Word-Line-Aware Garbage Collector for QLC-based NAND Flash Memories

Cassiano Silva de Campes, Rodrigo Marques de Figueiredo, and Sandro José Rigo

***Abstract*—The Garbage Collector (GC) on NAND Flash memories is one of the most expensive operations in modern Solid-State Drives (SSD). However, it is essential to claim more free pages on SSDs. Various researches attempt to reduce the penalty of GC operations at different levels of the Input/Output (IO) path. Nevertheless, within the reviewed related works in Host-side, Open-Channel SSDs, and Device-side areas, we did not observe overlap with our research scope in how the logical pages are scattered in the NAND Word-Lines (WL) at the physical level, and how they impact performance degradation during the Garbage Collection operation. To reduce this bridge gap, we propose a new GC policy: Word-Line Aware Garbage Collector (WLA-GC). The WLA-GC focuses on reducing the Garbage Collector overhead by observing the physical distribution of valid logical pages within the WLs of a NAND Flash block to best select the victim blocks to be erased. Based on these criteria, the GC decisions are made by considering the latency the victim block will take to be fully erased. Analytical modeling shows that the proposed method outperforms the Vanilla GC in all cases, with an average performance of 25.4%. In comparison, the experimental results present a performance improvement of 55% in best-case scenarios.**

Link to graphical and video abstracts, and to code:
https://latamt.ieeer9.org/index.php/transactions/article/view/10214

***Index Terms*—NAND-Flash Memory, Garbage Collector, QLC Cells, Word-Line Awareness**

I. Introduction

NAND flash memories are available at different cell bit densities to increase the capacity at the cost of latency: Single-, Multi-, Triple-, or even Quad-Level Cells (for short: SLC, MLC, TLC, QLC, respectively). As the number of bits-per-cell gets denser, the width of the word-lines that compose a physical NAND flash page also increases. In a QLC memory, a single cell can be in one of the 16 different voltage levels (L0-L15), as shown in Fig. 1, which can logically represent 4 bits of persisted data.

To read from a QLC cell, several comparison steps must be taken to determine the logical value stored [1]. Sense amplifiers are used to compare the voltage levels of the cells, identifying in which bin the threshold is and what is its equivalent logical binary value to which logical page. More comparisons are required to determine the logical value stored.

The associate editor coordinating the review of this manuscript and approving it for publication was Giner Alor-Hernández (*Corresponding author: Cassiano Campes*).

This work has been supported by Fundo Loyola.

Cassiano Campes, R. M. D. Figueiredo, and S. J. Rigo are, with with Universidade do Vale do Rio dos Sinos, São Leopoldo, Rio Grande do Sul, Brazil (e-mails: scampes@edu.unisinos.br, marquesf@unisinos.br, and rigo@unisinos.br).

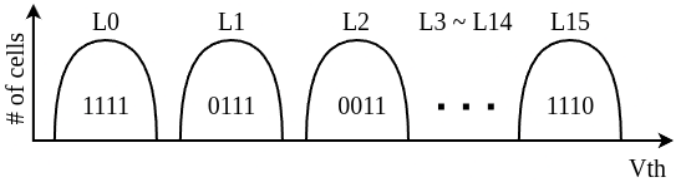

Fig. 1. Read method for QLC (4 bits/cell) voltage levels (16 voltage levels).

Also, since a QLC cell can store 4 bits of data, each one of those bits map a different logical page. The write (also known as program) operation is done by injecting electrons into the floating gate of the QLC cell. Similarly, when erasing the QLC cell, the stored voltage levels are drained out from the floating gate, and different latencies are experienced according to the current voltage level on the cell. This latency difference is known in the literature as fast- and slow-pages [1]–[4]. Fig. 2 shows the relationship between the physical voltage level stored in the QLC cell (L0-L15) and the logical value that the voltage level represents in which logical page. Thus, lower pages take fewer steps than higher pages to determine their logical value. To read the *bit-0* of *Page 0*, for example, a single comparison determines whether the bit is set or unset, while for *the bit-0* of *Page 3*, up to 15 comparisons must be taken to determine the logical value of the cell.

These cells are organized into strings of 32k to 128k cells, which represent the physical page width. Depending on the cell type, the usable capacity can vary significantly. For example, a 32k-cell string in SLC mode corresponds to 4kB of logical data storage. Using the same 32k-cell string in QLC mode increases the logical capacity fourfold to 16kB. Likewise, in a 128k-cell string organization in QLC mode, the logical data capacity increases to 64kB of logical data. One or multiple strings compose a word-line (WL), which is the physical unit for read and write operations. Several word-lines constitute a block, which is the physical erase unit of a NAND flash. Multiple blocks are organized into NAND Flash planes, which can be single-, or multi-planes, to provide parallel access to the WLs [5].

The logical and physical organization of the NAND flash devices is managed by the Flash-Translation Layer (FTL). The FTL is responsible for managing and translating the logical requests (therefore, logical pages) from the host to the physical locations within the device. This translation is done transparently to the host, meaning that the host is not aware of the idiosyncrasies of the NAND Flash physical characteristics, in this case, the WL width, nor the cell type. Nonetheless, the

Voltage Level	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	L15
bit[0].page0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
bit[0].page1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1
bit[0].page2	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1
bit[0].page3	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1

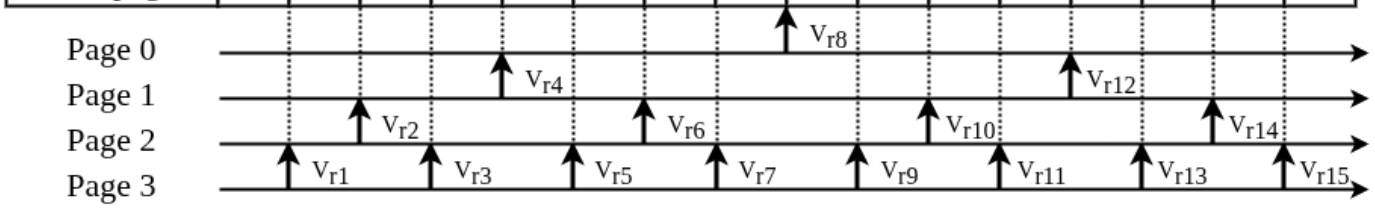

Fig. 2. Read method for QLC voltage levels: Physical voltage level stored in the QLC cell (L0-L15); logical bit value (0-1); comparison to determine the logical bit value on that page (Page 0 - Page 3).

