

Exploiting Zero-Entropy Data for Efficient Deduplication

by

Mu'men Al Jarah

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2025

© Mu'men Al Jarah 2025

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Research presented in Chapter 4 is included in a paper titled “The Impact of Low-Entropy on Chunking Techniques for Data Deduplication”, published in the IEEE 17th International Conference on Cloud Computing (CLOUD), 2024 [14]. I am the first author of this paper, and it has been done in collaboration with Sreeharsha Udayashankar and Abdelrahman Baba under the supervision of Professor Samer Al-Kiswany. Sreeharsha Udayashankar and Abdelrahman Baba helped in building a benchmarking tool for data chunking techniques used in data deduplication, which I utilize and extend in this thesis.

Abstract

As the volume of digital data continues to grow rapidly, efficient data reduction techniques, such as deduplication, are essential for managing storage and bandwidth. A key step in deduplication is file chunking, which is typically performed using Content-Defined Chunking (CDC) algorithms. While these algorithms have been studied under random data, their performance in the presence of zero-entropy data, where long sequences of identical bytes appear, has not been explored. Such zero-entropy data are common in real-world datasets and introduce challenges for CDC in deduplication systems.

This thesis studies the impact of zero-entropy data on the performance of both hash-based and hashless state-of-the-art CDC algorithms. The results show that existing algorithms, particularly hash-based ones, are inefficient at detecting and handling zero-entropy blocks, especially when these blocks are small, which reduces space savings. To address this issue, I propose ZERO (Zero-Entropy Region Optimization), a system that can be integrated into the deduplication pipeline. ZERO identifies and extracts zero-entropy blocks prior to chunking, compresses them using Run-Length Encoding (RLE), and stores their metadata for later reconstruction. ZERO improves deduplication space savings by up to 29% without impacting throughput.

Acknowledgements

I would like to express my heartfelt thanks and gratitude to all the people who made this thesis possible.

First and foremost, I would like to express my deepest gratitude and appreciation to my supervisor, Professor Samer Al-Kiswany, for his mentorship, unwavering support, and continuous guidance, which have been essential to my development as a researcher. His dedication, insightful feedback, and patience made this work possible and greatly enhanced its quality. On a personal level, Samer has shown exceptional kindness, encouragement, and wise advice during pivotal moments. His support has left a lasting impact, guiding my path and inspiring me toward a brighter future.

I am grateful to my colleagues in Waterloo Advanced Systems Lab (WASL) for all the moments we have shared. Thank you for your friendship, collaboration, and the many discussions and exchanges of ideas that have been invaluable to me. I will always remember your willingness to help, your encouragement, and the friendly, supportive environment you created. I also thank my colleague Sreeharsha Udayashankar for his fruitful collaboration and continuous feedback in debugging.

I am profoundly grateful to my parents, Zeyad and Hana, for shaping who I am today. Your trust, guidance, and belief in me made it possible for me to take this step with my family. I would not be here without you. I also thank my brother, Mohammad, and my sisters, Alaa, Batool, and Joud, for their constant motivation and encouragement of me throughout this journey. Your care and support made this achievement possible.

I am deeply grateful to my wife, Sara, for her constant support and patience at all times. She always found a way to lift my spirits when I lost motivation and ensured I stayed on track. Her understanding and encouragement were vital to completing this work.

Last but not least, I thank my two-year-old daughter, Salma, for bringing happiness to our home. I will always cherish her smiles, hugs, and the excitement she shows whenever she sees me.

Table of Contents

Author’s Declaration	ii
Statement of Contributions	iii
Abstract	iv
Acknowledgements	v
List of Figures	ix
List of Tables	xi
1 Introduction	1
2 Background	4
2.1 Data Deduplication	4
2.2 File Chunking	5
2.3 Content-Defined Chunking (CDC)	5
2.4 Data Compression	7
2.5 Data Entropy	8

3	Characterizing Zero-entropy Data in Real Datasets	9
3.1	Datasets Overview	10
3.2	Identifying Zero-Entropy Data	10
3.3	Block Size Distribution	11
3.4	Distribution Within Files	12
3.5	Common Repeated Bytes	15
3.6	Summary	15
4	The Impact of Zero-Entropy Data on CDC Algorithms	18
4.1	Synthetic Dataset Construction	18
4.2	Performance of Content-Defined Chunking Algorithms	20
4.2.1	Space Savings	21
4.2.2	Chunking Throughput	24
4.2.3	Chunk Size Distribution	25
4.2.4	Artificial Chunk Boundaries	25
4.3	Summary	28
5	ZERO: A System for Zero-Entropy Region Optimization in Data Deduplication	30
5.1	Design	30
5.2	Evaluation	32
5.2.1	Comparison of Data Reduction Techniques (Space Savings vs. Throughput)	33
5.2.2	Space Savings	35
5.2.3	End-to-end Evaluation	37
6	Related Work	40
6.1	Chunking Performance Improvement	40
6.2	Enhancing Deduplication Efficiency	42
6.3	Data Compression	43
6.4	Delta Compression	43

7 Conclusion and Future Work	44
References	46

List of Figures

2.1	Data deduplication workflow.	5
2.2	Hash-based CDC algorithms.	6
2.3	Asymmetric Extremum (AE).	7
2.4	Rapid Asymmetric Maximum (RAM).	7
3.1	CDF of zero-entropy data across block size ranges.	12
3.2	CDF of zero-entropy ratios across chunks for all datasets.	13
3.3	Overview of the most frequently repeated bytes in zero-entropy blocks for each dataset.	16
4.1	Synthetic datasets with random vs zero-entropy data insertion.	19
4.2	Space savings under random vs zero-entropy data insertions.	21
4.3	Impact of zero-entropy update ratio on space savings.	22
4.4	Impact of inserted zero-entropy block size on space savings of CDC algorithms.	23
4.5	Chunking throughput comparison.	24
4.6	CDF of chunk size distributions under zero-entropy vs random data insertions.	26
4.7	Artificial chunk boundaries under random and zero-entropy data insertion.	27
4.8	Artificial Chunk Boundary Comparison.	28
5.1	Overview of ZERO System.	31
5.2	Data Reconstruction Workflow.	32

5.3	Comparison of data reduction techniques across three datasets in terms of space savings and end-to-end throughput.	35
5.4	Space savings of ZERO compared to state-of-the-art deduplication on three datasets: VDI, LNX, and CORD-19.	36
5.5	Processing time breakdown of the deduplication pipeline across all datasets and CDC algorithms. Each group shows the times with and without ZERO.	38

List of Tables

3.1	Zero-entropy information in real-world datasets.	11
3.2	Summary of statistical properties for zero-entropy data across 128MB chunks in three datasets.	14

Chapter 1

Introduction

The amount of digital data is growing rapidly. In 2020, global data production reached 64.2 zettabytes, and it is expected to exceed 390 zettabytes by 2028 [13]. Previous studies show that large amounts of data stored on the cloud are redundant [31]. To cope with this massive growth and eliminate redundant content, cloud storage providers rely on numerous mechanisms, including distributed file systems [10], [15], data compression [24], and data deduplication [6], [36]. Among these, data deduplication plays a vital role in reducing this redundancy by identifying the redundant copies and avoiding their storage and transmission, thus reducing storage requirements and conserving network bandwidth [31], [19], [18].

Data deduplication typically involves multiple stages [6], where file chunking comes first. In this stage, the deduplication systems divide the source data into smaller chunks, as chunk-level data deduplication is more effective than file-level data deduplication [19]. Then, the system hashes each chunk using a collision-resistant hashing function to generate a unique fingerprint. After that, the system compares these fingerprints against an index of previously seen fingerprints to identify duplicates. Finally, the system discards duplicate chunks, stores the unique chunks, and adds their fingerprints to the index for future reference.

Most deduplication systems use Content-Defined Chunking (CDC) algorithms, which divide files based on their content. Numerous CDC algorithms exist in the current literature [44], [41], [33], [20], each offering different levels of deduplication space savings and chunking throughput. There are two categories of CDC algorithms: hash-based and hashless algorithms. Hash-based algorithms, such as Rabin’s chunking [20] and FastCDC [41], use a rolling hash function to determine the boundary of chunks. Hashless chunking

algorithms, such as Asymmetric Extremum (AE) [44] and Rapid Asymmetric Maximum (RAM) [33], identify chunks based on local minima/maxima in the data stream.

There are a large number of real-world datasets that include low-entropy regions. For example, virtual machine (VM) disk images often contain large sections filled with zeros, and scientific datasets may contain repeated measurement values. These regions offer an opportunity for optimization because they are easy to compress and can be detected with minimal computation. However, most modern deduplication systems do not explicitly handle such low-entropy patterns. In my thesis, I focus on the extreme case of low-entropy, known as zero-entropy, which is a long sequence of repeated bytes, such as zeros. I chose zero-entropy because it is easy to detect and process, and working with it provides greater benefit compared to general low-entropy regions.. My hypothesis is that leveraging zero-entropy blocks can improve deduplication performance.

I begin with a characterization study of zero-entropy data blocks across real-world datasets, focusing on their ratios, sizes, distributions, and repeated byte values. I selected datasets that were expected to contain zero-entropy regions, and verified that they do. I study three datasets from different domains: Linux kernel source codes [17], virtual machine disk images [29], and the CORD-19 scientific research dataset [32]. I developed a tool that detects and analyzes zero-entropy blocks. The results show that zero-entropy blocks exist in some production datasets with significant ratios ranging from 10% to 28% of the total size of the dataset. Most of these zero-entropy blocks are small, typically 8 to 32 bytes, and also appear across the entire dataset, not just in certain regions. Additionally, I have found that each dataset has a dominant repeated-byte value, which can simplify the encoding process.

Then, I study the impact of the zero-entropy blocks on the behavior of CDC algorithms. I use synthetic datasets with inserted zero-entropy blocks of varying sizes to evaluate this impact. I evaluate four state-of-the-art CDC algorithms, AE, FastCDC, Rabin, and RAM, across four key metrics: space savings, chunking throughput, chunk size variance, and artificial boundary percentage. The results reveal that CDC algorithms often struggle to detect and isolate small zero-entropy blocks, resulting in lower deduplication efficiency. However, hashless methods like AE and RAM exhibit more consistent performance with larger zero-entropy blocks due to their lower chunk size variance. These findings highlight the limitations of state-of-the-art CDC techniques and motivate the exploration of techniques to leverage zero-entropy data.

Based on these findings, I propose ZERO (Zero-Entropy Region Optimization), a lightweight preprocessing stage that runs before chunking and fingerprinting. ZERO scans the input data stream, extracts zero-entropy blocks, and encodes them using Run-Length

Encoding (RLE) [11], which is appropriate to encode long sequences of the same byte. ZERO stores these encoded blocks separately, along with metadata that allows them to be restored later. By reducing the amount of data passed to CDC algorithms, ZERO improves deduplication efficiency.

I evaluate ZERO using experiments on three real-world datasets: VDI, LNX, and CORD-19, and four state-of-the-art CDC algorithms: AE, FastCDC, Rabin, and RAM, using average chunk sizes of 4 KB, 8 KB, and 16 KB. The evaluation focuses on three main metrics: space savings, throughput, and end-to-end processing time. Compared to deduplication systems, ZERO improves space savings by 6–29 % across all datasets and CDC algorithms, with the largest gains in datasets containing more zero-entropy data. For throughput, ZERO presents comparable or slightly higher performance than deduplication. For the end-to-end evaluation, most CDC algorithms benefit from ZERO, with the Rabin algorithm presenting the greatest improvements of 24% faster than without ZERO due to reduced chunking overhead. While compression achieves higher space savings, ZERO presents high throughput, achieving $25\times$ faster processing than compression while still offering high storage reduction.

The rest of this thesis is structured as follows. Chapter 2 presents background details on chunking and deduplication. Chapter 3 provides a study of the characterization of zero-entropy data in real-world datasets. Chapter 4 examines the impact of zero-entropy data on CDC algorithms, and Chapter 5 shows the proposed design with its evaluation. Chapter 6 summarizes related work, and I conclude in Chapter 7.

Chapter 2

Background

This chapter covers the background information used in this thesis on data deduplication, chunking algorithms, and data entropy.

2.1 Data Deduplication

The rapid growth of digital data increases the demand on modern storage systems [13]. To handle the increasing volume of data effectively, these systems rely on data reduction methods such as data deduplication [6], [36]. In particular, backup and archival systems commonly use data deduplication. This helps optimize storage efficiency within large-scale storage systems [19] by eliminating redundant data. It removes redundancy by dividing files into chunks and storing only one copy of each unique chunk.

Figure 2.1 shows the workflow of the data deduplication process, which consists of four main steps:

1. File Chunking: The system divides each file into smaller parts called chunks using *chunking algorithms*.
2. Chunk Hashing: The system computes a fingerprint for each chunk using a collision-resistant hash function, such as SHA-1 [2] to generate *fingerprints*.
3. Fingerprint Indexing: The system compares fingerprints against a database of previously observed fingerprints. A duplicate fingerprint indicates a duplicate chunk, which the system eliminates without losing any unique data.

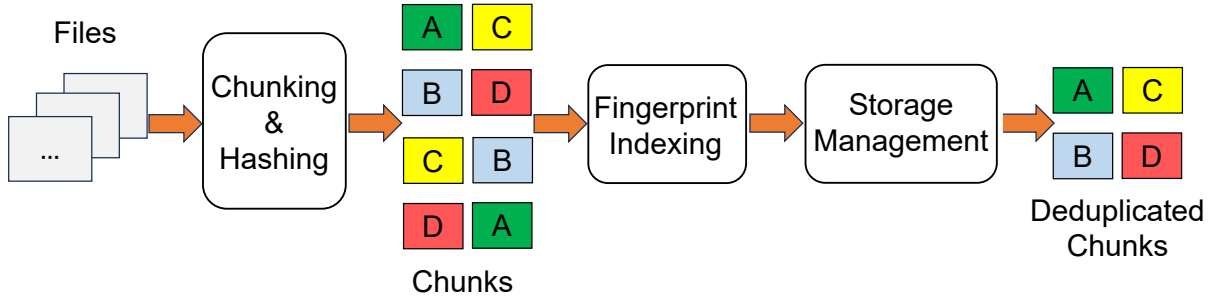


Figure 2.1: Data deduplication workflow.

