Cluster Computing (2020) 23:623–639
https://doi.org/10.1007/s10586-019-02948-8

COWCache: effective flash caching for Copy-on-Write virtual disks

Jianyu Fu[1,3] · Youyou Lu[2] · Jiwu Shu[2] · Guangming Liu[1] · Ming Zhao[3]

Received: 16 December 2018 / Revised: 16 December 2018 / Accepted: 28 May 2019 / Published online: 11 June 2019

Abstract
Host-side flash caching emerges as an effective solution for improving the performance of virtual machines (VMs) in cloud computing environment. However, for VMs with the commonly used Copy-on-Write (COW) virtual disks, flash caching in fact has negative impacts since it brings lots of unnecessary cache writes, hurting both the VM performance and the flash endurance. This paper presents COWCache, a COW-aware caching solution that addresses this problem by co-designing flash caching with COW-based virtual disks. First, COWCache designs a new architecture that bridges the semantic gap between flash caching and virtual disk management for realizing the VMs' real data locality. Second, it separately manages COW metadata with *fine-grained caching and journaling* to improve the metadata caching efficiency. Third, it provides a novel *decoupled Copy-on-Write* mechanism, which decouples the amplified data requests from the critical I/O path and only admits the data with real VM locality into flash cache. COWCache also introduces a new data structure, the *virtual cache map*, to mitigate the memory footprint overhead for indexing the cached data in flash. Evaluations show that COWCache improves the application performance by up to 122.7% and reduces the flash cache writes by up to 78.5% compared to traditional flash caching solutions.

Keywords Caching · Flash memory · Copy-on-Write · Virtual disk

1 Introduction

Virtualization has been widely used in modern data centers to provide services such as cloud computing. However, the performance of virtual machine (VM) storage, especially for the commonly used Copy-on-Write (COW) virtual disks, still remains a major limitation [4, 7, 20, 33, 39]. COW-based virtual disks provide rich features (e.g., fast snapshot, thin provisioning) that enhance the flexibility of virtualization [19, 27, 30], but their complex operations also introduce performance overhead to the VM storage. This overhead mainly comes from two sources: (1) metadata management for maintaining the metadata of COW-based virtual disks (e.g., lookup and update); and (2) disk I/O amplification caused by the *Copy-on-Write* of data [6, 22], i.e., the COW penalties. Our study uncovers that the amount of I/O requests issued from the guest VM can be amplified by COW-based virtual disks to $2\times \sim 13\times$ to the VM's backing storage (Sect. 2.1.2).

Recently, flash-based SSDs are being increasingly deployed at the VM host side, as local flash cache for virtual disks, to accelerate the VM storage performance. To better utilize the high-speed and low-endurance flash device, researchers have made great efforts on the cache management, including the designs of caching architecture [5, 13, 38] and the optimizations of policies for cache allocation [18, 25, 28], replacement [14, 21], write-back [17, 35], admission [1, 45], etc. These optimizations

✉ Jianyu Fu
jianyufu@asu.edu

Youyou Lu
luyouyou@tsinghua.edu.cn

Jiwu Shu
shujw@tsinghua.edu.cn

Guangming Liu
liugm@nscc-tj.cn

Ming Zhao
mingzhao@asu.edu

1 School of Computer, National University of Defense Technology, Changsha, China

2 Department of Computer Science and Technology, Tsinghua University, Beijing, China

3 School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, USA

have been proposed mainly by monitoring the guest VM's I/O pattern. For VMs with COW-based virtual disks, however, their complex semantics dramatically change the guest I/O pattern and the use of flash caching actually introduce additional performance overhead. The essential cause is that the management of COW-based virtual disks is not cache-friendly and it brings lots of unnecessary writes to the cache, which hurts both the VM performance and the flash endurance. E.g., for the random write workload, COW-based virtual disks can write up to 40× more data to flash cache than the writes issued by the guest (Sect. 5.2.2).

To provide efficient flash caching for COW-based virtual disks, only monitoring the VM's I/O pattern becomes inadequate, because COW-based virtual disks semantically introduce additional metadata requests and the COW mechanism. Although one can take existing virtual disk management and flash caching solutions and simply stack them together, the semantic gap and lack of integration between the two layers will lead to a two-fold challenge.

On one hand, for metadata caching, the in-memory metadata cache typically employed by the COW systems does not work well with flash cache, due to the mismanaged metadata locality between them. This mismanagement is reflected in two aspects. First, existing coarse-grained metadata caching overestimates the metadata locality, which results in low hit ratio in memory cache and repetitive metadata access to flash cache. Second, metadata is usually updated by a couple of bytes, but coarse-grained metadata update leads to high write amplification in flash cache. Simply decreasing the caching granularity is also inefficient, because fine-grained update will induce internal write amplification in flash cache [23].

On the other hand, for data caching, one guest write may be transformed into multiple data requests by the COW layer to ensure the virtual disk consistency, and they are processed in a tightly-coupled way for traditional COW mechanism. This tight coupling leads to that the guest write cannot be acknowledged to the VM until all the amplified requests complete, which means all the requests are processed in the critical I/O path. As a result, the flash cache manager is also forced to handle more data requests and even cache more data than the guest expects, which causes that the data locality it captures actually exceeds the real VM locality. Reducing the virtual disk *cluster granularity* may mitigate the data or cache amplification but is impractical, due to the substantial increase of metadata size and the loss of data locality in the virtual disk backend, as discussed in Sect. 2.1.3.

In summary, simple stacked flash caching solutions for COW-based virtual disks miss the opportunity to exploit the benefits from the virtual disk semantics, and they will even make the advantages of flash (e.g., high performance) being underutilized and its shortcomings (e.g., limited endurance) being aggravated. Unfortunately, there has been little work concentrating on the flash caching inefficiency induced by the virtual disk semantics.

We propose *COWCache*, a COW-aware flash caching solution to optimize the performance and endurance of flash caches for COW-based virtual disks. COWCache designs a new architecture to manage flash caching at the COW layer of the hypervisor, which bridges the semantic gap between flash cache management and virtual disk management and enables the cross-layer optimizations. To meet the metadata's special locality requirements and update pattern, COWCache manages metadata caching separately from data caching with fine-grained caching and journaling between memory cache and flash cache. This approach improves the memory cache's hit ratio by preserving more metadata locality in memory and mitigates the repetitive metadata access to flash cache by journaling the fine-grained update to flash.

COWCache provides a novel *decoupled Copy-on-Write* mechanism that *decouples* the amplified data requests from the critical I/O path and admits only the data with real VM locality into flash cache. The decoupled COW improves the VM performance and the lifetime of flash by fully exploiting the non-volatile property of flash cache. Moreover, the mechanism is also a general approach that can mitigate the long standing COW penalties in COW-based virtual disks without sacrificing their flexibility.

Inspired by the efficient large-cluster design of virtual disks that mitigates the metadata size, COWCache proposes a new *virtual cache map* data structure to index the cached data in flash. It breaks traditional one-to-one cache address mapping and employs a one-to-many approach for the index that reduces the memory fingerprint overhead.

To the best of our knowledge, COWCache is the first to use the virtual disk semantics to improve the flash caching efficiency. Although the discussion in the paper focuses on flash-based caches, the general COWCache approach is also applicable to new non-volatile memory technologies (e.g., 3DXpoint [11]), which will likely be used as a caching layer between DRAMs and the slower storage. The new technologies may have higher bandwidth and lower latency, and they will still benefit greatly from COWCache by reducing the cache writes and mitigating the unnecessary data processing in the critical I/O path to fully exert their high performance.

The rest of this paper is organized as follows. Section 2 presents the background of COW-based virtual disks and host-side flash caching as well as the motivations for COWCache. Section 3 describes the design of COWCache, including its COW-aware caching architecture, fine-grained metadata caching and journaling, and decoupled Copy-on-Write. Section 4 presents the design and

implementation details. Section 5 discusses the evaluation results. Section 6 examines the related work. Section 7 concludes the paper.

2 Background and motivation

2.1 COW-based virtual disks

COW-based virtual disks (e.g., QCOW2 [27], VHD [31], VMDK [42]) have been widely deployed in major cloud infrastructures for their rich features. They have similar features and we use the QCOW2 format as an example to present their common semantics.

2.1.1 Basics

COW-based virtual disks allocate storage space on demand starting from a nearly empty file or volume, and use additional metadata to organize the guest data. As Fig. 1 shows, QCOW2 virtual disk is structured in units of fixed-size cluster, including metadata tables and data clusters. A two-level lookup table (i.e., L1 and L2) is used for address translation from the guest virtual block address to the host block address in the disk backend. A two-level reference table is used for cluster allocation [27, 37, 40]. For each guest I/O, corresponding metadata tables should be properly traversed or updated to ensure correct guest-to-host data mapping. Typically, memory cache is explicitly maintained for the metadata tables to accelerate their lookup and update.