FTL is also in charge of hiding some mismatches between the host and the device, such as the smallest addressable logical unit from the host (4kB page size in a logical block device in the Linux kernel [6]) into the smallest addressable logical unit of the NAND flash, a WL. Furthermore, the WL can be composed of up to 128k cells. In an SLC NAND cell composition, the cells can represent 16kB of logical data - or 4 logical pages from the host; or in a QLC cell organization, representing 64kB of logical data - or 16 logical pages seen from the host [7], [8]). Because of this mismatch between the host and device, we can observe that in a NAND flash device, a single physical WL can represent a range of 1 logical page (32k SLC cells) up to 16 logical pages (128k QLC cells) - 4kB up to 64kB of data, respectively - as seen by the host, depending on its NAND flash type.

As the host reads data from the NAND Flash device, it sends logical page addresses, and the FTL is in charge of translating them into physical page addresses to read from the respective physical page. Furthermore, when the host performs a write operation, the FTL finds a suitable empty physical page to be used to store the data, and then translates the logical page address into the physical page address. Overwrites carried by the host are not done in place at the physical level of the NAND flash. Rather, the FTL invalidates the old physical page and assigns a new one to write the updated data. Deletions performed by the host are translated into physical page invalidation by the FTL. Consequently, as the number of invalid pages grows, the NAND Flash device needs to reclaim these invalidated physical pages to be erased. This operation is called the Garbage Collector (GC). The erase unit of NAND Flash memories is performed at the block level, and thus, the FTL is in charge of selecting the blocks that will be marked as victim blocks, and later will be erased.

The GC selects the blocks that have the largest number of invalid pages to reduce the impact of this operation. Any remaining valid pages must be written somewhere else first, and once it is done, the GC can finally erase the victim block. The GC operation uses a basic heuristic that selects the victim blocks just based on the number of invalid pages. It does not consider how the physical invalid pages are distributed, nor how the physical characteristics would impact the speed of the erase process, or the cell lifespan.

This paper proposes the WLA-GC, a Word-Line Aware Garbage Collector policy that selects the victim blocks not just by inspecting the number of invalid pages but also by considering the physical distribution of the pages within the NAND Flash Word-Lines. The policy allows for better decisions about picking the block as the victim.

II. Background

A. NAND Flash Memory Architecture

The NAND Flash memories are organized into numerous cells that store data. These cells are in fact physical transistors that store voltage levels which logically represent the binary data. Depending on the type of the cell, different voltage levels can be used to represent a wider range of logical data. NAND Flash memories have architectural variations such as the Single-Level Cell (SLC) that can store one voltage level, where it can store a single bit of data (1-bit cell). Multi-Level Cell (MLC) can store two different voltage levels, representing 2-bits per cell. Triple- and Quad-Level Cells can store 3-bits and 4-bits, respectively. As more voltage levels are supported by the cell, more data bits can be stored, and denser the NAND Flash memories become.

However, cramming more bits into a NAND Flash cell comes at the cost of performance and lifespan. The denser the cells are, the lower the read and write performance are, and the lifespan is reduced. These drawbacks occur because the cell has more voltage levels, so more comparisons are needed to determine which voltage level is stored. Also, higher voltage levels wear out the cells faster than lower voltage levels during read, write, and erase operations.

Multiple cells are organized into strings to form a Word-Line (WL). A WL can also be seen as the physical page of the NAND Flash memory, which is the minimum addressable read or write unit in the NAND Flash memory. To read or write from the WL, Sense Amplifiers (SA) are organized in parallel to the Bit Lines of the cells. The SAs determine the logical value of the bits by comparing the voltage levels depending on the cell type. For a QLC type, for example, up to 15 different comparisons can take place to determine the logical

value of the data. Once the logical data is decoded, it is stored temporarily in the Buffer Block.

Several strings are grouped to form a NAND Flash Block, which is the minimum erasable unit of the NAND Flash memory. Because of their architecture, NAND Flash memories cannot erase a single WL; they must erase the entire block to claim more free pages. Consequently, any valid pages within a block that need to be erased must first be written in a different block. Once it is complete, the erase procedure can be executed on that block.

