Academic Journal of Science and Technology
ISSN: 2771-3032 | Vol. 13, No. 3, 2024

Data Correlation Aware Buffer Cache Management for Improved I/O Performance on Log-Structured File System-based NAND Flash Storage Devices

Tian Tian

Luoyang Municipal Science and Technology Bureau, Luoyang 471000, China

Abstract: Buffer cache was critical in bridging the access speed gap between main memory and disks. Conventional buffer cache management approaches focus on improving the cache hit ratio by exploiting the locality of I/O workloads. However, conventional buffer caching strategies do not take the write characteristics of log-structured file systems into consideration, and thus they would suffer from a poor cache hit ratio as well as the degraded I/O performance. This study proposes a novel buffer cache management strategy at SSD controllers to improve the data caching performance. It takes advantages of the I/O pattern of log-structured file systems, e.g., F2FS, by correlating the current block I/O address with its previous I/O address such that the I/O requests issued to the same file location can be efficiently managed by the buffer cache. Experimental results demonstrate that the proposed approach substantially improve the read I/O performance by 14.5%, write I/O performance by 92.4%, and enhance the flash lifetime by 92.0%.

Keywords: Buffer cache; Data correlation; Log-structured File System; I/O Performance; Flash Storage.

1. Introduction

Flash storage devices are increasingly used as external storage in various computing environments, including mobile devices, personal computers, and enterprise servers. Compared to traditional hard disk drives (HDDs), flash devices offer a range of advantages, such as excellent read/write performance, shock resistance, and low power consumption. To fully leverage these benefits, flash-friendly log-structured file systems (LFS) have been proposed and widely adopted[1]. LFS uses out-of-place updating to transform random I/O into sequential I/O, thereby achieving superior I/O performance. For example, the F2FS file system, one of the most popular LFS file systems, can deliver a 2.5x performance gain over the traditional EXT4 file system on SATA SSD devices[2].

Flash-based solid-state drives (SSDs) typically integrate on-chip cache devices to balance the performance gap between memory and flash storage. These devices mainly maintain data caches and mapping caches. Data caches are the primary tool for balancing memory and flash performance, used to cache hot I/O data (data that is likely to be accessed soon) [3]. When new I/O accesses the flash device, the storage controller first checks the data cache for the target data. If found, the controller can immediately return the target data to the host, avoiding the lengthy flash access overhead and significantly reducing I/O access time.

Existing data cache management schemes enhance cache hit rates by exploiting the locality of I/O workloads [4, 5, 6]. For example, Park [6] proposed using access frequency and the time gap between accesses of each data block to prevent cache pollution by infrequently accessed data. However, due to the out-of-place updating mechanism used by LFS to produce sequential writes, the locality of I/O accesses is severely disrupted, and existing schemes cannot optimize performance in an LFS environment. Worse still, out-of-place updates lead to a large amount of invalid data occupying valuable cache resources, causing a sharp drop in cache hit rates. Experiments in this paper show that this drastic reduction in cache hit rates due to out-of-place updates in LFS not only severely damages the system's I/O performance but also significantly shortens the lifespan of flash devices.

To address these issues, this paper proposes a data association-aware cache management strategy for SSDs, which enhances cache hit rates by establishing address associations between sequential I/O updates. Specifically, the cache management process is implemented through the following steps:

(1) When a file update occurs, the logical block address (LBA) of the new write I/O data and its corresponding invalidated data blocks in LFS are associated. From the perspective of the file, the associated data blocks belong to the same file block. Therefore, the LBAs of the related blocks are packaged and transmitted along with the write I/O to the storage device, where they are used by the device's cache.

(2) By recognizing the new and old LBAs of the associated data blocks, the data cache can recover the I/O locality characteristics disrupted by LFS, thereby improving cache hit rates. This approach is highly feasible because, although out-of-place updating in LFS changes the LBAs of write I/Os and disrupts I/O locality, the associated data blocks still belong to the same file block. Thus, the flash controller can identify file write I/Os within the same block based on the associated data information and achieve cache hits by remapping the old and new I/O addresses.

Experimental results show that compared to existing cache management strategies, the proposed approach can effectively restore the I/O characteristics disrupted by LFS out-of-place updates, resulting in a significant improvement in cache hit rates. Specifically, the average read performance increases by 14.5%, write I/O performance improves by 92.4%, and the flash lifespan extends by 92.0%.

In summary, the proposed strategy for LFS-based flash SSDs aims to restore I/O locality characteristics through address remapping of associated file data I/O within the flash controller. According to the research, this is the first cross-

layer optimization addressing I/O locality degradation caused by LFS in flash device cache management. It is completely orthogonal to existing cache management algorithms and can seamlessly integrate into current flash device cache management systems as an extension module. The main contributions of this paper are as follows:

Identifying and validating that current LFS disrupts I/O locality, leading to a decline in flash device cache hit rates, reduced system I/O performance, and a shorter flash lifespan.

Proposing a data association-aware cache management strategy that restores I/O locality by remapping new and old addresses of associated I/Os, thus improving cache hit rates.

Verifying through experiments that the proposed cache management strategy effectively improves cache hit rates, enhances overall flash device performance, and significantly extends the flash lifespan.

The structure of the paper is as follows: Chapter 2 introduces the relevant background and reviews the latest related work. Chapter 3 quantifies the system performance loss and flash lifespan reduction caused by the disruption of I/O locality in LFS. Chapter 4 presents the proposed cache management strategy in detail. Chapter 5 compares and analyzes the experimental results of the proposed strategy with related work. Finally, Chapter 6 concludes the paper.

2. Background and Related Work

2.1. Log-Structured File Systems

Log-structured file systems (LFS) are increasingly being adopted in flash-based storage systems, including enterprise servers, personal computers, and mobile devices, due to their flash-friendly characteristics. For example, F2FS [2] has been adopted as the default file system in several recently released smartphones [7]. LFS handles file update I/O using out-of-place updates, where a new sequential logical address is allocated for each write I/O, while the old logical address for the updated file I/O is invalidated. This method transforms random I/O into sequential I/O, thus significantly improving system performance.

However, the out-of-place update method used by LFS disrupts the locality characteristics of I/O workloads. I/O locality includes temporal locality and spatial locality. Temporal locality refers to the likelihood that recently accessed I/O addresses (LBAs) will be accessed again in the near future. Spatial locality refers to the likelihood that a process that accesses a continuous range of LBAs will continue to access adjacent addresses in the same range. Since current flash device internal cache management primarily relies on the LBA of I/O requests to exploit locality characteristics and improve cache hit rates, the disruption of I/O locality by LFS significantly reduces cache hit rates, resulting in a decline in system performance and a shortening of flash device lifespan. The situation worsens when the old data of an updated file I/O is already stored in the cache, and the new I/O data is assigned a new LBA by the file system. In this case, the cache is unable to recognize the old invalid data, and as a result, invalid data can occupy valuable cache resources, further reducing cache hit rates. Although the TRIM mechanism can alleviate this problem to some extent, its significant overhead forces developers to carefully balance the tradeoff between the costs and benefits of TRIM. Therefore, addressing the reduction in cache hit rates caused by LFS out-of-place updates is critical to improving storage performance.

2.2. Flash Storage Devices

2.2.1. Sub heading

The section headings are in boldface capital and lowercase letters. Second level headings are typed as part of the succeeding paragraph (like the subsection heading of this paragraph). All manuscripts must be in English, also the table and figure texts, otherwise $_{s}w$ we cannot publish your paper. Please keep a second copy of your manuscript in your office.

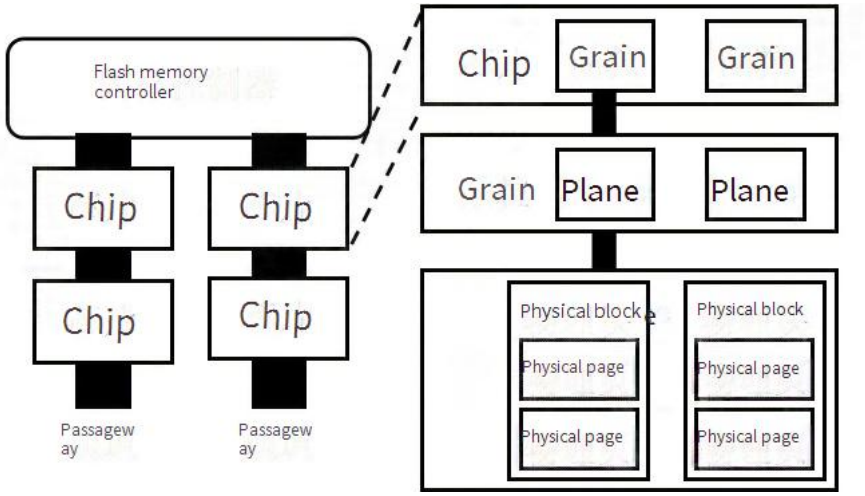


Figure 1. Flash Memory Device Structure Diagram

Flash storage devices are rapidly dominating the storage market. Compared to traditional hard disk storage, flash devices eliminate the time-consuming seek operations, enabling faster response times. Figure 1 shows the internal structure of a flash storage device. Each flash controller interacts with flash chips through multiple channels. Each chip encapsulates several flash dies, and each die consists of multiple storage planes. Each storage plane contains a large number of physical blocks, and each block is further divided into multiple physical pages. To manage the internal workings of the flash device, the flash controller maintains the Flash Translation Layer (FTL) algorithm, which serves two primary functions: address mapping and garbage collection (GC). The unit of flash read/write operations is a physical page, while

the unit for garbage collection is a physical block.

