Is Low Similarity Threshold A Bad Idea in Delta Compression?

Hongming Huang
City University of Hong Kong
Hong Kong, China
honhuang7-c@my.cityu.edu.hk

Chun Jason Xue
Mohamed bin Zayed University of Artificial Intelligence
Masdar City, Abu Dhabi, United Arab Emirates
jason.xue@mbzuai.ac.ae

Nan Guan
City University of Hong Kong
Hong Kong, China
nanguan@cityu.edu.hk

Hong Xu
The Chinese University of Hong Kong
Hong Kong, China
hongxu@cuhk.edu.hk

Abstract

Delta compression attracts many researchers' interest for its high efficiency in eliminating redundant data. It identifies a similar block for the incoming block and stores only the differences between them. The key challenge lies in detecting suitable similar blocks. Existing approaches have their limitations. Hash-based solutions like NTransform miss many similar blocks due to the high similarity detection threshold, while complex-threshold solutions like DeepSketch and Palantir have high computation overhead.

This paper explores whether a lower detection threshold can help delta compression. We propose new criteria to identify those wrongly recognized similar blocks and prevent their harm to the compression ratio. Moreover, using lower detection thresholds with a base block extension can effectively utilize potential duplicate data adjacent to the base block and achieve a higher compression ratio. Preliminary evaluation with six datasets shows that our approach, on average, improves the overall compression ratio (including deduplication and lossless compression) by 15.1% and finds 29.6% more similar blocks over the state-of-the-art approach Odess, with a throughput degradation of 3.7%.

CCS Concepts: • **Theory of computation** → Data structures and algorithms for data management; • **Information systems** → **Deduplication**.

Keywords: Storage, Delta Compression, Deduplication

ACM Reference Format:
Hongming Huang, Chun Jason Xue, Nan Guan, and Hong Xu. 2024. Is Low Similarity Threshold A Bad Idea in Delta Compression?. In *16th ACM Workshop on Hot Topics in Storage and File Systems (HOTSTORAGE '24), July 8–9, 2024, Santa Clara, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3655038.3665940

1 Introduction

Modern data centers have witnessed surging data volumes from various applications such as AI/machine learning and data analytics [14]. The trend creates significant pressure on storage systems, especially backup storage systems where data is replicated among multiple backups. Besides the well-studied deduplication [16, 20] and lossless compression [5, 7, 9, 13, 17, 23] techniques, *delta compression* becomes increasingly important to mitigate the pressure by storing only the differences between a pair of similar blocks [6, 18, 19, 22].

The critical challenge in delta compression lies in detecting similar blocks. Most prior work relies on locality-sensitive hash (LSH) functions with fixed detection thresholds [2, 3, 22], which are likely to generate the same hash result (a.k.a. *sketch*) for similar blocks. Those with at least one identical sketch are considered similar blocks. The common wisdom is to avoid using low similarity thresholds for hash functions because they may falsely recognize many non-similar blocks as similar and suffer from reduced compression performance. In practice, traditional approaches like NTransform, Finesse, and Odess [3, 22, 24] adopt a high similarity threshold to avoid such a risk, albeit at the cost of missing potential similar blocks.

Recent works utilize more complex similarity detection mechanisms [4, 8, 15]. For example, DeepSketch [15] leverages deep neural networks to produce similarity vectors and employs the approximate nearest neighbor algorithm to match similar blocks. Palantir [8], on the other hand, generates a set of sketches with multiple distinct LSH detection thresholds, and matches similar blocks from high to low thresholds hierarchically. Their improvements come at

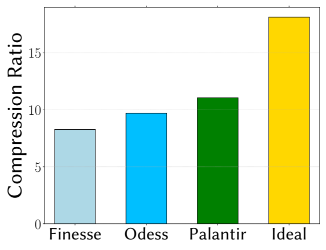


Figure 1. The compression ratio (including deduplication and delta compression) on DB_Stock dataset.

the expense of substantial computational overheads: DeepSketch's throughput is 55% lower than Finesse [22]; on a large dataset with a size of 1.2 TB, Palantir reports up to 45% lower throughput than Finesse.