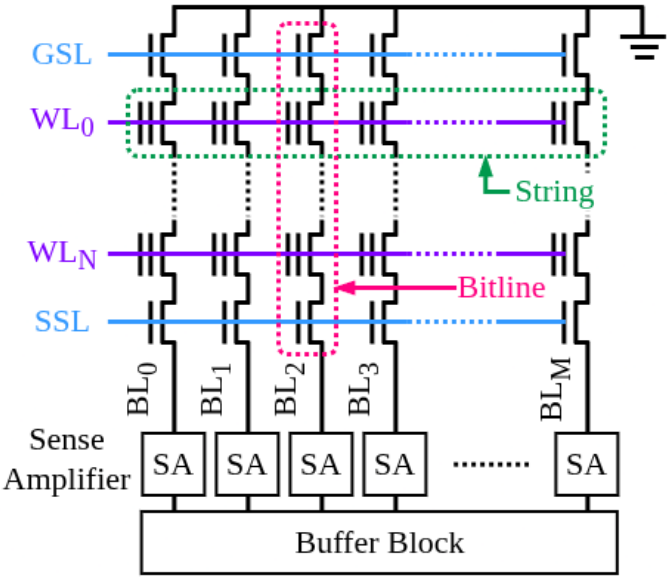

Fig. 3. Architectural organization of NAND Flash cells

Fig. 3 shows the architectural organization of the NAND Flash cells that compose the WLs. The WL, composed of thousands of cells, can be as wide as 128k cells to represent 16kB (in 1-bit per cell SLC mode), or 64kB (in 4-bits per cell QLC mode) of logical data addressable by the host. All these idiosyncrasies related to the physical aspects of the NAND Flash cells are transparent to the host due to the Flash Translation Layer (FTL).

B. Flash Translation Layer in NAND Flash Memories

The FTL is in charge of translating logical pages from the host into physical pages within the NAND Flash memory. These translations are required because the host should not need to understand how the NAND Flash memory is managed, nor how the physical aspects must be handled. FTL resolves all these aspects to the host by managing the logical to physical addresses, controlling the empty and invalid pages, as well as claiming more free space whenever needed by the host.

As the NAND Flash memory is written by the host, valid and invalid logical pages are produced within the physical blocks, which are managed by the FTL. From time to time, or when the system is under pressure for free pages, invalid logical pages must be claimed to make physical free space for new incoming write operations. The GC operation is responsible for selecting victim blocks to be erased based on the number of invalid logical pages a block has. If the number of invalid logical pages of a block is greater than a predetermined threshold, the block is marked as the victim. Thus, the GC picks the victim block and moves (physically writes) the valid logical pages into a different block. Once the valid pages are moved, the victim block can be erased to generate free physical space [9].

The general equation to describe the latency of a victim block ($latency_{vb}$) will be:

$$latency_{vb} = (t_{read} + t_{program}) * N_{valid} + t_{erase} \quad (1)$$

Where it is the sum of the time to read (t_{read}) and program ($t_{program}$) each valid page within the block, times the number of valid pages (N_{valid}), plus the time to erase the entire block (t_{erase}). However, the GC operates at the logic level, and it does not consider the distribution of the fast- and slow-pages within a block during its decision for a victim block. In fact, the GC only considers the number of valid pages (given a threshold) that must be moved somewhere else to make the decision of picking the block as a victim block.

Although two or more blocks may have the same number of invalid pages to be selected as the victim block, the general Equation (1) does not consider how the logical pages are physically distributed within the block. Consequently, the selection of a victim block based on this single criterion of valid/invalid page thresholds may result in an unexpected increase in latency. For that reason, a new GC policy that is aware of the physical distribution of written pages into the WL of a NAND Flash block is necessary to reduce the latency impact during erase operations.

III. Our Approach: Word-Line-Aware Garbage Collector

The GC algorithms do not consider the physical aspects of the NAND Flash memories, although the GC algorithm resides within the FTL. Because of that, GC lacks improvements to consider the physical distribution of the pages within the NAND Flash cells to make a better decision for selecting the victim block. We call these algorithms as *Vanilla GC* throughout the remain of this paper. Based on the problem description, this research proposes a new policy to help the GC to select the best victim blocks based on its physically valid page distribution over the WLs within a block. The proposed method is called Word-Line-Aware Garbage Collector, or WLA-GC for short.

Before selecting the victim block to be erased, the WLA-GC considers the valid page distribution over the WLs, and thus, determines in advance the approximate latency the block will take to have its pages migrated, not just the number of valid or invalid pages, as performed by the Vanilla GCs. Considering this decision criterion, we derived from Equation 1 a more detailed analytical model to describe the latency that a victim block will suffer based on the physical page distribution over the WLs within that block, as follows:

$$latency_{vb_x} = \sum_{j=1}^{n} latency_{WL_{x,j}} \quad (2)$$

where n = WLs per block.

Equation (2) shows that the estimated latency experienced by the victim block will be the sum of all WLs latencies belonging to that victim block x ($latency_{vb_x}$) - where x is

the candidate block to be selected as victim block - and n is the number of WLs in a block according to the NAND flash architecture. To calculate the latency per WL ($latency_{WL_{x,j}}$), we present the Equation (3). The $latency_{WL_{x,j}}$ is equal to the latency to read the valid pages according to their voltage levels within the WL; plus the sum of the time to program all the valid pages to another block; plus the sum of the time it takes to erase the victim block.

Expanding Equation (2) in more detailed terms to describe the validity of pages within a block, and the latency those pages will take to be moved, we have the following:

$$latency_{WL_{x,y}} = \begin{pmatrix} lvl_{slc} & lvl_{mlc} & lvl_{tlc} & lvl_{qlc} \end{pmatrix} * \begin{pmatrix} b_{11} & \cdots & b_{1j} \\ b_{21} & \cdots & b_{2j} \\ b_{31} & \cdots & b_{3j} \\ b_{41} & \cdots & b_{4j} \end{pmatrix} + t_{program} * \sum_{i=1,j=1}^{n} \begin{pmatrix} b_{11} & \cdots & b_{1j} \\ b_{21} & \cdots & b_{2j} \\ b_{31} & \cdots & b_{3j} \\ b_{41} & \cdots & b_{4j} \end{pmatrix} + t_{erase} \quad (3)$$

The valid pages within the WL are represented by the $b_{i,j}$ binary matrix. The index i and j in the binary matrix are the n^{th}-depth bit of the QLC cell ($n = 1, n = 2, n = 3, n = 4$ for SLC, MLC, TLC, and QLC, respectively), and the logical page shift seen by the host (4kB width) on that WL, respectively. Furthermore, the read latency according to the cell level is described by the vector $level_{slc}$, ..., to $level_{qlc}$, where is the read latency for the n^{th}-depth bit of the QLC cell; The valid pages within the WL is again represented by the $b_{i,j}$ binary matrix which is multiplied by the $t_{program}$, which is the time each valid page will take to be written somewhere else; and finally, the t_{erase} that is the time the victim block takes to be erased.

