Comparison of UNIX Compression Tools

UBC STAT 404

Aidan Meharg - 20112579
Abhinav Kansal - 24054660

December 6, 2024

1 Abstract

File compression is an essential computing technique used in a variety of modern day applications - from data archival to real time messaging. The goal of compression is to reduce the amount of bits required to represent a file (or set of files). This might be done to allow for storing "more" information in a given amount of disk space, or to reduce the amount of data needed to transfer files across a network (e.g. downloading a large program from the internet).

There are several file compression tools available on modern UNIX computers which use a combination of different compression algorithms. In this report, we investigate the performance and tradeoffs between three common lossless compression tools.

2 Introduction

We have chosen to investigate the performance of three prevalent UNIX command line compression tools: **gzip**, **bzip2**, and **zstd**. All three use varying combinations of compression algorithms to achieve their results (outlined in Table 5). The performance indicators we are focusing on are compression time and compression ratio. Naturally, compression time is highly dependent on the size of the file, hence we have standardized all of our files to be exactly 3MB (that is, 3 x 1024 x 1024 bytes). Performance on different file types was also an area of interest, and representative files of several types were used in our experiment.

Further details on treatment and blocking factors, response variables, and general experimental design are given next in section 3. This is followed by a statistical analysis in section 4, with results summarized in section 5. Supporting figures, tables, and additional information can be found in the appendix (section 6).

3 Experimental Design

The main response of interest is time to compress (in seconds). This was measured via the UNIX `time` utility which measures the time for a process to complete. Real time (as opposed to system or user time) was measured to account for IO operations and give a more accurate estimate for real applications. Another response of interest was compression ratio (simply the old size of the file divided by the new size), however this value was not incorporated into our models as it is completely deterministic for a given file and tool (and hence assumptions of random errors do not make sense). A table of compression ratios is given in Table 4. Treatment and blocking factors are summarized in the table below:

Name	Type	Levels
Tool	Treatment (3 Levels)	gzip, bzip2, zstd (all at default compression settings)
File	Blocking (4 Levels)	minecraft_binary, shakespeare.txt, song.wav, textbook.pdf
Processor	Blocking (4 Levels)	x86_64 (MacBook Air 2020), ARM_64 (Macbook Pro 2023)

Table 1: Experimental Factors

The contents of the file types were chosen as follows:

- the binary file contains the executable launcher for the popular video game Minecraft (truncated to 3MB)
- the txt file contains the complete works of William Shakespeare (originally 5.2MB truncated to 3MB)
- the wav file contains a short recording of a chord progression on a piano
- the pdf file is the first 3MB of a textbook *Computer Vision: Algorithms and Applications* 2nd Edition by Richard Szeliski

We take care not to attempt any vast generalizations about file types given that it is not the file type that causes files to compress better, but the contents of the files themselves. For example, wav files of electronic music (often with many repeating sections) may compress much better than wav files of classical music. Since including file type as a treatment factor would require some strong assumptions about the generalizability of our chosen files, we have decided to treat it as a blocking factor and focus on the main effects of Tool.

We wrote a python script to carry out batch execution of this experiment. Due to the relatively "cheap" data collection process, we did not make any analytical compromises in order to reduce the number of runs required. A replicated 2 x 3 x 4 factorial design was carried out (taking 10 replications for each of the 24 combinations of M, T, F to give 240 runs total).

In each replication on a given Processor (block) k, the 12 combinations of Tool x File were randomly permuted and run in that order.

4 Statistical Analysis

4.1 Model Checking & Improvement

We originally planned to model the compression time in seconds of tool i on file j and Processor k as:

$$Y_{ijk} = \mu + \alpha_i + \beta_j + \gamma_k + (\alpha\beta)_{ij} + E_{ijk}$$

$$(i = 1, 2, 3;\ j = 1, 2, 3, 4;\ k = 1, 2)$$

where: μ is an overall effect, α_i is an effect on the mean response from compression tool i, β_j is a blocking effect on the mean response from file type j, and γ_k is a blocking effect from the Processor used for compression. $(\alpha\beta)_{ij}$ is an interaction effect between tool i and file j, and $E_{111}, ..., E_{342}$ are random error terms assumed to be independent $N(0, \sigma^2)$ random variables.

