Comparison of lossless data compression methods

Preprint · March 2015

CITATIONS
0

READS
315

5 authors, including:

Ulrich Göhner
Hochschule für angewandte Wissenschaften Kempten
61 PUBLICATIONS **22** CITATIONS

SEE PROFILE

Technical Reports in Computing Science No. CS-07-2015

Comparison of lossless data compression methods

Dominic Berz
Marco Engstler
Moritz Heindl
Florian Waibel

March 2015

Hochschule Kempten
University of Applied Sciences
Fakultät Informatik

Comparison of lossless data compression methods

Dominic Berz, Marco Engstler, Moritz Heindl, Florian Waibel

Abstract— **When storing data in backup systems or transferring large files over data networks the use of data compression methods are often considered. A problem, that might arise, is the decision which compression method is suitable for the given situation.**

The aim of this paper is to find the best lossless compression method in terms of compression speed and ratio. To achieve this goal, the functionality of the four compression tools compress[1], gzip[2], bzip2[3] and xz[4] are compared with each other. The comparison is done by compressing and decompressing different files, from a Portable Network Graphics (PNG) with 3 MB to an ISO image with more than 600 MB.

Based on the measured results, the pros and cons of each tool are discussed, taking their compression and transformation algorithms[5][6][7][8] into account.

As conclusion, criteria for the selection of lossless compression tools are found, giving suggestions which compression tool is better suited in which situation.

I. Introduction

DATA compression tools are designed to reduce the size of data, so that it requires less storage space and less bandwidth to be transmitted on a data communication channel.

In the different data compression tools a variety of compression algorithms are used, including Lempel-Ziv-Welch, Lempel-Ziv-Markov chain, DEFLATE and Burrows-Wheeler. Data compression algorithms are generally classified into either lossless or lossy.

Lossless data compression involves a transformation of the representation of the original data set in a way, that it is possible to reproduce exactly the original data set by performing a decompression transformation. Lossless compression is used in compressing text files, executable codes, word processing files, database files, tabulation files, and whenever it is important that the original and the decompressed files must be identical. Thus, lossless data compression algorithms find numerous applications in information technology. For example, packing utilities in Windows, Linux, and Unix operating systems. [9]

The question that arises when using lossless compression tools, is which tools and with that also which algorithm to choose. Therefore this paper will compare four such tools, namely compress, gzip, xz and bzip2 and point out differences in terms of data compression speed and ratio.

II. Evaluated compression methods

A. compress

1) description: [1] Compress is a Unix shell lossless compression program based on the Lempel-Ziv-Welch (LZW) compression algorithm. Compared to more modern compression utilities - for example gzip, bzip2 and xz - compress performs faster and with less memory usage at the cost of a significantly lower compression rate.

Compress and it's decompression component uncompress can be used to compress and decompress single files or file archives (e. g. TAR). Files compressed by compress are given the ".Z" extension.

2) compress: [1] Spencer Thomas of the University of Utah implemented compress in 1984, after the modified Lempel-Ziv algorithm was publicized in [10]. Compress has since fallen out of favor in particular user-groups because it makes use of the LZW algorithm, which was covered by a Unisys patent. Compress has however, maintained a presence on Unix and BSD systems.

3) LZW algorithm: LZW is an universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. It was published by Welch in 1984 as an improved implementation of the LZ78 algorithm created by Lempel and Ziv in 1978. The algorithm is simple to implement and has got the potential for very high throughput in hardware implementations. [1] It was the algorithm of the widely used Unix file compression utility compress, and is still used in the GIF image format.

4) usage: LZW compression became the first widely used universal data compression method on computers. A large text file can typically be compressed via LZW to about half its original size.

[10] LZW was widely spread when it became part of the GIF image format in 1987. It may even be used in TIFF and PDF files.

5) compression: [6] [8] [10]

Compressing a file with compress:
compress -c */path/to/input* > */path/to/output*.Z

A dictionary is initialized to contain the single-character strings corresponding to all the possible input characters. The algorithm works by scanning through the input string for successively longer substrings until it finds one that is not in the dictionary. When such a string is found, the index for the string, without the last character (i.e., the longest substring that is in the dictionary), is retrieved from the dictionary and sent to output, and the new string is added to the dictionary with the next available code. The last input character is then used as the next starting point to scan for substrings. In this way, successively longer strings are registered in the dictionary and made available for subsequent encoding as single output values. The algorithm works best on data with repeated patterns, so the initial parts of a message will see little compression.

Functionality:
1) Initialize the dictionary to contain all strings of length one.
2) Find the longest string 'W' in the dictionary that matches the current input.
3) Emit the dictionary index for 'W' to output and remove W from the input.
4) Add W followed by the next symbol in the input to the dictionary.
5) Go to Step 2.

6) uncompression: [8] [10]

Decompressing a file with uncompress:
uncompress */path/to/input*

Files can be returned to their original state using uncompress. The usual action of uncompress is not merely to create an uncompressed copy of the file, but also to restore the timestamp and other attributes of the compressed file. The decoding algorithm works by reading a value from the encoded input and outputting the corresponding string from the initialized dictionary. In order to rebuild the dictionary in the same way as it was built during encoding, it also obtains the next value from the input and adds to the dictionary the concatenation of the current string and the first character of the string obtained by decoding the next input value, or the first character of the string just output if the next value can not be decoded. The decoder then proceeds to the next input value and repeats the process until there is no more input, at which point the final input value is decoded without any more additions to the dictionary. In this way the decoder builds up a dictionary which is identical to that used by the encoder, and uses it to decode subsequent input values. Thus the full dictionary does not need be sent with the encoded data; just the initial dictionary containing the single-character strings is sufficient.

