FSDedup: Feature-Aware and Selective Deduplication for Improving Performance of Encrypted Non-Volatile Main Memory

CHUNFENG DU, School of Informatics, Xiamen University, Xiamen, China
ZIHANG LIN, School of Informatics, Xiamen University, Xiamen, China
SUZHEN WU, Xiamen Key Laboratory of Intelligent Storage and Computing, School of Informatics, Xiamen University, Xiamen, China and Wuhan National Laboratory for Optoelectronics, Wuhan, China
YIFEI CHEN, School of Informatics, Xiamen University, Xiamen, China
JIAPENG WU, School of Informatics, Xiamen University, Xiamen, China
SHENGZHE WANG, School of Informatics, Xiamen University, Xiamen, China
WEICHUN WANG, Hikivision, Hikivision, Wuhan, China
QINGFENG WU, School of Informatics, Xiamen University, Xiamen, China
BO MAO, School of Informatics, Xiamen University, Xiamen, China

Enhancing the endurance, performance, and energy efficiency of encrypted Non-Volatile Main Memory (NVMM) can be achieved by minimizing written data through inline deduplication. However, existing approaches applying inline deduplication to encrypted NVMM suffer from substantial performance degradation due to high computing, memory footprint, and index-lookup overhead to generate, store, and query the cryptographic hash (fingerprint). In the preliminary ESD [14], we proposed the Error Correcting Code (ECC) assisted selective deduplication scheme, utilizing the ECC information as a fingerprint to identify similar data effectively and then leveraging the selective deduplication technique to eliminate a large amount of redundant data with high reference counts. In this article, we proposed FSDedup. Compared with ESD, FSDedup could leverage the prefetch cache to reduce the read overhead during similarity comparison and utilize the cache refresh mechanism to identify further and eliminate more redundant data. Extensive experimental evaluations demonstrate that FSDedup can enhance the performance of the NVMM system further than the ESD. Experimental results show that FSDedup can improve both write and read speed by up to 1.8×,

This work was supported by the National Key R&D Program of China No. 2023YFB4502703, the National Natural Science Foundation of China under Grant No. U22A2027 and No. 61972325, and Open Project Program of Wuhan National Laboratory for Optoelectronics No. 2021WNLOKF011.
Authors' Contact Information: Chunfeng Du, School of Informatics, Xiamen University, Xiamen, Fujian, China; e-mail: duchunfeng@stu.xmu.edu.cn; Zihang Lin, School of Informatics, Xiamen University, Xiamen, Fujian, China; e-mail: linzihang@stu.xmu.edu.cn; Suzhen Wu, Xiamen Key Laboratory of Intelligent Storage and Computing, School of Informatics, Xiamen University, Xiamen, Fujian, China and Wuhan National Laboratory for Optoelectronics, Wuhan, Hubei, China; e-mail: suzhen@xmu.edu.cn; Yifei Chen, School of Informatics, Xiamen University, Xiamen, Fujian, China; e-mail: chenyifei1024@stu.xmu.edu.cn; Jiapeng Wu, School of Informatics, Xiamen University, Xiamen, Fujian, China; e-mail: jiapengwu@stu.xmu.edu.cn; Shengzhe Wang, School of Informatics, Xiamen University, Xiamen, Fujian, China; e-mail: wangshengzhe@stu.xmu.edu.cn; Weichun Wang (Corresponding author), Hikivision, Hikivision, Wuhan, Hubei, China; e-mail: wangweichun@hikvision.com; Qingfeng Wu, School of Informatics, Xiamen University, Xiamen, Fujian, China; e-mail: qfwu@xmu.edu.cn; Bo Mao (Corresponding author), School of Informatics, Xiamen University, Xiamen, Fujian, China; e-mail: maobo@xmu.edu.cn.

ACM 1553-3077/2024/08-ART22
https://doi.org/10.1145/3662736

enhance Instructions Per Cycle by up to 1.5×, and reduce energy consumption by up to 2.0×, compared to ESD.

CCS Concepts: • **Information systems** → **Storage management**; • **Hardware** → **Non-volatile memory;**

Additional Key Words and Phrases: Non-volatile main memory, ECC mechanism, content locality, selective deduplication, prefetch cache, refresh mechanism

ACM Reference Format:
Chunfeng Du, Zihang Lin, Suzhen Wu, Yifei Chen, Jiapeng Wu, Shengzhe Wang, Weichun Wang, Qingfeng Wu, and Bo Mao. 2024. FSDedup: Feature-Aware and Selective Deduplication for Improving Performance of Encrypted Non-Volatile Main Memory. *ACM Trans. Storage* 20, 4, Article 22 (August 2024), 33 pages. https://doi.org/10.1145/3662736

1 Introduction

As the demand for data and in-memory workloads continues to grow, there is a growing need for larger main memory capacities. However, traditional **dynamic random-access memory (DRAM)** faces limitations in terms of scalability and power leakage, which diminish its attractiveness as a solution [26]. Researchers are exploring **non-volatile memory technologies (NVMs)** such as **Phase Change Memory (PCM)**, Spin-Torque Transfer RAM, and Resistive RAM as promising alternatives, owing to their high density and scalability [19, 37, 40]. Generally speaking, the PCM [51] is often used as a representative NVM for research and analysis purposes.

Developing high-performance and energy-efficient encrypted NVMM poses significant challenges due to the inherent characteristics of the media. First, writing data to NVMM introduces higher latency compared to read operations, consumes substantial energy, and diminishes the limited endurance of the memory [50]. Common NVMM has inherent limitations in write endurance due to variations in manufacturing processes [51]. When the number of data writes to NVMM's cell exceeds the designated limit, it can incur *write penetration* that can cause data loss, read/write errors, or irreparable damage for the entire NVMM. Therefore, reducing the frequency of write operations to NVMM is crucial for optimizing their performance and longevity. Second, existing research has predominantly focused on minimizing write operations through fine-grained bit changes, aiming to improve energy efficiency and overall system performance. However, data written to NVMM are typically encrypted to address the security concerns associated with the non-volatile nature of the memory. Unfortunately, most encryption algorithms alter approximately half of the bits, even if only a single bit of the data is modified [4, 49]. While ensuring data security, this also exacerbates the wear-out of NVMM, rendering conventional bit-level data reduction techniques [11] ineffective when applied to encrypted NVMM. Therefore, developing efficient strategies that simultaneously address data security and reduce the impact on NVMM endurance is paramount. By exploring novel encryption algorithms and innovative data reduction techniques designed explicitly for encrypted NVMM, researchers aim to strike a delicate balance between safeguarding data integrity and extending the lifespan of NVMM, ultimately enabling the realization of high-performance and energy-efficient encrypted NVMM systems.

Previous studies [41, 56] and our analysis have illustrated that a significant number of cache lines evicted from the **Last-Level Cache (LLC)** are redundant, illustrated in Figure 1, which implies that the system performance can be improved by utilizing deduplication technology to eliminate the duplicate cache lines. Moreover, existing studies have been considered to combine deduplication technology in encrypted NVMM systems [46, 56], e.g., **Deduplication-after-Encryption (DaE)**, **Deduplication-before-Encryption (DbE)**, and **Parallelism of**

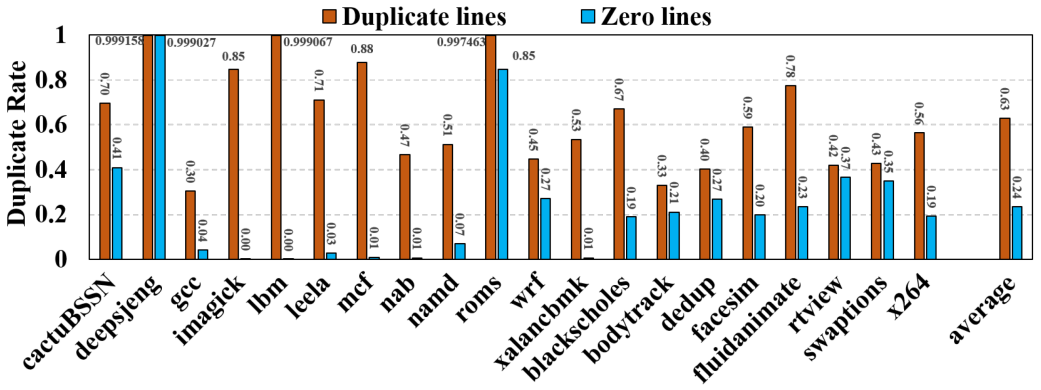


Fig. 1. Duplicate rate of cache lines *(the first 12 applications are from SPEC CPU 2017 and the rest 8 applications are from PARSEC)*. Note that the duplicate rate represents redundant data in proportion to the total data.

Deduplication and Encryption (PDE). The *DaE* approach is not viable, since the data change dramatically after encryption in an encrypted system, resulting in distinct data before and after encryption. Moreover, the *PDE* approach is also impractical as it can incur high calculated energy overhead, even though the computational latency overhead of non-duplicate cache lines is hidden [56]. However, directly implementing the DbE approach is also non-trivial and can considerably impair the overall system performance under worst-case conditions. The main reason includes three aspects: (1) High computation and energy overhead. Eliminating redundancy using fingerprints is a crucial step of a deduplication system. The fingerprint is usually calculated leveraging a hash function, such as MD5 or SHA-1/256, introducing high latency in the order of hundreds of nanoseconds. However, not only is the deduplication-induced computational overhead more than the latencies of reading and writing, but this is exceptionally high for latency-sensitive NVMM systems. Moreover, traditional deduplication methods apply a hash function on all data blocks, whether redundant or not, which adds an extra hash calculating latencies for non-duplicate cache lines and increases access latencies. (2) High memory space and lookup overhead. Generally speaking, deduplication-based NVMM systems need to store the fingerprint in NVMM and maintain a small proportion in the cache to quickly identify redundant data. When the fingerprint of data is coming, the system first searches in the cache and might also search in NVMM if not matched. However, as data volume increases, it becomes unrealistic to store such increasing fingerprints in memory. This also results in a fingerprint lookup bottleneck from NVMM, similarly to the fingerprint lookup bottleneck from disk in traditional disk-based deduplication systems [34], as the fingerprint lookup from NVMM procedure is on the I/O critical path. (3) Read overhead on the write path. Lightweight hash similarity judgment inevitably requires reading similar data from the underlying device for further comparison [30, 45, 56]. However, these similar data include redundant and collision data, leading to the need to determine whether the data are redundant. Therefore, the read overhead associated with the similarity comparison process is not overlooked.

To address these challenges, we built ESD [14]. Specifically, ESD leverages ECC values associated with cache lines to identify data similarity. This can eliminate costly hash computations and reduce energy consumption. Additionally, selective deduplication is exploited to remove redundant cache lines by only storing ECC values with high reference counts, taking advantage of content locality within cache lines. However, we found that overall performance in practical systems could be improved. There are two main reasons: (1) determining data similarity through

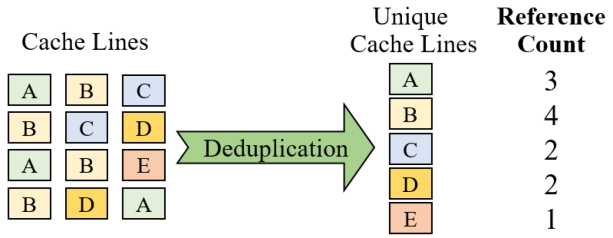


Fig. 2. The reference count (RC) in deduplication system.

the ECC mechanism requires reading data with the same ECC fingerprint for further byte-by-byte comparisons, resulting in significant read overhead, and (2) selective deduplication only targets items with high reference counts, lacking adaptability to dynamic workload changes. To overcome these limitations, we proposed FSDedup further, an enhancement version compared with ESD. First, FSDedup uses a prefetch cache to respond to read operations during similarity comparisons. Second, FSDedup can leverage a cache refresh mechanism to provide dynamism for identifying more redundant data. Experiment results show that FSDedup significantly outperforms the other methods, e.g., Dedup_SHA1, DeWrite [56], and ESD [14] regarding read/write performance, energy efficiency, and tail latency. Additionally, Dedup_SAH1 is a comparative traditional deduplication scheme implemented by us. It uses the SHA1 algorithm to calculate the hash value as the unique identifier to identify whether the data are redundant, detailed in Section 4.1.

2 Background and Motivation

In this section, we first introduce the features of the workload, including data redundancy and content locality, in Section 2.1. Subsequently, we discuss employing the device ECC mechanism, emphasizing their potential for detecting similar data patterns in Section 2.2. Finally, considering these challenges that have emerged when applying deduplication technology in the NVMM system, we delve into the motivation behind this article in Section 2.3.

2.1 Workload Feature

Understanding the workload characteristics is important for memory design. We observed that multiple duplicate cache lines, accounting for a total volume of 33.1% to 99.9% with an average of 62.9%, are evicted from the LLC on the CPU side, as illustrated in Figure 1. Meanwhile, further analysis discovers that these duplicate cache lines consist largely of non-zero duplicates, apart from zero duplicate cache lines.

Furthermore, our investigation revealed a skewed distribution of cache lines across all applications, a phenomenon we refer to as content locality. This indicates that a small subset of unique cache lines is referenced frequently. In Figure 2, we illustrate the concept of **reference count (RC)** in the deduplication process. To gain deeper insights into the characteristics of data workloads, we conducted an in-depth analysis and compiled statistics from 20 different applications sourced from SPEC CPU 2017 and PARSEC. The results, presented in Table 1, highlight the cache line analysis before and after deduplication for all 20 applications. The result underscores the prevalence of high content locality across these applications. For instance, cache lines with reference counts of 1,000 and above represent a mere 0.08% (approximate 1%) of the total unique cache lines. However, they contribute significantly to the total storage data volumes, accounting for an average of 42.6% before deduplication application in the NVMM system. These findings shed light on the substantial impact of content locality on storage utilization and emphasize the importance of efficient deduplication strategies.

Table 1. The Analysis of Cache Lines across All 20 Applications from SPEC CPU 2017 and PARSEC (before and after Deduplication)

	Before Deduplication	After Deduplication
RC = 1	31.5%	81.3%
1 < RC <= 10	15.6%	16.0%
10 < RC <= 100	7.2%	2.4%
100 < RC <= 1,000	3.1%	0.2%
1,000 < RC	42.6%	0.1%

Nevertheless, the inherent redundancy and content locality within memory access patterns underscores the potential of employing inline deduplication within the NVMM system to effectively curb write traffic. Therefore, exploring how to design and effectively leverage the deduplication technique to enhance system performance based on content locality within these workloads remains an intriguing question warranting thoughtful consideration.

