Dynamic Webpage Compression with Mod_gzip and Apache

Teaching Penguins to Fly

HTML files compress quite well. Content encoding and compressed transfers allow vast improvements in the effective transfer rate of a web server. The Mod_gzip module for Apache provides Linux based web admins with jet propulsion for their favorite penguin. BY ULRICH KEIL

Despite a modern modem or ISDN connection you sometimes feel that a mere dribble of data is getting through to you from the Web. The reasons for slow page build up are many and varied

- inadequate server hardware resources,
- slow transfer of data across the Internet or
- a slow Internet connection on the part of the end user.

A sysadmin can normally update the server hardware with a minimal investment, and there is certainly very little one can do to remove the Internet bottleneck. Thus the last-line issue (the end user, or client, internet connection) provides the greatest potential for enhancing performance.

The majority of Internet users still do not have access to a broadband connection and are forced to resort to modems or ISDN. This means that any data will have to cross the telephone line bottleneck, no matter how quickly they are generated and served up. Figure 1 shows the download time for a 300 kbyte HTML file using a 28k modem.

Content Encoding

Content encoding was introduced in 1999 as part of the HTTP 1.1 standard and allows you to compress the content to be transferred using the GZIP/ZLIB compression algorithms. Common web browsers (MS Internet Explorer version 4.0 or later, Netscape Navigator version 4.0 or later, Opera, Lynx, W3m) transfer the "Accept-Encoding: gzip, …" string during the HTTP handshake to inform the server of the compression techniques they support, if any. The server will then compress any data it needs to transfer, and the client will in turn decompress the data and continue to process it.

These steps are done transparently, that is the user will not notice anything – apart from an obvious performance gain. If a client that does not support HTTP 1.1, such as a search engine, a proxy, or an old browser, sends a request, the data will simply be transferred as it is.

Since a compressed HTML file is only a fraction the size (20 to 30 percent) of the original, transfer rates can be improved considerably by compression techniques. Figure 2 shows the throughput of a 56 K modem, which can achieve values of more than 30 kbyte/s for compressed data.

Friends in need: Mod_gzip

Although it might seem obvious that compression can cause an additional load on the server, field tests have shown that the opposite is in fact true. Due to the fact that client requests do not take up so much of the Httpd child processes' time, system resources can be released more quickly. This in turn allows more requests to be processed. If the web

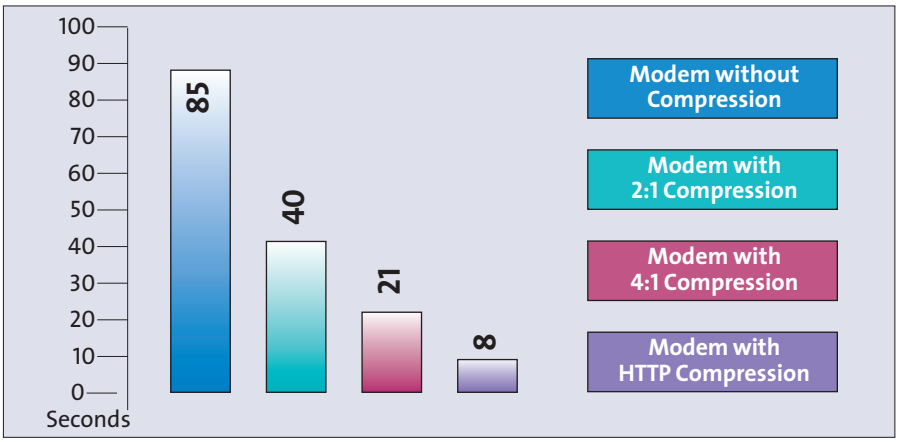

Figure 1: Download time for a 300 kbyte HTML file using a 28k modem

server is hosted by a provider that charges you for traffic volume, you can use content encoding to achieve considerable traffic reductions, and thus save a substantial amount of money.

