Digital Object Identifier 10.1109/ACCESS.2017.DOI

EyanaSSDSim: Explore the Inner Workings of Solid-State Drives with Data Visualization

MD HABIBUR RAHMAN [1], OMAR FAROQUE [2], CHOE MUN SEOK [3], AND JAEHO KIM [4]
[1]Dept. AI Convergence Engineering, Gyeongsang National University, Jinju, South Korea (e-mail: habib@gnu.ac.kr)
[2]Dept. Computer Science, The University of Texas at Austin, Texas, USA (e-mail: faroque.ae@gmail.com)
[3]Dept. AI Convergence Engineering, Gyeongsang National University, Jinju, South Korea (e-mail:ch011015@gnu.ac.kr)
[4]Dept. AI Convergence Engineering, Gyeongsang National University, Jinju, South Korea (e-mail: jaeho.kim@gnu.ac.kr)

Corresponding author: Jaeho Kim (e-mail: jaeho.kim@gnu.ac.kr).

ABSTRACT Massive block I/O systems serve as the backbone of many of today's critical applications, including financial trading platforms, data streaming platforms, cloud storage, and social media platforms. Since these applications involve data-intensive operations, the performance of the data storage system directly affects the user experience. The scale and complexity of the applications workloads make it challenging to identify and resolve issues when they arise. Visualization plays a key role in uncovering workload patterns and enhancing our understanding.

~~This work introduces~~ `EyanaSSDSim` ~~, a web-based SSD simulator designed to analyze and visualize the internal workings of NAND flash-based SSDs. Drawing from experience in analyzing and visualizing applications workloads,~~ `EyanaSSDSim` ~~provides visual and quantitative insight into workload patterns and data placement relationships and their impact on storage performance and lifespan. Our simulator visually provides the process of how the workload is processed inside SSDs, enabling various analyses through it.~~
`EyanaSSDSim` ~~enables real-time tracking of data movement, invalidation, and erasure, supporting trace uploads for custom analysis. It provides workload characterization by detecting patterns like sequentiality, randomness, and transitions (e.g., from sequential to random writes), and maps real-world traces to closest synthetic patterns (e.g., sequential, uniform random, Zipf) for optimizations like adaptive GC or allocation tuning.~~

is a web-based SSD simulator that provides real-time visual tracking of internal SSD operations, including data placement, page invalidation, garbage collection, and block erasure. The simulator enables performance analysis through quantitative metrics such as write amplification factor (WAF), wear-leveling distribution, and workload characterization. We validate `EyanaSSDSim` against established simulators (FEMU, FTLSim), demonstrating consistent results. As an accessible, installation-free tool, `EyanaSSDSim` bridges the gap between SSD complexity and comprehension for both research and education.

~~The simulator aims to bridge the gap between complexity and comprehension, offering an intuitive tool for learners and researchers. By enabling workload trace uploads and real-time visual comparisons,~~ `EyanaSSDSim` ~~simplifies performance analysis and helps identify issues, making it easier to understand SSD behavior and optimize performance.~~

INDEX TERMS Flash SSD Simulator, Data Placement, Visualization, Write Amplification, Wear Leveling, Garbage Collection, Flash Translation Layer

I. INTRODUCTION

NAND flash memory-based solid-state drives (SSDs) are widely used as data storage in a wide range of computer platforms today, such as mobile devices, cloud computing platforms, autonomous vehicles, etc.

~~The internal structure of SSDs, including firmware, is evolving to become more complex due to optimization technologies to improve performance, lifespan, energy efficiency, etc. One crucial aspect of these optimizations is data placement, which directly impacts SSD performance and durability. The data placement on flash devices is influenced by various allocation policies, dictating how data is distributed~~

~~within the SSD. Whenever data is updated, it is moved to a different location: the existing data is rewritten on a clean page, while the previous location is invalidated. This process is managed by the flash translation layer (FTL), responsible for mapping logical addresses to physical addresses. To maintain a pool of clean blocks, the garbage collection process periodically erases blocks with invalid pages, copying valid pages to other available blocks. This requires additional space allocation, known as over-provisioning space (OPS). These internal writes, known as write amplification, contribute to data movement within the device. However, accurately quantifying write amplification is not always straightforward and requires a deep understanding of the underlying causes of data movement within each device.~~

~~While current simulators output internal state and statistics, deriving internal processes from these outputs can be challenging due to the complexity and inter-dependencies of SSD operations. For instance, understanding how garbage collection (GC) was triggered, how many times a block was erased, and when these events occurred requires detailed tracking of low-level operations. Additionally, visualizing block erases, invalid page distributions, and other internal behaviors can be difficult without clear representation and logging mechanisms. The increasing complexity of state-of-the-art flash management calls for new research and analysis techniques. Our on-demand web-based visualization tool offers a more intuitive way to understand data layout and movement on complex devices. We presents a novel technique that enables full visual tracking of data placement and movement inside SSDs over time. Unlike traditional simulators that only output logs or summary metrics, our simulator allows users to visually observe how data is written, invalidated, erased, and relocated within SSD blocks at each moment of execution. This real-time visualization reveals critical data characteristics-such as sequentiality, randomness, grouping behavior, and write locality-making it easier to understand workload patterns and identify potential performance bottlenecks or irregularities. A key contribution is the ability to analyze SSD behavior dynamically, enabling researchers to spot inefficiencies or issues (e.g., excessive GC, poor wear-leveling) as they emerge. The system also supports real-world workload analysis, providing insight into trace characteristics and how they evolve over time. Users can detect transitions in data behavior, e.g., shifts from sequential to random writes, and interpret the impact on internal SSD operations. Furthermore, the simulator enables mapping real-world traces to their closest synthetic workload patterns, allowing workload-specific firmware optimizations such as adaptive GC tuning or allocation adjustment. This integration of visualization and modeling supports intelligent, data-driven SSD optimization. Moreover, as a fully web-based tool, the simulator is accessible without setup, making it ideal for both research and educational environments. This combination of visualization, interactivity, trace analysis, and accessibility represents a significant step forward in SSD simulation and workload understanding.~~

Understanding the internal behavior of SSDs is essential for optimizing storage performance and extending device lifespan. Key factors such as data placement, garbage collection (GC), wear-leveling, and write amplification factor (WAF) are tightly interconnected and significantly affect both performance and endurance [1]–[3]. However, analyzing these internal operations is challenging because they involve complex, dynamic interactions that are difficult to observe and quantify [4]–[6].

Existing SSD simulators [7]–[15] provide valuable performance metrics and statistics, but they primarily output logs or summary data, making it difficult to visualize and interpret internal SSD operations in real time. Understanding how GC is triggered, tracking block erase counts, and observing invalid page distributions require detailed, intuitive visualization that current tools do not adequately provide.

To address these limitations, we present `EyanaSSDSim`, a web-based SSD simulator that provides real-time visualization of internal SSD operations. Unlike existing simulators, `EyanaSSDSim` enables users to visually track data placement, page invalidation, block erasure, and GC processes as they occur. The simulator supports workload trace uploads, allowing researchers to analyze real-world workloads and compare them with synthetic patterns (sequential, uniform random, Zipf) for firmware optimization.

The major contributions of this paper are as follows:

- **Real-time visual tracking**: A novel visualization technique that enables full tracking of data placement and movement inside SSDs, revealing workload characteristics such as sequentiality, randomness, and write locality (Section IV-B).
- **Quantitative wear-leveling analysis**: Introduction of metrics including Degree of Invalid Page Distribution (DoIPD) and Degree of Erase Count (DoEC), along with Fourier Transform-based analysis to evaluate wear-leveling performance across different allocation policies (Section V-C).
- **Workload characterization and mapping**: A new metric called Workload Similarity Composition (WSC) that classifies real-world traces into synthetic workload patterns, enabling workload-aware firmware optimizations (Section V-E).
- **Web-based accessibility**: An on-demand, installation-free simulator accessible via web browser, validated against established simulators (FEMU, FTLSim) with 95% user satisfaction from 1,011 survey participants (Section VI, VIII).

In the remainder of this paper, we introduce related work and comparative study in Section II. The features, architecture, and functionality of our SSD simulator are described in Section III. In Section IV, we showcase the simulator's core capabilities through vivid visualizations of internal SSD operations across representative workloads. Leveraging these visualizations, Section V provides in-depth analyses of key phenomena including data placement dynamics, write amplification (WAF), garbage collection (GC) efficiency, and wear-leveling revealing patterns. In Section VI, we validate our simulator ~~by comparing a key indicator with existing simulators~~against FEMU and FTLSim. Section VII explores broader implications and future enhancements. Section VIII presents feedback from users of our simulator, and Section IX concludes our work. Readers are encouraged to explore our on-demand web-based simulator for demonstration purposes[1], and the publicly available code[2].

II. RELATED WORK AND COMPARATIVE STUDY

~~As a general rule, most~~ Most storage products include visualization tools~~, typically~~ showing ~~plots of input/output operations per second (IOPS),~~ throughput, and latency over time. What sets our visualizations apart is the ability to ~~effectively~~ display access patterns within the internal SSD blocks. ~~Using a two-dimensional display, we can visualize the motion of data from the beginning to the end of a workload.~~ MQSim [7] offers ~~an extensive set of~~ features~~, but may present challenges or require understanding the complex components of SSDs and internal workings of FTL~~. FlashSim [16] provides ~~versatile~~ simulation capabilities ~~that meet educational and research needs~~. ~~Meanwhile,~~ SSDModel [8] and other traditional simulators ~~often~~ demand significant ~~study~~ time for installation~~ and understanding~~. NANDFlashSim [10], VSSIM [11], WiscSim [12], SSDPlayer [17] and SimpleSSD [15] offer varying functionality and installation complexities. ~~However, our proposed simulator,~~ EyanaSSDSim ~~,~~ distinguishes itself by prioritizing analytical, educational and research-oriented design. EyanaSSDSim ~~further~~ enhances accessibility and interactivity through on-demand web access, real-time monitoring, and ~~comprehensive~~ documentation~~, which highlights its importance as an educational resource, analytical and research tool in the SSD simulation domain~~.

Table 1 summarizes the key features of EyanaSSDSim and their availability in other simulators. This comparison highlights the gaps in the literature, particularly the lack of real-time visualization and web-based accessibility, which EyanaSSDSim addresses. Real-time monitoring and visualization have proven valuable in other domains such as IoT-based monitoring systems [18], demonstrating the importance of intuitive visual feedback for system analysis.

