Journal of Computer Networks, Architecture and High Performance Computing
Volume 5, Number 1, January 2023
https://doi.org/10.47709/cnahpc.v5i1.1993

Submitted : January 12, 2023
Accepted : February 18, 2023
Published: February 19, 2023

Base-Delta Dynamic Block Length and Optimization on File Compression

Tommy[1)*], Ferdy Riza[2)], Rosyidah Siregar[3)], Manovri Yeni[4)],Andi Marwan Elhanafi[5)], Ruswan Nurmadi[6)]
[1)3)5)6)] Universitas Harapan Medan, [2)]Universitas Muhammadiyah Sumatera Utara, [4)]Universitas Muhammadiyah Aceh, Indonesia
[1)]tomshirakawa@email.com, [2)]ferdyriza@umsu.ac.id, [3)] rosyidah_siregar.unhar@harapan.ac.id,
[4]manovri.yeni@unmuha.ac.id, [5)]andimarwanelhanafi@gmail.com, [6)]ruswannurmadi@yahoo.com

ABSTRACT

Delta compression uses the previous block of bytes to be used as a reference in the compression process for the next blocks. This approach is increasingly ineffective due to the duplication of byte sequences in modern files. Another delta compression model uses the numerical difference approach of the sequence of bytes contained in a file. Storing the difference value will require fewer representation bits than the original value. Base + Delta is a compression model that uses delta which is obtained from the numerical differences in blocks of a fixed size. Developed with the aim of compressing memory blocks, this model uses fixed-sized blocks and does not have a special mechanism when applied to file compression in general. This study proposes a compression model by developing the concept of Base+Delta encoding which aims to be applicable to all file types. Modification and development carried out by adopting a dynamic block size using a sliding window and block header optimization on compressed and uncompressed blocks giving promising test results where almost all file formats tested can be compressed with a ratio that is not too large but consistent for all file formats where the ratio compression for all file formats obtained between 0.04 to 12.3. The developed compression model also produces compression failures in files with high uncompressed blocks where the overhead of additional uncompressed blocks of information causes files to become larger with a negative ratio obtained of -0.39 to -0.48 which is still relatively small and acceptable.

Keywords: File Compression, Delta, Difference, Dynamic Block, Optimization

1. INTRODUCTION

Delta encoding is an encoding scheme that uses delta as the encoding result. The concept of delta here is the difference value contained in the data. The value of the difference between one data and another data usually has a simpler representation. This concept is the basis of delta encoding applications in data compression. Initially, delta encoding was developed and applied to HTTP communications as can be seen in RFC-3229 (Mogul, et al., 2002). Using the cache from the previous request, data from a new request can be constructed using the delta value from the previous request thereby reducing the size of the data transmitted to the client where the delta value has a lower value than the original data so that it can be represented with simpler symbols or bits. Another application of the previous delta encoding in the HTTP communication domain can also be seen in several previous studies which show a fairly good potential of delta encoding in increasing the efficiency of the size of the transmitted data (Banga, Douglis, & Rabinovich, 1997) (Housel & Lindquist, 1996) which was then developed further by using modified request headers to obtain better efficiency (Mogul, Douglis, Feldmann, & Krishnamurthy, 1997).

In modern applications, delta encoding has been used for various purposes such as file backup (Zhang, et al., 2020) and archiving (Dolgorsuren, Khan, Rasel, & Lee, 2019), IoT systems (Hanumanthaiah, Gopinath, Arun, Hariharan,, & Murugan, 2019), file copy and data deduplication (Zhang, et al., 2021) (Vestergaard, Zhang, & Lucani, 2019) (Samteladze & Christensen, 2012). Delta encoding variants have been developed such as Xdelta (MacDonald, 2000), Zdelta (Trendafilov, Memon, & Suel, 2002), Ddelta (Xia, et al., 2014), Edelta (Xia, et al., 2015) and Gdelta (Tan, Zhang, Zou, Liao, & Xia, 2020). Delta encoding variants such as Xdelta and Zdelta use “Copy” and “Insert” instructions on the output encoding. The main concept of the variant is to detect matching strings contained in the same chunk and replace them with a “Copy” or “Insert” instruction based on the match/unmatched strings in the

* Corresponding author

Journal of Computer Networks, Architecture and High Performance Computing
Volume 5, Number 1, January 2023
https://doi.org/10.47709/cnahpc.v5i1.1993

Submitted : January 12, 2023
Accepted : February 18, 2023
Published: February 19, 2023

chunk. Another approach to applying delta encoding is to use the diff value obtained from the subtraction operation of the data bytes.

