ENGINEERING Information Technology & Electronic Engineering
https://www.academax.com/EITEE; https://www.engineering.org.cn
https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=11344065
E-mail: eng.itee@zju.edu.cn

Engineering Journals

Available on IEEE Xplore®

Research Article
https://doi.org/10.1631/ENG.ITEE.2025.0152

GC bypass: decoupling GC from the flash translation layer to eliminate GC-induced long-tail latency inside SSD

Shiqiang NIE[1], Jie NIU[1], Yingzhao SHAO[2], Xiaobo LI[2], Mingming ZHANG[2], Weiguo WU[1✉]

[1]*School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China*
[2]*Intelligent Computing Center, China Academy of Space Technology, Xi'an 710000, China*

Abstract: NAND flash-based solid-state drives (SSDs) have been adopted by many data centers due to their high performance and low power consumption. However, the physical characteristics of the underlying flash memory necessitate garbage collection (GC) operations. Valid page migration during GC contributes significantly to latency overhead while competing for flash channel bandwidth and controller resources with user I/O requests through shared physical paths, leading to path conflicts and elevated long-tail latency. The existing Venice scheme introduces a low-cost interconnected network with path reservation mechanisms to provide substantial path diversity for SSDs. Nevertheless, its fair scheduling policy lacks priority differentiation between I/O and GC requests. In this paper, we propose GC bypass, which leverages Venice's path diversity while enforcing GC request transmission through dedicated controllers. GC bypass decomposes GC requests into sub-requests and assigns low priority to valid page writes, enabling high-priority operations including user I/O, valid page reads, and block erases, to preempt paths reserved by low-priority requests. Valid pages failing to secure reserved paths are temporarily buffered for retry. Experimental results demonstrate that GC bypass reduces the 99.99^{th} percentile long-tail latency by up to 25% compared to Venice. GC bypass effectively mitigates interference between critical I/O operations and background maintenance tasks while maintaining the architectural benefits of path diversity.

Key words: Solid-state drive (SSD); NAND flash; Garbage collection (GC); Interconnected network; Flash channel

1 Introduction

With data-intensive workloads, such as artificial intelligence and high-performance computing, increasingly dominating data centers, NAND flash-based solid-state drives (SSDs) have widely replaced traditional hard disk drives (HDDs) as the primary storage components in data centers. SSDs exhibit significant advantages in multiple aspects due to their superior bandwidth, performance, reliability, and energy efficiency. In data center environments, SSDs face numerous challenges, one of the most critical being the long-tail latency caused by garbage collection (GC). GC not only occupies storage cells within flash chips but also directly competes with user I/O requests on the physical channel bus, leading to temporary suspension of I/O requests and creating a significant performance bottleneck. Although SSDs are designed with multiple independent physical channels connecting flash chips, theoretically enabling parallel processing of multiple requests, when valid page migration operations and user I/O access the same channel or chip simultaneously, SSDs are forced to serialize concurrent requests to serial execution, a phenomenon known as "path conflict." Path conflicts undermine the parallel advantages of SSD multi-channel architectures and lead to two key performance issues. When GC operations preempt channel resources, interrupted user I/O may experience millisecond-level waiting delays. In sustained write scenarios or when SSDs approach full capacity, frequently triggered GC prolongs the queuing time of I/O requests, resulting in long-tail latency. Additionally, since GC typically employs a threshold-triggered mechanism, its execution causes intermittent I/O performance fluctuations.

To mitigate the long-tail latency caused by GC, researchers focus on optimizing GC firmware algorithms and improving scheduling schemes between GC and I/O. However, firmware-based GC optimization still has limitations: it can operate only within the constraints of the existing hardware architecture and cannot fundamentally alter the allocation and

✉ Weiguo WU, wgwu@xjtu.edu.cn
Weiguo WU, https://orcid.org/0009-0000-8298-0572
CLC number: TP333.93

Received: Nov. 22, 2025; Revision accepted: Jan. 29, 2026;
Crosschecked: Feb. 3, 2026

© The Authors 2026. Published by Zhejiang University Press Co., Ltd. This is an open access article distributed under the terms of the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0/)

utilization of hardware resources. Moreover, the effectiveness of firmware optimization largely depends on the specific implementation and the quality of the optimization algorithms, which may vary across different SSD manufacturers and models, thereby increasing the difficulty and uncertainty of optimization. Existing studies have proposed various optimization techniques at the logical scheduling level, such as dynamically adjusting the priority weights of GC and user I/O (Lee et al., 2013; Yan et al., 2017; Mao et al., 2018), performing fine-grained GC operations during I/O request intervals (Kang et al., 2017; Paik et al., 2018; Sha et al., 2021), or exploiting intra-flash parallelism by parallelizing GC tasks (Gao et al., 2017, 2020b). While these schemes can partially alleviate resource contention under traditional multi-channel shared bus architectures, they remain constrained by the rigid structure of physical channels. When GC operations and user I/O requests are forced to share the same channel due to path conflicts, logical-layer scheduling optimizations become ineffective due to physical transmission bottlenecks. For instance, even if the system schedules GC execution during I/O off-peak periods, multiple GC requests accessing flash chips on the same channel will still incur delays due to path serialization.

This contradiction highlights the limitations of GC scheduling schemes in traditional SSD architectures, which cannot overcome the inherent bottleneck of hardware-level path conflicts. In recent years, researchers have sought to mitigate path conflicts between I/O operations by redesigning the organizational structure of flash channels and flash chips. For instance, pSSD (Kim et al., 2022) improves data transfer efficiency by grouping communication interfaces to increase the effective bandwidth of flash channels. Meanwhile, pnSSD (Kim et al., 2022) introduces vertical bus channels, enabling interconnections among flash chips within the same row or column, thereby providing more path options and partially alleviating path conflicts. The existing Venice (Nadig et al., 2023) scheme further addresses this issue by incorporating a low-cost interconnected network and path reservation based on a fully adaptive routing algorithm, offering rich path diversity that effectively mitigates path conflicts among I/O requests. Venice provides inherent architectural advantages for optimizing GC scheduling schemes, allowing for differentiated scheduling policies tailored to different types of requests. However, in Venice, GC operations share the same path resources with user I/O requests. Without proper scheduling, GC may occupy critical path resources, leading to increased latency for I/O requests. Therefore, we propose to leverage Venice's design principles to optimize GC scheduling schemes. We extend the Venice architecture, shifting its focus from resolving merely user I/O conflicts to enabling coordinated scheduling between GC and I/O operations. To address the aforementioned issues, we propose a novel scheme, GC bypass. GC bypass confines GC requests to a specific controller for transmission and decomposes them into sub-requests. Specifically, valid page write requests are assigned low priority, allowing high-priority requests, including user I/O, valid page writes, and block erases, to preempt the reserved paths of low-priority requests. Valid page write requests that fail to secure conflict-free reserved paths are redirected to a valid page buffer for retry attempts, while obsolete blocks can still be promptly erased to free up available space. GC bypass prevents GC operations from occupying critical path resources, thereby mitigating their interference with user I/O. Meanwhile, GC bypass ensures timely GC through prioritized block erases, maintaining storage availability.

Overall, this paper makes the following contributions:

1. We conduct the preliminary experiment to demonstrate that the Venice scheme fails to fully mitigate GC's impact on long-tail latency of SSDs, revealing that merely expanding physical paths without adding actual storage nodes in Venice's interconnected network proves ineffective for long-tail latency reduction.

2. We propose GC bypass, which introduces priority differentiation for GC sub-requests based on Venice's system, allowing high-priority I/O requests, valid page reads, and block erases to preempt paths reserved by low-priority valid page writes. GC bypass achieves entry isolation between I/O and GC requests through a dedicated GC controller, while maintaining a valid page buffer for temporarily storing write requests awaiting path reservation retries.

3. We conduct a series of experiments and validate the effectiveness of our scheme. Experimental results indicate that GC bypass reduces the 99.99^{th} percentile long-tail latency by up to 25% compared to Venice.

2 Background and research motivation

2.1 SSD architecture and FTL

The flash translation layer (FTL), serving as the critical firmware in SSDs, acts as a bridge connecting standard storage interfaces with underlying NAND flash memory. It not only conceals flash-specific complexities, such as out-of-place updates and block-level erase requirements, but also provides disk-like interfaces to upper-layer file systems and databases. The core functionalities of FTL encompass address translation, GC, and wear leveling, ensuring efficient and durable SSD operation. Specifically, address translation forms the foundation of FTL. By maintaining an address mapping table that records the physical storage location corresponding to each logical page, FTL effectively addresses the need for redirection of flash writes, where multiple writes to the same logical address are spread across different physical pages, thus alleviating the inherent limitation that flash memory is erased before it is written. As an SSD operates, invalid pages gradually scatter across used flash blocks and accumulate over time. GC reclaims space by identifying and erasing blocks containing substantial invalid data. Wear leveling strives to evenly distribute erase operations, preventing flash blocks from prematurely reaching their program/erase (P/E) cycle limits and thus extending the SSD's lifespan. Moreover, given that modern SSDs often integrate multiple NAND flash chips, FTL design incorporates considerations for parallelism and load balancing to maximize the utilization of chip-, die-, and even plane-level parallel processing capabilities. In a typical SSD, the controller interfaces with multiple NAND chips via a multi-channel, shared-bus interconnect. Each channel serves a group of chips on a shared bus, and the per-channel flash controller translates page-sized requests from the FTL into the corresponding low-level flash I/O command sequences and dispatches them to the chips.

While the multi-channel design enhances data transfer parallelism, it also introduces path contention issues. Since flash chips connected to the same flash controller share a common channel/path, if the path is occupied by one I/O request, other I/O requests must wait until the path becomes available. This path-sharing mechanism leads to path conflicts, constraining the SSD's parallel processing capability and potentially causing I/O request delays, particularly under high-concurrency workloads.

2.2 Network SSD

SSDs employ a multi-channel shared bus architecture for communication between the SSD controller and NAND flash memory chips, where the controller connects to flash chips through multiple channels. Consequently, each flash chip maintains only one communication path with the controller, and multiple flash chips on the same channel must share this single path. This design creates resource contention when multiple I/O requests access flash chips on the same channel, significantly limiting SSD parallelism. In response, researchers have proposed various methods to enhance internal parallelism, primarily focusing on increasing bandwidth and providing path diversity:

1. Packetized SSD (pSSD) (Kim et al., 2022). pSSD represents a bandwidth-enhancement approach through packetized communication interfaces. By replacing traditional dedicated control signals with packet-based communication between flash controllers and chips, pSSD approximately doubles the effective channel bandwidth without requiring additional signals or increased signaling rates.

