

Web Server Performance HOWTO

Patrice Neff

Web Server Performance HOWTO

by Patrice Neff

0.1 Edition

Copyright © 2005 Patrice Neff

This document explains some processes and hints for increasing the performance of a Web server.

Revision History

Revision 0.1 2005-03-29 Revised by: PN

Initial release

Table of Contents

Foreword.....	??
1. Feedback	??
2. Conventions used in this document.....	??
3. Copyright.....	??
4. Disclaimer	??
1. System Health.....	??
1.1. System Tools	??
1.1.1. uptime	??
1.1.2. vmstat	??
1.1.3. top.....	??
1.2. Monitoring	??
1.2.1. Before deciding	??
1.2.2. MRTG.....	??
1.3. Benchmarking	??
2. Operating system	??
3. Web server	??
4. PHP	??
5. Database.....	??
6. Application specific.....	??
7. Further Information.....	??
7.1. Literature.....	??
8. Concluding Remarks.....	??
9. Bits and Pieces.....	??
10. Examples.....	??
Bibliography	??

List of Tables

1. Typographic and usage conventions??

Foreword

Servers running Linux, Apache, MySQL and PHP - generally abbreviated as LAMP - are very popular nowadays. Such servers are increasingly easy to set up and keep running. Additionally the LAMP platform has proven to be a stable platform available at low costs.

Web servers running a LAMP package can perform very well out of the box. But there are many ways to improve performance both through server configuration and application changes. This document gives you a good starting point for where to start and what to look at. It will also equip you with tools and processes to monitor the server's performance.

This document is for you, if you are in charge of a web server running Linux and you want to improve the performance without just throwing more hardware at the problem.

1. Feedback

If you have any additions, corrections or any feedback at all, please write me:

<software@patrice.ch>

2. Conventions used in this document

The following typographic and usage conventions occur in this text:

Table 1. Typographic and usage conventions

Text type	Meaning
“Quoted text”	Quotes from people, quoted computer output.
<code>terminal view</code>	Literal computer input and output captured from the terminal, usually rendered with a light grey background.
command	Name of a command that can be entered on the command line.
VARIABLE	Name of a variable or pointer to content of a variable, as in \$VARIABLE.
option	Option to a command, as in “ the -a option to the ls command ”
<i>argument</i>	Argument to a command, as in “ read man ls ”.

Text type	Meaning
command options arguments	Command synopsis or general usage, on a separated line.
filename	Name of a file or directory, for example “ Change to the /usr/bin directory. ”
Key	Keys to hit on the keyboard, such as “ type Q to quit ”.
Button	Graphical button to click, like the OK button.
Menu → Choice	Choice to select from a graphical menu, for instance: “ Select Help → About Mozilla in your browser. ”
<i>Terminology</i>	Important term or concept: “ The Linux <i>kernel</i> is the heart of the system. ”
See Section 1.1	link to related subject within this guide.
The author (http://www.patrice.ch/)	Clickable link to an external web resource.

3. Copyright

Copyright © 2005 by Patrice Neff. This documentation is licensed under the Attribution-NoDerivs 2.0 Creative Commons license (<http://creativecommons.org/licenses/by-nd/2.0/>). Source codes are also licensed under the GNU General Public License 2.0

4. Disclaimer

Use the information in this document at your own risk. I disavow any potential liability for the contents of this document. Use of the concepts, examples, and/or other content of this document is entirely at your own risk.

Chapter 1. System Health

When running a server it's very important to be able to quickly check the current health and compare it with the past. This allows you to detect current anomalies and to see the system development over time. That part is covered in Section 1.1 and Section 1.2.

Benchmarking becomes important when tuning your system. You really want to be able to say if a change improves the system performance and by how much. So keep a box of benchmarking applications ready when applying changes intended to improve the system performance. This is covered in Section 1.3.

1.1. System Tools

There are a few general system tools which allow you to quickly find out the current health of the system. I will look into **uptime**, **vmstat**, **free** and **top** here.

1.1.1. uptime

The name of the tool **uptime** indicates that it tells you the current system uptime. And it does. But it does more than just that. A typical output of **uptime** looks like:

```
02:11:47 up 45 days, 16:55, 1 user, load average: 1.80, 1.37, 0.94
```

The first column is the current time. It is followed by how long the system has been running. Close to 46 days in this case. The next number tells you how many users are logged in. This is usually not of much use on a Web server, because Web site users are not counted here. The following three numbers are what we really want to know. It's the load average for the past one, five and fifteen minutes respectively. The load is defined as the number of processes currently waiting for execution.

A quick glance at this number will tell you the very basic health status of the system at the moment. What is tolerable can vary from system to system. Generally I have had the experience, that numbers below four are no problem at all. From there up to ten it might be good to look at it a bit closer and above a load of about ten the response time of the system may suffer considerably. But this numbers may vary highly on the individual server and even the cause of the current load.