2.2 Device Feature

ECC is commonly used in modern systems, especially server machines, to provide error detection and correction in case of hardware memory errors. Conventional error correction and checking technology, before ECC, was Parity [2]. Parity can only check single-bit errors at unknown locations but cannot correct the error. In contrast, ECC provides both error detection and correction capabilities in the case of memory bit errors [21, 28, 32] and is widely used in computer systems. Additionally, ECC values can be applied to a fine-grained cache line, i.e., Per-Word, e.g., an 8-byte word is matched with an 8-bit ECC. Each 8-byte word generates an 8-bit ECC, and a 64-byte cache line generates a 64-bit ECC [47]. Many processors with large capacity caches have ECC-protected L2 and L3 (also known as the last-level cache) caches [48]. For example, Qualcomm Centriq 2400 Processor has a 512KB L2 cache with ECC and a distributed 60 MB L3 Cache (12 × 5 MB) with ECC [31], AMD EPYC processors have internal ECC-enabled L2 and L3 caches [3], and Intel Xeon Processors have internal ECC logic to detect errors for L2 and L3 caches [15, 20]. Most importantly, existing works demonstrate that the memory controller computes ECC for each cache line and sends it on specific bits along with the data [47] when cache lines are evicted to NVMM from LLC.

In general, the ECC from the memory controller is considered absolutely secure and cannot be compromised by an attacker [44]. Besides the error detecting and correcting abilities, ECC can be used for data similarity identification. For instance, if two ECC values are different, then it indicates that their corresponding cache lines are different. Conversely, if their ECC values are the same, then there is a high possibility, though not 100%, that their cache lines are also the same. The corresponding cache line is further fetched to the memory for a byte-by-byte comparison [42, 56]. Therefore, it is implied that this property can be exploited to effectively filter out non-duplicate cache lines based on similar data identification without extra overhead.

2.3 Challenges and Motivation

Previous studies reveal that applying inline deduplication can decrease the system performance in the worst case, even though the duplicate cache lines can be eliminated significantly for saving space [27, 36, 56]. To improve the efficiency of NVMM systems with inline deduplication, there are yet some challenges remaining to be faced, illustrated as follows.

The first challenge is the high computational and energy overhead. Traditional inline deduplication technologies use the fingerprint (e.g., MD5 or SHA1) to identify duplicate data, which leads to

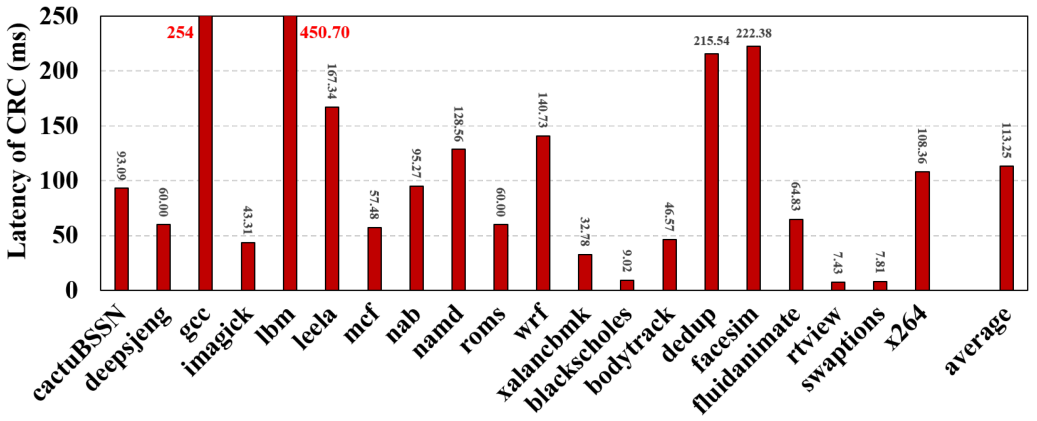


Fig. 3. The total latencies of applying the CRC algorithm redundancy detecting in the DeWrite under different benchmarks after running for some time.

significant computational latency and energy overhead [42, 56]. Moreover, Figure 3 shows the total increased time of applying the CRC algorithm under different applications in DeWrite [56] after running for some time. The increased time is significant, up to 450.7 ms with an average of 113.3 ms, particularly for applications with moderate deduplication and high write rates. The reasons for this are obvious. First, the computation overhead is required for all cache lines in DeWrite, not just for non-duplicate cache lines. The reason is that the parallel encrypting mechanism is only for non-duplicate cache lines. However, in practice, all data are subject to CRC calculation, regardless of whether the data redundancy is known. Second, even with strictly guaranteed correct predictions, the energy overhead for all cache lines due to hashlike computation cannot be overlapped, inevitably imposing a significant energy overhead on the system.

The second challenge is the memory footprint and lookup overhead in NVMM for the deduplication-induced metadata. During the process of deduplication, the fingerprint index is important metadata. For instance, the memory footprint required to store the metadata of DeWrite [56] is 6.25% of the total unique storage space for a 256-byte cache line. This increases to 25% when the cache line size is 64 bytes, a size widely used in previous studies [11, 49]. Meanwhile, storing whole fingerprints in NVMM and only maintaining a small number of fingerprints in the memory cache will introduce fingerprints NVMM_lookup bottleneck. This affects system performance in addition to the non-negligible NVMM space overhead. Therefore, effectively placing metadata and improving data access efficiency are questions worth considering. Figure 4 illustrates the rate of duplicate cache lines filtered by the fingerprints stored in the memory cache and NVMM, as well as the performance overhead induced by the fingerprint NVMM_lookup operation in the NVMM system. The data show that, on average, 51.0% of duplicate cache lines can be filtered by the fingerprints stored in the memory cache, while 13.7% can be filtered by the fingerprints stored in NVMM. However, the additional fingerprint NVMM_lookup processes required to filter these 13.7% duplicate cache lines result in a performance degradation of up to 90.7%, with an average degradation of 49.2%. It is important to note that dedicating significant system resources, such as NVMM space and performance overhead, to achieve a mere 13.7% improvement in filtering duplicate cache lines is not an efficient design choice in computer architecture.

The third challenge is read operations overhead on the write path. Even though traditional deduplication leverages strong hashes (e.g., SHA1 or MD5) to identify redundant data, the state-of-the-art DeWrite proposes to exploit the light hash (e.g., CRC-32) to identify data similarity to

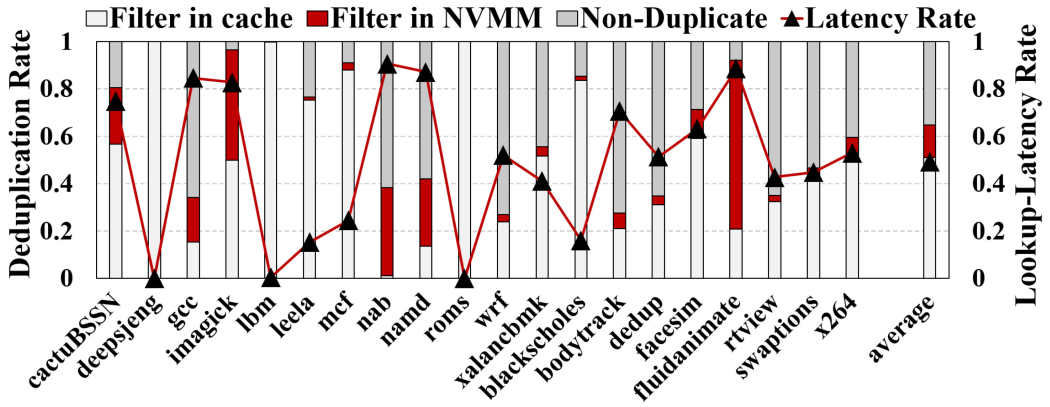


Fig. 4. The rate of duplicate cache lines filtered by the fingerprints stored in memory cache and NVMM, and the fingerprints NVMM_lookup latency rate of writes.

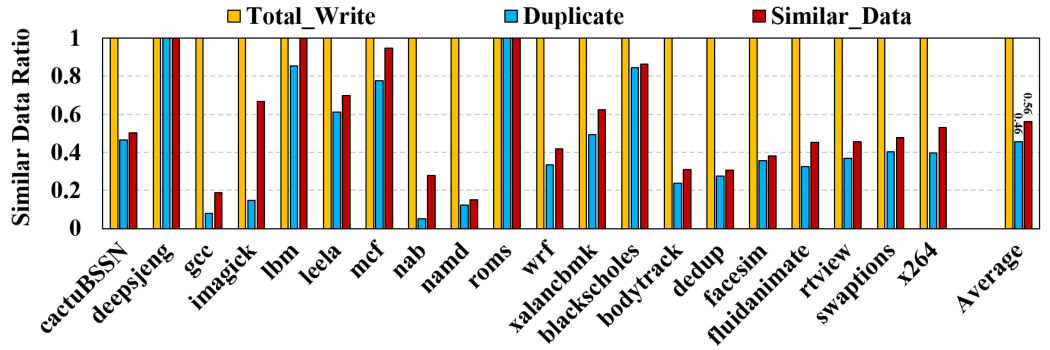


Fig. 5. The similar data ratio in all cache lines on the write critical path after running for a period of time. The Total_Write denotes the whole cache lines written to NVMM; the Duplicate denotes the redundant ratio; and the Similar_Data denotes the similar data ratio in all cache lines.

avoid heavy hash computation [56], which inevitably incurs the problem of hash conflicts. To avoid the influence of hash conflicts, existing methods usually need a read operation to retrieve similar data from the underlying memory device, followed by a byte-by-byte comparison to determine redundancy [14, 56]. Figure 5 shows a similar data ratio in all cache lines on the write critical path after running for a period of time. As we can see, the average percentage of similar data is as high as 56%, which is also 10% higher than redundant data. This also implies that similar data are usually more than redundant data. However, all these similar cache lines must be considered collisions, because the similarity property of the ECC identification mechanism does not enable us to accurately distinguish between identical ECC fingerprint entries due to the same/different data. Therefore, multiple read operations to the NVMM must be performed, which no doubt incurs significant overhead.

The fourth challenge is the inefficiency of caching. In our previous implementation, considering the need to minimize the storage space occupation and the lookup overhead of metadata in the underlying device, we employed a selective deduplication mechanism to cache only the metadata entries with higher reference counts. The data eliminated in this way are multiple duplicate data and are also more valuable. However, in a real system, due to the limited cache space, the reservation of high reference count data is also limited, resulting in the data that become high reference

count later with the time change is not well stored in the cache. As a result, such a caching mechanism is not dynamic and becomes a key factor limiting caching efficiency. For example, in the previous version of our implementation, the deduplication ratio of ESD [14] was 47.8%, compared to full deduplication methods (e.g., Baseline, Dedup_SHA1, DeWrite), there is approximately 18.2% of redundant data that is not detected and removed.

In summary, to improve the performance of the NVMM system, eliminating the redundant data written to NVMM by utilizing inline deduplication is a good choice. However, the above challenges are the ones we have to face. Therefore, to tackle these challenges and improve the NVMM system performance, we should also consider the following factors carefully. First, we should consider whether the features of the device (ECC mechanism) and the workload (content locality) should be carefully considered and exploited. This is for identifying the similarity of cache lines without inducing expensive hash calculating. It is also for performing the deduplication effectively while decreasing the storage overhead of fingerprints. Therefore, we design the ECC-based Similarity Identification illustrated in Section 3.3 and Selective Deduplication demonstrated in Section 3.4. Second, in practice, we found that similarity identification requires many read operations to compare the data for consistency, resulting in a large read overhead. Therefore, we designed the cache Prefetching Mechanism Section 3.5.1. Finally, we found that the effectiveness of the selective deduplication depends on the cache's efficiency. Therefore, we designed a Cache Refreshing Mechanism, as described in Section 3.5.2, to give the cache dynamism and enable the identification of more cache entries.

3 The Design of FSDedup

In this section, we propose FSDedup, an enhanced version compared to ESD [14]. The primary distinction between the two lies in the cache optimization mechanism. We begin with the architecture overview of FSDedup. We then introduce the data structures employed in FSDedup. Subsequently, we present the selective deduplication. Subsequently, we delve into exploring the cache mechanisms, which encompass the prefetch cache mechanism and the refresh mechanism. Finally, we address the security and consistency issues associated with FSDedup.

3.1 The Architecture Overview of FSDedup

FSDedup aims to eliminate computational overhead, reduce metadata footprint, eliminate NVMM_lookup overhead, reduce read overhead, and improve selective deduplication efficiency. The approach consists of applying ECC-based similarity recognition to encrypted NVMM with the assistance of the device (ECC mechanism) and implementing selective deduplication based on cache line characteristics (content locality), optimized using caching mechanisms. Specifically, FSDedup first intercepts existing ECC values associated with cache lines as the fingerprints to identify data similarity and filter the non-duplicate cache lines on the critical write path, illustrated in Section 3.3. Subsequently, for these similar cache lines that have a high probability of being redundant, FSDedup introduces a selective deduplication to remove the cache lines with high reference counts merely on the write path by leveraging the fingerprint cache based on the reference counts, illustrated in Section 3.4. Finally, FSDedup introduces a cache optimization mechanism, consisting of the Prefetch Cache and Cache Refresh techniques. The Prefetch Cache aims to reduce the read overhead caused by comparing similar data during the write path. However, the Cache Refresh technique is designed to enhance the performance of the fingerprint cache by identifying and eliminating more redundant data. Additionally, the written data must be encrypted when evicted from the CPU-side to the memory bus to defend against a series of data attacks, e.g., hardware theft and data leakage. Similarly to previous studies [4, 49, 56], FSDedup leverages the counter model encryption algorithm for encrypting cache lines.

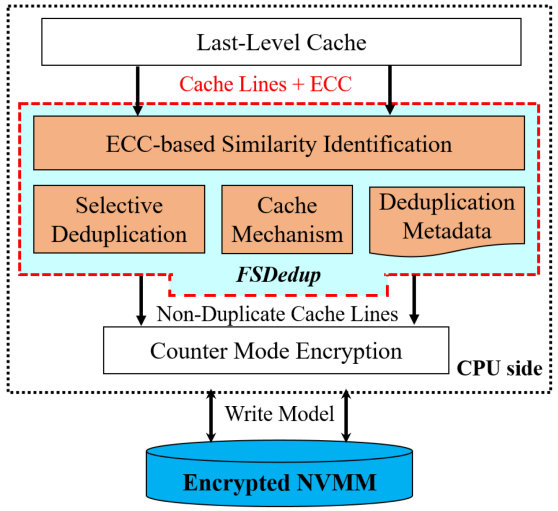


Fig. 6. The architecture overview of FSDedup.