III. NAVIGATING OUR SSD SIMULATOR: FEATURES, ARCHITECTURE, AND FUNCTIONALITY

Fig. 1 illustrates a high-level overview of EyanaSSDSim. Our simulator manages incoming I/O requests through several layers. ~~Initially, a~~ A request is received either from the Host Interface Layer (HIL) or by uploading a trace file. ~~When a workload is uploaded,~~ HIL process each request, schedules them ~~appropriately~~, and assigns them for execution on the SSD. The Flash Translation Layer (FTL) then translates the ~~corresponding~~ targeted address. ~~Next,~~ The Allocation Scheme

[1]https://ssd.qbithabib.com

[2]https://github.com/bithabib/flash_ssd_simulator_web

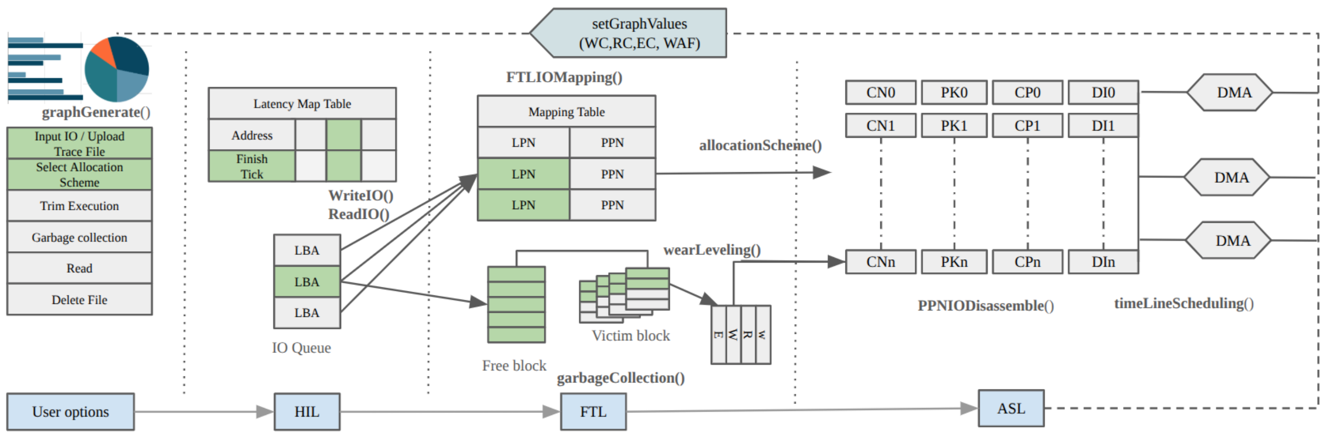
TABLE 1: Comparison of EyanaSSDSim features with other simulators in terms of inner workings and visualization

Feature	Description	Availability
Wear-leveling Analysis	Analyze wear-leveling using erase count and valid, invalid page count	a, b, c, d, e, i, j, l
Valid and Invalid Page Layout	Displays the layout of valid and invalid pages	a, l
GC Visualization	Visualizes garbage collection processes	a, e
Over Provisioning Visualization	Shows the over-provisioned space and its usage	a only
Data Placement Layout	Visualizes data placement based on erase count, write count, and valid/invalid pages	a, c, l
Write Amplification Factor (WAF) Analysis	Analyze the write amplification factor	All
Allocation Policies	Displays data placement based on different allocation policies	a, c
Dynamic SSD Size	Allows for the simulation of SSDs with varying sizes	a, d, e, i, j, k
Web-Based Access and Monitoring	Provides web-based access for ease use and real-time monitoring	a only
Automatic Log File Downloads	Automatically downloads logs for further analysis	a only
Real-time Visualization	Visual tracking of data movement in real-time	a only
Workload Trace Upload	Supports uploading custom workload traces	a, b, c, f, g, h, i, j

a. EyanaSSDSim, b. MQSim [7], c. SSDModel [8], d. SSDSim [9], e. NANDFlashSim [10], f. VSSIM [11], g. WiscSim [12], h. SimpleSSD [13], i. FEMU [14], j. FTLSim [15], k. FlashSim [16], l. SSDPlayer [19]

Layer (ASL) processes the request based on the host-selected allocation policy, which determines how data is distributed across the flash memory ~~to optimize performance and endurance. This abstraction accounts for the physical layout of interconnection buses and flash dies.~~ Upon completion of the I/O request, data is transferred to the host system using direct memory access (DMA)~~, minimizing central processing unit (CPU) overhead and improving efficiency~~. The ASL reports back to the host-side controller. ~~Additionally, it~~ It monitors the data written into the SSD and signals the controller to execute GC when necessary. GC reclaims space by erasing invalid data blocks~~, ensuring efficient storage management~~. The process starts when usage reaches GC-threshold-percent-high (99.9995%) and continues until it reduces to GC-threshold-percent (99.9990%). ~~Furthermore, users~~ Users can manually trigger TRIM operations from the host side to free unused data blocks. ~~TRIM is an ATA (Advanced Technology Attachment) command that allows the operating system to inform the SSD which data blocks are no longer in use and can be erased internally.~~ The OPS is a reserved portion of the SSD that enhances wear-leveling, performance, and GC efficiency. Table 2 presents the range of adjustable parameters for our NAND flash device simulator, with the corresponding experimental values provided in parentheses.

A. COMPREHENSIVE FIRMWARE SIMULATION



HIL: Host Interface Layer, **FTL:** Flash Translation Layer, **LPN:** Logical Page Number, **PPN:** Physical Page Number, **ASL:** Allocation Scheme Layer, **WC:** Write Count, **RC:** Read Count, **ER:** Erase Count, **WAF:** Amplification Factor, **DMA:** Direct Memory Access, **CN:** Channel, **PK:** Package, **CP:** Chip, **DI:** Die

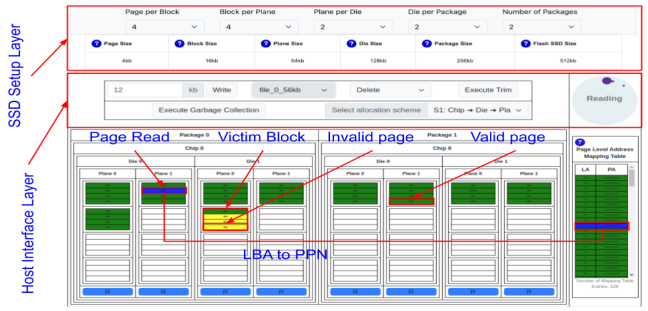
ReadIO: Read I/O requests from the host or an uploaded file, **WriteIO:** Send I/O requests to the FTL, **FTLIOMapping:** Translate LPNs to PPNs, **PPNIODisassemble():** Find duplicate LPNs, **allocationScheme():** Determines where new data should be place, **wearLeveling():** Ensures LBAs distribution across blocks evenly, **garbageCollection():** Selecting a victim block with the highest number of invalid pages, migrating valid data, and updating the address table and releases the block for reuse, **timeLineScheduling():** Process each request, schedules them appropriately, and assigns them for execution on the SSD, **setGraphValues (WC,RC,EC, WAF):** Prepare data for generate graph, **graphGenerate():** Generate graph.

FIGURE 1: High-level architecture of `EyanaSSDSim`, showing the data flow from host interface through Flash Translation Layer (FTL) and Allocation Scheme Layer (ASL) to NAND flash memory, including garbage collection (GC) and TRIM operations

TABLE 2: NAND flash specifications of `EyanaSSDSim`

Parameter	Value	Parameter	Value
PS	4096B	No. of Channel	2(2)
SS	512B-4KB (512B)	PPC	01-02 (02)
SPP	01-32 (08)	Total Capacity	3.27MB-536GB (5.03GB)
PPB	64-256 (256)	GC policy	Greedy
BPP	100-1000 (300)	Allocation Policy	S1-S6 (S1-S5)
PPD	01-04 (04)	OPS(%)	Any% (10,20,25)
DPC	01-04 (02)	GC_thres_pcent	99.9990
CPC	01-02 (1)	GC_thres_pcent_high	99.9995

a. PS: Page Size, b. SS: Sector Size, c. SSP: Sector Per Page, d. PPB: Page Per Block, e. BPP: Block Per Plane, f. PPD: Plane Per Die, g. DPC: Die Per Chip, h. CPC: Chip Per Channel



Page per Block 4, Block per Plane 4, Plane per Die 2, Die per Package 2, Number of Packages 2
Page Size 4kb, Block Size 16kb, Plane Size 64kb, Die Size 128kb, Package Size 256kb, Flash SSD Size 512kb
12 kb Write file_0_56kb Delete
S1: Chip → Die → Pla

FIGURE 2: Web interface of the basic SSD simulator showing user controls for I/O operations, allocation scheme selection, and real-time visualization of write count (WC), read count (RC), erase count (EC), and WAF metrics

1) SSD Setup and Host Interface Layer

In `EyanaSSDSim`, users can specify I/O size or upload a trace file, choose an allocation scheme, and perform operations like TRIM and garbage collection shown in Fig. 1 and 2. After ~~performing~~ read and write operations, HIL creates graphs for WC, RC, EC, and WAF.

The Host Interface Layer (HIL) shown in Fig. 1 and 2 handles incoming requests, translating ~~user inputs or~~ workload trace files into logical block addresses (LBAs), request types, ~~sector counts,~~ and timestamps. It forwards these details to the FTL. ~~The HIL also maintains a latency map table for request completion times, updated asynchronously by simulation modules.~~

2) Flash Translation Layer

In `EyanaSSDSim`, the FTL shown in Fig. 1 handles I/O requests by maintaining a mapping table of logical and physical pages. For reads, it translates logical page numbers (LPNs) to physical page numbers (PPNs) ~~through the mapping table~~. For writes, it allocates new physical pages, updates the mapping table, and manages data placement based on user selected allocation scheme. When an update occurs, the previous physical location is marked invalid. ~~A free block is a block that contains no valid or invalid pages and is fully available for new data writes. The allocation scheme determines where new data should be placed, ensuring efficient utilization of free blocks.~~ When no free pages are available, the FTL initiates garbage collection (GC) using a greedy policy: ~~it~~ selecting a victim block with the highest number of invalid pages, migrating valid data, ~~and updating the mapping table~~ and releases the block for reuse. Wear-leveling ensures even LBAs distribution across blocks~~ and execute GC when necessary~~. The efficiency of GC is measured using the write amplification

TABLE 3: Variables used in static allocation schemes equations

Variable	Explanation
lpn	Logical page number (LPN)
n_{ch}	The number of channels in an SSD
n_{cp}	The number of chips in a channel
n_{die}	The number of dies in a chip
n_{pl}	The number of planes in a die
$n_{\text{A-D}}$	The number of the component A-D

factor (WAF), defined as the ratio of SSD internal writes to host writes. ~~Additionally, user-defined OPS settings allow control over free block thresholds, enhancing SSD endurance and performance.~~

3) Page Allocation Schemes

FTL maps logical page numbers (LPN) to physical pages, with allocation schemes that influence SSD parallelism and data placement [4], [20]. Static allocation pre-assigns LPNs to channels, chips, dies, and planes using predetermined formulas, while dynamic allocation distributes erase operations more evenly, improving wear-leveling. ~~The priority order of the components in static schemes (e.g., channel (CN), package(PK), chip (CP), die (DI), and plane (PL)) slightly impacts the wear-leveling.~~

~~Based on our evaluation, we~~ We compared five static allocation strategies, S1 to S5. ~~Our results show that different policies perform better under different workloads.~~ Detailed descriptions ~~of this evaluation~~ are provided in subsection V-C.

