The Impact of Low-Entropy on Chunking Techniques for Data Deduplication

Mu'men Al Jarah
University of Waterloo
Waterloo, Canada
mumen.aljarah@uwaterloo.ca

Sreeharsha Udayashankar
University of Waterloo
Waterloo, Canada
s2udayas@uwaterloo.ca

Abdelrahman Baba
University of Waterloo
Waterloo, Canada
ababa@uwaterloo.ca

Samer Al-Kiswany
University of Waterloo
Waterloo, Canada
alkiswany@uwaterloo.ca

***Abstract*—While numerous Content-Defined Chunking (CDC) algorithms exist for data deduplication, their relative performance has not been analyzed in the presence of low-entropy induced byte-shifting. This paper explores and evaluates hash-based and hashless CDC algorithms in the presence of low-entropy data regions, using synthetic datasets. Our evaluation shows that modern CDC algorithms are poor at handling low-entropy blocks when the block sizes are small and that their low-entropy detection ability depends upon the expected average chunk size.**

Contrary to previous studies focusing on conventional byte-shifting, hash-based algorithms achieve poor space savings compared to their hashless counterparts when low-entropy induced byte-shifting is involved. This can be explained by the greater variability in chunk sizes and the higher percentage of artificial boundaries they exhibit in the presence of these regions. All of these factors together highlight the need for specialized CDC algorithms to detect and eliminate low-entropy data blocks during the deduplication process.

***Index Terms*—Data Deduplication, Content-Defined Chunking, Low-Entropy, Storage Systems, Cloud Computing**

I. Introduction

The amount of data in the digital world is continuously growing; the global data production in 2020 was 64.2 zettabytes and is expected to exceed 180 zettabytes by 2025 [1]. Cloud-storage providers utilize numerous mechanisms to cope with this data influx, such as distributed file systems [2], [3], data compression [4] and data deduplication [5], [6]. Previous studies have shown that a large amount of data stored on the cloud is redundant [7]. Data deduplication reduces storage requirements by identifying these redundant copies and avoiding their storage and transmission, conserving both storage space and network bandwidth [7]–[9]. Data deduplication consists of multiple phases [5], the first of which is file chunking. File chunking divides the source data into smaller chunks before deduplicating them, as chunk-level deduplication is more effective than file-level deduplication [8].

Content-Defined Chunking (CDC) algorithms are used to perform file chunking. Numerous chunking algorithms [10]–[13] exist in current literature, each offering different levels of deduplication space savings and throughput. There are two categories of CDC algorithms; hash-based and hashless algorithms. Hash-based algorithms, such as Rabin's chunking [13] and FastCDC [11], use a rolling hash function to determine chunk boundaries while their hashless counterparts, such as Asymmetric Extremum (AE) [10] and Rapid Asymmetric Maximum (RAM) [12], utilize local minima / maxima.

While studies in the past [9], [14] have analyzed the performance of CDC algorithms in datasets with conventional byte-shifting, they do not examine their performance in the presence of *low-entropy data blocks*. Low-entropy blocks contain long sequences of a single repeated byte or pattern. Our analysis shows that low-entropy blocks are prevalent in real-world datasets, amounting to 10-30% of the total size depending upon the dataset characteristics (§III). Efficient chunking algorithms should be able to handle these low-entropy regions by isolating and eliminating them, achieving high deduplication ratios.

In this paper, we examine four state-of-the-art CDC algorithms, evaluating their performance and effectiveness on datasets with low-entropy regions. We compare their performance under scenarios with conventional and low-entropy induced byte-shifting. We evaluate each algorithm using four metrics: space savings achieved, chunking throughput, chunk size variance and artificial boundary percentage.

Our analysis shows that the ability of modern CDC algorithms to detect and eliminate low-entropy blocks is poor and dependent upon the *block size* and the *expected average chunk size*. Modern CDC algorithms cannot detect and eliminate low-entropy blocks of small sizes, resulting in poor space savings. This highlights the need for chunking techniques specialized in low-entropy elimination. At large block sizes, contrary to previous studies focusing on datasets with conventional byte-shifting [11], we find that AE [10] is better at deduplicating data with high levels of low-entropy induced byte-shifting. This is because hashless algorithms such as AE [10] and RAM [12] possess lower chunk size variance and artificial boundary percentages when handling such data (§V).

The rest of the paper is organized as follows: §II provides necessary background and §III motivates our work. §IV describes our methodology and provides details about the synthetic datasets we use. §V presents the evaluation of the state-of-the-art CDC algorithms. §VI introduces additional related work and §VII concludes our discussion.

