The Improvised GZIP, A Technique for Real Time Lossless Data Compression

Ahmad Saeed Shah*, Muhammad Athar Javed Sethi

Department of Computer Systems Engineering, University of Engineering and Technology, Peshawar, Pakistan
engrasshah@gmail.com, atharsethi@uetpeshawar.edu.pk

Abstract

Whenever it comes to data processing, the user always faces two major constraints. One is storage capacity and second is bandwidth. These two resources must be efficiently utilized by compressing the data. Enormous algorithms are used to compress data. As far as, compression in storage is concern, GZIP is used on large scale for lossless data compression. However, it is not desirable to carry out lossless data compression for real time data. In this paper, an improvisation is proposed in the existing GZIP algorithm for compressing real time data by a contemporary concept of introducing Adaptive Huffman algorithm by replacing the traditional Huffman encoder (static). Simulations have proved that improvised GZIP has approximate 18% better compression ratio and space saving than traditional GZIP for real time data. This research paper extends the usability of GZIP algorithm to carry out lossless compression for real time data.

Keywords: Lossless data compression, LZ77, Huffman encoding, GZIP and Adaptive Huffman.

Received on 09 June 2019, accepted on 23 June 2019, published on 26 June 2019

doi: 10.4108/eai.1-10-2019.160599

*Corresponding author. Email: engrasshah@gmail.com

1. Introduction

Any technique through which the data size is significantly reduced is known as data compression. If data is not compressed, then ultimately cost of resources like storage and bandwidth will be increased [1]. Data compression is categorized in two types.

1.1 Lossy Compression

In lossy compression, data is compressed in such a way that hundred percent data is not extracted after decoding the compressed data. Due to this fact, it is known as irreversible compression. Common techniques for lossy compression are scalar quantization, vector quantization and wavelet [2]. This type of compression is mostly used in compressing general pictures, audios and videos. JPEG, PNG are the best examples of lossy compression techniques.

1.2 Lossless Compression

In most of the cases the loss of information is not tolerated due to the precious nature of data. For example, in the field of medical imaging, finger print data, computer programs etc. In addition, lossless data compression is more desirable in case of text. In such cases, data must be compressed in such a way that hundred percent data is extracted after decoding the compressed data. Hence lossless data compression is more preferred over lossy compression. As the data is decoded hundred percent as of the original data, that's why it is referred as reversible compression.
We know the fact that data contains redundancy. The data can be compressed by processing the redundant part of data. As a result, less hardware optimization will be required. There are two major approaches for carrying out lossless data compression.

1.2.1 Dictionary Based

According to this strategy, data is encoded by maintaining a list or dictionary of the most frequently occurring symbols or literals. The encoded data contains the index or position number of the symbol in the dictionary. Examples are LZ77, LZSS and LZ78.

Dictionary based is further categorised into static and dynamic dictionary. In static dictionary, the list of symbols is not changed once the dictionary is maintained. In some cases, addition to the dictionary is permissible but deletion is not allowed. On the other hand, in dynamic dictionary, the list of symbols is updated with the arrival and removal of symbols. Compression ratio for Dictionary based compression is greater for highly corelated data.

1.2.2 Entropy Based

According to this type of compression strategy, the data is encoded using the statistical information of the symbols. For this reason, it is also referred as statistical coding. Examples include Huffman coding [3, 4] and Shannon Fano coding.

This paper is divided into nine sections. Section 2 is about related work. In section 3, existing GZIP is discussed followed by the improvisation in existing GZIP in section 4. Simulations results are discussed in section 5. At the end, results are discussed in section 6. Conclusions are presented in section 7 followed by recommendations and limitations in section 8 and 9 respectively.

