ZERO: Zero-Entropy Region Optimization 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

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

***Abstract*—Data deduplication reduces storage requirements by identifying and eliminating duplicate data, typically using content-defined chunking (CDC) to divide data into chunks for duplicate detection. Some workloads contain zero-entropy regions, contiguous regions composed of a single repeated byte value, which can be efficiently compacted. However, deduplication pipelines still process these regions through chunking, fingerprinting, and index lookup. We propose ZERO (Zero-Entropy Region Optimization), a preprocessing layer that identifies and extracts qualifying regions before CDC while preserving metadata for exact reconstruction. We characterize these regions across four real-world datasets and evaluate ZERO with six CDC algorithms. Results show that ZERO achieves up to 30% payload-level space savings, while improving space savings from 46.08% to 59.07% and maintaining or reducing processing time in many configurations.**

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

I. Introduction

Cloud storage providers use data deduplication techniques [1] to improve storage efficiency and throughput in the face of ever-increasing storage demand. Data deduplication identifies duplicate content and stores only unique data [2], [3]. A typical deduplication pipeline first divides incoming data into chunks, computes a fingerprint for each chunk, and compares the fingerprint against an index of previously stored chunks. Content-defined chunking (CDC) algorithms [1], [4]–[8] determine chunk boundaries based on data content so that duplicate regions can still be identified when data is shifted by insertions or deletions [5]. CDC algorithms directly influence both the space savings achieved by deduplication and the end-to-end throughput of the deduplication pipeline [3].

The behavior of CDC algorithms depends on the properties of the input data [9], [10]. An important case is low-entropy data, which exhibits limited variation or repetitive patterns. In this work, we focus on *zero-entropy regions*: contiguous regions containing a single repeated byte value. We first study their impact on modern deduplication pipelines, then characterize their properties in real workloads, and finally investigate how they can be leveraged to improve deduplication.

Zero-entropy regions are not limited to synthetic or pathological inputs. We identify them across virtual-machine images, source-code repositories, scientific datasets, and database snapshots, although their prevalence varies substantially across workloads. In some datasets, zero-entropy regions account for up to 30% of the input data, while in others they are negligible.

In deduplication pipelines, zero-entropy regions are processed as part of the ordinary input stream. These regions participate in CDC boundary detection, become part of generated chunks, are fingerprinted, and require fingerprint-index lookups before unique chunks are passed to subsequent storage-processing stages, such as compression. Existing low-entropy-aware CDC techniques reduce some of these effects by modifying chunk-boundary selection. For example, an optimized variant of AE [11] detects zero-entropy regions, while UltraCDC [12] incorporates mechanisms to handle low-entropy input during chunking. However, these approaches retain the corresponding bytes within the generated chunks, so they continue through the remaining deduplication pipeline. Consequently, even though zero-entropy regions can be deduplicated and compressed, they still incur processing overhead. We study the impact of zero-entropy regions on deduplication systems. We use four CDC algorithms: AE [4], RAM [6], Rabin [5], and TTTD [7]. We find that zero-entropy regions can reduce the space savings achieved by the evaluated CDC algorithms by up to nine percentage points compared with high-entropy regions of the same sizes and at the same positions (§III). This result shows that the structure of the inserted data, in addition to the byte shift itself, can affect CDC boundary generation and the resulting deduplication effectiveness.

We then characterize zero-entropy regions in real-world workloads (§IV) by examining their sizes, locations, and repeated byte values. We find that zero-entropy regions are distributed throughout datasets rather than concentrated in a small number of locations, meaning that a mechanism designed to identify them must examine the entire input stream. We also find that at least 80% of zero-entropy regions are smaller than 64 bytes, which can increase metadata overhead when many regions must be represented individually. Finally, zero-entropy regions within a dataset are often dominated by a small number of repeated byte values, such as `NULL` or `SPACE`, while the dominant value varies across workloads. These findings suggest that a mechanism for exploiting zero-entropy regions must efficiently detect small regions throughout the input, represent them using compact reconstruction metadata, and support arbitrary repeated byte values.

Guided by these findings, we design ZERO (§V), a preprocessing layer that complements existing deduplication systems. ZERO scans the incoming data stream, identifies and extracts zero-entropy regions before they enter the deduplication pipeline, and encodes each region using its position, length, and repeated byte value. The remaining filtered data is passed to the deduplication pipeline. During reconstruction, ZERO uses the recorded metadata to restore the extracted regions to their original positions.

Our evaluation (§VI) integrates ZERO with six latest CDC algorithms (AE, FastCDC, Rabin, RAM, TTTD, and UltraCDC) as well as with lossless compression. ZERO improves space savings by up to 26 percentage points over vanilla deduplication and reduces end-to-end data-reduction time by up to 24%. When composed with UltraCDC, a low-entropy-aware CDC algorithm, ZERO provides an additional improvement of approximately 7.2 percentage points. The benefit is workload-dependent and becomes smaller when little zero-entropy data is available.

The rest of this paper is organized as follows. Section II provides background and Section III motivates our work. Section IV characterizes zero-entropy regions. Section V presents ZERO's design, followed by its evaluation in Section VI. Section VII discusses additional related work, and Section VIII concludes the paper.