Simple delta encoding performs a subtraction operation from the data byte with the previous byte where the result of the operation is a delta which is the result of the encoding process. Later improvement on delta encoding uses a base value which is used to calculate the delta value of the data bytes. The base value can be selected to obtain the smallest possible delta value thereby increasing the compression ratio. One of the latest compression models that use this approach is Base-Delta-Immediate compression (Pekhimenko, et al., 2012). "Copy" / "Insert" based delta encoding uses a base in the form of chunks to minimize duplication in the input chunk if similarities are found between the input chunk and the base chunk. So, the effectiveness of the model is very dependent on how the base is generated. This delta encoding model works best when the base chunk has high similarity to the input chunk. Because of this nature, reference data or files that have a high similarity to the input are used as the base. If the reference data or file is not available, then the base chunk is obtained from the previous data bytes. In general file compression, the base used during the encoding process is the base obtained from the previous byte. Depending on the content contained in a file, the text usually has a high enough duplication of chunks that the "Copy"/"Insert" methods can be used on this type of content. Data collections or datasets that contain repetitive strings are also good inputs to this encoding model. This can be seen in several previous papers which discussed the application of delta encoding to datasets (Suel, 2019). Specific data such as genome sequencing data (Cogo, Paulo, & Bessani, 2020) has a low number of string duplications which results in a poor compression ratio of the encoding results in delta encoding based on similarity. Current file formats that exist today have stored bytes of data in a more efficient way as so string duplication is rare to be found.

In the case of a low level of string duplication, the delta encoding scheme using the numerical delta of the difference in data bytes is a better alternative compared to the similarity-based model. However, the optimal compression ratio produced in this scheme is much lower than the similarity-based delta encoding model. In modern file formats, duplicated byte strings are rare to be found. On the other hand, low byte variation within a block of a given size is easier to find than finding duplicate block strings in modern files. However, like compression models in general, the delta encoding model using numerical differencing deltas has additional information bits needed by the decoder during reconstruction. Additional bits of information that are not managed properly can result in excessive overhead which will burden the encoding output (Pekhimenko, Guo, Jeon, Huang, & Zhou, 2018). This paper proposes a base delta encoding model that divides data into compressed blocks where each block will use a base value to calculate the delta value obtained from the difference between the base and the data bytes contained in the block. Compared to the previous delta encoding model, the data block size will change dynamically to achieve a more optimal compression ratio, especially in byte strings that have low variance. Restricting procedures and different header types on uncompressed block was used to minimize additional information bit overhead. So the model can effectively compress the part of the file that has low variance byte sequence.

2. LITERATURE REVIEW

Delta Encoding and Compression

As we can see in RFC-3229 (Mogul, et al., 2002), delta encoding is an encoding scheme used to encode content in the form of a delta representation of the content transmitted via HTTP and has been continuously developed and implemented in various applications (Henziger & Carlsson, 2019) (Italiano, Prezza, Sinaimeri, & Venturini, 2021). The delta encoded update that is sent to the client for the request made is obtained from the difference between the old instance contained in the cache and the new instance that will be sent by the server, where the encoded delta update will have a smaller size compared to sending all new content. Simple notation of delta encoding can be seen in equation 1.

$$\Delta(R:V) = A(R,V) \qquad (1)$$

Where R is the reference string and V is the new version string. Compression in delta encoding is obtained from the encoding process of a file or data using references from files or other data. The performance of delta encoding depends on the magnitude of the difference between the pairs of files used (Hunt, Vo, & Tichy, 1998) where the minimum delta is obtained by getting smaller the difference between R and V. The difference between two files or instances can be measured using the Longest Common Subsequence (LCS) which can be seen in equation 2.

* Corresponding author

Journal of Computer Networks, Architecture and High Performance Computing
Volume 5, Number 1, January 2023
https://doi.org/10.47709/cnahpc.v5i1.1993

Submitted : January 12, 2023
Accepted : February 18, 2023
Published: February 19, 2023

$$difference = \frac{size(file_1)+size(file_2)}{2} - size(LCS) \quad (2)$$

Where LCS is the maximum length of the same string sequence between two files. In delta encoding, encoded delta is an instruction that contains a pointer to data contained in a reference as can be seen in delta encoding algorithms such as diff (Hunt & MacIlroy, 1976), vcdiff (Korn, Macdonald, Mogul, & Vo, 2002), and the Xdelta family.

In the data compression domain, delta compression is broadly divided into two approaches. The first approach is to use a matched or unmatched string approach between the reference and the new string as can be seen in delta encoding in general, and the second approach uses numerical differences. The delta representation obtained is in the form of encoding from the numerical differences of the data pairs. A simple equation for the difference between data can be seen in equation 3 (Engelson, Fritzson, & Fritzson, 2000).

$$\Delta b_i = b_i - b_{i-1} \quad (3)$$

Where b is an integer array block with a certain size. The performance of delta compression using the numerical difference approach depends on the size of the variation in the values contained in the data. Delta compression stores the difference or delta value which has a smaller bit representation compared to the original value so that data with a smaller size is obtained (Jahani, Cafarella, & Ré, 2011).

Base+Delta Compression

Base+Delta Compression is a compression mechanism aimed at compressing data contained in cache memory (Pekhimenko, et al., 2012). The main concept of this compression mechanism is to represent the data bytes contained in the cache line with fewer bytes by using the value of the numerical difference between the words contained in the cache line. Base+Delta compression uses cache lines or fixed-sized blocks of 64 bytes. Within a block, each block is divided into sub-blocks in several alternatives, namely 8 sub-blocks that have 8 bytes in each sub-block, 16 sub-blocks containing 4-bytes each, 32 sub-blocks containing 2-bytes each, which is then followed by selecting a base value. as a delta reference for each block.