After inspecting the residuals from this model for violations of our assumptions of normality and homoscedasticity (see Normal QQ-plot in Figure 2 and plot of residuals in Figure 1) it was clear this model did not meet our error assumptions. The Box-Cox analysis was also suggesting an inconvenient transformation (Figure 3) with the 95% CI for λ roughly centered around 0.2. Upon inspection of Figure 4, it became clear that modeling the blocking factor Processor as a purely additive factor was a naïve approach. For example: zstd performed only moderately faster on the ARM64 processor, however bzip2 performed much faster on the ARM64 than the x86. This suggested the addition of an interaction term between Processor and Tool.

The Box-Cox plot of a new model with a Processor x Tool interaction (Figure 5) weakly suggested applying a square root transformation to the response (with the 95% CI for λ approximately covering 0.5).

Applying the square root transformation, we proceed with the following model:

$$\sqrt{Y_{ijk}} = \mu + \alpha_i + \beta_j + \gamma_k + (\alpha\beta)_{ij} + (\alpha\gamma)_{ik} + E_{ijk}$$

$$(i = 1, 2, 3;\ j = 1, 2, 3, 4;\ k = 1, 2)$$

where $(\alpha\gamma)_{ik}$ is an interaction effect between Processor k and tool i, and all other parameters are as described above.

The plot of residuals (Figure 6), and normal QQ plot (Figure 7) align much more closely with our error assumptions. Although we notice some outlier values, we do not consider these outliers to warrant a complete violation of our error assumptions.

4.2 ANOVA

Source	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Processor	1	0.545	0.5453	847.92	$< 2 \times 10^{-16}$ ***
File	3	0.044	0.0146	22.75	6.7×10^{-13} ***
Tool	2	5.076	2.5380	3946.69	$< 2 \times 10^{-16}$ ***
Processor:Tool	2	0.095	0.0476	74.01	$< 2 \times 10^{-16}$ ***
File:Tool	6	0.307	0.0511	79.48	$< 2 \times 10^{-16}$ ***
Residuals	225	0.145	0.0006		
*Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1*					

Table 2: ANOVA Results

Our ANOVA results (Table 2) indicate that all treatment, blocking, and interaction effects are significant at the $\alpha = 0.01$ level. In terms of practical significance, our treatment factor Tool has the strongest effect based on relative sum of squares magnitudes.

Table 3 lists all parameter estimates and 95% confidence intervals. Overall, our baseline tool bzip2 was shown to be the slowest of the 3 compression tools. gzip was estimated to have an overall compression time 0.114 seconds$^{0.5}$ (square root seconds) faster than bzip2. zstd was the fastest overall, achieving an overall estimated speedup of 0.279 seconds$^{0.5}$ in comparison to bzip2. These results are visually confirmed in Figure 4.

5 Discussion

In terms of original units (as sqrt seconds are not very intuitive), bzip2 took an average of 0.252 seconds to compress a 3MB file (across all files and processors), gzip took 0.108 seconds, and zstd took 0.0214 seconds overall. The differences in compression time are of a practically significant magnitude, with zstd averaging over 10x faster compression times in comparison to bzip2.

Table 4 provides information on the compression ratios achieved by these 3 tools. Somewhat surprisingly, zstd (our fastest tool) also achieved better compression ratios than both bzip2 and gzip with the exception of the wav file. The 3 tools achieved similar compression ratios on the wav (audio) file, however bzip2 performed slightly better than gzip and zstd.