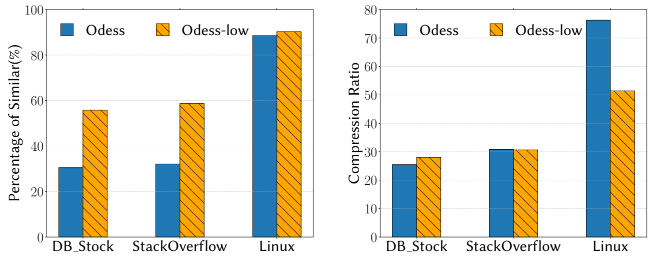
The fundamental question here is, how can we effectively and efficiently detect blocks that are truly similar? In sharp contrast to conventional wisdom, we explore the use of low similarity thresholds as a potential way out. A lower threshold naturally detects more similar blocks while keeping the computation overhead as low as traditional LSH-based solutions. Clearly, the key challenge is to identify and reduce false similar blocks effectively, preventing their harm to the overall compression ratio.

To address this challenge, this paper explores a comprehensive modeling and estimation of the gain and opportunity cost of conducting delta compression, including delta encoding, lossless compression, and the impact on other data's compression. This modeling approach can effectively filter out false similar blocks without heavy computation. Moreover, we find that lower detection thresholds can help to solve the boundary-shift problem[1], a well-known and not perfectly resolved issue that harms the compression ratio, by effectively finding original blocks. Then, we propose a novel base block extension solution to exploit potential duplicate data adjacent to the original block and promote the compression ratio further.

We conduct preliminary evaluations on six datasets and with the state-of-the-art algorithm Odess [24]. Using a lower threshold with our optimizations achieves a 15.1% higher end-to-end compression ratio and detects 29.6% more similar blocks on average, while only decrease the system's throughput by 3.7%. This result shows the potential of the lower similarity detection solution. A well-designed false similar filter can help it better detect similar blocks with low overhead. Its broader similarity detection range can also support other potential optimization solutions like the base block extension, which helps solve the boundary-shift problem.

2 Motivation

While many studies have delved into delta compression, a noticeable gap exists between state-of-the-art algorithms



(a) Percentage of similar **(b)** Compression ratio

Figure 2. Number of detected similar blocks and overall compression ratio on different datasets. Odess-low means using a lower threshold by generating 6 super-features from 12 raw features.

and the upper bound of achievable compression ratios. Figure 1 shows an experiment on a synthetic DB_Stock dataset (see Sec 4.1) using deduplication and different delta compression algorithms. The ideal reduction ratio means the maximum compression ratio when all duplicate data is eliminated, which is calculated by dividing the size of total data into the size of unique data. The result shows a 39% gap between the ideal and the best existing algorithm.

The primary reason for this gap lies in a fundamental dilemma when setting the similarity detection threshold. Setting a high threshold leads to omitting similar blocks and losing delta compression opportunity. Conversely, employing a low threshold can find more similar blocks but introduces false similar blocks, wherein dissimilar blocks are mistakenly marked as similar. The subsequent lossless compression of these delta-encoded blocks becomes tough, potentially resulting in a lower overall compression ratio.

To investigate the performance of these two options, we measure the overall compression ratio (including deduplication, delta compression and ZSTD [5] lossless compression) of the Odess algorithm [24] with different detection thresholds. Figure 2a shows the proportion between the detected similar blocks and total non-duplicate blocks. The lower threshold solution finds 55% more similar blocks on average. However, we find that about 21.9% of similar blocks detected by the lower threshold are false similar blocks that contribute none or even negatively to the overall compression ratio. As Figure 2b shows, the lower threshold solution has only minor or negative compression ratio gain.

This investigation implies that if we can effectively filter out false similar blocks, the lower detection threshold solution can fully exploit its advantage in detecting more similar blocks while not harmed by false similar blocks, and eventually achieve a higher compression ratio. This finding motivates us to explore this problem missed by conventional wisdom. Currently, the sole existing work addressing this issue is Palantir [8]. It involves an intuitive false similar filter that compares the compression ratio of delta encoding and lossless compression, but does not comprehensively investigate the impact of delta compression on such blocks.