When a flash page has been written (either by Write or Program), any update data for the I/O data associated with that page cannot be written in-place. Therefore, the FTL algorithm is responsible for redirecting updated file data I/O to a new free physical page while invalidating the old physical page containing the original data. When the number of available physical pages drops below a specified threshold, the controller triggers the GC process to reclaim invalid space, thus creating more free pages. During the GC process, the controller first selects a target physical block according to a specified algorithm, copies all valid data from that block to new physical pages, and then erases the original block. Since flash devices have a limited number of write/erase cycles (Programming/Erase Cycle, P/E Cycle), the migration of valid data during GC reduces the flash device's lifespan.

TRIM is one of the most important advanced I/O commands for improving flash device data management efficiency [8]. This command notifies the flash device of invalid data from the host side, thereby reducing the device's data management overhead. Upon receiving the TRIM command, the device controller can mark the corresponding flash pages as invalid, stop maintaining that data within the device, and release it from the cache. Although the TRIM command helps improve device performance, it incurs significant system processing overhead. To minimize the impact of TRIM commands on other I/O requests, modern flash devices typically schedule TRIM operations during system idle times.

2.3. Cache Management

Random Access Memory (RAM), such as Static RAM (SRAM) and Dynamic RAM (DRAM), has much faster access speeds than hard disk storage and flash devices. To balance the performance gap between memory and external storage, flash devices, such as SSDs, often integrate an on-chip RAM component to cache hot data. Modern cache components are increasingly using non-volatile memory, such as RAM with backup batteries, Spin-Transfer Torque RAM (STT-RAM), or Phase Change Memory (PCM). When a new write I/O request arrives at the flash device, the device controller first stores the write I/O data in the cache and returns the I/O completion signal. The I/O data (or "dirty data," since it has not yet been written to the flash chip) is eventually written back to the flash chip at an appropriate time, such as when the device is idle, using the Flush command. Due to the limited size of the cache compared to the overall size of the flash device, only a small portion of the I/O data can be stored in the on-chip cache. Therefore, efficient cache management is crucial for optimizing I/O performance.

2.4. Related Work

Li et al. [9] proposed allowing controllable bit errors for fault-tolerant applications to reduce cache misses caused by erroneous cache pages. Park [6] suggested using the access frequency and reuse distance of each data block to assess the "hotness" of data and evict cold data from the cache accordingly. Liu et al. [10] addressed the issue of process interference in multi-process mixed-load environments by isolating processes and allocating cache resources to meet the performance requirements of each process. Zhou et al. [11] proposed optimizations to improve the accuracy of flow I/O identification in multi-stream SSDs and reduce interference between I/Os of different streams in the cache. Gao [12] proposed a read-write separation mapping cache optimization scheme based on the read-write I/O characteristics of the F2FS file system. Gao et al. [13] addressed data loss issues caused by integrated capacitor aging within SSD caches, proposing a scheme to limit the number of dirty pages in the cache based on the remaining charge of the integrated capacitor. Liu [14] developed a deep learning model to optimize cache resource allocation efficiency in NVMe multi-queue I/O environments by analyzing cache allocation reuse distances. Sun et al. [15] proposed separating hot and cold I/O workloads in the cache to perform dirty page flushing and cache evictions. Yoo et al. [16] addressed the issue of adapting cache parameters to I/O workload characteristics and device operating conditions in QLC SSDs by proposing a reinforcement learning model for real-time parameter adjustment. Additionally, several works [17-20] have proposed optimization solutions for cache data compression and deduplication. Although recent research has optimized SSD cache management from various angles, no related work has addressed the optimization of cache hit rates in LFS environments, where the locality of I/O is disrupted.

3. Problem Description

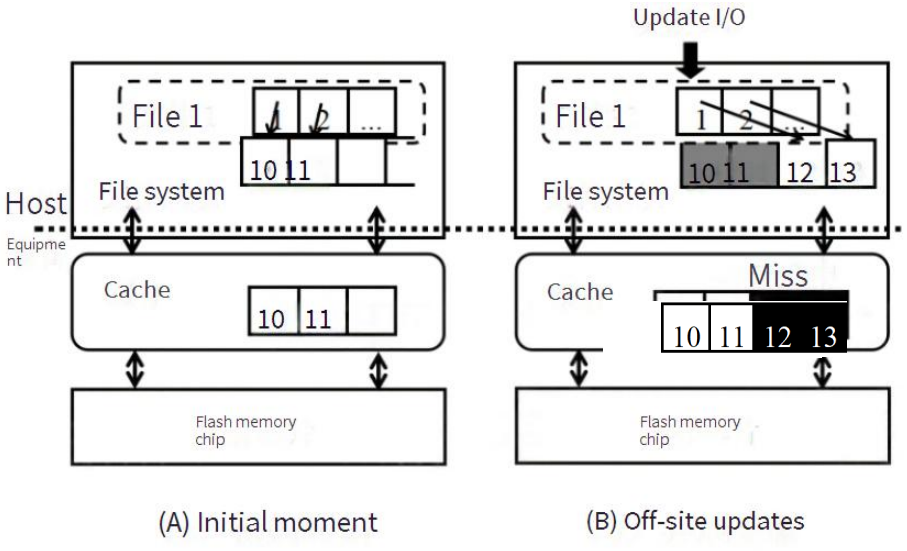
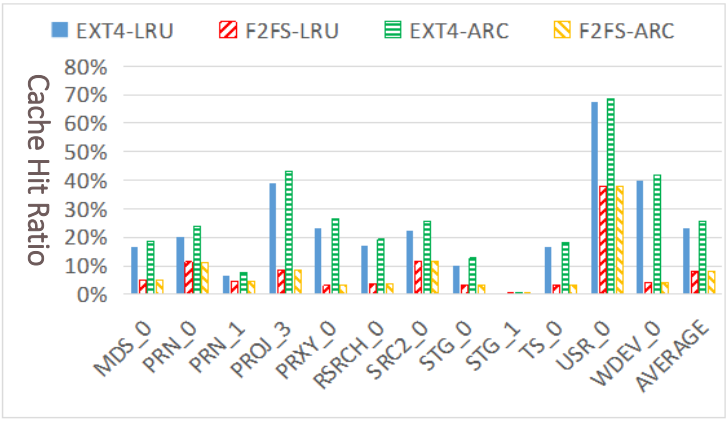


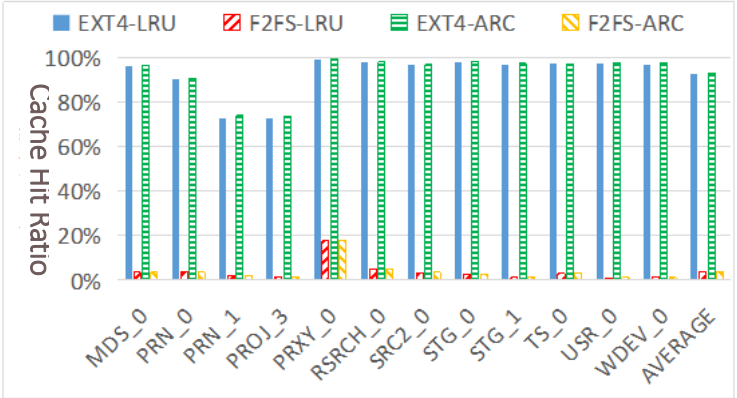
Figure 2. I/O System State Diagram

When a new I/O request is received, a flash storage device can only identify I/O data based on the requested block address. Previous work has aimed to improve the device's internal cache hit rate by exploiting temporal and spatial locality in I/O access addresses. From the perspective of the device's internal cache, I/O data is distinguished by the block address of the I/O request. However, for log-structured file systems (LFS), the locality of I/O requests becomes extremely low. Figure 2 shows a block diagram of the I/O system when using LFS with out-of-place updates. Although each file is assigned a new address by the file system, the old invalidated address data still resides in the cache without further processing. This phenomenon renders efforts to enhance cache hit rates using I/O locality ineffective. As shown in Figure 2(a), suppose the first two data blocks of File 1 are initially assigned logical addresses 10 and 11 by the file system. At the next time step, the file system receives an update for these two data blocks and assigns them new logical addresses 12 and 13, as shown in Figure 2(b). At this point, even though the previous versions of the two data blocks still exist in the cache, the current cache management mechanism, which relies solely on the logical address of the I/O request to identify the data, results in both updated I/Os missing the cache. Worse still, the old invalidated data still occupies limited cache resources, leading to the need for more dirty page writes to free up space for new I/Os. This further affects system performance and shortens the lifespan of the flash storage device.

