DedupHR: Exploiting Content Locality to Alleviate Read/Write Interference in Deduplication-Based Flash Storage

Suzhen Wu, *Member, IEEE*, Chunfeng Du, *Student Member, IEEE*, Weiwei Zhang, Bo Mao, *Member, IEEE*, and Hong Jiang, *Fellow, IEEE*

Abstract—Data deduplication has been widely employed in flash-based storage as an effective technique for enhancing reliability and cost efficiency. However, the read/write interference problem of flash storage, where on-going write requests prevent read requests from accessing available data and significantly increase read latency, remains a critical concern with the rapid evolutions of flash chips from SLC through MLC to TLC and on to QLC, with PLC on the horizon, especially under workloads with a mixture of read and write requests. To significantly improve the read performance in face of read/write interference, which is often more critical than the write performance, we propose an enhanced Hot Read Data Replication scheme for deduplication-based flash storage, called DedupHR, to alleviate the read/write interference problem. DedupHR exploits the content locality in the deduplication-based flash storage and outsources the data blocks with high reference count to a surrogate space such as a dedicated spare flash chip or an over-provisioned space in an SSD. By servicing some conflicted read requests on the surrogate flash space, DedupHR can significantly alleviate, if not entirely eliminate, the contention between the read requests and the on-going write requests. The evaluation results show that DedupHR significantly improves the state-of-the-art schemes in terms of the system performance and cost efficiency. Consequently, the tail-latency of the deduplication-based flash storage is also reduced.

Index Terms—Deduplication-based flash storage, read/write interference, reference count, data replication

1 Introduction

Flash-based storage devices have become an attractive alternative to HDDs and received a great deal of attention from both academia and industry [1]. Apart from its deployment on mobile devices and desktop/laptop PCs, flash storage has been employed in the high performance computing and enterprise environments. Different from magnetic HDDs, flash storage consists of semiconductor chips and can provide many benefits, such as low power consumption, high robustness to vibrations, and most importantly, high small-random-read performance [2], [3]. However, flash storage also has some disadvantages, such as the asymmetric read and write performance characteristics that induces the read/write interference problem [1], [4]. This interference refers to the phenomenon when on-going write requests preempt available flash memory resource to block read requests, which can cause significant read delays.

Currently, there are different types of NAND flash chips on the storage market [5]. Earlier Single-Level Cell (SLC) NAND flash chips store a single bit of data in each cell, i.e., *a single floating gate transistor*. Each transistor can be set to a specific threshold voltage within a fixed range of voltages. SLC NAND flash chips divide this fixed range into two voltage windows, where one window represents the bit value 0 and the other represents the bit value 1. In Multi-Level Cell (MLC) NAND flash chips, which have been widely commercialized, the same voltage range is instead divided into four voltage windows that represent each possible 2-bit value (00, 01, 10, and 11). Each voltage window in MLC NAND flash chips is therefore much smaller than a voltage window in SLC NAND flash chips, which makes it more difficult to distinguish the value stored in a cell. Recently, Triple-Level Cell (TLC) and Quadruple-Level Cell (QLC) [6] flash chips have been commercialized, which respectively divide the voltage range into eight and sixteen windows, representing a 3-bit and 4-bit value. In the meantime, Penta-level Cell (PLC) flash chips appear to be on the horizon. Encoding more bits per cell increases the capacity of the SSD without increasing the chip size, yet it also decreases reliability by making it more difficult to correctly store and read the bits. As moving from SLC through MLC to TLC and on to QLC, NAND flash chips are offering higher density and lower cost but at the expense of lower performance and endurance [7].

Fig. 1 shows the read and write latency for different types of NAND flash chips [7]. The performance gap between read and write accesses are widened significantly along the evolution path of flash chips from SLC to QLC, with increasing severity of read and write interference problem [4]. Such

- *Suzhen Wu, Chunfeng Du, Weiwei Zhang, and Bo Mao are with the School of Informatics of Xiamen University, Xiamen 361005, Fujian, China. E-mail: {suzhen, maobo}@xmu.edu.cn, dcf_wy@163.com, 1597474912@qq.com.*
- *Hong Jiang is with the Computer Science and Engineering Department, University of Texas at Arlington, Arlington, TX 76019 USA. E-mail: hong.jiang@uta.edu.*

Manuscript received 20 Jan. 2021; revised 4 Apr. 2021; accepted 16 May 2021. Date of publication 26 May 2021; date of current version 10 May 2022. (Corresponding author: Suzhen Wu.) Recommended for acceptance by J. Zhai. Digital Object Identifier no. 10.1109/TC.2021.3084116

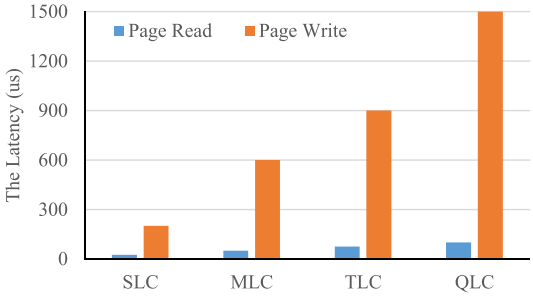


Fig. 1. The read and write latency for different types of flash chips.

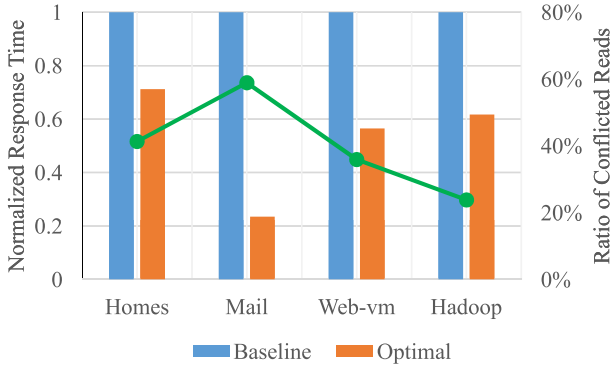


Fig. 2. The performance impact of the read/write interference problem by comparing the baseline and optimal systems. The baseline deduplicating system is the default configuration with no optimizations and the optimal system refers to one where all the interfered read requests are serviced without being blocked by any of the interfering write requests.

read and write interference results in unpredictable performance and creates significant challenges in consolidated and enterprise environments where mixed read and write requests are norm. Unfortunately, this read/write interference problem has not received adequate attention. Many existing studies focus on the garbage-collection (GC) induced performance degradation problem because GC operations are much more expensive for flash storage than for other types of storage [8], [9], [10], [11]. However, our evaluations and analysis reveal that the read and write interference is also very harmful for read performance because the write requests are much more frequent than the GC operations.

On the other hand, data deduplication has become a commodity feature in flash-based storage for many leading companies, such as HPE Nimble Storage [12] and Pure Storage [13], for the purpose of enhancing the flash reliability and space efficiency by eliminating duplicate write data [14], [15], [16]. Our workload analysis of deduplicating storage workloads shows that data blocks with high reference count have higher probability of read accesses, which can be classified as the hot read data in our previous HotR scheme [17]. Moreover, write or update operations to the data blocks with high reference count will not make the original data blocks invalid, but only decrease their reference count, which implies that these data blocks have a longer lifetime in flash storage.

Based on the above observations, we propose a new Hot Data Replication scheme for deduplication-based flash storage, inspired by HotR [17] and called DedupHR, to alleviate the read/write interference problem. DedupHR exploits the content locality in the deduplication-based flash storage to detect and outsource the data blocks with high reference count, also referred to as hot read data, to a surrogate space such as a dedicated spare flash chip or an over-provisioned space in an SSD. By servicing some conflicted read requests on the surrogate flash space, DedupHR can significantly alleviate, if not entirely eliminate, the contention between the read requests and the on-going write requests. The evaluation results show that, compared with the state-of-the-art approaches, DedupHR improves the system performance and cost efficiency significantly. Consequently, the tail-latency of the deduplication-based flash storage is also reduced.

The rest of this paper is organized as follows. Background and motivation are presented in Section 2. We describe the design of the proposed new hot data replication in Section 3. The performance evaluation is presented in Section 4. We conclude this paper in Section 6.

2 Background and Motivation

In this section, we first describe how read and write interference degrades the performance of flash storage. Then we illustrate deduplication-based flash storage and analyze the workload characteristics of deduplicating storage to motivate our DedupHR optimization for deduplication-based flash storage.