We formularize a general law to determine the placement of a logical page based on any given priority sequence ~~of components~~. Table 3 presents ~~descriptions of~~ the variables used ~~in the allocation law~~. If the priority order is represented as A, B, C, and D, the general allocation laws are:

- Position of the first component (A):

$$A = \text{lpn}\% n_A$$

- Position of the second component (B):

$$B = \left\lfloor \frac{\text{lpn}}{n_A} \right\rfloor \% n_B$$

- Position of the third component (C):

$$C = \left\lfloor \frac{\text{lpn}}{(n_A \times n_B)} \right\rfloor \% n_C$$

- Position of the fourth component (D):

$$D = \left\lfloor \frac{\text{lpn}}{(n_A \times n_B \times n_C)} \right\rfloor \% n_D$$

Whenever we have a priority sequence, we apply this formula by taking the first component as A, the second as B, the third as C, and the fourth as D to calculate the placement of the logical page example given in Table 4.

Using this method, we can create formulas for any given allocation pattern based on the priority order of the components. In our simulator, we support six different allocation

TABLE 4: Examples of S1 Allocation Scheme

Allocation Scheme	Priority Order	Formulas
S1	chip(CP)→die(DI) →plane(PL) →channel(CN)	- Chip: chip = $\text{lpn}\% n_{\text{cp}}$ - Die: $\left\lfloor \frac{\text{lpn}}{n_{\text{cp}}} \right\rfloor \% n_{\text{die}}$ - Plane: $\left\lfloor \frac{\text{lpn}}{(n_{\text{cp}} \times n_{\text{die}})} \right\rfloor \% n_{\text{pl}}$ - Channel: $\left\lfloor \frac{\text{lpn}}{(n_{\text{cp}} \times n_{\text{die}} \times n_{\text{pl}})} \right\rfloor \% n_{\text{ch}}$

schemes: S2: CN→ CP→ DI→ PL, S3:CN→PL→CP→DI, S4: CN→ DI→ CP→ PL, S5: CN→ PL→ DI→ CP, and S6: CN→ DI→ PL→ CP.

IV. FULL SYSTEM EVALUATION

The main feature of our simulator is visualizations of ~~the complex operations of~~ an SSD. In this section, we show ~~the process of~~ visualizing and analyzing the internal operations that ~~significantly~~ affect the performance and lifespan of SSDs. ~~Recall that SSDs handle write requests by placing a new location, which causes data movement inside an SSD through GC operation.~~ We present visualization examples of data movement inside an SSD and introduce performance and wear-leveling analysis models. The configuration of the SSD simulator for evaluations is specified by values in parentheses in Table 2. In this evaluation, we use three ~~different~~ workloads: sequential, uniform random, and Zipf~~. distribution~~ with a total request volume of ~~each workload is~~ 5.03GB, 4KB write requests, and 80% valid blocks of the SSD capacity.

Implementation and System Configuration: `EyanaSSDSim` is implemented as a web-based application using Python (Flask framework) for the backend simulation engine and JavaScript for the frontend visualization interface. The simulator runs on a server with an Intel Xeon E5-2680 v4 processor (2.40GHz, 14 cores), 64GB RAM, and Ubuntu 20.04 LTS. The web interface is accessible via standard web browsers (Chrome, Firefox, Safari) without requiring any client-side installation. All experiments were conducted by executing workloads through the simulator's web interface, with results automatically logged and exported for analysis.

Evaluation Metrics: The following metrics are used throughout our evaluation:

- **Write Amplification Factor (WAF):** Ratio of total physical writes to logical writes, calculated as WAF = Physical Writes / Logical Writes. Lower WAF indicates better efficiency.
- **Degree of Invalid Page Distribution (DoIPD):** Standard deviation of invalid page counts across blocks, measuring GC efficiency (Equation 1).
- **Degree of Erase Count (DoEC):** Standard deviation of erase counts across blocks, measuring wear-leveling uniformity (Equation 2).
- **Workload Similarity Composition (WSC):** Percentage-based metric classifying real-world traces into synthetic workload patterns.

Table 5 provides a comprehensive summary of all mathematical notations used throughout this paper.

TABLE 5: Summary of mathematical notations used in this paper

Symbol	Definition
	General SSD Parameters
n	Total number of blocks in the SSD
N	Length of sequence (number of data points) in Fourier Transform
LPN	Logical Page Number
PPN	Physical Page Number
LBA	Logical Block Address
	Invalid Page Distribution (DoIPD)
I_i	Number of invalid pages in block i
μ_I	Mean number of invalid pages per block: $\mu_I = \frac{1}{n}\sum_{i=1}^{n} I_i$
DoIPD	Degree of Invalid Page Distribution (std. dev. of invalid pages)
	Erase Count Distribution (DoEC)
E_i	Erase count of block i
μ_E	Mean erase count per block: $\mu_E = \frac{1}{n}\sum_{i=1}^{n} E_i$
DoEC	Degree of Erase Count (std. dev. of erase counts, wear_degree)
	Fourier Transform Analysis
$x[n]$	Sequence of erase counts per block
$X[k]$	Fourier Transform of sequence at frequency k
$A[k]$	Amplitude of Fourier Transform: $A[k] = \lvert X[k]\rvert$
μ_A	Average amplitude: $\mu_A = \frac{2}{N}\sum_{k=0}^{N/2-1} A[k]$
σ_A	Standard deviation of amplitudes
k	Frequency component index in Fourier Transform
	Allocation Scheme Variables
n_{ch}	Number of channels in the SSD
n_{cp}	Number of chips per channel
n_{die}	Number of dies per chip
n_{pl}	Number of planes per die
n_{A-D}	Number of components A, B, C, or D in priority sequence

TABLE 6: Legend for block color and symbol representation in visualizations

Symbol/Color	Meaning	SSD Behavior Implication
Tick-sign	All pages are valid	Block actively storing live data
Cross-sign	All pages are obsolete	Ideal candidate for GC erasure
Dark green	High number of valid pages	Lower priority for GC
Bright green	High number of invalid pages	High priority for GC
White / Empty block	Free block (contains no data)	Ready for new data writes

A. VISUAL INTERPRETATION GUIDANCE

All visual symbols and color codes used in the SSD layout figures are explained in Table 6. ~~These include the meanings of tick and cross signs, and variations in green shading, which indicate the validity state of blocks. Figures such as Figure 3 also illustrate two key workload stages: the first stage (before garbage collection is triggered) and the second stage (after the workload completes).~~ This table serves as the reference for interpreting all SSD visualizations ~~throughout the paper~~.

B. COMPARISON OF DATA PLACEMENT FOR DIFFERENT WORKLOADS

Fig. 3 illustrates data placement layouts for the three workloads at two stages: The first stage is when user writes reach GC_thres_pcent_high as mentioned in Table 2 ~~of SSD capacity, right before GC occurs~~. The second stage is when the workload ~~streaming~~ is complete. Fig. 3a, 3b, and 3c show the data layout at the first stage, while Fig. 3d, 3e, and 3f represent the second stage.

Each subfigure represents a single channel with 3,072 blocks. `EyanaSSDSim` features two channels, as specified in Table 2. ~~Since both channels exhibit nearly identical behavior across all workloads, we present only a single channel for simplicity.~~ The total number of blocks in the SSD is 6,124 and each block is represented as color-coded square boxes (256 pages per block)~~: the darkest green blocks contain the highest number of valid pages, the brightest green blocks have the most invalid pages, cross-signed blocks are fully invalid, and tick-signed blocks are fully valid~~.

In Fig. 3a, sequential workloads exhibit a structured pattern of invalidation's and GC, resembling a kernel (CNN, image processing) [21] sliding over a grid of blocks. As data is written sequentially, updates cause older pages to become invalid in a ~~continuous,~~ wave-like manner. When the GC_thres_pcent_high reached, the SSD reclaims space by relocating valid pages and erasing blocks~~, progressing forward systematically~~. This kernel-like movement of GC ensures that erasures are spread evenly across the drive, minimizing ~~the migration of~~ valid pages~~. As a result, it~~ migration, reduces write amplification and lowers GC overhead.

In the Zipf workload, as shown in Fig. 3c and Fig. 3f, both ~~the initial and final~~ stages show a specific group of blocks highlighted in bright green, indicating that these blocks have been erased and rewritten ~~during the workload~~. Invalid pages tend to concentrate within this ~~specific~~ group. When GC occurs, blocks are selected from this group, and as data is rewritten, pages within this group are more likely to be invalidated. This leads to an increased erasure count for ~~the blocks in~~ this group, which can be detrimental to wear-leveling. However, this clustering of invalid pages also benefits garbage collection, as ~~entire~~ blocks can be efficiently reclaimed with minimal valid page migration. ~~As a result,~~ Zipf writes achieve a better WAF compared to sequential writes, as shown in Fig. 4 and Fig. 12 ~~(Detailed explanations of figures are given in section~~ V~~,~~ VI~~)~~.

In the uniform random workload shown in Fig. 3b and Fig. 3e, most blocks appear dark green, indicating that invalid pages are distributed across all blocks. ~~When erasure occurs, blocks can be selected from any part of the drive, and as data is written, pages are invalidated randomly throughout the SSD.~~ This results in an even distribution of erasure counts across all blocks, which is beneficial for wear-leveling~~, as it ensures uniform wear-leveling across the drive~~. However, this even distribution is less efficient for GC. Since valid pages are scattered across blocks, it leads to higher write amplification~~, as more valid pages need to be migrated during~~

~~the GC process~~.

In summary, ~~our observations indicate that~~ in terms of reducing the WAF, the Zipf workload is the most effective, followed by sequential and ~~then~~ uniform random. ~~Conversely, in terms of~~ For wear-leveling ~~efficiency~~, sequential ~~provides the most balanced~~ is best, followed by uniform random, and then Zipf ~~workloads~~. These results highlight the trade-offs between WAF reduction and wear-leveling ~~across different workload patterns~~.

C. IMPACT OF OPS ON WAF FOR VARIOUS WORKLOADS

Fig. 4 compares WAFs for the sequential, uniform random, and Zipf workloads with 10% and 20% OPS. ~~The WAF decreases when SSDs operate efficiently, contributing to a longer lifespan.~~ Each line represents a different workload~~, illustrating how WAF evolves over time during write operations~~. The gray dots mark the peak WAF points, while the black dots indicate the final WAF ~~for each workload~~.