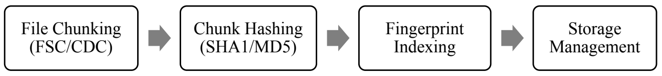


Fig. 1: Data deduplication workflow.

II. Background

In this section, we discuss relevant background information about deduplication, chunking algorithms and low-entropy data regions.

A. Data Deduplication

Data deduplication [6] helps optimize storage efficiency within large-scale digital storage systems by eliminating redundant data. The major steps involved in the deduplication process are shown in Fig. 1.

i. File Chunking: Files are divided into smaller parts called chunks using *chunking algorithms*.
ii. Chunk Hashing: Chunks are hashed using a collision-resistant hashing function (such as SHA-1 [15]) to generate unique *fingerprints*.
iii. Fingerprint Indexing: Fingerprints are compared against an existing database of previously observed fingerprints. A duplicate fingerprint indicates a duplicate chunk, which can be safely eliminated.
iv. Data Storage: Non-duplicate chunks are stored and their fingerprints are added to the fingerprint database.

File Chunking. File chunking is the first step in the deduplication pipeline and is a critical stage in the deduplication process [8]. Files can be sliced into either fixed-size or variable-size chunks. In fixed-size chunking, files are divided into chunks with equal predefined sizes. While this approach is simple, it achieves poor space savings due to the boundary-shift problem [13]. The boundary-shift problem causes all the chunk boundaries to shift when bytes are added to the data, resulting in completely new chunks and poor space savings.

On the other hand, deduplication systems in production use Content-Defined Chunking (CDC) algorithms to divide files into chunks based on data characteristics, effectively solving the boundary-shift problem. However, as this requires scanning each file in its entirety, it is slower than fixed-size chunking. Chunking throughput represents the speed at which a CDC algorithm can divide source data into chunks.

Chunking algorithms impact deduplication efficiency as well. Tuning chunking algorithms to generate larger chunks can reduce indexing and metadata overheads while smaller chunk sizes can enhance the space savings achieved by deduplication.

CDC Algorithms. Content-Defined Chunking (CDC) algorithms work by sliding a window across the source data. When the data within the window matches certain target characteristics, they insert chunk boundaries to divide the data into chunks. CDC algorithms can be classified into hash-based and hashless algorithms. All chunking algorithms insert artificial chunk boundaries when content-defined boundaries cannot be found before a maximum chunk size is reached [11], [13].

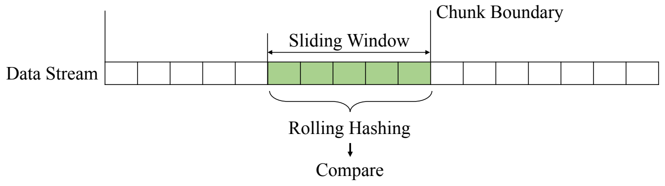


Fig. 2: Hash-based CDC algorithms.

As shown in Figure 2, hash-based algorithms slide a fixed-size window over the source data stream, one byte at a time. With each slide, the hash value of the data within the window is calculated using a rolling hash algorithm. Whenever this hash value matches a target value, a chunk boundary is inserted. Examples of hash-based algorithms include Rabin's chunking [13] and FastCDC [11]. These algorithms insert a chunk boundary when the lower *n* bits of the window's rolling hash value equal zero. We note that the hashing algorithms used here differ from the collision-resistant algorithms used for fingerprint indexing.

On the other hand, hashless algorithms treat each byte within the window as a separate value, and insert chunk boundaries based on local minima / maxima. Figure 3 shows the operational procedure of Asymmetric Extremum (AE) [10]. AE slides a fixed-size window over the data and inserts chunk boundaries when the target byte i.e. byte at the head of the window is a local maxima / minima. Figure 4 shows the operational procedure of RAM [12]. RAM also slides a fixed-size window over the data, inserting chunk boundaries when the byte immediately outside the window (Target Byte in Figure 4) is at least as large as the maximum valued byte within.

B. Data Entropy

Entropy, borrowed from information theory [16], measures the randomness or variance within data. Claude Shannon's work [17] proposed foundational steps for the understanding of how entropy influences the processing and transmission of data. We use this information to identify *low-entropy* data regions. Low-entropy regions are data segments characterized

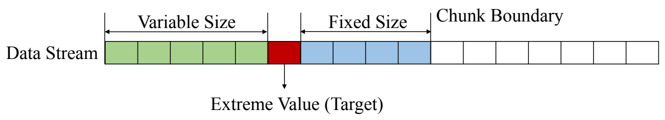


Fig. 3: Asymmetric Extremum (AE).

