CDAC: Content-Driven Deduplication-Aware Storage Cache

Yujuan Tan, Jing Xie, Congcong Xu
Chongqing University

Zhichao Yan
HP

Hong Jiang
University of Texas Arlington

Yajun Zhao, Min Fu
Sangfor

Xianzhang Chen, Duo Liu
Chongqing University

Wen Xia
Harbin Institute of Technology

Abstract

Data deduplication, as a proven technology for effective data reduction in backup and archive storage systems, also demonstrates the promise in increasing the logical space capacity of storage caches by removing redundant data. However, our in-depth evaluation of the existing deduplication-aware caching algorithms reveals that they do improve the hit ratios compared to the caching algorithms without deduplication, especially when the cache block size is set to 4KB. But when the block size is larger than 4KB, a clear trend for modern storage systems, their hit ratios are signif cantly reduced. A slight increase in hit ratios due to deduplication may not be able to improve the overall storage performance because of the high overhead created by deduplication.

To address this problem, in this paper we propose CDAC, a Content-driven Deduplication-Aware Cache, which focuses on exploiting the blocks' content redundancy and their 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 19.49X in read cache hit ratio, with an average of 1.95X under real-world traces when the cache size ranges from 20% to 80% of the working set size and the block size ranges from 4KB to 64 KB.

1 Introduction

Due to the exceptional performance relative to hard drive disks (HDDs), solid state drives (SSDs) have been widely adopted as a storage cache to boost the storage performance in large-scale HDD-based primary storage systems [1, 28, 7, 16, 25, 27, 2, 6, 20, 23, 26]. 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 [14, 12, 3, 10, 5] or compression techniques [8, 17, 29] to effectively increase the cache logical capacity by reducing data footprints.

Data deduplication focuses on identifying and removing redundant data to reduce data footprints. It uses an appropriate hash algorithm [21] to generate a unique content-based identif er, commonly referred to as a data f ngerprint, for each data unit (e.g., a f le or data chunk). However, due to the high cost in generating f ngerprints and their use for uniqueness identif cation [30, 15], deduplication is less popular in performance-sensitive primary storage systems [24, 4, 9] than in backup and archival storage systems [19, 22] 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 identif cation and elimination operations lay on the data read/write critical path. That is, the duplicate data are identif ed and removed before writing to the SSD caches. The deduplication overhead, in terms of the time spent on generating the data f ngerprints 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 reap the benef ts of increased logical capacity and cache hit ratios without paying a high price for the deduplication-induced overhead, thus improving the overall storage performance.

A recent study, CacheDedup [14], addressed this problem by proposing two deduplication-aware cache replacement algorithms, D-LRU and D-ARC, which decomposed metadata from the data in the cache to enhance the performance of deduplication-based SSD caches. This is the only published work addressing the problem so far, to the best of our knowledge. Our in-depth evaluation of D-LRU and D-ARC reveals that they do improve the hit ratios compared to LRU and ARC algorithms [18], especially when the block size is set to 4KB. However, as the block size increases, the hit ratios of D-ARC and D-LRU are signif cantly reduced due to the lower deduplication ratio and less expansion of the logical capacity. Compared to the OPT algorithm, they have a growing gap in hit ratios when the cache size is f xed. A slight increase in hit ratios of D-ARC and D-LRU relative to LRU and ARC may not be able to improve the overall storage performance due to the high overhead in the deduplication process.

The reason for the ineff cient cache design in CacheDedup [14] can be explained in part by the fact that data deduplication changes how blocks are cached and evicted signif cantly, thus their locality properties. *In conventional caches, each block is identif ed by a unique source address*; the source addresses of all blocks are independent of one another. But *with data deduplication, each block is identif ed 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 further renders effective in deduplication-based caches, compared to any conventional cache replacement algorithm that treats each source address independently and replaces the cached blocks based on the access locality of each independent source address. However, existing deduplication-aware cache algorithms, like D-LRU and D-ARC, using only the access time and frequency of source addresses as a hint for block replacement, missed the opportunity to effectively leverage the intensity of content redundancy and sharing in caching replacement algorithms.

Based on these observations, in this paper we propose CDAC, a Content-driven Deduplication-Aware Caching management approach, which focuses on exploiting 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 Identif cation (BHI). In particular, RCE focuses on evicting a cold block based on its reference count [13], 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 f ner-grained access patterns to parts of the large 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 preserves more valid data in the cache to some extent, improving cache space utilization by avoiding the false-positive identif cation of hot blocks, in which one or a few tiny parts of a block being "hot" render that entire block "hot". The combination of BHI and RCE enables the cache replacement algorithm to fully and accurately exploit the content sharing intensity and hotness.