COW-based virtual disks support disk chains [34], i.e., using writable *delta* disk linking logically to read-only *base* disk. The basis is the Copy-on-Write mechanism [40, 42]. As illustrated in Fig. 2, when the guest modifies the area G_w of a *guest* cluster G (in the logical disk) which is mapped to cluster B in the base disk, the COW layer will do the following steps before acknowledging the guest: (1) allocate a new data cluster D in the delta disk; (2) write to area D_w; (3) read area B_a; (4) write to area D_a; (5) read area B_b; (6) write to area D_b; (7) write a new L2 table entry in the delta disk to point to cluster D. Then the guest cluster G

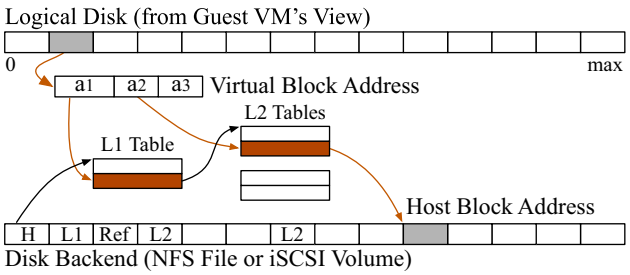


Fig. 1 Organization of QCOW2 virtual disk

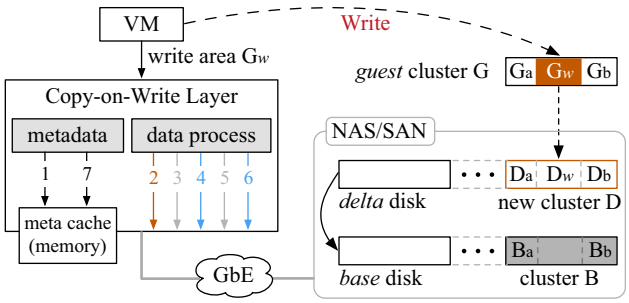


Fig. 2 The Copy-on-Write mechanism

is mapped to cluster D (cluster B will not be accessed by the guest thereafter) and the data process (including the *amplified* requests) is necessary to ensure the delta disk's consistency. Besides, the data reads or writes can also be merged respectively, which are implementation policies. As the COW mechanism supports rich features (e.g., fast snapshot, shared VM images) for COW-based virtual disks, it also introduces disk I/O amplification.

2.1.2 I/O pattern change

Figure 3 shows the impact of COW-based virtual disks (here it is one single delta disk backed by one base disk) on the guest VM's I/O pattern. For a guest new-write, the host receives up to 13× requests (with data amplification and multiple potential metadata update [7]); for a guest over-write, the host receives 2× requests (with metadata lookup). For a guest read, the host receives 2× or 3× requests, which depends on whether the data exists in the delta or base disk. Four real-world MSR workloads [32] are also replayed in the VM, and the results show that the host receives 2.5×∼3.5× requests than the guest issues. When the VM's logical disk is abstracted from a longer virtual disk chains (e.g., after making multiple snapshots), the host typically receives even more I/O requests.

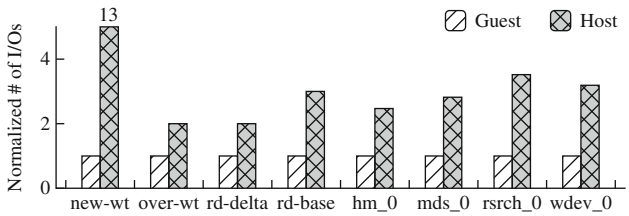


Fig. 3 Number of I/O requests observed from the guest/host. 'new-wt' means a new allocating write, 'over-wt' means a write to one previously written area, 'rd-delta' means a read directly to the delta disk, and 'rd-base' means a read to the base disk after checking that the data does not exist in the delta disk

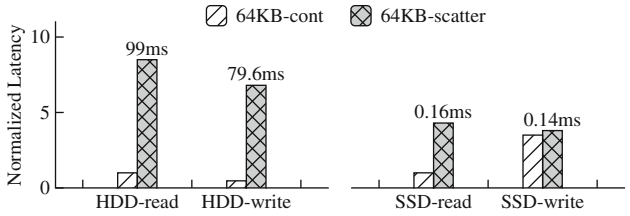


Fig. 4 Impact of virtual disk fragmentation (the latencies are normalized to 64 KB-cont at HDD-read or SSD-read respectively.)

2.1.3 Cluster granularity

Decreasing the cluster granularity (e.g., from QCOW2's default 64 KB to 4 KB) can mitigate the aforementioned data I/O amplification problem; however, two other severe challenges arise: (1) more data fragmentation. Figure 4 shows the comparison for accessing a 64 KB block between the block is continuously stored in one 64 KB cluster and the block is separately stored in sixteen 4 KB clusters that each may be completely scattered in the backing storage for worst case. The results show that the access latency of 64 KB-scatter can be slower than 64 KB-cont by up to 13.5× for HDD (at 'HDD-write') and up to 3.3× for SSD (at 'SSD-read'). This data fragmentation causes high performance overhead for COW-based virtual disks; (2) more metadata. For one single 1TB QCOW2 virtual disk, using 4 KB cluster versus 64 KB cluster will increase the metadata size from 160 MB to 2.5 GB, which incurs large management overhead. With the use of metadata replication in virtual disks like VMDK [7, 42] and longer virtual disk chains, the metadata overhead is more severe. As such, COW-based virtual disks typically employ large clusters to reduce data fragmentation and metadata management overhead [2, 36, 44].

2.2 Host-side flash caching layers

Figure 5 shows the generally employed I/O layers to manage host-side flash caching in virtualized environment.

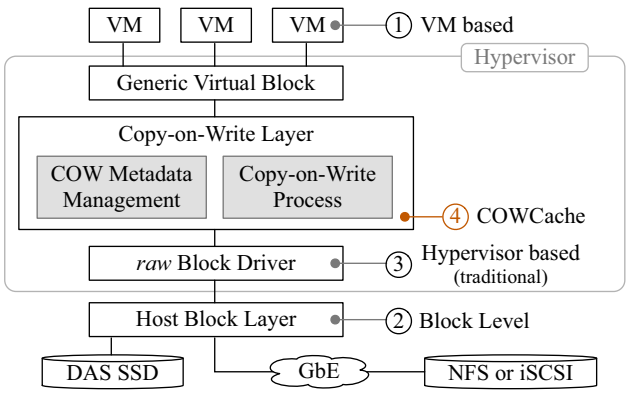


Fig. 5 I/O stacks and flash caching layers

VM-based flash caching (option ①) is beneficial for guest applications to manage flash cache according to their specific requirements [24]. However, the burden of manual modifications on users and the difficulty of dynamically sharing cache device among VMs make the benefits limited [5, 25]. Moreover, COW-based virtual disks are abstracted as logical disks in VMs and their semantic information are hidden from the cache manager, which can hardly optimize flash caching.

Host block-level flash caching (option ②) is a general-purpose approach for both virtualized and non-virtualized environments. However, due to the complex host I/O stacks, little VM semantic information can be delivered to the cache manager like option ① [25, 29]. Further, COW-based virtual disks are cached as multiple regular files without considering their logical relationship, which may induce unnecessary data caching, e.g., both the delta and base disks' data blocks that belong to the same logical block may be cached but the base disk's block is no longer useful for the VM.

In virtualized environment, managing flash cache in the hypervisor (option ③) is a commonplace due to the easy control for both the VMs and flash caches [4, 5, 18, 25, 28]. It is able to collect all the VM-identified I/O requests, to manage flash cache transparently to the VMs, and to support both high-level networked storage protocols and system block-level protocols. However, the cache manager in this option still lacks crucial semantic information about the virtual disk, which raises new challenges to provide caching for COW-based virtual disks, as described in Sect. 2.3.

2.3 COW caching challenges

The above design and results for COW-based virtual disks show that traditionally monitored guest I/O pattern has been changed dramatically by the COW layer [10, 12], and the new challenges of flash caching for them come from two aspects.

2.3.1 Metadata inefficiency

Since every guest I/O request to COW-based virtual disks needs metadata lookup or update, efficient metadata caching is important to the overall performance. Traditional locality model does not distinguish between metadata and data; however, metadata have different properties from data: (1) one small metadata table maps to a large range of data address space (e.g., for default 64 KB cluster in QCOW2, one L2 table can translate 512 MB consecutive logical disk space), which means the metadata locality is much smaller than data; (2) the modifications are usually very small in entry level (e.g., one L2 table entry is 8

bytes). Simply stacking metadata flash cache under memory cache is very inefficient due to the mismanaged metadata locality between them. On one hand, since the requested metadata is usually much less than one whole table, managing metadata in coarse granularity between memory cache and flash cache, which is default in tables for existing metadata caching, lowers the memory cache hit ratio and thus induces frequent table replacement in memory cache and table reads from flash cache. Specially, when dirty tables are frequently evicted from memory cache, they will cause repetitive writes for unmodified entries to flash cache. On the other hand, managing metadata in fine granularity (e.g., in entries) will induce much internal write amplification in flash due to the fine-grained update pattern to flash cache [23].

2.3.2 Copy-on-Write amplification

In the COW process, the guest write is transformed into multiple data requests to the disk backend (Sect. 2.1). The underlying flash cache manager receives *all* these requests, and processes and caches them as normal guest requests. To be specific, for a single guest write, the cache manager needs to: (1) read data from the base disk through network; (2) cache unmodified areas that have the same content twice (e.g., B_a and D_a), since the cache manager considers them as different data from different files or volumes. Although it is possible to employ cache deduplication [9, 21], it incurs unnecessary computation overhead. It is also a choice to deploy the cache manager below the VMs but above the COW layer, and then use individual VM's logical address indexing cached data to mitigate the above caching issues. However, due to the address space isolation among the VMs, the cache will still store multiple copies for different logical blocks that have identical content (e.g., from the same base disk), and the COW penalties of COW-based virtual disks still exist. In short, the cache manager is forced to do more caching work than the VMs expect, due to the lack of virtual disk semantics. The above is the *Copy-on-Write amplification* problem with flash caching.