2. Related Work

In the field of data compression, a lot of research has been carried out in different ages to address different issues. Some algorithms are data specific, such as text [5], picture [6], audio [7] and video [8] etc. whereas some algorithms focused on compression ratio while some focused-on compression time. Hence, there is a rich literature regarding data compression. We know that, data may be in any form. It may be in the form of picture, audio, video or a text file. In [9] better compression ratio and comparable PSNR value has been achieved as compared to traditional JPEG by using histogram-based block optimization & arithmetic coding technique is used for compressing and decompressing a lossy image. But the algorithm was only implemented for 8-bit grey scale image. Similarly, in case of text data, a comparative study [5] was carried out in 2010. Different lossless compression algorithms were implemented and analysed in terms of compression ratio, compression and decompression time. Better results were obtained for Shannon Fano algorithm.

One of the well-known algorithms is GZIP. GZIP is a single file compressor. It successfully compresses a single file. Latest research has been carried out to compressed already compressed files. Such applications are known as archivers. In 2019, [10] presents a novel scheme of archiving GZIP compressed files.

In [11] the bzip2 (based on Burrows-Wheeler Transform (BWT) was improvised for block size of 500 KB and implemented it over FPGA that supports 4KB block size.

The GZIP basically works on the principle of lossless data compression algorithm called DEFLATE [12], which is a hybrid algorithm of dictionary based algorithm, LZ77 [[13, 14] and entropy based algorithm, the Huffman coding. LZ algorithms searches the incoming symbol in the previously arrived symbols. If a symbol is successfully found in previous dictionary then that symbol in the upcoming data is encoded as pointer to the matching symbol. [14]

GZIP is a widely implemented in many applications to carry out lossless compression. For Example. GZIP is used different webservers like Apache and Microsoft Webserver, IIS [15]. But it has not a remarkable performance to compress real time data.

3. Existing GZIP

GZIP has two flavours as given in Figure 1. To compress real time data, GZIP uses a combination LZ77 and static Huffman encoder. Where as to compress non real time data, GZIP uses combination of LZ77 and dynamic Huffman algorithm. Static Huffman encodes the data in one go by assigning every symbol a fix length of code. Static Huffman encoder does not bother to know the frequency distribution of data. Hence the data is encoded efficiently with a compromise on compression ratio. The poor compression ratio is due to the fix length codes. Because the symbols that have less occurrence also gets the same length code.

On the other side, dynamic Huffman encoder encodes the data in two passes. In the first go, frequency distribution of data is calculated and in the second go the symbols are

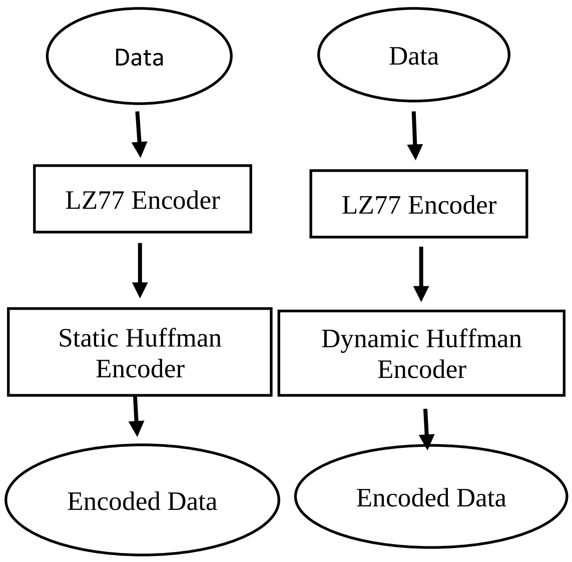

Figure 1. Two flavors of GZIP

encoded using the statistical information of every symbol. a kind of moderate homework is done by dynamic Huffman encoder to encode the symbols. Dynamic Huffman encodes the symbols with variable length codes in such a way that the symbol which occurs the most gets lesser length code and the symbol which has low density, is encoded with comparatively lengthy codes. As a result, good compression ratio is achieved. But the problem associated with dynamic Huffman to encode real time data is its two-pass property.

3.1 Lempel-Ziv 77

It is a dictionary based lossless data compression algorithm that compresses data sequentially. LZ77 maintains a dictionary of previously encoded literals or symbols. According to this algorithm, the stream of symbols is examined by a sliding window. Sliding window is comprised of search buffer and look ahead buffer. Search buffer contains the recently encoded symbols and look ahead buffers contain the incoming symbols that are going to be encoded.