II. Background

A. Data Deduplication

A data deduplication pipeline consists of four main steps:

1) **Data Chunking**: Incoming files are divided into smaller parts called chunks using *chunking algorithms*.
2) **Chunk Fingerprinting**: A *fingerprint* is computed for each chunk using a hash function, such as SHA-1 [19].
3) **Fingerprint Indexing**: Fingerprints are compared against an index of fingerprints of data stored in the system to identify duplicate chunks.
4) **Unique-Chunk Storage**: Chunks that are not identified as duplicates are stored, and their fingerprints are added to the index.

Data chunking is critical to deduplication because the selected chunk boundaries affect space savings and processing throughput [20]. Systems use different CDC mechanisms to determine chunk boundaries from data content. Hash-based approaches, such as Rabin-Karp [5] and TTTD [7], use content-dependent hash values, while hashless approaches, such as AE [4] and RAM [6] use local extrema or related byte-level features to identify a chunk boundary. FastCDC [1] uses Gear-based hashing together with techniques designed to accelerate chunk-boundary detection.

B. Data Compression

Lossless compression is a data reduction method complementary to deduplication that encodes repeated patterns to save space. GZIP [21] uses deflate, a lossless compression technique that identifies repeated byte patterns and replaces them with shorter codes. Run-Length Encoding (RLE) [22] is a simpler lossless compression technique that stores a single copy of repeated data values, along with how many times they occur. Zstandard (Zstd) [23] is a modern lossless compression algorithm that combines fast compression speeds with high compression ratios using entropy coding and dictionary-based techniques. Lossless compression is often paired with data deduplication to maximize storage efficiency [24]. However, it is computationally expensive, achieving lower throughput than deduplication (§VI).

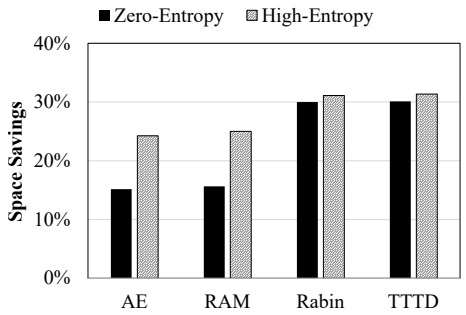


Fig. 1: Impact of zero-entropy and high-entropy insertions on space savings achieved by CDC algorithms.

C. Zero-Entropy Regions

Entropy [25] measures the uncertainty or variability of data. Messages with more randomness have higher entropy. Zero-entropy regions are regions in which every byte has the same value. Because there is no variation in the byte values within such a region, its byte-level entropy is zero.

Existing techniques handle low-entropy regions at different stages of storage processing. Low-entropy-aware CDC algorithms, such as UltraCDC [12] and an optimized variant of AE [11], modify chunk-boundary selection to better handle low-entropy regions. The optimized AE variant detects zero-entropy regions, while UltraCDC incorporates mechanisms for handling low-entropy input during chunking. However, the corresponding bytes remain part of the generated chunks and continue through the deduplication pipeline, incurring computation overhead. Lossless compression techniques, such as GZIP [21] and Zstd [23], can compactly encode repeated patterns, but they impose significant overhead.

III. Motivation

We conduct an experiment to understand the impact of zero-entropy regions on the CDC algorithms [4]–[7]. We begin with a source file containing randomly generated data and create two modified versions of the file. In the first version, we insert high-entropy random byte sequences. In the second, we insert zero-entropy sequences consisting of repeated byte values. The insertions in the two files have the same sizes and occur at the same positions.

We deduplicate each modified file against the source using four CDC algorithms: AE [4], RAM [6], Rabin-Karp [5], and TTTD [7]. AE and RAM are hashless CDC algorithms, whereas Rabin-Karp and TTTD are hash-based. Figure 1

TABLE I
ZERO-ENTROPY DATA IN REAL-WORLD DATASETS.

Dataset	Input	Files	Description	Qualifying ZE	ZE Ratio
CORD	80.0 GB	717,120	COVID-19 Open Research Dataset [13]	24.13 GB	30.17%
LNX	65.0 GB	158	Linux kernel distributions [14] in TAR [15] format	6.70 GB	10.32%
RDS	121.6 GB	100	Redis [16] snapshots generated with redis-benchmark	0.017 GB	0.014%
VDI	121.1 GB	48	Virtual-machine images in VDI format [17] from SourceForge [18]	34.34 GB	28.36%

shows the space saving with zero-entropy and high-entropy insertions. All CDC algorithms achieve lower space savings for files containing zero-entropy regions than for files containing high-entropy insertions. This indicates that zero-entropy regions directly impact the CDC algorithms effectiveness in identifying a chunk boundary, which in turn impacts deduplication effectiveness.

Two recent works aim to address the impact of low-entropy regions on chunking. UltraCDC [12] tries to avoid low-entropy regions during chunking. UltraCDC incorporates a jumping-based mechanism to identify and skip low-entropy regions during chunk-boundary detection. An optimized version of AE [11] tracks both the maximum and minimum byte values and identifies zero-entropy regions when the two are equal. These approaches operate within the chunking stage, and the low-entropy regions remain part of the generated chunks and continue through subsequent deduplication stages. ZERO explores a different approach for handling zero-entropy regions. ZERO extracts zero-entropy regions from the data before the CDC step.

