LZ4 vs. GZIP comparison

Alan Ghobadi & Fredrik Ingebrigtsen

June 2, 2015

1 Introduction

For this project the two lossless compression formats, gzip [8] and LZ4 [4] has be compared in different experiments. Speed, compression ratio as well as end to end network transmission. Real world text data is compared, such as JSON-documents. Gzip is a well established algorithm and widely used. This project will compare gzip to the new contender LZ4 to see if there are any advantages of using LZ4.

In general gzip is slower than LZ4 while having a slightly better compression ratio, both have an compression ratio of about 1.5-3x depending on data and settings used. We will see in our experiments that this trade off matters in different environments. In network transmission for example it depends on how how much bandwidth is available. If one can get good throughput LZ4 is a better choice over gzip as more of the time will be spend on compressing than actual transferring of data. Additionally LZ4 has a high-compression mode (LZ4-HC) that compresses more slowly but with a better compression ratio, while still being able to be decompressed with the same speed as the normal LZ4 mode. This is beneficial in particular when pre-compressing data or when compression time is not an issue.

2 Background

2.1 Compression algorithms

Both of the lossless compression algorithms focuses mainly on speed over compression ratio. They do however reach a good compression ratio considering speed which is why they are widely used.

2.2 Gzip

Gzip is a compression format based on the compression algorithm DEFLATE [1] which combines LZ77 [6] and Huffman coding [2]. The format was developed by the GNU Project for UNIX systems in the early 1990s.

The DEFLATE algorithm uses the LZ77 algorithm with a sliding window of 32kB to compress a block. After Huffman trees are used to compress the result even more.

Gzip is commonly used in the real world for many different applications. It can be used by web servers to deliver compressed responses (if the browser

supports this) to saved bandwidth. For example to delivery large JSON- or XML-documents over HTTP.

It is widely available across different platforms such as Windows, UNIX systems such as BSD and Linux and on embedded platforms. It is available as a component on these platforms as a library called zlib [7].

2.3 LZ4

The LZ4 compression algorithm is a relatively new compression algorithm based on previous LZ-algorithms and first released in 2011 by Yann Collet [4]. There are is a reference implementation in C [3] and there are also plenty of other implementations in other languages such as node.js or c++, and is available on a wide range of platforms.

The LZ4 algorithm is as the name suggests also part of the LZ-family, which means a sliding window is used to find matches. The actual way finding of matches (for example full search) however is up to the implementation. The specification only contains information about how to store matches. This allows for different flavors of LZ4, such as LZ4-fast which is a fast mode and LZ4-HC which is a high compression ratio mode, relative to the default mode. The reference implementation uses hash-chains to better find matches. One of LZ4s strong features is the fast decompression. It does not rely on compression mode due as long as the specification format is honored during compression.

The storage format consists of blocks of 64kB where each block contains sequences [5]. As seen in figure 1, each sequence starts with a 1 byte token which is split into two parts, 4 bits each. First 4 bits of the token indicates literal length (0-15) in bytes whereas the last 4 bits of the token indicates the match length (0-15) in bytes. If any of these values are 15, it means that the length described is bigger than 15 bytes. In this case the optional length bytes are added (0-255) if this in turn is 255, another byte is added and so on. After the token the optional literal length is stored, as described. Following this is the storage of actual literals and then the match offset, these values together with the token are mandatory. The rest is optional. Finally the sequence ends with the optional match length, as described before.

