Automating Endurance Test for Flash-based Storage Devices in Samsung Electronics

Jinkook Kim
Samsung Electronics Co.
Hwasung, Republic of Korea
jqook.kim@samsung.com

Minseok Jeon
Korea University
Seoul, Republic of Korea
minseok_jeon@korea.ac.kr

Sejeong Jang
Samsung Electronics Co.
Hwasung, Republic of Korea
sejeong.jang@samsung.com

Hakjoo Oh
Korea University
Seoul, Republic of Korea
hakjoo_oh@korea.ac.kr

***Abstract*—We present ARES, an automated framework for writing endurance tests on flash-based storage devices. Since flash-based storages such as solid-state drives and SD cards have a limited capacity for processing data write requests, it is important for manufacturers to accurately test and specify the maximum amount of data writes that their products are guaranteed to withstand. Unfortunately, however, writing such an endurance test is mostly conducted manually in practice, which is difficult, laborious, and sometimes inaccurate. To address this issue, we present ARES, a learning-based automated approach for generating endurance tests on flash-based storage devices. ARES is built on two ideas. First, we observe that the search space of endurance tests can be effectively reduced by devising abstract relative write patterns. Second, we use a learning algorithm based on genetic programming in order to find worse-case write patterns efficiently. The experimental results demonstrate that ARES is capable of successfully learning high-quality write patterns. The performance of the learned write patterns is superior to that of the manual tests designed by human engineers in Samsung Electronics. Especially for 32GB USB, ARES identified a write pattern that is 26% more effective than the manually crafted write pattern that has been used until recently.**

***Index Terms*—Flash based Storage, Non-functional property testing, Test input generation, Genetic algorithm**

I. Introduction

Flash-based storages are crucial devices widely used for reading and storing data. Thanks to the appearance of the MLC (Multi Level Cell) techniques [1], which enabled a single cell to store data larger than 1 bit, flash-based storages can efficiently store data and have become the most popular devices used in practice. Compared to hard disk drives, flash-based storages have better random I/O performance, efficient power consumption, and smaller sizes, etc. Currently, flash-based storage devices are widely used in PC, mobiles, and automotives, among others.

Flash-based storages, however, have a limited capacity for writing data. Though MLC is efficient, it exacerbates this write endurance problem in flash-based storage devices. If flash-based storage reaches a certain level of write activity (also known as write endurance), the stored data is no longer trusted. This issue is caused because the data storage medium of flash-based storage has a limited lifespan (program-erase count [2]). Figure 1 shows the impact of the MLC technology on the lifespan of flash memory. Figure 1 shows that as more data is stored in a given cell, the flash memory's wirte endurance declines. Currently, write endurance is regarded as a major criterion for evaluating the quality of flash-based storage. Different flash-based storages have varying degrees of endurance; it is therefore important to accurately specify their (guaranteed) endurance for users.

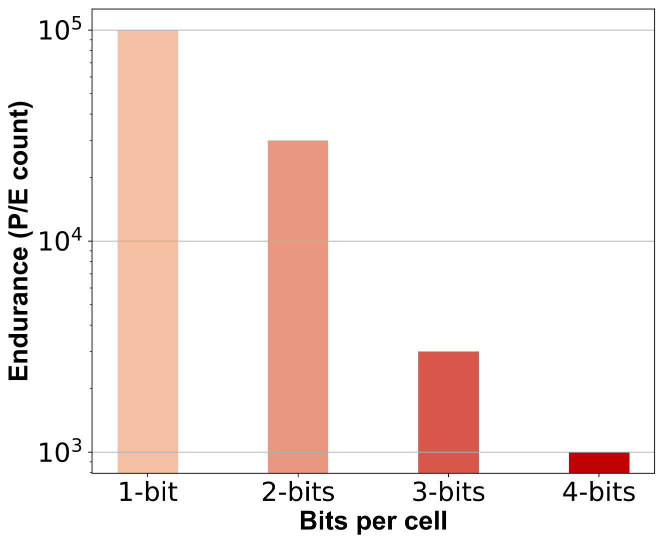


Fig. 1: How the endurance of flash memory has changed with respect to the advance of MLC technology.

