PLC-Cache: Endurable SSD Cache for Deduplication-based Primary Storage

Jian Liu[1,2,3], Yunpeng Chai[1,2*], Xiao Qin[4], and Yuan Xiao[2]

[1]Key Laboratory of Data Engineering and Knowledge Engineering, MOE, China
[2]School of Information, Renmin University of China, Beijing, China
[3]National Computer System Engineering Research Institute of China, Beijing, China
[4]Department of Computer Science and Software Engineering,
Samuel Ginn College of Engineering, Auburn University, Auburn, AL, USA
*Corresponding author: ypchai@ruc.edu.cn
liujian090032@163.com, xqin@auburn.edu, xiaoyuan@ruc.edu.cn

***Abstract*—Data deduplication techniques improve cost efficiency by dramatically reducing space needs of storage systems. SSD-based data cache has been adopted to remedy the declining I/O performance induced by deduplication operations in the latency-sensitive primary storage. Unfortunately, frequent data updates caused by classical cache algorithms (e.g., FIFO, LRU, and LFU) inevitably slow down SSDs' I/O processing speed while significantly shortening SSDs' lifetime. To address this problem, we propose a new approach – PLC-Cache – to greatly improve the I/O performance as well as write durability of SSDs. PLC-Cache is conducive to amplifying the proportion of the Popular and Long-term Cached (PLC) data, which is infrequently written and kept in SSD cache in a long time period to catalyze cache hits, in an entire SSD written data set. PLC-Cache advocates a two-phase approach. First, non-popular data are ruled out from being written into SSDs. Second, PLC-Cache makes an effort to convert SSD written data into PLC-data as much as possible. Our experimental results based on a practical deduplication system indicate that compared with the existing caching schemes, PLC-Cache shortens data access latency by an average of 23.4%. Importantly, PLC-Cache improves the lifetime of SSD-based caches by reducing the amount of data written to SSDs by a factor of 15.7.**

Keywords—Deduplication; SSD; Cache; Endurance; Primary Storage

I. INTRODUCTION

A study conducted by International Data Corporation (IDC) [29] concluded that 2.8ZB of data has been created and replicated in 2012; IDC predicted that the global data will reach 40ZB by 2020 [3]. Nearly 75% of the data is duplicated in our digital world [2] and; as such, data deduplication techniques [4, 5, 6, 7, 8] are deployed in modern storage systems to reduce cost and improve space-efficiency. Previously, data deduplication is mostly applied in secondary storage systems used for backup or archival purposes; these systems pay attention to high throughput of sequential accesses. Recently, the benefits of data deduplication have been extended to a wider range of storage systems, including primary storage like email, web, multimedia, and database systems (see, for example, iDedup [4], ZFS [37], SDFS [34], LessFS [32], and LBFS [11]).

Compared with secondary storage, primary storage systems aim at improving latency performance for good user experience. A big challenge is that scattered data distributions on disks coupled with additional deduplication operations (e.g., file chunking, fingerprint calculating, and looking up) inevitably increase I/O access latency. Some solutions like iDedup [4] have been proposed to decrease the latency of primary deduplication-based storage by reducing disk fragmentation. Furthermore, flash-based solid state drives or SSDs, which is much faster than hard drives, are a good candidate to significantly boost the performance of deduplication-based primary storage. SSDs with storage capacity as comparable those of hard drives can be adopted to store deduplication metadata; SSDs also may perform as a data cache of data chunks or containers [13]. An increasing attention has been paid toward the application of SSD cache to the secondary storage systems (see, for example, Dedupv1 [18], ChunkStash [6], and SAR [7]).

A. Challenges of SSD cache in primary deduplication

Different from the sequential restoring process in a backup storage (a.k.a., secondary deduplication), the primary deduplication faces random I/O requests issued by users. In other words, it is impossible to determine the most appropriate hot data blocks for SSD cache according to block popularity information collected in the backup phase (see SAR [7] for details). Caching algorithms are responsible for dynamically and frequently updating cached contents in SSDs according to the changes in data popularity, thereby achieving a high SSD-cache hit rate. When it comes to SSD cache of primary deduplication, an excessive number of SSD writes caused by cached data updates lead to two serious challenges:

1) Frequent write operations on SSD cache may degegrade the overall access performance of the primary storage due to the request congestion problem.

2) An excessive number of writes tend to quickly wear SSDs, because there is a limited number of erase operations per block .

Taking the 60GB Intel 520 SSD [22] for example, its total bytes written (TBW - the total allowed written amounts before wearing out) is approximately 36,500 GB. In a preliminary

978-1-4799-5671-5/14/$31.00 © 2014 IEEE

experiment, we configured a 60GB Intel 520 SSD as a data cache for a deduplication system, where the available capacity of SSD cache is 5% of the deduplicated data. After measuring SSD cache's write speed and TBW of Intel 520, we estimate the expected SSD lifetimes when FIFO, LRU, and LFU are respectively adopted (see Table 1).

Table 1 indicates that the SSD cache quickly wears out (i.e., in about eight days) when the conventional caching algorithms (e.g., FIFO, LRU, and LFU) are employed. Although expensive enterprise-class SSD products have long endurance, there is a lack of enough write endurance for SSD cache. For instance, TBW of the 400GB Intel 910 SSD - an off-the-shelf enterprise-class SSD - is 5 PB [12]. Thus, if five-year lifetime is expected, an Intel 910 SSD exhibits an average writing speed lower than 34.1 MB/s, which is less than the performance measured in our preliminary experiment (see Table 1). For a production storage system, SSD cache can be worn out quickly due to a high writing speed.

To enlarge SSD cache's lifetime in the context of deduplication-based primary storage, we aim at designing a cache replacement scheme to make SSD cache endurable by alleviating write pressure and achieving a high hit rate.

B. Basic idea of endurable SSD cache

In this paper, we classify data blocks written into SSD cache into four quadrants according to data blocks' popularity and their repeat counts. A block's popularity is quantified in terms of access frequency; the repeat writes of a block indicate the counts of writing the same block into SSD cache during an entire service time (see Fig. 1). Data represented in the left upper quadrant exhibit high popularity and low repeat writes (see quadrant *A* in Fig. 1); SSD cache should give high priority to this type of data referred to as popular and long-term cached data or PLC data throughout this paper. PLC data create ample opportunities to reduce SSD writes while maintaining a high cache hit rate. After evaluating SSD-write performance of the traditional FIFO, LRU, and LFU algorithms (see Section 3 for details), we observe that these three policies lead to very low proportions (i.e., less than 4.22%) of PLC data. Thus, most of the SSD written data has very low efficiency of yielding numerous SSD cache hits, thereby resulting in low performance and short lifetime of SSD devices.

To address this issue, we propose a PLC-Cache algorithm, which aims at increasing the proportion of PLC data in the entire SSD written data set. PLC-Cache is composed of a series of techniques (i.e., *Popularity Filter*, *Delayed Eviction*, and *In-Out Comparison*, see Section 4 for more details) of enlarging the amount of PLC data. To improve cache hits, PLC-Cache makes an effort to move data blocks located in quadrant *B* into quadrant *A*, while preventing data in quadrants *C* and *D* from being loaded into the SSD cache. Our experimental results show that PLC-Cache can significantly boost deduplication-based primary storage by 40%~50% with SSDs having only 20% of the capacity of the deduplicated data in disks. Moreover, PLC-Cache outperforms the traditional caching algorithms (e.g., LRU, LFU, FIFO) by 23.4% on average. Importantly, PLC-Cache reduces the total amount of data written to SSD cache by a factor of 15.7 on average compared with the traditional algorithms.

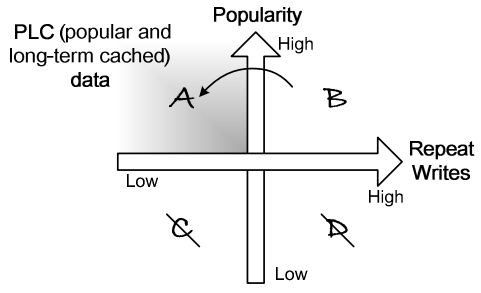


Fig. 1 **SSD cache should give top priority to popular and long-term cached or PLC data (i.e. quadrant A) with high popularity and low repeat counts;** the other quadrants' data should be converted to PLC data or excluded from being loaded into SSD cache.

Table 1: A 60GB Intel 520 SSD performed as a data cache will wear out in approximately 8 days under continuous usage if the traditional cache algorithms (e.g., LRU, LFU, and FIFO) are adopted.

Cache algorithm	The measured writing speed of SSD (MB/s)	Expected SSD lifetime (Days)
FIFO	56.7	7.6
LRU	50.7	8.5
LFU	53.1	8.1

Note: The writing speed of different cache algorithms is measured on our test bed. The expected lifetime is calculated with an assumption that the SSD cache is continuously active.

The rest of the paper is organized as follows. Section II describes background knowledge as well as related work. The characteristics of data written into SSD cache are analyzed in Sections III, which leads to our basic idea of reducing SSD writes. Then Section IV gives a description of our proposed PLC-Cache. An extensive performance evaluation based on a real-world deduplication system is exhibited in Section V. Finally, the last section concludes our paper.