IV. CHARACTERIZATION OF ZERO-ENTROPY REGIONS

In this section, we characterize zero-entropy regions across real-world datasets to understand their prevalence and properties across storage workloads. In particular, we examine the sizes of these regions, their location within a dataset, and their repeated byte values.

Datasets and Characteristics. We examined fifteen real-world datasets used in prior storage and deduplication studies [1], [3], [8], [10], [26], [27]. We consider zero-entropy regions with a minimum length of 8 bytes. We selected four datasets with varying amounts of zero-entropy regions. Table I shows the dataset details, size, and ratio of zero-entropy regions. CORD and VDI represent workloads with substantial zero-entropy content, LNX represents a workload with a moderate amount, and RDS represents a workload with negligible zero-entropy content. We intentionally retain RDS as a low-zero-entropy case to examine the behavior of ZERO when little zero-entropy data is available.

We observe substantial variation across workloads. In CORD and VDI, zero-entropy regions make up 24.13–34.34 GB of data, amounting to 30.17% and 28.36% of the respective datasets. LNX contains 6.70 GB of zero-entropy data, amounting to 10.32% of the dataset. In contrast, RDS contains only 0.014% of its input. We therefore do not further characterize the negligible zero-entropy content in RDS; instead, we retain it in the evaluation (§VI) as a workload where little data is available for extraction.

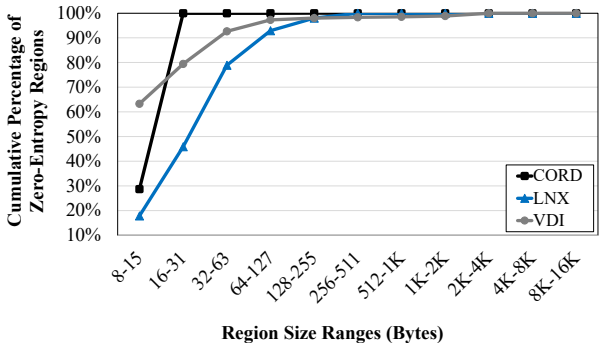


Fig. 2: Cumulative distribution of zero-entropy regions sizes across datasets, showing their ranges.

A. Zero-Entropy Region Sizes

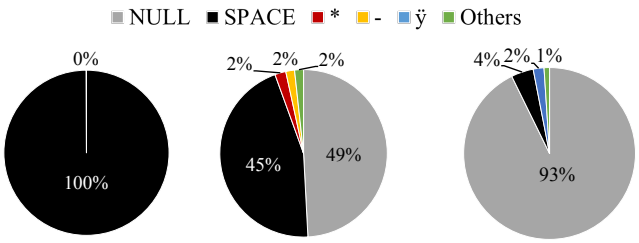
Figure 2 shows the cumulative distribution function (CDF) of the sizes of zero-entropy regions in CORD, LNX, and VDI. The majority of zero-entropy regions are small, with 80% of zero-entropy regions having a size less than 64 bytes. However, the exact distribution varies by dataset.

In VDI, 63% of the zero-entropy regions fall within the smallest region size range (8–15 bytes), and 98% are smaller than 256 bytes. In CORD, 99% of zero-entropy regions are smaller than 32 bytes. LNX shows a broader distribution, with zero-entropy regions spread over a range of region sizes up to 256 bytes. All three datasets have long tails, containing a small number of large zero-entropy regions. CORD contains zero-entropy regions up to 2KB while LNX and VDI contain zero-entropy regions as large as 16KB.

The prevalence of small regions creates a tradeoff for extraction: extracting more small regions removes additional data from the downstream deduplication pipeline, but can also increase the number of metadata records required for reconstruction.

B. Location of Zero-Entropy Regions

An important characteristic of zero-entropy regions is whether they are concentrated within a small portion of the input or distributed throughout the dataset. We divide each dataset into 128 MB fragments and study the zero-entropy regions in each fragment. We find that zero-entropy regions are distributed throughout each dataset with at least 80% of fragments containing zero-entropy regions.



(a) CORD (b) LNX (c) VDI

Fig. 3: Percentage of repeated byte values in zero-entropy regions across datasets.

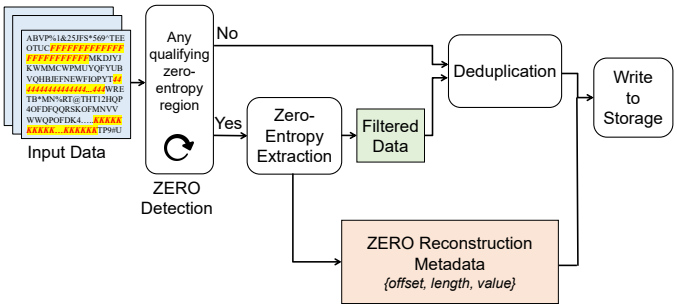


Fig. 4: ZERO's data ingestion workflow.

Zero-entropy regions are also distributed across files. They are present in approximately 80% of CORD's files and in 100% of files in LNX and VDI. These results show that zero-entropy regions may be present anywhere within a dataset, hinting that techniques leveraging them need to scan large amounts of data.