For the sequential workload, the WAF starts at 1.0 and ~~gradually~~ increases to ~~approximately~~ 1.35 with 10% OPS. ~~However,~~ With 20% OPS, WAF decreases to ~~around~~ 1.2 due to reduced GC demands. ~~In contrast,~~ The uniform random workload shows a sharp increase in WAF, starting at 1.0 and peaking at 2.58 with 10% OPS. After reaching this peak, WAF ~~gradually~~ declines~~, ending at approximately 2.39~~. ~~Similarly,~~ With 20% OPS, WAF reaches a maximum of 1.41 ~~before gradually decreasing to 1.17, emphasizing the benefits of additional space for GC efficiency~~. Zipf workloads demonstrate the lowest WAF ~~among all cases~~. WAF starts at 1.0, reaching ~~around~~ 1.07 with 10% OPS and ~~further~~ decreasing to ~~approximately~~ 1.03 with 20% OPS. ~~This improvement is attributed to concentrated write patterns, which enhance GC efficiency.~~

In summary, increased OPS reduces WAF across all workloads, with uniform random ~~patterns~~ gaining the most ~~from higher OPS~~.

V. ANALYSIS OF THE SIMULATION RESULTS

A. ANALYSIS OF RELATIONSHIP BETWEEN WAF AND INVALID PAGE DISTRIBUTION

The distribution of ~~variations in the number of~~ invalid pages per block ~~significantly~~ impacts SSD performance and longevity by influencing GC and wear-leveling ~~processes~~. Invalid pages, created by updates or deletions, must be efficiently managed to minimize WAF ~~, enhance endurance, and improve overall performance~~.

Fig. 5 presents the distribution of ~~variations in the number of~~ invalid pages per block for the sequential, uniform random, and Zipf workloads. Each histogram is overlaid with a bell curve to highlight the mean and Degree of Invalid Page Distribution (DoIPD)~~, which represents the standard deviation of invalid page distribution. The degree of invalid page distributions represented~~ by Equation 1 ~~to evaluate the distribution of variations in the number of invalid pages per block to understand the relationship between the distribution of variations and the efficiency of GC~~. The mean of invalid pages per block is expressed as $\mu_I = \frac{1}{n}\sum_{i=1}^{n} I_i$, where n is the number of blocks in an SSD, and I is the number of invalid pages per block.

$$DoIPD = \sqrt{\frac{\sum_{i=1}^{n}(I_i - \mu_I)^2}{n}} \tag{1}$$

The Zipf workload (μ : 25.23, DoIPD: 58.50) exhibits a broad distribution, with some blocks accumulating ~~significantly~~ more invalid pages. This high concentration ~~of distribution~~ supports efficient GC and lowers ~~write amplification factor (~~WAF~~)~~, making it optimal for WAF minimization. The sequential workload (μ : 25.26, DoIPD: 39.04) offers a balance between WAF and wear-leveling~~, providing moderate GC efficiency and wear-leveling performance~~.

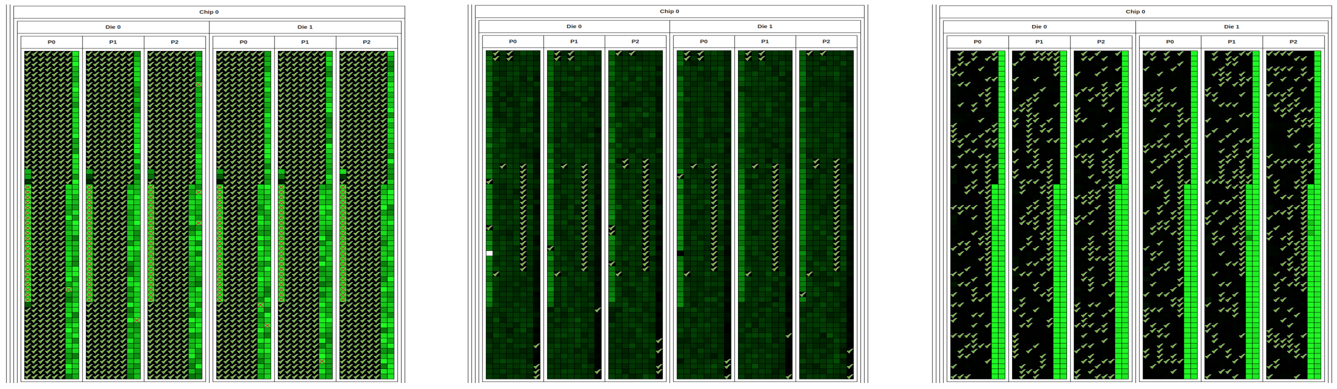
~~In contrast,~~ The uniform random pattern (μ : 25.40, DoIPD: 26.98) demonstrates a more even distribution of invalid pages. This uniformity is ~~highly~~ beneficial for wear-leveling~~, as it ensures write operations are distributed evenly across all blocks, reducing the risk of premature wear-leveling on specific blocks and enhancing the SSD's lifespan~~. However, ~~the downside is~~ lower GC efficiency~~. Since invalid pages are evenly spread,~~ results in more pages ~~need~~ to be copied during garbage collection, ~~increased~~ increasing WAFs compared to the Zipf and sequential workloads shown in Fig. 4

B. ANALYZING WEAR-LEVELING FOR VARIOUS WORKLOADS

Since flash memory has limited P/E cycles, wear-leveling of blocks on an SSD is necessary. ~~We examine the impact of wear-leveling according to workload patterns.~~ We use a simple formula to access the degree of wear-leveling ~~of blocks~~ as follows. The mean of erase counts is expressed as Mean $\mu_E = \frac{1}{n}\sum_{i=1}^{n} E_i$, where n is the number of blocks and E is the erase counts per block. The distribution of erase counts is quantified by Equation 2~~, which evaluates the variability in erase count distribution to understand its impact on wear-leveling~~.

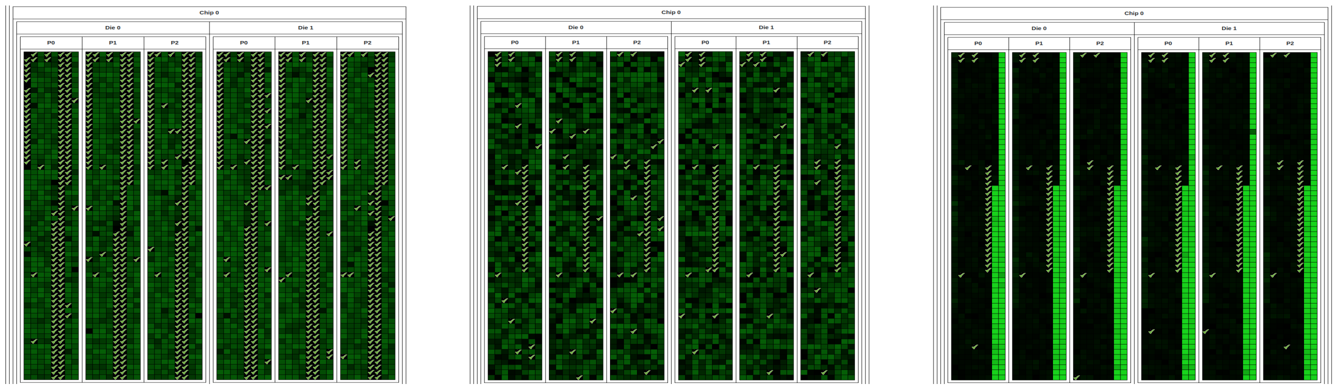
$$DoEC(wear\_degree) = \sqrt{\frac{\sum_{i=1}^{n}(E_i - \mu_E)^2}{n}} \tag{2}$$

Fig. 6 presents a histogram with bell curves to analyze the erase counts for SSD blocks under three ~~different~~ access patterns: sequential, uniform random, and Zipf ~~.~~ workloads. The sequential workload~~, depicted in blue,~~ has the lowest mean erase count (μ = 1.65) and the smallest ~~degree of erase count (~~DoEC~~), or wear_degree, which represents the standard deviation of erase count distribution~~ ($DoEC$ = 2.92), indicating ~~that the~~ wear on the SSD blocks is more uniform ~~and predictable~~. The uniform random workload ~~(green)~~ shows higher mean erase counts (μ = 3.02) and greater variability ($DoEC$ = 4.38), while Zipf ~~. workload (red)~~ shows the highest variability ~~with the largest standard deviation~~ ($DoEC$ = 4.93) suggesting uneven wear. ~~Based on these observations,~~ The sequential workload is ~~the most~~ optimal for wear-leveling~~,~~



(a) Sequential (b) Uniform random (c) Zipf

First stage: When user writes reach GC_thres_pcent_high of the SSD



(d) Sequential (e) Uniform random (f) Zipf

Second stage: When the workload streaming is complete

FIGURE 3: Visualization of SSD block states at two stages (before GC trigger and after workload completion) for sequential, uniform random, and Zipf workloads. (The darkest green block: the most valid pages, the brightest green: the most invalid pages, cross-signed blocks: fully invalid, and tick-signed: fully valid)

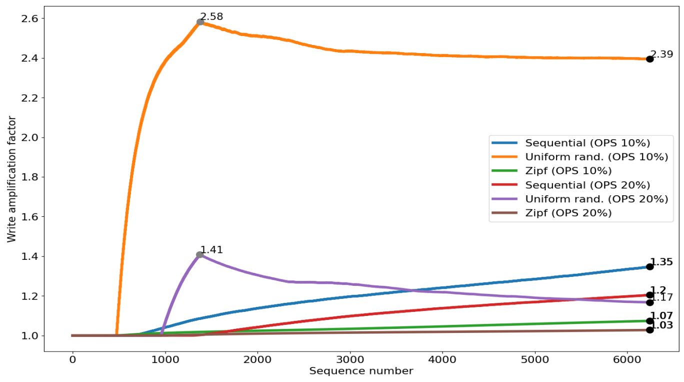


FIGURE 4: WAF progression over time for sequential, uniform random, and Zipf workloads comparing 10% vs 20% OPS configurations. Gray dots indicate peak WAF, black dots show final WAF values. Higher OPS reduces WAF across all workloads

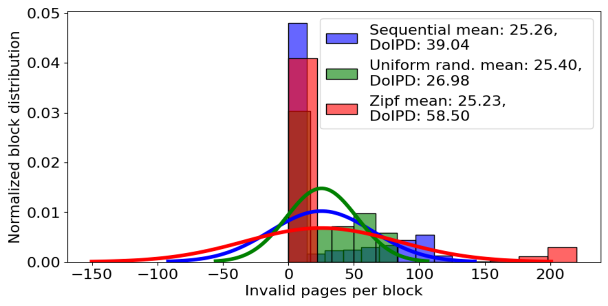


FIGURE 5: Histogram showing the distribution of invalid pages per block with overlaid bell curves for sequential, uniform random, and Zipf workloads. Mean (μ) and DoIPD values indicate the concentration of invalid pages, where higher DoIPD suggests better GC efficiency