An interaction plot of the File x Tool interaction is given in Figure 8. Once again, we do not claim that these files are representative of all files in their type, but inspect this interaction out of curiosity and perhaps motivations for another study. bzip2 appeared to work in opposite directions from the other 2 tools, working fastest on the text file and slowest on the wav and pdf files. The distinct behavior of bzip2 with different file types, compared to gzip and zstd, may be due to the fact that it is the only tool among the three that does not rely on the LZ77 compression algorithm as part of its core compression strategy (see Table 5).

6 Appendix

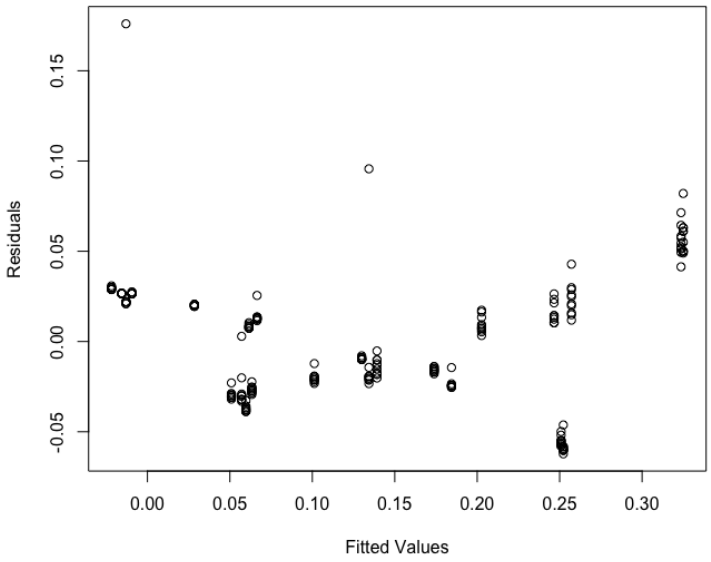

Figure 1: Residuals vs fitted values for original model (without Processor x Tool interaction)

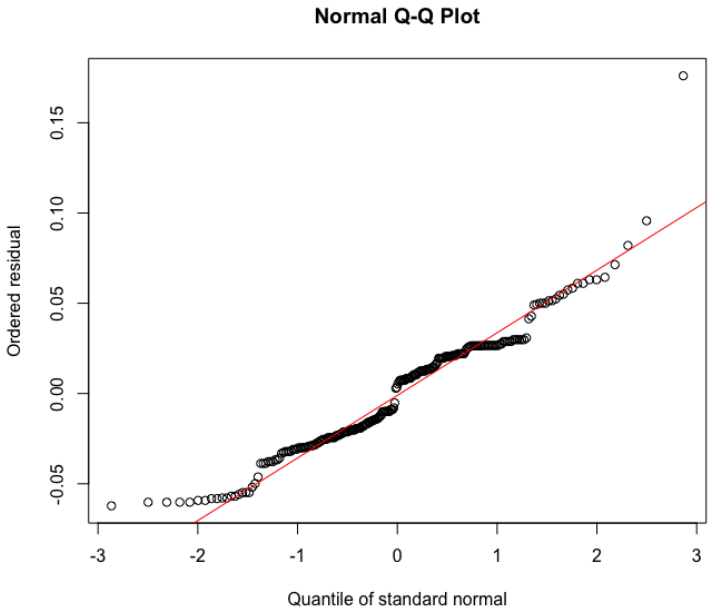

Figure 2: QQ plot for original model (without Processor x Tool interaction)