4. Storage Management: The system stores only the unique chunks and adds their fingerprints to the database for future comparisons.

2.2 File Chunking

File chunking is the first step in the deduplication pipeline and is a critical stage in the deduplication process [19]. Most deduplication systems in production use Content-Defined Chunking (CDC) algorithms to divide files into chunks based on data characteristics. This requires scanning each file in its entirety. Chunking algorithms impact deduplication efficiency as well. Tuning chunking algorithms to generate larger chunks reduces indexing and metadata overheads, while smaller chunk sizes can enhance the space savings achieved by deduplication.

2.3 Content-Defined Chunking (CDC)

CDC algorithms slide a window over the source data. When the data within the window matches certain target characteristics, they insert chunk boundaries to divide the data into chunks. To control chunk variability, the algorithm may not find a suitable chunk boundary, or it may find very small chunks that increase indexing overhead. To address this, algorithms have a minimum chunk size, ensuring that no boundary is placed smaller than this limit. Similarly, when the CDC cannot find a natural boundary before reaching the maximum chunk size, it inserts an artificial chunk boundary [20], [41]. A high percentage of

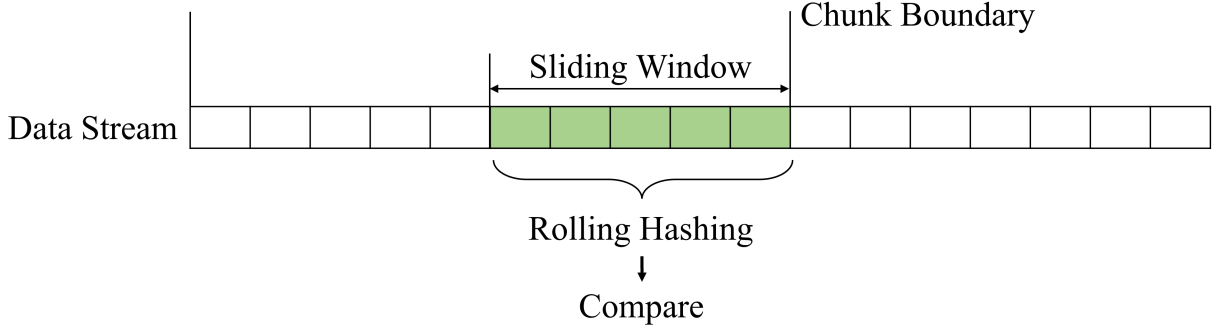


Figure 2.2: Hash-based CDC algorithms.

artificial boundaries typically reduces deduplication efficiency, since such boundaries may not align with repeated or redundant content.

CDC algorithms are generally categorized into hash-based and hashless algorithms. Figure 2.2 illustrates a hash-based CDC algorithm, where a fixed-size sliding window moves forward one byte at a time over the data stream. 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. The window size is typically chosen relative to the target average chunk size. Examples of hash-based algorithms include Rabin’s chunking [20] and FastCDC [41]. These algorithms insert a chunk boundary when the lower n bits of the window’s rolling hash value equal zero. I note that the hashing algorithms used here differ from the collision-resistant algorithms used for fingerprint indexing.

On the other hand, hashless CDC algorithms detect boundaries based on byte values rather than hash values, such as Asymmetric Extremum (AE) [44] and Rapid Asymmetric Extremum (RAM) [33]. They inspect the contents of the window to find local extrema (minima or maxima) and use those positions as chunk boundaries.

Figure 2.3 shows how the Asymmetric Extremum (AE) algorithm works. AE uses a variable-sized window and a fixed-sized window, with a target byte between them. AE slides over the data and checks if the target byte is an extreme value, either the minimum or maximum within the fixed-size window. If it is, the algorithm places a chunk boundary at the end of the fixed-size window. If not, the variable window expands by one byte and the check continues.

Figure 2.4 illustrates the Rapid Asymmetric Maximum (RAM) algorithm, which reverses the places of windows compared to AE. Each chunk starts with a fixed-size window

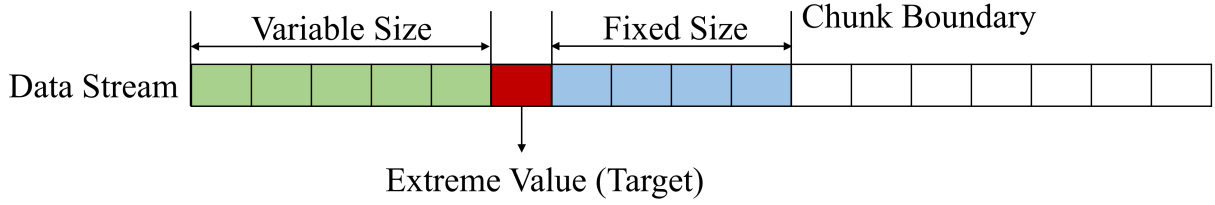


Figure 2.3: Asymmetric Extremum (AE).

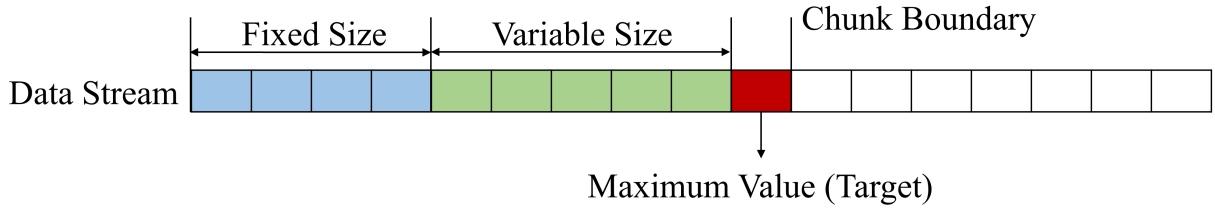


Figure 2.4: Rapid Asymmetric Maximum (RAM).

and ends with a variable-size one. A sliding window scans the variable region and places the chunk boundary when the next byte after the window is at least as large as the maximum value inside the variable window. The fixed-size window identifies the maximum byte value to serve as a threshold, while the variable-size window scans ahead to find the first byte that meets or exceeds this threshold, marking the end of the chunk.

2.4 Data Compression

Compression is another data reduction method that encodes repeated patterns to save space. However, it often processes data more slowly. For example, GZIP [9] is a lossless compression technique that identifies repeated byte patterns and replaces them with shorter codes, using fewer bits for common patterns to reduce file size while maintaining full data recovery. In addition, run-length encoding (RLE) [11] is a simple version of lossless compression techniques that stores one copy of a repeated data value along with how many times it occurs in a row. In this thesis, I focus on data deduplication, which is widely used in large-scale backup systems.

2.5 Data Entropy

Entropy, borrowed from information theory [34], measures the randomness or variability within data. Claude Shannon’s work [26] proposed foundational steps for understanding how entropy influences the processing and transmission of data. He explains that messages with more randomness have higher entropy. Mathematically, Shannon defines entropy using the formula:

$$H(X) = - \sum_{x \in X} p(x) \log_2 p(x), \quad (2.1)$$

where X is a discrete random variable representing all possible byte values, and $p(x)$ is the probability of occurrence of a particular value $x \in X$.

Low-entropy blocks represent a high degree of redundancy, such as repeated bytes or patterns [25]. These blocks are important in data deduplication because their low variability makes them good candidates for detection and extraction. In this thesis, I focus on the most extreme case of low-entropy, which is zero-entropy blocks. These are uniformly repeated byte sequences, such as (000000000000), rather than alternating patterns like (0101010101) or irregular repetitions such as (000010000000100). I chose zero-entropy blocks because they are easy to detect and can be processed with minimal effort. Encoding repeated data values is fast, and targeting such blocks provides the greatest benefit in improving deduplication efficiency.

My observation is that zero-entropy blocks are common in real-world datasets and are important for optimizing current deduplication techniques. This is important because deduplication plays a key role in reducing storage space in large backups and archival systems.

Chapter 3

Characterizing Zero-entropy Data in Real Datasets

The goal of this chapter is to study the characteristics of real-world data to find the percentage of zero-entropy blocks in real-world data sets. I focus on zero-entropy blocks because they are easier to detect and extract, making them a good target for space savings with minimal computational overhead during encoding. Studying zero-entropy blocks gives me a lower bound on the benefits of leveraging low-entropy regions for space savings. I aim to understand the size, distribution, and location of these blocks within files, as well as the values of the most frequently repeated bytes. These characteristics are important because they affect how I handle zero-entropy during deduplication.

To do this, I used datasets from three domains, including virtual disk images, Linux kernel archives, and scientific research files. I built a tool that scans these datasets, analyzes the content, and detects any repeated byte sequences. The tool focuses on identifying zero-entropy blocks, which represent no randomness. Specifically, I consider any sequence of eight bytes or more that has the same value a zero-entropy block. I chose eight as the threshold because it represents the size of the largest primitive data types, such as a double. Smaller repeated sequences could just be part of a single primitive value, rather than real redundancy.

Smaller repeated patterns may be part of common data types and do not yield savings when encoded.

3.1 Datasets Overview

For this study, I selected datasets that I expected to contain zero-entropy regions, and then verified that they do. This ensures the datasets are representative of real-world cases where zero-entropy blocks exist. I use three datasets in this characterization study:

- **Linux kernels (LNX)** [17]: This dataset includes 158 TAR files (.tar) of Linux kernel source code, ranging from 20 MB to 1 GB, with a total of 65 GB. It covers various kernel versions from Linux 2.0 to Linux 6.5. Each TAR file includes the full source tree for one version. These versions share common files and structures.
- **Virtual Machine Disk Images (VDI)** [29]: This dataset has 48 virtual disk image files (.vdi), each between 1 GB and 5 GB in size, with a total of 121 GB. The files contain Linux distributions including Ubuntu, Fedora, Debian, and CentOS. VDI files store full virtual machine disk images, which often include large sections of space or repeated byte patterns.
- **COVID-19 Open Research Dataset (CORD-19)** [32]: This dataset is a collection of articles and data released in response to the global COVID-19 pandemic. It includes over one million scholarly articles related to COVID-19 and similar viruses, with more than 400,000 full-text entries. The dataset contains different files in .csv, .txt, and .pdf formats, with a total of 81.5 GB. The files mainly contain raw data about COVID-19 cases, diagnostics, and research findings. This dataset was collected by researchers who removed redundant data during collection. Specifically, they group records that refer to the same paper using unique identifiers. From each group, they keep only one metadata record and discard any duplicates or incomplete versions.

3.2 Identifying Zero-Entropy Data

To analyze the zero-entropy blocks in datasets, I built a tool that scans all files in each dataset and detects zero-entropy blocks. The tool records the size of each block and its byte value. Then, it groups block sizes into ranges.

Table 3.1 presents key properties of datasets, including the dataset size, number of files, the percentage of zero-entropy blocks, and the percentage of files that have zero-entropy blocks. The table shows a large number of zero-entropy blocks in these datasets. For

Dataset	Size	Description	Number of files	Percentage of files with zero-entropy	Zero-entropy percentage
LNX [17]	65 GB	158 Linux kernel distributions in TAR format	158	100%	10%
VDI [29]	121 GB	48 Virtual Desktop Infrastructure Images	48	100%	28%
CORD-19 [32]	81.5 GB	COVID-19 Open Research Dataset	717,120	80.25%	28.5%

Table 3.1: Zero-entropy information in real-world datasets.

instance, in the VDI dataset, which has a total size of 121 GB, 34 GB (28%) of the data is zero-entropy. All files in this dataset contain zero-entropy data. Similarly, the CORD-19 dataset contains 81.5 GB of data and 717,120 files (80% of its files have zero-entropy), has a zero-entropy ratio of 28.5%. The LNX dataset contains a 10% zero-entropy ratio of its size, and 100% of its 158 files contain zero-entropy blocks. These percentages highlight the existence of high zero-entropy blocks in each dataset. Because the datasets come from different domains, the results show that zero-entropy blocks appear in a variety of real-world domains and motivate building systems that can exploit them.

3.3 Block Size Distribution

Figure 3.1 shows the cumulative distribution function (CDF) of the size of zero-entropy blocks. In all three datasets, the majority of zero-entropy blocks are small, between eight bytes and two kilobytes. However, the distribution varies by dataset.