If you see a high load using **uptime**, you may want to use **vmstat** to see a bit more detailed, where the system is slow or **top** to find out what processes are causing the high load.

1.1.2. vmstat

vmstat gives you a quick overview of some of the most important system statistics. The default output contains the following columns from left to right (straight from the man page):

- Procs: Process statistics
 - r: Number of processes waiting for run time.
 - b: Number of processes in uninterruptible sleep. (usually this processes are waiting for IO)
- Memory
 - spwd: Amount of virtual memory used.
 - free: Amount of idle memory.
 - buff: Amount of memory used as buffers.
 - cache: Amount of memory used as cache.
- Swap
 - si: Amount of memory swapped in from disk per second.
 - so: Amount of memory swapped out to disk (per second).
- IO: Input/Output subsystem
 - bi: Blocks received from a block device (blocks/s).
 - bo: Blocks sent to a block device (blocks/s).
- System
 - in: Number of interrupts per second.
 - cs: Number of context switches per second.
- CPU: this numbers are percentages of the total CPU time.
 - us: Time spent running non-kernel code. (user time)
 - sy: Time spent running kernel code. (system time)
 - id: Time spent idle.
 - wa: Time spent waiting for IO.

The first line of the output always is the average since system reboot. If **vmstat** is given a number as a command line parameter, it will show a new sampling every few seconds (e.g. every 10 seconds in **vmstat 10**). Every line after the first one then shows the values aggregated since the previous output.

Some common scenarios are:

- One or more CPU intensive processes running. This is indicated by a high user time and several processes waiting for run time. Use **top** to identify the CPU intensive process(es).

- Low on memory. If the currently free memory is low and the numbers for swapping in/out are high, your system is running low on RAM. Use **top** to identify the processes using much of the memory. As a short-term solution restarting the process may help, but look for long-term solutions such as configuration changes or memory extensions.
- Processes fighting. The context switches go especially high when processes are fighting for their CPU time. A common scenario is an application and database server on the same machine (Apache/PHP and MySQL for example). For every request the application may need to get a lot of information from the database server. In such cases it may actually be the best to get a dedicated database server or use a multi-processor machine.

1.1.3. top

top displays the current processes. By default it refreshes the display every three seconds and sorts the processes by current CPU usage. You can also change the sort order for example to show the processes with the highest memory utilization first. How this is done, may change from one version to the other. But pressing **h** will show you a detailed help screen.

When you encounter a high load it may be best to use **top** to get an idea what processes are causing the load. In times of unusually high load this might also be some runaway processes such as cronjobs that do not end in time.

1.2. Monitoring

Monitoring has different aspects. It can involve checking the system health at regular intervals and fixing things or notifying somebody when the system doesn't work as expected. That part is not covered in this HOWTO.

For our purposes it simply means that the system health is continually checked and recorded. I will show you how to use MRTG with an example setup. MRTG records the system status and show the development in a graphical way. There are other tools you may want to check out such as Cacti or cricket. Many of those tools are based on RRD Tool, a graphing tool written by the creator of MRTG. You'll find a large list of monitoring tools at <http://people.ee.ethz.ch/~oetiker/webtools/rrdtool-1.0.x/rrdworld/index.html>. So choose whatever suits your situation best.

1.2.1. Before deciding

You will have to decide what your environment is like and also what values you want to collect statistics on. Only if you have an idea what you're facing, you will be able to select the appropriate tool for the job.

Some important points to consider:

- Do you need to get values from multiple computers? If so it is quite useful to gather those statistics on one machine for easy access.
- How long do you want to store values? For example MRTG stores values for no more than about 1.5 years.
- How many values do you want to plot into one graph? MRTG can't do more than two values, while RRD Tool is able to do many values in one graph.
- Can you extend the solution easily to gather new values? This should be very easy to do.

1.2.2. MRTG

In this document I'll only look at MRTG. Many of the principles will probably be applicable to other monitoring solutions as well.

MRTG is an abbreviation for Multi Router Traffic Grapher. As this name indicates, MRTG was originally created to graph the traffic of routers over time using values gathered using SNMP. But through external scripts, MRTG is now very flexible and can graph every value imaginable. It's from the router background where MRTG's main limitation comes from: it can only handle two values in a graphic. This used to be simple In/Out traffic, but can now be used for whatever you wish.

The following picture shows a sample system with the "Daily" graph - which actually shows the last two days' worth of data.

1.2.2.1. Configuration

MRTG only requires one command line parameter. The config file. This is often stored in `/etc/mrtg.cfg` or similar. A minimal configuration file will look something like the following.