Figure 6 shows the architecture overview of FSDedup, which operates in conjunction with other modules and resides within the secure confines of the memory controller on the CPU-side [44]. Meanwhile, FSDedup mainly consists of four functional modules, namely the ECC-based Similarity Identification, the Selective Deduplication, the Deduplication Metadata, and Cache Mechanism. When a cache line is evicted from the LLC, FSDedup obtains the ECC values associated with the corresponding cache lines and passes them to the ECC-based Similarity Identification module, which is mainly responsible for detecting non-duplicate cache lines by checking their ECC values. However, collisions can occur for these cache lines that are not filtered out, since different cache lines may generate the same ECC. Though the probability is low, we cannot remove these cache lines based on their ECC values and must further compare their content. Therefore, when the ECC item is judged as a similar cache line, the ECC-based Similarity Identification module passes it to the Selective Deduplication module. The Selective Deduplication module is primarily responsible for conducting selective deduplication by removing identical cache lines. The cache line is considered a non-duplicate one and is written to the NVMM if its fingerprint or content is different from the existing one. However, regardless of redundancy or not, once the ECC-based fingerprints are the same, FSDedup needs to read the cache lines out of the NVMM and perform a byte-by-byte comparison, which will undoubtedly incur a huge read overhead. Therefore, the Cache Mechanism module plays a crucial role in enhancing the cache ratio of ECC-based fingerprints and minimizing the overhead of read operations on the critical write path compared with ESD. Moreover, the Deduplication Metadata module is mainly responsible for managing the metadata used in FSDedup and storing them in the memory cache and NVMM, undoubtedly increasing the memory space overhead. The Deduplication Metadata includes two primary data structures, the **ECC-based Fingerprint Index Table (EFIT)** and the **Address Mapping Table (AMT)**, which are described in the following Section 3.2.

3.2 Data Structures

There are two main data structures utilized in FSDedup to support the ECC-based data similarity identification and selective deduplication, namely EFIT and AMT, demonstrated in Figure 7.

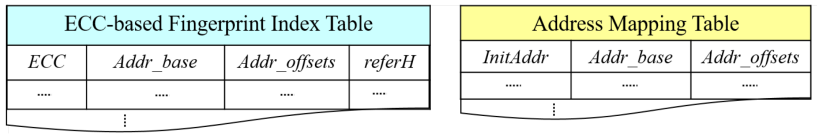


Fig. 7. Two data structures of FSDedup.

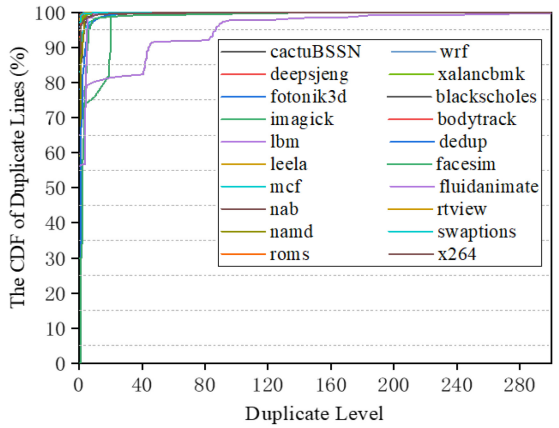


Fig. 8. The CDF of redundant cache lines.

The EFIT records the ECC-based fingerprint items with high reference counts for deduplication processes. Each entry in EFIT includes four variables, shown as $\langle ECC, Addr\_base, Addr\_offsets, referH \rangle$. The *ECC* is obtained from the memory controller when the corresponding cache line is evicted from LLC. The *Addr_base* occupies 4 bytes, which presents the base addressing space. However, with the release of the Intel Optane PM device [16], the total capacity of NVMM could reach 1.5 TB. The 4 bytes addressing space is not sufficient in practice, because the total addressing space is only 256 GB based on 64 bytes cache line size, which is a constant cache line size granularity generated by the CPU core and widely used in the previous studies [4, 11, 49]. Therefore, FSDedup adopts a simple tradeoff that adds *Addr_offsets* as the offset of addressing space, which means that the real physical address comprises two parts: *Addr_base* (4-byte) and *Addr_offsets* (1-byte). To get the physical address, FSDedup first shifts left 8-bit of *Addr_base* and then plus *Addr_offsets*. The 40-bit physical address could address up to 64 TB of total memory space, which is enough for current servers. The *referH* indicates the number of remapping to the same physical address. It is set to be 1 byte, which is enough, since more than 99.9% of the reference counts are less than 1,000 as shown in Table 1. Meanwhile, as we can see from Figure 8 most of these redundant cache lines are duplicated less than 255 times, which also corresponds to Section 2.1. When the reference counts increase more than the limited number, FSDedup will not consider it as a duplicate and avoids overflowing the reference.

The AMT plays a crucial role in a deduplication-based NVMM system by recording the mapping relationship between logical addresses and their corresponding physical addresses. The AMT, denoted as $\langle initAddr, Addr\_base, Addr\_offsets \rangle$, establishes a many-to-one relationship and is updated during the write path. The *initAddr* represents the logical address of a cache line allocated by the user when it is evicted from the CPU. The physical address, consisting of the *Addr_base* and *Addr_offsets*, carries the same meaning as in the EFIT. To provide efficient query performance, FSDedup maintains the AMT in NVMM and buffers frequently accessed items with high reference counts in the memory cache.

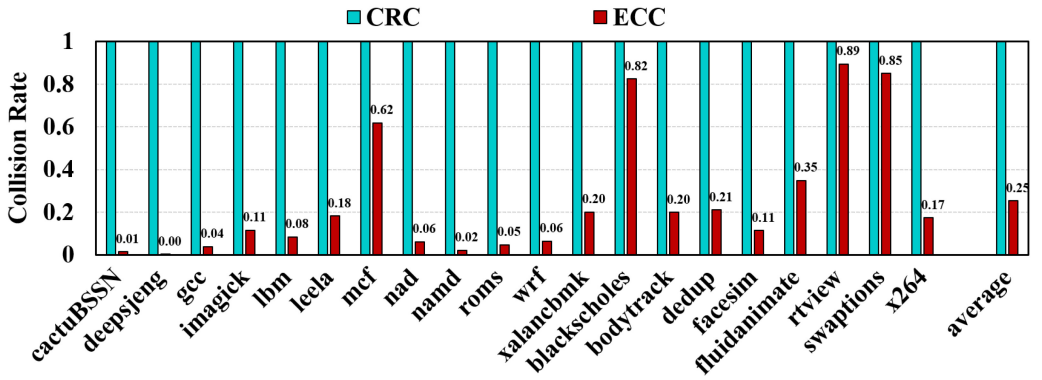


Fig. 9. The architecture overview of FSDedup.

With the ability to identify similarities in the ECC items stored in the EFIT, FSDedup efficiently filters out non-duplicate cache lines and determines the similarity of cache lines without introducing significant latency overhead, as discussed in Section 3.3. In cases where cache lines exhibit similarity, FSDedup conducts a byte-by-byte comparison to accurately determine the redundancy of the cache lines, as explained in Section 3.4.

3.3 ECC-based Similarity Identification

The fingerprint of data is utilized as a unique identifier to detect and filter data to eliminate redundancy and save efficient storage space for typical deduplication. However, while typical hash functions used to compute fingerprints (e.g., MD5 [13] or SHA1 [8]) can be considered as unique identifiers of the data, they are computed with extremely high latency, e.g., 312 ns for MD5 [56] and 321 ns for SHA1. This can lead to cascade blocking and greatly extend the write latency of deduplication-based memory systems, especially for NVMM with ultra-low latency. The state-of-the-art method, DeWrite [56], leverages the lightweight CRC algorithm to detect similar cache lines and exploits a parallel method for concealing the influence of computational overhead as much as possible. However, this parallelism strictly depends on the prediction mechanism, which can cause heavy overhead. The reason consists of two aspects. First, if the result of the prediction is true, then DeWrite needs to calculate the CRC for all duplicate cache lines, even though these redundant cache lines will be removed. Second, if the result of the prediction is false, then DeWrite will lead to two extreme cases, e.g., CRC calculation and encryption become serial operations or many duplicate cache lines for invalid encryption. In addition, CRC also needs extra hardware logic support, besides the high hash collision rate demonstrated in Figure 9. On the contrary, FSDedup utilizes the existing ECC value associated with each cache line as the fingerprint. This approach definitively filters out non-duplicate cache lines, preventing costly hash computation and identifying cache line similarity. Hence, the latency of the hash calculation can be eliminated on the critical write path in FSDedup.

Figure 10 illustrates the workflow of ECC-based Similarity Identification. When cache lines are evicted from the CPU's LLC, the memory controller will calculate the ECC value associated with the cache line and send it to the specified bit of data, as illustrated in Section 2.2. Hence, the FSDedup retrieves the corresponding ECC items from the memory controller. This retrieval process is highly efficient, incurring minimal overhead without compromising the error-checking function of ECC. Subsequently, FSDedup searches the EFIT to determine the presence of these ECC items. If a match is found, then it indicates that the cache lines with the associated ECC items are likely to be similar due to collision probability. A byte-by-byte comparison based on the data

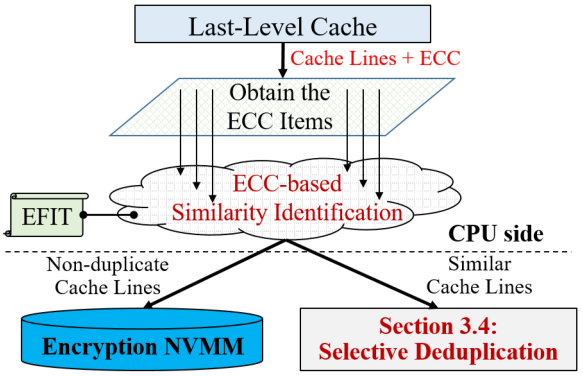


Fig. 10. The workflow of ECC-based similarity identification.

stored in NVMM must be performed to confirm cache line duplication. Therefore, the results of ECC-based cache line similarity identification are then passed to the Selective Deduplication module. Throughout this process, the latency introduced by the deduplication logic is negligible. The reasons include two aspects. First, FSDedup eliminates the need for fingerprint computation, reducing latency overhead. Second, metadata lookup operations are performed exclusively within the cache, as FSDedup executes selective deduplication, as depicted in 3.4. Additionally, FSDedup updates the AMT and caches information relevant to mapping items.

Though DeWrite leverages the lightweight CRC to reduce the computational overhead, the high CRC collision rate can exacerbate the write latency, because many read accesses need to be performed on NVMM for further comparison. Therefore, using existing ECC values instead of SHA1 or CRC not only significantly reduces the computational overhead in filtering non-duplicate cache lines but also decreases the overhead of reading from NVMM when collisions occur. To accelerate the read/write processes, the ECC-based fingerprints should be stored in the memory cache as much as possible. It is not realistic to store all fingerprint items in the memory cache. In general, the whole fingerprints should be stored in NVMM in typical deduplication-based NVMM systems, such as the Dedup_SHA1 and DeWrite [56]. Unlike the full deduplication method used in the traditional Dedup_SHA1 and the DeWrite, however, FSDedup only stores the ECC-based fingerprints with high reference counts in the memory cache and performs a selective deduplication method for the NVMM system, which keeps the same design with ESD [14]. Thus, not only can the latency of accessing the fingerprint metadata stored in NVMM on the critical write path be eliminated, but also the memory space overhead can be decreased by FSDedup.

Although we leverage ECC to identify similar data, once the ECC-based fingerprint of data matches an existing one, it is judged as similar data. Subsequently, FSDedup still needs to go to the NVMM device to read the data and then perform the byte-by-byte comparison, as illustrated in Section 2.3. Unlike ESD, we propose a Prefetching Cache in FSDedup, as illustrated in Section 3.5.1, which can respond to read requests in the similarity comparison through the cache mechanism and reduce access to underlying devices.

3.4 Selective Deduplication

As mentioned above, the fingerprint Lookup overhead from NVMM during deduplication could cause significant performance degradation. The reason is that Dedup_SHA1 and DeWrite perform full deduplication, even though the reference count of cache lines is only 1. Once the fingerprint is not in the fingerprint cache, read accesses of the fingerprints stored in the NVMM are performed. These additional fingerprint lookup operations from NVMM minimally contribute to the overall

ALGORITHM 1: GET (fingerprint)

```
// cacheline_table - key type: string - value type: cacheline struct pointer
// frequency_table - key type: int - value type: cacheline struct pointers list
cacheline = cacheline_table.find(fingerprint)
if (cacheline == NULL)
    return NULL
cl_val = cacheline.value
cl_freq = cacheline.frequency
frequency_table.getCachelineList(cl_freq+1).add(cacheline)
frequency_table.getCachelineList(cl_freq).remove(cacheline)
// If the cacheline list is empty, we need to delete it in the frequency table and then update the
min_frequency
if (frequency_table.getCachelineList(cl_freq).size == 0)
   frequency_table.removeCachelineList(cl_freq)
   if (min_frequency == cl_freq)
      min_frequency += 1
if (cacheline_table.size != 0)
   REFRESH()
return cl_val
```

redundancy elimination, by less than 13.7%. However, they induce a considerable performance overhead on the write path by up to 90.7% with an average of 49.2%, as shown in Figure 4. It is unwise to eliminate a small amount of redundant data while spending colossal overhead.

In ESD [14], we introduced selective deduplication as a solution to mitigate the overhead associated with NVMM fingerprint lookup. This approach leverages the LRCU cache strategy to maintain only partial ECC entries within the ECC-based fingerprint cache on the CPU side. The LRCU policy is designed to prioritize the replacement of ECC-based fingerprints with lower reference counts, thereby allowing fingerprints with higher reference counts to remain in the cache longer. However, we observed that the previously implemented LRCU version stored only cache index entries with high reference counts and did not dynamically adapt to workload changes. As a result, the identification of redundant data was limited. We utilize an enhanced LRCU policy with a refresh mechanism to address this limitation to enable dynamic adaptation to load changes. By incorporating mechanisms for real-time monitoring and adjustment of cache entries based on evolving reference counts, we aim to improve the effectiveness of selective deduplication in identifying and managing redundant data within the NVMM environment. This refined approach aims to optimize the ECC-based fingerprint cache utilization and contribute to the overall system performance and efficiency.

Effectively maintaining valuable ECC-based fingerprints in the cache for filtering cache lines is critical in selective deduplication. The effectiveness of the LRCU caching strategy, in other words, relies on the value of the fingerprint items stored in it. After multiple changes in the reference counts of cached entries, however, certain fingerprint items that were considered valuable in the previous period may remain unused for a significant duration. Despite this, they continue to occupy space in the cache, as depicted in Section 2.3. Hence, this prompts us to propose a cache refresh mechanism in FSDedup, as demonstrated in Section 3.5.2, aimed at introducing dynamism into the LRCU cache based on the refresh mechanism.