In the VDI dataset, 63% of the zero-entropy data is found in the smallest block range (8–15 bytes). The cumulative distribution increases gradually, with 98% of zero-entropy data located in blocks smaller than 256 bytes. In the CORD-19 dataset, 99% of zero-entropy data appears in blocks smaller than 32 bytes. In contrast, the LNX dataset shows a broader distribution, with zero-entropy blocks spread over a wider range of block sizes, up to 256 bytes. Although the exact distributions differ across datasets, the presence of many small zero-entropy blocks in all three confirms that this is a common pattern. Additionally, these insights guide my synthetic dataset generation in Chapter 4

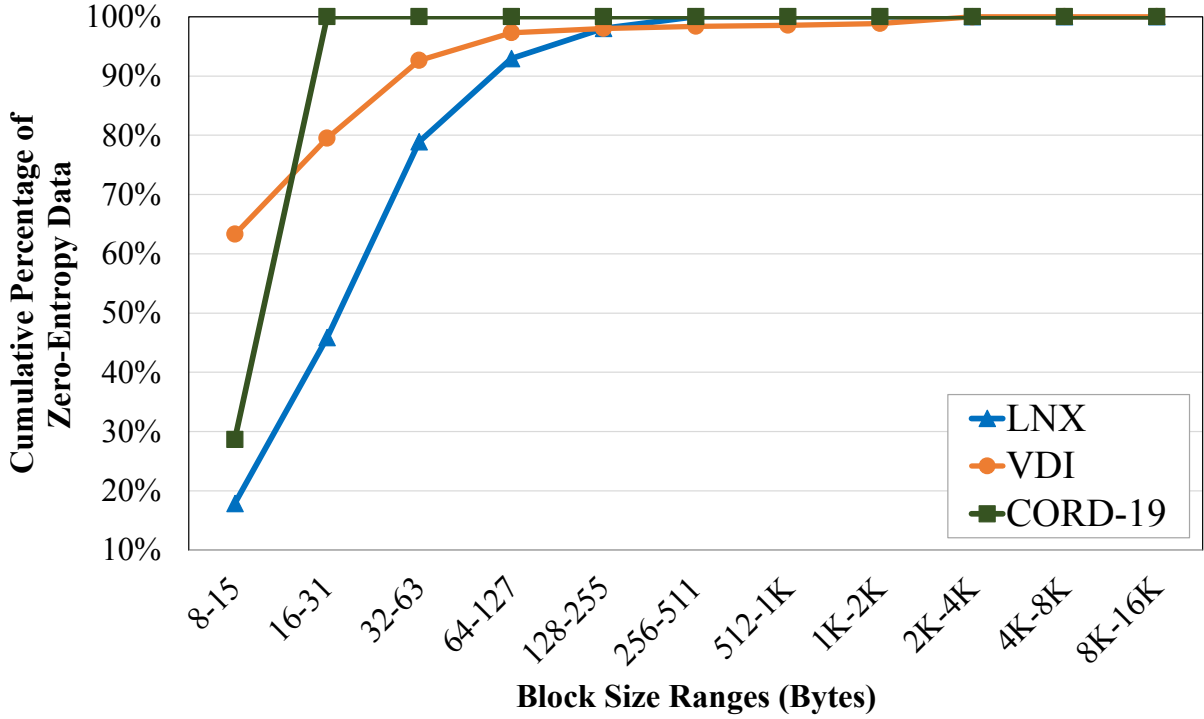


Figure 3.1: CDF of zero-entropy data across block size ranges.

3.4 Distribution Within Files

One important characteristic of zero-entropy blocks is their location in a file or data set. Understanding whether zero-entropy blocks are uniformly distributed or concentrated in a few regions of the dataset can enable optimizing the processing of datasets with zero-entropy regions. To examine the distribution of zero-entropy blocks across the datasets, I built a tool that divides the datasets into fixed-size chunks (128 MB) and computes the ratio of zero-entropy blocks within each chunk.

Figure 3.2 shows the cumulative percentage of chunks with different zero-entropy ratios for all datasets. The three datasets show different patterns of zero-entropy data distribution across chunks. In the CORD-19 dataset, most chunks have either low or high ratios of zero-entropy data, with a notable increase near 38%. The LNX dataset has a more gradual increase, and most chunks have zero-entropy ratios between 8% and 20%. For the VDI dataset, it shows a steady distribution across a wide range of zero-entropy ratios, without jumps, indicating a mix of very low and very high ratios across the chunks. This

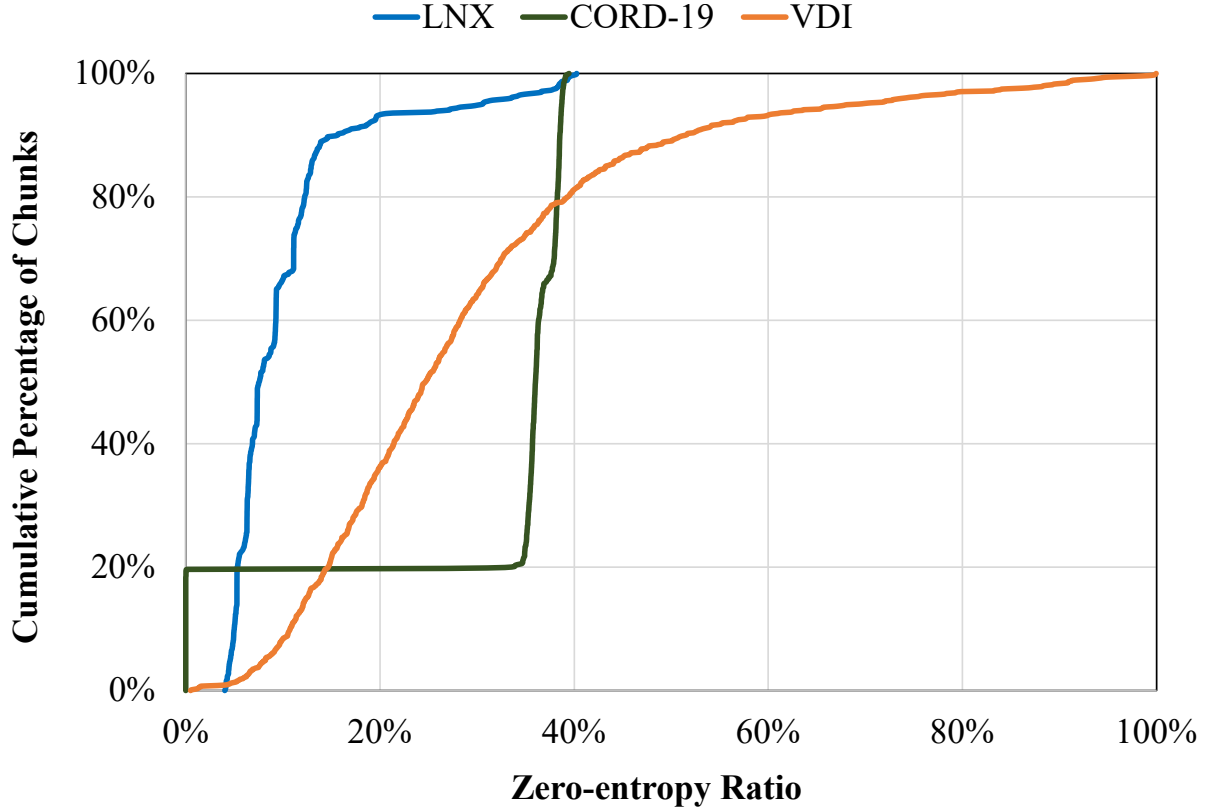


Figure 3.2: CDF of zero-entropy ratios across chunks for all datasets.

characteristic is important to know if zero-entropy blocks are clustered in specific regions, where the system can encode only these regions, or are distributed among all files, which requires scanning all data. So, in both cases, this provides insight into how to generate synthetic datasets to study the impact of zero-entropy data on CDC algorithms.

For a better understanding of these distributions, I measure key statistical metrics to analyze the results. These include the mean, which represents the average percentage of zero-entropy data in each 128MB chunk. The standard deviation shows how much the zero-entropy percentages vary across different chunks. The minimum and maximum values indicate the lowest and highest observed zero-entropy percentage in any single chunk. Finally, the skew describes the shape of the distribution, whether most chunks have lower or higher zero-entropy content.

Table 3.2 summarizes these metrics in all three datasets. My analysis shows that

Metric	Description	Datasets		
		CORD-19 [32]	LNK [17]	VDI [29]
Number of Chunks	Number of 128MB chunks analyzed from the dataset.	653	625	989
Mean (%)	Average percentage of each chunk that is made up of zero-entropy data (e.g., repeated bytes like zeros).	29.6	10.1	28.7
Standard deviation (%)	How much the zero-entropy percentages vary across chunks.	14.7	7.3	18.3
Min–Max (%)	Lowest and highest percentage of zero-entropy data observed in any single chunk.	0.0–56.1	4.0–54.9	0.5–99.9
Skew	Shape of the distribution: negative = left-skewed (more high values), positive = right-skewed (more low values).	−1.49	2.62	1.56

Table 3.2: Summary of statistical properties for zero-entropy data across 128MB chunks in three datasets.

the amount and distribution of zero-entropy data vary between datasets. The table shows different numbers of 128MB chunks that were analyzed, based on data size: 653 for CORD-19, 625 for LNK, and 989 for VDI. On average, 29.6% of the data in each chunk of CORD-19 and 28.7% in VDI consists of zero-entropy data, while LNK has less, with only 10.1%. Also, the table shows variability in these percentages across chunks by using the standard deviation. VDI has the highest variability (18.3%), followed by CORD-19 (14.7%), and LNK has the most consistent values (7.3%).

The range of zero-entropy ratios per chunk also varies. In VDI, chunks contain between 0.5% and 99.9% zero-entropy data. For CORD-19, the range starts from 0.0% to 56.1%, and for LNK, from 4.0% to 54.9%. Lastly, the skewness describes the shape of the distribution in each dataset. CORD-19 is left-skewed (−1.49), which means that most chunks have high zero-entropy blocks, while a few toward the end have significantly low values. In

contrast, LNX and VDI are right-skewed (2.62 and 1.56), meaning most chunks have low zero-entropy content, with fewer chunks containing high amounts.

3.5 Common Repeated Bytes

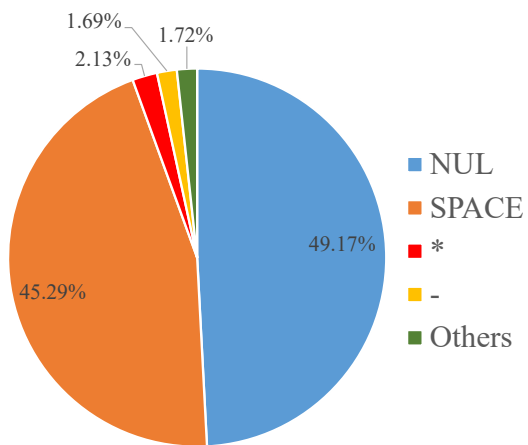
Another characteristic to understand is the nature of the repeated content, specifically, the values of the most frequently repeated bytes. This is needed because if a single character dominates zero-entropy blocks, such as zero, this makes the extraction easier by only scanning for zero and avoiding scanning for other byte values, and gives a view of the nature of repeated bytes in these regions. To analyze these repeated bytes, I developed a tool that records the most frequent byte values found in zero-entropy blocks for each dataset.

Figure 3.3 illustrates the percentages of these repeated bytes across datasets. The results show that each dataset has a dominant repeated byte in its zero-entropy blocks, but the dominant byte differs across datasets. In the LNX dataset, the two most common repeated bytes are null and space characters, which together make up 94% of all zero-entropy blocks. Other characters like (*) and (-) also appear, but much less often. For the VDI dataset, the dominant character is the NUL byte, which makes up 92% of all repeated bytes. In CORD-19, the space character alone accounts for over 99% of the repeated bytes.

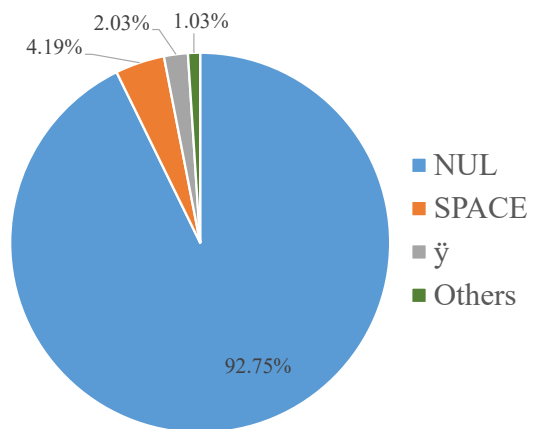
Implementation. I developed a C++ program of 340 lines of code. It analyzes files to detect zero-entropy blocks. The code takes a directory as input. It scans all files inside that directory and any other subdirectories. When the code finds a sequence of the same byte repeated at least eight times or more in a row, it counts it as a zero-entropy block. For every block, it records its starting position, its length, and the specific repeated byte. Then, it uses this information to build a detailed report of zero-entropy blocks across all files in the dataset. The program also generates statistics. It categorizes zero-entropy block sizes into ranges such as 8–1,000 bytes, 1,001–10,000 bytes, etc. It also records the count of each byte value that appears in zero-entropy blocks.

3.6 Summary

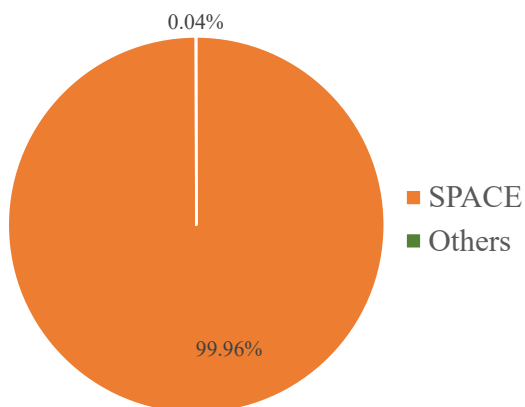
In this chapter, I analyze real-world datasets to identify and characterize zero-entropy blocks. I developed tools to detect these blocks, measure their size and distribution within files, and record the most frequently repeated byte values within them. I performed this



(a) LNX dataset.



(b) VDI dataset.



(c) CORD-19 dataset.

Figure 3.3: Overview of the most frequently repeated bytes in zero-entropy blocks for each dataset.

analysis across datasets from different domains, including virtual disk images, Linux kernel archives, and scientific research files. The analysis shows that all datasets contain a noticeable percentage of zero-entropy data, most of which is concentrated in small blocks. Also, I found that the repeated byte values are dominated by a small set of specific characters. From the characteristics study in this chapter, I summarize the following observations:

- Some production datasets have zero-entropy data. A significant amount of zero-entropy data, up to 28% in the datasets I explored. This amount of zero entropy creates an opportunity to extract it, which can reduce the volume of data that the deduplication system needs to process.
- The majority of zero-entropy blocks are typically small in the range of 8-32 bytes. Future techniques that want to exploit these blocks need to be efficient in processing this level of zero entropy to extract the benefit.
- My evaluation shows that zero-entropy data are spread across files. This means that any future techniques aiming to leverage zero-entropy data need to process the entire dataset. There is no opportunity to optimize by processing only a specific portion of the data, such as the beginning or end of a file, because zero-entropy blocks may appear anywhere within it.
- My analysis shows that in every dataset, there is one dominant repeated byte, which represents the majority of zero-entropy data. This can help in finding an efficient encoding technique by using that byte as a default value and just storing how many times it repeats. Since the byte is known, this reduces encoding overhead.

