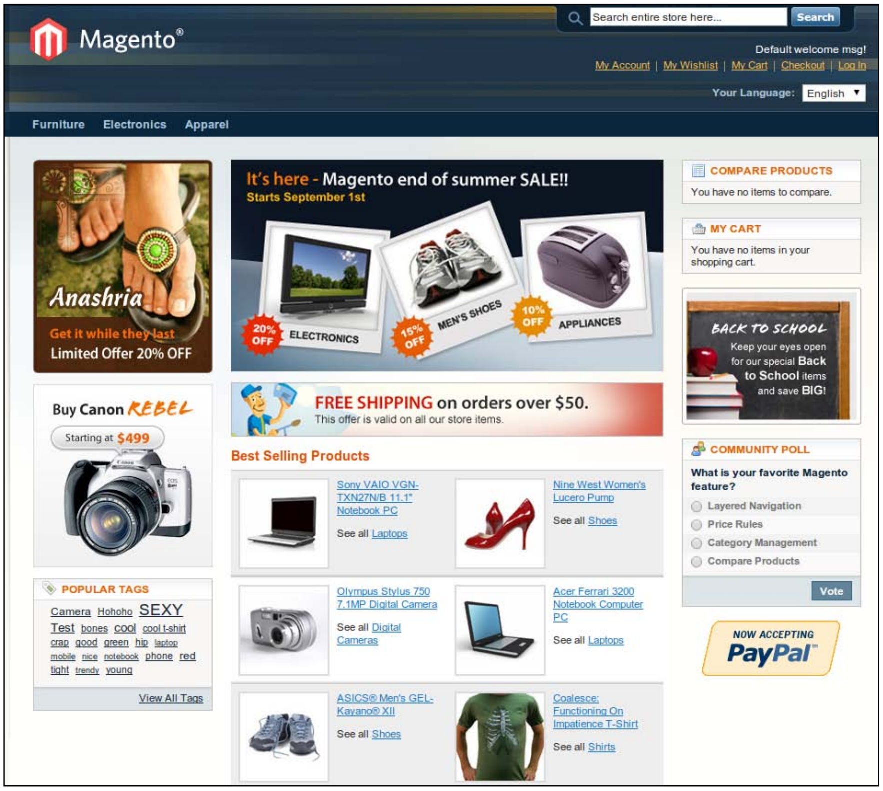
As expected, the `ab` part refers to Apache Benchmark, `-n` the number of requests and `-c` the concurrent request. At last, the target address comes. The result will be something similar to the following output:

```
Benchmarking www.google.com (be patient).....done
Document Path:          /
Document Length:        258 bytes
Concurrency Level:      10
Time taken for tests:   0.816 seconds
Complete requests:      100
Failed requests:        4

Total transferred:      108684 bytes
HTML transferred:       25792 bytes
Requests per second:    122.50 [#/sec] (mean)
Time per request:       81.635 [ms] (mean)
Time per request:       8.163 [ms] (mean, across all concurrent requests)
Transfer rate:          130.01 [Kbytes/sec] received
```

> If you don't have Apache Benchmark installed yet, you can have it quickly by entering the following command:
>
> `sudo apt-get install apache2-utils`

It is now time to introduce you to our test environment and the first result. We will use a fresh install of Ubuntu 12.04.3 server with the LAMP version that we can install when installing the system. The server runs with 512 MB of RAM, 8 GB of hard drive (SSD), and one CPU at 1.80 GHz.

If you are not familiar with the default Magento and the associated test data, the following screenshot shows Magento Community Edition 1.8 and the test data 1.6.1 as they look out of the box:


Magento®
Search entire store here...
Search
Default welcome msg!
My Account | My Wishlist | My Cart | Checkout | Log In
Your Language: English
Furniture
Electronics
Apparel
Anashria
Get it while they last
Limited Offer 20% OFF
It's here - Magento end of summer SALE!!
Starts September 1st
20% OFF
ELECTRONICS
15% OFF
MEN'S SHOES
10% OFF
APPLIANCES
COMPARE PRODUCTS
MY CART
BACK TO SCHOOL
Keep your eyes open for our special Back to School items and save BIG!
Buy Canon REBEL
Starting at $499
FREE SHIPPING on orders over $50.
This offer is valid on all our store items.
Best Selling Products
Sony VAIO VGN-TXN27N/B 11.1" Notebook PC
See all Laptops
Nine West Women's Lucero Pump
See all Shoes
Olympus Stylus 750 7.1MP Digital Camera
See all Digital Cameras
Acer Ferrari 3200 Notebook Computer PC
See all Laptops
ASICS® Men's GEL-Kayano® XII
See all Shoes
Coalesce: Functioning On Impatience T-Shirt
See all Shirts
POPULAR TAGS
Camera Hohoho SEXY Test bones cool cool t-shirt crap good green hip laptop mobile nice notebook phone red tight trendy young
View All Tags
COMMUNITY POLL
Vote
NOW ACCEPTING PayPal

You have no items to compare.

You have no items in your shopping cart.

What is your favorite Magento feature?

○ Layered Navigation
○ Price Rules
○ Category Management
○ Compare Products

> To install the test data, you have to download them at `http://www.magentocommerce.com/download` before installing Magento. The test data archive contains a media folder which must be present in the Magento directory, and a SQL script that must be executed on your database prior to the Magento installation.

Our first attempt at this configuration was with 10,000 request and 200 concurrent ones. Unfortunately, Apache didn't handle it and starts killing processes. Each time a request is submitted to Apache, it creates an associated process. Therefore, when Apache is killing processes because of the lack of memory, it flushes requests. After that, we have to restart the whole server in order to access to the website again.

Then, we tried with 1000 requests and 30 concurrent ones and the result was the same: Apache flushing requests. Finally, we had to do the initial test with 100 requests and only five concurrent ones. The following is the command we will run for the next chapters:

```
ab -n 100 -c 5 http://your.magento.store.com /index.php/furniture/living-room.html/
```

In the preceding command, `your.magento.store.com` is the default address of the store you want to test. We chose to stress the display of all the items belonging to the living-room category in order to perform a lot of database and PHP treatments.

The following results were not very encouraging:

```
Concurrency Level:       5
Time taken for tests:    110.396 seconds
Complete requests:       100
Failed requests:         83

   (Connect: 0, Receive: 0, Length: 83, Exceptions: 0)

Write errors:            0
Non-2xx responses:       83
Total transferred:       840645 bytes
HTML transferred:        792932 bytes
Requests per second:     0.91 [#/sec] (mean)
Time per request:        5519.809 [ms] (mean)
Time per request:        1103.962 [ms] (mean, across all concurrent
requests)
Transfer rate:           7.44 [Kbytes/sec] receive
```

As you can see, the time per request is 5519 ms. In other words, we have to wait almost 6 seconds in order to visualize the living room page.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you. You can also directly download the code sample from `http://mathieu-nayrolles.com/magento-optimization/`.

Evaluating your needs

Before choosing your hardware, you should know how any changes will directly impact the performance of your website. In the following sections, we will successively change the available amount of RAM and the number of CPU, and we will even over-stress our hard drives.

Doubling the CPU

Our first test will be to double the CPU in order to see whether the CPU should be in the balance while choosing your hardware. We will not double the frequency as we don't have a 3.6 GHz core to give at our **virtual machine** (**VM**); we will add another 1.8 GHz core instead.

While the server was under stress, we use the command on it in order to see the CPU usage. The `www-root` user, which is in charge of running the webserver, does use 100 percent of the available CPU, distributed across several processes. However, did the results improve? The default performance in terms of time and requests per second is shown as follows:

```
Document Path:          /magento/index.php/furniture/living-room.html/

Document Length:        36778 bytes
Concurrency Level:      5
Time taken for tests:   137.210 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Total transferred:      3725700 bytes
HTML transferred:       3677800 bytes
Requests per second:    0.73 [#/sec] (mean)
Time per request:       6860.487 [ms] (mean)
Time per request:       1372.097 [ms] (mean, across all concurrent
requests)
Transfer rate:          26.52 [Kbytes/sec] received
```

As you can see, `Time per request` reaches a new peak at `6860.487` ms. However, this time no requests have failed. Therefore, we can conclude that the CPU will not necessarily directly improve the needed time to load a page, but can drastically improve the number of served pages.

Doubling the RAM

In this second test, we use a single CPU 1.8 Ghz but double the RAM. The new amount of available RAM is 1 GB. The default performance, after doubling the RAM, in terms of time and requests per second is shown as follows:

```
Document Path:          /magento/index.php/furniture/living-room.html/

Document Length:        36778 bytes
Concurrency Level:      5
Time taken for tests:   94.596 seconds
```

```
Complete requests:      100
Failed requests:        0
Write errors:           0
Total transferred:      3725700 bytes
HTML transferred:       3677800 bytes
Requests per second:    1.06 [#/sec] (mean)
Time per request:       4729.795 [ms] (mean)
Time per request:       945.959 [ms] (mean, across all concurrent
requests)
Transfer rate:          38.46 [Kbytes/sec] received
```

The 512 MB of added RAM seems to have a very beneficial effect. As you can see, the time per request drops to 4.7 seconds and there no requests have failed.

Let's try to double it again. We are now at 2048 MB of dedicated RAM. If you check the server offered by your hosting company, that amount of RAM is rarely guaranteed on first prices:

```
Concurrency Level:      5
Time taken for tests:   96.854 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Total transferred:      3725700 bytes
HTML transferred:       3677800 bytes
Requests per second:    1.03 [#/sec] (mean)
Time per request:       4842.707 [ms] (mean)
Time per request:       968.541 [ms] (mean, across all concurrent
requests)
Transfer rate:          37.57 [Kbytes/sec] received
```

There are not really any improvements. In the actual configuration, to go over 1 GB is superfluous.

Doubling CPU and RAM

In this third test, we will stress a server with two CPUs at 1.8 Ghz core and 1 GB of RAM. The following is the performance after doubling the CPU and RAM:

```
Document Path:          /magento/index.php/furniture/living-room.html/
Document Length:        36778 bytes

Concurrency Level:      5
Time taken for tests:   58.580 seconds
Complete requests:      100
```

```
Failed requests:        0
Write errors:           0
Total transferred:      3725700 bytes
HTML transferred:       3677800 bytes
Requests per second:    1.71 [#/sec] (mean)
Time per request:       2928.976 [ms] (mean)
Time per request:       585.795 [ms] (mean, across all concurrent
requests)
Transfer rate:          62.11 [Kbytes/sec] received
```

Here, we can see some improvements! We are now under 3 seconds with 1 GB of RAM and two CPUs at 1.8 Ghz each.

Let's try with two CPUs and 2 GB of RAM:

```
Document Path:          /magento/index.php/furniture/living-room.html/
Document Length:        36778 bytes
Concurrency Level:      5
Time taken for tests:   55.333 seconds
Complete requests:      100
Failed requests:    0
Write errors:           0
Total transferred:      3725700 bytes
HTML transferred:       3677800 bytes
Requests per second:    1.81 [#/sec] (mean)
Time per request:       2766.668 [ms] (mean)
Time per request:       553.334 [ms] (mean, across all concurrent
requests)
Transfer rate:          65.75 [Kbytes/sec] received
```

Once again, there is not much improvement in doubling the available RAM. Indeed, we only drop from 2.9 seconds per request to 2.7 seconds.

With four CPUs, we might have enough horsepower to allocate our memory:

```
Document Path:          /magento/index.php/furniture/living-room.html/
Document Length:        36778 bytes
Concurrency Level:      5

Time taken for tests:   45.624 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Total transferred:      3725700 bytes
HTML transferred:       3677800 bytes
```

```
Requests per second:    2.19 [#/sec] (mean
Time per request:       2281.198 [ms] (mean)
Time per request:       456.240 [ms] (mean, across all concurrent
requests)
Transfer rate:          79.75 [Kbytes/sec] received
```

There is some improvement. We have now reached 2.2 seconds per request. We can conclude that we need horsepower to take advantage of our RAM. But it is definitely not worth paying for 2 GB of RAM, given the performances that it offers.

Measuring the impact of disk rate

As a reminder, our server has SSD disks and, almost certainly, you will not have these kind of disks. In order to simulate the slowness of classical mechanical disks, we will transfer files from a disk to the USB and copy files from the disk to another directory. The file we are moving to the USB is the Ubuntu virtual disk (ISO) of size 697 MB, while the files being copied are in fact, a very large bunch of small files (17,000). The server stays with 2 GB of RAM and four CPUs:

```
Concurrency Level:      5
Time taken for tests:   80.814 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Total transferred:      3725700 bytes
HTML transferred:       3677800 bytes
Requests per second:    1.63 [#/sec] (mean)
Time per request:       3490.707 [ms] (mean)
Time per request:       588.141 [ms] (mean, across all concurrent
requests)
Transfer rate:          70.22 [Kbytes/sec] received
```

As you can see, the disk's velocity has a crucial impact on the performance. Indeed, we lost more than 1 second, giving us 3.4 seconds per request, by asking a busy disk to serve us pages.

We make our hardware range from one CPU at 1.8 Ghz and 512 MB of RAM to four CPUs at 1.8 Ghz and 2 GB of RAM. All these hardware changes have led us to a minimum of 2281 ms average (four CPUs, 2 GB RAM, and no disk stress). We identified what the bottlenecks of an out-of-the box Magento site on a default LAMP stack are. First of all, 512 MB of RAM and a single core with a low frequency (under 2 GHz) are not enough to handle five concurrent clients.

In conclusion, if you intend to serve more than five customers at a time, you should consider a configuration ranging from a CPU with a high frequency or many smaller ones along with, at least, 1 GB of RAM. In a perfect scenario, you should also go for SSD disks, which are a big plus point as Magento makes an intensive use of them. Still, we are at light years of the test performance on Google, in this introductory chapter. As a reminder, the average time per request was 81.635 ms. We can nuance these results because we test the Google homepage, nevertheless, there is a long way to go.

If the hardware improves only that much our performance, the responses must be elsewhere. In the next chapter, we will see how to improve our performances by installing another webserver.

Selecting a trusted company

Your web host can become a friend you never hear from or your worst nightmare. Indeed, it will be a shame to see all our further attempts in optimizing our Magento site annihilated by a poor host.

Magento Inc. maintains a country-wise list of hosting companies that install Magento on your servers for free. This list is available at `http://www.magentocommerce.com/wiki/1_-_installation_and_configuration/magento-web-hosting`.

Beside this list, the following are some points that you should look at when picking your hosting company:

- Amount of web space and I/O performances
- Reliability and speed of access
- Data transfer (traffic/bandwidth)
- Root access FTP, PHP, Perl, SSI, .htaccess, SSH, MySQL, and crontabs
- SSL (secure server)
- E-mail, POP
- Control panel
- Multiple domain hosting and subdomains
- Web server and operating system
- Price
- Monthly/quarterly/annual payment plans
- Customer support 24 x 7 DDoS protection and virus scanner

Handling more than what you physically can

In addition to all the optimization you will find in this book, you will certainly need a little help externally. In this part of the chapter, we will learn how to distribute our requests for external services.

Content Delivery Network

Several hosting companies offer the possibility to replicate your files on many servers scattered across the globe. The advantage here is that, if you have a worldwide traction, your customers will get their files from the closest location instead of a unique and very distant server.

Of course, for this kind of service you need to shell out some extra cash monthly. However, if you do have customers scattered all over the planet that will be money well spent.

A **Content Delivery Network** (**CDN**) is a large distributed system of servers that is deployed globally; each of them owns a copy of a file such as pictures or scripts that your customers are downloading frequently. The goal is to serve files from the closest location to the customer. The difference here is that you will not own the CDN. Indeed, there are plenty of CDN operators who will offer different options and prices. Among the most known are Windows Azure and Amazon CloudFront.

Eventually, you could go with CloudFlare, which I use for all my clients. There is a free solution that provides a free CDN, a page cache mechanism, an anti-intrusion mechanism, and the possibility to supercharge your website with a ton of tiny yet useful services.