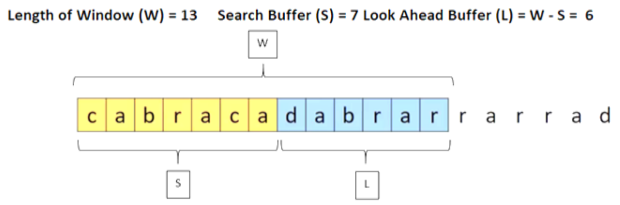
Length of Window (W) = 13
Search Buffer (S) = 7 Look Ahead Buffer (L) = W - S = 6
W
c a b r a c a d a b r a r r a r r a d
S
L

Figure 2. Working of LZ77

The symbols are scanned by the sliding window and matches are found between the two buffers. The lengthier match found means the more data is compressed. The encoded data is obtained in the form of three different information like offset (distance between two matched symbols in two buffers), length (number of symbols matched in two buffers) & literal (next unencoded symbol in the lookahead buffer).

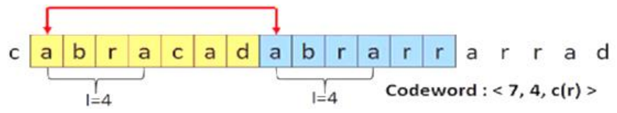
c a b r a c a d a b r a r r a r r a d
l=4
l=4
Codeword : < 7, 4, c(r) >

Figure 3. Representation of LZ77 encoded data

3.2 Huffman Coding

Huffman coding is the example of statistical based compression. In Huffman coding, the statistical information of symbols such as frequency distribution are used to encode symbols.

There are two types of Huffman coding. One is static Huffman and other is dynamic Huffman.

3.2.1 Static Huffman

In static Huffman fix length codes are used to encode symbols. Look-up tables are used to encode and decode data. All the symbols are encoded with fix length codes. Length of codes is not related to the frequency of symbols. It means both less and frequently occurring symbols gets the same length code. This phenomenon is explained in Table 1.

Table 1. Static Huffman coding

Symbols	Frequency	Fix code	
A	45	000	45×3 = 135
B	13	001	13×3 = 39
C	12	010	12×3 = 36
D	16	011	16×3 = 48
E	9	100	09×3 = 27
F	5	101	05×3 = 15
Total	**100 B (Uncompressed)**		300 bits or approx.**38 B**

Hence the symbols are encoded in one go without calculating the weights of frequency. This property makes static Huffman coding desirable for encoding real time data. Because it is time efficient. The compression ratio is not good because all the symbols are assigned fix length codes.

3.2.2 Dynamic Huffman

Dynamic Huffman is a two-pass algorithm. In first pass, frequency distribution of symbols is calculated and in second pass symbols are encoded. In dynamic Huffman, variable length codes are assigned to symbols depending upon their occurrence. Such that symbols with less frequency are encoded with greater bits and symbols with high occurrence are encoded with fewer bits. That results in good compression ratio.

In dynamic Huffman, a binary tree, as shown in Figure 4, is maintained for the data in Table 1.