All these insights are critical for generating synthetic datasets to study the impact of zero-entropy data on the performance of CDC algorithms, which I present in the next chapter.

Chapter 4

The Impact of Zero-Entropy Data on CDC Algorithms

As I observed in Chapter 3, zero-entropy blocks are prevalent in real datasets in significant amounts. The primary objective of this chapter is to study the impact of these zero-entropy data on the performance of CDC algorithms in deduplication systems.

4.1 Synthetic Dataset Construction

To evaluate CDC behavior under controlled conditions, I developed a tool that constructs two synthetic datasets derived from the same base data. Figure 4.1 shows examples of them. The base consists of multiple files filled with uniformly random bytes, representing high-entropy content. The tool inserts blocks of data at random locations within these generated files. Figure 4.1a shows the first data set, which has inserted blocks of randomly generated bytes. These blocks represent typical high-entropy changes. Figure 4.1b shows the second dataset, which has inserted blocks of repeated byte values to mimic zero-entropy data.

I use random data as a baseline because many CDC algorithms are designed and analyzed using random input [44] [20], where their boundary detection is more predictable. In contrast, real datasets are hard to understand at the byte level due to their complexity and variability, and it is difficult to track how data changes over time. Prior work [27] shows that modeling real-world byte patterns is challenging. Also, the way data shifts or changes depends heavily on the specific dataset, so there is no single pattern that works

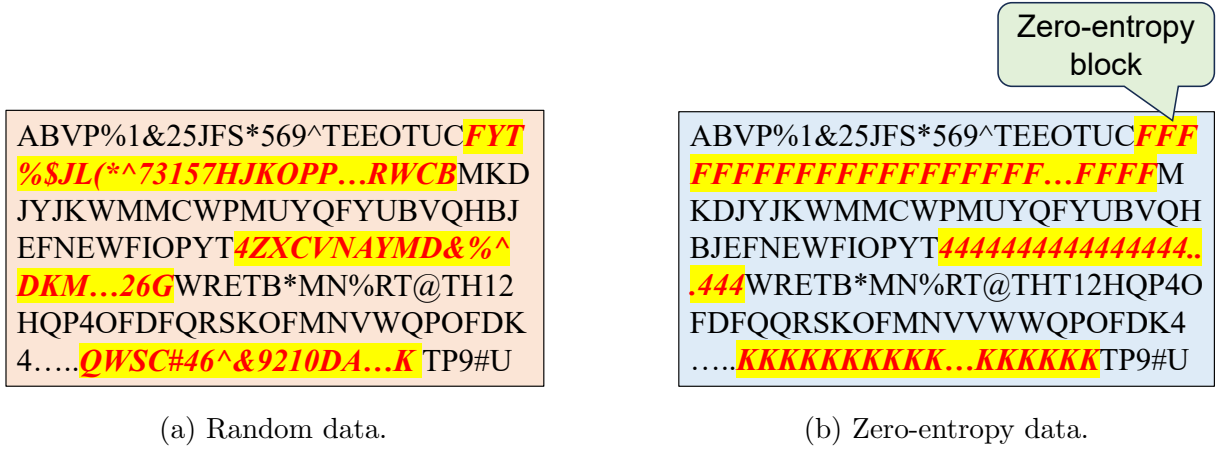


Figure 4.1: Synthetic datasets with random vs zero-entropy data insertion.

for all cases. My goal is to study how zero-entropy blocks affect chunk boundaries and deduplication, not to analyze how CDC handles other byte-shift patterns. Using a random baseline to control helps better highlight these effects.

I evaluate four state-of-the-art CDC algorithms, AE, FastCDC, Rabin, and RAM, using four metrics: space savings, chunking throughput, chunk size distribution, and artificial boundary percentage. This setup enables a comparative analysis of how these CDC algorithms perform under various conditions in the presence of zero-entropy data. I vary the following parameters to generate datasets possessing different characteristics:

1. **Block size:** I insert zero-entropy or random data as fixed-size blocks within the original data. I vary the size of these inserted blocks and examine their impact on CDC algorithms.
2. **Update Ratio:** I define the ratio of inserted data (random or repeated) relative to the total file size using the following formula:

$$\text{Update Ratio} = \frac{\text{Inserted Data Size}}{\text{Total Data Size}} \quad (4.1)$$

4.2 Performance of Content-Defined Chunking Algorithms

Modern CDC algorithms can be tuned to generate chunks with different average sizes. I experiment with configurations that use average chunk sizes of 4 KB, 8 KB, and 16 KB.

Metrics. I quantify the impact of zero-entropy blocks on these algorithms using the following metrics:

1. **Space Savings:** The space saving ratio that the CDC algorithm achieves.

$$\text{Space Savings} = \frac{\text{Original Size} - \text{Deduplicated Size}}{\text{Original Size}} \times 100 \quad (4.2)$$

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 [18] [8].
4. **Artificial Boundary Percentage:** The percentage of chunk boundaries that the CDC algorithm inserts due to the maximum chunk size limit being exceeded, rather than based on actual content features.

For each metric, I compare zero-entropy to random data insertions and also examine the impact of update ratio and block sizes.

Alternatives. I compare the following CDC algorithms:

- AE: The Asymmetric Extremum (AE) [44] algorithm. I use AE-Max; that is, the target byte is a local maximum.
- FastCDC: The rolling hash-based FastCDC [41] algorithm. I use a normalization level of 2.
- Rabin: Rabin’s chunking algorithm [20]
- RAM: Rapid Asymmetric Extremum (RAM) [33].

I use minimum and maximum chunk sizes of $0.5\times$ and $2\times$ the average chunk size, respectively. Unless otherwise noted in this chapter, the expected average chunk size is configured to be 16 KB. These parameters follow standard settings commonly used in the CDC algorithms, so I can compare the results fairly.

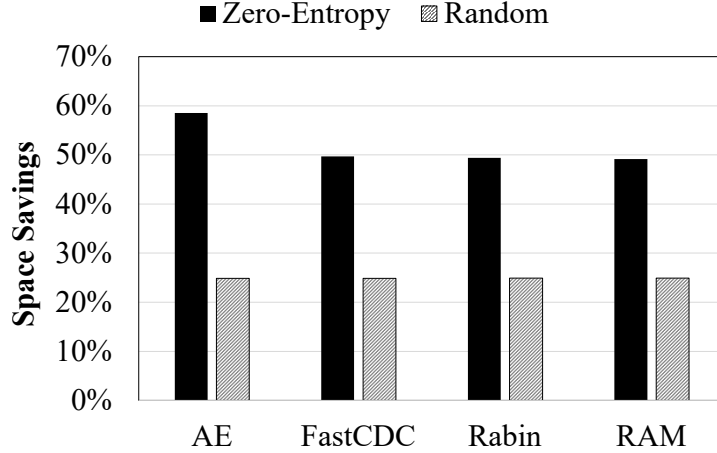


Figure 4.2: Space savings under random vs zero-entropy data insertions.

4.2.1 Space Savings

I evaluate the space savings achieved by the four CDC algorithms. I configure an expected average chunk size of 4 KB and a zero-entropy block size of 16 KB for this experiment.

Figure 4.2 presents a comparison of the space savings that CDC algorithms achieve under two synthetic datasets containing equal percentages of random and zero-entropy inserted blocks. The results show that all four CDC algorithms achieve significantly higher space savings when the inserted data consists of zero-entropy blocks compared to random data. For example, AE achieves a space savings ratio of 58% on datasets with zero-entropy blocks, compared to 24% with random data. Similarly, FastCDC, Rabin, and RAM each achieve a 50% space savings with zero-entropy data, compared to 25% with random data. Among all the CDC algorithms, AE [44] achieves the highest space savings in the presence of zero-entropy blocks.

Figure 4.3 shows the space saving ratios CDC algorithms achieve with increasing zero-entropy update ratio. The figure illustrates that, at low ratios, FastCDC and Rabin achieve the highest space savings among CDC algorithms, similar to the results of previous studies on datasets with random data [41]. However, as mentioned above, zero-entropy blocks can be eliminated. When I increase zero-entropy ratios, AE [44] is the only algorithm that identifies and efficiently eliminates these blocks. This causes AE to achieve higher space savings with increasing zero-entropy update ratios, as Figure 4.3 illustrates. Thus, in contrast to previous studies using random data [41], AE is the optimal CDC algorithm to handle high ratios of zero-entropy blocks.

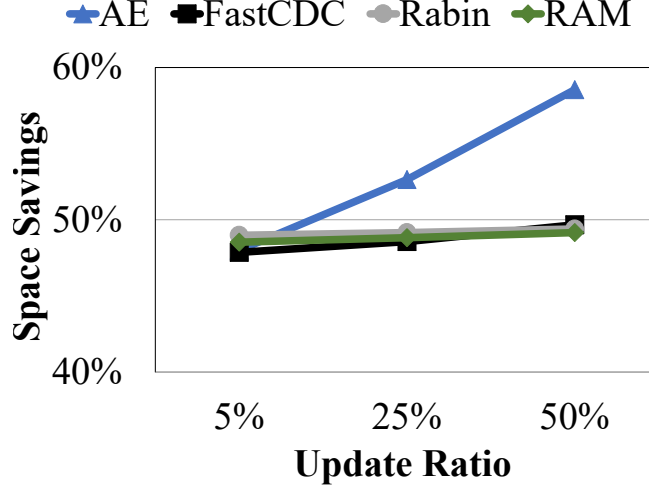
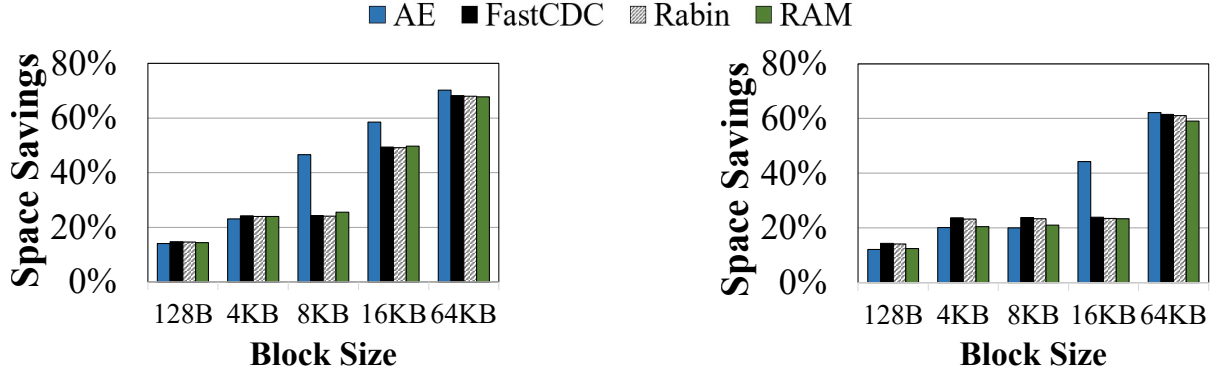


Figure 4.3: Impact of zero-entropy update ratio on space savings.

Figure 4.4 shows the space saving ratios that CDC algorithms achieve with increasing block sizes for three different average chunk size configurations: 4 KB, 8 KB, and 16 KB. Note that the zero-entropy update ratio was fixed at 50% for this experiment. The figure shows that CDC algorithms achieve higher space savings with increasing zero-entropy block size.

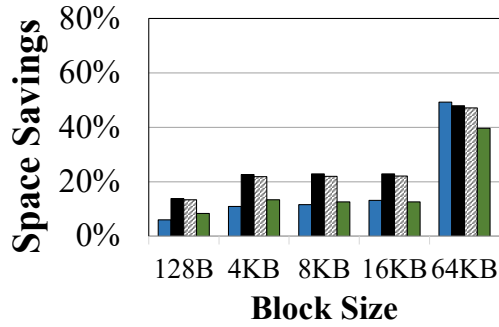
Across all three average chunk sizes, CDC algorithms identify and eliminate zero-entropy blocks more effectively as the block size increases. For instance, all CDC algorithms achieve a space saving ratio of $\sim 64\%$ in Figure 4.4a with a zero-entropy block size of 64 KB, but only 18% with a block size of 4 KB. Thus, modern CDC algorithms are poor at eliminating zero-entropy blocks with small block sizes.

I also note that some CDC algorithms start identifying and eliminating zero-entropy blocks when the block size is greater than or equal to the expected average chunk size, especially in AE. Once again, AE [44] can identify zero-entropy blocks at lower block sizes than other CDC algorithms. This is because the expected average chunk size determines how frequently a CDC algorithm places chunk boundaries. If a zero-entropy block is smaller than the average chunk size, it may be fully contained within a single chunk and cannot be detected. However, when the zero-entropy block is equal to or larger than the average chunk size, the CDC algorithm is more likely to consider a boundary near or inside the block, increasing the chance of isolating it. Thus, the ability of CDC algorithms to eliminate zero-entropy blocks depends on the expected average chunk size.



(a) 4 KB avg chunk size.

(b) 8 KB avg chunk size.



(c) 16 KB avg chunk size.

Figure 4.4: Impact of inserted zero-entropy block size on space savings of CDC algorithms.

Finally, I note an interesting trend when comparing hash-based and hashless algorithms with varying zero-entropy block sizes. At lower block sizes, hash-based algorithms, such as FastCDC and Rabin, achieve better space savings than hashless ones, such as AE and RAM. However, as the zero-entropy block size increases, this gap is quickly bridged. At higher block sizes, AE achieves the best space savings among all CDC algorithms, due to its ability to identify and eliminate low-entropy data [44]. Thus, the optimal CDC algorithm to eliminate zero-entropy blocks depends on their sizes.