The objective of a write endurance test is to quantify the level of endurance, so that we can check that the devices satisfy the requirement of market or customers before they are shipped. A write endurance test takes a flash-based storage and write pattern (e.g., test case). Then, it measures the amount of data write that the storage can endure. We note that the primary goal of the endurance test is to evaluate the firmware's quality, rather than the hardware; one of the main jobs of the firmware is to manage data writes properly.

The success of a write endurance test depends heavily on the quality of the write patterns it uses. Worst-case write patterns, which mirror the worst-case user scenarios, effectively consume the devices' lifespan throughout the test; the write endurance test aims to find a worst-case write pattern for a given storage device. Unfortunately, however, discovering such a write pattern has been manually conducted by field

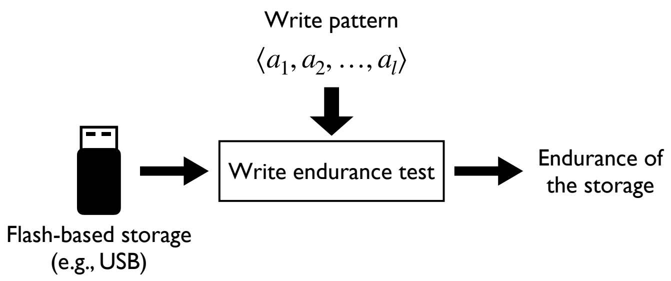

Write pattern
$\langle a_1, a_2, \ldots, a_l \rangle$
Write endurance test
Endurance of the storage
Flash-based storage (e.g., USB)

Fig. 2: Conventional write endurance test process.

engineers.

Manually designing worse-case write patterns is difficult and time-consuming. To develop worst-case write patterns, test engineers need to examine various user scenarios (e.g., workload [3]), developer interviews, S/W development documentation review, and industry standards review. Our experience is that these are difficult and laborious tasks even for experienced test engineers, and may lead to human errors.

To address this problem, we present ARES, an automated technique for generating endurance tests. ARES is developed with two key ideas. First, we design abstract relative write patterns. Each abstract relative write pattern represents a set of concrete write patterns that are likely to show similar performance in the write endurance test. Using abstract relative write patterns instead of concrete write patterns prevents redundant learning and significantly reduces the search space. In our evaluation, the search space is reduced from about $3,000,000^{100}$ to 33^{100}. The second idea is a learning algorithm based on genetic programming. The algorithm exploits the structure of our abstract relative write patterns to effectively learn a good concrete write pattern (e.g., worst-case write pattern) that will be used in write endurance test.

Our evaluation results show that ARES is effective at learning high-quality write patterns that perform better than existing manually crafted write patterns used in Samsung Electronics. We use ARES to produce worst write patterns for 32GB USB flash drive and 256GB micro SD Card, two popular flash-based storage devices used in practice. For the two storages, ARES produced write patterns that perform better than those developed by test engineers in Samsung. Especially for 32GB USB, the learned write pattern was significantly (26%) more effective than the baseline, the manually crafted write pattern.

Contributions. Our contributions are summarized as follows:

- We present ARES, an automated technique for generating write endurance tests. The key ideas are our abstract relative write patterns to reduce search space and our learning algorithm that uses abstract relative write patterns to effectively search qualified write patterns.
- We experimentally demonstrate the effectiveness of ARES in comparison with existing manually designed write patterns that have been used in industry.

II. Preliminaries

In this section, we introduce the write endurance test in detail and related terms.

Flash-based storage. Flash-based storages are popular devices that store data in flash memories. Solid State Drive [4] and USB flash drive [5] are representative flash-based storages. Multi Media Card [6], Universal Flash Storage [7], embedded in mobile [8], automotive [9], and IoT [10], also belong to flash-based storages. A flash-based storage consists of flash memory [11], controller, and firmware where a controller processes instructions, and firmware operates the hardware.

Sector. A sector is the minimum unit space in flash-based storages that stores user data. A unit space is 512KBytes or 4096KBytes.

Address. Addresses are the locations of sectors where each sector has a unique address. The write endurance test uses addresses for writing data on the corresponding sectors.