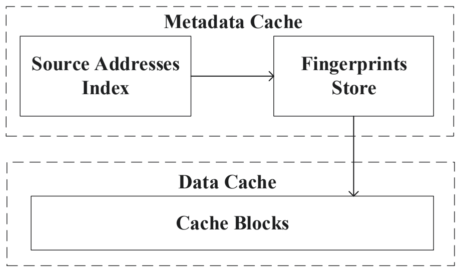


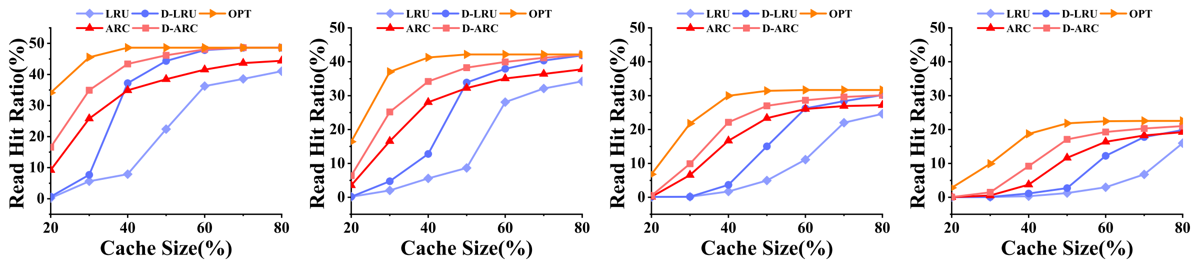
Figure 1: Architecture of CacheDedup.

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 and Section 5 concludes the paper.

2 Background and Motivation

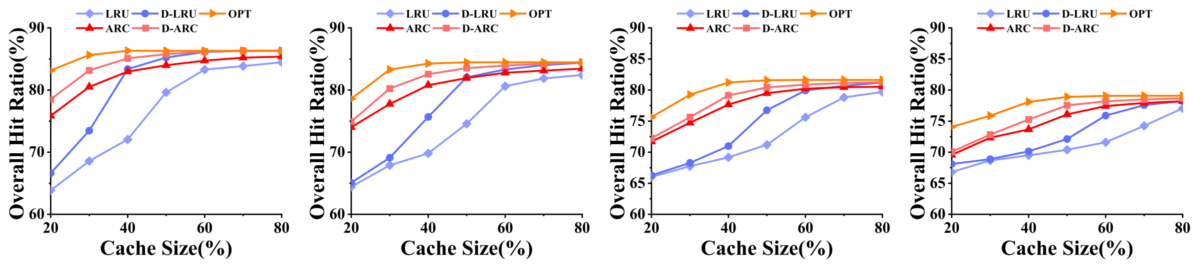
CacheDedup is the f rst and only known work to address the issue of deduplication-aware cache management, to the best of our knowledge. It proposed an architecture using separate Data Cache and Metadata Cache to integrate the data caching and deduplication metadata caching, as illustrated in Figure 1. Data Cache stores the cached data blocks, and Metadata Cache stores the source addresses and data f ngerprints of these blocks. Based on this architecture, two deduplication-aware caching replacement algorithms, D-LRU and D-ARC, are designed based on the traditional LRU and ARC algorithms. D-LRU and D-ARC respectively perform deduplicatation on the cached blocks, and separately manage Data Cache and Metadata Cache according to the corresponding LRU and ARC algorithms, respectively. Moreover, since Data Cache and Metadata Cache are separately managed and accessed, the blocks stored in Data Cache and their corresponding deduplication metadata stored in Metadata Cache do not need to be fetched in or evicted out synchronously.

D-LRU and D-ARC can increase the cache's logical capacity by removing redundant cacheblocks, thereby improving the cache hit ratio to a certain extent. However, such improvements are limited due to their inability to explore the cache blocks' intensity of content redundancy and content sharing. In this section, we will investigate D-LRU and D-ARC on empirical evidence from extensive experiments and uncover their limitations. In our experiments, we created a practical CacheDedup prototype and replayed the WebVM trace. Its statistical characteristics are detailed in Section 4. We implemented LRU, D-LRU, ARC, and D-ARC algorithms respectively.



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

Figure 2: Read Hit Ratio from WebVM



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

Figure 3: Overall Hit Ratio from WebVM

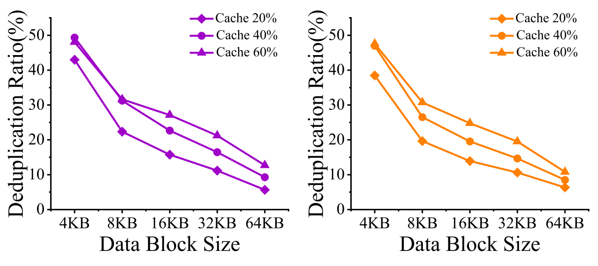
In addition, we also implemented the theoretically optimal cache replacement (OPT) algorithm as a reference.

2.1 Hit Ratio

Figure 2 and Figure 3 show the read hit ratios and overall hit ratios of ARC, D-ARC, LRU, D-LRU and OPT when the cache size ranges from 20% to 80% of the working set size. It can be seen from the results that both D-ARC and D-LRU improve the read hit ratios and overall hit ratios of ARC and LRU, respectively.

