Improving the Performance of Deduplication-Based Storage Cache via Content-Driven Cache Management Methods

Yujuan Tan, Congcong Xu, Jing Xie, Zhichao Yan, Hong Jiang, *Fellow, IEEE*, Witawas Srisa-an, Xianzhang Chen, and Duo Liu

Abstract—Data deduplication, as a proven technology for effective data reduction in backup and archiving storage systems, is also showing promises in increasing the logical space capacity for storage caches by removing redundant data. However, our in-depth evaluation of the existing deduplication-aware caching algorithms reveals that they only work well when the cached block size is set to 4 KB. Unfortunately, modern storage systems often set the block size to be much larger than 4 KB, and in this scenario, the overall performance of these caching schemes drops below that of the conventional replacement algorithms without any deduplication. There are several reasons for this performance degradation. The first reason is the deduplication overhead, which is the time spent on generating the data fingerprints and their use to identify duplicate data. Such overhead offsets the benefits of deduplication. The second reason is the extremely low cache space utilization caused by read and write alignment. The third reason is that existing algorithms only exploit access locality to identify block replacement. There is a lost opportunity to effectively leverage the content usage patterns such as intensity of content redundancy and sharing in deduplication-based storage caches to further improve performance. We propose CDAC, a Content-driven Deduplication-Aware Cache, to address this problem. CDAC focuses on exploiting the content redundancy in blocks and intensity of content sharing among source addresses in cache management strategies. We have implemented CDAC based on LRU and ARC algorithms, called CDAC-LRU and CDAC-ARC respectively. Our extensive experimental results show that CDAC-LRU and CDAC-ARC outperform the state-of-the-art deduplication-aware caching algorithms, D-LRU, and D-ARC, by up to 23.83X in read cache hit ratio, with an average of 3.23X, and up to 53.3 percent in IOPS, with an average of 49.8 percent, under a real-world mixed workload when the cache size ranges from 20 to 50 percent of the workload size and the block size ranges from 4KB to 32 KB.

Index Terms—Data deduplication, storage cache, content sharing

1 Introduction

Due to the exceptional performance of solid state drives (SSDs) relative to hard drive disks (HDDs), SSDs have been widely adopted as a storage cache to boost the storage performance in large-scale HDD-based primary storage systems [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]. However, with the increasing intensity of modern workloads, the demand on the cache capacity is poised to quickly outgrow the limited capacity of SSD devices. Thus, some researchers have proposed to apply the data deduplication [12], [13], [14], [15], [16] or compression techniques [17], [18], [19] to effectively increase the cache logical capacity by reducing data footprints.

- *Yujuan Tan, Congcong Xu, Jing Xie, Xianzhang Chen, and Duo Liu are with the College of Computer Science, Chongqing University, Chongqing 400044, China. E-mail: {tanyujuan, xzchen109}@gmail.com, xucc1996@foxmail.com, {jing.xie, liuduo}@cqu.edu.cn.*
- *Zhichao Yan is with the HewlettPackard Enterprise, San Jose, CA 95002. E-mail: yanzhichao.hust@gmail.com.*
- *Hong Jiang is with the University of Texas Arlington, Arlington, TX 76019. E-mail: hong.jiang@uta.edu.*
- *Witawas Srisa-an is with the University of Nebraska Lincoln, Lincoln, NE 68588. E-mail: witty@cse.unl.edu.*

Manuscript received 16 Oct. 2019; revised 9 June 2020; accepted 16 July 2020. Date of publication 29 July 2020; date of current version 18 Aug. 2020.
(Corresponding author: Yujuan Tan.)
Recommended for acceptance by J. Wang.
Digital Object Identifier no. 10.1109/TPDS.2020.3012704

Data deduplication focuses on identifying and removing redundant data to reduce data footprints. It uses an appropriate hash algorithm [20] to generate a unique content-based identifier, commonly referred to as a data fingerprint, for each data unit (e.g., a file or data chunk). However, due to its high cost in generating fingerprints and their use for uniqueness identification [21], [22], deduplication is less popular in performance-sensitive primary storage systems [23], [24], [25] than in backup and archival storage systems [26], [27] where performance is less critical than the former. In many cases, to avoid the performance degradation, deduplication is implemented away from the critical data path, i.e., in an off-line mode. However, for many other cases where data must be deduplicated along the critical data path, such as SSD-based storage caches in primary storage systems, in-line deduplication is a requirement and also the focus of this paper.

While implementing the in-line deduplication in SSD caches, the duplicate data identification and elimination operations lay on the data read/write critical path. That is, the duplicate data are identified and removed before writing to the SSD caches. The deduplication overhead, in terms of the time spent on generating the data fingerprints and their use to identify duplicate data, would extend the data read/write access latency and degrade the performance of the overall storage system. Thus, the deduplication-based SSD caches need to be carefully designed and managed to

1045-9219 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on August 24,2020 at 17:28:02 UTC from IEEE Xplore. Restrictions apply.

reap the benefits of increased logical capacity and cache hit ratios brought by data deduplication without paying a high price for the deduplication-induced overhead, thus improving the overall storage performance.

A recent study, CacheDedup [12], addressed this problem by proposing two deduplication-aware cache replacement algorithms, D-LRU and D-ARC, that decomposed metadata from data in the cache to enhance the performance of deduplication-based SSD caches. This is the only published work addressing the problem thus far, to the best of our knowledge. However, our in-depth evaluation of D-LRU and D-ARC reveals that they only work well when the cache block size is set to 4KB. When block size is larger than 4 KB, i.e., 8 KB, 16 KB, 32 KB and 64 KB, with lower deduplication rate, they performed worse because of the deduplication overhead than the conventional replacement algorithms LRU and ARC [28] without any deduplication, as described in Section 2. These results clearly indicated that, when the block size is larger than 4 KB, a likely trend in SSD products as device capacity keeps growing rapidly and each cached block consists of multiple pages to increase throughput and amortize cache management cost, the deduplication overhead more than offsets the benefits of deduplication, significantly degrading the storage performance. In other words, D-ARC and D-LRU will no longer be useful for future SSDs due to their heavy deduplication overhead and inefficient caching policies.

The reason for the inefficient cache design in CacheDedup [12] can be explained in part by the fact that data deduplication changes how blocks are cached and evicted significantly, thus changing their locality properties. *In conventional caches, each block is identified by a unique source address*; the source address is the address of the block in the underlying storage system, and the cache maintains a mapping from the source address to the cache block address in the storage cache. The source addresses of all blocks are independent of one another. But *with data deduplication, each block is identified by its data content that can be common to and pointed to by multiple source addresses*; As a result, this content sharing among multiple source addresses whose block contents are identical causes their accesses to be dependent of one another. The more source addresses are associated with the same content, the more intense their content sharing will be. This renders any conventional cache replacement algorithms ineffective in deduplication-based caches due to their tendency to consider each source address independently and replace the cached blocks based on the access locality of each independent source address. As such, existing deduplication-aware cache algorithms, like D-LRU and D-ARC, only use the access frequency of source addresses as a hint for block replacement. For example, D-LRU identifies the hot/cold source addresses according to the last access time and removes the coldest source address when the cache is full. This is very similar to LRU except that D-LRU may need to remove multiple source addresses to find one free block while LRU needs to remove only one source address, since each block in deduplication-aware cache may be pointed to by multiple source addresses. In doing so, *both D-LRU and D-ARC miss the opportunity to be even more effective and efficient by leveraging the intensity of content redundancy and sharing in caching replacement algorithms.*

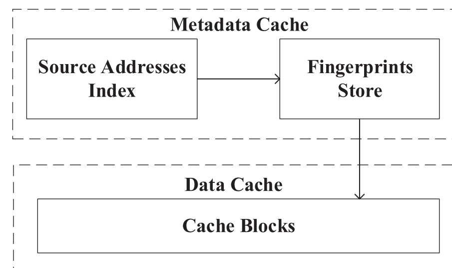


Fig. 1. Architecture of CacheDedup.

This observation motivates us to propose CDAC, a Content-driven Deduplication-Aware Caching management approach. CDAC exploits the intensity of content sharing and hotness in the design of its cache algorithms. CDAC consists of two complementary techniques: Reference-Count based Eviction (RCE) and Bitmap based Hotness Identification (BHI). In particular, RCE focuses on evicting a cold block based on its reference count [29], which is a measure of content sharing intensity and hotness, in terms of the total number of the source addresses pointing to that block. BHI helps identify a hot/cold block based on finer-grained access patterns to parts of the block captured by an access bitmap that records the access status of each individual small part within the block. If most individual small parts within a block are accessed recently, it would be regarded as a hot block; otherwise, as a cold block. This avoids the false-positive identification of hot blocks, in which one or a few tiny parts of a block being “hot” render that entire block “hot”.

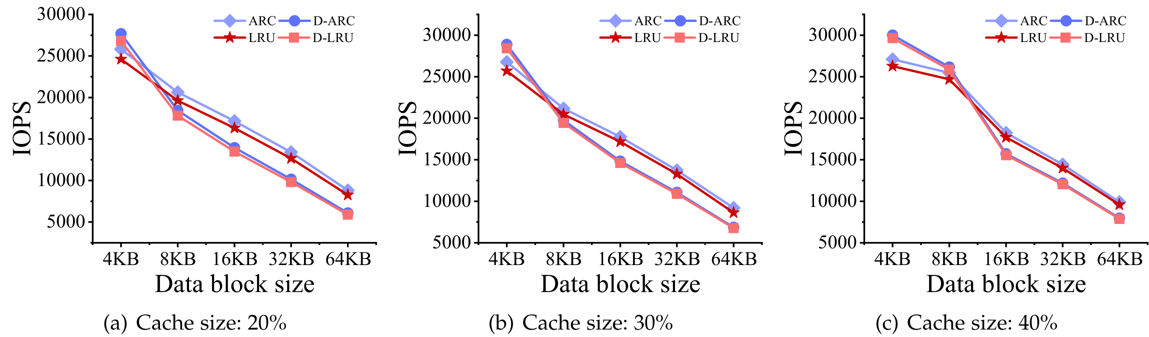
When the cache is full, CDAC will first use BHI to identify candidate cold blocks and then use RCE to determine whether the cold blocks need to be deleted. The combination of BHI and RCE enables the cache replacement algorithm to fully and accurately exploit the content sharing intensity and hotness. We have implemented CDAC based on LRU and ARC, called CDAC-LRU and CDAC-ARC respectively. Our extensive experimental results showed that CDAC-LRU and CDAC-ARC outperform D-LRU and D-ARC by up to 23.83X in read cache hit ratio, with an average of 3.23X, and up to 53.3 percent in IOPS, with an average of 49.8 percent under real-world workloads.