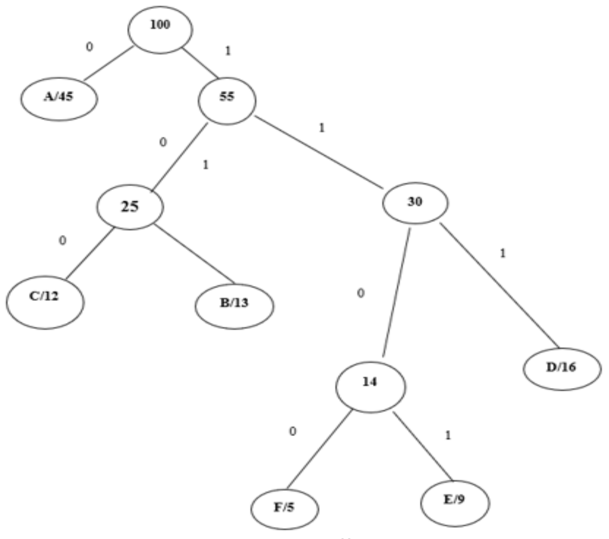

Figure 4. Dynamic Huffman - Binary tree

The variable length code obtained from Figure 4 is shown in Table 2

Table 2. Variable length codes - dynamic Huffman

Symbols	Frequency	Variable	
A	45	0	45 × 1 = 45
B	13	101	13 × 3 = 39
C	12	100	12 × 3 = 36
D	16	111	16 × 3 = 48
E	9	1101	09 × 4 = 36
F	5	1100	05 × 4 = 20
Total 100 B (uncompressed)			**224 Bits**

4. Improvised GZIP

Improvisation is brought to the existing GZIP by using Adaptive Huffman coding by replacing Static Huffman coding as seen in Figure 5. Adaptive Huffman has functionalities of both static Huffman and dynamic Huffman. like static Huffman encoder, adaptive Huffman encodes data in one go. initially the symbols are encoded with fix length codes but the Huffman tree is updated with respect to the arrival of new symbol and length of code for every symbol change as per its density. At the end, like dynamic Huffman encoder the most frequently occurring symbols is encoded with less length of code.

In first step, the data is encoded by LZ77 compressor. A sliding window which is further split into two buffers, one is search buffer and second is look ahead buffer. The symbols are scanned by the sliding window and matches are found between the two buffers. The encoded data is obtained in the form of three different information like offset (distance between two matched symbols in two buffers), length (number of symbols matched in two buffers) & literal (next unencoded symbol in the lookahead buffer). In second step data encoded by LZ77 encoder is passed to adaptive Huffman encoder and the data is finally compressed.

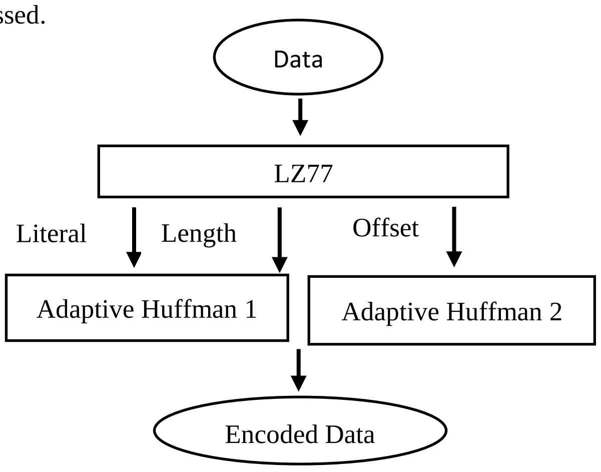

Figure 5. Generic Representation of Improvised GZIP

4.1 Adaptive Huffman

Encoding procedure of Adaptive Huffman is shown in Figure 6. The symbol is initially assigned some fixed length code, which is updated gradually as the encoder encounters new symbols. If the symbol arrives for the first time then the symbol is encoded with Not Yet Transmitted node in association with that initial fix length code. On the other hand, if the symbol is encountered before then the symbol is encoded with a code, from root node to the corresponding node. The update procedure is called and tree is updated in such a way that if weight of left node is greater than the weight of right one, then they are swapped. This procedure continues until the last symbol is arrived.

As far as, the decoding procedure is concerned, it is done quickly because the same tree is used for decoding as well. The detailed decoding procedure of Adaptive Huffman is shown in Figure 7. Decoding starts from the top, such as root node. First the current node is checked for the leaf node followed by the NYT node. In case of the current node is NYT node, the symbol is decoded in accordance to the node. If the current node is not an NYT node then the symbol is decoded by fix length code. Then update procedure is called. This process continues until the last symbol is decoded.