In the following sections, we perform a comprehensive investigation of the influence of delta encoding operations. We observe that delta encoding not only modifies the compression characteristics of the current block, but also exerts an impact on the compression of other data. The unique block can serve future delta compression as a base block, while a delta-encoded block cannot. Therefore, we explore an innovative model that quantifies all these impacts and leverages them as the criteria for identifying false similar blocks. We aim to enhance the effectiveness of the LSH-based solutions with a lower detection threshold, enabling them to realize their potential to detect more real similar blocks and eventually achieve a higher compression ratio.

3 Preliminary Design

In this section, we explore new criteria to identify false similar blocks that help the lower detection threshold solution avoid their hazard. Then, we find that the lower detection threshold can help to solve the boundary-shift problem by effectively finding original blocks, and explore a base block extension solution to further improve the compression ratio.

3.1 Identification of False Similar Blocks

A naive way to identify false similar blocks is to compare the real compression ratio with and without delta encoding the block. However, modern lossless compression usually leverages an input buffer that can contain tens of data blocks. Directly measuring the compression ratio requires repetitively compressing the whole buffer with/without each similar block, which is very inefficient.

We regard this problem as a decision problem. The goal is to reduce as much data as possible, and the two options are conducting delta compression or treating the current block as a unique block and directly conduct lossless compression to it. Therefore, our idea is to model the gain of the delta compression option by estimating the size of data it can reduce and the opportunity cost (the maximum gain of the other options). For the opportunity cost, we divide it into two parts: the compression of itself and the impact on the compression of other data.

Gain of delta compression. Since the commonly used delta encoding algorithm, like x-delta [11], contains a local compression, the encoded delta block is difficult to compress further. Therefore, we can estimate the gain of delta compression by conducting delta encoding to the current block and calculating the reduced data size.

Lossless compression of itself. If we do not conduct delta compression, the block will be directly compressed by lossless compression. An existing work [8] proposes an assumption that the lossless compression ratio will not change sharply. Based on this assumption, we can estimate the lossless compression ratio C of the current block is the same as the average of previous L lossless compression records. Then, we estimate this part of the opportunity cost as follows:

$$Cost_{self} = block\_size - block\_size/C \tag{1}$$

Impact on other data. Most delta compression algorithms do not allow delta blocks to act as base blocks. If we regard a block as a unique block, it can serve as a base block in the future and provide delta compression opportunities. The gain of those future delta blocks is also a part of the opportunity cost of delta compressing the current block. To estimate this cost, we have to estimate 1) the expected gain of a future delta block and 2) the expected number of new delta blocks can be matched by a unique block.

Notably, those future delta blocks also have their opportunity cost when being delta compressed. Therefore, we define the $real\_gain$ of a delta block as:

$$real\_gain = delta\_gain - Cost \tag{2}$$

In formula 2, $delta\_gain$ means how many bytes can be eliminated by delta compression, and $Cost$ means the total opportunity cost of this block. Since we cannot foresee the information of future blocks, we estimate them using the average value of existing delta blocks:

$$E(real\_gain) = \frac{\sum_{delta\_block}(delta\_gain - Cost)}{NUM(delta\_block)} \tag{3}$$

For the number of new delta blocks that can be matched by an independent block, we estimate it using the number of existing delta and unique blocks:

$$E(NUM(new\_delta)) = \frac{NUM(delta\_block)}{NUM(unique\_block)} \tag{4}$$

Finally, we can estimate the opportunity cost of becoming a future base block by combining formula 3 and 4:

$$Cost_{be\_base} = \frac{\sum_{delta\_block}(delta\_gain - Cost)}{NUM(unique\_block)} \tag{5}$$

Boundary constraint. We notice that the $Cost_{be\_base}$ is independent of the current block's size. Then, the opportunity cost might be close to or even exceed the size of small blocks. In this case, a minor estimation error will cause a considerable fluctuation of the criteria. Therefore, we add an upper bound to the opportunity cost based on the estimated lossless compression ratio C of the current block:

$$Cost_{self} + Cost_{be\_base} <= block\_size - \frac{block\_size}{\beta * C} \tag{6}$$

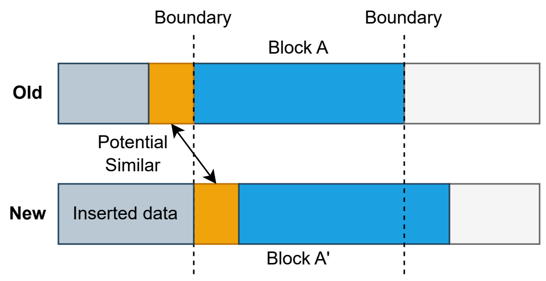


Figure 3. An example of boundary-shift case.