For a simple comparison, consider the Vanilla GC policy described by Equation (1) and the WLA-GC policy described by Equation (2) and (3). Considering two victim blocks with the same number of valid pages, the Vanilla GC, which is not aware of the WL page distribution, can be up to 37,5% slower if compared to WLA-GC. This difference is the result of the heuristic adopted by selecting the victim blocks. This performance difference can be achieved by the WLA-GC due to its awareness of how the pages are physically distributed within the NAND Flash cells, and not simply by the number of invalid pages. Consequently, it can estimate the read, write, and erase latencies that the victim block will take if selected. On the other hand, the Vanilla GC policy simply selects the victim block by considering solely the number of invalid pages, which is proven to not be the optimal selection method.

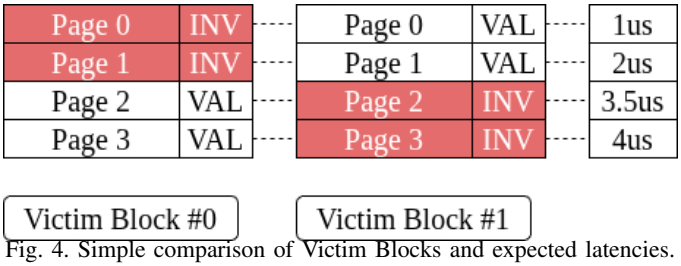

Fig. 4. Simple comparison of Victim Blocks and expected latencies.

Fig. 4 shows two victim blocks with the same number of invalid pages to describe this simple comparison between Vanilla GC and WLA-GC. Although the number of invalid pages are the same (two, in this case), different latencies are expected from each one as we run the different GC operations on them. For the sake of simplicity, let's assume that the $t_{program}$ is the same as the time to read the cell as shown on the right side of the figure ($1\mu s$, $2\mu s$, $3.5\mu s$, and $4\mu s$, respectively), and a t_{erase} of $20\mu s$. When the WLA-GC is going to select one of the two blocks, it will pick the *Victim Block #1* because it has a total latency of $1\mu s + 2\mu s + 1\mu s + 2\mu s + 20\mu s = 26\mu s$. On the other hand, the Vanilla GC could select the *Victim Block #0* because of the incremental order of the blocks, and also because *Victim Block #0* and *#1* have the same number of valid pages. Thus, it would result in a latency of $3.5\mu s + 4\mu s + 4.5\mu s + 4\mu s + 20\mu s = 35\mu s$, which is about 34% slower than WLA-GC.

Considering the background concepts and the proposed approach, in the next section, we present our analytical and experimental evaluation methods to validate the performance enhancements resulting from the WLA-GC when compared to the Vanilla GC.

IV. Evaluation

The baseline selected for comparison in this work is the Vanilla GC for our experiments - summarized by Equation (1) and Equation (2). To the best of our knowledge, and based on the related work review depicted in Section V, we did not identify specific work that focuses on the same assumptions of our research: *how the physical page distribution with the Word-Line affects the overall performance of the Garbage Collector based on the NAND flash cell types*. Related work is complementary to our policy and can be combined with our method at FTL level. To better understand the effects of the proposed WLA-GC policy compared to the Vanilla GC, we divided the Evaluation into two parts: (1) Analytical Modeling; and (2) Experimental Modeling. Subsection IV-A presents the analytical modeling we developed, detailing the implementation of the equations, while Subsection IV-B shows the experimental results conducted and the model implementation in C language. For both Analytical and Experimental modeling, we use the parameters as shown in Table I to simulate the NAND flash device.

A. Analytical Modeling

For the Analytical Modeling, we implemented the Equation (1) and Equation (2) in Python language and used the parameters depicted in Table I to figure out the latencies as we increased the number of valid pages within the victim block for both Vanilla GC and the WLA-GC. In other words, finding the minimization and maximization of the equations for each algorithm used.

For simplicity, we assume that the t_{erase} is constant for both cases, so we can omit it from the equations, and the $t_{program}$ (time to program each valid page within the block) is equal to the write latency to write the page at the same voltage level on a different block, as shown in Table I. Unless

TABLE I
PARAMETERS FOR THE ANALYTICAL AND EXPERIMENTAL MODELING

Total blocks	1,000,000	
Logical page size	4kB	
WL width (# of cells)	128k*	
Cell type	QLC	
WL per Block	1024	
Pages per Block	4096	
Read Delay	SLC	1us
	MLC	2us
	TLC	2.5us
	QLC	3us
Write Delay	SLC	2us
	MLC	3us
	TLC	3.5us
	QLC	4us
$t_{program}$	SLC	2us
	MLC	3us
	TLC	3.5us
	QLC	4us
t_{erase}	60us	

* The WL in QLC mode represents 16 logical pages seen by the host.