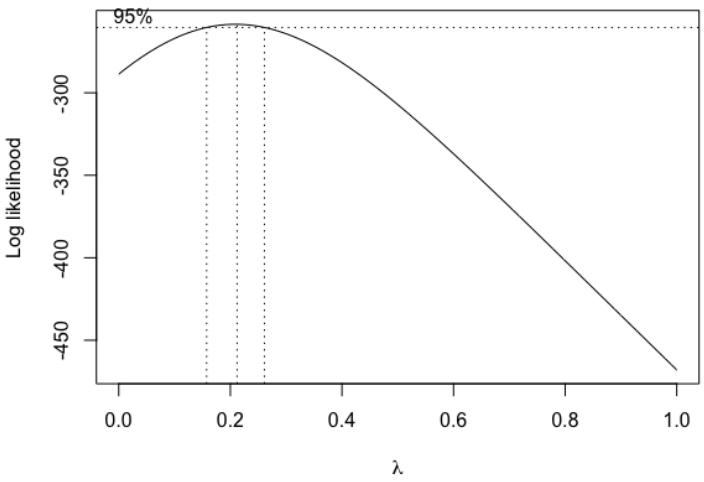

Figure 3: Box-Cox plot for original model

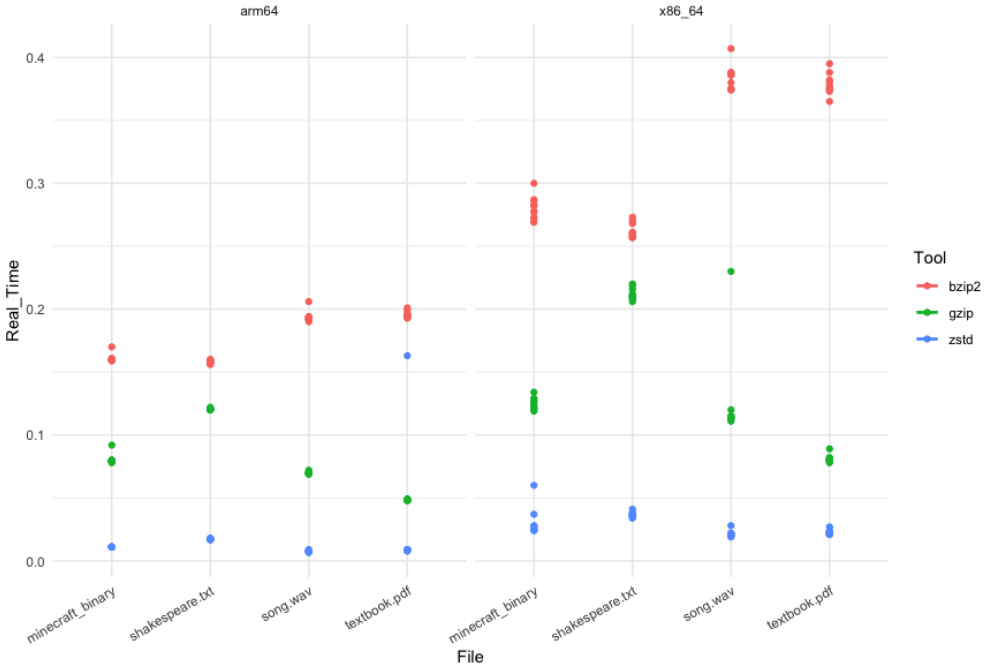

Figure 4: Full dataset visualized (split by Processor)