~~as it promotes a more consistent and even distribution of wear across the SSD blocks, enhancing the overall longevity~~

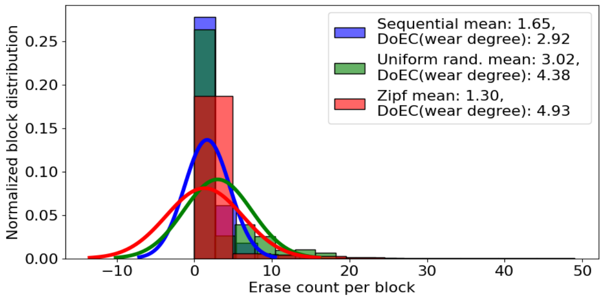


FIGURE 6: Histogram of erase count distribution per block with bell curves for sequential, uniform random, and Zipf workloads. Lower mean (μ) and DoEC indicate better wear-leveling; sequential workload achieves the most uniform wear distribution

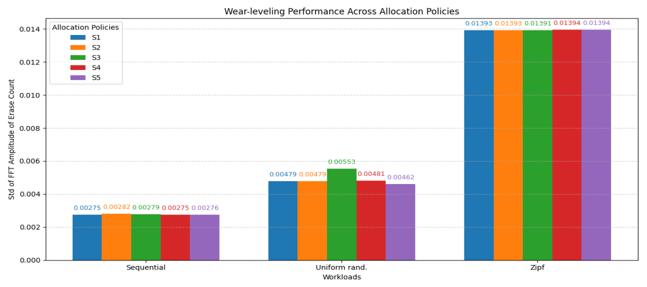


FIGURE 7: Heatmap comparing standard deviation of Fourier Transform amplitudes (σ_A) across five allocation policies (S1-S5) for sequential, uniform random, and Zipf workloads. Lower values (darker colors) indicate better wear-leveling performance

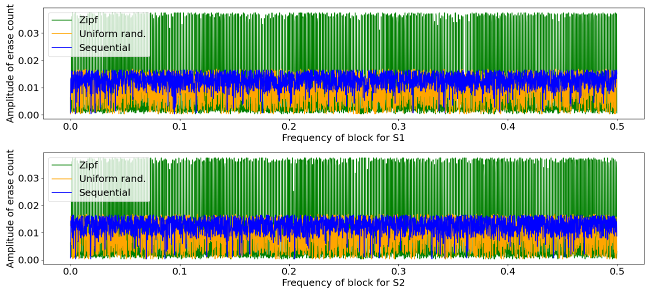


FIGURE 8: Fourier Transform amplitude spectra of erase counts for sequential, uniform random, and Zipf workloads. X-axis shows frequency (block variation rate), Y-axis shows amplitude magnitude. Zipf exhibits highest peaks indicating concentrated erasures on specific blocks

~~and reliability of the SSD~~. According to the invalid page distribution at the first stage in Fig. 3b, the uniform random ~~access pattern~~ appears ~~to be~~ best for wear-leveling. However, the erase count analysis in Fig. 6 and the second stage in Fig. 3e show that sequential ~~access pattern~~ is better for wear-leveling. ~~As per the DoIPD values in Fig.~~ 5~~, uniform random shows 26.98, whereas sequential has 39.04, indicating that invalid pages are more evenly distributed in the uniform random workload. However, Fig.~~ 5 ~~and DoEC values suggest that the sequential workload achieves a better erase distribution.~~ This is due to the inherently sequential nature of the workload, as explained ~~above~~ in sub-section IV-B.

C. FOURIER TRANSFORM INSIGHTS: WEAR LEVELING PERFORMANCE COMPARISON BETWEEN DIFFERENT ALLOCATION SCHEMES

Allocation policies in data storage systems are critical for performance and longevity as they determine data distribution across storage blocks, affecting read/write speeds, wear-leveling, and efficiency [4]. ~~Our~~ `EyanaSSDSim` can be used ~~as an analytical tool~~ to analyze different allocation policies.

~~To gain deeper insights into erase count distributions across different workloads, we explore approaches beyond direct comparison of erase count values.~~ Simply comparing raw erase count differences did not reveal meaningful insights~~, as statistical similarities made it difficult to distinguish workload-dependent behaviors~~. To address this, we applied the Fourier Transform to analyze the time-domain data in the frequency domain~~, which provided several advantages~~. Below is the mathematical representation~~ of the Fourier Transform and calculating the standard deviation of the amplitudes~~:

Let $x[n]$ be the sequence of erase counts per block, where n ranges from 0 to $N-1$. The Fourier Transform $X[k]$ of this sequence is given by: $X[k] = \sum_{n=0}^{N-1} x[n]e^{-i\frac{2\pi}{N}kn}$. The amplitude $A[k]$ of the Fourier Transform at frequency component k is: $A[k] = |X[k]|$. The average amplitude μ_A is: $\mu_A = \frac{2}{N}\sum_{k=0}^{N/2-1} A[k]$. The standard deviation of the amplitudes σ_A is Equation 3 :

$$\sigma_A = \sqrt{\frac{2}{N}\sum_{k=0}^{N/2-1}(A[k]-\mu_A)^2} \tag{3}$$

In the Fourier Transform results shown in Fig. 8, the x-axis represents the frequency of block which means variations in erase count between blocks~~, where each block corresponds to 1 second (or 1 time step), and the total number of blocks is 6,124. The frequency values on the x-axis indicate how often variations in the erase counts occur~~.

The y-axis represents the amplitude, indicating the magnitude of these variations. ~~A higher amplitude signifies greater fluctuations in erase counts across blocks, reflecting poor wear-leveling, while a lower amplitude indicates more uniform erase counts, signifying better wear-leveling. Therefore,~~ Lower variation in amplitude corresponds to better wear-leveling~~, as it signifies a more uniform distribution of erase counts~~.

This transformation revealed subtle periodic patterns~~, indicating that specific blocks exhibit higher erase counts in each frequency cycle~~. ~~This pattern suggests that a particular group of blocks consistently experiences the highest erase counts.~~ As highlighted in Fig. 3c, 3f, this distribution follows a Zipf-like behavior, where certain blocks undergo ~~significantly~~ more erasures than others, as shown in Fig. 8. ~~Moreover, differentiating between sequential and uniform random workloads are particularly difficult because their erase counts are very close as per the erase count distribution given in Fig. 6. However, by applying the Fourier Transform, we can clearly identify this difference specifically, in the sequential workload, the gap between blocks with high and low erase counts is smaller, which becomes more distinguishable in the frequency domain. The Fourier Transform thus serves as a valuable tool for uncovering hidden patterns that are not evident in direct comparisons.~~ Applying Fourier transformation [22] [23], we identified patterns in erase counts~~, on which~~ and applied standard deviation analysis to measure amplitude variations. ~~Our results show that different policies perform better under different workloads. For instance,~~ S1 and S4 demonstrate the lowest standard deviations for the sequential workload (see Fig. 7), indicating better wear-leveling ~~compared to other policies~~. ~~Similarly,~~ S5 exhibited lower standard deviations for the uniform random workload~~, suggesting better performance in this scenario~~. ~~On the other hand,~~ S3 shows improved results for the Zipf workload. This highlights `EyanaSSDSim` ability to differentiate between allocation strategies~~ and provide deeper insights into storage performance~~.

Based on the Fourier Transform results in Fig. 8 and the standard deviation values shown in Fig. 7, ~~we can conclude the following regarding wear-leveling performance:~~ the sequential workload yields the best wear-leveling, ~~the~~ uniform random ~~workload~~ performs well, and Zipf ~~. workload~~ performs the worst. Among the allocation schemes, S3 provides ~~slightly~~ better wear-leveling for Zipf~~.~~, S1 and S4 for sequential ~~workloads~~, while S5 shows better performance for uniform random ~~workloads compared to other allocation policies, though the difference is minimal~~.

D. REAL-WORLD WORKLOAD ANALYSIS: WAF AND INVALID PAGE DISTRIBUTION

In this section, we analyze the relationship between data placement inside the SSD and WAF in real-world workloads. We used two ~~real world~~ workloads TPC-C [24], [25] and MSR prxy [26]. Prxy was chosen ~~among the MSR traces~~ because it is highly write-intensive, with 97% ~~of its operations being~~ writes and ~~an average~~ I/O ~~write request~~ size of 7.07 KB [26], [27]~~, making it ideal for studying write-heavy workloads~~. TPC-C was selected because~~, although it has a mix of reads (64%) and writes (36%) requests, it still involves~~ it has a ~~significant number of write requests with a small average~~ I/O ~~write request~~ size of 7.55 KB ~~as per our measurement~~. Both workloads emphasize small write operations~~, making them suitable for analyzing storage systems' behavior under write-intensive conditions with small request sizes~~.

Fig. 9a illustrates the distribution of invalid pages across SSD blocks for different workloads and OPS configurations~~, where each dot represents the number of invalid pages per block, and the colored lines with arrows depict the cumulative DoIPD (C. DoIPD)~~. The table above Fig. 9a summarizes key metrics: ~~the Mean Invalid Page Count indicates the average number of invalid pages per block, in our case we got with~~ higher values correlating to lower WAF (e.g., prxy at 25% OPS has a mean of 21.8 and a WAF of 1.008, whereas TPC-C at 10% OPS has a mean of 5.4 and a WAF of 5.12).

~~However, this relationship does not always hold. If the distribution of invalid pages follows a zigzag pattern where one block has a high number of invalid pages, the next has a low count, and so on WAF can remain low despite having the same mean value. Conversely, if invalid pages are evenly distributed across blocks, WAF tends to be higher. To better analyze the impact of invalid page distribution on WAF, we introduce DoIPD, which accounts for the standard deviation of invalid pages per block. This provides a more comprehensive measure of the relationship between invalid page distribution and WAF.~~ The DoIPD represents how invalid pages are distributed across blocks, with higher values indicating greater variation (prxy at 25% OPS has 25.45, while TPC-C at 10% OPS has 2.16). ~~The WAF column highlights how inefficient invalid page distribution increases write amplification, as seen in TPC-C at 10% OPS, where WAF reaches 5.12, compared to 2.3 at 25% OPS.~~ The Fig. 9 visually confirms these trends~~, demonstrating how clustering of invalid pages significantly impacts garbage collection efficiency and SSD performance~~.