2.1 Read/Write Interference Problem

Besides erase operations, write operations are the performance bottleneck of flash storage, not only because they require more time than read operations, but also because they block the subsequent read operations in the waiting queue [18], [19], [20]. Under a concurrent workload with a mixture of read and write requests, the on-going write requests preempt available flash storage resource so as to block read requests, which is the so-called the read/write interference problem. Such read/write interference results in unpredictable performance and creates significant challenges for flash storage design.

Fig. 2 shows what percentage of read requests are blocked by the on-going read requests and how the system performance is degraded by the read/write interference, under the four different deduplicating workloads summarized in Table 2. An average of 40.0 percent, with a maximum of 58.9 percent read requests are blocked by the on-going write requests, resulting of an average of 46.8 percent performance degradation, and as much as 76.5 percent for the Mail workload. These results indicate that the read/write interference problem significantly degrades the system performance and should be carefully addressed when designing flash-based deduplicating storage systems.

2.2 Deduplication-Based Flash Storage

Recent studies have revealed that moderate to high data redundancy exists in primary and enterprise storage systems [21], [22]. Due to the unique characteristics of flash memory, applying data deduplication to flash-based storage has also been well studied [13], [14], [15]. CAFTL [14] and CA-SSD [15] employ data deduplication to eliminate redundant write data to improve the endurance and performance of flash-based SSDs. Moreover, the data deduplication

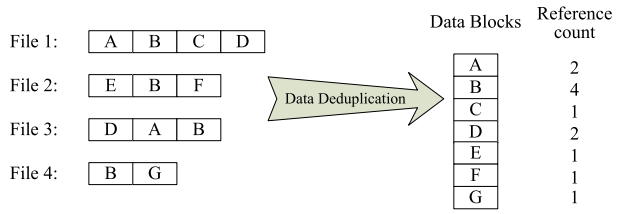


Fig. 3. The data reference characteristics in data deduplication systems.

technique has been widely employed in flash-based storage products from leading companies, such as HPE Nimble Storage [12] and Pure Storage [13], for the purpose of enhancing reliability and space efficiency.

Fig. 3 shows an example of the data reference count in the data deduplication system. In the context of deduplication-based flash storage, reference count of a data block, which refers to the number of different files a data block belongs to or is shared by, is an important measure of content locality. It is evident that data blocks with high reference count are shared by many files, for example, data block *B* in Fig. 3. Whenever any of the files containing *B* is accessed, this data block will be accessed. It implies that data blocks with high reference count are likely to be accessed much more frequently, because they are shared by many files (or data blocks). Our extensive analysis of deduplicating workloads further confirms that data blocks with high reference count are accessed frequently, as elaborated in Section 2.3.

2.3 Workload Characteristics of Deduplicating Storage

Understanding the workload characteristics is important for the design of storage systems. For this purpose, we analyze the content locality exhibited in workloads from deduplication-based storage systems. Fig. 4 shows the percentages of read requests that hit the data blocks with different reference counts in a deduplication-based storage system. The reference count is an important indicator of data content locality in deduplication-based storage systems [21], [22], [23], [24]. For example, a previous study on the 113 virtual machines (VMs) from 32 VMware ESX hosts reveals that a high degree of similarity among the VM disk images results in more than 80 percent of reduction in the storage space. Moreover, about 6 percent of unique data blocks are referenced more than 10 times and some data blocks are referenced over 100,000 times at 4KB block size [23].

Fig. 4 shows that the unique data blocks that are referenced more than 5 times account for over 42.2 percent of the total read accesses, but only amount to about 4.5 percent of all data blocks [24], [25]. Moreover, these unique data blocks with a reference count larger than 2 are likely to be accessed much more frequently, with an average of over 50 percent of all read accesses, since they are shared by many files or data blocks. In addition, these unique data blocks only account for less than 10 percent of total capacity. Moreover, these unique data blocks with high reference count are rarely deleted because they are referenced by many different files. A file deletion or update of the unique data blocks with high reference count only decreases the reference count by 1 for the data block.

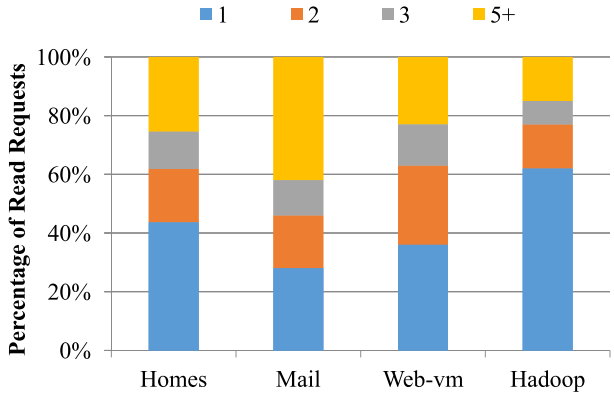


Fig. 4. The percentages of read requests that hit in data pages with different reference counts in the deduplicating storage systems.

With the evolution of flash chips from SLC to MLC, TLC and QLC, the performance gap between the read and write accesses are widened substantially. Thus, the read performance will not only be significantly degraded but also become unpredictable by the on-going write requests. On the other hand, data deduplication has become a commodity feature in flash-based storage systems. This, combined with awareness of the content locality of deduplicating workloads, motivates the design of DedupHR that proactively migrates the data blocks with high reference count to a pre-reserved space (e.g., a staging space such as a dedicated flash chip or the dedicated flash space in each flash chip in an SSD) to significantly alleviate, if not entirely eliminate, the contention between the on-going write requests and the incoming read requests. This helps simultaneously improve the user I/O performance and reduce the tail latency of deduplication-based flash storage.

3 DedupHR

In this section, we first outline the main principles guiding the design of DedupHR. Then we present a system overview of DedupHR, followed by a description of the data structures, data migration and request processing workflow in DedupHR. The design choice and data consistency issues of DedupHR are discussed at the end of this section.

3.1 Design Principles

DedupHR focuses on alleviating the read/write interference problem to achieve high performance, flexibility and extendibility, as follows.

High Performance. Due to high write processing overhead of modern flash chips, the user read response time in face of the read/write interference must be significantly reduced. DedupHR strives to achieve this goal by significantly alleviating, if not entirely eliminating, the contention between the read requests and the on-going write requests, by migrating the high reference data pages to the staging space.

Flexibility. In the DedupHR design, the staging space can be a dedicated flash chip or the dedicated flash space within each flash chip in an SSD. How to choose an appropriate staging space is based on the requirements on performance and maintainability, along with the trade-offs between them.

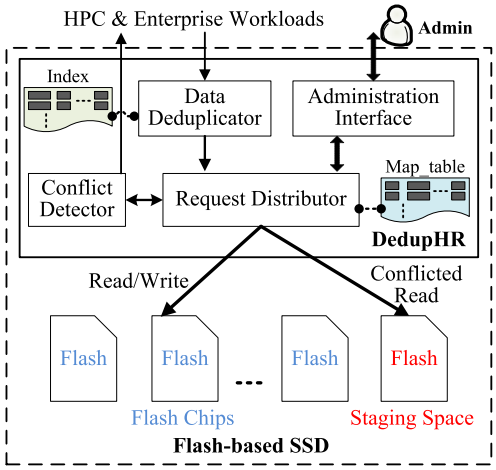


Fig. 5. System overview of DedupHR.

Extendibility. Since I/O interference in an SSD is not only between user read and write operations but also between external user read/write operations and internal garbage collection operations, the idea of DedupHR should be readily extendable to alleviate interferences among all these operations, i.e., user read, user write and garbage collection operations.

3.2 System Overview of DedupHR

The main goal of the DedupHR scheme is to improve the performance of the deduplication-based flash storage system by addressing the read/write interference problem. Fig. 5 shows a system overview of our proposed DedupHR. In our current design, DedupHR is incorporated into the Flash Transactions Layer of SSDs. However, DedupHR can also be implemented in other system software layers, such as the I/O scheduler and RAID controller layers of deduplication-based flash storage systems [26], [27].