2. Packetized-network SSD (pnSSD) (Kim et al., 2022). Similarly, employing packetized interfaces, pnSSD improves bandwidth while providing path diversity through horizontal and vertical bus channels. Maintaining equivalent per-chip bandwidth to baseline SSDs, pnSSD introduces vertical channels enabling direct interconnections among flash chips within the same row or column. This architecture establishes direct chip-to-chip communication paths and multiple controller-to-chip routes. However, both pSSD and pnSSD incur substantial costs due to significant NAND flash chip modifications, with non-negligible area overhead from packetization circuitry per chip.

3. Network-on-SSD (NoSSD) (Tavakkol et al., 2013). NoSSD replaces the conventional shared bus architecture with a two-dimensional (2D) mesh network interconnecting flash chips. Despite this innovation, NoSSD's integration of buffered routers within flash chips introduces considerable area and cost penalties. Furthermore, its simple deterministic routing algorithm, dimension-order routing, cannot adaptively utilize multiple idle paths between controllers and target chips.

4. Venice (Nadig et al., 2023). Venice proposes a cost-effective flash interconnection solution by introducing a low-cost network between the SSD controller and flash chips to resolve path conflicts through diversity. This architecture incorporates three key innovations: simple router chips adjacent to flash chips forming flash nodes without modifying flash designs, bidirectional network links, and conflict-free path reservation before data transmission. Venice's fully adaptive routing algorithm ensures both path diversity and conflict avoidance. The framework's primary advantage lies in its cost efficiency and rich path selection capability, effectively reducing contention and improving I/O parallelism without altering existing flash chip designs. Compared to pSSD, pnSSD, and NoSSD, Venice delivers superior I/O performance, representing the state-of-the-art in SSD path diversity schemes.

2.3 Garbage collection

GC is a critical operation in SSDs designed to reclaim space occupied by invalid data. Data read/write operations occur at page granularity while erases are performed at the block level in SSDs. During data updates, SSDs employ an out-place update mechanism where new data are written to free pages while the original pages are marked invalid rather than being overwritten directly. As SSDs continue operating, these invalid pages progressively accumulate and occupy valuable storage capacity, necessitating periodic GC operations to maintain sufficient free space for subsequent write requests. The standard GC procedure involves three sequential phases: initially the FTL selects candidate victim blocks based on comprehensive evaluation criteria including the valid page ratio, erase count, and retention time through specific GC algorithms; subsequently remaining valid pages are migrated from victim blocks to designated target blocks via the flash controller, potentially incorporating data classification and reorganization schemes during this process; finally the prepared victim blocks are erased to restore them to usable state. Note that when the flash controller is busy with GC, incoming I/O requests are queued until the controller becomes available.

Contemporary research efforts have concentrated on optimizing various aspects of the GC workflow. The determination of optimal GC triggering timing represents a critical research direction, as premature GC initiation induces excessive reclamation operations that waste flash resources and degrade I/O performance, whereas delayed execution leads to acute space shortages and severe I/O latency spikes during intensive GC periods. Victim block selection methodologies constitute another focal point, where advanced algorithms evaluate multiple dimensions, including invalid page concentration for migration minimization, data temperature for wear leveling optimization, and erase cycle distribution for endurance enhancement. Valid page migration schemes have evolved to incorporate intelligent data placement techniques that leverage access patterns and temporal locality, transforming random write sequences into sequential layouts that promote future invalid page clustering. Furthermore, researchers have investigated granular GC execution models that decompose monolithic collection procedures into interruptible stages, enabling intermittent I/O request servicing to alleviate path contention between background maintenance and foreground operations. These multifaceted optimization approaches collectively address the inherent trade-offs among space reclamation efficiency, write amplification reduction, and quality-of-service maintenance in modern SSDs.

2.4 Motivation

During GC operations, flash controller resources may be occupied, resulting in delays in I/O requests. Specifically,

unless specially designed greedy internal migration schemes are employed, the GC process tends to monopolize channel resources, forcing all page-level I/O requests on that channel into a waiting state. Since SSDs perform read/write operations at the page level, upper-layer I/O requests are decomposed into multiple sub-requests (transactions) and distributed across different channels and chips for execution. When a particular channel becomes overloaded, the transactions allocated to that channel will enter a busy-wait state until resources are released. Particularly, in scenarios with constrained storage capacity, the increased frequency of GC triggering persistently exacerbates channel congestion, preventing accumulated transactions from timely completion. This blocking effect propagates upward along the bus, causing continuous deterioration in upper-layer request response latency and ultimately forming a vicious cycle of I/O request accumulation that results in long-tail latency, severely compromising the system's real-time responsiveness. In the traditional architecture of SSDs, the exclusive access nature of the data bus creates contention and waiting among chips. When one chip occupies the data bus for transmission, other chips requiring bus access are forced into a waiting state. This phenomenon becomes particularly pronounced during GC operations, not only interrupting I/O services on the current chip but also impairing the I/O access capability of other chips on the same channel. The page migration in GC temporarily occupies channel resources, increasing the waiting time for target chip requests. Frequent GC triggering further suppresses the I/O service capability of other chips, making it challenging to fully exploit the parallel advantages of multi-channel architectures.

The latency of read/write operations in SSDs primarily depends on their internal architecture and workflow. As illustrated in Fig. 1, the process can be divided into three main phases:

1. Controller processing phase. The host sends requests through the interface; the FTL parses commands and schedules the FTL to perform logical-to-physical address translation, and then dispatches operations to channels; each channel links the flash controller to one or more NAND chips (corresponding to steps 1–3 in Fig. 1). This phase contributes relatively minor time overhead.

2. Data transfer phase. Data are transmitted from the flash controller through channels to target idle chips (step 4 in Fig. 1).

3. On-chip operation phase. Physical operations on NAND memory cells are performed by the chips themselves (step 5 in Fig. 1). NAND flash operations include read, program (write), and erase operations.

Among these, write and erase operations exhibit significantly higher latency due to their involvement in charge movement. While read operations are faster than writes, they remain considerably slower compared to DRAM. Typical NAND read latencies range in tens of microseconds, writes in hundreds of microseconds, and erases reach millisecond levels. The physical characteristics of NAND flash result in operation latencies within the chip that are substantially higher than the electrical signal transmission time on channels, making on-chip operations the dominant contributor to overall read/write latency. Although channel-level parallelism can improve throughput via multiple channels, the response time of individual requests remains constrained by single-chip operations. While internal dies or planes within chips support parallel operations, physical operations within the same die must still proceed sequentially, creating a fundamental performance bottleneck.

Existing GC scheduling schemes demonstrate limited performance optimization in conventional SSD architectures. Venice introduces an innovative flash chip interconnected network that provides substantial path diversity, effectively mitigating path conflicts among I/O requests and establishing a new hardware foundation for GC scheduling optimization. Venice's path reservation mechanism treats all requests equally, including both GC operations and host I/O requests. Once reserved, these requests can transmit concurrently on their respective paths without mutual interference. While Venice exhibits excellent performance in fair scheduling scenarios, its lack of priority awareness fails to completely resolve resource competition between GC and I/O operations. GC operations are typically executed in batches when available space becomes insufficient or during intensive write operations. Venice's uniform first-come-first-served scheduling policy for all request types leads to concentrated path occupation by numerous GC requests within short time intervals under such conditions. When multiple GC requests simultaneously reserve paths, they may exhaust the interconnected network's available paths, forcing subsequent I/O requests to wait. Consequently, user I/O requests, particularly latency-sensitive read

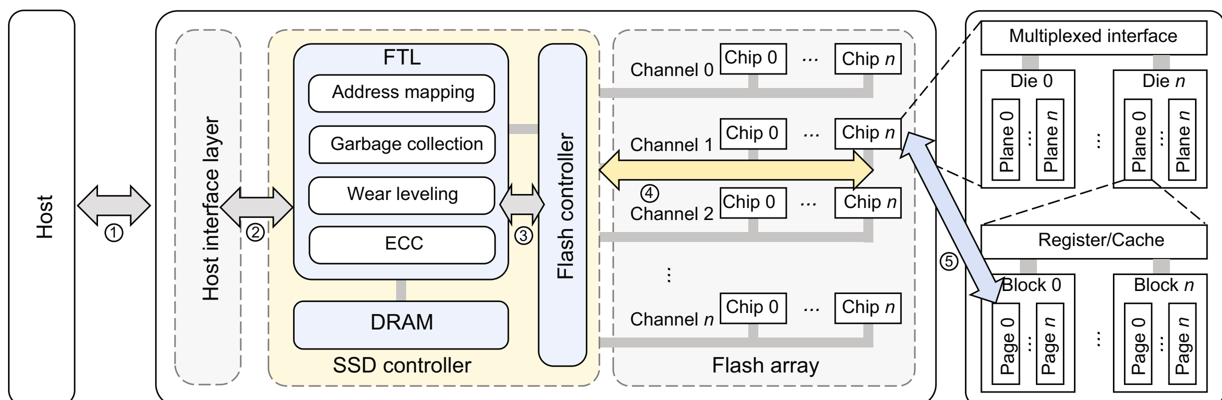

Fig. 1 Workflow of read/write operations in SSDs

operations, experience long-tail latency due to path occupation by GC requests. Furthermore, Venice's treatment of GC requests as ordinary I/O requests in path competition results in spatially random conflict hotspots for both operation types across the interconnected network. The scheduling scheme lacks fine-grained priority differentiation for GC requests, allowing numerous low-value migration requests to compete with high-priority I/O operations for optimal paths. Our preliminary experiment investigates GC's impact on long-tail latency under Venice. Comparative analysis of request response time cumulative distribution between GC-disabled and normal operation scenarios reveals that Venice fails to effectively alleviate GC-induced long-tail latency in SSDs, as demonstrated in Fig. 2. The experimental results demonstrate that Venice fails to effectively alleviate GC-induced long-tail latency in SSDs.

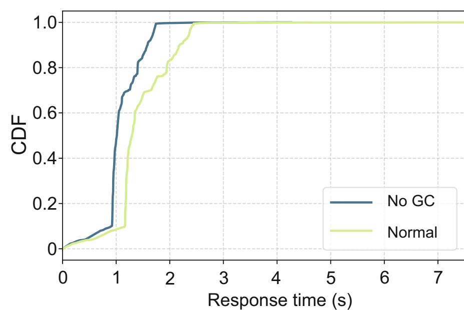

Fig. 2 Cumulative distribution function (CDF) of response time in workload HM_1

The parallelism of conventional SSDs is fundamentally constrained by the number of available channels. We have conducted a preliminary experiment to investigate whether Venice's interconnected network, which expands physical paths without actually adding storage nodes, could effectively mitigate long-tail latency in SSDs. Our experimental setup maintained the original number of functional flash chips while adding "chips" solely for path interconnection purposes, incapable of actual data storage. As shown in Fig. 3, the long-tail latency remained at comparable levels despite the increased number of network entry points. This phenomenon stems from Venice's inability to specifically address GC-I/O contention. Although requests gain additional entry points into the interconnected network, the underlying scheduling mechanism remains unoptimized, and the number of usable chips stays unchanged, resulting in marginal performance improvement. The persistent long-tail latency (still far from ideal levels) under Venice can be partially attributed to the lack of priority differentiation for GC requests. We therefore propose to dedicate one existing channel exclusively for GC requests, physically separating the entry paths for I/O and GC operations. This approach enables co-optimization at both hardware and scheduling levels, creating distinct pathways that prevent resource competition between critical I/O operations and GC tasks.