The rest of this paper is organized as follows. Section 2 quantitatively analyzes CacheDedup, the only known related work, to provide insight and motivation to the CDAC design. Section 3 details the design of CDAC. Section 4 evaluates CDAC, Section 5 describes the related work and Section 6 concludes the paper.

2 Background and Motivation

To the best of our knowledge, CacheDedup is the only known work that addresses the issue of deduplication-aware cache management [12]. It proposes an architecture that uses a separate Data Cache and Metadata Cache to integrate the data caching and deduplication metadata caching, as illustrated in Fig. 1. Data Cache stores the cached data blocks, and Metadata Cache stores the source addresses and data fingerprints of these blocks and maintains the mapping. Based on this architecture, two deduplication-aware caching replacement algorithms, D-LRU and D-ARC, are designed

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on August 24,2020 at 17:28:02 UTC from IEEE Xplore. Restrictions apply.



(a) Cache size: 20% (b) Cache size: 30% (c) Cache size: 40%

Fig. 2. A comparison between ARC, D-ARC, D-LRU, and LRU in IOPS as a function of the block size and cache size.

based on the traditional LRU and ARC algorithms. D-LRU uses two LRU lists to manage Metadata Cache and Data Cache separately. In Data Cache, it performs deduplication on the blocks pointed to by the fingerprints stored in Metadata Cache. D-LRU does not synchronize the block evictions in Data Cache and corresponding metadata eviction in Metadata Cache to improve performance. For example, when Data Cache is full, D-LRU will first replace unmapped blocks in Metadata Cache. If no such block is available, it then evicts the source addresses from Metadata Cache until such block is found. Except for using the ARC algorithm, D-ARC manages Metadata Cache and Data Cache in the same way as D-LRU. In addition, compared with the traditional ARC algorithm, D-ARC can retain more source addresses in Metadata Cache even if it needs to be deleted, so when the data blocks pointed to by these source addresses remain in Data Cache, the hit ratios can be improved.

D-LRU and D-ARC can improve cache hit rates by removing redundant cached blocks to increase the logical cache capacity. However, such improvements are limited due to their inability to explore the intensity of content redundancy and content sharing in cache replacement algorithms. In conventional caches, each block is identified by a unique source address and all the source addresses of all blocks are independent of one another. On the other hand, data deduplication identifies each block by its data content that can be common to and pointed to by multiple source addresses. Thus it is inappropriate to treat each source address independently and select the blocks to be replaced based on the access locality of each independent source address alone. As such, D-LRU and D-ARC only use the access frequency of each source address as a hint to identify the hot/cold blocks. This fundamentally changes the caching behaviors to the point that considering spatial and temporal localities of source addresses to improve caching performance is no longer sufficient,as will be shown by the results of our investigation to evaluate the performances and shortcomings of D-LRU and D-ARC.

In our experiments, we created a practical CacheDedup prototype as a virtual block device under the hypervisor used by multiple virtual machines. Our workload consists of Webmail server and FTP file server, with a total of about 68.7GB data. We created a virtual machine and exercised it with the specified workload. We then recorded the I/O results, which are shown and analyzed next. We provide the detailed description of our experimental setup in Section 4.

2.1 IOPS and Hit Ratios

Fig. 2 shows the IOPS for ARC and D-ARC, LRU and D-LRU, as a function of the block size, ranging from 4 KB to 64 KB, when the cache size is between 20 to 40 percent of the working set size. Here, we only show results with cache size of 20, 30 and 40 percent to save space, because the results under other cache sizes are similar. From these results, it can be seen that for 4 KB blocks, D-ARC and D-LRU obtain higher IOPS than ARC and LRU for all cache sizes tested. However, for blocks larger than 4 KB (i.e., 8 KB, 16 KB, 32 KB and 64 KB), the IOPS of D-ARC and D-LRU are lower than that of ARC and LRU, even though D-ARC and D-LRU obtain higher hit ratios than ARC and LRU, as shown in Tables 1 and 2. As shown in Fig. 2, when the cache size is 20 percent, the IOPS of D-ARC and D-LRU are about 7.2 and 9.0 percent higher than ARC and LRU for 4 KB blocks. Note that we calculated the former by dividing the IOPS of ARC with the difference between the IOPS of D-ARC and that of ARC. The latter was calculated by dividing the IOPS of LRC with the difference between IOPS of D-LRU and that of LRU.

However, when the block size grows to 8 KB, the IOPS of D-ARC and D-LRU are about 10.9 and 9.1 percent lower than those of ARC and LRU, respectively. When the block size grows to 16 KB, the IOPS of D-ARC and D-LRU further degrade to 19 and 17.8 percent lower than those of ARC and LRU, respectively. This occurs in spite of higher cache hit ratios of 8.10 (63.00-54.90 percent) and 13.01 percent (59.51-46.50 percent) than those of ARC and LRU for 8 KB blocks, and 6.53 (46.80-40.27 percent) and 13.67 percent (44.89-31.22 percent) higher than those of ARC and LRU for 16 KB blocks. These results clearly show that for D-ARC and D-LRU, increased cache hit ratios when the block size is 4 KB can improve overall storage performances. However, if the block size exceeds 4KB, the increased cache hit ratios do not

TABLE 1
A Comparison Between ARC and D-ARC in the Cache Hit Ratios as a Function of the Block Size and Cache Size

	Cache 20%		Cache 30%		Cache 40%	
	ARC	D-ARC	ARC	D-ARC	ARC	D-ARC
4KB	69.76	79.05	72.37	86.44	75.15	93.74
8KB	54.90	63.00	59.34	72.17	67.52	83.02
16KB	40.27	46.80	47.67	57.92	55.38	68.32
32KB	29.21	35.02	37.20	46.02	45.22	56.80
64KB	23.48	27.70	30.27	37.48	37.04	46.56

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on August 24,2020 at 17:28:02 UTC from IEEE Xplore. Restrictions apply.

TABLE 2
A Comparison Between LRU and D-LRU in the Cache Hit Ratios as a Function of the Block Size and Cache Size

	Cache 20%		Cache 30%		Cache 40%	
	LRU	D-LRU	LRU	D-LRU	LRU	D-LRU
4KB	62.55	74.66	69.48	84.69	72.86	92.70
8KB	46.50	59.51	56.93	70.45	65.31	81.82
16KB	31.22	44.89	44.80	55.46	53.41	67.52
32KB	23.69	32.91	33.12	41.75	43.61	56.08
64KB	18.91	25.69	25.80	33.35	36.95	46.41

improve the overall storage performances due to the offsetting overheads of deduplication, which we examine next.

2.2 Deduplication Ratio and Overhead

To better understand the inefficiency of D-ARC and D-LRU, we measured their deduplication ratios and overhead. The deduplication ratio here is defined by the amount of removed data divided by the total amount of processed data. The deduplication overhead is defined by the percentage of the time spent on the data deduplication during data caching, including the fingerprints generation and uniqueness identification.

Figs. 3 and 4 show the deduplication ratios, deduplication overhead, and increased cache hit ratios of D-ARC and D-LRU, as a function of the block size ranging from 4 KB to 64 KB, and the cache size ranging from 20 to 40 percent. As shown in Figs. 3a and 4a, when the block size increases from 4 KB to 64 KB, the deduplication ratios are significantly reduced. This is because, when the cache size is fixed and the block size becomes large, the number of blocks stored in the cache becomes smaller, and when the deduplication granularity is the block size, fewer identical blocks can be found. Therefore, the deduplication ratio is lower. Further, when the deduplication ratio becomes low, the redundant data to be eliminated becomes small, and the logical capacity of the cache becomes small. As a result, the hit ratio also become lower. As can be seen from the results shown in Fig. 3b and 4b, as the block size increases, the increased hit ratios of D-ARC and D-LRU relative to ARC and LRU becomes smaller. When the block is large, a small increase in hit ratio is not able to improve the IOPS and overall storage performances.

Figs. 3c and 4c show the deduplication overhead of D-ARC and D-LRU. As can be seen from the results, the deduplication overhead increases when the block size becomes large. Taking D-ARC for example , when the block size is 4 KB and the cache size is 20 percent, the deduplication time only accounts for 6.48 percent of the total time. But when the block size increases to 64 KB, the deduplication time can be up to 50.58 percent of the total time (as shown in Fig. 4c). This high deduplication overhead stems from two reasons. First, when the block size is large, the cache hit ratio will be lower, so more read requests will be missed in the cache, requiring more new data to be read from the underlying storage system. For each new data read into the cache, its fingerprint needs to be calculated and the uniqueness of the fingerprint needs to be identified. Therefore, the more new data is read from the underlying storage system, the more fingerprint calculation and uniqueness identification it requires, leading to longer deduplication time and higher deduplication overhead. Second, when the block size is larger, the time for calculating the fingerprint of each block is also long, and the deduplication overhead is also high.

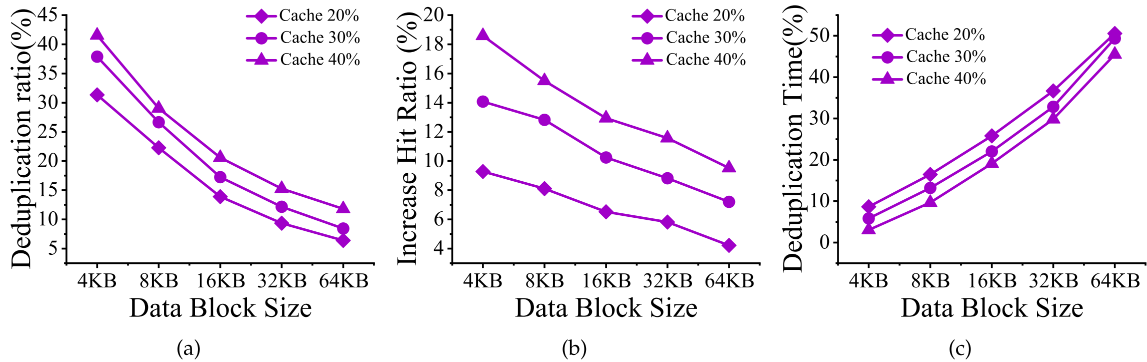


Fig. 3. The deduplication ratio, increased hit ratios, and deduplication overhead of D-ARC as a function of the block size and cache size.