Figure 11 compares redundant data identification with different deduplication methods. It shows that FSDedup introduces two cache mechanisms: Prefetch and Refresh. This distinguishes it from

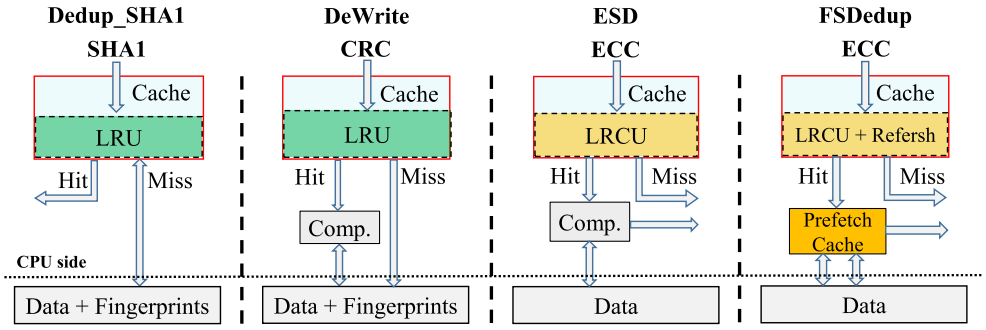


Fig. 11. A comparison of redundant data identification process under different deduplication method, (e.g., Dedup_SHA1, DeWrite, ESD, and FSDedup).

typical deduplication methods and ESD. Selective deduplication is designed to identify and eliminate duplicate cache lines through a byte-byte comparison of similar cache lines. Initially, the system searches for the ECC item in the LRCU cache, as in ESD. When an ECC item is not found, the system deems the cache line non-duplicate and writes it to the NVMM system. Conversely, if the ECC item exists, then the data are evaluated as similar, necessitating the reading of the corresponding cache line from NVMM for further comparison by byte by byte.

To reduce the read operation overhead in the critical write path, FSDedup introduces the Prefetch Cache mechanism, detailed in Section 3.5.1. This mechanism involves the system first searching the Prefetch Cache when considering the cache line associated with the ECC entry as similar to others during the FSDedup process. Only when the fingerprint items are not found in the prefetch cache, FSDedup needs to read the corresponding cache line from NVMM for further byte-by-byte comparison. The sensitivity analysis of the prefetch cache is presented in Section 4.5. Overall, the cache mechanisms in FSDedup play a crucial role in improving the selective deduplication ratio and reducing the read operation overhead in NVMM operations.

The design of the cache optimization is crucial for selective deduplication. After analyzing typical methods and ESD, two key designs have been proposed in FSDedup. The first design involves a refresh mechanism based on LRCU, which dynamically adjusts the internal cache contents to accommodate immediate changes in different workloads. The second design introduces the Prefetch caching mechanism, aimed at reducing the overhead of read operations caused by comparing similar data. Next, we introduce two foundational algorithms for the cache mechanism in FSDedup. Algorithm 1 outlines the pseudo-code for the get() operation in the LRCU strategy. Specifically, it first determines whether the corresponding fingerprint entry exists in the cache based on the incoming fingerprint value. If it is not found, then no special processing is needed; if the corresponding fingerprint entry is found, then its reference count needs to be increased. If the chain table of cache entries is empty, then the entry should be deleted, and the minimum reference rate value should be updated; if the chain table is not empty, then a refresh operation is triggered. Algorithm 2 presents the pseudo-code for the put() operation in the same strategy. Specifically, the specified fingerprint entry is first looked up in the cached fingerprint chain table. It is necessary to check whether the chain table is full if it is not found. If the chain table is full, then the entry with the lowest reference rate should be deleted. If the chain table is empty, then it should be deleted entirely, and a new fingerprint metadata chain table should be initialized, setting the reference rate of the entry with the lowest reference rate to 1. An entry will be removed if the specified fingerprint entry is found in the chain table. If the reference rate table is empty, then the entry is added, and the minimum reference rate is added to the fingerprint entry

ALGORITHM 2: PUT (fingerprint, value)

```
// cacheline_table - key type: string - value type: cacheline struct pointer
// frequency_table - key type: int - value type: cacheline struct pointers list
cacheline = cacheline_table.find(fingerprint)
if (cacheline == NULL)
   // When ECC cache is full, it needs deletion
   if (cacheline_table.IsFull())
      frequency_table.getCachelineList(min_frequency).removeLastCacheline()
      if (frequency_table.getCachelineList(min_frequency).size == 0)
         frequency_table.removeCachelineList(min_frequency)
   //Adding new cacheline, frequency is 1
   new_cacheline.initialize(fingerprint, value, 1)
   frequency_table.getCachelineList(1).insert_at_front(new_cacheline)
   min_frequency = 1
else
   frequency_table.getCachelineList(cl_freq).remove(cacheline)
   if (frequency_table.getCachelineList(cl_freq).size == 0)
      frequency_table.removeCachelineList(cl_freq)
      if (min_frequency == cl_freq)
            min_frequency += 1
   frequency_table.getCachelineList(cl_freq + 1).insert_at_front(cacheline)
   cacheline.frequency += 1
if (cacheline_table.size != 0)
   REFRESH()
```

chain table by adding 1. A refresh operation is triggered if the fingerprint cache chain table is not empty.

3.5 Cache Optimization Mechanism

Caching technology is an important tool to improve system performance and is widely used in computer systems. The large number of read operations due to similar data comparison will inevitably lead to serious system overhead. Therefore, we propose the prefetch cache mechanism, which is designed to accelerate the process of selective deduplication by reducing the read operations overhead for similar data comparison on the write path, demonstrated in Section 3.5.1. Moreover, in ESD, LRCU in selective deduplication adopts a caching strategy similar to LFU, aiming to maintain entries with high reference counts in the cache as much as possible. However, this caching strategy does not adapt to dynamic workload changes, leading to inefficiency. Therefore, we propose the cache refreshing mechanism to make LRCU detect more duplicate cache lines, illustrated in Section 3.5.2.

3.5.1 Prefetch Cache. To provide more accurate duplicate data detection for eliminating redundant data, FSDedup needs to perform additional read operations for byte-to-byte comparison to determine if it is redundant after being identified as similar data. Therefore, we aim to further reduce the read operation overhead on the critical path by introducing a prefetch cache mechanism, which fully exploits content locality as illustrated in Section 2.1.

The essence of prefetch is to retrieve data from the underlying device into the cache in advance, with the core aim being to prefetch the most valuable data so it can be directly fetched from the cache upon reaccess. However, how to cache the most useful data is critical. Traditional file

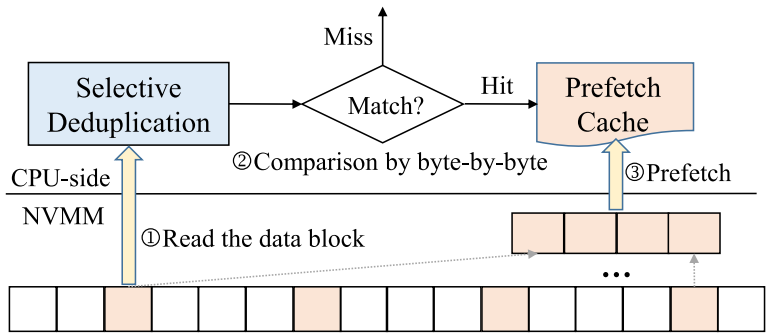


Fig. 12. The workflow of prefetch cache optimization.

system-level prefetching methods prefetch blocks at a granularity of 4K or more as the base unit, but this is ineffective in FSDedup for two reasons: (1) the precious on-chip memory cache space makes the cache prefetch and storage size limited. (2) Both parallel and online threaded read methods are bound to impose an additional latency burden on the system, which is not worth it. Inspired by the content locality, as illustrated in Section 2.1, we believe the value of caching data lies in its ability to reduce access to the underlying device through cache hits. Therefore, the prefetch cache should store as much data as possible with a high data reference rate, i.e., those with a high redundancy rate. Specifically, after the data are detected as redundant, the FSDedup will read the data from the underlying device and compare it further to determine whether it is redundant. Note that prefetching cached data do not directly read much data from the underlying device and place it in the cache. Once the data are determined to be redundant, the prefetching mechanism is activated and cached.

Moreover, the core of the prefetch cache is to solve the hash collision problem. Figure 12 shows the workflow of the prefetch cache. FSDedup first reads a similar data block from the underlying device and compares it to the incoming cache line. Because of content locality, we can store redundant data in the cache to reduce the read overhead on the critical write path after a collision occurs. However, the nature of collisions makes us aware that multiple different data can produce the same ECC value, which still needs to avoid the underlying device lookup process. Therefore, all similar data should be kept in the cache, which avoids the read overhead problem when different data collide. In the current implementation, FSDedup uses an asynchronous prefetch method. When similar data are compared identically, FSDedup prefetches all similar data that yield the same ECC values into the cache by an asynchronous read request operation. The key for FSDedup to set up a prefetch cache is to take full advantage of the device characteristics to enhance the parallelism of read data processing. Although we are dealing with a cache line unit of 64 B, to utilize the throughput bandwidth of NVM efficiently, we utilize the maximum access granularity (e.g., 256 B) inside NVM. Also, we can use prefetching instructions to parallelize the processing and loading of 16 data blocks with a granularity of 256 B into the NVMM read buffer or CPU cache, reading the data into the read buffer sequentially with a granularity of 256 B based on read access. the internal parallelism of NVMM allows us to take maximum advantage of the content localization feature of the data. Experimental evaluation demonstrates that a cache size of only 512 KB is required to substantially reduce the read operation overhead on critical write paths.

3.5.2 Refresh Mechanism. FSDedup utilizes the LRCU cache to achieve selective deduplication, owing to the content locality of workloads, as illustrated in Section 2.1. LRCU cache can maintain the ECC items with higher reference counts in the LRCU cache to identify and eliminate

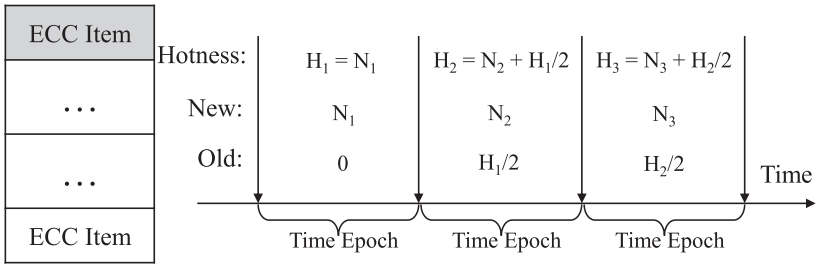


Fig. 13. The hot ECC item identification mechanism for FSDedup system. Note that the N_* denotes the reference counter of ECC items, and the H_* is defined as the hot degree during a time epoch.

ALGORITHM 3: Refresh

```
// cacheline_table - key type: string - value type: cacheline struct pointer
// frequency_table - key type: int - value type: cacheline struct pointers list
currentRefreshTimeSecond = now_time()
if ((currentRefreshTimeSecond - lastRefreshTimeSecond) >= refresh_interval)
   for each cacheline_list in frequency_table
      for each cacheline in cacheline_list
         cur_freq = cacheline_list.frequency
         last_refreshed_freq = cacheline.last_refreshed_frequency
        // The refresh mechanism.
         refreshed_freq = last_refreshed_freq/2 + (cur_freq - last_refreshed_freq)
         cacheline.last_refreshed_frequency = refreshed_freq
         if (refresh_freq < min_frequency)
            min_frequency = refresh_freq
        // Removing cacheline in this cacheline list, and insert it into another cacheline list.
         cacheline_list.remove(cacheline)
        // Due to the frequency of this cacheline has been refreshed, we need to add it to another
cacheline list.
         frequency_table.getCahelineList(cacheline.frequency).add(cacheline)
   lastRefreshTimeSecond = now_time()
```

corresponding redundant cache lines. However, if we only keep these ECC items in the cache, then this will inevitably cause "cache solidification," meaning that some cached items will never be kicked out. To take full advantage of the content locality feature and to avoid the "cache solidification" problem, therefore, we propose an LRCU refresh mechanism that dynamically identifies hot data entries and evicts the less desirable data entries with the timely refresh.

Identifying hot data items is a fundamental function of the LRCU cache, based on the content locality of data. Meanwhile, to defend against the "cache solidification" problem, dynamic hot data item identification would be an effective mechanism. Figure 13 shows the hot ECC item identification mechanism for the FSDedup system. The *New* denotes the frequency of accesses in the current time epoch. The *Old* means one-half of the last time epoch hotness value. The *hotness* represents the hotness value that can be calculated in the current time epoch. Specifically, FSDedup refreshes the LRCU cache after each time epoch, during which the hotness of a frequently accessed data entry continues to increase in line with access frequency during the current epoch. Similarly, if the cache entry is not frequently accessed, then its *Hotness* will gradually decrease and eventually be evicted from the cache. Most importantly, calculating the hotness value in the current time epoch

is determined by combining the current frequency of access to the ECC item and the hotness value in the last epoch. For instance, as illustrated in Figure 13, the value of H_3 is equal to N_3 plus half of the H_2. The value of H_3 will increase with the frequency of accesses adequate. Otherwise, the H_3 value will decrease.

Moreover, Algorithm 3 outlines the pseudo-code for the refresh() function within the LRCU cache. For example, in the refresh mechanism, the *refreshed_freq* represents the *Hotness*, calculated as the difference between *cur_freq* and *last_refreshed_freq*, plus half of *last_refreshed_freq*. To execute the refresh() function, we initially record the start time of each interval. Upon reaching the interval, the cache's refresh mechanism is activated. During the refresh() process, FSDedup refreshes all cache entries as described in Figure 13, enhancing the maintenance of new cache items to adapt better to dynamic workload changes. Consequently, the LRCU's refresh mechanism can dynamically identify more hot ECC items within the FSDedup system, facilitating the exploitation of data's content locality feature and averting the "cache solidification" issue. Furthermore, by leveraging the LRCU cache with the refresh mechanism, FSDedup achieves selective deduplication without persistent metadata to NVMM, ensuring the absence of consistency issues.

3.6 Security and Consistency Issues

Security and consistency are two critical issues in the system design process. Compared with previous ESD versions, FSDedup achieves better performance gains by leveraging cache optimization mechanisms on top of previous versions. Of course, this does not result in additional security and consistency overhead for FSDedup. Regarding the security and consistency of FSDedup, there are two essential aspects to consider.

