Comparison of Brotli, Deflate, Zopfli, LZMA, LZHAM and Bzip2 Compression Algorithms

Jyrki Alakuijala, Evgenii Kliuchnikov, Zoltan Szabadka, and Lode Vandevenne
Google, Inc.

Abstract—This paper compares six compression techniques, and based on the results proposes that brotli could be used as a replacement of the common deflate algorithm. We compared the performance of brotli by measuring the compression ratio and speed, as well as decompression speed on three different corpora: the Canterbury compression corpus, an ad hoc crawled web content corpus, and enwik8. On all three corpora we show performance superior to that of deflate. Further, we show that Zopfli, LZMA, LZHAM and bzip2 use significantly more CPU time for either compression or decompression and could not always work as direct replacements of deflate.

Introduction

Much of the practical lossless data compression is done with the deflate algorithm, not only because it is well supported by existing systems, but also because it is relatively simple and fast to encode and decode. In 2013 we launched Zopfli [1], a compression algorithm that allows for denser compression while remaining compatible with the deflate format. While Zopfli is now well-accepted in the field, there were opinions expressed that we should move on from the deflate file format to a modern solution. Brotli [2] is our attempt at building a compression format and an example implementation of this format that is fundamentally more efficient than deflate. In this paper we measure the performance of our implementation and compare it with deflate and a few other compression algorithms.

Methods

The tests were run with a 22 bit window size for brotli, LZMA and LZHAM, and a 15 bit window size for deflate and zopfli. We used a 22 bit window size because past experience showed that larger windows can be slower to decode. Larger window sizes tend to give a higher compression ratio at the expense of decoding speed. For deflate and zopfli we used the maximum size allowed by the format. The versions of the algorithms tested are:

- brotli version 0.2.0 [2],
- deflate algorithm from zlib 1.2.8 [3],
- Zopfli version from github 2015-09-01 [4],
- LZMA implementation in 7zip 9.20.1 [5],
- LZHAM 1.0 stable 1 [6], and
- bzip2 1.0.6, 6-Sept-2010 [7].

The test computer we used is an Intel® Xeon® CPU E5-1650 v2 running at 3.5 GHz with six cores and six additional hyper threading contexts. We run linux 3.13.0. All codecs were compiled using the same compiler, GCC 4.8.4 at -O2 level optimization. All tests were run single-threaded on an otherwise idle computer.

The compression corpora we used in the testing are the Canterbury compression corpus [8], an ad hoc crawled web content corpus, 1'285 files, 70'611'753 bytes total, and enwik8, a single file corpus that is used in the Hutter prize [9]. The average file size on the web content corpus is only 55 kB, so the larger window size advantage of advanced algorithms over deflate mostly disappears there.

We measured the compression ratio, compression speed and decompression speed for selected algorithms and compression levels. The compression and decompression speed of each algorithm were measured with the same benchmark program that called the compression and decompression routines of each algorithm from statically linked libraries.

We limited the selection of algorithms to those that generally have a higher compression ratio than that of deflate. For this reason we excluded algorithms like lz4 and zstd from this study.

Unlike other algorithms compared here, brotli includes a static dictionary. It contains 13'504 words or syllables of English, Spanish, Chinese, Hindi, Russian and Arabic, as well as common phrases used in machine readable languages, particularly HTML and JavaScript. The total size of the static dictionary is 122'784 bytes. The static dictionary is extended by a mechanism of transforms that slightly change the words in the dictionary. A total of 1'633'984 sequences, although not all of them unique, can be constructed by using the 121 transforms. To reduce the amount of bias the static dictionary gives to the results, we used a multilingual web corpus of 93 different languages where only 122 of the 1285 documents (9.5 %) are in languages supported by our static dictionary.

In averaging over the results of individual files and over the corpora we chose to use geometric mean instead of the more common arithmetic mean. The geometric mean gives a bit more weight for poor performance, i.e., if a particular algorithm compresses one file type extremely fast or densely, it will not be propagated into the results as strongly as with an arithmetic mean.