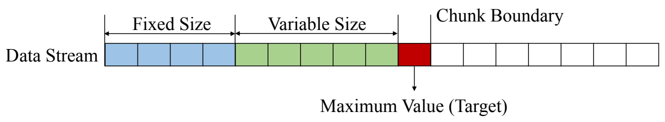


Fig. 4: Rapid Asymmetric Maximum (RAM).

TABLE I: Low-entropy data regions in real datasets.

Dataset	Size	Description	LE %
VDI	122 GB	Virtual Desktop Infrastructure images [20]	27.43%
LNX	65 GB	160 Linux kernel distributions in TAR format [21]	9.87%

by high predictability and redundancy, like repeated bytes or patterns [18].

These regions play a key role in data deduplication; as these regions have low variability, they are prime candidates for elimination. While limited efforts have been previously made to study their impact on deduplication [10], their effect on CDC algorithms has not been extensively studied.

III. Motivation

Low-entropy data regions are a common occurrence in many real datasets. For instance, virtual machine disk images (VDI) have a large amount of low-entropy. as empty regions are represented by a single repeated byte value (such as '0'). Table I shows the percentage of low-entropy regions in two diverse datasets. Note that a minimum of 10 adjacent occurrences of a byte are required for a region to be classified as low-entropy in the table.

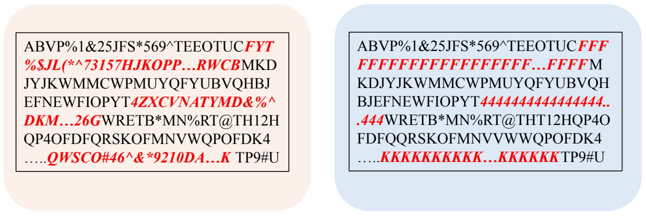
Comparing CDC Algorithms. A plethora of CDC algorithms exist within previous literature [10]–[13], [19]. Previous studies [9], [14] have compared these algorithms using evaluation metrics such as space savings and chunking throughput. However, none of the studies analyze the impact of low-entropy regions on CDC algorithms, instead focusing on datasets with conventional byte-shifting. As low-entropy regions are prevalent in real datasets, the motivating factor behind our study is to quantify their impact on CDC algorithms and compare it to the impact of conventional byte-shifting.

IV. Study Design

As shown in §III, low-entropy data regions are common in real datasets. The primary objective of our paper is to comprehensively study the impact of low-entropy data regions on data deduplication. In order to assess the impact of byte shifting caused by low-entropy regions (i.e. repeated byte patterns) and compare it against conventional byte shifting (i.e. random bytes), we generate synthetic datasets containing equal amount of low-entropy and conventional byte shifts (§IV-A). We compare the performance of modern CDC algorithms on these datasets using the metrics discussed in §IV-B.

A. Synthetic Datasets

To compare the impact of byte shifting caused by low-entropy regions to conventional byte shifting, we generate two kinds of synthetic datasets. We generate a base dataset of multiple files filled with uniformly random bytes. We then insert blocks of data at uniformly random locations within these files, to mimic data insertion in real datasets [22]. Within the first dataset, inserted data blocks contain random bytes to represent conventional byte shifting. For the second dataset,


ABVP%1&25JFS*569^TEEOTUCFYT
%$JL(*^73157HJKOPP...RWCBMKD
JYJKWMMCWPMUYQFYUBVQHBJ
EFNEWFIOPYT4ZXCVNATYMD&%^
DKM...26GWRETB*MN%RT@TH12H
QP4OFDFQRSKOFMNVWQPOFDK4
.....QWSCO#46^&*9210DA...K TP9#U
ABVP%1&25JFS*569^TEEOTUCFFF
FFFFFFFFFFFFFFFFF...FFFFM
KDJYJKWMMCWPMUYQFYUBVQH
BJEFNEWFIOPYT4444444444444444..
.444WRETB*MN%RT@THT12HQP4O
FDFQQRSKOFMNVVWWQPOFDK4
.....KKKKKKKKKK...KKKKKKTP9#U

(a) Conventional (b) Low-Entropy Induced

Fig. 5: Datasets with byte-shifting.

inserted blocks contain repeated byte sequences and patterns to represent byte-shifting caused by low-entropy data regions. Figure 5 shows an example of the conventional and low-entropy byte-shifted datasets arising from the same base data.