Summary

In this first chapter, we successively upgraded CPU, RAM, and disk rate in order to evaluate their impact on Magento performances. We found out that each of them will directly improve Magento performance. However, hardware upgrades come at high costs. In the next chapters, we will focus on the software side of server optimization.

2

Choosing the Best Web Server

In this chapter, we will tackle the critical choice of a web server for Magento. The web server is actually the best lever to improve your performance and please your customers. This chapter will lead you towards a better understanding of your needs and the major differences between web servers. In the second phase, we will learn how to go beyond simple installation and learn the following topics:

- Install, tweak, and benchmark Apache
- Install, tweak, and benchmark lighttpd
- Install, tweak, and benchmark Nginx

Evaluating your needs

In order to clearly evaluate your needs, you have to understand how the time taken to load a page impacts your customer's satisfaction. The basic categories of a page load have been (almost) the same for the past three decades. Here is an excerpt of Miller who started to study this in 1968:

- 0.1 second is the limit for the user to feel that the system is reacting instantaneously, that is, no special feedback is necessary except to display the result.
- 1.0 second is the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. Normally, no special feedback is necessary during delays of more than 0.1 but less than 1.0 second, but the user does lose the feeling of operating directly on the data.

- 10 seconds is the limit for keeping the user's attention focused on the dialogue. For longer delays, users will want to perform other tasks while waiting for the computer to finish, so they should be given feedback indicating when the computer expects to be done. Feedback during the delay is especially important if the response time is likely to be highly variable, since users will then not know what to expect.

Obviously, we want our web servers to serve pages as fast as possible; however, we have to keep the maintainability and the usability high too. Indeed, we can surely reach this kind of response time if we propose customers webpages without images to our or so. In addition, we definitely want our server to be understandable for us as backend users/developers.

In this chapter, we will try to get rid of the sacrosanct Apache HTTP web server, which is simply called Apache. This server is maintained by the Apache software company and its development started back in 1995. Nowadays, the Apache HTTP web server still holds 44.89 percent of the web server market. In other words, it is the web server of 344 million out of the 767 million websites in the world. Nevertheless, every month—according to `http://news.netcraft.com/`—it loses almost 5 percent of market shares. So, what are our options? What is the next web servers on line that can potentially beat the king? In this chapter, we will interest ourselves in **lighttpd**, which has a slogan *flight light*.

Our second candidate is **Nginx**, which, for now, holds 16.05 percent of the market shares. However, unlike Apache, its adoption is ascending. Indeed, Nginx won a percent a month. So, why those two in particular? Lighttpd is an open source web server released for the first time in March 2003 and was developed to obtain better performances on speed-critical environment. In spite of these promises, the adoption of lighttpd is limited, indeed its powers only 0.3 percent of the overall website on earth and it actually lost users (almost 0.5 percent in January 2013). Nginx was brought to the world with the same objectives as lighttpd—to handle more than 10,000 simultaneous connections—in speed-critical systems. In addition, Nginx aims to keep its memory footprint ranging from barely noticeable to low.

For our main problem, that is, optimizing Magento site performances, Apache and Nginx are supported by Magento Inc. Apache, is reputed to be very straightforward to set up and to get your Magento e-commerce up and running. On the other hand, Nginx is not mainstream concerning Magento hosting and it is reputed to be very hard to configure to get your Magento's website running. In the middle, lighttpd aims to provide better performances than Apache HTTP server without any complexity overhead.

Our main objective in this chapter is that we will endeavor to verify all these legends and, in particular, with Magento.

Choosing the best server for your e-commerce website

As in the previous chapter, we will use the Apache benchmark tool to benchmark the different web servers. Here is the command that sends 100 requests with a concurrency level of 5:

```
ab -n 100 -c 5 http://localhost/magento/index.php/furniture/living-room.html
```

Once again, we target the living-room page which displays the furniture/living-room category. This page stresses our web servers and database. We will test all the web servers with the exact same hardware as follows:

```
Ubuntu 12.04 64bits

2 Proc @ 1.80 GHz

2 GB Ram

10 Gig SSD disks
```

In the next sections, we will pass through the installation, evaluation, and optimization of the Apache HTTP server, lighttpd, and Nginx.

Installing the Apache HTTP server 2.2.22

This is the very first hands-on section of this book. Let's install the latest release of the Apache HTTP server, which is the 2.2.22. You can get it run by simply invoking the following command:

```
sudo apt-get install apache2
```

We will not dig deep into the details of the classical installation. This book tackles the optimization of such servers with the assumption that they are already running.

> To execute the command, you need to own root access to your server.

Before optimization

The following is the result of our test against an out-of-the-box Apache2 server:

```
Document Path:          /magento/index.php/furniture/living-room.html
Document Length:        36634 bytes
```

```
Concurrency Level:      5
Time taken for tests:   53.697 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Total transferred:      3711200 bytes
HTML transferred:       3663400 bytes
Requests per second:    1.86 [#/sec] (mean)
Time per request:       2684.860 [ms] (mean)
Time per request:       536.972 [ms] (mean, across all concurrent
requests)
Transfer rate:          67.49 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   0.2      1       1
Processing:  1546 2656 515.1   2669    4051
Waiting:     1349 2446 485.9   2427    3756
Total:       1547 2657 515.1   2670    4052

Percentage of the requests served within a certain time (ms)
  50%   2670
  66%   2805
  75%   2942
  80%   3164
  90%   3450
  95%   3535
  98%   3737
  99%   4052
 100%   4052 (longest request)
```

As you can see, the results aren't good. It takes almost 54 seconds to complete 100 requests and handle about 1.86 requests per second. Useless to say, the results are terrible.

Let's see how we can improve Apache2's server performance in the next sections.

Configuration tweaks

In the following subsection, we will tweak the Apache configuration in order to improve its performance. We will see the `mod_deflate` and `keepAlive` modules and set the maximum number of processes that Apache can create.

Using mod_deflate

Apache2 can be configured via the `.htaccess` files. The `.htaccess` files are the files that can configure web servers at the directory level. As such, you can tweak your configuration depending on the web application you host in each directory. Magento does embed a `.htaccess` file that contains a commented optimization. The first thing we will do is to uncomment everything between `<IfModule mod_deflate.c>` and `</IfModule>`. The complete piece of code looks like the following:

```
<IfModule mod_deflate.c>
AddOutputFilterByType DEFLATE text/html text/plain      text/xml
text/css text/javascript
BrowserMatch ^Mozilla/4 gzip-only-text/html
BrowserMatch ^Mozilla/4.0[678] no-gzip
BrowserMatch bMSIE !no-gzip !gzip-only-text/html SetEnvIfNoCase
Request_URI .(?:gif|jpe?g|png)$ no-gzip dont-vary Header append
Vary User-Agent env=!dont-vary

</IfModule>
```

Then, you can activate the deflate mode by typing the following command:

```
sudo a2enmod deflate
```

Finally, you can restart Apache2 in order to load the new configuration:

```
sudo /etc/init.d/apache2 restart
```

Let's find out if this simple tweak improves our performances. The following is the result of the test after using `mod_deflate`:

```
Document Path:          /magento/index.php/furniture/living-room.html
Document Length:        36634 bytes

Concurrency Level:      5
Time taken for tests:   49.228 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
```

```
Total transferred:      3711200 bytes
HTML transferred:       3663400 bytes
Requests per second:    2.03 [#/sec] (mean)
Time per request:       2461.392 [ms] (mean)
Time per request:       492.278 [ms] (mean, across all concurrent
requests)
Transfer rate:          73.62 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   0.1      1       1
Processing:  1346 2441 408.0   2482    3225
Waiting:     1180 2256 386.8   2292    3122
Total:       1346 2442 408.0   2482    3226

Percentage of the requests served within a certain time (ms)
  50%   2482
  66%   2608
  75%   2742
  80%   2793
  90%   2908
  95%   3145
  98%   3213
  99%   3226
 100%   3226 (longest request)
```

We pass from 54 seconds (to complete 100 requests) to 49 seconds with a 2.03 requests per second rate. This is almost a 10 percent amelioration, which is definitely lovely. The `mod_deflate` module is an Apache module that allows the compression of materials about to be sent to customers. Doing this, you surely stress a little bit more on your processor because compression is very resource consuming. On the other hand, as you send less data the sending packet process is faster. Therefore, we won some time here.

The keepAlive and max processes

In this second round for optimizing Apache2, we will use the `keepAlive` module, which authorizes to keep a connection alive longer than usual. As such, our server will not deal with opening and closing the TCP connection as often as before. In order to do so, you have to again open the `.htaccess` file at the root directory of your Magento and add the following code:

```
KeepAlive On
MaxKeepAliveRequests 100
KeepAliveTimeout 15
```

Activate the corresponding module and restart the Apache service using the following commands:

```
sudo a2enmod header
```

```
sudo /etc/init.d/apache2 restart
```

We will continue to optimize our Apache2 server by reducing the number of processes it is allowed to create in our server. The first thing is to determine how much memory a process needs to run. In order to discover this value, you can simply run a test using Apache Benchmark and run the following command on your host server:

```
$ top
```

The preceding command will show you the running processes on your server and the memory that each process will occupy.

In our case, the Apache2 processes require 42 MB of memory while under charge and we have 2048 MB of memory available. As we have other services running on your server (MySQL and so on), we cannot dedicate all our memory to Apache2. We choose to dedicate 1 GB. So, we have divided 1048 MB of memory by 42 MB, which gives us almost 25. This means that Apache2 can create 25 processes, which each requiring 42 MB of RAM before jamming the other services on the server. To set the new value of the max, add the following to the `.htaccess` file:

```
<IfModule prefork.c>
StartServers       25
MinSpareServers    25
MaxSpareServers    35
ServerLimit       256
MaxClients        256
MaxRequestsPerChild  10000
</IfModule>
```

Activate the `prefork` module as follows:

```
Sudo a2enmod prefork
sudo /etc/init.d/apache2 restart
```

Theses optimizations will make you win a lot of additional time. In the next section, we will see how we can do even better by compiling Apache 2.2 ourselves.

Compiling Apache 2.2

The version of Apache 2.2 you get when you install with the packet manager is built to function with any hardware. In other words, this is not a performance-focused build. In this section, we will compile Apache 2.2 from its sources.

Here is a list of the tools required to build Apache2:

```
sudo apt-get install gcc make libpcre3 libpcre3-dev bison flex php5
php5-dev
```

Then, we have to download the so-called sources:

```
cd /opt/
$ sudo wget http://archive.apache.org/dist/httpd/httpd-2.4.7.tar.gz
http://archive.apache.org/dist/apr/apr-1.5.0.tar.gz  http://archive.
apache.org/dist/apr/apr-util-1.5.3.tar.gz
```

The preceding command will place you in the `/opt` directory of your server and download files at three different URL. These URLs are pointing towards, `httpd-2.4.7.tar.gz`, `apr-1.5.0.tar.gz`, and `apr-util-1.5.3.tar.gz` respectively. The first file is concretely the web server while the last two are the Apache runtime libraries required to run the httpd server.

In the following commands, we will untar all these files and place the Apache runtime libraries under the `srclib` directory of the httpd server sources:

```
$ sudo tar zxvf httpd-2.4.7.tar.gz
$ sudo ln -s httpd-2.4.7/ httpd
$ sudo cd httpd/srclib/
$ sudo tar zxvf ../../apr-1.5.0.tar.gz
$ sudo ln -s apr-1.5.0/ apr
$ sudo tar zxvf ../../apr-util-1.5.3.tar.gz
$ sudo ln -s apr-util-1.5.3/ apr-util
```

Note that we create aliases of all the directory in order to match the naming convention recommended by Apache. In the next commands, we configure and compile the `apr` libraries:

```
cd /opt/httpd/srclib/apr

sudo CFLAGS="-Os" ./configure

make

make install

cd ../apr-util

sudo CFLAGS="-Os" ./configure –with-apr="/opt/httpd/srclib/apr"

make

make install
```

For both configuration steps we use the argument `CFLAGS="-Os"`, which is an option telling our compiler that the source files must be compiled in such a way that the memory consumption will be reduced. The cost of doing this is a much longer compilation time. Finally, we can configure, compile, and install our web servers. As you have certainly noticed by now, Apache2 is based on modules that we can activate or deactivate at will. However, this flexibility costs us time because the module isn't really a part of the Apache2 server and they have to be loaded at runtime when we need it. Fortunately, we can configure Apache in such a way that a list of modules will be compiled and integrated into the core of Apache2 annihilating the loading time of these modules. Also, we will deactivate some modules that Magento doesn't need and that are known to be resource consuming:

```
sudo CFLAGS="-Os" ./configure \

    --prefix=/opt/httpd \

    --with-apr=/opt/httpd/srclib/apr \

    --with-apr-util=/opt/httpd/srclib/apr-util \

    --enable-ssl \

    --with-ssl=/opt/openssl-1.0.1e \

    --enable-ssl-staticlib-deps \

    --enable-mods-static='rewrite prefork deflate headers expires
php5 mime dir auth' \

       --disable-status \

       --disable-userdir \

       --disable-threads \

       --disable-ipv6


make

make install
```

If we start this server and stress it using `ab`, we can see that the needed memory per processes drop to 35 MB. Therefore, we won 5 MB per process and we can adjust the configuration values that we saw in the previous section accordingly.

Installing the lighttpd 1.4.28 web server

In this section, we will try to prove that the lighttpd slogan, *flight light*, means something and really offers better performances than Apache2.

In order to install lighttpd on a new server, we have to enter the following command:

```
$ sudo apt-get install php5-mcrypt php5-curl php5-gd lighttpd
```

Before optimization

As for Apache2, we have run a first test on our Magento powered by a lighttpd server out of the box and the results are slightly better than Apache2:

```
Document Path:          /magento/index.php/furniture/living-room.html
Document Length:        37459 bytes

Concurrency Level:      5
Time taken for tests:   49.391 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Total transferred:      3709025 bytes
HTML transferred:       3664225 bytes
Requests per second:    2.02 [#/sec] (mean)
Time per request:       2469.562 [ms] (mean)
Time per request:       493.912 [ms] (mean, across all concurrent
requests)
Transfer rate:          73.33 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    3  20.7      1     208
Processing:  1484 2437 434.1   2317    3507
```

```
Waiting:      1364 2282 431.9   2154    3333
Total:        1485 2439 435.2   2318    3509

Percentage of the requests served within a certain time (ms)
  50%   2318
  66%   2606
  75%   2712
  80%   2920
  90%   3120
  95%   3193
  98%   3425
  99%   3509
 100%   3509 (longest request)
```

We need a little less than 50 seconds to complete the 100 requests, and that gives us a rate of 2.02 requests per second. However, we do not fight as light as expected because the performances are equal to the basic optimizations of Apache2.

In the following section, we will try to play a little with the lighttpd configuration in order to obtain better performances.

Configuration tweaks

The default configuration for the server can be found in the `/etc/lighttpd/lighttpd.conf` directory, and it is as follows:

```
server.max-keep-alive-requests = 128
server.max-keep-alive-idle = 30
server.max-read-idle = 60
server.max-write-idle = 360
```

We will now modify them to the following:

```
server.max-keep-alive-requests = 4
server.max-keep-alive-idle = 4
```

It may sound a little counterintuitive in comparison to the configuration of the Apache2 web server in which we actually raise these values. However, our tests prove that lighttpd performs better with this little tweak.