II. Background and Related Work

A. Data Deduplication

1) Basic data deduplication

Data deduplication conserves storage capacity by detecting data redundancy and storing only one copy of the redundant data. When storing data, a system first splits the data stream into numerous data chunks using the content-based Rabin chunking algorithm [25], and then calculates the fingerprint (i.e., unique identifier) of each data chunk using the SHA-1 hash function. Recently proposed solutions combine many unique data chunks into a container with fixed size (e.g. 4MB) to reduce disk fragmentation. As Fig. 2 shows, the metadata of data deduplication includes *Chunk Index* (i.e., the mapping of all the fingerprint and related container ID), and *File Recipe* (i.e., the related data chunks' fingerprints of all the files). In the process of output, the system first reads *File Recipe* for the chunks' fingerprints of a target file, and then searches the related container IDs in *Chunk Index*. Finally, related containers are read to construct a target file.

Data deduplication can be classified into groups, namely, primary and secondary deduplication. Primary data deduplication is implemented in a native file system on a host or a cluster supporting deduplication operations [1]. Secondary storage usually refers to data backup or archive systems; its storage capacity is usually 10-20x [13] than primary storage.

2) Secondary Deduplication

Secondary deduplication systems adopt a standard deduplication method. A secondary storage system - facilitating data backup and restore operations - requires high I/O throughput.

Both Sparse Indexing [5] and SiLo [14] efficiently affiliate disk bottleneck through exploiting similarity among data segments. Deduplication is performed if two data segments are similar to each other. Nam *et al.* introduced a Chunk Fragmentation Level (CFL) monitor [31]. If the CFL monitor indicates a chunk fragmentation is becoming worse, the monitor will selectively rewrite some chunks to reduce fragmentation, thereby achieving high restore performance.

Dedupv1 [18] and ChunkStash [6] store metadata in flash memory for chunk index exploiting high random-read performance of SSD to accelerate chunk index. SAR [7] stores unique data chunks in SSDs with high reference count exploiting high random-read performance of SSDs to greatly improve restore performance.

3) Primary Deduplication

Data of emails, multimedia, and databases is stored in primary deduplication systems, which are frequently accessed by users in a random fashion. Compared with data restore in secondary systems, primary storage systems simply read files demanded by users each time rather than restoring the entire dataset. As such, the reading latency can be significantly reduced.

iDedup [4] exploits spatial locality to deduplicate data sequence that are contiguously stored on a disk to reduce disk fragmentation, thereby improving reading speed. What is more, iDedup takes advantage of temporal locality with a smaller memory to cache metadata and; therefore, iDedup dramatically reduces disk I/Os and improves writing speed. More importantly, data deduplication has been integrated into practical storage products (e.g., NetApp ASIS [35] and EMC Celerra [36]) and file systems (e.g. ZFS [37], SDFS [34], LessFS [32], and LBFS [11]).

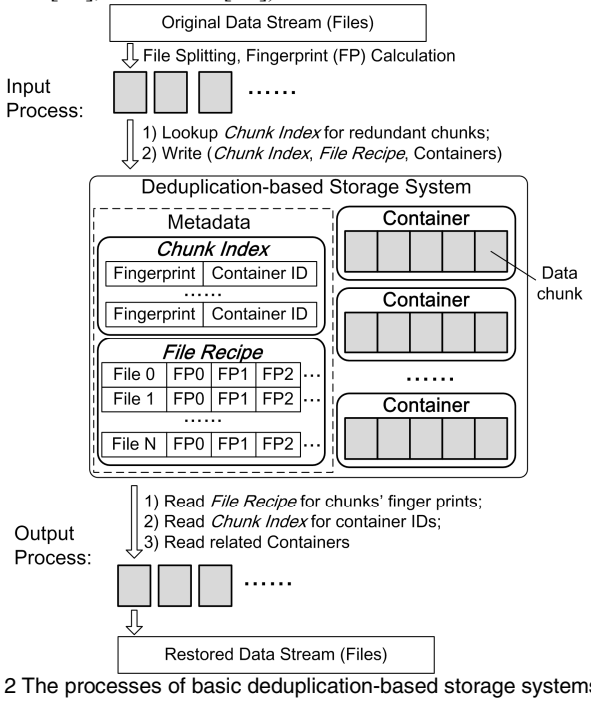


Fig. 2 The processes of basic deduplication-based storage systems.

B. SSD Endurance

SSDs are favored by IT companies and end users thanks to SSDs' low access latency, high energy efficiency, and high storage density. However, the most serious challenge of SSDs lies in limited write endurance (i.e., SSDs only stand very limited number of write bytes) [27]. This problem is caused by two reasons:

1) Storage units inner flash chips have to be erased before any re-write operation. To pursue high storage density and low cost, mainstream SSDs have adopted multi-level cell flash (MLC) instead of single-level cell (SLC). A SLC flash memory array typically supports approximately 100,000 erasures cycles for each basic unit. The value of MLC flash, which can store more than one bit in each unit, drops as low as 5,000 ~ 10,000 cycles or even lower [16, 28].

2) Write amplification. Each erase-unit inner flash chips generally contains hundreds of pages – a basic read/write unit of flash [28]. When an erase-unit – containing mixed valid pages and invalid pages (old version of updated data) – is about to be erased, the valid pages must be written to other place, thereby imposing extra writes. Therefore, the amount of data written on flash chips of an SSD are much larger than the amount of data issued to the SSD.

In recent years, a considerable amount of research has been done to enhance SSD endurance. For example, Griffin employed hard disk drives or HDDs as a write cache for SSDs to coalesce overwrites, aiming to significantly reduce write traffic to the SSDs [16]. Chen *et al.* proposed a hetero-buffer, which consists of DRAM and reorder area [17]. DRAM devotes to improving hit ratio, reorder area aims at exhibiting write amplification. Both of these techniques allow SSDs to reduce the number of erased physical blocks. I-CASH arranges SSDs to store seldom-changed and mostly read reference data, whereas a HDD stores a log of changed deltas of SSD data [30]. In the I-CASH case, SSDs obviate the need of handling random writes. Kim *et al.* apply data deduplication technology in SSDs to reduce write amplification effects [10].

C. Cache Management for SSDs

Traditional cache replacement algorithms (e.g., FIFO, LRU, LFU, LIRS [15], and ARC [9]) make great efforts in obtaining high cache hit rates by frequently updating cached contents without any restriction. The write endurance of SSD products is inadequate to meet such excessive write requests, which inevitably lead to a short lifetime of the SSD cache (see an example in Table 1). To overcome this problem, vendors and researchers have proposed various solutions that exploit application characteristics to limit the number of writes to SSD cache.

For example, Oracle Exadata's Smart Flash Cache [24] combines the cache management and database logic together, skipping some unimportant data for Flash cache to reduce SSD writes. Netapp [33] - employing SSDs as read cache - categories data into different priorities, namely, metadata, normal data, and low-priority data. SSDs can be configured to only cache data with specified priorities, with the result of reducing write load. LARC [19] manages a virtual LRU queue with a limited length prior to SSD cache; hitting the LRU queue is the only condition of entering SSD to reduce the amount of written data. EMC's FAST Cache [20], Intel's

Turbo Memory (ITM) [21], and SieveStore [38] use reference counts to limit SSD writes.

In the realm of data deduplication, little attention has been paid toward applying SSD cache to boost performance while addressing the write endurance problem of SSDs. Therefore, the endurable SSD cache algorithm in deduplication-based storage systems is still an important and open research field.

III. Analysis of SSD Write Reduction

In order to reduce the amount of written data to SSD cache for extending the lifetime of SSDs, we need to keep track of components of written data first. In this section, let us classify all written data of cache into several categories based on the analysis of real-system traces.

A. Four Quadrants of Cache's Written Data

We classify all the data written into SSD cache into four quadrants (i.e. the quadrants A, B, C, and D in Fig. 1), according to data blocks' access counts and their repeat write counts, which are the counts of writing one block into SSD during the entire service time. A high repeat count means a block is often evicted from cache, and then written into cache again for many times.

1) Popular and Long-term Cached (PLC) Data

The data in quadrant A or QA, referred to as popular and long-term cached (i.e., PLC) data, exhibit high access counts and low repeat counts. After a PLC data block is fetched into cache, the block resides in the cache for a long time period to yield a large number of cache hits. PLC data is the best candidates to be loaded into SSD cache, because caching PLC data is likely to boost I/O performance by obviating the need of writing SSDs.

2) Popular and Short-term Cached Data

Like PLC data, the popular and short-term cached data (see Quadrant B or QB in Fig. 1) induces many accesses on average for each data block. However, PLC data differs from the popular and short-term cached data in that PLC data introduce low repeat counts. The short-term cached data have two salient features: (1) They are frequently fetched to SSD cache thanks to their high popularity; (2) they are likely to be evicted by caching algorithms frequently due to a combination of reasons (e.g., a long time since the last access). Keeping QB data in SSDs tends to improve performance because of high cache hit ratio; on the other hand, short-term cached data seriously and adversely affect SSDs' lifetime due to numerous writes.

3) Non-popular Data

The data in quadrant C (i.e. QC) and quadrant D (i.e. QD) both have low access counts, so they both belong to non-popular data. Each time when a non-popular data block enters cache, few or no hits are generated before it is evicted due to a long idle time. Non-popular data is of little value to improve performance, so they should be kept out of SSD cache.

