

LAC: A Workload Intensity-Aware Caching Scheme for High-Performance SSDs

Hui Sun , Haoqiang Tong , Yinliang Yue , and Xiao Qin 

I. INTRODUCTION

Abstract—Inside an NAND Flash-based solid-state disk (SSD), utilizing DRAM-based write-back caching is a practical approach to bolstering the SSD performance. Existing caching schemes overlook the problem of high user I/Os intensity due to the dramatic increment of I/Os accesses. The hefty I/O intensity causes access conflict of I/O requests inside an SSD: a large number of requests are blocked to impair response time. Conventional passive update caching schemes merely replace pages upon access misses in event of full cache. Tail latency occurs facing a colossal I/O intensity. Active write-back caching schemes utilize idle time among requests coupled with free internal bandwidth to flush dirty data into flash memory in advance, lowering response time. Frequent active write-back operations, however, cause access conflict of requests – a culprit that expands write amplification (WA) and degrades SSD lifetime. We address the above issues by proposing a *workLoad* intensity-aware and Active parallel Caching scheme - LAC - that is powered by collaborative-load awareness. LAC fends off user I/Os' access conflict under high-I/O-intensity workloads. If the I/O intensity is low – intervals between consecutive I/O requests are large – and the target die is free, LAC actively and concurrently writes dirty data of adjacent addresses back to the die, cultivating clean data generated by the active write-back. Replacing clean data in priority can reduce response time and prevent flash transactions from being blocked. We devise a data protection method to write back cold data based on various criteria in the cache replacement and active write-backs. Thus, LAC reduces WA incurred by actively writing back hot data and extends SSD lifetime. We compare LAC against the six caching schemes (LRU, CFLRU, GCaR-LRU, MQSim, VS-Batch, and Co-Active) in the modern MQSim simulator. The results unveil that LAC trims response time and erase count by up to 78.5% and 47.8%, with an average of 64.4% and 16.6%, respectively.

Index Terms—Caching scheme, I/O-intensity awareness, parallel write, die-level monitor, solid state disk, NAND flash.

Manuscript received 20 July 2023; revised 12 February 2024; accepted 30 March 2024. Date of publication 4 April 2024; date of current version 11 June 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62072001. The work of Xiao Qin was supported in part by the U.S. National Science Foundation under Grant IIS-1618669, Grant OAC-1642133, and Grant CCF-0845257. Recommended for acceptance by M. T. Kandemir. (Corresponding author: Hui Sun.)

Hui Sun and Haoqiang Tong are with the School of Computer Science and Technology, Anhui University, Hefei 230201, China (e-mail: sunhui@ahu.edu.cn).

Yinliang Yue is with Zhongguancun Laboratory, Beijing 100049, China (e-mail: yylhust@qq.com).

Xiao Qin is with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849-5347 USA (e-mail: xqin@auburn.edu).

Digital Object Identifier 10.1109/TC.2024.3385290

NAND Flash-based solid-state drives (SSDs) are widely deployed in storage systems [1] thanks to exceptional random performance and low power consumption [2]. Unfortunately, NAND flash inside an SSD has a limited number of Program/Erase cycles (P/E) [3], which determines SSD lifetime. SSDs use built-in DRAM as a cache [4] to slash the number of user requests of updating flash memory [5] – and the performance difference between the host-side I/O interface and flash memory [6] can be balanced out. As such, user response time is significantly reduced. Existing workloads are featured with high I/O intensity [7] and burstiness [8]. In an SSD, data volume per unit time - intensity - increases dramatically in real-world loads in recent years, affecting user response time and causing high tail latency¹. Taking the Systor'17 traces [8] for example, the I/O intensity is large, and the distribution of the request intervals is unbalanced. The percentage of logical page addresses (LPAs) that are rewritten is high, accompanied by a short rewrite distance. This metric is the number of requests between two write requests to the same LPA. In our tested SSD (see Section III) serving user requests, there are numerous free chips and dies in the flash memory. Although the chip is busy, most dies are free.

Conventional caching schemes applied in SSDs are primarily classified into three camps. First, the SSD front-end caching algorithms [9], [10], [11] are designed based on I/O access patterns - hot and cold, temporal locality, spatial locality, and I/O intensity. More often than not, the above caching schemes overlook the I/O intensity issue. Facing high I/O intensity, the replacement of dirty data causes access conflict with subsequent requests, the process of which is delayed. Second, the back-end caching schemes [12], [13], [14] exploit the read-write imbalance to preferentially replace clean data. This technique is adequate for the chip-level state and idle time in flash memory to prioritize replacing pages in target idle chips [15]. However, the fully parallelism in SSDs has been rarely studied. Third, the front-back-end caching schemes [3], [16] combine both I/O access pattern and flash features, including I/O hotness, I/O intensity, cache-miss cost, and access conflict between user I/Os and internal I/Os in flash memory. These schemes ignore the relationship between I/O intensity and flash write latency, thereby wasting flash memory resources. When the I/O intensity

¹The term *tail latency* refers to extreme values at the tail end of the latency distribution under workloads with high I/O intensity in this paper

is high, a large number of I/O conflicts are incurred by flash transactions, and response time is enlarged.

To solve the above problems, we propose a caching scheme that is collaboratively aware of user I/O access patterns and flash memory status – and simultaneously detects (1) I/O hotness and I/O intensity at the user level and (2) write latency and state in flash memory. The overarching goal of LAC is to mitigate access conflict between user requests in workloads with high I/O intensity. When LAC detects that a request interval becomes long and a target die in flash memory is free, the dirty data with adjacent logical addresses is actively written back to flash memory in parallel. Afterward, LAC increases the amount of the clean data generated by active write back, lowers response time, and avoids flash transactions from being blocked. To reduce write amplification (WA) caused by actively writing hot data to flash memory, we divide cache space into cold, warm, and hot regions. We propose a data protection method to evict cold data in the cache based on different criteria in the cache replacement and active write-backs, aiming to extend SSD lifetime. Our contributions are tabulated as follows.

▷ We propose a workload-intensity-aware and active parallel caching scheme named *LAC* that is adaptive. *LAC* alleviates access conflict arising from user I/Os under workloads with high I/O intensity. When the I/O intensity is low, *LAC* increases the number of active write-backs for dirty data and expands the size of clean data to be preferentially replaced, thereby minimizing cache replacement cost.

▷ We devise an approach in *LAC* to predicting I/O intensity by leveraging the status in flash memory. This method mitigates access conflict and minimizes response time by slashing the parallel write backs of cold and dirty data in the context of low I/O intensity accompanied by idle target die.

▷ We design an algorithm focused on safeguarding hot data from excessive write-backs to flash memory, enhancing the SSD lifetime. *LAC* protects hot data during the cache replacement and parallel write-back, averting redundant flash transactions due to repeated dispatch² after a write-back.

▷ We validate the advantages of *LAC* by comparing against the popular caching schemes under realistic traces. Our findings demonstrate that *LAC* reduces the response time and erase count of the alternative caching schemes by up to 78.5% and 47.8% with averages of 64.4% and 16.6%, respectively.

The rest of the paper is organized as follows. We introduce the background in Section II. We analyze the motivation behind the development of *LAC* in Section III. Section IV elaborates on the design of *LAC*. After presenting the experimental setup in Section V, we validate *LAC*'s performance by comparing it with the state-of-the-art caching schemes in Section VI. We present the related work in Section VII. Finally, Section VIII discusses and concludes this paper.

II. BACKGROUND

SSD Architecture. A typical NVMe SSD [14] comprises host- and device-side subsystems. Within NVMe drives, multiple submission queues (SQs) and completion queues (CQs) are

²Writing hot data back may result in multiple rewrites of the same data and dispatch duplicate flash write transactions.

configured in the host-side DRAM. I/O requests, generated by a host bypassing the OS I/O stack and block-level queue and directly inserted into the SQ, are submitted to an SSD device for scheduling. After the I/O requests are executed, the status is returned to the CQ. The device side is divided into a front end and a back end: the front end includes the host interface logic (HIL) [14] and flash translation layer (FTL)), whereas the back end embraces storage NAND flash arrays. The HIL converts user requests into flash transactions. In the FTL, flash transactions pass through the cache units. The cache space, containing a portion of the DRAM, stores accessed page requests with a low latency. The FTL maintains the I/O queue for each chip, and the flash transactions in the I/O queue are then written/read to the flash chip through the transaction schedule and flash channel controller (FCC). An SSD has channel-, chip-, die-, and plane-level parallelisms. A die is a basic unit for executing commands; a block is the basic unit of an erase operation; and a page is the basic unit of read and write operations. An erase duration is around ten times of a write duration. The delay of read is lower than those of the other two operations. Flash memory has a limited number of program/erases (P/Es), which determines SSD lifetime [2], [3].

Caching scheme. Caching influences the SSD performance. Caching schemes encompass active and passive write-back schemes [3], which are determined by the data-update method. In the passive write-back mode, pages are written back to the flash memory solely when the cache reaches its capacity or when a cache miss happens. The cache space then can be freed for incoming user I/O requests. The active write-back mode writes dirty data back to flash memory while converting the dirty data into clean ones. This method minimizes cache replacement costs at the cost of increasing WA. The replaced data are flushed to the flash memory – and logical-and-physical address mapping is updated

III. MOTIVATIONS

In this section, we analyze the workload and flash parallelism, which motivates us to design *LAC*. We focus on the *I/O-intensity* workloads from Systor'17 traces [8], see Table I. The prior active write-back modes [3], [17] flush dirty cache data into the target idle chip, lowering the cache replacement cost. If the interval between two consecutive requests is smaller than the duration of an active write-back, subsequent requests are blocked, and the write latency is worsened. Recognizing that the request interval depends on access patterns, we analyze traces to unveil the interval between two consecutive requests.