However, as the size of the cache block increases, the hit ratios of both D-ARC and D-LRU are signif cantly reduced. First, the increase in the hit ratios of D-ARC and D-LRU relative to ARC and LRU becomes smaller. For example, when the block size is 4KB, the read hit ratios of D-ARC and D-LRU is 6.91% and 11.85% higher than ARC and LRU on average, respectively; but when the block size grows to 8KB, 16KB and 32KB, the read hit ratios of D-ARC are only 5.38%, 3.00% and 2.65% higher than ARC on average, and D-LRU's read hit ratios are 8.70%, 5.58% and 3.77% higher than LRU's on average. Second, as the block size increases, the hit ratios of D-ARC and D-LRU become lower than that of OPT when the cache space is f xed. Taking the cache size as 40% of the working set size for example, when the block size is 4KB, the read hit ratios of D-LRU and D-ARC are 76.5% and 89.2% of that of OPT; but when the block size grows to 8KB, 16KB and 32KB, the read hit ratios of D-LRU are only 31%, 12.3% and 6% of that of OPT, and D-ARC's read hit ratios are only 82.9%, 73.9% and 49% of OPT's. Moreover, when the block size increased, D-ARC and D-LRU require more cache space to bring their hit ratios close to that of OPT. When the block size is 4K-B, D-ARC and D-LRU's read hit ratios are close to that of OPT when the cache size is 50% of the working set size. However, when the block size is 8KB and 16KB, to make the read hit ratio of D-ARC and D-LRU close to that of OPT, the cache space size needs to reach 70% and 80% of the working set size respectively.

Based on these observations, it is found that D-ARC and D-LRU can improve the hit ratios by removing duplicate data to increase the logical capacity of the cache. They work well when the size of the cache blocks is 4K-B. But when the block size increases, the benef ts of deduplication become limited and their hit ratios decreases signif cantly, greatly reducing the overall storage performance.



(a) D-LRU (b) D-ARC

Figure 4: Deduplication Ratio

2.2 Deduplication Ratio

To better understand the ineff ciency of D-ARC and D-LRU, we measured their deduplication ratios. The d-eduplication ratio here is def ned by the amount of data blocks not written to SSD due to deduplication divided by the total amount of processed data.

Figure 4 shows the deduplication ratios of D-ARC and D-LRU. As shown in Figure 4, when the block size increases from 4KB to 64KB, the deduplication ratios are signif cantly reduced. When the block size is large, fewer identical blocks can be found, so the deduplication ratios are lower. Further, the redundant data to be eliminated becomes less, and the logical capacity of the cache becomes small. As a result, the cache hit ratios will become low. It also can be seen from the results shown in Figure 2 and Figure 3, as the block size increases, the increased cache hit ratios of D-ARC and D-LRU relative to ARC and LRU becomes smaller. When the block is large, a slight increase in hit ratio is not able to significantly improve the cache hit ratios and overall storage performance.

As shown in Figure 4, we can conclude that when the block size gets larger, the deduplication ratios decrease; which directly lead to a lower cache hit ratio. Therefore, in order to improve the cache hit ratios and storage system performance, we propose CDAC, a content-driven deduplication-aware caching management approach, which will be described in detail in the next section.

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 Identif cation (BHI). Both are designed based on the architecture of CacheDedup shown in Figure 1.

According to the architecture, there are two caches, Metadata Cache and Data Cache; Metadata Cache stores the source addresses of the data blocks stored in Data Cache; a free block in the Data Cache means that there is no source address in the metadata cache pointing to it. 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.

Table 1: Variable Def nition

Symbol	Def nition
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 L-RU position
S_L	The source address in the LRU position in M
$Flags_{S_i}$	A f ag that is used to indicate block $f(S_i)$ that has been accessed in the current cycle for BHI
P_{S_i}	The percentage of the small parts that have been accessed for S_i

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, in terms of 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 being pointed to by more source addresses implies that more data read requests will likely be directed to this block to make it a hotter target, which helps increase cache hit ratios and improve the storage performance.

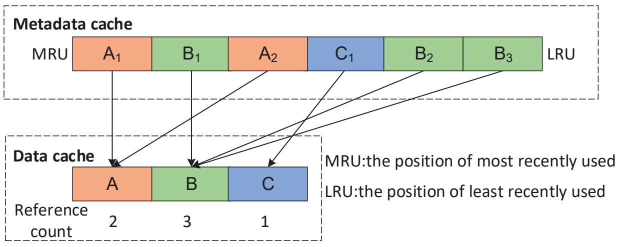


Figure 5: The working process of RCE.

