AGC: An Adaptive Workload Burst-Aware Garbage Collection Mechanism for High-Performance SSDs

Hui Sun, *Member, IEEE*, Haisheng Ding, Haoqiang Tong, Xiang Cheng, Honggang Chai, and Xiao Qin, *Senior Member, IEEE*

***Abstract*—In NAND flash-based solid-state drives (SSDs), system performance and quality of service tend to be degraded by frequent conflicts between garbage collection (GC) I/Os and user I/Os. To address this challenge, we propose an adaptive workload burst-Aware Garbage Collection (AGC) mechanism to build high-performance SSDs. AGC minimizes conflicts between user I/Os and GC I/Os by jointly considering access patterns in workloads, internal GC, and allocation strategies. The running time of a workload is divided into multiple five-millisecond time windows. AGC incorporates a burst I/O access pattern prediction mechanism, which anticipates I/Os patterns in subsequent time windows according to historical access patterns. Pattern predictions allow AGC to proactively buffer valid pages during GC to prevent interference with forthcoming user I/O requests. According to the status of the underlying flash memory, we implement a hybrid allocation strategy to reduce GC-induced user writes blocking while preserving read parallelism performance. AGC classifies data into four hotness levels (hot, warm, cool, and cold) according to the update interval of the same request. Then, we design a hotness-aware victim block selection policy that prioritizes hot blocks to avert valid data migration during GC, thereby reducing write amplification in SSDs. We evaluate AGC through extensive experiments driven by real-world traces. The findings confirm that compared with state-of-the-art schemes (Baseline, CachedGC, FFT-GC, HIPA, and RDA), AGC reduces read and write response time by up to 81.57% and 69.66%, respectively, with average reductions of 54.34% and 39.21%. Furthermore, AGC decreases GC counts by up to 24.31% with an average of 12.04%, thereby reducing GC overhead and extending SSD lifetime across diverse workloads.**

***Index Terms*—Access conflicts, garbage collection, data allocation, workload aware, solid state drives.**

Received 31 July 2025; revised 26 November 2025; accepted 7 December 2025. Date of publication 12 November 2025; date of current version 12 February 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 62472002 and Grant 62072001. The work of Xiao Qin's was supported in part by U.S. National Science Foundation under Grant IIS-1618669 and Grant OAC-1642133, in part by the National Aeronautics and Space Administration under Grant 80NSSC20M0044, in part by the National Highway Traffic Safety Administration under Grant 451861-19158, and in part by Wright Media, LLC under Grant 240250 and Grant 240311. Recommended for acceptance by G. Sun. *(Corresponding author: Hui Sun.)*

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

Honggang Chai is with Wuhan HOSTOR Information Technology Company Ltd, Wuhan 430079, China (e-mail: jackeychai@adritas.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.2025.3642636

I. Introduction

NAND Flash-based solid-state drives (SSDs) have been widely deployed in data centers, high-performance computing, and consumer electronic devices [1], [2], [3] thanks to their superior random performance, low latency, and low power consumption [4], [5], [6]. The out-of-place update of flash memory rewrites valid data to free blocks while marking invalid pages, leading to invalid page accumulation. When available space in an SSD falls below a predetermined threshold, the garbage collection (GC) mechanism is sparked to reclaim invalid pages [7]. This process involves valid data migration (*i.e.*, GC I/Os) and block erasure operations, which interfere with user I/O requests – and cause a long-tail latency problem. The interference between user and GC I/Os is introduced by internal SSD activities: GC and data allocation. Thus, optimizing GC and data allocation mechanisms is essential for enhancing SSD performance.

Prior proposed methods – addressing the challenges posed by GC in SSDs – can be categorized into three camps: victim block selection, GC scheduling, and hotness-aware data separation in the workload.

When it comes to the victim block selection, DA-GC [8] presents a dynamic approach to balance wear leveling with GC operations, whereas Aware-GC [9] selects the victim blocks that have the minimum value of valid pages minus cached valid pages among all blocks in flash memory for GC, thereby reducing write amplification and enhancing GC efficiency through the elimination of redundant page migration. These techniques focus on optimizing the block reclamation during GC to maximize GC efficiency while balancing wear-out across flash memory inside the SSD. In the GC scheduling scenarios, Lee et al. [10] prioritized user I/O requests by preemption; however, the accumulation of GC operations ultimately degrades SSD performance and lifetime. Similarly, Wu et al. [11] suspended erase operations for preset time intervals to mitigate the performance impact at the cost of hardware. Gao et al. [7] proposed CachedGC to reduce resource contention by buffering valid pages during GC and writing them back once the chip is idle; however, subsequent writes can still conflict with GC I/Os. Sha et al. [12] designed the Fourier transformation to perform partial GC in the case of low I/O intensity, which mitigates long-tail latency while increasing write amplification due to premature GC. Additionally, a hotness-aware data separation potentially places similar hotness pages in a block, curtailing

0018-9340 © 2025 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:46 UTC from IEEE Xplore. Restrictions apply.

valid page migration [13], [14]. To improve performance by classifying data hotness, a GC mechanism proposed in SUP-GC [14] defines user-written data and GC-induced data as hot and cold, respectively.

We conducted extensive experiments in a popular SSD simulator, MQSim, and identified that the burst characteristics in user I/O access patterns significantly impair SSD performance. When a sufficient number of user I/O requests within a short time window are blocked by GC, these requests are referred to as *"burst conflict requests"* in this study. Notably, numerous burst conflict requests cause significant tail latency and degrade SSD performance. Furthermore, we found that when the available space in the SSD is limited, dynamic allocation strategies effectively prevent GC blocking by allocating user I/O requests to free resources (channel, chip, die, and plane), thereby outperforming static allocation. Additionally, the data hotness in workloads exhibits various types instead of the binary classification based on the update intervals of write requests.

In this paper, we propose an adaptive workload burst-**A**ware **G**arbage **C**ollection (AGC) mechanism for high-performance SSDs. AGC minimizes conflicts between user I/Os and GC I/Os by jointly considering access patterns in workloads, internal GC, and allocation strategies. In AGC, we implement a hybrid allocation strategy that reduces the number of user-blocking writes induced by GC while preserving read parallelism. When AGC predicts burst user I/O requests in the upcoming time window, it preemptively migrates valid pages from the current GC process to a valid page buffer, thereby mitigating GC interference with user I/O requests. Furthermore, to reduce write amplification caused by GC, we classify user data into four types of hotness: cold, cool, warm, and hot. The contributions of this study are tabulated as follows.

- We gauge the impact of burst and non-burst conflict requests on SSD performance, particularly highlighting the severe effect of burst conflict requests on tail latency. To study the impact of burst conflict requests on response time, we introduce the '*Burst Conflict Impact Factor (BC_IF)*' as a new performance metric.
- In AGC, we study the performance of static and dynamic allocation strategies under various SSD free space conditions. We design a hybrid allocation strategy to achieve an optimal balance between read parallelism and conflict avoidance. We also propose a four-type hotness classification based on update intervals, allocating data to flash blocks corresponding to their hotness.
- AGC incorporates a predictive model for user burst I/O requests in upcoming time windows and employs a page caching mechanism to effectively mitigate tail latency caused by burst conflict requests.
- We implement AGC in a popular SSD simulator—MQsim. We conducted extensive experiments under real-world traces. AGC reduces read and write response time by up to 81.57% and 69.66%, respectively. Furthermore, AGC decreases the number of GC counts by an average of 12.04%, extending the SSD lifetime.

The rest of the paper is organized as follows. We present the background in Section II. In Section III, we articulate related work and elaborate on the motivations behind our proposed AGC. We describe the system and its implementation in Section IV. In Section V, we present the experimental setups and results. In Section VI, we discuss related work. Finally, we draw the conclusion in Section VII.

II. Background

Modern SSDs implementing the Non-Volatile Memory Express (NVMe) protocol comprise multiple components that collaborate to deliver optimal performance [15]. The host interface logic (HIL) module partitions user requests into page-sized transactions. A flash transaction is distinguished by metadata information, such as logical page address (LPA), bitmap, and time stamp, which locates the address to store transaction data [16]. The flash translation layer orchestrates an array of modules and algorithms to schedule transactions and improve SSD performance.

The cache management module writes requests sharing the same LPAs as those in the prior request to the cache, thereby improving performance with fewer writes to flash memory [17], [18]. GC reclaims invalid pages incurred by out-of-place updates in flash memory. The wear-leveling balances block erasures in flash memory to prolong SSD lifetime [19]. The address mapping module manages a mapping table from an LPA to a physical page address (PPA) [20]. When data is updated, to-be-removed physical pages will be set to invalid ones by mapping entries. In addition, the data allocation module determines data layout from static allocation or dynamic allocation [21], [22]. Transactions with PPAs enter chip queues [23] and are then scheduled by the transaction scheduling unit according to a policy [24]. Finally, the flash channel controllers issue commands to execute read, program, and erase operations in the flash memory.

III. Motivation

Existing studies on SSD performance exhibit certain limitations. Many focus primarily on improving internal GC or allocation strategies while inadequately addressing *I/O access patterns* in workloads [25]. Although some studies consider both I/O access patterns and *GC* (or *allocation*) strategies [12], [26], there remains ample opportunities for performance improvement by jointly optimizing all three factors – I/O access patterns, internal GC, and allocation strategies – which collectively offer promising avenues for SSD performance enhancement.

In this research, we employ a popular SSD simulator – MQSim [15] – to study the impact of the three factors on SSD performance under Microsoft Research Cambridge Repository [27], Virtual Desktop Infrastructure (VDI) [28], and Alibaba block traces [29]. MSR workloads include block I/O traces of enterprise servers at Microsoft. VDI workloads capture I/Os patterns in enterprise virtualization environments with multiple users. Alibaba block traces exemplify modern write-intensive access patterns in large-scale cloud services. Table I presents the characteristics of workloads.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:46 UTC from IEEE Xplore. Restrictions apply.