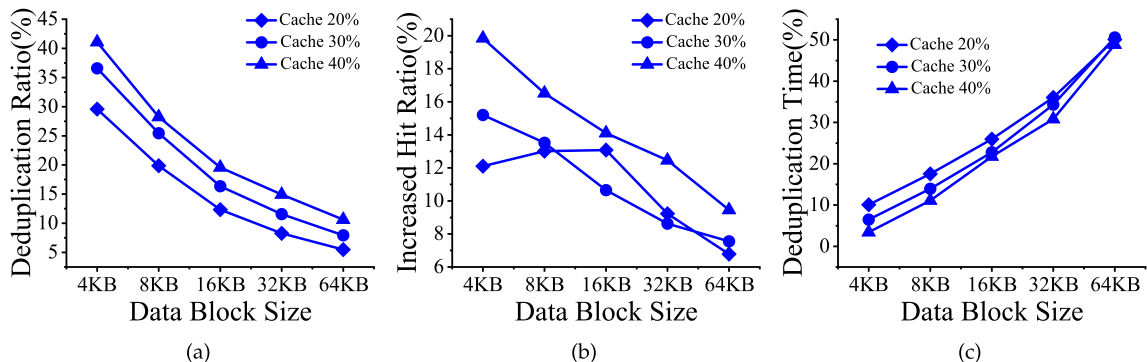


Fig. 4. The deduplication ratio, increased hit ratios, and deduplication overhead of D-LRU as a function of the block size and cache size.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on August 24,2020 at 17:28:02 UTC from IEEE Xplore. Restrictions apply.

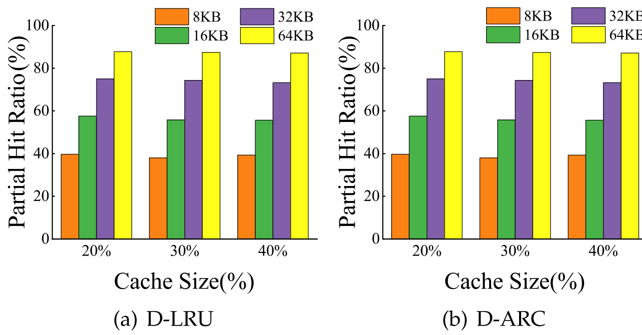


Fig. 5. Partial hit ratios of D-LRU and D-ARC.

According to our preliminary study, as shown in Figs. 3 and 4, we can conclude that as the block size gets larger, the deduplication ratio and the cache hit ratio decrease; while the cache hit ratio decreases, the deduplication overhead becomes higher. This high deduplication overhead (represented the deduplication time) directly leads to increased access latency and reduced IOPS.

2.3 Misidentified Hot Blocks

In addition, in order to further explore the reasons for the low efficiency of D-ARC and D-LRU, we studied the hot/cold blocks determined by the D-ARC and D-LRU algorithms. According to our preliminary study, it is found that when the cache block is large, a large portion of hot blocks have only a small hot portions. These blocks are generated because a small part of each block is frequently accessed. We refer to these blocks as misidentified hot blocks. In D-ARC and D-LRU, since they treat these misidentified hot blocks as completely hot blocks, they will not actively evict them to make room for some potentially hotter blocks, which will seriously affect the cache utilization rate and cache hit rate.

Fig. 5 plots the partial hit ratios to measure the severity of the misidentified hot blocks using D-LRU and D-ARC algorithms. Here, we define each request to access less than 50 percent of the cache block data as a partial hit of the cache block. When a partial cache block hit occurs, the corresponding block will be identified as a fully hot block. Therefore, the number of partial cache block hits is able to represent the number of misidentifications of hot blocks. The partial hit ratios (i.e., defined as the number of the total hits divided by the number of the partial cache block hits) can represent the percentage of hot blocks that are not correctly identified among all blocks.

As can be seen from the results of Fig. 5, in most cases, there are a large number of partial cache block hits in D-LRU and D-ARC. For example, when the cache size is set to 20 percent of the working set size and the cache block is 8 KB, the partial hits of the cache block using the D-LRU algorithm account for 39.73 percent of the total hits, and the partial hits of the cache block using D-ARC algorithm account for 37.55 percent of the total hits. When the block size is increased to 16 KB, the partial hit ratio is increased to 57.60 and 55.06 percent respectively. These results clearly show that using D-LRU and D-ARC algorithms, there are a large number of partial cache block hits. These partial cache block hits will generate many incorrectly identified cache blocks and severely affect cache utilization and hit ratios.

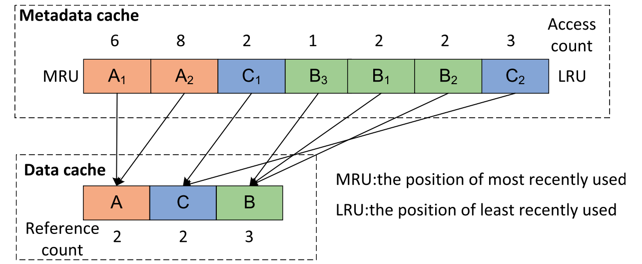


Fig. 6. An example of cached blocks.

3 CDAC Design

CDAC focuses on exploiting the intensity of content sharing and hotness in cache management strategies. It consists of two complementary techniques, Reference-Count based Eviction (RCE) and Bitmap based Hotness Identification (BHI). Both are designed based on the architecture of Cache-Dedup shown in Fig. 1.

According to the architecture, there are two caches, Data Cache and Metadata Cache. Data Cache stores the cached data blocks, while Metadata Cache stores the source addresses and data fingerprints of the data blocks. A free block in Data Cache refers to a block that does not have any source address in Metadata Cache. When Data Cache is full and no free blocks exist in Data Cache, it needs to delete some of the source addresses in Metadata Cache to generate free blocks. The selection of source addresses to be deleted and free blocks to be generated is closely related to the cache hit ratios. Therefore, in CDAC, RCE and BHI focus on how to select the source addresses to be deleted and free blocks to be generated to improve the cache hit ratios. In this section, we will describe their design in detail. In addition, CDAC supports write-back policy. For any block written to CDAC due to a new write request or read miss, CDAC will calculate its fingerprint to check if it is already in Metadata Cache. If so, it means that the same block is already stored in Data Cache, and only the source address of the block needs to be inserted into the Metadata Cache. Otherwise, it will allocate a separate new block in Data Cache to store the new block data, and insert the corresponding source address and fingerprint into Metadata Cache.

3.1 Referenced-Count-Based Eviction

RCE selects the free blocks and the associated source addresses to be deleted based on the reference count of each data block. The reference count, which is the total number of the source addresses pointing to that block, is a measure of the intensity of content sharing and hotness. A block with a high reference count should stay in cache longer than one with a lower reference count since it is pointed to by more source addresses. This higher reference count implies that more data read requests will likely be directed to this block, making it a hotter target that can help increase the hit rate and improve the storage performance.

However, while reference-count based eviction can help produce more cache hit rates, using reference count as the only hint to find the block to be replaced is not effective, unless it is jointly considered with access temporal locality of the associated source addresses. As an example, Fig. 6

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on August 24,2020 at 17:28:02 UTC from IEEE Xplore. Restrictions apply.

shows three blocks in Data Cache: A, B and C. Block A and Block C are referenced 2 times; Block B is referenced 3 times. While using the referenced count to identify the hot/cold blocks, Block B is the hottest, and Block C and Block A are the coldest. Therefore, since the referenced count of Block B is greater than that of Block C and Block A, Block A and Block C and their associated source addresses will be removed before Block B to make room for the new block. However, among all blocks, Block A has the highest access count, and Block C is not at the LRU position. Therefore, according to the access locality, in the near future, Block A and Block C is more likely to be accessed than Block B. Replacing Block A and Block C will be likely to reduce the cache hit ratio. Furthermore, storing Block B wastes both the space of Data Cache (i.e., storing block B) and Metadata Cache (i.e., storing three source addresses of Block B, B_1, B_2 and B_3), if Block B will not be accessed in the near future.

To address this problem, RCE takes both reference count and access locality into consideration to find the free blocks and associate source addresses to be deleted. While for the access locality, RCE considers both the access recency and access frequency based on the following three basic assumptions.

- First, the source address of a highly referenced data block is likely to be accessed again in the near future;
- Second, the source address that is frequently accessed recently is likely to be accessed again.
- Third, the source address of the LRU position located in Metadata Cache may no longer be accessed.

Based on the above three assumptions, RCE focuses on the source addresses in the LRU position and divides the data blocks pointed to by these source addresses into two categories: one is the data block that is referenced only once, and the other is the block that is referenced multiple times. For each source address in the LRU position, if it points to the block of the former category, RCE will delete it. Otherwise RCE moves it to the MRU position to keep it and further observe how its access frequency and the reference count of the data block pointed to by this source address will change in the next cycle. Here a cycle refers to the time required for the source address to go from the MRU position to the LRU position, and the access frequency is represented by the access count. During each cycle, RCE will reduce its access count periodically to attenuate the contribution of the old accesses to the access locality. Therefore, in the next cycle, if the access count of the source address is reduced below zero or the reference count of the data block pointed by the source address is reduced to zero, the source address will be deleted; otherwise it will be retained and go to next cycle. In addition, before moving the source address to the MRU position, RCE will decrement the reference count of the data block it points to by 1 if its access account is smaller than zero. This is because the source address should actually be deleted since its access locality (i.e., stays in the LRU position with very low access frequency) is poor, but considering the data block it points to is very hot, RCE retains it, only reducing the corresponding reference count without actually deleting it, referred to false deletion.

Algorithm 1 shows the pseudocode of RCE. Table 3 defines the corresponding symbols. It uses C_{B_i} to represent the number of the source addresses pointing to block B_i, C_{S_i} to represent the number of the source addresses pointing to block $f(S_i)$ when S_i is last located at the LRU position, and A_{S_i} to represent the access count of the source address S_i. Each time a new source address S_i enters M, C_{S_i} is set to 0. When the source address S_i reaches the LRU position, C_{S_i} will be set to C_{B_i}. At this time, if S_i is the first time to reach the LRU position, and C_{B_i} is greater than 1, meaning that more than one source address pointing to block B_i, then S_i will be retained and moved to the MRU position. In the next cycle, when S_i reaches the LRU position again, it will check both the reference count C_{B_i} and the access count A_{S_i}, if the $C_{B_i} = 0$ or $A_{S_i} \leq 0$, RCE will delete it.