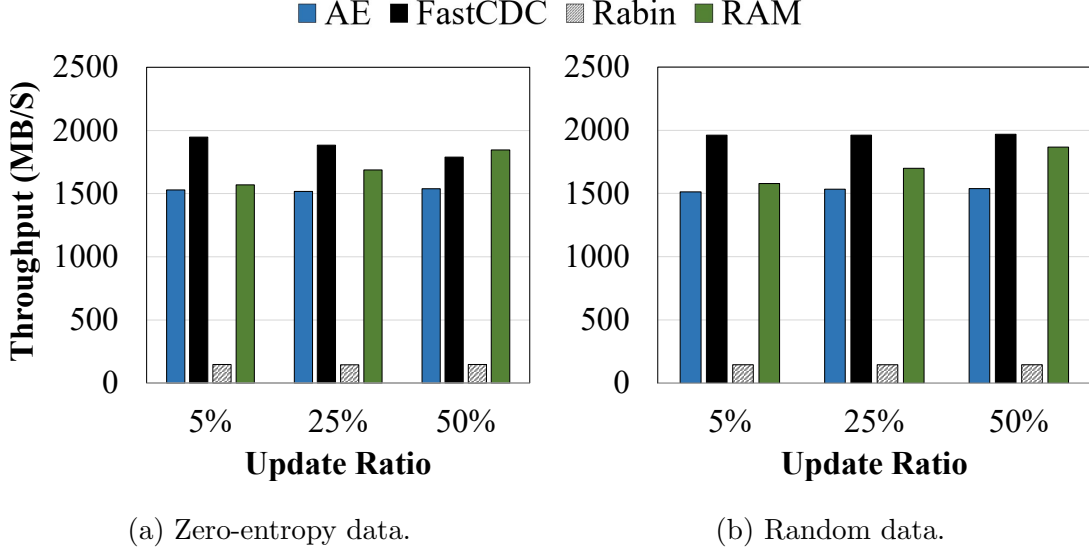


Figure 4.5: Chunking throughput comparison.

4.2.2 Chunking Throughput

Figure 4.5 shows the chunking throughput that all CDC algorithms achieve with different update ratios of random and zero-entropy blocks. The zero-entropy block size was set to 16 KB for these experiments. All results are the average of 5 runs, and the standard deviation was less than 5%.

All algorithms maintain their throughputs between equal ratios of zero-entropy (Figure 4.5a) and random (Figure 4.5b) data. FastCDC [41] achieves the highest chunking throughput among all the CDC algorithms, in line with previous studies using datasets with random data [41].

With increasing zero-entropy block ratios (Figure 4.5a), FastCDC [41] shows a slight decrease in chunking throughput. This is because it needs to scan larger amounts of data before finding chunk boundaries. The throughputs of other CDC algorithms remain stable across different update ratios. This is because their boundary detection logic involves fixed-cost, constant-time operations per byte, which do not depend on the type or amount of inserted data.

I tested different zero-entropy block sizes to see if they affect the throughputs of all CDC algorithms. The results showed no change in throughput. I do not include the figures because there are no interesting results, and to make the thesis more concise and avoid

clutter.

4.2.3 Chunk Size Distribution

In this section, I demonstrate the distributions in chunk sizes that all CDC algorithms perform under random and zero-entropy data. I use an update ratio of 50% and a 16 KB block size for these experiments.

Figure 4.6 shows the chunk size distributions all CDC algorithms exhibit under equal update ratios of random and zero-entropy data. Hashless CDC algorithms exhibit no differences in chunk size distributions between random and zero-entropy data. AE (Figure 4.6a) and RAM (Figure 4.6d) exhibit identical chunk size distributions in both cases. On the other hand, hash-based algorithms, such as FastCDC (Figure 4.6b) and Rabin’s chunking (Figure 4.6c) exhibit differences in chunk sizes between zero-entropy and random data insertion. Both algorithms produce larger chunk sizes on average when zero-entropy blocks are involved. Their chunk size variance is higher as well, which helps explain the lower space savings ratios observed in Figure 4.2.

While the chunk size distributions vary with update ratio and zero-entropy block size, the differences between hash-based and hashless algorithms remain the same as those outlined above. To ensure clarity, I only present the results for the block size and update ratio configuration above within the thesis.

4.2.4 Artificial Chunk Boundaries

Modern CDC techniques insert artificial chunk boundaries when content-defined boundaries cannot be determined within a maximum size limit. A higher percentage of artificial chunk boundaries results in lower deduplication space saving ratios. In this section, I examine the impact of zero-entropy blocks on artificially inserted boundaries. I use an update ratio of 50% and a zero-entropy block size of 16KB for this experiment.

Figure 4.7 shows the percentage of artificial chunk boundaries inserted by chunking techniques in the presence of random and zero-entropy data. The figure shows that hash-based CDC algorithms exhibit a higher percentage of artificially inserted chunk boundaries in the presence of zero-entropy data insertion.

This helps explain the space savings differences observed in Figure 4.2, as artificial chunk boundaries reduce deduplication effectiveness. This occurs because the rolling hash value fails to match the target value across the zero-entropy blocks, as they are composed of

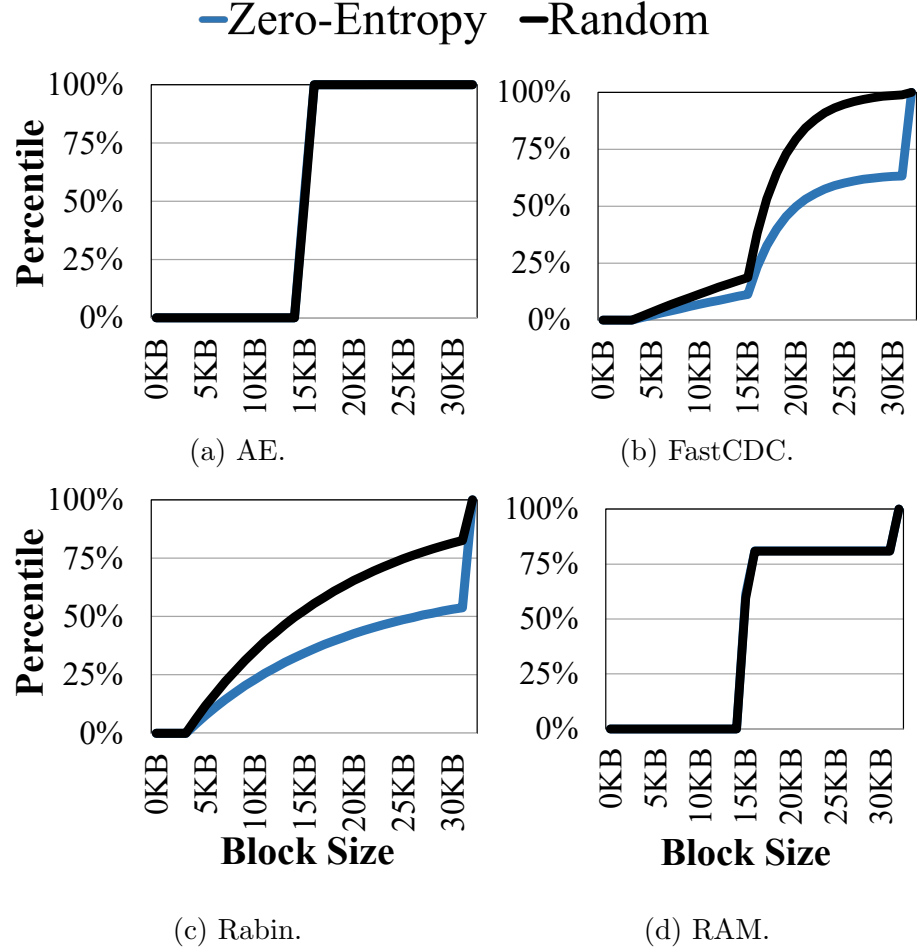


Figure 4.6: CDF of chunk size distributions under zero-entropy vs random data insertions.

repeating byte sequences. Hashless CDC algorithms do not suffer from the same drawback. For example, as Figure 4.7 illustrates, AE and RAM exhibit nearly identical percentages of artificial chunk boundaries under both random and zero-entropy data, while FastCDC and Rabin show a significant increase when processing zero-entropy data.

Figure 4.8a shows the impact of zero-entropy update ratio on the artificial chunk boundary percentages. With increasing zero-entropy update ratios, all CDC algorithms exhibit higher percentages of artificial chunk boundaries. In particular, hash-based algorithms have higher artificial boundaries compared to their hashless counterparts. For example, all algorithms produce more artificial boundaries with increasing update ratio, but the effect

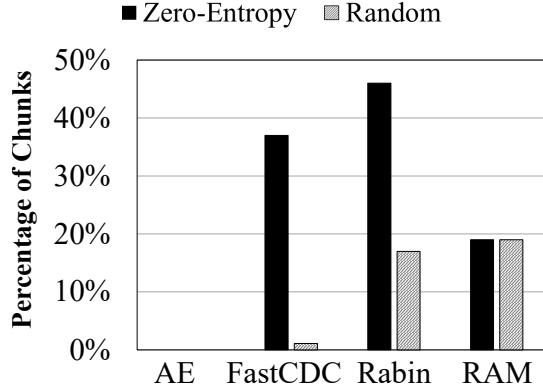


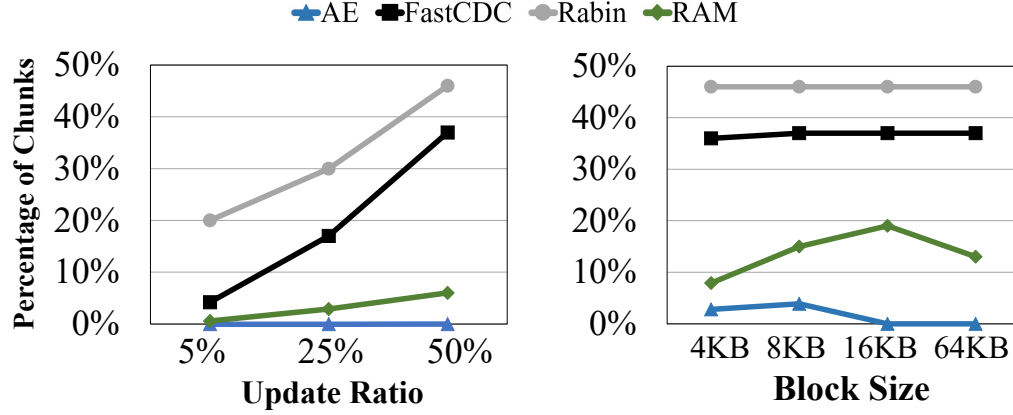
Figure 4.7: Artificial chunk boundaries under random and zero-entropy data insertion.

is much more pronounced in hash-based algorithms such as FastCDC and Rabin. At a 50% update ratio, Rabin exceeds 40% artificial boundaries and FastCDC reaches 38%, while AE and RAM remain below 10%.

For the block size effect, Figure 4.8b shows the impact of zero-entropy block size on the artificial chunk boundary percentages. It suggests that zero-entropy block sizes impact CDC algorithms in differing ways. Hash-based algorithms once again suffer from much higher artificial chunk boundaries compared to their hashless counterparts, regardless of zero-entropy block size. For example, FastCDC and Rabin have a consistently high rate of artificial boundary percentages with 38% and 46%, respectively, across all zero-entropy block sizes. In contrast, AE and RAM maintain low artificial boundary rates regardless of block size.

All four CDC algorithms, which are AE, FastCDC, Rabin, and RAM, have been implemented in C++ using 150 – 300 lines of code each and integrated into DedupBench [18], which I use to benchmark their performance. To detect duplicate chunks, I use SHA-1 [2] as the collision-resistant hash function. The fingerprints generated by SHA-1 are compared to detect identical chunks. For the implementation, I use the SHA-1 function from the OpenSSL `libssl` library [23]. All of these implementations, including modifications made to DedupBench, are publicly available in the project GitHub repository¹. For the synthetic dataset, I use a Python tool to produce synthetic files composed of strings that mix random and zero-entropy regions, with user-defined parameters for dataset size, block size, and update ratio.

¹<https://github.com/UWASL/dedup-bench>



(a) Update ratio impact.

(b) Block size impact.

Figure 4.8: Artificial Chunk Boundary Comparison.

4.3 Summary

In this chapter, I study how zero-entropy blocks affect the performance of modern CDC algorithms used in data deduplication. I evaluate both hash-based algorithms (FastCDC and Rabin) and hashless ones (AE and RAM), using data with inserted random and zero-entropy blocks. I use synthetic datasets to control the amount of zero-entropy and random data, allowing for better comparisons. The evaluation is based on metrics such as space savings, chunking throughput, chunk size distribution, and the percentage of artificial boundaries.

I summarize the main findings in this chapter as follows:

- All CDC algorithms perform better in space savings on zero-entropy data than random data.
- AE consistently achieves the best space savings, especially with high zero-entropy ratios.
- CDC algorithms are poor at eliminating zero-entropy blocks with small block sizes.
- The ability of CDC algorithms to eliminate zero-entropy blocks is dependent upon the expected average chunk size.
- The optimal CDC algorithm to eliminate zero-entropy blocks depends on their sizes.

- FastCDC leads in throughput, although it slightly slows with more zero-entropy, while hashless algorithms remain stable.
- Hash-based algorithms (FastCDC and Rabin) show greater chunk size variance and more artificial boundaries, reducing their deduplication effectiveness in zero-entropy conditions.
- Hashless algorithms (AE and RAM) maintain low artificial boundary rates and consistent chunk sizes, making them more resilient to zero-entropy patterns.

Chapter 5

ZERO: A System for Zero-Entropy Region Optimization in Data Deduplication

This chapter introduces ZERO (Zero-Entropy Region Optimization), a system that improves deduplication performance by identifying and extracting zero-entropy blocks prior to the deduplication process. The chapter details the design, implementation, and evaluation of the ZERO system.

5.1 Design

Figure 5.1 shows the components and the workflow of the ZERO system. The process begins with input data that includes both high-entropy (random bytes) and zero-entropy blocks. The data first enters a pre-screening stage that scans the input to detect zero-entropy blocks. If there are no zero-entropy blocks, the data flows directly into the deduplication process, ignoring the zero-entropy optimization process. However, when the pre-screening detects zero-entropy blocks, ZERO proceeds to the zero-entropy extraction stage.