As shown in Fig. 5, DedupHR consists of four key functional modules: Data Deduplicator, Conflict Detector, Request Distributor and Administration Interface. *Data Deduplicator* is responsible for deduplicating the incoming write data and passing the reference count information to the *Request Distributor* to migrate these data pages. *Conflict Detector* is responsible for detecting read requests that are conflicting with the on-going write requests being processed in flash chips. *Request Distributor* is responsible for redirecting the user I/O requests to the proper places, either the original addresses or ones in the staging space, according to the Conflict Detector module. DedupHR also migrates the popular read data pages to, and manages the data layout of the redirected data in the staging space. The staging space can be a dedicated flash chip or the dedicated flash space within each flash chip, which we call *staging space* in the rest of the paper, as indicated in red in Fig. 5. The original addresses in SSDs are referred to as *flash chips* in the rest of the paper, as indicated in blue in Fig. 5.

It must be noted that although DedupHR interacts with the data deduplication module, it is implemented independently of the latter's internal details. DedupHR can be incorporated into any Flash Transactions Layer of SSDs, to alleviate the aforementioned interferences among read, write and GC operations. In this paper, however, we focus on the read/write interference problem and the discussion on how DedupHR works with the garbage collection can be found in Section 4.

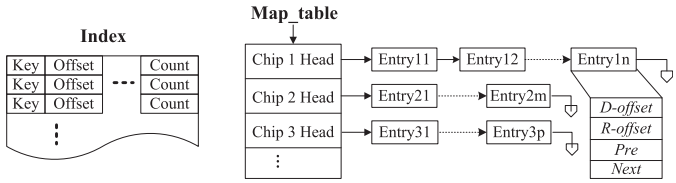


Fig. 6. The two data structures in DedupHR.

3.3 Data Structures

There are two key data structures in DedupHR, namely, Index and Map_table, as shown in Fig. 6. Index is an important data structure in deduplication-based systems. Upon each write, the fingerprints contained in Index are searched to identify whether the written data blocks are redundant or not. Generally speaking, the deduplication process can be divided into the following four stages: (1) data chunking that divides data streams/files into roughly equal-sized blocks (often based on content), (2) hash computing for each unique data block to generate a fingerprint that uniquely identifies that data block, (3) index querying that determines whether incoming data blocks are duplicated for removal, and (4) index and metadata updating.

In order to improve the efficiency of garbage collections and increase effective storage space utilization, studies on deduplication-based systems have started to leverage the reference count of data blocks. The reference *count* value of a unique data block is equal to the number of different files that reference, or share that block. When a unique data block is first stored, its count variable is initialized to be "1" and increased by one each time it is referenced again. When the count variable reaches a threshold (e.g., "3"), the corresponding unique data block is marked and will be migrated to the staging space. The reference count is tracked on the write path.

Map_table stores the mapping information for the unique data blocks that are migrated to the staging space. Map_table stores the metadata information of the migrated data pages, including the D-offset and R-offset values. It must be noted that Map_table is internally divided by chip boundaries. As an example for a single SSD, it is easy to query and determine whether a conflicted read request should be serviced in the staging space if an on-going write request is located in the same flash chip. For flash-based storage systems, it can also be divided by the regions that are convenient to physically isolate access operations, sometimes including garbage collection operations.

- *D-offset* indicates the offset of read data page on the flash chip. It is initialized when the read data page is migrated to the staging space and inserted into Map_table.
- *R-offset* indicates the offset of read data page in the staging space. It is filled when the read data page is requested and migrated to the staging space.
- *Count* indicates how many times the read data page has been accessed to capture the access frequency pattern.

- *Pre* and *Next* are two pointers used to link the sorted list.

3.4 Hot Read Data Migration

Identifying hot read data pages is very important for DedupHR to migrate popular read data pages in advance to alleviate the read/write interference [17]. As shown in Section 2.3, unique data blocks with high reference count are shared by many files and have a high probability of being accessed frequently. Thus the reference count value is an important indicator of hotness of read data pages, which is the key factor that substantially differentiates DedupHR from HotR [17].

In deduplication-based storage systems, the reference count value is updated on the write path. When receiving a write request, the Data Deduplicator module first splits the data into multiple data pages, identifies the redundant data pages by comparing the fingerprints of these data pages with those in the Index structure, and increments by one the corresponding reference counts of the unique data pages that hit in the Index lookups. Then DedupHR decides whether the unique data pages of the data (i.e., those hit the Index lookups) should be marked to be migrated based on their reference count values. The best reference count threshold can be determined through a sensitivity study, as detailed in Section 4. For pages meeting the threshold, the write data will be written to the staging space and a new entry will be inserted in Map_table to record the metadata information of the migrated write data. As a result, the subsequent conflicted read requests can be serviced by the staging space without a cold miss. The reference count threshold can be configured through the *Administration Interface*.

It must be noted that the hot read data migration does not incur extra read overhead. Once the reference count of the data page reaches the predefined threshold, the data page is directly written to the staging space. Migrating the hot read data page takes place on the write path when the data page is already in memory. Thus, DedupHR does not need to fetch the previously stored data from flash chips. Moreover, the hot read data pages are migrated on the write path, the subsequent read requests to these data pages can be directly serviced by the staging space. Compared with HotR, the cold start misses are completely eliminated.

3.5 Request Processing Workflow

Fig. 7 shows the processing workflows for write and read requests. In DedupHR, all write requests are processed by the Data Deduplicator module to eliminate redundant write data pages. If a write data page is redundant, the corresponding reference count variable is updated. Although the write operations only incur the updates of the reference count variable in Index, Map_table is also checked and queried to ensure data consistency. If the write request hits an entry in Map_table and the corresponding reference count value is decreased to "1", that entry is deleted from Map_table to make sure that subsequent read requests will fetch up-to-date data pages. The corresponding read data pages in the staging space are marked as invalid simultaneously. No matter the write requests hit Map_table or not, data consistency of the subsequent read requests are not affected

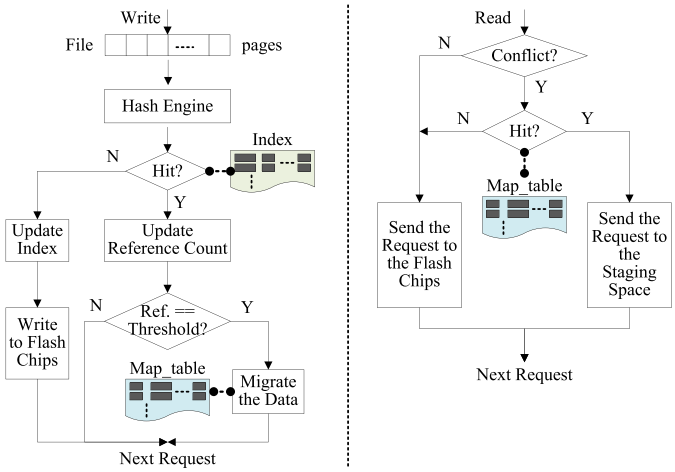


Fig. 7. The processing workflows of the write and read requests in DedupHR.

because the migrated read data pages in the staging space are always up-to-date and identical to the data pages stored on the flash chips.

Due to the deduplication process and I/O scheduler optimizations, the write requests arriving at the device level are usually bursty [26]. The large number of write requests and long program latency upon a write burst make most flash chips busy and block the incoming read requests. The Conflict Detector module monitors the program status of the flash chips and returns the results indicating whether the incoming read requests are blocked. Since the popular read data pages are classified by their locations within each flash chip of the SSD, it is easy to query and determine whether a conflicted read request should be serviced in the staging space if an on-going write request is located in the same flash chip. If a read request is blocked, the Request Distributor module queries Map_table to check whether the read request hits Map_table. If it hits, the read request is serviced by the staging space and the corresponding count value is incremented by 1. If not, the read request is serviced by the flash chips.

3.6 Design Choices

DedupHR can migrate hot read data to different persistent configurations of flash, such as a dedicated flash chip, or the dedicated flash space within each flash chip in an SSD.

A dedicated flash chip: DedupHR can store the migrated hot read data in a free flash chip, namely, a dedicated flash chip in an SSD. The advantage of this design option is its simple space management, while its disadvantage is also obvious: relatively low performance gain due to the lack of I/O parallelism. Fig. 5 shows an example of the dedicated flash chip as the staging space with an SSD.

The dedicated flash space in each flash chip: DedupHR can utilize the free space of each flash chip in an SSD as the staging space. In this case, DedupHR gains high performance owing to its high parallelism, but requires complicated maintenance. Fig. 8 shows an example of the free space of each flash chip as the staging space in an SSD.

