Empirical Study on Effects of Compression Algorithms in Web Environment

Lukáš Čegan[1]

[1]University of Pardubice, Faculty of Electrical Engineering and Informatics,
Department of Information Technology, Pardubice, Czech Republic

Web resource compression is one of the most useful tools, which is utilized to accelerate website performance. Compressed resources take less time to transfer from server to client. This leads to faster rendering of web page content resulting in a positive impact on the user experience. However, content compression is time consuming and also brings extra demands on system resources. For these reasons, it is necessary to know how to choose a suitable algorithm in relation to particular web content. In this paper we present an empirical study on effects of the compression algorithms which are used in web environment. This study covers Gzip, Zopfi and Brotli compression algorithms and provides their performance comparison.

Keywords: Compression, Website, Gzip, Zopfi, Brotli.

1. INTRODUCTION

Today's web users are not very patient. They expected delivery content from web servers to their devices in a flash. Therefore, web developers, UX designers, software architects, network experts and many others care about many optimization technics and appropriate solutions that help them to delivery whole web content to the client as fast as a possible. One of these optimization techniques is appropriate usage of compression algorithms to compress web page resources. Compressing resources is a very effective way of reducing their size which is a very significant help in reducing time needed to transfer these resources between server and user's web browser. Unfortunately, every optimization solution has its pros and cons. The cons of compression consist in resource consumption, like CPU and memory, that are used during data processing. In the web environment are a number of different algorithm and many of them are very effective at quickly processing and compressing files. But not all of them are suitable for the various data formats that are in the WWW world.

*Email Address: Lukas.cegan@upce.cz

Some of them are ideal for frequently changing files which are encoded on-the-fly, because these algorithms are very fast. But these algorithms have not such a big compression ratio as others which are useful for static files such as images, CSSs, JavaScripts. These others algorithms have a big compression ratio, but they are slow. However, it may not be a hindrance for static content, because it can be easily preprocessed and deployed to the web server. But this practice is definitely inapplicable for dynamic generated content because it is created on-the-fly, on the server side. For this reasons it is necessary to have a deep knowledge of the performance data of different algorithms in different kinds of deployment. In this paper an empirical study on effects of different compression algorithms is performed, that brings performance results for mutual comparison.

The paper is organized as follows. After introducing the objective of this paper, the compression algorithms are presented in Section II. The Section III described the practical experiments and benchmark settings. The results of the experimental analysis are discussed in section IV. Finally, the last section gives conclusions and future research opportunities followed by references at the end.

2. BACKGOUND OF COMMPRESSION ALGORITHS

Compression algorithms are used in the digital world everywhere. Music is compressed by MP3, video by MPEG4, images by GIF, etc. In general, compression algorithms can be divided into two different group. The first group are lossless algorithms, which can reconstruct the original data exactly from the compressed data. These algorithms are mainly used to compress text information. The second group are lossy algorithms, which can only reconstruct an approximation of the original data. These algorithms are useful, for example, to compress audio, video and image data. The modern web browser can work with both groups of algorithm. For efficient communication between server and client it is especially important to compress text files such as source code of websites (HTML, CSS, JavaScripts, etc.). The web server mainly use compression formats such as Gzip, DEFLATE, Zlib and new one Zopfi or Brotli.

Gzip, DEFLATE, Zlib

Gzip (GNU zip) file format is based on the DEFLATE algorithm that is combination of the LZ77 (Lempel–Ziv, 1977) dictionary-based algorithm and Huffman coding. DEFLATE provides very good compression on a wide variety of data with minimal use of system resources. It was created as a free software replacement for LZW and other patent-encumbered data compression algorithms. The first version of algorithm was released in 1993. Zlib is a software library used for lossless data compressing and it is an abstraction of the DEFLATE compression algorithm [1]. Zlib was developed by Jean-loup Gailly (compression) and Mark Adler (decompression) and the initial version of Zlib was released in 1995 as free software under the Zlib license.

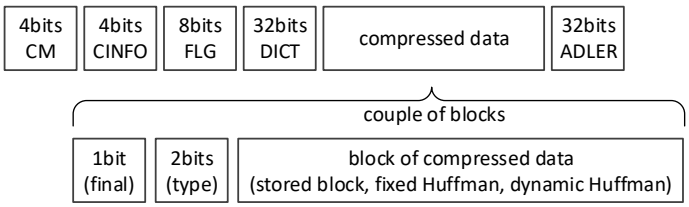

Fig.1. Zlib data structure

Zopfli

The Zopfli is a compression algorithm that is compatible with the DEFLATE algorithm used in Zlib. The algorithm was developed by the Google corporation and got its name from a Swiss bread recipe. The initial release of the algorithm was introduced in February 2013. The reference implementation of the Zopfli compression algorithm from Google is programmed in C language. It is an open source and it is distributed under the Apache License, Version 2.0 [2]. The performance of this algorithm is very good. It reduces files to sizes 3.7–8.3 percent smaller than other similar algorithms, but data processing is slow and consumes two to three times the CPU power of its competition [3]. For this reason, this algorithm is best suited for applications where data is compressed once, and then used many times, like static content for the web.