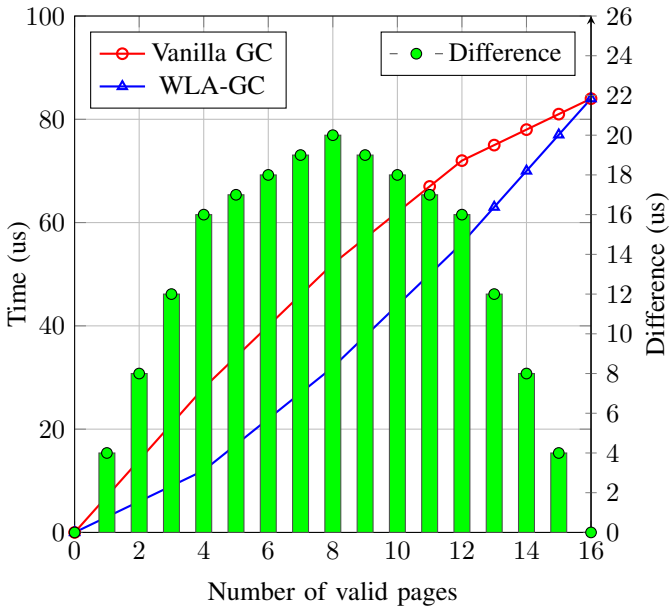

Fig. 5. Comparison of Vanilla GC, WLA-GC, and their difference.

stated, any other delays (transfer time from the bus, ECC, NAND Flash plane is busy, etc) are disregarded from our analysis. Fig. 5 shows the results, where *x-axis* represents the number of valid pages within a QLC WL with 128k cells width (128 k cells $\times$ 4 bits per cell / 8 bits / 4 kB = 16 logical pages). The *y-axis* on the left side is the approximate latency time taken to erase the victim block; while the *y-axis* on the right side is the latency difference between the Vanilla GC and the WLA-GC policies. It is visible that WLA-GC outperforms the Vanilla GC performance in most of the cases, while in the worst-case scenario - where the distribution of the pages is the same for both policies - they show no performance difference. The worst-case scenario represents only 12.5% of the total possible page distribution, where WLA-GC shows the same performance, while 87,5% of the cases WLA-GC outperforms Vanilla GC. On average, WLA-GC shows a latency of 38.1 μs while Vanilla GC shows a latency of 51.1 μs. On average, the WLA-GC outperforms Vanilla GC by 25.4%.

B. Experimental Modeling

For the Experimental Modeling, we also used the same parameters presented in Table I to run a simulation of victim block selection for both Vanilla GC and WLA-GC algorithms. To emulate the NAND Flash, we developed a C program that mimics the base architecture of a NAND Flash block with its WL organization as well as the different cell levels. The emulator allows us to instantiate as many blocks as requested, initialize them, and execute the Vanilla GC and the WLA-GC. Table II shows the specification of the computer used to run the experiments.

TABLE II
TECHNICAL CONFIGURATIONS OF THE HOST MACHINE USED TO RUN THE PROPOSED EXPERIMENTS

Component	Specification
Processor	AMD Ryzen 9 5950X
Number of Cores	16
Number of Threads	32
Memory	64 GB
Disk	1 TB SSD NVMe

The execution of the experiments was conducted at 5% steps, meaning the total number of valid pages within the emulated blocks to run the algorithms. Each step ran a batch of 10 executions with 1,000,000 NAND Flash blocks, and calculated the average latencies of the execution for each algorithm. To execute the experiments, we divided the simulation into three steps: (1) Initialization of the blocks; (2) Victim block selection; (3) Execution of the algorithm.

The *(1) Initialization of the blocks* consists of initializing the 1,000,000 blocks and the internal structures (WL and cell type) that will be used to run the experiments. The blocks will be initialized with the valid pages according to the step it is currently at. On step *(2)Victim block selection*, we ran each GC policy to select the victim blocks from the 1,000,000 blocks pool. Each algorithm is set to select a quarter of the available blocks, so, in total 250,000 blocks are selected as victim blocks per algorithm. Finally, on step *(3) Execution of the algorithm* is when the write and erase operations take place. In total, 200,000,000 blocks were emulated to complete these experiments.

Fig. 6 shows the results of Vanilla GC versus the WLA-GC. Our experiments running on the emulated NAND Flash scenario shows similar results as those presented in the analytical modeling, showing that the WLA-GC performs better than Vanilla GC in most cases, and it has the same performance as Vanilla GC in the worst-case scenarios. In our experimental results, the WLA-GC presents a performance gain of up to

55% compared to the Vanilla GC, and an average performance gain of 17.7%. As the number of valid pages increase, WLA-GC will have fewer options to select the best victim blocks, converging to the same performance as the Vanilla GC when victim blocks have higher valid pages. Regardless of the number of valid pages, WLA-GC outperforms Vanilla GC from 5% up to 95% of valid pages within the blocks in our experiments.

C. FTL Requirements to Support WLA-GC

The FTL resides inside the NAND flash device, and it is composed of runtime data structures that are built from the persistent metadata stored in the NAND flash memory. The runtime data structures for a simple page-level mapping FTL implementation are generally composed of the Logical to Physical (L2P) mappings, Validity Page Bitmap, and Block Information Table [3]

WLA-GC was developed on top of a page-level mapping FTL. We added an extra metadata in the Block Information Table (BIT) - a *block weight* - to accumulate the weight of the valid pages based on their physical distribution over the WL (following the same logic as depicted in Figure 4). When the GC operation needs to execute, the FTL iterates over the Block Information Table to scan for the victim blocks based on the *weight* heuristic. In this work, we did not focus on the FTL penalties that this heuristic would imply. To have a fair baseline, both *Vanilla GC* and *WLA-GC* policies do a full scan on the BIT to look for victim blocks. Improvements on the data structures used by the FTL are left for future work to be implemented on a real Open-Channel SSD. Detailed implementation of this heuristic can be found on our GitHub page, publicly available here: https://github.com/campescassiano/ieee-latin-america-wla-gc-paper.