The zero-entropy extraction process separates zero-entropy blocks from the input stream. This process produces two distinct outputs: zero-entropy data, consisting of the extracted zero-entropy blocks, and filtered data, containing the remaining non-zero-entropy content after those blocks are removed. The system then encodes the extracted zero-entropy blocks

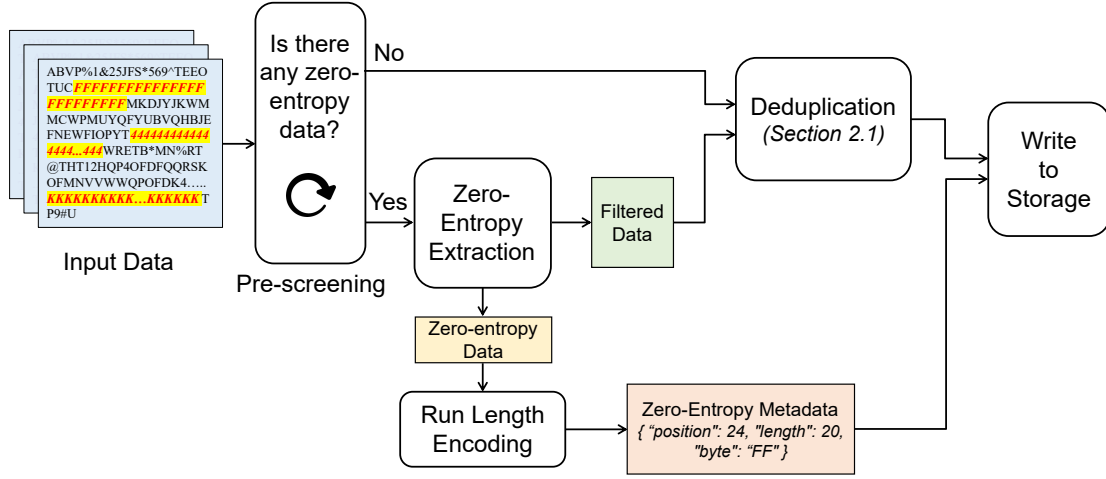


Figure 5.1: Overview of ZERO System.

using run-length encoding (RLE) [11] into compact metadata. This encoding process creates zero-entropy metadata that stores essential information for each zero-entropy block. The metadata includes the position indicating the starting location, the length representing the number of repeated bytes, and the byte value specifying the repeated character.

After extracting and encoding zero-entropy blocks, ZERO passes the filtered data to the standard deduplication workflow (Section 2.1), including chunking, hashing, and identifying duplicates through fingerprint indexing. The system then writes both the deduplicated chunks and the zero-entropy metadata to storage. This ensures that all necessary data is preserved for later data reconstruction.

ZERO provides several advantages, where it enhances deduplication performance by eliminating repetitive data that consumes CPU resources during fingerprinting, improves chunking by eliminating patterns that interfere with CDC boundary detection, and increases space efficiency through RLE.

Figure 5.2 shows how ZERO reconstructs the original data without loss. The reconstruction process involves several stages to restore the original file structure. The system begins with deduplicated chunks that were processed through the standard deduplication workflow. At the same time, the system decodes the encoded zero-entropy blocks back into the original sequences.

The reassembly engine combines both the deduplicated chunks and the decoded zero-entropy blocks. The run-length decoding process reconstructs the zero-entropy blocks from metadata by expanding compact representations (e.g., position: 24, length: 20, byte: FF)

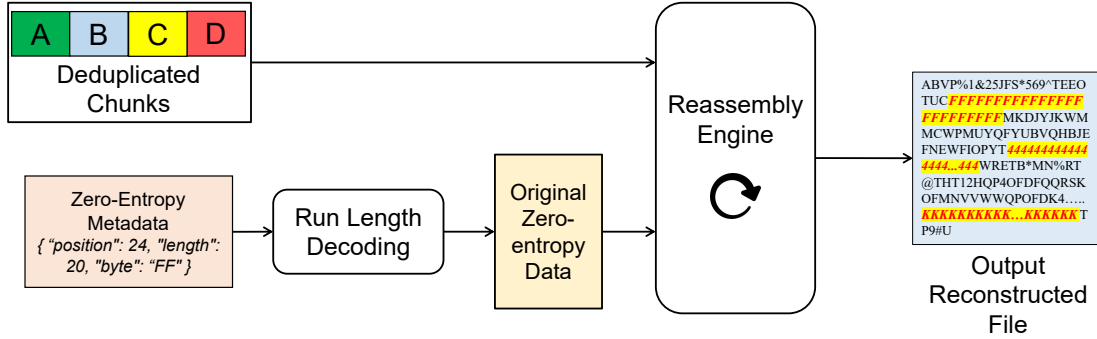


Figure 5.2: Data Reconstruction Workflow.

into their original sequences. The reassembly engine then integrates decoded zero-entropy blocks and the deduplicated chunks back into their exact original positions, ensuring the final reconstructed output is identical to the original input data.

Implementation. I designed a system that extracts zero-entropy blocks from files and stores them separately. This step removes repeated byte sequences before the deduplication stage. The extracted zero-entropy blocks are compressed using RLE. The system then stores metadata for each block, including its position, size, and byte value, which can be used later during data restoration. After removing zero-entropy blocks, the remaining data, typically higher entropy content, is passed to the deduplication pipeline.

5.2 Evaluation

I evaluate the performance of the ZERO system against state-of-the-art deduplication and compression systems. The experiments evaluate the effectiveness of ZERO in terms of space savings, throughput, and processing overhead, using various state-of-the-art CDC algorithms and real-world datasets.

Testbed. I conducted all my experiments on a c6525-25g machine from CloudLab Utah [5]. The machine has a 16-core AMD EPYC 7302P processor with hyperthreading, 128 GB of RAM, and a 25 Gbps Mellanox NIC. I use Ubuntu 22.04, gcc v11.4.0, and libssl v3.0.2 to deploy the code.

Metrics. To evaluate the effectiveness of the ZERO system, I use the following metrics:

1. **Space Savings:** The percentage of storage space saved by using a data reduction technique, including deduplication, compression, or/and ZERO. It is calculated as:

$$\text{Space Savings} = \frac{\text{Before Size} - \text{After Size}}{\text{Before Size}} \times 100 \quad (5.1)$$

where **Before Size** refers to the original input size before applying the data reduction technique, and **After Size** refers to the final size after applying the data reduction technique.

2. **Throughput:** The overall throughput of the data reduction technique. The throughput includes all stages, such as preprocessing (e.g., zero-entropy extraction), chunking, hashing, and any compression or post-processing steps.
3. **Processing Time:** An analysis of processing time for each stage of the pipeline, including zero-entropy extraction for the ZERO, chunking and hashing stages.

Alternatives. I compare ZERO against different data reduction techniques implemented in various combinations:

1. **Compression:** I apply standard lossless compression (e.g., GZIP) directly to the raw dataset without deduplication.
2. **Deduplication:** I apply deduplication using different CDC algorithms, AE, FastCDC, Rabin, and RAM, without any additional preprocessing.
3. **Deduplication + Compression:** I apply deduplication first (using the same CDC algorithms as above), followed by compression of the deduplicated data.
4. **ZERO:** My proposed system, where I apply zero-entropy extraction before deduplication.

Dataset. I evaluate the performance of each technique on the same three real-world datasets in Table 3.1, including VDI, LNX, and CORD-19.

5.2.1 Comparison of Data Reduction Techniques (Space Savings vs. Throughput)

This section compares the effectiveness of the four alternatives in terms of both space savings and throughput. I evaluate three different average chunk sizes: 4 KB, 8 KB, and

16 KB. For compression, I use the GZIP algorithm [9]. For clarity, I show only the results for AE using an average chunk size of 8 KB to make the thesis concise. The results of other CDC algorithms and chunk sizes show similar conclusions.

The goal of this experiment is to evaluate the performance of space savings and end-to-end throughput that each technique achieves. Figure 5.3 shows the results for all alternatives and datasets. The results demonstrate that ZERO achieves the best tradeoff between space savings and throughput across all datasets. While compression techniques achieve higher space savings across all datasets, they suffer from low throughput. On the other hand, state-of-the-art deduplication systems maintain higher throughput, but provide fewer space savings. ZERO offers higher space savings than deduplication alone while keeping throughput comparable or slightly higher than others.

Across all datasets, ZERO achieves the best trade-off between space savings and throughput. In the VDI dataset (Figure 5.3a), ZERO improves space savings by 11% over deduplication while maintaining similar throughput. Compared to compression techniques, ZERO achieves 25x higher throughput; however, it shows lower space savings by 14%. In the LNX dataset (Figure 5.3b), ZERO improves space savings by 5% over deduplication while maintaining high throughput and achieves over 18 $\times$ higher throughput than compression. Also, ZERO achieves 18 $\times$ higher throughput than compression, while it achieves lower space savings by 30%. For the CORD-19 dataset (Figure 5.3c), deduplication is ineffective. This is because the dataset was cleaned of redundancy during collection and contains a few duplicate chunks [32], but ZERO achieves 29% space savings and presents a comparable throughput. Compared to compression, ZERO maintains over 24 $\times$ higher throughput with lower space savings of 50%.

The benefit of ZERO is also dependent on the dataset characteristics. It performs better on datasets containing higher percentages of zero-entropy, such as VDI and CORD-19. Even when redundancy is minimal, as CORD-19 shows, ZERO achieves more space savings where deduplication fails. In LNX, ZERO provides fewer benefits due to less zero-entropy data, but ZERO still outperforms compression-based techniques in throughput.

Overall, deduplication-based techniques maintain high throughput but deliver lower space savings, while compression-based techniques achieve higher space savings but suffer from very low throughput. ZERO combines the strengths of both without suffering their trade-offs.

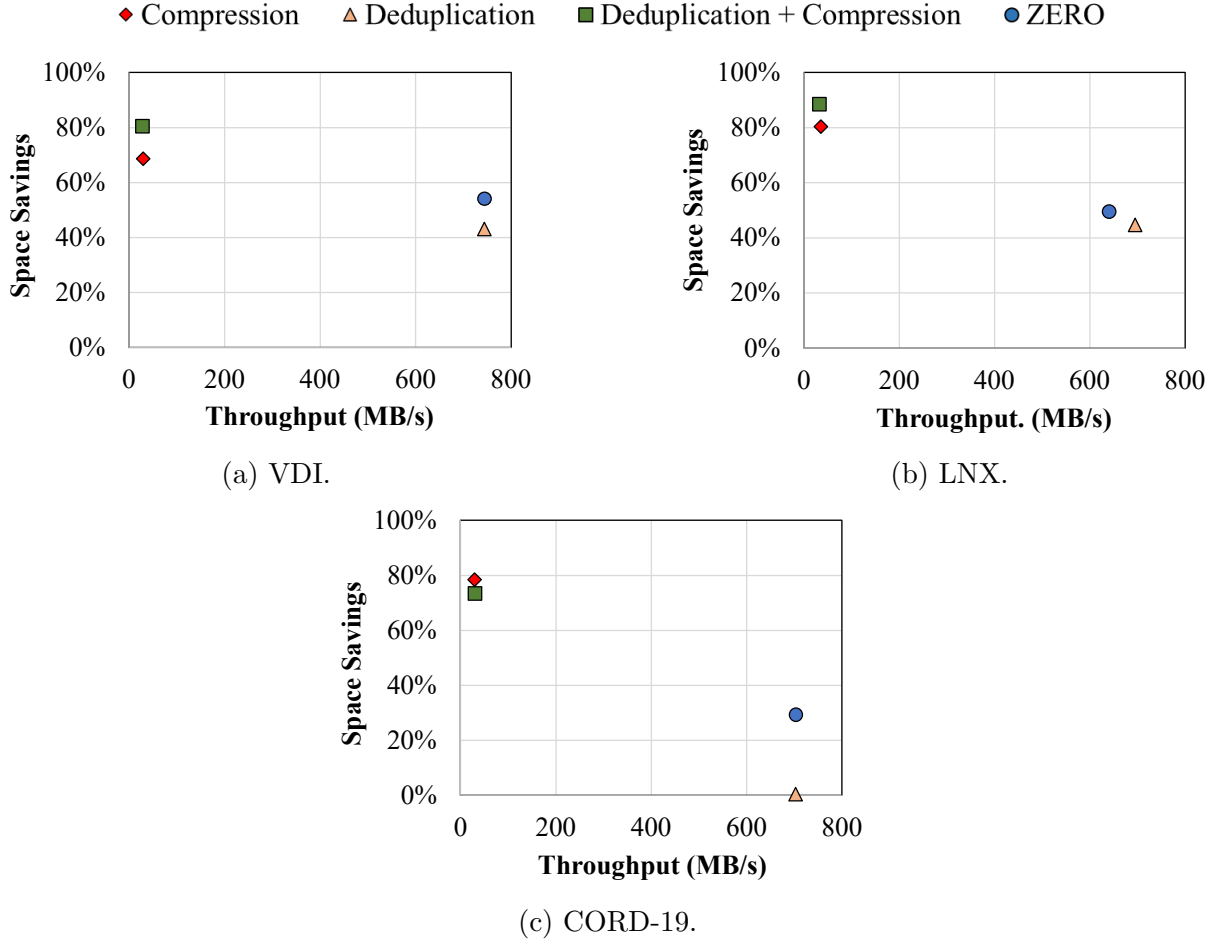


Figure 5.3: Comparison of data reduction techniques across three datasets in terms of space savings and end-to-end throughput.

5.2.2 Space Savings

In this section, I evaluate the storage space savings of ZERO compared to state-of-the-art deduplication systems, using an average chunk size of 8 KB across four CDC algorithms. Although I also evaluate ZERO with chunk sizes of 4 KB and 16 KB, I focus on the 8 KB case to avoid redundancy, as the results for different chunk sizes follow the same patterns.