7) error tolerance: "A characteristic of most reversible compression systems is that if one bit is altered in the compressed image, much of the message is garbled. Therefore, a given error rate may be acceptable for the transmission of uncompressed data, but the same rate may be unacceptable for compressed data. Special efforts to detect errors are needed, especially to catch any errors in the compression and decompression processes. ... This intolerance to errors often limits the use of compression. For example, digitized speech can tolerate occasional bit losses, but compression turns these errors into serious losses. On the other hand, since most commercial data storage devices, such as disk and tape, have acceptably low error rates and there is no attempt to use their data with errors present, compression works well in these circumstances." [10]

8) parallelization: [11] The LZW-algorithm used in compress is the most well known sequential compression technique and was not yet implemented in a parallel manner, but there are dissertation that theorize and test such implementations. For example in [11] is claimed that a coarse grain parallelization method on LZW compression technique can improve the existing sequential algorithms.

9) performance: Some Files are not compressed by compress, as can be seen in table II-A9, as their

TABLE I
COMPRESS - COMPRESSION RATIO AND SPEED.

File	compression		uncompression
	ratio	time	time
File A	2,500	13,471s	5,074s
File B	fail		
File C	2,498	4,472s	1,742s
File D	fail		
File E	fail		
File F	fail		
File G	fail		
File H	1,454	0,297s	0,163s
File I	fail		
File J	fail		

compressed file size would have been greater than in the uncompressed state. This condition leads to the conclusion that the LZW algorithm can effectively be used only on files which are available in text format. Such text files are compressed to less than half their original file size.

B. *gzip*

[2][7][12] gzip is a compression and decompression software application and file format, created by Jean-Loup Gailly and Mark Adler. Their intent was to have a free replacement for the compression program compress, which used the patent protected LZW algorithm. It was therefore integrated into the GNU Project and put under the GNU Public Licence.

gzip uses a combination of the LZ77[8] and Huffman coding[13], which makes up the DEFLATE algorithm.

1) File format: The gzip file format consists of:

- a 10-byte header, containing a magic number (1f 8b), a version number and a timestamp
- optional extra headers, such as the original file name
- a body, containing a DEFLATE-compressed payload
- an 8-byte footer, containing a CRC-32 checksum and the length of the original uncompressed data

This format is only used for single file compression. To achive compression of multiple files, these files first need to be put into a single uncompressed tar archive, basically turning them into one file. This can then be compressed by gzip again. The resulting file is then called a tarball with the .tar.gz or .tgz extension.

Compared to gzip the ZIP archive format, which also uses DEFLATE, does not need this external archiver. It compresses files individually. However, this makes it less compact, because of the missing advantage of redundancy between files.

2) DEFLATE algorithm: A compressed data set consists of a series of blocks of input data. The block size is optional, except for non-compressible blocks with a limit of 65535 bytes. Each block consists of a pair of Huffman code trees and the compressed data.

The DEFLATE algorithm uses the LZ77 algorithm to find duplicated strings within the previous 32K input bytes and add pointers of string lenght and the backward distance to these matches. The compressed data along with the non-duplicated strings are then represented in a Huffman tree for the literals and lenghts and another for the distances. The two trees are themselves compressed through Huffman coding.

Because of the seperation into blocks and seperate compression for each, the data compression can be spread over multiple threads. Gzip doesn't support multi-threading, there is however a parallel implementation of gzip called pigz[14].

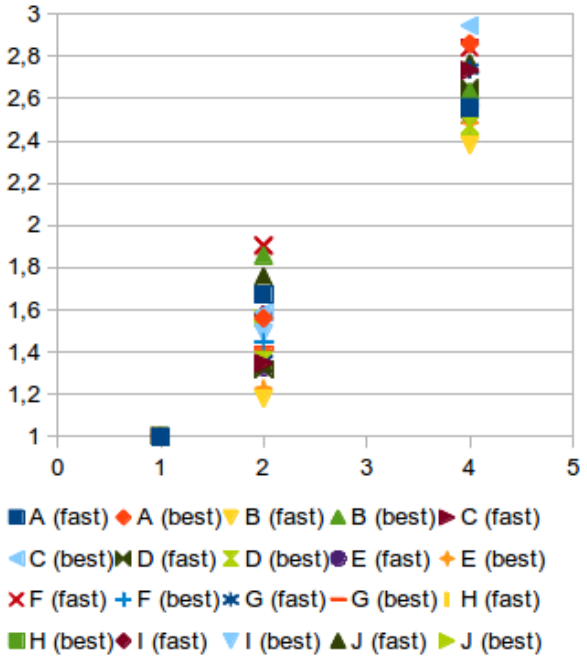

Fig. 1. gzip - Speedup per file using one, two or four threads.

Figure 1 shows the speedup for each file in the test using pigz. The result is a linear speedup of the compression time.

3) Performance options: The usage of the gzip command is:

```
gzip [OPTION]... [FILE]...
```

As performance option the numbers 1 - 9, in single steps, can be added with 1 being the fastest compression mode and 9 the mode with the highest compression ratio.

4) Tested command line options: For the performance test of the gzip program, the files from section III where compressed and decompressed with the limits of the performance option, meaning fastest and best compression.

The commands used for the test:

```
Fastest compression mode:
gzip -1 /path/to/input
Best compression mode:
gzip -9 /path/to/input
```

5) Test results: In tables II-B5 and II-B5 the measured metrics of gzip are listed for each file described in III-B.