To validate the above analysis, we conducted a series of experiments using the I/O workload published by MSR Cambridge [21] under different file system configurations to test flash device cache performance. Detailed experimental configurations are provided in Chapter 5. Figure 3 illustrates the cache hit rate for read and write I/Os, where EXT4- represents I/O generated by the EXT4 file system, F2FS- represents I/O generated by the F2FS file system, and LRU and ARC represent the cache replacement strategies using LRU and ARC, respectively. It can be observed that F2FS with out-of-place updates completely disrupts the locality of write I/Os. Under EXT4, the write I/O hit rates for LRU and ARC strategies are 93.0% and 93.3%, respectively. However, in F2FS, the write I/O hit rates drop to 3.8% and 3.9%, respectively. This is because out-of-place updates change the write I/O LBAs, causing the cache management mechanism to fail to recognize whether the updated I/Os hit the cache. It should be noted that F2FS still uses in-place updates for files below a certain size threshold, which allows for a small number of write I/O cache hits. Furthermore, out-of-place updates also have a significant impact on the cache hit rate for read I/Os. Under the LRU strategy, the average hit rate for I/Os drops from 23.3% to 8.1%, and under the ARC strategy, the average hit rate drops from 25.6% to 8.1%. This is because a large amount of outdated invalid data from updated file I/Os occupies cache resources, which causes a reduction in cache hit rates. The experimental results also show that for ARC, which relies on both I/O access frequency and temporal locality, the hit rate is better than LRU, which only depends on temporal locality in EXT4. However, in F2FS, the performance degrades to be almost identical to LRU, because out-of-place updates disrupt the I/O access frequency characteristics, and ARC can only rely on temporal locality for cache replacement management.

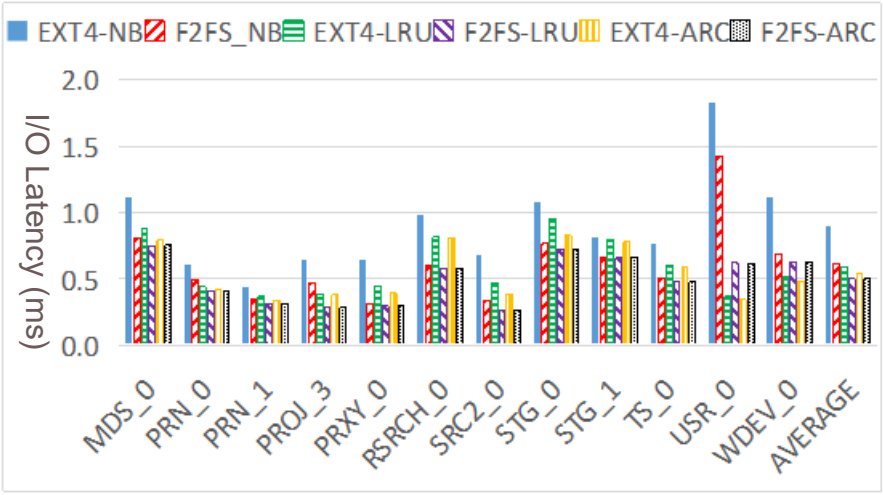


(a) Read I/O Cache Hit Ratio

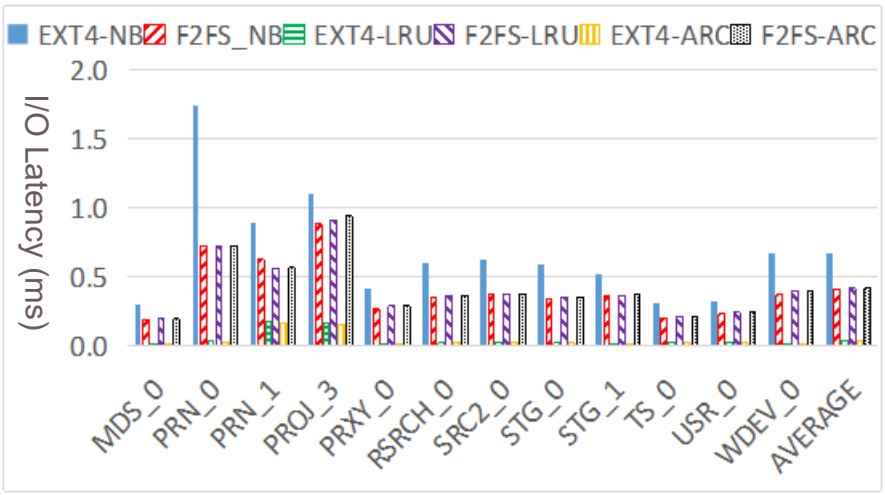


(b) Write I/O Cache Hit Ratio

Figure 3. Cache Hit Rate Comparison



(a) Read I/O Latency



(b) Write I/O Latency

Figure 4. I/O Average Latency Comparison

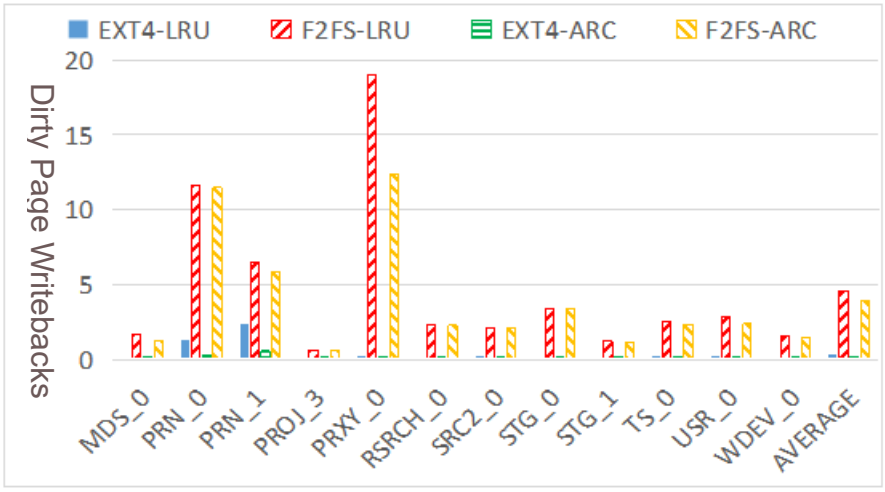


Figure 5. Dirty Page Writebacks Comparison

The decline in I/O cache hit rates directly manifests in increased I/O latency. Figure 4 compares the average I/O latency under different file systems and cache replacement strategies. NB represents a storage device without an integrated cache. It can be observed that without cache integration, F2FS shows better I/O performance than EXT4, owing to its flash-friendliness. However, once a cache device is integrated, F2FS with out-of-place updates fails to fully leverage cache resources due to the reduced cache hit rate. For write I/Os, although all write I/Os are written to the cache and an I/O completion interrupt is returned, when the cache becomes full, non-hitting I/Os need to write back dirty pages

to free up space. Therefore, the cache hit rate directly influences write I/O latency. As shown in Figure 4(b), when using LRU and ARC strategies, F2FS's write I/O latency increases by 91.7% and 92.2%, respectively, compared to EXT4. The primary reason for the large performance difference is that in F2FS, all write I/Os miss the cache, resulting in a large number of dirty page writes. Meanwhile, the reduced cache hit rate also has a direct impact on read I/O latency. Even though F2FS shows better read I/O performance than EXT4 when the cache is integrated, due to the cache hit rate decline, F2FS's performance advantage for read I/Os is significantly weakened. As shown in Figure 4(a), without cache integration, F2FS reduces read I/O latency by 30.5% compared to EXT4. However, when integrated with a cache and using the LRU and ARC management algorithms, F2FS only reduces read I/O latency by 14.8% and 8.1%, respectively. In some I/O workloads, such as USR_0 using LRU, F2FS even exhibits worse read I/O performance than EXT4, with a 41.1% increase. The two main reasons for the significant reduction in read I/O benefits with integrated cache in F2FS are: (1) the decline in the read I/O cache hit rate, and (2) a large number of dirty page writes causing write I/Os to stall, further exacerbating the interference of write I/Os on read I/Os.