3 Design

3.1 Design overview

Based on the Venice architecture, GC bypass incorporates the design as illustrated in Fig. 4. To achieve path entry isolation between GC and I/O operations, GC bypass introduces the novel concept of a GC controller; all GC requests exclusively enter the interconnected network through this dedicated controller, while user I/O requests access the network via

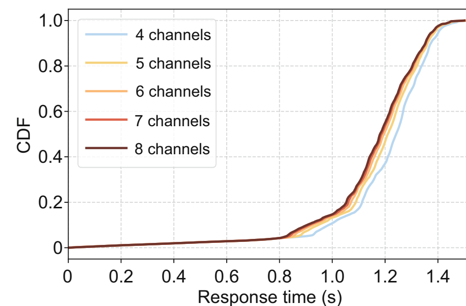

Fig. 3 Cumulative distribution function (CDF) of response time under more channels

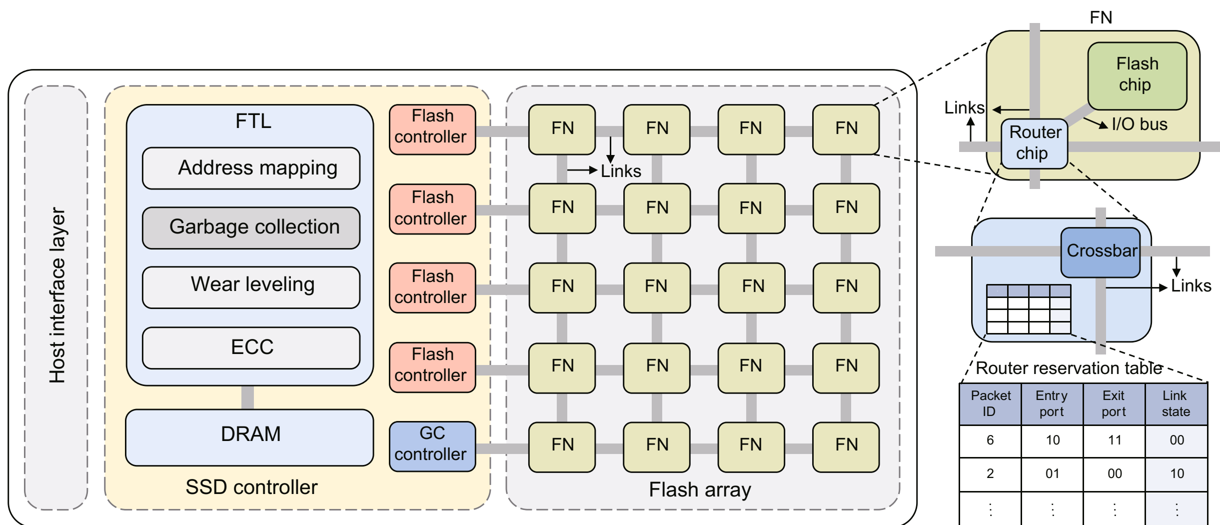

Fig. 4 Overview of GC bypass (FN: flash node)

conventional flash controllers. GC bypass leverages Venice's existing scout packet structure by repurposing a reserved bit in the tail flit as a priority flag. High-priority requests can preempt paths reserved by lower-priority requests. Additionally, a 1-bit marker in both the header and tail flits' type field distinguishes between scout packets originating from the GC controller versus the flash controller. Normal I/O requests transmitted through standard flash controllers are marked as high-priority ones. In the scout packets sent by the GC controller, read requests for valid pages during GC are set as high priority to ensure their rapid completion, and erase requests for victim blocks are also assigned high priority to enable timely release of flash memory resources. Conversely, write requests for valid pages are designated as low priority, making their reserved paths subject to preemption by higher-priority requests. When high-priority requests seek reserved paths, they may invalidate existing low-priority path reservations, requiring the low-priority requests to resend scout packets to find other available paths. To guarantee eventual completion, low-priority requests automatically escalate to high-priority status after multiple failed reservation attempts. While GC bypass separates entry paths for GC and I/O requests at the controller level, all requests share common path resources and routers within the network. The differentiation occurs solely through entry points and priority logic without physical isolation. For preempted valid page write requests, GC bypass maintains a dedicated valid page buffer with an accompanying buffer mapping table, enabling efficient batch processing and management of deferred operations. GC controller and flash controller requests utilize the same non-minimal fully adaptive routing algorithm for path reservation, ensuring architectural consistency while implementing priority-based differentiation.

The interconnected network in Venice essentially establishes a pooled path resource system within the flash memory chip array, whose distributed routing nodes and fully adaptive routing algorithm provide the foundation for fine-grained control of requests with different priorities. GC bypass leverages Venice's existing hardware framework, implementing logical partitioning of the path resource pool into high- and low-priority sections solely through extended router priority logic, without introducing additional physical overhead. Furthermore, the native router reservation table of Venice inherently supports multi-entry request scheduling. Building upon this capability, GC bypass introduces the GC controller to centrally dispatch GC requests, thereby reducing the probability of path conflicts caused by random request distribution at the network entry points.

3.2 Adaptive routing algorithm with priority

GC bypass introduces priority bits into the scout packets of Venice, allowing high-priority requests to preempt the reserved paths of low-priority ones. It assigns high priority to I/O operations and critical GC operations, such as reading valid pages and erasing victim blocks, while assigning low priority to writing valid pages into flash chips. This ensures that during GC, the process of reading valid pages is not blocked by I/O operations, whereas the reserved paths for writing valid pages can be preempted. Additionally, even if not all valid pages in victim blocks have been migrated, the blocks can still be reclaimed in a timely manner, thereby freeing up available space in the SSD. GC bypass repurposes a 1-bit reserved field in the tail flit of the scout packet as a priority flag, where 0 indicates low priority (writing valid pages) and 1 indicates high priority (I/O operations, reading valid pages, and erasing victim blocks). The structure of the GC bypass scout packet is illustrated in Fig. 5, consisting of two 8-bit flits: a header flit and a tail flit, which can represent up to 8 controllers and 32 flash chips. Each flit contains 3 bits of type information: the first bit indicates whether the flit is a header flit or a tail flit; the second bit indicates whether the flit is in path-reservation mode or cancellation mode; the third bit indicates whether the flit is sent from a flash controller or a GC controller. The ID of the target flash chip is stored in the last 5 bits of the header flit, supporting up to 32 chips. In the tail flit, 3 bits are used to represent the source controller ID, 1 bit indicates the priority level, and the remaining 1 bit is unused. It should be noted that the source controller identifier must match the ID of the corresponding scout packet. By default, GC bypass assigns the GC controller ID to the last position in the current controller sequence. For example, in an SSD with five active channels, the GC controller is automatically assigned the identifier ID 100. For a given I/O request, GC bypass checks whether the flash controller closest to the target flash chip is available. If available, it selects that controller to handle the I/O request; otherwise, it uses the nearest idle flash controller. For GC requests, however, GC bypass can utilize only the dedicated GC controller to process the request.

Before request transmission, the source controller sends a scout packet in reservation mode to identify a path to the target chip. GC bypass employs a non-minimal, fully adaptive

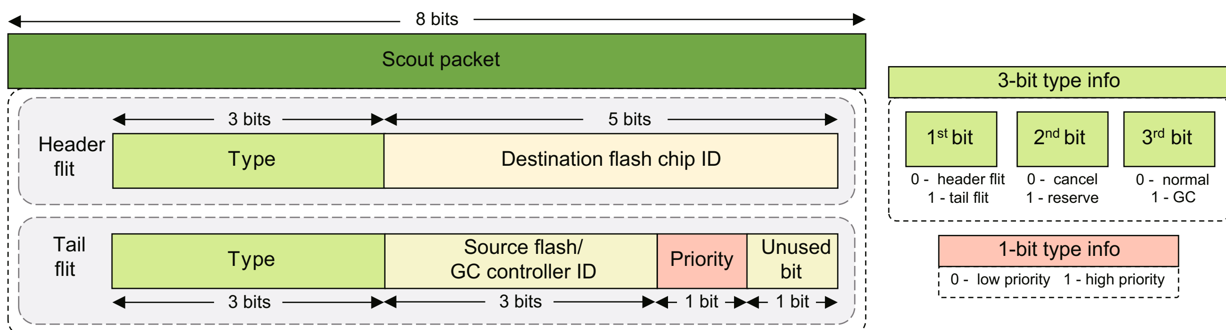

Fig. 5 Structure of the scout packet

routing algorithm similar to Venice to route the scout packet from the source controller to the target flash chip. During this process, all interconnected network links traversed by the scout packet are reserved. Each reserved link is bidirectional, allowing data to be transmitted via the forward path (from controller to flash chip, e.g., user write/valid page write requests) or the reverse path (from flash chip to controller, e.g., user read/valid page read requests). Each router chip maintains a router reservation table, as illustrated in Fig. 4. The router reservation table records the packet ID, which matches the source flash controller ID. The ID field has a length of $\log_2 n$ bits, where n denotes the maximum number of concurrent requests (i.e., the total number of controllers), ensuring unique identification for each reserved path. The table also includes 2-bit entry port and 2-bit exit port fields, indicating the input and output ports of the router (e.g., East=00, South=10, West=11, North=01). These fields are consistent with Venice's design. GC bypass extends this structure by introducing a 2-bit link state field to indicate the link status, which also encodes priority information. The encoding is as follows: 00—Link is idle and unreserved; 01—Link is reserved by a low-priority request and can be preempted; 10—Link is reserved by a high-priority request and cannot be preempted; 11—Link is actively transmitting data, regardless of priority, and cannot be preempted. The path status records the priority level of the current occupant. If the target port is occupied by a low-priority request, a cancellation packet is issued to forcibly release the path. When updating the router reservation table, the priority of the current packet is recorded. By leveraging the priority field and preemption logic, GC bypass enables high-priority requests to reserve paths efficiently, facilitating rapid path preemption and minimizing the impact of low-priority valid page write operations on normal I/O performance.