B. Proportions of the Four Quadrants

Ideally, SSD cache should be in favor of PLC data and against the other three types of data. In what follows, we perform an analysis of the proportions of the four quadrants of data in SSD cache according to the classical cache replacement algorithms (i.e., FIFO, LRU, and LFU).

We replayed real-world traces collected from typical applications (see Table 2 for more details) and a synthetic trace of randomly accessing the collected data from the servers of Renmin University of China under the Zipf distribution (marked as *rucserver*). The real-world traces include *buildserver* (a Windows build server trace), *cctvvod* (a video-on-demand trace from *cctv.com*), and *hive-select*, which is collected by running BigDataBench [39] to perform a cloud database test. We studied the distribution of the aforementioned three types of data. In the trace analysis, we set the cache size as 5% of the storage capacity, and set the thresholds of "Access Count" and "Repeat Count" as 5 both to split the four quadrants.

Fig. 3 illustrates the proportions of the three types of written data under the four traces driven by FIFO, LRU, and LFU, respectively. The first four columns in Fig. 3 illustrate the results of caching in the unit of small data chunk (e.g. 4KB) of deduplication for the four traces, while the last columns shows the result of caching big containers (e.g. 2MB) for *rucserver*. We make the following observations on the distributions of the four quadrants of data:

(1) In all the situations, the proportion of PLC data (i.e. quadrant A) is very low, between 0% and 4.22%. Some traces have high proportions of QB data, while others have high proportions of QC+QD data.

(2) Comparing the two columns of *rucserver* results in chunk-level and container-level caching, respectively, we have the following findings:

- ✧ The proportion of QA data in container-level caching is smaller compared with chunk-level, because some data chunks from QB or QD mixed with QA data chunk makes the entire container locate in QB.
- ✧ The proportion of QC+QD data in container-level caching becomes smaller too, because mixing some QD data chunks together, mixing QC/QD data chunks with QB ones, or mixing QD data chunks with QA ones will all make the entire container locate in QB too.
- ✧ Therefore, the proportion of QB obviously increases.

(3) According to the above difference between chunk-level and container-level caching, and the proportion distributions of the real-world traces, we can conclude that the proportion of QA data in container-level caching of these real-world trace will be lower further.

(4) Due to such a small proportion of PLC data (i.e. QA data), the popular and short-term cached data or QB data plays a decisive role for total cache hits. Therefore, an ideal caching policy should not prevent QB data from being loaded into SSD. We discover that an optimized solution is to move QB data to QA; thus, making a massive amount of PLC data in SSD cache in the long term without repeatedly evicting and reloading popular data.

(5) Regardless of the tested traces, the proportion distributions among FIFO, LRU and LFU are quite similar. Thus, the classical caching algorithms are close to each other

Table 2: Characteristics of selected real-world workload traces.

Trace Name	Application Type	Req Number
buildserver	Windows Build Server	957,596
cctvvod	Search Engine	5,047,596
hive-select	Cloud Database	419,723

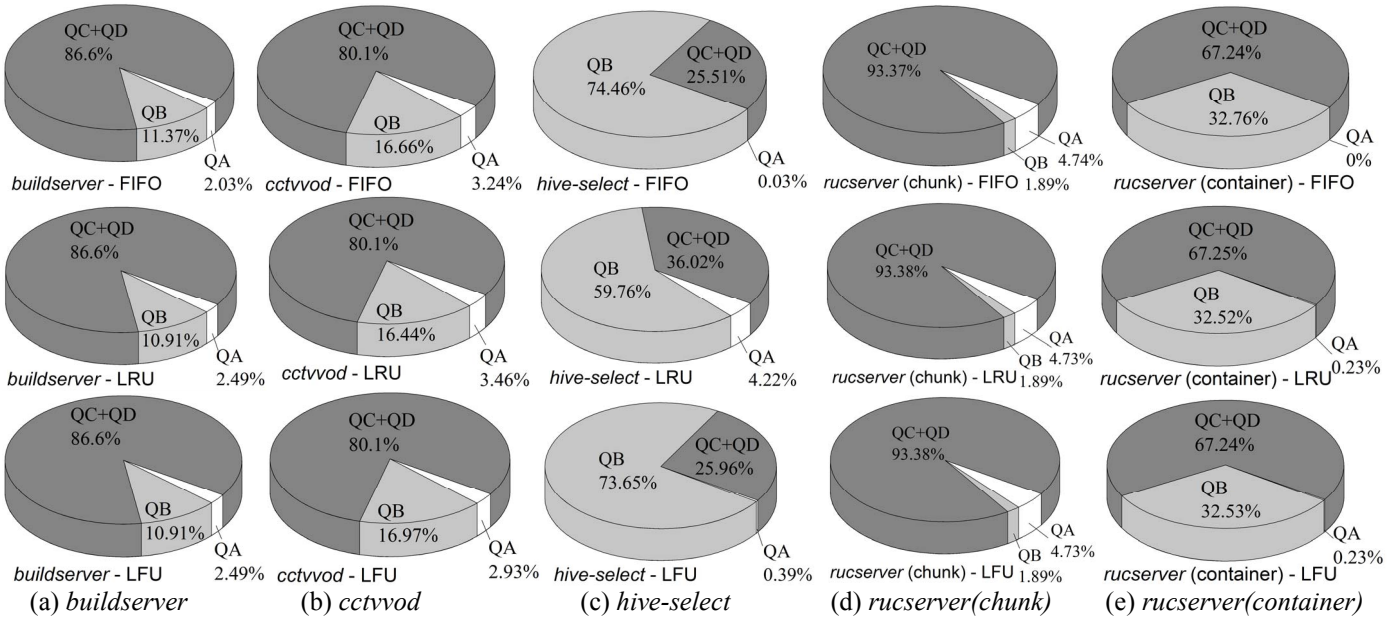


Fig. 3 Proportions of the data in the four quadrants under various traces coupled with respectively FIFO, LRU, and LFU.

in the terms of affecting proportions of the three types of data. The above observations motivate us to design a novel caching algorithm aiming to reduce the number of writes to SSD cache and to promote the proportion of PLC data at the same time.

IV. PLC-CACHE

In this section, an overview of our proposed PLC-Cache algorithm will be given in the first part to present the principles and working steps of SSD written data reduction. And then, three important modules of PLC-Cache, i.e. *Popularity Filter*, *Delayed Eviction*, and *In-Out Comparison*, will be respectively introduced in the following subsections. Finally, we will exhibit some implement details of PLC-Cache based on a practical system.

A. Overview of PLC-Cache

Storing QA data with both high popularity and low writing amounts in SSD cache is the key of achieving both high cache hit rates and long lifetime of SSDs. Based on this idea, we design a new PLC-Cache algorithm, which aims to reduce the total SSD written data amounts and to increase the proportion of PLC data in the SSD written data set, by excluding some obviously unpopular data, and converting some potential PLC data into real PLC data through a series of mechanisms.

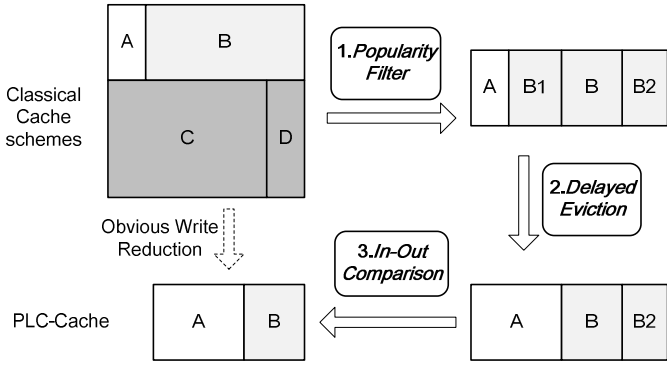


Fig. 4 **Steps of SSD write reduction in PLC-Cache:** 1) ***Popularity Filter*** – kicking QC and QD data from SSD written data set; 2) ***Delayed Eviction*** – transforming QB1 data to QA data; 3) ***In-Out Comparison*** – keeping QB2 data away from SSD cache.

Fig. 4 shows the steps of reducing unnecessary written data and improve the average benefits of each SSD written block in our proposed PLC-Cache. They are implemented by the following three key modules:

1) *Popularity Filter* is responsible for preventing the low-value data (e.g. data in quadrant C and D) from entering SSD cache. The trace analysis in the last section indicates that the majority of the written data to cache usually locates in quadrant C and D. *Popularity Filter* helps to reduce the SSD write amounts directly, and also improves the average quality of cached data in SSD.

2) *Delayed Eviction* plays the role of delaying the evicting time of cache data in SSDs. Part of data blocks in quadrant B (i.e. QB1) have high popularity in the long term. It does not fall into quadrant A, only because its occasional two adjacent accesses are far apart from each other. They have potential of falling into quadrant A. Delayed eviction can increase the proportion of QA data by reduce the redundant writes of QB1data.

3) *In-Out Comparison* is designed for another kind of data in quadrant B (i.e. QB2). One reason that the QB data has high repetitive write counts of one data block on average lies in that many data blocks with similar popularity replace each other in rotation. This consumes the write endurance of SSD consistently, but don't bring additional benefits of the cached data. Therefore, *In-Out Comparison* is a mechanism to prevent this to happen. When cache replacement is going to be performed, we compare the internal data of SSD cache with the external incoming data. Only when the external data is indeed significantly better than the internal data, PLC-Cache replaces the internal data with external one; otherwise, cache replacement will not happen.