Results

Tables 1, 2 and 3 show the results of three different corpora. Figure 1 shows the decompression speed vs. compression ratio on Canterbury corpus, showing graphically some of the results from Table 1. We can see from the tables that brotli at quality setting 1 (short-hand notation in this document brotli:1) compresses and decompresses roughly the same speed as deflate:1, but offers 12–16 % higher compression ratio. Brotli:9 is again roughly similar with deflate:9 on the Canterbury and web content corpora, but gives a speed increase of 28 % in decoding of enwik8, and a compression ratio increase of 13–21 %. Brotli:11 is significantly faster in compression than zopfli and gives 20–26 % higher compression ratio.

Brotli gives slightly faster decompression than deflate for the tested corpora, while other advanced algorithms (LZMA, LZHAM and bzip2) are slower than deflate. The geometric means for all reported decompression speeds in the table are 342.2 MB/s for brotli and 323.6 MB/s for deflate, a 5.7 % advantage for brotli.

In compression brotli:1 is similarly 5.7 % faster than deflate:1, but brotli:9 happens to be 32.3 % slower than deflate:9. However, one should not compare compression speed simply by the quality setting. A more useful comparison is to consider compression speed for an aimed compression ratio. Often brotli:1 is close to deflate:9, and sometimes even exceeding its compression ratio. For example, when compressing the Canterbury corpus down to 3.3 ratio one could use either brotli:1 at 98.3 MB/s or deflate:9 at 15.5 MB/s.

LZMA can compress enwik8 with a 2.5 % higher compression ratio, but that comes with a penalty of 3.5 times longer decompression time. LZHAM:4 is somewhat similar in performance with brotli:11: 1 % higher compression ratio at a cost of 25 % slower decompression speed. On shorter files, like Canterbury corpus and web documents, brotli's compression ratios are unmatched by LZMA and LZHAM.

Bzip2 was able to compress some text files rather well, but in the overall results it falls behind.

Table 1. This table shows the results of compression algorithms on the Canterbury corpus. The Canterbury corpus contains 11 files, and we show the geometric mean for the measured attributes: compression ratio, compression speed and decompression speed.

Algorithm: quality setting	Compression ratio	Compression speed [MB/s]	Decompression speed [MB/s]
brotli:1	3.381	**98.3**	334.0
brotli:9	3.965	17.0	**354.5**
brotli:11	**4.347**	0.5	289.5
deflate:1	2.913	93.5	323.0
deflate:9	3.371	15.5	347.3
zopfli	3.580	0.2	342.1
lzma:1	3.847	10.2	70.0
lzma:9	4.240	3.9	71.7
lzham:1	3.836	3.9	116.0
lzham:4	3.952	0.5	117.7
bzip2:1	3.757	11.8	40.4
bzip2:9	3.869	12.0	40.2