Example 1-1. MRTG example configuration

```
WorkDir: /var/www/kaywa_intranet_kaywa_biz/mrtg/calvin
XSize[^]: 600
YSize[^]: 100
Options[^]: growright

# CPU usage
Target[load]: `/usr/local/lib/mrtg/load.pl`
Options[load]: nopercents,gauge
```

```

Title[load]: cpu load
MaxBytes[load]: 100000
YLegend[load]: load * 100
ShortLegend[load]: /100]
LegendI[load]: 5 min average [
LegendO[load]: 15 min average [
Legend1[load]: cpu load 5 min average
Legend2[load]: cpu load 15 min average
PageTop[load]: <h1>cpu load</h1>

```

The only required value is `workDir`. It specifies where MRTG will put its output files. `xSize` and `ySize` specify the size of the generated graphics. The `Options` value `growright` causes the graph to be in left-to-right order. By default MRTG puts the current value to the left, which doesn't suit my way of reading at all.

The second part defines one target. Every target has a unique name which is specified inside the square brackets after the parameters. The target name used in the example is `load`. This will get the current system load average using the script `/usr/local/lib/mrtg/load.pl`.

1.2.2.2. Writing scripts

Writing scripts for MRTG is very simple. The script just has to output four lines.

1. First value for the target
2. Second value for the target
3. Uptime of the target in a human-readable format
4. Name of the target

The last two lines are not even necessary. So this makes the `load.pl` script is quite simple.

Example 1-2. `load.pl`

```

#!/usr/bin/perl
use strict;

my $data;
my $data1;
my $mrtg = 100;
my $uptime = `/usr/bin/uptime`;

$uptime =~ /average:.\+, ([\d.]+), ([\d.]+)/;
$data = $1;
$data1 = $2;
$data *= $mrtg;
$data1 *= $mrtg;

```

```
#answering :
print "$data\n";
print "$data1\n";
```

And please do really keep your MRTG scripts simple. You don't want those scripts to be the cause for a major performance problem.

1.2.2.3. Executing MRTG

While MRTG can run as a daemon, I have always executed it using cron. In my case it runs the following script every five minute.

```
# Gathers the mrtg statistics for the different servers

for servercfg in /etc/mrtg.d/*
do
    workdir=$(grep -i WorkDir $servercfg | sed 's/WorkDir: *//i')
    [ -d $workdir ] || mkdir -p $workdir

    mrtg $servercfg

    indexmaker --output $workdir/index.html --columns 1 $servercfg
done
```

This script allows me to put multiple configuration files into `/etc/mrtg.d`. I use that to have one configuration file and also a different working directory for every server I query. It also runs `indexmaker` to make sure the `index.html` file in the working directory reflects the current configuration.

1.2.2.4. Accessing other servers

I have a simple password-less SSH setup to get MRTG data from multiple servers. On every server I want to get data from I set up an account with very limited access. This I accomplish by giving that user an `rbash` shell and limiting the `PATH` to the directory of MRTG scripts. Additionally I allow logins to that account with a specify SSH private key.

The master server then accesses this clients using the SSH private key and can execute any MRTG commands in the `PATH`. Only the four lines of script output go over the wire, thereby this data aggregation setup is very efficient.

For detailed information on how to set up this SSH configuration, consult [Barrett2001], specifically chapter 11.1.

1.2.2.5. More information

Get more information about MRTG configuration in the man page or online at the MRTG Web site (<http://people.ee.ethz.ch/~oetiker/webtools/mrtg/>), especially the MRTG reference (<http://people.ee.ethz.ch/~oetiker/webtools/mrtg/mrtg-reference.html>).

1.3. Benchmarking

Chapter 2. Operating system

Linux configuration

- tempfs, noatime

Chapter 3. Web server

- Apache configuration
 - General
 - Caching on Apache level
 - mod_gzip
- Lighthttpd (and other similar servers)

Chapter 4. PHP

PHP configuration; Turck MMCache

Chapter 5. Database

- Optimize MySQL performance
- memcached
- SQL optimization

Chapter 6. Application specific

Cache on the application level

Chapter 7. Further Information

7.1. Literature

Chapter 8. Concluding Remarks

Chapter 9. Bits and Pieces

Chapter 10. Examples

Bibliography

Books

[Barrett2001] Daniel J. Barrett and Richard E. Silverman, 2001, Edited by Mike Loukides, 0-596-00306-4, O'Reilly & Associates, Inc., *SSH, the Secure Shell: The Definitive Guide*.

[Zawodny2004] Jeremy D. Zawodny and Derek J. Balling, 2004, Edited by Andy Oram, 0-596-00306-4, O'Reilly Media, Inc., *High Performance MySQL: Optimization, Backups, Replication & Load Balancing*.