Another consequence of the reduced cache hit rate is the shortening of the flash device's lifespan. Since write I/Os that miss the cache result in a large number of dirty page writes, and flash devices only support a limited number of P/E cycles, the total amount of writes directly affects the device's lifespan. Figure 5 compares the number of dirty page writes under different file systems and cache replacement strategies. It can be seen that out-of-place updates in F2FS cause a significant increase in the number of dirty page writes. Compared to EXT4, F2FS with LRU and ARC strategies leads to a 92.6% and 98.0% increase in the average number of dirty page writes, respectively. These dirty page writes severely deplete the P/E cycles of the flash device, leading to a rapid decline in its lifespan.

In conclusion, the reduction in cache hit rate caused by out-of-place updates has a significant impact on both system performance and flash device lifespan, making it an urgent issue that needs to be addressed.

4. Data Association-Aware Cache Management

This chapter presents in detail the cross-layer optimization strategy of data association-aware cache management for flash storage devices.

4.1. Overall Architecture

Due to the isolation mechanism between the flash device and the host, the internal cache management efficiency of the flash device is low and severely affects I/O performance. Based on this observation, this paper proposes a correlation-aware cache management strategy (CAC) to improve cache management efficiency for flash devices based on log-structured file systems. The proposed cache management strategy consists of two components: the address association tracker on the host side and the cache management strategy on the device side, as shown in Figure 4.

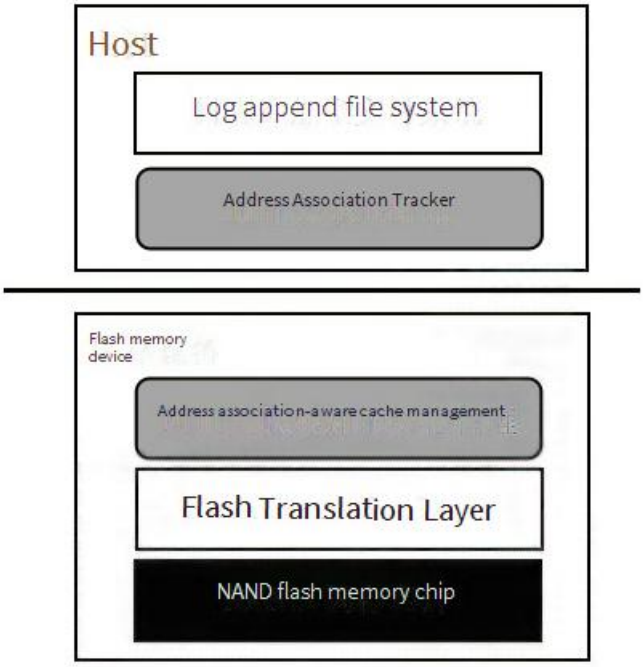


Figure 4. System Overall Structure Diagram

4.2. Related Work

Log-structured file systems generate flash-friendly I/O workload characteristics using out-of-place updates. Therefore, the I/O data of updated files at the same file location is assigned different logical block addresses by the file system. This address information is transparent on the host side but invisible to the device side. To provide this reference information to the cache management mechanism on the device side, CAC reorganizes the write I/O structure pointing to the same file block.

To obtain the associated addresses for each file block, CAC integrates an address association tracker module on the host side. This module maintains a list of address pairs to

continuously track logical addresses assigned to the same file block. Each associated address pair (LBA_pre, LBA_Cur) represents the old address (LBA_pre) and the new address (LBA_Cur) of the updated file I/O. Figure 5 illustrates how the tracker module tracks associated data addresses. For example, the third block of File 1 is stored at logical address 102 in the file system. To update this block, a new logical address (LBA 103) is assigned to the update I/O, and the old address LBA 102 is marked invalid. At this point, the associated address pair (LBA 102, LBA 103) is created and will be bound to the update I/O and sent to the flash device.

During execution, the biggest challenge in the address tracking process is how to transfer the address association information from the file system layer to the flash device. This paper uses the private member of the bio structure, which represents the I/O block device layer, to carry the LBA_pre information. Since the bio structure is created in memory on the host side for write data, which already carries LBA_Cur, the proposed tracking scheme incurs minimal overhead.

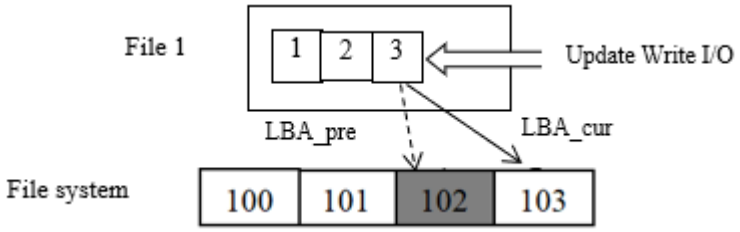


Figure 5. Updated File I/O Data Management Example

4.3. Related Work

When a new write I/O is received, the device controller first checks whether the write I/O is an update I/O based on the associated address pair information. If it is, the controller replaces the logical address LBA_pre in the cache with LBA_Cur. If it is not an update I/O, the controller keeps the current address. To implement this, CAC introduces an address remapping table (ART) in the controller to store remapping information (LBA_Cur, LBA_pre).

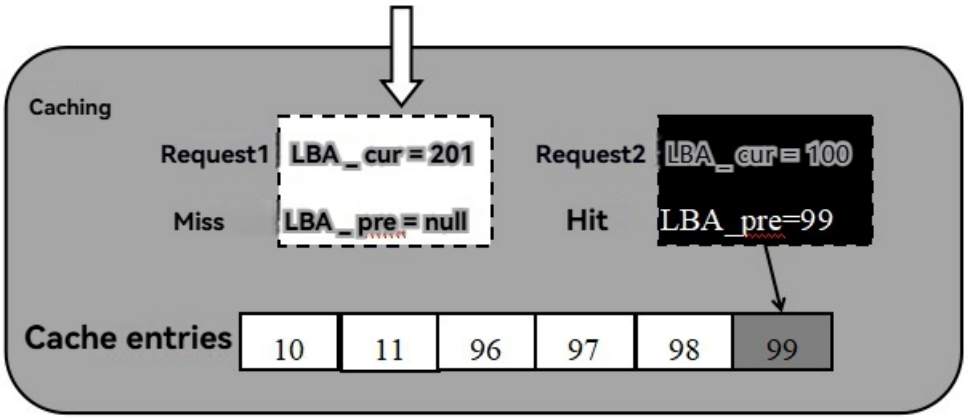


Figure 6. CAC Cache Management Strategy Example

Figure 6 provides an example of the CAC cache management strategy. For Request 1, since neither LBA_Cur nor LBA_pre hits the cache, it results in a cache miss. However, after the associated data remapping, Request 2's LBA_pre hits the cache. The mapping of file block addresses is done in 4KB blocks.

The associated address remapping technique in CAC enhances the locality characteristics of I/O workloads in log-structured file systems, and it can be coupled with existing cache management techniques that utilize I/O locality to improve cache hit rates. It is important to note that log-structured file systems may activate atomic write operations to reduce write pressure from database transactions, thereby eliminating write operations for database logs and ensuring data consistency. In atomic write operations, the file system uses a copy of the old version of the new file data, which is marked invalid in logical space, to enable rollback recovery in case of system crashes or power outages. In this case, both the old and new versions of updated file I/Os must be stored on the storage device. To meet this requirement, the flash internal cache needs to retain both versions of the file block's data. Therefore, CAC should not directly remap two consecutive version updates, but instead remap the current version's write data and the version before the previous one.

4.4. Overhead Analysis

In CAC, the associated address information, i.e., the associated address pair, can be transmitted from the host to the flash device using current high-level block commands, such as SCSI commands for SATA SSDs. For SSDs, the information transmission technique required by CAC is supported in current mainstream SATA and NVMe interface protocols. For mobile devices, the associated address pair can be embedded in write I/O requests using the Tag Identification method in the eMMC protocol [22]. Therefore, CAC does not introduce any additional I/O overhead.