GC bypass maintains a 1-bit status vector for each chip to indicate its availability. A target chip is marked as 0 when it is either reserved by a path or busy, and as 1 when idle and available. This status determination directly influences the subsequent selection of write locations for valid pages. Upon receiving a scout packet from the controller, the router checks the link status flag of the target path. If the path is idle, it is immediately reserved. If the incoming scout packet carries a high-priority request, it may also preempt a path previously reserved by a low-priority request. If no available path can be preempted or the reserved path is already in use for data transmission, the router activates the cancellation mode in the scout packet. This removes the corresponding entry from the router reservation table, effectively canceling the reservation. The scout packet then backtracks along its path to upstream routers. Leveraging the adaptive nature of the routing algorithm, the scout packet may attempt alternative idle output links at upstream routers or continue backtracking further. If no idle or preemptable output link is found during backtracking, the scout packet returns to the flash controller without reserving a path. When the source controller receives the scout packet in cancellation mode, it retries the path reservation process by issuing a new scout packet. Upon successful path reservation, the source controller receives a scout packet in the reserved state.

The adaptive routing algorithm of GC bypass determines the forwarding direction of scout packets in a 2D mesh network. The core logic of its non-minimal fully adaptive routing algorithm selects the shortest path based on the relative positions of the current and target routers. If the shortest path is blocked, the algorithm attempts detour paths before resorting to backtracking. The algorithm operates in two phases. In the minimal-path phase, it prioritizes idle or preemptable ports. If the minimal path is occupied by a high-priority request, the algorithm enters the non-minimal-path phase, exploring alternative directions. For preemption, low-priority scout packets can only reserve idle links, whereas high-priority packets can forcibly release low-priority reserved paths, updating the router reservation table to reflect the new high-priority reservation. Low-priority packets may dynamically elevate their priority after repeated failures. During the minimal-path phase, possible directions are selected based on the relative coordinates (Diff_x and Diff_y) between the current and target positions. For instance, if the target is northeast, the algorithm attempts east and north ports. Nine possible cases cover all relative position combinations. Each port has four states, corresponding to the router's link status: Free (00), Reserved_Low (01), Reserved_High (10), and Active (11). A port state-checking function, can_use, is defined to select idle output ports. If a target port is reserved but the current request has higher priority, preemption is permitted. All eligible idle or preemptable ports are collected, and one is randomly selected using a router-internal linear-feedback shift register (LFSR) for lightweight load balancing. If a preemptable port is chosen, a cancellation signal is sent to release the low-priority reservation before updating the router table with the new high-priority reservation.

If no minimal-path ports are available for preemption or are active, the algorithm proceeds to the non-minimal-path phase. To avoid backtracking, the input port is excluded, and other directions are tested for idle or preemptable ports, following the same selection logic as the first phase. If no ports remain, the algorithm attempts backtracking by returning to the input port, prompting upstream routers to reselect paths. To prevent the starvation of low-priority requests, GC bypass implements a backtrack counter for each packet. If a low-priority packet fails to secure a path and returns to the source controller, the counter is incremented. Once the counter exceeds a threshold, the priority of the packet is forcibly elevated. The backtrack counter is typically adjusted based on the available free space in the SSD. When there are more free pages, the counter can be larger, as the GC operation is less urgent. Conversely, when fewer free pages are available, the counter is smaller to expedite the GC process.

The time complexity of our algorithm in an $N \times N$ matrix, where $N \times N$ represents the total number of chips, and the controller starts at $(0, i)$ and searches for a specific destination at (m, n), is influenced by multiple factors, including path selection, priority control, path preemption, and backtracking. In the shortest path phase, the algorithm attempts to find the most direct path from the source to the destination, typically using BFS or A* algorithms, which have a time complexity of $O(N^2)$, as the entire $N \times N$ matrix may need to be explored in the worst case. If the shortest path is unavailable or blocked, the algorithm may resort to non-minimal (detour) paths, leading to an additional time complexity of $O(N^2)$.

During path preemption and priority handling, the algorithm checks the status of up to four potential paths (up, down, left, right) in constant time $O(1)$, but dynamic adjustments and preemption introduce overhead. Furthermore, if the path is blocked, backtracking may be required, which in the worst case adds a time complexity of $O(N^2)$. Considering all these factors, the overall worst-case time complexity is dominated by the need to traverse all nodes in the matrix, making the total time complexity be $O(N^2)$. While path preemption and backtracking add additional complexity, the algorithm generally maintains a worst-case time complexity of $O(N^2)$, similar to that of BFS or A*, but with enhanced flexibility and dynamic path selection. Since the value of $O(N^2)$ corresponds to the number of flash chips, which is typically between 16 and 64 (Cui et al., 2024), the complexity can be considered negligible.

3.3 GC process

Compared with Venice, GC bypass introduces a dedicated GC controller to achieve entry isolation for different types of requests in the interconnected network composed of flash memory chips. All GC requests enter the network exclusively through the GC controller, which decouples path management from conventional GC operations, enabling more flexible priority scheduling. As the core component of GC bypass, the GC controller integrates three key functions: GC buffer, priority management, and path management (Fig. 6).

1. GC buffer: The GC controller maintains a valid page buffer (GC buffer) and a buffer mapping table. The GC buffer serves as a temporary storage area for low-priority valid page write requests delayed due to path preemption, with a capacity equivalent to one flash block. To prevent data loss during sudden power outages, the capacitor equipped in SSD caches could flush all valid pages from the cache (Gao et al., 2020a; Ren et al., 2025). The buffer mapping table records the logical page number (LPN), buffer page number (BPN), process-specific type tag (Type), and dirty bit (Dirty) for each buffered valid page. When reading valid pages from victim blocks, the LPN and BPN are bound, and pages from the same GC process share the same Type value for batch write-back management. If a host I/O modifies a buffered valid page, the Dirty bit is set to 1, indicating that the page must be written to a new physical location. By grouping buffered pages with the same Type into batch requests, the buffer mapping table reduces path reservation overhead. This caching mechanism significantly mitigates interference from valid page writes on real-time I/O while minimizing the amount of redundant path reservations caused by conflicts.

2. Priority management: The priority management module assigns dynamic priorities to GC requests, ensuring that I/O operations, valid page reads, and victim block erases are prioritized. While maintaining compatibility with Venice's flash controller, including command scheduling and data randomization, this module introduces priority-based preemption logic, enabling routers to swiftly release low-priority link resources.

3. Path management: This module selects target chips for valid page writes by integrating the wear-leveling algorithm of the FTL with real-time chip status vectors, achieving balanced wear and load distribution. During valid page writes, it prioritizes idle chips with lower wear levels and reserves paths via low-priority scout packets. Collaborating with the routing strategy of the interconnected network, this module supports non-minimal path selection.

Fig. 7 illustrates the scheduling diagram for various types of requests in the SSD. Upon arrival at the host queue, host I/O requests are divided into page-sized I/O transactions (subrequests) and distributed to multiple channel queues. Building upon Venice, GC bypass introduces GC I/O queues to store fine-grained GC sub-requests, isolating them from user I/O. Additionally, a valid page buffer is incorporated to temporarily hold valid pages requiring migration during GC. While I/O requests may dispatch scout packets via the nearest available flash controller to the target chip, all GC requests must exclusively utilize the dedicated GC controller for scout packet transmission. GC bypass decomposes GC requests into N valid page read requests (GC_R), multiple valid page write requests (GC_W), and a single block erase request (GC_E), where N denotes the number of valid pages in the victim block. GC is triggered when the SSD's remaining free space falls below a predefined threshold. The FTL first selects a victim block, a process orthogonal to existing victim block selection algorithms

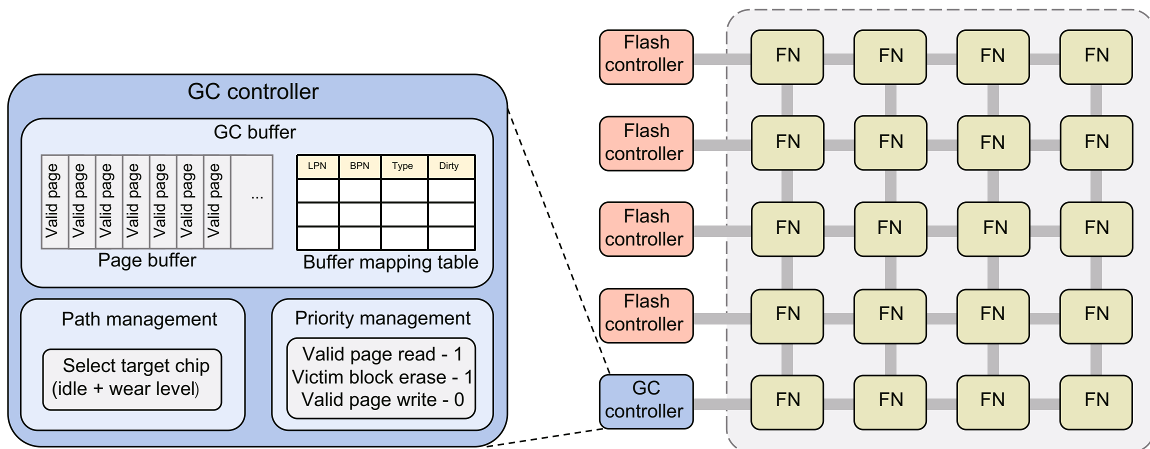

Fig. 6 Structure of the GC controller (FN: flash node)

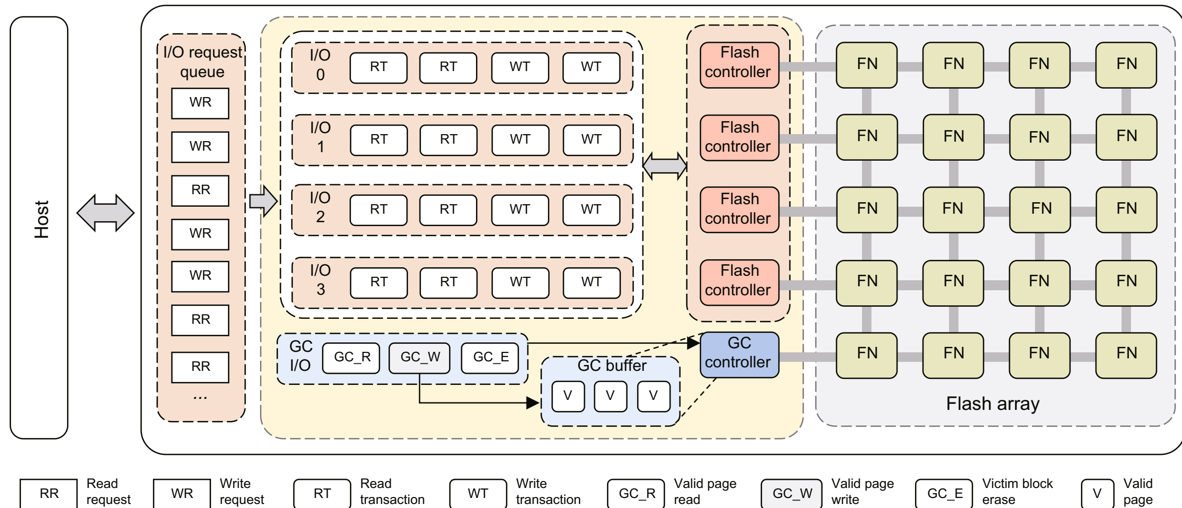

Fig. 7 Workflow of GC and I/O request scheduling (FN: flash node)