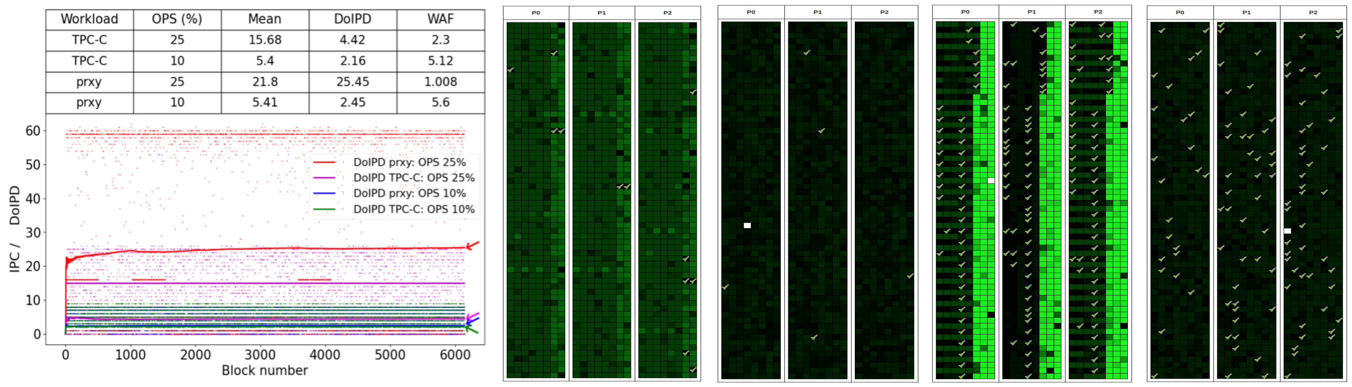
For TPC-C with 25% OPS ~~(purple dots)~~, invalid pages are evenly distributed~~, as shown in the data placement layout~~ in Fig. 9b, with ~~a moderate~~ DoIPD of 4.42. This ~~balanced~~ distribution results in ~~a moderate~~ WAF of 2.3 ~~given in Fig.~~ 9a~~, indicating controlled GC overhead~~. ~~However, for~~ For TPC-C with 10% OPS ~~(green dots)~~, the invalid page distribution is ~~even more~~ uniform~~, as depicted~~ in Fig. 9c, with ~~the lowest~~ DoIPD of 2.16. ~~While this improves wear-leveling, it leads to~~ This leads to a higher WAF of 5.12 due to increased GC ~~operations~~ at lower OPS.

For MSR prxy with 25% OPS ~~(red dots)~~, invalid pages are ~~highly~~ clustered~~, as illustrated in the data placement layout~~ in Fig. 9d, with ~~a high~~ DoIPD of 25.45. This clustering minimizes GC overhead, leading to ~~an extremely~~ low WAF of 1.008 ~~given in Fig.~~ 9a, but results in poor wear-leveling ~~since certain blocks endure excessive writes~~. ~~In contrast, for~~ For MSR prxy with 10% OPS ~~(blue dots)~~, invalid pages are more evenly distributed~~, as shown in the data placement layout~~ in Fig. 9e, with ~~a lower~~ DoIPD of 2.45. This enhances wear-leveling but ~~significantly~~ increases WAF to 5.6 ~~as shown in Fig.~~ 9a due to frequent GC ~~operations~~.

This analysis highlights a clear trade-off: clustering reduces GC overhead and lowers WAF but worsens wear-leveling, while ~~a more~~ uniform distribution improves wear-leveling but increases GC ~~activity, leading to a higher~~ and WAF.

E. WORKLOAD CHARACTERIZATION AND MAPPING

Simulator `EyanaSSDSim` uses each real-world workload trace to analyse and determine its underlying access characteristics. For each trace file, `EyanaSSDSim` continuously monitors the



Workload	OPS (%)	Mean	DoIPD	WAF
TPC-C	25	15.68	4.42	2.3
TPC-C	10	5.4	2.16	5.12
prxy	25	21.8	25.45	1.008
prxy	10	5.41	2.45	5.6

(a) Invalid page distribution. Dots represents invalid page count (IPC) and line represents DoIPD. (b) TPC-C (OPS 25%) (c) TPC-C (OPS 10%) (d) prxy (OPS 25%) (e) prxy (OPS 10%)

FIGURE 9: (a) Scatter plot of invalid page counts per block with cumulative DoIPD lines for TPC-C and prxy workloads at different OPS levels; (b-e) corresponding SSD block state visualizations showing the impact of OPS on data clustering. (The darkest green block: the most valid pages, the brightest green: the most invalid pages, cross-signed blocks: fully invalid, and tick-signed: fully valid)

sequence of logical block addresses (LBAs) and classifies them into uniform random, sequential or Zipf patterns. For example, in a sequential pattern, LBAs are written in strictly increasing order, indicating sequential writes, which is typical of logging or streaming data. In a uniform random pattern, LBAs are uniformly distributed across the entire address space, with each block having an equal probability of being written. In contrast, a Zipf distribution pattern follows an asymmetrical distribution, with a small subset of active areas (LBAs) receiving most writes, indicating strong data locality. In addition to these patterns, we also observed a fourth category, referred to as an irregular random pattern. In this pattern, the LBAs exhibit stochastic write behaviour that does not conform to the aforementioned patterns. The accesses are irregular and dispersed across the address space, but without clear locality or hot/cold regions [28]. To quantitatively represent the contribution of each access pattern in a given trace, we introduce a new metric called Workload Similarity Composition (WSC). WSC is defined as the percentage breakdown of uniform random, sequential, Zipf and irregular random accesses observed in the workload Table 7.

Once these access distributions are determined, we compare the four types of workload access ratios for each real-world workload with the synthetic workload profiles generated by `EyanaSSDSim` to find the closest match. In this comparison, we did not consider irregular random patterns, as they do not correspond to any defined synthetic workload profile. We identified this pattern while analyzing the workload behavior. As shown in Table 7, the TPC-C workload closely aligns with the uniform random workload when the OPS is set to 25%, 20%, and 15%. Although the WAF of TPC-C is higher than that of the synthetic uniform random workload, this can be attributed to the presence of high number of unique LBA as mentioned in Table 7 and a small proportion of Zipf, sequential, and irregular random patterns within TPC-C, which together result in limited data locality and weak sequential write behaviour. As OPS decreases, both the ratio of irregular random access patterns and the WAFs increase.

A similar trend is observed for the prxy workload, which exhibits a strong Zipf-like access pattern, leading to a slightly higher WAF with only a marginal difference. However, when the OPS is reduced to 15%, a noticeable increase in WAF occurs for prxy, as shown in Table 7. A similar increase in WAF is observed when the OPS is 5% for the synthetic Zipf pattern. This can be attributed to the fact that prxy has more frequently written LBAs compared to the Zipf pattern; therefore, the active area (where frequently written LBAs are located) is larger in prxy than in Zipf. As illustrated in Fig. 9d for prxy and Fig. 3f for the Zipf data placement layout, the bright-green regions indicate a high number of invalid pages, as indicated by Table 6 and represent the active areas approximately 35% for prxy and 19% for Zipf indicating that Zipf has a smaller active area. For this reason, a sudden WAF increase occurs at 15% OPS for prxy and at 5% OPS for Zipf.

Another observation when the OPS is reduced, the irregularity in data placement increases for the prxy workload, as demonstrated in Fig. 10. In Fig. 10a (initial state, with 5% OPS), the SSD is initially fully written(tick signed) except for the OPS-reserved region. After the OPS space is written shown in Fig. 10b, we can see that invalid pages are mostly concentrated in the 5% OPS region, referred to as the active area. When 40% of the workload write requests are completed shown in Fig. 10c, the active area becomes darker, indicating that invalid pages are now distributed throughout the SSD. At 60% workload completion shown in Fig. 10d, the active regions get even more darker, and some blocks appear completely dark, meaning that almost all pages in those blocks are valid. At this stage, WAF starts to increase, as invalid pages are distributed irregularly over a wider area. Finally, when the full workload has been written shwon in Fig. 10e, almost no distinct active area remains. In summary, invalid LBAs become irregularly distributed referred to as irregular random for all workloads when OPS is reduced. To compare the effect of OPS size, by comparing data placement layout 10% OPS in Fig. 9e with OPS 25% in Fig. 9d, we observe that a large OPS accommodates all active LBAs and suppresses the increase in WAF. This phenomenon occurs because the size of the OPS directly determines how many hot LBAs can be kept within a confined active area. When the OPS is sufficiently large (e.g., OPS 25% in Fig. 9d), the garbage collector can repeatedly reuse the reserved blocks for all hot LBAs, keeping invalid pages highly concentrated in a small bright-green region and maintaining excellent data locality. In contrast, with only 10% OPS Fig. 9e, the reserved space is no longer large enough to absorb all updates from the hottest LBAs. Once the active area overflows the OPS region, new versions of hot data must be written to previously cold blocks outside the original active area. This forces the garbage collector to scatter invalid pages across a much larger portion of the SSD, rapidly transforming the original localized (Zipf-like or sequential-like) access pattern into a diffuse irregular random pattern. Consequently, victim blocks selected for GC now contain far fewer invalid pages, dramatically increasing valid-page migration and therefore causing the sharp rise in WAF that is visible in Table 7 for both real-world traces and the synthetic Zipf workload at low OPS values. In addition, the increasing ratio of irregular random depends on the OPS size and workload type. For TPC-C, it increases gradually, whereas for prxy and Zipf, irregular random increases sharply depending on the number of frequently accessed LBAs.

By mapping real-world traces to synthetic traces, if we can determine what synthetic workload pattern (uniform random, irregular random, sequential or Zipf) a real-world trace most closely follows, then we can apply or tune existing SSD firmware policies (GC policies, allocation policies, OPS, etc.) that are already known to perform best for that pattern. For example, if your proxy workload behaves like a Zipf distribution, you can improve the performance and endurance of your proxy workload through firmware optimisations designed for Zipf workloads, such as proper OPS size, adaptive hot/cold region management [28] or zone-based allocation [29] [30]. Consistent with this approach, for optimization of prxy workload increasing the OPS from 20% to 25% yields only limited improvement in WAF, as shown in Fig. 9a and Table 7, because writes remain concentrated within compact hot/cold regions [28]; beyond a modest level, additional OPS provides diminishing returns. This indicates that maintaining an OPS of 20% instead of 25% can save up to 5% of SSD space without significantly affecting WAF.

F. LESSONS LEARNED AND KEY INSIGHTS

From our comprehensive analysis using `EyanaSSDSim`, several key insights emerge regarding SSD behaviour under diverse workloads and allocation strategies. The summarized results are presented in Table 8, which highlights each figure's observations, implications, and corresponding benefits.