However, while reference-count based eviction can help produce more cache hit ratios, using reference counting as the only hint to f nd the block to be replaced is not suff ciently effective, unless it is jointly considered with access temporal locality of the associated source addresses. Taking Figure 5 for example, there are three blocks in Data Cache, A, B and C; Block A is referenced 2 times; Block B is referenced 3 times; Block C is referenced once. While using the referenced count to identify the hot/cold blocks, Block B is the hottest and Block C is the coldest; Block C and its associated source address C_1 will be removed to make room for new blocks. However, if block C is to be accessed in the near future and Block B will not be accessed for a long time, replacing Block C will reduce the cache hit ratios and storing Block B wastes both the space of Data Cache(i.e., storing block B) and Metadata Cache (storing three source addresses of Block B, B_1, B_2 and B_3).

To address this problem, RCE takes both reference counts and access locality into consideration to f nd the free blocks and associate source addresses to be deleted. RCE has two basic assumptions: f rst, the source address of a highly referenced data block is likely to be accessed again in the near future; second, the source address of the LRU position located in Metadata Cache may no longer be accessed recently. Based on these two assumption-s, RCE only focuses on the source addresses in the L-RU position and divides the data blocks pointed to by these source addresses into two categories: one is the da-ta 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 ob-serve how 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 ad-dress to go from the MRU position to the LRU position. If the blocks reference count is signif cantly reduced in the next cycle, the source address will be deleted; other-wise it will be retained and go to next cycle.

Algorithm 1: RCE pseudocode

```
Remarks: Each time a new source address $S_i$ enters into $M$, $C_{S_i}$ is set to be 0;
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
        Value = $(C_{S_i}$ - $C_{B_i})/C_{S_i}$;
        if $Value \leq \lambda$ then
            $C_{S_i} = C_{B_i}$;
            Move $S_i$ to the MRU location in $M$;
        else
            Remove $S_i$ to the delete queue;
    else
        $C_{S_i} = C_{B_i}$;
        $C_{B_i} = C_{B_i}$ - 1;
        Move $S_i$ to the MRU location in $M$;
else
    Move $S_i$ to the delete queue;
```

Algorithm 1 shows the pseudocode of RCE. Table 1 def nes the corresponding symbols.It uses C_{B_i} to repre-sent the number of the source addresses pointing to block B_i, and C_{S_i} to represent the number of the source address-es pointing to block $f(S_i)$ when S_i is last located at the LRU position. 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 f rst time to reach the LRU position, and C_{B_i} is greater than 1, meaning that more than one source address point-ing 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 L-RU position again, it will use the formula (C_{S_i} - C_{B_i})/C_{S_i} to quantify how C_{B_i} changes during this period. If (C_{S_i} - C_{B_i})/C_{S_i} is greater than the preset threshold λ, it means that C_{B_i} is greatly reduced and block B_i is no longer hot, and S_i will be removed to the delete queue to be deleted; otherwise if (C_{S_i} - C_{B_i})/C_{S_i} is lower than or equal to the preset threshold λ or C_{B_i}) increases beyond C_{S_i}, it mean-s that block B_i is very hot, then S_i will be retained and moved to the MRU position.

3.2 Bitmap based Hotness Identif cation

In storage caches, the block size is f xed and all requests need to be aligned to the cache's block size. In conven-tional cache replacement algorithms, a block is identif ed as hot or cold completely determined by the access fre-quency or the last access time of its source address, re-gardless of the valid content for each access. For exam-ple, there are two cached blocks, A and B, with a block size of 4KB; if block A is accessed before block B, block B will be identif ed as hotter than block A, even if block B only accesses 1KB of data, and Block A accesses 4KB 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 1KB 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.

To solve this problem, CDAC identif es hot/cold blocks based on f ner-grained access patterns. It breaks a block 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 multi-ple individual parts of a block makes it possible to more accurately identify the content hotness of the block, e-specially for large blocks, minimizing false-positive hot block identif cations.

Based on the architecture shown in Figure 1, BHI di-vides the address space of each source address into mul-tiple small parts and uses bitmaps to record the access status of each part. If one part is accessed, the corre-

```
Algorithm 2: BHI pseudocode
Remarks: Each time some part of the source address $S_i$ is
accessed, $Flag_{S_i}$ is set to be 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 then
    if $P_{S_i} \geq \alpha$ And $Flag_{S_i}$ = 1 then
        $Flag_{S_i} = 0$;
        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;
```

sponding position in the bitmap is set to 1, otherwise it is set to 0. In addition, BHI adds a f ag to indicate whether the source address was accessed in a cycle. Algorithm 2 shows the pseudocode of BHI. For each source address S_i in the LRU location in M, BHI f rst 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 source address S_i is accessed during this period (i.e., $Flag_{S_i}$ is equal to 1), the source address S_i will be retained and moved the MRU position and $Flag_{S_i}$ will be reset to 0. Otherwise S_i will be identif ed 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.

