Data compression using Zopfli

Jyrki Alakuijala, Ph.D. and Lode Vandevenne, M.Sc.
Google Inc.

Abstract — We measure the performance of the Zopfli compression algorithm and compare it with other implementations of deflate compression. We show that Zopfli has the highest compression density of all deflate compatible algorithms we compared, on four compression corpora we used for testing. Zopfli uses significantly more time in compression, but decompression speed of Zopfli-generated output is comparable with other algorithms.

Introduction

Zopfli is a new deflate compatible compressor that was inspired by compression improvements developed originally for the lossless mode of WebP image compression. Being compatible with deflate makes Zopfli compatible with zlib and gzip. Most internet browsers support deflate decompression, and it has a wide range of other applications. This means that Zopfli-compatible decompression is readily widely available.

In this study we compare the compression density of the Zopfli compressor with the compression density of zlib [1], the most common deflate implementation used today, as well as two lesser known deflate implementations, 7-zip [2] and kzip [3]. We use four data compression corpora [4–7] to measure the compression density. We also measured the compression and decompression speeds for one test corpus. In the conclusion we suggest a potentially important use case for the new Zopfli data compression algorithm.

Data compression works by eliminating statistical redundancy from the data. The redundancy can be, for example, in the form of some symbols occurring more often or sequences of symbols repeating.

There are many benefits to higher compression density. The smaller compressed size allows for storing more items in less space, faster data transmission, and lower web page load latencies. Furthermore, the smaller compressed size has additional benefits in mobile use, such as lower data transfer fees and reduced battery use. The higher data density is achieved by

using more exhaustive compression techniques, which make the compression a lot slower, but the decompression speed is not affected.

Methods

We chose several sets of files (corpora) for running the compressors:

- a web-centric benchmark by downloading the homepages of the 10000 most popular websites as given in the Alexa Internet directory [4]. 9148 pages were successfully loaded to form our corpus.
- The Calgary Corpus is a collection of small text and binary data files, commonly used for comparing data compression algorithms. [5]
- Canterbury Corpus, a compression corpus designed for lossless data compression. It was suggested as a replacement for the older Calgary Corpus. [6]
- enwik8 [7] has been developed as a large text compression benchmark, consisting of 100 million bytes of English Wikipedia.

For running the benchmarks, we used an Ubuntu-derivative Linux operating system with kernel 3.2.5 (x86_64) on Dell Precision T3500 Intel Xeon CPU X5650 running at 2.67 GHz. The versions of the various software used in experiments are Zopfli (https://code.google.com/p/zopfli/source/browse/ revision acc035299f8d), gzip 1.4, 7-Zip (A) [64] 9.20, and kzip (release 20091108). The compiler we used is gcc version 4.6.3. We run Zopfli and kzip with default arguments, gzip with -9, and 7-zip with -mm=Deflate -mx=9.

Results

Compression results (Table 1) indicate that Zopfli produces the most dense output, but is slowest (Table 2) of all the algorithms we tested. Uncompression time (Table 3) is unaffected by the selection of the compression algorithm.

Table 1. Compressed data size for the four file corpora and for common compression algorithms along with Zopfli. The output produced by Zopfli is 3.7–8.3 % smaller than that of gzip -9.

Benchmark	Corpus size	gzip -9	7-zip	kzip	Zopfli
Alexa-top-10k	693’108’837	128’498’665	125’599’259	125’163’521	123’755’118
Calgary	3’141’622	1’017’624	980’674	978’993	974’579
Canterbury	2’818’976	730’732	675’163	674’321	669’933
enwik8	100’000’000	36’445’248	35’102’976	35’025’767	34’995’756

Table 2. Compression times for enwik8. Zopfli is 81 times slower than the fastest measured algorithm gzip -9.

Compression algorithm	Compression time
gzip -9	5.60 s
7-zip -mm=Deflate -mx=9	128 s
kzip	336 s
Zopfli	454 s

Table 3. Uncompression times for data that were compressed with different algorithms are tested with running “gzip -d” for the compressed enwik8 corpus. We obtained the run times from 9 runs, and chose the median time. The difference between fastest and slowest are within 2.5 %.

Compression algorithm	Uncompress time for “gzip -d” of enwik8
gzip -9	934 ms
7-zip -mm=Deflate -mx=9	949 ms
kzip	937 ms
Zopfli	926 ms

Discussion

Zopfli gives smaller deflate-compatible output size than gzip (3.7–8.3 % smaller), 7-zip, and kzip, with more CPU time used at compression phase. This makes Zopfli ideal for uses where the cost of CPU is small in relation to the output size. Such use could include denser compression of static content for making websites faster.

Zopfli and gzip compress to gzip format, whereas kzip and 7-zip compress to zip format. This may alter the sizes slightly as the container format has slightly different overhead. For enwik8, the header overhead difference is below 0.0001 % in relation to the output size.

7-Zip can operate with the deflate format, but it can read and write several other archive formats, and achieve higher compression ratios. In this study we only measured deflate-compatible compression.

We could achieve faster results with gzip and other algorithms by specifying lower compression density options. In this study we are interested on finding the smallest possible

compressed size, and because of this we have only run every algorithm with maximum compression options. Zopfli also can run even longer to achieve slightly higher compression density, but we chose to run it with default settings.

In the light of the results we presented, we recommend Zopfli for compression of static content and other content where data transfer or storage costs are more significant than the increase in CPU time. To our knowledge Zopfli typically produces the highest compression density of any deflate-compatible algorithm.

Zopfli is opensourced at https://code.google.com/p/zopfli/. We invite everyone to try it out, and hope that it will find many practical uses.

References

1. http://en.wikipedia.org/wiki/Zlib
2. http://en.wikipedia.org/wiki/7-Zip
3. http://www.advsys.net/ken/utils.htm
4. Alexa-top-10k corpus: https://code.google.com/p/httparchive/source/browse/trunk/lists/Alexa10K.txt
5. Calgary corpus: http://www.data-compression.info/Corpora/CalgaryCorpus.zip
6. Canterbury corpus: http://corpus.canterbury.ac.nz/resources/cantrbry.zip
7. enwik8 corpus: http://mattmahoney.net/dc/text.html http://mattmahoney.net/dc/enwik8.zip
8. P.Deutsch, RFC 1952—GZIP file format specification version 4.3, http://www.ietf.org/rfc/rfc1952.txt
9. P. Deutsch, RFC 1951—DEFLATE Compressed Data Format Specification version 1.3, http://www.ietf.org/rfc/rfc1951.txt