TABLE 3
Variable Definition

Symbol	Definition
D	Data Cache
M	Metadata Cache
S_i	A source address in M
B_i	A data block in Data Cache
$f(S_i)$	Function that maps a source address to a data block
C_{B_i}	The number of the source addresses that pointing to the block B_i
C_{S_i}	The number of the source addresses pointing to the block $f(S_i)$ when S_i is last located in the LRU position
A_{S_i}	The access count of the source address S_i
S_L	The source address in the LRU position in M
$Flag_{S_i}$	A flag that is used to indicate the last hit type of block $f(S_i)$ in the current cycle for BHI, including INIT, READ, WRITE and NULL
P_{S_i}	The percentage of the small parts that have been accessed for S_i
H_{S_i}	The hotness value of S_i

Algorithm 1. RCE Pseudocode

Remarks: Each time a new source address S_i enters into M, C_{S_i} is set to be 0 and C_{B_i} is incremented by 1; Each time source address S_i is accessed, A_{S_i} is incremented by 1;
Input: A list of source addresses in M, the data blocks in D;
Initialization: Set $S_i = S_L$, $B_i = f(S_L)$;
if $B_i \in$ D ***And*** $C_{B_i} > 1$ **then**
  if $C_{S_i} \neq 0$ **then**
    if $A_{S_i} \leq 0$ **then**
      Remove S_i to the delete queue;
    else
      $A_{S_i} = A_{S_i} - \lambda * (C_{S_i} / C_{B_i})$;
      if $A_{S_i} \leq 0$ **then**
        $C_{B_i} = C_{B_i} - 1$;
      $C_{S_i} = C_{B_i}$;
      Move S_i to the MRU location in M;
  else
    $A_{S_i} = A_{S_i} - \lambda$;
    $C_{S_i} = C_{B_i}$;
    if $A_{S_i} \leq 0$ **then**
      $C_{B_i} = C_{B_i} - 1$;
    Move S_i to the MRU location in M;
else
  Move S_i to the delete queue;

In addition, whenever S_i arrives LRU position, RCE decreases A_{S_i} to reduce the contribution of its old accesses to

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on August 24,2020 at 17:28:02 UTC from IEEE Xplore. Restrictions apply.

the access locality. While it first reaches LRU position, RCE reduces A_{S_i} by a preset default value λ. At other times, RCE reduces A_{S_i} by $\lambda * (C_{S_i}/C_{B_i})$. Here C_{S_i}/C_{B_i} represents the hotness change of the data block pointed to by the source address in a cycle from MRU position to LRU position. If C_{B_i} is greater than C_{S_i} and C_{S_i}/C_{B_i} is smaller than 1, it means that more source addresses have been added to the cache and the data block pointed to by the source addresses are becoming hotter; otherwise if C_{B_i} is smaller than C_{S_i} and C_{S_i}/C_{B_i} is greater than 1, it means that a few source addresses have been deleted and the hotness of this data block has been reduced. Therefore, based on the value of $\lambda * (C_{S_i}/C_{B_i})$, the value of the access count will also change with the hotness of the corresponding data block f_{S_i}. If the data block gets hotter, the value $\lambda * (C_{S_i}/C_{B_i})$ will be less than the default value λ; otherwise if the data block gets cold, the value $\lambda * (C_{S_i}/C_{B_i})$ will be greater than the default value λ, which can be used to speed up the reduction of the access count.

We take Fig. 6 as an example to briefly introduce the working process of RCE. To simplify the description, the λ is set to 3. As shown in Fig. 6, the Data Cache contains three blocks, and the Metadata Cache shows the corresponding source address and the number of accesses (access counts). When a new block arrives, CDAC will selectively delete the source address from the LRU position in Metadata Cache to generate some free blocks available in Data Cache (i.e., the block without a source address point). Therefore, CDAC first deletes the source address C_2 located at the LRU position. However, according to RCE, since the reference count of Block C pointed by C_2 is greater than 1, RCE does not directly delete C_2, but reduce its access count by λ. Since its access count is still greater than 0 after attenuation, it will be moved to the MRU position without changing the reference count according to the algorithm. Then B_2 reaches the LRU position, because the reference count of Block B pointed by B_2 is greater than 1, B_2 will also move to the MRU position through the same process, but because the access counts is smaller than λ , So the reference count of Block B is reduced from 3 to 2. Then when B_1 reaches LRU position, its operation is the same as B_2 because they point to the same block and have the same access count. Thus, the reference count of Block B is reduced to 1. Finally, when B_3 reaches LRU position. Since the reference count of the Block B pointed by C_3 is only 1, RCE will directly deleted B_3. Therefore, in Data Cache, Block B becomes a free block and is evicted to make room for a new block.

3.2 Bitmap-Based Hotness Identification

In storage caches, the block size is fixed and all requests need to be aligned to the cache's block size. In conventional cache replacement algorithms, a block is identified as hot or cold completely determined by the access frequency or the last access time of its source address, regardless of the valid content for each access. For example, there are two cached blocks, A and B, with a block size of 4 KB; if block A is accessed before block B, block B will be identified as hotter than block A, even if block B only accesses 1 KB of data, and Block A accesses 4 KB of data. At this point, if the cache is full, Block A will be deleted and Block B will be retained. However, Block B contains only 1 KB of valid data, while Block B requires 4KB of cache space, resulting in lower space utilization. Furthermore, as the size of the cached block increases, this space utilization will decrease, which will seriously affect the cache hit ratio.

Algorithm 2. BHI Pseudocode

(1)The block whose source address is in the LRU location in *M*;
Remarks: A new source address enters the cache, $Flag_{S_i}$ is set to be INIT; Then if it is hit by a read or write request, $Flag_{S_i}$ is changed to READ or WRITE;
Input: a list of source addresses in *M*, the data blocks in *D*;
Initialization: Set $S_i = S_L$, $B_i = f(S_L)$;
if $B_i \in$ D **then**
  if $P_{S_i} \geq \alpha$ ***And*** $Flag_{S_i}$ == *READ* **then**
    $Flag_{S_i}$ = NULL;
    Move S_i to the MRU position in M;
  else
    identify S_i as a candidate cold source address to be deleted;
else
  remove S_i to the delete queue;
(2)The block whose source address is just accessed;
Input: a list of source addresses in *M*, the data blocks in *D*;
Initialization: S_i is accessed, $B_i = f(S_i)$;
mark $Flag_{S_i}$ as READ or WRITE;
recalculate H_{S_i} and update;
if $H_{S_i} \geq \beta$ **then**
  Move S_i to the MRU position in M;

To solve this problem, BHI uses fine-grained access patterns to identify hot/cold blocks. In a block I/O system, before generating a block request, it merges multiple consecutive I/O requests at the physical address into one block I/O request. Therefore, it is feasible to explore fine-grained access patterns to help identify hot/cold blocks. For each block, BHI breaks it into multiple small parts and then uses bitmaps to record the access status of each part. If most of the parts in a block have been accessed recently, the block will be recognized as a hot block; otherwise, it will be considered as a cold block. The access status of multiple individual parts of a block makes it possible to more accurately identify the content hotness of the block, especially for large blocks, minimizing false-positive hot block identifications. In addition, BHI only recognizes the hot blocks from the read-intensive cache blocks, regardless of the write-intensive cached blocks. This is because each write changes the block content every time, we believe that unless another read or write operation of the same content occurs, the written content should not be considered hot content. Therefore, when identifying the hot block, if the write content pointed by one source address is not shared with other source address (processed by RCE) or is no longer read, it is excluded from the hot block identification in BHI.

Based on the architecture shown in Fig. 1, BHI divides the address space of each source address into multiple small parts and uses bitmaps to record the access status of each part. If one part is accessed, the corresponding position in the bitmap is set to 1, otherwise it is set to 0. In addition, BHI adds a flag to indicate the last hit type for each source address in a cycle, including INIT, READ, WRITE, and NULL. INIT indicates the new source address added to the cache; NULL indicates that the

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on August 24,2020 at 17:28:02 UTC from IEEE Xplore. Restrictions apply.

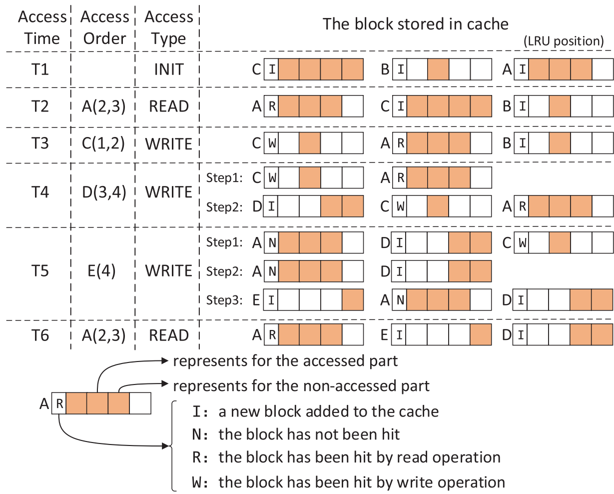


Fig. 7. The working process of BHI.

source address did not hit during the cycle; READ and WRITE indicate that the last hit of the source address in the cycle is data read or write. Algorithm 2 shows the pseudocode of BHI. As described in Algorithm 2, we divide these blocks into two categories: the block whose source addresses is at the LRU position to be deleted and the block whose source address is just accessed at any position. BHI uses separate mechanism to deal with them.

(1) For the block whose source address S_i is in the LRU location in M, BHI first checks the number of the accessed parts. If the percentage of the accessed parts P_{S_i} is greater than or equal to the preset threshold α, and the last hit of the source address S_i is data read during this period (i.e., $Flag_{S_i}$ is equal to READ), the source address S_i will be retained and moved the MRU position and $Flag_{S_i}$ will be reset to NULL. Otherwise S_i will be identified as a candidate cold source address to be deleted. It is worth noting that BHI processes each source address separately, even though some of them share the same data block. This is reasonable because the bitmap used by BHI mainly focuses on the effective access content for each access, and RCE has resolved the content sharing among source addresses.

We use an simple example to illustrate how this process works. As show in Fig. 7, the cache can only hold 3 blocks, and the percentage threshold for access portion is set to 50 percent. For simplicity, we assume that each block has only one source address, so we use blocks instead of the source addresses to illustrate how BHI works. In this example, at times T1, Block A, B and C enter the cache; their flags are set to INIT. At time T2 and T3, Block A is read and Block C is write, respectively, and their flags are changed to READ and WRITE, respectively. At time T4, Block D needs to enter the cache but there is no free blocks. So BHI checks the access status of Block B at the LRU position. Since the access portion of block B is smaller than 50 percent but its flag is INIT, Block B is deleted. At time T5, Block E needs to enter the cache. At this time, Block A is at the LRU position. Because the access portion of Block A exceeds 50 percent and its flag is READ, Block A will not be deleted; it moves to the MRU position and its flag is set NULL. Then BHI checks Block C. While for Block C, since its access portion is only 25 percent and its flag is WRITE, it is deleted. So after time T5, Block E enters the MRU location in the cache, Block C is deleted, and Block A's flag is set to NULL. At time T6, when accessing Block A again, block A moves to the MRU position and its flag is set to READ again.