In Table 8, No.1 - Visual tracking of data placement and movement provides a clear understanding of how different workload patterns influence internal SSD operations. The simulator reveals that Zipf workloads exhibit low WAF due to strong locality but suffer from uneven wear-leveling, whereas uniform random workloads achieve better wear-leveling at the cost of higher WAF. Sequential workloads strike a

TABLE 7: Validation of similarity between real-world and synthetic workloads using access patterns and WAF. (TPC-C, UR, prxy, and Zipf have 362K, 284K, 148K, and 108K unique LBAs, respectively)

WL (OPS%)	**UR (%)**	**IR (%)**	**SQ (%)**	**ZF (%)**	**Closest Synth.**	**OBS WAF**	**Synth. WAF**
TPC-C (25%)	75.3	10.2	2.4	12.1	UR	2.3	1.26
TPC-C (20%)	68.7	19.4	2.5	9.3	UR	2.84	1.41
TPC-C (15%)	62.3	29.2	1.9	7.5	UR	3.66	1.82
TPC-C (10%)	50.3	43.9	1.1	4.7	UR	5.12	2.58
TPC-C (5%)	36.2	62.2	0.5	1.1	UR	8.21	4.13
prxy (25%)	11.9	8.1	1.5	78.2	Zipf	1.008	1.004
prxy (20%)	9.7	13.5	1.5	75.3	Zipf	1.071	1.03
prxy (15%)	6.4	50.9	1.5	41.2	Zipf	3.43	1.05
prxy (10%)	5.7	83.6	1.3	9.4	Zipf	5.06	1.07
prxy (5%)	4.2	89.8	0.7	5.3	Zipf	10.13	4.14

WL: Workload, OPS: Over-provisioning space, UR: Uniform Random, IR: Irregular Random, SQ: Sequential, ZF: Zipf, Synth.: Synthetic, OBS: Observed, UQ: Unique

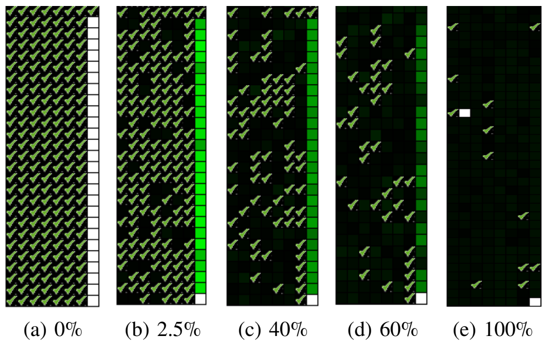

(a) 0% (b) 2.5% (c) 40% (d) 60% (e) 100%

FIGURE 10: Time-series visualization of SSD block states during MSR prxy workload execution with 5% OPS, showing progression from initial state (0%) to completion (100%). Demonstrates how invalid pages gradually spread from OPS-reserved region to entire SSD, causing WAF increase. (The darkest green block: the most valid pages, the brightest green: the most invalid pages, cross-signed blocks: fully invalid, and tick-signed: fully valid)

balance between these two extremes, showing predictable GC behaviour and consistent endurance performance. The visualization also helps identify hot and cold regions, enabling comparison of real workloads with their closest synthetic counterparts described in Section V-E.

In Table 8, No.2 - Fourier analysis of erase-count amplitudes uncovers hidden periodic wear-leveling patterns, offering a new diagnostic perspective for evaluating allocation policies. Certain strategies (e.g., S1/S4) perform best under sequential workloads, while others (e.g., S3/S5) demonstrate advantages under random or Zipf-like patterns, emphasising the workload dependence of optimal allocation design.

In Table 8, No.3 - Analysis of valid and invalid page distributions in real workloads such as TPC-C and prxy reveals a trade-off between GC efficiency and wear-leveling uniformity. High clustering of invalid pages accelerates GC but may compromise wear-leveling balance, whereas more uniform invalid-page distributions improve endurance but increase WAF. In the case of Zipf-like behaviour, the access pattern is highly asymmetrical, with a small subset of active LBAs receiving the majority of writes. Consequently, increasing OPS from 20% to 25% yields limited improvement in WAF given in Fig. 9 because writes remain concentrated in those compact hot regions; beyond a modest level, additional OPS provides diminishing returns. In our experiments, this localization suggests that modest OPS values can be near-optimal for space efficiency while still containing WAF growth.

In Table 8, No.4 - Visualisation of valid and invalid page layouts for real workloads helps identify hot and cold [28] regions and match them with their closest synthetic workloads. By mapping real traces to these representative patterns, `EyanaSSDSim` enables the application of well-established firmware algorithms tailored to each workload type, facilitating more accurate and workload-aware firmware tuning.

Overall, `EyanaSSDSim` transforms SSD research into an interactive, interpretable, and data-driven process, empowering researchers and practitioners to visualise internal operations, detect inefficiencies, and optimise firmware behaviour in real time.

VI. PERFORMANCE VALIDATION

The validation of `EyanaSSDSim` ~~, our simulator,~~ is demonstrated through a comparative analysis of the WAF against `FTLSim` [15] and `FEMU` [14], and read latency against `FEMU` ~~across different workloads~~.

The simulation configuration ~~for validation~~ is the same as ~~the settings in~~ Table 2, with 10% OPS for `FTLSim` comparison and 25% OPS for `FEMU` ~~comparison~~. ~~Overall, the trend of the comparison results given in Fig.~~ 12 ~~matches each other. However, in some cases, EyanaSSDSim does not achieve close WAF values compared to FTLSim, indicating some discrepancies in performance. Despite this, the simulator demonstrates effectiveness and reliability in simulating SSD performance, but the WAF values do not match closely due to certain differences in the underlying simulation approaches.~~

TABLE 8: Summary of Figures: Observations, Implications, Insights, and Benefits

No.	Fig.	What It Shows	Implications	Key Insights	Benefit
1	Fig. 3a-3f	Data placement at GC threshold and after workload	Page validity during/after GC	Zipf: low WAF; Seq: balanced; Uniform: better wear-leveling and high WAF	Visualize workload impact on GC, WAF and wear-leveling
2	Fig. 8	Fourier transform of erase counts	Hidden periodic wear-leveling patterns	S1/S4 best for sequential; S3 best for Zipf; S5 best for uniform random	Compare allocation policy efficiency
3	Fig. 9a	Invalid page distribution (TPC-C and prxy with OPS 10% and 25%)	Effect on WAF and GC	High DoIPD: better GC; low DoIPD: better wear-leveling	Trade-off: performance vs. endurance
4	Fig. 9b-9e	Layouts of valid/invalid pages	Compare data clustering across workloads	Clustering: helps GC but hurts wear-leveling; Uniformity: helps wear-leveling but raises WAF	Demonstrates OPS impact in real workloads

The differences in WAF values between `EyanaSSDSim` and `FTLSim` can be attributed to the ~~additional~~ LSB (Least Significant Bit) backup pages used in `FTLSim`. ~~Specifically, when the OPS is set to 10%, FTLSim utilises 87.5K backup pages for sequential workloads, 51.2K pages for uniform random, 49.4K pages for Zipf, 1.5M pages for TPC-C, and 2.8M pages for the prxy workloads.~~ This ~~additional~~ backup mechanism in `FTLSim` influences the GC behaviour, leading to ~~the observed~~ higher WAF ~~compared to EyanaSSDSim. The presence of these extra backup pages in FTLSim results in additional GC overhead, which increases the WAF~~.

~~Additionally, we~~ We tested our workloads using `FEMU` with 25% OPS~~, as recommended~~ [31]. The results for `EyanaSSDSim` and `FEMU` show some differences in WAF values ~~as well~~. ~~These differences arise because~~ `FEMU` employs a queueing-based FIFO strategy for GC, whereas `EyanaSSDSim` follows a greedy policy ~~for GC~~. ~~The FIFO strategy in FEMU affects how blocks are selected for GC, leading to slight variations in the WAF values across different workloads. This difference in GC strategies contributes to the observed discrepancies between EyanaSSDSim and FEMU.~~

~~Despite these differences in WAF values, the~~ The overall similarity in trends ~~across different configurations and tools~~ suggests that `EyanaSSDSim` performs similarly to `FTLSim` and `FEMU` ~~in terms of overall SSD behaviour~~, thereby validating its effectiveness ~~and reliability in simulating SSD performance under various configurations~~. To evaluate the read performance of `EyanaSSDSim`, we compared its latency

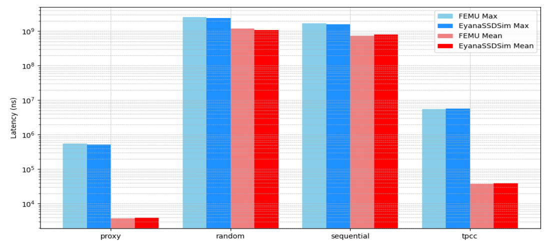


FIGURE 11: Read latency distribution comparing EyanaSSDSim and FEMU , showing maximum latency (510ms vs 560ms) and average latency (3.921ms vs 3.758ms). The similar latency patterns validate EyanaSSDSim 's timing accuracy

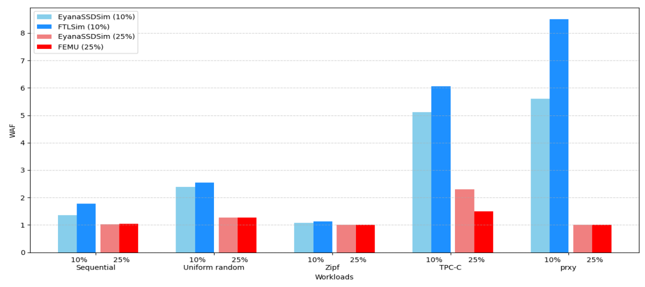


FIGURE 12: WAF comparison across sequential, uniform random, Zipf, TPC-C, and prxy workloads for EyanaSSDSim , FTLSim (10% OPS), and FEMU (25% OPS). Similar trends across simulators validate EyanaSSDSim 's WAF calculation accuracy

results with FEMU ~~using similar workload settings~~. As shown in Fig. 11, EyanaSSDSim shows a maximum read latency of 510 ms, ~~which is slightly~~ lower than FEMU's 560 ms. The average latency of EyanaSSDSim is 3.921 ms, which closely matches FEMU's 3.758 ms. ~~These results indicate that EyanaSSDSim achieves comparable latency performance to FEMU, with minor differences resulting from changes in GC strategy and I/O scheduling processes. Overall, the close alignment of latency values reinforces the accuracy and reliability of EyanaSSDSim in simulating SSD read performance.~~

VII. DISCUSSION AND IMPLICATION

EyanaSSDSim is not limited to traditional SSD analysis-it also provides a foundation for exploring emerging storage technologies. Its visualisation capabilities can be extended to support Zoned Storage (ZNS) by tracking per-zone operations~~, including sequential writes, zone open/close events, and write pointer progression~~. For Flexible Data Placement (FDP), EyanaSSDSim enables visualisation of how host placement hints affect physical layout~~, supporting optimisation of FDP-aware systems. Additionally, as SSD architectures evolve, the simulator can be extended to offers a flexible platform for observing and prototyping advanced features like custom FTLs and computational storage. Furthermore,~~ EyanaSSDSim can function as a digital twin [32] for SSD subsystems by ~~virtually~~ replicating their structure, behavior, and workload interactions. ~~By integrating real-time performance metrics and system states, it enables predictive analysis, anomaly detection, and optimization of storage performance under diverse workloads and configurations similar to how digital twins~~ [32] ~~are employed in data center environments to improve operational insight and efficiency.~~ These strengths position EyanaSSDSim as a versatile tool for ~~both~~ research and educational purposes ~~in next-generation storage system design~~.