C. Byte Values in Zero-Entropy Regions

Figure 3 illustrates the percentage shares of different byte values within zero-entropy regions across datasets. The figure shows that each dataset is dominated by a small number of repeated byte values. In CORD, the SPACE value dominates. In LNX, NULL and SPACE are the two most common repeated byte values. In VDI, NULL is the dominant repeated byte value. These results show that zero-entropy regions are not limited to all-zero sequences. A mechanism targeting these regions should therefore support arbitrary repeated byte values.

D. Characterization Summary

To summarize, our characterization study shows that:

- The prevalence of zero-entropy data is workload-dependent, reaching approximately 30% in some production datasets, while being negligible in others.
- The majority of observed zero-entropy regions are small, with 80% smaller than 64 bytes, requiring efficient identification techniques.
- Zero-entropy regions are spread across datasets, requiring the scanning of all incoming data and not just specific regions.
- Zero-entropy regions typically have one dominant repeated byte, varying per dataset. Simple encoding techniques are sufficient to handle this.

We use these insights to design ZERO.

V. System Design

We propose ZERO, a preprocessing module that identifies and extracts zero-entropy regions before the deduplication pipeline. In this section we discuss the steps to store new data and to reconstruct stored data.

A. Data Ingestion

Figure 4 shows the pipeline for ingesting new data. The pipeline first scans the data to identify zero-entropy regions, extracts them, and updates the file metadata. The remaining data is processed through the deduplication technique. The pipeline produces two outputs: filtered data, which continues through the deduplication pipeline, and reconstruction metadata describing the extracted regions.

Zero-Entropy Detection. ZERO scans the input looking for regions with a single repeated byte value. If such a region is at least as long as a configurable threshold T. Unless otherwise stated, we use $T = 8$ bytes.

ZERO Extraction. For each qualifying region, ZERO removes the corresponding bytes from the stream. All remaining bytes are preserved in their original order, forming the *filtered data*. For every extracted region, ZERO records its original byte offset, length, and repeated byte value. To reduce reconstruction-metadata overhead, these records are stored using a compact representation: positions are delta-encoded as gaps between successive extracted regions (typically 1–2 B), and region lengths are encoded using tagged variable-length integers (typically 1–2 B). The dominant repeated-byte value is stored once per file (1 B). Regions with a different byte value are identified by the tagged length field and store the exceptional byte explicitly (1 B). Thus, each region requires a minimum of 2 B, plus a 6 B header per file containing extracted regions. With $T = 8$, each extracted region contains at least 8 B, while its compact metadata requires a minimum of 2 B. These records form the ZERO reconstruction metadata.

Because extraction occurs before CDC, it may produce different chunk boundaries on the filtered data than on the original input. This does not affect correctness because the deduplication pipeline only needs to reconstruct the filtered data, after which ZERO restores the extracted zero-entropy regions. ZERO is designed as a preprocessing stage. Its benefit depends on the amount and granularity of zero-entropy data in the workload.

B. Data Reconstruction

Figure 5 shows the steps to reconstruct a stored file. Reconstruction proceeds in two stages: First, the deduplication system uses the stored unique chunks and file recipe to reconstruct the filtered data. ZERO then uses the ZERO reconstruction metadata to restore the extracted zero-entropy regions. Reconstruction records are processed in increasing order of their original byte offsets. For each record, ZERO copies the required bytes from the filtered data until the output

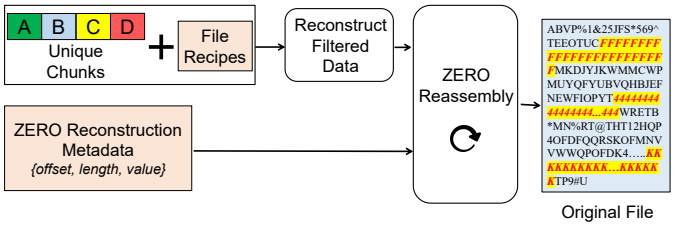


Fig. 5: ZERO's data reconstruction workflow.

reaches the recorded offset, and then inserts the recorded number of copies of the corresponding byte value. The number of extracted regions affects both metadata size and reconstruction cost. We quantify these tradeoffs, together with the effect of the extraction threshold, in §VI.

VI. Evaluation

We evaluate ZERO to answer four questions: (1) how the extraction threshold and reconstruction metadata affect space savings; (2) how ZERO interacts with different CDC algorithms, including low-entropy-aware CDC; (3) how it interacts with lossless compression; and (4) what processing and reconstruction costs it introduces and whether the reconstructed output matches the original data.

A. Experimental Setup

Testbed. We run all experiments on a `c6525-25g` machine from CloudLab Utah [28]. The machine has a 16-core AMD EPYC 7302P processor with hyperthreading, 128 GB of RAM, and a 25 Gbps Mellanox NIC.

Datasets. We evaluate ZERO using the four workloads described in Table I: `CORD`, `LNX`, `RDS`, and `VDI`.

Deduplication. We use the implementations from DedupBench [3] and evaluate AE [4], FastCDC [1], Rabin-Karp [5], RAM [6], TTTD [7], and UltraCDC [12]. Unless otherwise stated, the minimum, average, and maximum chunk sizes are 2 KB, 8 KB, and 32 KB, respectively.