Security Vulnerability: It is important to note that the ECC generated from plaintext data does not pose significant security vulnerabilities. As seen in previous studies [44, 53, 54], the trust base of FSDedup lies solely in the processor chip. Any data stored outside the processor chip are susceptible to external attacks. In general, ECC items are associated with plaintext cache lines as they are generated by the memory controller based on the data. Storing these ECC items could potentially reveal the original cache line, presenting a serious security vulnerability. However, it is important to note that these ECC items are only stored in the memory controller cache on the CPU-side, making them secure and impervious to manipulation by attackers [44, 56].

Data integrity and Metadata Consistency: The consistency issues of FSDedup include data consistency and metadata consistency. It is essential to prevent any potential data loss caused by deduplication-induced write eliminations, ensuring the integrity of the stored data. Typical deduplication methods may eliminate non-duplicate data blocks identified as duplicate data blocks due to the possibility of hash collisions, which can lead to data loss. However, the ECC-based fingerprints are only used for similarity identification to filter the non-duplicate cache lines. FSDedup conducts a byte-by-byte comparison for these similar cache lines to judge whether they are identical. Moreover, though FSDedup only eliminates a significant part of duplicate cache lines, the undetected duplicate cache lines cannot result in data loss or correctness issues.

Therefore, special attention must be given to persistently securely storing the critical data structures involved in FSDedup, guaranteeing their availability and reliability over time. FSDedup only keeps the fingerprints of cache lines with high reference counts in the memory controller cache, effectively avoiding consistency issues. Meanwhile, FSDedup improves the system performance with the help of a cache optimization mechanism. Nonetheless, the consistency guarantee of the cache mechanism is beyond the scope of this article. Additionally, the new Optane PMem 200 series is equipped with extended asynchronous DRAM refresh capabilities [16], which can be leveraged for persistence. Furthermore, other persistence methods, such as battery-backed write cache [4] and programming primitive enabled write back [24], are independent yet complementary to FSDedup.

Table 2. The System Configurations in the Simulators

Processor and Cache	
CPU	8 cores, X86-64 processor, 2 GHz Clock
L1 cache (private)	32 KB, 8-way with 64 B cache line, 2-cycle latency
L2 cache (private)	256 KB, 8-way with 64 B cache line, 8-cycle latency
L3 cache (shared)	16 MB, 8-way with 64 B cache line, 25-cycle latency
Cache Line Size	64 B
Main Memory (PCM) for the System	
Capacity	16 GB
PCM Latency	Read/Write: 75 ns/150 ns [5, 22]
PCM Energy	Read/Write: 1.49 nJ/6.75 nJ [10]
Metadata Cache	EFIT (512 KB), AMT (512 KB), Prefetch Cache (512 KB)

4 Performance Evaluation

In this section, we first describe the experimental setup. Then, we evaluate the performance of FSDedup in write/read speedup, energy consumption, **Instructions Per Cycle (IPC)**, and tail latency, driven by different applications compared with the typical approaches (i.e., Baseline, Dedup_SHA1, and DeWrite) and ESD. Finally, we give the sensitivity study and space overhead.

4.1 Experimental Setup

We implement and evaluate the prototype of FSDedup on gem5 [6] with NVMain 2.0 [29]. As shown in Table 2, which describes the system configuration parameters used in the experimental simulator. The system cache consists of three levels, including dedicated caches (e.g., L1 and L2) and shared L3 cache, which is a popular trend in cache hierarchical design [55, 56]. Meanwhile, the cache line evicted from the CPU core is 64 B, corresponding to the real system configuration [4, 38, 49]. The parameters of the Main Memory (PCM) for the system remain consistent with the previous studies [22]. For instance, the latency of read/write for PCM is 75 ns/150 ns, and the energy consumption for read/write operations in PCM is 1.49 nJ/6.75 nJ. The metadata cache is configured to be 512 KB for the EFIT and AMT tables. Meanwhile, the Prefetch Cache is also set at 512 KB.

For the purpose of experimental evaluation, we employed a set of 20 applications with their default settings, chosen to represent various domains in real-world scenarios. This selection includes 12 applications sourced from SPEC CPU 2017 [7] and 8 applications from PARSEC [17]. The SPEC CPU 2017 applications encompass a diverse range of types, such as integer throughput (int_rate), integer speed (int_speed), floating-point throughput (fp_rate), and floating-point speed (fp_speed). However, the PARSEC applications consist of a collection of multi-threaded benchmark tests. These 20 applications have been extensively employed in previous studies to evaluate memory performance [4, 35, 49]. Meanwhile, for effectively and comprehensively evaluating the performance of FSDedup, we also implement and evaluate the other comparative methods, e.g., the Baseline, the Dedup_SHA1, the DeWrite [56], and ESD [14].

— *Baseline*: is a general scheme without deduplication, which has normal data read and write operations for Encrypted NVMM.
— *Dedup_SHA1*: Dedup_SHA1 generates a unique fingerprint (SHA1) to detect and filter duplicate cache lines. Meanwhile, it leverages full deduplication to eliminate all redundant cache lines by storing all fingerprints in memory and the NVMM.

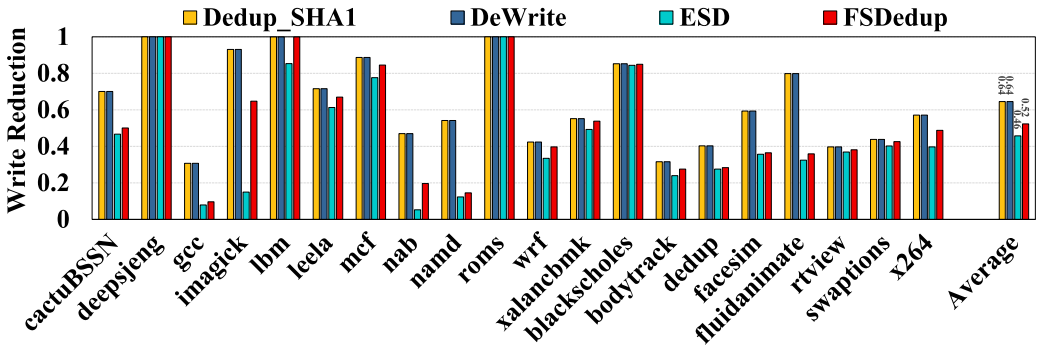


Fig. 14. Write reductions by different methods.

— *DeWrite*: DeWrite [56] generates a lightweight CRC-based fingerprint to identify similar cache lines. Based on the identified results, it needs to read the cache line from the NVMM for a further byte-by-byte comparison to determine. DeWrite also uses full deduplication to eliminate all redundant cache lines by storing all fingerprints in memory and the NVMM.
— *ESD*: ESD leverages the already existing ECC value to identify the similarity of cache lines and maintain high reference count cache lines in fingerprint cache for selective deduplication.
— *FSDedup*: FSDedup is an enhanced version compared with ESD. It proposed the cache optimization mechanism, including the prefetch cache and refresh mechanism, to improve the overall system's performance further.

For all deduplication-based methods, the NVMM system sends about 1 billion requests for warm-up during the initialization phase before each evaluation. By default, we compare all schemes to the Baseline for a comprehensive comparison of NVMM endurance, system performance, and tail latency. At last, we conduct the experimental evaluation of sensitivity analysis and space overhead.

4.2 NVMM Endurance

FSDedup is designed to enhance the NVMM's endurance by inline deduplication, reducing write traffic by identifying and eliminating writes of duplicate cache lines. As shown in Figure 14, normalized to Baseline without deduplication, we can observe that our scheme identifies and eliminates more redundant cache lines compared to ESD across all 20 applications. Meanwhile, FSDedup reduces cache line writes by 52.3% on average across all 20 applications, and by up to 99.9% for the *deepsjeng*, *lbm*, and *roms* applications. Although the deduplication ratio of FSDedup is less than the Dedup_SHA1 and DeWrite, both of which are full deduplication schemes, FSDedup also eliminates more duplicate cache lines compared to ESD.

The main reason FSDedup can enhance the NVMM endurance is that FSDedup can eliminate the redundancy of cache lines. However, as shown in Figure 14, FSDedup has approximately 10.2% undetected redundant cache lines compared with full deduplication methods, e.g., Dedup_SHA1 and DeWrite. The reason is that FSDedup is a selective deduplication, exploiting content locality by storing only ECC-based fingerprint entries with high reference counts in the cache. Meanwhile, compared to ESD, FSDedup achieves a higher deduplication ratio. The reason is that FSDedup leverages the refresh mechanism to refresh the items in the LRCU cache. Hence, FSDedup can detect and eliminate more duplicate cache lines, but may miss some with reference counts not large enough, a characteristic of selective deduplication.

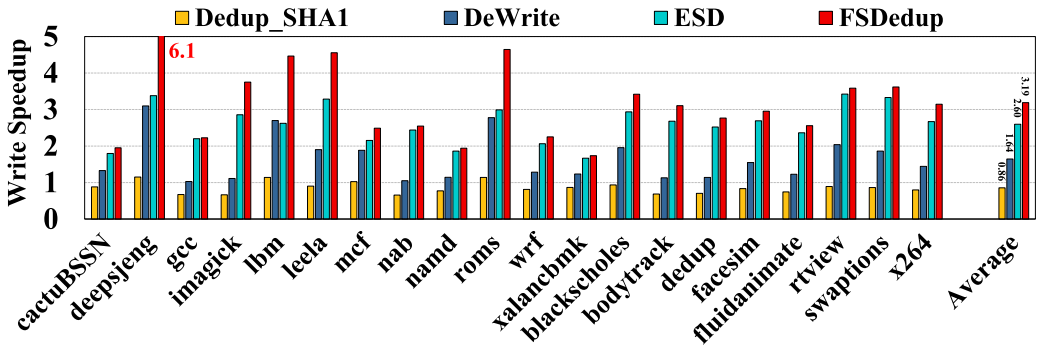


Fig. 15. Write speedup normalized to the Baseline.

4.3 System Performance

In this section, we give the experimental evaluations about write/read speedup, energy consumption, tail latency, and overall system performance. By default, the metrics used in the evaluation are represented as Baseline divided by other methods, e.g., Dedup_SHA1, DeWrite, and ESD.

Write Speedup: Figure 15 shows that the write speedup normalized to the Baseline. We can observe that FSDedup can improve the system's write performance across all 20 applications compared with Baseline. The improvement ranges by up to 6.1×, with an average speedup of 3.1×. Moreover, FSDedup can further enhance the system's write performance by up to 5.3×, 3.4×, and 1.8× for Dedup_SHA1, DeWrite, and ESD, respectively.

In contrast, Dedup_SHA1 often decreases system write performance in most applications and only shows a speedup over the Baseline scheme in a few applications, such as *deepsjeng*, *lbm*, and *roms*. Hence, the average performance of write speedup for Dedup_SHA1 lowers Baseline. Even for DeWrite, a full deduplication method with a weak hash (i.e., CRC algorithm), has limited write acceleration capability. This is mainly because of the hash computation and metadata lookup overhead in the critical path of writes.

FSDedup can provide significant improvements in write speed through three key aspects. First, FSDedup eliminates the computational latency associated with expensive hash calculations by leveraging existing ECC information on the critical write path. While the reduced write traffic may be less than that of full deduplication methods like Dedup_SHA1 and DeWrite, like ESD, the elimination of costly hash calculations greatly enhances performance.

Second, FSDedup incorporates a refresh mechanism for the LRCU cache, to adapt the dynamic workload changes so that more redundant cache lines can be detected. This mechanism can further enhance the contribution to improving the write performance compared with the typical Dedup_SHA1, the latest DeWrite, and ESD.

Third, FSDedup adopts the LRCU cache to store a portion of the fingerprint metadata items, like ESD, thereby avoiding the overhead of fingerprint NVMM_lookup seen in schemes like Dedup_SHA1 and DeWrite. By checking only the ECC-based fingerprints in memory, the fingerprint lookup process is significantly accelerated on the critical write path. Moreover, FSDedup uses a prefetch mechanism for further corresponding the similar cache lines to reduce the read overhead from the similarity comparison.

Overall, FSDedup offers superior write performance by addressing computational latency, optimizing fingerprint lookup, and efficiently identifying redundant data.

Read Speedup: Figure 16 demonstrates that FSDedup apparently improves performance across all 20 applications compared to Baseline. Specifically, FSDedup enhances read performance by

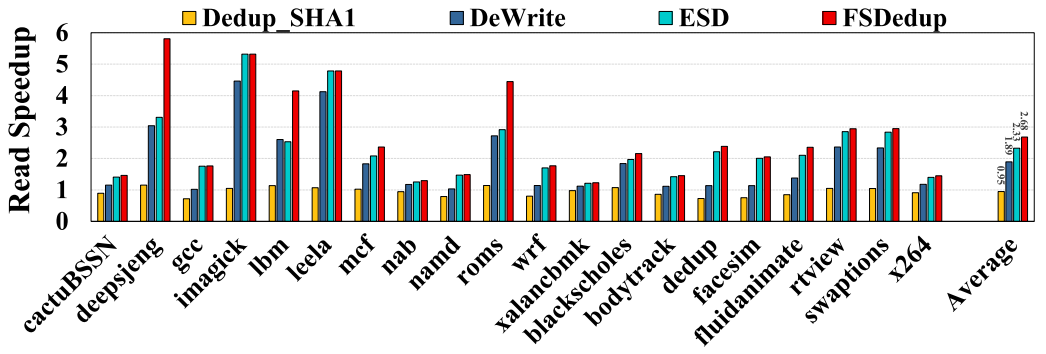


Fig. 16. Read speedup normalized to the Baseline.

up to 5.8× for *deepsjeng*, 5.3× for *deepsjeng*, 2.1× for *dedup*, and 1.8× for *deepsjeng* applications, compared to Baseline, Dedup_SHA1, DeWrite, and ESD, respectively. Additionally, we also can see that FSDedup improves the performance of read with 2.8×, 1.4×, and 1.2× on average compared with Dedup_SHA1, DeWrite, and ESD.

While Dedup_SHA1 can eliminate redundant data, it only improves read performance in a few applications. However, FSDedup also can degrade the read performance on average due to the heavy computation and loolup overhead. Correspondingly, the read performance of DeWrite is superior to that of Dedup_SHA1 due to the use of the lightweight CRC algorithm for computing the fingerprint. However, the read performance improvement is still limited because of the overhead of lookup for metadata and read overhead from similarity data comparison. Moreover, the FSDedup outperforms Dedup_SHA1, DeWrite, and ESD in terms of read performance, demonstrating its effectiveness in improving system performance.