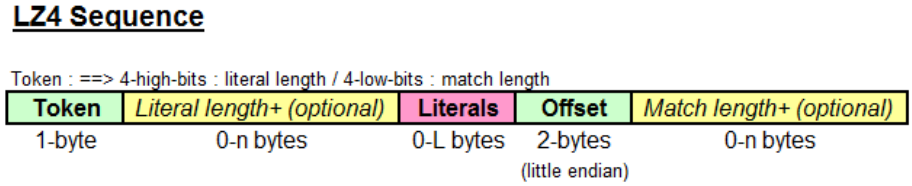


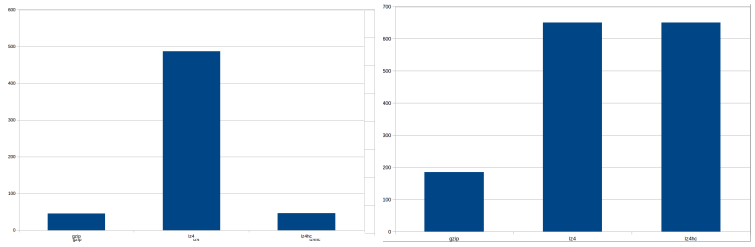
Figure 1: Structure of LZ4 block sequence [5]

3 Experiments and Results

There are three factors that make up a compression algorithms performance, compression ratio (how much it compresses), compression speed and decompression speed. In our experiments we decided to put our focus on JSON-documents, and their performance in a network situation. The JSON-documents used are

of a varying size, from as big as 200 MB down to 1 MB. The results presented are all taken from the average of the different JSON-documents performance in a given experiment. The experiments compare three compression algorithms, Gzip, LZ4 and LZ4 HC (the high compression variant of LZ4)

3.1 Speed



(a) Compression speed (b) Decompression speed

Figure 2: Speed

As expected LZ4 is a much faster algorithm than Gzip. This is particularly visible in the decompression speed, which is one of the strongest points of LZ4/LZ4 HC. In our case LZ4 decompresses at around 650 MB/s, but it has been shown on more powerful systems to reach speeds up to 2000 MB/s, almost reaching RAM speeds.

3.2 Ratio

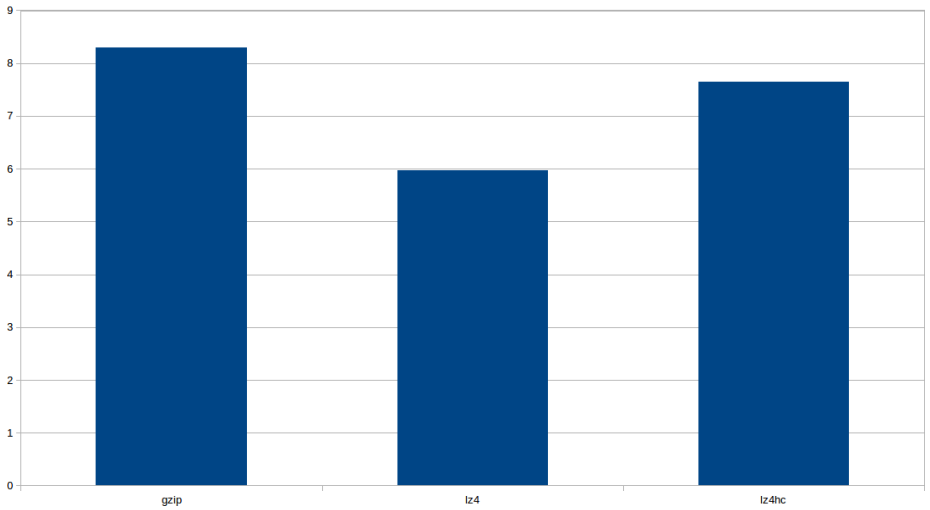


Figure 3: Compression ratios

Our experiments for compression ratio produced very high ratios, something that can be explained by the documents used. The JSON-documents proved

indeed to be very compressible, something not too surprising given their repetitive structure. Still the results we got are comparable to others have found in that Gzip will give the best ratio, closely followed by LZ4 HC.