Although COW-based virtual disks are widely deployed and the VM snapshot and clone operations are commonly used by IaaS systems, unfortunately, the aforementioned issues have not received enough attention. Drop-in flash caching solutions cannot mitigate the performance overhead of COW-based virtual disks. Even worse, the cache-unfriendly management of COW-based virtual disks and the semantic gap between virtual disk management and flash caching result in extremely inconsistent VM performance and serious flash endurance problem.

3 Design

COWCache is a COW-aware flash caching solution, which carefully utilizes the virtual disk semantics to improve the flash caching efficiency. In this section, we first discuss the architecture of COWCache. We then describe the fine-grained metadata caching and journaling. Finally, we present the decoupled Copy-on-Write mechanism.

3.1 COW-aware caching architecture

As Fig. 5 shows, COWCache is a hypervisor-based flash caching solution (option ④). But different from traditional caching layers in the hypervisor (option ③), COWCache is designed in the COW layer. Then COWCache can be easily aware of all the necessary knowledge of COW-based virtual disks' metadata management and data processing, which is a key advantage to optimize the flash caching.

Figure 6 shows the detailed architecture of COWCache. It consists of two modules, i.e., the **MC** module for fine-grained metadata management and the **DC** module for decoupled COW. COWCache provides good modularity so that it is easy to integrate existing caching algorithms just under the DC module, without the requirements to know what the COW mechanism is or how COWCache operates. Virtual cache map is a new design to mitigate the memory footprint for the description information of cached metadata and data in flash, which will be elaborated in Sect. 4.

The new architecture design of COWCache makes it easier for optimizing the flash caching for COW-based virtual disks without changing much to existing I/O stacks (e.g., the interfaces). First, COWCache can easily identify the metadata and data requests, so that it can cache them separately and adopt different caching policies for each. Second, COWCache can manage the metadata more naturally and support fine-grained metadata management, i.e., fine-grained metadata caching and journaling, to reduce the metadata lookup overhead and the metadata write

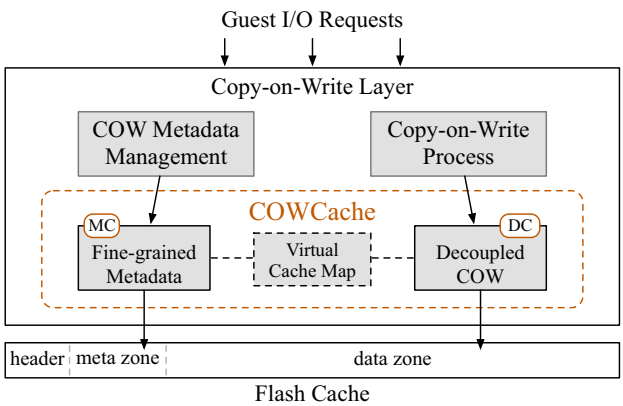


Fig. 6 Architecture of COWCache

amplification on flash cache. Third, COWCache can distinguish the original guest I/O requests from the extra data requests amplified by the COW layer and understand the real data requirements of the VM. As such, it can employ a *decoupled COW* mechanism to decouple the amplified data requests from the critical I/O path and to cache only the data with real VM locality. Finally, COWCache is aware of both the guest requests' logical disk address and disk backend address, so that it can semantically deduplicate the cached data from different snapshots but has the same logical disk address.

3.2 Fine-grained metadata caching and journaling

To provide efficient metadata caching, COWCache manages metadata in both memory cache and flash cache at the same fine granularity (by default in sectors, and other granularities are also supported), and updates them to flash cache in journaling, as shown in the right subgraph of Fig. 7. As a comparison, the left subgraph of Fig. 7 shows traditional coarse-grained metadata management. Note that COWCache does not change the organization of metadata in the disk backend (i.e., they are still stored in original granularity), but only changes their caching management to be cache-friendly. The metadata I/O flows are illustrated as follows.

3.2.1 Metadata I/O flow

For a metadata entry read, the metadata *sector* that contains this entry will be checked. If it is in memory cache, then the entry is returned; otherwise, if it is only in flash cache, then the sector will be read into memory cache and returned. If neither memory nor flash cache has the sector, the whole metadata *table* will be read from the disk backend, which is to mitigate the slow backend access. Then the *table* will be cached in flash and the *sector* will be cached in memory.

For a metadata entry write, when COWCache updates the corresponding metadata sector in memory cache, the dirty sector is not written to flash cache directly, but firstly copied into a *journal buffer* in memory. The journal buffer is a small memory area (e.g., 64 KB) that groups the fine-grained metadata update to prevent them inducing more internal write amplification to the flash device. Then the journal will be written back to flash cache when it reaches either a pre-defined size or age.

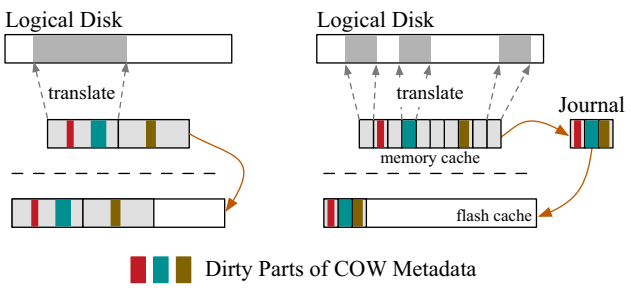


Fig. 7 Coarse- versus fine-grained metadata management

3.2.2 Advantages

With fine-grained metadata caching and journaling between memory and flash cache, COWCache has three advantages: (1) it caches more valuable metadata information in memory that can translate more distributed logical disk space, compared to several large consecutive disk space that can be translated by fewer metadata tables, which improves the metadata hit ratio in memory cache; (2) higher metadata hit ratio in memory cache and fine-grained metadata interaction bring less metadata reads from flash cache; (3) less frequent metadata replacement in memory cache and fine-grained metadata journaling induce less metadata writes to flash cache. Moreover, less metadata access to flash cache also reduces its bandwidth contention to normal data caching in flash which improves the data caching efficiency.

In addition, metadata operations for COW-based virtual disks need to get a exclusive lock, and almost all the metadata access are in these periods holding the lock. So compared to traditional coarse-grained metadata management, COWCache not only improves the metadata caching efficiency, but also mitigates the impact of metadata on the parallelization of I/O requests and thus improves the overall VM performance.

3.3 Decoupled Copy-on-Write

Traditional COW mechanism is a tightly-coupled process that all the amplified data requests are handled in the critical I/O path along with the guest write to ensure consistency. With flash caching, the cache manager is also forced to process the amplified requests and even to cache these extra data that exceed the VM's real locality into flash. However, we observe that both the traditional COW process and the flash caching efficiency can be improved by exploiting the non-volatile property of flash cache, and we propose the new *decoupled Copy-on-Write* mechanism.

The key idea for the decoupled COW mechanism with flash caching is two-fold: (1) only process the real guest requests in the critical I/O path; and (2) only admits the requests that have real VM locality into flash cache. After bridging the semantic gap between the management of virtual disks and flash caches, the guest or amplified requests can be easily distinguished. Then for the virtual

disk management, the use of non-volatile flash caches makes it a good opportunity to decouple the amplified I/O requests from the critical path (especially for the *writeback* policy, as discussed in Sect. 3.3.1), and maintain necessary description information to ensure correct and consistent data access; for the flash cache management, the cache manager only admits the guest requests into flash cache to cache the real guest I/O demand.

As Fig. 8 shows, when the guest writes to $D1_w$ and $D2_w$, previous caching solutions caches more data under traditional COW, while COWCache only caches the real guest writes into flash, which utilizes the cache space more efficiently. An optimization is introduced by recognizing a block's logical and physical addresses: if the guest reads $B2_b$ before writing to $D2_w$, then $B2_b$ is cached since it has VM locality; when the guest writes to $D2_w$, although it has different host address from $B2_b$, $B2_b$ loses VM locality because the guest will not access it again (logically masked by $D2_w$). COWCache evicts $B2_b$ from the cache to further improve the space efficiency.

There are two common polices for writing back cached data in flash, i.e., *writeback* and *writethrough* [17, 35]. COWCache supports both policies but designs different caching I/O flows for them considering different tradeoffs between the storage performance and consistency.

3.3.1 Caching I/O flow

Writeback policy For an allocating write in the COW process, the guest write is cached in flash and the other amplified requests are not processed. Instead of issuing the amplified reads to the base disk and writes to the delta disk, COWCache only marks the newly allocated delta disk cluster as *partially valid* and then acknowledges the guest write to the VM. As such, COWCache decouples the amplified data requests from the critical I/O path. When such cached guest write needs to be written back to the delta disk (e.g., evicted out from flash cache), the guest write is written back and the unmodified areas in the delta disk cluster still remain invalid (necessary description information are persisted to avoid inconsistency under crash, as discussed in Sect. 3.3.3). For subsequent guest writes to the same cluster, they are also cached into flash like the first write to this cluster, and the remaining unmodified areas (if still exist) are not processed as well.