We vary the following parameters to generate conventional and low-entropy byte-shifted datasets possessing different characteristics:

1) **Block size:** Byte-shifting is caused by *blocks* of data inserted within the original data. We vary the size of these inserted blocks and examine their impact on CDC algorithms.
2) **Byte-shift Level:** We define the ratio of byte-shifting (conventional or low-entropy induced) as the *byte-shift level* within the data. The formula we use to calculate byte-shift level is:

$$\text{Byte-shift Level} = \frac{\text{Inserted Data Size}}{\text{Total Data Size}} \tag{1}$$

We choose to generate synthetic datasets for our study as it is difficult to achieve fine-grained control over these characteristics in real world datasets.

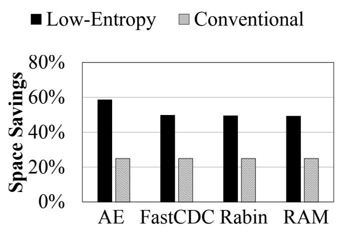
B. Evaluation Metrics

Modern CDC algorithms can be tuned to generate chunks with different average sizes. We experiment with configurations that generate average chunk sizes of 4-16 KB. We quantify the impact of low-entropy regions on these algorithms using the following metrics:

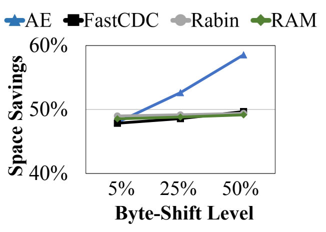
1) **Space Savings:** The space savings achieved by using the CDC algorithm within a deduplication system (§II-A).

$$\text{Space Savings} = \frac{\text{Original Size} - \text{Deduplicated Size}}{\text{Original Size}} \tag{2}$$

2) **Chunking Throughput:** The rate at which the CDC algorithm can divide source data into chunks.
3) **Chunk Size Distribution:** Frequency distribution of chunk sizes. Higher chunk size variance is typically associated with lower deduplication efficiency [9], [14].
4) **Artificial Boundary Percentage:** The percentage of artificial chunk boundaries (§II-A) inserted by the CDC algorithm.



(a) Conventional vs low-entropy induced byte-shifting.



(b) Impact of low-entropy induced byte-shift levels.

Fig. 6: Space savings comparison.

V. Evaluation and Insights

We compare four state-of-the-art CDC algorithms. We evaluate their space savings, chunking throughput, chunk size variance, and artificial boundary percentages (§IV). For each metric, we compare low-entropy induced byte shifting to conventional byte-shifting and also examine the impact of byte-shift levels and block sizes.

Alternatives. We compare the following CDC algorithms:

- *AE:* The Asymmetric Extremum (AE) [10] algorithm. We use `AE-Max` i.e. the target byte is a local maximum.
- *FastCDC:* The rolling hash-based FastCDC [11] algorithm. We use a normalization level of 2.
- *Rabin:* Rabin's chunking algorithm [13].
- *RAM:* Rapid Asymmetric Extremum (RAM) [12].

We use minimum and maximum chunk sizes of 0.5× and 2× the average chunk size. Unless otherwise noted, we use an expected average chunk size of 16KB.

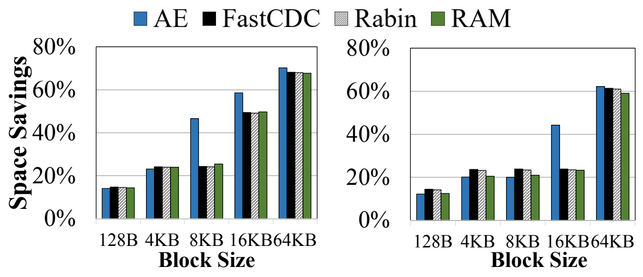
Implementation. All four CDC algorithms have been implemented in C++ using $150 - 300$ lines of code each and integrated into DedupBench [9]. We implemented the tool to generate synthetic datasets (§IV-A) using $\sim$ 50 lines of Python code. We use SHA-1 [15] as the collision resistant hashing function to generate the chunk fingerprints used in space savings calculations. We use the SHA-1 implementation provided by OpenSSL's `libssl` API [23]. Our code is publicly available on GitHub [24].

Testbed. We use a `c6525-25g` [25] machine from CloudLab Utah [26] for our experiments. The machine is an AMD EPYC Rome server containing a 16 core AMD 7302P with hyperthreading, 128GB of RAM and a 25 GBps Mellanox NIC. We use Ubuntu 22.04. We use gcc v11.4.0 and libssl v3.0.2 to deploy our code.