3.3 Simulations

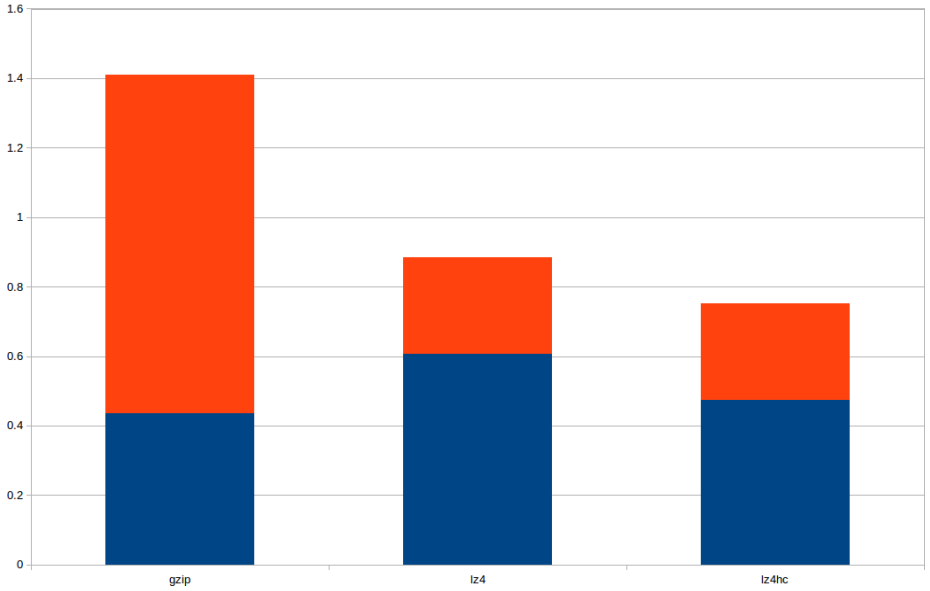


Figure 4: Static asset load

In this simulation we reproduced a scenario similar to loading already compressed files from the disk to memory and decompressing. This is quite a common scenario where compression is used, two examples are video games and OS kernels.

We used the largest JSON document for this simulation, but as it was only 200 MB we had to tone down the speed of the imagined disk transfers otherwise the decompression was the only participating factor.

LZ4 HC gave the best results, not too surprising when it compresses at almost the same ratio as Gzip, but can follow it up with some impressing decompression speeds.

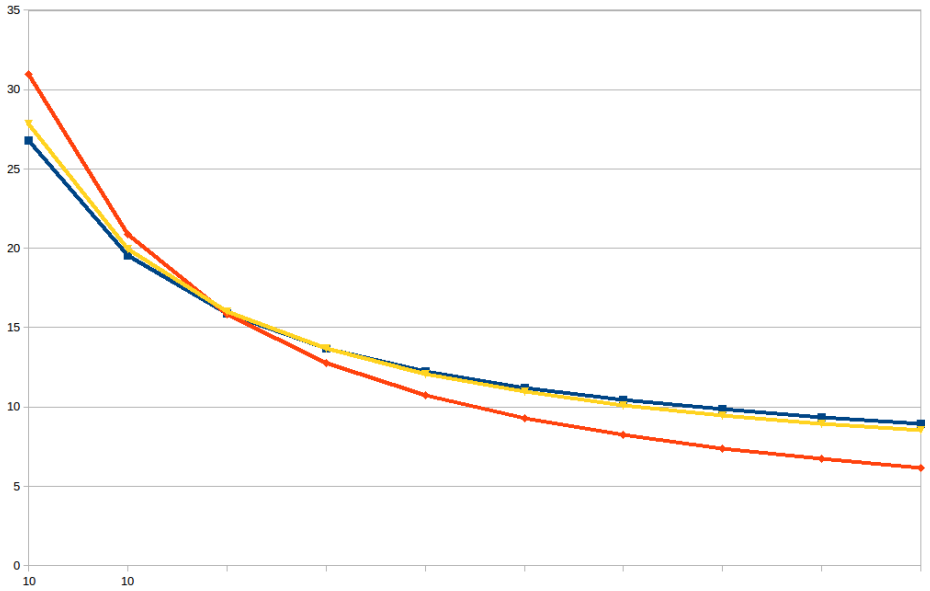


Figure 5: Network simulation. Red-Gzip, Yellow-LZ4HC, Blue-LZ4

This is the network simulation where we take a look at performance over varying network speeds. The x-axis on the graph is the network speed, starting at 1 Mbit/s and increasing in 0.5 MB/s steps. Y-axis is time as in the previous simulation, but here compression time is also taken into consideration. We are measuring here the average time it takes for all JSON-documents to be compressed, transferred and decompressed.