Figure 5.4 presents the space saving ratios for all datasets and CDC algorithms. ZERO improves space savings across all CDC algorithms. It improves space savings by 6–29 %

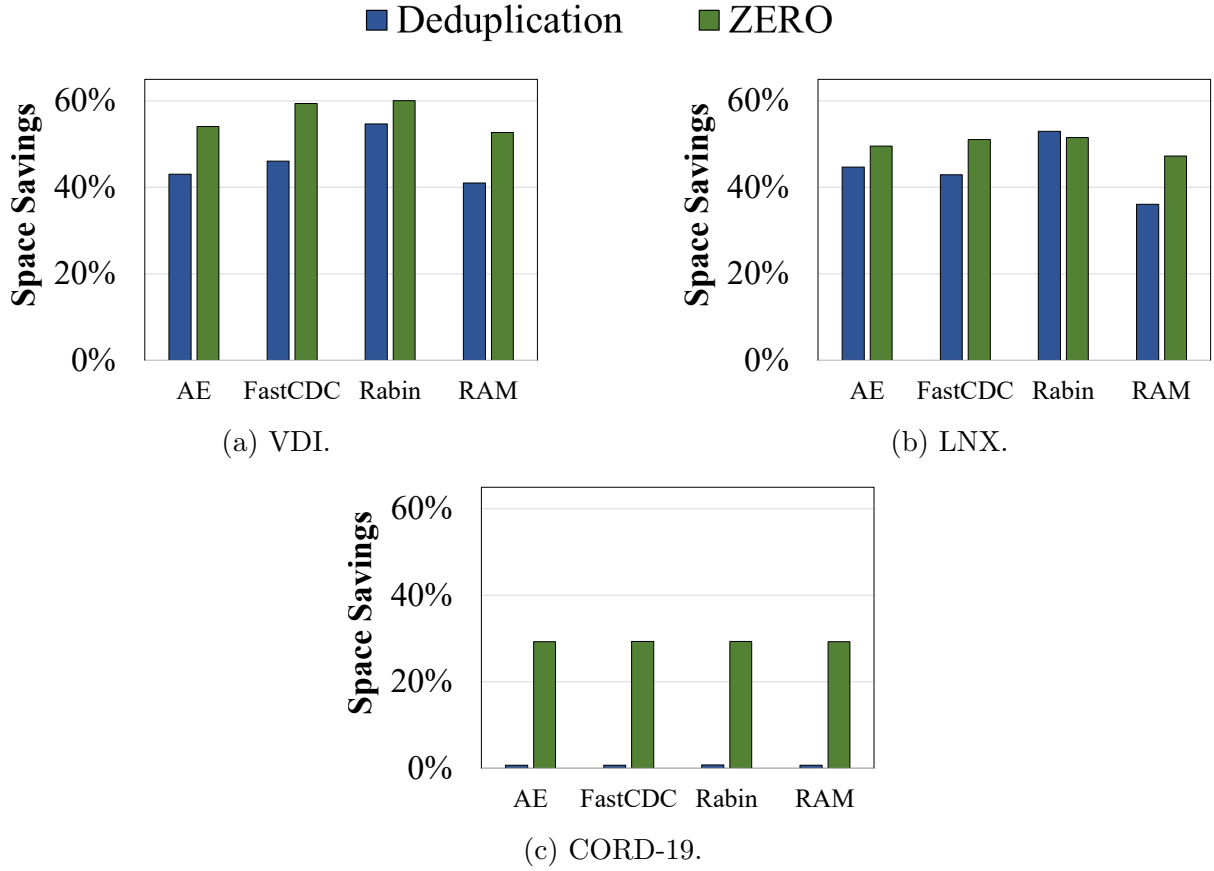


Figure 5.4: Space savings of ZERO compared to state-of-the-art deduplication on three datasets: VDI, LNX, and CORD-19.

points across datasets. In the VDI dataset (Figure 5.4a), ZERO outperforms deduplication across all CDC algorithms. With AE, space savings are 11% higher, while FastCDC presents the largest improvements at 13%. RAM also benefits with a gain of an 11% gain, and Rabin shows a smaller but still measurable improvement of 5%. In the LNX dataset (Figure 5.4b), the benefit of ZERO is still present, although the gains are smaller due to the lower zero-entropy data. AE improves by 5%, while FastCDC achieves 8% more space savings. RAM shows the largest increase in this dataset with 9%. In contrast, Rabin performs slightly worse with ZERO, showing about 1% lower space savings compared to deduplication. In the CORD-19 dataset (Figure 5.4c), ZERO shows the most noticeable improvement. It provides 29% space savings across all CDC algorithms, while deduplica-

tion fails to achieve space savings, because this dataset has very low duplicate contents.

My experiments show that extracting and encoding zero-entropy blocks before chunking improves space savings across all datasets and using all CDC algorithms except the Rabin algorithm in the LNX dataset, where it has lower zero-entropy data. This is especially helpful for datasets with a higher percentage of zero-entropy data, but it also brings noticeable benefits in less redundant datasets [32]. Even a small amount of zero-entropy data leads to better storage efficiency. This highlights that ZERO is an effective way to optimize deduplication efficiency.

5.2.3 End-to-end Evaluation

This section evaluates the processing time across the deduplication pipeline, including the zero-entropy extraction step in the ZERO system. I conduct the experiments using three average chunk sizes (4 KB, 8 KB, and 16 KB), but since the results for all sizes show similar patterns, I show only the results for the 8 KB, as it represents the overall patterns of the results. I compare two configurations: with ZERO (zero-entropy extraction, chunking, and hashing) and without ZERO (chunking and hashing only).

Figure 5.5 shows the processing time for all datasets and CDC algorithms. The results demonstrate that the deduplication pipeline with ZERO performs as fast or faster than the pipeline without ZERO in most cases, especially in datasets that contain more zero-entropy data.

For AE (Figure 5.5a), the results vary across datasets. On the VDI dataset, both with and without ZERO take the same time. On LNX, which has little zero-entropy data, ZERO adds a small overhead of 5%. In contrast, CORD-19 benefits from ZERO, completing 9% faster.

FastCDC (Figure 5.5b) shows a similar pattern. On VDI and CORD-19, the total processing times are the same for both with and without ZERO. For LNX, ZERO adds some overhead, making it about 18% slower than without ZERO, for the same reason as before, where LNX has less zero-entropy data.

The most significant improvements appear with Rabin (Figure 5.5c), where ZERO substantially reduces processing time. On the VDI dataset, using ZERO completes deduplication 24% faster than without ZERO, primarily due to reduced chunking time. The LNX dataset also shows a moderate reduction of 7% in total processing time when using ZERO. For the CORD-19 dataset, the improvement is large, with ZERO achieving 24% faster. This demonstrates that Rabin, with its higher chunking overhead [20], benefits significantly from the zero-entropy extraction process.

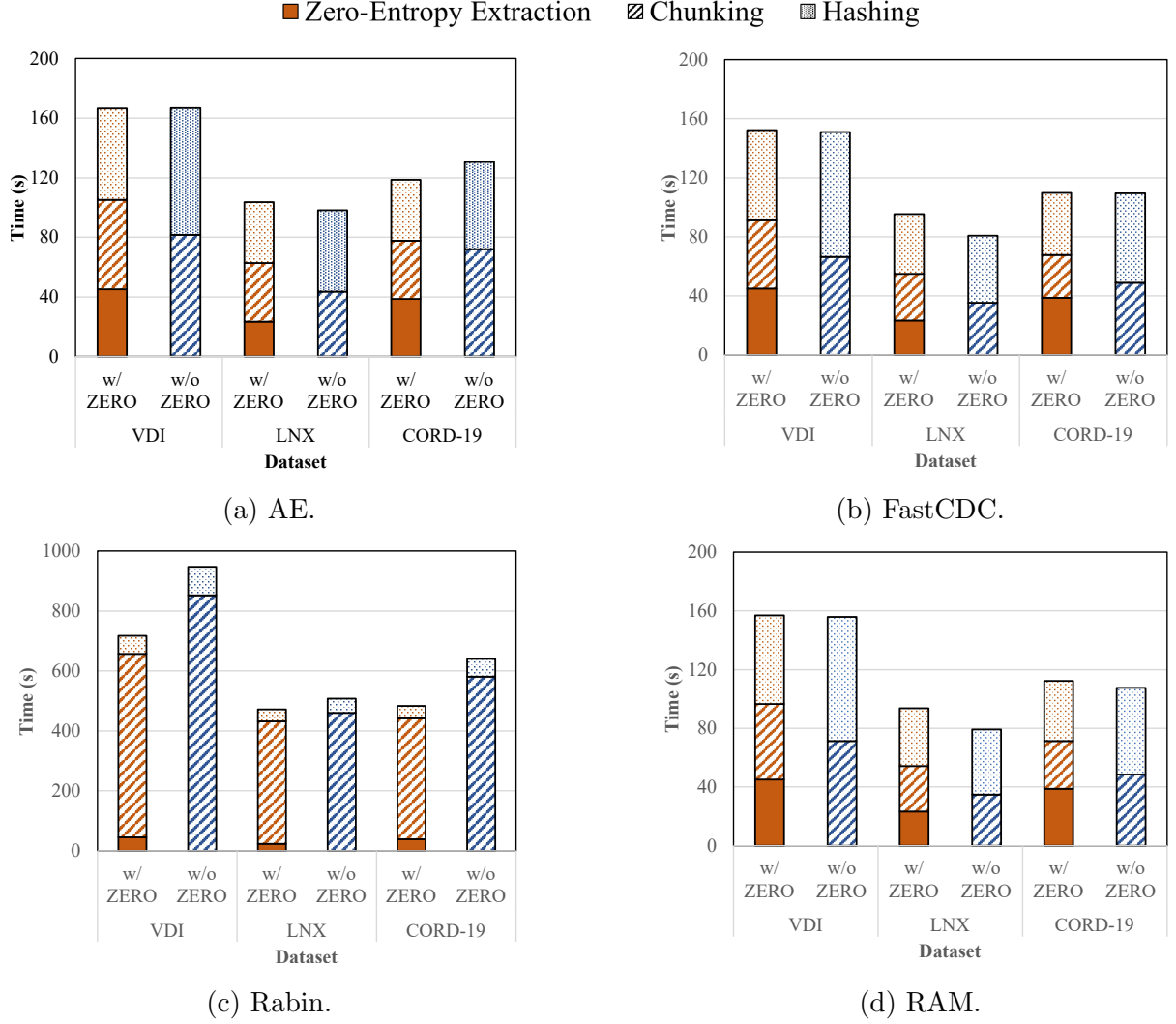


Figure 5.5: Processing time breakdown of the deduplication pipeline across all datasets and CDC algorithms. Each group shows the times with and without ZERO.

RAM (Figure 5.5d) shows different characteristics in all datasets. On VDI, the total processing times are similar. On the LNX dataset, the pipeline with ZERO is slightly slower, following the general trend of low zero-entropy ratios for this dataset. For CORD-19, the difference is small, with the version without ZERO being slightly faster.

Among all datasets, the results show that the VDI dataset benefits the most from

ZERO, then CORD-19. The LNX dataset shows moderate improvements across CDC algorithms. Overall, from the end-to-end evaluation, I observe that removing zero-entropy data saves time, even though it adds some overhead at the beginning. This zero-entropy extraction step speeds up data deduplication, or at least maintains comparable performance. Rabin achieves the greatest improvements because the zero-entropy extraction process reduces the chunking overhead that Rabin introduces.

Chapter 6

Related Work

This chapter reviews previous research related to this work, including studies on improving chunking performance and data deduplication efficiency, and provides an overview of data compression techniques.

6.1 Chunking Performance Improvement

This section summarizes efforts to improve the performance of the chunking phase in deduplication systems.

RapidCDC [22] accelerates chunk boundary detection by leveraging the locality of duplicate data. It predicts the next chunk boundary using hints learned from previously seen duplicate chunks, instead of scanning byte-by-byte. When a duplicate chunk is detected, RapidCDC checks a small set of possible chunk sizes associated with the fingerprint of that chunk to jump directly to the likely boundary positions. This avoids unnecessary rolling and accelerates chunking. It also includes fallback strategies when predictions fail, switching back to regular rolling when needed.

QuickCDC [42] improves chunking speed by combining fast skipping techniques with traditional CDC. It uses a feature vector of the first and last few bytes of a chunk as signatures to identify and skip over duplicate chunks quickly. If both the front and end feature vectors match, QuickCDC assumes the chunk is a duplicate and jumps directly to the next boundary, avoiding byte-by-byte scanning. In the case of unique chunks, where feature vector matching fails, it skips a fixed minimum chunk length and resumes normal CDC scanning, which reduces unnecessary processing. QuickCDC [42] uses dynamic mask

adjustment to control chunk size and avoid very large or very small chunks, which balances speed and deduplication accuracy.

MUCH [35] is a variable-size chunking technique that parallelizes chunking using multiple threads. It leverages multicore CPUs for file chunking. It splits each file into smaller segments and assigns them to parallel chunker threads, which independently process their segments to find chunk boundaries. MUCH ensures that chunk boundaries remain consistent no matter how many threads are used by starting chunking in a slower, careful mode, then switching to faster processing after a reliable boundary is detected. It also includes a cleanup phase, which adjusts any chunks that are too small or too large to make sure all chunks follow the expected size limits. For small files where using many threads is not efficient, it applies dynamic segment prefetching to group data before assigning it to threads, thereby improving performance.

P-Dedupe [39] is a deduplication system built to eliminate the write bottleneck in storage systems based on parallel pipelining for deduplication tasks. It runs all of the deduplication stages, chunking, fingerprinting, indexing, and writing, concurrently using multiple CPU cores. This gives better resource utilization and reduces latency. Due to the challenges of data dependencies in the parallelization of CDC, P-Dedupe solves that by dividing the input into large sections and processing them in parallel. Also, it applies boundary rechecking to ensure chunking correctness, supports parallel fingerprinting, and uses a hybrid synchronization approach to keep correct chunk ordering without stalling the pipeline.

Minimal Incremental Interval (MII) [43] is another CDC algorithm designed for incremental file synchronization. It defines chunk boundaries based on patterns of increasing byte values between adjacent bytes in the data stream, with no need to use rolling hashes. Once a sequence of increasing values reaches a predefined length, a chunk boundary is marked. This allows MII to detect changes efficiently and maintain boundary stability even under byte insertions, deletions, or shifts. MII operates in linear time with constant space, enabling efficient detection of changes across file versions. It performs well under byte insertions, deletions, and shifts, but exhibits low throughput and inflexible chunk sizes. Its primary design goal is synchronization rather than deduplication, and it does not address repetitive or uniform data scenarios.