Selection of size variations in block size and base value requires prior analysis. If a block is C bytes in size, the size of each value within the block or sub-block is k bytes and the set of values to be compressed is $S = (v_1, v_2, \dots, v_n)$, where $n = C/k$. Then it is necessary to determine the base values of B^* and k which will produce the optimal compression block. Where the notation of the compressed output can be seen in equation 4 below.

$$\{k, B^*, \Delta = (\Delta_1, \Delta_2, \dots, \Delta_n)\} \quad (4)$$

Where $\Delta_i = B^* - v_i \; \forall_i \in \{1, \dots, n\}$

Based on equation 4, it can be seen that compressed blocks are divided into block headers and block content. The block header consists of the byte length of the sub-block and the base value. Block content consists of sub-blocks containing delta bytes of data relative to the base. A block will be compressed if $\forall_i, \max(size(\Delta_i)) < k$, otherwise the block will not be compressed.

3. METHOD

In this paper, we propose a compression mechanism that adopts the concept of Base+Delta compression with the addition of dynamic block size features and optimization of block headers to optimize block output for both compressed and uncompressed blocks. As can be seen in the Base+Delta compression mechanism, the block size used is fixed according to the size of the cache memory. In file compression applications, fixed-size blocks can reduce compression potential if the file contains many byte sequences with low variance. In file compression applications, each block requires additional information in the form of a block header, both in the compressed block and in the uncompressed block, which will later be used by the decoder. With a fixed block size, uncompressed blocks will incur overhead due to the additional information needed by the decoder in the block. The compression mechanism we propose aims to minimize this overhead and optimize the compression potential, especially for files in general where the duplication and dynamics of the bytes contained in the data varies from one file to another.

* Corresponding author

Journal of Computer Networks, Architecture and High Performance Computing
Volume 5, Number 1, January 2023
https://doi.org/10.47709/cnahpc.v5i1.1993

Submitted : January 12, 2023
Accepted : February 18, 2023
Published: February 19, 2023

Block

The compression mechanism proposed in this study divides the data bytes into blocks which will then be processed into the compression function. Each C block consists of a header block and a content block. The blocks used in this mechanism are divided into compressed blocks and uncompressed blocks with different block headers and block content. To simplify the process, this mechanism does not use the sub-block concept where each byte will be processed separately from other bytes, although it does not rule out the possibility of further modification by implementing sub-blocks where each byte in a block can be combined into sub-blocks to form words. By using k = 1 byte, the output for a compressed block can be denoted as follows:

$$\{f, l, dl, B^*, \Delta = (\Delta_1, \Delta_2, \dots, \Delta_n)\} \quad (5)$$

Where f is the flag information bit, l is the block length and dl are the max length of the delta bit which will be discussed in the block header section. While the output for uncompressed blocks can be denoted as follows:

$$\{f, t, v = (v_1, v_2, \dots, v_n)\} \quad (6)$$

Where t is additional information for the uncompressed block.

Block Header

The block header for each block contains different information between compressed blocks and uncompressed blocks. However, each block starts with a single bit as an identifier or flag whether the block is a compressed block or an uncompressed block. In a compressed block, the flag bit will then be followed by bits that represent the length of the block (l). Due to the dynamic block length, each block must include block length information as a delimiter with the next block. The bit length that represents the block length can vary from 4 bits to 1 byte. The bit length of this information can be determined early in the compression process and uniformized for all compressed blocks so that it acts as the maximum block length for all compressed blocks. The bit length can also be obtained from the sum of the maximum block lengths of all compressed blocks.

$$l = \max\big(length(C_i)\big), \forall_i \in \{1, \dots, n\} \quad (7)$$

If applied globally to each compressed block, the bits length of l must be included at the beginning of the compressed file. dl is additional information that tells the decoder how many bits each delta value is

.

$$\mathrm{dl} = \max\big(size(\Delta_i)\big), \forall_i \in \{1, \dots, n\} \quad (8)$$

For k = 1 byte, then dl = 3 bits even though in a compressed block not all the bits of dl will be used to represent the bit length of the delta. In an uncompressed block, in addition to the 0 bit flag as the identifier of the uncompressed block, t is additional information that has two possibilities depending on the length of the uncompressed block ul, while the conditions that determine additional information from t are as follows:

1. If $(ul + 1) < \max\big(length(C_i)\big), \forall_i \in \{1, \dots, n\}$, then t is not used but any raw data contained in the uncompressed block will start with bit 0.
2. If $(ul + 1) \geq \max\big(length(C_i)\big), \forall_i \in \{1, \dots, n\}$, then t will contain bit 1 followed by ul.

As with dl, the ul bit length can be uniform for all blocks.