Brotli

Brotli compressed data format is a lossless compressed data format that compresses data using a combination of the LZ77 algorithm and Huffman coding. Development of this algorithm was initiated in Google labs and now it is distributed as open-sourced code under the MIT License. The Brotli specification is published in RFC7932 [4]. One of the main advantages of this algorithm is much faster decompression than common LZMA [5] implementations. The Brotli offers approximately the same speed of compression, but results of compression are denser. Brotli is currently used by several web browsers such as WOFF2 font compression [6]. The results of WOFF 2.0 Compression on Google Fonts, from a study, shows a significant reduction of the data size. The maximum improvement with WOFF 2.0 comes up to 61%. The average improvement reaches 26% [7]. Brotli is currently only supported in a few web browsers – Chrome, Opera, Firefox, Android browser, Chrome for Android [8].

3. EXPERIMENT DESIGN

Most modern browsers support web content decompression. They inform web servers about supporting algorithms by header “Accept-Encoding” in the HTTP request. Currently, most modern web browsers support GZIP and DEFLATE decompression. Other compression algorithms have only partial support in a small group of web browsers and very often they are supported only for experimental purpose. A web server informs a browser about the type of compressed algorithm which was used for compression content of a HTTP response via the header “Content-Encoding”. The possible values are:

- gzip - a format using the Lempel-Ziv coding with a 32-bit CRC,
- compress - a format using the Lempel-Ziv-Welch algorithm,
- deflate - using the zlib structure with the deflate compression algorithm,
- identity - indicates the identity function (no compression),
- br - a format using the Brotli algorithm.

Compression is a CPU and memory consumed process, with higher compression levels resulting in smaller files at the expense of CPU and memory. For this reason, it is always necessary to choose the best ratio among many parameters like compression density, the time needed for processing and consumption of system resources. Furthermore, the right processing method must be selected: pro-compression or compression on-the-fly. The performance impact of these parameters on the overall user experience is considerable and therefore we provided an empirical evaluation of the degree of impact. The evaluation was performed on Apache web servers with.

Testbed platform

A testbed platform consists of the physical machine Dell Latitude E6440, Intel(R) Core(TM) i5-4310M, 2.70 GHz, 8GB RAM, Windows 10 64 bit. and virtualization platform VMware Workstation 12. The virtual machine host server provides computing resources, such as processing power, memory, disk and network I/O, and so on. The guest is a completely separate and independent instance of the operating system. The virtual machine host represents the desktop client with web browser Chrome 53. The guest represents the server side with operation system Debian 8.6 and the web server Apache 2.4.10. The Apache server was configured with module: mod_deflate and apache-mod-brotli (see source code below).

```
1   # BROTLI
2   <IfModule mod_brotli.c>
3   LoadModule brotli_module
    modules/mod_brotli.so
4   BrotliCompressionLevel 11
5   BrotliWindowSize 22
6   BrotliFilterNote Input brotli_in
7   BrotliFilterNote Output brotli_out
8   BrotliFilterNote Ratio brotli_ratio
9   LogFormat '"%r" %{Brotli_out}n/%{Brotli_in}n
    (%{Brotli_ratio}n)' brotli
10  AddOutputFilterByType BROTLI text/htm
    text/html text/plain text/xml text/css image/gif
    image/png image/jpeg application/x-javascript
    application/javascript
11  </IfModule>
12
13  # DEFLATE
14  <IfModule mod_deflate.c>
15  DeflateCompressionLevel 9
16  AddOutputFilterByType DEFLATE text/htm
    text/html text/plain text/xml text/css image/gif
    image/png image/jpeg application/x-javascript
    application/javascript
17  </IfModule>
```

Experiment methodology

The impact of each compression algorithm was conducted on commonly used JavaScript library jQuery 3.1.0, on the very popular CSS framework Boostrap 3.3.7 and Foundation 6.2.3. Each of these libraries has been compressed with Gzip, Zopfli and Brotli with different levels of compression quality. In each measurement were monitored:

- Compress ratio – the ratio between the uncompressed and compressed data
- Time – the time required for data compression, measured by Linux utility Time
- CPU usage – CPU needed to compress data, measured by Valgrind tool.