V. Related Work

The Garbage Collector overhead is widely studied and researched in the literature to find new methods to reduce its performance penalty. To better understand the solutions proposed in the literature, we organized the related works into three major categories: (1) Host-side; (2) Open-Channel SSDs; and (3) Device-side. Furthermore, these approaches can be divided into the subcategories: (1.1) Data temperature; (1.2) File type; (2.1) Open-channel; (3.1) FTL level; and (3.2) Architecture level. Table III summarizes these categories and the related works in the respective category.

TABLE III
Reviewed Related Work and the Scope Categorization

Scope	Approach level	Related works
Host-side	Data temperature	[10]–[14]
	File type	[14]–[17]
OCSSD	Open-Channel	[2], [18], [19]
Device-side	FTL level	[20], [21]
	Architecture level	[22]–[24]

The GC overhead is well known in the literature, and several works have been proposed at different layers of the NAND flash memory ecosystem, ranging form host-side FTLs using Open Channel SSDs, down to architectural NAND flash memory operation parallelism [2], [10]–[24].

In [10] the authors presented the ShadowGC, a method that can take advantage of the write buffers located in the host-side (page cache in Linux systems) to reduce the GC overhead caused by the movement of valid pages into new blocks. The separation of the pages relies on the data temperature to better predict when it will be invalidated [13]. This combination of host-side and device-side write buffers combines an interesting method of bridging the gap between the host and device.

The work presented by *Choi and Ahn* [11] exploits the new Zoned SSDs (ZSSDs) that divide the NAND flash memory space into different zones which allows the host to directly assign zones to certain write requests. However, they have identified that journaling file systems do not take full advantage of ZSSDs and can cause a considerable GC overhead due to the nature of writes caused by the journaling mechanism. Considering this, *Choi and Ahn* [11] proposed a new file system zone placement to separate the journal data from the file data, thus reducing the GC overhead by up to 26.8%. Other related works also consider the journaling file systems to improve the GC operations [12], [22].

Furthermore, several works converge on the understanding that the host-side is Open Channel SSDs (OCSSDs) are widely used by host-based FTLs to let the host directly control the NAND Flash memories. A common benefit proposed by OCSSDs work are due to the possibility of the host to separate hot and cold data based on its own implementation decisions [15], [22].

In [16], the authors propose a multi-level parallel GC operation, taking advantage of the new modern NAND Flash memories. The work uses the parallelism of the modern NAND Flash architectures to execute the steps of the GC operation in parallel, such as: migrating valid pages in parallel; and erasing victim blocks of multiple planes are also reclaimed in parallel. *Garret et al.* [2] also focuses on increasing the parallelism of the erase operations within the NAND Flash memories. Their work proposes the Intra-Plane Parallel Block Erase (IPPBE) that reduces the impacts of the GC operations.

In [18] a novel method of erasing the blocks of a NAND Flash memory is presented. Although the work is more focused on the memory wearout caused by the erase stress, their work presents a very detailed understanding of the WL organization and the impacts of the erase operations. Hot and cold data classification at FTL is another common technique to reduce the performance impacts caused by the GC operation [20], [21].

In this section, we summarize the related works that are close to our research study. Although they share similar motivations in improving GC overhead, we could not find one that makes the same assumptions and proposes the same mechanism as our work. Our work focuses on the physical page distribution within the WL of the NAND flash memory, and how to use this information to better predict the GC execution time. WLA-GC resides on the Device-Side, which

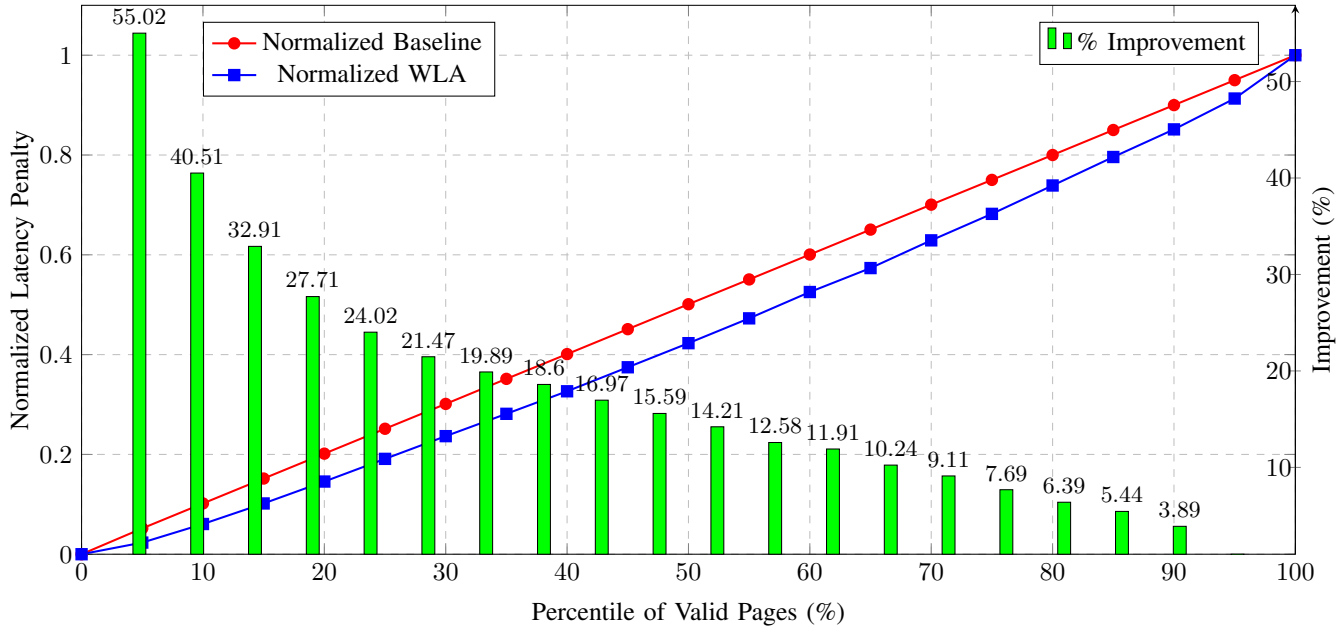

Fig. 6. Normalized Latency Penalty vs. Percentile of Valid Pages with % Improvement.

means its implementation is done at the FTL level within the storage device. Considering this, where applicable, these techniques found in the related works are complementary and could be combined with our approach at the FTL level.