A. Deduplication Space Savings

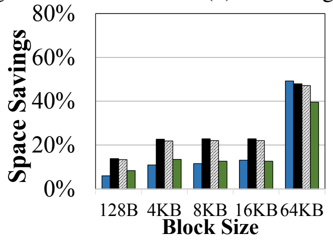
We evaluate the space savings achieved by the four CDC algorithms. We use an expected average chunk size of 4KB and a block size of 16KB for this experiment.

Low-entropy vs conventional byte-shifts. Figure 6a illustrates the differences in space savings achieved by CDC algorithms on synthetic datasets with equal percentages of conventional and low-entropy induced byte-shifts. *CDC algorithms achieve higher space savings in the presence of byte-shifts induced by low-entropy regions compared to conventional shifts.*



(a) 4KB Avg Chunk Size

(b) 8KB Avg Chunk Size



(c) 16KB Avg Chunk Size

Fig. 7: Impact of inserted block size on space savings of CDC algorithms.

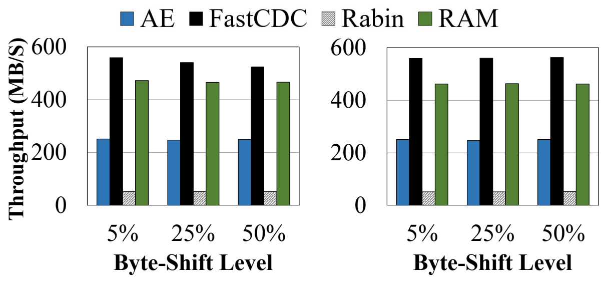
This is because the low-entropy data regions have repeating patterns and bytes, which can be detected and eliminated by isolating them into separate chunks. AE [10] achieves the highest space savings in the presence of low-entropy induced byte-shifts, performing slightly better than other CDC algorithms.

Byte-shift level. Figure 6b shows the space savings achieved by all CDC algorithms with increasing low-entropy induced byte-shift levels. At low byte-shift levels, FastCDC and Rabin achieve the highest space savings among CDC algorithms, similar to the results observed by previous studies on datasets with conventional byte-shifting [11].

However, as mentioned above, low-entropy data regions can be eliminated. When low-entropy induced byte-shift levels are increased, AE [10] is the only algorithm which identifies and efficiently eliminates these regions. This causes AE to achieve higher space savings with increasing low-entropy induced byte-shift levels, as shown in Figure 6b. Thus, in contrast to previous studies using conventional byte-shifting [11], *AE is the optimal CDC algorithm to handle high levels of low-entropy induced byte-shifts.*

Block size. CDC algorithms exhibit increases in space savings with increasing low-entropy block size i.e. they are better at handling large block sizes over smaller ones. Figure 7 shows the space savings achieved with increasing block sizes, for three different average chunk size configurations. The low-entropy induced byte-shift level was fixed at 50% for this experiment.

We note that in all three configurations, CDC algorithms identify and eliminate low-entropy regions more efficiently when the block sizes are higher. For instance, all algorithms



(a) Low-entropy induced (b) Conventional

Fig. 8: Chunking throughput comparison.

achieve a space savings of $\sim 60\%$ in Figure 7a with a block size of 64KB but only 20-25% with a block size of 4KB. Thus, *modern CDC algorithms are poor at eliminating low-entropy regions with small block sizes.*

We also note that CDC algorithms only start identifying and eliminating low-entropy regions when the block size is greater than or equal to the expected average chunk size. Once again, AE [10] can identify low-entropy regions at lower block sizes than other CDC algorithms. Thus, *the ability of CDC algorithms to eliminate low-entropy byte-shifts is dependent upon the expected average chunk size.*

Finally, we note an interesting trend when comparing hash-based and hashless algorithms with varying block size. At lower block sizes, FastCDC and Rabin's chunking achieve better space savings than AE and RAM. However, as the block size increases, this gap is quickly bridged. At higher block sizes, AE achieves the best space savings among all CDC algorithms. Thus, *the optimal CDC algorithm to eliminate low-entropy byte-shifts depends on the block size.*