First, we define I/O intensity in a workload as $\mathbf{W}_{intensity} = \frac{\sum S_{page}}{t}$, which is the data volume from the host per unit time. The denominator t denotes the length of time, whose default value is one second. A hefty I/O intensity $W_{intensity}$ entails a small average request interval T_{avg} , and we have $T_{avg} = \frac{t}{N_{req}-1} = \frac{S_{page}}{W_{intensity} - S_{page}}$. The ratio of I_{max} to I_{avg} for workloads is higher than 10 times. The Systor'17 traces are featured with high I/O intensity, frequent bursts of requests, and short inter-request intervals. In addition, user requests are split into page-sized requests with an interval less than the flash write delay (e.g., 0.75 ms). When an active write-back occurs within

TABLE I
CHARACTERIZATION OF SYSTOR'17 TRACES

Trace ¹	TotalReq (10 ⁶)	Write Ratio	Average Size(KB)	WriteSize (GB)	T_{avg} ² (ms)	T_{max} (s)	I_{avg} ³ (MB/s)	I_{max} (MB/s)	Seq ⁴ Ratio(%)	Rewrite Ratio ⁵
LUN0-0	1.07	0.73	13.42	5.68	3.35	6.49	3.91	174.3	1.97	0.94
LUN0-1	2.31	0.70	15.47	18.90	1.56	1.51	9.69	84.2	1.64	0.93
LUN2-0	2.70	0.69	12.50	13.82	1.33	1.32	9.17	92.8	1.56	0.94
LUN2-1	1.83	0.66	13.57	7.97	1.97	0.61	6.73	99.7	1.93	0.93
LUN2-2	4.44	0.80	10.35	24.14	0.81	0.60	12.50	223.7	0.80	0.97
LUN2-3	2.90	0.79	10.91	13.90	1.24	0.52	8.59	147.8	1.11	0.97
LUN2-4	2.47	0.69	11.32	10.37	1.45	4.39	7.60	89.4	1.13	0.96
LUN2-5	1.47	0.72	11.77	7.66	2.44	5.89	4.71	68.4	0.73	0.94
LUN2-6	2.41	0.86	9.39	13.48	1.49	4.44	6.13	114.8	0.63	0.98
LUN3-0	4.24	0.06	25.06	2.22	0.85	0.13	28.83	151.5	0.96	0.31

*¹ Column 1 represent traces 2016021615-LUN0, 2016031614-LUN0, 2016030117-LUN2, 2016030415-LUN2, 2016030817-LUN2, 2016030819-LUN2, 2016030917-LUN2, 2016031015-LUN2, 2016031115-LUN2, and 2016022113-LUN3, respectively. ² T_{avg} and T_{max} are the mean and maximum values of the request interval, respectively. ³ I_{avg} and I_{max} are the mean and maximum values of the I/O intensity, respectively. ⁴ The Seq Ratio indicates the ratio of sequential I/O requests to ones. ⁵ The Rewrite Ratio denotes the requests with the same LPA occurring re-write.

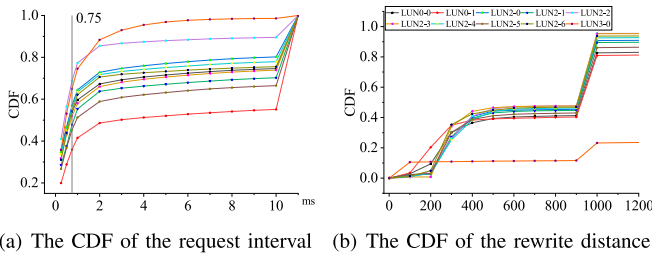


Fig. 1. The CDF of workload analysis.

this interval, the write-back delays the subsequent request. The ratio of T_{max} to T_{avg} is close to 10^3 times. The larger the disparity between the two metrics, the greater the imbalance in the workload's access pattern and the more pronounced the burstiness. Fig. 1(a) plots the cumulative distribution function (CDF) curve of request intervals in the workloads, where the requests with an interval less than 0.75 ms account for 35.9% to 68.8% of all the requests.

Observation 1: The Systor'17 traces show both high I/O intensity and frequent bursts of intense I/O activity. Request interval sizes show uneven distribution, and active write-backs in short intervals without I/O intensity prediction tend to block requests. It is wise to leverage a large-size request interval to actively write dirty data back to flash memory amid low I/O intensity at the host side; otherwise, active writes should be averted at all costs.

Second, in the last column, we refer to two writes on the same LPA as one rewrite operation; similarly, two read operations on the same LPA are named reread operation. In addition, there are write-read and read-write operations. The rewrite ratio is gauged as a proportion of rewrite operations among the four-type operations. Given an LPA in an SSD, the *rewrite distance* refers to the total number of requests between two write operations on the same LPA. Actively writing back a short-rewrite-distance page can cause extra writes, impairing the response time and SSD lifetime. More often than not, the percentage of rewrite operations exceeds 90% – meaning that a written page is likely be re-accessed by a write request. After studying the CDF curve for rewrite distances ranging from 1 to 10^4 , we observe

that requests with a rewrite distance of less than 10^3 accounted for more than 80%.

Observation 2: A substantial portion of requests involve rewriting the same LPA, with a tendency for these rewrites to occur in close temporal proximity. If dirty data – given a short rewrite distance – are actively updated to flash memory, subsequent host write requests will repeatedly flush back to flash memory to worsen WA. Conversely, a long rewrite distance alleviates this issue.

Third, the fluctuation of rewrite distance is large, as evidenced in prior work [18]. We count the rewrite distances of all requests with the same access frequency in a workload. We study the variance and median of these rewrite distances, see Fig. 2, which signifies similar trends under all the workloads. As the number of I/O accesses goes up, the two metrics drop – and the rate of decrement tendency slows down. We categorize the data points based on the position along the curve: those near the two ends of the curve are defined as cold and hot data, respectively. Cold data are characterized by high variance (10^8) and median (10^4). Hot data have low variance (10^6) and median (3×10^3). Moreover, warm data accesses, which represent intermediate access metrics between hot and cold data, are evident. There are several peaks with the number of I/Os accesses ranging from 10 to 100, as plotted in Fig. 2(b), which unveils that a large number of I/O accesses cannot guarantee low variance and median.

Observation 3: The Systor'17 traces suggest that the rewrite distance of requests is very high due to low access frequency. The rewrite distance becomes low and stable for high access frequency of requests. The rewrite distance highly fluctuates for moderate access frequency of requests. Dividing cache space into – cold, warm, and hot regions, we propose the cache replacement strategies coupled with a data protection method for each region. Our design aims to, by the virtue of improved accuracy of identifying cold and hot data, fend off high WA incurred by write-back hot data.

Fourth, in the Systor'17 traces, the consecutive request intervals are mostly smaller than a flash write latency. In this case, active write-backs commandeer request intervals, exacerbating access conflict between user requests. The number of active write-backs decreases, and this operation converts dirty data

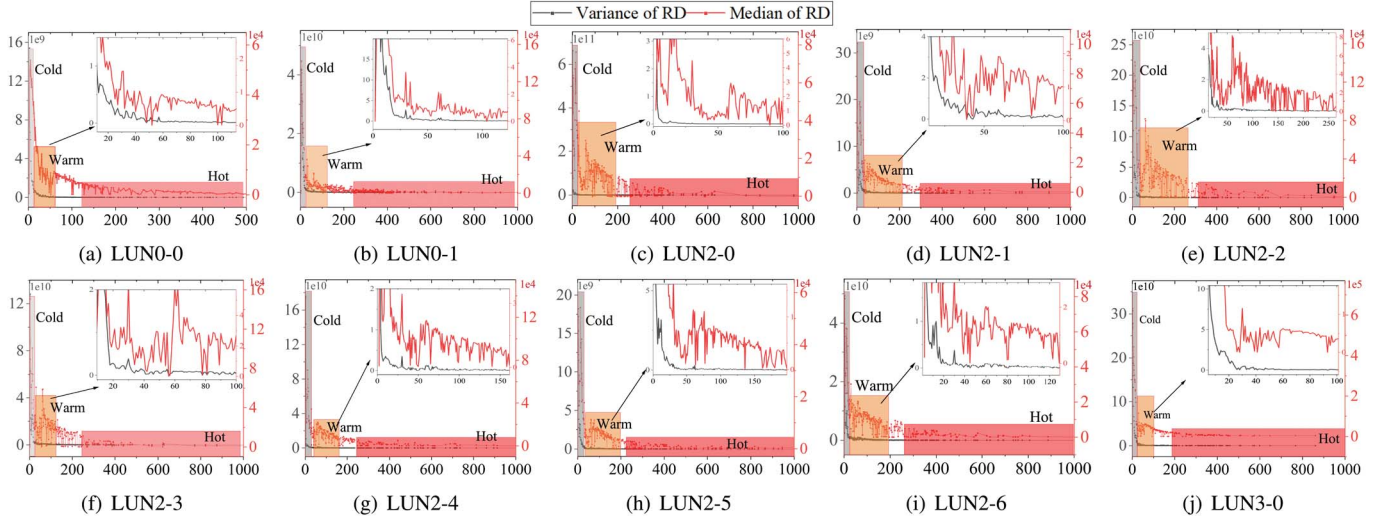


Fig. 2. The analysis of rewrite distance (RD) in workloads. X-axis represents the I/Os access frequency of requests. Left y-axis and right y-axis denote the variance and median of RD of requests with the same number of I/Os access frequencies, respectively.

TABLE II
THE BUSINESS OF A DIE IN A CHIP INSIDE AN SSD

Trace	LUN0-0	LUN0-1	LUN2-0	LUN2-1	LUN2-2
ChipBusy DieFree	0.91736	0.93558	0.93838	0.91684	0.95667
ChipBusy DieBusy	0.08264	0.04735	0.06162	0.08316	0.04333
Trace	LUN2-3	LUN2-4	LUN2-5	LUN2-6	LUN3-0
ChipBusy DieFree	0.93328	0.90093	0.95081	0.95667	0.95265
ChipBusy DieBusy	0.06672	0.09907	0.04919	0.04333	0.06442

to clean one. The amount of dirty data becomes large in the cache. Consequently, significant overhead is incurred in the cache replacement. In an SSD, the die is the smallest unit of execution. If the chip is busy, the die in the chip is still idle, which affects the amount of active write-back data and the dies' utilization. We define *ChipBusyDieBusy* as all dies being busy when the chip is busy, and *ChipBusyDieFree* as there being one free die when the chip is busy. Our study, examining a 32-chip and 64-die SSD configuration (see Section V), reveals that die idle time when the chip is busy occupies about 90% (see Table II). This ratio increases with the number of dies in a chip, wasting resources in flash memory.

Fig. 3 shows the CDF of the number of chips and dies in parallel when a new request arrives. When the chip-level status is disclosed, a chip allows only one die to perform the active write-back, which can significantly reduce dies' utilization. For example, in LUN3-0, with the chip-level detection, the number of chips in parallelism less than 9 occupies 80%. With the die-level detection, the number of parallelisms increases to 14 at the same probability, increasing the utilization of die to a larger extent – and enhancing the active write-back throughput. Then, more clean data are originated to shorten response time.

Observation 4: When a chip is busy, the dies in the chip have a high probability of being idle. There are a large number of idle

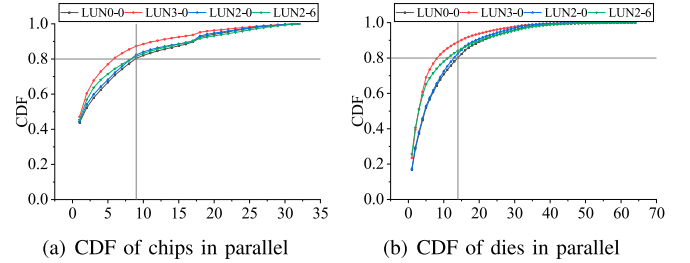


Fig. 3. CDF between chip- and die-level utilization.

dies in flash memory whenever a request arrives. There are multiple dies in a chip. With a chip-level monitoring scheme, if one die is busy, the other dies cannot be fully utilized. However, a die-level load monitor (see Section IV-D for detail) can discover idle dies with finer granularity, increasing dies' utilization.