TABLE I
CHARACTERIZATION OF TRACES

Trace[1]	TotalReq (10^6)	Write Ratio	Average Size (KB)	Write Size (GB)	T_{avg}[2] (ms)	T_{max} (s)	I_{avg}[3] (MB/s)	I_{max} (MB/s)	Update Ratio[4]
web_0	2.03	0.70	14.99	11.67	297.94	29.84	0.02	19.03	0.96
hm_0	3.99	0.65	7.99	20.48	151.45	18.72	0.03	33.69	0.93
LUN0-1	1.53	0.65	15.44	9.33	2.36	3.63	2.66	270.60	0.81
LUN1-1	1.12	0.32	23.88	4.66	3.21	0.77	1.33	206.20	0.64
LUN2-1	1.83	0.66	13.57	7.97	1.97	0.61	2.27	110.30	0.90
LUN3-1	1.36	0.36	23.69	6.06	2.64	3.91	1.73	138.60	0.64
LUN3-2	2.07	0.17	35.62	6.19	1.74	3.08	1.76	184.70	0.56
LUN6-1	2.65	0.14	20.71	6.50	1.36	0.95	1.85	178.09	0.61
ali-65	1.09	0.59	23.06	12.51	899.12	57.92	0.01	115.14	0.86
ali-168	0.93	0.99	8.52	7.34	280.52	14.45	0.03	118.01	0.88

Note: *[1] Traces from three workload sources: web_0 and hm_0 are from MSR workloads; LUN0-1 through LUN6-1 are from VDI workloads representing traces 20160217l5-LUN0, 20160223l8-LUN1, 20160304l15-LUN2, 20160224l4-LUN3, 20160318l8-LUN3, 20160318l9-LUN6 respectively; ali-65 and ali-168 are from Alibaba block traces; [2] T_{avg}, T_{max}: mean and maximum request interval; [3] I_{avg}, I_{max}: mean and maximum I/O intensity; [4] Update Ratio: proportion of write requests that will be subsequently updated.

A. Conflicts Between User I/Os and GC I/Os

User I/O requests in workloads may be blocked upon arrival at the SSD due to undergoing internal GC, thereby impacting response time. We refer to these GC-blocked user I/O requests as '*conflict requests*'. Notably, existing GC strategies [7], [26] cannot differentiate between various conflict requests, failing to identify burst conflict requests, which are a major contributor to severe SSD performance degradation. Previous work [17] identified user burst-request phases in which more than 1,000 sectors are accessed within a one-second time window. With the SSD performance enhancement and exponential data processing growth; however, this coarse-grained definition is hardly adequate for burst I/O access patterns. To study the impact of conflict requests on SSD performance, we refine the time window to 5 milliseconds, designating time windows with over 200-sector accesses as *'burst request phases'*. This 5ms time window size represents an optimal balance between prediction accuracy and the ability to capture fine-grained I/O access patterns (see V-C3). Consequently, user I/O requests blocked by GC in these phases are classified as *'burst conflict requests'*, which are primarily investigated in our work.

We conduct experiments to study the heterogeneity of conflict requests. Fig. 1 unveils the distribution of request response time. The x-axis represents the sequence of consecutive read/write requests in the trace, with each operation type (read or write) separately plotted while preserving their original relative ordering within their respective sequences. The y-axis employs a logarithmic scale to show the distribution of response time variations. The majority of read requests maintain a stable response time of around 100 µs, whereas write requests stabilize at about 850 µs, representing normal requests unaffected by GC interference. Additionally, some read response time exhibits between 500 and 1,000 µs, corresponding to requests during burst request phases without GC. Importantly, read requests blocked by GC during burst request phases demonstrate response time spikes of 10-100 ms. In comparison, the peak response time of write requests reaches approximately 100 times the stable value (blue line in Fig. 1). Such a disparity demonstrates that burst conflicts between the user and GC I/Os largely affect read response time compared to write.

The experimental results demonstrate the effect of burst and non-burst conflict requests on SSD performance under traces. Our findings confirm that burst conflict requests are a critical bottleneck to the response time. Hence, we propose novel metrics to quantify the impacts of non-burst and burst conflict requests on SSD performance. In this section, OP denotes the operation type, which can be either R (read) or W (write). We denote the total number of user requests and the average response time as N_{u_OP} and T_{u_OP}. We define the number of burst conflict requests and their average response time as N_{bc_OP} and T_{bc_OP}. Meanwhile, N_{c_OP} and T_{c_OP} represent the quantities and average response time of non-bursty conflict requests.

We define ΔT_{bc_OP} as the additional latency of burst conflict requests compared to regular user requests, see (1). Similarly, ΔT_{c_OP} represents the additional latency of non-burst conflict requests compared to user requests, see (2).

$$\Delta T_{bc_OP} = T_{bc_OP} - T_{u_OP} \tag{1}$$

$$\Delta T_{c_OP} = T_{c_OP} - T_{u_OP} \tag{2}$$

Furthermore, F_{bc_OP} in (3) denotes the proportion of burst conflict requests among total user requests. Likewise, F_{c_OP} in (4) represent the proportion of non-burst conflict requests among total requests.

$$F_{bc_OP} = N_{bc_OP}/N_{u_OP} \tag{3}$$

$$F_{c_OP} = N_{c_OP}/N_{u_OP} \tag{4}$$

To study the impact of conflict requests on SSD response time, we introduce metric *'Garbage Collection Impact Factor'* (GC_IF) in (5). A high GC_IF value reflects a substantial performance degradation due to conflict requests. Similarly, we define metric *'Burst Conflict Impact Factor'* (BC_IF) in (6) to measure the impact of burst conflict requests on SSD performance.

$$\text{GC_IF} = F_{c_R} \cdot \Delta T_{c_R} + F_{c_W} \cdot \Delta T_{c_W} \tag{5}$$

$$\text{BC_IF} = F_{bc_R} \cdot \Delta T_{bc_R} + F_{bc_W} \cdot \Delta T_{bc_W} \tag{6}$$

Table II lists the normalized GC_IF and BC_IF values under the traces. The results reveal that BC_IF values substantially

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:46 UTC from IEEE Xplore. Restrictions apply.

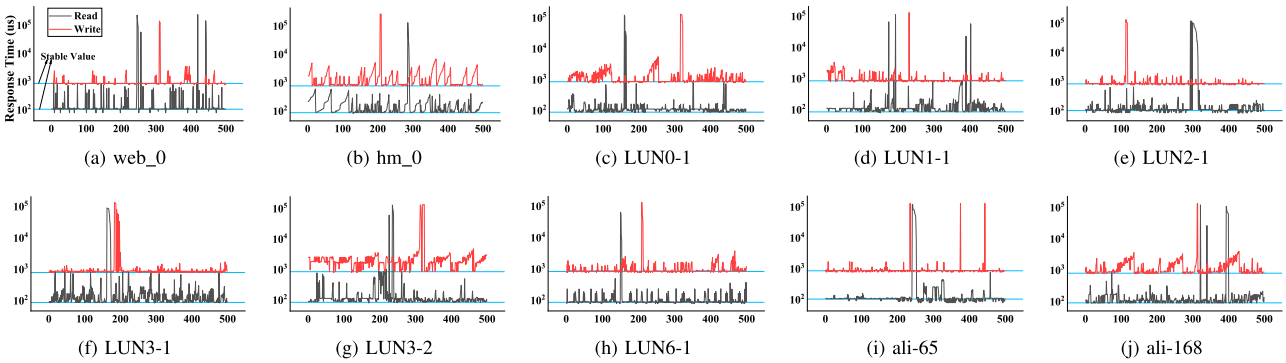


Fig. 1. Distribution of user request response times.

TABLE II
NORMALIZED GC_IF AND BC_IF

	web_0	hm_0	LUN0-1	LUN1-1	LUN2-1
GC_IF	1.00	1.00	1.00	1.00	1.00
BC_IF	1.77	3.59	12.72	21.11	12.39
	LUN3-1	**LUN3-2**	**LUN6-1**	**ali-65**	**ali-168**
GC_IF	1.00	1.00	1.00	1.00	1.00
BC_IF	9.91	32.79	18.55	10.29	1.90

exceed GC_IF values in most traces. In an extreme case, a BC_IF shows 32.79 times higher than its GC_IF counterpart. These results demonstrate that burst conflict requests incur much higher performance overhead than non-burst conflicts. Thus, they are a critical factor in SSD performance optimization.

We investigate the conflicts between user and GC I/Os, which reveals that GC-blocked requests during burst request phases largely degrade overall system performance, leading to severe tail latency issues —- a problem rarely mentioned in prior studies. Based on this finding, we propose a quantitative metric, BC_IF, to precisely measure the impact of burst conflict requests on SSD performance. Experimental results confirm that burst conflict requests negatively impact SSD performance to a greater extent than non-burst conflict requests. These findings motivate us to design a method to identify and prioritize burst conflict requests, thereby mitigating tail latency in SSDs.

B. Impact of SSD Free Space on Allocation Strategies

The existing allocation strategies [30], [31] inside SSDs primarily focus on enhancing parallelism using static and dynamic allocation strategies. To shorten read latency, the static allocation employs striping techniques to maximize SSD parallelism. The dynamic allocation optimizes write performance by preventing write blocking. The resource consumption during GC affects allocation strategies differently, which is overlooked in the prior studies. Thus, we evaluate the performance variations of allocation strategies across different available space inside SSDs. We simulate these cases using various preconditioning ratios, where higher ratios indicate fewer available free blocks, increasing the probability of GC.

Fig. 2 depicts the average read, write, and overall response time under the traces at various preconditioning ratios (5% intervals) in MQSim [15]. When preconditioning ratios range from 0% to 40%, response time remains relatively stable. This consistency occurs because there is sufficient free space in the SSD to accommodate user writes without triggering GC, thereby preventing GC-induced interference with user I/O requests. When preconditioning ratios increase to 40%-85%, response time rises significantly compared to the earlier stages. The change trend in Fig. 2 indicates that GC has been triggered to enlarge the response time with blocked user requests.

Based on SSD performance under varying free-space conditions, we observe that static and dynamic allocation strategies exhibit complementary advantages. Static allocation maintains high read performance through striping but suffers significant performance degradation as free space decreases. In contrast, dynamic allocation exhibits superior write performance, especially when GC operations are frequently invoked. These findings indicate that the design of an effective SSD data allocation should consider as GC mechanisms, adaptively combining both strategies to optimize overall performance.

C. Update Interval Distribution of Write Requests

To study the hotness of write requests, we analyze write request update intervals in the traces, where the update intervals are measured by the number of requests between successive updates to the same request's logical address. In Fig. 3, the x-axis represents the update interval of write requests, while the CDF on the y-axis indicates the proportion of write requests with update intervals less than or equal to a specific value.