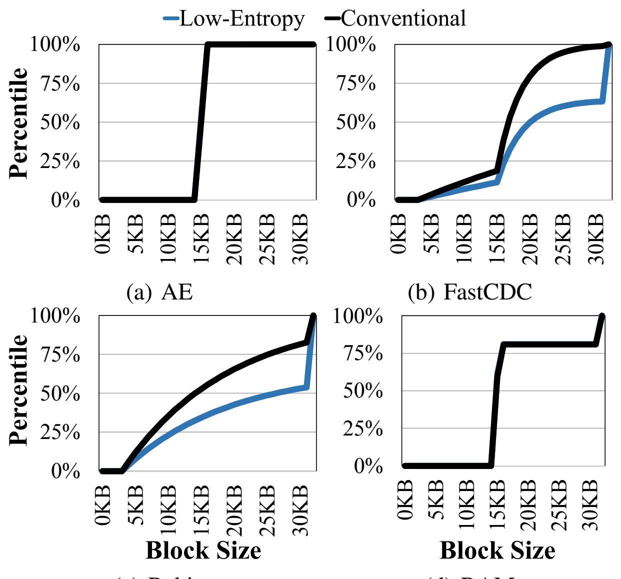
B. Chunking Throughput

Figure 8 shows the chunking throughput achieved by all algorithms with different levels of conventional and low-entropy induced byte shifting. The byte-shift level was set to 50% and the low-entropy block size was set to 16KB for these experiments. All our results are the average of 5 runs and the standard deviation was less than 5%.

Low-entropy vs conventional byte-shifts. All techniques maintain their throughputs between equal levels of low-entropy induced (Figure 8a) and conventional (Figure 8b) byte-shifting. FastCDC [11] achieves the highest chunking throughput among all the CDC algorithms, in line with previous studies using datasets with conventional byte-shifting [11].

Byte-shift level. With increasing levels of low-entropy induced byte-shifting (Figure 8a), FastCDC [11] exhibits slightly lower chunking throughputs. This is because it needs to scan larger amounts of data before finding chunk boundaries (§V-D). The throughputs of other CDC algorithms remain unaffected by byte-shift levels.

Block size. The throughputs of all CDC algorithms are unaffected by block size.



(a) AE (b) FastCDC (c) Rabin (d) RAM

Fig. 9: Chunk size distributions (CDF) under low-entropy induced and conventional byte-shifting.

C. Chunk Size Distribution

Within this section, we demonstrate the distributions in chunk sizes exhibited by all CDC algorithms under conventional and low-entropy induced byte-shifting. We use a byte-shift level of 50% and a 16KB block size for these experiments.

Low-entropy vs conventional byte-shifts. Figure 9 shows the chunk size distributions exhibited by all CDC algorithms under equal percentages of conventional and low-entropy induced byte-shifting.

Hashless CDC algorithms exhibit no differences in chunk size distributions between conventional and low-entropy induced byte-shifts. AE (Figure 9a) and RAM (Figure 9d) exhibit identical chunk size distributions in both cases.

Hash-based algorithms, such as FastCDC (Figure 9b) and Rabin's chunking (Figure 9c) on the other hand, exhibit differences in chunk sizes between low-entropy and conventionally induced byte-shifts. Both algorithms produce larger chunk sizes on average when low-entropy regions are involved. Their chunk size variance is higher as well, partially explaining their lower space savings in Figure 6a.

Byte-shift level and block size. While the chunk size distributions vary with byte-shift level and block size, the differences between hash-based and hashless algorithms remain the same as those outlined above. To ensure clarity, we only present the results for the block size and byte-shift level configuration above within the paper.

D. Artificial Chunk Boundaries

Modern CDC techniques insert artificial chunk boundaries when content-defined boundaries cannot be determined within a maximum size limit (§IV). A higher percentage of artificial

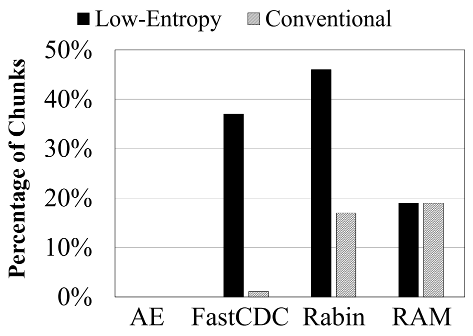


Fig. 10: Artificial chunk boundaries under conventional and low-entropy induced byte-shifting.

chunk boundaries results in lower deduplication space savings. Within this section, we examine the impact of low-entropy regions on artificially inserted boundaries. We use a byte-shift level of 50% and a block size of 16KB for this experiment.