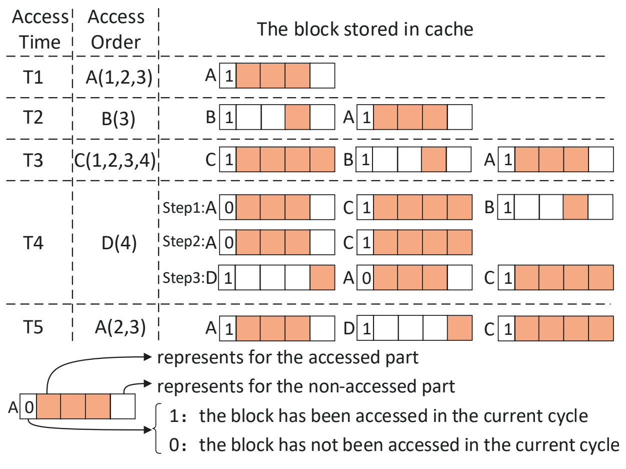

Access Time
Access Order
The block stored in cache
T1
A(1,2,3)
T2
B(3)
T3
C(1,2,3,4)
T4
D(4)
Step1:A
Step2:A
Step3:D
T5
A(2,3)
represents for the accessed part
represents for the non-accessed part
1: the block has been accessed in the current cycle
0: the block has not been accessed in the current cycle

Figure 6: The working process of BHI.

Figure 6 presents an example to illustrate how BHI works. In this example, the cache can only hold 3 blocks, and the percentage threshold for access portion is set to 50%. 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, T2 and T3, block A, B and C enter the cache; their f ags are set to 1. At time T4, Block D needs to enter the cache but there is no free blocks. So BHI f rst checks the access status of block A at the LRU position. Because the access portion of block A exceeds 50% and its f ag is 1, block A will not be deleted; it moves to the MRU position and its f ag is reset 0. Then BHI checks block B. While for block B, since its access portion is only 25%, it is deleted. So after time T4, block D enters the MRU location in the cache, block B is deleted, and block A's f ag is set to 0. At time T5, when accessing A again, block A moves to the MRU position and its f ag is set to 1 again.

3.3 CDAC: combining RCE and BHI together

When the cache is full, CDAC combines RCE and BHI together to f nd free blocks. It f rst 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. 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, as quantitatively evidenced in Section 4.

4 Experimental Evaluation

In this section, we will assess the benef ts 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. During experiments, we replace D-LRU and D-ARC with CDAC-LRU and CDAC-ARC in the CacheDedup prototype to measure CDAC's performance. At the same time, we also implemented LRU and ARC, which are used as baseline non-deduplication-aware caching algorithms to highlight the benef ts of data deduplication brought about by C-DAC.

(2)Experimental Workload. We replayed the public FIU traces [11] in our experiments. These traces were collected from a VM hosting the departmental websites for webmail and online course management (WebVM), a f le server used by a research group (Homes), and a departmental mail server (Mail). Table 2 shows the statistical characteristics of all 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

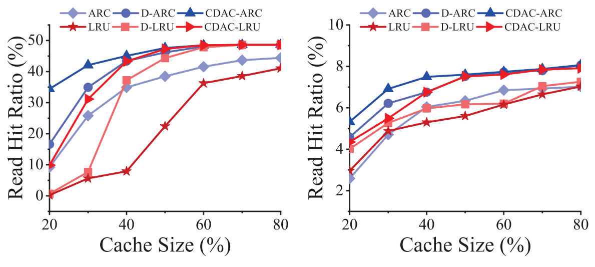
Table 2: Trace statistics

4.2 Performance

We evaluate the cache performance for each dataset with different cache sizes from 20% to 80% of the working set size, and different block sizes from 4KB to 64KB. For FIU traces, since the public dataset has only 4KB sized request, we merged the requests with consecutive source addresses into larger sized request and generate corresponding new f ngerprints. In CDAC-ARC and CDAC-LRU, We set the threshold λ in RCE to 50%, the threshold α in BHI to 50%, and the size of the smallest part of each block in BHI to 4KB. In this section, we will show the performance results for these conf gurations.

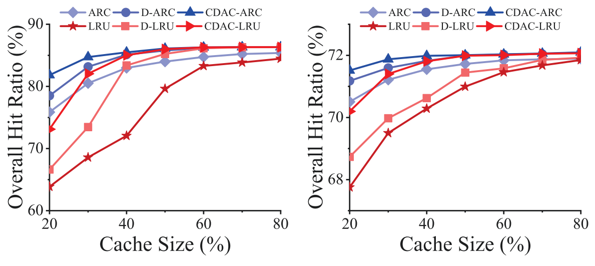
Cache hit ratio is a key metric to measure the caching eff ciency. Here we mainly show the overall hit ratio and the read hit ratio to evaluate CDAC's benef ts. We did not show write hit ratios because the write requests in these datasets are very intensive and all of the cache replacement algorithms handle them well.