CDF curves reveal that most workloads exhibit distinct bimodal characteristics. Within the 0-500 request interval, CDF curves demonstrate steep gradients, which indicates frequently updated data within short time periods. Under web_0 and LUN2-1, approximately 75% of write requests are updated within this interval, suggesting that the majority of writes in these traces localize in a small subset of hot requests. Furthermore, almost all the workloads exhibit a significant increase in request quantity within the 7000-8000 request interval. This trend indicates that write-once data remains un-updated throughout workloads. Within the 3000-7000 request interval, workloads embrace different I/O access patterns. LUN2-1

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:46 UTC from IEEE Xplore. Restrictions apply.

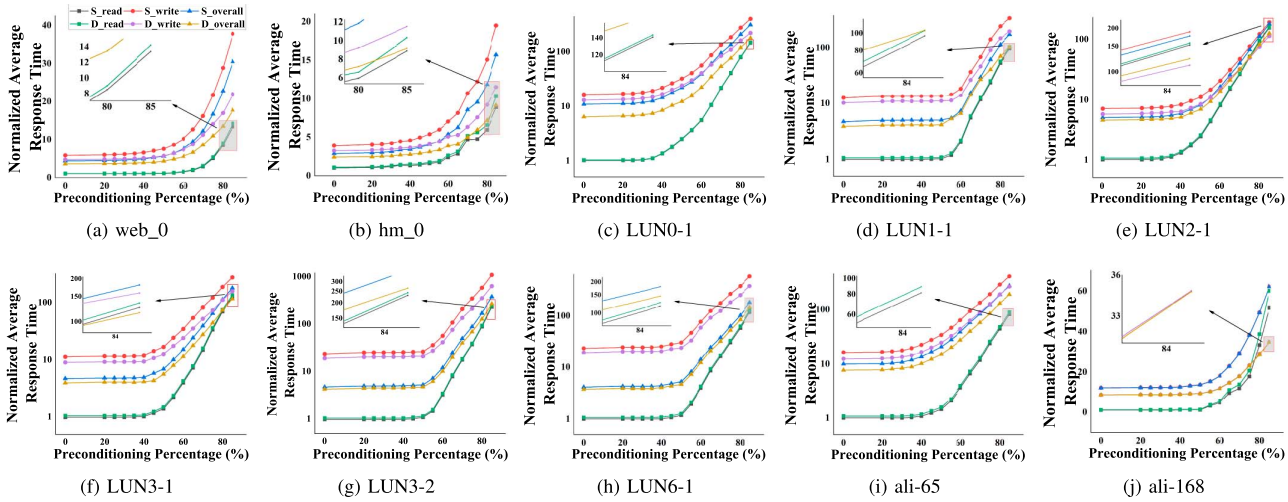


Fig. 2. Static vs. Dynamic allocation performance under different SSD free space levels.

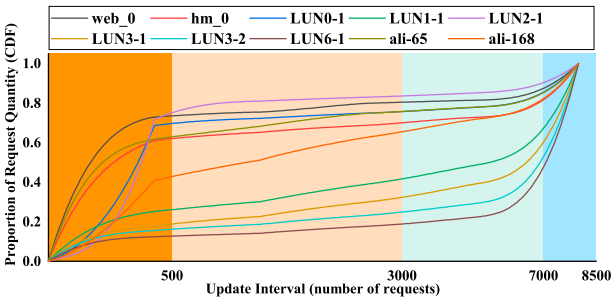


Fig. 3. Update interval distribution of write requests.

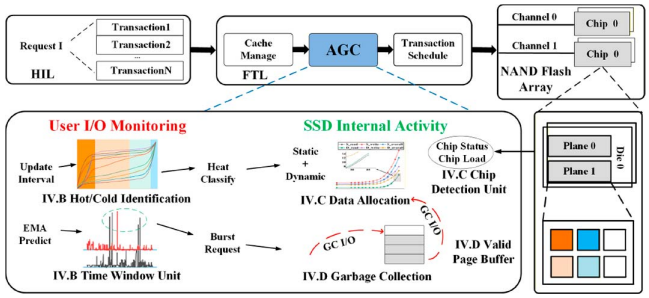


Fig. 4. System overview of AGC.

presents minimal curve growth in the two intervals: we recognize highly polarized I/O access patterns in which the requests predominantly consist of either hot or cold data, with negligible intermediate-hotness data. In contrast, ali-168 displays distinct gradients in the intermediate-hotness regions, meaning that there exists substantial intermediate-hotness data.

These findings lead to new ideas for designing data allocation policies: an effective allocation mechanism should identify multi-level data hotness rather than binary hot/cold classifications. Inspired by these insights on request intervals, we categorize data hotness into four regions: hot (0-500), warm (500-3,000), cool (3,000-7,000), and cold (7,000-8,000). This fine-grained hotness classification enables storing data with similar hotness in a cluster, which reduces valid page migrations during GC and mitigates write amplification. As a result, GC interference with user I/O requests is minimized during burst request phases.

IV. Design of AGC

A. System Overview

Fig. 4 illustrates the architecture of our proposed AGC. AGC comprises a user I/O request monitoring module, a chip status detection unit, a data allocation module, and a garbage collection module. The I/O request monitoring module coordinates a hot/cold identification unit and a time window unit. The hot/cold identification unit tracks I/O request arrival sequences and calculates intervals between consecutive accesses to determine data hotness, which guides subsequent data allocation and GC processes. The time window unit is implemented based on an Exponential Moving Average (EMA) [32] to detect burst I/O requests in workloads from the host. It predicts I/O intensity for the upcoming time window based on historical I/O patterns within defined time intervals. Then, the prediction results are forwarded to the garbage collection module.

The chip detection unit gleans the load status of chips, including idle status and the frequency of read, write, and erase operations in each chip and its waiting queue. The data allocation module leverages this load status along with SSD internal parallelism to optimize resource utilization: the module determines the optimal target plane for data allocation and assigns data to blocks in the flash memory corresponding to their hotness. When available space in flash memory falls below a predetermined GC threshold, the garbage collection module will utilize the burst I/O information from the time window unit to decide whether to temporarily store valid pages during GC migration in the valid page buffer.

In the workflow, the HIL divides user requests into page-sized transactions. The I/O monitoring module determines request hotness and predicts I/O intensity. Then, the data allocation

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:46 UTC from IEEE Xplore. Restrictions apply.

module assigns write transactions to chips based on hotness and chip load status. When free space drops below the GC threshold, the GC module decides whether to buffer valid pages from GC migration based on predicted I/O intensity and available buffer space. The buffer data is written back to blocks according to hotness when chips are sitting idle.

B. I/O Request Monitoring Module

To capture burst request phases in real time, we advocate using a time window unit within the HIL to partition the entire running time into equal-sized intervals. Then, we obtain the data volume in a time window to pinpoint the burst request phase. Specifically, we employ the EMA algorithm to predict the number of sectors accessed in the next time window. This prediction is then compared with the threshold of "burst request phases" (see section III-A) to determine whether a burst request phase occurs in the next time window. The EMA algorithm is expressed as

$$\mathrm{EMA}_t = \alpha \times X_t + (1-\alpha) \times \mathrm{EMA}_{t-1} (t \in \mathbb{Z}^+) \quad (7)$$

where EMA_t represents the moving average in the current period. X_t denotes the actual observation for the current period. EMA_{t-1} is the moving average for the previous period. α is the smoothing coefficient, and its value is in the range between zero and one ($\alpha \in (0, 1)$). When we have $\alpha > 0.5$, the current moving average is largely determined by the current observation; otherwise, the moving average largely depends on the moving average in the previous period.

We employ the current number of sectors (N_i) in time window t_i as the actual observation for the current period. A time window corresponds to an interval. The exponential moving average EMA_{t-1} is used as the predicted number of sectors P_i in time window t_i ($i >= 6$, $i \in \mathbb{Z}^+$).

To predict whether burst requests will occur in window t_i, we track the number of sectors accessed by users in the previous five consecutive windows. N_{i-1}, N_{i-2}, N_{i-3}, N_{i-4}, and N_{i-5} denote the number of sectors in time windows t_{i-1}, t_{i-2}, t_{i-3}, t_{i-4}, and t_{i-5}, respectively. The EMA algorithm is initially kicked in after the sixth window. Thus, when $i >= 6$, we predict the number of sectors P_i in time window t_i as follows

$$P_i = \mathrm{EMA}_{i-1} = \alpha \sum_{j=0}^{4} (1-\alpha)^j \cdot N_{i-j-1} + (1-\alpha)^5 \cdot \mathrm{EMA}_{i-5} \quad (8)$$

EMA_{i-5} is initialized as

$$\mathrm{EMA}_{i-5} = (N_{i-1} + N_{i-2} + N_{i-3} + N_{i-4} + N_{i-5})/5 \quad (9)$$

We decompose the calculation formula of EMA_{i-1} as follows:

$$\mathrm{EMA}_{i-4} = \alpha \times N_{i-4} + (1-\alpha) \times \mathrm{EMA}_{i-5} \quad (10)$$

$$\mathrm{EMA}_{i-3} = \alpha \times N_{i-3} + (1-\alpha) \times \mathrm{EMA}_{i-4} \quad (11)$$

$$\mathrm{EMA}_{i-2} = \alpha \times N_{i-2} + (1-\alpha) \times \mathrm{EMA}_{i-3} \quad (12)$$

$$\mathrm{EMA}_{i-1} = \alpha \times N_{i-1} + (1-\alpha) \times \mathrm{EMA}_{i-2} \quad (13)$$

Parameter α in EMA affects the algorithm's sensitivity. Considering the variable user access patterns, we set α to 0.6 to balance the weights of historical data volume and the latest data volume (see Section V-C3). Then, EMA_{i-1} denotes the predicted number of sectors that will be accessed by users in time window t_i. When this value exceeds 200, AGC recognizes that I/O request burstiness occurs in the next time window. To illustrate the EMA algorithm, we provide an example, and it assumes that the number of accessed sectors in five consecutive time windows are 150, 180, 210, 190, and 230, respectively. We first calculate $\mathrm{EMA}_{i-5} = 192$ (average value), and we used the smoothing factor $\alpha = 0.6$ to calculate the following EMA values. Second, we have $\mathrm{EMA}_{i-4} = 184.8$, $\mathrm{EMA}_{i-3} = 199.92$, $\mathrm{EMA}_{i-2} = 193.97$, $\mathrm{EMA}_{i-1} = 215.59$. Since the predicted value $P_i = 215.59$ exceeds the threshold of 200, AGC recognizes t_i as a burst request window. Through this mechanism, AGC preemptively buffers valid page data during GC, thereby cutting back the long-tail latency incurred by conflicts between user and GC I/Os.