The measured values indicate the following characteristics:

- The compression ratio is best for text files and text file archives.
- For multimedia files (compressed and uncompressed) compression ratio is very poor.
- The time for decompression is not dependant on the compression mode.

TABLE II
GZIP - COMPRESSION RATIO FOR FASTEST MODE AND BEST MODE.

File	compression ratio	
	fast mode	best mode
File A	3,783	4,750
File B	3,155	3,718
File C	2,153	2,328
File D	1,010	1,011
File E	1,004	1,005
File F	1,011	1,012
File G	1,022	1,019
File H	1,147	1,110
File I	1,049	1,052
File J	1,000	1,000

TABLE III
GZIP - AVERAGE COMPRESSION SPEED AND AVERAGE DECOMPRESSION SPEED FOR FASTEST MODE AND BEST MODE.

File	compression speed in s		decompression speed in s	
	fast mode	best mode	fast mode	best mode
File A	10,784	44,795	5,392	4,787
File B	3,910	10,233	1,929	1,773
File C	0,370	1,047	0,143	0,150
File D	0,493	0,563	0,133	0,150
File E	2,157	2,367	0,550	0,543
File F	6,473	7,097	1,567	1,567
File G	1,603	1,737	0,423	0,387
File H	0,170	0,270	0,083	0,060
File I	28,540	32,775	6,402	6,178
File J	51,437	54,840	10,270	10,513

- The time for decompression is dependant on the compression ratio.

C. xz

1) Description: xz is a lossless data compression program and file format which incorporates the LZMA/LZMA2 compression algorithms.

The Lempel-Ziv-Markov chain algorithm (LZMA) is an algorithm used to perform lossless data compression.

LZMA uses a dictionary compression algorithm (a variant of LZ77 with huge dictionary sizes and special support for repeatedly used match distances), whose output is then encoded with a range encoder, using a complex model to make a probability prediction of each bit. The dictionary compressor finds matches using sophisticated dictionary data structures, and produces a stream of literal symbols and phrase references, which is encoded one bit at a time by the range encoder: many encodings are possible, and a dynamic programming algorithm is used to select an optimal one under certain approximations.

Prior to LZMA, most encoder models were purely byte-based (coded each bit using only a cascade of contexts to represent the dependencies on previous bits from the same byte). The main innovation of LZMA is that instead of a generic byte-based model, LZMA's model uses contexts specific to the bitfields in each representation of a literal or phrase: this is nearly as simple as a generic byte-based model, but gives much better compression because it avoids mixing unrelated bits together in the same context. Furthermore, compared to classic dictionary compression, the dictionary sizes are much larger, taking

advantage of the large amount of memory available on modern systems.

The LZMA2 container supports multiple runs of compressed LZMA data and uncompressed data. Each LZMA compressed run can have a different LZMA configuration and dictionary. This improves the compression of partially or completely incompressible files and allows multithreaded compression and multithreaded decompression by breaking the file into runs that can be compressed or decompressed independently in parallel.

xz is a general-purpose data compression tool with command line syntax similar to *gzip* and *bzip2*. The native file format is the .xz format, but the legacy .lzma format used by LZMA Utils and raw compressed streams with no container format headers are also supported.

xz compresses or decompresses each file according to the selected operation mode. If no files are given or file is -, xz reads from standard input and writes the processed data to standard output. xz will refuse (display an error and skip the file) to write compressed data to standard output if it is a terminal. Similarly, xz will refuse to read compressed data from standard input if it is a terminal.

After successfully compressing or decompressing the file, xz copies the owner, group, permissions, access time, and modification time from the source file to the target file. If copying the group fails, the permissions are modified so that the target file doesn't become accessible to users who didn't have permission to access the source file. xz doesn't support copying other metadata like access control lists or extended attributes yet.

2) Memory usage: The memory usage of xz varies from a few hundred kilobytes to several gigabytes depending on the compression settings. The settings used when compressing a file determine the memory requirements of the decompressor. Typically the decompressor needs 5% to 20% of the amount of memory that the compressor needed when creating the file. For example, decompressing a file created with xz -9 currently requires 65 MB of memory. Still, it is possible to have .xz files that require several gigabytes of memory to decompress.

Especially users of older systems may find the possibility of very large memory usage annoying. To prevent uncomfortable surprises, xz has a built-in memory usage limiter, which is disabled by default.

3) XZ Utils: XZ Utils (previously LZMA Utils) is a set of free command-line lossless data compressors, including LZMA and xz, for Unix-like operating systems and, from version 5.0 onwards, Microsoft Windows.

XZ Utils consists of two major components:

- xz, the command-line compressor and decompressor (analogous to gzip)
- liblzma, a software library with an API similar to zlib

XZ Utils can compress and decompress both the xz and lzma file formats, but since the LZMA format is now legacy, XZ Utils compresses by default to xz.

4) Implementation: Both the behavior of the software as well as the properties of the file format have been designed to work similarly to those of the popular Unix compressing tools gzip and bzip2. It consists of a Unix port of Igor Pavlov's LZMA-SDK that has been adapted to fit seamlessly into Unix environments and their usual structure and behavior.

Just like gzip and bzip, xz and lzma can only compress single files as input. They cannot bundle multiple files into a single archive, to do this an archiving program is used first, such as tar.