Figure 7 and Figure 8 show the cache hit ratio for WebVM and Homes traces with a block size of 4KB. It is worth noting that when the block size is 4KB, BHI will not work since the smallest part of each block is set to 4KB. Therefore, the results of CDAC shown in Figure 7 and Figure 8 contain only the performance results of RCE. From these results, it can be seen that CDAC-LRU and CDAC-ARC signif cantly improve the read hit ratios for all the traces, which 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. Figure 9 and Figure 10 show the read hit ratios and overall hit ratios for WebVM and Mail traces when the block size is from 8KB to 64KB. 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 their corresponding baseline approaches. By analyzing these results, we can conclude that these



(a) WebVM (b) Homes

Figure 7: Read hit ratio with 4KB block size.



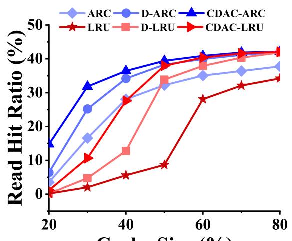
(a) WebVM (b) Homes

Figure 8: Overall hit ratio with 4KB block size.

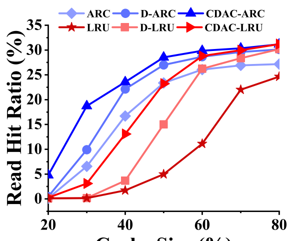
improvements to the cache hit ratios have three characteristics.

First, CDAC's performance improvement in read hit ratios is greater than overall hit ratios. There are two reasons: at f rst, 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; secondly, for all the datasets, the proportion of write requests is much higher than read requests (see Table 2), so the read hit ratios contribute much less to the overall hit ratios than the write hit ratios. Therefore, CDAC does not signif cantly improve the overall hit ratios as it does for read hit ratios.

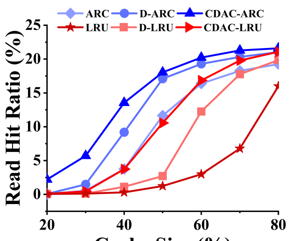
Second, as the block size increases, the amount of cache space required for maximum improvement in CDAC increases. Taking the read hit ratio as an example, for the WebVM trace, when the block size is 8KB, CDAC-LRU has the greatest improvements to the baselines when the cache size is 40%; but when the block size is increased to 32KB and 64KB, CDAC achieves the greatest improvements on the baselines when the cache size is 50%, 60%, respectively. This is because as the block size increases, the percentage of valid content per data block may decrease in each access, resulting in lower cache utilization; 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



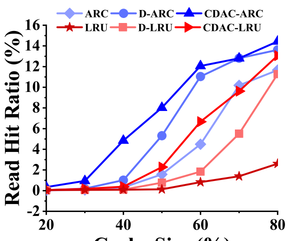
(a) Block Size: 8KB



(b) Block Size: 16KB

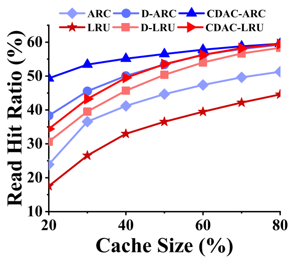


(c) Block Size: 32KB

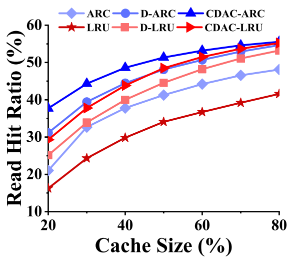


(d) Block Size: 64KB

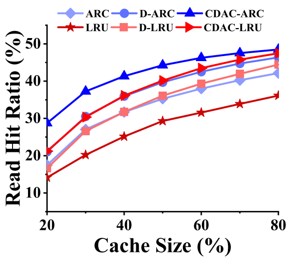
(A)WebVM



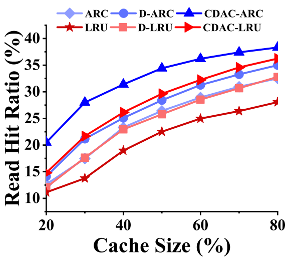
(e) Block Size: 8KB



(f) Block Size: 16KB



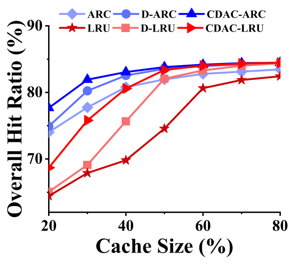
(g) Block Size: 32KB



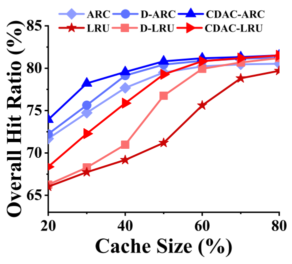
(h) Block Size: 64KB

(B)Mail

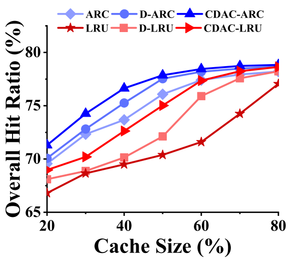
Figure 9: Read hit ratio of WebVM and Mail