In this formula, β is a parameter that defines the constraint: we regard it as a true similar block if the delta compression's compression ratio is far higher than the lossless compression.

By quantifying the gain and opportunity cost, we define a block as a false similar block if the gain is smaller than the total opportunity cost. If so, we abort the delta compression's result and regard it as a unique block.

3.2 Base Block Extension

After designing the false similar filter for the low detection threshold solution, we came up with the idea that the solution can now help solve the boundary-shifting problem by finding the original block.

The boundary-shift problem is an old problem for data deduplication systems. When a small amount of data is inserted or deleted from a duplicated area, it will affect the chunking point and shift the blocks' boundary. Consequently, the following blocks in the non-modified area are no longer identical. As Figure 3 shows, blocks A and A' are not identical, and even delta compression can only eliminate a part of A' because the shifted data (orange part in the figure) is not included in the original block A.

The common solution to this problem is content-defined chunking (CDC) [7, 12, 13, 21], which decides the chunking point by sliding hash values. However, CDC algorithms may generate blocks of various sizes, affecting the effectiveness of other deduplication techniques. Therefore, CDC algorithms are usually accompanied by a min/max block size limit. These limits make the CDC algorithm unable to solve the boundary-shift problem completely. To investigate the remaining boundary-shift problem, we do a test that adds 4.5% new data to an original dataset (adding method is same as Sec4.1). The fastCDC [21] algorithm generates 10.0% unique blocks, meaning that boundary-shift affects 55% of unique blocks. We measure that the average compression ratio of unique blocks is 2.0. Assuming the average shifting size is 50% of the block size, that means the boundary-shift problem introduces a $55\% * 50\%/2.0 = 13.75\%$ loss to the total compression ratio.

An insight into this problem is that the duplicate data is adjacent to the original block A (the orange part in Figure 3). If we extend the base block using the data adjacent to it, the extended block will include the shifted data, and delta encoding can eliminate all data of A'. However, traditional similarity detection algorithms with high thresholds prevent the insight's exploitation. According to Broder's theorem [2], if the block is shifted by 20% of its length, the original NTransform or Odess algorithm only has a 48% chance of detecting the base block. There is no chance to leverage the adjacent duplicate data without finding the original block.

Our solution is using a lower detection threshold with the false similar filter to find the original block effectively. Then, we extend the detected base block with the entire block before and after it and do delta compression with the extended base block. A lower threshold will enable the system to find the original block, solve the boundary-shift problem better, and further promote the end-to-end compression ratio.

A problem of our preliminary design is that it needs to load both the base block and those two blocks adjacent to it, which means extra data loading and probable decompression works. Our solution is introducing an in-memory cache to keep base blocks together with those adjacent blocks to mitigate this problem.

4 Evaluation

We evaluate our preliminary designs using five backup datasets and a Linux code repository. The evaluation includes a comparison with a state-of-the-art algorithm, Odess, using different similarity detection thresholds. Our key findings are that our two optimizations with a lower threshold promote the end-to-end compression ratio by 15.1% and only introduce a 3.7% throughput downgrade.

4.1 Experiment Settings

Testbed. All experiments were conducted on a server with Intel Xeon 8260 CPU, 512 GB of RAM and a 14 TB SSD disk, and Ubuntu 16.04 LTS. We used the FastCDC [21] algorithm to chunk the data with the expected block size of 8 KB, xDelta-3 [11] for delta encoding, and ZSTD-1.3.1 with level 10 for lossless compression.

Datasets. The evaluation contains results for five synthetic backup datasets and a Linux code repository (Table 1). The initial versions of all backup datasets are collected from a global cloud provider. We create 20 backup versions for each dataset by first dividing them into 8 kB blocks, adding 1%, modifying 3.5% and deleting 0.5% of blocks in each version. To add, we insert a new 8 kB block after the original block. To modify, we randomly modify (0, 50%) of the block's data. To delete, we remove the entire block. All added and modified bytes are randomly selected from the original block to maintain the same byte distribution.