On the host side, the associated address information is carried in the reserved area of the bio structure that corresponds to the write I/O request's block data structure, thus incurring no extra space overhead. On the device side,

the flash controller maintains the address remapping table in the flash internal RAM cache to record the remapping information of updated write I/Os. Assuming a cache capacity of 128MB, with each cache block being 4KB, and each cache mapping entry requiring 4B of cache space, the space overhead introduced by the address remapping table in CAC would be at most 128KB, which accounts for only 0.1% of the entire cache space.

5. Experiments

5.1. Experimental Environment

5.1.1. Experimental Platform

Table 1. SSDsim Parameter Configuration

SSD Capacity	128GB
Cache Capacity	128MB
Number of Channels	8
Number of Chips per Channel	8
Number of Dies per Chip	2
Number of Planes per Die	2
Number of Blocks per Plane	2048
Number of Pages per Block	64
Page Size	4KB
Over-provisioning Ratio	10%
Garbage Collection Strategy	Greedy Algorithm
Address Allocation Strategy	Static Allocation
Read Physical Page Latency	25μs
Write Physical Page Latency	300μs
Erase Latency	2ms
Cache Access Latency	1us

The cache management strategy proposed in this paper for flash storage devices requires modifications to the flash device firmware. Therefore, SSDsim, a simulator, was used to validate the proposed correlation-aware cache management strategy [23]. SSDsim is an SSD simulator developed by the Huazhong University of Science and Technology research team. It not only provides accurate simulations of the real-time operation state of SSD devices but also supports advanced parallel instructions within SSDs, making it widely recognized by researchers in academia [13, 16, 24-25, 3]. To simulate the file system layer, we performed I/O workload replay on real devices configured with EXT4 and F2FS file systems, and used the Blktrace tool to recapture the logical addresses of I/O workloads generated by both file systems to replace the original logical addresses. These workloads were then replayed in the SSDsim simulator to ensure the experimental process closely mirrors the real device operation. In the experiments, SSDsim was configured as a 128GB SSD with a 128MB RAM cache, interacting with the host via 8 channels, each connected to 8 flash chips, with each physical page sized at 4KB. The read/write and erase latency overheads of the SSD device were configured strictly according to commercial SSD product manuals [22]. Table 1 describes the specific configuration parameters of the simulator used in this paper.

5.1.2. I/O Workloads

Table 2. I/O Load Characteristics

	Total Size (GB)	Duration (h)	Read/Write Ratio	I/O Interval (ms)	Update Write Ratio
MDS_0	7.72	166.91	88.11%	0.50	98.25%
PRN_0	46.98	167.98	89.21%	0.11	93.94%
PRN_1	32.27	167.04	24.66%	0.05	87.68%
PROJ_3	2.75	167.83	5.18%	0.27	80.46%
PRXY_0	56.41	167.41	96.93%	0.05	97.70%
RSRCH_0	11.34	167.93	90.68%	0.42	98.25%
SRC2_0	9.79	194.09	88.66%	0.45	97.26%
STG_0	15.82	167.89	84.81%	0.30	98.67%
STG_1	6.28	167.74	36.25%	0.27	97.68%
TS_0	11.89	192.71	82.42%	0.39	97.65%
USR_0	13.71	166.07	59.58%	0.27	98.18%
WDEV_0	7.49	167.98	79.92%		

In this paper, we use the open-source I/O workload set released by MSR Cambridge (MSRC), which was collected and released by Microsoft and the Cambridge University team from product servers [21]. Since these I/O workloads were collected based on traditional Hard Disk Drive (HDD) devices, they do not reflect the I/O characteristics of SSD devices. Therefore, Jung [26-27] re-played these workloads on an SSD hardware platform and re-collected and published them for use with SSDs. In this paper, we use 12 re-collected MSRC workloads to validate the proposed cache

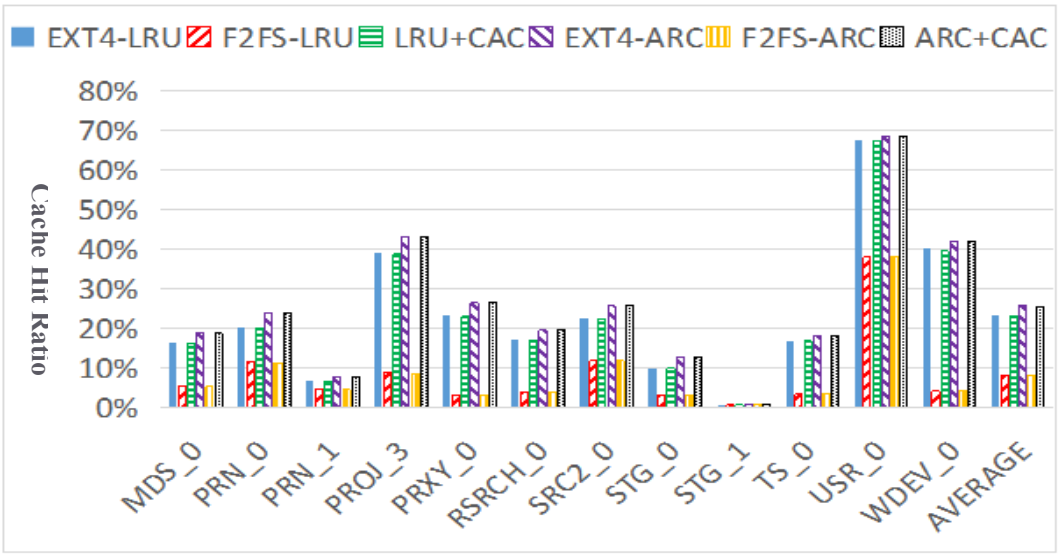
management strategy. Table 2 describes the relevant features of the I/O workloads used in this paper, including the total size (the total size of all write I/Os), duration (the time difference between the first and last I/O), I/O interval (the average time interval between adjacent I/Os), and the update write ratio (the ratio of file update I/Os to all write I/Os). It can be seen that the I/O workloads used in this paper are diverse in terms of total size, read/write ratio, and I/O intervals, which can fully validate the optimization effects of the proposed strategy under different workload environments. Additionally, all I/O workloads have long durations, sufficient to verify the long-term stability of the proposed strategy. Moreover, all I/O workloads have a high proportion of update I/Os, which would maintain a high write I/O hit rate in in-place update environments but are significantly impacted by out-of-place updates in LFS.

5.1.3. Comparison Approaches

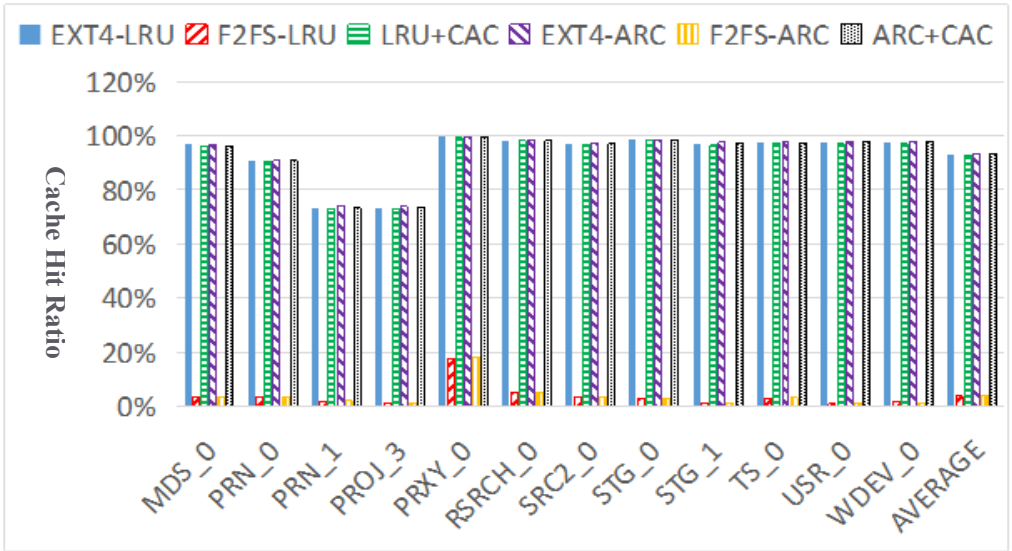
Since the proposed cache management strategy is coupled with existing strategies that rely on I/O locality features to improve cache hit rates, two classic cache management strategies are adopted in the experiments. These strategies are validated both with and without the proposed management strategy. The two algorithms used are Least Recently Updated (LRU) [10] and Adaptive Replacement Cache (ARC) [20]. The LRU algorithm suggests that the most recently accessed I/O data is identified as hot data and stored in the cache, with the hottest data placed at the front of the cache and the least hot data more likely to be evicted. The ARC algorithm maintains two I/O data lists: the time-domain list for recently accessed data and the frequency-domain list for frequently accessed data. It also maintains shadow lists for each main list to store evicted data and dynamically adjusts the sizes of the two main lists based on the cache misses in the shadow lists. We selected these two strategies as the coupling algorithms for the proposed cache management strategy because most related work is based on these classic algorithms [10-15, 17-19]. If the proposed management strategy achieves better cache management efficiency when coupled with these two algorithms, it is likely to also improve cache management efficiency when used with other related work.