The above two design options are both feasible and can be made available for system administrators to choose from through the Administration Interface based on their

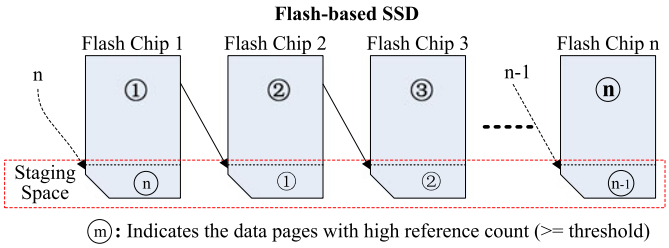


Fig. 8. An illustration of the staging space with dedicated flash space within each flash chip.

characteristics and tradeoffs. In this paper, the prototype implementation and performance evaluations are centered around a dedicated flash chip, although sample results from the other design choice are also given to show the quantitative differences between them.

3.7 Data Consistency

Data consistency in DedupHR includes the following two aspects: (1) The user read data must be up-to-date, (2) The Map_table data structure and the data stored in the staging space should be safe.

First, DedupHR must ensure that read requests fetch the up-to-date data pages. In DedupHR, the user read requests are serviced by either the flash chips or the staging space. However, write requests can render the data pages stored in the staging space invalid. Thus, if a write request hits an entry in Map_table, the corresponding entry of Map_table is deleted and the data pages in the staging space is marked as invalid. The subsequent read requests to the data pages are serviced by the flash chips. In this way, the fetched data pages by the read requests are always up-to-date.

Second, a power loss may also cause Map_table to lose its entire content. Moreover, a failure of the staging space may also render the stored data in it inaccessible. However, since the data pages stored in the staging space are only hot read data pages, they also have identical copies on the flash chips. The failures of the Map_table data structure and staging space, as a result of, say, a power failure or other data corruption, will not cause any data loss. If failures happen, DedupHR is initialized to restart. In the normal state, the Map_table data structure is stored in the memory and flushed to the backend flash storage periodically.

4 Performance Evaluation

In this section, we first describe the experimental setup and methodology. Then we evaluate the effectiveness and performance of our proposed DedupHR scheme by comparing it with the relevant state-of-the-art schemes, i.e., Rails [4], TTFlash [10], and HotR [17], through extensive trace-driven experiments with deduplication-based workloads.

4.1 Evaluation Setup and Methodology

To evaluate the efficiency of our proposed DedupHR scheme, we have implemented a prototype of the DedupHR scheme by integrating it into an open-source SSD simulator developed by Microsoft Research (MSR) [1]. The MSR SSD simulator, an extension of Disksim from the Parallel Data Lab of CMU [28], has been released to the public and widely used to evaluate the performance of the flash-based storage systems [1], [18]. The SSD-extended Disksim simulator is enhanced with a data deduplication module to implement our proposed DedupHR scheme. The values of the SSD-specific parameters used in the simulator and performance characteristics of different NAND flash types are summarized in Table 1 and Fig. 1. By default, the TLC-based NAND flash chip is used in the experimental evaluation. The read and write latencies of different flash types are the same as that shown in Fig. 1.

TABLE 1
The Default SSD Model Parameters

Parameter	Value	Parameter	Value
Total Capacity	80GB	Flash Chip Elements	10
Reserved Free Blocks	15%	Planes Per Package	8
Minimum Free Blocks	5%	Blocks Per Plane	2048
Cleaning Policy	Greedy	Pages Per Block	64
Page Size	4KB	Latencies (SLC-QLC)	Fig. 1

We use the FIU IODedup traces, including the Homes, Mail and Web-vm workloads [25], [29], and the Hadoop trace used in the CA-FTL study [14]. The three FIU traces are collected from a file server (Homes), an email server (Mail), and a virtual machine running two web servers (Web-vm) [25], [29]. The Hadoop trace is collected from 5 machines for kernel development in Department of Computer Science and Engineering at the Ohio State University [14]. The main characteristics of these workloads, including the write ratio, total I/O requests, average request size, and deduplication ratio, are summarized in Table 2. Since these workloads were collected on disk-based storage systems, we use the TraceTracker [30] to convert them into flash-based traces in our experiments.

We compare the DedupHR scheme with the original deduplicating system without any optimizations (Baseline), Rails [4], TTFlash [10], and HotR [17] in terms of average response time and cost-latency product that provides a measure of cost-effectiveness. Rails [4] is based on mirrored redundancy that physically separates reads from writes to improve read-only performance in the presence of writes. TTFlash [10] was originally designed to address GC-induced performance degradation by using the RAID5-based redundancy inside SSDs and a read reconstruction technique. We leverage TTFlash to address the read/write interference problem by replacing the GC operations with on-going write operations. Our previous proposed HotR [17] scheme outsources the popular read data to a surrogate space to alleviate the contention between the read requests

TABLE 2
The Workload Characteristics

Traces	W. Ratio	Total I/Os	Aver. Req. Size	Dedup. Ratio
Homes	80.5%	64,819	13.1 KB	33.3%
Mail	78.5%	328,145	40.8 KB	91.0%
Web-vm	69.8%	154,105	14.8 KB	47.3%
Hadoop	59.3%	9,441,783	15.1 KB	20.7%

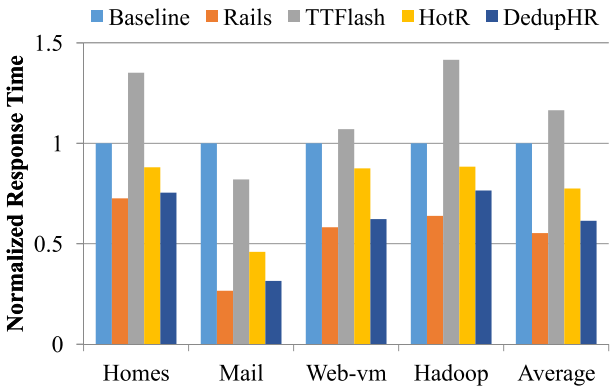


Fig. 9. Average response times, normalized to that of the baseline deduplicating system with no optimization, under the four workloads.

and the on-going write requests. However, the hot read data identification in HotR is based on logical block addresses. It must be noted that all the schemes work below the data deduplication module within the FTL layer in flash-based SSDs. By default, DedupHR uses the reference count threshold of 3 and a dedicated flash chip as the staging space in the experiments.

4.2 Performance Results and Analysis

We first conduct experiments on different schemes driven by the four deduplicating workloads. Fig. 9 shows the average response times, normalized to that of the baseline deduplicating system with no optimizations. By default, Rails uses the 4+4 RAID1 configuration and TTFlash uses the 4+1 RAID5 configuration in SSD. Hot read data pages, accounting for 10 percent of read data pages, are migrated in HotR by default [17]. First, DedupHR reduces the average response time of the baseline deduplicating scheme by up to 68.5 percent with an average of 38.6 percent. The significant performance improvement comes from the fact that an average of 31.3 percent user read requests are redirected to the staging space, as indicated in Fig. 10. Most of these migrated user read requests would have been blocked by the on-going write requests without migration, resulting in the contention between read requests and writes requests being significantly alleviated. From the point of view of

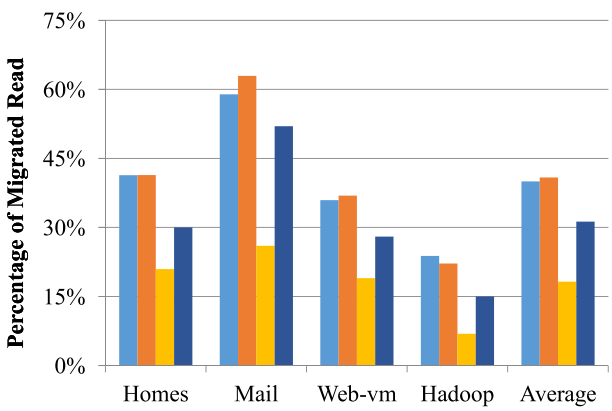


Fig. 10. The percentages of redirected read requests by the different schemes. The"Blocked" bars indicate the percentages of read requests that are blocked in the baseline deduplicating system with no optimizations.

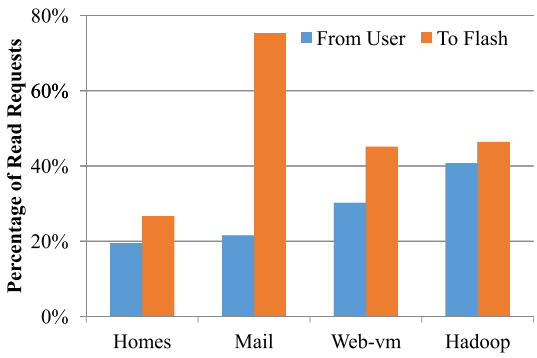


Fig. 11. Percentages of user-issued read requests versus those issued to the flash device.

read requests, the average response time is notably reduced by avoiding much of the long latency caused by programming operations (writes).