Baseline Setup. We evaluate our works with the state-of-the-art delta compression algorithm Odess [24]. It has high speed with minimum compression ratio loss compared to the raw N-Transform approach[3]. We apply our designs to

Dataset	Size (GB)	Description
DB_Stock	5.211	Stock price database
DB_Forex	55.006	Forex database
VSI_StackOverflow (VSI_SOF)	200.375	Web pages of StackOverflow
VDI_server	318.193	Server images
VDI_ipod	1256.018	Ipod images
Linux	16.888	Linux kernel code 5.0.1 - 5.0.21

Table 1. Test datasets

Algorithm	E2E ratio	DCC	Thrpt (MB/S)
Odess	1.000	0.446	181.558
Odess+fsf+ext	1.019	0.497	177.482
OdessLow	1.018	0.669	181.795
OdessLow+fsf	1.058	0.562	179.952
OdessLow+fsf+ext	**1.151**	**0.578**	**175.053**

Table 2. Average result over all datasets. The E2E ratio is normalized to Odess. **Fsf** means false similar filter, and **ext** means base block extension.

Odess with two settings: the original Odess, and the other is named OdessLow. OdessLow adopts a lower threshold by generating 6 super-features from 12 raw features. Then, we evaluate them with our two designs: false similar filter(**fsf**) and base block extension(**ext**).

Evaluation Metrics. We adopt three metrics:

- The **end-to-end compression ratio (E2E ratio)** of the entire system includes deduplication, delta compression, and lossless compression.
- **Delta Compression Coverage (DCC)**. The ratio between the number of similar blocks and all unique blocks, indicating the ability to detect similar blocks.
- **Throughput**. We measure the end-to-end writing throughput of the entire system to directly compare the efficiency of all algorithms.

4.2 Preliminary Results

This section shows the end-to-end result of all algorithms on different datasets. Table 2 shows the average result for all metrics, and the following figures illustrate results on different datasets. When calculating the average E2E ratio, we normalize the E2E ratio to Odess's ratio to provide a fair comparison and remove the effect of dataset variances. From this table, we can get three key findings:

- When applied to a lower detection threshold, the false similar filter and base block extension promote the end-to-end compression ratio by **15.1%**, and find **29.6%** more similar blocks than the original algorithm. The system's throughput is only affected by **3.7%**. This result proves that a lower threshold can help the delta compression system without introducing much computational overhead.
- Though it can find 49.9% more similar blocks, directly using a lower threshold will bring only a 1.8% higher compression ratio because of the negative impact of false similar blocks. The proposed filter successfully identifies them, leading to a 5.8% higher compression ratio.
- The base block extension improves the compression ratio by 8.8% on average with the lower threshold, but can only improve 1.9% with the original high threshold. This proves our insight that using lower similarity detection thresholds can help to find the original block, which is missed by high threshold solutions.

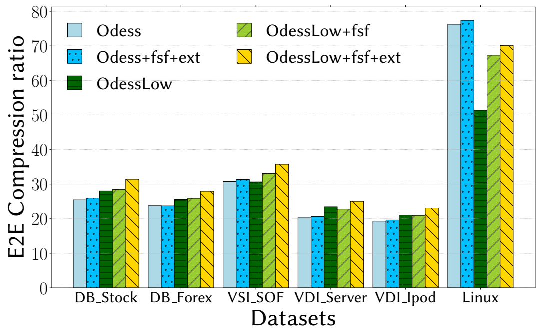


Figure 4. End-to-end compression ratio on all datasets.

Figure 4 shows the E2E compression ratio on all datasets. We can interpret this paper in two aspects: the compression ratio and the stability. Using the lower threshold alone can achieve a higher compression ratio on some datasets. However, it has lower compression ratios on some datasets like VSI_SOF and Linux, while the false positive filter and base block extension outperform and effectively improve OdessLow's compression ratio by 13.1% on average and show higher robustness on different datasets. Notably, the false similar filter slightly downgrades the compression ratio on the VDI_Server dataset, which points out that we should improve the filter further.