Compression. We evaluate GZIP using `zlib` and Zstd [23]. For Zstd, we use `L3` as the storage-efficiency configuration and `Fast` as the throughput-oriented configuration. Compression is applied after deduplication to the unique stored data. We also evaluate simple byte-level run-length encoding (RLE), where each maximal byte run is represented by its byte value and a variable-length run length.

ZERO Configuration. Unless otherwise stated, ZERO uses the default extraction threshold $T = 8$ bytes. We additionally evaluate larger thresholds to quantify the tradeoff between the amount of zero-entropy data extracted and the number of reconstruction-metadata records generated.

Space Savings. We calculate space savings relative to the original input size as

$$\text{Space Savings} = \frac{\text{Input Size} - \text{Stored Size}}{\text{Input Size}} \qquad (1)$$

For vanilla deduplication, *Stored Size* is the unique chunk payload. For ZERO, *Stored Size* includes both the unique chunk payload and the ZERO reconstruction metadata. Conventional deduplication file-recipe metadata is excluded from both configurations. When compression is enabled, the payload term is the compressed unique payload.

Timing. Processing-time results include ZERO detection and extraction together with the subsequent data-reduction stages used in each configuration. Unless otherwise noted, timing results are averaged over five runs, and the standard deviation is below 5%.

Implementation. We implement ZERO in approximately 3000 lines of C++ code and use the C++ implementations from DedupBench [3] for the downstream deduplication pipeline.

TABLE II
Effect of ZERO's extraction threshold on `VDI` with FastCDC. Extracted and metadata values are percentages of the original input; regions is a count.

Configuration	Extracted	Regions	Metadata	Space Savings
FastCDC	–	–	–	46.08%
ZERO $T = 8$	28.36%	418.86M	0.79%	**59.07%**
ZERO $T = 32$	24.95%	74.69M	0.16%	57.94%
ZERO $T = 64$	23.28%	26.78M	0.07%	56.95%

B. Space Savings

Extraction Threshold and Metadata. We first examine how the extraction threshold affects space savings. Table II reports metadata-inclusive results on `VDI` using FastCDC.

FastCDC achieves 46.08% space savings while ZERO improves it by 12.99 percentage points at $T = 8$. Raising T to 32 bytes reduces reconstruction records by 82.2% and metadata overhead by 79.7%, but decreases space savings by 1.13 percentage points. At $T = 64$, space savings decrease by another 0.99 percentage points.

These results show that the extraction threshold balances two competing effects: a lower T removes more zero-entropy data from the downstream pipeline but generates more regions and reconstruction metadata. With the compact representation, metadata is sufficiently small that the default $T = 8$ provides the highest overall space savings on `VDI`.

Table III reports metadata-inclusive space savings across CDC algorithms. ZERO improves savings by approximately 26 percentage points on `CORD` and by 4.50–12.99 points on `VDI`. Results are CDC-dependent on `LNX` and negligible on `RDS`.

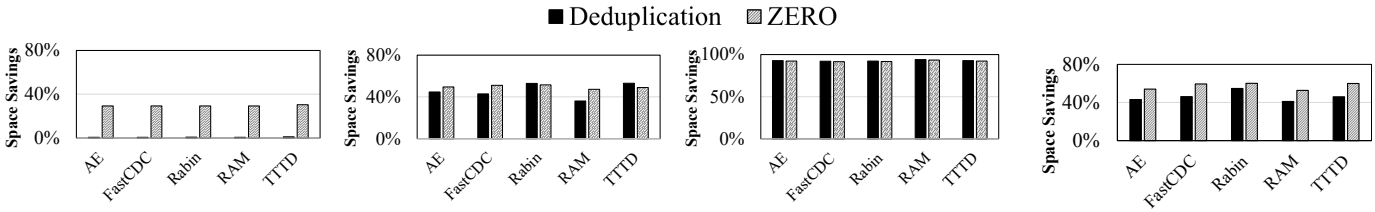
ZERO vs Deduplication. Figure 6 compares the payload-only space savings across CDC algorithms with and without ZERO. ZERO improves space savings by up to 13.98, 29.38, and 11.10 percentage points on `VDI`, `CORD`, and `LNX`, respectively. On `RDS`, the two configurations achieve similar savings because little zero-entropy data is available for extraction.

Low-Entropy-Aware CDC. We also compare ZERO against UltraCDC [12], which already includes mechanisms for handling low-entropy input. Figure 7 compares their metadata-inclusive space savings across four datasets. ZERO

TABLE III

METADATA-INCLUSIVE SPACE SAVINGS (%) AT $T = 8$. METADATA IS REPORTED AS A PERCENTAGE OF THE ORIGINAL INPUT. D DENOTES VANILLA DEDUPLICATION AND Z DENOTES ZERO INCLUDING RECONSTRUCTION METADATA.