(2) While for the block whose source address is just accessed, BHI checks its accessed part to determine its access popularity, and then determines whether to move it as a hot block to the MRU position as. The access popularity quantification is described in

$$Hotness = Max\left(\frac{NP}{TNP}, \frac{(CHP - TNBM * TNP)}{TNP}\right). \tag{1}$$

In Formula (1), TNP represents the Total Number of Parts in the cache block, and NP represents the Number of the Parts accessed this time, so (NP/TNP) represents the percentage of the accessed data this time; $TNBM$ represents the Total Number of times the Block is moved to the MRU position, and CHP represents the Cumulative Hits of all Parts after entering the cache, so $(CHP - TNBM * TNP)/TNP$ represents the average number of times each data part has not been moved to the MRU position when hit. Therefore, Hotness (i.e., used to quantify the access popularity) is equal to the maximum value between NP/TNP and $(CHP - TNBM * TNP)/TNP$.

BHI uses the hotness value obtained from Formula (1) to determine whether the block just accessed is a hot block. The process is describe in Algorithm 2. As shown in Algorithm 2, if the Hotness value exceeds a preset hot threshold, BHI will recognize it as a hot block and move it to the MRU position; otherwise, the block will remain stationary. In other words, it means that if the percentage of the data parts accessed this time is large enough, or the block is hit multiple times without moving to the MRU position, the block will reach the MRU position and be recognized as a hot block.

3.3 CDAC: Combining RCE and BHI Together

When the cache is full, CDAC combines RCE and BHI together to find free blocks. It first uses BHI to check if the source address in the LRU position is recognized as a cold source address. If it is a cold source address, CDAC then uses RCE to identify if it needs to be deleted. Algorithm 3 describes its working process. The source address in the LRU position needs to be constantly checked and deleted until a free block is found. The combination of BHI and RCE enables CDAC to more accurately identify the cold blocks and associated addresses to improve the cache hit ratios and IOPS, as quantitatively evidenced in Section 4.

Algorithm 3. CDAC Pseudocode

```
Input: a list of source addresses in M, the data blocks in D;
while There is no free block in D do
    S_i = S_L, B_i = f(S_L);
    if B_i ∈ D then
        run BHI;
        if S_i is identified as a candidate cold source address then
            run RCE;
    else
        move S_i to the delete queue;
remove the source addresses in the delete queue;
```

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on August 24,2020 at 17:28:02 UTC from IEEE Xplore. Restrictions apply.

4 Experimental Evaluation

In this section, we assess the benefits of CDAC with extensive experimental evaluations.

4.1 Experimental Setup

1) *Baseline Approaches*. The most relevant work to CDAC is CacheDedup with its two deduplication-aware caching algorithms, D-LRU and D-ARC, as described in Section 2. We have developed a CacheDedup prototype and use D-LRU and D-ARC as the baseline deduplication-aware caching algorithms. Moreover, to fairly compare the efficiency of CDAC to that of D-LRU and D-ARC, we also implemented CDAC based on the LRU and ARC algorithms, referred to as CDAC-LRU and CDAC-ARC respectively. CDAC-LRU and CDAC-ARC are also implemented based on CacheDedup architecture, that is, they manage Metadata Cache and Data Cache separately like D-LRU and D-ARC. But unlike D-LRU and D-ARC, CDAC-LRU and CDAC-ARC implements the hot/cold block identification module through RCE and BHI instead of using the traditional hot/cold block identification mechanism used in LRU and ARC. At the same time, we also implemented LRU and ARC, which are used as baseline non-deduplication-aware caching algorithms to highlight the benefits of data deduplication brought about by CDAC.
2) *Storage Setup*. We implemented CacheDedup as the client-side flash cache in Gluster Filesystem [30], which is a free and open source scalable network file system. The client and server each run on a node with an Intel i7-6700K CPU and 16 GB of RAM. The client side uses a 256 GB MLC SATA SSD as the cache, and the server uses a 2 TB 7.2K RPM SATA disk as the target. Both nodes run Linux with kernel 3.10.0. We installed a hypervisor in Gluster and created a virtual machine to run our experimental workload.
3) *Experimental Workload*. To evaluate CDAC, we used two datasets. First, we replayed the public FIU traces [31] collected from a VM hosting the departmental websites for webmail and online course management (WebVM), a file server used by a research group (Homes), and a departmental mail server (Mail). Second, we collected a real dataset from a Webmail server and a FTP file server hosted in a unit within a corporation; we refer this dataset as a mixed dataset. This mixed dataset contains the real data read and write operations with real data content, unlike the public FIU traces that only have the statistical features. Table 4 shows the statistical characteristics of all the datasets.
4) *Performance Metrics*. We compare CDAC to baseline approaches on two performance metrics, cache hit ratio and IOPS. Cache hit ratio is a key metric to measure the caching efficiency of any caching algorithm. A higher cache hit ratio will produce better storage performance for a specified storage system. IOPS measures the overall storage performance. Improving IOPS is the goal for all the deduplication-aware caching algorithms implemented in SSD caches. As a side note, in a virtual environment, even if we are not running any other applications, the performance results (especially IOPS) will be slightly different each time the same algorithm and the same workload are used. Therefore, in this section, we only show the average performance results after three runs.

TABLE 4
The Statistical Characteristics of the Datasets

Name	Total I/Os I/Os (GB)	Working Set(GB)	Write-to-read ratio	Unique Data(GB)
WebVM	54.5	2.1	3.6	23.4
Homes	67.3	5.9	31.5	44.4
Mail	1741	57.1	8.1	171.3
Mixed	68.7	6.2	3.98	28.9

4.2 Performance

We evaluated the cache performance for each dataset with different total cache sizes from 20 to 80 percent, and different block sizes from 4 KB to 64 KB. For FIU traces, since the public dataset has only 4 KB sized request, we merged the requests with consecutive source addresses into larger sized request and generate corresponding new fingerprints. Therefore, the minimum part of each block is set to 4 KB. In addition, in CDAC-ARC and CDAC-LRU, we set the threshold λ in RCE to the average access count of all cache blocks, and set the threshold α and β used in BHI as the median of the access part of all blocks and the median of the hotness value of all blocks, respectively. All three parameters change dynamically according to the change of block access characteristics.

4.2.1 Cache Hit Ratio

Cache hit ratio is a key metric to measure the caching efficiency. Here we mainly show the overall hit ratio (i.e., for both read and write requests) and the read hit ratio alone to evaluate CDAC's benefits. We did not show write hit ratios because the write requests in these datasets are very intensive and all of the cache replacement algorithms handle them well.

Figs. 8 and 9 show the cache hit ratio for WebVM, Homes and Mail traces with a block size of 4 KB. It is worth noting that when the block size is 4 KB, BHI will not work since the smallest part of each block is set to 4 KB. Therefore, the results of CDAC shown in Figs. 8 and 9 contain only the performance results of RCE. From these results, it can be seen that CDAC-LRU and CDAC-ARC significantly improve the read hit ratios for all the traces.

For example, with WebVM trace (see Fig. 8a), the average read hit ratio of CDAC-LRU is 13.09X ($\sum_{size=20\%}^{80\%} \frac{CDAC_LRU}{LRU}/7$) higher than that of LRU and 4.67X ($\sum_{size=20\%}^{80\%} \frac{CDAC_LRU}{D_LRU}/7$) higher than that of D-LRU. The CDAC-ARC's average read hit ratios is 1.73X higher than that of ARC and 1.27X than that of D-ARC. These results clearly show that using the reference counts that represent the intensity of the content sharing among source addresses helps preserve a lot of hot data blocks and associated source addresses.

Figs. 10 and 11 show the read hit ratios and overall hit ratios for WebVM and Mail traces when the block size is from 8 KB to 64 KB. As can be seen from the results, both CDAC-LRU and CDAC-ARC obtain higher read hit ratios and overall hit ratios, especially the read hit ratios, than

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on August 24,2020 at 17:28:02 UTC from IEEE Xplore. Restrictions apply.

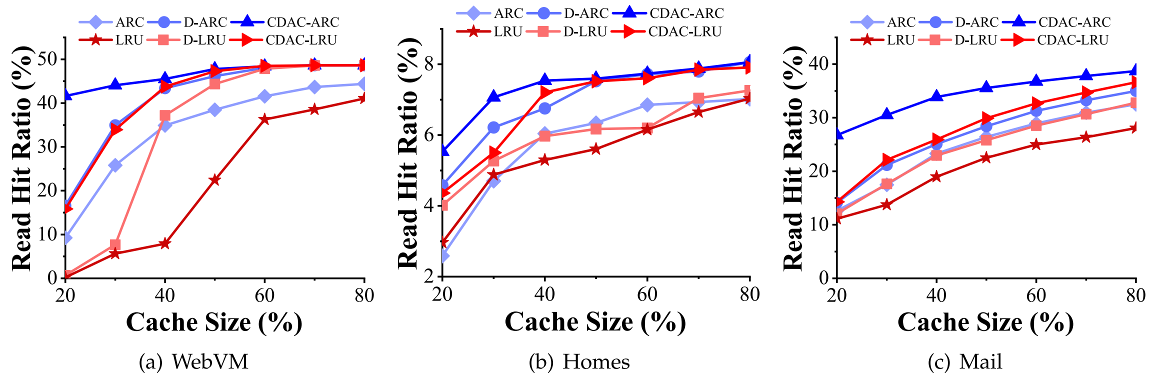


Fig. 8. Read hit ratio of FIU traces with the block size of 4KB.

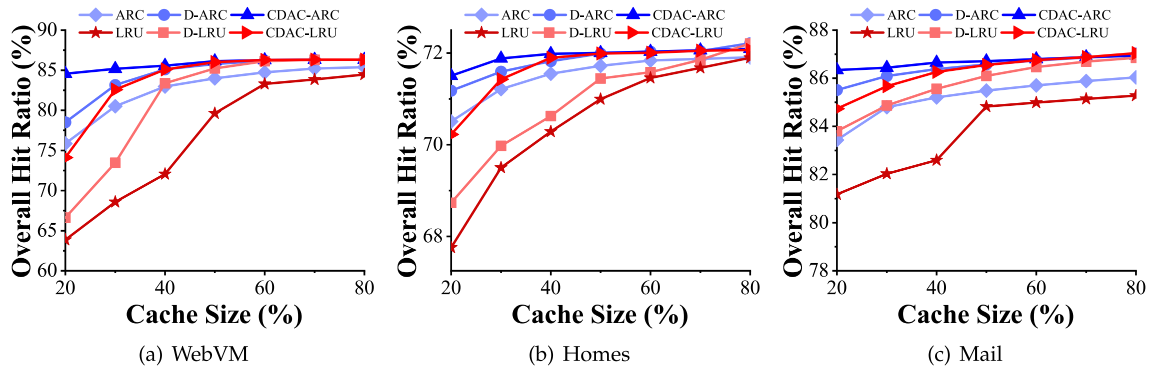


Fig. 9. Overall hit ratio of FIU traces with the block size of 4KB.