5.2. Experimental Results and Analysis

5.2.1. Cache Hit Rate



(a) Read I/O Cache Hit Ratio



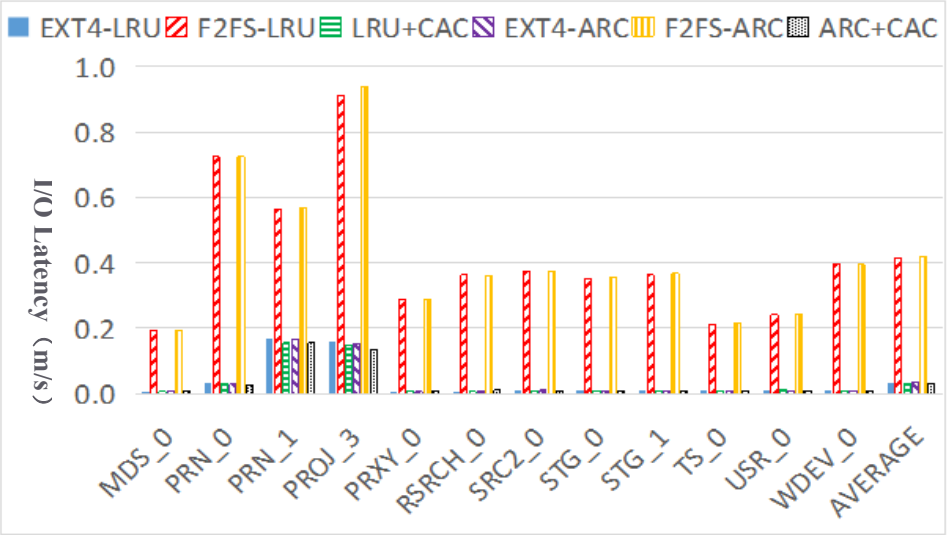
(b) Write I/O Cache Hit Ratio

Figure 7. Cache Hit Ratio Optimization Effect

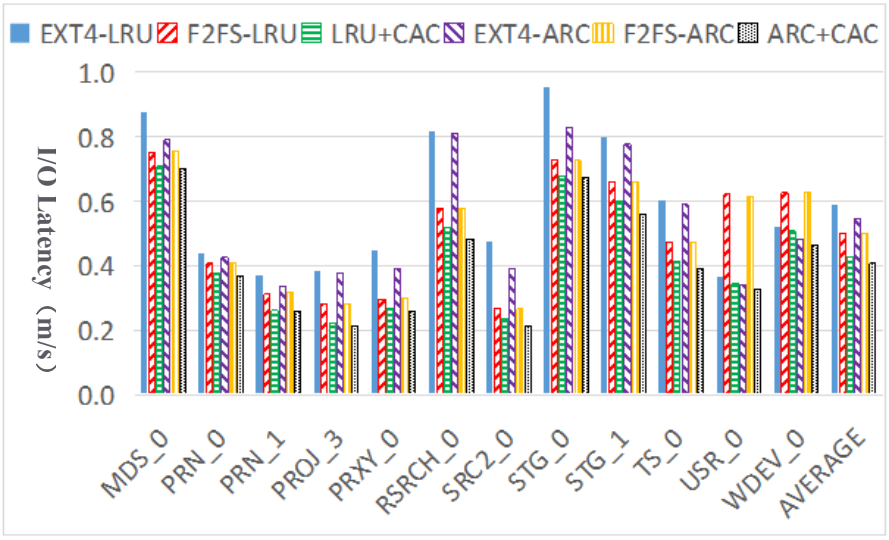
Figure 7 shows the read and write I/O cache hit rates for all schemes, both with and without CAC. The "+CAC" label indicates the scenario where the LRU and ARC strategies work in conjunction with CAC under the F2FS file system. It can be observed that when working in conjunction with CAC for cache management, the read and write cache hit rates for LRU and ARC strategies essentially recover to the levels observed under the EXT4 file system. This demonstrates that CAC can effectively restore I/O locality disrupted by out-of-place updates through correlation-aware I/O address remapping, thereby improving cache hit rates and fully leveraging the flash-friendly advantages of LFS. As shown in Figure 7, under the F2FS file system, compared to LRU working alone, LRU+CAC improves the average read I/O hit rate from 8.1% to 23.1% and the average write I/O hit rate from 3.8% to 92.7%. Compared to ARC, ARC+CAC improves the average read I/O hit rate from 8.1% to 25.5% and the average write I/O hit rate from 3.9% to 93.0%. The improved cache hit rates directly contribute to better I/O performance and extended flash device lifespan.

5.2.2. I/O Performance

Figure 8 compares the I/O latency with and without CAC under LRU and ARC strategies. The experimental results confirm that CAC effectively improves cache hit rates under LFS, thus fully utilizing its flash-friendly characteristics. When LRU is used, CAC reduces the average read I/O latency by 14.5% and write I/O latency by 92.4%. When ARC is used, CAC reduces the average read I/O latency by 18.4% and write I/O latency by 92.8%. The reason for the long write I/O latency under LFS is that out-of-place updates to logical addresses prevent the cache from recognizing the hit I/O, causing many write I/Os to trigger dirty page writes, which significantly reduce cache performance efficiency. CAC, with its correlation-aware remapping, enables the cache management mechanism to recognize the hit I/Os by transmitting the old addresses of updated I/Os, fundamentally resolving this issue. For read I/Os, the long latency is mainly caused by the large amount of invalid data occupying the cache, which lowers the cache hit rate, and the large number of dirty page writes causing write I/O to stall and indirectly interfere with read I/O. CAC improves the cache hit rate for write I/Os, further mitigating write I/O stalling and thereby enhancing read I/O performance.



(a) Write I/O Latency



(b) Read I/O Latency

Figure 8. I/O Average Latency Optimization Chart

Furthermore, when compared to EXT4's I/O performance, CAC can simultaneously leverage the flash-friendly characteristics and cache performance advantages of F2FS. In F2FS, LRU+CAC reduces the average read I/O latency by 27.2% and write I/O latency by 7.5%, compared to EXT4's LRU strategy. Similarly, ARC+CAC reduces the average read I/O latency by 25.0% and write I/O latency by 8.0%, compared to EXT4's ARC strategy. This is because, for I/Os that hit the cache under EXT4, ARC can restore their hit rate under F2FS, and for non-hitting I/Os, F2FS's flash-friendly nature provides better performance than EXT4.

5.2.3. Flash Device Lifespan

Figure 9 compares the number of dirty page writes with and without CAC under LRU and ARC strategies. Since the number of dirty page writes directly reflects the consumption of the flash P/E cycle resource, we use this metric to assess CAC's contribution to extending the flash device's lifespan. When LRU is used, CAC reduces the number of dirty page writes by 90.8%, compared to its individual performance. When ARC is used, CAC reduces the number of dirty page writes by 97.8%, compared to its individual performance. The reasons for the increased number of dirty page writes under LFS are twofold: the main reason is the inability of the cache to perceive the associated addresses before and after the I/O update, which causes a sharp decline in write hit rates and leads to a large number of dirty page writes. The indirect reason is that a large amount of invalid data occupies the cache, further compressing available cache resources. As previously analyzed, both of these issues can be fundamentally solved by CAC using correlation-aware remapping, resulting in significantly fewer dirty page writes under LFS and effectively extending the flash device's lifespan.