Compressing an archive:

xz testarchive.tar results in *testarchive.tar.xz*

Decompressing the archive:

unxz testarchive.tar.xz results in *testarchive.tar*

5) Used Options:

- *-k, –keep* Don't delete the input files
- *-f, –force* If the target file already exists, delete it before compressing or decompressing
- *-0 to -9* Select a compression preset level. The default is -6.
  - -0 to -3 These are somewhat fast presets.
  - -4 to -6 Good to very good compression while keeping decompressor memory usage reasonable even for old systems.
  - -7 to -9 These are like *-6* but with higher compressor and decompressor memory requirements.

TABLE IV
XZ - COMPRESSION RATIO FOR FASTEST MODE AND BEST MODE.

File	compression ratio	
	fast mode	best mode
File A	5,53	7,2
File B	1,07	1,10
File C	4,45	6,05
File D	1,02	1,04
File E	1,19	1,21
File F	1,01	1,01
File G	1,00	1,00
File H	2,53	2,69
File I	1,00	1,00
File J	1,02	1,02

TABLE V
XZ - AVERAGE COMPRESSION SPEED AND AVERAGE DECOMPRESSION SPEED FOR FASTEST MODE AND BEST MODE.

File	compression speed in s		decompression speed in s	
	fast mode	best mode	fast mode	best mode
File A	43,883	396,780	10,110	8,247
File B	17,120	147,260	3,797	2,907
File C	2,127	5,877	0,457	0,420
File D	3,773	3,390	0,503	0,697
File E	17,850	18,903	0,753	1,420
File F	52,920	69,347	6,977	8,487
File G	12,943	13,680	2,730	2,603
File H	1,117	1,403	0,263	0,257
File I	247,953	343,850	12,033	12,753
File J	455,030	630,887	3,77	3,963

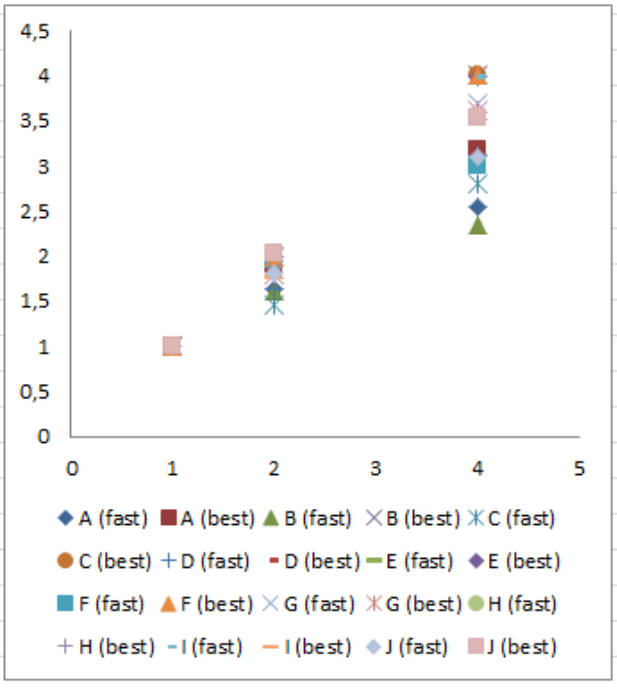

Fig. 2. xz - Speedup per file using one, two or four threads.

By using the xz a linear speedup of the runtime can be achieved. Figure 2 shows the speedup per file.

D. *bzip2*

Another widely used compression program is the bzip2 program. bzip2 was mainly developed by Julian Seward and is available for the operating systems Linux, Unix and Windows. The source code of bzip2 is released under a BSD-like license and non of the used algorithms are covered by software patents.[3]

bzip2 can be used to compress single files or file archives (e. g. TAR). A file compressed by bzip2 is usually recognizable by the file name extension .bz2.

1) Algorithms: [5][13] bzip2 uses two major algorithms to compress data streams. The first one is the Burrows-Wheeler sorting algorithm, which is used to group together identical characters inside the input stream. After this the compression is actualy done by applying the Huffman coding on the previously sorted input stream.

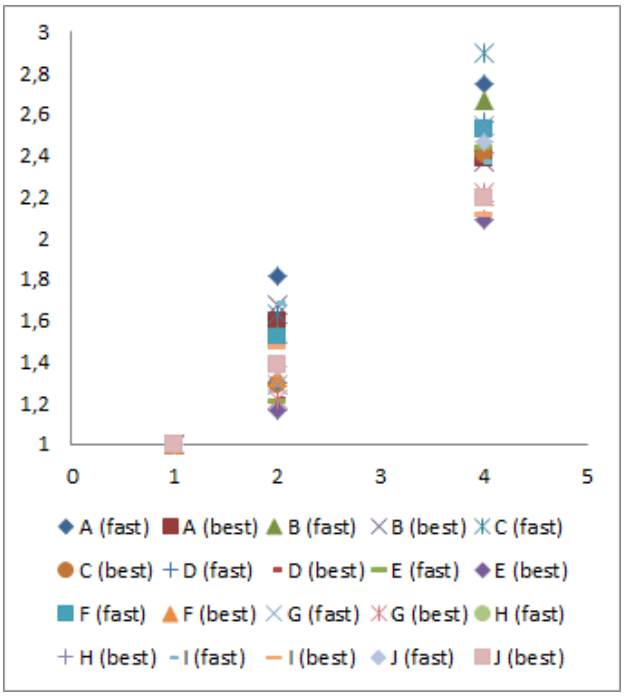

Fig. 3. bzip2 - Speedup per file using one, two or four threads.

Both the Burrows-Wheeler sorting and the Huffman coding are done block-wise by splitting the input stream into multiple, equal sized data chunks. With this in mind, bzip2's compression speed can be increased by spreading the compression onto multiple threads. Though bzip2 itself doesn't support

multi-threading. For multi-threading, an implementation variant called pbzip2[15] (Parallel bzip2) is available.