Workload. Workloads are sequences of read and write, and describe user scenarios. In our evaluation, we consider workloads as described in JESD219 [12] including various user scenarios in practice. Note that different storage devices (e.g., USB flash drive vs SSD) have varying user scenarios. The write endurance test should consider suitable user scenarios.

Write pattern. A write pattern is a sequence of write requests that affects (e.g., declines) the endurance of flash-based storages.

A. Write Endurance

The endurance of a flash-based storage presents the amount of data writes that the storage can endure. A storage device has a finite endurance, and the write endurance of the storage is determined by write patterns [3], [13] to be processed. Endurance-critical write patterns make the storage have smaller write endurance.

B. Write Endurance Test

The objective of a write endurance test is to determine the endurance guarantee of flash-based storages. Figure 2 presents the write endurance test process used in Samsung Electronics. A write endurance test takes a flash-based storage and a write pattern. The storage device iteratively processes the write pattern until the device uses all of its lifespan. Then, the write endurance test returns the amount of written data that the given storage endured. Note that the success of the write endurance test depends heavily on the quality of the used write pattern as guaranteed write endurance can be found with well-designed write patterns reflecting worst-case user scenarios. Our goal is to automatically generate high-quality write patterns for the endurance test.

III. Method

In this section, we present ARES, a new approach for learning write patterns for write endurance test. We first model flash-based storages and write endurance test (Section III-A). Then, we illustrate how ARES automatically generates good write patterns (Section III-B and III-C).

A. Problem

We model a flash-based storage ($Storage$) as a triple:

$$Storage = Sector \times Age \times Lifespan.$$

$Sector \subseteq 2^{\mathbb{N}}$ denotes the set of sector addresses in the storage, $Age \subseteq \mathbb{N}$ the spent endurance (i.e., program-erase count) of the storage, and $Lifespan \subseteq \mathbb{N}$ the maximum number of program-erase cycle that the storage can endure. We define two projection functions `Age` and `Lifespan`: $\texttt{Age}(s)$ and $\texttt{Lifespan}(s)$ denote the age and lifespan of the storage s, respectively. A write pattern w is a sequence of sector addresses (e.g., $\langle a_1, a_2, \ldots, a_l \rangle \in \mathbb{N}^l$). We assume the length (e.g., l) of write pattern is given.

In write endurance testing, the given write pattern iteratively increases the age of the given storage. We model an iteration of the endurance test as follows:

$$Test : Storage \times WritePattern \to \mathbb{N}.$$

Given a storage s and a write pattern w, *Test* returns an integer presenting the age increased (e.g., $\Delta\texttt{Age}(s)$) during the test. The endurance test iteratively runs $Test(s, w)$ until the age of the stoarge reaches its lifespan (e.g., $\texttt{Age}(s) \geq \texttt{Lifespan}(s)$). Then, the endurance test returns the total amount of written data as the endurance of the storage s.

Goal. For a given length l and storage s, our goal is to find an effective write pattern $w = \langle a_1, a_2, \ldots, a_l \rangle$ that would maximize the increased age in the endurance test:

$$\underset{w}{\operatorname{argmax}}\ Test(s, w).$$

Note that good write patterns effectively increase the age in each iteration; they make the storage s retire with small numbers of iterations in the endurance test.

Challenge. Write patterns have a huge search space; a naive search algorithm is impractical to find good write patterns. If there are $|S|$ addresses in the given flash-based storage s and the length of the write pattern is l (e.g., $|w| = l$), the search space poses $|S|^l$. For example in a 32GB NAND flash-based storage ($|S| \approx 6,000,000$), the search space of write pattern poses about $6,000,000^l$. A naive search algorithm would fail to learn effective write patterns.

B. Our Approach

The first idea of ARES is our two-step abstractions of write patterns to reduce the search space. We first define (normal) concrete write patterns on which we define abstractions. We define concrete write patterns as follows:

$$WritePattern : Addr^l$$

where addresses are natural numbers ($Addr \subseteq \mathbb{N}$). A concrete write pattern $w \in WritePattern$ is a sequence of addresses in the given storage device s. For example in the endurance test (e.g., *Test*), a concrete write pattern $w = \langle i, i, i, i, i \rangle$ writes data five times at the i'th sector of the storage s.