FSDedup can speed up read operations by reducing duplicate cache lines during the write path, optimizing resource utilization of the I/O bus and memory banks. This enables more efficient handling of read requests. Moreover, FSDedup can further eliminate these duplicate cache lines through its refresh mechanism and prefetch cache mechanism. Consequently, this minimizes read/write interference and reduces waiting times for read operations in the queue.

Energy Consumption: Energy consumption is a vital performance metric for memory system efficiency, encompassing read/write energy, encrypted energy, and deduplication-induced computing energy, similar to the energy consumption model for DeWrite and SHA1 [39]. FSDedup directly contributes to reducing the energy consumption of the NVMM's system by leveraging inline deduplication to eliminate duplicate writes, like ESD.

Figure 17 shows the energy consumption results normalized to Baseline. It demonstrates that FSDedup achieves an energy consumption reduction of 83.6% on average across all 20 applications compared with Baseline. Moreover, FSDedup exhibits a decrease in energy consumption compared with the Dedup_SHA1, Dewrite, and ESD across all 20 applications, which further highlights the effectiveness of FSDedup in minimizing energy consumption in the NVMM system.

Note that Dedup_SHA1 can eliminate more redundant data. However, the performance improvement is only evident for a few applications, such as *gcc*, *nab*, *namd*, *wrf*, and *xalancbmk*. In contrast, most applications perform worse than Baseline. DeWrite, which employs a weak hash, also has limited energy improvement for the system. In the worst-case scenario, DeWrite consumes more energy than Baseline. The main reason is that all the data hash computation and metadata lookups are in the critical write path, resulting in a significant energy consumption for the NVMM system.

FSDedup outperforms the DeWrite, Dedup_SHA1, and ESD in reducing energy consumption. First, like ESD, FSDedup leverages the existing ECC information to eliminate computational energy

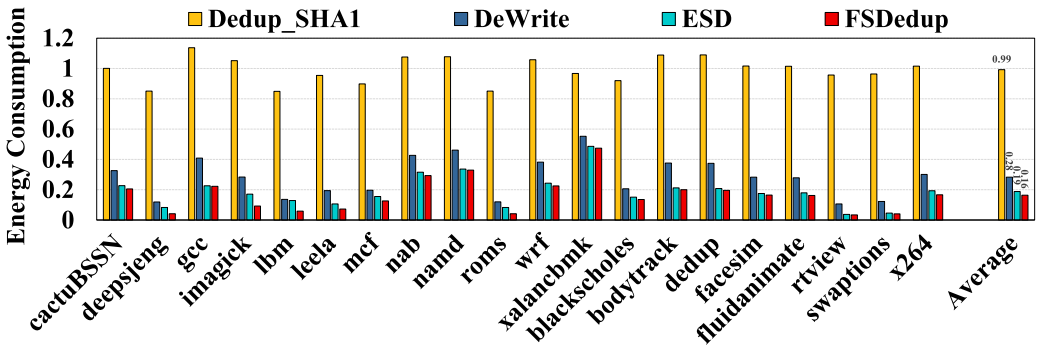


Fig. 17. The energy consumption results normalized to the Baseline.

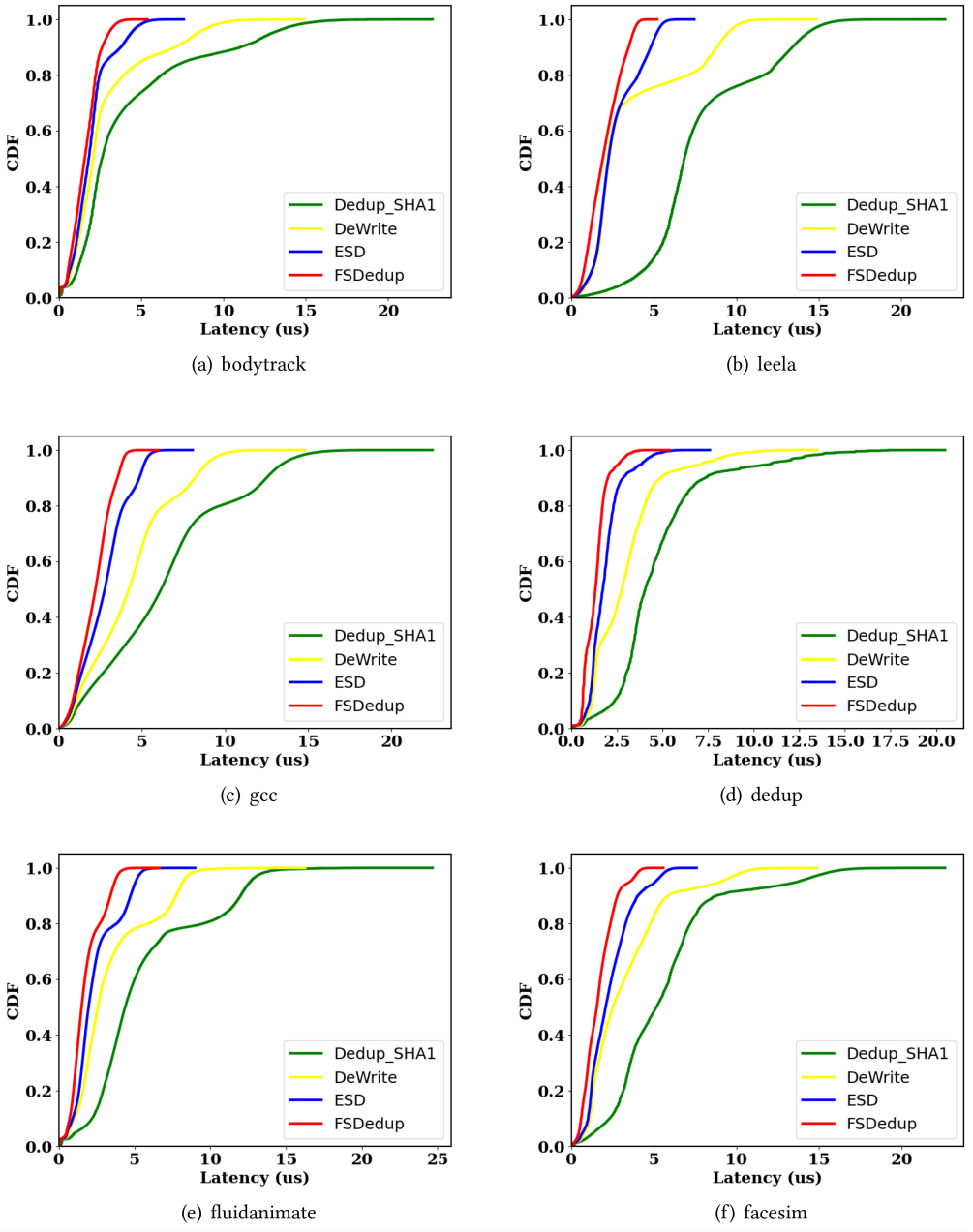
consumption in the deduplication logic. In contrast, Dedup_SHA1 and DeWrite require significant energy consumption for calculating SHA1 and CRC fingerprints by CPU cores. Second, FSDedup exploits the selective deduplication method to avoid NVMM-based fingerprint accesses, similar to ESD, reducing energy consumption. In contrast, the full deduplication method (a.k., Dedup_SHA1, and DeWrite) incurs many extra NVMM accesses to search specific fingerprints, leading to significant energy consumption. Third, FSDedup utilizes cache optimization mechanisms to reduce energy consumption further. The refresh mechanism can detect more duplicate data, thereby reducing data writes. The prefetch cache mechanism reduces read operations overhead from similar data comparisons during the critical write path. As a result, FSDedup avoids both computation and fingerprint-induced energy consumption in deduplication-based NVMM systems.

Tail Latency: To further demonstrate the advantages of FSDedup, we conducted tests on tail latency, which reflects the **Quality of Service (QoS)** of a storage system [52]. This is especially important for memory systems being gradually replaced by NVMM, especially in latency-sensitive applications. Our investigation into the latency distributions of the Dedup_SHA1, DeWrite, ESD, and FSDedup across 20 applications revealed that the FSDedup significantly outperforms others in tail latency.

Figure 18 illustrates the CDF of write latency across different applications under various methods. As we mentioned above, these applications demonstrate the effectiveness of FSDedup in reducing tail latencies and improving QoS for NVMM-based storage systems. Moreover, FSDedup can significantly reduce tail latency through several fundamental mechanisms.

First, FSDedup eliminates the computational latency caused by expensive hash calculations. This relieves the critical write path from the burden of hash calculation latency, resulting in improved performance. Second, FSDedup utilizes a selective deduplication approach assisted by ECC. This approach effectively reduces latency for both read and write operations by eliminating a majority of duplicate writes. FSDedup minimizes latency and enhances overall system efficiency by avoiding unnecessary data accesses. The reasons analyzed above are also reflected in ESD.

Additionally, compared with ESD, FSDedup incorporates the cache optimization mechanism consisting of a prefetch cache mechanism and a refresh mechanism to reduce the latency overhead further and detect more redundant data. The refresh mechanism can effectively update the fingerprint items in the LRCU cache to adapt to the changed workloads. Hence, FSDedup can detect more redundant data by the refresh mechanism, reducing the overhead of the write path. Moreover, when judged similarly, the prefetch mechanism can take the *valuable* fingerprint items to the prefetch cache to quickly detect duplicate data. Therefore, FSDedup can decrease the overhead of read operation during the similar data comparison process. As a result, the corresponding cascade operation latency is reduced.



(a) bodytrack (b) leela

(c) gcc (d) dedup

(e) fluidanimate (f) facesim

Fig. 18. The CDF of write latency of different applications, such as *bodytrack*, *leela*, *gcc*, *dedup*, *fluidanimate*, and *facesim*

Furthermore, there is no doubt that the latency performance of FSDedup is better than the Dedup_SHA1 and DeWrite. However, FSDedup may reduce slightly fewer duplicate cache lines compared to other methods. The reason is that both can introduce additional costly operations, such as hash calculations and fingerprint NVMM_lookup operations, on the critical write path.

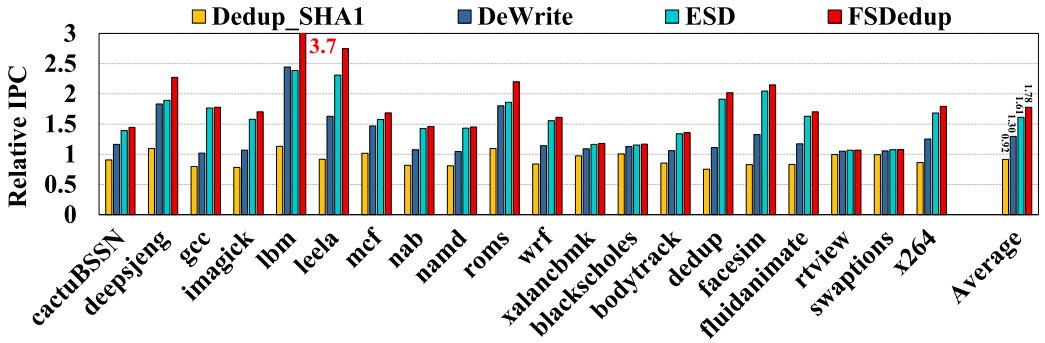


Fig. 19. IPC improvements normalized to the Baseline.

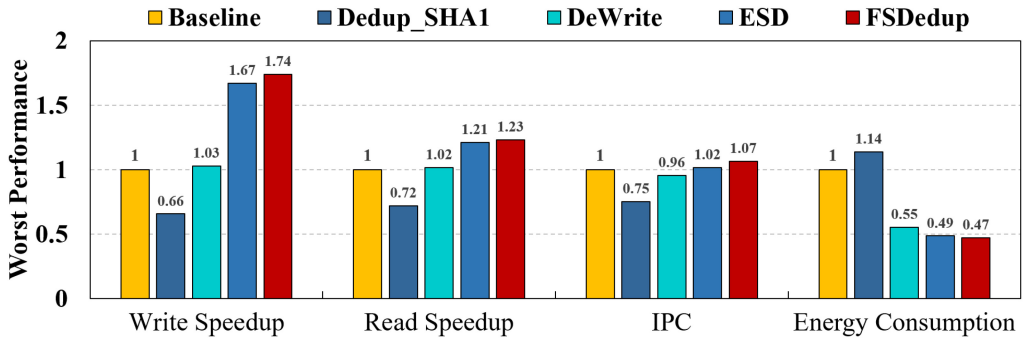
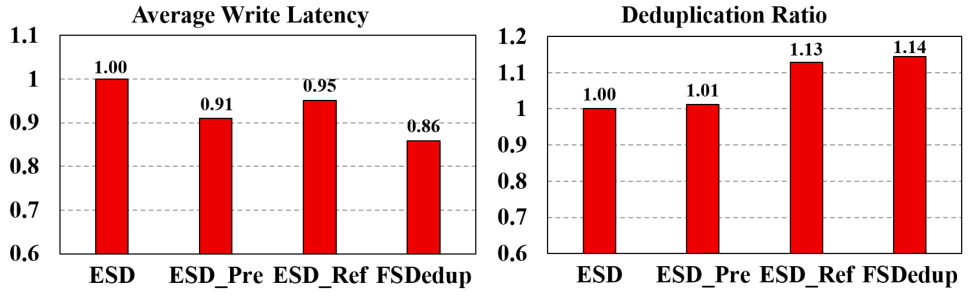


Fig. 20. The worst performance of various schemes normalized to Baseline, such as write speedup, read speedup, IPC, and Energy consumption.

Overall, FSDedup can remarkably reduce tail latency by eliminating the hash calculation latency, utilizing selective deduplication, and avoiding the additional overhead from read operations due to similarity comparison on the critical write path. Therefore, combining these mechanisms sets FSDedup apart from other methods, making it a superior choice for minimizing latency in storage systems.

The Overall System Performance: FSDedup demonstrates an increase in IPC, depicted in Figure 19, a crucial metric for evaluating overall system performance. FSDedup can improve overall system performance in two aspects. First, FSDedup can reduce system writes through selective deduplication, thus enhancing the system's ability to process data. This is the main reason why the performance of FSDedup is better than other traditional methods, such as Dedup_SHA1 and DeWrite. Meanwhile, FSDedup adopts a cache optimization mechanism. It can better adapt to workload changes through the refreshing mechanism, eliminating more redundant data. Additionally, it reduces read access to the underlying devices through the prefetching mechanism in case of similar data collisions.