Second, DedupHR significantly outperforms TTFlash but slightly underperforms Rails. This is because, while avoiding the interference between user read requests and write requests by read reconstruction in parity-based redundancy, TTFlash's read reconstruction generates a large number of internal read requests that worsen the read/write interference problem. As a result, TTFlash is neither extendable to nor suitable for addressing the read/write interference problem for flash storage. By contrast, Rails can not only eliminate the read/write interference problem, but also take advantage of read balance offered by replication redundancy. However, the significant performance improvement of Rails by leveraging replication redundancy comes at a high space overhead. Previous studies have shown that replication is not cost effective for flash-based storage [31], which is consistent with our cost-effectiveness results.

Third, DedupHR reduces the average response time by up to 46.0 percent with an average of 29.7 percent, compared with HotR. This is because, as shown in Fig. 10, DedupHR redirects much more user read requests to the staging space than HotR by using the reference count characteristics in deduplicating systems to identify hot read data pages. By migrating these data pages with high reference count, DedupHR can proactively identify hot read data pages to avoid many compulsory misses. Moreover, with the data deduplication technique, some redundant write requests are eliminated before they arrive at the flash device. Fig. 11 compares the percentages of read requests issued by user and those issued to, i.e., actually arrive at, the flash device, which are identical for all the schemes. Because data deduplication decreases the number of write requests actually arriving at the flash devices and the total read requests are not changed, it causes the percentage of read requests to increase. Since all the schemes work under the data deduplication module, the performance of read requests becomes much more important and directly affects the system performance.

Tail Latency. In modern large-scale storage systems, such as Google, Microsoft Bing, Facebook and Amazon, the long tails of the service latency are of particular concerns [32]. With the wide deployment of flash-based storage devices in large-scale storage systems, the tail latency of flash-based devices ought to be a very important consideration for the

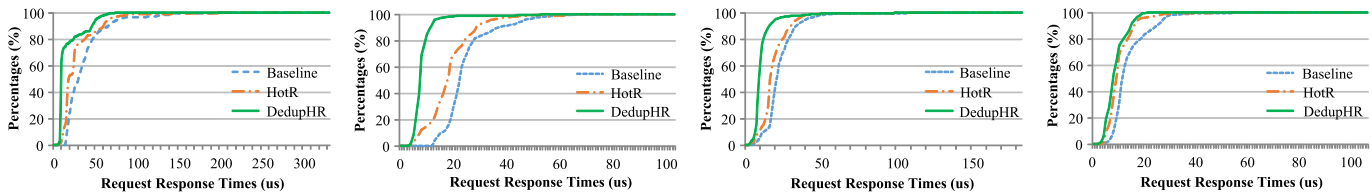


Fig. 12. Response time distributions for the baseline deduplicating system, HotR and DedupHR schemes under the four deduplicating workloads, where the X-axis indicates the request response times while the Y-asis indicates the fraction of requests whose response times are lower than the corresponding values on the X-axis.

design of storage systems [33]. To investigate the impact of DedupHR on tail latency under different workloads, we plot the response time distributions for the baseline deduplicating system, HotR and DedupHR schemes in Fig. 12. DedupHR is shown to consistently and significantly outperform the baseline deduplicating system in terms of the tail latency performance. While tail latency is mainly caused by the programming operations of flash chips, the write-induced read performance degradations are alleviated by the read request redirections in DedupHR. As a result, the percentage of requests with long latency is reduced accordingly. Moreover, DedupHR also outperforms HotR. The reason is that DedupHR proactively identifies the hot read data by exploiting the reference count characteristics in deduplicating workloads. It can redirect much more read requests that are conflicted with on-going write requests. As a result, DedupHR can improve the system performance significantly, especially for deduplicating workloads with high deduplication ratios, such as the Mail workload.

Cost-Effectiveness. To meaningfully estimate and quantify the cost-effectiveness of DedupHR in comparison with the state-of-the-art redundancy-based schemes, we use the cost-latency product as a measure for cost-effectiveness, taking inspiration from the energy-latency product that is commonly used in the computer architecture literature to quantify the energy efficiency [34]. The cost is mainly on the extra storage space that each scheme consumes, normalized to that of the baseline system without any extra space overhead. The lower the cost-latency product value of a scheme, the more cost-effective the scheme is. Fig. 13 shows the cost-effectiveness, in terms of the cost-latency product, of the different schemes based on the latency and cost results obtained from the trace-driven experiments presented earlier in this section.

Clearly, DedupHR is the most cost effective, outperforming Rails, TTFlash and HotR by 38.3, 52.7 and 22.0 percent respectively, and achieving even better cost-effectiveness than that of the baseline deduplicating system by 32.4 percent. The reasons behind the superiority of DedupHR in the cost-effectiveness measure are two-fold. First, by exploiting the workload characteristics of deduplicating systems, only data pages with high reference count are migrated to the staging space, which costs much smaller space overhead than Rails and TTFlash. Although not all the conflicted read requests are migrated, an average of 31.3 percent conflicted read requests are avoided by DedupHR. Second, by placing hot read data pages in the staging space, the access latency of the conflicted read requests is reduced significantly. These conflicted read requests are serviced without further interference from write requests, thus reducing their access latency and improving system performance. TTFlash avoids the interference between user read requests and user write requests by read reconstruction that generates a large number of internal read requests. These increased internal read requests can conflict with the user write requests, thus offsetting the advantage of read reconstruction and worsening the read/write interference problem. While TTFlash has smaller space overhead than Rails, its performance is poorer than Rails. Consequently, neither Rails nor TTFlash is cost effective and they are even worse than the baseline deduplicating system without any optimizations.

By contrast, both HotR and DedupHR have a better balance between system performance and space overhead by exploiting the workload characteristics, thus achieving the best cost effectiveness among all the schemes. However, DedupHR further utilizes the already available reference count characteristics in deduplicating systems to identify the hot read data pages and proactively migrates them to the staging space. Thus, DedupHR is much more cost effective than HotR.

4.3 Sensitivity Study

The performance of DedupHR is affected by various factors, such as the types of NAND flash chips and the reference count threshold. In order to evaluate their impact on DedupHR's performance, we conduct several sensitivity studies driven by the four deduplicating workloads. Here we only report the performance speedup results of DedupHR over the baseline deduplicating system for simplicity, since all the other schemes share the same trend.

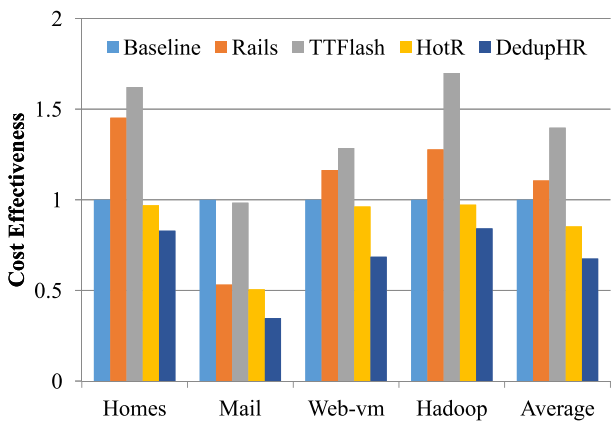


Fig. 13. The cost-effectiveness results of the different schemes under the four deduplicating workloads, normalized to that of the baseline deduplicating system with no optimizations and space redundancy. *Note that the lower the value, the better the the cost effectiveness.*

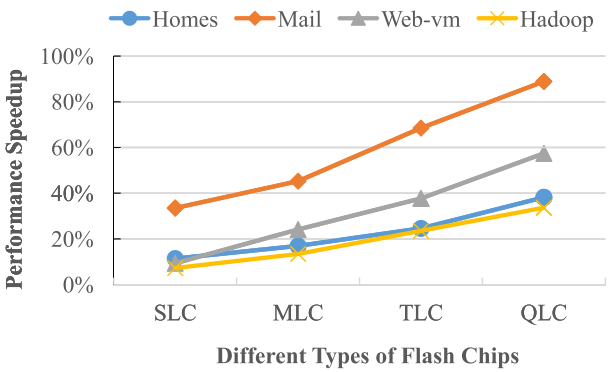


Fig. 14. The performance speedup results of DedupHR over the baseline deduplicating system with a reference count threshold of 3 and a dedicated flash chip, as a function of the different flash chips from SLC, to MLC, TLC, and QLC, driven by the four deduplicating workloads.