Relative write pattern. The first abstraction is our relative write patterns $RelWritePattern$ defined as follows:

$$RelWritePattern : \Delta Addr^l$$

where $\Delta Addr \subseteq \mathbb{N}$ presents how each address in the concrete write pattern changes. A relative write pattern $w = \langle \Delta a_1, \Delta a_2, \ldots, \Delta a_l \rangle$ presents a set of concrete write patterns:

$$\begin{aligned} \llbracket \langle \Delta a_1, \Delta a_2, \ldots, \Delta a_l \rangle \rrbracket_{RelWritePattern} &= \\ \{ \langle a_1, a_2, \ldots, a_l \rangle \in WritePattern \mid & \\ \forall i \in [2, l].\, a_i - a_{i-1} = \Delta a_i \}. \end{aligned}$$

For example, concrete write patterns $\langle 1, 2, 3, 4, 5 \rangle \in WritePattern$ and $\langle 2, 3, 4, 5, 6 \rangle \in WritePattern$ belong to the same relative write pattern $\langle 0, 1, 1, 1, 1 \rangle \in RelWritePattern$ because all the addresses in the concrete write patterns are increased by 1.

Relative write patterns are designed to exploit the behavior of the firmware in flash-based storage. The firmware in Flash-based storages uses an address mapping policy based on address locality (e.g., Delta FTL [14] and LAST [15]). Address locality is a tendency of memory addresses that an address is likely to be accessed if an adjacent one is accessed; the firmware is designed to consider relative addresses (e.g., whether the addresses in the write are adjusted) rather than absolute addresses.

We also experimentally checked concrete write patterns have the following property:

$$\begin{aligned} &\forall s.(\forall i \in [2, l].a_i - a_{i-1} = a'_i - a'_{i-1}) \implies \\ &\quad Test(s, \langle a_1, a_2, \ldots, a_l \rangle) \approx Test(s, \langle a'_1, a'_2, \ldots, a'_l \rangle) \end{aligned}$$

where $\langle a_1, a_2, \ldots, a_l \rangle$ and $\langle a'_1, a'_2, \ldots, a'_l \rangle$ are two different concrete write patterns. That is, concrete write patterns belonging to the same relative write pattern are likely to show similar performance. Using relative write patterns instead of concrete ones would reduce the search space without losing a chance to find qualified write patterns. Note that though using relative write patterns reduces the search space (e.g., from $|S|^l$ into $|S|^{l-1}$), the search space is still impractical (e.g., $6,000,000^{l-1}$). We need the following additional abstraction to make the learning practical.

Abstract relative write pattern. As an additional (second-step) abstraction, we define abstract relative write patterns:

$$\widehat{RelWritePattern} : \widehat{\Delta Addr}^{\,l}.$$

$\widehat{\Delta Addr} \subseteq \mathbb{N}$ is an abstracted one from $\Delta Addr$ with Table I which maps a relative address $\Delta a \in \Delta Addr$ into the corresponding abstract relative address $\hat{\Delta a} \in \widehat{\Delta Addr}$. For example, an abstract relative address $4 \in \widehat{\Delta Addr}$ presents a set of

TABLE I: Table that maps $\Delta a \in \Delta Addr$ to $\hat{\Delta a} \in \widehat{\Delta Addr}$.

$\hat{\Delta a}$	**Category**	**Criterian**
0	Overwrite	$\Delta a = 0$
1	Seq. forward	$\Delta a = 1$
2	Seq. backward	$\Delta a = -1$
3		$1 < \Delta a \leq 2$
4		$2 < \Delta a \leq 8$
5		$8 < \Delta a \leq 16$
6		$16 < \Delta a \leq 32$
7		$32 < \Delta a \leq 64$
8		$64 < \Delta a \leq 128$
9		$128 < \Delta a \leq 256$
10	Jump forward	$256 < \Delta a \leq 512$
11		$512 < \Delta a \leq 1024$
12		$1024 < \Delta a \leq 16384$
13		$16384 < \Delta a \leq 32768$
14		$32768 < \Delta a \leq 65536$
15		$65536 < \Delta a \leq 131072$
16		$131072 < \Delta a \leq 262144$
17		$262144 < \Delta a$
18		$-1 > \Delta a \geq -2$
19		$-2 > \Delta a \geq -8$
20		$-8 > \Delta a \geq -16$
21		$-16 > \Delta a \geq -32$
22		$-32 > \Delta a \geq -64$
23		$-64 > \Delta a \geq -128$
24		$-128 > \Delta a \geq -256$
25	Jump backward	$-256 > \Delta a \geq -512$
26		$-512 > \Delta a \geq -1024$
27		$-1024 > \Delta a \geq -16384$
28		$-16384 > \Delta a \geq -32768$
29		$-32768 > \Delta a \geq -65536$
30		$-65536 > \Delta a \geq -131072$
31		$-131072 > \Delta a \geq -262144$
32		$-262144 > \Delta a$