Additionally, AGC assigns a unique access sequence number to each incoming write request. Then, update intervals can be diagnosed by calculating the sequence difference between consecutive updates to the same logical address of the write request. Based on the empirical analysis of update intervals in Section III-C, we categorize data hotness into four types. This four-level hotness guides both initial data allocation and valid page migration decisions amid GC, which can allocate data with similar access hotness in a block. This strategy minimizes unnecessary data migrations, reduces write amplification, and extends SSD lifetime. Our experimental results unveil that when a write request's update interval exceeds 8,000 – a threshold, the probability of subsequent updates to the write request's logical address becomes insignificant. In this case, we implement a memory optimization mechanism that frees memory space occupied by request sequence numbers. Such an approach optimizes memory space without compromising classification accuracy.

C. Hybrid Data Allocation

To fully exploit SSD internal parallelism and reduce tail latency caused by GC blockings, we propose a hybrid data allocation strategy combining the advantages of both static and dynamic allocation strategies. Traditional static allocation strategies establish target addresses for channel ID (C), chip ID, die ID (D), and plane ID (P) through mathematical formulas. To achieve high parallelism, AGC maintains channel-level parallelism [21] by employing a C-type [22] allocation method (see Algorithm 1). This strategy statically determines the channel ID while keeping the other levels dynamic, thereby preserving channel parallelism and enhancing resource flexibility. Subsequently, AGC polls for idle chips within the channel by querying the chip detection module. Given an idle chip, its ID is elected for a target address, with the die ID and plane ID allocated sequentially according to die- and plane-level parallelism (Lines 3-7). Otherwise, no idle chips implies that the CDP-type [22] allocation strategy is employed, which statically allocates the channel, die, and plane IDs while dynamically selecting the chip with the lowest transaction load (Lines 10-13).

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:46 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 Hybrid Page Allocation Strategy in AGC

Require:
The channel count, C; The die count per chip, D;
The plane count per die, P; Die ID of last allocation, Did;
Plane ID of last allocation, Pid;
Ensure:
Target address ID of LPA, target.xxID;

```
target.CID = LPA % C;      /* Preserve channel-level parallelism */
/* Query the chip detection module to find idle chips in target channel */
If Has_Idle_Chip(target.CID) then
    target.WID = Find_Idle_Chip(target.CID)
    /* Round-robin for die-level and plane-level parallelism */
    target.DID = (Did + 1) % D
    target.PID = (Pid + 1) % P
else    /* No idle chips available */
    /* Find the least loaded chip in target channel */
    target.WID = Find_Least_Busy_Chip(target.CID)
    /* CDP-type allocation for die ID and plane ID */
    target.DID = (LPA / C) % D
    target.PID = (LPA / (D * C)) % P
return target.xxID;
```

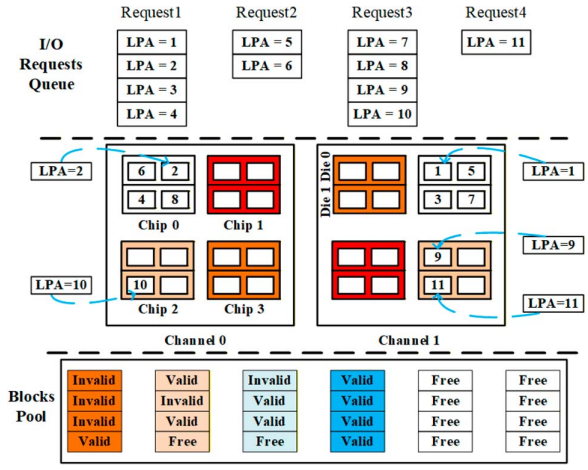


Fig. 5. Hybrid page allocation in AGC.

Finally, this hybrid strategy allocates the logical page address to a physical block in the corresponding hotness region based on hotness classification from the hot/cold identification module. New request arrivals without any access history are initially classified as cold data.

Fig. 5 presents an example demonstrating the hybrid allocation strategy deployed in AGC. In this storage architecture, each chip contains two dies (Die 0, Die 1)– and each die is comprised of two planes. The I/O request queue contains four various-sized write requests, which are decomposed into page-sized transactions. Each transaction is assigned a unique LPA and mapped to a physical location. For the transaction with LPA = 1, the target channel is determined as Channel 1 through a modulo (*i.e.*, 1% 2 = 1). Then, an idle chip within Channel 1 is identified, *e.g.*, Chip 1. Considering data distribution across hierarchical resources, Die 0 and Plane 0 are designated as the targeted locations. Transactions with LPAs 2 - 8 undergo the identical allocation procedure. After these allocations are completed, all chips across all the channels are occupied. In this case, new transactions are routed to chips with the shortest queue execution time. For instance, the transaction with LPA = 9 is allocated to Chip 3, with its target die and plane addresses according to Algorithm 1. We have the die number is 0 ((9/2) % 2 = 0) and the plane number is 0 ((9/(2 ∗ 2)) % 2 = 0). Similarly, allocation addresses for transactions with LPA = 10 and LPA = 11 are shown in Fig. 5.

Initially, blocks within each plane have no hotness. During programming operations, the SSD controller allocates a new block from the free block pool and assesses a hotness based on the update frequency of the incoming data. Subsequently, data with the same hotness are sequentially written into blocks with the corresponding hotness. Following erase operations, the hotness of blocks is reset before returning them to the free-block pool. This dynamic block allocation and GC mechanism ensures that hotness is effectively adapted to access patterns in workloads.

D. Burst-and-Hotness-Aware GC Strategy

The prior GC strategies [7], [12] store valid pages from victim blocks in a valid page buffer to minimize GC-induced blocking of user I/Os. In Section III-A, we find that burst conflict requests significantly impact overall performance, whereas the valid page buffer size is constrained. Consequently, AGC prioritizes the valid page buffer for storing migration data when GC is triggered during a user burst access period.

From the GC's block selection policy, the previous studies [7], [14] adopted the greedy strategy to reduce GC overhead. Nonetheless, this strategy struggles to ensure wear-leveling across blocks. To address this issue, we employ the random greedy algorithm (RGA) [33] to balance GC overhead and wear leveling. Similar to work [34], we implement our block selection policy anchored on the RGA.

Fig. 6 illustrates the GC process in AGC. GC is initiated when the free space inside an SSD falls below a preset threshold. The traditional RGA randomly picks victim blocks, which can cause high write amplification due to the candidate blocks containing many valid pages. While selecting victim blocks, AGC employs a hotness-aware prioritization strategy powered by the RGA policy [33]. Initially, AGC prioritizes electing hot blocks, gathering a preset number of candidate blocks. If the number of hot blocks is insufficient for RGA, AGC chooses warm blocks, then cool blocks, and finally cold blocks if necessary, until the required number of candidate blocks is satisfied. This selection procedure optimizes GC efficiency by prioritizing blocks containing data with higher update frequencies, and these blocks typically have fewer valid pages during GC. AGC's hotness-aware victim selection strategy is slated to reduce write amplification while ensuring efficient GC under dynamically changing workloads.

Next, AGC determines whether to buffer the valid pages requiring migration based on two factors: the I/O intensity forecast for the upcoming time window (provided by the I/O requests monitoring module) and the available capacity in the valid page buffer. When the two conditions are satisfied, AGC executes the erase operation to reclaim blocks while deferring the out-of-place update in GC, thereby diminishing the chance

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:46 UTC from IEEE Xplore. Restrictions apply.

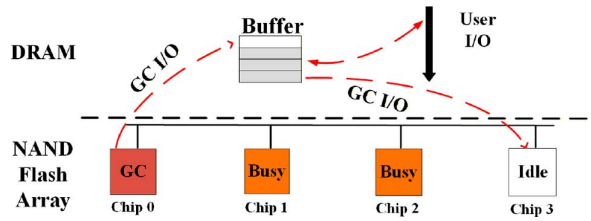


Fig. 6. Valid page buffer management in AGC.

of GC processes interfering with incoming user burst I/Os. If the valid page buffer lacks sufficient capacity for the GC's valid pages, the operation follows the conventional GC sequence, completing valid page migration before executing the block erasure. When the requested data has been cached in the valid page buffer, read requests retrieve the data from the buffer, while write requests are released after the buffer is updated [7]. When chips are idle and no burst requests are predicted for the upcoming interval, buffered data is proactively flushed to flash blocks, ensuring efficient buffer utilization. Based on this burst-and-hotness-aware GC strategy, AGC reduces tail latency while expanding SSD lifetime.

E. Discussions

Time Overhead. The prediction algorithm exploited by the load-aware mechanism applies the EMA method, which is required to run once at the end of each time window. Given n time windows, the complexity of a prediction is a constant $O(1)$ thanks to the straightforward arithmetic operations involving only addition and multiplication. Thus, the total time complexity for n time windows is $O(n)$, where n – being a very small constant like 5 – entails small time overhead. In this case, the main operations include update-distance computation and migration of hot-cold data. The update distance is calculated for each block, and its time complexity is $O(m)$ given a total m of blocks.

Space Overhead. AGC's space overhead includes two metadata components: the workload-aware prediction and the hotness-aware data separation mechanisms. The workload-aware prediction mechanism maintains I/O access logs across n time windows with $O(n)$ space complexity. With $n = 5$ in AGC, this overhead of around 40 Bytes is negligible. The hotness-aware data separation mechanism requires hotness classification metadata and access sequence numbers. The SSD in this study is configured with 262,144 blocks (see Section V-A), the hotness classification uses two bits per block for four categories, resulting in 64 KB space cost. Each write request record consumes 12 B (8 B request ID and 4 B access sequence number). The memory optimization mechanism purges records with update intervals exceeding 8,000, limiting active write requests to around 10^6 (around 11.44 MB). The total space overhead is about 11.51 MB, which is insignificant in the DRAM of an SSD.

V. Experiment and Evaluation

A. Experimental Setups

1) Settings: We implement AGC in an open-source multi-queue SSD simulator named MQSim [15]. MQSim accurately simulates the front- and back-end components of modern multi-queue SSDs. The simulation parameters are listed in Table III. The approach to selecting blocks in GC is RGA. Its default hot-cold data separation method defines the user data as hot while the valid page data as cold. Initially, we utilize a preconditioning function in MQSim to initialize the SSD space before loading the trace, which ensures that GC is triggered under traces. Similar to work [6], the preconditioning ratio is set to 80%. We set the buffer size to 1 MB, accommodating 128 valid pages for GC. The time window size is configured as 5 ms. The parallel resource configurations in Table III include 2-4-4-4 [35], 8-1-4-4, 8-4-2-2 [6], and 16-4-2-2 [12], where the numbers represent the number of channels, chips per channel, dies per chip, and planes per die, respectively. These configurations are widely adopted in SSD architectures in the past studies. Through these parallelism configurations, we are positioned to investigate the performance of AGC with various SSD organizations.