The second part of the experiment was aimed at evaluating the impact of compression from the user's perspective. The impact of each compression algorithm was conducted on widely used CMS WordPress 4.6.1 and Joomla 3.6.2. Each algorithm was tested with several different parameters (if allowed). Individual measurements were made in three different simulated network environments: (A) Fiber – unlimited Mbit/s bandwidth and 50ms latency, (B) LTE – 10 Mbit/s and 50ms latency and (C) 3G – 1 Mbit/s bandwidth and 300ms latency. For creating a simulation environment Linux tool Netem (Network Emulator) was used which provides functionality for variable delay, loss, duplication and re-ordering with combination of traffic shaper tool TBF (Token Bucket Filter), which allows the slowing down of transmitted traffic, to the specified rate. For the impact of each compression algorithm, tests were performed repeatedly under HTTP/1.1 + SSL. In each scenario we measured:

- Compress ratio – the ratio between the uncompressed and compressed data
- PLT – page load time, measured by our own JavaScript application based on Navigation Timing API [13] which obtain performance data (DNS lookup, TCP connection, DOM loading, etc.) of every request in the browser.

All tests were performed with a cleaned cache.

4 EXPERIMENTAL RESULT AND DISCUSSION

Table 1 showed a compression density of the jQuery library, which is just one file in minification version. Further, Table 1 shows compression density of the framework Booststrap, which covers: bootstrap.min.css, bootstrap-theme.min.css, boot-strap.min.js, glyphicons-halflings-regular.svg files, and framework Foundation, which covers: foundation.min.css, app.js, foundation.min.js files.

Table.1. Compression density [B]

	jQuery	Bootstrap	Foundation
Uncompressed	86351	290392	185299
Gzip1	35010	73716	47349
Gzip5	30148	60291	37923
Gzip9	29885	58620	37257
Zopfli1	29040	55431	35792
Zopfli50	29013	55103	35642
Zopfli1000	29013	55076	35604
Brotli1	35982	70311	47331
Brotli5	29474	55470	35370
Brotli9	29147	54058	34560

Comparison of the compression density is shown in the following chart. Compression density is expressed as total_size_of_all_files_after_compression / total_size_of-_all_files_before_compression * 100%. As the graph shows, the best result was achieved by a Brotli with compression level 9 (see Figure 2).

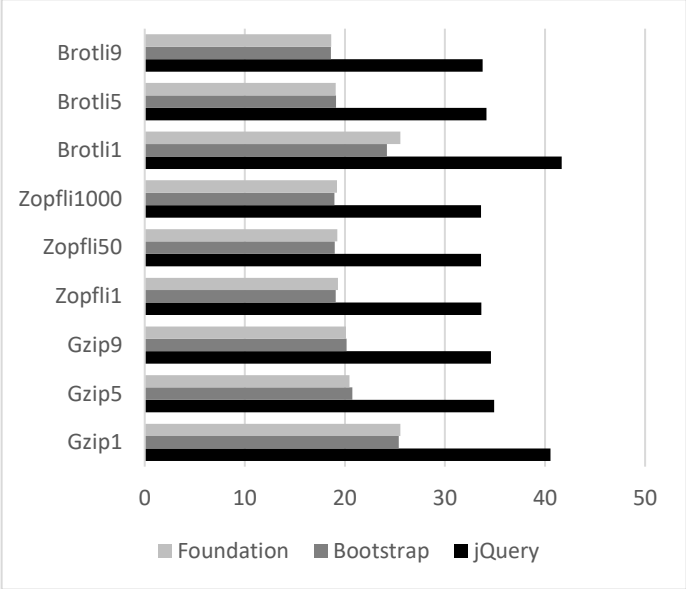

Fig.2. Compression density [%]

Table 2 shows the compression rate. The speed value is specified in bytes per millisecond. As results show, Zopfli is really slow.

Table.2. Compression speed [ms]

	jQuery	Bootstrap	Foundation
Gzip1	0.003	0.006	0.004
Gzip5	0.005	0.010	0.006
Gzip9	0.006	0.023	0.010
Zopfli1	0.103	0.609	0.405
Zopfli50	1.038	4.660	3.284
Zopfli1000	14.139	83.227	59.112
Brotli1	0.003	0.007	0.005
Brotli5	0.017	0.017	0.011
Brotli9	0.458	0.073	0.396

The next measured parameter was CPU usage. Table 3 shows the amount of CPU time spent in user-mode code (outside the kernel) and sys-mode (inside the kernel) within the process. Time is given in milliseconds. Again, the worst result was achieved by Zopfli.

The second part of the empirical study has focused on the evaluation of compression algorithms from the user experience perspective, which is also very important. The effectiveness of the compression algorithms has been investigated in three network scenarios: FIBER, LTE, 3G, and each scenario was tested on two websites based on Wordpress and Joomla CSM.