(e.g., FIFO, Greedy, and Random). After selecting the victim block, the FTL transmits the target chip information and valid page addresses to the GC controller, which then issues a high-priority scout packet to the victim block's chip to read its valid pages. Leveraging GC bypass's adaptive routing algorithm, if a path's link is reserved by a low-priority request, preemption is triggered, releasing the link and updating its reservation to high priority. Should no low-priority or idle path be available, the corresponding entry is removed from the routing table, and the scout packet backtracks. If backtracking fails to secure an idle path, the scout packet returns to the GC controller with a cancellation flag, leaving no path reserved. Upon successful path reservation, valid pages from the GC process are read into the buffer, with their FTL mapping table entries marked as invalid and the buffer mapping table updated. After all valid pages are read, a high-priority scout packet reserves a path to erase the victim block, releasing the original flash chip resources. For buffered valid page write requests, GC bypass aggregates pages of the same Type into batch requests to minimize path reservation overhead. GC bypass queries the chip status vector and incorporates the FTL's wear-leveling algorithm to select an idle, low-wear target chip, dispatching a low-priority scout packet to reserve a path. Successful reservation enables writing the buffered valid pages to their new locations, followed by updates to the FTL mapping table and clearance of buffer mapping table entries. If a valid page write request fails to reserve a path three consecutive times, its priority is temporarily elevated, and the scout packet is retransmitted. If the host modifies a buffered page (i.e., Dirty=1), the page must be written to a new physical location during write-back rather than overwriting the originally migrated page.

3.4 I/O process

The SSD's host interface layer (HIL) converts host I/O requests into multiple page-sized transactions, which are then distributed by the FTL to multiple channel queues before being dispatched to target chips via flash controllers. For read transactions, the buffer mapping table is first queried to check whether the target LPN exists. If a hit occurs, data are directly read from the buffer corresponding to the BPN, eliminating the need for flash access. In case of a miss, a high-priority scout packet is sent from the flash controller to reserve a path, and upon successful reservation, data are read from the flash memory. For write transactions, if the buffer mapping table yields a hit, the data in the buffer are updated at the corresponding BPN, with the Dirty bit set to 1, indicating that the data must be written back to the flash. If a miss occurs, a high-priority scout packet is sent from the flash controller to reserve a path, allowing the data to be written to a new physical page, followed by an update to the FTL mapping table. Similar to Venice, GC bypass handles deadlocks through scout packet backtracking. When a path conflict occurs during path reservation, the scout packet backtracks along its path to previously visited routers. Consequently, scout packets are never blocked due to resource unavailability in the interconnected network, preventing deadlocks. To address live locks, GC bypass restricts the number of times a scout packet can revisit the same router. Each scout packet reserves each output port of a router only once, meaning that in a 2D mesh topology, a scout packet can revisit the same router at most three times. The number of retries is based on the number of available router ports minus one, as the scout packet can revisit a router up to four times, with one of the ports reserved for the entry of the scout packet. We also explore the impact of retry times on the success ratio, with the results shown in Table 1. We calculate the success ratio of finding the path from the flash controller to the target chips.

Table 1 Success rate vs. the maximum number of retries (N)

N	1	2	3	4	5	6
Success rate	0.954	0.968	0.983	0.980	0.981	0.979

If a scout packet revisits the same router three times, it traces its path back to an upstream router by querying the router reservation table and attempts to reserve a different output port at the upstream router. Even in the worst-case

scenario, where a scout packet fails to reserve a path to the destination after visiting all routers three times, it returns to the source controller, which then sends a new scout packet to retry path reservation.

3.5 Overhead analysis

GC bypass builds upon Venice's scheme of attaching a dedicated router chip adjacent to each flash memory chip to form flash nodes. The router directly utilizes the existing I/O pins of the flash chip, eliminating the need for modifications to flash chip design. The scout packets used for path reservation contain minimal control information, resulting in extremely low transmission latency. The transfer latency could be calculated with

$$T_{\text{transfer}} = \left(\text{distance} + \frac{\text{transfer_size}}{\text{link_width}}\right) \cdot \text{latency}_{\text{link}}, \quad (1)$$

where distance, transfer_size, link_width, and $\text{latency}_{\text{link}}$ represent the number of hops between the flash controller and the flash chip (including scout packet retries and path preemption), the command/data transfer size in bytes, the link width in bytes, and the latency of a single transfer (of size link width) on the link in seconds, respectively (Nadig et al., 2023). In comparison to the IO command and data transfer latency (e.g., 16 KB), the scout packets are small (8 bits). During the data transfer phase, delays are primarily driven by data volume and flash operations, with minimal impact from additional hops introduced by non-minimal paths. Venice achieves a path reservation success rate as high as 99.98%, and its retry mechanism ensures rapid response to failures, further reducing waiting time. Compared to Venice's low-cost path diversity solution with minimal overhead, GC bypass introduces the following additional costs:

1. Priority configuration overhead: GC bypass repurposes a 1-bit reserved field in the scout packet's tail flit as a priority identifier without increasing the total packet length. The expansion of the path reservation table adds only 2 bits of link state information per entry, representing a negligible storage overhead.

2. GC controller overhead: Traditional NAND interfaces share the same set of pins for command, address, and data transmission. As bus speeds increase, command and address transfer efficiency becomes a bottleneck, reducing overall bus utilization. In November 2024, JEDEC Solid State Technology Association introduced the JESD230G standard, incorporating the SCA (separate command address) protocol (JSST Association, 2024). SCA decouples command/address signals from data signals, introducing a dedicated CA (command and address) channel for serial transmission, simplifying wiring and circuit design while enabling higher speeds and lower power consumption. Under conventional interfaces, the GC controller requires independent CA channel support. However, SCA's serialized protocol allows the GC controller to reuse existing CA channels for command transmission, theoretically eliminating the need for additional physical pins. While traditional architectures require a dedicated channel entry for GC requests, SCA necessitates only logical module extensions, resulting in minimal hardware overhead.

If we use the conventional design, each router occupies about $8\,\text{mm}^2$, which is 8% of a typical $100\,\text{mm}^2$ NAND flash chip (Nadig et al., 2023). The GC controller introduces additional hardware overhead. According to the flash controller overhead listed in Qiu et al. (2021), the GC controller consumes approximately 3774 LUTs and 4799 FFs, which account for about 10% of the resources on an FPGA board. However, in our design, we only split the NAND flash controller into the GC controller for GC and the regular NAND flash controller for servicing host requests. Thus, the GC controller mainly adds connections to other chips. Additionally, we use the CACTI tool (Balasubramonian et al., 2017) to model the additional memory overhead. Since the SSD is equipped with DRAM (ranging from MB to GB (Wang et al., 2024)), we quantify only the DRAM used by the GC controller. The overhead for the GC controller is as follows: the power consumption is 1.87 nJ, and the DRAM core area is $72.5\,\text{mm}^2$ for 1 GB 3D DRAM. Given that the SSD already has DRAM, the existing DRAM resources can be utilized by the GC controller, making the hardware overhead relatively negligible.

3. Buffer overhead: In GC bypass, the valid page buffer is sized to one flash block (e.g., 1 MB), accommodating 256 valid pages in one block at maximum, and the size could be varied with the block size. We also investigate the impact of page buffer size on IO performance and DRAM consumption. When the GC DRAM buffer size equals the block capacity, all valid pages can be moved into the DRAM buffer at once. In contrast, if the buffer size is half or one-quarter of the block capacity, multiple steps are required to migrate the valid pages. We measure the relationship between IO overhead and GC DRAM buffer size, with results presented in Table 2. These results indicate that splitting the GC in one block into several steps benefits normal IO requests, although the improvement is modest. Consequently, we adopt the simple configuration where the GC DRAM buffer equals the block size. Since the buffer resides in the GC controller for fast access, its read/write latency is significantly lower than flash access latency, making the time cost of transferring valid pages negligible. The buffer mapping table introduces marginal space overhead—each entry requires 8 bits for buffer location and GC process type, compared to 4 bytes per entry in traditional FTL mapping tables for logical-to-physical address translation. Thus, the buffer mapping table's storage impact remains negligible.

Table 2 GC strategy performance comparison (50% valid pages)

Number of batches	GC total time (normalized)	Normalized DRAM	Normalized IO avg latency
1*	1.00	1.00	1.00
2	1.00	0.50	1.00
3	1.00	0.34	0.999
4	1.00	0.24	0.999

* All-at-once. avg: average

4 Experiments

4.1 Experimental environment setup

We employ the open-source SSD simulator MQSim (Tavakkol et al., 2018) to evaluate and compare GC bypass. Developed by CMU-SAFARI, MQSim represents the

state-of-the-art SSD simulator capable of accurately modeling key front- and back-end components of modern SSDs. The experimental SSD configuration is shown in Table 3. To validate the effectiveness of GC bypass, we compare the following schemes:

1. Baseline: a conventional SSD serving as the reference benchmark;

2. Venice: a solution providing rich path diversity through interconnected networks and path reservation techniques;

3. GC bypass: an enhanced Venice implementation that introduces priority mechanisms, where valid page write requests are assigned low priority to allow their reserved paths to be preempted by higher-priority requests, thereby mitigating path conflicts between GC and I/O operations.

The test workloads are derived from the MSR Cambridge dataset (Narayanan et al., 2009) and Alibaba Group (Li et al., 2020), whose characteristics are detailed in Table 4. The Alibaba dataset contains I/O activities recorded in January 2020 from 1000 virtual disks, randomly sampled from the Alibaba cluster. Our experiments focus on the first five disks from the Alibaba dataset to validate the effectiveness of our proposed scheme. Collected from server environments, the MSR traces encompass diverse I/O patterns typical of data centers, including high randomness, mixed read/write ratios, and dynamic load fluctuations, making them well-suited for assessing SSD performance under intensive workloads, as well as Alibaba Cloud traces.

4.2 Experimental results analysis

We quantitatively evaluate and compare our scheme with existing schemes in terms of end-to-end latency, bandwidth, tail latency, and other relevant performance metrics under the current experimental environment.