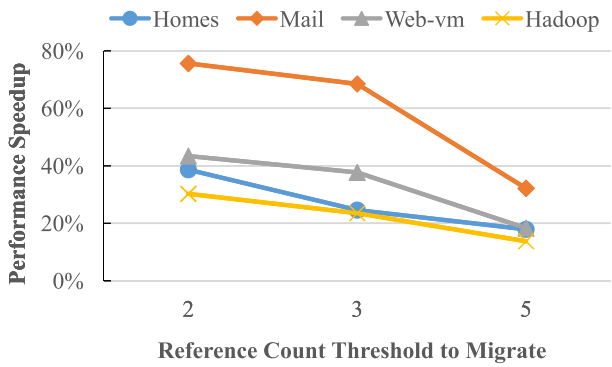


Fig. 15. The performance speedup results of DedupHR over the baseline deduplicating system with TLC-based flash chips, as a function of different reference count thresholds, under the four deduplicating workloads.

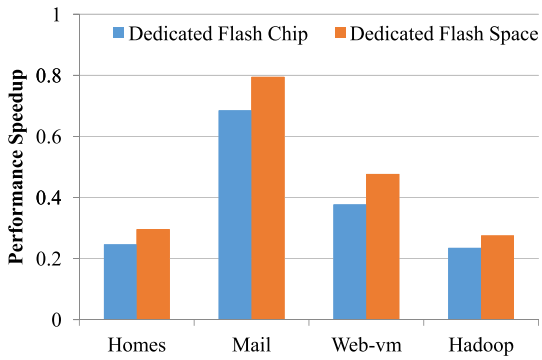


Fig. 16. The performance speedup results of DedupHR over the baseline deduplicating system with TLC-based flash chips, as a function of different design choices for the staging space, under the four deduplicating workloads.

Flash Chips. Fig. 14 shows the performance speedup results of DedupHR over the baseline deduplicating system with a reference count threshold of 3 and a dedicated flash chip, as a function of the different flash chips from SLC, to MLC, TLC and QLC, driven by the four deduplicating workloads. As bit-per-cell density increases from SLC to QLC, the performance speedup achieved by DedupHR increases accordingly. The reason is that TLC- and QLC-based flash chips have longer program latency, as shown in Fig. 14, which significantly increases of read-write conflict probability and prolongs the latency of the conflicted read requests. That is, a longer program latency causes a larger percentage of read requests being blocked by the on-going write requests. On the other hand, we also see that DedupHR improves the system performance for SLC- and MLC-based flash storage, though the performance improvement is not as significant as that for TLC- and QLC-based flash storage.

Moreover, for 3D NAND flash, the memory cells are stacked vertically in multiple layers, such as 64 or 96. Thus, it becomes popular to build large capacity SSDs [35]. Due to the overlap of flash blocks among vertical layers, the write latency is significantly increased in 3D NAND flash chips, compared with their 2D-based counterparts. For example, a TLC-based 64-layer 3D NAND chip has a 90us-180us page read latency, 900us page write latency and 10ms block erase latency [36]. In these 3D NAND-based SSDs, the advantage of DedupHR will be much more predominant.

Reference Count Threshold. Fig. 15 shows the performance speedup results of DedupHR over the baseline deduplicating system with TLC-based flash chips, as a function of different reference count thresholds, under the four deduplicating workloads. With a lower reference count threshold, much more hot read data pages can be proactively migrated to the staging space. As a result, much more conflicted read requests can be redirected to the staging space to avoid interference by the on-going write requests, but at the cost of a higher reserved space overhead. By default, DedupHR uses the reference count threshold of 3 to make a balance between improved performance and induced space overhead. It is obvious that the larger percentage of read data pages are migrated to the staging space, the larger performance speedup is achieved by DedupHR over the baseline deduplicating system. Moreover, the warm or cold read data pages only account for a small percentage of user accesses but account for a large part of the overall system data. Thus, only migrating the hot read data pages, not replicating all the data blocks as Rails, is the most cost effective.

4.4 Different Design Choices for the Staging Space

All experiments reported up to this point in this paper adopt a dedicated flash chip. To examine the impact of different types of staging space on the performance of DedupHR, we also conduct experiments with a dedicated flash space within each flash chip. Fig. 16 shows the performance speedup results of DedupHR over the baseline deduplicating system with TLC-based flash chips, as a function of different design choices for the staging space, under the four deduplicating workloads.

From Fig. 16, we can see that the dedicated flash space within each flash chip outperforms a dedicated flash chip in user response time by an average of 20.0 percent. The reason is that the chip-level access parallelism is larger in a dedicated flash space within each flash chip than that in a dedicated flash chip. With the dedicated flash space within each flash chip, as shown in Fig. 8, all the chips can be accessed in parallel to accelerate the redirected read operations, similar to the performance advantages of RAID5 over RAID4 [37]. Moreover, the internal parallelism is an important factor to improve the performance of flash-based SSDs [38], [39]. Thus, by exploiting the internal chip-level parallelism, the dedicated flash space within each flash chip

TABLE 3
Comparisons Between DedupHR and the State-of-the-Art Schemes

Scheme	Main Idea	Implementation	Space Overhead	Migration Overhead
FIOS [26]	Separates reads and writes in batches and exploits parallelism	Host	No	No
Amphibian [40]				
ParDispatcher [41]				
PIQ [42]				
P/E suspension [43]	Suspends the on-going P/E operations	Hardware modification		
Rails [4]	Separates reads from writes by replication	Host	High	Double writes
TTFlash [10]	Read reconstruction under parity redundancy	Controller	Moderate	Frequent parity updates
HotR [17]	Migrate frequently accessed data blocks to surrogate flash chips	Controller	Moderate	Hot data migration
DedupHR	Migrate data blocks with high reference count to surrogate flash chips	Controller	Moderate	Proactive data migration

can achieve better system performance than a dedicated flash chip. Nevertheless, the superior performance achieved by the dedicated flash space comes at the increased management complexity.

Under the workloads with more writes and less reads at a low deduplication ratio, more writes will be issued to the flash storage after data deduplication. Thus, the benefit of DedupHR will be somewhat limited in such an environment because the write performance will dominate the overall system performance. To fully exploit the device resources, the staging space can be dynamically used to store the incoming write data as normal flash storage driven by these workloads.

5 Related Work

Solutions proposed to address the read/write interference problem fall into two broad categories, i.e., I/O scheduler based and redundancy based schemes. Among the most representative in the I/O scheduler based solutions, FIOS [26] schedules requests with the awareness of read/write interference of SSDs. Amphibian [40] and ParDispatcher [41] take advantage of internal parallelism of SSD by separating read and write requests in batches. However, both Amphibian and ParDispatcher only consider the logical address space regions, which can not fully exploit the parallelism offered by flash-based SSDs due to their out-of-place-update nature. Gao *et al*. proposed PIQ [42] to minimize the access interferences among I/O requests in one batch by exploiting the rich parallelism of SSD. However, all these I/O schedulers only separate read and write requests in batches and the read/write interference problem still exists between read and write request batches. To avoid these drawbacks, Wu *et al.* [43] proposed a P/E (program/erase) suspension scheme, which is a device-level scheduler, to reduce the write-induced interference. However, frequently suspending/resuming the on-going P/E operation introduces system overhead and suppresses the write performance. Moreover, P/E suspension requires hardware modification.

As an alternative to scheduler-based solutions, redundancy-based solutions add space redundancy to separate interfered reads from writes to minimize the impact of their interference. Recently, Skourtis *et al.* proposed Rails [4] that utilizes the mirrored redundancy to provide predictable read performance under read/write mixed workloads by physically separating reads from writes. However, the space overhead of Rails is quite high due to the replication redundancy. In contrast, TTFlash [10] explores the parity redundancy among flash chips in SSDs to alleviate the GC-induced read performance variability by read reconstruction. The idea of TTFlash may be applicable to alleviating the write-induced read performance degradation. However, our experimental results indicate that TTFlash is harmful due to the increased number of reconstruction-induced internal read requests on flash chips, which worsens the read/write interference problem, as revealed in Section 4. Moreover, both Rails [4] and TTFlash [10] address the read/write interference problem by adding and reorganizing the low-level flash device resources, but without exploiting the high-level workload characteristics. By contrast, our previously proposed HotR scheme exploits the workload characteristics by only storing the hot read data pages on the surrogate space to alleviate the read/write interference problem, thus achieving much better cost effectiveness than Rails with comparable performance results. Moreover, HotR is much better than TTFlash in terms of both performance and cost effectiveness.