Dataset	Metadata	AE		FastCDC		Rabin		RAM		TTTD	
		D	Z	D	Z	D	Z	D	Z	D	Z
CORD	3.78%	0.49	**26.76**	0.78	**27.07**	1.04	**27.25**	0.41	**26.68**	1.05	**27.25**
VDI	0.79%	42.95	**53.78**	46.08	**59.07**	54.42	**58.92**	39.58	**51.73**	53.34	**59.12**
LNX	0.60%	44.71	**48.51**	42.90	**50.03**	**52.93**	47.75	36.11	**40.45**	**53.07**	48.34
RDS	0.006%	92.86	92.85	92.11	92.10	92.27	92.27	93.97	93.97	92.80	92.79



(a) CORD (b) LNX (c) RDS (d) VDI

Fig. 6: Payload-only space savings achieved by ZERO and vanilla deduplication across CDC algorithms.

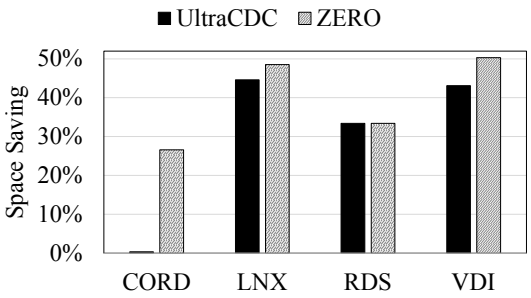


Fig. 7: Metadata-inclusive space savings achieved by UltraCDC with and without ZERO across four datasets.

improves UltraCDC's space savings by 26.24, 7.20, and 3.93 percentage points on CORD, VDI, and LNX, respectively, while the improvement on RDS is negligible.

C. Lossless Compression

Lossless compression increases space savings because it captures redundancy that remains after duplicate chunks are removed. Figure 8 compares the payload-only space savings achieved when lossless compression is applied after deduplication.

The additional space savings from ZERO become smaller when strong compression is enabled. For example, on LNX, *ZERO + ZLIB* achieves 88.74% payload-only space savings, compared with 88.20% for *Dedup + ZLIB*. After including the 0.60% compact reconstruction metadata, the corresponding ZERO result is 88.14%, eliminating this small payload-only advantage. On CORD, *ZERO + ZSTD* underperforms *Dedup + ZSTD* by approximately 7 percentage points even before reconstruction metadata is included.

Therefore, when the estimated metadata-inclusive gain after compression is negligible or negative, an adaptive deployment could bypass ZERO. Thus, ZERO complements rather than replaces lossless compression.

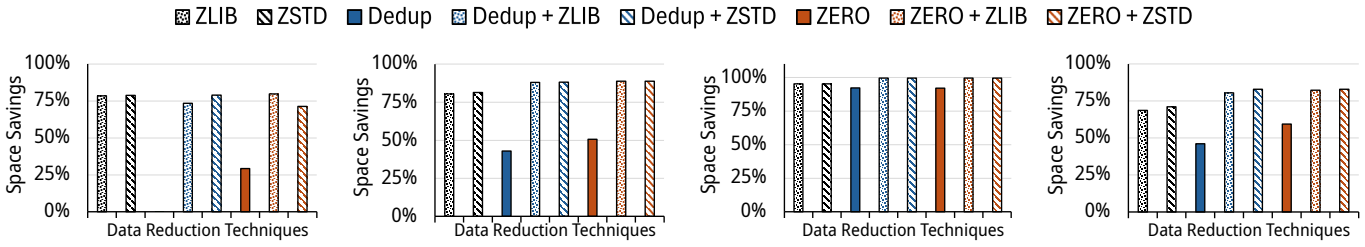
We also compare against simple byte-level RLE. RLE expands CORD, LNX, and VDI by 38.16%, 69.54%, and 30.04%, respectively. Most maximal byte runs are short, so encoding every run introduces substantial overhead. In contrast, ZERO extracts only repeated-byte regions of at least T bytes and leaves the remaining data unchanged for downstream deduplication and compression. Thus, ZERO is not equivalent to applying RLE to the entire input.

D. End-to-end Data Reduction Time

Figure 9 shows the processing-time breakdown for VDI across the evaluated CDC algorithms. ZERO introduces ZERO detection and extraction, but also reduces the amount of data processed by the downstream stages. The net effect therefore depends on the workload and the CDC algorithm.

On VDI, ZERO reduces or approximately maintains the total processing time across the evaluated CDC algorithms. With Rabin-Karp, the processing time decreases by 24%, while AE shows approximately the same total time. Other workloads exhibit both positive and negative outcomes. On LNX, for example, ZERO increases processing time by 5% with AE and 18% with FastCDC, reduces it by 7% with Rabin-Karp, and is approximately unchanged with RAM.

A similar workload dependence appears when lossless compression is enabled. With ZSTD (L3), ZERO reduces processing time on CORD from 531.55 to 415.59 seconds, a 21.8% reduction. On VDI, both configurations take approximately 380 seconds. In contrast, RDS incurs approximately 22% additional processing time because little input is removed before the downstream stages. Thus, the processing-time benefit of ZERO depends on whether the amount of extracted data is sufficient to offset the additional detection and extraction work.



(a) CORD (b) LNX (c) RDS (d) VDI

Fig. 8: Payload-only space savings achieved when ZERO and vanilla deduplication are combined with lossless compression. We use FastCDC [1] with an average size of 8 KB for chunking.