The Mod_gzip module available under the Apache license provides a complete implementation of the content encoding standard (RFC 2613) for Apache 1.3. The module, which was developed by Hyperspace Communications, which is currently available as version 1.3.19.1a and is part of the commercial Hyperspace package.

The module is capable of compressing both static and dynamically generated content on the fly, and is used by freshmeat.net, slashdot.org and web-hostlist.de just to name a few. Although it is possible to link Mod_gzip into Apache, installing Mod_gzip as a DSO module is probably a better alternative since it saves you the time recompiling Apache, and the entire installation process can be completed in less than a quarter of an hour.

Installation: Plug & Play

After downloading the sources from [1], make sure you are superuser, root, and type the following:

```
apxs -i -a -c mod_gzip.c
```

to compile and install the module. The entries required to load Mod_gzip must be placed in "httpd.conf". But before you can finally use the module, you must insert the configuration directives from Listing 3 into "httpd.conf". All that remains now, is to restart your Apache by typing "apachectl restart". Mod_gzip is now ready and willing to deal with user requests.

Just a note at this point: Although it is theoretically possible to compress any data transferred via HTTP, under normal conditions it only makes sense to compress ASCII files where a considerable reduction in size can be achieved (for example.html, .pl, .php, .txt). Double compression of files such as images will normally cause an overhead that delays the data transfer.

More Speed?

For files that exceed the size defined in "mod_gzip_maximum_inmem_size" (that is, 64 kbytes by default), Mod_gzip will create a file in the temp directory, although this causes hard disk activity

Listing 1: "ramdisk.sh"

```
#/bin/bash

# Initialize the ramdisk
dd if=/dev/zero of=/dev/ram ➥
bs=1k count=4096
mke2fs -vm0 /dev/ram 4096
<\c>
# Create a mountpoint
mkdir -p /var/cache/ramdisk
chown nobody.nobody /var/cache➥
/ramdisk
chmod 770 /var/cache/ramdisk

#Mount the ramdisk
mount -t ext2 /dev/ram /var/➥
cache/ramdisk
```

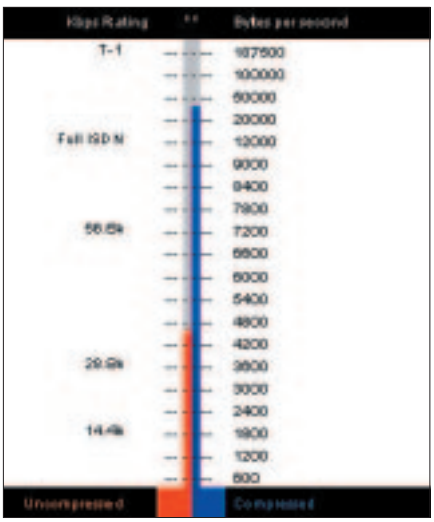

Figure 2: Compression improves the client data throughput by several orders of magnitude. This figure shows a 56k modem with a throughput of more than 30 kbyte/s

Listing 2: "compress.sh"

```
#!/bin/bash
ARGS=2
E_BADARGS=65
if [ $# -ne $ARGS ]
then
        echo "Syntax:   `basename $0` path suffix"
        echo "Example: `basename $0`  /home/httpd html"
        exit $E_BADARGS
fi
for directory in `find $1 -type d`
do
        for filename in $directory/*.$2
        do
                if [ -f $filename ]
                then
                        gzip -c9 -v $filename >$filename.gz
                fi
        done
done
```

INFO

[1] Mod_gzip Homepage: *http://www.remotecommunications.com/apache/mod_gzip*

[2] Michael Schröpls Mod_gzip Page: *http://www.schroepl.net/projekte/mod_gzip*

[3] Content Encoding via Perl Scripts: *http://www.schroepl.net/projekte/gzip_cnc*

[4] Performance Test on the Hyperspace Homepage: *http://www.ehyperspace.com/ html/solutions/performance.html*

and therefore wastes time. To increase your Apache's performance, you might like to swap your temp directory out to a RAM disk.