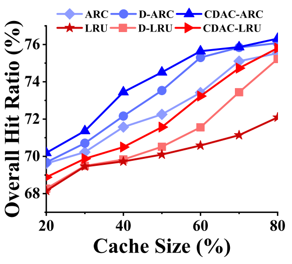
(a) Block Size: 8KB



(b) Block Size: 16KB

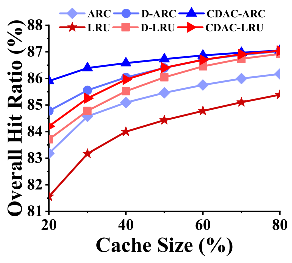


(c) Block Size: 32KB

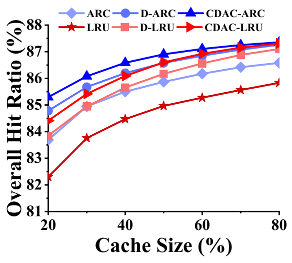


(d) Block Size: 64KB

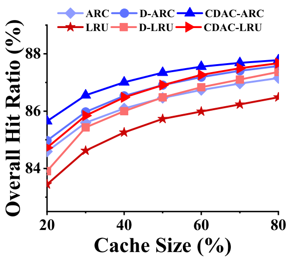
(A)WebVM



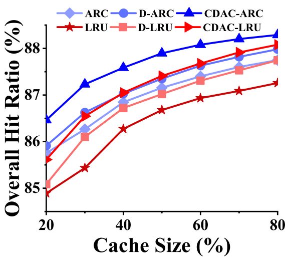
(e) Block Size: 8KB



(f) Block Size: 16KB



(g) Block Size: 32KB



(h) Block Size: 64KB

(B)Mail

Figure 10: Overall hit ratio of WebVM and Mail

achieve better performance improvements. But if the cache size exceeds a certain amount, the cache replacement algorithm is less eff cient and the contribution of CDAC will be smaller.

Third, CDAC's overall performance advantages over the baselines become more pronounced as the block size increases. 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 benef t is further magnif ed by larger blocks. But for D-LRU, D-ARC, LRU and ARC, when the block size increases, it becomes more diff cult for them to f nd the redundant blocks and accurately identify the hot/cold blocks, leading to lowered overall cache hit ratios.

5 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 adequately, we propose CDAC, a Content-driven Deduplication-Aware Caching management approach to signif cantly increase the performance of deduplication-based SSD caches. C-DAC focuses on mining data blocks' content redundancy and exploiting their 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 Identif cation (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 of the state-of-the-art deduplication-aware cache algorithms, D-ARC and D-LRU, driven by real-world datasets.

Acknowledgment

We are very grateful to Professor Ming Zhao for providing us with CacheDedup Prototype and many instructive comments. This work is partially supported by grants from Fundamental Research Funds for the Central Universities (2018CDXYJSJ0026), National Natural Science Foundation of China (61402061, 61672116 and 61802038), Chongqing High-Tech Research Program (cstc2016jcyjA0274 and cstc2016jcyjA0332), China Postdoctoral Science Foundation (2017M620412), and Chongqing Postdoctoral Special Science Foundation (XmT2018003).

References

[1] Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J. D., Manasse, M., and Panigrahy, R. Design tradeoffs for ssd performance. In *USENIX 2008 Annual Technical Conference* (Berkeley, CA, USA, 2008), ATC'08, USENIX Association, pp. 57–70.

[2] Byan, S., Lentini, J., Madan, A., and Pabn, L. Mercury: Host-side f ash caching for the data center. In *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)* (April 2012), pp. 1–12.

[3] Chen, F., Luo, T., and Zhang, X. Caftl: A content-aware f ash translation layer enhancing the lifespan of f ash memory based solid state drives. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies* (Berkeley, CA, USA, 2011), FAST'11, USENIX Association, pp. 6–6.

[4] Clements, A. T., Ahmad, I., Vilayannur, M., and Li, J. Decentralized deduplication in SAN cluster f le systems. In *USENIX'09* (Jan. 2009).

[5] Gupta, A., Pisolkar, R., Urgaonkar, B., and Sivasubramaniam, A. Leveraging value locality in optimizing nand f ash-based ssds. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies* (Berkeley, CA, USA, 2011), FAST'11, USENIX Association, pp. 7–7.

[6] Holland, D. A., Angelino, E., Wald, G., and Seltzer, M. I. Flash caching on the storage client. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference* (Berkeley, CA, USA, 2013), USENIX ATC'13, USENIX Association, pp. 127–138.

[7] Huang, S., Wei, Q., Feng, D., Chen, J., and Chen, C. Improving f ash-based disk cache with lazy adaptive replacement. *Trans. Storage 12*, 2 (Feb. 2016), 8:1–8:24.

[8] Huang, W.-T., Chen, C.-T., Chen, Y.-S., and Chen, C.-H. A compression layer for nand type f ash memory systems. In *Third International Conference on Information Technology and Applications (ICITA'05)* (July 2005), vol. 1, pp. 599–604 vol.1.