Compression

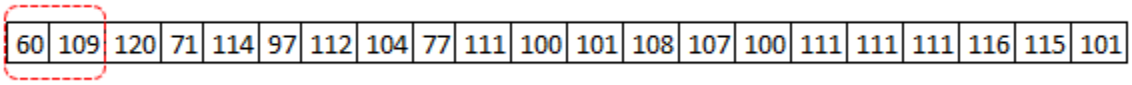
The compression process is carried out by using a byte wise sliding window to form a candidate block. Sliding window sizes start with a length of 2–byte, 4–byte or 8-byte. A base value is then selected from the bytes contained in the sliding window, the minimum value contained in the sliding window will be used as the base to avoid negative delta.

* Corresponding author

Journal of Computer Networks, Architecture and High Performance Computing
Volume 5, Number 1, January 2023
https://doi.org/10.47709/cnahpc.v5i1.1993

Submitted : January 12, 2023
Accepted : February 18, 2023
Published: February 19, 2023

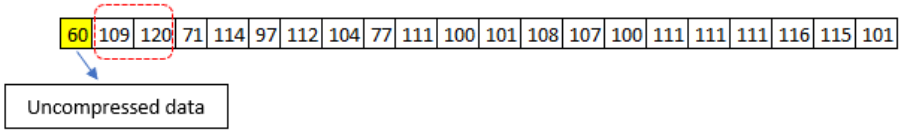
$$B^* = min(v_1, v_2, \dots, v_n) \quad (9)$$



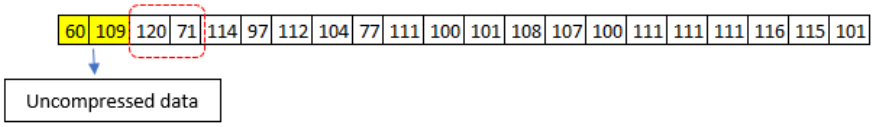
From the sliding window above we get Δ= (0,49), with dl = $\max(size(\Delta_i)) = 6\ bits$. Furthermore, the compressed length of the current sliding window is then calculated using the following equation.

$$CompressedLength = (dl * length(slidingwindow)) + length(B^*) \quad (10)$$

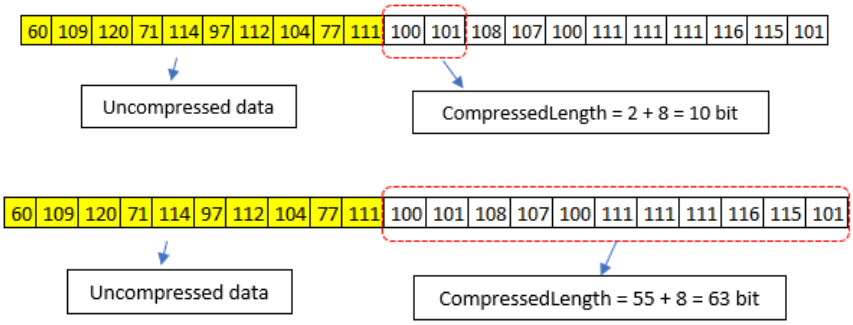
If the Compressed Length is less than the original length, the sliding window size will be enlarged by 1 byte. In the example above, we get the compressed length of 20 bits which is bigger than the original length of 16 bits. Then the sliding window will shift by 1 byte and the previous 1 byte will be included in the uncompressed block candidate.



In the next sliding window, obtained Δ= (0,11), with dl = $\max(size(\Delta_i)) = 4\ bits$, obtained Compressed Length of 16 bits which is the same length as the original length so that the sliding window will shift by 1 byte and the previous 1 byte will be marked uncompressed.



When the Compressed Length condition is less than the original length, the sliding window size will be enlarged by one byte until the Compressed Length condition is greater than or equal to the original length.



From the above process, a compressed block is obtained with the following bit sequence:

f	*l*	*dl*	*B*	*Content*
1	1011	101	01100100	00000 00001 01000 00111 00000 01011 01011 10000 01111 00001

Then we will get 71 bit compressed block of 88 bit original bytes. For uncompressed blocks, the structure will be arranged as follows:

f	*t*	*Content*

* Corresponding author

Journal of Computer Networks, Architecture and High Performance Computing
Volume 5, Number 1, January 2023
https://doi.org/10.47709/cnahpc.v5i1.1993

Submitted : January 12, 2023
Accepted : February 18, 2023
Published: February 19, 2023

0	1 1010	60 109 120 71 114 97 112 104 77 111

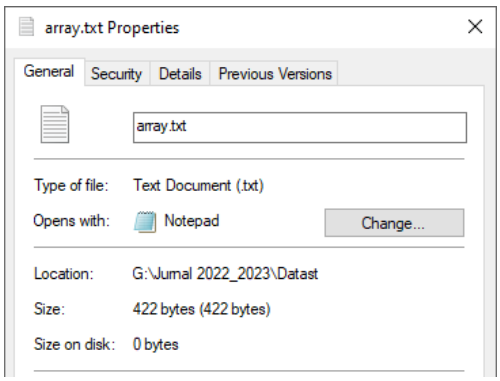
Then in total we will get a compressed file of 71 + 86 = 157 bits from the original 168 bit file. Bit 1 at the beginning of t indicates that this uncompressed block uses a block length that can be obtained from the next 4 bits or 1 byte because using 4 extra bits as a block length is more efficient than providing flag bits for each byte contained in the uncompressed block.