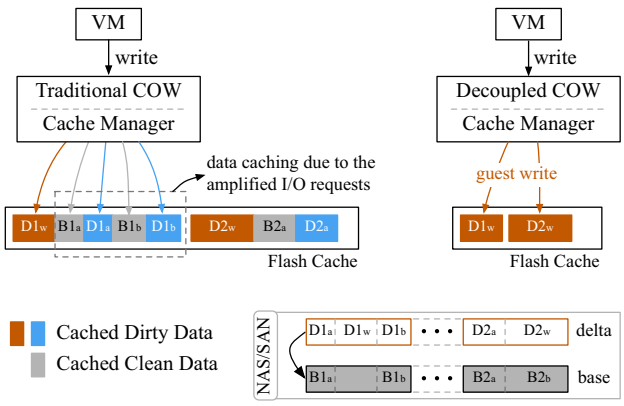


Fig. 8 Flash caching with traditional versus decoupled Copy-on-Write

Meanwhile, the guest read I/O flow is also changed to ensure correct disk access. For a guest read, since some delta disk clusters may be partially valid, there are three routines for COWCache: (1) if the data is cached in flash, it will be read directly from flash cache; (2) if the data is not cached and its delta disk cluster is *completely valid*, it will be read normally from the delta disk; (3) if the data is not cached and its delta disk cluster is partially valid, although the base disk cluster has been logically masked, the data may still be redirected to read from the base disk if not in the delta disk.

Writethrough policy In this policy, the guest write is cached in flash and the amplified data requests are not admitted into cache either, but COWCache still issues the amplified data requests to the backing storage, i.e., read data from the base disk and write them to the delta disk, to ensure the delta disk is always completely valid, so as to satisfy the strong consistency requirements of writethrough.

For both policies, COWCache reduces the flash writes by not admitting the data without real VM locality into flash cache. In addition, COWCache also mitigates the access latency of allocating guest writes and the network traffic by eliminating the amplified data requests from the critical I/O path in the writeback policy.

3.3.2 Consistency analysis

Although there are other consistency problems and solutions related to flash caching (e.g., in [17, 35]), here we mainly concentrate on the inconsistency potentials while caching for COW-based virtual disks.

For the writethrough policy, it is designed to provide strong consistency, which is also ensured by COWCache. In the COW process, COWCache does not cache the amplified requests, but still issues them along with the guest write, i.e., the guest write (e.g., $D1_w$) is sent to the delta disk, and the unmodified areas are read from the base disk (e.g., $B1_a$ and $B1_b$) and written to the delta disk (e.g., $D1_a$ and $D1_b$). Before persisting the lookup entry to point to the newly allocated cluster in the delta disk, the cluster is already completely valid, so that the delta disk is always consistent to use.

For the writeback policy, the caching I/O flow is redesigned to process minimum I/Os in the critical path but still

ensures the guest to access consistent disk data. However, if the host crashes, there will be inconsistency potentials, because some delta disk clusters may be partially valid but considered as completely valid after recovery. Then the guest may read invalid areas in some delta disk clusters. Note that the inconsistency problems also exist in traditional flash caching solutions for COW-based virtual disks, since out-of-order cache writeback may result in (partially) invalid clusters. To illustrate them, we give some denotations for the decoupled COW process: the updated COW metadata (*Meta*), e.g., a new L2 table entry; the data written by the *guest* (*GData*); when Meta and GData are cached in flash, their description information, e.g., the cache-to-backend address mapping and the cluster status, are denoted as *FInfo*.

The conditions at the host crash point that may induce delta disk inconsistency are: Meta is already written back to the delta disk, GData exists in either the flash cache or delta disk, and FInfo is still just in memory (i.e., not persisted to flash yet). When Meta has been updated to the delta disk, it should point to a completely valid cluster. However, if FInfo has not been persisted, the description information will be lost. Then if GData only exists in flash cache, it cannot be recovered, and Meta will point to a completely invalid cluster in the delta disk; even though GData has been written back to the delta disk cluster, the cache manager cannot identify which parts are valid or invalid without the cluster status information, so that Meta will point to a partially valid cluster. The above conditions cause the delta disk to be inconsistent.

For other cases, if FInfo has not been persisted, and Meta and GData are only cached in flash (or only GData is written back to the delta disk but Meta is not), then both Meta and GData are lost but it does not induce inconsistency; if FInfo has been persisted and each of Meta and GData can be either only in flash or already written back to the delta disk, then both the Meta and GData can be recovered and the decoupled COW mechanism still functions correctly.

3.3.3 Consistent cache writeback

In COWCache, while writing back cached data in flash that are related to the partially valid clusters, some write orders must be kept to avoid the above inconsistent cases. To be specific, before writing back cached Meta (dirty COW metadata) or GData (partially updated data) from flash to the delta disk, COWCache firstly ensures that in-memory FInfo (address mapping and cluster state) has been persisted to flash. Thus, even if the host crashes, all the information about the partially valid delta disk clusters can be recovered and the consistency of the decoupled COW mechanism can be ensured. In addition, the address mapping and cluster state information are also periodically persisted to mitigate new data loss.

An alternative approach is that before writing back cached Meta from flash to the delta disk, GData should be written back first and the unmodified areas in corresponding clusters should be filled with data from the base disk, thus Meta will not point to invalid clusters. But this approach will induce some overhead, since during the whole VM running process, there is no need to specially read data from the base disk to fill partially valid delta disk clusters. First, there are write coalescing [17] in flash cache that inconsistent clusters will be less (i.e., new allocated clusters are written completely by the VM), and the amplified data reads from the base disk and data writes to the delta disk will become unnecessary. Second, the extra copying can be done while shutting down the VM, or just mark the delta disk as inconsistent and do the copying work offline, which will mitigate COW-based virtual disks' negative impact on the running VM's performance.

3.3.4 Recovery

There are recoverable or destructive failures for the host and flash regarding whether the flash device can be recovered or not [35]. For both kinds of failures, COWCache provides strong virtual disk consistency under the writethrough policy, but the performance is limited. The writeback policy is suitable for recoverable failures and provides better performance. Note that by employing the peer-replication caching technique as introduced in [4], COWCache can also provide strong consistency under the writeback policy, but it is out of the scope of this paper.

COWCache designs two recovery approaches for host crashes in the writeback policy: *fast recovery* and *full recovery*. Fast recovery means COWCache only recovers the address mapping and cluster state information for all cached data in flash, then the VM can go on running as before the crash. Although there may be partially valid clusters, COWCache can identify them from the cluster state, so that the VM still runs correctly. Full recovery means that COWCache not only recovers the information as in fast recovery, but also reads all cached dirty data and writes them into the delta disk, including read data from the base disk to fill all the partially valid clusters in the delta disk, so as to make the delta disk completely consistent before using it again.

Fast recovery is fast and efficient but needs the VM recovered in the same host. Full recovery needs more time to copy data but recovers the delta disk backend to a consistent state, and then the VM can be restarted in a different host, which provides more flexibility.

4 Implementation

We implemented a COWCache prototype based on the QCOW2 driver in the QEMU [3] emulator with KVM [16] enabled.

4.1 Virtual cache map

Cached data on flash is usually managed as fixed-size block, e.g., 4 KB [1, 5, 25, 28]. For every cached block, there is an in-memory entry preserving the address mapping from the original block number in the disk backend to the cache block number in flash, i.e., a *one-to-one mapping* like (LBN, CBN). Inspired by the cluster-style organization of COW-based virtual disks, we observe that, although different data blocks in the same cluster have different LBNs, they actually have the same cluster number (LCN), which means duplicated address mapping information exist in memory. We propose a new design for the in-memory address mapping structures, called *virtual cache map*, to mitigate the memory footprint overhead for flash caching.

Virtual cache map decouples traditional one-to-one address mapping, and one entry maps to multiple cached blocks. For each entry, it has a *single* LCN identifying one cluster in the disk backend and *multiple* CBNs for the cached blocks inside this cluster. It also has a guest cluster number (GCN) to recognize the data from different snapshots whereas with the same logical disk address. Since every entry maps to multiple cached blocks, there are two small bitmaps: one is the data bitmap (one bit represents whether a block in the cluster is cached or not); the other is the dirty bitmap (one bit represents whether a cached block is dirty or not). Moreover, there is a *dcow* bitmap for every entry to store the cluster's consistency state. It indicates which parts in the cluster are valid or not, and it is the pivotal information for the decoupled COW running correctly. So, for a guest I/O request, after identifying its cluster address in the disk backend, the corresponding virtual cache map entry will be checked to see the data block's status (e.g., whether cached or not, dirty or not, in the delta disk or in the base disk), according to which to process the request as aforementioned in Sect. 3.3.1.

For default 64 KB cluster size and 4 KB cache block size, one virtual cache map entry contains a 32-bit LCN, a 32-bit GCN, a 4-bit disk ID, a 64-bit pointer to a dynamic array that contains 1–16 32-bit CBNs, three 16-bit bitmaps and one 32-bit reference counter for each cached block. So the memory overhead for a cluster is about 0.23%. Since not all the blocks in a cluster are always cached, the real memory overhead depends on the workloads.