Table 3 provides a high-level comparison between DedupHR and the state-of-the-art schemes. Rails introduces migration overhead for double writes due to the employment of mirroring redundancy, while TTFlash incurs frequent parity updates induced by the parity redundancy. Both Rails and TTFlash will incur data updates overhead for normal write requests. By contrast, both HotR and DedupHR only incur limited migration overhead for the hot read data blocks. Especially for DedupHR, it further reduces the migration overhead by proactively identifying the hot read data and migrating it on the write path without any extra read overhead. Our proposed DedupHR scheme is orthogonal to and can be easily incorporated into any existing system level I/O schedulers to further alleviate the read/write interference problem.

On the other hand, applying data deduplication on the write path to eliminate redundant write data to improve the endurance and performance of flash-based SSDs has been well studied [13], [14], [15], [16], [44]. In fact, inline data deduplication has become a commodity feature in flash-based storage products for many leading companies, such as Nimble Storage [12], Pure Storage [13] and Tintri [45], for the purpose of enhancing the system performance, reliability and space efficiency by utilizing data deduplication to reduce write traffic. These schemes can alleviate the read and write interference problem by reducing the redundant write data blocks on the write path. However, the interference between read and write requests issued to the flash storage remains. Our proposed DedupHR scheme exploits the reference count

characteristics in deduplication-based flash storage systems to further alleviate the read/write interference problem. Thus, DedupHR is orthogonal to the existing data deduplication studies for flash-based storage systems.

6 Conclusion

With the evolution of flash chips from SLC to MLC, TLC, and QLC, the read/write performance gap has been increasing sharply, which worsens the read/write interference problem in deduplication-based flash storage systems. To address this problem, we propose a Hot Data Replication scheme for deduplication-based flash storage, called DedupHR, by exploiting both the workload characteristics and device characteristics. DedupHR can alleviate, if not entirely eliminate, the contention between the read requests and the on-going write requests by removing some conflicted read requests from the original addresses in flash chips. The extensive trace-driven experimental results show that DedupHR significantly improves the system performance and cost efficiency of the state-of-the-art approaches. Consequently, the tail-latency of the deduplication-based flash storage systems is also reduced.

DedupHR is an ongoing research project that offers several directions for future researches. First, we will further investigate the internal parallelism among flash chips within SSDs, particularly 3D stacked SSDs, to alleviate the read/write interference by exploiting both the up-level workload characteristics and low-level device characteristics. Second, we will implement DedupHR in a real platform, to investigate the efficiency of DedupHR under real applications and platforms.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant U1705261, 61972325, and Grant 61872305, and in part by the US NSF under Grant CCF-1704504 and Grant CCF-1629625.

References

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC'08)*, Boston, MA, USA, 2008, pp. 57–70.

[2] C. Liu, J. Kotra, M. Jung, and M. Kandemir, "PEN: Design and evaluation of partial-erase for 3D NAND-based high density SSDs," in *Proc. 16th USENIX Conf. File Storage Technol. (FAST'18)*, Oakland, CA, USA, 2018, pp. 67–82.

[3] Y. Lu, J. Shu, and W. Zheng, "Extending the lifetime of flash-based storage through reducing write amplification from file systems," in *Proc. 11th USENIX Conf. File Storage Technol. (FAST'13)*, San Jose, CA, USA, 2013, pp. 257–270.

[4] D. Skourtis, D. Achlioptas, N. Watkins, C. Maltzahn, and S. Brandt, "Flash on rails: Consistent flash performance through redundancy," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC'14)*, Philadelphia, PA, USA, 2014, pp. 463–474.

[5] Y. Cai, S. Ghose, E. Haratsch, Y. Luo, and O. Mutlu, "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proc. IEEE*, vol. 105, no. 9, pp. 1666–1704, 2017.

[6] QLC NAND Technology, Oct. 2019. [Online]. Available: https://www.micron.com/products/advanced-solutions/qlc-nand

[7] A. Alsalibi, S. Mittal, M. Al-Betar, and P. Sumari, "A Survey of techniques for architecting SLC/MLC/TLC hybrid flash memory-based SSDs," *Concurrency Comput.: Pract. Experience*, vol. 30, no. 6, 2018, Art. no. e4420.

[8] W. Choi, M. Jung, and M. Kandemir, "Parallelizing garbage collection with I/O to improve flash resource utilization," in *Proc. ACM Symp. High-Perform. Parallel Distrib. Comput. (HPDC'18)*, Tempe, AZ, USA, 2018, pp. 243–254.

[9] S. Wu, Y. Lin, B. Mao, and H. Jiang, "GCaR: Garbage collection aware cache management with improved performance for flash-based SSDs," in *Proc. 30th Int. Conf. Supercomputing (ICS'16)*, Istanbul, Turkey, 2016, pp. 1–12.

[10] S. Yan *et al.*, "Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs," in *Proc. 15th USENIX Conf. File Storage Technol. (FAST'17)*, Santa Clara, CA, USA, 2017, Art. no. 22.

[11] J. Zhang, J. Shu, and Y. Lu, "ParaFS: A log-structured file system to exploit the internal parallelism of flash devices," in *Proc. USENIX Annu. Tech. Conf. (ATC'16)*, Denver, CO, USA, 2016, pp. 87–100.

[12] HPE Nimble Storage, Nov. 2019. [Online]. Available: https://www.hpe.com/us/en/storage/nimble.html

[13] J. Colgrove *et al.*, "Purity: Building fast, highly-available enterprise flash storage from commodity components," in *Proc. ACM SIGMOD Int. Conf. Management Data (SIGMOD'15)*, 2015, pp. 1683–1694.

[14] F. Chen, T. Luo, and X. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in *Proc. 9th USENIX Conf. File Storage Technol. (FAST'11)*, San Jose, CA, USA, 2011, pp. 77–90.

[15] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam, "Leveraging value locality in optimizing NAND flash-based SSDs," in *Proc. 9th USENIX Conf. File Storage Technol. (FAST'11)*, San Jose, CA, USA, 2011, pp. 91–103.

[16] C. Li, P. Shilane, F. Douglis, H. Shim, S. Smaldone, and G. Wallace, "Nitro: A capacity-optimized SSD cache for primary storage," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC'14)*, 2014, pp. 501–512.

[17] S. Wu, W. Zhang, B. Mao, and H. Jiang, "HotR: Alleviating read/write interference with hot read data replication for flash storage," in *Proc. 22nd Design Automat. Test Eur. (DATE'19)*, 2019, pp. 1367–1372.

[18] Q. Li *et al.*, "Access characteristic guided read and write cost regulation for performance improvement on flash memory," in *Proc. 14th USENIX Conf. File Storage Technol. (FAST'16)*, Santa Clara, CA, USA, 2016, pp. 125–132.

[19] G. Belelloch, J. Fineman, P. Gibbons, Y. Gu, and J. Shun, "Sorting with asymmetric read and write costs," in *Proc. 27th ACM Symp. Parallelism Algorithms Architectures (SPAA'15)*, Portland, OR, USA, 2015, pp. 1–12.

[20] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, and M. Kandemir, "HIOS: A host interface I/O scheduler for solid state disks," in *Proc. 41st Int. Symp. Comput. Archit. (ISCA'14)*, Minneapolis, MN, USA, 2014, pp. 289–300.

[21] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti, "iDedup: Latency-aware, inline data deduplication for primary storage," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST'12)*, 2012, pp. 299–312.

[22] B. Mao, H. Jiang, S. Wu, and L. Tian, "POD: Performance oriented I/O deduplication for primary storage systems in the cloud," in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp. (IPDPS'14)*, May 2014, pp. 767–776.

[23] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li, "Decentralized deduplication in SAN cluster file systems," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC'09)*, 2009, pp. 101–114.

[24] B. Mao, H. Jiang, S. Wu, Y. Fu, andL. Tian, "Read performance optimization for deduplication-based storage systems in the cloud," *ACM Trans. Storage*, vol. 10, no. 2, pp. 1–22, 2014.