To do so, your kernel will need RAM disk support. You can use the following rule of thumb to calculate the size of the disk: "Maximum number of simultaneous requests multiplied by 100 kbytes" – this should allow you enough leeway to process requests even at peak times. However, since the kernel will only support RAM disks of 4 mbytes by default, you may need to create a new kernel, specifying the required size for "Block Devices | Default RAM disk size" in the configuration file.

After ensuring that a properly dimensioned RAM disk is available, you can use Listing 1 to initialize and mount the disk. Now simply add

```
mod_gzip_temp_dir ➥
/var/cache/ramdisk
```

to "httpd.conf" – and after a restart your Apache will save resources by writing temporary files to memory.

Since a RAM disk is volatile and disappears into the happy hunting grounds when you restart your computer, you might also like to add Listing 2 to an init script. This script needs to be run before you launch your Apache to initialize the RAM disk automatically on rebooting.

If you want your Apache to go faster than the speed of light, note that Mod_gzip allows you to serve up pre-compressed files. This saves the dynamic compression process, thus reducing the interval between the request and the return data by more than half.

If you add the following entry to "httpd .conf"

```
mod_gzip_can_negotiate Yes
```

the web server will check for a pre-compressed version of the file to be served up (with the ".gz" suffix) and return this to the client. If there is none, the file is compressed on the fly and returned. Listing 2 should help you create a compressed version of any files with a specific suffix (for example, ".html" or ".txt") in a specific directory and its subdirectories.

If you require additional information on Mod_gzip, you might like to visit Michael Schröpl's site [2]. Michael has his own project [3] running – a content encoding solution based entirely on scripts, that will be of interest to readers with FTP only access to their web servers who are unable to install Mod_gzip themselves. Anyone who would like to feel the difference that compression can make to a download should also try out the Hyperspace website [4]. ■

Listing 3: "httpd.conf"

```
# enable mod_gzip
mod_gzip_on Yes

# Temp directory
# (must be writable for Apache)
mod_gzip_temp_dir /tmp

# Keep temporary files?
# Set this to Yes for
# debugging only
mod_gzip_keep_workfiles No

# Exclude browsers whose
# content encoding
# implementation is faulty
mod_gzip_item_exclude reqheader➥
 "User-agent: Mozilla/4.0[678]"

# Exclude CSS and Javascript,
# since Netscape 4 cannot
# decompress these files
# properly
mod_gzip_item_exclude file \.js$
mod_gzip_item_exclude file ➥
\.css$

# Limit file size (in Bytes)
# Compression causes an overhead
# for files <0.5 kb
mod_gzip_minimum_file_size 500

# File >1 MB compression causes
# a delay in serving
mod_gzip_maximum_file_size ➥
1000000

# All files <60 Kb can be
# compressed in memory and need
# not be transferred to the swap
# directory
mod_gzip_maximum_inmem_size 60000

# Files to be compressed?
# HTML:
mod_gzip_item_include file \.htm$
mod_gzip_item_include file ➥
\.html$

# Text:
mod_gzip_item_include mime text/.*

# Scripts:
mod_gzip_item_include file \.pl$
mod_gzip_item_include file ➥
\.php$
mod_gzip_item_include handler ➥
^cgi-script$

# And finally a logfile
LogFormat "%h %l %u %t \"%V ➥
%r\" %&gt;s %b ➥
mod_gzip: %{mod_§§ gzip_result}n

In:%{mod_gzip_input_size}n ➥
Out:%{mod_gzip_output_size}n:%

{mod_gzip_compression_ratio}➥
npct." common_with_mod_gzip_info2

CustomLog /var/log/httpd/mod_➥
gzip common_with_mod_gzip_info
```

THE AUTHOR

Ulrich Keil studies Computer Science in Mannheim, and works part-time as a system administrator for 9 Net Avenue. In his leisure time Ulrich works as a volunteer emergency medical technician, and still finds time to take care of his personal Sparc station 10, that is still faithfully serving up his homepage at http://www.der-keiler.de.