Low-entropy vs conventional byte-shifts. Figure 10 shows the percentage of artificial chunk boundaries inserted by chunking techniques in the presence of conventional and low-entropy induced byte-shifts. *Hash-based CDC algorithms exhibit a higher percentage of artificially inserted chunk boundaries in the presence of low-entropy induced byte-shifting.*

This partially explains the space savings differences observed in Figure 6a. This occurs because the rolling-hash value fails to match the target value across the low-entropy regions, as they are composed of repeating byte-sequences. Hashless CDC algorithms do not suffer from the same drawback.

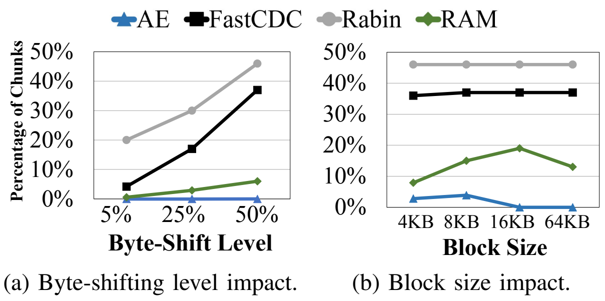
Byte-shift levels. With increasing low-entropy induced byte-shift levels, all CDC techniques exhibit higher percentages of artificial chunk boundaries, as seen in Figure 11a. Hash-based algorithms suffer in particular, exhibiting much higher artificial boundaries compared to their hashless counterparts.

Block size. Block size impacts CDC algorithms in differing ways. Hash-based algorithms once again suffer from much higher artificial chunk boundaries over their hashless counterparts regardless of block size.

VI. Related Work

Enhancing Deduplication Efficiency. There have been various efforts to increase the efficiency of the other phases within data deduplication. StoreGPU [27] uses GPUs for hashing, in order to improve throughput. SiLo [28] presents a scalable deduplication system with minimal RAM utilization. These are orthogonal to our work, as we target CDC algorithms for file chunking.

Data Compression. Huffman coding [29] is a popular entropy coding method, forming optimal prefix codes for data compression. It creates a binary tree sorted by frequency, to minimize byte redundancy by substituting input symbols with shorter codes. Lempel-Ziv compression [30] speeds up the data compression process by referencing the positions and lengths of redundant data. These methods are typically used in tandem with data deduplication, and are orthogonal to our work.



(a) Byte-shifting level impact. (b) Block size impact.

Fig. 11: Artificial Chunk Boundary Comparison.

Delta Compression. DARE [31] is an approach that is aware of deduplication and has low overhead for delta compression. It uses information about adjacent duplicates to make the process of resemblance detection and elimination efficient. Furthermore, Ddelta [32] is a fast deduplication-inspired delta compression method, improving the speed of both delta encoding and decoding through deduplication concepts while maintaining the same compression ratio. These are orthogonal to our work as we focus on data deduplication.

VII. Conclusion

While numerous studies exist targeting CDC algorithms, they only focus on datasets with conventional byte-shifting. Though low-entropy data blocks are prevalent in real-world datasets, not much attention has been devoted to analyzing their impact on CDC algorithms.

We present the first detailed analysis on the impact of low-entropy induced byte-shifting on modern CDC algorithms. Our analysis shows that modern CDC algorithms are poor at identifying and eliminating low-entropy data blocks with small block sizes. In addition, their ability to eliminate low-entropy blocks depends upon the expected average chunk size. Finally, the choice of optimal algorithm to handle low-entropy induced byte-shifting is highly dependent on the byte-shift level and block size. All of these together highlight the need for the design of novel CDC algorithms to detect and eliminate low-entropy blocks during the data deduplication process.

Acknowledgments

We would like to thank the anonymous reviewers for their detailed feedback which helped us improve this paper.

References

[1] IDC and Statista, "Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025 (in zettabytes) [Graph]," https://www.statista.com/statistics/871513/worldwide-data-created/, Jun. 2021, [Online; accessed 2024-02-27].

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003, pp. 29–43.

[3] C. Lam, *Hadoop in action*. Simon and Schuster, 2010.

[4] D. Salomon, *Data compression*. Springer, 2002.

[5] A. El-Shimi, R. Kalach, A. Kumar, A. Ottean, J. Li, and S. Sengupta, "Primary data Deduplication—Large scale study and system design," in *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. Boston, MA: USENIX Association, Jun. 2012, pp. 285–296. [Online]. Available: https://www.usenix.org/conference/atc12/technical-sessions/presentation/el-shimi

[6] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou, "A comprehensive study of the past, present, and future of data deduplication," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.