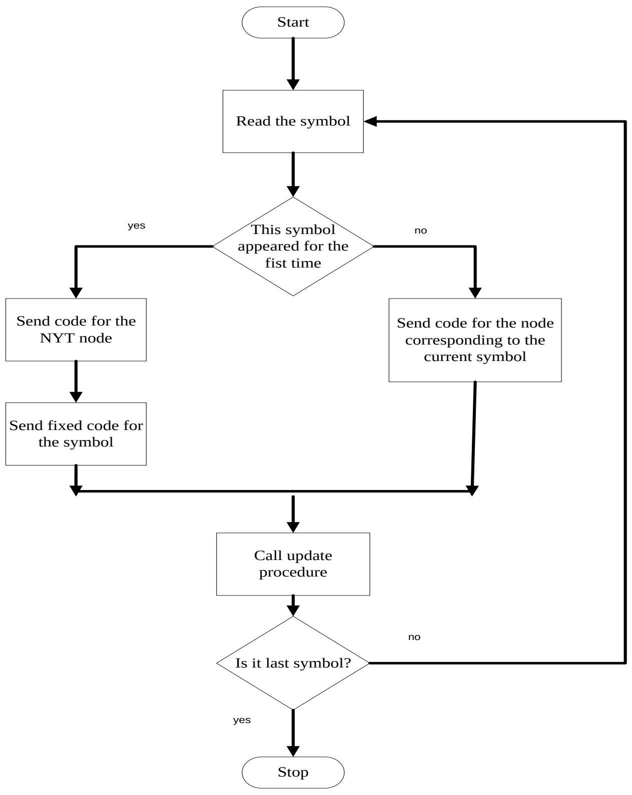
Start
Read the symbol
This symbol appeared for the fist time
yes
no
Send code for the NYT node
Send code for the node corresponding to the current symbol
Send fixed code for the symbol
Call update procedure
Is it last symbol?
no
yes
Stop

Figure 6. Adaptive Huffman encoding procedure

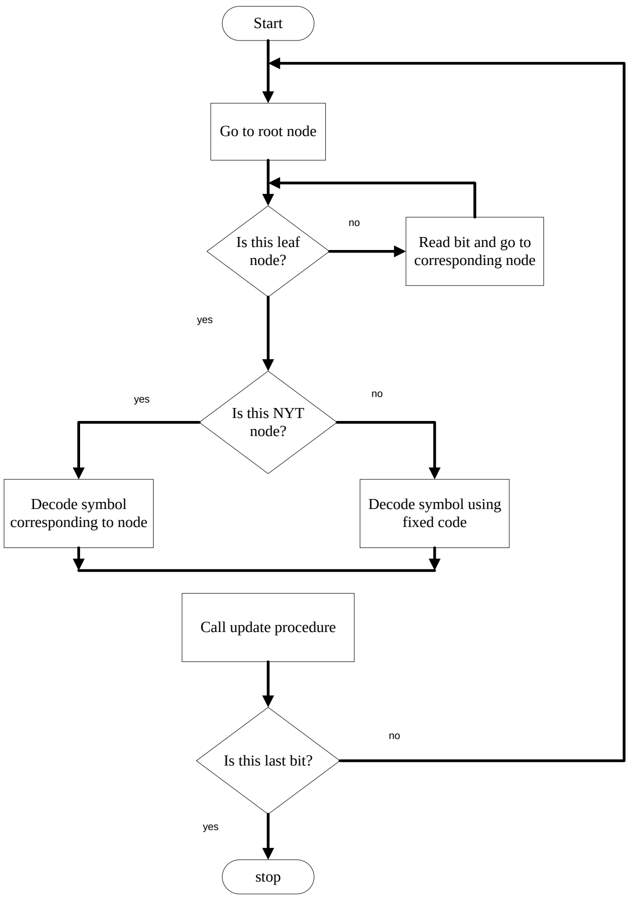

Figure 7. Adaptive Huffman decoding procedure

5. Results

Results are recorded in terms of compression ratio and space savings. Where, Compression ratio (CR) is the ratio of original or uncompressed file size to the encoded or compressed file size.

$$\text{Compression Ratio} = \frac{Uncompressed\ file\ zise}{Compressed\ file\ size}$$