Summary. When data are replaced in an aggressive manner, it can quickly free up space at the cost of serious WA issues (Observations 2 and 3). Moreover, frequent replacement of data at high I/O intensity leads to tail latency due to access conflict between user I/Os (Observation 1). Replacing small-sized data fails to fully utilize the parallel resources (dies) of flash memory (Observation 4). We are motivated by the aforementioned observations to propose LAC optimizing the amount of actively write-back data when I/O intensity is low and dies are idle. LAC incorporates the data protection method to control the amount of data, thereby reducing WA.

IV. DESIGN OF LAC

A. System Overview

Fig. 4 sketches the LAC system architecture that embraces five modules - hot/cold separation, data protection, I/O-intensity-aware, die-level load monitor, and parallel-write modules. In Section IV-B, hot/cold separation classifies incoming page-sized requests into cold, warm, and hot data. The hot

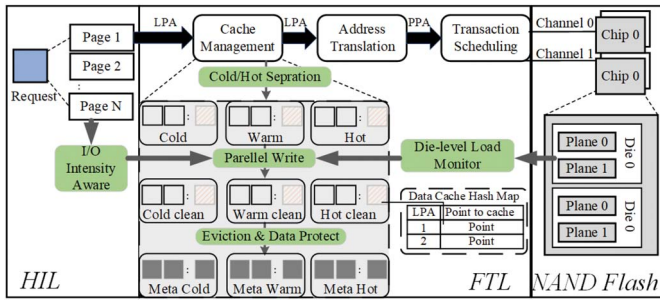


Fig. 4. System overview of LAC.

TABLE III
NOTATIONS AND DESCRIPTION USED IN THIS PAPER

$W_{intensity}$	The amount of data per unit time
S_{page}	A page size in sectors
T_{avg}	Average interval of all requests in a second
N_{req}	The number of page requests
I_x	The selected item in the linked-list
$N_{curPage}$	The number of pages currently requested
$N_{channel}$	The number of channels of an SSD
N_{page}	Average number of pages in a second
$N_{parallel}$	The number of channel parallel operations
$R_{parallel}$	The LPA region size for a parallel write
P_{degree}	The number of channels utilized in parallel write
$N_{writeback}$	The number of write-back operations
L_{Die}	The load of a die
L_{extern}	The waiting time of requests on the chip queue
L_{excute}	The remaining execution time
T_r, T_w, T_e	The wait time on the chip queue for read, write, and erase
$C_{user}, C_{GC}, C_{Mapping}$	The number of user read/write, GC, and mapping
L_r, L_w, L_e	Execution time of a read, write, erase
F_r, F_w, F_e	The flag for a read, write, and erase
T_{cur}	The current system time

data-protection method protects hot data from being mistakenly evicted from the cache, thereby avoiding multiple active write backs that deteriorate WA. In Section IV-C, the I/O-intensity-aware method detects the magnitude of the I/O intensity from the host and the size of a request interval through the sliding window. The die-level load monitor module monitors the status - idle or busy - of the flash memory, see Section IV-D. In Section IV-E, active write-backs are triggered when the I/O intensity is low and the die is idle. To originate clean data, the parallel-write module uses channel-level parallelism to write cached cold and dirty data to the flash memory in parallel when the die is idle. The parallel write-back process is sparked based on die-level load monitor and I/O-intensity-aware modules. These page requests with adjacent logical addresses are written back into flash memory. This method can enhance the throughput and convert much dirty data into clean data, minimizing the cost of cache miss. Table III lists the notations and description used in this paper.

B. Hot/Cold Management

Inspired by observations 2 and 3, we propose a hot-and-cold management scheme that integrates cold-hot separation and data protection methods. In LAC, cold data is actively

written back into the flash memory because there will be no access to the cold data in the future. The cold data is actively written back in advance to axe WA, whereas writing back hot data that is accessed again escalates WA. Thus, we design an accurate hot-and-cold separation module that distinguishes hot and cold data amid dynamically changing I/O access patterns, thereby bolstering the accuracy of cold-hot data identification. When replacing cache data, the data protection method of LAC avoids active write-backs of hot data, with a significant reduction of write overhead caused by multiple repeated write-backs, thereby reducing WA. As described in Section III, the cache space is divided into cold, warm, and hot regions. An LRU-linked list manages the cache space, and each real region is partitioned into the clean- and dirty-linked lists. Replaced pages are stored in the meta region based on metadata, which contains the addresses of page requests.

We demonstrate cold, warm, and hot data from perspectives of classification, data movement, and data protection below.

1) **Cold Data: Conditions for Cold Classification:** Cold pages are those that are infrequently accessed or have not been accessed for a considerable amount of time. They are likely to have a low access frequency F and a long rewrite distance RD . In this paper, the term accesses includes read and write accesses, which register the total number of reads and writes handled by the LPA. The rewrite distance's accesses include writes; otherwise, read and write accesses refer to the number of reads and writes, respectively.

Moving to Cold Region: Pages are classified as cold data when (1) they are first visited or (2) moved from the warm region when the cold region is empty.

Data Protection: The cold region stores first-access pages. C denotes the cache capacity; $F \times C$ represents the maximum value at which a page request can survive. If the lifetime after the last access exceed this maximum value, it signifies that the request, which has been kept in the cache for a long period, should be expelled to mitigate cache pollution. Hot data is protected by moving it to the head of the current linked list and halving the number of times it is accessed.

2) **Warm Data: Conditions for warm classification:** Warm pages are those that are accessed less frequently than hot pages but more frequently than cold ones. Warm pages have an intermediate access frequency and large rewrite distance.

Moving to Warm Region: A page may transition to the warm category from cold under the conditions: (1) its access frequency increases but is insufficient to be classified as hot data, (2) cache hits in the meta-cold region, or (3) data from a hot region is moved to the warm region as it is empty.

Data Protection: Access frequency F in the warm region is small – and functions in conjunction with rewrite distance RD . $F \times RD$ denotes the time data are kept in the cache. If the lifetime after the last access is less than this value, the page request ought to be protected.

3) **Hot Data: Conditions for Hot Classification:** Hot pages are accessed frequently or has been accessed recently. Specific metrics for this classification include a high access frequency, short rewrite distance, or both.

Moving to Hot Region: A page transitions to the hot region from the warm region a cache hit occurs in the meta-warm or meta-hot region.

Data Protection: The data in the hot region is frequently accessed with a stable rewrite distance. If Age is less than the median rewrite distance of all requests RD_{median} , the request – being hotter than a majority of its peers, is a promising candidate for the cache.

Notably, the data protection method in the above three cases exhibits two features. First, it improves SSD lifetime by maximizing the retention of hot data in the cache, increasing hit rate, and reducing the number of flash writes. Second, it avoids extra writes incurred by multiple flushes of hot data back to the flash memory – and access conflict between user I/Os is alleviated to lower response time.

4) *Cache Replacement in LAC:* If a flash transaction hits the real region, the page request should be relocated to the higher-level region. If it hits the meta-cold region, most cold data is accessed. Then, high priority is given to replacing pages from the real warm region. Hit meta-warm region indicates that the current request involves infrequently accessed data. Therefore, a preference should be given to replacing a page from a linked list in the real hot region. If it hits the meta-hot region, high priority is given to replacing pages from the linked list in the real-cold region because more hot data is accessed. The data protection method traverses the linked list to locate data that is inappropriate for storage in the real cold region. Subsequently, the data is evicted from the cache. If any region has a cache miss, cache replacement will be provoked. Cache replacement occurs in the real cold region.

5) *Time to Transition From Hot to Cold:* The time for a page transition from hot to cold is not fixed; it is dynamically determined by access patterns and the data protection algorithm. If a page in the hot region has not yet been accessed for a while, the page's access frequency and rewrite distance will change, potentially leading to its cache replacement by the protection algorithm if the page is deemed cold data.

C. I/O-Intensity-Aware Module

The I/O intensity of Systor'17 traces is high. If the active write-back time is not well orchestrated under short request intervals, the response time of subsequent requests increases. Thus, we defined the interval write-back threshold (S_w) as

$$S_w = \frac{T_{avg} * N_{curPage} * S_{page} * N_{channel}}{N_{page} * L_w}. \quad (1)$$

This threshold is the maximum amount of data written back during the time interval between two consecutive requests. Active write-back data volume beyond this threshold prevents subsequent requests from being served. Below this interval, on the other hand, dirty data cannot be swiftly transformed at the maximum. The threshold is adjusted based on the request interval, and the request interval measure is inversely proportional to the intensity.

Eq. 1 unveils the relationship between the I/O access pattern and flash write latency. The number of active write-backs is

estimated using the I/O intensity-aware method. T_{avg}/N_{page} is an interval between individual page requests, determining the number of active write-backs within the interval. $T_{avg}/N_{page}/L_w * N_{channel}$ is the maximum number of write-backs within an interval. The amount of written data accumulates in the active back-pressure buffer. When the SSD controller receives a new request, the request is appended to the back-pressure buffer queue. When the number of I/O requests in the buffer exceeds its maximum capacity, the buffer will pause receiving new requests while processing the buffered requests. When the value of the active back-pressure buffer is less than S_w , an active write-back will be possible. The active back-pressure buffer gradually decreases as flash transactions are executed. Subsequently, new active write-back requests are admitted.

Based on the size of the historical request intervals, we propose a time-smoothing approach with a one-second window to predict the request intervals and the number of pages within the subsequent one-second period. In this prediction, the time window moves on the time series in one-second steps, and we obtain five consecutive one-second time windows before the current time. After each movement step, we calculate the average sizes of the request interval within five windows. Then, we employ this average value as a prediction value of the request interval in the next one-second period. The predicted request interval is applied in S_w (see Eq. 1) to determine the maximal amount of write-back data after a request is completed. T_i is an interval time between requests i and $i + 1$ in a second; n is the number of requests in the second. The average interval of all requests in the second is expressed as

$$T_{avg} = \sqrt[n-1]{\prod_{i=1}^{n-1} T_i}. \quad (2)$$

The average number of pages of all requests in a second is

$$N_{page} = \frac{\sqrt[n]{\prod_{i=1}^n S_i}}{S_{page}}, \quad (3)$$

where S_i denotes the size of sectors in request i . The time-smoothing approach bolsters the calculation accuracy by smoothing data and reduces the interference of outliers in request intervals.