4.2 COW awareness

To be aware of COW-based virtual disks, COWCache designs more flexible caching interfaces that support the QCOW2 driver to give more hints about the request, which includes not only its backend address, but also its logical address, its type (metadata/data), and whether it is a guest request or amplified request due to the COW process.

4.3 Flash cache organization

Flash cache is split into three regions: cache header, metadata zone, and data zone. The cache header consists of the superblock, the address mapping, and the cluster state information for all cached data. The metadata zone caches the journaled COW metadata default in sector granularity, and the data zone caches the normal data with an in-place update manner default in 4 KB blocks and with the LRU replacement policy.

5 Evaluation

5.1 Experimental setup

We performed the experiments on a machine with two Intel Xeon E5-2680 v3 12-core CPU (2.50GHz), 384 GB main memory, 1 TB Seagate HDD, and a 400 GB Intel SSD DC S3610 as host-side flash cache. The host runs Fedora 23 and the guest VM runs CentOS 7. Each VM is configured with 1 vCPU, 2 GB RAM and two QCOW2 virtual disks: one is the OS and the other is a newly created 1 TB sparse delta disk backed by a preallocated base disk to conduct experiments. Both the delta and base disks are stored in an NFSv4 datastore, backed by a 2 TB Intel SSD DC P3700 and connected via IPoIB (the network bandwidth is up to 1.6 GB/s).

5.2 Micro-benchmark evaluation

FIO [15] is used to do the micro benchmarks. We run FIO in the VM to produce different types of guest workloads to the 1 TB virtual disk. We will elaborate the workloads and configurations for each experiment. Compared to COWCache (*COWC*), TRDCache (*TRDC*) means the COW-*unaware* flash caching solutions with table-grained metadata management and traditional COW process. 'wt' means the *writethrough* policy for flash cache and 'wb' means *writeback*. E.g., COWC-wb means COWCache with the writeback policy.

5.2.1 Fine-grained metadata caching and journaling

We evaluate COWCache's metadata optimization with host-side flash caching. The guest workload consists of totally one million random 64 KB write requests to the 1 TB virtual disk. The request has the same size as the virtual disk cluster so that there are no COW operations (i.e., no data amplification that copies data from the base disk to the delta disk).

Figure 9 shows the guest write throughput of two metadata caching mechanisms at different memory cache sizes for L2 tables. The results show that the throughput of COWCache outperforms TRDCache by 6.7–116.5% in 'wt' mode and by 2–175.7% in 'wb' mode, which are because COWCache caches more valuable metadata information in memory and produces less metadata flash cache reads and writes (more details in Table 1).

Table 1 shows that with different memory cache sizes, the hit ratio of COWCache outperforms TRDCache by up to 124% (e.g., 75.1% to 33.6% with 1 MB memory cache). The reason is that COWCache manages metadata memory cache at fine granularity, which translates more distributed logical disk space than TRDCache. TRDCache has lower hit ratio and thus needs more times of metadata reads from flash cache in granularity of metadata table. E.g., with 1 MB memory cache, TRDCache induces surprisingly 340× more metadata flash reads than COWCache. COWCache also writes back much less metadata to flash cache. E.g., with 1 MB memory cache, COWCache induces only 0.7% metadata flash writes compared to TRDCache; even with 128 MB memory cache, COWCache writes only 9.4% metadata size of TRDCache. Meanwhile, when the memory cache size decreases, the metadata flash writes of TRDCache are proportionally increased, while COWCache always writes the same size. These results demonstrate that COWCache improves the metadata hit ratio in memory cache and reduces the bandwidth contention to data caching in flash that lead to higher guest throughput.

5.2.2 Decoupled Copy-on-Write

We evaluate COWCache's decoupled COW mechanism with host-side flash caching. The guest workload consists of totally one million random write requests to the 1 TB virtual disk. The memory cache size for L2 tables is set to the maximal value (i.e., 128 MB) to minimize the impact of metadata management.

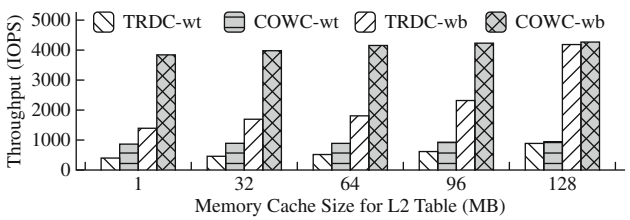


Fig. 9 Impact of COW metadata

Figure 10a shows the guest write throughput at different block sizes and Fig. 10b shows the stacked traffic of flash writes and network I/O. First, in 'wt' mode, the throughput of COWCache outperforms TRDCache by 3.8–10.9%, and the improvement is mainly because COWCache eliminates the flash writes for those amplified data without real VM locality. Compared to TRDCache, to be specific, COWCache reduces the flash writes by 31.6–96.7%, e.g., 96.7% at 4 KB block size, and it induces the same amounts of network I/O to ensure all the clusters in the virtual disk backend to be valid thus to provide stronger consistency. Second, in 'wb' mode, the throughput of COWCache outperforms TRDCache by 60%∼4.4×, and the improvement comes from not only the flash writes reduction which is the same as in 'wt' mode, but also the decoupling of the amplified read and write data requests from the critical I/O path. For the guest writes, TRDCache needs to read unmodified areas from the base disk through network in the critical path and thus it still bears latency of networked storage, while COWCache eliminates those read requests from the critical path and thus it can fully benefit from the high performance of flash cache.

Note that at aligned 64 KB block size ('64-a'), none guest writes induce the COW operations, so COWCache and TRDCache process the same amounts of flash writes and network I/O and have the same throughput. While for unaligned 64 KB block size ('64-u'), the guest write spans in two consecutive clusters. TRDCache still needs to fill the unmodified areas in the beginning and ending parts of the clusters, but COWCache processes them in a decoupled manner so that it has obvious performance improvement than TRDCache again.

The above results are achieved while maximizing the memory cache size for L2 tables to minimize the metadata's impact. When decrease this memory cache size to 1 MB, TRDCache induces 40× more flash writes than COWCache for the random write workload at 4 KB block size.

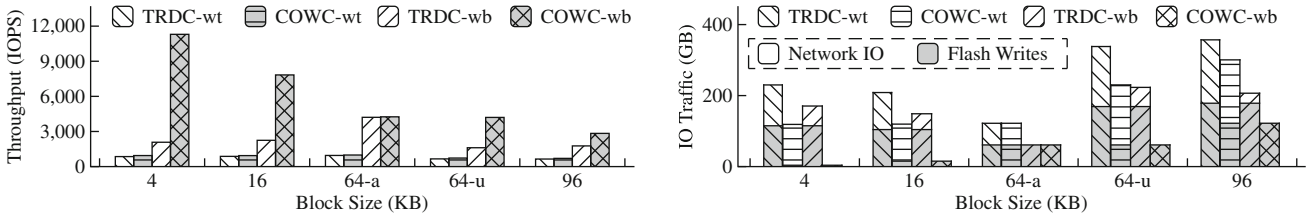
5.2.3 Impact of COW ratio

The ratio of allocating writes, which induce the COW process, in all the writes is defined as *COW Ratio*. For COW-based virtual disks, the COW ratio is high at the beginning, since most writes are allocating writes; then the COW ratio gradually decreases if not taking snapshot, because more latter writes are issued to previously allocated clusters. We evaluate the impact of COW ratio on the

Table 1 Memory cache hit ratio of L2 tables, and metadata flash reads and writes

Memory cache (MB)	Hit ratio of L2 (%)		Meta flash reads (GB)		Meta flash writes (GB)	
	TRDC	COWC	TRDC	COWC	TRDC	COWC
1	33.6	75.1	119.64	0.35	61.15	0.47
32	74.7	83	45.66	0.24	46.61	0.47
64	83.1	90.4	30.39	0.14	31.95	0.47
96	91.6	96.6	15.13	0.05	18.08	0.47
128	∼100	∼100	0	0	4.95	0.47

The modes 'wt' and 'wb' have the same values for the above results of TRDCache or COWCache



(a) Throughput

(b) Stacked Traffic of Flash Writes and Network I/O

Fig. 10 Impact of Copy-on-Write

VM performance with different write/read ratios, i.e., 2:1 (write-intensive) and 1:2 (read-intensive) respectively. The guest workload consists of totally one million 4 KB I/O requests to the 1 TB virtual disk. The workload is random but controlled necessarily to achieve the specific COW ratios. The memory cache size for L2 tables is also maximal.

The left subgraph of Fig. 11 shows the VM throughput at 2:1 write/read ratio and the results are normalized against TRDCache at 80% COW ratio in 'wb' mode. At different COW ratios, COWCache outperforms TRDCache by 4.1–7% in 'wt' mode and 1.8× ∼3.4× in 'wb' mode. When the COW ratio gets lower, both COWCache and TRDCache have better performance. E.g., at 40% COW ratio (vs. 80%) in 'wb' mode, the throughput increases 16.3% for COWCache and 79.6% for TRDCache, but COWCache still outperforms TRDCache by 1.8×. The improvement for COWCache is mainly because that lower COW ratio means less allocating writes and thus less metadata update, while the improvement for TRDCache is mostly because there are less COW operations and less amplified data requests need to be processed. The results also show that with the COW ratio changing, the VM performance for TRDCache is not only low but also inconsistent, thus is more unpredictable. The right subgraph of Fig. 11 shows that for read-intensive guest workloads, COWCache still outperforms TRDCache by 2.3–8.2% in 'wt' mode and 1.0× ∼1.7× in 'wb' mode.