[9] Jin, K., and Miller, E. L. The effectiveness of deduplication on virtual machine disk images. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference* (New York, NY, USA, 2009), SYSTOR '09, ACM, pp. 7:1–7:12.

[10] Kim, J., Lee, C., Lee, S., Son, I., Choi, J., Yoon, S., u. Lee, H., Kang, S., Won, Y., and Cha, J. Deduplication in ssds: Model and quantitative analysis. In *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)* (April 2012), pp. 1–12.

[11] Koller, R., and Rangaswami, R. I/o deduplication: Utilizing content similarity to improve i/o performance. In *Usenix Conference on File & Storage Technologies* (2010).

[12] Li, C., Shilane, P., Douglis, F., Shim, H., Smaldone, S., and Wallace, G. Nitro: A Capacity-Optimized SSD Cache for Primary Storage. In *USENIX FAST'14* (Feb. 2014).

[13] Li, C., Shilane, P., Douglis, F., and Wallace, G. Pannier: A container-based f ash cache for compound objects. In *Proceedings of the 16th Annual Middleware Conference* (2015), Middleware '15, ACM, pp. 50–62.

[14] Li, W., Jean-Baptise, G., Riveros, J., Narasimhan, G., Zhang, T., and Zhao, M. CacheDedup: In-line Deduplication for Flash Caching. In *USENIX FAST'16* (Feb. 2016).

[15] Lillibridge, M., Eshghi, K., Bhagwat, D., Deolalikar, V., Trezise, G., and Campbell, P. Sparse Indexing: Large scale, inline deduplication using sampling and locality. In *FAST'09* (Feb. 2009).

[16] Luo, T., Ma, S., Lee, R., Zhang, X., Liu, D., and Zhou, L. S-cave: Effective ssd caching to improve virtual machine storage performance. In *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques* (Piscataway, NJ, USA, 2013), PACT '13, IEEE Press, pp. 103–112.

[17] Makatos, T., Klonatos, Y., Marazakis, M., Flouris, M. D., and Bilas, A. Using transparent compression to improve ssd-based i/o caches. In *Proceedings of the 5th European Conference on Computer Systems* (New York, NY, USA, 2010), EuroSys '10, ACM, pp. 1–14.

[18] Megiddo, N., and Modha, D. S. Arc: A self-tuning, low overhead replacement cache. In *Proceedings of the 2Nd USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2003), FAST '03, USENIX Association, pp. 115–130.

[19] Meister, D., and Brinkmann, A. Multi-Level Comparison of Data Deduplication in a Backup Scenario. In *SYSTOR'09* (2009).

[20] MENG, F., ZHOU, L., MA, X., UTTAMCHANDANI, S., AND LIU, D. vcacheshare: Automated server f ash cache space management in a virtualization environment. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2014), USENIX ATC'14, USENIX Association, pp. 133–144.

[21] NIST. Secure Hash Standard. In *FIPS PUB 180-1* (May 1993).

[22] QUINLAN, S., AND DORWARD, S. Venti: A new approach to archival storage. In *FAST'02* (Jan. 2002).

[23] SAXENA, M., SWIFT, M. M., AND ZHANG, Y. Flashtier: A lightweight, consistent and durable storage cache. In *Proceedings of the 7th ACM European Conference on Computer Systems* (New York, NY, USA, 2012), EuroSys '12, ACM, pp. 267–280.

[24] SRINIVASAN, K., BISSON, T., GOODSON, G., AND VORUGANTI, Y. iDedup: Latency-aware, inline data deduplication for primary storage. In *USENIX FAST'12* (Feb. 2012).

[25] SUEI, P. L., YEH, M. Y., AND KUO, T. W. Endurance-aware f ash-cache management for storage servers. *IEEE Transactions on Computers 63*, 10 (Oct 2014), 2416–2430.

[26] TANG, L., HUANG, Q., LLOYD, W., KUMAR, S., AND LI, K. Ripq: Advanced photo caching on f ash for facebook. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2015), FAST'15, USENIX Association, pp. 373–386.

[27] YANG, J., PLASSON, N., GILLIS, G., TALAGALA, N., SUNDARARAMAN, S., AND WOOD, R. Hec: Improving endurance of high performance f ash-based cache devices. In *Proceedings of the 6th International Systems and Storage Conference* (New York, NY, USA, 2013), SYSTOR '13, ACM, pp. 10:1–10:11.

[28] YANG, Q., AND REN, J. I-cash: Intelligently coupled array of ssd and hdd. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture* (Washington, DC, USA, 2011), HPCA '11, IEEE Computer Society, pp. 278–289.

[29] YIM, K. S., BAHN, H., AND KOH, K. A f ash compression layer for smartmedia card systems. *IEEE Trans. on Consum. Electron. 50*, 1 (Feb. 2004), 192–197.

[30] ZHU, B., LI, K., AND PATTERSON, H. Avoiding the disk bottleneck in the Data Domain deduplication f le system. In *FAST'08* (Feb. 2008).