relative addresses $\{3,4,5,6,7,8\} \subseteq \Delta Addr$ because the abstract relative address 4 corresponds to $2 < \Delta a \leq 8$ in Table I. Note that any relative address $\Delta a \in \Delta Addr$ can be mapped into an abstract relative address $i \in [0, 32]$ in Table I. An abstract relative write pattern $\langle \hat{\Delta a}_1, \hat{\Delta a}_2, \ldots, \hat{\Delta a}_l \rangle \in \widehat{RelWritePattern}$ presents a set of relative write patterns as follows:

$$\begin{aligned}
&[\![\langle \hat{\Delta a}_1, \hat{\Delta a}_2, \ldots, \hat{\Delta a}_l \rangle]\!]_{\widehat{RelWritePattern}} = \\
&\quad \{\langle \Delta a_1, \Delta a_2, \ldots, \Delta a_l \rangle \in RelWritePattern \mid \\
&\qquad\qquad \forall i \in [1, l].\, \Delta a_i \in Table\ I(\hat{\Delta a}_i)\}
\end{aligned}$$

where $Table\ I(\hat{\Delta a}_i)$ presents the set of Δa that corresponds to the abstract relative address $\hat{\Delta a}_i$. Note that an abstract relative write pattern $\hat{\hat{w}} \in \widehat{RelWritePattern}$ also can be concretized into a set of concrete write patterns as follows:

$$\begin{aligned}
\gamma(\hat{\hat{w}}) = \{w \mid w \in\ &[\![\hat{w}]\!]_{RelWritePattern}, \\
&\hat{w} \in\ [\![\hat{\hat{w}}]\!]_{\widehat{RelWritePattern}}\}.
\end{aligned}$$

We designed Table I based on the observation of [16] and our domain knowledge. Kim et al. [16] present that the size of the cluster block, cluster page, and read/write buffer affect the performance of flash-based storages. It shows that the values of relative addresses affects the performance of the storage. Based on the observation, we designed our abstract relative addresses, which are clusters of relative addresses where relative addresses in the same cluster would result in similar effect in the endurance test. Therefore, an abstract relative write pattern $\hat{\hat{w}}$ presents a set of concrete write patterns $\gamma(\hat{\hat{w}})$ that would show similar performance in the write endurance test.

In the learning procedure, the search space is drastically reduced by devising abstract relative write patterns. In a 32GB NAND flash-based storage used in our evaluation, for example, the search space is reduced from $6,000,000^l$ into $(33)^l$.

C. Learning Write Patterns

Now we present our learning procedure that learns a qualified concrete write pattern for write endurance test. Our algorithm is an instance of the conventional genetic algorithm [17]; our algorithm iteratively produces and evaluates offspring abstract relative write patterns. In this section, we call abstract relative write patterns as abstract write patterns for brevity.