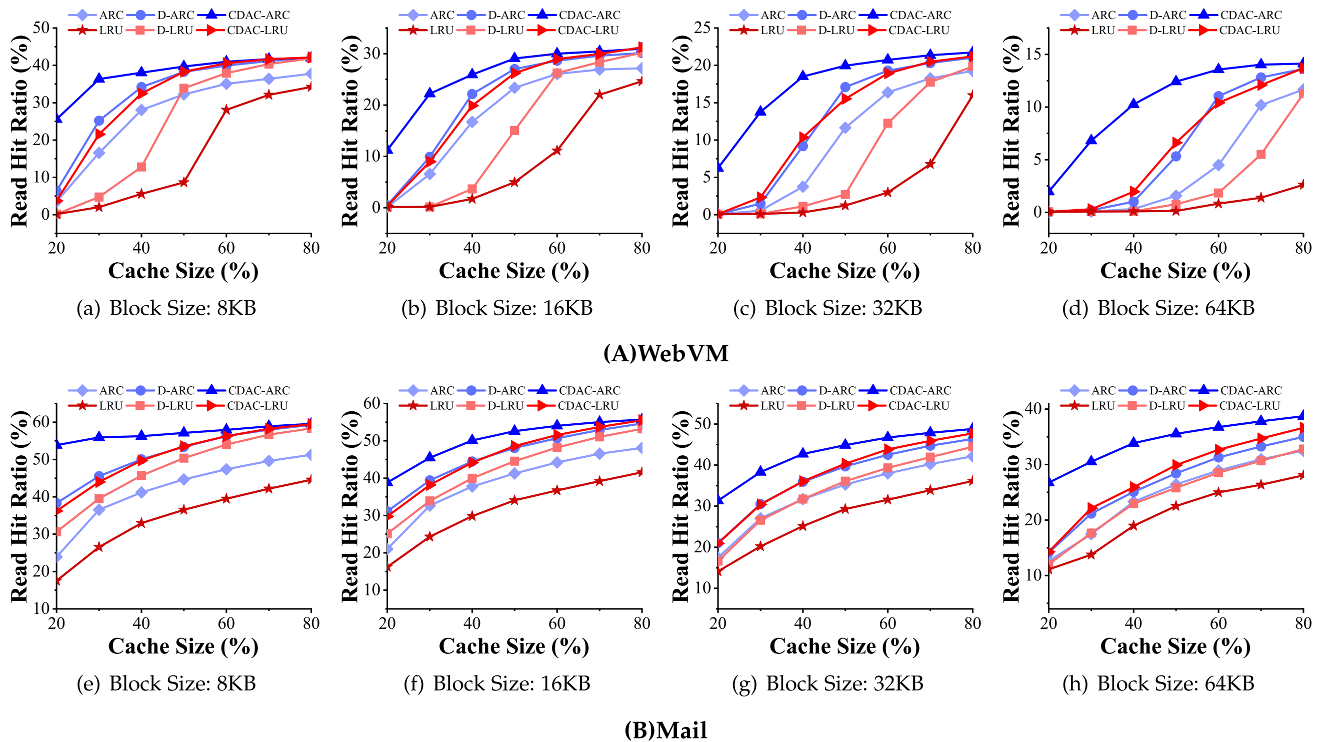


Fig. 10. Read hit ratio of WebVM and Mail traces with block size from 8KB to 64KB.

their corresponding baseline approaches. For example, when the page size is 16 KB and the cache size is 20 percent, the performance gap between D-ARC's read hit ratio and CDAC-ARC's read hit ratio is the largest, reaching 23.83X (i.e., $\frac{11.20}{0.47} = 23.83$). We observe that these improvements to the cache hit ratios have three characteristics.

First, CDAC's performance improvement in read hit ratios is greater than overall hit ratios. There are two reasons: at first,

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on August 24,2020 at 17:28:02 UTC from IEEE Xplore. Restrictions apply.

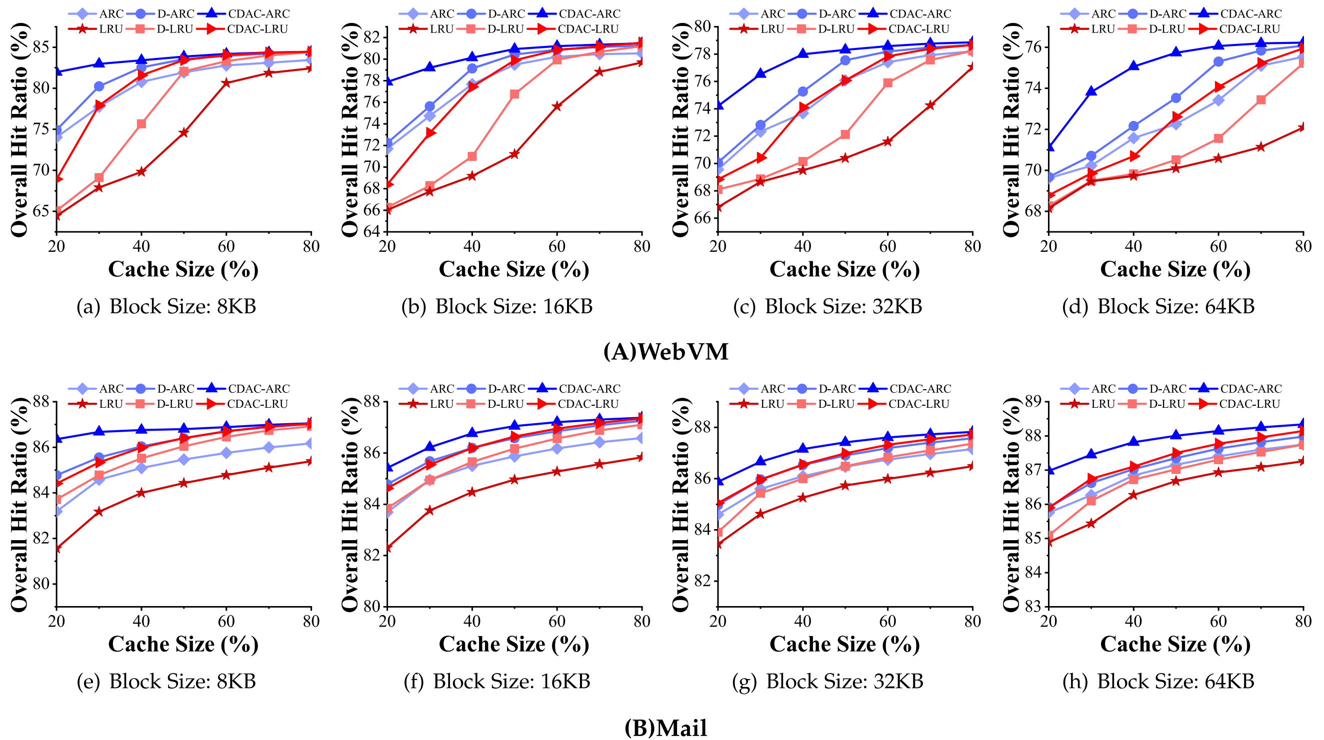


Fig. 11. Overall hit ratio of WebVM and Mail traces with block size from 8KB to 64KB.

in our experimental datasets, the write requests are very intensive and all cache replacement algorithms handle them well, so CDAC's improvement in write hit ratio is very limited; second, for all the datasets, the proportion of write requests is much higher than read requests (see Table 4), so the read hit ratios contribute much less to the overall hit ratios than the write hit ratios. Therefore, CDAC does not significantly improve the overall hit ratios as it does for read hit ratios.

Second, as the block size increases, the amount of cache space required for maximum improvement in CDAC increases. Taking the read hit ratio as an example, for the WebVM trace (see Fig. 10, when the block size is 8 KB, CDAC-LRU has the greatest improvements to the baselines when the cache size is 30 percent; but when the block size is increased to 16 KB and 32 KB, CDAC achieves the greatest improvements on the baselines when the cache size is 40 and 50 percent respectively. This is because as the block size increases, the percentage of valid content per data block may decrease in each access; in addition, the larger the blocks, the less the number of data blocks that can be stored in the cache. Therefore, at the same cache size, the sum of the effective block content in the cache becomes less. If the cache is larger, CDAC can take advantage of the more popular content to improve the cache hit ratios and achieve better performance improvements. But if the cache size exceeds a certain amount, the cache replacement algorithm is less efficient and the contribution of CDAC will be smaller.

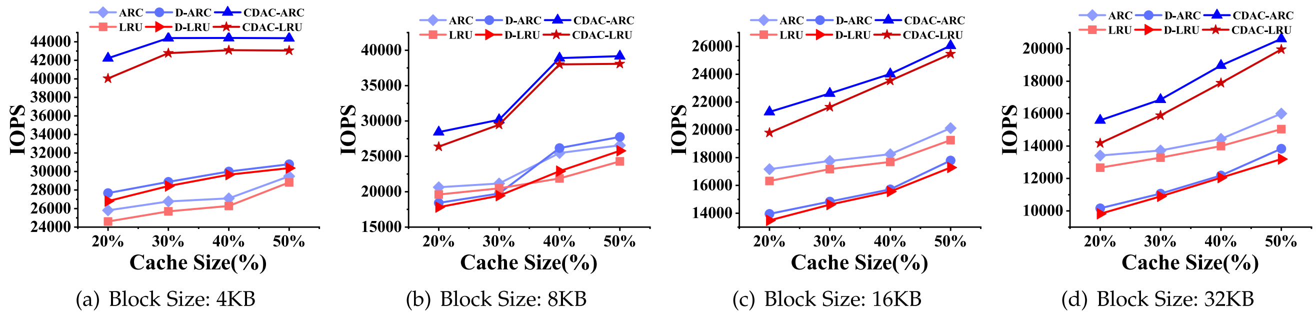
Three, when the block size is fixed, CDAC's overall performance advantages over the baselines become more pronounced as the block size decreases. This achievement can be contributed to BHI technique. Recall that, BHI divides a large block into multiple small parts. The combination of the access status of multiple small parts within a block is able to more accurately identify the hotness/coldness of each block. This benefit is further magnified by larger blocks. But for D-LRU, D-ARC, LRU and ARC, when the block size increases, it becomes more difficult for them to find the redundant blocks and accurately identify the hot/cold blocks, leading to lowered overall hit ratios.