If, CR = 1, it means no compression

CR > 1, Desirable

CR <1, Blunder

Where the space saving is the amount of space saved after compression.

$$\text{Space Saving [5]=} \left[1 - \frac{Compressed\ file\ size}{Uncompressed\ file\ size}\right]\%$$

Table 3. Compression Ratio & Space Saving of GZIP

#	File Name	Uncom-pressed (bytes)	Com-pressed (bytes)	CR	Space Saving %
1	sample.html	24603	17047.875	1.443	30.71
2	sample.c	11150	7509.625	1.484	32.65
3	sample.php	3721	2404.625	1.547	35.38
4	sample1.txt	38240	28130.625	1.359	26.44
5	sample2.txt	125179	105374.37	1.187	15.82
6	sample3.txt	152089	123295.12	1.233	18.93

Table 4. Improvised GZIP for Real time Data

#	File Name	Uncom-pressed (bytes)	Com-pressed (bytes)	CR	Space Saving %
1	sample.html	24603	13039	1.886	47
2	sample.c	11150	5754.625	1.937	48.39
3	sample.php	3721	1888	1.970	49.26
4	sample1.txt	38240	23052.625	1.658	39.72
5	sample2.txt	125179	74678.5	1.676	40.34
6	sample3.txt	152089	87458.125	1.739	42.5

Table 5. Traditional GZIP for Storage data

#	File Name	Uncom-pressed (bytes)	Com-pressed (bytes)	CR	Space Saving %
1	sample.html	24603	12937.875	1.901	47.41
2	sample.c	11150	5662.75	1.969	49.21
3	sample.php	3721	6.3775	2.041	51.02
4	sample1.txt	38240	22406.25	1.706	41.41
5	sample2.txt	125179	74595.375	1.678	40.41
6	sample3.txt	152089	87370	1.740	42.55

After oberving Table 3and Table 4, it is revealed that improvised GZIP (combination of LZ77 and adaptive Huffman) have good compression ratio as compared to traditional GZIP (combination of LZ77 and static Huffman). Whereas, Table 4 and Table 5 show that adaptive huffman and dynamic huffman has almost similar results in combination with LZ77.