2) Alternative Schemes: We compare AGC with four state-of-the-art schemes which are implemented on MQSim with a default hotness-aware data separation method. We adopt MQSim [15] as the *baseline*, which uses a conventional GC strategy when the free space falls below a predefined threshold. CachedGC [7] leverages a valid page buffer to store migrated valid pages during GC. It is passively triggered when available space inside an SSD drops below a threshold. Blocks are erased after valid pages are moved to the buffer. CachedGC prioritizes user I/Os when the buffer is full and writes migrated valid pages to flash memory when chips are idle. FFT-GC [12] employs the Fourier transform to predict periods with low I/O intensity and constructs a mathematical model to determine GC based on I/O intensity, read/write ratio, and SSD status. HIPA [30] is a page allocation strategy that dynamically determines allocation methods based on die-level load status, improving read parallelism while maintaining write performance. RDA [35] dynamically adjusts parallelism levels based on historical read request size, distributing writes across parallel resources to achieve high parallelism for large-sized reads while reducing fragmentation for small reads.

3) Tested Metrics: We use the following metrics to validate AGC compared with the alternative schemes. The average read/write response time is a metric that measures SSD read/write performance in terms of individual I/O requests. The GC count reflects the overhead of GC and its impact on SSD lifetime. The write amplification quantifies the excessive write operations incurred by GC activities. This metric reflects the efficiency of the GC mechanism and its impact on SSD endurance. The tail latency signifies the stability of a GC strategy: a low tail latency reflects the performance stability of GC. The **BC_IF** quantifies the performance impact of burst conflict requests between the user and GC I/Os.

B. Performance Evaluation

To improve results' readability, we use the performance results of MQSim as the baseline, the ratios of other schemes' results to that of the baseline are referred to as the normalized value in our experiments.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:46 UTC from IEEE Xplore. Restrictions apply.

TABLE III
SIMULATION PARAMETERS IN MQSIM

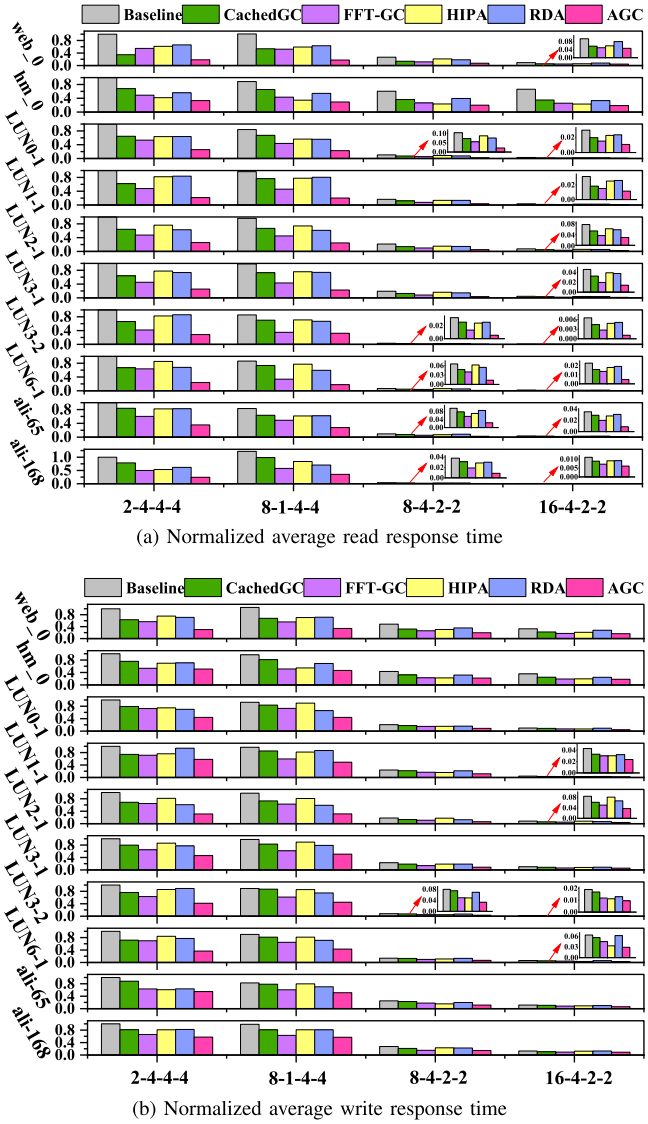
Host Interface	Total Capacity	Flash Interface	Address Mapping	TSU Policy
PCIe 3.0 (NVMe 1.2)	512 GB	ONFI 3.1 (NV-DDR2)	Page level	Sprinkler [24]
GC policy	**GC threshold**	**Over-provisioning**	**Pages per block**	**Page size**
RGA (Default)	0.05	0.07	256	8 KB with 448 B metadata

Parallel Resource Configurations						Flash Latency Parameters			
Config. Type	**Channel Counts**	**Chips /Channel**	**Dies /Chip**	**Planes /Die**	**Blocks /Plane**	**Flash Type**	**Read (μs)**	**Program (μs)**	**Erase (ms)**
2-4-4-4	2	4	4	4	2048	SLC	25	250	2
8-1-4-4	8	1	4	4	2048	MLC	LSB: 50, MSB: 100	LSB: 500, MSB: 1,000	3
8-4-2-2	8	4	2	2	2048	TLC	LSB: 90, CSB: 120, MSB: 180	LSB: 900, CSB: 1,200, MSB: 1,800	4
16-4-2-2	16	4	2	2	1024	QLC	LSB: 100, CSB: 180, MSB: 300, TSB: 450	LSB: 1,000, CSB: 1,800, MSB: 3,000, TSB: 4,500	5

1) Average Response Time: Fig. 7(a) unveils the average read response time of AGC under traces. As the number of parallel units increases, particularly in the 8-4-2-2 and 16-4-2-2 configurations, all schemes show significant improvement in read performance. With all parallelism configurations, AGC achieves the lowest read response time compared to alternative schemes. The superior performance of AGC is attributed to its approach to minimizing conflicts between user and GC I/Os through the hybrid data allocation and burst-and-hotness-aware GC. Compared with baseline, CachedGC, FFT-GC, HIPA, and RDA, AGC maximally improves the read response time by 66.01%, 54.16%, 42.07%, 53.77%, and 55.67% on average, with 81.57%, 75.87%, 66.47%, 78.22%, and 75.57%, respectively.

Fig. 7(b) plots the average write response time. As shown in Fig. 7(b), when the number of parallel units is fixed (*i.e.*, for 2-4-4-4, 8-1-4-4, and 8-4-2-2), a higher number of channels and chips enhances SSD parallelism. AGC delivers the best write performance, while the other schemes exhibit various performance fluctuations contingent upon access patterns in workloads, demonstrating their sensitivity to access patterns and intensity variations in workloads. AGC reduces write response time by up to 69.66%, 57.12%, 51.99%, 63.26%, and 57.4%, compared with baseline, CachedGC, FFT-GC, HIPA, and RDA, respectively, with average improvements of 52.64%, 41.44%, 25.8%, 36.19%, and 39.99%. This is because AGC effectively reduces conflicts between user and GC I/Os through its adaptive GC scheduling and hybrid data allocation strategies.

2) GC Counts and Write Amplification: Fig. 8(a) and 8(b) demonstrate the GC counts and write amplification of AGC. FFT-GC has the highest number of GC counts under workloads since early GC migrates some valid pages that have the potential to be invalid pages. FFT-GC enlarges write amplification and shortens SSD lifetime. The baseline, CachedGC, and HIPA share the same number of GC counts and the same write amplification factor. This is because these schemes do not implement additional data separation optimization. RDA, however, reduces GC counts by addressing the data fragmentation problem that emerges from indiscriminate parallel distribution of consecutive pages. AGC further refines GC efficiency through a data separation strategy that classifies data into four



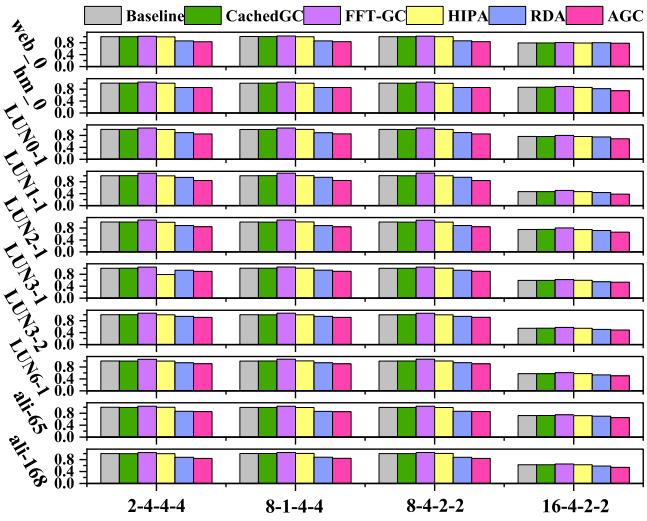
(a) Normalized average read response time

(b) Normalized average write response time

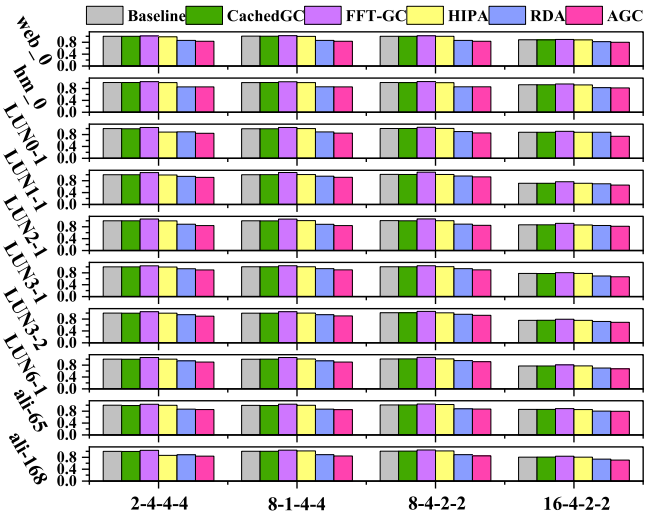
Fig. 7. Normalized read and write response time.

hotness – hot, warm, cool, and cold – based on the requests' update intervals. This method provides efficient GC by grouping data with similar update frequency. Furthermore, when the

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:46 UTC from IEEE Xplore. Restrictions apply.