Table.3. CPU usage [ms]

	jQuery	Bootstrap	Foundation
Gzip1	0.000	0.004	0.004
Gzip5	0.004	0.008	0.004
Gzip9	0.004	0.020	0.008
Zopfli1	0.092	0.596	0.392
Zopfli50	0.888	4,636	3.264
Zopfli1000	14.084	82.964	56.914
Brotli1	0.000	0.004	0.000
Brotli5	0.004	0.008	0.012
Brotli9	0.132	0.052	0.072

The total size of each website is shown in Table 4. The uncompressed size of tested web pages is from 2.7 to 3.4, which is, according to available statistics, a common size of web pages today.

Table.4. Size of website [Mb]

	WordPress	Joomla
Uncompressed	3.4	2.7
Gzip1	2.9	2.1
Gzip9	2.9	2.1
Brotli1	2.9	2.1
Brotli11	2.8	2.0

Table.5. Page load time [s]

	WordPress	Joomla
LTE		
Uncompressed	1.422	1.357
Gzip1	1.262	1.230
Gzip9	1.221	1.259
Brotli1	1.282	1.311
Brotli9	1.189	1.282
3G		
Uncompressed	32.243	26.376
Gzip1	27.125	22.052
Gzip9	28.012	21.297
Brotli1	27.237	21.998
Brotli9	28.068	22.138
FIBER		
Uncompressed	3.674	3.478
Gzip1	3.543	3.286
Gzip9	3.571	3.287
Brotli1	3.552	3.129
Brotli9	3.491	2.933

Table 5 shows page load time for each scenario and each website, which expresses the time required to fully display the content of a specific page.

5. CONCLUSIONS

This paper presents an empirical study on effects of compression algorithms in the web environment. Assessment of the algorithms were divided into two branches: static and dynamic web content. The demonstrated results in the static web branch show, that commonly used Gzip is very fast and has a small CPU footprint. Zopfi is better than Gzip in compressing, but it is much slower. However, for a static web it is not a disadvantage, because all web resources are pre-compressed and stored in the web server for use. From this perspective, Zopfi is the most appropriate tool for the static web. In the dynamic web branch, the situation is different. Zopfi is very slow, therefore it is totally inappropriate for dynamically generated content. The results demonstrate that Brotli offers a significantly better compression ratio while keeping decompressing speed relatively close to Gzip. From the user perspective, even this small improvement can mean a significantly faster rendering of a web page with large files, which leads to the achievement of better user experience. Brotli has potential to become the most commonly used compression algorithm in WWW for on-the-fly compression. Unfortunately, the disadvantage of Brotli is incompatibility with the current most widely used format DEFLATE, which can lead to a slower expansion of support in major browsers.

ACKNOWLEDGMENTS

This work is published thanks to the financial support Faculty of Electrical Engineering and Informatics, University of Pardubice under grant TG02010058 “Podpora aktivit proof-of-concept na Univerzitě Pardubice”.

REFERENCES

[1] P. DEUTSCH and J-L. GAILLY, ZLIB Compressed Data Format Specification version 3.3, In: Internet Engineering Task Force (IETF) [cit. 2016-09-17]. Received: https://tools.ietf.org/html/rfc1950

[2] Zopfli Compression Algorithm [online]. [cit. 2016-09-17]. Received: https://github.com/google/zopfli

[3] JYRKI ALAKUIJALA a LODE VANDEVENNE. Data compression using Zopfli. [online]. [cit. 2016-09-17]. Received: https://ru.scribd.com/document/319797551/Data-compression-using-Zopfli-pdf.

[4] Z. SZABADKA a J. ALAKUIJALA. Brotli Compressed Data Format [online]. In: Internet Engineering Task Force (IETF) [cit. 2016-09-17]. Received: https://www.ietf.org/rfc/rfc7932.txt.

[5] LI, Bing, Lin ZHANG, Zhuangzhuang SHANG a Qian DONG. Implementation of LZMA compression algorithm on FPGA. Electronics Letters. 2014, 50(21), 1522-1524. DOI: 10.1049/el.2014.1734. ISSN 0013-5194. Received: http://digital-library.theiet.org/content/journals/10.1049/el.2014.1734.

[6] WOFF File Format 2.0: W3C Candidate Recommendation 15 March 2016 [online]. [cit. 2016-09-17]. Received: https://www.w3.org/TR/ WOFF2/# table_format.

[7] KUETTEL, David. WOFF 2.0 Compression w/ Google Fonts [online]. In: Google [cit. 2016-09-17]. Received: https://docs.google.com/spreadsheets/d/1DxoOZLA1QywIzwmWr0Pc0GAp15YDnB-4JbJiKWQNgo8/edit#gid=0

[8] Can I Use, Brotli, Received: http://caniuse.com/#search=brotli