An interesting finding is that on the Linux dataset, which is extremely compressible, our lower threshold solution is 7.9% worse than Odess. According to our analysis, the reason is that the degree of similarity of the Linux code repository is very high, and the lower threshold will find many similar candidates. However, the first-fit mechanism makes it unable to select the best candidate. Therefore, it will sometimes pick the sub-optimal base block and lose compression ratio, which points to future work to explore a better mechanism to select proper base blocks.

Figure 5 shows that OdessLow can find many more similar blocks. However, this does not always mean a higher E2E ratio. Odess-low finds more similar blocks on the VSI_SOF

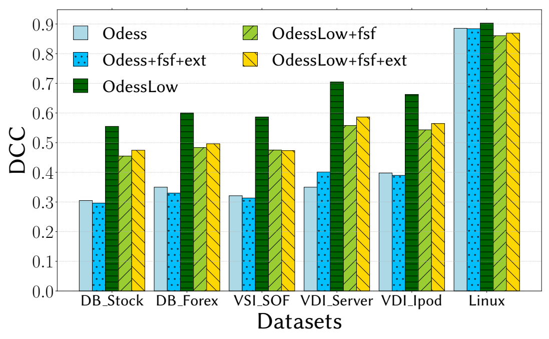


Figure 5. Proportion of detected similar blocks.

Dataset	OdessLow+PFSF	OdessLow+FSF	Gain(%)
DB_Stock	28.021	28.518	1.77%
DB_Forex	25.522	25.813	1.14%
VSI_SOF	32.056	33.631	**4.91%**
VDI_Server	23.747	23.530	-0.91%
VDI_Ipod	21.247	21.108	-0.23%
Linux	62.484	67.365	**7.81%**

Table 3. Compression ratio using different false similar filter algorithms with OdessLow. PFSF means the intuitive false similar filter of Palantir, FSF means this paper's false similar filter.

and Linux datasets, but the E2E ratio is even lower than Odess. This result proves the negative impact of false similar blocks and the importance of a false similar filter, which can promise the system's robustness under different scenarios.

Table 3 compares the two false similar filter algorithms of Palantir and this paper. They are all applied to the baseline OdessLow. The result shows that our false similar filter performs better than Palantir's filter, especially on datasets that suffer significantly from the false similar problem, including the VSI_SOF and Linux. This result supports our idea that we should consider and estimate delta compression's impact on other data, not only the delta block itself. However, on some datasets like VDI_Server, our filter performs slightly worse than Palantir's. This phenomenon indicates that the false similar filter can still be further improved.

5 Discussion

In the future, there will still be many problems to explore further. First, the lower compression ratio on the Linux dataset reveals that the traditional first-fit mechanism does not perfectly match the lower threshold solutions. A new fitting mechanism that can select the best candidates will help the system to handle such scenarios. Second, the preliminary design of base block extension requires to load the entire block before/after the base block. On an HDD-based platfor, this may cause performance downgrade. We should design a mechanism to optimize this I/O overhead to avoid this problem. What is more, the lower threshold solutions can help other research, such as migratory compression [10], which first groups all similar blocks together and compresses them together. The lower threshold solution can detect more similar blocks and help build larger and more effective similar clusters.

References

[1] Muthitacharoen Athicha, Benjie Chen, and David Mazières. 2001. A low-bandwidth network file system. In *ACM Symposium on Operating Systems Principles (SOSP)*. 174–187.

[2] Andrei Z Broder. 1997. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES)*. 21–29.

[3] Andrei Z. Broder. 2000. Identifying and Filtering Near-Duplicate Documents. In *11th Annual Symposium on Combinatorial Pattern Matching (CPM), Montreal, Canada, June 21-23*. 1–10.

[4] ChunzhiWang, Keguan Wang, Min Li, Feifei Wei, and Neal Xiong. 2024. Chunk2vec: A novel resemblance detection scheme based on Sentence-BERT for post-deduplication delta compression in network transmission. *IET Communications* (2024), 145–159.

[5] Yann Collet and Murray Kucherawy. 2021. *Zstandard Compression and the 'application/zstd' Media Type*. Technical Report. https://www.rfc-editor.org/rfc/rfc8878

[6] Fred Douglis and Arun Iyengar. 2003. Application-specific Delta-encoding via Resemblance Detection. In *USENIX Annual Technical Conference (ATC)*. 113–126.