In the next tweak for lighttpd, we will see the event and network handler used to handle the requests of our users. Depending on the operating system you use to run your web server and power your Magento, you should use the following event handler:

```
all select select
Unix poll poll
Linux 2.4+ rt-signals linux-rtsig
Linux 2.6+ epoll linux-sysepoll
Solaris /dev/poll solaris-devpoll
FreeBSD, ... kqueue freebsd-kqueue
```

As we use a Linux 2.6+ we will set our event handler to `lynux-syspool`. In order to do so, we have to set this value in the configuration file of lighttpd in the same way as follows:

```
server.event-handler = "linux-sysepoll"
```

In the same way, here is the configuration for the network handlers:

```
all write
Unix writev
Linux 2.4+ sendfile
Linux 2.6+ sendfile64
Solaris sendfilev
FreeBSD sendfile
```

Accordingly, we adjust the value in our configuration to math our operating system as follows:

```
server.network-backend = "sendfile64"
```

Our next optimization is to increase the limit of file descriptors for lighttpd. Indeed, lighttpd is a single threaded server unlike Apache2. Therefore, the limit of file descriptor is a bottleneck. In order to raise this limit you have to run the lighttpd server as root and set the `max-fds` argument as follows:

```
server.max-fds = 2048
```

The default value for this parameter is 1024. It is now time to see if all these configuration tweaks allow a better performances. The performance result is as follows:

```
Document Path:          /magento/index.php/furniture/living-room.html
Document Length:        36634 bytes

Concurrency Level:      5
```

```
Time taken for tests:   48.420 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Total transferred:      3708200 bytes
HTML transferred:       3663400 bytes
Requests per second:    2.07 [#/sec] (mean)
Time per request:       2420.991 [ms] (mean)
Time per request:       484.198 [ms] (mean, across all concurrent 
requests)
Transfer rate:          74.79 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    3  20.4      1     205
Processing:  1302 2382 611.6   2126    4135
Waiting:     1211 2230 610.8   1971    3966
Total:       1302 2385 615.1   2127    4135

Percentage of the requests served within a certain time (ms)
  50%   2127
  66%   2357
  75%   2608
  80%   2986
  90%   3544
  95%   3657
  98%   3949
  99%   4135
 100%   4135 (longest request)
```

The improvement isn't much. Indeed, we improve our performances by only 2 percent. As such, we can conclude that lighttpd is, in fact, well configured as it is and tweaks will not give you a lot of additional performances.

Installing the Nginx 1.1.19 web server

In this section, we will see how much Nginx 1.1.19 can improve our performance in serving Magento. Unlike Apache2 and lighttpd, Nginx cannot serve Magento without a lot of configuration. Therefore, we will spend the first part of this section to install and configure Nginx in order to serve Magento and, in the second step, we will optimize Nginx.

Before optimization

The first step is to install all the required components using the following command:

```
$ sudo apt-get install nginx spawn-fcgi php5-cgi php5-cli php5-mcrypt
php5-gd php5-curl
```

Then, we need a script to force PHP to start in the `fastcgi` mode. This script has been built by `atoc.com` and can be downloaded at `http://mathieu-nayrolles.com/magento-optimization/chapter2/php5-fcgi`. This script is placed in `/etc/init.d/`:

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          php5-fcgi
# Required-Start:    $all
# Required-Stop:     $all
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: PHP FastCGI
# Description:       PHP FastCGI
### END INIT INFO

# Author: AITOC <www.aitoc.com>

PATH=/sbin:/usr/sbin:/bin:/usr/bin
DESC="PHP FastCGI"
NAME=php5-fcgi
DAEMON=/usr/bin/spawn-fcgi
FCGI_SOCKET=/var/run/php5-fcgi.sock
FCGI_USER=www-data
FCGI_MODE=0600
PIDFILE=/var/run/$NAME.pid
SCRIPTNAME=/etc/init.d/$NAME
DAEMON_ARGS="-s $FCGI_SOCKET -M $FCGI_MODE -u $FCGI_USER -U $FCGI_USER
-P $PIDFILE -- /usr/bin/php5-cgi"
```

```
[ -x "$DAEMON" ] || exit 0

[ -r /etc/default/$NAME ] && . /etc/default/$NAME

. /lib/init/vars.sh

. /lib/lsb/init-functions

do_start()
{
  start-stop-daemon --start --quiet --pidfile $PIDFILE --exec
/usr/bin/php5-cgi --test >/dev/null \
    || return 1
  start-stop-daemon --start --quiet --pidfile $PIDFILE --exec
$DAEMON -- \
    $DAEMON_ARGS >/dev/null \
    || return 2
}

do_stop()
{
  start-stop-daemon --stop --quiet --retry=TERM/30/KILL/5 --
pidfile $PIDFILE
  RETVAL="$?"
  rm -f $PIDFILE $FCGI_SOCKET
  return "$RETVAL"
}


case "$1" in
  start)
  [ "$VERBOSE" != no ] && log_daemon_msg "Starting $DESC" "$NAME"
  do_start
  case "$?" in
    0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
    2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
  esac
  ;;
  stop)
  [ "$VERBOSE" != no ] && log_daemon_msg "Stopping $DESC" "$NAME"
  do_stop
  case "$?" in
    0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
    2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
```

```
      esac
      ;;
      status)
           status_of_proc "$DAEMON" "$NAME" && exit 0 || exit $?
           ;;
      restart|force-reload)
      log_daemon_msg "Restarting $DESC" "$NAME"
      do_stop
      case "$?" in
       0|1)
        do_start
        case "$?" in
          0) log_end_msg 0 ;;
          1) log_end_msg 1 ;; # Old process is still running
          *) log_end_msg 1 ;; # Failed to start
        esac
        ;;
          *)
        # Failed to stop
        log_end_msg 1
        ;;
      esac
      ;;
      *)
      echo "Usage: $SCRIPTNAME {start|stop|status|restart|force-
    reload}" >&2
      exit 3
      ;;
    esac
```

Then, we can add the default `php5-fcgi` configuration file under `/etc/default/`. In a file named `php5-fcgi`, pass the following values:

```
    export PHP_FCGI_CHILDREN=10
    export PHP_FCGI_MAX_REQUESTS=1000
```

Finally, we can add `php5-fcgi` to the list of automatically started services with the following command:

```
$ sudo chmod +x /etc/init.d/php5-fcgi
$ sudo update-rc.d php5-fcgi defaults
$ sudo invoke-rc.d php5-fcgi start
```

We have set all the surrounding technologies; we can now handle the web server configuration. Here is the configuration proposed by Nginx developers for Magento 1.7+, which has been modified to run `fastcgi`:

```
server {
  root     /home/magento/web/;
  index     index.php;
  server_name  magento.example.com;
  location / {
    index index.html index.php;
    try_files $uri $uri/ @handler;
    expires 30d;
  }
  location ~ ^/(app|includes|lib|media/downloadable|pkginfo|report/
config.xml|var)/ { internal; }
  location /var/export/ { internal; }
  location /. { return 404; }
  location @handler { rewrite / /index.php; }
  location ~* .php/ { rewrite ^(.*.php)/ $1 last; }
  location ~* .php$ {
    if (!-e $request_filename) { rewrite / /index.php last; }
    expires off;
    fastcgi_pass unix:/var/run/php5-fcgi.sock;
     fastcgi_param SCRIPT_FILENAME
/var/www/magento-site-name$fastcgi_script_name;
    fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
    fastcgi_param MAGE_RUN_CODE default;
    fastcgi_param MAGE_RUN_TYPE store;
    include /etc/nginx/fastcgi_params;
  }
}
```

This file must be created under `/etc/nginx/sites-available/` and all the values in bold must be adapted to your configuration.

After all this, we can restart Nginx and finally get the default performances on this web server:

```
sudo /etc/init.d/nginx restart
```

Here is the default performance of Nginx:

```
Document Path:          /magento/index.php/furniture/living-room.html
Document Length:        16538 bytes
```

```
Concurrency Level:      5
Time taken for tests:   34.846 seconds
Write errors:           0
Total transferred:      1618209 bytes
HTML transferred:       1573709 bytes
Requests per second:    2.87 [#/sec] (mean)
Time per request:       1742.280 [ms] (mean)
Time per request:       348.456 [ms] (mean, across all concurrent
requests)
Transfer rate:          45.35 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       1
Processing:   745 1723 382.8   1676    2928
Waiting:      744 1722 382.9   1675    2927
Total:        746 1724 382.8   1677    2928

Percentage of the requests served within a certain time (ms)
  50%   1677
  66%   1818
  75%   1872
  80%   1971
  90%   2279
  95%   2552
  98%   2788
  99%   2928
 100%   2928 (longest request)
```

As you can see, the differences are amazing from web servers to web servers. Indeed, by default, Nginx is 35 percent faster than Apache2. Moreover, Nginx can handle 100 requests in no more than 35 seconds and have a score of 2.87 requests per seconds.

In the next section, we will try, as usual, to get even better performances out of Nginx.

Configuration tweaks

In this subsection, we will only modify the `/etc/nginx/nginx.conf` file. This file is the main configuration file of the Nginx web server. The first tweak in this configuration file will be to get the event section to look similar to the following code:

```
events {
    worker_connections  1024;
    multi_accept on;
    use epoll;
 }
Then, we have to uncomment the following parameters:
    server_tokens       off;
    sendfile            on;
    tcp_nopush          on;
    tcp_nodelay         on;
```

You can then set the `gzip` section to the following:

```
gzip                on;
    gzip_vary           on;
    gzip_proxied        any;
    gzip_types          text/css application/x-javascript;
    gzip_buffers        16 8k;
    gzip_comp_level     6;
    gzip_min_length     1024;
```

It may seem little, but Nginx is already the most powerful server. Let's discover how the performances have been increased by these modifications. The Nginx performance after the modifications is as follows:

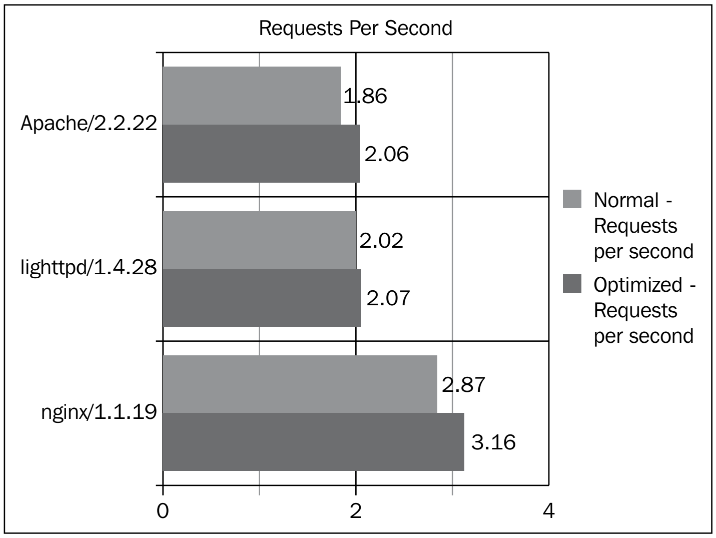
```
Document Path:          /magento/index.php/furniture/living-room.html
Document Length:        15729 bytes

Concurrency Level:      5
Time taken for tests:   31.614 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Non-2xx responses:      100
Total transferred:      1619000 bytes
HTML transferred:       1572900 bytes
Requests per second:    3.16 [#/sec] (mean)
```

```
Time per request:       1580.699 [ms] (mean)
Time per request:       316.140 [ms] (mean, across all concurrent
requests)
Transfer rate:          50.01 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    8  44.5      0     322
Processing:   978 1561 189.9   1590    1987
Waiting:      977 1561 190.0   1589    1986
Total:        978 1569 194.9   1591    2022

Percentage of the requests served within a certain time (ms)
  50%   1591
  66%   1668
  75%   1709
  80%   1722
  90%   1777
  95%   1933
  98%   1987
  99%   2022
 100%   2022 (longest request)
```

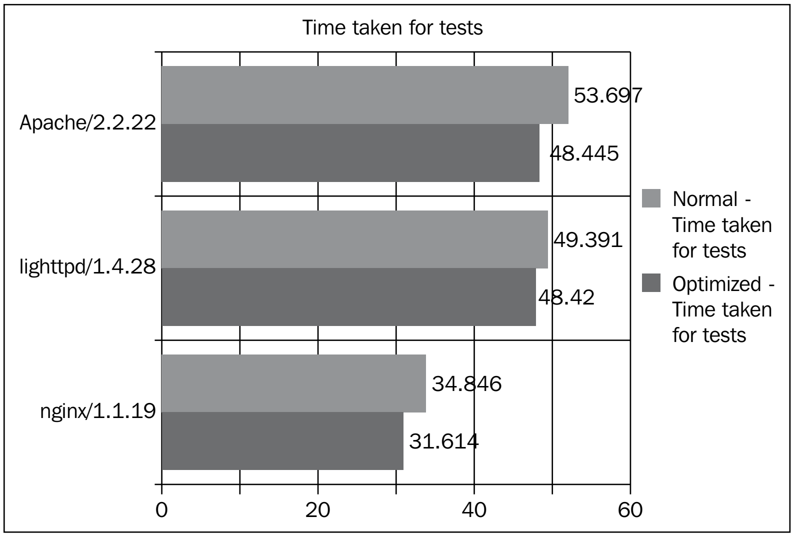
The time needed to complete our test is now 31 seconds, which is a solid 10 percent improvement. Also, our requests per second has been raised to 3.16.

Summary

In this very hands-on chapter, we learned how to install and configure three of the most popular web servers: the Apache HTTP server, lighttpd, and Nginx. For each of them, we first tried the default configuration with the **Apache Benchmark** (**ab**) stress tool and then optimized them in order to obtain better performances for Magento. As shown in the following graph, Nginx performs way better than Apache and lighttpd in terms of requests per second. Indeed, Nginx outperforms the two other competitors by a comfortable margin of 1.09 RPS:



As the requests per seconds could be hard to interpret for newcomers in the web servers optimization field, we also compute a new graph showing the total time needed to complete the test. Once again, Nginx is far ahead by requiring only 31.614 seconds while the two other one need more than 48.42 seconds:



We can conclude that Nginx wins by far this round and, in order to improve your Magento site performance, you should definitely consider switching to Nginx. In the next chapter, we will combine our newly acquired, optimized Nginx web server with an optimized database server.

In the next chapter, we will interest ourselves to the data engine which powers Magento: MySQL. We will tune, scale, and replicate MySQL in order to take advantage of our hardware and improve Magento's performance.

3

Tuning, Scaling, and Replicating MySQL

In this chapter, we will learn about an important and often neglected part of complex systems—the data and the engine that serve them. Magento only works with MySQL as database engine. Although MySQL obtains its well-founded reputation mainly because it's free and easy to use, it is becoming a solid choice for web applications in terms of reliability and ease of installation, configuration, and replication. However, these advantages have a principal drawback—MySQL is shipped in such a way that it can run in all hardware, even the most modest one. Therefore, we have to tune the configuration of MySQL in order to take advantage of our hardware. In this chapter, we will cover the following points:

- Finding the bottleneck in MySQL performance
- Tweaking the MySQL configuration
- Optimizing Magento database tables
- MySQL database replication

Understanding why MySQL is too stressed

MySQL is our database engine that serves all the data ever needed by our customers, and it is needless to say that there is a lot of data to be served in Magento installation. Indeed, every single label on your website is stored in the database and if you wish to add an article to your shopping basket, the Magento core has to check whether the product is still available in stock, and so on.

We can easily bring proof of this stress imposed on MySQL by Magento to the forefront. First, we have to delete the cache file of Magento, using the following command, in order to be sure that all the data will be fetched from the database:

```
$ sudo rm -R /var/www/magento-optimization/var/cache/mage--*
```

Be sure to adapt the path to your installation. If the used path matches the different installation steps discussed in the previous chapter, then we can activate the logging mechanisms of MySQL. This mechanism is, by definition, performance killer as they will write into a file every single operation our database engine does. Therefore, be sure to deactivate it in a production environment. To do so, you have to edit the main configuration file of MySQL with the following command:

```
$ sudo nano /etc/mysql/my.cnf
```

Look for `#general_log_file` in line 72 and uncomment it (remove the "`#`"). You can now restart your MySQL server for the new configuration to be loaded by using the following command:

```
$ sudo /etc/init.d/mysql restart
```

To be sure that we have sufficient data in our logging file, we first request the home page of our Magento and then the usual furniture category. The generated logging file is 352 lines long and is characterized by the following values:

- 179 selects
- 2 updates (log visitor)
- 4 inserts
- 0 delete

Some of the requests are very simple and we can assume that they do not cost too much to be completed. However, some of them are composed of many jointure and imbricated select. Here is an example of a fairly simple query and a much more complex query:

```
select count(*) into @discard from 'information_schema'.'EVENTS';
```

In the preceding code, we have a simple count on the `Events` table. The following one, however, has five inner jointures and many count and order operations:

```
SELECT 'attr_table'.* FROM 'catalog_category_entity_varchar' AS
'attr_table'
 INNER JOIN 'eav_entity_attribute' AS 'set_table' ON
attr_table.attribute_id = set_table.attribute_id AND
```

```
set_table.attribute_set_id = '12' WHERE (attr_table.entity_id =
'22') AND (attr_table.store_id IN (0, 1)) UNION ALL SELECT
'attr_table'.* FROM 'catalog_category_entity_text' AS 'attr_table'
 INNER JOIN 'eav_entity_attribute' AS 'set_table' ON
attr_table.attribute_id = set_table.attribute_id AND
set_table.attribute_set_id = '12' WHERE (attr_table.entity_id =
'22') AND (attr_table.store_id IN (0, 1)) UNION ALL SELECT
'attr_table'.* FROM 'catalog_category_entity_int' AS 'attr_table'
 INNER JOIN 'eav_entity_attribute' AS 'set_table' ON
attr_table.attribute_id = set_table.attribute_id AND
set_table.attribute_set_id = '12' WHERE (attr_table.entity_id =
'22') AND (attr_table.store_id IN (0, 1)) UNION ALL SELECT
'attr_table'.* FROM 'catalog_category_entity_datetime' AS
'attr_table'
 INNER JOIN 'eav_entity_attribute' AS 'set_table' ON
attr_table.attribute_id = set_table.attribute_id AND
set_table.attribute_set_id = '12' WHERE (attr_table.entity_id =
'22') AND (attr_table.store_id IN (0, 1)) UNION ALL SELECT
'attr_table'.* FROM 'catalog_category_entity_decimal' AS
'attr_table'
 INNER JOIN 'eav_entity_attribute' AS 'set_table' ON
attr_table.attribute_id = set_table.attribute_id AND
set_table.attribute_set_id = '12' WHERE (attr_table.entity_id =
'22') AND (attr_table.store_id IN (0, 1)) ORDER BY 'store_id' ASC;
```

We will not dig into the details of why Magento requires so much information from the database to display the home page and then the category page. Needless to say, a good optimization of the MySQL server will, for sure, improve our performance.

Configuring MySQL for high performance

As said in the introduction, the MySQL server is shipped in such a way that it can run on every hardware, even the most modest ones. In this section, we will learn how to configure MySQL to take advantage of our hardware. Nonetheless, there is good news. This is a well-known fact, and skilled developers have produced scripts that allow the analysis of a running MySQL server and provide advice and customization. In the next subsection, we will learn how to use these scripts.

Tuning MySQL using the mysqltuner.pl script

MySQL tuner is a handy Perl script that will analyze your MySQL configuration and provide some advice on the configuration you should have considered for your installation and needs. Indeed, this script will also analyze how your MySQL server was used since the last time you shut it down. On a production server that is online 24 hours a day and seven days a week, it can provide very pertinent data. However, on our development servers that are restarted many times a day, the advice can be somewhat inaccurate.

Here are the commands to retrieve the script and run it:

```
$ wget http://mysqltuner.pl mysqltuner.pl
$ sudo chmod +x mysqltuner.pl
$ ./mysqltuner.pl
```

In the preceding command lines, the first line downloads the script, the second line allows it to be executed, and the third line executes it.

Here is an example of the recommendations part of this particular script on a voluntarily messy configuration:

```
-------- Recommendations --------------------------------------------
---------
General recommendations:
    Run OPTIMIZE TABLE to defragment tables for better performance
    MySQL started within last 24 hours - recommendations may be
inaccurate
    Reduce your overall MySQL memory footprint for system stability
    Enable the slow query log to troubleshoot bad queries
Variables to adjust:
  *** MySQL's maximum memory usage is dangerously high ***
  *** Add RAM before increasing MySQL buffer variables ***
    query_cache_limit (> 148M, or use smaller result sets)
```

As you can see, the script advises us to run an optimized table operation in order to defragment our tables. The script mainly tells us that the maximum authorized amount of RAM for MySQL is dangerously high. Once again, the longer your MySQL has been online and actually serving data for your Magento, the more accurate the recommendation will be.

Tuning MySQL using the tuning Primer

MySQL tuning Primer script is another script written to analyze and propose modifications in your MySQL configuration. Moreover, this script is able to create a configuration file, including your old settings and the new recommendations for you. This script is actually the one that inspires MySQL tuner. In order to use it, the following are the required commands:

```
$ wget https://launchpad.net/mysql-tuning-primer/trunk/
1.6-r1/+download/tuning-primer.sh
$ sudo chmod +x tuning-primer.sh
$ ./tuning-primer.sh
```

At runtime, the script will ask you some questions and the following are the standard responses:

```
Would you like to provide a different socket?: [y/N] N
Do you have your login handy ? [y/N] : y
User: root
Password: your-password
Would you like me to create a ~/.my.cnf file for you? [y/N] : N
```

In the output of this script, you will find many sections and most of them will contain either an OK flag, which means that your configuration is all right, or a recommendation to improve your configuration. The different sections in the output are: `SLOW QUERIES`, `BINARY UPDATE LOG`, `WORKER THREADS`, `MAX CONNECTIONS`, `INNODB STATUS`, `MEMORY USAGE`, `KEY BUFFER`, `QUERY CACHE`, `SORT OPERATIONS`, `JOINS`, `OPEN FILES LIMIT`, `TABLE CACHE`, `TEMP TABLES`, `TABLE SCANS`, and `TABLE LOCKING`. The following is an example of the recommendations that this script is able to do:

```
QUERY CACHE SECTION
Your query_cache_size seems to be too high.
Perhaps you can use these resources elsewhere
MySQL won't cache query results that are larger than
query_cache_limit in size


MEMORY USAGE
Max Memory Ever Allocated : 794 M
Configured Max Per-thread Buffers : 4.02 G
Configured Max Global Buffers : 784 M
Configured Max Memory Limit : 4.79 G
```

```
Physical Memory : 1.95 G

Max memory limit exceeds 90% of physical memory
```

As a reminder, the configuration on which I ran the script was not the default one but a misconfigured one in order to check the benefits of the recommendation system.

Tuning my.cnf for performance

As a base to work with, my advice will be to generate the configuration file by using the MySQL tuning Primer script by analyzing a real production server. Then, copy the generated files onto your development server in order to test out the recommendation of the script. After that, you can use the MySQL `tuner.pl` script and add the additional recommendation.

Follow the next steps to add the additional recommendation:

1. Run the Primer script on your production server.
2. Make sure it generates a `my.cnf` file.
3. Copy this file to your development environment.
4. Replace the `/etc/mysql/my.cnf` file of your environment server with the one generated by the `Run` Primer, and then play around with your Magento in order to generate new usage statistics of your database engine.
5. Run the `mysqltuner.pl` script on your production environment and add the new recommendation to your `my.cnf` file.

Then, follow the recommendations given here; they are targeting better performances for Magento:

- `innodb_buffer_pool_size`: This should be around 50 percent of the physical server RAM when the server hosts the web server and the database engine both. If MySQL has its own server, then the allocated RAM should be 80 percent of the total physical RAM.
- `innodb_thread_concurrency`: This must be two more than twice the number of your CPUs. This means that on a true dual core, the value must be *2 x 2 + 2 = 6*.
- `thread_concurrency`: This should be equal to three times the number of CPUs.
- `query_cache_size`: This should be 64 MB.
- `query_cache_limit`: This should be 2 MB.

- `table_cache`: This should be 500.
- `table_open_cache`: This should be 96.
- `join_buffer_size`: This should be 64 MB.
- `concurrent_insert`: This should be true.
- `sort_buffer_size`: This should be 124.
- `order_buffer_size`: This should be 8 MB.
- `group_buffer_size`: This should be 8 MB.

By order of appearance, here is what each option does. The `innodb_buffer_pool_size` parameter defines the amount of memory our database engine is allowed to use while the next two define how many threads MySQL can create. Then, we take advantage of the caching mechanisms of MySQL by setting an overall cache and the query cache. In the last part of the option, we define a buffer for special operations such as join, sort, order, and group.

Optimizing our table

One of the recommended ameliorations provided by both scripts used in the previous section was to optimize our table. Optimizing tables can be a misleading expression, especially for us because we definitely don't want to modify the database model of Magento. Here, the optimization refers to the potential physical fragmentation of data, which means the data composing a table can be scattered over many places on the hard drive. The optimize table option can improve the performances in a sense that we can economize I/O operation while retrieving data from our database.

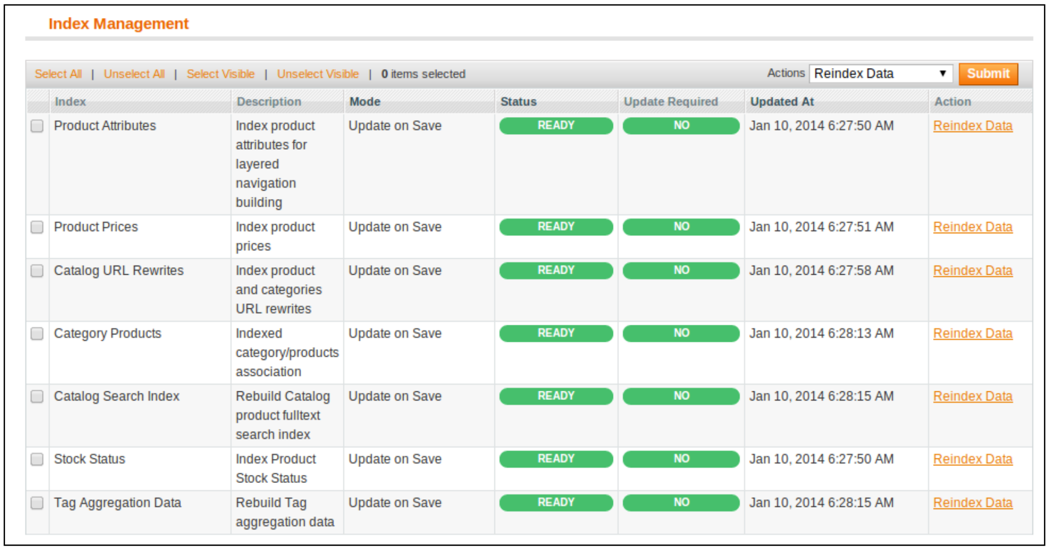
In order to optimize all the tables of your Magento database, we have to type the following command:

```
mysqlcheck -o magento_database_name
```

As you might expect, this command isn't related to Magento and is a more general tip for every system that uses Magento. While useful, this command should be used with parsimony and only when you don't have any stress on your servers. Indeed, it will require a huge amount of your disks rate to complete and, therefore, slow down everything else on your servers.

Another useful tip, this time related to Magento is the index management. As the date changes on your Magento, for example, you add new rules for prices discount and basket or client location they need to be reindexed. Indexes are special data structures that improve the speed of data retrieval.

Magento uses many indexes in order to access easily to, for example, rules for each product each time they are added to the basket of each customer. Indexes are very important in Magento and can be managed from the admin panel by navigating to **System | Index Management**. In this menu, each table that is in need of a reindexing will appear in red. You can reindex a table by selecting the table, selecting **Reindex** in the upper-right corner, and clicking on the **Submit** button. Here is a screenshot of the table after indexation:



Index Management

Select All | Unselect All | Select Visible | Unselect Visible | 0 items selected — Actions: Reindex Data ▾ Submit

	Index	Description	Mode	Status	Update Required	Updated At	Action
☐	Product Attributes	Index product attributes for layered navigation building	Update on Save	READY	NO	Jan 10, 2014 6:27:50 AM	Reindex Data
☐	Product Prices	Index product prices	Update on Save	READY	NO	Jan 10, 2014 6:27:51 AM	Reindex Data
☐	Catalog URL Rewrites	Index product and categories URL rewrites	Update on Save	READY	NO	Jan 10, 2014 6:27:58 AM	Reindex Data
☐	Category Products	Indexed category/products association	Update on Save	READY	NO	Jan 10, 2014 6:28:13 AM	Reindex Data
☐	Catalog Search Index	Rebuild Catalog product fulltext search index	Update on Save	READY	NO	Jan 10, 2014 6:28:15 AM	Reindex Data
☐	Stock Status	Index Product Stock Status	Update on Save	READY	NO	Jan 10, 2014 6:27:50 AM	Reindex Data
☐	Tag Aggregation Data	Rebuild Tag aggregation data	Update on Save	READY	NO	Jan 10, 2014 6:28:15 AM	Reindex Data

Truncating some tables for performance

The logging system for user interactions with your e-commerce website is exclusively based on MySQL. This means that whatever your users are doing on your website, their actions are stored in your MySQL database. On one hand, it's a very handy way of informing you what your user is looking for in your store and to adjust yourself to fit your customers' needs, but on the other hand, this information comes with an incredibly heavy weight. On average, these data weigh 1 MB per visitor per month. Knowing that, doing the math is fairly easy, 1,000 visitor/month weigh 1 GB of logs in your MySQL database. Needless to say, the heavier the database becomes, the slower your data will be served. Therefore, you should dump that log from time to time in order to learn about your customers from them, and also, delete them on a monthly basis.

The following are the commands to truncate a table:

```
$ mysql -user magento-user -password
```

```
Password: database-password
mysql > TRUNCATE dataflow_batch_export;
mysql > TRUNCATE dataflow_batch_import;
mysql > TRUNCATE log_customer;
mysql > TRUNCATE log_quote;
mysql > TRUNCATE log_summary;
mysql > TRUNCATE log_summary_type;
mysql > TRUNCATE log_url;
mysql > TRUNCATE log_url_info;
mysql > TRUNCATE log_visitor;
mysql > TRUNCATE log_visitor_info;
mysql > TRUNCATE log_visitor_online;
mysql > TRUNCATE report_viewed_product_index;
mysql > TRUNCATE report_compared_product_index;
mysql > TRUNCATE report_event;
mysql > TRUNCATE index_event;
mysql > TRUNCATE catalog_compare_item;
```

The `TRUNCATE` command will clean up every table on which it's invoked, free some space on your hard drive, and speed up your databases.

Moving MySQL to its own dedicated server

One of the easiest ways of optimizing Magento is to get a dedicated server for MySQL. As such, the hardware capacities will not be shared between the web server and the database engine. It is very easy to pass from a shared server from a dedicated server, and in this section, we will show how with a step-by-step tutorial. Indeed, MySQL comes with all the tools required to do this:

1. The very first thing to do is to dump our current database. This is done using the following command:

   ```
   $ mysqldump --user username --password your_magento_database >
   dump.sql
   ```

2. After typing your password, this command will dump all the content of `your_magento_database` to a file named `dump.sql`. Then, you can move this file to your new database server. If you need to install MySQL to your new server, use the following command:

   ```
   $ sudo apt-get install mysql-server mysql-client
   ```

3. After the installation, you have to create the Magento database using the following commands:

   ```
   mysql --user username -password
   create database magento;
   exit;
   ```

4. The previous command logs us in to MySQL and creates a new database named `magento`. We can finally import the data dumped from the original server:

   ```
   mysql -u username -p -h localhost magento < dump.sql
   ```

5. After the previous command, all the data of your Magento instance is now on the new server. However, we still have to configure Magento for it to use this new database server. First, we have to create a MySQL user that is allowed to connect to the database engine and interact with the data from the outside. Indeed, when MySQL and your applications are on the same server, the connection is granted de facto because it comes from localhost. In order to access data from another server, a special user is required. To create this user, log in to your MySQL engine with the following commands:

   ```
   mysql -u username -p
   ```

6. We are now logged in to MySQL and we can create the user:

   ```
   GRANT ALL PRIVILEGES
   ON magento
   TO 'magento'@'IP_ADDRESS'ADDRESS'
   IDENTIFIED BY 'your_password';
   ```

You have to replace the value in bold according to your system. The first instance of `magento` must be replaced by the name of your database and the second one refers to the name of the user you are about to create. The `IP_ADDRESS` value must be replaced by the IP of the server where your Magento application is.

You can find out what is your current public IP by using the terminal and the following command:

```
wget -qO- http://ipecho.net/plain ; echo
```

Finally, you have to replace the password with one of yours and adjust the Magento database user on your Magento configuration. You can do it in the `magento_home/app/etc/local.xml` file. Around line 48, the database configuration starts. You must adapt the value in bold to your environment:

```
<default_setup>
<connection>
<host>
<![CDATA[ IP_ADDRESS_OF_MYSQL ]]>
</host>
<username>
<![CDATA[ MYSQL_REMOTE_USER ]]>
</username>
<password>
<![CDATA[ MYSQL_REMOTE_USER_PASSWORD ]]>
</password>
<dbname>
<![CDATA[ MAGENTO_DATABSE_NAME ]]>
</dbname>
```

To be sure that the new configuration is loaded, you should delete the cache of your Magento installation as follows:

```
$ sudo rm -R /magento_home/var/cache/mage--*
```

Voila! Your Magento server now uses a remote MySQL server.

Replicating MySQL on a slave server

The last thing to attempt if you still feel that your MySQL engine is under too much pressure is to separate the insert query from the select query between the two databases. In other words, one database will be responsible for every read while a second one will handle insertion, updates, and deletion.

As we saw in the beginning of this chapter, the simple action of crawling a webpage triggers inserts. For example, if your visitor transforms into a buyer and wants to add an item to his or her basket, then a very huge amount of inserts and selects will be required. In order to discover what the ratio between inserts and selects is, we crawl the home page of our Magento server and then the category page. In this page, we choose an item and add it to the cart. Finally, we buy this item as a guest user. The MySQL log file is now 1820 lines long and contains 1121 queries that are organized as follows:

- 980 selects
- 68 inserts

- 70 updates
- 3 deletes

Therefore, having a database to handle the reads and another one to handle the writes will free the first database from more than 15 percent of its duty. However, we have to keep the two databases synchronized.

The following are the steps to replicate MySQL to a slave server:

1. The first step towards the replication is to create yet another up-to-date database server using the previous section *Moving MySQL to its own dedicated server*. Then, you have to choose which one will be the master and which one will be slave. If your two servers are identical, the choice doesn't matter. However, if they are not identical, you must choose as slave the one with the better hardware. Indeed, the slave will be used for the reads, while the master will handle the writes.
2. Then, we have to document which of the databases are powered by our master server database engine, and then activate logs on it. These logs will be used for the slave server to know what happens in the master server. Add the following lines to the `my.cnf` file of the master server:

   ```
   log-bin = /var/log/mysql/mysql-bin.log
   binlog-do-db=magento_db
   server-id=1
   ```

   Now, adjust the variable in the bold to your installation. Here, you must replace `magento_db` by the name of the database that powers your Magento.

3. The third step is the creation of a user with replication privileges on the master server. Log in to your MySQL engine as root, and then create the user by using the following commands:

   ```
   $ mysql -u root -p
   GRANT REPLICATION SLAVE ON *.* TO 'slave_user'@'%' IDENTIFIED
   BY '<some_password>';
   ```

4. Next, we will select the `magento` database and then release the tables that have a read lock:

   ```
   USE magento_db;
   FLUSH TABLES WITH READ LOCK;
   SHOW MASTER STATUS;
   ```

5. The previous command should output something similar to the following:

   ```
   +--------------+---------+-------------+
   | File         | Position | Binlog_do_db |
   +--------------+---------+-------------+
   | mysql-bin.007 | 297      | magento_db   |
   +--------------+---------+-------------+
   ```

We have now completed the configuration on the master server. You must save the data produced by the previous command as they will be needed in a few steps. If you followed all the previous steps, the Magento database should already be installed on your slave server. If it is not the case, refer to the previous section.

Open the `my.cnf` file on the slave server and add the following lines (by default, this file is under `/etc/mysql`):

```
server-id=2
master-host=MASTER_USER_IP
master-user=slave_user
master-password=slave_user_password
master-connect-retry=60
replicate-do-db=magento_db
```

The following is the next and final command to set up the MySQL slave server. Be sure to replace all the values in bold according to the output of the `SHOW MASTER STATUS` command on the user we created earlier in this chapter:

```
SLAVE STOP;
CHANGE MASTER TO MASTER_HOST=Master_IP, MASTER_USER='slave_user',
MASTER_PASSWORD='slave_password',
MASTER_LOG_FILE='mysql-bin.007',
MASTER_LOG_POS=297;
START SLAVE;
quit;
```

Finally, we must inform Magento that it has two databases to work with. In the `magento_home/app/etc/local.xml` file, locate the `<default_setup> </default_setup>` line and duplicate it. Then, modify the `<default_setup> </default_setup>` line into `<default_read> </default_read>` and adjust the values inside, as shown in the following code:

```
<default_read>
<connection>
<host>
<![CDATA[ IP_ADDRESS_OF_MYSQL_SLAVE ]]>
```

```
</host>
<username>
<![CDATA[ MYSQL_REMOTE_USER_ON_SLAVE ]]>
</username>
<password>
<![CDATA[ MYSQL_REMOTE_USER_PASSWORD_ON_SLAVE ]]>
</password>
<dbname>
<![CDATA[ MAGENTO_DATABASE_NAME ]]>
</dbname>
[. . . ]
</default_read>
```

After deleting the cache of Magento once again, your new configuration will be loaded and the reads will be handled by the slave server while the writes will be handled by the master server.

Summary

In this very hands-on chapter, we learned how to first optimize our database server configuration by using scripts that are capable of analyzing our configuration and proposing some amelioration. Then, we optimized our table and truncated the tables responsible for logging the visitor's move, in order to limit the growth of our database. Finally, we gave MySQL its own server and even learned how to replicate MySQL into another server in order to separate reads and writes. Unfortunately, the Magento core doesn't support more than two databases. The only way to achieve such clustering is to duplicate the whole software stack, including the web server. On each server, we have a web server connected to a database server for the reads, and all these read database servers are replicated from a unique write database server. However, the configuration of such an architecture is complex and doesn't fit in the scope of this book. You can find more about clustering the whole architecture in *Instant Magento Performance Optimization How-to*, *Packt Publishing*.

In the next chapter, we will learn how to install and configure the most effective caching mechanisms for Magento.

4
Caching Them All

This last chapter will introduce advanced caching mechanisms in order to serve a page more often than you actually generate it. That is, you can dramatically reduce the time needed to serve a page and give some rest to your CPUs and hard drive. As a reminder, the configuration of the server is the same as the last chapter. Magento is served by the Nginx web server and we have 2048 MB of RAM along with a dual core processor. In this final chapter, we learn how to install and configure the following:

- Varnish
- **Alternative PHP Cache** (**APC**)
- Memcached
- **Full page cache** (**FPC**)
- HipHop Virtual Machine from Facebook Inc.

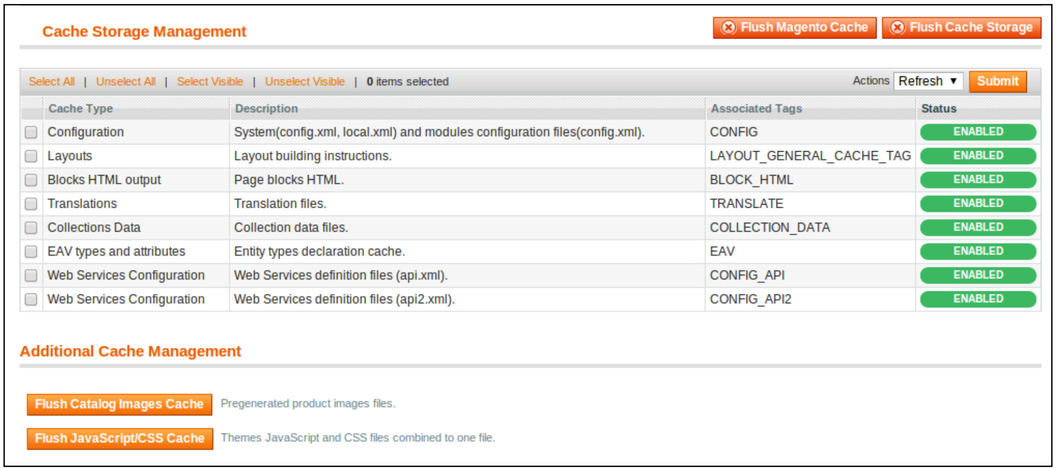
What is caching?

A cache is a system that stores generated or computed data so that future requests for that data can be served faster. There are several forms of cache mechanisms that allow the caching of different types of data to be cached. For example, web cache stores copies of documents passing through it, and subsequent requests may be satisfied from the cache if a set of conditions exists. Indeed, for each customer who asks for web pages from your server, the web server has to generate them. This generation or assemblage takes time; time that the web caching mechanisms will try to annihilate by storing the generated page and serve it again—without regenerating it—in case the same request comes again. In addition to web pages, we can cache a tremendous amount of things. Indeed, the PHP code composing Magento has to be transformed into bytecode in order to be executed by the processor of your server.

This also takes time and, as expected, we can cache the generated bytecode in order to save some resources. In one sentence, everything can be cached from the full web pages to random objects retrieved from libraries or databases. In this chapter, we will take advantage of the most efficient caching mechanisms to optimize Magento.

Exploring built-in Magento caching mechanisms

Before installing any third-party software to handle the caching in Magento, we can first see how the caching mechanisms embedded in Magento works. You have to go to **System** | **Cache Management** in order to manage them. The default settings look similar to what is shown in the following screenshot:



	Cache Type	Description	Associated Tags	Status
□	Configuration	System(config.xml, local.xml) and modules configuration files(config.xml).	CONFIG	ENABLED
□	Layouts	Layout building instructions.	LAYOUT_GENERAL_CACHE_TAG	ENABLED
□	Blocks HTML output	Page blocks HTML.	BLOCK_HTML	ENABLED
□	Translations	Translation files.	TRANSLATE	ENABLED
□	Collections Data	Collection data files.	COLLECTION_DATA	ENABLED
□	EAV types and attributes	Entity types declaration cache.	EAV	ENABLED
□	Web Services Configuration	Web Services definition files (api.xml).	CONFIG_API	ENABLED
□	Web Services Configuration	Web Services definition files (api2.xml).	CONFIG_API2	ENABLED

Additional Cache Management
Flush Catalog Images Cache
Flush JavaScript/CSS Cache

Pregenerated product images files.

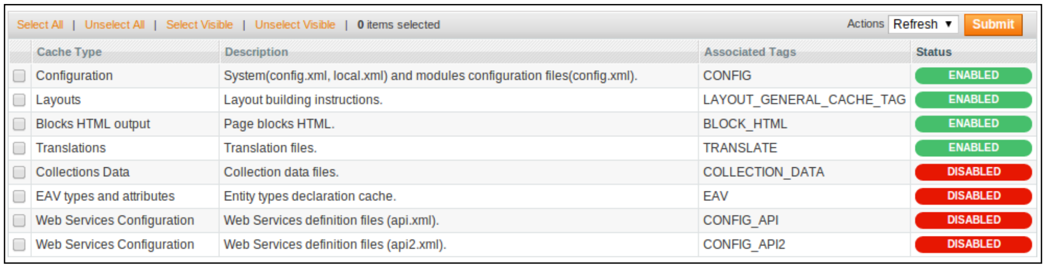
Themes JavaScript and CSS files combined to one file.

As you can see, all the caching categories are enabled and you have the possibility of refreshing them using the action list at the top-right corner or to flush them using the button at the bottom-left corner. The performances at this point are exactly the same as the previous chapter as we did not modify anything yet. However, we will go with a very counter intuitive optimization. Indeed, we will deactivate half of the caches. The following cache needs to be deactivated:

- **Collections Data**
- **EAV types and attributes**
- Both the **Web Services Configuration** options

In order to deactivate the cache, you have to select them using the checkboxes on the left side of the cache table and then select the deactivate action in the action list on the top-right corner.

The configuration is now as shown in the following screenshot:


Select All | Unselect All | Select Visible | Unselect Visible | 0 items selected
Actions
Refresh ▾
Submit

	Cache Type	Description	Associated Tags	Status
□	Configuration	System(config.xml, local.xml) and modules configuration files(config.xml).	CONFIG	ENABLED
□	Layouts	Layout building instructions.	LAYOUT_GENERAL_CACHE_TAG	ENABLED
□	Blocks HTML output	Page blocks HTML.	BLOCK_HTML	ENABLED
□	Translations	Translation files.	TRANSLATE	ENABLED
□	Collections Data	Collection data files.	COLLECTION_DATA	DISABLED
□	EAV types and attributes	Entity types declaration cache.	EAV	DISABLED
□	Web Services Configuration	Web Services definition files (api.xml).	CONFIG_API	DISABLED
□	Web Services Configuration	Web Services definition files (api2.xml).	CONFIG_API2	DISABLED

As counterintuitive as it may sound, the results are improved by this modification. Indeed, we can expect an improvement of 3 seconds on our classical test. As a reminder, following is the command to test the performance:

```
$ ab -n 100 -c 5 http://192.168.0.103/index.php/furniture/living-
room.html
```

There are many hypotheses out there to explain this weird optimization. The main one is that the Magento core has to parse the cache and check in MySQL to compare updated data, and this causes a huge delay. In fact, by allowing Magento to do these kind of operations, we don't use the full resources of our systems. As a conclusion, generating files could actually be faster than retrieving cached files. Once again, the *do not optimize until you need to* proverb is true. Indeed, these premature (and by default) optimizations hinder the performances of our Magento instance.

Using RAM to store cache files

A simple question about any computer is, "what is the slowest component?". The response is (almost) always the hard drive. Considering this well-known fact, why are we storing our cache files that are supposed to speed up our Magento on that component? Once again, the response is pretty simple: because it's reliable. Every created and updated file in the hard drive will handle a power shutdown, which isn't the case of the RAM.

However, what if we could put the entire Magento cache file in the RAM and synchronize them from time to time to the reliable hard drive? It will be a win-win configuration in which we have the access rate of the RAM, and we will suffer from a limited and controlled data loss in case of power shutdown or any major failure that requires the server to be stopped:

1. Create the in-RAM directory with the following command:

   ```
   $ sudo mount -t tmpfs -o size=256M,mode=0777 tmpfs
   /var/www/YOUR_DOMAIN.COM/var/cache/
   ```

   We use the `sudo` and the `mount` command to mount the `/var/www/YOUR_DOMAIN.COM/var/cache/` directory using the `tmpfs` filesystem. We also specify the maximum size of the directory using the `-o=256M` argument. Finally, we put the directory in `0777` mode. This mode will enable any process to write and read in or from that directory.

> **Tmpfs** stands for **temporary file storage** and is hosted directly into the RAM. Unfortunately, directories don't stay mounted in RAM between reboot.

2. In order to automate the creation of this directory in the RAM, you will have to add the following entry in the `/etc/fstab` file:

   ```
   $ tmpfs /var/www/YOUR_DOMAIN.COM/var/cache/ tmpfs
   size=256M,mode=0777 0 0
   ```

   The `/etc/fstab` file is the filesystem's table and can be found on almost every Linux-like system. This file is a configuration file that specifies the volume to be mounted at the start time. You should see your real disks on this table.

   The outcomes of this command will really depend on how slow your hard drive is on your production server. On our test server the optimization isn't much (around 10 percent) because we are using last generation SSD disks. If you are in the same case, the trouble may not worth it. However, if you are using good old 5400 RPM disks, the optimization obtained here will definitely put your Magento on steroids.

3. We now have to periodically synchronize the files inside the RAM to the disk in order to avoid major data losses. In order to do so, you can create a directory named `cache-backup` under `/var/www/YOUR_DOMAIN.COM/var/` and the following script under `/etc/init.d/` and name it `syncache.sh`:

```
#! /bin/sh
# /etc/init.d/syncache.sh
#

echo "Synching Magento Cache to  Hard drive"
echo [`date +"%Y-%m-%d %H:%M"`] Magento Cache Synched to Disk >> /
var/log/magento_ram_cache.log
    rsync -av --delete --recursive --force /var/www/YOUR_DOMAIN.
COM/var/cache/ /var/www/YOUR_DOMAIN.
COM/var/cache-backup/


exit 0
```

The previous script will, indeed, if launched, synchronize the file between the RAM mounted folder and the backup folder. It will echo `Synching Magento Cache to Hard drive` every time it's launched. Also, this script will write the following line in a file named `/var/log/magento_ram_cache.log`:

```
Year month day Hours minutes Magento Cache Synched to Disk
```

4. However, it's not yet automatized. In order to do it, you have to add an entry in the cron job table to our system. In Linux systems, the cron table is used to schedule commands that will be executed periodically and is used as follows:

   ```
   * * * * * user command to be executed
   ```

   The asterisks stand for minutes (0 to 59), hours (0 to 23), days of the month (1 to 31), days of the week (0 to 7, both 0 and 7 are for Sunday).

5. To edit the cron table, use the following command:

   ```
   $ crontab -e
   ```

   Add the following line to run the script every 10 minutes:

   ```
   * /10 * * * * root        /etc/init.d/syncache
   ```

From now on, the cache files belonging to the cache mechanisms built into Magento will be stored in an automatically mounted in-RAM directory. Moreover, the content of this directory will be synchronized with the hard drive every 10 minutes in order to avoid important data losses.

Installing other caching tools

In this section, we will learn how to install different third-party software in order to speed up your Magento installation. The third-party software that we are about to install is beyond the scope of pure Magento and could be suitable to improve the performances of any large-scale PHP application. However, we will see how to customize these tools specifically to improve Magento. Each one of the following tools is doing only one thing and doing it well. That's why they are tools on their own and not integrated with Magento.

Varnish Cache

Using the words of the Varnish's creators from `https://www.varnish-cache.org/`, here's what **Varnish** is:

> "*Varnish Cache is a web application accelerator also known as a caching HTTP reverse proxy. You install it in front of any server that speaks HTTP and configure it to cache the contents. Varnish Cache is really, really fast. It typically speeds up delivery with a factor of 300 - 1000x, depending on your architecture.*"

In other words, Varnish stores the response related to the received requests and if the same request comes again, then Varnish will send the stored response. Therefore, the generation of the response by the server is avoided. Of course, Varnish will check, from time to time, whether the response it has is still a valid one. The steps to install Varnish are as follows:

1. The first step towards the installation of Varnish is to update our package repository with the following commands:

   ```
   $curl http://repo.varnish-cache.org/debian/GPG-key.txt | sudo
   apt-key add –
   ```

   This line will fetch the GPG key from the Varnish website and add it to the apt keys. Then, we need to add the Varnish repository to our apt sources list by using the following command:

   ```
   $echo "deb http://repo.varnish-cache.org/ubuntu/ precise
   varnish-3.0" | sudo tee -a /etc/apt/sources.list
   ```

2. Finally, update our repository and install Varnish using the following commands:

   ```
   $sudo apt-get update
   $sudo apt-get install varnish
   ```

3. Varnish is now installed on our system, but it's still far away from helping us in our optimization quest. Indeed, we need to configure it to this purpose.
4. Open the `/etc/default/varnish` file using your favorite text editor and locate the `Alternative 2` block on line 40.
5. Modify the `-a` option so that it looks similar to the following:

   ```
   DAEMON_OPTS="-a :80 \
                -T localhost:6082 \
                -f /etc/varnish/default.vcl \
                -S /etc/varnish/secret \
                -s malloc,256m"
   ```

   The configuration exposed here will make Varnish listen on the port `80`, have its administration panel listening on the `6082` ports, and use the `256M` fixed size cache file. The forwarding address (that is, our Nginx server address) will be set in `/etc/varnish/default.vcl`. Therefore, we have to modify this file in order to complete the varnish configuration. In `/etc/varnish/default.vcl`, you will find the following block on the seventh line:

   ```
   backend default {
       .host = "127.0.0.1";
       .port = "8080";
   }
   ```

   This is where Varnish should look for the concrete web pages. As you can see, the default configuration may be adapted for us. Indeed, the host is the localhost (`127.0.0.1`) and the port is `8080`. However, our Nginx server does listen on port `80` and not on `8080`; therefore, we have to adapt its configuration.
6. Open the Nginx configuration file under `/etc/nginx/sites-enables/YOUR_MAGENTO_SITE_NAME` and edit the second line so it looks similar to the following:

   ```
   server{
     listen  listen  127.0.0.1:8080;
     server_name …
     […]
   }
   ```

7. We will also have to delete the default in the same folder and test whether the Nginx configuration is correct, by using the following commands:

   ```
   $ sudo rm /etc/nginx/sites-enables/default
   $ nginx -t
   ```

> You may wonder if it is safe to delete the default configuration of Nginx. The response is yes, because the default configuration was overridden by Magento's site configuration, which was also on port `80`. Now that the our custom configuration is on port `8080`, the default configuration will listen on port 80 and prevent Varnish to work correctly as port 80 will be occupied by Nginx.

8. We can finally restart `nginx` and `varnish` using the following commands:

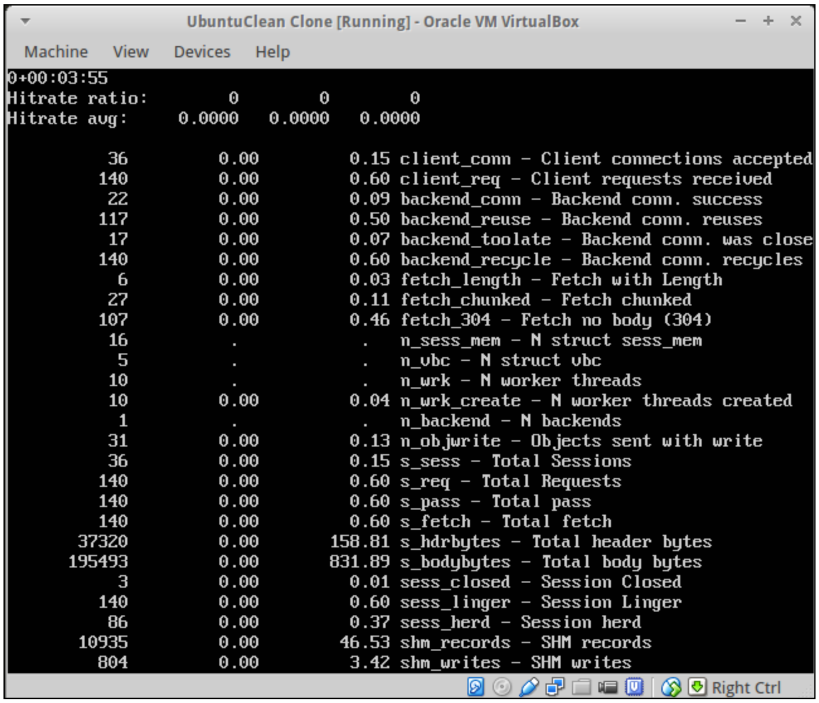
   ```
   $ sudo service nginx restart
   $ sudo service varnish restart
   ```

From now on, Varnish will handle the request coming from outside and periodically pull the page generated by Magento hosted in the Nginx server. The Nginx server listens for calls coming from localhost; therefore, from the outside, the 8080 port is still unreachable.

You can have a look at the Varnish stats by executing the following command:

```
$ varnishstat
```

Following is the output of the `varnishstat` command:



```
0+00:03:55
Hitrate ratio:        0        0        0
Hitrate avg:     0.0000   0.0000   0.0000

          36         0.00         0.15 client_conn - Client connections accepted
         140         0.00         0.60 client_req - Client requests received
          22         0.00         0.09 backend_conn - Backend conn. success
         117         0.00         0.50 backend_reuse - Backend conn. reuses
          17         0.00         0.07 backend_toolate - Backend conn. was close
         140         0.00         0.60 backend_recycle - Backend conn. recycles
           6         0.00         0.03 fetch_length - Fetch with Length
          27         0.00         0.11 fetch_chunked - Fetch chunked
         107         0.00         0.46 fetch_304 - Fetch no body (304)
          16          .            .   n_sess_mem - N struct sess_mem
           5          .            .   n_vbc - N struct vbc
          10          .            .   n_wrk - N worker threads
          10         0.00         0.04 n_wrk_create - N worker threads created
           1          .            .   n_backend - N backends
          31         0.00         0.13 n_objwrite - Objects sent with write
          36         0.00         0.15 s_sess - Total Sessions
         140         0.00         0.60 s_req - Total Requests
         140         0.00         0.60 s_pass - Total pass
         140         0.00         0.60 s_fetch - Total fetch
       37320         0.00       158.81 s_hdrbytes - Total header bytes
      195493         0.00       831.89 s_bodybytes - Total body bytes
           3         0.00         0.01 sess_closed - Session Closed
         140         0.00         0.60 sess_linger - Session Linger
          86         0.00         0.37 sess_herd - Session herd
       10935         0.00        46.53 shm_records - SHM records
         804         0.00         3.42 shm_writes - SHM writes
```

The previous screenshot depicts the Varnish performance and status.

If, despite the proven performances of Nginx, you cannot switch from Apache2 to Nginx, you will have to adapt the previous sections by following the next commands. Add the following code in the sudo nano /etc/apache2/ports.conf file:

```
NameVirtualHost 127.0.0.1:8080
Listen 127.0.0.1:8080
```

Change the settings of the virtual host to <VirtualHost 127.0.0.1:8080> in /etc/apache2/sites-available/default and restart Apache using the following command:

```
$ sudo service apache2 restart
```

Alternative PHP Cache

Alternative PHP Cache (**APC**) is a free open source framework that is aiming to accelerate PHP as a whole. How? As you may know, the PHP language is an interpreted language, but still has to be compiled into byte code to be executable by our CPUs. This process can be resource consuming for applications with a large source code base, as Magento owns more than 27000 files. So, the purpose of APC is to store these generated byte code into the shared memory in order to reuse them later instead of regenerating them every time.

> We advise the reader that this section will trigger the reinstallation of your Magento. You must do a backup before going further.

The following are the steps in order to get APC running:

1. The first step is to install the required libraries and APC:

   ```
   $ sudo apt-get install php-pear php5-dev make libpcre3-dev
   php-apc
   ```

2. Then, we have to modify the apc.ini file under /etc/php5/conf.d/ as follows:

   ```
   extension=apc.so
   apc.enabled = 1
   apc.cache_by_default = 1
   apc.shm_segments = 1
   apc.shm_size = 128
   apc.mmap_file_mask = /tmp/apc.XXXXXX
   ```

```
apc.stat = 0
apc.num_files_hint = 10000
apc.max_file_size = 5M
apc.ttl = 7200
apc.user_ttl = 7200
```

This file can be downloaded from `http://mathieu-nayrolles.com/magento-optimization/chapter4/apc.ini`.

3. The next, and final step for the installation of APC is to configure Magento to use APC. In the `Magento_root/app/etc/local.xml` file, in the line 57, you will find a tag named `<session>`. You have to add the following line just after the `<session></session>` tag line:

```
<cache>
       <backend>apc</backend>
       <prefix>mage_</prefix>
       <fast_backend>apc</fast_backend>
       <slow_backend>database</slow_backend>
       <slow_backend_store_data>0</slow_backend_store_data>
       <auto_refresh_fast_cache>0</auto_refresh_fast_cache>
</cache>
```

This piece of code specifies that the cache back-end to use is `apc` and the file should be prefixed by the string `mage_`. Indeed, you might want to use the very same APC to accelerate other PHP applications hosted on your server. Therefore, files must be prefixed by their application name in order to avoid discrepancies.

4. Clean the cache and session files using the following command:

```
$ sudo rm -R magento_root/var/cache
$ sudo rm -R magento_root/var/session
```

> As advised at the beginning of this section, the installation of Magento should have been triggered. The installation means that you will have to reset the time zone, currency, database address/user/passwords, credit card encryption key, and so on. However, your products and everything related to your store should not be impacted by the manipulation.

The memcached object caching system

The words of the creators of memcached cannot be clearer:

> "*Memcached is a free and open source, high-performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load. Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.*"

> Once again, the installation of memcached will trigger the reinstallation of Magento. Make sure that you have a copy in a handy.

We will use this mechanism to store the sessions as it is particularly well fitted and Magento has support for it by default. In order to install it, execute the following command:

```
$ sudo apt-get install memcached php5-memcache
```

Then, you will have to replace the `<sessions_save></session_save>` tag in the `magento_root/app/etc/local.xml` file by the following code:

```
<session_save><![CDATA[memcache]]></session_save>
<session_save_path><![CDATA[tcp://localhost:11211?persistent=1&wei
ght=2&timeout=10&retry_interval=10]]></session_save_path>
```

The previous lines of code make memcache our session manager and define how the session should be stored and retrieved with the `session_save_path` tag.

Lesti::Fpc – full page cache for Magento

We will now dig into full page cache. Full page cache mechanisms are used to cache the results of large dynamic blocks and therefore reduce the needed time to serve a page. If you look through the Magento Connect website with the FPC keyword, you will find dozens of free and paid extension designed to provide FPC for Magento. As we wish to rely only on free (and preferably open source) tools, we will only look at the free ones. Among them, there are two popular choices: **zoom full page cache** and **Lesti::Fpc**. In this book, we strongly advise you to choose Lesti::FPC because zoom full page cache is really painful to configure and any misconfiguration ends in a Magento White Screen of Death.

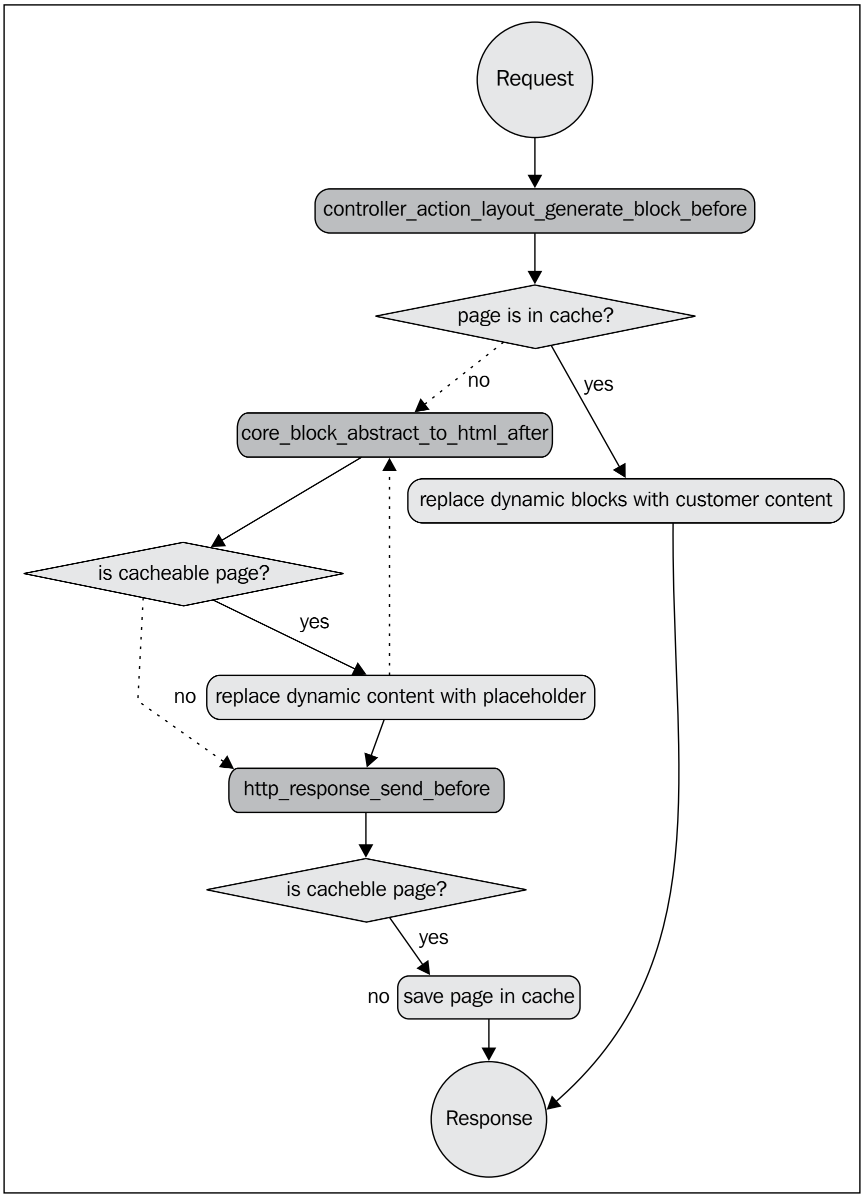
> **Magento White Screen of Death** (**MWoD**) is a reference to the **Windows Blue Screen of Death**. Indeed, in the case of a severe crash, the Magento core will not output anything and let the page desperately blank.

The Lesti::Fpc extension has been created by Gordon Lesti and you can get additional information on his website: `http://gordonlesti.com/`. As usual now, the words of the author describe this product:

> "*Lesti::Fpc is an internal full page cache for Magento. This Cache needs no varnish or any other external software and works with events. It is an internal cache and so it replaces dynamic blocks before sending a response to customer. Here is little post that explains the workflow of Lesti::Fpc.*"

The main difference between Lesti::Fpc and its competitor is that Lesti::Fpc doesn't rely on Ajax to refresh the dynamic content. Indeed, other FPC systems are said external as they save the outputs of requests into caches and serve these saved outputs for further requests. However, when the requests involve dynamic content, then, the FPC mechanisms have to fetch the dynamic content using Ajax. Ajax is a great technology, but can come with a lot of limitations and complex bugs. For example, if your customers use old or mobile browsers, they will experience difficulties. Gordon Lesti wanted an FPC mechanism that would be directly integrated into the Magento core and replace content by cached outputs directly during the process of generating pages.

The following figure can be found in the official Lesti:: Fpc website and exposes how the page generation is handled when it is installed:

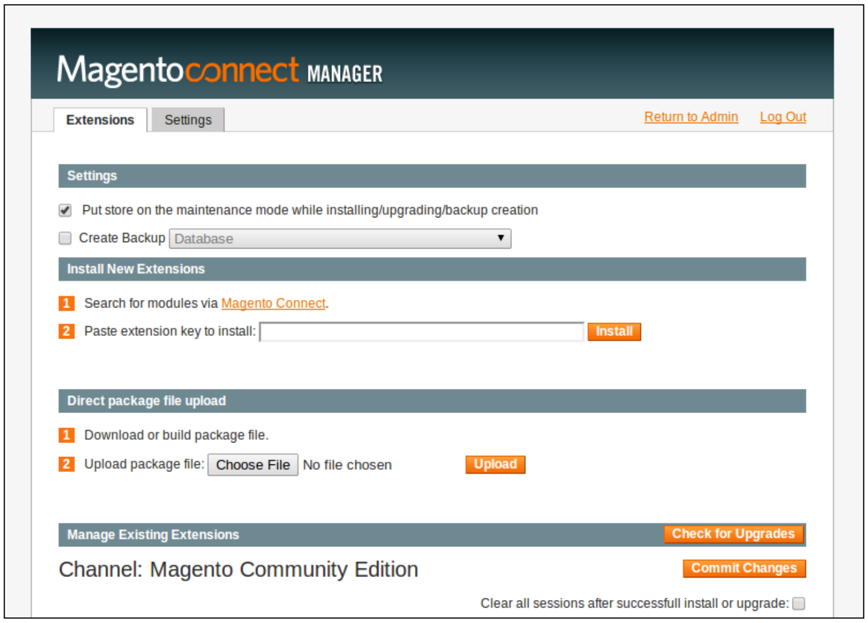

Request
controller_action_layout_generate_block_before
page is in cache?
no
yes
core_block_abstract_to_html_after
replace dynamic blocks with customer content
is cacheable page?
yes
no
replace dynamic content with placeholder
http_response_send_before
is cacheble page?
yes
no
save page in cache
Response

In the preceding figure, the white rectangles are Magento events while gray ones are Lesti::Fpc events and processes.

Installing Lesti::Fpc

The following steps show how to install the Lesti::Fpc:

1. Open the admin panel of your Magento and locate **Connect Manager** in the **Configuration** menu.



Settings

☑ Put store on the maintenance mode while installing/upgrading/backup creation

□ Create Backup Database ▾

Install New Extensions

1 Search for modules via Magento Connect.

2 Paste extension key to install: [Install]

Direct package file upload

1 Download or build package file.

2 Upload package file: [Choose File] No file chosen [Upload]

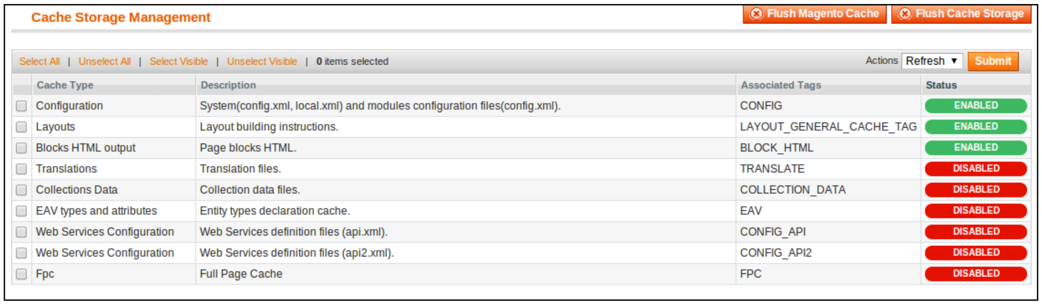
Manage Existing Extensions [Check for Upgrades]

Channel: Magento Community Edition [Commit Changes]

Clear all sessions after successfull install or upgrade: □

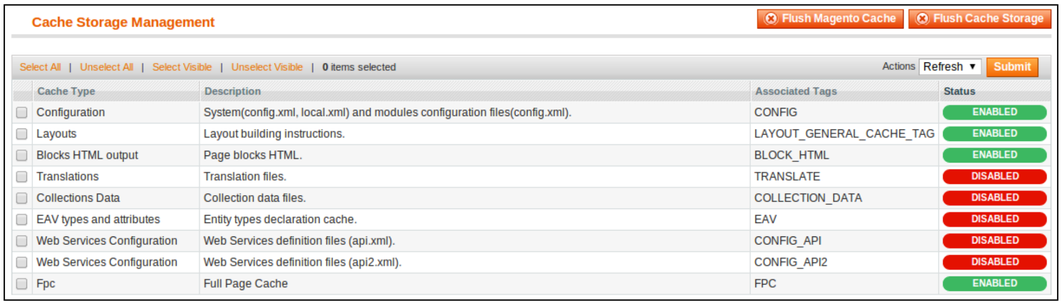
2. In the **Paste extension key to install** textbox, write the following: `http://connect20.magentocommerce.com/community/Lesti_Fpc`.

3. Click on **Install**. If everything went well, you can head for the **Cache Storage Management** menu under **System** and see **Fpc** in addition to the one already seen earlier in this chapter.



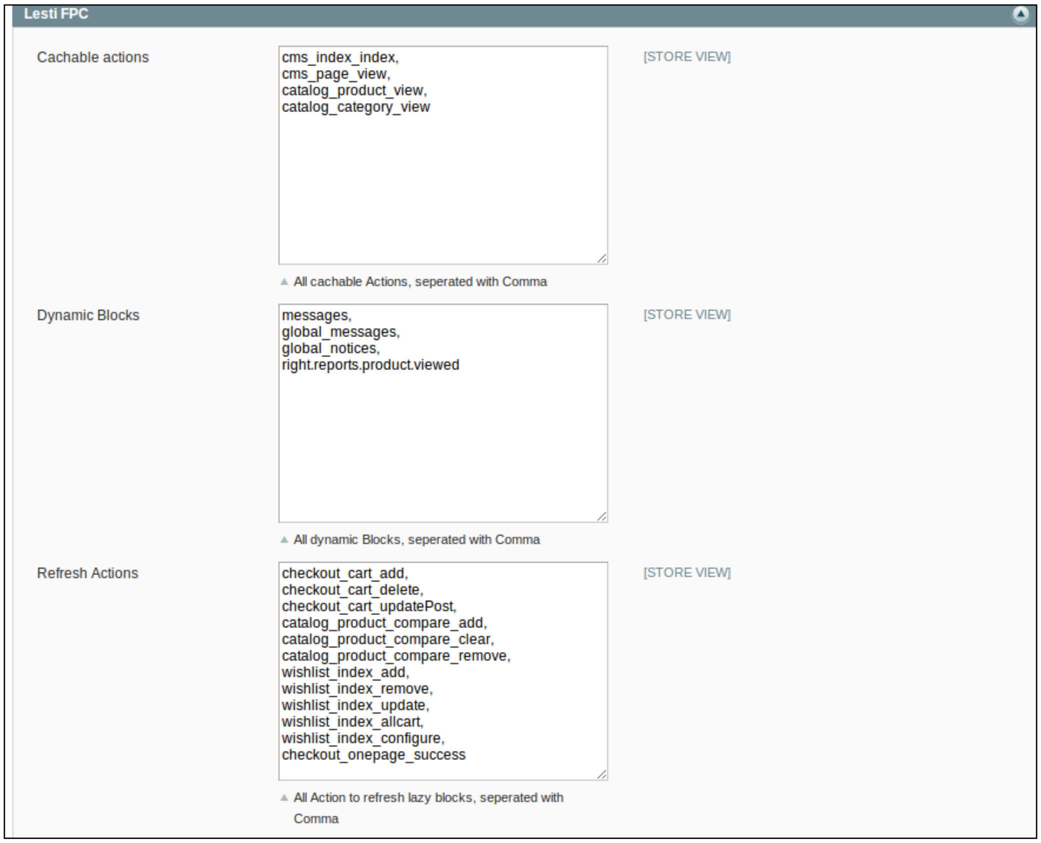
	Cache Type	Description	Associated Tags	Status
□	Configuration	System(config.xml, local.xml) and modules configuration files(config.xml).	CONFIG	ENABLED
□	Layouts	Layout building instructions.	LAYOUT_GENERAL_CACHE_TAG	ENABLED
□	Blocks HTML output	Page blocks HTML.	BLOCK_HTML	ENABLED
□	Translations	Translation files.	TRANSLATE	DISABLED
□	Collections Data	Collection data files.	COLLECTION_DATA	DISABLED
□	EAV types and attributes	Entity types declaration cache.	EAV	DISABLED
□	Web Services Configuration	Web Services definition files (api.xml).	CONFIG_API	DISABLED
□	Web Services Configuration	Web Services definition files (api2.xml).	CONFIG_API2	DISABLED
□	Fpc	Full Page Cache	FPC	DISABLED

4. Select **Fpc** and enable it using the action box, as shown in the following screenshot:



	Cache Type	Description	Associated Tags	Status
□	Configuration	System(config.xml, local.xml) and modules configuration files(config.xml).	CONFIG	ENABLED
□	Layouts	Layout building instructions.	LAYOUT_GENERAL_CACHE_TAG	ENABLED
□	Blocks HTML output	Page blocks HTML.	BLOCK_HTML	ENABLED
□	Translations	Translation files.	TRANSLATE	DISABLED
□	Collections Data	Collection data files.	COLLECTION_DATA	DISABLED
□	EAV types and attributes	Entity types declaration cache.	EAV	DISABLED
□	Web Services Configuration	Web Services definition files (api.xml).	CONFIG_API	DISABLED
□	Web Services Configuration	Web Services definition files (api2.xml).	CONFIG_API2	DISABLED
□	Fpc	Full Page Cache	FPC	ENABLED

5. The setting can also be found by navigating to **System** | **Configuration** | **Advanced** | **System** | **Lesti FPC**.



Cachable actions

```
cms_index_index,
cms_page_view,
catalog_product_view,
catalog_category_view
```

[STORE VIEW]

All cachable Actions, seperated with Comma

Dynamic Blocks

```
messages,
global_messages,
global_notices,
right.reports.product.viewed
```

[STORE VIEW]

All dynamic Blocks, seperated with Comma

Refresh Actions

```
checkout_cart_add,
checkout_cart_delete,
checkout_cart_updatePost,
catalog_product_compare_add,
catalog_product_compare_clear,
catalog_product_compare_remove,
wishlist_index_add,
wishlist_index_remove,
wishlist_index_update,
wishlist_index_allcart,
wishlist_index_configure,
checkout_onepage_success
```

[STORE VIEW]

All Action to refresh lazy blocks, seperated with Comma

6. At this point, Lesti::Fpc is fully functional but doesn't use APC or the memcached we settled earlier. In order to let Lesti:: Fpc use them, we have to rename the `app/etc/fpc. xml. sample` to `app/etc/fpc.xml` under `Magento_Home`.
7. Locate the section related to APC and memcached in `app/etc/fpc.xml` and modify these sections so that they look like the one we added in `app/etc/local.xml` in the previous two sections.

Let's have a look at the performances now that we have done so much in our Magento server:

```
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/

Licensed to The Apache Software Foundation, http://www.apache.org/
Benchmarking 192.168.0.103 (be patient).....done
Server Software:        nginx
Server Hostname:        192.168.0.103
Server Port:            80
Document Path:          /index.php/furniture/living-room.html
Document Length:        35611 bytes
Concurrency Level:      5
Time taken for tests:   29.797 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Total transferred:      3616000 bytes
HTML transferred:       3561100 bytes
Requests per second:    3.36 [#/sec] (mean)
Time per request:       1489.826 [ms] (mean)
Time per request:       297.965 [ms] (mean, across all concurrent
requests)
Transfer rate:          118.51 [Kbytes/sec] received
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    3  20.8      0     208
```

```
Processing:   781 1467 335.2   1473    2703
Waiting:      780 1465 335.3   1471    2701
Total:        782 1469 337.0   1474    2703