1. End-to-end delay. The end-to-end delay of a request refers to the total time elapsed from the time when the host initiates an I/O request until the request is fully processed by the SSD, and the result is returned to the host. This latency includes the queuing delay in the HIL submission queue, the scheduling delay within the SSD, as well as the time consumed by path reservation, data transmission, and flash memory operations. Additionally, it encompasses extra waiting delays caused by path conflicts, GC, or concurrent requests. Fig. 8 presents a comparison of the average end-to-end delay across the three schemes under different workloads, normalized against the Baseline. GC bypass demonstrates the best performance across all workloads, achieving up to 62% and 26% reductions in average end-to-end delay compared to the Baseline and Venice, respectively. Baseline employs a traditional multi-channel shared bus architecture, where multiple flash chips share the same channel for controller communication. When multiple requests access the same channel simultaneously, path conflicts arise, forcing serialized request processing and increasing latency. Both GC and I/O requests—as well as contention among I/O requests—compete for channel access, leading to I/O request blocking. In Venice, all requests share the same set of channel entry points but leverage adaptive routing algorithms to reserve and select distinct paths. Although path diversity increases, the lack of priority differentiation means that GC operations, when frequently triggered, may still occupy critical paths, adversely affecting the response time of real-time I/O requests. GC bypass further optimizes Venice's approach by assigning high priority to I/O requests and critical GC operations (e.g., reading valid pages and erasing), allowing them to preempt path resources and directly reduce waiting time. Low-priority valid page write requests, whose reserved paths can be preempted, avoid blocking high-priority requests, ensuring faster response for user I/O.

Table 3 SSD configuration

Parameter	Value
Number of SSD channels	4
Number of chips per channel	8
Number of dies per chip	2
Number of planes per die	2
Number of blocks per plane	512
Number of pages per block	256
Flash page size	4 KB
Read latency	75 µs
Write latency	750 µs
Erase latency	3.8 ms

Table 4 Workload characterization

Trace	Write ratio	Average write request size (KB)	Average read request size (KB)
HM_0	75.1%	11.2	11.7
HM_1	3.1%	22.9	18.1
PRN_0	89.4%	14.0	26.6
PRXY_0	97.1%	6.3	9.7
RSRCH	91.0%	12.5	15.7
STG_0	76.9%	12.7	33.6
USR_0	62.9%	13.5	47.4
WDEV_0	79.9%	12.1	16.6
Ali0	99.3%	79.7	6.87
Ali1	99.8%	6.33	25.8
Ali2	97.6%	13.6	14.3
Ali3	98.9%	17.45	4.01
Ali4	98.1%	4.78	4.03

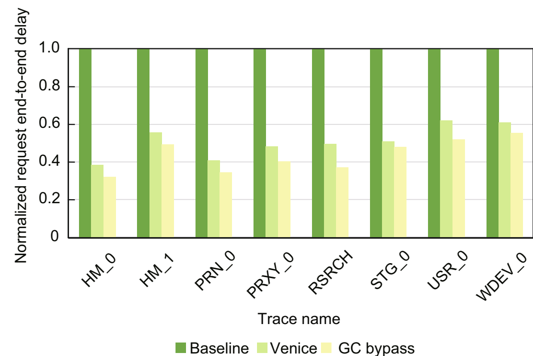

Fig. 8 Normalized request end-to-end delay

2. Average latency of read/write transaction. The total latency of each read/write transaction consists of queuing time, path reservation time, and data transfer time, where a transaction refers to each read or write sub-request. The data transfer time includes command transmission, flash operations, and data migration, primarily determined by the SSD's configuration parameters, making it identical across all three schemes. Venice and GC bypass employ a non-minimal fully adaptive routing algorithm to swiftly select candidate paths. Upon path reservation failure, they immediately resend scout packets rather than waiting for a fixed interval, thereby minimizing idle waiting time and reducing the impact of additional path

reservation latency. Consequently, under real-world workloads, the average read/write transaction latency of all three schemes is predominantly determined by queuing time, the duration a transaction waits in the queue for resource allocation.

Under MSR workloads with intensive access patterns, the SSD controller's processing queue may become saturated, forcing new I/O requests to wait until preceding requests are completed, directly increasing queuing delays. In SSDs, path conflicts are widely recognized as a major contributor to elevated queuing latency. Figs. 9 and 10 present the normalized average read/write transaction latency relative to the Baseline. GC bypass achieves the lowest read and write latency across all workloads, reducing average read and write transaction delays by up to 18% and 39%, respectively, compared to Venice. Baseline exhibits the highest queuing delay due to the serial access constraints of fixed channels. Venice mitigates this by providing path reservation and adaptive routing, offering rich path diversity for I/O requests. However, in scenarios where GC is frequently triggered, GC requests may occupy substantial path resources, increasing queuing delays for I/O requests. GC bypass ensures priority for I/O requests and critical GC operations, minimizing resource contention between valid page migration and I/O while guaranteeing timely victim block reclamation. When path resources are scarce, high-priority I/O requests can directly preempt paths reserved by low-priority requests, reducing queuing delays caused by GC operations occupying critical links.

3. Bandwidth. SSD bandwidth refers to the amount of data that can be transmitted per unit time, which is influenced by the number of channels, the bandwidth of each channel, and the actual channel utilization rate. Fig. 11 displays the normalized bandwidth with Baseline as the reference. Experimental results demonstrate that GC bypass achieves the highest bandwidth across all workloads, showing improvements of up to 3.1 and 1.3 times compared to Baseline and Venice, respectively. In Baseline, bandwidth is wasted due to path conflicts in the multi-channel shared bus architecture. While Venice alleviates such conflicts through its interconnected network, all requests still compete equally for path resources. GC bypass enables high-priority requests to preempt paths reserved by low-priority requests, forcibly releasing idle reserved links for high-priority reservations. This mechanism allows high-priority requests to reclaim links reserved for low-priority valid page writes, avoiding the bandwidth wastage observed in Venice when valid page migration occupies critical links. Furthermore, in GC bypass, aggregating valid page write requests from the buffer reduces the number of path reservations, thereby increasing the proportion of effective data transmission.

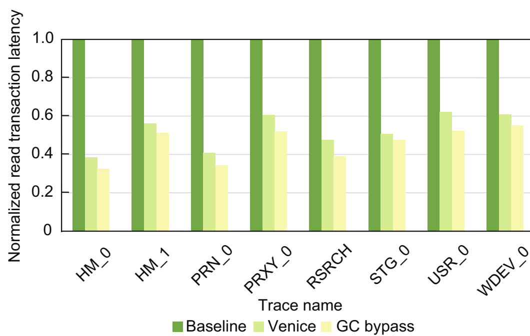

Fig. 9 Normalized read transaction latency

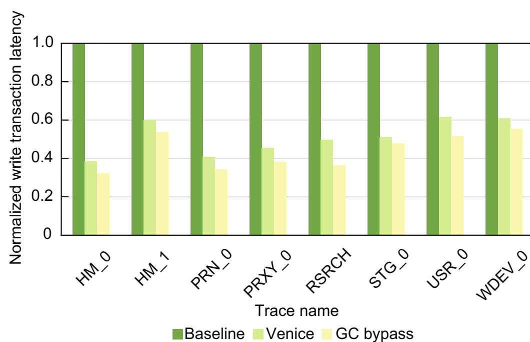

Fig. 10 Normalized write transaction latency

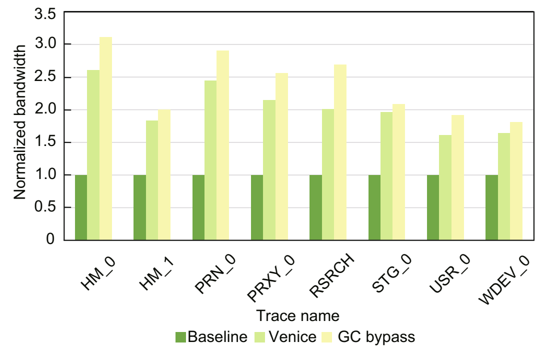

Fig. 11 Normalized bandwidth

4. Long-tail latency. The CDF curves of request response times are illustrated in Figs. 12a–12h, demonstrating the latency control capabilities of the three schemes under extreme scenarios. Experimental results indicate that GC bypass maintains the lowest latency distribution across all workloads. Under the RSRCH workload, GC bypass reduces the tail latency of 99.99% I/O requests by 62.7% and 25.2% compared to Baseline and Venice, respectively. In Baseline, requests can only be transmitted through the channel corresponding to the target chip, resulting in high path contention among I/O requests. During GC, I/O requests are further blocked by GC operations for extended periods, exhibiting the worst long-tail latency. Venice mitigates this issue by providing abundant path diversity, reducing long-tail latency caused by path conflicts and prolonged queuing delays. However, due to the absence of priority differentiation, valid page migration and user I/O requests compete equally for path resources in scenarios with frequent GC triggers, leading to persistent long-tail latency issues. GC bypass ensures that user I/O requests always receive priority access to path resources through its priority-based path preemption mechanism, significantly reducing interference from GC operations and directly decreasing I/O waiting times. Even during peak flash block reclamation periods, I/O requests can still be serviced promptly, avoiding extreme delays caused by resource contention. Furthermore, by centralizing GC requests through the dedicated GC controller entry point, GC bypass isolates them from user I/O paths, concentrating GC-related routing within the interconnected network. This design reduces random collisions between I/O and GC requests. GC bypass effectively suppresses latency spikes induced by