VIII. SURVEY RESULT

We conducted a survey targeting students and professionals in computer engineering ~~and semiconductor technology~~ to gather feedback on our ~~SSD and floating gate transistor~~ simulators. A total of 1,011 participants participated ~~in the survey~~, providing insight through ~~a combination of~~ multiple-choice ~~questions~~ and open-ended responses. ~~Impressively, approximately~~ 95% of ~~the~~ users expressed satisfaction with the simulators. ~~Furthermore,~~ 97% ~~of the participants~~ reported that the simulators ~~significantly~~ reduced their learning time~~, allowing them to allocate more time to research~~.

IX. CONCLUSION

We proposed a ~~high-fidelity~~ web-based SSD simulator, EyanaSSDSim, that constructs a ~~complete,~~ visualizable storage stack ~~from scratch~~ and models ~~all detailed~~ characteristics of SSD internal hardware and software. This simulator serves educational, analytical and research purposes.

REFERENCES

[1] J.-U. Kang, J. Hyun, H. Maeng, and S. Cho, "The multi-streamed Solid-State drive," in 6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 14), 2014. [Online]. Available: https://www.usenix.org/conference/hotstorage14/workshop-program/presentation/kang
[2] Q. Wang, J. Li, P. P. C. Lee, T. Ouyang, C. Shi, and L. Huang, "Separating data via block invalidation time inference for write amplification reduction in log-structured storage," in Proceedings of the 20th USENIX Conference on File and Storage Technologies (FAST 22), 2022, pp. 429–444.
[3] S. Oh, J. Kim, S. Han, J. Kim, S. Lee, and S. H. Noh, "Midas: Minimizing write amplification in log-structured systems through adaptive group number and size configuration," in Proceedings of the 22nd USENIX Conference on File and Storage Technologies (FAST 24), 2024, pp. 259–275.
[4] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, "Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity," in Proceedings of the International Conference on Supercomputing, ser. ICS '11. Association for Computing Machinery, 2011, pp. 96–107. [Online]. Available: https://doi.org/10.1145/1995896.1995912
[5] W. Bux and I. Iliadis, "Performance of greedy garbage collection in flash-based solid-state drives," Perform. Eval., vol. 67, no. 11, pp. 1172–1186, Nov. 2010. [Online]. Available: https://doi.org/10.1016/j.peva.2010.07.003
[6] R. Liu, Z. Tan, L. Long, Y. Wu, Y. Tan, and D. Liu, "Improving fairness for ssd devices through dram over-provisioning cache management," IEEE Transactions on Parallel and Distributed Systems, vol. 33, no. 10, pp. 2444–2454, October 2022.
[7] 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 16th USENIX Conference on File and Storage Technologies (FAST 18), 2018, pp. 49–66. [Online]. Available: https://www.usenix.org/conference/fast18/presentation/tavakkol

[8] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for ssd performance," in USENIX 2008 Annual Technical Conference, ser. ATC'08. USENIX Association, 2008, pp. 57–70.

[9] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, "Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity," in Proceedings of the International Conference on Supercomputing, ser. ICS '11, 2011, pp. 96–107. [Online]. Available: https://doi.org/10.1145/1995896.1995912

[10] M. Jung, W. Choi, S. Gao, E. H. Wilson III, D. Donofrio, J. Shalf, and M. T. Kandemir, "Nandflashsim: High-fidelity, microarchitecture-aware nand flash memory simulation," ACM Trans. Storage, vol. 12, no. 2, Jan. 2016. [Online]. Available: https://doi.org/10.1145/2700310

[11] J. Yoo, Y. Won, J. Hwang, S. Kang, J. Choi, S. Yoon, and J. Cha, "Vssim: Virtual machine based ssd simulator," in 2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST), 2013, pp. 1–14.

[12] J. He, S. Kannan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "The unwritten contract of solid state drives," in Proceedings of the Twelfth European Conference on Computer Systems, ser. EuroSys '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 127–144. [Online]. Available: https://doi.org/10.1145/3064176.3064187

[13] M. Jung, J. Zhang, A. Abulila, M. Kwon, N. Shahidi, J. Shalf, N. S. Kim, and M. Kandemir, "Simplessd: Modeling solid state drives for holistic system simulation," IEEE Computer Architecture Letters, vol. 17, no. 1, pp. 37–41, 2018.

[14] H. Li, M. Hao, M. H. Tong, S. Sundararaman, M. Bjørling, and H. S. Gunawi, "The CASE of FEMU: Cheap, accurate, scalable and extensible flash emulator," in 16th USENIX Conference on File and Storage Technologies (FAST 18), Oakland, CA, 2018, pp. 83–90. [Online]. Available: https://www.usenix.org/conference/fast18/presentation/li

[15] S.-H. Kim, J. Lee, and J.-S. Kim, "Gcmix: An efficient data protection scheme against the paired page interference," ACM Trans. Storage, vol. 13, no. 4, Nov. 2017. [Online]. Available: https://doi.org/10.1145/3149373

[16] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, "Flashsim: A simulator for nand flash-based solid-state drives," in Proceedings of the 2009 First International Conference on Advances in System Simulation, ser. SIMUL '09. IEEE Computer Society, 2009, p. 125–131. [Online]. Available: https://doi.org/10.1109/SIMUL.2009.17

[17] G. Yadgar and R. Shor, "Experience from two years of visualizing flash with ssdplayer," ACM Trans. Storage, vol. 13, no. 4, Nov. 2017.

[18] N. Ahmed, D. De, and I. Hussain, "Technology-assisted decision support system for efficient water utilization: A real-time testbed for irrigation using wireless sensor networks," IEEE Access, vol. 6, pp. 1322–1337, 2018.

[19] G. Yadgar, R. Shor, E. Yaakobi, and A. Schuster, "It's not where your data is, It's how it got there," in 7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 15), Santa Clara, CA, 2015. [Online]. Available: https://www.usenix.org/conference/hotstorage15/workshop-program/presentation/yadgar

[20] M. Jung and M. Kandemir, "An evaluation of different page allocation strategies on High-Speed SSDs," in 4th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 12), 2012. [Online]. Available: https://www.usenix.org/conference/hotstorage12/workshop-program/presentation/Jung

[21] Ş. Öztürk, U. Özkaya, B. Akdemir, and L. Seyfi, "Convolution kernel size effect on convolutional neural network in histopathological image processing applications," in 2018 International Symposium on Fundamentals of Electrical Engineering (ISFEE), 2018, pp. 1–5.

[22] B. McFee, Digital Signals Theory. W3K Publishing, 2007, ch. Similarity.

[23] A. K. Verma, "Fourier transform visualization," 2025, accessed: 2025/03/10. [Online]. Available: https://github.com/thatSaneKid/fourier

[24] S. T. Leutenegger and D. Dias, "A modeling study of the tpc-c benchmark," in Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '93, 1993, pp. 22–31.

[25] W. W. Hsu, A. J. Smith, and H. C. Young, "Characteristics of production database workloads and the tpc benchmarks," IBM Systems Journal, vol. 40, no. 3, pp. 781–802, 2001.

[26] D. Narayanan, A. Donnelly, and A. Rowstron, "Write Off-Loading: Practical power management for enterprise storage," in 6th USENIX Conference on File and Storage Technologies (FAST 08), 2008.

[27] Y. Oh, E. Lee, C. Hyun, J. Choi, D. Lee, and S. H. Noh, "Enabling cost-effective flash based caching with an array of commodity ssds," in Proceedings of the 16th Annual Middleware Conference, ser. Middleware '15. Association for Computing Machinery, 2015, pp. 63–74.

[28] J. Lee and J.-S. Kim, "An empirical study of hot/cold data separation policies in solid state drives (ssds)," in Proceedings of the 6th International Systems and Storage Conference, ser. SYSTOR '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2485732.2485745

[29] K. Han, H. Gwak, D. Shin, and J. Hwang, "Zns+: Advanced zoned namespace interface for supporting in-storage zone compaction," in 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21), 2021, pp. 147–162. [Online]. Available: https://www.usenix.org/conference/osdi21/presentation/han

[30] M. Bjørling, A. Aghayev, H. Holmberg, A. Ramesh, D. L. Moal, G. R. Ganger, and G. Amvrosiadis, "ZNS: Avoiding the block interface tax for flash-based SSDs," in 2021 USENIX Annual Technical Conference (USENIX ATC 21), 2021, pp. 689–703. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/bjorling

[31] MoatLab, "FEMU Blackbox Script, v9.0.1," June 2024, latest release: v9.0.1, June 21, 2024. Accessed: 2025/03/05.

[32] J. Athavale, C. Bash, W. Brewer, M. Maiterth, D. Milojicic, H. Petty, and S. Sarkar, "Digital twins for data centers," Computer, vol. 57, no. 10, pp. 151–158, 2024.

HABIBUR RAHMAN received his Bachelor's degree in Computer Science and Engineering from Daffodil International University, Bangladesh, in 2019, where he was among the top three students. He then pursued a Master's degree in AI Convergence Engineering at Gyeongsang National University, South Korea. His major fields of study include artificial intelligence, solid-state drives, and system engineering.

He worked as a Software Engineer at ResPay, Dallas, Texas, USA, from 2018 to 2019, leading the development of the ResPay application. From 2019 to 2023, he served as a Software Engineer at Daffodil International University.

Habibur is an expert in C, Python, JavaScript, cloud technologies and remains an active contributor to open-source projects.

OMAR FAROQUE is a Software Engineer at Meta, specialising in high-performance computing, distributed systems, and cloud infrastructure. With over 15 years of experience, he has worked at Amazon Web Services (AWS) and Apple, contributing to Big Data engineering, AI infrastructure, and IoT solutions.

He holds an M.S. in Electrical and Computer Engineering from Southern Illinois University, Carbondale, and has received notable accolades, including 2nd place in the MOBI Grand Challenge (2019) and the Greater Chicago Area Android Wear Hackathon Champion (2015).

Omar is an expert in Java, Go, Python, and cloud technologies and remains an active contributor to open-source projects.

CHOI MOON SEOK received his Bachelor's degree in Aerospace and Software Engineering from Gyeongsang National University, Korea. Since March 2024, He has been a Master's student at Department of AI Convergence Engineering from Gyeongsang National University, Korea. He is interested in storage, embedded software and operating systems.

JAEHO KIM is an Associate Professor in the Department of Software Engineering at Gyeongsang National University (GNU). Before joining GNU, I was a senior research engineer at Huawei Technologies in Germany from November 2019 to August 2020. I was a postdoctoral researcher at the Department of Electrical and Computer Engineering of Virginia Tech from October 2017 to October 2019 and Ulsan National Institute of Science and Technology (UNIST) from October 2015 to September 2017. I received the PhD degree from the School of Computer Science, University of Seoul, Korea in 2015.