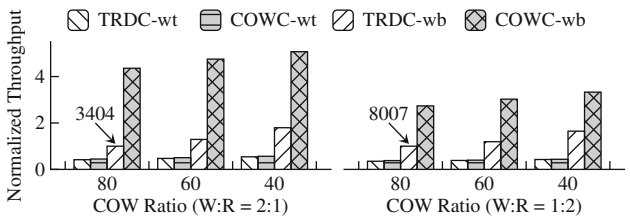


Fig. 11 Impact of COW ratio

5.3 Application evaluation

We evaluate COWCache using typical application workloads, i.e., OLTP, Varmail, and Fileserver from Filebench [26], YCSB [8], and the SNIA MSR traces [32].

5.3.1 Workloads

OLTP emulates an online transaction processing service. It has two typical I/O sizes, 2 KB (*OLTP2*) and 8 KB (*OLTP8*). Varmail emulates a mail server and Fileserver emulates a file server. YCSB is a framework used for benchmarking cloud serving systems. We use MySQL as the database and choose the workload A (*YCSB-A*), which consists of a mix of 1:1 write/read requests. We select ten SNIA MSR traces that represent a variety of real-world workloads: hardware monitoring (*hm_0*), media server (*mds_0*), print server (*prn_0*), project directories (*proj_0*), web proxy (*proxy_0*), research projects (*rsrch_0*), web staging (*stg_0*), terminal server (*ts_0*), user home directories (*usr_0*), and test web server (*wdev_0*).

We run the OLTP workloads in the VMs with different memory cache sizes for L2 tables, e.g., OLTP2-1 means the I/O size is 2 KB and the memory cache is 1 MB.

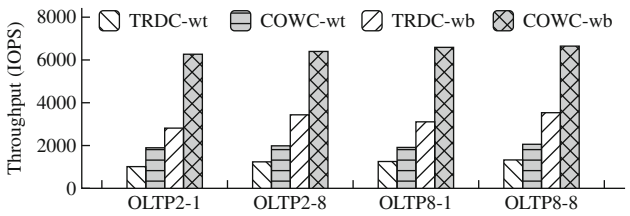


Fig. 12 The OLTP workloads

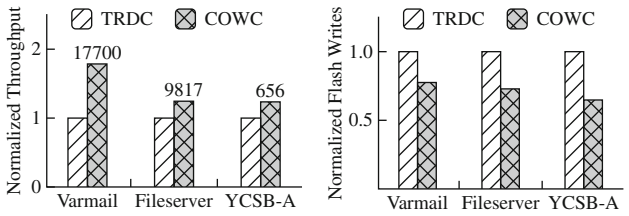


Fig. 13 Application workloads

Figure 12 shows that the throughput of COWCache outperforms TRDCache by 52.5–86.8% in 'wt' mode and 86.2–122.7% in 'wb' mode. Moreover, COWCache is less affected by the metadata cache size and its performance is more consistent and predictable than TRDCache. E.g., for OLTP2-1 (vs. OLTP2-8) in 'wb' mode, the performance degradation of TRDCache is 18.1%, while it is only 2% for COWCache.

Table 2 shows the detailed results. We use OLTP2 as an example to elaborate them and OLTP8 has similar results. For OLTP2-1, the memory cache hit ratio of L2 tables for COWCache (69.9%) outperforms TRDCache (39.8%) by 75.6% due to the fine-grained metadata management. Thus, COWCache induces only 0.32% flash reads and 0.73% flash writes for metadata compared to TRDCache. Due to the decoupled COW mechanism and only caching data with real VM locality, COWCache writes 35.3% data to flash cache of that written by TRDCache. So COWCache reduces the overall flash writes by 77.2%, which can obviously extend the lifetime of flash device.

When increase OLTP2's memory cache for L2 tables to 8 MB, the metadata reads for both TRDCache and COWCache can be accessed from the memory cache and their hit ratios are nearly 100%. But TRDCache still induces 125× more metadata flash writes than COWCache. Since OLTP is a sync-intensive workload, and for every sync operation, TRDCache updates all the dirty metadata tables to flash cache, while COWCache only updates the dirty metadata sectors in a journaled way to flash cache, which induces significantly less metadata flash writes.

Figure 13 shows the throughput and flash writes of Varmail, Fileserver, and YCSB-A in 'wb' mode and the results are normalized against TRDCache. Their metadata memory cache sizes are maximal. The left figure shows that COWCache has higher throughput than TRDCache by 78.6%, 24.5%, and 23.3% respectively, and the right figure shows that it reduces the flash writes by 22.5%, 27.1%, and 35.2% respectively. While in 'wt' mode, the throughput improvement for Varmail is 13% and less than 2% for both Fileserver and YCSB-A.

5.3.2 Impact of snapshot frequency

We evaluate COWCache using the MSR traces to see the impact of snapshotting on the flash writes. In this test, we select one week-long part from each trace (most traces are one week long) to replay. 'No Snapshot' (S_{no}) means at the whole replaying process, there is only one delta disk. 'Snapshot Per Day' (S_{per}) means making one snapshot per day, so there is a longer disk chain. Figure 14 shows COWCache's flash writes reduction compared to TRDCache in both cases. The Y-axis means the reduction ratio of flash writes since last snapshot. Because we do not make intermediate snapshot for S_{no}, the reduction ratio is always from the beginning day.

For S_{no}, the flash writes reduction ratios are 1.4–32.8% at different days for the workloads. And for the same workload, the reduction ratio gradually decreases, because for the single delta disk, the ratio of allocating writes gradually decreases. But the reduction ratio may increase. E.g., for *prn_0*, the curve has an increase in the fourth day,

Table 2 Detailed results of the OLTP workloads for TRDCache and COWCache

Workloads	COW ratio (%)	Memory cache hit ratio of L2 (%)		Meta flash reads (GB)		Meta flash writes (GB)		Data flash writes (GB)		Reduction ratio of flash writes (%)
		TRDC	COWC	TRDC	COWC	TRDC	COWC	TRDC	COWC	
OLTP2-1	73.1	39.8	69.9	87.46	0.28	21.9	0.16	38.75	13.66	77.2
OLTP2-8	68.8	~100	~100	0	0	23.84	0.19	46.48	17.08	75.4
OLTP8-1	74.5	31.4	56.3	90.05	0.33	20.97	0.17	41.31	13.19	78.5
OLTP8-8	69.9	~100	~100	0	0	21.5	0.16	48.13	20.75	70

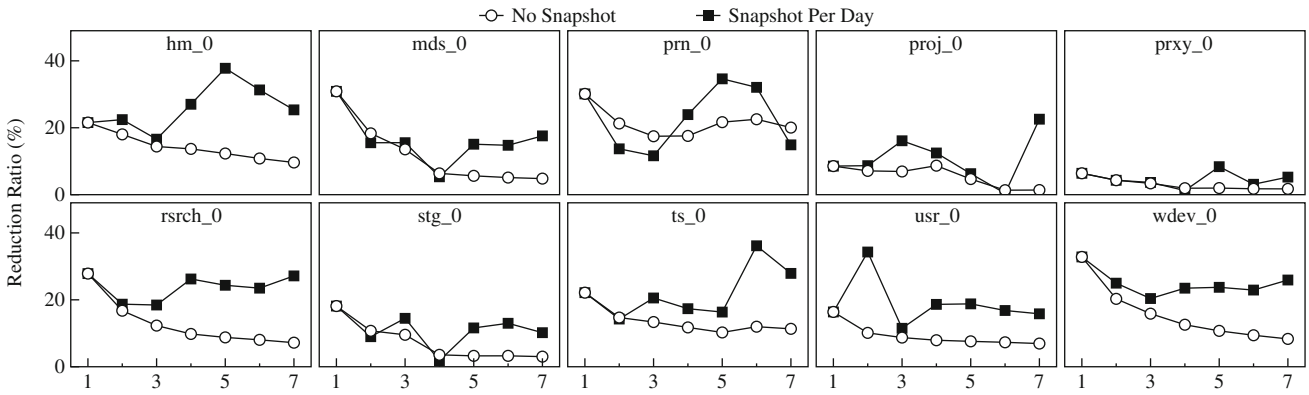


Fig. 14 Flash writes reduction (the X-axis is the day)

and the reason is that the workload writes to more new clusters in the delta disk, so the overall COW ratio becomes higher. For S_{per}, the reduction ratios are 0.4–37.8% at different days. Compared to S_{no}, the curve of S_{per} is mostly higher, since after each day's snapshot, the previous delta disk becomes read-only, and a new delta disk is created based on the previous one. Then there are more allocating writes in the new delta disk and COWCache induces less flash writes by not caching the data without VM locality.