B. Organization and Modules of PLC-Cache

PLC-Cache manages the SSD cache in the unit of deduplication containers [13], which has fixed size (e.g., 1MB

~ 8MB). This leads to two benefits: Firstly, the access manner of overwriting SSDs in the unit of large containers can weaken the fragmentation of garbage data inner Flash chips of SSDs. Thus the write performance of SSD can be improved, and the write endurance of SSDs can also be extended because of decreasing the write amplification of SSDs. Secondly, when the popular capping technique [8] is employed in data deduplication, each file is only related to very limited containers in hard disk drives (HDDs), with the results of less HDD seeking time for better performance. Caching complete containers in SSD cache can further reduce the seeking counts and access time of HDD.

PLC-Cache's organization is illustrated in Fig. 5. It employs a typical clock algorithm to manage the cached blocks (i.e. containers) in SSDs. Furthermore, three modules (i.e. *Popularity Filter*, *Delayed Eviction*, and *In-Out Comparison*) are designed around the basic clock to reduce the SSD write amounts and improve the average quality of cached data (i.e. increasing the proportion of the PLC data).

The access information of each container recorded by PLC-Cache is summarized in Table 3. Among these properties, *access_count* and *reference_count* of all the containers in the storage system are maintained in memory, while only the cached containers in SSD cache record *inner_priority* and *delay_count*. Assuming the size of one container is 2MB and the capacity of deduplication-based primary storage is 4TB, there are 2M containers all together. If the size of container ID, the priorities of *access_count* and *reference_count* are all 4 bytes, 24 MB (2M*3*4B) metadata for recording the information is needed in memory. Therefore, maintaining the information in memory is not a burden.

When an accessed container (e.g., the container *b* in Fig. 5) is missed by cache, it has to pass the following three modules of PLC-Cache in order to enter SSD cache physically. The three modules exclude most of the low-benefit containers from writing into SSD, and simultaneously convert other kinds of data into PLC data as much as possible. The detailed processing of the three modules of PLC-Cache is discussed below:

1) Popularity Filter

Because of the limited write endurance, SSD cache is only appropriate for the containers with good and lasting popularity. The history access count of containers is a simple but effective recognizer of these long-term hot containers and the other. Therefore, PLC-Cache sets an appropriate threshold of *access_count* in *Popularity Filter*. Any missed containers (e.g., the container *b* in Fig. 5) with *access_count* higher than the threshold can pass through the first module. If the threshold is set properly, the situation shown in Fig. 1 will appear, i.e. both

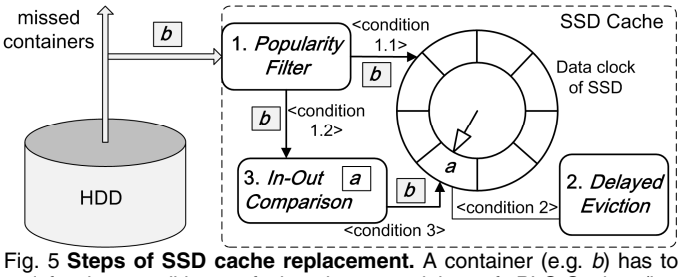


Fig. 5 **Steps of SSD cache replacement.** A container (e.g. *b*) has to satisfy the conditions of the three modules of PLC-Cache (i.e. *Popularity Filter*, *Delayed Eviction*, and *In-Out Comparison*) to enter SSD cache.

Table 3: Properties that need to be recorded for each container.

Properties	Description
access_count	The total access count in the past.
reference_count	The file number related to this container collected during the deduplicating files.
*inner_priority**	The priority of staying in the clock of SSD cache. If a container's *inner_priority* is 0, it may be replaced soon. It increases in the case of cache hit, and decreases when the clock sweeps over it.
*delay_count**	It indicates whether the selected victim of SSD cache (whose inner_priority is 0) should be replaced or not. When *delay_count* is not 0, the selected container's eviction is delayed, with the value of *delay_count* decreased.

Note: the properties marked with "*" are only for the cached containers in SSD.

QA and QB data with high *access_count* can pass through the *Popularity Filter*, while QC and QD with low *access_count* are kept out by the filter. In addition, the *access_count* of a container evicted from SSD cache will be cleared to be 0 to avoid cache pollution of the previously hot data.

When a missed container (e.g., container *b*) satisfies the above condition, there are two possibilities here:

a) If there is free space in SSD cache to accommodate new incoming data, we write container *b* into SSDs directly. In this case, the following two modules (i.e. *Delayed Eviction* and *In-Out Comparison*) will not be enabled in this round (See <condition 1.1> in Fig. 5).

b) After the SSD cache is full, container *b* has to replace a cached container inner SSD. Of course, this process is much stricter. Container *b* has to pass the following two modules of PLC-Cache for entering SSD physically (See <condition 1.2> in Fig. 5).

2) Delayed Eviction

PLC-Cache utilizes a classical clock algorithm to manage the cached containers in SSDs for low overhead. Each time when a container is written into SSD cache, it will be inserted into the circular list of clock algorithm, with an initial value of *inner_priority* according to its hotness. The value of *inner_priority* of a container increases by 1 if it is hit. When PLC-Cache looks for a victim to evict, the clock pointer scans the containers in the ring one by one, decreasing each container's *inner_priority* by 1, until a container whose *inner_priority* is 0 is found.

Different with classical clock algorithm, PLC-Cache set an additional priority, i.e. *delay_count,* for each cached container. Each time when a container victim is selected by the clock algorithm, PLC-Cache will decrease the value of *delay_count* by 1, and then check its value. If *delay_count* is 0, the victim is passed to the third module (i.e. *In-Out Comparison*) for the possible cache replacement (See <condition 2> in Fig. 5). When the value of *delay_count* is larger than 0, the cache replacement is terminated here, and container *b* is ignored by PLC-Cache.

In fact, the mechanism of *Delayed Eviction* extends the staying time of cached containers in SSD. Thus some cached containers with high popularity but unstable access intervals (i.e. Quadrant B1) can be protected to avoid too early eviction, with the result of transferring some data in QB1 which frequently flushes SSD cache into the desired QA data. It can

enhance the endurance of SSDs and improve cache hit rate at the same time.

3) In-Out Comparison

If a cached container in SSD satisfies the condition of *Delayed Eviction* (e.g., container *a* in Fig. 5), PLC-Cache will compare containers *b* and *a* in the module of *In-Out Comparison*. The conditions (i.e. the <condition 3> in Fig. 5) of replacing container *a* with *b* is one of the following two:

a) The *reference_count* of container *b* is larger than that of container *a*, which indicates that *b* is related to more files and then may have more chances to be accessed than *a*.

b) The value of container b's *access_count* is larger than the multiple of Popularity Filter's threshold and a parameter calculated according to container a's popularity.

For instance, assuming *Popularity Filter*'s threshold is 5, and the parameter is 1.8 according to container *a*'s popularity, container *b* can replace container *a* only when the *access_count* of container *b* is no less than 9 (i.e. 1.8*5). Otherwise container *b* will be discarded, and container *a* is still cached in SSD. In this way, only the external data which is obviously better than the internal one is worthy of making the cache replacement and SSD writes. Thus the data in quadrant B2 is excluded from SSD cache, with the result of much less SSD writes and similar average popularity in SSD cache (because container *b* is not obviously better than container *a*).

C. Summary of Cache Replacment in PLC-Cache

The cache replacement of PLC-Cache is described in the following pseudo code. Note that PLC-Cache is designed to boost the read accesses of a deduplication-based primary storage. The boost of write requests (including file splitting, deduplicating, and storing) is beyond the focus of this paper.

```
0: def REQ PROCESS (Read Request x):
1:   case 1: x hits a SSD container c.
2:     Read data from SSD.
3:     c's access_count and inner_priority both plus 1.
4:   case 2: SSD cache miss, x hits a HDD container b.
5:     Read data from HDD, and b's access_count pluses 1.
6:     if b's access_count > threshold of the filter.
7:       case 2.1: SSD is not full.
8:         write b into SSD cache.
9:       case 2.2: SSD is full.
10:        container a = CLOCK_EVICT().
11:        a's delay_count decreases by 1.
12:        case 2.2.1: a's delay_count > 0.
13:          return.
14:        case 2.2.2: a's delay_count = 0.
15:          if IN_OUT_COMPARE(b, a)
16:            writing container b into SSD to replace a.
17: def CLOCK_EVICT():
18:    while (the pointed container's inner_priority > 0):
19:      the pointed container's inner_priority decreases by 1.
20:      clock pointer moves forward.
21:    return the pointed container.
22: def IN_OUT_COMPARE(b, a):
23:      return b's reference_count > a's reference_count or b's
access_count > f(a.access_count)*threshold of the fileter.
24: Note: f(x) is a mapping function, ranging from 1.0 to 2.0.
```

D. System Implement

We have implemented our proposed PLC-Cache in a practical open-source deduplication prototype system called Destor [23] developed by Huazhong University of Science & Technology, China. Destor works in a Linux 64bit environment, and has implemented a variety of latest algorithms of data deduplication with about 10,000 line codes in C.