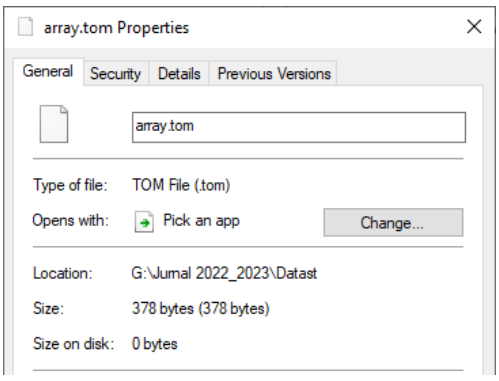
4. RESULT

Delta compression works well with text where the difference between bytes is smaller than for other types of content. In this study, we tested the modification mechanism of Base+Delta compression on various file formats to see the performance of the developed compression model. The main goal of developing this compression model is that it can be applied to all file formats so that it can be applied in general. The test dataset consists of several file formats which are divided into several categories such as images, text, executable, document files, and compressed files. The dataset used was obtained from various sources that can be accessed online.

The test was carried out using a compression application that implemented the compression model proposed in this study. During the compression process, information about compressed blocks and uncompressed blocks will be recorded as information for the performance analysis needs of the developed compression model. In figures 1 and 2 can be seen from testing the compression process in one of the test files.

Fig 1. Compression Test





* Corresponding author

Journal of Computer Networks, Architecture and High Performance Computing
Volume 5, Number 1, January 2023
https://doi.org/10.47709/cnahpc.v5i1.1993

Submitted : January 12, 2023
Accepted : February 18, 2023
Published: February 19, 2023

Fig 2. Compression Test Result

Compression testing on image files using several different images. The test results can be seen in the table below.

Table 1
Images Files Compression Test

File	Original Size	Compressed Size	Ratio
Image1.jpg	5,782	5,731	0.88
Image2.png	3,815	3,830	-0.39
Image3.jpg	675	594	12
Image4.jpg	32,775	28,748	12.3
Image5.png	27,877	27,865	0.04
Image6.jpg	45,448	41,485	8.72
Image7.png	1,429	1,435	-0.42

Text testing is carried out using several file formats that store data in text formats such as txt, sql, log, source codes, etc. The testing of text files can be seen in the following table.

Table 2
Text Files Compression Test

File	Original Size	Compressed Size	Ratio
Text1.txt	423	379	10.4
Text2.csv	409	380	7.1
Text3.sql	70,642	63,095	10.68
Text4.ris	434	389	10.37
Text5.md	2,725	2,462	9.65
Text6.dist	2,416	2,168	10.26
Text7.html	79,082	72,087	8.84

Document file testing uses several files with commonly used document format files such as word doc, excel, ppt etc. The testing of files with document format can be seen in the following table.

Table 3
Document Files Compression Test

File	Original Size	Compressed Size	Ratio
Doc1.docx	21,807	20,399	6.46
Doc2.pdf	11,158	10,866	2.62
Doc3.xlsx	15,519	14,046	9.49
Doc4.pptx	187,961	186,747	0.64
Doc5.docx	33,075	31,713	4.12
Doc6.pdf	99,858	97,691	2.17
Doc7.pdf	118,743	116,588	1.81

The last test was carried out using several different files such as executable and compressed files. The test results can be seen in the following table.

Table 4
Other Files Compression Test

File	Original Size	Compressed Size	Ratio
File1.zip	18,491	18,484	0.04
File2.zip	6,795	6,193	8.86

* Corresponding author

Journal of Computer Networks, Architecture and High Performance Computing
Volume 5, Number 1, January 2023
https://doi.org/10.47709/cnahpc.v5i1.1993

Submitted : January 12, 2023
Accepted : February 18, 2023
Published: February 19, 2023

File3.exe	145,409	143,571	1.26
File4.exe	25,601	23,838	6.89
File5.dll	457,217	444,015	2.89
File6.tgz	145,310	146,018	-0.48
File7.zip	61,789	61,731	0.09

DISCUSSIONS

Based on the results of tests that have been carried out on various file formats, it can be seen that the compression model developed in this study can generally compress various file types. Even so, compression ratios obtained especially in modern file – file formats such as document and specialty files such as compressed files and executables produce a relatively low compression ratio where one of the test results is on the file type it can be seen in figure 3 and 4.