5.3.3 Impact of flash cache size

Figure 15 shows the normalized stacked traffic of flash writes and network I/O for each MSR trace at different flash cache sizes in 'wb' mode. The cache size is the percentage of each workload's working set size (WSS). At different cache sizes, COWCache reduces the flash writes by 2.8–25.5% and the network traffic by 18–76.2%. The reduction are mainly from two sources. First, COWCache needs less reads from the disk backend in the decoupled COW process due to the decoupling of the extra read requests, and the invalid areas are much likely written by the VM later so the base reads are unnecessary. Second, COWCache only caches the data that are accessed by the VM, which in turn allows for caching more data with real VM locality. By comparison, TRDCache eagerly completes the whole COW process in the critical I/O path and caches the amplified data in flash, which evicts out the data with real VM locality and induce more subsequent network reads and writeback. Moreover, COWCache not only produces less network traffic, but also reduces the contention to the storage server and network bandwidth.

5.3.4 Overhead

Memory overhead We replay the MSR traces and the memory overhead for flash caching in COWCache is 0.23–0.26%. If we also use 4-bit reference counter to differentiate hot and cold blocks like S-CAVE [25], the memory overhead can be further reduced to 0.14–0.17%, lower than 0.23% in S-CAVE and 0.5% in Mercury [5], which mainly benefits from the deduplication of the address information in memory.

Recovery time We replay the MSR traces, crash the VM, and recover the flash cache and virtual disk with fast/full recovery. For fast recovery, all need less than one second, since only the address mapping and cluster state

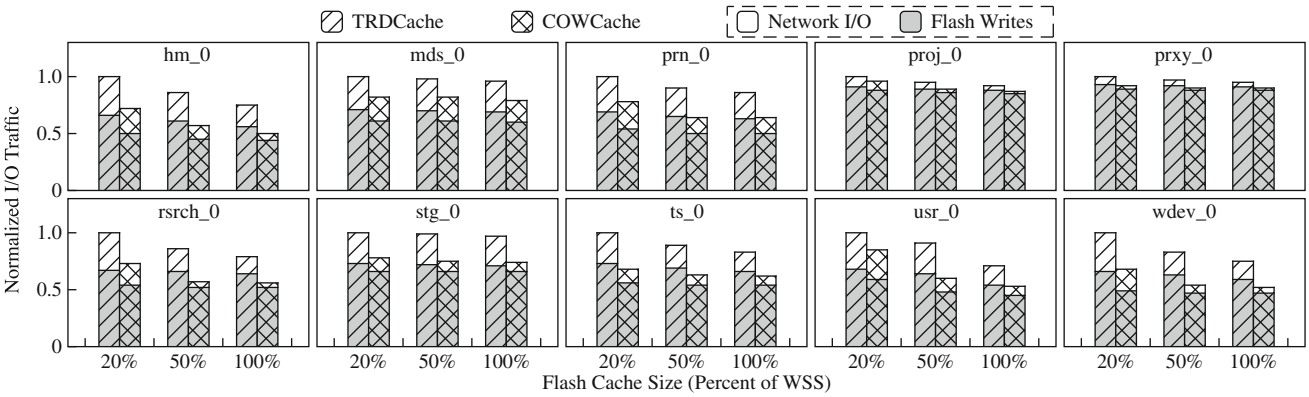


Fig. 15 Stacked traffic of flash writes and network I/O

information need to be read into memory to recover the cache. For full recovery, since the dirty data in flash cache need to be written back and the partially valid clusters in the delta disk still need to be filled with data from the base disk, so the recovery time is longer. The evaluation results show that the full recovery time for the MSR traces are 2.3–30.3 s, which depend on the workloads' pattern. E.g., for *prn_0*, the recovered dirty data set is 13.6 GB, and the recovery time is 30.3 s.

6 Related work

Deploying flash memory caching for virtual machines has been extensively researched in the literatures. They can be categorized into the following parts.

6.1 Caching layers

Mercury [5] makes a comprehensive discussion about the layers to deploy flash device, and chooses the hypervisor to manage flash cache *under* the Copy-on-Write layer. S-CAVE [25] also discusses the effectiveness of hypervisor-based flash caching and VMWare VAIO [43] has a similar architecture. Dm-cache [41], CloudCache [1] and CacheDedup [21] manage cache at the block level, which benefits for the generality for both virtualized and non-virtualized environments. Different from them, COWCache is not just hypervisor-based flash caching, but also manages cache in the COW layer, and bridges the semantic gap between the management of flash cache and virtual disk and fully exploits the virtual disk semantics to optimize flash caching.

6.2 Cache allocation and admission

S-CAVE proposes a rECS metric to determine the relative cache demand of different VMs, and then dynamically allocates space among them. vCacheShare [28] adopts a cache utility model and makes dynamic and automated flash cache space allocation based on multiple I/O access characteristics (e.g., locality changes). Centaur [18] relies on a workload's MRC to direct cache sizing and to achieve the QoS goals. CloudCache proposes a new cache demand model called Reuse Working Set to predict a VM's cache demand and to make cache allocation. CacheDedup and [9] use cache deduplication to reduce flash writes and to improve I/O performance. However, the I/O pattern they monitor to do cache allocation or admission has been changed greatly by COW-based virtual disks, which is very cache-unfriendly. COWCache makes co-design between flash caching and COW-based virtual disks to improve the caching efficiency. In addition, their policies can also be adopted together with COWCache.

6.3 Cache writeback policies

Mercury and S-CAVE adopt the writethrough policy, and vCacheShare use the writearound policy. In [17, 35], the authors discuss the problems of poor performance of writethrough and inconsistency potentials of writeback. They propose several writeback based caching policies to achieve both high performance and strong consistency. However, as discussed in front, while caching for COW-based virtual disks, even in writeback mode, the guest VM cannot fully experience the flash latency due to the Copy-on-Write amplification problem. COWCache solves the problem using the decoupled Copy-on-Write mechanism.

Different from traditional host-side flash caching, to the best of our knowledge, COWCache presents some new flash caching challenges induced by the virtual disk semantics, which have not been discussed in the literatures. Drop-in flash caching for COW-based virtual disks is inefficient, due to the cache-unfriendly management of COW-based virtual disks and the semantic gap between flash caching and virtual disk management. COWCache bridges the semantic gap and improves the flash caching efficiency by being aware of and utilizing the virtual disk semantics.

Optimizations for COW-based virtual disks are also researched. In [7], the authors uncover the sync amplification in COW-based virtual disks, and proposes two journaling approaches to mitigate the sync operations. However, the write amplification of metadata and data still exist. In [36], the authors use data cache to mitigate the COW penalties, but it only works for strictly sequential writes. FVD [40] is a new virtual disk format designed for both Cloud and non-Cloud environments. Selfie [44] states the significance of metadata for COW-based virtual disks, and proposes to mitigate the metadata writes by co-locating compressed metadata and data. Different from them, as flash caching is increasingly deployed for virtual disks, COWCache observes a good opportunity to use the non-volatile flash devices to enhance the metadata and data management of COW-based virtual disks, which achieves more improvement for the VM storage.

7 Conclusions

As flash caching is increasingly deployed for the VM disks expected to improve the VM storage performance, we uncover that while caching for COW-based virtual disks, it brings severe challenges of inefficient metadata caching and Copy-on-Write amplification, which makes the high-

performance flash caches underutilized and their endurance problem aggravated. We propose COWCache, a COW-aware flash caching solution to address the above challenges. First, COWCache is designed in the Copy-on-Write layer of the hypervisor to bridge the semantic gap between virtual disk and flash cache management and to enable cross-layer optimizations. Second, COWCache makes fine-grained metadata caching and journaling between memory cache and flash cache to improve the metadata management efficiency. Third, COWCache adopts a decoupled Copy-on-Write mechanism to decouple the amplified data processing from the critical I/O path and to cache only the data with real VM locality, which reduce unnecessary I/O processing and cache writes and improve the VM performance. Finally, as the design of COWCache is not specific to flash-based caching, we believe that the COWCache approach can also be applied to emerging non-volatile memory devices and improve their performance and endurance while used as caches for the VMs.

Acknowledgements The work is supported by the National Natural Science Foundation of China (Grant Nos. 61772300, 61832011) and National Science Foundation CAREER Award CNS-1619653, Award CNS-1562837, and Award CNS-1629888. Jianyu Fu thanks the China Scholarship Council (CSC) for the financial support.