Performance in Worst-case Scenarios: Generally, we enhance system performance by eliminating substantial amounts of redundant data. However, the existing deduplication methods may not positively impact the system performance but instead may degrade the system performance when there is not so much redundant data or in extreme scenarios where the data are extremely scattered. To evaluate the performance of the various methods in specific situations, we conducted a benchmark test using content with relatively decentralized characteristics and a relatively low duplicate rate. Figure 20 shows the performance of various methods in terms of write speedup, read speedup, IPC, and energy consumption.



(a) Average Write Latency (b) Deduplication Ratio

Fig. 21. The performance comparison of different schemes, i.e., ESD, ESD_Pre, ESD_Ref, and FSDedup.

We can see that Dedup_SHA1 leads to system performance degradation no matter which evaluation metrics are used. In the worst-case scenarios, read/write performance can be decreased by up to 28% and 34%. This is because the deduplication technique imposes a certain performance overhead on the write critical path, bringing some additional overhead to the system. DeWrite utilizes a lightweight hash to reduce the computational overhead during the write process. However, the performance improvement of the system is still limited. In the worst-case scenarios, read/write performance only can be increased by 2% and 3%. In addition, even in the worst case, ESD can still improve the system's performance, attributed to its ECC identification mechanism and selective deduplication mechanism. However, FSDedup can improve the system performance even further compared with ESD. By considering specific features and cache optimization mechanisms, FSDedup outperforms other methods, making it a valuable choice for enhancing system performance in storage systems. Even for the worst-case scenarios, the IPC can be increased by 7%.

4.4 Other Analysis

In this section, we perform two main experiments. The first experiment compares ESD, ESD_Pre, ESD_Ref, and FSDedup under different scenario settings. The second experiment tests how well the various methods perform in real-world *key-value* scenarios.

Experimental Evaluations under four setups: To further analyze the performance benefits of FSDedup, we further evaluate four different setups: ESD, ESD_Pre, ESD_Ref, and FSDedup. Figure 21 shows the comparison of different schemes. Compared with ESD, ESD_Pre has the Prefetch mechanism. The purpose is to reduce the read latency cost due to similarity comparison on the write path, which is reflected by the average latency of the write process. Figure 21(a) shows the average write latency under different experimental scheme Settings. It can be seen that compared with ESD, ESD_Pre can further reduce the write latency of the system, because the prefetch mechanism can reduce the read operation latency caused by similar data comparisons on the write path. At the same time, we can see that the write latency of ESD_Ref is slightly higher than that of ESD_Pre, only 4%, but it is still lower than that of ESD. The reason is that the refresh mechanism also introduces some overhead while it can enhance the deduplication ratio. Meanwhile, similar data also needs to be read out from NVMM for byte-by-byte comparison.

Compared with ESD, the Refresh mechanism is added to ESD_Ref. This mechanism mainly adds a refresh mechanism to the original cache mechanism (LRCU), illustrated in Section 3.5.2. The purpose is to refresh the entries in the cache in a timely manner to adapt to workload changes and increase the deduplication rate. The deduplication ratio is expressed as the proportion of the data reduced by deduplication to the total data size. Figure 21(b) shows the proportion of

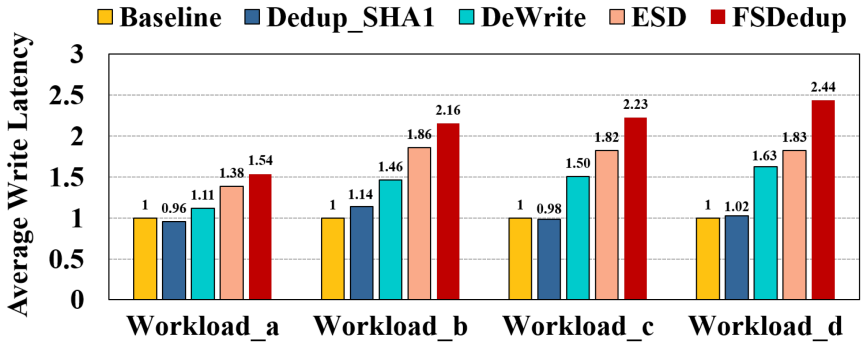


Fig. 22. The average write latency of different schemes under four workloads of YCSB. Note that *Workload_a* denotes the first workload, and the rest can correspond similarly.

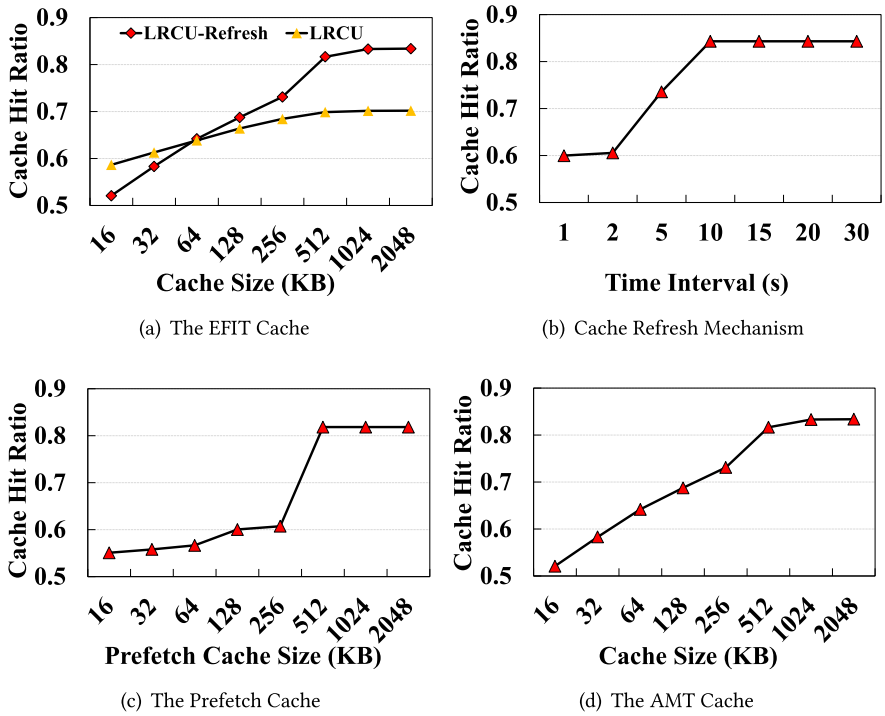
the Deduplication ratio under different experimental Settings. We can observe that the ESD_Ref has better deduplication performance than ESD. Meanwhile, ESD_Pre can also achieve a slight performance improvement. This is likely due to our proposed prefetch mechanism, which can fetch similar items into the prefetch cache further to improve the accuracy in the similarity comparison process. Secondly, we can see that the deduplication ratio of FSDedup is also higher than that of the ESD_Ref experimental scheme with only a refresh mechanism, which further confirms our speculation just now.

YCSB Evaluation: To satisfy the real experimental evaluation scenarios as much as possible, we also include the workloads test of *Key-value* store. Typically, redundant data can be eliminated by performing FSDedup on the system write critical path, which is the core idea of FSDedup. Therefore, we use the YCSB [12] benchmark to generate five workloads with insert operations, which are (a) write-read (25% insert and 75% read), (b) write-read (50% insert and 50% read), (c) write-read (75% insert and 25% read), and (d) write-only. Then, we leverage Memcached [1] to run these workloads based on our prototype system. Figure 22 shows the performance of different schemes under four different workloads of YSCB.

As we can see from Figure 22, from *workload_a* to *workload_d*, as the percentage of INSERT operations increases, the average write performance of the multiple schemes with deduplication also improves compared with Baseline. This is because as the amount of data written increases, the various deduplication schemes play a stronger role. Meanwhile, we can see that despite the addition of the deduplication technique, Dedup_SHA1 does not bring enough performance improvement to the system. Instead, it may lead to system performance degradation, because it has a massive overhead of hash computation. The maximum increase of the DeWrite solution is 1.6× compared with Baseline. This is because, although the CRC algorithm can reduce the latency on the write path, there is also a certain percentage of read operation overhead in the similarity comparison process. The average write latency of ESD is improved by up to 1.86× compared to the Baseline, which is higher than the DeWrite and Dedup_SHA1. However, FSDedup can improve up to 2.44× and is higher than ESD because of the cache optimization mechanism compared to ESD. Moreover, FSDedup can utilize the cache refresh mechanism to dynamically adjust the cache items to adapt to the workload changes and identify more redundant data. Finally, FSDedup utilizes the prefetching mechanism to reduce the access overhead of the underlying data during similar data detection.

4.5 Sensitivity Analysis and Space Overhead

As a selective deduplication, the performance of the EFIT cache in FSDedup directly affects the overall system performance. Therefore, compared with ESD, FSDedup proposed the cache refresh



(a) The EFIT Cache

(b) Cache Refresh Mechanism

(c) The Prefetch Cache

(d) The AMT Cache

Fig. 23. The sensitivity analysis of cache hit ratio under different cache sizes and time intervals. Panels (a), (c), and (d) show that the cache hit ratio changes with the increase in cache size. Panel (b) shows that the hit ratio of LRCU changes under different time intervals.

mechanism further, described in Section 3.5.2. Figure 23(a) illustrates the performance of the basic LRCU cache and the LRCU cache with the Refresh mechanism under different cache sizes in FSDedup. As expected, the hit ratio of both caches increases as the cache size grows. However, due to its limited cache space, the initial LRCU-Refresh cache has a lower hit rate than the basic LRCU cache. This is because the refresh mechanism may evict hot data entries from the cache. Nevertheless, we observe that as the cache space expands, the LRCU cache with the refresh mechanism apparently outperforms the basic LRCU cache.

Moreover, beyond a cache size of 512 KB, the hit rate of LRCU-Refresh shows no substantial increase. For instance, the hit rate only increases by 0.1% from 512 KB to 1024 KB. These findings suggest that consistently increasing the cache size beyond a certain point does not significantly contribute to the detection and elimination of duplicate cache lines. As a result, we have set the cache size for LRCU-Refresh as 512 KB. These results demonstrate the effectiveness of the Refresh mechanism in improving cache performance and overall system efficiency. By optimizing the cache size and incorporating the Refresh mechanism, FSDedup can detect and eliminate duplicate cache lines more efficiently, enhancing system performance and providing users with a more efficient storage solution.

Furthermore, the refresh interval is an essential factor that influences cache performance. To evaluate its impact, we conducted a sensitivity analysis of cache refresh, as depicted in Figure 23(b). The results indicate that the performance of LRCU-Refresh is optimized when the refresh interval is set to 10 s.

Like the ESD and DeWrite, the process of identifying redundant data needs extra read operations from the NVMM to compare by byte by byte further, which introduces additional overhead to the

system, as discussed in Section 3.5.1. To mitigate this performance impact, FSDedup designed the prefetch cache mechanism. Compared with ESD, FSDedup demonstrates a significant increase in write acceleration. This improvement is achieved by eliminating many read operations on the write critical path.

However, the prefetch cache's performance is affected by size variations, mainly due to expensive on-chip caches. The cache hit rate continues to increase as the prefetch cache size increases, as depicted in Figure 23(c). As the cache size exceeds 512 KB, the cache hit rate continues to increase by a negligible percentage. Similarly, Figure 23(d) shows that the cache hit rate increases steadily with the gradual increase in prefetch cache size. However, the growth rate decreases significantly when the cache size exceeds 512 KB. For example, the cache hit rate increases by only 0.5% from 512 bytes to 1024 KB and 0.6% from 1024 to 2048 KB. These findings highlight the importance of optimizing the cache size of the prefetch cache in FSDedup. By carefully choosing the right cache size, system performance can be significantly improved, thus ensuring efficient read operations and increasing the efficiency of the entire storage system.

The fingerprint metadata store can impose a significant NVMM space overhead in Dedup_SHA1 and DeWrite, which both utilize full deduplication methods. In DeWrite, each physical cache line requires (16 bytes + 3 bits) for maintenance, resulting in a metadata space overhead of approximately 25.59%. Dedup_SHA1, with its 160-bit SHA1 fingerprints, incurs an even larger metadata space overhead than DeWrite.

In contrast, FSDedup takes a different approach by selectively storing fingerprints with high reference counts in the memory cache for deduplication, while maintaining the AMT table in the NVMM. This eliminates the NVMM space overhead caused by the fingerprint store. Additionally, in the current implementation, the refresh mechanism for the EFIT table has been introduced to detect more duplicate cache lines, without requiring any extra space overhead.

Furthermore, to further reduce the overhead of read operations due to byte-by-byte data matching, we have incorporated the prefetch cache mechanism, which only requires 512 KB of space. The experiments were conducted to validate that FSDedup significantly outperforms the Dedup_SHA1 and DeWrite.

In summary, FSDedup effectively addresses the NVMM space overhead issue associated with fingerprint metadata storage. By employing selective deduplication and efficient cache mechanisms like the refresh mechanism and the prefetch mechanism, FSDedup achieves superior performance compared to alternative schemes.

5 Related Work

Previous studies either apply multiple threads to multicore processors [23, 43] or integrate the data deduplication process into the GPU/co-processor hardware architecture to accelerate the compute-intensive hash calculations [18, 33].

Fingerprints Calculating Acceleration: Calculating hashes for deduplication can be time-consuming and taxing system resources, especially for NVMs with ultra-low latency. To address this challenge, existing studies have explored techniques such as sampling or lightweight pre-hashing to filter out non-duplicate data. For instance, CAFTL [9] utilizes samples to filter non-duplicate pages for flash storage, while EaD [42] leverages existing ECC functions to optimize write operation to SSD. Moreover, DeWrite [56] uses CRC to detect data similarity. Based on EaD's and DeWrite's insights, ESD also taps into existing ECC information within the LLC to identify similar cache lines, thereby reducing computational overhead. by using existing ECC information to eliminate hash computation overhead, ESD can also avoid the metadata storage overhead issue, albeit at the cost of handling a small portion of redundant data. In comparison, FSDedup, like ESD, utilizes existing ECC information to replace hash computation. Still, it proposes a prefetch

cache mechanism to reduce metadata access when hash collisions occur, ultimately enhancing system performance.

Selective Deduplication: Selective deduplication is not new and has been studied in the deduplication for primary storage systems, such as iDedup [34] and POD [25]. However, both iDedup and POD are designed for HDD-based primary storage systems with high disk seek overhead. Unlike iDedup and POD, ESD is designed for NVMM, which does not have seek overhead. NVMs are promising candidates to replace or co-exist with DRAM, acting as a scalable memory solution where space efficiency is critically important. Deduplication-induced memory overhead is significant for NVMs evaluated in the DeWrite study [56]. To address the problem, ESD performs selective deduplication by only storing ECC values with high reference counts in the cache. This can eliminate a major part of duplicate cache lines, though not all of them, as in DeWrite. As a result, the selective deduplication method in ESD differs significantly from that in iDedup and POD, which aims to reduce seek overhead for HDD-based primary storage systems. The crucial thing about selective deduplication is how to implement the selectivity. ESD proposed LRCU caching to maintain high-reference metadata entries in the cache. However, such an approach makes the cache solidified and not adaptable to immediate changes in load. Therefore, FSDedup proposes a refresh mechanism to make the LRCU caching policy dynamic and able to update the cache entries in time to keep them fresh.