[7] "Characteristics of backup workloads in production systems," in *10th USENIX Conference on File and Storage Technologies (FAST 12)*. San Jose, CA: USENIX Association, Feb. 2012. [Online]. Available: https://www.usenix.org/conference/fast12/characteristics-backup-workloads-production-systems

[8] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," in *9th USENIX Conference on File and Storage Technologies (FAST 11)*. San Jose, CA: USENIX Association, Feb. 2011. [Online]. Available: https://www.usenix.org/conference/fast11/study-practical-deduplication

[9] A. Liu, A. Baba, S. Udayashankar, and S. Al-Kiswany, "Dedupbench: A benchmarking tool for data chunking techniques," in *2023 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2023, pp. 469–474.

[10] Y. Zhang, H. Jiang, D. Feng, W. Xia, M. Fu, F. Huang, and Y. Zhou, "Ae: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 1337–1345.

[11] W. Xia, Y. Zhou, H. Jiang, D. Feng, Y. Hua, Y. Hu, Q. Liu, and Y. Zhang, "FastCDC: A fast and efficient Content-Defined chunking approach for data deduplication," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. Denver, CO: USENIX Association, Jun. 2016, pp. 101–114. [Online]. Available: https://www.usenix.org/conference/atc16/technical-sessions/presentation/xia

[12] R. N. Widodo, H. Lim, and M. Atiquzzaman, "A new content-defined chunking algorithm for data deduplication in cloud storage," *Future Generation Computer Systems*, vol. 71, pp. 145–156, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X16305829

[13] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 174–187. [Online]. Available: https://doi.org/10.1145/502034.502052

[14] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, W. Xia, Y. Zhang, and Y. Tan, "Design tradeoffs for data deduplication performance in backup workloads," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, 2015, pp. 331–344.

[15] D. E. E. 3rd and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," Internet Engineering Task Force, Internet-Draft draft-eastlake-sha1-02, Apr. 2001, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-eastlake-sha1/02/

[16] Wikipedia contributors, "Information theory," https://en.wikipedia.org/w/index.php?title=Information_theory&oldid=1206249062, 2024, accessed: February 16, 2024.

[17] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[18] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: local algorithms for document fingerprinting," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 76–85. [Online]. Available: https://doi.org/10.1145/872757.872770

[19] K. Eshghi and H. K. Tang, "A framework for analyzing and improving content-based chunking algorithms," *Hewlett-Packard Labs Technical Report TR*, vol. 30, no. 2005, 2005.

[20] "VirtualBoxes - Free VirtualBox(R) Images — sourceforge.net," https://sourceforge.net/projects/virtualboximage/, [Accessed 20-03-2024].

[21] Linux, "The Linux Kernel Archives," https://www.kernel.org/, 2023, [Online; accessed ¡Date-You-Accessed-The-Site¿].

[22] V. Tarasov, A. Mudrankit, W. Buik, P. Shilane, G. Kuenning, and E. Zadok, "Generating realistic datasets for deduplication analysis," in *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, 2012, pp. 261–272.

[23] I. OpenSSL Foundation, "Openssl," https://www.openssl.org/.

[24] "DedupBench Code Repository," https://github.com/UWASL/dedup-bench, 2023.

[25] "Cloudlab hardware," https://docs.cloudlab.us/hardware.html.

[26] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb *et al.*, "The design and operation of cloudlab," in *2019 USENIX annual technical conference (USENIX ATC 19)*, 2019, pp. 1–14.

[27] S. Al-Kiswany, A. Gharaibeh, E. Santos-Neto, G. Yuan, and M. Ripeanu, "Storegpu: exploiting graphics processing units to accelerate distributed storage systems," in *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, ser. HPDC '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 165–174. [Online]. Available: https://doi.org/10.1145/1383422.1383443

[28] W. Xia, H. Jiang, D. Feng, and Y. Hua, "Similarity and locality based indexing for high performance data deduplication," *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 1162–1176, 2015.

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

[30] 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.

[31] W. Xia, H. Jiang, D. Feng, and L. Tian, "Combining deduplication and delta compression to achieve low-overhead data reduction on backup datasets," in *2014 Data Compression Conference*, 2014, pp. 203–212.

[32] W. Xia, H. Jiang, D. Feng, L. Tian, M. Fu, and Y. Zhou, "Ddelta: A deduplication-inspired fast delta compression approach," *Performance Evaluation*, vol. 79, pp. 258–272, 2014, special Issue: Performance 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166531614000790