Our proposed PLC-Cache works on the basis of the latest container capping and the rolling forward assembly area technique [8], in which the deduplicated data are managed in containers with fixed size (e.g. 2MB or 4MB), and each data segment with fixed size has limited number of containers to reduce the seeking counts to boost the process of file reading. SSD cache of PLC-Cache also manages cached data in the unit of containers. An additional benefit is that the containers' sizes are usually the integer multiples of Flash chips' erase unit (e.g. 256KB) inner SSDs. Therefore, the fragmentation of garbage data pages inner SSDs and the write amplification rate will be significantly reduced, with results of better write performance and longer lifetime of SSDs.

However, Destor is designed as a secondary deduplication system (i.e. a backup system), so we have made some modification to rebuild it to be a primary deduplication system. As we know, *File Recipe* is critical information for data reading. In the original program, the recipe of each file is stored sequentially in a certain file named *job_meta_data* during the backup phase. It will be read sequentially from the beginning of this file when we restore the file. In our modified program, we can get the number of bytes occupied by each file recipe, so the offset of each file recipe in the file *job_meta_data* can be obtained directly. In this way, we can randomly access to any files.

Based on the above modification, we have implemented our proposed PLC-Cache scheme in Destor. The cached containers are managed by a circular linked list according to clock algorithm, and the recorded metadata of each container in the list includes *access_count, reference_count, inner_priority*, and *delay_count* for the usage of the three modules in PLC-Cache (*Popularity Filter*, *Delayed Eviction*, and *In-Out Comparison*). The *access_count* and *reference_count* of all the containers in the system are managed by a linked list. In addition, we have implemented some classical cache algorithms such as LRU, LFU, FIFO in Destor for performance evaluation.

What's more, all the metadata of data deduplication (e.g., *Chunk Index*, and *File Recipe* shown in Fig. 2) is maintained in SSD to fully utilize the high performance of SSD. This helps a lot in reducing the read latency of a deduplication-based primary storage system with the cost of occupying only a very small part of SSD.

V. EVALUATIONS

A. Experimental Setup

Recall that we built our experimental environment by extending a real-world open-source deduplication system – Destor [23]. We compare our PLC-Cache with the following two systems:

1) A HDD-only deduplication system without SSD cache;

2) A SSD-cache-enabled deduplication system coupled with the FIFO, LRU and LFU caching algorithms.

The caching algorithms are designed to reduce read latency of deduplication-based primary storage systems; three metrics considered in this study are average read latency, cache hit rates, and the amount of written data in SSD. The first two reflect system performance, whereas the third one represents SSD lifetime.

The real-world data set used in the experiments was collected from Renmin University's laboratory servers; the size of the data set is around 200 GB. We split the collected data into variable-size chunks with mean size 10 KB using the content-based Rabin fingerprinting chunking algorithm [25], which is the most popular chunking algorithm that achieves higher deduplication efficiency than the fixed chunking algorithm. Fingerprints of chunks are generated using the SHA-1 hash function for each data chunk to identify duplicate chunks. A synthetic trace used in our experiments randomly accesses the collected data based on a Zipf distribution, and the amount of read data exceeds 520GB.

All the experiments are performed on a Ubuntu 12.04 LTS system, and the hardware configuration includes an Intel Core$^{(TM)}$ i7-2600 quad-core CPU running at 3.40GHz, coupled with 4GB RAM, one 2TB 5400rpm Western Digital hard disk, and a 60GB Intel 520 Flash-based SSD [22] with SATA 6Gb/s interface.

B. Overall Performance Comparison

In the first part of the evaluation, we compare the overall performance of PLC-Cache with the HDD-only non-cache system, and an SSD-cache-enabled system under the FIFO, LRU, and LFU policies. Figs. 6, 7, and 8 plot the average read latency, the amount of data written to SSD, and the hit rates of SSD cache, respectively.

All the three evaluated systems share the same settings: the size of containers for duplication is 2MB; the available capacity of SSD cache is 20% of the deduplicated data in disks; the total amount of accessed data is about 520GB.

Fig. 6 shows the average read latency of randomly accessing one file for a HDD-only non-cache system and an SSD-cache-enabled system coupled with FIFO, LRU, LFU, and PLC-Cache respectively. PLC-Cache improves the read performance by 41.9% with the support of additional SSD with only 20% capacity of the deduplicated data on HDDs. Moreover, PLC-Cache outperforms FIFO, LRU and LFU by 24.0%, 23.2%, and 23.0%, respectively. Interestingly, the three classical algorithms almost share the same performance.

A second significant advantage of PLC-Cache over the other caching algorithms lies in PLC-Cache's capability of extending SSD lifespan by reducing SSD writes. Fig. 7 reveals that the total amount of written data under LRU and LFU are almost identical (i.e., 48.77GB vs. 48.55GB). FIFO leads to a large amount of written data (i.e., 53.05GB). In contrast, PLC-Cache significantly reduces the total amount of data written in SSD down to 3.18GB, which is less than 6.7% of the data amount in the LFU case. Thus, PLC-Cache can extend the lifetime of SSD cache by a factor of 15 compared with the classical caching algorithms.

Fig. 8 shows that the cache hit rate of PLC-Cache is 66.9%, whereas those of FIFO, LRU, and LFU are respectively 61%, 61.9%, and 62%. The hit-rate improvement of PLC-Cache over the three existing algorithms is an average of 8.55%. In fact, the classical caching algorithms should exhibit good cache hit rates, because there exist many chances for updating cached data. More importantly, PLC-Cache not only improves the average quality of data cached in SSD through the *Popular Filter* and *In-Out Comparison* modules, but also increases the hit rate by the *Delayed Eviction* module.

Comparing PLC-Cache's improvements in terms of both cache hit rate and latency reduction, we observe that the performance improvement of PLC-Cache is more pronounced than its hit-rate improvement (i.e., 23.0%~24.0% vs. 8.55%). The performance improvement is more significant thanks to PLC-Cache's prominent reduction in SSD writes, which is much slower than SSD read operations. The existing algorithms generate a huge amount of data written to SSD, which may cause I/O congestions during the SSD writing process while slowing down the SSD read performance. On the contrary, PLC-Cache alleviates the I/O congestion problem by minimizing the number of SSD writes, thereby substantially improving SSD read speed.

In order to clearly observe the distribution changes of SSD written data set, we conduct an experiment to compare PLC-Cache and LRU – an existing algorithm widely adopted in many real-world storage systems. In this experiment, random reads of 520GB data are issued to the SSD-cache-enabled deduplication system. The SSD cache size is 10% of the total deduplicated data. Fig. 9 shows that LRU's total amount of written data is 235.9GB, whereas PLC-Cache reduces the amount of SSD written data down to 4.4GB. Thus, PLC-Cache reduces the written data amount of LRU by 98.1% (i.e., 1/53.6 of LRU's written data amount). Fig. 9 also indicates the distribution of the four quadrants. In LRU, non-popular data (i.e., QC and QD data) account for more than 73% of all the

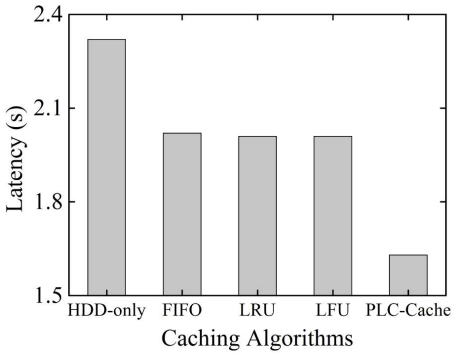


Fig. 6 PLC-Cache improves the performance by 23.4% on average compared with the classical algorithms.

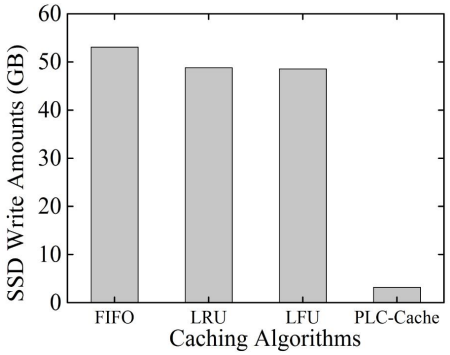


Fig. 7 PLC-Cache reduces the SSD write amounts for more than 15 times.

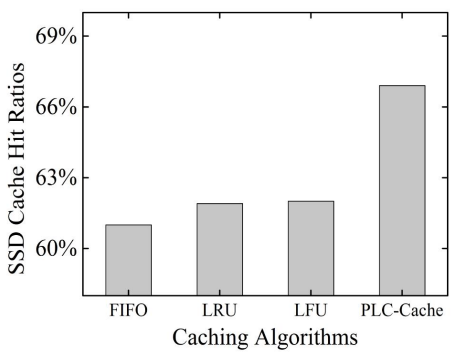
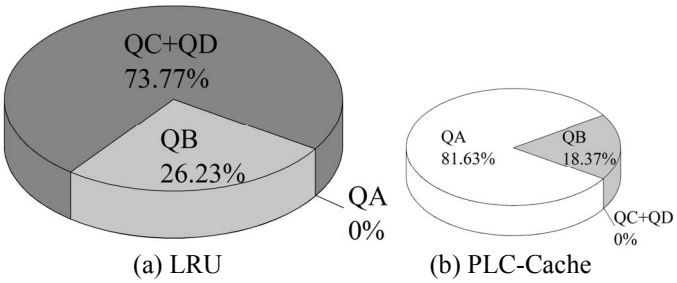


Fig. 8 PLC-Cache improves the cache hit rate by 8.55% on average.