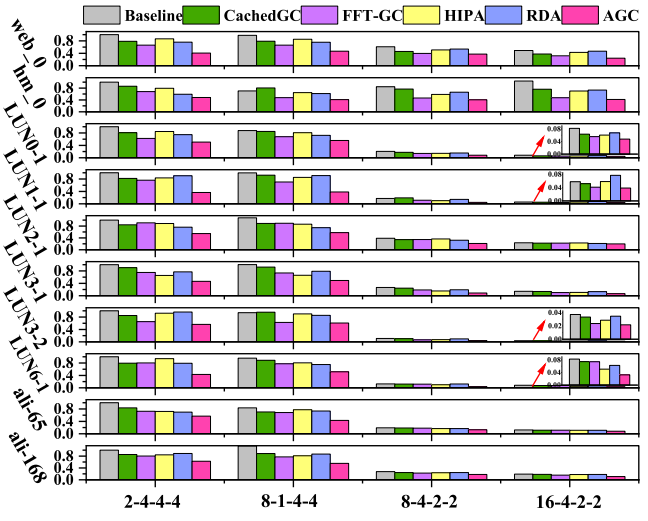


(a) Normalized total GC counts

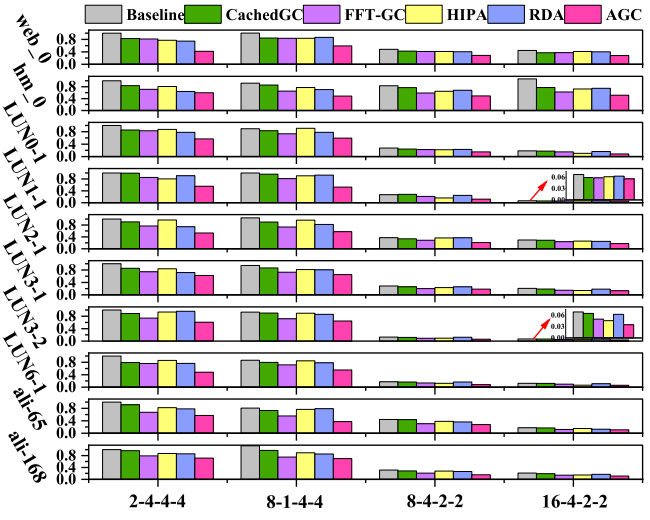


(b) Normalized write amplification

Fig. 8. Normalized GC counts and write amplification.



(a) Normalized P99.99 read tail latency



(b) Normalized P99.99 write tail latency

Fig. 9. Normalized P99.99 read and write tail latency.

flash memory organization is configured as 16-4-2-2, compared to other configurations, the write pressure is more effectively distributed due to the increased number of parallel units, thereby reducing the frequency of garbage collection triggers. AGC trims the number of GC counts by up to 24.31% with an average reduction of 12.04%. AGC also decreases write amplification by up to 21.04%, with an average of 11.26%. In summary, AGC reduces GC overhead and extends SSD lifetime through its hotness-aware data placement.

3) Tail Latency: Fig. 9(a) and 9(b) show the read and write long-tail latency of AGC. As the number of channels and chips in the SSD goes up, all the schemes achieve significant improvements in read and write tail latency under most workloads with the 16-4-2-2 configuration. As illustrated in Fig. 9(a), alternative GC schemes exhibit a higher the 99.99th percentile (P99.99) read latency than AGC. This is because, first, HIPA and RDA fail to mitigate conflicts between user and GC I/Os. As it cannot anticipate future workload access patterns, CachedGC causes user-GC I/O conflicts. FFT-GC exhibits limitations under workloads with irregular access patterns, leading to a high tail latency when prediction errors occur. Second, AGC effectively eliminates conflicts between user and GC I/Os by strategically caching valid pages prior to anticipated burst access periods, thereby reducing tail latency.

AGC achieves the lowest P99.99 write tail latency under workloads, see Fig. 9(b). First, AGC reduces conflicts between user and GC I/Os through its hybrid data allocation and GC strategies. Second, AGC's data separation mechanism based on request update intervals substantially reduces GC counts. Notably, the improvement of write tail latency is less pronounced than that of read tail latency. This is because GC conflicts affect read performance more significantly. However, AGC reduces P99.99 write latency by an average of 42.88%, 37.28%, 24.05%, 29.96%, and 31.95% compared to the baseline, CachedGC, FFT-GC, HIPA, and RDA, respectively, with maximum reductions of 58.08%, 57.21%, 48.43%, 51.36%, and 52.78%.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:46 UTC from IEEE Xplore. Restrictions apply.

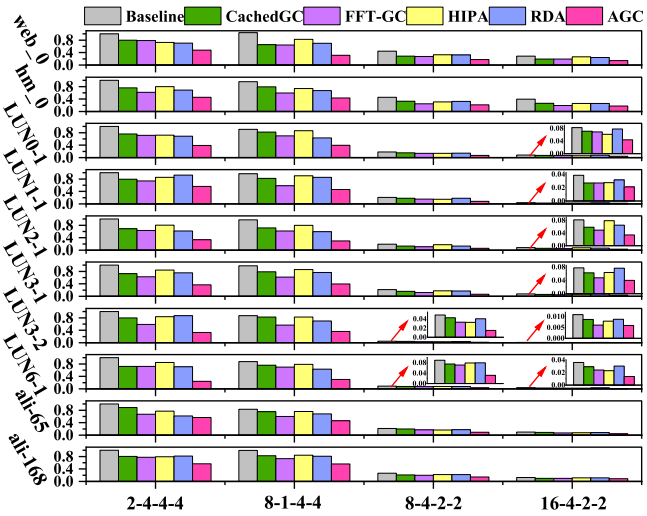


Fig. 10. Normalized BC_IF.

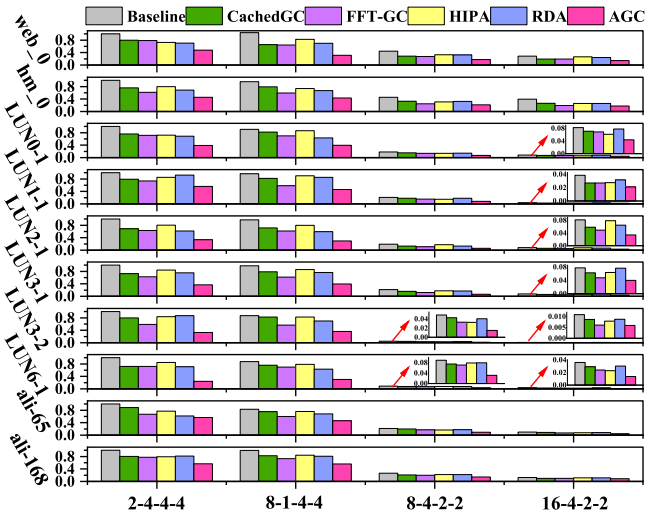


Fig. 11. Impact of valid page buffer size (KB).

4) BC_IF: To study the effectiveness of our proposed AGC in mitigating performance degradation caused by conflicts between user and GC I/Os, we employ metric BC_IF defined in Section III-A. This metric quantifies the performance impact of burst conflict requests, which is a primary factor contributing to SSD performance bottlenecks. Fig. 10 shows the comparison of BC_IF values for various schemes, and the results demonstrate that AGC achieves optimal performance under all parallel configurations. Specifically, compared to baseline, CachedGC, FFT-GC, HIPA, and RDA, AGC reduces the BC_IF value by an average of 56.29%, 44.57%, 34.92%, 45.52%, and 43.85%, respectively, with maximum reductions of 76.06%, 64.71%, 56.61%, 71.80%, and 66.29%. This substantial improvement indicates that our proposed AGC can address the performance cliff issue caused by burst conflict requests. The excellent performance of AGC can be attributed to AGC's hybrid allocation strategy and burst-aware GC scheduling strategy. This enables AGC to reduce conflicts between user and GC I/Os, thereby avoiding the performance impact of burst conflict requests.

C. Extended Experiments

Since the 8-4-2-2 configuration in MQSim achieves the best performance across all tests among configurations with the same number of parallel units (*i.e.*, 2-4-4-4, 8-1-4-4, and 8-4-2-2), we adopt the 8-4-2-2 parallel configuration with MLC flash memory for extended experiments (see Table III).

1) Impact of Valid Page Buffer Size: Fig. 11 shows the impact of valid page buffer size on the overall response time of CachedGC, FFT-GC, and AGC. We find that the average response time decreases with a large-sized buffer capacity because a large-sized buffer can accommodate more valid pages migrated by GC, thereby reducing conflicts between user and GC I/Os. However, the performance improvement diminishes as buffer size increases, which indicates that an excessively large buffer provides diminishing returns. Under web_0, hm_0, and ali-168, the performance improvements of the three schemes are negligible when increasing the buffer size from 1 MB to 2 MB because of the limited number of valid pages migrated by GC in these workloads. As listed in Table III, AGC with a 1-MB buffer sufficiently handles valid page migrations during GC. Similarly, under other workloads, when the buffer size increases from 4 MB to 8 MB, the response time improvement of each scheme is also limited. This suggests that once the buffer reaches a sufficient capacity, further expansion yields only marginal improvements in response time. Furthermore, a large buffer incurs increased implementation costs. Therefore, we recommend that the buffer size should not exceed 4 MB, which can balance SSD performance with design cost.

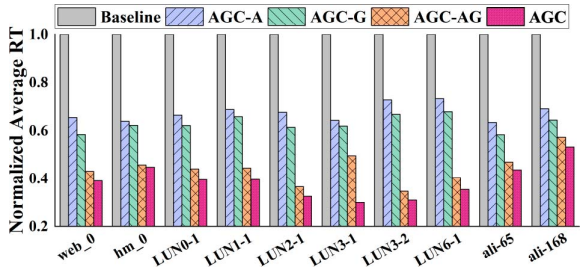
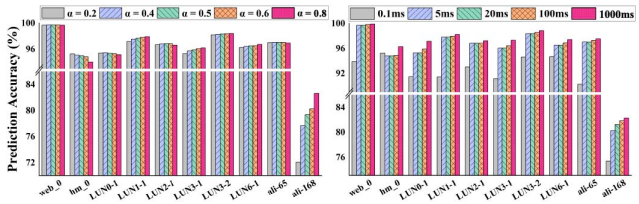


Fig. 12. Contribution of individual module in AGC.