VI. Conclusion

As the width of physical word-lines of a NAND flash increases and the bit-density scales, the role of the FTL becomes more crucial in how to better manage the NAND flash memories to take the best performance. Furthermore, the FTL ends up hiding the physical characteristics of the NAND flash memories, not leaving room for improvement at the software layer.

The FTL is the layer that bridges the logical and physical aspects of the NAND flash memories, but it still lacks enhancements to take the best advantage of it during the GC operation. Based on these discoveries, WLA-GC is proposed to reduce the latency gap between the logical and physical operations caused by the GC operations.

WLA-GC showed that providing awareness of the physical distribution of the logical pages within a WL can considerably reduce the erase latency of NAND flash blocks. Analytical modeling has shown that on best-case victim block selection, latency can be reduced up-to 37,5%, while on Experimental Modeling the improvement can be up to 55%, and on the worst-case scenario, it is compared to a Vanilla GC policy.

As future work, we expect to implement our WLA-GC policy on a real OCSSD to open-source a real FTL implementation. Thus, it is possible to precisely evaluate the benefits as well as the impacts of the applied heuristic in the FTL. Nonetheless, another relevant topic is the usage of multi-streamed SSDs [25], which can further increase the probabilities of maximizing the number of valid pages in the fast page region of the blocks.

Acknowledgments

This work has been supported by Universidade do Vale do Rio dos Sinos, São Leopoldo, Rio Grande do Sul, Brazil via Fundo Loyola. All the source-code artefacts related to this paper can be found on our GitHub page, publicly available here: https://github.com/campescassiano/ieee-latin-america-wla-gc-paper.

References

[1] L. Shijun, Z. Xuecheng, and W. Baocun, "Program and read methods with offset in quad-level-cell nand design," in *2017 International Conference on Electron Devices and Solid-State Circuits (EDSSC)*. IEEE, 2017, pp. 1–2. [Online]. Available: https://doi.org/10.1109/EDSSC.2017.8126461

[2] T. Garrett, J. Yang, and Y. Zhang, "Enabling intra-plane parallel block erase in nand flash to alleviate the impact of garbage collection," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2018, pp. 1–6. [Online]. Available: https://doi.org/10.1145/3218603.3218627

[3] N. Li, M. Hao, H. Li, X. Lin, T. Emami, and H. S. Gunawi, "Fantastic ssd internals and how to learn and use them," in *Proceedings of the 15th ACM International Conference on Systems and Storage*, ser. SYSTOR '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 72–84. [Online]. Available: https://doi.org/10.1145/3534056.3534940

[4] W. Wang, W. Pan, T. Xie, and D. Zhou, "How many mlcs should impersonate slcs to optimize ssd performance?" in *Proceedings of the Second International Symposium on Memory Systems*, ser. MEMSYS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 238–247. [Online]. Available: https://doi.org/10.1145/2989081.2989095

[5] C.-W. Yoon, "The fundamentals of nand flash memory: Technology for tomorrow's fourth industrial revolution," *IEEE Solid-State Circuits Magazine*, vol. 14, no. 2, pp. 56–65, 2022. [Online]. Available: https://doi.org/10.1109/MSSC.2022.3166466

[6] W. Sul, K. Kim, M. Ryu, H. Jung, and H. Han, "Fast journaling made simple with nvm," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ser. SAC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1214–1221. [Online]. Available: https://doi.org/10.1145/3341105.3373865

[7] K. Koo, J. Oh, K. Min, Y. Kwon, and Y. Won, "C2j: compulsory compound transaction for journaling file system," in *Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems*, ser. APSys '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 17–24. [Online]. Available: https://doi.org/10.1145/3476886.3477514

[8] H. Park, E. Lee, J. Kim, and S. H. Noh, "Lightweight data lifetime classification using migration counts to improve performance and lifetime of flash-based ssds," in *Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems*, 2021, pp. 25–33. [Online]. Available: https://doi.org/10.1145/3476886.3477520

[9] S. Li, W. Tong, J. Liu, B. Wu, and Y. Feng, "Accelerating garbage collection for 3d mlc flash memory with slc blocks," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8. [Online]. Available: https://doi.org/10.1109/ICCAD45719.2019.8942097

[10] J. Cui, Y. Zhang, J. Huang, W. Wu, and J. Yang, "Shadowgc: Cooperative garbage collection with multi-level buffer for performance improvement in nand flash-based ssds," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1247–1252. [Online]. Available: https://doi.org/10.23919/DATE.2018.8342206

[11] Y.-i. Choi, S. Ahn *et al.*, "Separating the file system journal to reduce write amplification of garbage collection on zns ssds," *Journal of Multimedia Information System*, vol. 9, no. 4, pp. 261–268, 2022. [Online]. Available: https://doi.org/10.33851/JMIS.2022.9.4.261