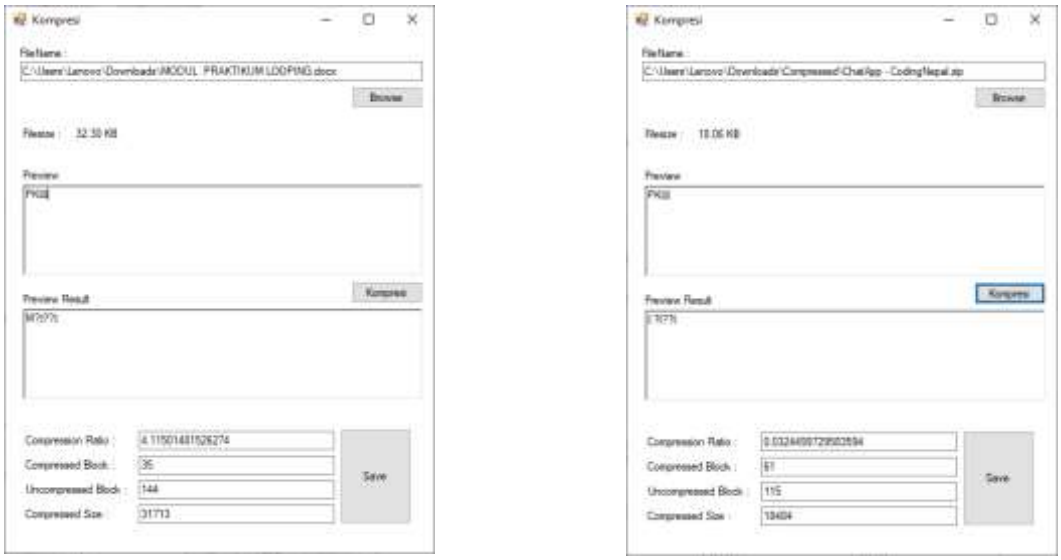
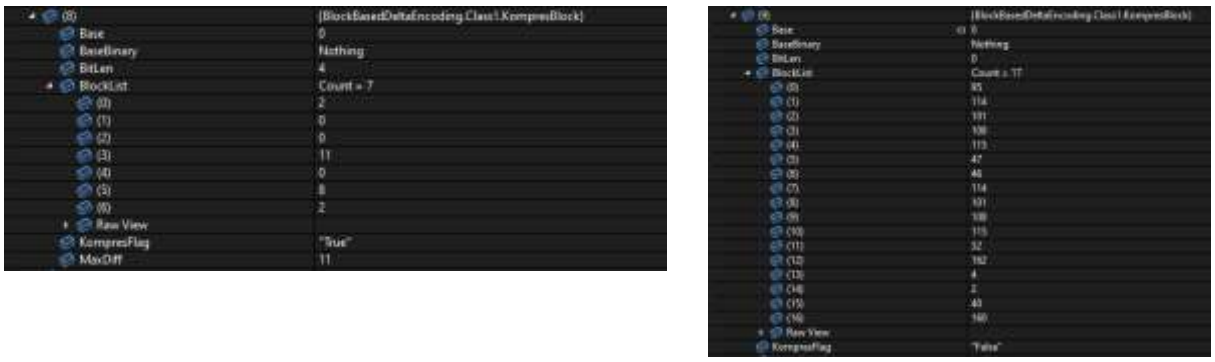


Fig 3. Compression Test on Document and Compressed Files

The low compression ratio on special files such as document and compressed file is due to the low number of compressed blocks formed compared to the number of uncompressed blocks. Higher uncompressed block shows the high variant of the difference between adjacent bytes contained in the file, where in figure 3 we can see on the docx file the number of compressed blocks is much smaller compared to uncompressed block (35 : 144).



* Corresponding author

Journal of Computer Networks, Architecture and High Performance Computing
Volume 5, Number 1, January 2023
https://doi.org/10.47709/cnahpc.v5i1.1993

Submitted : January 12, 2023
Accepted : February 18, 2023
Published: February 19, 2023

Fig 4. Compressed and Uncompressed Block Sample of Document Test

Good compression ratio is obtained on files with low difference variants between bytes such as in text files and some image formats as can be seen in figure 5 where the number of compressed blocks is usually larger compared to the number of uncompressed blocks so that it can produce better compression ratio.

Fig 5. Compression ratio on text files

Although in most of the tests the number of compressed blocks and uncompressed blocks determines the low and high compression ratios obtained. It is also possible to obtain a low compression ratio in compressed files with the number of compressed blocks and uncompressed blocks that are not that much different, as can be seen in the zip file test in Figure 3.