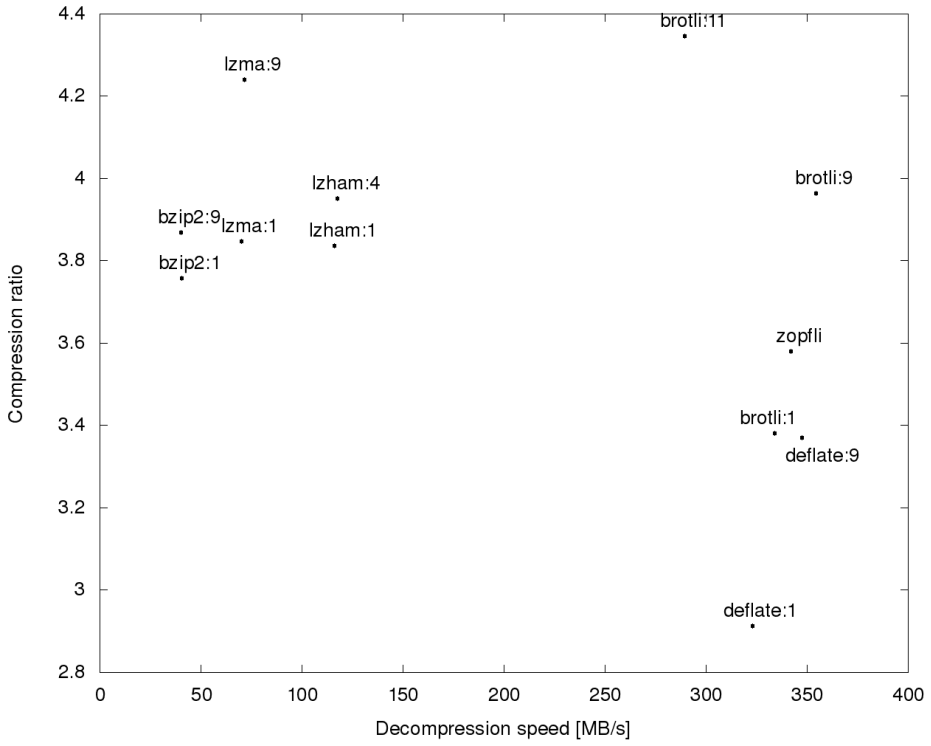

Figure 1. *The decompression speed vs. compression ratio of the Canterbury corpus as a scatter plot. For the decompression speed vs. compression ratio, brotli:9 and brotli:11 form the pareto-optimal front.*

Table 2. Results of the compression algorithms on a sample of documents crawled from the Internet. The sample consists of 1285 HTML documents, with 93 different languages.

Algorithm: quality setting	Compression ratio	Compression speed [MB/s]	Decompression speed [MB/s]
brotli:1	5.217	145.2	508.4
brotli:9	6.253	30.1	**508.7**
brotli:11	**6.938**	0.6	441.8
deflate:1	4.666	**146.9**	434.8
deflate:9	5.528	32.9	484.1
zopfli	5.770	0.2	460.1
lzma:1	5.825	7.9	100.5
lzma:9	6.231	4.4	102.2
lzham:1	5.580	4.7	168.7
lzham:4	5.768	0.2	172.7
bzip2:1	5.710	11.0	52.3
bzip2:9	5.867	11.1	52.3

Table 3. Results of different compression algorithms on the enwik8 file.

Algorithm: quality setting	Compression ratio	Compression speed [MB/s]	Decompression speed [MB/s]
brotli:1	2.711	**78.3**	228.6
brotli:9	3.308	5.6	**279.4**
brotli:11	3.607	0.4	257.4
deflate:1	2.364	70.8	211.7
deflate:9	2.742	18.1	217.4
zopfli	2.857	0.6	227.7
lzma:1	3.106	9.8	60.6
lzma:9	**3.696**	3.44	71.8
lzham:1	3.335	2.4	177.9
lzham:4	3.643	0.4	192.2
bzip2:1	3.007	12.3	30.8
bzip2:9	3.447	12.4	30.3

Discussion and Conclusions

These comparisons are done with a fixed width (22 bits) backward reference window. Other algorithms could possibly benefit from different widths. LZMA and LZHAM are commonly applied with a larger window. In this study we are looking for a replacement candidate algorithm for the deflate algorithm, and a larger window could slow down encoding and decoding as well as use more memory during decoding. Applying the same backward reference window size in LZMA, LZHAM and brotli removes one complication from the comparison.

Brotli uses a static dictionary that can be helpful for compressing short files. Other algorithms could be easily modified to do the same, and they would obtain slightly better compression ratios. For a long file like enwik8 a static dictionary is not very helpful. Canterbury corpus contains short documents with English, and there brotli's static dictionary might be giving it an unfair advantage.

Our results indicate that brotli, and only brotli out of all the benchmarked algorithms, would be a good replacement for the common use cases of the deflate algorithm in all three aspects, compression ratio, compression speed, and decompression speed.

References

1. https://zopfli.googlecode.com/files/Data_compression_using_Zopfli.pdf
2. https://github.com/google/brotli/releases/tag/v0.2.0
3. http://www.zlib.net/
4. https://github.com/google/zopfli/commit/89cf773beef75d7f4d6d378debdf299378c3314e
5. http://www.7-zip.org/history.txt
6. https://github.com/richgel999/lzham_codec/releases/tag/v1_0_stable1
7. http://www.bzip.org/
8. http://corpus.canterbury.ac.nz/
9. http://prize.hutter1.net/