The reasons behind employing one second as the time window are elaborated as follows. In the extensive experiments, we reckon that the I/O intensity in the workload varies over a short period. If the time window size is too small – for example, less than 1 ms, the predicted result might be inaccurate due to random short-term fluctuations. When the window size is larger than one second, the window cannot capture the true size of request intervals: the time window is too large and the difference in the average value of request intervals per unit time is small. A time window between 1 ms and 1 s can well capture short-term variations; thus, we choose 1s to achieve high accuracy and minimal computational overhead. In Section VI-B1, we tested the performance impact of I/O intensity predictions based on a one-second time window – and the module can indeed significantly cut back response time.

The above strategy combines host I/O intensity with flash write latency. The interval write-back threshold becomes low under high I/O intensity, mitigating access conflict between user I/Os. In contrast, the interval write-back threshold hikes given low I/O intensity – and SSD performance is optimized.

D. Die-Level Load Monitor

We propose a die-level load monitor to detect busy flash memory, the load L_{Die} of which is denoted as

$$L_{\text{Die}} = L_{\text{extern}} + L_{\text{execute}}. \quad (4)$$

where L_{extern} is the waiting time of requests on chip queue. L_{execute} is the remaining execution time of the current die. L_{Die} being zero signifies an idle die. Notably, L_{extern} is given as

$$L_{\text{extern}} = T_r + T_w + T_e, \quad (5)$$

which includes read time T_r , write time T_w , and erase time T_e . C_{user} , C_{GC} , and C_{Mapping} denote the number of user read/write, garbage collection, and mapping operations, respectively. Read (write) operations have three types: user read (write), GC read (write), and mapped read (write). Erase only originate from GC. Symbols L_r , L_w , and L_e denote time of a basic read, write, and erase, respectively, in flash memory. Thus, die-level wait time of read, write, and erase requests is gauged as $T_r = (C_{\text{user}} + C_{\text{GC}} + C_{\text{Mapping}}) \cdot L_r$, $T_w = (C_{\text{user}} + C_{\text{GC}} + C_{\text{Mapping}}) \cdot L_w$, and $T_e = C_{\text{GC}} \cdot L_e$, respectively.

The remaining execution time on the die can be given as

$$L_{\text{execute}} = F_w \cdot L_w + F_r \cdot L_r + F_e \cdot L_e - T_{\text{cur}}, \quad (6)$$

which is the expected completion time of a basic operation being executed in the die minus the current system time. T_{cur} is the current system time. F_w , F_r , and F_e are the Boole value ('0' or '1') for a basic operation. Active write back is sparked only when the die-level load is '0'. This approach has a finer granularity than that of chip-level load monitoring, discovering more idle target dies in time – and improving utilization of dies by increasing the amount of write-back data.

E. Parallel-Write Mode

To write as much data as possible in a limited time, LAC incorporates a multi-channel parallel write-back method, where parallel active write-backs fully exploit the spatial locality of the workload in conjunction with the parallelism of flash memory. Page requests with close spatial locality are written back in parallel through channel-level parallelism, increasing throughput. The HIL module in an SSD slices requests into page-sized requests with increasing logical addresses, high spatial locality, and similar hotness. Writing these requests back amid an active write-back increases the write data volume during low hotness.

Parallel Region. A static page allocation assigns logically adjacent pages to different channels. The number of channel-parallel operations (N_{parallel}) is given as $N_{\text{parallel}} = \frac{T_{\text{avg}}}{N_{\text{page}} \cdot L_w}$, which depends on the request interval and flash memory parameters. Given a fixed write latency, a large request interval leads to high parallelism degree. The LPA region size for a

parallel active write-back (R_{parallel}) is derived from the number of parallel operations as ($R_{\text{parallel}} = N_{\text{parallel}} \cdot N_{\text{channel}}$).

A parallel write only selects pages within the address region, and pages beyond this region compete for parallel resources while suffering from low spatial locality due to wide discrepancies in LPAs. This problem results in misjudgment of data elected for active write-backs. Notably, a large-sized request is split into multiple page-sized requests having adjacent LPAs and similar hotness, which is prioritized in the active write-back. If a large-sized request does not exist, we search pages with adjacent or close LPAs – and write them back to flash memory in parallel. The number of channels utilized for parallel write-backs depends on the workloads. Parallel active write-backs cause high WA, which is more pronounced in face of writing back hot data. Frequent updates of hot data incur multiple write-backs that downgrade SSD lifetime. We integrate parallel write-backs with the data protection method: only cold data are written back in parallel.

Parallel-write Procedure. Algorithm 1 presents the parallel-write process, in which LAC nominates cache item I_x from the tail of the dirty linked list to warrant an active write back. The conditions under which the active write back is invoked include: 1) Dirty data exist in the cache, and the number of clean pages is fewer than the preset write-back threshold. Dirty data may trigger an active write back. The write-back threshold is set to increase the number of clean pages without many write-backs (Lines 4–6). 2) The die-level load is zero. In this case, the active write-back pages are served in time – preventing a request from awaiting while reducing access conflict between flash transactions. The algorithm calculates I_x 's channel, chip, and die number under the static allocation strategy using the LPA (see Lines 7–13). The die-level load monitor detects whether a physical address is free. The active write-back procedure moves the page request from the dirty linked list to the clean list, followed by delivering the write transaction. Then, the parallel write-back module writes back the other requests with adjacent LPAs based on the spatial locality, using the flash bandwidth to enhance throughput. 3) The proactive back-pressure buffer is less than S_w , which is limited by I/O intensity – and the write-back data volume is expanded when the intensity is low. Increasing the write-back data volume under hefty intensity maximizes the flash bandwidth and alleviates access conflict between user I/Os (see Line 23). Lines 16–22 articulate the parallel active write-back process under two conditions. First, when an active write back is initiated, a parallel region is generated based on the current LPA. The request that satisfies the region is located. Therefore, the spatial locality is fully utilized to increase write-back throughput. Second, only data that do not trigger the data protection procedure are written back in parallel because writing back hot data shortens the SSD lifetime.

V. EXPERIMENTAL SETUP

Settings. We implement LAC on an open-source NVMe SSD simulator, MQSim [14]. We configure cache sizes from 8 MB to 1024 MB. Table IV lists parameters of MQSim. We employ the realistic enterprise-scale Systor'17 traces [8], [22] collected

Algorithm 1 Parallel-Write Mode

Require: Flash Transaction $T1, T2, T3, \dots, Ti, \dots$

```

1: procedure ACTIVEPARALLELWRITEBACK
2:   repeat
3:     Get the cache entry from LRU in dirty_list:  $I_x$ 
4:     if  $N_{dirty} == 0 \parallel$  Write-back threshold  $\leq N_{clean}$ 
5:       return & Exit
6:     end if
7:     Get the LPA( $I_x$ )
8:      $channelID \leftarrow LPA \bmod N_{channel}$ 
9:      $chipID \leftarrow (\lfloor LPA / channelN \rfloor) \bmod N_{chip}$ 
10:     $dieID \leftarrow (\lfloor (LPA / channelN) / chipN \rfloor) \bmod N_{die}$ 
11:    if The Die( $channelID, chipID, dieID$ ) is in Busy
12:      then
13:        return
14:      end if
15:      ACTIVEWRITEBACKTONANDFLASHDIE( $I_x$ )
16:      MOVETOList( $LPA, clean\_list$ )
17:      while  $I_x.next.LPA - I_x.LPA \leq N_{channel} \times N_{parallel}$  do
18:        if ! HOTPROTECT( $LPA$ ) then
19:          ACTIVEPARALLELWRITEBACK
20:           $I_x.next \leftarrow I_x.next.next$ 
21:        end if
22:      end while
23:       $I_x \leftarrow I_x.next$ 
24:    until Proactive_back_pressure  $\leq S_w$ 
25:  end procedure

```

TABLE IV
SIMULATION PARAMETERS IN MQSIM

SSD Organization	Host Interface: PCIe 3.0 (NVMe 1.2) Capacity: 512 GB, 8 channels, 4 chips/channel
Flash Interface	ONFI 3.1 (NVDDR2)
Flash Structural	256 pages/block, 2048 blocks/plane 2 planes/die, 2 dies/chip, 8 KB/page
Flash Latency	Read: 25 μ s (S), 50 μ s (M), 75 μ s (T), 100 μ s (Q) Program: 0.3 ms (S), 0.75 ms (M), 1.3 ms (T), 2 ms (Q) Erase: 2 ms (S), 3 ms (M), 4 ms (T), 5 ms (Q)
FTL	GC Policy: Greedy [19], GC Threshold: 0.05 Mapping: DFTL [20], TSU: Sprinkler [21] Over provisioning Ratio: 0.07

*Symbols S, M, T, and Q means SLC, MLC, TLC, and QLC flash memory, respectively.

from the Fujitsu Laboratories in Japan to evaluate the performance of the caching scheme. Systor'17 traces have higher I/Os per second and higher I/O intensity than those in prior work [23], [24] MQSim performs a preconditioning step prior to simulation, where it processes an input trace to extract the LPA access patterns of application I/O requests. The extracted information is used to feed a valid/invalid page distribution model [25], which preconditions the model for SSD use by marking physical pages as valid or invalid. Similar to [5], [26], the preconditioning ratio is set to 80%. Extensive experiments



Fig. 5. Normalized response time.

are performed to validate LAC under workloads with write- and read-intensive scenarios. We compare LAC with LRU, CFLRU [12], GCaR-LRU [27], VS-batch [11], MQSim [14], and Co-Active [3].

Tested metrics. *Average response time* represents the time interval between an I/O request entering the SQ and the host receiving feedback from the SSD. We normalize the performance metrics to investigate the performance of the caching schemes. *Tail latency* is the longest latency in a set of requests, and it reveals the system's performance in the worst-case scenario. The tail latency in this paper can help us improve system performance under workloads with high I/O intensity. *Write amplification (WA)* is the ratio of the data volume written to flash memory to that issued by a host. WA unveils the impact of the cache scheme on SSD lifetime. *Erase count* in the flash memory quantifies the SSD lifetime. *Write/read hit ratio* is the ratio at which an I/O request hits the cache; a high hit ratio reduces the response time.

VI. PERFORMANCE EVALUATION

We conducted basic experiments (Section VI-A) and extended experiments (Section VI-B) to evaluate the performance of LAC.

A. Basic Experiments

1) *Response Time and Tail Latency:* **Response Time.** In Fig. 5, LAC outperforms other caching schemes under various cache sizes and workloads. Given a 32-MB cache, LAC reduces the response time by averages of 50.6%, 52.1%, 48.4%, 33.2%, 42.4%, and 8.29%, compared with LRU, CFLRU, GCaR-LRU, MQSim, VS-Batch, and Co-Active, respectively. The 256-MB cache-based LAC shortens the response time by 76.8%, 76.0%, 74.1%, 69.3%, 66.1%, and 16.2% on average. Under the read-intensive LUN3-0, LAC optimizes the first three cache schemes

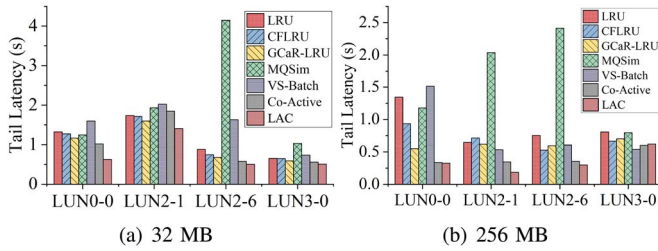


Fig. 6. Tail latency of caching schemes with 32- and 256-MB cache.

by averages of 83.4%, 82.8%, and 83.0% because LAC addresses the flash read/write performance imbalance while processing writes during idle time. The response time of the most cache schemes decreases as the cache size increases, and LAC delivers an outstanding optimization under a large-sized cache.