Percentage of the requests served within a certain time (ms)
  50%   1474
  66%   1593
  75%   1643
  80%   1670
  90%   1799
  95%   2143
  98%   2661
  99%   2703
 100%   2703 (longest request)
```

That's quite an achievement! We reached the impressive score of 3.36 requests per second for a total of less than 30 seconds. As a reminder, the default time obtained with Apache and no optimization at the beginning of the book was more than 110 seconds.

Understanding the limitations of cache mechanisms

Cache mechanisms, regardless what they are caching (PHP byte code, HTML responses, or MySQL requests), have one common limit: they are storing (computed) files in order to serve them later in case the same requests come several times. What does that imply? We are moving the load that was generated by the files every time for the hard drive from the CPU. Consequently, you have to monitor your resources in order to identify where the new bottlenecks are. Indeed, you could end up in a situation where the problem for your Magento is no longer the time required for the generation, but the time to retrieve the files from the disk. In such extreme scenarios, you might consider deactivating some of your caching mechanisms in order to bring back some load on your CPU.

HipHop Virtual Machine

HipHop Virtual Machine (HHVM) is the cornerstone of the PHP processing stack of Facebook and is currently able to increase the number of requests handle by a server by 9. It is open source and you can download it on GitHub. How does it work? As we can write a whole book (or two) about HHVM, we will just give the key ideas here. HHVM is a virtual machine that will translate any called PHP file into a HHVM byte code in the same spirit as the Java or .NET virtual machine. HHVM transforms your PHP code into a lower level language that is much faster to execute. Of course, the transformation time (compiling) does cost a lot of resources, therefore, HHVM is shipped with a cache mechanism similar to APC. This way, the compiled PHP files are stored and reused when the original file is requested. With HHVM, you keep the PHP flexibility and ease in writing, but you now have a performance like that of C++. Hear the words of the HHVM team at Facebook:

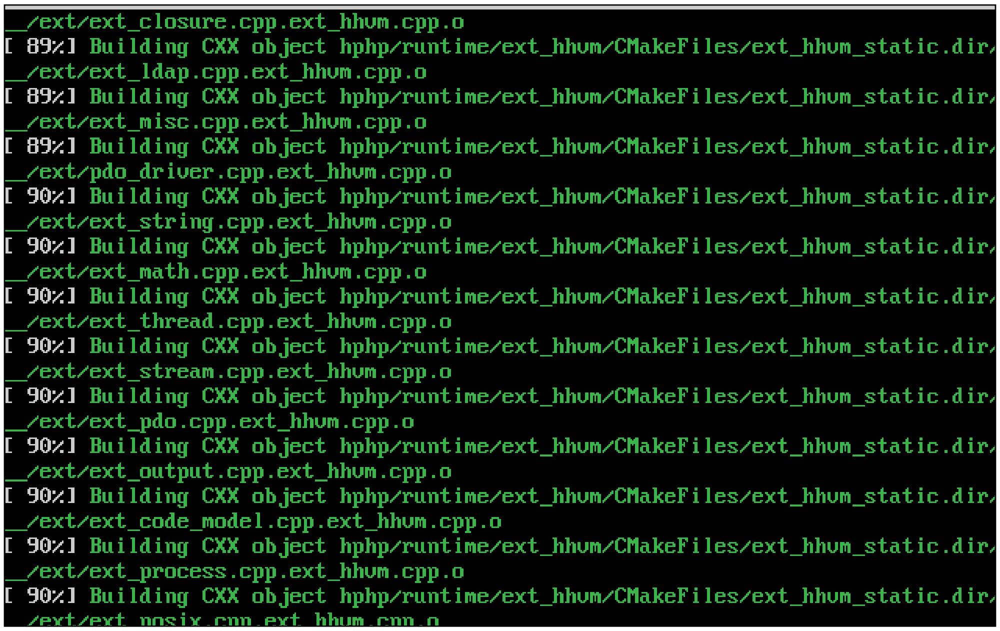
> *"HHVM (aka the HipHop Virtual Machine) is a new open-source virtual machine designed for executing programs written in PHP. HHVM uses a just-in-time compilation approach to achieve superior performance while maintaining the flexibility that PHP developers are accustomed to. To date, HHVM (and its predecessor HPHPc) has realized over a 9x increase in web request throughput and over a 5x reduction in memory consumption for Facebook compared with the Zend PHP 5.2 engine + APC.*
>
> *HHVM can be run as a standalone webserver (in other words, without the Apache webserver and the "modphp" extension). HHVM can also be used together with a FastCGI-based webserver, and work is in progress to make HHVM work smoothly with Apache."*

If you think this is too good to be true, you're right! Indeed, HHVM have a major drawback. HHVM was and still is focused on the needs of Facebook. Therefore, you might have a bad time trying to use your custom made PHP applications inside it. Nevertheless, this opportunity to speed up large PHP applications has been seen by talented developers who improve it, day after day, in order to support more and more framework. As our interest is in Magento, I will introduce you Daniel Sloof who is a developer from Netherland. More interestingly, Daniel has done (and still does) an amazing work at adapting HHVM for Magento.

Here are the commands to install Daniel Sloof's version of HHVM for Magento:

1. `$ sudo apt-get install git`
2. `$ git clone https://github.com/danslo/hhvm.git`
3. `$ sudo chmod +x configure_ubuntu_12.04.sh`
4. `$ sudo ./configure_ubuntu_12.04.sh`
5. `` $ sudo CMAKE_PREFIX_PATH=`pwd`/.. make ``

If you thought that the first step was long, you will be astonished by the time required to actually build HHVM. Nevertheless, the wait is definitely worth it. The following screenshot shows how your terminal will look for the next hour or so:



```
__/ext/ext_closure.cpp.ext_hhvm.cpp.o
[ 89%] Building CXX object hphp/runtime/ext_hhvm/CMakeFiles/ext_hhvm_static.dir/
__/ext/ext_ldap.cpp.ext_hhvm.cpp.o
[ 89%] Building CXX object hphp/runtime/ext_hhvm/CMakeFiles/ext_hhvm_static.dir/
__/ext/ext_misc.cpp.ext_hhvm.cpp.o
[ 89%] Building CXX object hphp/runtime/ext_hhvm/CMakeFiles/ext_hhvm_static.dir/
__/ext/pdo_driver.cpp.ext_hhvm.cpp.o
[ 90%] Building CXX object hphp/runtime/ext_hhvm/CMakeFiles/ext_hhvm_static.dir/
__/ext/ext_string.cpp.ext_hhvm.cpp.o
[ 90%] Building CXX object hphp/runtime/ext_hhvm/CMakeFiles/ext_hhvm_static.dir/
__/ext/ext_math.cpp.ext_hhvm.cpp.o
[ 90%] Building CXX object hphp/runtime/ext_hhvm/CMakeFiles/ext_hhvm_static.dir/
__/ext/ext_thread.cpp.ext_hhvm.cpp.o
[ 90%] Building CXX object hphp/runtime/ext_hhvm/CMakeFiles/ext_hhvm_static.dir/
__/ext/ext_stream.cpp.ext_hhvm.cpp.o
[ 90%] Building CXX object hphp/runtime/ext_hhvm/CMakeFiles/ext_hhvm_static.dir/
__/ext/ext_pdo.cpp.ext_hhvm.cpp.o
[ 90%] Building CXX object hphp/runtime/ext_hhvm/CMakeFiles/ext_hhvm_static.dir/
__/ext/ext_output.cpp.ext_hhvm.cpp.o
[ 90%] Building CXX object hphp/runtime/ext_hhvm/CMakeFiles/ext_hhvm_static.dir/
__/ext/ext_code_model.cpp.ext_hhvm.cpp.o
[ 90%] Building CXX object hphp/runtime/ext_hhvm/CMakeFiles/ext_hhvm_static.dir/
__/ext/ext_process.cpp.ext_hhvm.cpp.o
[ 90%] Building CXX object hphp/runtime/ext_hhvm/CMakeFiles/ext_hhvm_static.dir/
__/ext/ext_posix.cpp.ext_hhvm.cpp.o
```

Create a file named `hhvm.hdf` under `/etc/hhvm` and write the following code inside:

```
Server {
  Port = 80
  SourceRoot = /var/www/_MAGENTO_HOME_
}

Eval {
  Jit = true
}
Log {
  Level = Error
  UseLogFile = true
  File = /var/log/hhvm/error.log
  Access {
    * {
      File = /var/log/hhvm/access.log
      Format = %h %l %u %t \"%r\" %>s %b
    }
  }
}

VirtualHost {
  * {
    Pattern = .*
    RewriteRules {
      dirindex {
        pattern = ^/(.*)/$
        to = $1/index.php
        qsa = true
      }
    }
  }
}

StaticFile {
  FilesMatch {
    * {
      pattern = .*\.(dll|exe)
      headers {
        * = Content-Disposition: attachment
      }
    }
  }
  Extensions {
    css = text/css
    gif = image/gif
    html = text/html
    jpe = image/jpeg
    jpeg = image/jpeg
    jpg = image/jpeg
    png = image/png
    tif = image/tiff
    tiff = image/tiff
    txt = text/plain
  }
}
```

You can also download it from `http://mathieu-nayrolles.com/magento-optimization/chapter4/hhvm.hdf`.

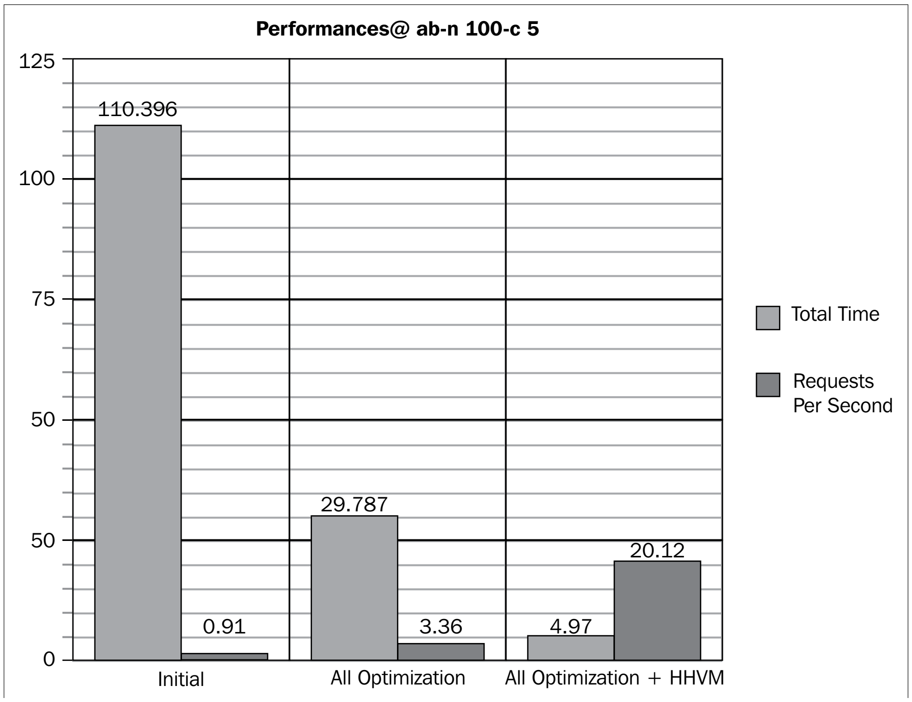
Now, run the following command:

```
$ sudo ./hhvm -mode daemon -config /etc/hhvm.hdf
```

The `hhvm` executable is under `hhvm/hphp/hhvm`. Is all of this worth it? Here's the response:

```
ab -n 100 -c 5 http://192.168.0.105/index.php/furniture/living-
room.html


Server Software:
Server Hostname:        192.168.0.105
Server Port:            80

Document Path:          /index.php/furniture/living-room.html
Document Length:        35552 bytes

Concurrency Level:      5
Time taken for tests:   4.970 seconds
Requests per second:    20.12 [#/sec] (mean)
Time per request:       248.498 [ms] (mean)
Time per request:       49.700 [ms] (mean, across all concurrent
requests)
Transfer rate:          707.26 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    2  12.1      0      89
Processing:   107  243  55.9    243     428
Waiting:      107  242  55.9    242     427
Total:        110  245  56.7    243     428
```

We literally reach a whole new world here. Indeed, our Magento instance is six times faster than after all our previous optimizations and about 20 times faster than the default Magento served by Apache exposed in the first chapter of this book. The following graph shows the performances:



Our Magento instance is now flying at lightening speed, but what are the drawbacks? Is it still as stable as before? All the optimization we did so far, are they still effective? Can we go even further? In what follows, we present a non-exhaustive list of answers:

- Fancy extensions and modules may (and will) trigger HHVM incompatibilities.
- Magento is a relatively old piece of software and combining it with a cutting edge technology such as HHVM can have some unpredictable (and undesirable) effects.
- HHVM is so complex that fixing a Magento-related bug requires a lot of skill and dedication.
- HHVM takes care of PHP, not of cache mechanisms or the accelerator we installed before. Therefore, APC, memcached, and Varnish are still running and helping to improve our performances.

- If you become addicted to performances, HHVM is now supporting Fast-CGI through Nginx and Apache. You can find out more about that at: `http://www.hhvm.com/blog/1817/fastercgi-with-hhvm`.

Summary

In this final chapter, we learned how to take advantage of different caching mechanisms in order to store—instead of recomputing—the requests of your customers, in terms of HTTP requests, PHP byte code, random objects, and dynamic blocks using Varnish, APC, memcached, and Lesti::Fpc, respectively. We also successfully used the HHVM from Facebook to serve Magento. This last improvement brings incredible performances.

Overall, this book allows Magento to be served in a fraction of the time required before. Indeed, the default time was over 110 seconds for our stress test, while the required time at the end of this chapter is less than 5 seconds. In conclusion, using all the technics seen in this book will result in Magento being sped up 20 times faster, if you are fearless enough to use HHVM, or 6 times otherwise.

Index

[PACKT] open source
PUBLISHING community experience distilled

Thank you for buying

Magento Site Performance Optimization

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

About Packt Open Source

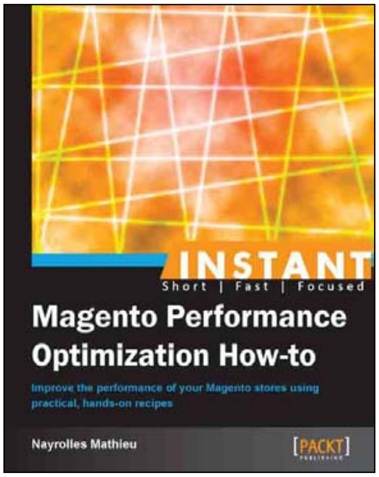
In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

[PACKT] open source
PUBLISHING community experience distilled


INSTANT
Short | Fast | Focused
Magento Performance Optimization How-to
Improve the performance of your Magento stores using practical, hands-on recipes
Nayrolles Mathieu
[PACKT]

Instant Magento Performance Optimization How-to

ISBN: 978-1-78216-542-2 Paperback: 56 pages

Improve the performance of your Magento stores using practical, hands-on recipes

1. Learn something new in an Instant! A short, fast, focused guide delivering immediate results.
2. Tune your Magento installation for optimal performance.
3. Identify misconfigurations that can cause slow down.
4. Prepare your installation for clustering.

Community Experience Distilled
Magento Responsive Theme Design
Leverage the power of Magento to successfully develop and deploy a responsive Magento theme
Richard Carter
[PACKT] open source

Magento Responsive Theme Design

ISBN: 978-1-78398-036-9 Paperback: 110 pages

Leverage the power of Magento to successfully develop and deploy a responsive Magento theme

1. Build a mobile-, tablet-, and desktop-friendly e-commerce site.
2. Refine your Magento store's product and category pages for mobile.
3. Easy-to-follow, step-by-step guide on how to get up and running with Magento.

Please check **www.PacktPub.com** for information on our titles

[PACKT] open source
PUBLISHING community experience distilled

INSTANT
Short | Fast | Focused
E-Commerce with Magento: Build a Shop
A fast-paced, practical guide to building your own shop with Magento
Branko Ajzele
[PACKT]

Instant E-Commerce with Magento: Build a Shop

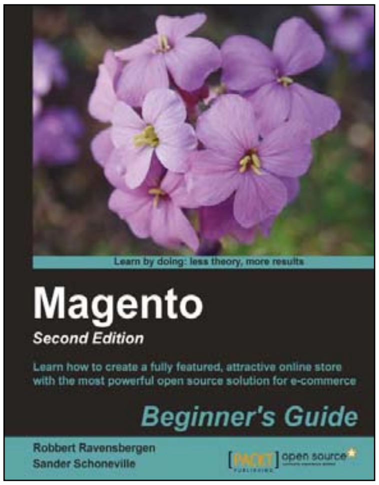
ISBN: 978-1-78216-486-9 Paperback: 52 pages

A fast-paced, practical guide to building your own shop with Magento

1. Learn something new in an Instant! A short, fast, focused guide delivering immediate results.

2. Learn how to install and configure an online shop with Magento.

3. Tackle difficult tasks like payment gateways, shipping options, and custom theming.


Learn by doing: less theory, more results
Magento
Second Edition
Learn how to create a fully featured, attractive online store with the most powerful open source solution for e-commerce
Beginner's Guide
Robbert Ravensbergen
Sander Schoneville
[PACKT] open source

Magento Beginner's Guide

Second Edition

ISBN: 978-1-78216-270-4 Paperback: 320 pages

Learn how to create a fully featured, attractive online store with the most powerful open source solution for e-commerce

1. Install, configure, and manage your own e-commerce store.

2. Extend and customize your store to reflect your brand and personality.

3. Handle tax, shipping, and custom orders.

Please check **www.PacktPub.com** for information on our titles