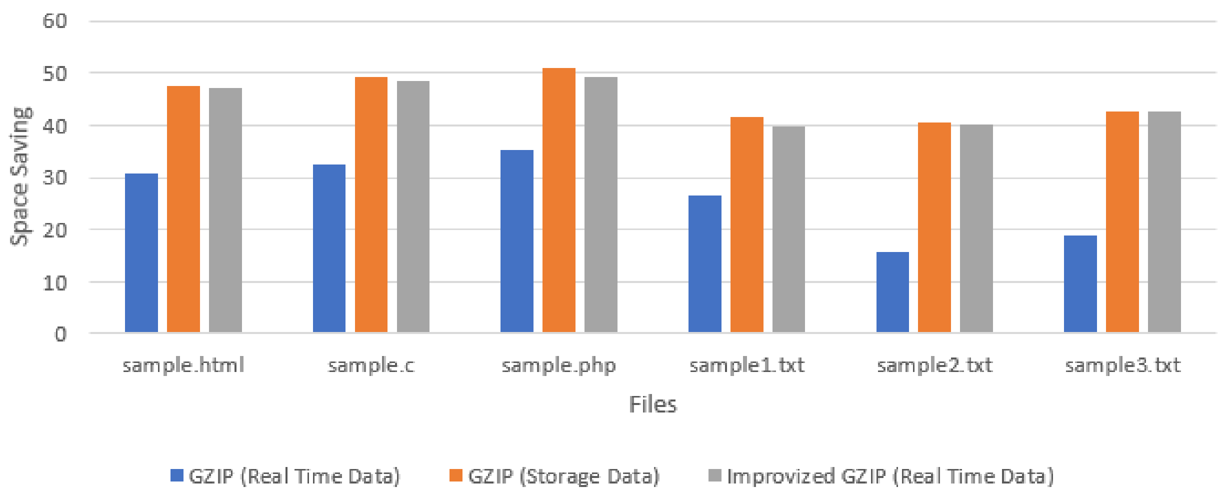

Figure 8. Comparison of Compression Ratio

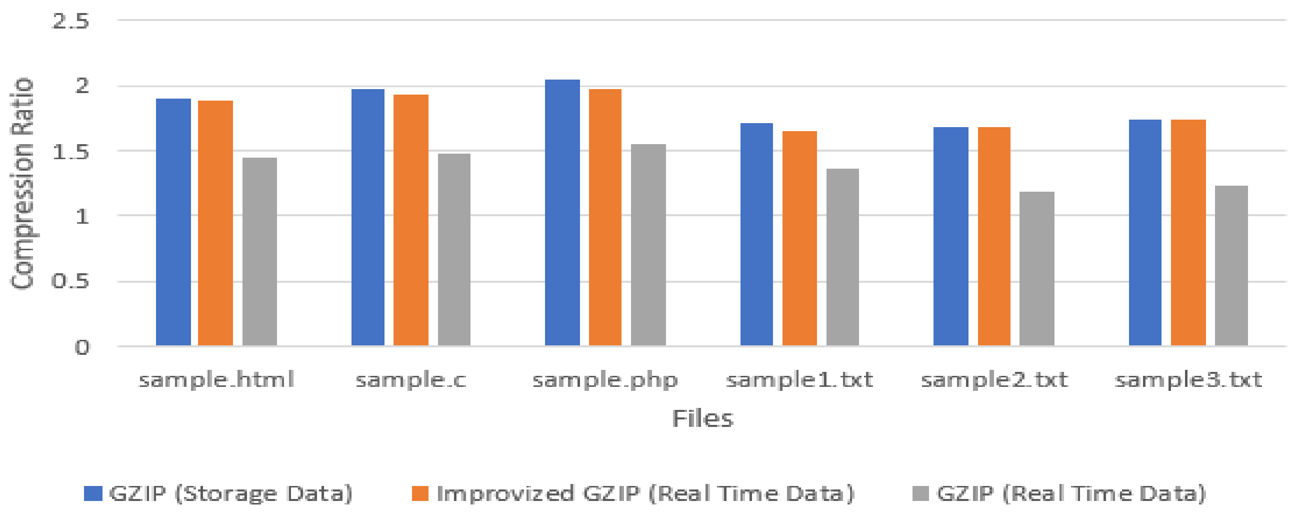

Figure 9. Comparisons of Space Savings

6. Discussion

Traditional GZIP was only suitable for compressing non real time data without any loss in data. In traditional GZIP, where the dynamic Huffman encoder has its significance in terms of good compression ratio. On the other hand, it has several disadvantages which cannot be ignored. One of the major disadvantages is its two-pass nature. Which makes it a bad choice for compressing real time data. On the other hand, this property makes it more complex i.e. higher specification hardware is required plus greater encoding and decoding time is required by GZIP.

Whereas, Improvised GZIP is suitable for hardware implementation compared to traditional GZIP because of less memory requirements for code storage. Secondly, Improvised GZIP can be used to carry out lossless data compression in real time data. The role of Adaptive Huffman encoder in improvised GZIP gives it an edge. As it encodes data in one go. Secondly in Adaptive Huffman the processing time is quite small as compared to Dynamic Huffman, because same tree is used during encoding and decoding processes [5].

Table 6. Comparison of Space Savings for real-time data

#	File Name	Space Saving for GZIP (%)	Space Saving for Improvised ZIP (%)	Difference %
1	sample.html	31	47	16
2	sample.c	33	48	16
3	sample.php	35	49	14
4	sample1.txt	26	40	13
5	sample2.txt	16	40	25
6	sample3.txt	19	42	24
Average				18

Hence, in case of real time data, improvised GZIP gives approximate 18% better results than traditional GZIP.