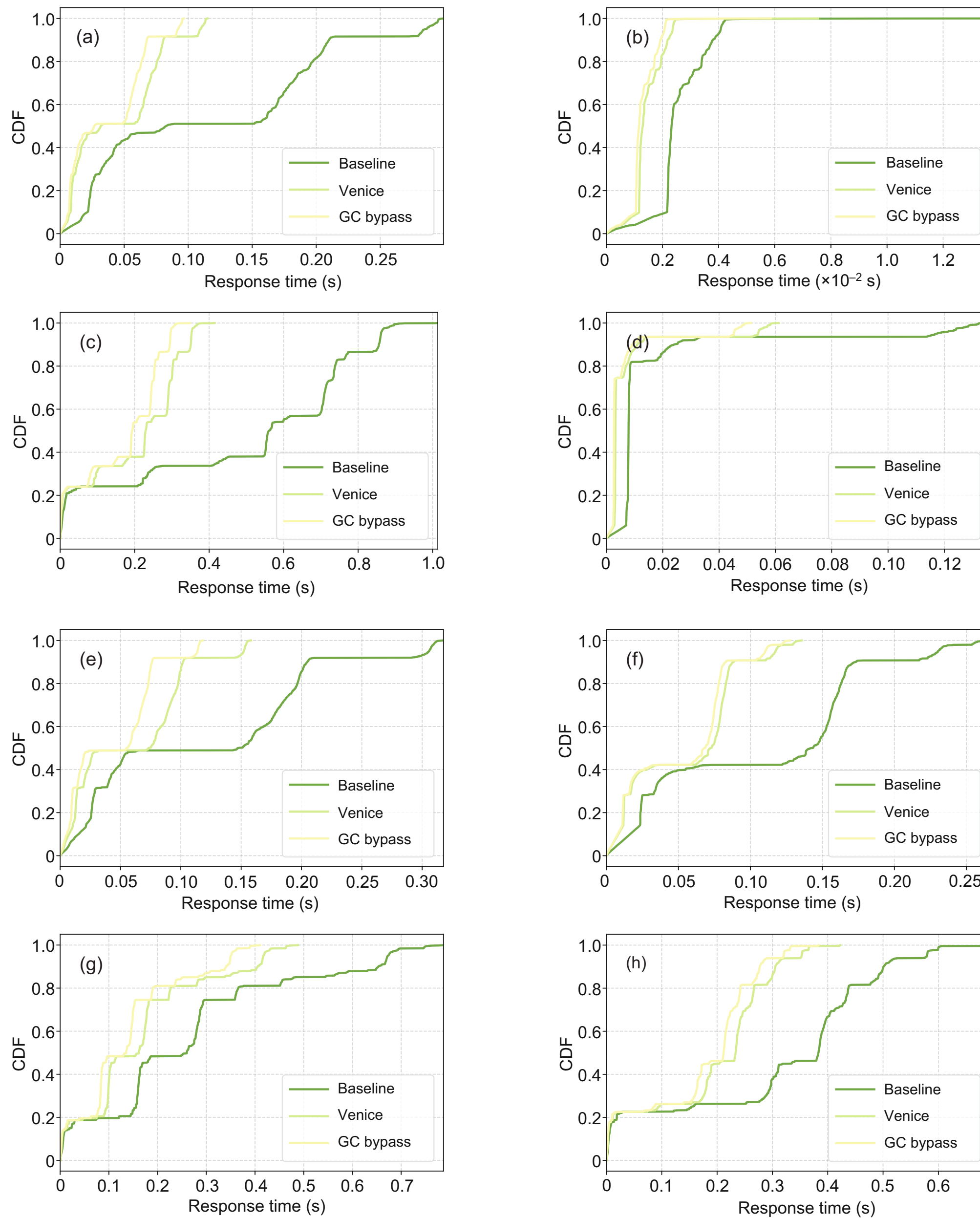

Fig. 12 Cumulative distribution function (CDF) of response time: (a) HM_0; (b) HM_1; (c) PRN_0; (d) PRXY_0; (e) RSRCH; (f) STG_0; (g) USR_0; (h) WDEV_0

sudden GC operations occupying critical paths, particularly under MSR workloads with intensive access patterns. The scheme demonstrates notable improvements in mitigating extreme delays under high-stress conditions.

5. Long-time running. The request response time curve reflects the stability of the I/O performance of different schemes under various workloads. As shown in Figs. 13a–13h, the request response time curve of GC bypass is lower than those of Baseline and Venice, with smaller jitter and lower peak values. The serial access mode in the same channel in the Baseline leads to random congestion, resulting in significant I/O jitter. Venice, with its fair competition between GC and user I/O, causes momentary resource contention during GC-intensive periods. GC bypass, even under sudden load surges, ensures that high-priority requests occupy the link first, avoiding an increase in response time due to resource exhaustion. The reserved path for valid page write requests allows preemption and temporarily stores them in the valid page cache, preventing them from monopolizing the link and blocking I/O requests.

6. Performance comparison on Alibaba block traces. We analyze the performance of three schemes using Alibaba Group's trace, which includes read/write records from 100 devices, with data from 5 devices selected for comparison. The results (Fig. 14) highlight key differences in response times. GC bypass consistently outperforms Baseline and Venice. Baseline uses a multi-channel shared bus architecture where multiple flash chips compete for the same channel, leading to path conflicts and increased latency. Venice improves path diversity with adaptive routing but lacks priority differentiation,

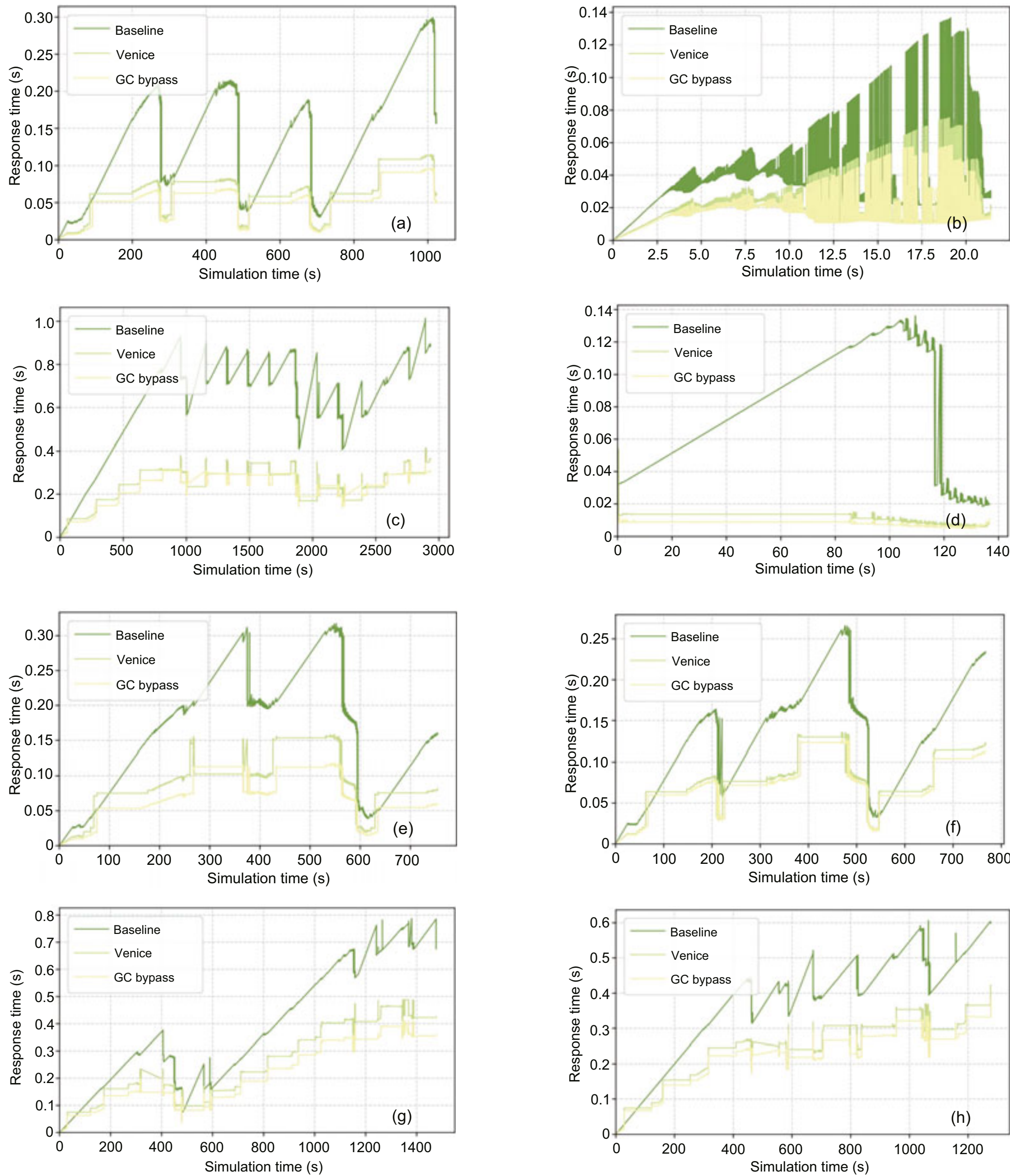

Fig. 13 Response time: (a) HM_0; (b) HM_1; (c) PRN_0; (d) PRXY_0; (e) RSRCH; (f) STG_0; (g) USR_0; (h) WDEV_0

which results in GC operations occupying critical paths and delaying real-time I/O requests. GC bypass optimizes Venice's approach by prioritizing I/O and critical GC operations, allowing them to preempt paths and reduce waiting time. Low-priority valid page write requests can be preempted, avoiding I/O blocking and ensuring faster response for user I/O requests.

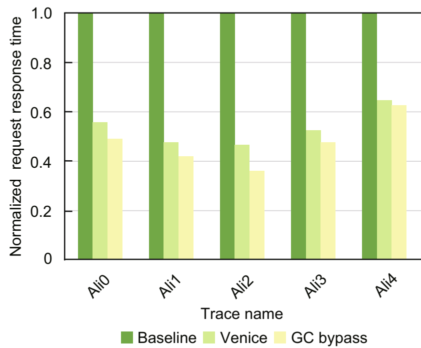

Fig. 14 Performance comparison on Alibaba block traces

7. Chip utilization comparison. We analyze the chip utilization across different workloads to assess the performance improvements of each scheme. The results (Fig. 15) clearly indicate a significant improvement in chip utilization with GC bypass. The chip utilization, represented as the fraction of time spent in execution, is compared for three schemes: Baseline, Venice, and GC bypass. As shown in Fig. 15, GC bypass demonstrates the most consistent and efficient chip utilization across all workloads. In contrast, Baseline and Venice exhibit noticeable variations in chip utilization, especially under high-load scenarios. Baseline suffers from inefficient utilization due to path conflicts and contention among other chips on the same channel, leading to lower overall chip utilization. Venice improves chip utilization by adopting adaptive routing, but

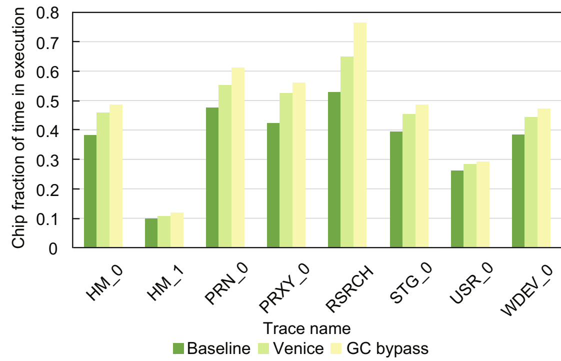

Fig. 15 Chip utilization comparison

the lack of prioritization results in occasional underutilization during GC-intensive periods. GC bypass further optimizes chip utilization by prioritizing critical I/O and GC operations, ensuring that the chip's resources are used more effectively. This scheme leads to a more balanced and efficient execution across different workloads, significantly reducing idle time and improving overall throughput. In summary, chip utilization analysis highlights the advantages of GC bypass in maintaining higher and more stable chip utilization, which contributes to improved overall system performance.