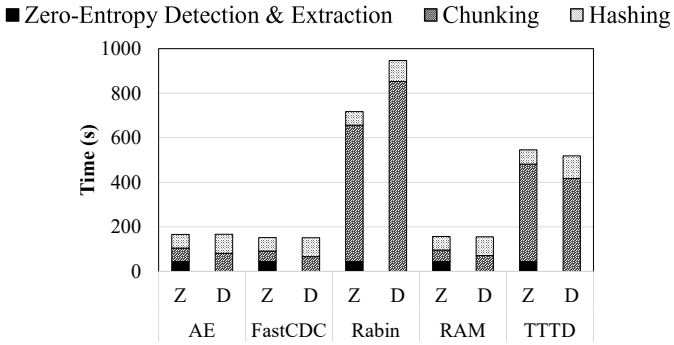
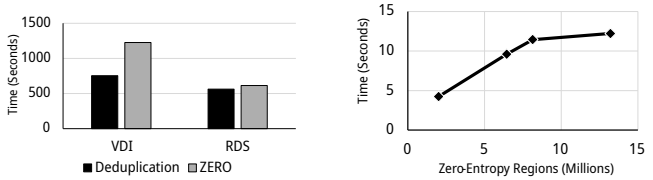


Fig. 9: Processing-time breakdown for VDI. Each group compares ZERO (Z) with vanilla deduplication (D).



(a) Reconstruction time across workloads. (b) Impact of zero-entropy region count.

Fig. 10: ZERO reconstruction overhead.

E. Reconstruction Overheads

ZERO adds a reassembly stage to the conventional reconstruction path because each extracted zero-entropy region must be restored using its reconstruction metadata. We verify reconstruction correctness by comparing reconstructed files with their original inputs. All reconstructed files in the evaluated workloads match their corresponding originals exactly.

Figure 10a compares the measured reconstruction phase for VDI and RDS. On VDI, ZERO requires an additional 473.68 seconds compared with vanilla deduplication, corresponding to a 62% overhead. On RDS, the overhead is 9%. Thus, reconstruction can introduce a material cost when a workload contains a large number of extracted regions. This overhead is important for restore-intensive workloads with strict latency requirements. Our current work performs ZERO reassembly as a separate stage after reconstructing the filtered data. A production implementation could instead overlap the two stages: as the deduplication system reconstructs the filtered stream, ZERO could insert zero-entropy regions when their recorded positions are reached, avoiding the need to wait for the complete filtered file before reassembly.

Impact of Region Count. To isolate this effect, we select four VDI files with similar sizes of 1–2 GB but different numbers of zero-entropy regions, ranging from approximately 2 to 13 million. Figure 10b shows that reconstruction time increases with the number of extracted regions. Region granularity therefore affects not only reconstruction-metadata size but also reconstruction cost. This makes ZERO less attractive for restore-intensive workloads or deployments with strict reconstruction-time requirements.

F. Summary

The benefit of ZERO depends on the workload and the downstream data-reduction configuration. CORD and VDI show the largest storage gains, whereas RDS shows essentially none. On LNX, the benefit varies across CDC algorithms. Strong lossless compression can reduce ZERO's incremental gain. Processing time also varies with the amount of data removed. Finally, reconstruction adds overhead that grows with the number of extracted regions.

VII. Additional Related Work

Impact of Low-Entropy on CDC. Prior work [29] characterizes the impact of low-entropy regions on CDC effectiveness and performance, while this work focuses on zero-entropy regions and extracts them before the deduplication pipeline.

Zero-Filled Data and Sparse Files. Storage systems also special-case all-zero content, including zero-filled blocks and zero chunks [30], [31]. Sparse files similarly represent some zero-valued ranges without allocating corresponding data blocks [32]. Filesystem interfaces such as SEEK_HOLE/SEEK_DATA can identify these ranges [33], while fallocate can preserve or recreate holes [34], [35]. ZERO differs because it operates on the logical byte stream, does not require block or chunk alignment, and supports any repeated byte value rather than only 0x00. Filesystem sparsity and ZERO are therefore complementary. A filesystem-aware implementation could bypass existing sparse holes and apply ZERO only to qualifying regions in allocated data.

VIII. Conclusion

We design ZERO, a preprocessing layer that extracts zero-entropy regions before CDC, records their original byte offset, length, and repeated byte value, and processes the remaining filtered data using a deduplication pipeline. Our evaluation shows that ZERO can improve metadata-inclusive space savings by up to 26 percentage points over vanilla deduplication and reduce end-to-end data-reduction time by up to 24%.

In the future, we plan to further improve ZERO's throughput and reduce its metadata by exploring techniques to integrate ZERO and CDC processing into a single step.

Acknowledgments

The research team was supported by grants from the National Cybersecurity Consortium (NCC), Natural Sciences and Engineering Research Council of Canada (NSERC) (ALLRP-561423-20 and RGPIN-2025-03332), Ontario Research Fund – Research Excellence program (ORF-RE012-051), Oracle Research Labs, Rogers Communication, and Acronis.

References

[1] W. Xia, Y. Zhou, H. Jiang, D. Feng, Y. Hua, Y. Hu, Y. Zhang, and Q. 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*, ser. USENIX ATC '16. USA: USENIX Association, 2016, p. 101–114.

[2] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu, "Characteristics of backup workloads in production systems," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, ser. FAST'12. USA: USENIX Association, 2012, p. 4.