7. Conclusion

From the above results it is concluded that improvised GZIP (LZ77 and adaptive Huffman) as approximate 18% good results in terms of compression ratio and space saving than traditional GZIP (LZ77 and Static Huffman) for real time data. On the other side, adaptive Huffman and dynamic Huffman have almost similar results in combination with LZ77 for storage data compression, but dynamic Huffman is not suitable for real time data due to its 2 – pass nature. Hence, GZIP is improved for real time data compression by using adaptive Huffman and LZ77 compressors.

8. Recommendations

There are many factors that has either direct or indirect relation with data compression. For example, the size of file, number of unique symbols in a file, occurrence and position of symbols in file, compression and decompression time etc. As this paper is about lossless data compression, therefore, the amount of space saved and compression ratio got an edge over other parameters. The efficiency of improvised GZIP is illustrated in terms of compression ratio and Space saving.

9. Limitations

In this paper, six different files with different file sizes are taken as source files. The results are obtained from these source files only. Hence, it is recommended that Improvised GZIP may be applied on files obtained from diverse sources and different file formats.

References

[1] S. S. Thakare and P. M. Parekar, "Lossless data compression algorithm–a review," *International Journal of Computer Science & Information Technologies*, vol. 5, no. 1, 2014.

[2] P. S. Asolkar, P. H. Zope and S. R. Suralkar, "Review of Data Compression and Different Techniques of Data Compression," *International Journal of Engineering Research & Technology (IJERT)*, vol. 2, no. 1, January 2013.

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

[4] D. Huffman, "A method for the construction of minimum-redundancy codes," *Resonance*, vol. 11, no. 2, pp. 91-99, 2006.

[5] S. R. Kodituwakku and U. S. Amarasinghe, "Comparison of lossless data compression algorithms for text data," *Indian journal of computer science and engineering*, vol. 1, no. 4, pp. 416-425, 2010.

[6] M. U. Ayoobkhan , E. Chikkannan , K. Ramakrishnan and S. B. Balasubramanian , "Prediction-based Lossless Image Compression," *International Conference on ISMAC in Computational Vision and Bio-Engineering. Springer, Cham, 2018.*, pp. 1749-1761, 2018.

[7] U. K. Mondal , "A novel approach to lossless audio compression (LAC)," *Annual Convention of the Computer Society of India, Springer, Singapore*, pp. 99-106, 2018.

[8] W. Gao , G. Jiang , M. Yu and T. Luo , "Lossless fragile watermarking algorithm in compressed domain for multiview video coding," *Multimedia Tools and Applications 2019*, vol. 78, no. 8, pp. 9737-9762, 2019.

[9] S. Dutta, A. Abhinav, P. Dutta, P. Kumar and A. Halder, "An Efficient Image Compression Algorithm Based on Histogram Based Block Optimization and Arithmetic Coding," *International Journal of Computer Theory and Engineering*, vol. 4, no. 6, p. 954, 2012.

[10] M. Kerbiriou and R. Chikhi, "Parallel decompression of gzip-compressed files and random access to DNA sequences," *arXiv preprint arXiv:1905.07224.*, 17 May 2019.

[11] W. Qiao , Z. Fang , . M. Chang and J. Cong , "An FPGA-Based BWT Accelerator for Bzip2 Data Compression," *IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE*, pp. 96-99, 2019.

[12] P. Deutsch, ""DEFLATE Compressed Data Format Specification version 1.3," *Network Working Group*, 1996.

[13] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337-343, 1977.

[14] B. Sailesh, . P. Fleming , L. Mosur , D. Cassetti and S. Palermo , "Data compression engine for dictionary based lossless data compression". U.S Patent Patent Application No. 16/228,300., 25 April 2019.

[15] J. Ouyang , H. Luo , Z. Wang , J. Tian , C. Liu and K. Sheng , "FPGA implementation of GZIP compression and decompression for IDC services," *2010 International Conference on Field-Programmable Technology IEEE*, pp. 265-268, 2010.

[16] P. M. Parekar and S. S. Thakare , "Lossless Data Compression Algorithm – A Review," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 1, pp. 276-278, 2014.