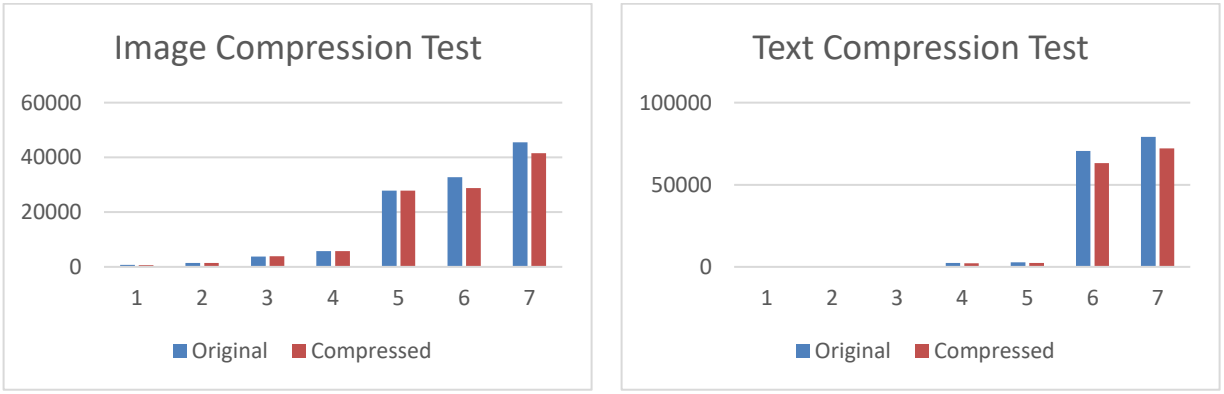


Fig. 6 Image and Text Files Size Comparison Chart

* Corresponding author

Journal of Computer Networks, Architecture and High Performance Computing
Volume 5, Number 1, January 2023
https://doi.org/10.47709/cnahpc.v5i1.1993

Submitted : January 12, 2023
Accepted : February 18, 2023
Published: February 19, 2023

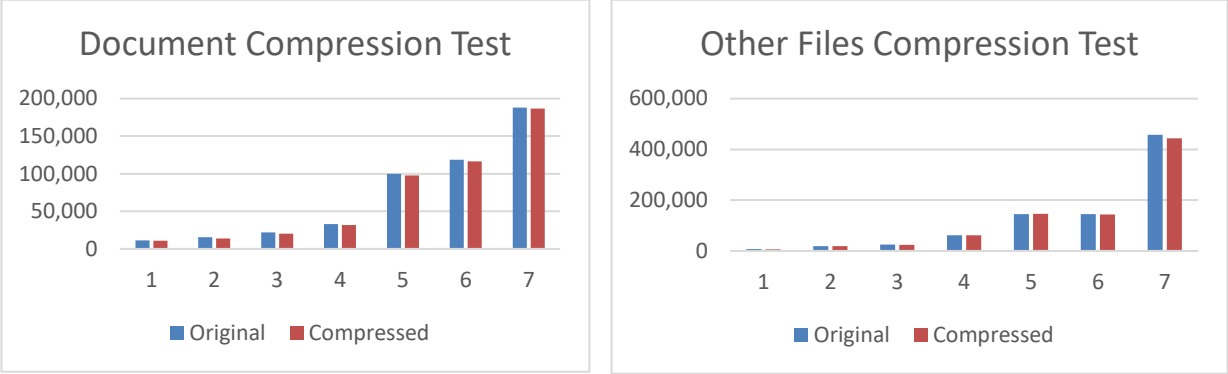


Fig. 7 Document and Other Files Size Comparison Chart

From the visualization of the comparison of the original size with the compressed file as shown in Figure 6 and 7, the compressed result gives good compression results at a larger file size. In text files, the difference in file size gives better compression results on files with larger sizes. However, in terms of ratio, there is no significant difference between compressed files of different sizes, especially in file formats other than text, where the compression ratio is very dependent on the variant bytes of data that the file has.

5. CONCLUSION

Base + Delta compression is developed with fixed sized blocks intended to apply compression to cache memory lines. This research develops a compression model by adopting the concept of Base compression + Delta compression. Modifications and additions are made by applying a dynamic block size by using a sliding window to determine the data bytes and compression block length. Dynamic block length makes it possible to increase compression efficiency where the fewer the number of blocks used, the smaller the overhead of additional information in the block. Dynamic block sizes also increase the compression potential of files that have a low variance byte order. Optimization is also carried out on uncompressed blocks to minimize the weaknesses of block-based compression which are usually burdened by additional bits of information. The test results show quite promising results where almost all file formats can be compressed properly where the highest compression ratio obtained is 12.3 and the lowest ratio is 0.04. Although compared to other compression models, the compression ratio obtained is relatively low, the compression model obtained can be applied to almost all file formats and has room for further development and applications.

6. REFERENCES

Banga, G., Douglis, F., & Rabinovich, M. (1997). Optimistic deltas for WWW latency reduction. In Proc. 1997 USENIX Technical Conference, Anaheim, CA, (pp. 289-303).

Cogo, V., Paulo, J., & Bessani, A. (2020). Genodedup: Similarity-based deduplication and delta-encoding for genome sequencing data. IEEE Transactions on Computers, 70(5), 669-681.

Dolgorsuren, B., Khan, K., Rasel, M., & Lee, Y. (2019). StarZIP: streaming graph compression technique for data archiving. IEEE Access, 7, 38020-38034.

Engelson, V., Fritzson, P., & Fritzson, D. (2000). Lossless compression of high-volume numerical data from simulations. Linkoping: Linkoping University Electronic Press.

Hanumanthaiah, A., Gopinath, A., Arun, C., Hariharan,, B., & Murugan, R. (2019). 2019. In 2019 9th International Symposium on Embedded Computing and System Design (ISED) (pp. 1-5). IEEE.