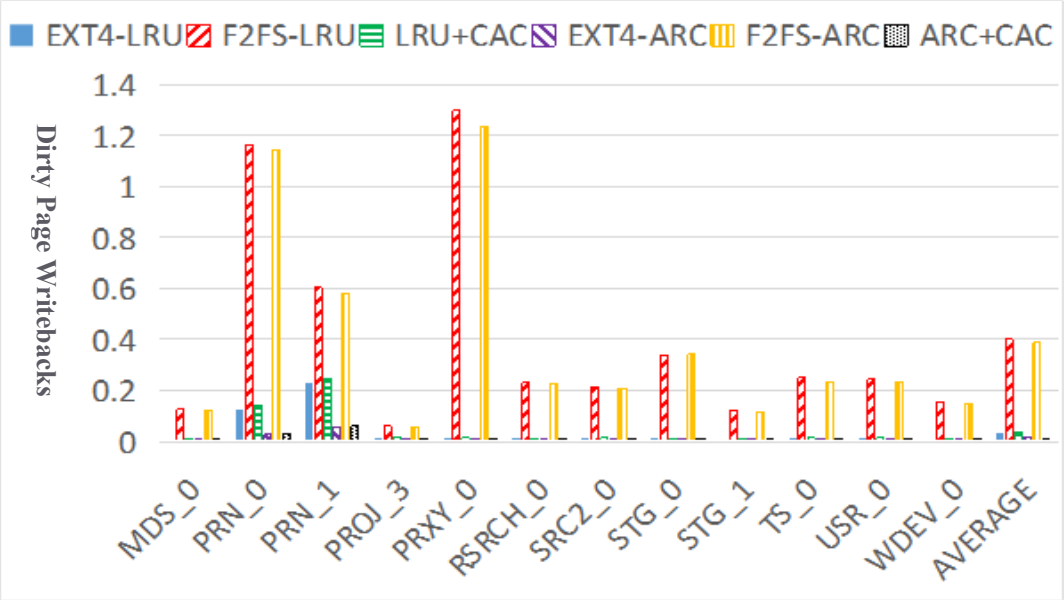


Figure 9. Dirty Page Writebacks Optimization Effect

5.2.4. Sensitivity Analysis

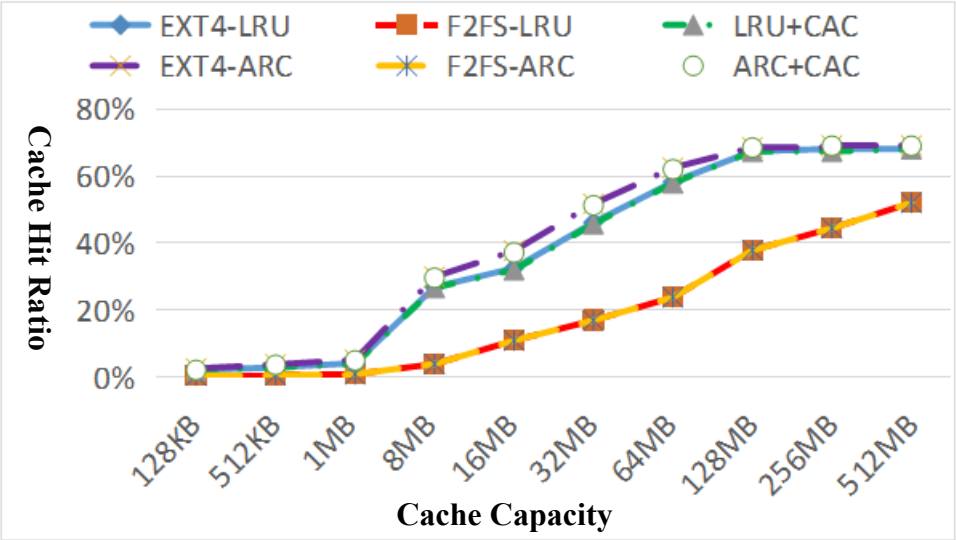
Cache size has a decisive impact on cache hit rates. Since this paper primarily addresses the impact of out-of-place updates in LFS on cache hit rates, this section presents experiments to analyze the sensitivity of the proposed CAC strategy to different cache size configurations. Although most mainstream SSDs configure their internal cache as 1% of the SSD's storage capacity, lower-end SSD products still use smaller cache sizes to reduce costs. This analysis is of great significance for SSD products with different cache configurations in LFS environments.

In this experiment, the USR_0 workload, which has a high cache hit rate in EXT4, was chosen as an example to test its cache hit rate nder different cache size configurations. Figure 10 presents the experimental results, from which the following conclusions can be drawn:

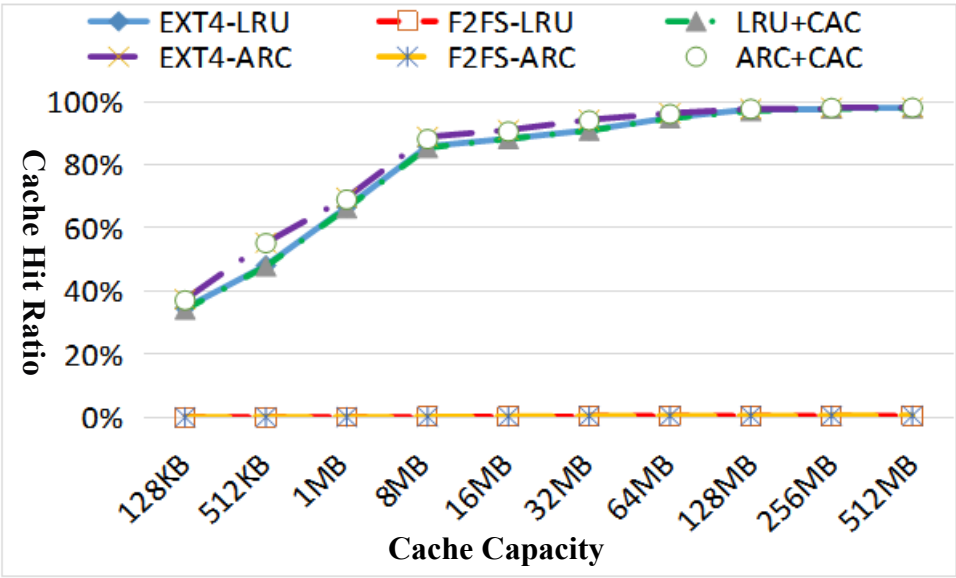
1) When the cache capacity is small, the I/O cache hit rate directly benefits from the increase in cache capacity. However, this benefit saturates once the cache capacity reaches a certain threshold, as the cache becomes large enough to hold most of the hit I/Os.

2) Under different cache capacity configurations, F2FS's out-of-place updates have a huge impact on the cache hit rate. Write hit rates drop to 0% due to out-of-place updates, and read hit rates also decrease to varying degrees.

3) Under different cache capacity configurations, the proposed CAC strategy can recover the cache hit rate significantly impacted by out-of-place updates, bringing it closer to the hit rate achieved by in-place updates in EXT4.



(a) Read I/O Cache Hit Ratio vs. Cache Capacity



(b) Write I/O Cache Hit Ratio vs. Cache Capacity

Figure 10. Sensitivity of USR_0 Load I/O Cache Hit Ratio to Cache Capacity

From these experiments, it can be concluded that the proposed strategy fundamentally addresses the issues identified in this paper. By enjoying the flash-friendly advantages of LFS, flash storage devices can fully leverage the performance advantages of internal caches, leading to significant improvements in I/O performance and greatly extending flash lifespan.

6. Summary

Traditional cache management strategies rely on using I/O locality to improve cache hit rates. However, when a computer system employs a log-structured file system (LFS), the out-of-place update method it uses disrupts the I/O locality characteristics, leading to a sharp decline in the cache hit rate within flash storage devices. From the perspective of the flash device, I/O updates that should point to the same file block data are treated as two separate write I/Os by the flash device due to the change in their logical block addresses after using the log-structured file system.To address this issue, this paper proposes a correlation-aware cache management strategy to enhance the cache hit rate within the flash storage device, thereby improving system I/O performance and extending the lifespan of the flash device. Experimental results demonstrate that the proposed strategy can effectively improve cache hit rates, achieving the expected optimization effects. Since the proposed solution is fully compatible with current cache management strategies and incurs minimal implementation overhead, we hope that this strategy can serve as a reference for future research and flash storage developers.

References

[1] Rosenblum Mendel, Ousterhout K. John, et al. The Design and Implementation of a Log-Structured File System. ACM Transactions on Computer Systems. 1992, Vol. 10 (No. 1), p. 26-52.

[2] Lee Changman, Sim Dongho, Hwang Joo-Young, Cho Sangyeun, et al. F2FS: A New File System for Flash Storage. Proceedings of the USENIX Conference on File and Storage Technologies. Santa Clara, CA, USA, 2015, p. 273-286.

[3] Wu Chao, Li Qiao, Ji Cheng, Kuo Teiwei, Xue Chun Jason, et al. Boosting User Experience via Foreground-Aware Cache Management in UFS Mobile Devices. IEEE Transactions on

Computer-Aided Design of Integrated Circuits and Systems. 2020, Vol. 39 (No. 11), p. 3263-3275.