[25] R. Koller and R. Rangaswami, "I/O deduplication: Utilizing content similarity to improve I/O performance," in *Proc. 8th USENIX Conf. File Storage Technol. (FAST'10)*, 2010, Art. no. 13.

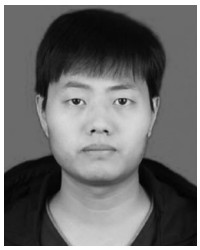
[26] S. Park and K. Shen, "FIOS: A fair, efficient flash I/O scheduler," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST'12)*, San Jose, CA, USA, 2012, pp. 155–169.

[27] S. Wu, W. Zhu, G. Liu, H. Jiang, and B. Mao, "GC-aware request steering with improved performance and reliability for SSD-based RAIDs," in *Proc. 32nd IEEE Int. Parallel Distrib. Process. Symp. (IPDPS'18)*, Vancouver, BC, Canada, 2018, pp. 296–305.

[28] J. S. Bucy, J. Schindler, S. W. Schlosser, G. R. Ganger and Contributors, "The DiskSim simulation environment version 4.0 reference manual," CMU-PDL-08-101, Parallel Data Lab., Carnegie Mellon Univ., Pittsburgh, PA, May 2008.

[29] FIU IODedup traces, Oct. 2020. [Online]. Available: http://iotta.snia.org/traces/391
[30] M. Kwon *et al.*, "Tracetracker: Hardware/software co-evaluation for large-scale I/O workload reconstruction," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC'17)*, 2017, pp. 87–96.
[31] M. Balakrishnan, A. Kadav, V. Prabhakaran, and D. Malkhi, "Differential RAID: Rethinking RAID for SSD reliability," in *Proc. 5th Eur. Conf. Comput. Syst. (EuroSys'10)*, Paris, France, 2010, pp. 15–26.
[32] J. DeanandL. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.
[33] M. Hao, G. Soundararajan, D. Kenchammana-Hosekote, A. A. Chien, and H. S. Gunawi, "The tail at store: A revelation from millions of hours of disk and SSD deployments," in *Proc. 14th USENIX Conf. File Storage Technol. (FAST'16)*, 2016, pp. 263–276.
[34] A. Banerjee, P. Wolkotte, R. Mullins, S. Moore, and G. Smit, "An energy and performance exploration of network-on-chip architectures," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 17, no. 3, pp. 319–329, Mar. 2009.
[35] C. Liu, J. Kotra, M. Jung, M. Kandemir, and C. Das, "SOML read: Rethinking the read operation granularity of 3D NAND SSDs," in *Proc. Twenty-Fourth Int. Conf. Architectural Support Program. Lang. Operating Syst. (ASPLOS'19)*, 2019, pp. 955–969.
[36] C. Kim *et al.*, "11.4 A 512Gb 3b/cell 64-stacked WL 3D V-NAND flash memory," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC'17)*, 2017, pp. 202–203.
[37] R. H. Katz, David A. Patterson, and Garth A. Gibson, "A case for redundant arrays of inexpensive disks (RAID)," in *Proc. 1988 ACM SIGMOD Int. Conf. Manage. Data (SIGMOD'88)*, 1988, pp. 109–116.
[38] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, "Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity," in *Proc. 25th Int. Conf. Supercomputing (ICS'11)*, Tucson, AZ, USA, 2011, pp. 96–107.
[39] F. Chen, R. Lee, and X. Zhang, "Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing," in *Proc. 17th Int. Symp. High Perform. Comput. Architecture (HPCA'11)*, 2011, pp. 266–277.
[40] B. Mao and S. Wu, "Exploiting request characteristics and internal parallelism to improve SSD performance," in *Proc. 33rd IEEE Int. Conf. Comput. Design (ICCD'15)*, New York, NY, USA, 2015, pp. 447–450.
[41] H. Wang, P. Huang, S. He, K. Zhou, C. Li, and X. He, "A novel I/O scheduler for SSD with improved performance and lifetime," in *Proc. IEEE 29th Symp. Mass Storage Syst. Technol. (MSST'13)*, Long Beach, CA, USA, 2013, pp. 1–5.
[42] C. Gao, L. Shi, M. Zhao, C. Xue, K. Wu, and E. Sha, "Exploiting parallelism in I/O scheduling for access conflict minimization in flash-based solid state drives," in *Proc. IEEE 29th Symp. Mass Storage Syst. Technol. (MSST'14)*, Santa Clara, CA, USA, 2014, pp. 1–11.
[43] G. Wu and B. He, "Reducing SSD read latency via NAND flash program and erase suspension," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST'12)*, San Jose, CA, USA, 2012, pp. 117–123.
[44] Q. Wang, J. Li, W. Xia, E. Kruus, B. Debnath, and P. P. C. Lee, "Austere flash caching with deduplication and compression," in *Proc. USENIX Annu. Tech. Conf. (ATC'20)*, 2020, pp. 713–726.
[45] Tintri VMstore, Oct. 2019. [Online]. Available: http://info.tintri.com/vmstore-whitepaper

Suzhen Wu (Member, IEEE) received the BE degree in computer science and technology and the PhD degree in computer architecture from the Huazhong University of Science and Technology, Wuhan, China, in 2005 and 2010, respectively. Since August 2014, she has been an associate professor with the Computer Science Department, Xiamen University. She has authored or coauthored more than 40 publications in journals and international conferences including the *IEEE Transactions on Computers*, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *IEEE Transactions on Parallel and Distributed Systems*, *ACM Transactions on Storage*, USENIX FAST, USENIX LISA, ICS, ICDCS, ICCD, MSST, DATE, and IPDPS. Her research interests include computer architecture and storage system. She is a member of ACM.



Chunfeng Du (Student Member, IEEE) is currently working toward the PhD degree with the Computer Science Department, Xiamen University. His research interests include flash and NVM storage systems. He has a paper accepted by IPDPS 2021 on deduplication enabled garbage collection optimization or flash storage.

Weiwei Zhang received the BE degree from the Computer Science Department, Xiamen University, Fujian, China, where she is currently working toward the master's degree. Her research interests include flash storage systems and I/O workload analysis. She has authored or coauthored a paper published in DATE 2019 which is selected as the Best Paper Nominees in the D track.

Bo Mao (Member, IEEE) received the BE degree in computer science and technology from Northeast University, Shenyang, China, in 2005 and the PhD degree in computer architecture from the Huazhong University of Science and Technology, Wuhan, China, in 2010. He is currently an associate professor with the Software School, Xiamen University. He has authored or coauthored more than 40 publications in international journals and conferences, including the *IEEE Transactions on Computers*, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *IEEE Transactions on Parallel and Distributed Systems*, *ACM Transactions on Storage*, USENIX FAST, USENIX LISA, ICS, ICDCS, ICCD, DATE, MSST, and IPDPS. His research interests include storage system, cloud computing, and big data. He is a member of ACM and USENIX.

Hong Jiang (Fellow, IEEE) received the BSc degree in computer engineering from the Huazhong University of Science and Technology, Wuhan, China, in 1982, the MASc degree in computer engineering from the University of Toronto, Toronto, ON, Canada in 1987, and the PhD degree in computer science from the Texas A&M University, College Station, TX, USA, in 1991. He is currently the chair and Wendell H. Nedderman endowed professor with the Computer Science and Engineering Department, University of Texas at Arlington. From January 2013 to August 2015, he was a program director with the National Science Foundation. In 1991, he joined the University of Nebraska-Lincoln as a Willa Cather professor of computer science and engineering. He has graduated 16 PhD students who upon their graduations either landed academic tenure-track positions in PhD-granting US institutions or were employed by major US IT corporations. He has authored or coauthored more than 300 publications in major journals and conferences, including the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *Proceedings of IEEE*, *ACM Transactions on Architecture and Code Optimization*, *ACM Transactions on Storage*, *Journal of Parallel and Distributed Computing*, ISCA, MICRO, USENIX ATC, FAST, EUROSYS, SOCC, LISA, SIGMETRICS, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, INFOCOM, and ICPP. His research has been supported by the NSF, the DOD, the State of Texas, the State of Nebraska, and the industry. His current research interests include computer architecture, computer storage systems and parallel I/O, high-performance computing, big data computing, cloud computing, and performance evaluation. He was an associate editor for the *IEEE Transactions on Parallel and Distributed Systems*. He is a member of ACM.