[3] 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.

[4] 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.

[5] 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

[6] 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

[7] 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.

[8] S. Udayashankar, A. Baba, and S. Al-Kiswany, "Seqcdc: Hashless content-defined chunking for data deduplication," in *Proceedings of the 25th International Middleware Conference*, ser. Middleware '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 292–298. [Online]. Available: https://doi.org/10.1145/3652892.3700766

[9] M. Gregoriadis, L. Balduf, B. Scheuermann, and J. Pouwelse, "A thorough investigation of content-defined chunking algorithms for data deduplication," *arXiv preprint arXiv:2409.06066*, 2024.

[10] S. Udayashankar, A. Baba, and S. Al-Kiswany, "VectorCDC: Accelerating data deduplication with vector instructions," in *23rd USENIX Conference on File and Storage Technologies (FAST 25)*. Santa Clara, CA: USENIX Association, Feb. 2025, pp. 513–522. [Online]. Available: https://www.usenix.org/conference/fast25/presentation/udayashankar

[11] Y. Zhang, D. Feng, H. Jiang, W. Xia, M. Fu, F. Huang, and Y. Zhou, "A fast asymmetric extremum content defined chunking algorithm for data deduplication in backup storage systems," *IEEE Transactions on Computers*, vol. 66, no. 2, pp. 199–211, 2017.

[12] P. Zhou, Z. Wang, W. Xia, and H. Zhang, "Ultracdc:a fast and stable content-defined chunking algorithm for deduplication-based backup storage systems," in *2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2022, pp. 298–304.

[13] L. L. Wang, K. Lo, Y. Chandrasekhar, R. Reas, J. Yang, D. Burdick, D. Eide, K. Funk, Y. Katsis, R. Kinney *et al.*, "Cord-19: The covid-19 open research dataset," *ArXiv*, 2020.

[14] Linux Kernel Organization, "The linux kernel archives," https://www.kernel.org/, 2023.

[15] GNU, "GNU tar 1.35: Basic Tar Format," https://www.gnu.org/software/tar/manual/html_section/Standard.html, 2023.

[16] Redis, "Redis," https://redis.io/, 2023, accessed: 2023.

[17] Oracle, "Oracle virtualbox documentation," https://docs.oracle.com/en/virtualization/virtualbox/6.0/user/vdidetails.html, 2020.

[18] VirtualBoxes, "Free virtualbox(r) images," https://sourceforge.net/projects/virtualboximage/, 2024.

[19] D. E. E. 3rd and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," RFC 3174, Sep. 2001. [Online]. Available: https://www.rfc-editor.org/info/rfc3174

[20] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system." in *USENIX Conference on File and Storage Technologies (FAST)*, vol. 8, 2008, pp. 1–14.

[21] J.-L. Gailly and M. Adler, "The gzip compressor," http://www.gzip.org/, 1991.

[22] S. Golomb, "Run-length encodings (corresp.)," *IEEE transactions on information theory*, vol. 12, no. 3, pp. 399–401, 1966.

[23] Y. Collet and M. Kucherawy, "Zstandard compression and the application/zstd media type," RFC 8878, 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc8878

[24] C. Constantinescu, J. Glider, and D. Chambliss, "Mixing deduplication and compression on active data sets," in *2011 Data Compression Conference*. IEEE, 2011, pp. 393–402.

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

[26] O. Randall and P. Lu, "Predicting deduplication performance: An analytical model and empirical evaluation," in *2022 IEEE International Conference on Big Data (Big Data)*, 2022, pp. 319–328.

[27] O. Randall, L. Schultz, and P. Lu, "Optimizing deduplication parameters via a change-estimation analytical model," *IEEE Transactions on Big Data*, pp. 1–12, 2025.

[28] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The design and operation of CloudLab," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, Jul. 2019, pp. 1–14. [Online]. Available: https://www.flux.utah.edu/paper/duplyakin-atc19

[29] M. A. Jarah, S. Udayashankar, A. Baba, and S. Al-Kiswany, "The impact of low-entropy on chunking techniques for data deduplication," in *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, 2024, pp. 134–140.

[30] K. Jin and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," in *Proceedings of the Israeli Experimental Systems Conference (SYSTOR)*. ACM, 2009.

[31] D. Meister, A. Brinkmann, and T. Süß, "File recipe compression in data deduplication systems," in *11th USENIX Conference on File and Storage Technologies (FAST 13)*. San Jose, CA: USENIX Association, 2013, pp. 175–182. [Online]. Available: https://www.usenix.org/conference/fast13/technical-sessions/presentation/meister

[32] Free Software Foundation, "Gnu coreutils: Sparse files," https://www.gnu.org/software/coreutils/manual/html_node/cp-invocation.html.

[33] M. Kerrisk, "lseek(2) – linux manual page," https://man7.org/linux/man-pages/man2/lseek.2.html, linux man-pages project.

[34] OpenZFS Project, "zfs(4) – openzfs documentation," https://openzfs.github.io/openzfs-docs/man/master/4/zfs.4.html.

[35] M. Kerrisk, "fallocate(2) – linux manual page," https://man7.org/linux/man-pages/man2/fallocate.2.html, linux man-pages project.