MQSim uses an improved write-through method to write back the cache cold data to flash memory followed by the deletion of cache pages. This technique warrants a low response time under low-I/O-intensity workloads; however, it increases the response latency owing to access conflict between user I/Os caused by a large number of flash writes under workloads with high I/O intensity. LAC has a performance edge under the majority of workloads. LAC slashes the response time of Co-Active by up to 42.3%. Co-Active has no limited data volume actively written back into flash memory, causing access conflict among consecutive requests. LAC detects the low I/O-intensity duration – and maximizes the data volume written back in the shortest time through the parallel write-back mode, lessening access conflict and maximizing the amount of dirty data converted into clean ones. Thus, the replacement cost is suppressed amid cache misses with reduced response time. Under a few workloads, the response time of LAC is marginally smaller than that of Co-Active because LAC aims to reduce the number of active writes to prevent excessive writes from impairing the SSD lifetime.

Tail Latency. In Fig. 6, LAC mostly exhibits lower tail latency under all the workloads. With a 32-MB cache, LAC’s tail latency is dropped by averages of 33.9%, 30.5%, 24.4%, 53.8%, 47.8%, and 20.8% compared to LRU, CFLRU, GCaR-LRU, MQSim, VS-Batch, and Co-Active, respectively. In case of a 256-MB cache, LAC shortens tail latency by averages of 57.7%, 47.4%, 43.0%, 68.2%, 44.8%, and 15.4% compared to the other schemes. Notably, the reduction rate of tail latency is high in write-intensive workloads – and the rate is slightly low in read-intensive workloads. LAC writes back dirty data to flash memory to originate clean data as I/O intensity is low, optimizing the replacement latency of cache misses. Thus, tail latency becomes low under the high-I/O-intensity workloads.

2) **SSD Lifetime:** SSD lifetime, affected by WA and erase counts, is one of important metrics for validating the LAC. In Fig. 7, with a 32-MB cache, LAC reduces WA by 34.3%, 30.8%, 33.0%, 43.3%, 3.7%, and 14.8% on average compared with LRU, CFLRU, GCaR-LRU, MQSim, VS-Batch, and Co-Active, respectively. Given a 256-MB cache, LAC further curtails WA and erase counts. LAC’s optimization performance becomes more prominent in case of a large-sized cache. Fig. 8

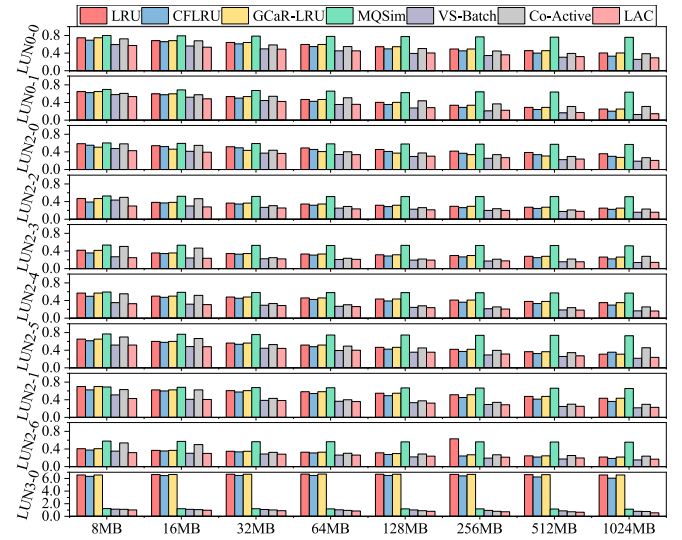


Fig. 7. Write amplification.

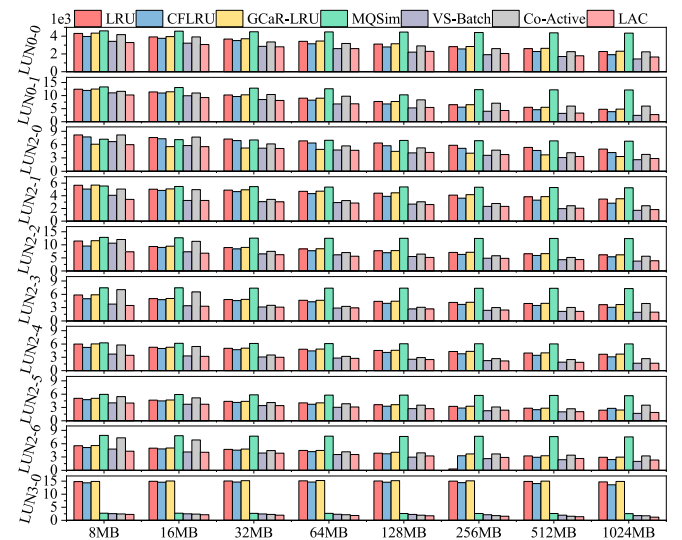


Fig. 8. Normalized erase count.

displays LAC’s erase counts which are 34.4%, 31.0%, 32.1%, 42.2%, 3.6%, and 15.2% smaller on average than those of LRU, CFLRU, GCaR-LRU, MQSim, VS-Batch, and Co-Active, respectively, under a 32-MB cache. When it comes a 256-MB cache, LAC outperforms LRU, CFLRU, GCaR-LRU, MQSim, and Co-Active in erase count by averages of 42.1%, 32.3%, 37.5%, 57.2%, 4.3%, and 20.8%, respectively. In the majority of cases, the erase count decreased as the cache size increased. Importantly, LAC exhibits the lowest erase count with a large-sized cache. In LAC, we design an accurate hot-and-cold separation module that distinguishes hot and cold data during dynamically changing I/O access patterns. The data protection method of LAC avoids active write backs of hot data, with a significant reduction of write overhead caused by multiple repeated write backs. MQSim has a high WA and

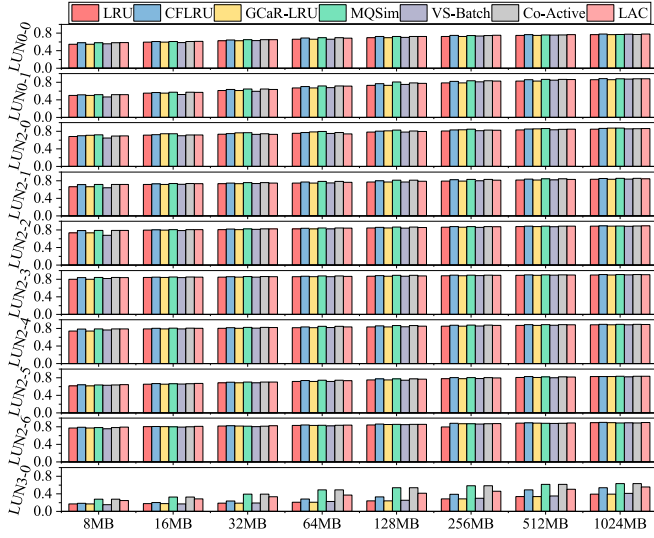


Fig. 9. Write hit ratio.

erase count because of the unrestricted write-back data volume, which severely diminishes SSD lifetime.

3) *Hit Ratio*: In Fig. 9, LAC curbs the write-hit rates of LRU, CFLRU, GCaR-LRU, MQSim, VS-Batch, and Co-Active by averages of 9.6%, 4.0%, 9.2%, -2.3%, 9.4%, and -1.9%, respectively, under a 32-MB cache. With a 256-MB cache, LAC trims the hit rates of LRU, CFLRU, GCaR, LRU, MQSim, and Co-Active by 8.9%, 1.3%, 7.7%, -2.9%, 6.8%, and -2.6% on average, respectively. The write-hit ratio of the schemes rises with an increasing cache size. MQSim and Co-Active present low read-hit ratios owing to the write cache mode, in which read requests bypass cache to enhance the write-hit ratio. Facing write-intensive workloads, the write-hit ratios are extremely close to each other because the majority of the rewrite distances are smaller than the cache size. Under the read-intensive workload LUN3-0, the write-hit ratio significantly goes up as the cache size increases because a small number of writes are more likely to remain in the cache as the cache size expands. Under write-intensive workloads (LUN0-0 and LUN2-0), the write-hit rate converges with an increasing cache capacity.

In Fig. 10, LAC improves the read-hit rate by averages of -14.7%, -12.6%, -14.5%, 51.3 $\times$, -3.8%, and 53.1 $\times$ compared with LRU, CFLRU, GCaR-LRU, MQSim, VS-Batch, and Co-Active based on a 32-MB cache. LAC optimizes, in the 256-MB cache case, LRU, CFLRU, GCaR-LRU, MQSim, VS-Batch, and Co-Active by averages of 2.2%, 3.9%, -6.7%, 25.2 $\times$, 8.1%, and 25.4 $\times$, respectively. Write-intensive workloads include few reads, and the alternative caching schemes share similar read hits because these schemes are extended from LRU. MQSim and Co-Active have a low read ratio due to the bypassing of reads from the cache. Under read-intensive workloads, the caching schemes that prioritize writes present marginally lower hit ratios than the peers.

B. Extended Evaluation

1) *Impact of I/O-Intensity-Aware Module*: To study the impact of the I/O intensity-aware module on the LAC



Fig. 10. Read hit ratio.

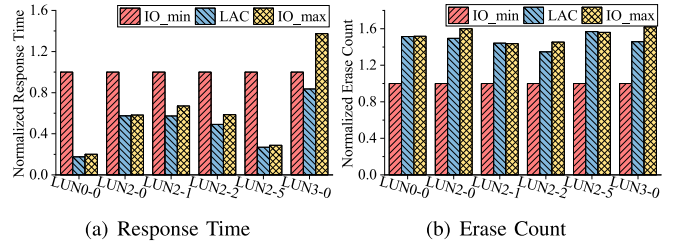


Fig. 11. Impact of I/O-intensity-aware module on performance.

performance, we set up three sets of experiments. Symbol IO_{min} indicates that S_w is set to “0”, meaning that no occurred active write back. IO_{max} entails that S_w is set to the maximum value – the maximum depth of the back-pressure buffer. In Fig. 11, LAC trims the average response time of IO_{min} and IO_{max} by 51.4% and 15.1%, respectively, with a reduction of the maximum response time by 82.5% and 39.1%, respectively. IO_{min} cuts down LAC’s erase count by 31.8% – and IO_{max} increases LAC’s WA by 4.2%. IO_{min} -based LAC enjoys the smallest erase count; however, its response time is longer than that of LAC due to the lack of active write backs that fail in fully utilizing idle intervals. The amount of data written back in the intervals is unrestricted – causing access conflict that expand response times with high WA. LAC adjusts S_w between zero and the maximum depth of the back-pressure buffer by detecting any change in I/O intensity. LAC effectively mitigates the access conflict between user I/Os amid high I/O intensity – and the flash memory’s utilization maximized to curb response time.