By using the pbzip2 a linear speedup of the runtime can be achieved. Figure 3 shows the speedup per file.

2) Performance options: [3] Both compression ratio and runtime memory usage are affected only by the block size used. bzip2 supports block sizes from 100000 bytes to 900000 bytes in steps of 100000 bytes. The compression speed is independent from the block size.

If x is the block size in byte, the amount of memory consumed by bzip2 at runtime can be calculated as shown in II-D2.

Fig. 4. Block size dependent runtime memory usage in compression mode and decompression mode.

$$400000byte + (8 * x)$$

$$100000byte + (4 * x)$$

3) Tested command line options: The files described in section III, where tested by compressing and decompressing each with the minimum block size and the maximum block size. The compression block size is adjusted by appending -x to the command line options, where x is an element from 1, 2, ..., 9.

These are the to command lines used:

Fastest compression mode:
bzip2 -1 */path/to/input*
Best compression mode:
bzip2 -9 */path/to/input*

4) Test results: In tables II-D4 and II-D4 the measured metrics of bzip2 are listed for each file described in III-B.

The measured values indicate the following characteristics:

- The compression ratio is best for text files and text file archives.
- For multimedia files (compressed and uncompressed) compression ratio is poor.
- Large block size results in imperceptible slower compression speed.

TABLE VI
BZIP2 - COMPRESSION RATIO FOR FASTEST MODE AND BEST MODE.

File	compression ratio	
	fast mode	best mode
File A	6,013	5,041
File B	3,912	4,935
File C	2,342	2,420
File D	1,005	1,013
File E	0,998	1,001
File F	1,003	1,008
File G	1,029	1,051
File H	1,144	1,201
File I	1,044	1,053
File J	0,993	0,996

TABLE VII
BZIP2 - AVERAGE COMPRESSION SPEED AND AVERAGE DECOMPRESSION SPEED FOR FASTEST MODE AND BEST MODE.

File	compression speed in s		decompression speed in s	
	fast mode	best mode	fast mode	best mode
File A	64,838	69,402	18,722	21,681
File B	23,035	25,767	7,553	8,916
File C	1,813	1,870	0,667	0,847
File D	2,043	2,090	0,843	1,113
File E	9,270	9,483	3,913	4,727
File F	27,263	27,993	11,483	14,683
File G	5,577	5,610	2,703	3,477
File H	0,633	0,610	0,310	0,357
File I	137,603	140,591	54,300	73,117
File J	245,447	250,627	100,597	127,130

- Large block size results in slightly slower decompression speed.

III. EXPERIMENTAL SET-UP

To get the data needed to compare the previously mentioned compression tools, an experimental set-up was created.

A. Test environment

This includes the following conditions:

- All tests are executed on the same machine (4 x 2.40 GHz).
- All tools are tested in their fastest compression mode.
- All tools are tested in their best compression mode.
- Each tool and mode is tested three times.
- The Unix tool *time* is used to measure the processor time used.

B. Dummy data

The data used for the compression tests was selected in terms of comprehensibility, so all files are available for free. Also different file formats and types where chosen to cover a wide variety of commonly used data types.

1) Text files:

Linux kernel 3.17.3 source code (File A)
: The source tarball of the Linux kernel version 3.17.3.[16]

Wikinews Dump 2014-11-19 (File B)
: A Dump of Wikinews from 2014-11-29.[17]

Docker 1.4.0 source code (File C)
: The source tarball of the Docker software version 1.4.0.[18]

2) Sound files:

Cannnncion sound (File D)
: An OGG sound file.[19]

Synthesizer sound (File E)
: An 14 seconds synthesizer generated OGG sound file.[20]

Sintel master sound (File F)
: The FLAC encoded master sound of the Sintel movie.[21]

3) Image files:

Panorama Auenfeld-Hochtannber (File G)
: A 360 degree JPEG-Panorama.[22]

Wealth Gini Map 2013 (File H)
: A PNG of the World map visualizing the Wealth Gini Index.[23]

4) Other files:

Debian 7.7.0 ISO (File I)
: The first Disc Image (ISO) for Debian 7.7.0 i386.[24]

Sintel movie (File J)
: The 1080p MKV version of the Sintel movie.[25]

IV. Results

The experimental results already mentioned for each tool in the previous sections, are now going to be compared with each other.

To illustrate the differences between the tools, the results are visualized in three bar charts, side by side. The first chart (Figure 5) describes the achieved compression ratio for each file. There are also charts for compression runtime (Figure 6) and decompression runtime (Figure 7).

The most striking characteristic when evaluating the compression ratio, is the file type dependancy. Only for text files, there are significant differences. For all other files, the differences are not only small, but there is marginally any space saving achieved by compression. Especially compress fails to process the non-text files. On the other hand, a precedence order referring to the ratio can be created.

- compress is always worse than the other tools
- gzip is always better then compress but less effective than bzip2 and xz
- the best mode of xz always leads to the best result
- the fast mode of xz and the fast mode and best mode of bzip2 are not generally classifiable

When speaking of marginal space saving for non-text files, this must be seen in relation to the compression achieved for text files, where the compressed file size is up to seven times smaller. For the image files a space saving of up to 18 % was achieved. This means that the space saving of non-text file compression is primarily significant for very large amounts of data.

The high compression ratio of xz's best mode is achieved at the expense of a long runtime, which is distinctly higher than for all other tested tools. Generally the runtime order is inverse to the ratio order with one exception: compress is always worse then gzip. The differences are much higher than for compression ratio.