(a) LRU (b) PLC-Cache

Fig. 9 PLC-Cache noticeably reduces the amount of SSD written data (represented by the size of pies) by excluding non-popular data (i.e., QC, QD data), and a part of short-term-popular (i.e., QB data) from the written data set, and significantly improves the proportion of QA data (i.e. high up to 81.63%) compared the one of LRU (0%).

written data, and PLC data (i.e., QA data) only accounts for 0%. In PLC-Cache, non-popular data is excluded from the SSD cache, and the proportion of PLC data is increased up to 81.63%. In addition, a part of QB data that has similar popularity with the cached data in SSD (i.e. data in QB2) has been eliminated, and a part of QB data been promoted to QA (i.e. data in QB1 in Fig. 4). As a result, the percent of QB data drops from 26.23% to 18.37%.

C. *Effects of Various System Settings*

Now we evaluate the impacts of various system settings, including the SSD-cache capacity, the access popularity distribution, and the size of containers in deduplication.

1) *Impacts of SSD Cache Capacity*

Fig. 10 exhibits the average latency of reading one file when we vary the capacity ratio (i.e., from 10% to 40%) between SSD cache and deduplicated HDD data ranging. Fig. 10 shows that when it comes to LRU and PLC-Cache, read latency almost linearly decreases with the increasing SSD capacity for both. Regardless of the SSD cache capacity, PLC-Cache is superior to LRU.

Fig. 11 plots the SSD write amounts for LRU and PLC-Cache. The experimental results visualized in Fig. 11 indicate that PLC-Cache rather than LRU has a steady write pressure under various capacity settings of the SSD cache. In other words, LRU writes more when SSD cache is relatively small; LRU reduces the number of writes in case of a large SSD cache. This trend can be explained by the fact that LRU achieves higher cache hit rates when deploying a large SSD cache, leading to decreased number of missed requests - the source of SSD writes.

Fig. 12 shows that the cache hit ratios almost linearly increase when the SSD capacity goes up. When the capacity ratio between SSD cache and the deduplicated HDD data is smaller than 25%, PLC-Cache can achieve a similar or even higher cache hit rate than LRU. When the capacity ratio is above 25%, LRU's hit rate becomes better than PLC-Cache. Unlike LRU, PLC-Cache significantly reduces the frequency of updating SSD cache, thereby resulting in a slightly lower cache hit rate than LRU. Nevertheless, thanks to PLC data cached by PLC-Cache in SSD, PLC-Cache maintains a similar or even better hit rate when cache size is relatively small.

Due to the high cost of SSDs, SSD-cache size is usually not very large. In all the subsequent experiments, we set the SSD-cache capacity to be 20% of deduplicated HDD data.

2) *Impacts of the Parameter α in Zipf*

Now we investigate the behavior of PLC-Cache and LRU under synthetic traces with various data popularity distribution. For a storage system with F files, a Zipf distribution is used to select the ID of an accessed target file. Zipf distribution is expressed in (1), where i is the file ID ranging from 1 to F, and α is a data popularity distribution parameter ranging from 0 to 1. A large α value results in concentrated I/O accesses.

$$p(i,\alpha,F) = \frac{1/i^{1-\alpha}}{\sum_{i=1}^{F} 1/i^{1-\alpha}} \quad (1)$$

In this group of experiments, we vary the α value in a range between 0.2 and 0.9 to represent different levels of access concentration. Recall that if the parameter α is large, there exist data with extremely high access counts, which usually leads to a high cache-hit rate and good performance.

Figs. 13, 14, and 15 respectively show the average latency, SSD write amounts, and the SSD cache hit rates of LRU and PLC-Cache under various α settings. Not surprisingly, a large α value leads to a high cache-hit rate, a low read latency, and a small number of SSD writes, which are caused by a reduced amount of missed data. Compared with LRU, PLC-Cache always achieves better performance and a much lower write pressure in SSD cache; the behavior of PLC-Cache under the traces with various popularity distributions is more stable than LRU. Note that the default parameter α of Zipf is set to 0.47, representing common cases in all the other experiments.

3) *Impacts of Container Size*

A container consisting of many small data chunks is a basic unit of cache replacement in SSD caches. The size of a container is usually a few megabytes; data chunk size in a deduplication system is only several kilobytes. Because the container size is usually a multiple of SSDs' basic erase unit, writing or overwriting data to SSDs in the unit of a container is good for reducing SSD write amplification. We conduct a experiment to measure a practical erase cycles of the SSD caches by reading the S.M.A.R.T. [26] information of SSDs before and after the experiment. The experimental results show that compared with a SAR-like scheme that writes in the unit of small data chunks, PLC-Cache, which writes SSD in the unit of containers, reduces write amplification of inner SSDs by 54.4%.

Figs. 16, 17, and 18 illustrate the latency, SSD write amounts, and the SSD hit rates of LRU and PLC-Cache under various container sizes ranging from 128KB to 16MB. For the large-container case (i.e., larger than 1MB), the performance drops along with the increase of container size, because reading a file needs extra time to scan larger containers to search for target data chunks. For all the container sizes, PLC-Cache outperforms LRU in terms of average latency. The SSD write amounts increase along with the increasing container size, because updating one container in the SSD cache leads to a larger amount of written data. PLC-Cache alleviates the high SSD write pressure of LRU; such a benefit offered by PLC-Cache becomes more pronounced under large containers.

LRU and PLC-Cache exhibit similar cache-hit rates. Their hit rates fluctuate anywhere between 60% to 65% when the container size is smaller than 4MB. When the container size is larger than 4MB, the cache hit rates quickly drops due to a large number of irrelative cold data chunks mixed in large

containers. The default container size in all the other experiments is 2MB.

D. Effects of Different Parameters in PLC-Cache

The threshold settings of *Popularity Filter* and *Delayed Eviction* - two important components of PLC-Cache algorithm, affect the amount of written data in SSD caches. Here we conduct a series of experiments to analyze the impacts of these thresholds of PLC-Cache.

1) Impacts of Popularity Filter threshold

The threshold of *Popularity Filter* is used to improve the average quality of cached data in SSD caches according to historical access counts. If the threshold is set too low, there will be an excessive amount of non-popular data (i.e., QC or QD data) polluting the SSD caches. On the contrary, when the threshold is set too high, many popular data will lost opportunities to be loaded into the SSD caches. Thus, setting an appropriate threshold value is indispensable for optimizing PLC-Cache.

Fig. 19 shows that when the *Popularity Filter* threshold ranges from 0 to 14, the average latency decreases until the threshold value reaches 4; after that, there is no noticeable changes in performance. Furthermore, the increasing *Popularity Filter* threshold indeed causes the decline in the amount of SSD written data; this trend is especially true when the threshold is less than 4 (see Fig. 20).

Fig. 21 reveals that when the threshold exceeds 4, SSD cache-hit rate rapidly declines due to a decreasing number of PLC data in SSD caches. The dropping cache-hit rate is a leading factor for the increased read latency. On the other hand, amount of written data into SSD declines, which in turn avoids I/O congestions while improving read performance. Therefore, the latency almost remains unchanged when the threshold is larger than 4. For example, when the threshold is 4, the hit rate is 62.3%. When the threshold is 14, the hit rate is 55.5%, representing a reduction of 12%; the amount of data written to the SSD cache is reduced by more than 66%. Note that the default threshold value is set to 4.

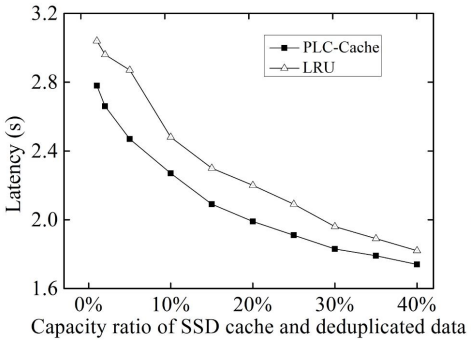


Fig. 10 Comparison of average latency under various size of SSD cache.

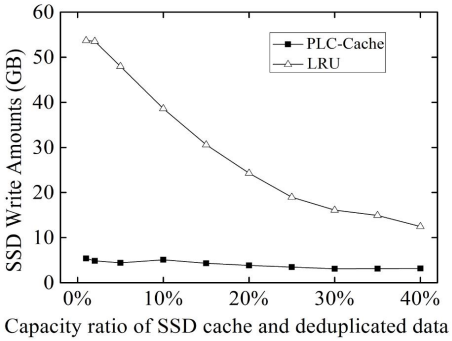


Fig. 11 Comparison of total amount of data written to SSD under various size of SSD cache.

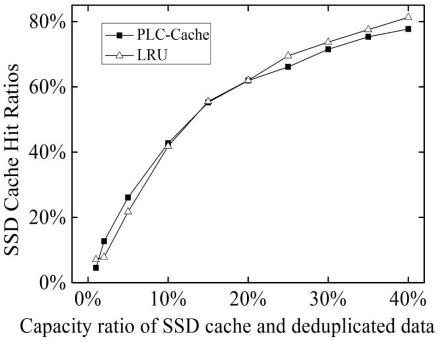


Fig. 12 Comparison of SSD cache hit rate under various size of SSD cache.