2) *Parallel-Write Mode*: Fig. 12 plots the impacts of parallel write backs on the response time and lifetime. *Parallel_off* and *Parallel_on* represents the parallel write-back module is turn off and on scenarios. Under the six workloads, parallel write-back shortens the response time by 39.1% and the maximum erase count by 10.0%. The parallel write-back module determines

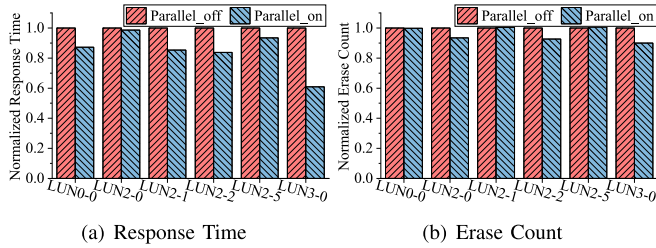


Fig. 12. Impact of parallel-write module on performance.

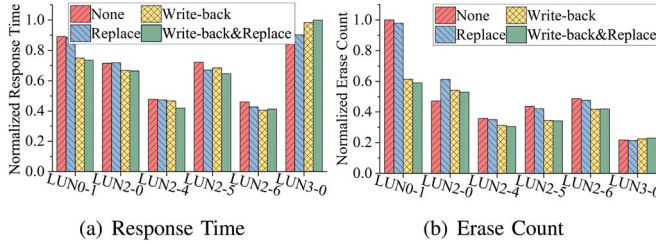


Fig. 13. Impact of data protection module on performance.

the maximum number of parallel write-backs according to I/O intensity. Then, the adjacent cache pages are simultaneously written back to the flash memory anchored on the spatial locality and high sequential write performance of the flash memory, alleviating access conflict and lowering response time. The traditional cache write-back scheme replaces data in the linked list upon a full cache. Parallel active write-back changes the order of conventional write-back based on the LRU – and cold data that have a close spatial locality are written back to suppress erase count.

3) *Data Protection*: We validate the data protection module in the context of four caching schemes: (1) turning off data protection (None), (2) triggering data protection in the cache replacement (Replace), (3) enabling data protection in the cache replacement and write-back (Write-back & Replace), and (4) kicking off data protection in the write-back (Write-back). Fig. 13 shows that for the above three schemes under the write-intensive workloads, the Write-back & Replace case reduces the response time by up to 21.2%, 20.3%, and 6.5%. the highest declines in erase count for the above three are 69.1%, 65.6%, and 3.9%. The results confirm that the three data protection methods trim response time and erase count. The Replace case reduces these metrics the least whereas the Write-back & Replace case achieves the most optimization because most dirty data are converted into clean data by active write backs. Write-back & Replace, which can refine hot data, is marginally improved compared to Write-back. Under a read-intensive workload, the erase counts of the four schemes are similar: the response time of Write-back & Replace is marginally higher than those of its peers because the write-back feature cannot be fully exploited in LAC under read-intensive conditions.

4) *Die-Level Load Monitor*: We study the impact of the die-level load monitor module on the response time and erase count. Symbols *CHIP-LAC* and *DIE-LAC* in Fig. 14 represent the

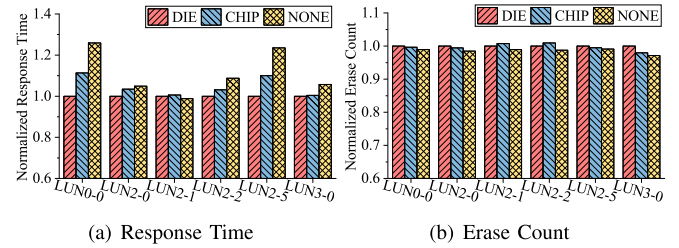


Fig. 14. Impact of die-level load monitor module on performance.

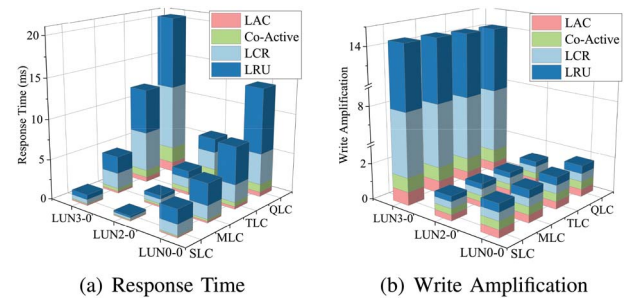


Fig. 15. Impact of flash memory cell on performance.

chip- and die-level load monitor methods in LAC, respectively. *NONE* means that the access conflict between internal I/Os in flash memory and user I/Os is neglected in the write-back. *DIE-LAC* slashes the response time of *CHIP-* and *NONE-LAC* by up to 11.3% and 20.6% but these schemes share similar erase counts. Compared with *CHIP-LAC*, *DIE-LAC* shortens latency by mitigating access conflict between flash transactions in the active write-backs; however, *DIE-LAC* pushes up the number of pages converted to clean data and erase count. Compared with *CHIP-LAC*, the *DIE-LAC* has limited performance improvement because the static allocation strategy in LAC handles the cold-and-hot access patterns, and most requests concentrate on a small portion of LPAs, which incurs infrequent address mappings in some idle dies.

5) *Impact of Flash Memory Cell*: We study the impact of write latency in flash memory on LAC performance in the I/O intensity-aware module. We deploy four flash memory cells - SLC, MLC, TLC, and QLC [28], see Table IV. LAC is a winner among its peers – and the response time of Co-Active, LCR, LRU, and LAC climbs when the write latency enlarges. In the SLC case, LAC cuts down the response time of Co-Active, LCR, and LRU by -6.7%, 75.2%, and 77.8%; LAC reduces WA by 5.4%, 35.4%, and 28.9%. With respect to MLC, LAC slashes the response time of Co-Active, LCR, and LRU by 24.3%, 80.4%, and 79.4% with a WA reduction of 9.4%, 37.6%, and 31.5%. TLC-based LAC shortens the response time by 39.6%, 82.5%, and 82.7%, compared against Co-Active, LCR, and LRU, in which WA is optimized by 18.4%, 42.3%, and 37.0%. Similarly, QLC-based LAC curtails the response time by 46.4%, 81.8%, and 82.4% with the WA reduction of 22.1%, 43.8%, and 38.5%, respectively. LAC's I/O intensity-aware module, adjusting S_w in accordance to I/O intensity and flash latency,

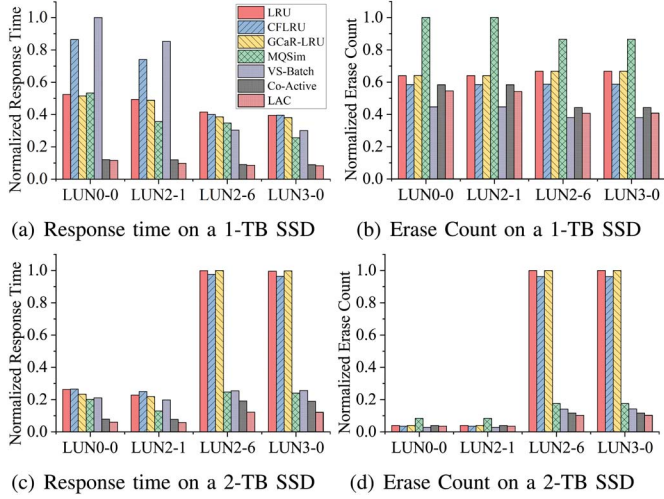


Fig. 16. Impact of large-capability SSDs.

determines an appropriate write-back data volume to mitigate access conflict between user I/Os and optimize response time.

6) *Impact of Large-Capability SSDs*: We investigate the performance of LAC in MQsim configured with 1 TB and 2 TB capacity, see Fig. 16. Due to the large-sized SSD, we preconditions a stable-status SSD to generate garbage collection. Compared with LRU, CFLRU, GCaR-LRU, VS-Batch, MQSim, and Co-Active, 1-TB-SSD LAC achieves lower response time than by 79.0%, 82.7%, 78.3%, 73.4%, 80.1%, and 8.4%, respectively. The erase count is slashed by 27.0%, 18.8%, 26.9%, 49.2%, -14.6%, and 7.3%, respectively. On the 2-TB SSD, LAC reduces the response time by 81.6%, 82.0%, 80.6%, 56.1%, 61.4%, and 29.7%. Furthermore, LAC cuts back the erase count by 49.6%, 44.5%, 49.5%, 49.0%, 0.3%, and 11.1%. The performance of LAC increases when the capacity is scaled up because LAC can find low I/O-intensity write-back time dramatically, avoiding request access conflict and – generating a large amount of clean data. Thus, LAC streamlines the cache replacement process, reducing replacement latency. On the 1-TB SSD, VS-batch has a lower erase count thanks to its unified temporal localization, batch processing of requests, and reduced WA. LAC and VS-batch share a similar result on the 2-TB SSD because of the inverse relationship between SSD capacity and erase count.

7) *Performance of LAC Under Other Workloads*: We undertake empirical studies on LAC under workloads from Microsoft Research’s Cambridge traces [23] and the UMass Trace Repository [24]. Fig. 17 shows that LAC shortens the response time for LRU, CFLRU, GCaR-LRU, VS-Batch, MQSim, and Co-Active by an average of 11.5%, 34.5%, 10.6%, 11.2%, 3.3%, and 1.4%, respectively. The request intervals in these workloads are more than a hundred times larger compared with that of Systor’17, which has larger request intervals and fewer access conflicts. The advantage of LAC to locate time for active write backs cannot be fully utilized under low-I/O intensity workloads. Moreover, LAC reduces erase count by 1.3%, 1.5%, 2.6%, and -8.1%, respectively, compared to the passive write-back

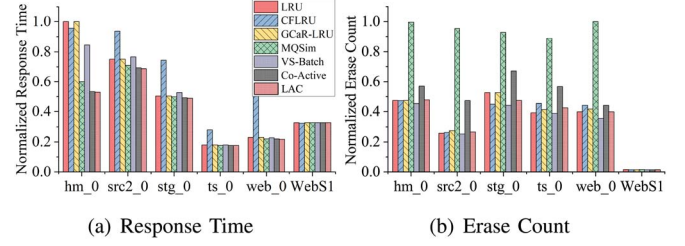


Fig. 17. Performance of LAC under other workloads.