The differences for decompression runtime are similar to the compression runtime, but the orders of xz and bzip2 are reversed.

V. Summary

After evaluating the results, several conclusions can be drawn. As the different options that can be applied to the tools affect the results, following recommendations should be seen as general guidance.

compress is the one tool that was identified as the lowest performing. While it is fast in compressing data, it lacks decompression speed. Also the compression ratio is the worst of all compared tools and there is no multithreading support. However all of this isn't relevant for non text file compression,

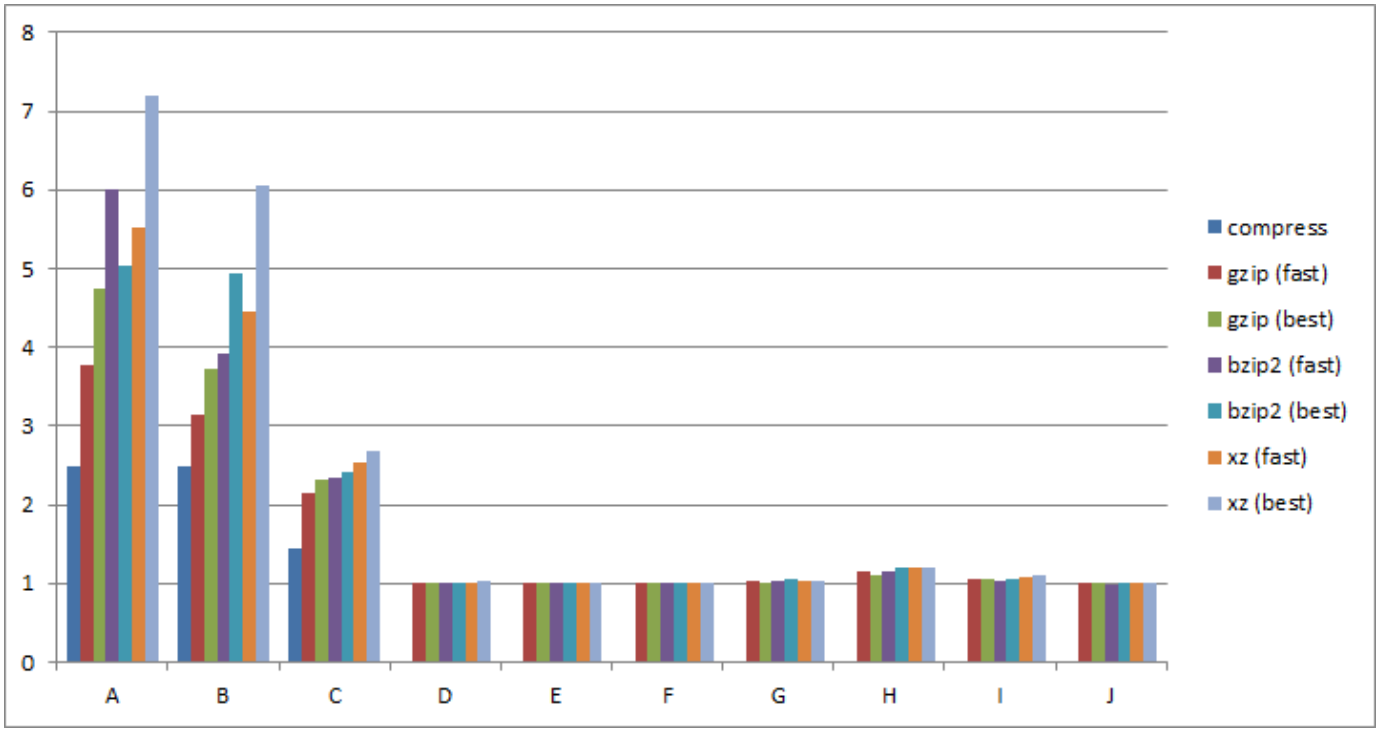

Fig. 5. Comparison of compression ratio. A higher value is better.