References

1. Arteaga, D., Cabrera, J., Xu, J., Sundararaman, S., Zhao, M.: CloudCache: on-demand flash cache management for cloud computing. In: Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16), pp. 355–369. USENIX (2016)
2. Basu, G., Nadgowda, S., Verma, A.: LVD: lean virtual disks. In: Proceedings of the 15th International Middleware Conference (Middleware'14), pp. 25–36. ACM (2014)
3. Bellard, F.: QEMU, a fast and portable dynamic translator. In: Proceedings of the 2005 USENIX Annual Technical Conference (USENIX ATC'05), pp. 41–46. USENIX (2005)
4. Bhagwat, D., Patil, M., Ostrowski, M., Vilayannur, M., Jung, W., Kumar, C.: A practical implementation of clustered fault tolerant write acceleration in a virtualized environment. In: Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15), pp. 287–300. USENIX (2015)
5. Byan, S., Lentini, J., Madan, A., Pabon, L.: Mercury: host-side flash caching for the data center. In: Proceedings of the 28th International Conference on Massive Storage Systems and Technology (MSST'12), pp. 1–12. IEEE (2012)
6. Chen, J., Wang, J., Tan, Z., Xie, C.: Recursive updates in Copy-on-Write file systems-modeling and analysis. J. Comput. **9**(10), 2342–2351 (2014)
7. Chen, Q., Liang, L., Xia, Y., Chen, H., Kim, H.: Mitigating sync amplification for Copy-on-Write virtual disk. In: Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16), pp. 241–247. USENIX (2016)
8. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC'10), pp. 143–154. ACM (2010)
9. Feng, J., Schindler, J.: A deduplication study for host-side caches in virtualized data center environments. In: Proceedings of the 29th International Conference on Massive Storage Systems and Technology (MSST'13), pp. 1–6. IEEE (2013)
10. Hajnoczi, S.: An updated overview of the QEMU storage stack. In: LinuxCon Japan. The Linux Foundation (2011)
11. Handy, J.: Understanding the Intel/Micron 3D XPoint memory. In: Proceedings of Storage Developer Conference. SNIA (2015)
12. Hellwig, C.: The KVM/QEMU storage stack. In: Japan Linux Symposium. The Linux Foundation (2009)
13. Holland, D.A., Angelino, E., Wald, G., Seltzer, M.I.: Flash caching on the storage client. In: Proceedings of the 2013 USENIX Annual Technical Conference (USENIX ATC'13), pp. 127–138. USENIX (2013)
14. Huang, S., Wei, Q., Chen, J., Chen, C., Feng, D.: Improving flash-based disk cache with lazy adaptive replacement. In: Proceedings of the 29th International Conference on Massive Storage Systems and Technology (MSST'13), pp. 1–10. IEEE (2013)
15. Jens, A.: FIO—flexible I/O tester. https://github.com/axboe/fio (2005)
16. Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: KVM: the linux virtual machine monitor. In: Proceedings of the 2007 Linux Symposium, pp. 225–230 (2007)
17. Koller, R., Marmol, L., Rangaswami, R., Sundararaman, S., Talagala, N., Zhao, M.: Write policies for host-side flash caches. In: Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST'13), pp. 45–58. USENIX (2013)
18. Koller, R., Mashtizadeh, A.J., Rangaswami, R.: Centaur: host-side SSD caching for storage performance control. In: Proceedings of the 2015 IEEE International Conference on Autonomic Computing (ICAC'15), pp. 51–60. IEEE (2015)
19. Lagar-Cavilla, H.A., Whitney, J.A., Scannell, A.M., Patchin, P., Rumble, S.M., De Lara, E., Brudno, M., Satyanarayanan, M.: SnowFlock: rapid virtual machine cloning for cloud computing. In: Proceedings of the 4th ACM European Conference on Computer Systems (EuroSys'09), pp. 1–12. ACM (2009)
20. Le, D., Huang, H., Wang, H.: Understanding performance implications of nested file systems in a virtualized environment. In: Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12), pp. 87–100. USENIX (2012)
21. Li, W., Jean-Baptise, G., Riveros, J., Narasimhan, G., Zhang, T., Zhao, M.: CacheDedup: in-line deduplication for flash caching. In: Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16), pp. 301–314. USENIX (2016)
22. Lu, Y., Shu, J., Zheng, W.: Extending the lifetime of flash-based storage through reducing write amplification from file systems. In: Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST'13), pp. 257–270. USENIX (2013)
23. Lu, Y., Shu, J., Wang, W.: ReconFS: a reconstructable file system on flash storage. In: Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST'14), pp. 75–88. USENIX (2014)
24. Lu, T., Huang, P., He, X., Zhang, M.: Understanding the impact of cache locations on storage performance and energy consumption of virtualization systems. In: Proceedings of the 2016 USENIX Workshop on Cool Topics on Sustainable Data Centers (CoolDC'16). USENIX (2016)

25. Luo, T., Ma, S., Lee, R., Zhang, X., Liu, D., 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 (PACT'13), pp. 103–112. IEEE (2013)
26. McDougall, R., Jim, M.: Filebench. https://github.com/filebench/filebench (2004)
27. McLoughlin, M.: The QCOW2 image format. https://people.gnome.org/~markmc/qcow-image-format.html (2008)
28. Meng, F., Zhou, L., Ma, X., Uttamchandani, S., Liu, D.: vCacheShare: automated server flash cache space management in a virtualization environment. In: Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC'14), pp. 133–144. USENIX (2014)
29. Mesnier, M., Chen, F., Luo, T., Akers, J.B.: Differentiated storage services. In: Proceedings of the 23th ACM Symposium on Operating Systems Principles (SOSP'11), pp. 57–70. ACM (2011)
30. Meyer, D.T., Aggarwal, G., Cully, B., Lefebvre, G., Feeley, M.J., Hutchinson, N.C., Warfield, A.: Parallax: virtual disks for virtual machines. In: Proceedings of the 3th ACM European Conference on Computer Systems (EuroSys'08), pp. 41–54. ACM (2008)
31. Microsoft: VHDX format specification v1.00. https://www.microsoft.com/en-us/download/details.aspx?id=34750 (2012)
32. Narayanan, D., Donnelly, A., Rowstron, A.: Write off-loading: practical power management for enterprise storage. In: Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST'08), p. 17. USENIX (2008)
33. Oh, M., Eom, H., Yeom, H.Y.: Enhancing the I/O system for virtual machines using high performance SSDs. In: Proceedings of the 2014 International Performance Computing and Communications Conference (IPCCC'14), pp. 1–8. IEEE (2014)
34. Pfaff, B., Garfinkel, T., Rosenblum, M.: Virtualization aware file systems: getting beyond the limitations of virtual disks. In: Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI'06), pp. 353–366. USENIX (2006)
35. Qin, D., Brown, A.D., Goel, A.: Reliable writeback for client-side flash caches. In: Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC'14), pp. 451–462. USENIX (2014)
36. Reitz, M., Wolf, K.: Qcow2—why (not)? In: KVM Forum. The Linux Foundation (2015)
37. Ribot, F.Z.: QLOOP: Linux driver to mount QCOW2 virtual disks. PhD thesis (2010)
38. Saxena, M., Swift, M.M., Zhang, Y.: Flashtier: a lightweight, consistent and durable storage cache. In: Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys'12), pp. 267–280. ACM (2012)
39. Shafer, J.: I/O virtualization bottlenecks in cloud computing today. In: Proceedings of the 2nd Workshop on I/O Virtualization (WIOV'10), pp. 5–5. USENIX (2010)
40. Tang, C.: FVD: a high-performance virtual machine image format for cloud. In: Proceedings of the 2011 USENIX Annual Technical Conference (USENIX ATC'11), pp. 18–18. USENIX (2011)
41. Van Hensbergen, E., Zhao, M.: Dynamic policy disk caching for storage networking. IBM technical report RC24123 (2006)
42. VMware: virtual disk format 5.0. https://www.vmware.com/support/developer/vddk/vmdk_50_technote.pdf (2011)
43. VMware: VMware vSphere APIs for I/O filtering (VAIO). https://storagehub.vmware.com/export_to_pdf/vmware-vsphere-apis-for-i-o-filtering-vaio (2017)
44. Wu, X., Shao, Z., Jiang, S.: Selfie: co-locating metadata and data to enable fast virtual block devices. In: Proceedings of the 8th International Systems and Storage Conference (SYSTOR'15), p. 2. ACM (2015)
45. Yang, J., Plasson, N., Gillis, G., Talagala, N.: HEC: improving endurance of high performance flash-based cache devices. In: Proceedings of the 6th International Systems and Storage Conference (SYSTOR'13), p. 10. ACM (2013)

Jianyu Fu is a Ph.D. candidate in the School of Computer at National University of Defense Technology, China. He received the B.S. degree in Software Engineering from Nankai University, China in 2013, and the M.S. degree in Computer Science from National University of Defense Technology, China in 2015. His research interests include storage systems and cloud computing.

Youyou Lu is an assistant professor in the Department of Computer Science and Technology at Tsinghua University, China. His current research interests include storage systems, distributed systems and computer architecture. He received the B.S. degree in Computer Science from Nanjing University, China in 2009, and the Ph.D. degree in Computer Science from Tsinghua University, China in 2015. He is a Member of the IEEE.

Jiwu Shu received the Ph.D. degree in Computer Science from Nanjing University, China in 1998, and finished the postdoctoral position research at Tsinghua University, China in 2000. Since then, he has been teaching at Tsinghua University. His current research interests include distributed (network/cloud/big data) storage systems, non-volatile memory systems and technologies, reliability for storage systems, parallel and distributed processing technology. He is a IEEE Fellow.

Guangming Liu received the B.S. and M.S. degrees in Computer Science from National University of Defense Technology, China in 1980 and 1986 respectively. He is now a professor in the School of Computer at National University of Defense Technology. His research interests include high performance computing, massive storage, and cloud computing.

Ming Zhao received the B.S. and M.S. degrees in Automation/Pattern Recognition and Intelligent Systems from Tsinghua University, China in 1999 and 2001 respectively, and the Ph.D. degree in Electrical and Computer Engineering from the University of Florida, USA in 2008. He is now an associate professor in the School of Computing, Informatics, and Decision Systems Engineering at Arizona State University, USA. His research interests include distributed/cloud computing, high-performance computing, virtualization, storage systems, and operating systems.