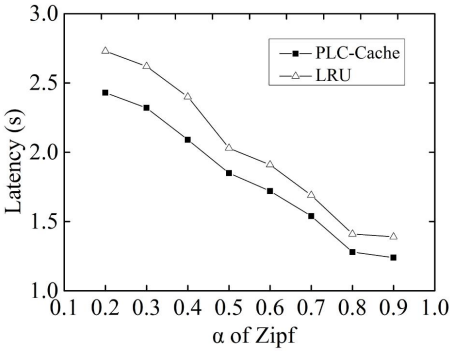


Fig. 13 Comparison of average latency under various value of Zipf distribution.

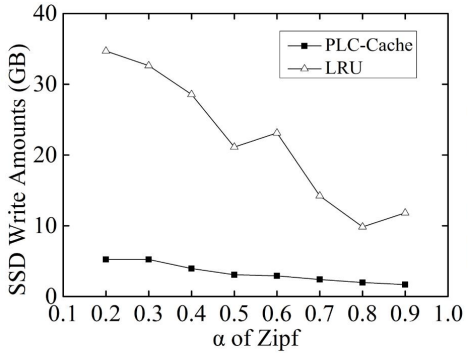


Fig. 14 Comparison of total amount of data written to SSD under various value of Zipf distribution

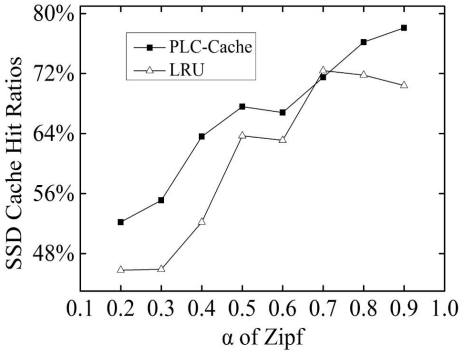


Fig. 15 Comparison of SSD cache hit ratios under various value of Zipf distribution.

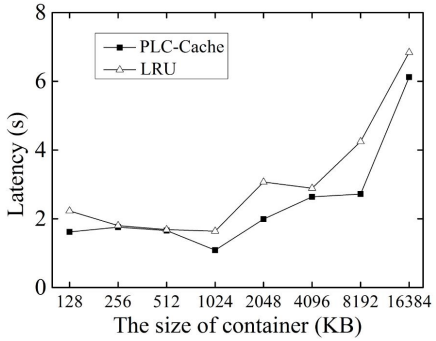


Fig. 16 Comparison of average latency under various container size.

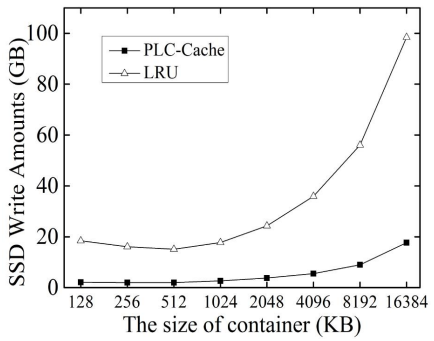


Fig. 17 Comparison of total amount of data written to SSD under various container size.

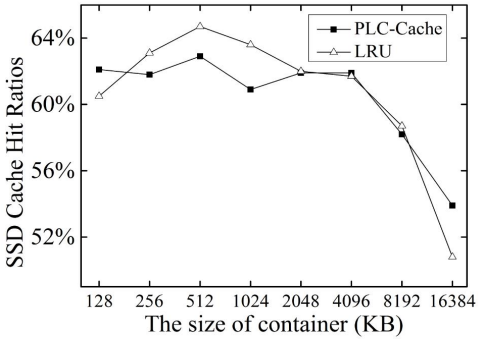


Fig. 18 Comparison of SSD cache hit ratios under various container size.

2) Impacts of Delayed Eviction threshold

Recall that the *Delay Eviction* module in PLC-Cache is used to convert the short-term-popular data (i.e., QB1 data) to PLC data (i.e., QA data) by delaying the evictions of data cached in SSDs. Thus, setting an appropriate threshold of *Delayed Eviction* can optimize the performance of PLC-Cache.

Figs. 22, 23 and 24 respectively plot the impacts of the *Delayed Eviction* threshold on average latency, total SSD write amounts, and SSD cache-hit rate. When the threshold increases from 0 to 14, the SSD write amounts are declining, because the number of data updates in the SSD cache is dropping.

Interestingly, there is no clear co-relationship between the threshold and the SSD-cache hit rate (see Fig. 24). On one side, a larger threshold allows PLC-Cache to promote more short-term-popular data to PLC data, which help in increasing hit rate. On the other side, a larger threshold leads to few SSD-cache updates to miss some hot data, which in turn make hit rate declining. Increasing the threshold might increase or decrease the cache hit rate..

The average latency drops when the threshold increases all the way up to 6; the latency becomes unstable when the threshold is larger than 6. Comparing Figs. 22 and 23, we observe that the decreases in SSD write amount contribute to the improved performance (i.e., smaller latency), because reducing SSD writes boosts SSD read bandwidth. When the decline of write amount becomes impossible after the threshold is larger than 6, the latency stops dropping. Note that the default threshold of *Delayed Eviction* is set to 2 in this group of experiments.

VI. CONCLUSIONS

The limited write performance and the poor write endurance of flash-based SSDs bring serious performance and lifetime challenges for SSD caches in deduplication-based primary storage systems. In this study, we proposed a new SSD-based caching algorithm called PLC-Cache to improve storage system performance while extending the lifespan of SSD caches. PLC-Cache seamlessly integrates three modules, namely, *Popularity Filter*, *Delayed Eviction*, and *In-Out Comparison* to achieve two overarching goals. First, PLC-Cache prevents non-popular data from being fetched to SSD caches. Second, PLC-Cache promotes some of the popular and short-term cached data into long-term cached one to contribute cache hits in a long time period with much less cost of SSD writes. By achieving these two goals, PLC-Cache improves the amount of long-term-popular data residing in SSD caches to improve performance and to reduce the number of writes in SSD caches at the same time.

We implemented PLC-Cache in a real-world deduplication system. Our experimental results confirm that PLC-Cache outperforms the three classical caching algorithms (e.g., FIFO, LRU, and LFU) in terms of read latency by an average of 23.4%. Compared with the three algorithms, PLC-Cache reduces the total amount of data written to SSD caches by about a factor of 15.7, indicating that PLC-Cache extends the lifetime of SSD caches by at least a factor of 15.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (No. 61202115), and open research program of State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Science (No. CARCH201302). Xiao Qin's work is supported by the U.S. National Science Foundation under Grants CCF-0845257 (CAREER), CNS-0917137 (CSR), CNS-0757778

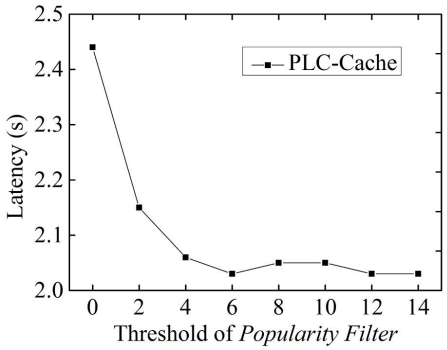


Fig. 19 Comparison of latency under various thresholds of *Popularity Filter*.

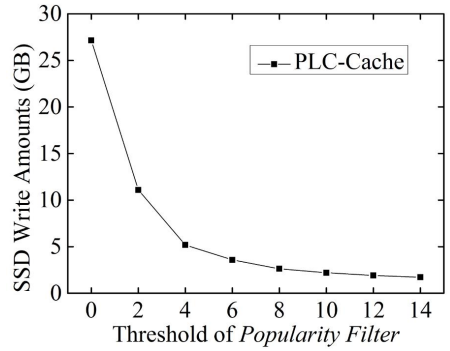


Fig. 20 Comparison of total SSD write amounts under various threshold of *Popularity Filter*.

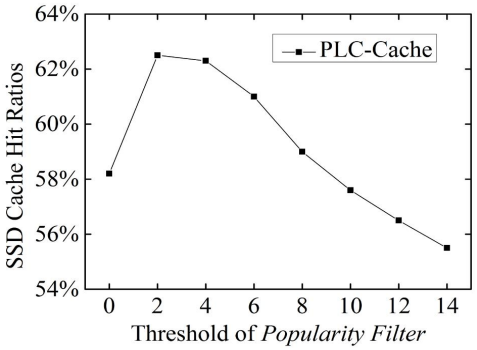


Fig. 21 Comparison of SSD cache hit ratios under various threshold of *Popularity Filter*.

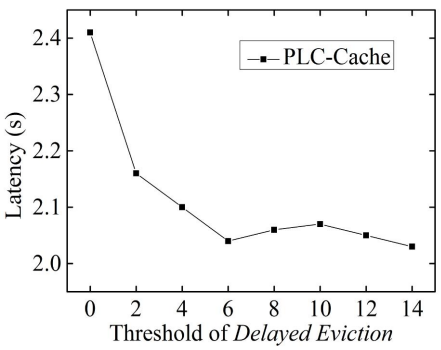


Fig. 22 Comparison of latency under various threshold of *Delayed Eviction*.