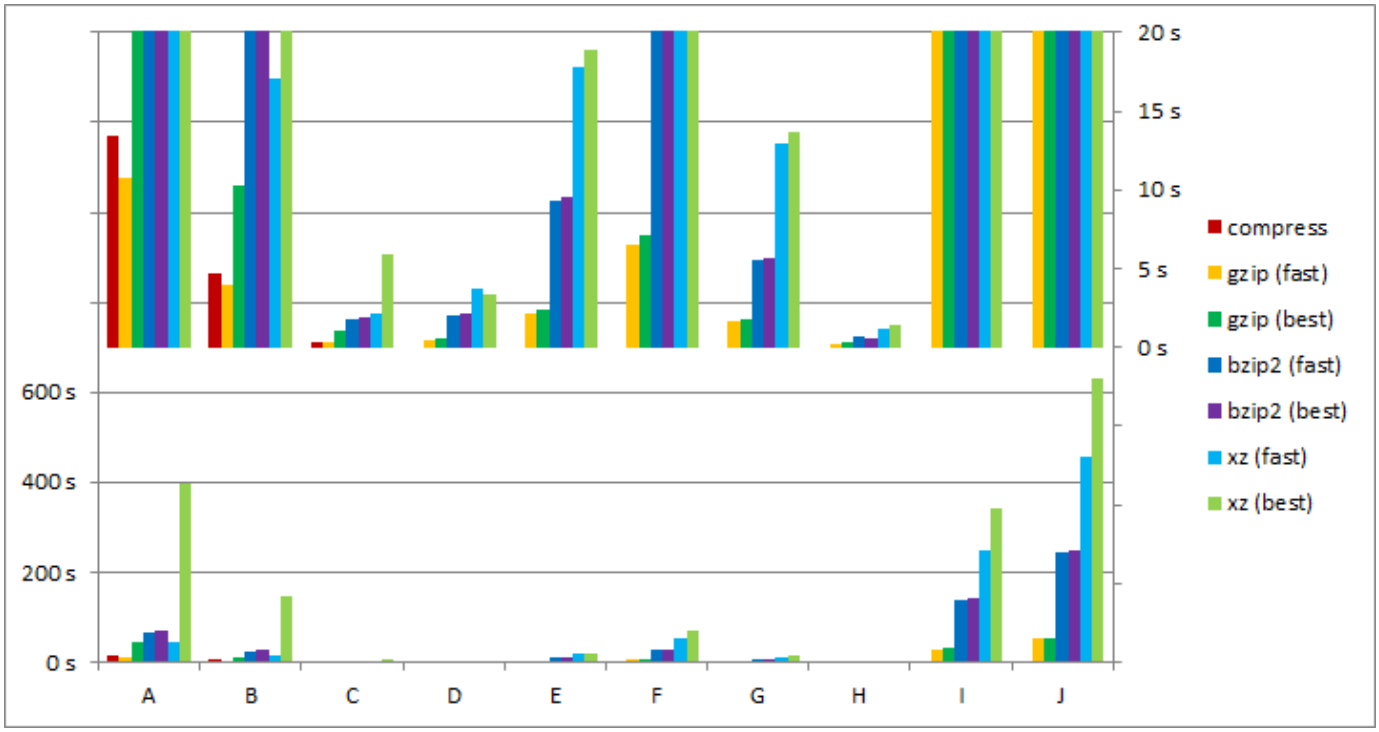

Fig. 6. Comparison of compression run time. A lower value is better. Cut view (top) and complete view (bottom).

as *compress* fails to compress such data. Because there are much better alternative solutions for data compression, *compress* should be viewed as a legacy tool and not to be used in upcoming applications.

Compression ratio of *gzip* isn't outstanding but reliable. It is a fast alternative to other compression tools and should therefore be considered as a compression option for communication applica-

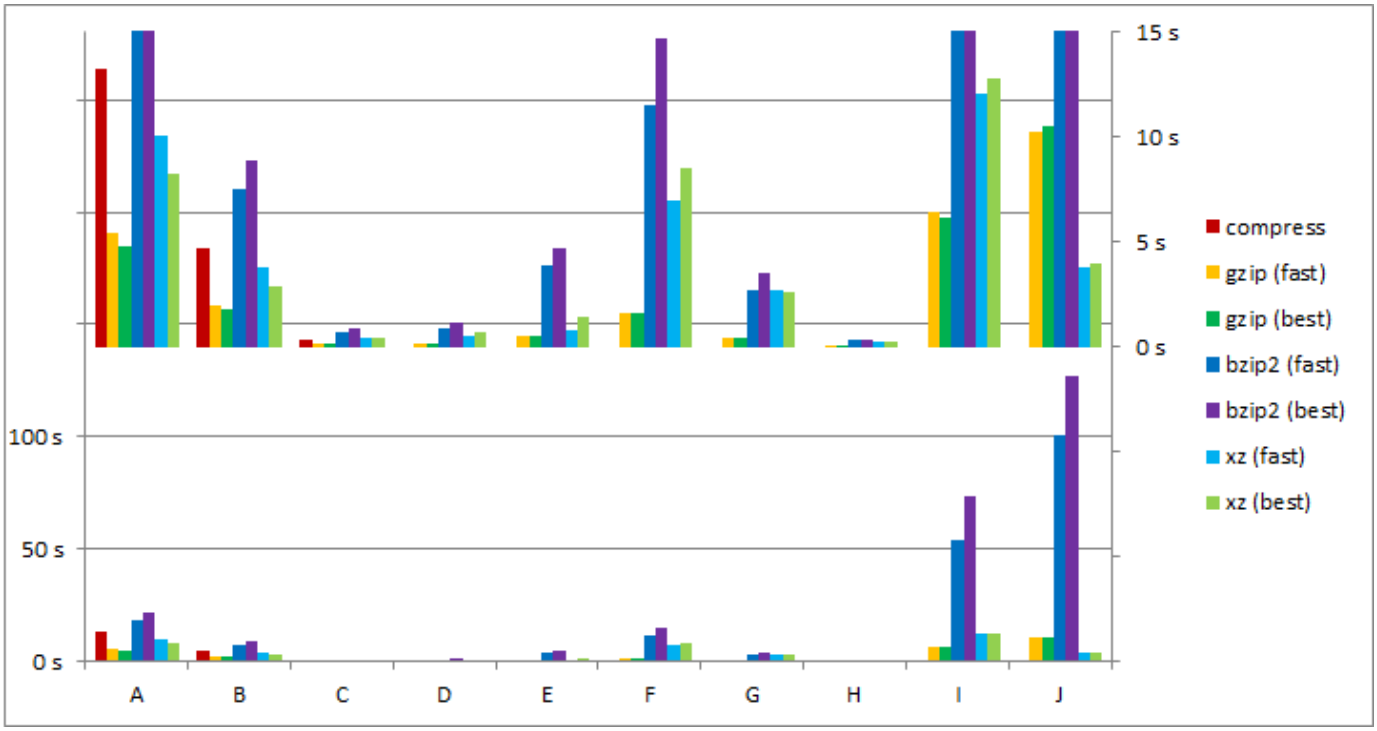

Fig. 7. Comparison of decompression run time. A lower value is better. Cut view (top) and complete view (bottom).

tions. Because they are officialy supported by the HTTP-Standard, *gzip* implementations are widely applicable.

bzip2 is slower than *gzip* but most of the time it achieves a higher compression ratio. Its drawback is the slow decompression speed, which was the worst of the tested tools. *bzip2* can be seen as a compromise of *gzip* and *xz*, providing higher compression ratio but lower compression speed.