8. GC overhead among GC bypass, Venice, pSSD, and SmartNetSSD. We evaluate the GC overhead for four schemes—GC bypass, Venice, pSSD, and SmartNetSSD—across multiple workloads: Ali0, Ali1, Ali2, HM_0, PRN_0, and PRXY_0 (Fig. 16). GC bypass consistently exhibits the lowest GC overhead, with values ranging from 0.73 to 0.82, demonstrating its efficiency in reducing GC overhead. In contrast, Venice shows slightly higher overheads, ranging from 0.80 to 0.89, due to its adaptive routing strategy. pSSD has the highest GC overhead, as it could not utilize the flash controller fully. Overall, GC bypass outperforms the other schemes in minimizing GC overhead and ensuring more stable system performance across varying workloads.

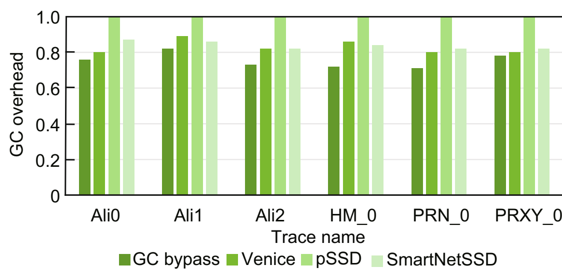

Fig. 16 GC overhead

5 Related works

5.1 Path diversity

The traditional SSD architecture employs a rigid connection structure between controllers and flash memory chips, where limited path diversity makes it difficult to achieve effective resource isolation between GC and foreground I/O operations. To address this challenge, recent research has focused on reconfiguring internal SSD architectures by adopting channel reconfiguration mechanisms to mitigate path conflicts between requests. To alleviate path conflicts by increasing flash channel bandwidth, Kim et al. (2022) proposed a packetized SSD architecture that enhances flash memory interconnect bandwidth using packetized communication technology. This design replaces dedicated signals with packet-based transmission to achieve a higher effective flash channel bandwidth. However, increasing system bus bandwidth does not fundamentally resolve resource contention between GC and I/O operations. In terms of improving path diversity, Kim et al. (2022) introduced pnSSD, which utilizes vertical bus channels to enable interconnections among flash chips within the same column, providing each chip with two access paths to reduce performance overhead caused by path conflicts. Nevertheless, these methods fail to fully resolve path contention issues due to insufficient path diversity between flash controllers and chips, along with non-negligible additional hardware overhead. Tavakkol et al. (2013) replaced the shared multi-channel bus with an interconnected network, proposing the NoSSD architecture that supports pipelined multi-router access to flash memory. However, direct modifications to flash chips incur high costs, and the deterministic routing algorithm employed exhibits limitations in path diversity. Nadig et al. (2023) introduced Venice, which incorporates a low-cost interconnected network between SSD controllers and chips without altering flash chip structures. Leveraging path reservation technology, Venice ensures conflict-free paths from flash controllers to target chips for each I/O request, providing substantial path diversity. However, in Venice, each I/O request exclusively occupies a single access path. Building upon Venice, Cui et al. (2024) proposed SmartNetSSD, which adopts a multipath routing algorithm to reserve multiple access paths per request, further enhancing SSD path diversity. SmartNetSSD pipelines successive read retry steps, allowing multiple sensing operations on the same data page using fine-tuned read reference voltages after initial sensing, thereby reducing read response time.

5.2 Scheduling GC

Recent studies addressing the potential impact of GC on host I/O performance in SSDs have adopted various approaches to mitigate interference. Some works prioritize I/O requests blocked by GC operations, ensuring that critical I/O processing remains timely even during GC execution. Alternative research explores interruptible or coalesced GC mechanisms to improve concurrency efficiency. Lee et al. (2013) proposed semi-preemptive GC, permitting GC tasks to be interrupted when processing queued I/O requests while employing pipelining to merge underlying GC and I/O operations such as data migration and writes. However, GC interruption may induce fragmentation issues, increasing GC trigger frequency over prolonged operation. Yan et al. (2017) developed tiny-tail flash, which reduces GC interference through physical isolation and parallel processing. Leveraging modern SSDs' multi-plane architecture and redundancy techniques, this approach confines GC operations to specific physical planes while allowing others to service I/O requests. Nevertheless, such hardware-dependent schemes exhibit limited applicability in cost-sensitive SSDs. Another research direction focuses on executing fine-grained GC operations during I/O request

intermissions to minimize critical path interference. Kang et al. (2017) employed reinforcement learning to predict optimal idle intervals for fine-grained GC execution. Sha et al. (2021) used Fourier transforms to identify sparse I/O request windows, constructing dynamic GC workload allocation models that comprehensively consider I/O intensity, read/write ratios, and device status for temporal separation between GC and I/O operations. Paik et al. (2018) introduced a selectively delayed GC scheme that triggers collection only when pending read operations are not at risk of resource preemption by GC.

6 Conclusions

We propose GC bypass to optimize the long-tail latency caused by page migration during GC in SSDs. By introducing a dedicated GC controller, GC requests are transmitted through independent interfaces, and the process of writing valid pages is assigned low priority, allowing high-priority requests to preempt path reservations from low-priority ones. Experimental results indicate that GC bypass reduces the 99.99^{th} percentile long-tail latency by up to 25% compared to Venice.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 62202368) and the National Key Research and Development Program of China (No. 2022YFB2902703).

Author contributions

Jie NIU and Shiqiang NIE designed the research. Xiaobo LI and Mingming ZHANG processed the data. Jie NIU drafted the paper. Yingzhao SHAO helped organize the paper. Weiguo WU and Yingzhao SHAO revised and finalized the paper.

Conflict of interest

All the authors declare that they have no conflict of interest.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Declaration on the use of generative AI tools

During the preparation of this work, the authors used ChatGPT and Grammarly to improve the language and readability. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

References

Balasubramonian R, Kahng AB, Muralimanohar N, et al., 2017. CACTI 7: new tools for interconnect exploration in innovative off-chip memories. *ACM Trans Archit Code Optim*, 14(2):14. https://doi.org/10.1145/3085572

Cui JH, Chen FY, Li L, et al., 2024. SmartNetSSD: exploiting path resources for read performance improvement in network-based SSDs. IEEE 42nd Int Conf on Computer Design, p.356-359. https://doi.org/10.1109/iccd63220.2024.00061

Gao CM, Shi L, Di YJ, et al., 2017. Exploiting chip idleness for minimizing garbage collection-induced chip access conflict on SSDs. *ACM Trans Des Autom Electron Syst*, 23(2):15. https://doi.org/10.1145/3131850

Gao CM, Shi L, Li Q, et al., 2020a. Aging capacitor supported cache management scheme for solid-state drives. *IEEE Trans Comput Aided Des Integr Circ Syst*, 39(10):2230-2239. https://doi.org/10.1109/tcad.2019.2949541

Gao CM, Shi L, Liu K, et al., 2020b. Boosting the performance of SSDs via fully exploiting the plane level parallelism. *IEEE Trans Parall Distrib Syst*, 31(9):2185-2200. https://doi.org/10.1109/TPDS.2020.2987894

JEDEC Solid State Technology Association (JSST Association), 2024. JESD230G: NAND Flash Interface Interoperability. Arlington, VA, USA.

Kang W, Shin D, Yoo S, 2017. Reinforcement learning-assisted garbage collection to mitigate long-tail latency in SSD. *ACM Trans Embed Comput Syst*, 16(5s):134. https://doi.org/10.1145/3126537

Kim J, Kang S, Park Y, et al., 2022. Networked SSD: flash memory interconnection network for high-bandwidth SSD. 55th IEEE/ACM Int Symp on Microarchitecture, p.388-403. https://doi.org/10.1109/micro56248.2022.00038

Lee J, Kim Y, Shipman GM, et al., 2013. Preemptible I/O scheduling of garbage collection for solid state drives. *IEEE Trans Comput Aided Des Integr Circ Syst*, 32(2):247-260. https://doi.org/10.1109/TCAD.2012.2227479

Li JH, Wang QP, Lee PPC, et al., 2020. An in-depth analysis of cloud block storage workloads in large-scale production. IEEE Int Symp on Workload Characterization, p.37-47. https://doi.org/10.1109/iiswc50251.2020.00013

Mao B, Wu SZ, Duan LD, 2018. Improving the SSD performance by exploiting request characteristics and internal parallelism. *IEEE Trans Comput Aided Des Integr Circ Syst*, 37(2):472-484. https://doi.org/10.1109/TCAD.2017.2697961

Nadig R, Sadrosadati M, Mao HY, et al., 2023. Venice: improving solid-state drive parallelism at low cost via conflict-free accesses. Proc 50th Annual Int Symp on Computer Architecture, p.1-16. https://doi.org/10.1145/3579371.3589071

Narayanan D, Thereska E, Donnelly A, et al., 2009. Migrating server storage to SSDs: analysis of tradeoffs. Proc 4th ACM European Conf on Computer Systems, p.145-158. https://doi.org/10.1145/1519065.1519081

Paik JY, Cho ES, Jin RZ, et al., 2018. Selective-delay garbage collection mechanism for read operations in multichannel flash-based storage devices. *IEEE Trans Consum Electron*, 64(1):118-126. https://doi.org/10.1109/TCE.2018.2812062

Qiu YH, Yin WB, Wang LL, 2021. A high-performance open-channel open-way NAND flash controller architecture. 31st Int Conf on Field-Programmable Logic and Applications, p.91-98. https://doi.org/10.1109/fpl53798.2021.00023

Ren TY, Du YJ, Cui JH, et al., 2025. Device-level optimization techniques for solid-state drives: a survey. https://doi.org/10.48550/arXiv.2507.10573

Sha ZB, Li J, Song LH, et al., 2021. Low I/O intensity-aware partial GC scheduling to reduce long-tail latency in SSDs. *ACM Trans Archit Code Optim*, 18(4):46. https://doi.org/10.1145/3460433

Tavakkol A, Arjomand M, Sarbazi-Azad H, 2013. Network-on-SSD: a scalable and high-performance communication design paradigm for SSDs. *IEEE Comput Arch Lett*, 12(1):5-8. https://doi.org/10.1109/l-ca.2012.4

Tavakkol A, Gómez-Luna J, Sadrosadati M, et al., 2018. MQSim: a framework for enabling realistic studies of modern multi-queue SSD devices. 16th USENIX Conf on File and Storage Technologies, p.49-65.

Wang Y, Sun ZB, Zhou Y, et al., 2024. Balloon-ZNS: constructing high-capacity and low-cost ZNS SSDs with built-in compression. Proc 61st ACM/IEEE Design Automation Conf, p.125. https://doi.org/10.1145/3649329.3657368

Yan SQ, Li HC, Hao MZ, et al., 2017. Tiny-tail flash: near-perfect elimination of garbage collection tail latencies in NAND SSDs. *ACM Trans Stor*, 13(3):22. https://doi.org/10.1145/3121133