4.2.2 IOPS

IOPS and latency are widely used metrics to measure storage system performances. Here we only show the IOPS of the mixed dataset since the public FIU trace does not have the actual data content. Fig. 12 compares CDAC-ARC, CDAC-LRU, D-ARC, D-LRU, ARC and LRU in IOPS as a function of the block size, from 4 KB to 32 KB, and the cache size, from 20 to 50 percent of the working set size. As can be seen from the results in Fig. 12, CDAC-ARC and CDAC-LRU obtain much higher IOPS than D-ARC and ARC, D-LRU and LRU consistently in all cases. On average, CDAC-ARC and CDAC-LRU outperform D-ARC and D-LRU by 49.7 percent ($\sum_{size=20\%}^{50\%} \frac{CDAC_ARC-D_ARC}{D_ARC}/4$) and 46.7 percent ($\sum_{size=20\%}^{50\%} \frac{CDAC_LRU-D_LRU}{D_LRU}/4$) for 4 KB blocks, 49.2 and 53.3 percent for 8 KB blocks, 51.1 and 48.3 percent for 16 KB blocks, and 52.6 and 47.5 percent for 32KB blocks. The IOPS of CDAC-ARC and CDAC-LRU are higher than those of ARC and LRU by 61.04 and 60.65 percent for 4 KB blocks, 45.14 and 52.2 percent for 8 KB blocks, 28.12 and 28.13 percent for 16 KB blocks, and 24.81 and 22.97 percent for 32 KB blocks. Such high IOPS of CDAC benefits from its much higher cache hit ratios than those of the baseline systems due to its efficient caching replacement algorithms of RCE and BHI.

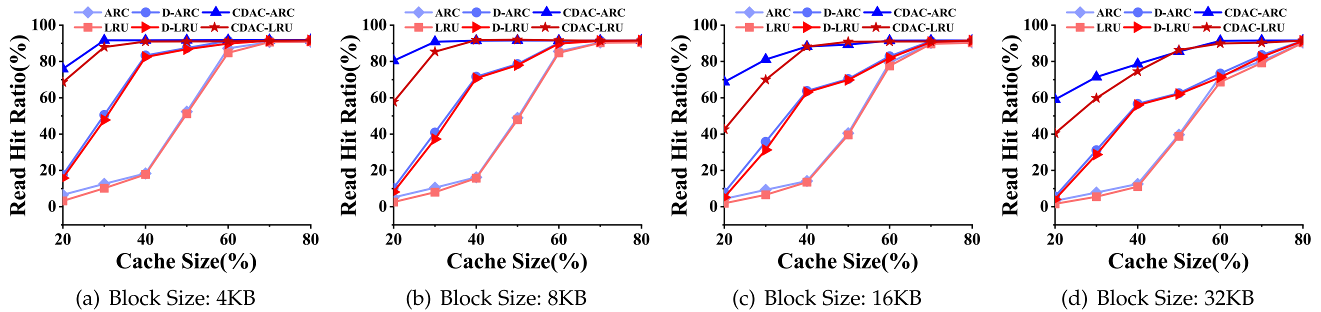
However, CDAC's improvements in IOPS are less drastic than those in cache hit ratios shown in Figs. 13 and 14, particularly read hit ratios. The reasons are twofold. First, CDAC's

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on August 24,2020 at 17:28:02 UTC from IEEE Xplore. Restrictions apply.



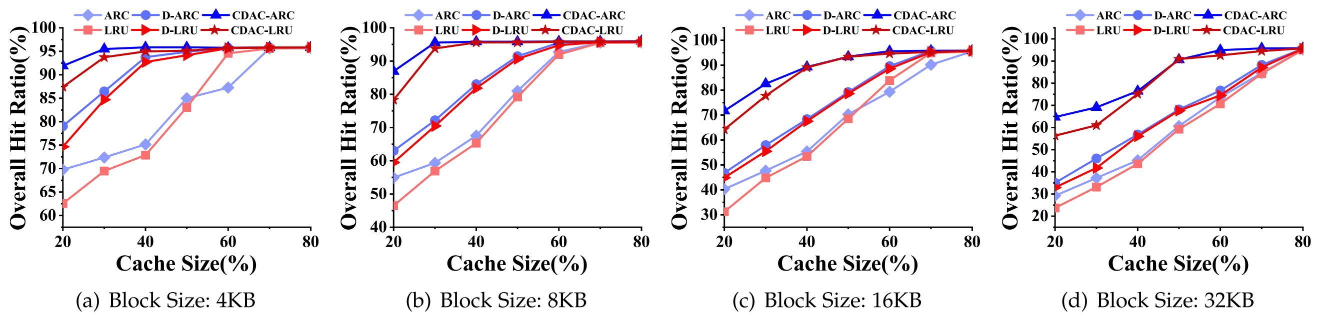
(a) Block Size: 4KB (b) Block Size: 8KB (c) Block Size: 16KB (d) Block Size: 32KB

Fig. 12. IOPS of mixed dataset with block size from 4KB to 32KB.



(a) Block Size: 4KB (b) Block Size: 8KB (c) Block Size: 16KB (d) Block Size: 32KB

Fig. 13. Read hit ratios of mixed dataset with block size from 4KB to 32KB.



(a) Block Size: 4KB (b) Block Size: 8KB (c) Block Size: 16KB (d) Block Size: 32KB

Fig. 14. Overall hit ratios of mixed dataset with block size from 4KB to 32KB.

two optimization techniques, RCE and BHI, operate on the critical path of data reads and writes. As a result, the processing overhead they incur necessarily lengthens the data read/write response time, which in turn negatively impacts IOPS. Second, in our experiment workload, the number of data read operations that directly benefit from caching is only 20 percent of the total I/O operations, making IOPS improvement far less drastic than what read hit ratios would imply.

5 Related Work

Data deduplication is usually used to reduce the data footprint to save the bandwidth required for data transmission and the capacity required for data storage. Existing research has proposed a series of related solutions to apply deduplication to different levels of storage hierarchy, including backup and archival storage [26], [27], primary storage [25], [32], and main memory [33]. The results of these studies indicate that deduplication is a viable solution to increase storage system capacity and performance.

In the SSD cache, due to the increasing demand for cache space in modern workloads and the write durability of SSD media, deduplication is also a popular way to reduce the footprint to increase cache capacity and reduce cache writes to improve durability [14], [15], [16]. CAFTL [14] is the first method to integrate the deduplication component in the SSD to alleviate the flash wear problem. It implements a deduplication module at the File Translation Layer (FTL). Nitro [13] and DEC [34] propose using a combination of deduplication and compression to reduce the amount of data in the flash cache. To reduce write amplification, Nitro organizes cached data according to the size of the SSD write unit. Other work [35], [36] have also studied deduplication of flash cache in virtualized environments. They found that due to the high degree of integration of virtual machines, flash cache deduplication could reduce a large number of duplicate blocks in the cache device.

However, unlike our work, none of these methods consider data replacement algorithms when integrating deduplication to the SSD cache. The only work focused on the deduplication-aware replacement algorithm is CacheDedup [12]. It proposes a novel architecture that separates the Metadata Cache and Data Cache and manages them separately. Based on this architecture, it proposes two deduplication-aware cache

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on August 24,2020 at 17:28:02 UTC from IEEE Xplore. Restrictions apply.

replacement algorithms, D-LRU and D-ARC, to further improve the cache hit ratios and reduce cache writes. Our proposed CDAC is a content-driven caching method, focusing on providing content-driven data replacement algorithms. Unlike CacheDedup, CDAC fully considers data sharing and content popularization caused by deduplication when identifying hot/cold blocks, which will greatly improve cache utilization and cache hit rate.

In addition to deduplication, some studies have proposed other methods to optimize flash cache. Flashied [37] and HEC [6] have studied to not cache data with weak temporal locality in the flash cache. Pannier [29] have proposed to reduce write amplification by caching frequently read and infrequently updated objects. In addition, some work directly changed the flash memory device. For example, Duracache [38] attempts to extend the life of SSD cache by dynamically increasing the error correction capability of flash memory devices; Shen *et al.* [39] allows the cache to map keys directly to the device itself, thereby eliminating the overhead of the flash garbage collector.

6 Conclusion

Motivated by the fact that the existing deduplication-aware cache algorithms, D-ARC and D-LRU, are not able to improve the cache hit ratios and IOPS adequately, we propose CDAC, a Content-driven Deduplication-Aware Caching management approach to significantly improve the performance of deduplication-based SSD caches. CDAC focuses on mining data blocks content redundancy and exploiting the intensity of content sharing among source addresses in cache management strategies. It consists of two complementary optimization techniques, Reference-Count based Eviction (RCE) and Bitmap based Hotness Identification (BHI), which are combined to leverage the intensity of content sharing and hotness in the cache replacement algorithm. Our extensive experimental results showed that CDAC significantly improves cache hit ratios and IOPS of the state-of-the-art deduplication-aware cache algorithms, D-ARC and D-LRU, driven by real-world datasets.

Acknowledgments

The authors were very grateful to Prof. Ming Zhao for providing them with CacheDedup Prototype and many instructive comments. This work was supported by grants from Open Project Program of Wuhan National Laboratory for Optoelectronics under Grant 2019WNLOKF009, Fundamental Research Funds for the Central Universities under Grant 2019CDJGFJSJ001, National Natural Science Foundation of China under Grant 61402061, 61672116, and 61802038, Chongqing High-Tech Research Program under Grant cstc2016jcyjA0274 and cstc2016jcyjA0332, China Postdoctoral Science Foundation under Grant 2017M620412, and Chongqing Postdoctoral Special Science Foundation under Grant XmT2018003.

References

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, “Design tradeoffs for SSD performance,” in *Proc. USENIX Annu. Tech. Conf.*, 2008, pp. 57–70. [Online]. Available: http://dl.acm.org/citation.cfm?id=1404014.1404019

[2] Q. Yang and J. Ren, “I-cash: Intelligently coupled array of SSD and HDD,” in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Architecture*, 2011, pp. 278–289. [Online]. Available: http://dl.acm.org/citation.cfm?id=2014698.2014865

[3] S. Huang, Q. Wei, D. Feng, J. Chen, and C. Chen, “Improving flash-based disk cache with lazy adaptive replacement,” *Trans. Storage*, vol. 12, no. 2, pp. 8:1–8:24, Feb. 2016. [Online]. Available: http://doi.acm.org/10.1145/2737832

[4] T. Luo, S. Ma, R. Lee, X. Zhang, D. Liu, and L. Zhou, “S-cave: Effective SSD caching to improve virtual machine storage performance,” in *Proc. 22nd Int. Conf. Parallel Architectures Compilation Techn.*, 2013, pp. 103–112. [Online]. Available: http://dl.acm.org/citation.cfm?id=2523721.2523739

[5] P. L. Suei, M. Y. Yeh, and T. W. Kuo, “Endurance-aware flash-cache management for storage servers,” *IEEE Trans. Comput.*, vol. 63, no. 10, pp. 2416–2430, Oct. 2014.