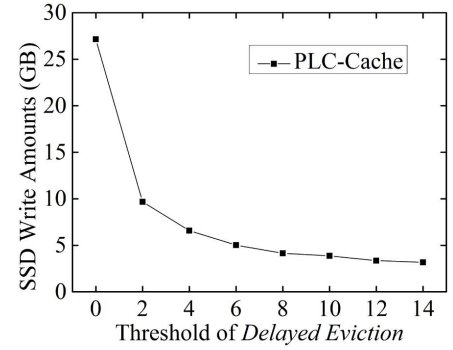


Fig. 23 Comparison of total SSD write amounts under various threshold of *Delayed Eviction*.

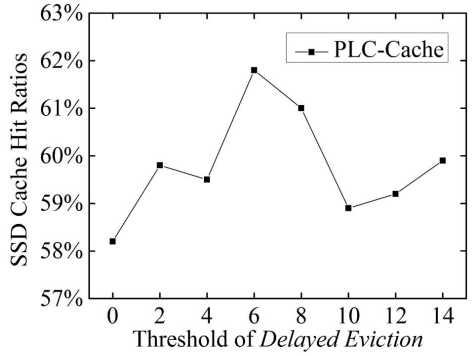


Fig. 24 Comparison of SSD cache hit ratios under various threshold of *Delayed Eviction*.

(CSR), CCF-0742187 (CPA), CNS-0831502 (CyberTrust), CNS-0855251 (CRI), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), and DUE-0830831 (SFS).

In addition, we thank Min Fu from Huazhong University of Science & Technology for giving us sincere help and guidance for the modification of Destor.

REFERENCES

[1] Ganesan, Pradeep, "Read performance enhancement in data deduplication for secondary storage", Master Thesis. University of Minnesota, 2013.

[2] J. Gantz, D. Reinsel. "The Digital Universe Decade-Are You Ready? ", IDC White Paper, 2010.

[3] D. Iacono, "Enterprise Storage: Efficient,Virtualized and Flash Optimized", IDC WhitePaper, May 2013.

[4] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti, "iDedup: Latency-aware, inline data deduplication for primary storage", *Proceedings of the 10th USENIX conference on File and Storage Technologies (FAST'12)*, Feb. 2012.

[5] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, andP. Camble, "Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality", *Proceedings of the 7th USENIX conference on File and Storage Technologies (FAST'09)*, Feb. 2009.

[6] B. Debnath, S. Sudipta, and J. Li, "ChunkStash: Speeding up Inline Storage Deduplication using Flash Memory", *Proceedings of the 2010 USENIX annual technical conference (ATC'10)*, 2010.

[7] B. Mao, H. Jiang, S. Wu, Y. Fu, and L. Tian, "SAR: SSD Assisted Restore Optimization for Deduplication-Based Storage Systems in the Cloud", *IEEE 7th International Conference on Networking, Architecture and Storage (NAS'12)*, 2012.

[8] M. Lillibridge, E. Kave, and B. Deepavali, "Improving Restore Speed for Backup Systems that Use Inline Chunk-based Deduplication", *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST'13)*, Feb.2013.

[9] N. Megiddo and D. Modha, "ARC: a Self-tuning, Low Over-head Replacement Cache", *Proceedings of the 2nd USENIX Symposium on File and Storage Technologies (FAST'03)*, Mar. 2003.

[10] Kim, J., Son, I., Choi, J., Yoon, S., Kang, S., Won, Y., & Cha, J. "Deduplication in SSD for Reducing Write Amplification Factor", *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST'11)*, 2011.

[11] A. Muthitachatoen, B. Chen, and D. Mazi eyes. "A lowbandwidth network file system. " *In Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '0l)*, pp. 174-187, Oct. 2001.

[12] Intel Corporation, "Intel Solid-StateDrive 910 Series: Product Specification",http://www.intel.com/content/www/us/en/solid-state-drives/ssd-910-series-specification.html, Jun. 2012.

[13] Zhu, Benjamin, Kai Li, and R. Hugo Patterson. "Avoiding the Disk Bottleneck in the Data Domain Deduplication File System." *Proceedings of the 7th USENIX Symposium on File and Storage Technologies (FAST'08)* , Feb.2008.

[14] W. Xia, H. Jiang, D. Feng, and Y. Hua. SiLo: "A Similarity-Localitybased Near-Exact Deduplication Scheme with Low RAM Overhead and High Throughput. " *Proceedings of the 2011 USENIX Annual TechnicalConference (USENIX ATC'11)*, Jun. 2011.

[15] S. Jiang and X. Zhang, "LIRS: An Efficient Low Inter-reference Recency Set Replacement Policy to Improve Buffer Cache Performance", *Proceeding of 2002 ACM SIGMETRICS*, pp. 31-42, Jun. 2002.

[16] Soundararajan, Gokul, et al. "Extending SSD Lifetimes with Disk-Based Write Caches." *Proceedings of the 9th USENIX Symposium on File and Storage Technologies (FAST'10)*, Feb. 2010.

[17] Chen, Zhiguang, Fang Liu, and Yimo Du. "Reorder the Write Sequence by Virtual Write Buffer to Extend SSD's Lifespan." *Network and Parallel Computing*. pp. 263-276,Springer Berlin Heidelberg, 2011.

[18] D. Meister and A. Brinkmann. "dedupv1: Improving DeduplicationThroughput Using Solid State Drives (SSD). " *Proceedings of the 29th International Conference on Massive Storagesystems and Technology* (MSST'10), May 2010.

[19] S. Huang, Q. Wei, J. Chen, C. Chen, and D. Feng, "Improv-ing Flash-based Disk Cache with Lazy Adaptive Replacement",*Proceedings of the 29th International Conference on Massive Storagesystems and Technology* (MSST'13), May 2013.

[20] EMC, "EMC FAST Cache: A Detailed Review",http://www.emc.com/collateral/software/white-papers/h8046-clariion-celerra-unified-fast-cache-wp.pdf, Oct. 2011.

[21] J. Matthews, S. Trika, D. Hensgen, Rick Coulson, and KnutGrimsrud, "Intel Turbo Memory: Nonvolatile Disk Caches inthe Storage Hierarchy of Mainstream Computer Systems", *ACM Transactions on Storage*, vol. 4, no. 2, May 2008.

[22] Intel Corporation, "Intel Solid-State Drive 520 Series: Product Specification", http://www.intel.com/content/www/us/en/solid-state-drives/ssd-520-specification.html.

[23] Min Fu, Huazhong University of Science & Technology, China. Destor: https://github.com/fomy/destor

[24] Oracle, "Exadata Smart Flash Cache Features and the Oracle Exadata Database Machine", http://www.oracle.com/technetwork/serverstorage/engineered-systems/exadata/exadata-smart-flashcache-366203.pdf, Jan. 2013.

[25] Michael O. Rabin. Fingerprinting by random polynomials.Technical Report TR-15-81, Center for Research in Comput-ing Technology, Harvard University, 1981.

[26] S.M.A.R.T.,http://en.wikipedia.org/wiki/Self-Monitoring,_Analysis_and_Reporting_Technology

[27] S. Boboila, and P. Desnoyers, "Write Endurance in Flash Drives: Measurements and Analysis", *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*, 2010.

[28] L. Grupp, J. Davis, and S. Swanson, "The Bleak Future of NAND Flash Memory", *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*, 2012.

[29] EMC, "New Digital Universe Study Reveals Big Data Gap" , http://www.emc.com/about/news/press/2012/20121211-01.htm

[30] Q. Yang, J. Ren, "I-CASH: Intelligently coupled array of ssd and hdd", *IEEE 17th International Symposium on High Performance Computer Architecture (HPCA'11)*, pp. 278-289, Feb. 2011.

[31] Young Jin Nam, Dongchul Park, and David HC Du. "Assuring demanded read performance of data deduplication storage with backup datasets." *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2012 IEEE 20th International Symposium on. IEEE, 2012.

[32] Open-source lessfs deduplication project. http://www.lessfs.com/wordpress/.

[33] Netapp, "Optimizing Storage Performance and Cost with Intelligent Caching", http://www.logismarket.pt/ip/elred-netapp-virtual-storage-tier-optimizing-storage-performance-and-cost-with-intelligent-caching-929870.pdf

[34] Open-source SDFS project. http://www.opendedup.org/.

[35] C. Alvarez. "NetApp deduplication for FAS and V-Series deployment and implementation guide". Technical Report TR-3505, NetApp, 2011.

[36] EMC. "Achieving storage efficiency through EMC Celerra data deduplication. " White paper, Mar. 2010.

[37] J.Bonwick.Zfsdeduplication.http://blogs.oracle.com/bonwick/entry/zfs_dedup, Nov. 2009.

[38] T. Pritchett, M. Thottethodi, "SieveStore: a highly-selective, ensemble-level disk cache for cost-performance", *Proceedings of* the 37th annual international symposium on Computer Architecture *(ISCA'10)*, pp. 163-174, 2010.

[39] Lei Wang, Jianfeng Zhan, Chunjie Luo, et al. "BigDataBench: a Big Data Benchmark Suite from Internet Services", *The 20th IEEE International Symposium On High Performance Computer Architecture (HPCA'14)*, Feb. 2014.