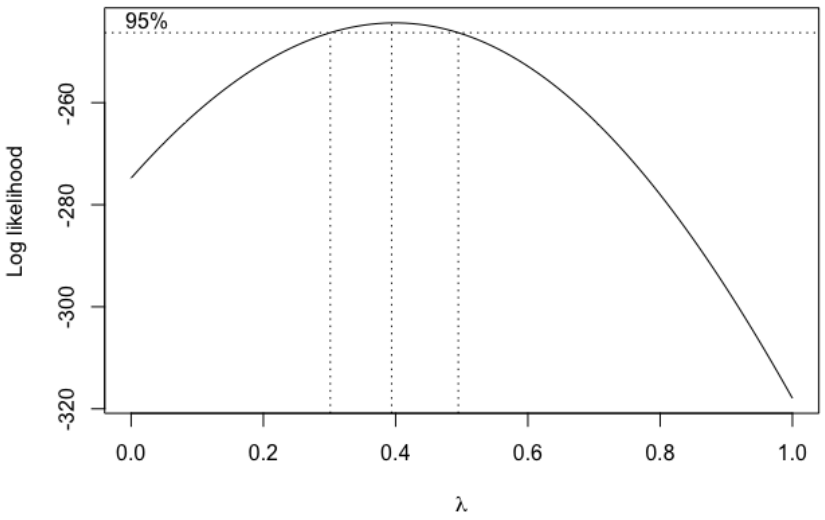

Figure 5: Box-Cox plot for model with Processor x Tool interaction

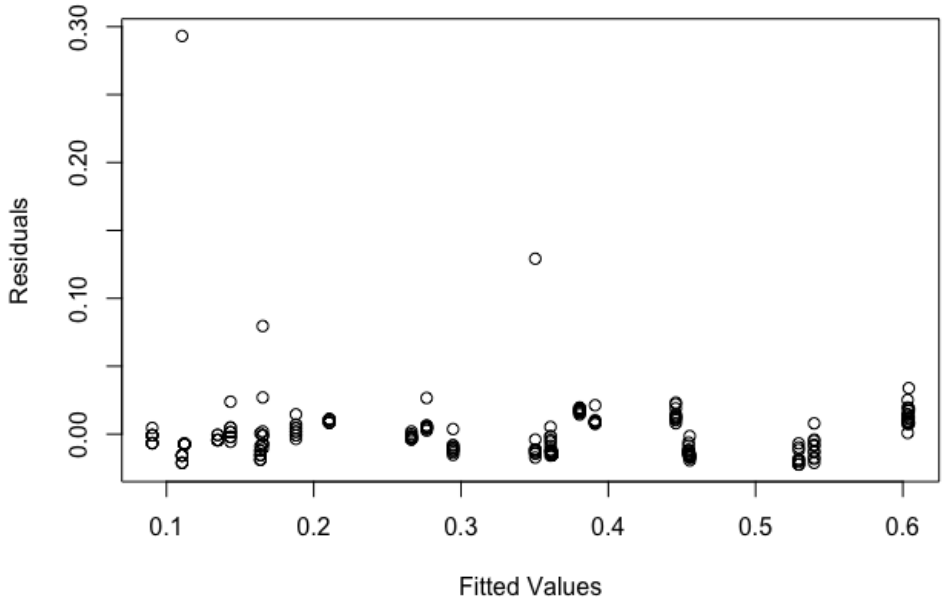

Figure 6: Residuals vs fitted values for transformed model

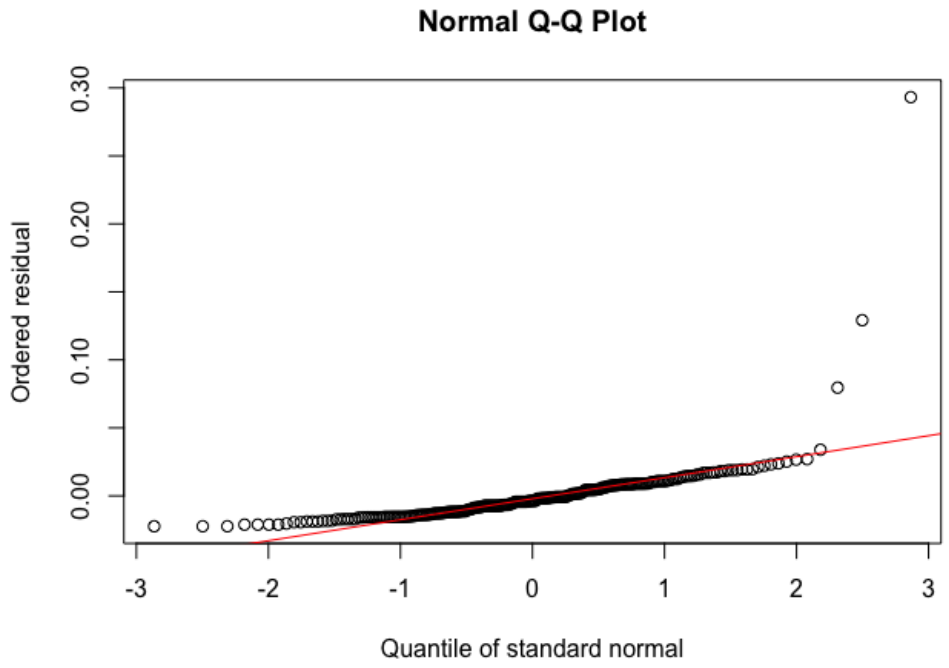

Figure 7: Normal QQ-plot for transformed model

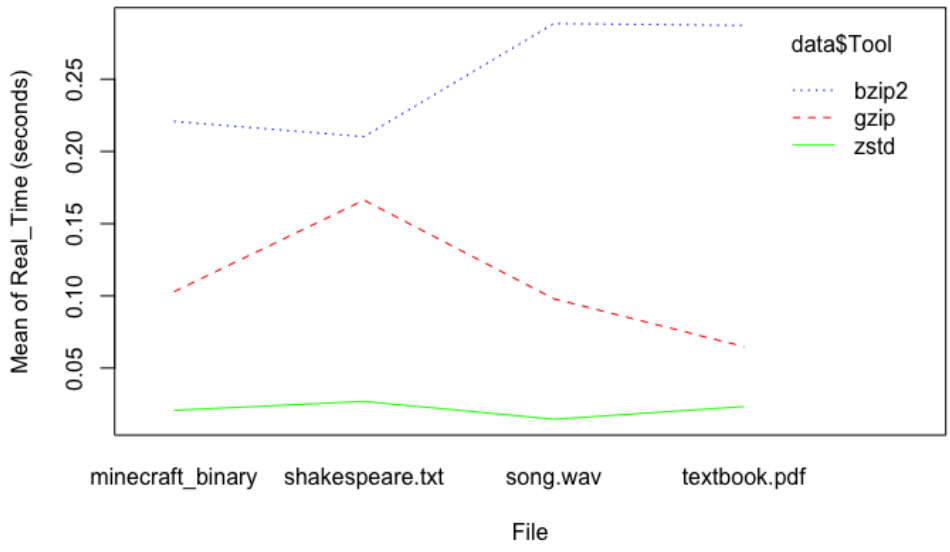

Figure 8: Interaction plot for (File x Tool) interaction

Term	**95% CI**
(Intercept)	0.391 ± 0.012
Processorx86_64	0.149 ± 0.011
Fileshakespeare.txt	-0.010 ± 0.016
Filesong.wav	0.064 ± 0.016
Filetextbook.pdf	0.063 ± 0.016
Toolgzip	-0.114 ± 0.018
Toolzstd	-0.279 ± 0.018
Processorx86_64:Toolgzip	-0.065 ± 0.016
Processorx86_64:Toolzstd	-0.096 ± 0.016
Fileshakespeare.txt:Toolgzip	0.096 ± 0.022
Filesong.wav:Toolgzip	-0.075 ± 0.022
Filetextbook.pdf:Toolgzip	-0.130 ± 0.022
Fileshakespeare.txt:Toolzstd	0.033 ± 0.022
Filesong.wav:Toolzstd	-0.086 ± 0.022
Filetextbook.pdf:Toolzstd	-0.065 ± 0.022

Table 3: 95% Confidence intervals of parameter estimates

File	**bzip2**	**gzip**	**zstd**
minecraft_binary	3.547880	3.383536	3.497095
shakespeare.txt	3.656340	2.682340	2.768776
song.wav	1.149453	1.084999	1.080648
textbook.pdf	1.630648	1.590124	1.714739

Table 4: Compression Ratios by Tool and File Type

Tool	Core Compression Strategy
gzip	LZ77 + Huffman Coding
bzip2	Burrows-Wheeler Transform + Huffman Coding
zstd	Finite State Entropy + Dictionary Compression + LZ77

Table 5: Summary of Compression Strategies for Chosen Tools