TABLE V
MIXED WORKLOADS USED IN THIS TEST

mix1	LUN0-0, LUN0-1	mix2	LUN2-0, LUN3-0
mix3	LUN0-0, LUN0-1, LUN2-0	mix4	LUN0-0, LUN0-1, LUN3-0
mix5	LUN0-0, LUN0-1, LUN2-0	mix6	LUN0-0, LUN0-1, LUN2-0
	LUN2-1		LUN3-0

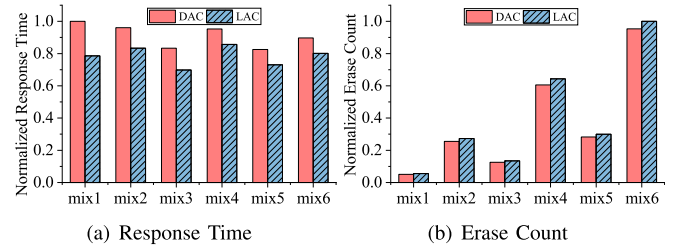


Fig. 18. Performance comparison of LAC and DAC.

schemes LRU, CFLRU, GCaR-LRU, and VS-Batch. It achieves lower erase count than the active write-back schemes MQSim and Co-Active by 49.3% and 20.6%, respectively. LAC exhibits optimized performance with a marginally increased erase count compared with VS-Batch. However, LAC significantly reduces response time and erase count than other schemes.

8) *Comparison of LAC and DAC*: We study the performance of DAC and LAC under mixed loads with a 256-MB cache in the 512-GB SSD. These mixed workloads used in this test are listed in Table V. In Fig. 18, LAC reduces response times by 21.5%, 13.2%, 16.1%, 10.0%, 11.5%, and 10.6% compared to DAC under mix1, mix2, mix3, mix4, mix5, and mix6, respectively. Nonetheless, LAC exhibits an increment in the erase count by 8.8%, 7.0%, 7.1%, 6.2%, 5.9%, and 4.8% across the tested workloads. There is a smaller request interval in the mixed workloads, and access conflict of user requests becomes more severe. LAC detects the appropriate time to write back within a small request interval, minimizing response time. The active write-back threshold of DAC indicates the proportion of clean data space in the cache. This threshold is judiciously adjusted according to the ratio of pages in the linked list, but the adjustment range is small. This threshold may be small for a long period, which leads to an insufficient frequency of active writes and cannot generate enough clean data. LAC exhibits greater WA than DAC due to its larger write-back threshold, which writes back a small portion of the hot data, and consequently increases WA.

TABLE VI
CACHING SCHEMES IN RELATED WORK

Caching Schemes	Prior work	I/O ¹ Intensity	Page ² Hotness	Miss ³ Penalty	Chip ⁴ State	Die ⁵ State	Parallel ⁶ Write	Dynamic ⁷ Adjustment
Front end	[9]	✗	✗	✗	✗	✗	✗	✗
	[11]	✗	✓	✗	✗	✗	✗	✗
	[10][29][30][31][32][33]	✗	✓	✗	✗	✗	✗	✓
	[6]	✗	✓	✗	✗	✗	✓	✓
	[34]	✗	✓	✓	✗	✗	✗	✗
	[35]	✗	✗	✗	✗	✗	✗	✓
Back end	[12]	✗	✗	✓	✗	✗	✗	✗
	[27][36][15]	✗	✗	✓	✓	✗	✗	✗
	[14]	✗	✓	✓	✗	✗	✗	✗
	[13]	✗	✗	✓	✓	✗	✗	✗
Front-Back end	[37][16]	✗	✓	✓	✗	✗	✗	✗
	[3]	✗	✓	✓	✓	✗	✗	✗
	LAC	✓	✓	✓	✓	✓	✓	✓

* ¹I/O Intensity is measured as the amount of data per second for the workload. ²Page Hotness refers to if cached data is separated based on temperature. ³Miss Penalty indicates if the replacement order of dirty and clean data is taken into consideration. ⁴Chip State and ⁵Die State refer to whether the idle states of flash chip and die are considered. ⁶Parallel write indicates if cached pages are replaced in parallel. ⁷Dynamic adjustment is the ability to adjust execution parameters according to access patterns.

VII. RELATED WORK

We classify caching schemes into three categories. Symbols ✓ or ✗ in Table VI indicates if a caching scheme incorporates or ignore a metric.

A. Front-End Caching Scheme

Front-end caching schemes – focusing on the HIL of SSDs advocate for algorithms driven by I/O access patterns. For example, ARC [10], SHARC [29], TinyLFU [30], Clock-Pro [31], and Wang [33] embrace the access frequency of page requests. SHARC allows the shadow recency cache to be used as an alternative to a virtual cache region, and the shadow recency cache grows as needed to achieve good performance under strong and weak localities. TinyLFU acts as an admission filter and inserts entries from the window cache that fulfill the conditions into the main cache, improving the cache hit ratio under skewed-access workloads. TinyLFU focuses on frequency, whereas LAC depends on multidimensional access history. CLOCK-Pro uses a circular queue to simulate the hotness of pages. The reference bits and access history is used to determine cache replacement and protect frequently accessed pages. However, LAC divides the cache space into three regions based on the number of accesses and the rewrite distance of the request. In each region, we design a data protection scheme to avoid high WA caused by repeated write-backs of hot data.

The other solutions [32], such as VS-Batch [11], Pattern+ [26], GALRU [34], and ARB [6] utilize I/O temporal and spatial locality. For instance, VS-Batch [11] employs the I/O temporal and spatial locality – and employs the visibility graph [38] to unify the temporal and spatial locality of I/O accesses [39]. In our study, LAC leverages the spatial locality to write back page requests with adjacent or close logical addresses in parallel, which ramps up the clean data conversion rate under small-request intervals. Cacheus [35] and work [40] increase the accuracy of classification for workloads and request types by using a machine learning model. Cacheus [35] adapts to various

workloads by using machine learning, and its two lightweight cache replacement policies can be dynamically adjusted based on real-time workloads. LAC relies on the I/O access patterns in workloads to adjust the hot, warm, and cold region sizes for accurate hot-cold data separation. LAC’s implementation is easier than that of Cacheus, and the resource overhead of LAC is marginal.

These algorithms mostly consider the I/O access pattern of the workloads rather than the hot-and-cold I/O access patterns over time and flash memory status inside the SSD. We should consider both the I/O access pattern and the flash memory status, aiming to maximize the overall system performance.

B. Back-End Caching Strategy

Back-end caching schemes exploit flash memory features: read/write imbalance, cache miss cost, and execution state. CFLRU [12] and CFDC [41] deal with read/write performance imbalance in flash memory, and replacing clean data is prioritized over updating data in flash memory. This method is inefficient because there are limited clean data under write-intensive workloads. Caching schemes such as GCaR [27], LCR [36], ECR [15], and caching-aware GC [13] incorporate the flash state of the SSD back-end in cache management algorithms, thereby alleviating access conflict between external and internal I/Os. With a quantified cost of cache misses in place [17], MQSim [14] proactively writes dirty data back from cache to flash memory. Uncontrolled write backs, however, tend to enlarge response times and WA under high I/O intensity. The majority of the above approaches fail to exploit flash parallelism coupled with a die-level load monitor. In contrast, LAC increases the utilization of parallelism by sequentially write requests with continued spatial addresses back to flash memory through parallel writes. The die-level monitor module revamp the die utilization, where the number of active write-backs is surged with reduced response time.

C. Front-Back-End Caching Scheme

Front-back-end caching schemes combine both I/O access patterns and flash memory status. LRU-WSR [37] and CCF-LRU [16] explore the hot and cold states of I/O requests and cache-miss costs. These schemes prioritize replacing cold-clean and cold-dirty pages, thereby minimizing replacement cost and response latency. Co-Active [3] detects the flash chip status through a chip-level monitor. Facing an idle chip, the active write-back module writes cold-dirty data from the cache to the flash memory: clean data are originated – and replacing clean data is prioritized in cache replacement. The analysis on the relationship between I/O intensity and write latency in flash memory is in its infancy: such an analysis makes it possible to fully utilize idle intervals among I/O requests and the idle flash memory. LAC writes back more cold data under low I/O intensity: access conflict between user I/Os are averted to slash response time and WA.

VIII. DISCUSSION AND CONCLUSION

In this section, we discuss the overhead and complexity of LAC and its trade-offs between performance and costs. Then, we conclude our work in this paper.

A. Discussion

1) *Overhead and Complexity*: To achieve a high-performance lookup of requests, we designed a set of hash tables. The time complexity of the hashing algorithm is $O(1)$. When an active write-back is invoked, data that meets the write-back conditions in the dirty linked list will be traversed – and the time complexity in the worst-case scenario is $O(n)$. The amount of dirty data is small under active write-back, which entails short traversal time. The space overhead includes the data structure of a linked list node, the linked list, and a hash table. The cache data structure has four items: *last_access*, *rewrite distance*, *F*, and *Age*. The *last_access* is the sequence number of the last page request in the cache. Each item in the data structure consumes 4 B. The size of each node in the linked list and hash map is 8 B; hence, the total space is 32 B. The size of each cached page is 8 KB. Therefore, the space overhead is approximately 0.4%. The other spatial costs, such as the I/O-intensity-aware and die-level load monitor modules, remain constant without introducing significant overheads.

LAC is a software-based caching layer within SSD firmware that enhances data access performance without altering the back-end hardware. LAC's parameters and policies can be tuned for various workloads and system requirements without additional hardware required and costs.

2) *Trade-Offs Between Performance, Lifespan, and Costs*: LAC does not sacrifice performance to enhance SSD lifetime and does not reduce the total amount of data to be actively written back. LAC spreads write-backs during low-I/O-intensity periods discovered by the I/O-intensity awareness module, which avoids blocking requests to shorten latency. LAC lowers WA by separating hot and cold data. It is preferential to write

back cold data in advance without increasing WA, whereas writing back hot data that is reaccessed escalates WA. We studied the performance of LAC in the case of multiple cache sizes in Section VI-A. We observed that LAC maintains stable performance under various cache sizes; thus, LAC can adapt to various workloads and system requirements. While increasing cache size may introduce extra costs, the significant performance improvements enabled by LAC, particularly in latency-sensitive applications, are likely to outweigh these costs.