[4] Jiang Song, Zhang Xiaodong, et al. LIRS: An Efficient Low Inter-Reference Recency Set-Replacement Policy to Improve Buffer Cache Performance. ACM SIGMETRICS Performance Evaluation Review. 2002, Vol. 30 (No. 1), p. 31-42.

[5] Megiddo Nimrod, Modha Devesh, et al. Arc: A Self-tuning, Low Overhead Replacement Cache. Proceedings of the USENIX Conference on File and Storage Technologies. San Francisco, USA, 2003, p. 115-130.

[6] Park Sejin, Park Chanik, et al. Frd: A Filtering Based Buffer Cache Algorithm That Considers Both Frequency and Reuse Distance. Proceedings of the International Conference on Massive Storage Systems and Technology. Santa Clara, CA, USA, 2017, p. 1-12.

[7] Products With F2FS on: https://f2fs.wiki.kernel.org/, 2016

[8] Liang Yu, Ji Cheng, Fu Chenchen, Ausavarungnirun Rachata, Li Qiao, Pan Riwei, Chen Siyu, Shi Liang, Kuo Tei-Wei, Xue Chun Jason, et al. iTrim: I/O-Aware Trim for Improving User Experience on Mobile Devices. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2020 (early access).

[9] Li Fei, Lu Youyou, Wu Zhongjie, Shu Jiwu, et al. ASCache: An Approximate SSD Cache for Error-Tolerant Applications. Proceedings of Design Automation Conference. Las Vegas, NV, USA, 2019, p. 1-6.

[10] Liu Jiahao, Wang Fang, Feng Dan, et al. CostPI: Cost-Effective Performance Isolation for Shared NVMe SSDs. Proceedings of the International Conference on Parallel Processing. Kyoto, Japan, 2019, p. 1-10.

[11] Zhou Bo, Ding Chuanming, Lv Yina, Xue Chun Jason, Zhuge Qingfeng, Sha H.-M. Edwin, Shi Liang, et al. SAC: A Stream Aware Cache Scheme for Multi-Streamed Solid State Drives. Proceedings of Asia and South Pacific Design Automation Conference. Tokyo, Japan, 2021, p. 645-650.

[12] Gao Congming, Di Yejia, Deng Aosong, Liu Duo, Ji Cheng, Xue Chun Jason, Shi Liang, et al. F2FS Aware Mapping Cache Design on Solid State Drives. Proceedings of Non-Volatile Memory Systems and Applications Symposium. Hakodate, Japan, 2018, p. 31-36.

[13] Gao Congming, Shi Liang, Li Qiao, Liu Kai, Xue Chun Jason, Yang Jun, Zhang Youtao, et al. Aging Capacitor Supported Cache Management Scheme for Solid-State Drives. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2020, Vol. 39 (No. 10), p. 2230-2239.

[14] Liu Weiguang, Cui Jinhua, Liu Junwei, Yang T, et al. Laurence. MLCache: A Space-Efficient Cache Scheme based on Reuse Distance and Machine Learning for NVMe SSDs. Proceedings of IEEE/ACM International Conference on Computer Aided Design. New York, USA, 2020, p. 1-9.

[15] Sun Hui, Dai Shangshang, Huang Jianzhong, Qin Xiao, et al. Co-Active: A Workload-Aware Collaborative Cache Management Scheme for NVMe SSDs. IEEE Transactions on Parallel and Distributed Systems. 2021, Vol. 32 (No. 6), p. 1437-1451.

[16] Yoo Sangjin, Shin Dongkun, et al. Reinforcement Learning-based SLC Cache Technique for Enhancing SSD Write Performance. Proceedings of USENIX Workshop on Hot Topics in Storage and File Systems. Boston, USA, 2020, p. 1-6.

[17] Tan Yujuan, Xie Jing, Xu Congcong, Yan Zhichao, Jiang Hong, Zhao Yajun, Fu Min, Chen Xianzhang, Liu Duo, Xia Wen, et al. CDAC: Content-Driven Deduplication-Aware Storage Cache. Proceedings of Symposium on Mass Storage Systems and Technologies. Santa Clara, CA, USA, 2019, p. 282-291.

[18] Wang Qiuping, Li Jinghong, Xia Wen, Kruus Erik, Debnath Biplob, Patrick P.C. Lee, et al. Austere Flash Caching with Deduplication and Compression. Proceedings of USENIX Annual Technical Conference. Virtual Conference, 2020, p. 713-726.

[19] Li Wenji, Baptise Jean Gregory, Riveros Juan, Narasimhan Giri, Zhang Tony, Zhao Ming, et al. CacheDedup: In-Line Deduplication for Flash Caching. Proceedings of USENIX Conference on File and Storage Technologies. Santa Clara, CA, USA, 2016, p. 301-314.

[20] Megiddo Nimrod, Modha S. Dharmendra, et al. ARC: A Self-Tuning, Low Overhead Replacement Cache. Proceedings of USENIX Conference on File and Storage Technologies. San Francisco, CA, USA, 2003, p. 115-130.

[21] Narayanan Dushyanth, Theresha Eno, Donnelly Austin, Elnikety Sameh, Rowstron I.T. Antony, et al. Migrating Server Storage to SSDs: Analysis of Tradeoffs. Proceedings of the European Conference on Computer Systems. Nuremberg, Germany, 2009, p. 145-158.

[22] Micron flash memory chip MLC specification on: https://www.micron.com/-/media/client/global/documents/products/data-sheet/nand-flash/20-series/2gb_nand_m29b.pdf 2004

[23] Hu Yang, Jiang Hong, Feng Dan, Tian Lei, Luo Hao, Zhang Shuping, et al. Performance Impact and Interplay of SSD Parallelism through Advanced Commands, Allocation Strategy and Data Granularity. Proceedings of the International Conference on Supercomputing. Tucson, Arizona, USA, 2011, p. 96-107.

[24] Wu Chao, Ji Cheng, Li Qiao, Gao Congming, Pan Riwei, Fu Chenchen, Shi Liang, Xue Chun Jason, et al. Maximizing I/O Throughput and Minimizing Performance Variation via Reinforcement Learning. IEEE Transactions on Computers. 2020, Vol. 69 (No. 1), p. 72-86.

[25] Wu Chao, Cui Yufei, Ji Cheng, Kuo Teiwei, Xue Chun Jason, et al. Pruning Deep Reinforcement Learning for Dual User Experience and Storage Lifetime. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2020, Vol. 39 (No. 11), p. 3993-4005.

[26] Liu Chun Yi, Kotra B. Jagadish, Jung Myoungsoo, Kandemir T. Mahmut, Das R. Chita, et al. SOML Read: Rethinking the Read Operation Granularity of 3D NAND SSDs. Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems. RI, USA, 2019, p. 955-969.

[27] Kwon Miryeong, Zhang Jie, Park Gyuyoung, Choi Wonil, Donofrio David, Shalf John, Kandemir T. Mahmut, Jung Myoungsoo, et al. TraceTracker: Hardware/Software Co-Evaluation for Large-Scale I/O Workload Reconstruction. Proceedings of the IEEE International Symposium on Workload Characterization. Seattle, WA, USA, 2017, p. 87-96.

[28] Jeong Sooman, Lee Kisung, Hwang Jungwoo, Lee Seongjin, Won Youjip, et al. Framework for analyzing Android I/O stack behavior: from generating the workload to analyzing the trace. Future Internet. 2013, Vol. 5 (No. 4), p. 591-610.

[29] Katcher Jefferey, et al. Postmark: A New File System Benchmark. NetAPP Inc, Technical Report: TR3022.California.1997.

[30] Kim Youngjae, Tauras Brendan, Gupta Aayush, Urgaonkar Bhuvan, et al. Flashsim: A Simulator for NAND Flash-Based Solid State Drives. In Proceedings of First International Conference on Advances in System Simulation. Porto, Portugal, 2009: 125–131.Kim Youngjae, Tauras Brendan, Gupta Aayush, Urgaonkar Bhuvan, et al. Flashsim: A Simulator for NAND Flash-Based Solid State Drives. Proceedings of First International Conference on Advances in System Simulation. Porto Portugal, 2009, p. 125-131.

[31] Min Changwoo, Lee Sang-Won, Eom Young Ik, et al. Design and Implementation of a Log-structured File System for Flash-based Solid State Drives. IEEE Transactions on Computers. 2014, Vol. 63 (No. 9), p. 2215-2227.

[32] Block-I/O Layer Tracing: Blktrace[J] on: https://www.mimuw.edu.pl/~lichota/09-10/Optymalizacja-open-source/Materialy/10%20-%2-Dysk/gelato_ICE06apr_blktrace_brunelle_hp.pdf 2006,4