Unlike all previous work in this section, which focuses on accelerating chunking through prediction, parallelism, and adaptive boundary detection, these approaches often assume regular data patterns or focus on general performance. As a result, they do not consider the impact of zero-entropy content on chunking accuracy, consistency, or efficiency. In contrast, my thesis investigates the behavior and limitations of chunking algorithms specifically in

the presence of zero-entropy.

6.2 Enhancing Deduplication Efficiency

This section summarizes some efforts to increase the efficiency of data deduplication by optimizing other deduplication phases, such as hashing and indexing.

StoreGPU [3] uses GPU acceleration to improve the performance of hash-based operations in storage systems. StoreGPU improves throughput by offloading hash computations to the GPU. It achieves up to $8\times$ speedup compared to CPU-based hashing. This enables more efficient similarity detection, though it primarily targets hashing rather than chunking.

SiLo [37] presents a scalable deduplication system by combining data locality and similarity to improve indexing efficiency in large-scale backup systems. It segments large files and groups similar small files to enhance duplicate detection, and it leverages locality to reduce index lookups on the disk, which minimizes RAM usage. SiLo uses stateless routing to distribute data across backup nodes, which improves throughput and memory efficiency.

HYDRASor [4] is a distributed storage system that handles backup and archival workloads. It uses a grid architecture of storage and access nodes, with a modified distributed hash table to manage data placement and resiliency. HYDRASor supports inline, global deduplication with variable-size blocks, fault-tolerance erasure coding, and load balancing based on the entropy of the system. HYDRASor chunks data and stores it using variable-sized blocks, and deduplication is performed through hash-based verification. HYDRASor maintains high availability using self-healing and online data rebuilding after failures. Its design optimizes storage utilization, deletion, and data recovery throughout the system.

Sparse Indexing [16] introduces a scalable deduplication approach to prevent full chunk indexing bottlenecks. It divides data into chunks and groups them into larger segments. It uses an efficient way to sparse the index to identify similar segments. Sparse Indexing [16] compares only a small number of segment manifests to detect and eliminate duplicate chunks. This reduces RAM usage and disk access and provides high deduplication efficiency.

Unlike all previous work in this section, which focuses on improving the efficiency of deduplication through indexing optimization, data routing, and hardware acceleration, my thesis focuses on optimizing deduplication efficiency in the presence of zero-entropy data.

6.3 Data Compression

Lossless data compression reduces the size of a file while still keeping its original information. It reduces file size by encoding repeated patterns. For example, if a file has the same word multiple times, compression records it once and notes the number of times it repeats. This makes compression work well for individual files.

The idea behind compression is that the more repeated the data is, the more it can be compressed. For example, repeating the same letter many times is easier to compress than repeating random text. Methods such as Huffman coding [12] use frequency-based rules to shorten common symbols, and others, such as LZ77 [45], find repeating strings and replace them with shorter references.

One of the popular compression methods is GZIP [9], which is a lossless compression method. It combines both LZ77 for finding repeated patterns in the data, and Huffman coding for shortening them. GZIP works using a two-stage approach. It uses LZ77 and Huffman coding. It first looks for repeated byte patterns and replaces them with shorter symbols. Then, it gives the most common patterns fewer bits, and the rare ones more bits. This reduces the overall size, but the original data can still be restored.

While compression techniques save large spaces, they add overhead, which makes them orthogonal to my work in this thesis, where I focus on data deduplication techniques.

6.4 Delta Compression

DARE [38] 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. Also, Ddelta [40] is a fast deduplication-inspired delta compression method, improves the speed of both delta encoding and decoding through deduplication concepts while maintaining the same compression ratio. While I can use delta compression to compress the zero-entropy data in my work, its slow speed makes it orthogonal to my work as I focus on optimizing the performance of data deduplication in the context of zero-entropy data.

Chapter 7

Conclusion and Future Work

The explosive growth of digital data and the high redundancy in cloud storage make efficient deduplication essential. Although numerous studies target CDC algorithms for data deduplication, they focus on datasets with general random data. However, zero-entropy data blocks are widespread in some production datasets such as virtual machine disk images, source code repositories, and scientific archives. Despite their prevalence, not much attention has been devoted to studying and understanding their behavior on CDC algorithms for deduplication systems.

In this thesis, I present a characterization study of zero-entropy data across three real-world datasets and evaluate its impact on four state-of-the-art CDC algorithms. The results show that modern CDC algorithms often fail to identify and handle zero-entropy blocks, especially with small sizes. In addition, their ability to handle zero-entropy blocks depends on the expected average chunk size. The choice of optimal algorithm to handle zero-entropy blocks is highly dependent on their sizes. All of these together highlight the need for a new approach to detect and extract zero-entropy blocks during the data deduplication process.

I design ZERO, a system that leverages zero-entropy blocks to optimize space savings in deduplication. ZERO detects and extracts zero-entropy blocks before the deduplication process. ZERO encodes these blocks using RLE, stores their metadata, and then passes the reduced data stream to the CDC algorithm for chunking. Finally, it stores their metadata for later use in the restoration phase. The results show that ZERO improved space savings by 6–29% across real-world datasets. In addition, ZERO presents comparable or slightly higher throughput than state-of-the-art deduplication systems. It also reduces end-to-end processing time, with up to 24% speedup while being up to 25× faster than traditional

compression methods.

While the current work focuses on zero-entropy data, I plan to extend this work by extending it to detect and extract other low-entropy patterns containing more irregular repetition.

References

- [1] Speed up your wordpress website with wp rocket. <https://wp-rocket.me/>, 2025. Accessed: 2025-06-10.
- [2] Donald E. Eastlake 3rd and Paul Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174, September 2001.
- [3] Samer Al-Kiswany, Abdullah Gharaibeh, Elizeu Santos-Neto, George Yuan, and Matei Ripeanu. Storegpu: exploiting graphics processing units to accelerate distributed storage systems. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, HPDC '08, page 165–174, New York, NY, USA, 2008. Association for Computing Machinery.
- [4] Cezary Dubnicki, Leszek Gryz, Lukasz Heldt, Michal Kaczmarczyk, Wojciech Kilian, Przemyslaw Strzelczak, Jerzy Szczepkowski, Cristian Ungureanu, and Michal Welnicki. Hydrastor: a scalable secondary storage. In *Proceedings of the 7th Conference on File and Storage Technologies*, FAST '09, page 197–210, USA, 2009. USENIX Association.
- [5] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.
- [6] Ahmed El-Shimi, Ran Kalach, Ankit Kumar, Adi Oltean, Jin Li, and Sudipta Sen-gupta. Primary data deduplication-large scale study and system design. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, page 26, USA, 2012. USENIX Association.

- [7] Kave Eshghi and Hsiu Khuern Tang. A framework for analyzing and improving content-based chunking algorithms. *Hewlett-Packard Labs Technical Report TR*, 30(2005), 2005.
- [8] Min Fu, Dan Feng, Yu Hua, Xubin He, Zuoning Chen, Wen Xia, Yucheng Zhang, and Yujuan Tan. Design tradeoffs for data deduplication performance in backup workloads. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, FAST’15, page 331–344, USA, 2015. USENIX Association.
- [9] Jean-Loup Gailly and Mark Adler. The gzip compressor. <http://www.gzip.org/>, 1991. Accessed: 2025-06-10.
- [10] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP ’03, page 29–43, New York, NY, USA, 2003. Association for Computing Machinery.
- [11] Solomon Golomb. Run-length encodings (corresp.). *IEEE transactions on information theory*, 12(3):399–401, 1966.
- [12] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.
- [13] IDC, Statista, and Various sources. Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2023, with forecasts from 2024 to 2028 (in zettabytes) [graph]. <https://www.statista.com/statistics/871513/worldwide-data-created/>, May 2024. Accessed: May 06, 2025.
- [14] Mu’men Al Jarah, Sreeharsha Udayashankar, Abdelrahman Baba, and Samer Al-Kiswany. The impact of low-entropy on chunking techniques for data deduplication. In *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, pages 134–140, 2024.
- [15] Chuck Lam. *Hadoop in action*. Simon and Schuster, 2010.
- [16] Mark Lillibridge, Kave Eshghi, Deepavali Bhagwat, Vinay Deolalikar, Greg Trezise, and Peter Camble. Sparse indexing: large scale, inline deduplication using sampling and locality. In *Proceedings of the 7th Conference on File and Storage Technologies*, FAST ’09, page 111–123, USA, 2009. USENIX Association.

- [17] Linux Kernel Organization. The linux kernel archives. <https://www.kernel.org/>, 2023. [Accessed 11-May-2025].
- [18] Alan Liu, Abdelrahman Baba, Sreeharsha Udayashankar, and Samer Al-Kiswany. Dedupbench: A benchmarking tool for data chunking techniques. In *2023 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 469–474, 2023.
- [19] Dutch T Meyer and William J Bolosky. A study of practical deduplication. *ACM Transactions on Storage (ToS)*, 7(4):1–20, 2012.
- [20] Athicha Muthitacharoen, Benjie Chen, and David Mazières. A low-bandwidth network file system. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP ’01, page 174–187, New York, NY, USA, 2001. Association for Computing Machinery.
- [21] MySQL. MySQL. <https://www.mysql.com/>, 2023. Accessed: 2025-06-17.
- [22] Fan Ni and Song Jiang. Rapidcdc: Leveraging duplicate locality to accelerate chunking in cdc-based deduplication systems. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC ’19, page 220–232, New York, NY, USA, 2019. Association for Computing Machinery.
- [23] OpenSSL Foundation. OpenSSL. <https://www.openssl.org/>, n.d. Accessed: 2025-05-19.
- [24] David Salomon. *Data compression*. Springer, 2002.
- [25] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’03, page 76–85, New York, NY, USA, 2003. Association for Computing Machinery.
- [26] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.
- [27] Vasily Tarasov, Amar Mudrankit, Will Buik, Philip Shilane, Geoff Kuenning, and Erez Zadok. Generating realistic datasets for deduplication analysis. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC’12, page 24, USA, 2012. USENIX Association.

- [28] Transaction Processing Council. TPC-C Overview. <https://www.tpc.org/tpcc/detail5.asp>, 2023. Accessed: 2025-06-17.
- [29] VirtualBoxes. Virtualboxes - free virtualbox(r) images. <https://sourceforge.net/projects/virtualboximage/>, 2024. [Accessed 11-May-2025].
- [30] VMWare. VMWare Marketplace. <https://marketplace.cloud.vmware.com/services>, 2023. Accessed: 2025-06-17.
- [31] Grant Wallace, Fred Douglass, Hangwei Qian, Philip Shilane, Stephen Smaldone, Mark Chamness, and Windsor Hsu. Characteristics of backup workloads in production systems. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST’12, page 4, USA, 2012. USENIX Association.
- [32] Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Douglas Burdick, Darrin Eide, Kathryn Funk, Yannis Katsis, Rodney Kinney, et al. Cord-19: The covid-19 open research dataset. *ArXiv*, pages arXiv–2004, 2020.
- [33] Ryan N.S. Widodo, Hyotaek Lim, and Mohammed Atiquzzaman. A new content-defined chunking algorithm for data deduplication in cloud storage. *Future Generation Computer Systems*, 71:145–156, 2017.
- [34] Wikipedia contributors. Information theory — Wikipedia, The Free Encyclopedia, 2025. [Online; accessed 30-April-2025].
- [35] Youjip Won, Kyeongyeol Lim, and Jaehong Min. Much: Multithreaded content-based file chunking. *IEEE Transactions on Computers*, 64(5):1375–1388, 2015.
- [36] Wen Xia, Hong Jiang, Dan Feng, Fred Douglass, Philip Shilane, Yu Hua, Min Fu, Yucheng Zhang, and Yukun Zhou. A comprehensive study of the past, present, and future of data deduplication. *Proceedings of the IEEE*, 104(9):1681–1710, 2016.
- [37] Wen Xia, Hong Jiang, Dan Feng, and Yu Hua. Similarity and locality based indexing for high performance data deduplication. *IEEE Transactions on Computers*, 64(4):1162–1176, 2015.
- [38] Wen Xia, Hong Jiang, Dan Feng, and Lei Tian. Combining deduplication and delta compression to achieve low-overhead data reduction on backup datasets. In *2014 Data Compression Conference*, pages 203–212, 2014.

- [39] Wen Xia, Hong Jiang, Dan Feng, Lei Tian, Min Fu, and Zhongtao Wang. P-dedupe: Exploiting parallelism in data deduplication system. In *2012 IEEE Seventh International Conference on Networking, Architecture, and Storage*, pages 338–347, 2012.
- [40] Wen Xia, Hong Jiang, Dan Feng, Lei Tian, Min Fu, and Yukun Zhou. Ddelta: A deduplication-inspired fast delta compression approach. *Performance Evaluation*, 79:258–272, 2014. Special Issue: Performance 2014.
- [41] Wen Xia, Yukun Zhou, Hong Jiang, Dan Feng, Yu Hua, Yuchong Hu, Yucheng Zhang, and Qing Liu. Fastcdc: a fast and efficient content-defined chunking approach for data deduplication. In *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '16, page 101–114, USA, 2016. USENIX Association.
- [42] Zhen Xu and Wenbo Zhang. Quickcdc: A quick content defined chunking algorithm based on jumping and dynamically adjusting mask bits. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 288–299, 2021.
- [43] Changjian Zhang, Deyu Qi, Zhe Cai, Wenhao Huang, Xinyang Wang, Wenlin Li, and Jing Guo. Mii: A novel content defined chunking algorithm for finding incremental data in data synchronization. *IEEE Access*, 7:86932–86945, 2019.
- [44] Yucheng Zhang, Hong Jiang, Dan Feng, Wen Xia, Min Fu, Fangting Huang, and Yukun Zhou. Ae: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1337–1345, 2015.
- [45] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.