Deduplication for NVMs: Recent studies, as demonstrated by NV-Dedup [36], DeWrite [56], and BCD deduplication [27], underscore the effectiveness of inline deduplication in minimizing write traffic to NVMs. NV-Dedup excels in data deduplication and performance in NVM-oriented file systems. At the same time, DeWrite leverages NVM's intrinsic read/write asymmetry and a lightweight CRC hashing algorithm for reducing duplicate cache lines. BCD deduplication follows a similar path with inter-block diff compression and deduplication for enhancing main memory capacity. However, these approaches, including DeWrite and BCD [27], predominantly pursue complete deduplication, aiming to eliminate all duplicate cache lines or redundant data blocks in NVMs. Consequently, computational and memory space overheads persist, prompting varied designs to address these challenges.

In contrast, the design of ESD mitigates computational overhead by utilizing ECC values within LLC cache rows to identify data similarity. Simultaneously, selective deduplication stores the fingerprints with higher reference counts in memory to exploit content locality. Experimental results highlight the importance of cache performance in selective deduplication. Therefore, FSDedup optimizes cache performance using prefetch cache and refresh mechanisms, effectively reducing read operation overhead during data comparison and detecting more redundant data. FSDedup demonstrates superior efficiency in eliminating duplicate cache lines with reduced space overhead compared to alternative schemes.

6 Conclusion

After conducting a workload analysis, it became evident that existing deduplication-based studies were impacting the performance of NVMM systems. ESD introduced ECC-assisted selective deduplication to reduce writes to the NVMM device and enhance overall system performance. However, initial experiments revealed that cache performance was a limiting factor, primarily due to reduced redundancy in writes and the modest improvements in read/write operations and energy consumption. As a result, we proposed FSDedup, which incorporates a cache refresh mechanism to identify more redundant data and a prefetch cache mechanism to mitigate the read overhead caused by similar data collisions, respectively, compared to ESD. Our experimental evaluation demonstrates that FSDedup significantly outperforms existing methods regarding read and write performance, energy efficiency, and tail latency.

References

[1] 2023. Memcached. Retrieved from https://memcached.org/

[2] Irina Alam, Clayton Schoeny, Lara Dolecek, and Puneet Gupta. 2018. Parity++: Lightweight error correction for last level caches. In *Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSNW'18)*.

[3] AMD EPYC Processors. 2021. Retrieved from https://www.amd.com/zh-hans/processors/epyc-server-cpu-family

[4] Amro Awad, Pratyusa Manadhata, Stuart Haber, Yan Solihin, and William Horne. 2016. Silent shredder: Zero-cost shredding for secure non-volatile main memory controllers. In *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'16)*.

[5] Amro Awad, Mao Ye, Yan Solihin, Laurent Njilla, and Kazi Zubair. 2019. Triad-nvm: Persistency for integrity-protected and encrypted non-volatile memories. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA'19)*.

[6] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, D. Hill, and A Wood. 2011. The Gem5 simulator. *ACM SIGARCH Comput. Arch. News* 39, 2 (May 2011), 1–7.

[7] James Bucek, Klaus Lange, and Jakim Kistowski. 2018. SPEC CPU2017: Next-generation compute benchmark. In *Proceedings of the 9th ACM/SPEC International Conference on Performance Engineering (ICPE'18)*.

[8] Ricardo Chaves, Leonel Sousa, Nicolas Sklavos, Apostolos Fournaris, Georgina Kalogeridou, Paris Kitsos, and Farhana Sheikh. 2016. Secure hashing: SHA-1, SHA-2, and SHA-3. In *Circuits and Systems for Security and Privacy*, 105–132.

[9] Feng Chen, Tian Luo, and Xiaodong Zhang. 2011. CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies (FAST'11)*.

[10] Zhengguo Chen, Youtao Zhang, and Nong Xiao. 2020. ExtraCC: Improving performance of secure NVM with extra counters and ECC. In *Proceedings of the 36th International Conference on Massive Storage Systems and Technology (MSST'20)*.

[11] Sangyeun Cho and Hyunjin Lee. 2009. Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'09)*.

[12] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SOCC'10)*.

[13] Guzman De, B Larry, Ariel Sison, and Ruji Medina. 2018. MD5 secured cryptographic hash value. In *Proceedings of the 2018 International Conference on Machine Learning and Machine Intelligence (MLMI'18)*.

[14] Chunfeng Du, Suzhen Wu, Jiapeng Wu, Bo Mao, and Shengzhe Wang. 2023. ESD: An ECC-assisted and selective deduplication for encrypted non-volatile main memory. In *Proceedings of the 29th IEEE International Symposium on High-Performance Computer Architecture (HPCA'23)*.

[15] ECC for L2 Cache Data Memory. 2022. Retrieved from https://www.intel.com/content/www/us/en/docs/programmable/683360/18-0/ecc-for-l2-cache-data-memory.html

[16] Extended Asynchronous DRAM Refresh (eADR). 2021. Retrieved from https://www.intel.cn/content/www/cn/zh/products/docs/memory-storage/optane-persistent-memory/optane-persistent-memory-200-series-brief.html

[17] Mark Gebhart, Joel Hestness, Ehsan Fatehi, Paul Gratz, and Stephen Keckler. 2009. *Running PARSEC 2.1 on M5*. Technical Report TR-09-32, The University of Texas at Austin, Department of Computer Science.

[18] Roksana Hossain. 2019. *Accelerating Sequence Calculations on Parallel GPU Architecture*. Unpublished Ph.D. Dissertation, University of Calgary.

[19] S. Ikeda, K. Miura, H. Yamamoto, K. Mizunuma, H. D. Gan, M. Endo, S. Kanai, J. Hayakawa, F. Matsukura, and H. Ohno. 2010. A perpendicular-anisotropy CoFeB–MgO magnetic tunnel junction. *Nat. Mater.* 9, 9 (Jul. 2010), 721–724.

[20] Intel Xeon Processor E5 Family. 2020. Retrieved from https://www.intel.com/content/www/us/en/processors/xeon/xeon-e5-family-spec-update.html

[21] Luc Jaulmes, Miquel Moreto, Mateo Valero, and Marc Casas. 2019. A vulnerability factor for ECC-protected memory. In *Proceedings of the 25th International Symposium on On-Line Testing and Robust System Design (IOLTS'19)*.

[22] Benjamin Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*.

[23] Sheng Li, Ho Ahn, Richard Strong, Jay Brockman, Dean Tullsen, and Norman Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'09)*.

[24] Sihang Liu, Aasheesh Kolli, Jinglei Ren, and Samira Khan. 2018. Crash consistency in encrypted non-volatile main memory systems. In *Proceedings of the 24th IEEE International Symposium on High Performance Computer Architecture (HPCA'18)*.

[25] Bo Mao, Hong Jiang, Suzhen Wu, and Lei Tian. 2014. POD: Performance oriented I/O deduplication for primary storage systems in the cloud. In *Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium (IPDPS'14)*.

[26] Prashant Nair, DaeHyun Kim, and Moinuddin Qureshi. 2013. ArchShield: Architectural framework for assisting DRAM scaling by tolerating high error rates. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13)*.

[27] Sungbo Park, Ingab Kang, Yaebin Moon, Jung Ho Ahn, and G Suh. 2021. BCD deduplication: Effective memory compression using partial cache-line deduplication. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'21)*.

[28] Minesh Patel, Jeremie Kim, Taha Shahroodi, Hasan Hassan, and Onur Mutlu. 2020. Bit-exact ecc recovery (BEER): Determining DRAM on-die ECC functions by exploiting DRAM data retention characteristics. In *Proceedings of the 53rd International Symposium on Microarchitecture (MICRO'20)*.

[29] Matthew Poremba, Tao Zhang, and Yuan Xie. 2015. NVMain 2.0: A user-friendly memory simulator to model non-volatile memory systems. *IEEE Comput. Arch. Lett.* 14, 2 (Feb. 2015), 140–143.

[30] Jiansheng Qiu, Yanqi Pan, Wen Xia, Xiaojia Huang, Wenjun Wu, Xiangyu Zou, Shiyi Li, and Yu Hua. 2023. Light-dedup: A light-weight inline deduplication framework for non-volatile memory file systems. In *Proceedings of the 2023 USENIX Annual Technical Conference (ATC'23)*.

[31] Qualcomm Centriq 2400 Processor. 2017. Retrieved from https://www.qualcomm.com/news/onq/2017/10/05/qualcomm-centriq-2400-designed-scalability-and-throughput-performance

[32] Gururaj Saileshwar, Prashant Nair, Prakash Ramrakhyani, Wendy Elsasser, and Moinuddin Qureshi. 2018. Synergy: Rethinking secure-memory design for error-correcting memories. In *Proceedings of the 24th IEEE International Symposium on High Performance Computer Architecture (HPCA'18)*.

[33] Karan Shetti and Rajendra. 2014. *Optimization and Scheduling of Applications in A Heterogeneous CPU-GPU Environment*. Ph.D. Dissertation. Nanyang Technological University.

[34] Kiran Srinivasan, Timothy Bisson, Garth Goodson, and Kaladhar Voruganti. 2012. iDedup: Latency-aware, inline data deduplication for primary storage. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*.

[35] Shivam Swami, Joydeep Rakshit, and Kartik Mohanram. 2016. SECRET: Smartly encrypted energy efficient non-volatile memories. In *Proceedings of the 53rd Design Automation Conference (DAC'16)*.

[36] Chundong Wang, Qingsong Wei, Jun Yang, Cheng Chen, Yechao Yang, and Mingdi Xue. 2018. NV-dedup: High-performance inline deduplication for non-volatile memory. *IEEE Trans. Comput.* 67, 5 (May 2018), 658–671.

[37] Rujia Wang, Lei Jiang, Youtao Zhang, and Jun Yang. 2015. SD-PCM: Constructing reliable super dense phase change memory under write disturbance. In *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'15)*.

[38] Zixuan Wang, Xiao Liu, Jian Yang, Theodore Michailidis, Steven Swanson, and Jishen Zhao. 2020. Characterizing and modeling non-volatile memory systems. In *Proceedings of the 53rd International Symposium on Microarchitecture (MICRO'20)*.

[39] Benedikt Westermann, Danilo Gligoroski, and Svein Knapskog. 2010. Comparison of the power consumption of the 2nd round SHA-3 candidates. In *Proceedings of the International Conference on ICT Innovations (ICCS'10)*.

[40] Philip Wong, Heng Lee, Yu Shimeng, Chen YuSheng, Wu Yi, Chen Pang-Shiu, Lee Byoungil, Chen Frederick, and Tsai Ming-Jinn. 2012. Metal–oxide RRAM. *Proc. IEEE* 100, 6 (May 2012), 1951–1970.

[41] Suzhen Wu, Jiapeng Wu, Zhirong Shen, Zhihao Zhang, Zuocheng Wang, and Bo Mao. 2021. SimiEncode: A similarity-based encoding scheme to improve performance and lifetime of non-volatile main memory. In *Proceedings of the 39th IEEE International Conference on Computer Design (ICCD'21)*.

[42] Suzhen Wu, Jindong Zhou, Weidong Zhu, Hong Jiang, Zhijie Huang, Zhirong Shen, and Bo Mao. 2020. EaD: A collision-free and high performance deduplication scheme for flash storage systems. In *Proceedings of the IEEE 38th International Conference on Computer Design (ICCD'20)*.

[43] Wen Xia, Hong Jiang, Dan Feng, Lei Tian, Min Fu, and Zhongtao Wang. 2012. P-dedupe: Exploiting parallelism in data deduplication system. In *Proceedings of the IEEE 7th International Conference on Networking, Architecture, and Storage (NAS'12)*.

[44] Fan Yang, Youyou Lu, Youmin Chen, Haiyu Mao, and Jiwu Shu. 2019. No compromises: Secure NVM with crash consistency, write-efficiency and high-performance. In *Proceedings of the 56th ACM/IEEE Design Automation Conference (DAC'19)*.

[45] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steve Swanson. 2020. An empirical guide to the behavior and use of scalable persistent memory. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20)*.

[46] Zuoru Yang, Jingwei Li, and Patrick PC Lee. 2022. Secure and lightweight deduplicated storage via shielded deduplication-before-encryption. In *Proceedings of the USENIX Annual Technical Conference (ATC'22)*. 37–52.

[47] Mao Ye, Clayton Hughes, and Amro Awad. 2018. Osiris: A low-cost mechanism to enable restoration of secure non-volatile memories. In *Proceedings of the 51st International Symposium on Microarchitecture (MICRO'18)*.

[48] Doe Hyun Yoon and Mattan Erez. 2009. Memory mapped ECC: Low-cost error protection for last level caches. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*.

[49] Vinson Young, Prashant Nair, and Moinuddin Qureshi. 2015. DEUCE: Write-efficient encryption for non-volatile memories. *ACM SIGARCH Comput. Arch. News* 43, 1 (Mar. 2015), 33–44.

[50] Jianhui Yue and Yifeng Zhu. 2013. Accelerating write by exploiting PCM asymmetries. In *Proceedings of the IEEE 19th International Symposium on High Performance Computer Architecture (HPCA'13)*.

[51] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. 2009. A durable and energy efficient main memory using phase change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*.

[52] Yanqi Zhou, Ramnatthan Alagappan, Amirsaman Memaripour, Anirudh Badam, and David Wentzlaff. 2017. *HNVM: Hybrid NVM Enabled Datacenter Design and Optimization*. Microsoft Research TR 8 (Feb. 2017).

[53] Abu Zubair and Amro Awad. 2019. Anubis: Ultra-low overhead and recovery time for secure non-volatile memories. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA'19)*.

[54] Abu Zubair, Sudhanva Gurumurthi, Vilas Sridharan, and Amro Awad. 2021. Soteria: Towards resilient integrity-protected and encrypted non-volatile memories. In *Proceedings of the 54th International Symposium on Microarchitecture (MICRO'21)*.

[55] Pengfei Zuo, Yu Hua, and Yuan Xie. 2019. SuperMem: Enabling application-transparent secure persistent memory with low overheads. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'19)*.

[56] Pengfei Zuo, Yu Hua, Ming Zhao, Wen Zhou, and Yuncheng Guo. 2018. Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'18)*.

Received 22 October 2023; revised 22 March 2024; accepted 15 April 2024