[6] J. Yang, N. Plasson, G. Gillis, N. Talagala, S. Sundararaman, and R. Wood, “HEC: Improving endurance of high performance flash-based cache devices,” in *Proc. 6th Int. Syst. Storage Conf.*, 2013, pp. 10:1–10:11. [Online]. Available: http://doi.acm.org/10.1145/2485732.2485743

[7] S. Byan, J. Lentini, A. Madan, and L. Pabn, “Mercury: Host-side flash caching for the data center,” in *Proc. IEEE 28th Symp. Mass Storage Syst. Technol.*, 2012, pp. 1–12.

[8] D. A. Holland, E. Angelino, G. Wald, and M. I. Seltzer, “Flash caching on the storage client,” in *Proc. USENIX Conf. Annu. Tech. Conf.*, 2013, pp. 127–138. [Online]. Available: http://dl.acm.org/citation.cfm?id=2535461.2535477

[9] F. Meng, L. Zhou, X. Ma, S. Uttamchandani, and D. Liu, “vCacheShare: Automated server flash cache space management in a virtualization environment,” in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2014, pp. 133–144. [Online]. Available: http://dl.acm.org/citation.cfm?id=2643634.2643649

[10] M. Saxena, M. M. Swift, and Y. Zhang, “FlashTier: A lightweight, consistent and durable storage cache,” in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, 2012, pp. 267–280. [Online]. Available: http://doi.acm.org/10.1145/2168836.2168863

[11] L. Tang, Q. Huang, W. Lloyd, S. Kumar, and K. Li, “RIPQ: Advanced photo caching on flash for facebook,” in *Proc. 13th USENIX Conf. File Storage Technol.*, 2015, pp. 373–386. [Online]. Available: http://dl.acm.org/citation.cfm?id=2750482.2750510

[12] W. Li, G. Jean-Baptise, J. Riveros, G. Narasimhan, T. Zhang, and M. Zhao, “CacheDedup: In-line deduplication for flash caching,” in *Proc. 14th Usenix Conf. File Storage Technol.*, 2016, pp. 301–314.

[13] 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 Conf. USENIX Annu. Tech. Conf.*, 2014, pp. 501–512.

[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. Stroage Techno.*, 2011, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1960475.1960481

[15] J. Kim *et al.*, “Deduplication in SSDs: Model and quantitative analysis,” in *Proc. IEEE 28th Symp. Mass Storage Syst. Technol.*, 2012, pp. 1–12.

[16] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam, “Leveraging value locality in optimizing NAND flash-based SSDs,” in *Proc. 9th USENIX Conf. File Stroage Technol.*, 2011, pp. 7–7. [Online]. Available: http://dl.acm.org/citation.cfm?id=1960475.1960482

[17] W.-T. Huang, C.-T. Chen, Y.-S. Chen, and C.-H. Chen, “A compression layer for NAND type flash memory systems,” in *Proc. 3rd Int. Conf. Inf. Technol. Appl.*, 2005, pp. 599–604.

[18] T. Makatos, Y. Klonatos, M. Marazakis, M. D. Flouris, and A. Bilas, “Using transparent compression to improve SSD-based I/O caches,” in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/1755913.1755915

[19] K. S. Yim, H. Bahn, and K. Koh, “A flash compression layer for smartmedia card systems,” *IEEE Trans. Consum. Electron.*, vol. 50, no. 1, pp. 192–197, Feb. 2004. [Online]. Available: http://dx.doi.org/10.1109/TCE.2004.1277861

[20] P. Gallagher and A. Director, “Secure hash secure hash standard (shs),” in *FIPS PUB*, vol. 180, p. 183, 1995.

[21] B. Zhu, K. Li, and H. Patterson, “Avoiding the disk bottleneck in the data domain deduplication file system,” in *Proc. 6th USENIX Conf. File Storage Technol.*, 2008, Art. no. 18.

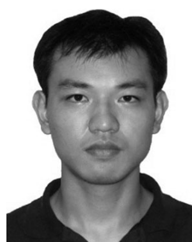
Authorized licensed use limited to: University of Texas at Arlington. Downloaded on August 24,2020 at 17:28:02 UTC from IEEE Xplore. Restrictions apply.

[22] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Campbell, "Sparse indexing: Large scale, inline deduplication using sampling and locality," in *Proc. 7th Conf. File Storage Technol.*, 2009, pp. 111–123.
[23] K. Srinivasan, T. Bisson, G. Goodson, and Y. Voruganti, "iDedup: Latency-aware, inline data deduplication for primary storage," in *Proc. 10th USENIX Conf. File Storage Technol.*, 2012, pp. 1–14.
[24] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li, "Decentralized deduplication in SAN cluster file systems," in *Proc. Conf. USENIX Annu. Tech. Conf.*, 2009, pp. 101–113.
[25] K. Jin and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," in *Proc. Israeli Exp. Syst. Conf.*, 2009, pp. 7:1–7:12. [Online]. Available: http://doi.acm.org/10.1145/1534530.1534540
[26] D. Meister and A. Brinkmann, "Multi-level comparison of data deduplication in a backup scenario," in *Proc. SYSTOR Israeli Exp. Syst. Conf.*, 2009, Art. no. 8.
[27] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage," in *Proc. 1st USENIX Conf. File Storage Technol.*, 2002, pp. 89–101.
[28] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in *Proc. 2nd USENIX Conf. File Storage Technol.*, 2003, pp. 115–130. [Online]. Available: http://dl.acm.org/citation.cfm?id=1090694.1090708
[29] C. Li, P. Shilane, F. Douglis, and G. Wallace, "Pannier: A container-based flash cache for compound objects," in *Proc. 16th Annu. Middleware Conf.*, 2015, pp. 50–62. [Online]. Available: http://doi.acm.org/10.1145/2814576.2814734
[30] "Gluster filesystem," [Online]. Available: https://www.gluster.org/
[31] R. Koller and R. Rangaswami, "I/O deduplication: Utilizing content similarity to improve I/O performance," in *Proc. Usenix Conf. File Storage Technol.*, 2010, Art. no. 13.
[32] B. Hong, D. Plantenberg, D. D. E. Long, and M. Sivan-Zimet, "Duplicate data elimination in a SAN file system," in *Proc. IEEE Conf. Mass Storage Syst. Technol.*, 2004, pp. 301–304.
[33] C. A. Waldspurger, "Memory resource management in VMware ESX serve," *ACM SIGOPS Operating Syst. Rev.*, vol. 36, pp. 181–194, 2002.
[34] Z. Han, X. Wen, Y. Hu, F. Dan, and G. Liang, "DEC: An efficient deduplication-enhanced compression approach," in *Proc. IEEE 22nd Int. Conf. Parallel Distrib. Syst.*, 2016, pp. 519–526.
[35] X. Chen, W. Chen, Z. Lu, P. Long, S. Yang, and Z. Wang, "A duplication-aware SSD-based cache architecture for primary storage in virtualization environment," *IEEE Syst. J.*, vol. 11, no. 4, pp. 2578–2589, Dec. 2017.
[36] J. Feng and J. Schindler, "A deduplication study for host-side caches in virtualized data center environments," in *Proc. IEEE Symp. Mass Storage Syst. Technol.*, 2013, pp. 1–6.
[37] A. Eisenman *et al.*, "Flashield: A key-value cache that minimizes writes to flash," 2017, *arXiv:1702.02588*.
[38] R. S. Liu, C. L. Yang, C. H. Li, and G. Y. Chen, "Duracache: A durable SSD cache using MLC NAND flash," in *Proc. Des. Autom. Conf.*, 2013, pp. 1–6.
[39] Z. Shen, F. Chen, Y. Jia, and Z. Shao, "Optimizing flash-based key-value cache systems," in *Proc. Usenix Conf. Hot Topics Storage File Syst.*, 2016, pp. 46–50.

Yujuan Tan received the BSc degree in computer science and engineering from Hunan Normal University, Changsha, China, in 2006, and the PhD degree in computer science and engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2012. She is currently a full professor with the College of Computer Science in Chongqing University, Chongqing, China. Her research interests include hybrid memory system, flash cache, and data deduplication.

Congcong Xu is working toward the master's degree majoring in computer architecture at Chongqing University, Chongqing, China. His current research interests include data deduplication and flash cache.

Jing Xie is working toward the master's degree majoring in computer architecture at Chongqing University, Chongqing, China. His current research interests include data deduplication, and flash cache.

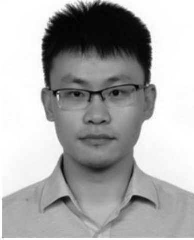


Zhichao Yan received the first PhD degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 2012, and the second PhD degree in computer engineering from the University of Texas at Arlington, in 2018. His research interests include flash SSD, data deduplication, and cloud storage.

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, Canada, in 1987, and the PhD degree in computer science from the Texas A&M University, College Station, Texas, in 1991. He is currently chair and Wendell H. Nedderman endowed professor of Computer Science and Engineering Department, University of Texas at Arlington. Prior to joining UTA, he has served as a program director at National Science Foundation (2013–2015) and he was at University of Nebraska-Lincoln since 1991, where he was 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 also supervised 15 post-doctoral fellows and visiting scholars. His present research interests include computer architecture, computer storage systems and parallel I/O, high-performance computing, big data computing, cloud computing, and performance evaluation. He recently served as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems*. He has more than 250 publications in major journals and international Conferences in these areas, including the *IEEE Transactions on Parallel 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, LISA, SIGMETRICS, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, INFOCOM, ICPP, etc., and his research has been supported by NSF, DOD, and industry. He is a member of the ACM.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on August 24,2020 at 17:28:02 UTC from IEEE Xplore. Restrictions apply.

Witawas Srisa-an received the PhD degree in computer science from the Illinois Institute of Technology, in 2002. He is currently an associate professor of Computer Science and Engineering, University of Nebraska-Lincoln. His research interests include the area of programming languages, software engineering, operating systems, runtime systems, and security.



Xianzhang Chen received the BS and MS degrees from the School of Computer Science and Engineering, Southeast University, Nanjing, China, and the PhD degree from the College of Computer Science, Chongqing University, in 2017. He is currently a research assistant with Chongqing University. His research interests include non-volatile memory-based file systems, memory management, and in-memory databases.

Duo Liu received the BE degree in computer science from the Southwest University of Science and Technology, Sichuan, China, in 2003, the ME degree from the Department of Computer Science, University of Science and Technology of China, Hefei, China, in 2006, and the PhD degree in computer science from the Hong Kong Polytechnic University, in 2012. He is a tenured associate professor with the College of Computer Science, Chongqing University, China. His current research interests include emerging nonvolatile memory (NVM) techniques for embedded systems, memory/storage management in mobile systems, and hardware/software co-design. He has served as program committee for multiple international conferences, and as reviewer for several ACM/IEEE journals and transactions.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on August 24,2020 at 17:28:02 UTC from IEEE Xplore. Restrictions apply.