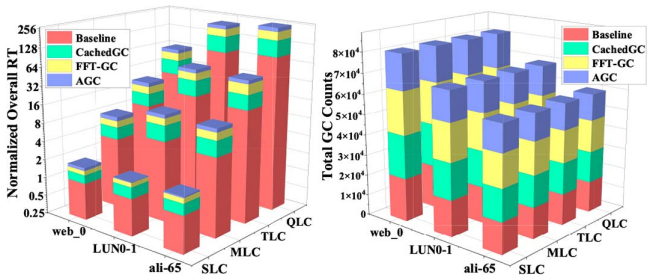
2) Modular Effect Analysis: We decompose AGC into independent modules to evaluate the effectiveness of each module. Fig. 12 presents the impact of the various modules in AGC on the SSD performance. The y-axis (RT) represents the response time, normalized to the baseline. AGC-A means that AGC is configured with the hybrid data allocation strategy without the burst-and-hotness-aware GC strategy. AGC-G focuses on optimizing the GC strategy but lacks the hybrid data allocation strategy, which limits its ability to perform hotness-aware optimizations. AGC-AG integrates both data allocation and GC strategies rather than the hotness-aware data separation strategy. AGC implementation encompasses all functional modules.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:46 UTC from IEEE Xplore. Restrictions apply.



(a) Value of α (b) Time Window Size (ms)

Fig. 13. Impact of parameters on EMA prediction accuracy.



(a) Overall Response Time (b) Total GC Counts

Fig. 14. Impact of NAND flash memory.

In Fig. 12, all AGC variants achieve significant improvements in average response time. Under most workloads, the GC strategy (AGC-G) contributes marginally more to performance enhancement than the allocation strategy (AGC-A), highlighting GC mechanisms in performance optimization. When allocation and GC strategies operate synergistically (AGC-AG), system performance improves substantially, near to the performance of AGC. The performance differential between these configurations is primarily attributed to the data separation module. These findings validate our designs and provide guidance for latency-oriented SSD optimization. In resource-constrained cases, the developers should give priority to the collaborative effect of GC and allocation strategies.

3) Impact of Parameters on EMA Prediction Accuracy: In EMA, the parameter α controls the weighting between the number of historically accessed sectors from the previous four time windows against those from the latest time window when predicting the number of sectors accessed in the subsequent time window.

Accordingly, to study the impact of α on the prediction accuracy, we conduct experiments with five α values: 0.2, 0.4, 0.5, 0.6, and 0.8. Fig. 13(a) shows the impact of five α values on the prediction accuracy. Larger α values generally improve prediction accuracy under most workloads. However, an opposite trend is shown under hm_0 and LUN0-1, where larger α values result in low prediction accuracy. We calculate the average prediction accuracy under all workloads for the five α values. When $\alpha = 0.6$, EMA algorithm delivers the highest average prediction accuracy. While optimizing α for a specific workload could achieve superior performance, we need to obtain access patterns in the workload, which is unavailable in real life. Therefore, we set α to 0.6 in this paper, which enables AGC to capture burst user I/O requests for diverse access patterns.

Furthermore, in section III-A, the time window size significantly influences the prediction accuracy of the EMA algorithm. Fig. 13(b) reveals the correlation between prediction accuracy and various time window sizes. Prediction accuracy is lowest for a time window of 0.1 milliseconds and improves monotonically as the time window size increases. However, excessively large time window sizes obscure differences in workload access patterns within the time window. Our studies confirm that a large-size time window significantly raises the proportion of idle periods from 43.96% at 5 ms to 61.67% at 100 ms. Therefore, for larger time window sizes, actual bursty phases can be misclassified as non-bursty phases. Such misclassification can impair the performance of AGC. We therefore recommend setting time window sizes between a few milliseconds and tens of milliseconds. Given that the VDI workloads used in our work exhibit the average request interval time below 5 milliseconds (see Table I), we set the time window size to 5 milliseconds.

4) Impact of NAND Flash Memory Cell: Fig. 14 demonstrates the impact of different flash cells (SLC, MLC, TLC, and QLC) on performance. As shown in Fig. 14(a), the overall response time increases as flash cells evolve from SLC to QLC, which aligns with the inherent characteristic of higher-density flash cells increasing read and write latency. Under workloads, AGC demonstrates the lowest response time compared to all tested GC schemes, with this performance advantage becoming more pronounced in QLC. Specifically, for SLC, AGC improves overall response time by 63.9%, 49.13%, and 32.31% compared to baseline, CachedGC, and FFT-GC, respectively. For QLC, AGC achieves overall response time improvements of 63.00%, 54.33%, and 42.12%, respectively.

Fig. 14(b) shows the total GC counts. Since we maintain consistent SSD internal activity policies and flash organization across different flash cells, the GC counts remain relatively consistent across various flash cells. The experimental results indicate that AGC effectively reduces GC frequency across all flash cells. Compared to Baseline, CachedGC, and FFT-GC, AGC reduces the number of GC counts by 15.43%, 15.42%, and 18.11%, respectively. These results demonstrate the adaptability and effectiveness of AGC in different flash cells.

5) Impact of Large-capacity SSDs: We study the performance of AGC in cases of 1 TB and 2 TB SSD capacities, see Fig. 15. AGC achieves optimal performance in terms of response time and GC counts. Compared to baseline, CachedGC, FFT-GC, HIPA, and RDA, AGC demonstrates average improvements of 47.83%, 31.94%, 22.67%, 27.37%, and 37.56% in terms of response time, respectively. For GC counts, AGC achieves reductions of 15.81%, 15.76%, 18.87%, 15.65%, and 5.51%, respectively. These performance improvements are primarily attributed to AGC's collaborative optimization of I/O access patterns in workloads, internal GC, and data allocation strategy inside the SSD.

6) Impact of Preconditioning Ratios: Fig. 16 illustrates the average response time of AGC under various preconditioning ratio conditions. The horizontal axis represents different preconditioning ratio values, demonstrating that AGC

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:46 UTC from IEEE Xplore. Restrictions apply.

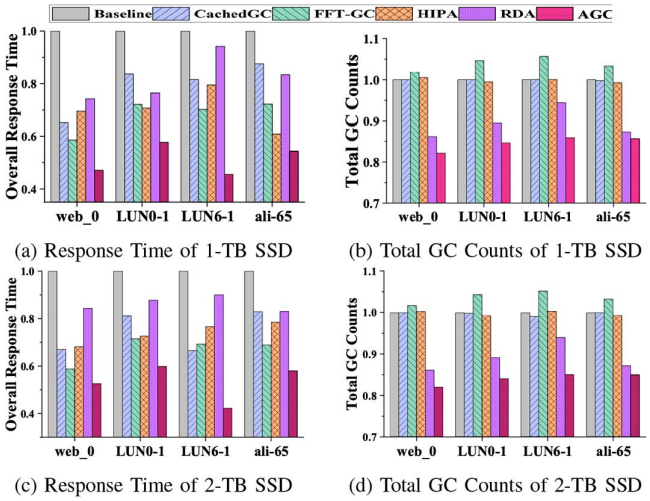


Fig. 15. Impact of large-capacity SSDs.

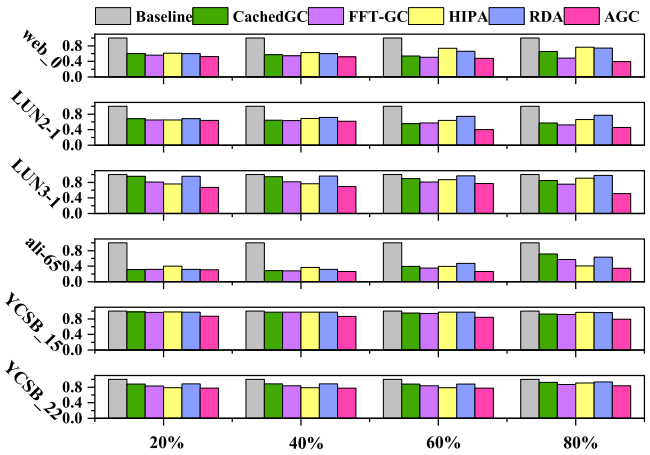


Fig. 16. Impact of preconditioning ratios.

shortens response time performance compared to the alternative schemes. As the preconditioning ratio increases, the number of GC counts within the SSD also increases, which, in turn, amplifies the effectiveness of AGC.

Furthermore, AGC improves – under YCSB [36] – the average response time by 22.25% on average compared to alternative schemes, with a maximum improvement of 65.46%. AGC maintains performance advantages under varying preconditioning ratios and diverse real-world workloads.

VI. Related Work

The performance of SSDs is largely affected by GC processes, which occupies internal channels and chip resources, blocking user I/O requests and leading to high tail latency. Existing studies address this challenge through GC optimization and data allocation strategies.

Prior work focuses on optimizing GC scheduling to lessen conflicts between user and GC I/Os. CachedGC [7] reduces resource contention between user and GC I/Os by buffering valid pages during GC and writing them back to flash memory once the target chip is idle. FFT-GC [12] employs Fourier transforms to identify periodic access patterns in workloads, executing GC operations in the low-I/O-intensity period to minimize user access blocking. However, FFT-GC's preemptive GC results in premature data migration, which increases write amplification and erase operations, thereby reducing SSD lifetime. While these GC scheduling optimization strategies have demonstrated effectiveness in specific scenarios, they exhibit limited adaptability to non-periodic burst I/O workloads, which is primarily considered in our proposed AGC.

In addition, research focuses on optimizing conflicts by lowering GC frequency. DA-GC [8] integrates GC with wear-leveling and dynamically modifies the GC algorithm based on real-time wear-out conditions, thereby reducing SSD wear frequency and GC overhead. Gao et al. [37] minimize GC overhead by aligning physical blocks during GC, optimizing utilization of multiple planes. However, these studies primarily focus on improving SSD internal activities without adequately considering the dynamic access patterns in workloads.