As seen from the graph Gzip is the preferred algorithm at lower network speeds but is quickly surpassed by the LZ4 variants (at 2 Mbit/s). This shows us that if you are transferring over high speed networks, and today most network operates at a higher speed than 2 Mbit/s, LZ4/LZ4 HC will be the favored algorithms.

As an interesting anecdote, Gzip's (or DEFLATE's rather) rise to popularity in the mid 90's was almost solely because of the eruption of the internet, which with the horrible network speeds of that day was in dire need of a good compression algorithm. Has the time for change finally come? Should we embrace this new guy, just because he bigger muscles? Lets head over to the conclusion (it's past the specifications) to find out!

3.4 Specifications

In conducting the experiments the following computer and software was used: 2.67 GHz Intel Core i7, 8 GB 1600 MHz DDR3, Ubuntu 14.04, Gzip 1.6, LZ4 r114.

4 Conclusion

We have looked at the performance of Gzip, LZ4 and LZ4 HC on JSON-documents, and the result have confirmed our initial expectations. LZ4 show

tremendous speeds both in compression and decompression, with the option of greater compression ratio in the high compression version. When simulating real world application of the algorithms it became apparent that LZ4 is almost always achieves better results. For transferring and decompressing a file from a storage device LZ4 HC is the best alternative out there, and is consequently seeing more and more use as such in the software industry. It is currently been adapted by several different companies in many different contexts ranging from operating systems (Linux, FreeBSD), search engines (Lucene, solr), databases (MySql, RocksDB) and games (Battlefield 4, Guild Wars 2). In our network simulation we saw that LZ4 is the stronger suit for network speeds above 2 Mbit/s. Network speeds today are indeed often higher (or much higher) than this, but here the new alternative have not been embraced on the same scale. The reasons that Gzip still is the most prevalent compression algorithm in network situation are many, the most significant being that the benefit of change is not that high. Change takes a lot of effort and time, so for it to happen the reward must generally outweigh the cost. In the network domain the limiting factor normally isn't compression, so it is understandable why adaptation of a slightly faster algorithm takes time.

But maybe one day a new and even faster compression algorithm is found, one that is universally adopted across the internet. Who knows when and where it will be created, or by who the creator will be. That is impossible to say. But one thing isn't:

> *"A good hockey player knows where the puck is. A great hockey player knows where the puck is going to be."* - Wayne Gretzky

References

[1] Deflate. http://en.wikipedia.org/wiki/DEFLATE, 2015. [Online; 2015-04-16].

[2] Huffman coding. http://en.wikipedia.org/wiki/Huffman_coding, 2015. [Online; 2015-04-16].

[3] Lz4 c reference implementation. https://github.com/Cyan4973/lz4, 2015. [Online; 2014-12-17].

[4] Lz4 compression algorithm. http://en.wikipedia.org/wiki/LZ4_(compression_algorithm), 2015. [Online; 2015-04-16].

[5] lz4explained. http://fastcompression.blogspot.fr/2011/05/lz4-explained.html, 2015. [Online; 2015-06-02].

[6] Lz77 and lz78. http://en.wikipedia.org/wiki/LZ77_and_LZ78, 2015. [Online; 2015-04-16].

[7] zlib. http://en.wikipedia.org/wiki/Zlib, 2015. [Online; 2015-06-02].

[8] Mark Adler Jean-loup Gailly. The gzip homepage. http://www.gzip.org/, 2015. [Online; 2015-04-16].