[7] Ahmed El-Shimi, Ran Kalach, Ankit Kumar, Adi Ottean, Jin Li, and Sudipta Sengupta. 2012. Primary Data Deduplication—Large Scale Study and System Design. In *USENIX Annual Technical Conference (ATC)*. 285–296.

[8] Hongming Huang, Peng Wang, Qiang Su, Hong Xu, Chun Jason Xue, and André Brinkmann. 2024. Palantir: Hierarchical Similarity Detection for Post-Deduplication Delta Compression. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[9] Byron Knoll and Nando de Freitas. 2012. A Machine Learning Perspective on Predictive Coding with PAQ8. In *Data Compression Conference (DCC)*. 377–386.

[10] Xing Lin, Guanlin Lu, Fred Douglis, Philip Shilane, and Grant Wallace. 2014. Migratory compression: coarse-grained data reordering to improve compressibility. In *USENIX Conference on File and Storage Technologies (FAST)*. 257–271.

[11] Joshua P. MacDonald. 2000. File system support for delta compression. http://www.xmailserver.com/xdfs.pdf.

[12] Dirk Meister, Jurgen Kaiser, Andre Brinkmann, Toni Cortes, Michael Kuhn, and Julian Kunkel. 2012. A study on data deduplication in HPC storage systems. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*. 1–11.

[13] Dutch T. Meyer and William J. Bolosky. 2012. A study of practical deduplication. *ACM Transactions on Storage (ToS)* (2012), 14:1–14:20.

[14] Naeem Muhammad, Jamal Tauseef, Diaz-Martinez Jorge, Butt Shariq Aziz, Montesano Nicolo, Tariq Muhammad Imran, De la Hoz-Franco Emiro, and De-La-Hoz-Valdiris Ethel. 2022. Trends and future perspective challenges in big data. In *Advances in Intelligent Data Analysis and Applications*. 309–325.

[15] Jisung Park, Jeonggyun Kim, Yeseong Kim, Sungjin Lee, and Onur Mutlu. 2022. DeepSketch: A New Machine Learning-Based Reference Search Technique for Post-Deduplication Delta Compression. In *USENIX Conference on File and Storage Technologies (FAST)*. 247–264.

[16] Calicrates Policroniades and Ian Pratt. 2004. Alternatives for Detecting Redundancy in Storage Systems Data. In *USENIX Annual Technical Conference (ATC)*. 73–86.

[17] Claude E. Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal,* (1948), 379–423.

[18] Philip Shilane, Mark Huang, Grant Wallace, and Windsor Hsu. 2012. WAN-optimized replication of backup datasets using stream-informed delta compression. *ACM Transactions on Storage (ToS)* (2012), 13:1–13:26.

[19] Philip Shilane, Grant Wallace, Mark Huang, and Windsor Hsu. 2012. Delta Compressed and Deduplicated Storage Using Stream-Informed Locality. In *USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*. 1–10.

[20] Grant Wallace, Fred Douglis, Hangwei Qian, Philip Shilane, Stephen Smaldone, Mark Chamness, and Windsor Hsu. 2012. Characteristics of backup workloads in production systems. In *USENIX Conference on File and Storage Technologies (FAST)*. 33–48.

[21] Wen Xia, Yukun Zhou, Hong Jiang, Dan Feng, Yu Hua, Yuchong Hu, Liu, and Yucheng Zhang. 2016. FastCDC: a Fast and Efficient Content-Defined Chunking Approach for Data Deduplication. In *USENIX Annual Technical Conference (ATC)*. 101–114.

[22] Yucheng Zhang, Wen Xia, Dan Feng, Hong Jiang, Yu Hua, and Qiang Wang. 2019. Finesse: Fine-Grained Feature Locality based Fast Resemblance Detection for Post-Deduplication Delta Compression. In *USENIX Conference on File and Storage Technologies (FAST)*. 121–128.

[23] Jacob Ziv and Abraham Lempel. 1978. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory (TIT)* (1978), 530–536.

[24] Xiangyu Zou, Cai Deng, Wen Xia, Philip Shilane, Haoliang Tan, Haijun Zhang, and Xuan Wang. 2021. Odess: Speeding up Resemblance Detection for Redundancy Elimination by Fast Content-Defined Sampling. In *IEEE International Conference on Data Engineering (ICDE)*. 480–491.