[12] J. Kim, M. Kim, M. D. Tehseen, J. Oh, and Y. Won, "{IPLFS}:{Log-Structured} file system without garbage collection," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 739–754. [Online]. Available: https://www.usenix.org/conference/atc22/presentation/kim-juwon

[13] 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

[14] S. Gong, H. Sun, Z. Fang, L. Liu, L. Chen, and Y. Gao, "Vstream: A distributed streaming vector search system," *Proc. VLDB Endow.*, vol. 18, no. 6, p. 1593–1606, Aug. 2025. [Online]. Available: https://doi.org/10.14778/3725688.3725692

[15] H. Lee, W. Choi, and Y. Hong, "On-demand garbage collection algorithm with prioritized victim blocks for ssds," *Electronics*, vol. 12, no. 9, p. 2142, 2023. [Online]. Available: https://doi.org/10.3390/electronics12092142

[16] T. Wang, P. Li, P. Xie, and X. Wang, "Ssd multi-level parallel garbage collection," in *2023 4th International Conference on Information Science, Parallel and Distributed Systems (ISPDS)*. IEEE, 2023, pp. 301–306. [Online]. Available: https://doi.org/10.1109/ISPDS58840.2023.10235643

[17] Y.-Y. Lu, C.-H. Wu, and Y.-S. Chen, "A machine-learning-based data classifier to reduce the write amplification in ssds," in *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, ser. RACS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 213–218. [Online]. Available: https://doi.org/10.1145/3400286.3418239

[18] D. Hong, M. Kim, G. Cho, D. Lee, and J. Kim, "{GuardedErase}: Extending {SSD} lifetimes by protecting weak wordlines," in *20th USENIX Conference on File and Storage Technologies (FAST 22)*, 2022, pp. 133–146. [Online]. Available: https://www.usenix.org/conference/fast22/presentation/hong

[19] R. Zhang, D. Liu, C. Yang, X. Chen, L. Qiao, and Y. Tan, "Optimizing cow-based file systems on open-channel ssds with persistent memory," in *Proceedings of the 2022 Conference & Exhibition on Design, Automation & Test in Europe*, ser. DATE '22. Leuven, BEL: European Design and Automation Association, 2022, p. 496–501. [Online]. Available: https://doi.org/10.23919/DATE54114.2022.9774695

[20] P. Singh *et al.*, "Mfgc: Minimal first garbage collection for monitoring of context-aware victim selection," *Journal of Electrical Systems*, vol. 20, no. 2s, pp. 92–101, 2024. [Online]. Available: https://doi.org/10.52783/jes.1092

[21] Y.-S. Tsai, S.-H. Chen, Y.-C. Liaw, and C.-Y. Wu, "Exploring hot/cold data separation for garbage collection efficiency enhancement on ocssds," in *2023 20th International SoC Design Conference (ISOCC)*. IEEE, 2023, pp. 241–242. [Online]. Available: https://doi.org/10.1109/ISOCC59558.2023.10396140

[22] S. Son and S. Ahn, "Optimizing garbage collection overhead of host-level flash translation layer for journaling filesystems," *International Journal of Internet, Broadcasting and Communication*, vol. 13, no. 2, pp. 27–35, 2021. [Online]. Available: https://doi.org/10.7236/IJIBC.2021.13.2.27

[23] X. Zhang, J. Bhimani, S. Pei, E. Lee, S. Lee, Y. J. Seong, E. J. Kim, C. Choi, E. H. Nam, J. Choi, and B. S. Kim, "Storage abstractions for ssds: The past, present, and future," *ACM Trans. Storage*, vol. 21, no. 1, Jan. 2025. [Online]. Available: https://doi.org/10.1145/3708992

[24] Z. Tan, L. Long, J. Shen, R. Liu, C. Gao, K. Zhong, and Y. Jiang, "Optimizing garbage collection for zns ssds via in-storage data migration and address remapping," *ACM Trans. Archit. Code Optim.*, vol. 21, no. 4, Nov. 2024. [Online]. Available: https://doi.org/10.1145/3689336

[25] 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)*. Philadelphia, PA: USENIX Association, Jun. 2014. [Online]. Available: https://dl.acm.org/doi/10.5555/2696578.2696591

Cassiano Silva de Campes received his BSc. in Computer Engineering at Universidade do Vale do Rio dos Sinos - UNISINOS, Brazil (2016); Sandwich PhD student in Electrical and Computer Engineering at Sungkyunkwan University - SKKU, South Korea (2016-2019); PhD Student in Applied Computing at Universidade do Vale do Rio dos Sinos - UNISINOS, Brazil; Senior Embedded Software Engineer at Venko Networks, Porto Alegre, Brazil.

Rodrigo Marques de Figueiredo received his BSc. in Electrical Engineering at Universidade do Vale do Rio dos Sinos - UNISINOS (2005); MSc. in Applied Computing at Universidade do Vale do Rio dos Sinos - UNISINOS (2009); PhD in Geology at Universidade do Vale do Rio dos Sinos - UNISINOS (2016); Professor at UNISINOS University in Electrical Engineering undergraduate and graduate programs.

Sandro José Rigo received his BSc. in Computer Science at Pontífica Universidade Católica do Rio grande do Sul - PUCRS (1990); MSc. in Computer Science at Universidade Federal do Rio Grande do Sul - UFRGS (1993); PhD in Computer Science at Universidade Federal do Rio Grande do Sul - UFRGS (2008); Post-doctorate at Friedrich-Alexander Universität Erlangen-Nürnberg/Germany (2018); Professor at UNISINOS University and Researcher in Applied Computing Graduate Program (UNISINOS/PPGCA); Dean of the UNISINOS Polytechnic School; Consultant on innovation and information technology; Research project coordinator at UNISINOS.