Several work optimizes GC through the data separation. WARM [13] exploits heterogeneity in data write frequency, selectively relaxing retention time requirements for frequently written data in NAND flash, thereby reducing unnecessary refresh operations. SUP-GC [14] introduces a simplified GC strategy that inherently classifies newly written data as hot and GC-migrated data as cold, achieving substantial performance improvements through this intuitive partitioning approach. However, these existing approaches lack the adaptive mechanism that dynamically adjusts to changes of access patterns in workloads, whereas AGC addresses this limitation through fine-grained hotness classification and sequence-number-based update intervals.

Additionally, research addresses access conflicts between user and GC I/Os inside an SSD through refined data allocation strategies. Tavakkol et al. [22] investigate the impact of varying freedom degrees in the dynamic resource allocation policies, and demonstrate that the dynamic allocation significantly reduces conflicts between user and GC I/Os by optimally distributing user I/O requests across parallel resources. RDA [35] dynamically adjusts the parallel distribution of consecutive pages based on historical read request patterns, effectively preserving parallelism for read operations while mitigating performance degradation caused by redundant parallel allocation. Unfortunately, these approaches rarely address the delicate balance between maximizing SSD internal parallelism and minimizing user access conflicts – a critical challenge that persists in modern SSD designs.

VII. Conclusion

In this study, we developed AGC – an adaptive workload burst-aware GC mechanism that minimizes access conflicts between user and GC I/Os in SSDs. By predicting burst access patterns in subsequent time windows, AGC migrates valid pages to the buffer during GC, preventing user I/O blockage. Based on the status of the underlying flash memory, we implement a hybrid allocation strategy to minimize GC-induced user

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:46 UTC from IEEE Xplore. Restrictions apply.

write I/O blocking while preserving read parallelism. AGC classifies data into four hotness levels – hot, warm, cool, and cold – and incorporates a hotness-aware block selection policy that prioritizes blocks with a higher hotness level, thereby reducing valid page data migration to optimize write amplification. We conducted extensive experiments to evaluate AGC using real-world traces. AGC shortens read and write response time by up to 81.57% and 69.66%, respectively, with average reductions of 54.34% and 39.21%. Additionally, AGC brings down GC counts by up to 24.31% with an average of 12.04%, extending SSD lifetime.

REFERENCES

[1] Y. Jin, H.-W. Tseng, Y. Papakonstantinou, and S. Swanson, "KAML: A flexible, high-performance key-value SSD," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA),* Piscataway, NJ, USA: IEEE Press, 2017, pp. 373–384.

[2] J. Wan, W. Wu, L. Zhan, Q. Yang, X. Qu, and C. Xie, "DEFT-cache: A cost-effective and highly reliable SSD cache for RAID storage," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, Piscataway, NJ, USA: IEEE Press, 2017, pp. 102–111.

[3] D. Li, H. Sun, and X. Qin, "iCache: An intelligent cache allocation strategy for multi-tenant in high-performance solid state disks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 44, no. 5, pp. 1690–1703, May 2025.

[4] J. Colgrove et al., "Purity: Building fast, highly-available enterprise flash storage from commodity components," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1683–1694.

[5] S. Yu et al., "Adaptive dram cache division for computational solid-state drives," in *Proc. Des., Automat. Test Europe Conf. Exhib. (DATE)*, Piscataway, NJ, USA: IEEE Press, 2024, pp. 1–6.

[6] H. Sun, H. Tong, Y. Yue, and X. Qin, "LAC: A workload intensity-aware caching scheme for high-performance SSDs," *IEEE Trans. Comput.*, vol. 73, no. 7, pp. 1738–1752, Jul. 2024.

[7] C. Gao et al., "Exploiting chip idleness for minimizing garbage collection—Induced chip access conflict on ssds," *ACM Trans. Des. Automat. Electron. Syst. (TODAES)*, vol. 23, no. 2, pp. 1–29, 2017.

[8] Z. Chen and Y. Zhao, "DA-GC: A dynamic adjustment garbage collection method considering wear-leveling for SSD," in *Proc. Great Lakes Symp. VLSI*, 2020, pp. 475–480.

[9] 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.

[10] J. Lee, Y. Kim, G. M. Shipman, S. Oral, and J. Kim, "Preemptible i/o scheduling of garbage collection for solid state drives," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 2, pp. 247–260, Feb. 2013.

[11] G. Wu and X. He, "Reducing ssd read latency via nand flash program and erase suspension," in *Proc. Fast*, 2012, vol. 12, p. 10.

[12] Z. Sha et al., "Low I/O intensity-aware partial gc scheduling to reduce long-tail latency in ssds," *ACM Trans. Archit. Code Optim. (TACO)*, vol. 18, no. 4, pp. 1–25, 2021.

[13] Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu, "WARM: Improving NAND flash memory lifetime with write-hotness aware retention management," in *Proc. 31st Symp. Mass Storage Syst. Technol. (MSST)*, Piscataway, NJ, USA: IEEE Press, 2015, pp. 1–14.

[14] K. Wang, H. Tan, and K. Li, "Simplicity as the ultimate principle: The art of garbage collection management in ssds inspired by natural data behavior," *ACM Trans. Storage*, vol. 21, no. 4, pp. 36:1–36:34, Nov. 2025.

[15] 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)*, 2018, pp. 49–66.

[16] H. Sun, C. Sun, H. Tong, Y. Yue, and X. Qin, "A machine learning-empowered cache management scheme for high-performance SSDs," *IEEE Trans. Comput.*, vol. 73, no. 8, pp. 2066–2080, Aug. 2024.

[17] 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.

[18] H. Sun, Q. Cui, J. Huang, and X. Qin, "NCache: A machine-learning cache management scheme for computational SSDs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 6, pp. 1810–1823, Jun. 2023.

[19] M. Murugan and D. H. C. Du, "Rejuvenator: A static wear leveling algorithm for nand flash memory with minimized overhead," in *Proc. IEEE 27th Symp. Mass Storage Syst. Technol. (MSST)*, Piscataway, NJ, USA: IEEE Press, 2011, pp. 1–12.

[20] C. Ji, L.-P. Chang, L. Shi, C. Wu, Q. Li, and C. J. Xue, "An empirical study of {File-System} fragmentation in mobile storage systems," in *Proc. 8th USENIX Workshop Hot Topics Storage File Syst. (HotStorage 16)*, 2016, pp. 1–5.

[21] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and C. Ren, "Exploring and exploiting the multilevel parallelism inside ssds for improved performance and endurance," *IEEE Trans. Comput.*, vol. 62, no. 6, pp. 1141–1155, Jun. 2013.

[22] A. Tavakkol, P. Mehrvarzy, M. Arjomand, and H. Sarbazi-Azad, "Performance evaluation of dynamic page allocation strategies in ssds," *ACM Trans. Model. Perform. Eval. Comput. Syst. (TOMPECS)*, vol. 1, no. 2, pp. 1–33, 2016.

[23] C. Gao et al., "Exploiting parallelism for access conflict minimization in flash-based solid state drives," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 168–181, Jan. 2018.

[24] 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)*, Piscataway, NJ, USA: IEEE Press, 2014, pp. 524–535.

[25] 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.

[26] T. Fujimori and S. Nomura, "Booster: Rethinking the erase operation of low-latency ssds to achieve high throughput and less long latency," in *Proc. 16th ACM Int. Conf. Syst. Storage*, 2023, pp. 94–104.

[27] "Microsoft block i/o traces." 2017. [Online], Available: http://iotta.snia.org/traces/block-io/388.

[28] 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.

[29] "Alibaba cloud traces." GitHub. 2025. [Online]. Available: https://github.com/alibaba/block-traces

[30] H. Sun, X. Cheng, C. Zhang, Y. Yue, and X. Qin, "HIPA: A hybrid load balancing method in SSDs for improved parallelism performance," *J. Syst. Archit.*, vol. 131, Oct. 2022. Art. no. 102705.

[31] B. Mao, S. Wu, and L. Duan, "Improving the ssd performance by exploiting request characteristics and internal parallelism," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 2, pp. 472–484, Feb. 2018.

[32] F. Klinker, "Exponential moving average versus moving exponential average," *Mathematische Semesterberichte*, vol. 58, no. 1, pp. 97–107, 2011.

[33] Y. Li, P. P. C. Lee, and J. C. S. Lui, "Stochastic modeling of large-scale solid-state storage systems: Analysis, design tradeoffs and optimization," *in Proc. ACM SIGMETRICS/Int. Conf. Meas. Model. Comput. Syst.*, 2013, pp. 179–190.

[34] Y. Zhou, F. Wang, Z. Shi, and D. Feng, "The static allocation is not a static: Optimizing SSD address allocation through boosting static policy," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 8, pp. 1373–1386, May 2024.

[35] S. Pang, Y. Deng, Z. Wu, G. Zhang, J. Li, and X. Qin, "RDA: A read-request driven adaptive allocation scheme for improving SSD performance," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 44, no. 2, pp. 416–429, Feb. 2025.

[36] G. Yadgar, M. Gabel, S. Jaffer, and B. Schroeder, "Ssd-based workload characteristics and their performance implications," *ACM Trans. Storage (TOS)*, vol. 17, no. 1, pp. 1–26, 2021.

[37] C. Gao, L. Shi, K. Liu, C. J. Xue, J. Yang, and Y. Zhang, "Boosting the performance of ssds via fully exploiting the plane level parallelism," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 9, pp. 2185–2200, Sep. 2020.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:46 UTC from IEEE Xplore. Restrictions apply.



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



Haisheng Ding is currently working toward the M.S. degree with Anhui University. His research interests include SSDs and garbage collection.



Haoqiang Tong received the M.S. degree from Anhui University. His research interests include SSDs, intelligent cache management, and storage systems.



Xiang Cheng received the M.S. degree from Anhui University. His research interests include SSDs and resource allocation.



Honggang Chai received the B.S. and M.S. degrees in computer science from Huazhong University of Science and Technology, in 2004 and 2007, respectively. He brought more than 18 years of storage industry experience in SSD controllers. He was a firmware R&D Engineer with VIA Technologies. He is currently a Co-Founder/Technical Director with Wuhan HOSTOR Information Technology Company Ltd. His research interests include solid-state disks, flash translation layer, cache management, and flash memory endurance.



Xiao Qin (Senior Member, IEEE) received the B.S. and M.S. degrees in computer science from Huazhong University of Science and Technology, in 1992 and 1999, respectively. He received 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 a NSF CAREER Award in 2009. His research interests include parallel and distributed systems, storage systems, and performance evaluation.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:46 UTC from IEEE Xplore. Restrictions apply.