On Exploiting gzip's Content-Dependent Compression

Matteo Franzil, Luis Augusto Dias Knob

1. Introduction

Despite the development of more recent compression techniques like `bzip2` and `xz`, `gzip` is still a well-liked UNIX compression tool. Gzip is employed as the default compression technique in the majority of Linux and UNIX distributions. Although it has a somewhat poor compression ratio when compared to other utilities, this can be justified by its simplicity and quick compression and decompression times [1, 4]. This is also true for some tools, like `containerd`, which employs gzip as its standard compression technique for its image layers [2, 3].

In this report, we explore the possibility of exploiting gzip's algorithm for artificially increasing the decompression time of a file. By creating files filled with semi-random data generated with various methods, we show that compression and decompression times may vary significantly depending on the content of the file and the compression level used. Indeed, we show that the decompression time of a file can be increased by a factor of 3 in the worst case, when compared to the decompression time of a file containing English text.

2. Results

2.1. System setup

We run our experiments on two machines, one with a 4-core Intel(R) Xeon(R) Silver 4112 CPU @ 2.60GHz and 64GB of RAM, and a MacBook Pro (late 2020) with a M1 chip and 16GB of RAM. The first machine runs Ubuntu 20.04.2 LTS, while the MacBook runs macOS Ventura 13.3.1 (a). We used gzip version 1.12 on both machines. Finally, tests were run in isolation and with CPU pinning, in order to reduce the impact of other processes on the results, and were run multiple times to reduce the impact of noise.

Our tests comprised the following steps:

1) Generate a file with a given size and content
2) Compress the file with gzip
3) Decompress the file with gzip

We measured the time taken by each step, the size of the compressed and uncompressed files, the CPU usage, and the compression ratios. We repeated this test for each of `gzip`'s compression levels (1-9).

Furthermore, we executed tests both with files between 100MB and 1GB in size, in order to verify if a file's size impacted the results [However, we found that the results were comparable, and thus we only report the results for files of size 1GB.

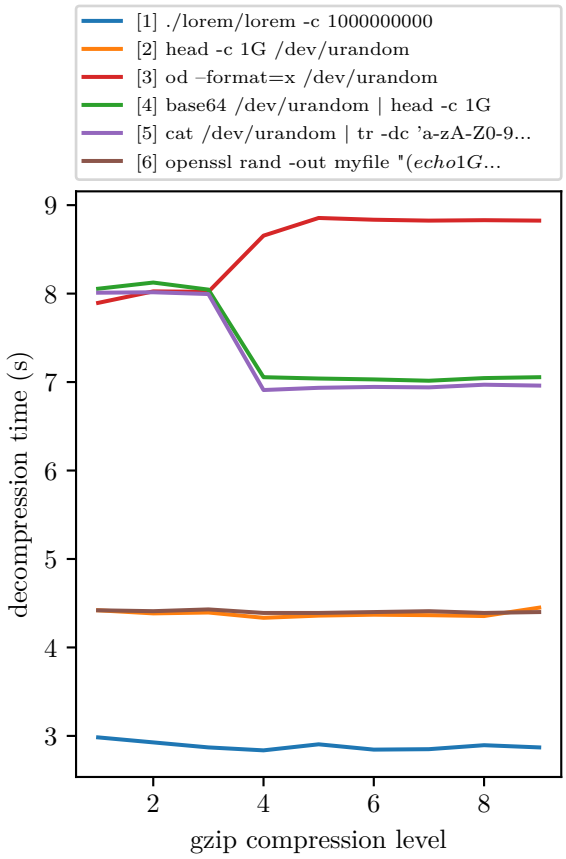

Figure 1: Decompression times for files generated with popular tools.

2.2. Popular tools

We first started by analyzing the compression and decompression times of files generated with some popular random data generators. We generated files with the following tools [5]:

- `od --format=x /dev/urandom`
- `base64 /dev/urandom | head -c 1G`
- `cat /dev/urandom | tr -dc 'a-zA-Z0-9' | head -c 1G`
- `openssl rand -out myfile "$( echo 1G | numfmt --from=iec )"`
- `lorem -c 1000000000`

We then compressed and decompressed these files with gzip, using the methods described above. The decompression times are shown in Figure 1.

We can see that the decompression times vary significantly depending on the tool used to generate the file. For example, the file generated by `lorem`, containing English text, is decompressed in 3 seconds. `openssl` and direct `/dev/urandom` output are decompressed in 4.5 seconds. Finally, the files generated by `base64` and `tr` require between 7 and 8 seconds to be decompressed, depending on the compression level used. Finally, `od`'s output is decompressed in 9 seconds.