B. Conclusion

LAC leverages I/O intensity awareness accompanied by flash load monitoring to boost the parallel write-back the dirty data, which generates clean data during low I/O intensity and idle die periods. LAC generates more clean data, prioritizes the replacement of clean data, and minimizes the overhead of cache replacement. LAC is adroit at avoiding access conflict between user I/Os and flash transaction bottlenecks, thereby cutting back response time. Furthermore, the data protection method is incorporated in LAC to fend off multiple refreshes of hot data back to flash memory, reducing WA and extending SSD lifetime. We conducted extensive experiments in MQSim to validate LAC compared with LRU, CFLRU, GCaR-LRU, MQSim, VS-Batch, and Co-Active. The results unveil that LAC curtails response time by up to 78.5% with 64.4% on average – and the number of erase counts is slashed by an average of 16.6% on average with a maximum of 47.8%.

ACKNOWLEDGMENT

We are grateful to Chen Sun, currently pursuing the M.S. degree in Anhui University, for helping us conduct extended experiments. His feedback and support improves our paper in this revised version.

REFERENCES

- [1] Z. Yang et al., "Autotiering: Automatic data placement manager in multi-tier all-flash datacenter," in *Proc. IEEE 36th Int. Perform. Comput. Commun. Conf.*, 2017, pp. 1–8.
- [2] B. S. Kim, J. Choi, and S. L. Min, "Design tradeoffs for SSD reliability," in *Proc. 17th USENIX Conf. File Storage Technol. (FAST)*, 2019, pp. 281–294.
- [3] H. Sun, S. Dai, J. Huang, and X. Qin, "Co-active: A workload-aware collaborative cache management scheme for NVMe SSDs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 6, pp. 1437–1451, Jun. 2021.
- [4] M. Tarihi, H. Asadi, A. Haghdooost, M. Arjomand, and H. Sarbazi-Azad, "A hybrid non-volatile cache design for solid-state drives using comprehensive I/O characterization," *IEEE Trans. Comput.*, vol. 65, no. 6, pp. 1678–1691, Jun. 2016.
- [5] J. Li, Z. Sha, Z. Cai, F. Trahay, and J. Liao, "Patch-based data management for dual-copy buffers in RAID-enabled SSDs," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3956–3967, Nov. 2020.
- [6] H. Lin et al., "Adaptive management with request granularity for DRAM cache inside nand-based SSDs," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 42, no. 8, pp. 2475–2487, Aug. 2023.
- [7] Z. Sha et al., "Low I/O intensity-aware partial GC scheduling to reduce long-tail latency in SSDs," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 4, pp. 1–25, 2021.
- [8] C. Lee, T. Kumano, T. Matsuki, H. Endo, N. Fukumoto, and M. Sugawara, "Understanding storage traffic characteristics on enterprise virtual desktop infrastructure," in *Proc. 10th ACM Int. Syst. Storage Conf.*, 2017, pp. 1–11.

- [9] A. Dan and D. Towsley, "An approximate analysis of the LRU and FIFO buffer replacement schemes," in *Proc. ACM SIGMETRICS Conf. Meas. Model. Comput. Syst.*, 1990, pp. 143–152.
- [10] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in *Proc. 2nd USENIX Conf. File Storage Technol. (FAST)*, vol. 3, 2003, pp. 115–130.
- [11] Z. Sha, Z. Cai, F. Trahay, J. Liao, and D. Yin, "Unifying temporal and spatial locality for cache management inside SSDs," in *Des., Automat. & Test Eur. Conf. & Exhib. (DATE)*, Piscataway, NJ, USA: IEEE Press, 2022, pp. 891–896.
- [12] S.-Y. Park, D. Jung, J.-U. Kang, J.-S. Kim, and J. Lee, "CFLRU: A replacement algorithm for flash memory," in *Proc. Int. Conf. Compilers, Archit. Synthesis Embedded Syst.* 2006, pp. 234–241.
- [13] Y. Pan, M. Lin, Z. Wu, H. Zhang, and Z. Xu, "Caching-aware garbage collection to improve performance and lifetime for NAND flash SSDs," *IEEE Trans. Consum. Electron.*, vol. 67, no. 2, pp. 141–148, May 2021.
- [14] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, and O. Mutlu, "MQSim: A framework for enabling realistic studies of modern multi-queue SSD devices," in *Proc. 16th USENIX Conf. File Storage Technol. (FAST 18)*, 2018, pp. 49–66.
- [15] H. Chen, Y. Pan, C. Li, and Y. Xu, "ECR: Eviction-cost-aware cache management policy for page-level flash-based SSDs," *Concurrency Comput., Pract. Exp.*, vol. 33, no. 15, 2021, Art. no. e5395.
- [16] Z. Li, P. Jin, X. Su, K. Cui, and L. Yue, "CCF-LRU: A new buffer replacement algorithm for flash memory," *IEEE Trans. Consum. Electron.*, vol. 55, no. 3, pp. 1351–1359, Aug. 2009.
- [17] M. Huang, O. Serres, V. K. Narayana, T. El-Ghazawi, and G. Newby, "Efficient cache design for solid-state drives," in *Proc. 7th ACM Int. Conf. Comput. Frontiers*, 2010, pp. 41–50.
- [18] S. Jiang and X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 1, pp. 31–42, 2002.
- [19] W. Bux and I. Iliadis, "Performance of greedy garbage collection in flash-based solid-state drives," *Perform. Eval.*, vol. 67, no. 11, pp. 1172–1186, 2010.
- [20] A. Gupta, Y. Kim, and B. Ugaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," *ACM Sigplan Notices*, vol. 44, no. 3, pp. 229–240, 2009.
- [21] M. Jung and M. T. Kandemir, "Sprinkler: Maximizing resource utilization in many-chip solid state disks," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2014, pp. 524–535.
- [22] "Systor'17 traces." SNIA IOTTA Repository. 2022. [Online]. Available: <http://iota.snia.org/traces/block-io/4928>
- [23] "MSR Cambridge traces." SNIA IOTTA Repository. 2023. [Online]. Available: <http://iota.snia.org/traces/388>
- [24] "Umass trace repository." UMass Trace Repository. 2023. [Online]. Available: <http://traces.cs.umass.edu/index.php/Storage/Storage>
- [25] B. Van Houdt, "On the necessity of hot and cold data identification to reduce the write amplification in flash-based SSDs," *Perform. Eval.*, vol. 82, pp. 1–14, Dec. 2014.
- [26] J. Li et al., "Pattern-based prefetching with adaptive cache management inside of solid-state drives," *ACM Trans. Storage*, vol. 18, no. 1, pp. 1–25, 2022.
- [27] S. Wu, Y. Lin, B. Mao, and H. Jiang, "GCaR: Garbage collection aware cache management with improved performance for flash-based SSDs," in *Proc. Int. Conf. Supercomputing*, 2016, pp. 1–12.
- [28] Q. Chen et al., "PACA: A page type aware read cache scheme in QLC flash-based SSDs," in *Proc. IEEE 40th Int. Conf. Comput. Des. (ICCD)*, Piscataway, NJ, USA: IEEE Press, 2022, pp. 59–66.
- [29] X. Du and C. Li, "SHARC: Improving adaptive replacement cache with shadow recency cache management," in *Proc. 22nd Int. Middleware Conf.*, 2021, pp. 119–131.
- [30] G. Einziger, R. Friedman, and B. Manes, "TinyLFU: A highly efficient cache admission policy," *ACM Trans. Storage*, vol. 13, no. 4, pp. 1–31, 2017.
- [31] S. Jiang, F. Chen, and X. Zhang, "CLOCK-Pro: An effective improvement of the CLOCK replacement," in *Proc. USENIX Annu. Tech. Conf.*, 2005, pp. 323–336.
- [32] C. Chakraborti and H. Litz, "Learning I/O access patterns to improve prefetching in SSDs," in *Machine Learning and Knowledge Discovery in Databases*. New York, NY, USA: Springer-Verlag, 2021, pp. 427–443.
- [33] Y. L. Wang, K. T. Kim, B. Lee, and H. Y. Youn, "A novel buffer management scheme based on particle swarm optimization for SSD," *J. Supercomput.*, vol. 74, pp. 141–159, Jan. 2018.
- [34] J. Kwak et al., "GALRU: A group-aware buffer management scheme for flash storage systems," *IEEE Access*, vol. 8, pp. 185360–185372, 2020.
- [35] L. V. Rodríguez et al., "Learning cache replacement with {CACHEUS}," in *Proc. 19th USENIX Conf. File Storage Technol. (FAST 21)*, 2021, pp. 341–354.
- [36] C. Liu et al., "LCR: Load-aware cache replacement algorithm for flash-based SSDs," in *Proc. IEEE Int. Conf. Netw., Archit. Storage (NAS)*, Piscataway, NJ, USA: IEEE Press, 2018, pp. 1–10.
- [37] H. Jung, H. Shim, S. Park, S. Kang, and J. Cha, "LRU-WSR: integration of LRU and writes sequence reordering for flash memory," *IEEE Trans. Consum. Electron.*, vol. 54, no. 3, pp. 1215–1223, Aug. 2008.
- [38] L. Lacasa, B. Luque, F. Ballesteros, J. Luque, and J. C. Nuño, "From time series to complex networks: The visibility graph," *Proc. Nat. Acad. Sci.*, vol. 105, no. 13, pp. 4972–4975, 2008.
- [39] C. Tang et al., "Cache eviction for SSD-HDD hybrid storage based on sequential packing," *J. Syst. Archit.*, vol. 141, 2023, Art. no.102930.
- [40] H. Wang, X. Yi, P. Huang, B. Cheng, and K. Zhou, "Efficient SSD caching by avoiding unnecessary writes using machine learning," in *Proc. 47th Int. Conf. Parallel Process.*, 2018, pp. 1–10.
- [41] Y. Ou, T. Härder, and P. Jin, "CFDC: A flash-aware replacement policy for database buffer management," in *Proc. 5th Int. Workshop Data Manage. New Hardware*, 2009, pp. 15–20.



Hui Sun received the Ph.D. degree from Huazhong University of Science and Technology, in 2014. He is an Associate Professor of computer science with Anhui University. His research interests include computer systems, performance evaluation, non-volatile memory-based storage systems, and file systems.



Haoqiang Tong is currently working toward the M.S. degree with Anhui University. His research interests include SSDs and intelligent cache management.



Yinliang Yue received the Ph.D. degree in computer architecture from Huazhong University of Science and Technology, China, in 2011. He is a Professor with the Institute of Information Engineering, Chinese Academy of Sciences. His research interests include computer architecture and storage systems. He has more than 10 publications in major journals and international conferences including the IEEE TRANSACTIONS ON COMPUTERS, ICDCS, and IPDPS.



Xiao Qin received the B.S. and M.S. degrees in computer science from Huazhong University of Science and Technology, in 1992 and 1999, respectively, and the Ph.D. degree in computer science from the University of Nebraska-Lincoln, in 2004. He is currently an Alumni Professor and the Director of Graduate Programs with the Department of Computer Science and Software Engineering, Auburn University. He won an NSF CAREER Award in 2009. His research interests include parallel and distributed systems, storage systems, and

performance evaluation.