* Corresponding author

Journal of Computer Networks, Architecture and High Performance Computing
Volume 5, Number 1, January 2023
https://doi.org/10.47709/cnahpc.v5i1.1993

Submitted : January 12, 2023
Accepted : February 18, 2023
Published: February 19, 2023

Henziger, E., & Carlsson, N. (2019). Delta encoding overhead analysis of cloud storage systems using client-side encryption. In 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), (pp. 183-190).

Housel, B., & Lindquist, D. (1996). WebExpress: A system for optimizing Web browsing in a wireless environment. In Proceedings of the 2nd annual international conference on Mobile computing and networking, (pp. 108-116).

Hunt, J., & MacIlroy, M. (1976). An algorithm for differential file comparison. Murray Hill: Bell Laboratories.

Hunt, J., Vo, K., & Tichy, W. (1998). Delta algorithms: An empirical analysis. ACM Transactions on Software Engineering and Methodology (TOSEM), 7(2), 192-214.

Italiano, G., Prezza, N., Sinaimeri, B., & Venturini, R. (2021). Compressed weighted de Bruijn graphs. 32nd Annual Symposium on Combinatorial Pattern Matching, 191(16), 1-16.

Jahani, E., Cafarella, M., & Ré, C. (2011). Automatic optimization for MapReduce programs. arXiv preprint arXiv:1104.3217.

Korn, D., Macdonald, J., Mogul, J., & Vo, K. (2002). RFC3284: The VCDIFF Generic Differencing and Compression Data Format.

MacDonald, J. (2000). File system support for delta compression. (Doctoral dissertation, Masters thesis. Department of Electrical Engineering and Computer Science, University of California at Berkeley).

Mogul, J., Douglis, F., Feldmann, A., & Krishnamurthy, B. (1997). Potential benefits of delta encoding and data compression for HTTP. In Proceedings of the ACM SIGCOMM'97 conference on Applications, technologies, architectures, and protocols for computer communication, (pp. 181-194).

Mogul, J., Krishnamurthy, B., Douglis, F., Feldmann, A., Goland, Y., van Hoff, A., & Hellerstein, D. (2002). Delta encoding in HTTP. No. rfc3229.

Pekhimenko, G., Guo, C., Jeon, M., Huang, P., & Zhou, L. (2018). {TerseCades}: Efficient Data Compression in Stream Processing. In 2018 USENIX Annual Technical Conference (USENIX ATC 18), (pp. 307-320).

Pekhimenko, G., Seshadri, V., Mutlu, O., Gibbons, P., Kozuch, M., & Mowry, T. (2012). Base-delta-immediate compression: Practical data compression for on-chip caches. In Proceedings of the 21st international conference on Parallel architectures and compilation techniques, (pp. 377-388).

Samteladze, N., & Christensen, K. (2012). DELTA: Delta encoding for less traffic for apps. In 37th Annual IEEE Conference on Local Computer Networks (pp. 212-215). IEEE.

Suel, T. (2019). Delta compression techniques. Encyclopedia of Big Data Technologies, 63.

Tan, H., Zhang, Z., Zou, X., Liao, Q., & Xia, W. (2020). Exploring the Potential of Fast Delta Encoding: Marching to a Higher Compression Ratio. In 2020 IEEE International Conference on Cluster Computing (CLUSTER) (pp. 198-208). IEEE.

Trendafilov, D., Memon, N., & Suel, T. (2002). Zdelta: An efficient delta compression tool. Technical report, Department of Computer and Information Science at Polytechnic University.

Vestergaard, R., Zhang, Q., & Lucani, D. (2019). Lossless compression of time series data with generalized deduplication. In 2019 IEEE Global Communications Conference (GLOBECOM) (pp. 1-6). IEEE.

Xia, W., Jiang, H., Feng, D., Tian, L., Fu, M., & Zhou, Y. (2014). Ddelta: A deduplication-inspired fast delta compression approach. Performance Evaluation, 79, 258-272.

* Corresponding author

Journal of Computer Networks, Architecture and High Performance Computing
Volume 5, Number 1, January 2023
https://doi.org/10.47709/cnahpc.v5i1.1993

Submitted : January 12, 2023
Accepted : February 18, 2023
Published: February 19, 2023

Xia, W., Li, C., Jiang, H., Feng, D., Hua, Y., Qin, L., & Zhang, Y. (2015). Edelta: A {Word-Enlarging} Based Fast Delta Compression Approach. In 7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 15).

Zhang, Y., Jiang, H., Shi, M., Wang, C., Jiang, N., & Wu, X. (2021). A High-performance Post-deduplication Delta Compression Scheme for Packed Datasets. In 2021 IEEE 39th International Conference on Computer Design (ICCD) (pp. 464-471). IEEE.

Zhang, Y., Yuan, Y., Feng, D., Wang, C., Wu, X., Yan, L., . . . Wang, S. (2020). Improving restore performance for in-line backup system combining deduplication and delta compression. IEEE Transactions on Parallel and Distributed Systems, 31(10), 2302-2314.

* Corresponding author