2.3. `od`'s output

Wishing to understand why `od`'s output was decompressed slower than the others, we decided to fully leverage `od`'s various output formats. We thus generated files with the following output formats:

- `x1`, `x2`, `x4`, `x8` (hexadecimal)
- `a`, `c` (ASCII both named and unnamed)
- `d1`, `d2`, `d4`, `d8` (decimal)
- `f` (floating point)
- `o` (octal)
- `u1`, `u2`, `u4`, `u8` (unsigned decimal)

We repeated the same tests as before, and the results are shown in Figure 2, although with 100MB files instead of 1GB files, due to the long time required to generate the files.

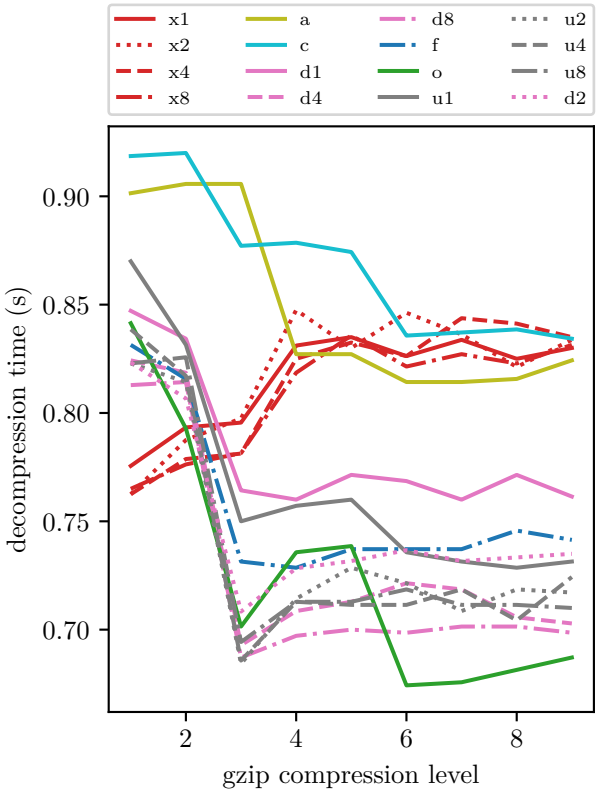

Figure 2: Decompression times for files generated with od 's various output formats.

We can see that the decompression times vary significantly depending on the output format chosen. For our nefarious purposes, the best output formats are `a` and `c` (ASCII named and unnamed), although `x2` slightly outperforms both at level 6, which is the default compression level.

2.4. Finding the optimal file

Finally, the graph in Figure 3 shows the decompression times for `od`'s best output formats along with the other tools.

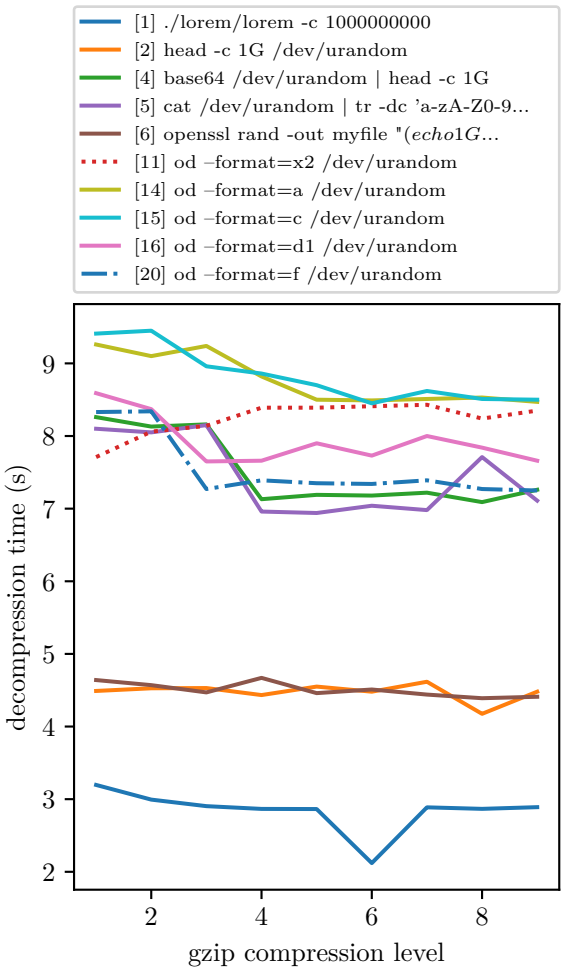

Figure 3: Decompression times for files generated with od 's best output formats and the other tools.

3. Bibliography

References

[1] GNU Gzip.
[2] Make image (layer) downloads faster by using pigz by sargun · Pull Request #35697 · moby/moby.
[3] Support parallel decompression (pigz) by mxpv · Pull Request #2640 · containerd/containerd.

[4] L. Peter Deutsch. DEFLATE Compressed Data Format Specification version 1.3. Request for Comments RFC 1951, Internet Engineering Task Force, May 1996.
[5] Per Erik Strandberg. Lorem, December 2022.