In the range of the compared tools, *xz* is the one with the highest compression ratio (dependent on the used quality mode). It is also the tool with the highest compression run-time, which must be considered before processing a large amount of data. Nevertheless its decompression run-time is much better than the one of *bzip2*, which makes it a good solution for spreading data to multiple receivers (e. g. file downloads).

The study also found that the exact achieved compression results are most effective for text data. For other data types compression is useful when handling large amounts of data or spreading data multiple times.

REFERENCES

[1] P. Jannesen, "ncompress." [Online]. Available: https://mirrors.kernel.org/gentoo/distfiles/ncompress-4.2.4.4.tar.gz

[2] J.-l. Gailly and M. Adler, "gzip." [Online]. Available: http://ftp.gnu.org/gnu/gzip/gzip-1.6.tar.gz

[3] J. Seward, "bzip2." [Online]. Available: http://bzip.org/1.0.6/bzip2-1.0.6.tar.gz

[4] L. Collin, "xz." [Online]. Available: http://tukaani.org/xz/xz-5.0.7.tar.gz

[5] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," May 10, 1994. [Online]. Available: http://apotheca.hpl.hp.com/ftp/pub/DEC/SRC/research-reports/SRC-124.pdf

[6] H. Yokoo, "Improved variations relating the ziv-lempel and welch-type algorithms for sequential data compression," *IEEE Transactions on Information Theory*, vol. 38, no. 1, pp. 73–81, 1992.

[7] P. Deutsch, "Deflate compressed data format specification," Network Working Group, May 1996. [Online]. Available: http://tools.ietf.org/pdf/rfc1951.pdf

[8] MatthÃd'us Wander, "Verlustfreie datenkompression mittels der lz-algorithmen," Ph.D. dissertation.

[9] Hussein Al-Bahadili, "A novel lossless data compression scheme based on the error correcting hamming codes."

[10] Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, 1984.

[11] Manas Kumar Mishra, Tapas Kumar Mishra, Alok Kumar Pani, "Parallel lempel-ziv-welch (plzw) technique for data compression." [Online]. Available: http://www.ijcsit.com/docs/Volume\%203/vol3Issue3/ijcsit2012030340.pdf

[12] Jean-loup Gailly, "Gnu gzip: The data compression program for gzip," 28 May 2013. [Online]. Available: www.gnu.org/software/gzip/manual/gzip.pdf

[13] D. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[14] M. Adler, "pigz." [Online]. Available: http://zlib.net/pigz/pigz-2.3.3.tar.gz

[15] J. Gilchrist, "pbzip2." [Online]. Available: https://launchpad.net/pbzip2/1.1/1.1.12/+download/pbzip2-1.1.12.tar.gz
[16] L. Torvalds and and others, "Linux kernel," 2014. [Online]. Available: https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.17.3.tar.xz
[17] Wikimedia Foundation Inc., "enwikinews-20141119-pages-articles-multistream.xml.bz2," 2014. [Online]. Available: http://dumps.wikimedia.org/enwikinews/20141119/enwikinews-20141119-pages-articles-multistream.xml.bz2
[18] I. Docker, "docker-v1.4.0.tar.gz," 2014. [Online]. Available: https://github.com/docker/docker/archive/v1.4.0.tar.gz
[19] Cristian109, "1vs0_fumesv4_movements1_2_3.ogg," 2012. [Online]. Available: http://upload.wikimedia.org/wikipedia/commons/e/e5/1vs0\textunderscoreFumesv4\textunderscoreMovements1\textunderscore2\textunderscore3.ogg
[20] MerveillePédia, "2s4._438_synthé_différents_en_104_minutes.ogg," 2012. [Online]. Available: http://upload.wikimedia.org/wikipedia/commons/e/e6/2s4.\textunderscore438\textunderscoresynth\%C3\%A9\textunderscorediff\%C3\%A9rents\textunderscoreen\textunderscore104\textunderscoreminutes.ogg
[21] J. Morgenstern, "sintel-master-51.flac," 2010. [Online]. Available: http://media.xiph.org/sintel/sintel-master-51.flac
[22] A. Vonbun, "360Âř_pano_auenfeld_hochtannberg.jpg," 2010. [Online]. Available: http://upload.wikimedia.org/wikipedia/commons/3/34/360\%C2\%BA\textunderscorePano\textunderscoreAuenfeld\textunderscoreHochtannberg.jpg
[23] Araz, "Wealth_gini_map_2013.png," 2013. [Online]. Available: http://upload.wikimedia.org/wikipedia/commons/5/53/Wealth\textunderscoreGini\textunderscoreMap\textunderscore2013.png
[24] Debian Project, "Debian: i386," 2014. [Online]. Available: http://cdimage.debian.org/debian-cd/7.7.0/i386/iso-cd/debian-7.7.0-i386-CD-1.iso
[25] C. Levy and and others, "Sintel.2010.1080p.mkv," 2010. [Online]. Available: http://download.blender.org/demo/movies/Sintel.2010.1080p.mkv