Algorithm 1 Our Learning Algorithm

```
Require: A storage $s$, write pattern length $l$, population $n$
Ensure: A concrete write pattern $w$
 1: procedure LEARN($s$)
 2:     $w_{best} \leftarrow \langle 0, 0, ..., 0 \rangle$                    ▷ length = $l$
 3:     $\{\hat{\hat{w}}_1, \hat{\hat{w}}_2, \ldots, \hat{\hat{w}}_n\} \leftarrow GenerateInitial(\mathrm{n})$
 4:     $W \leftarrow \{\hat{\hat{w}}_1, \hat{\hat{w}}_2, \ldots, \hat{\hat{w}}_n\}$
 5:     $score \leftarrow \lambda \hat{\hat{w}}.0$
 6:     while $\texttt{Age}(s) \geq \texttt{Lifespan}(s)$ do
 7:         $(w'_{best}, score) \leftarrow$ EVALUATE$(W, s, score)$
 8:         if $Test(s, w_{best'}) > Test(s, w_{best})$ then
 9:             $w_{best} \leftarrow w'_{best}$
10:         $\{\hat{\hat{w}}'_1, \ldots, \hat{\hat{w}}'_n\} \leftarrow$GENERATEOFFSPRING$(score, n)$
11:         $W \leftarrow \{\hat{\hat{w}}'_1, \ldots, \hat{\hat{w}}'_n\}$
12:     return $w_{best}$
 1: procedure EVALUATE($\{\hat{\hat{w}}_1, \hat{\hat{w}}_2, \ldots, \hat{\hat{w}}_n\}, s, score$)
 2:     $max \leftarrow 0$
 3:     $w_{best} \leftarrow \langle 0, 0, ..., 0 \rangle$
 4:     for $i = 1$ to $n$ do
 5:         $w_i \leftarrow ChooseConcrete(\hat{\hat{w}}_i)$
 6:         $score \leftarrow \lambda \hat{\hat{w}}.$ if $\hat{\hat{w}} = \hat{\hat{w}}_i$ then $Fitness(s, w_i)$ else $score(\hat{\hat{w}})$
 7:         if $Test(s, w_i) > max$ then
 8:             $w_{best} \leftarrow w_i$
 9:             $max \leftarrow Test(s, w_i)$
10:     return $(w_{best}, score)$
 1: procedure GENERATEOFFSPRING($score, n$)
 2:     $\{(\hat{\hat{w}}_1, \hat{\hat{w}}'_1), \ldots, (\hat{\hat{w}}_{\frac{n}{2}}, \hat{\hat{w}}'_{\frac{n}{2}})\} \leftarrow ChooseOld(score, n)$
 3:     $O \leftarrow \emptyset$
 4:     for $i = 1$ to $\frac{n}{2}$ do
 5:         $(\hat{\hat{w}}_{new}, \hat{\hat{w}}'_{new}) \leftarrow CrossOver(\hat{\hat{w}}_i, \hat{\hat{w}}'_i)$
 6:         $O \leftarrow O \cup \{\hat{\hat{w}}_{new}, \hat{\hat{w}}'_{new}\}$
 7:     return $O$
```

The goal of our algorithm is to learn a qualified concrete write pattern w that maximizes the fitness function [17] defined as follows:

$$\mathit{Fitness}(s, w) = \frac{\mathit{Test}(s, w)}{\texttt{Lifespan}(s)}. \tag{1}$$

where s is the given training storage. $\mathit{Fitness}(s, w)$ presents how effectively the write pattern w makes the storage s retire.

LEARN in Algorithm 1 presents the overall learning process. The algorithm takes a training storage s, length of write pattern l, and population size n as inputs. The algorithm iteratively enumerates n different write patterns with length l and evaluates them on the storage s. The iteration ends when the storage retires ($\texttt{Age}(s) \geq \texttt{Lifespan}(s)$). Then, it returns the learned concrete write pattern w_{best} that performed the best throughout the learning. Note that the length of the write pattern l and the population size n are tunable hyperparameters; we chose l and n as 100 and 24, respectively, with our experience.

The algorithm first generates an initial concrete write pattern (e.g., $\langle 0, 0, \ldots, 0\rangle$ at line 2) and abstract write patterns (line 3). $\mathit{GenerateInitial}(n)$ randomly generates n different abstract write patterns $\{\hat{\hat{w}}_1, \hat{\hat{w}}_2, \ldots, \hat{\hat{w}}_n\}$, and they are used as the initial work set W. score maps each abstract write pattern to its score, which is initialized as $\lambda\hat{\hat{w}}.0$ (line 5).

At lines 6-11, the algorithm iteratively evaluates abstract write patterns and generates offspring abstract write patterns for the next iteration. At line 7, the algorithm evaluates the abstract write patterns in W where $\text{EVALUATE}(W, s, \mathit{score})$ returns a qualified concrete write pattern w'_{best} produced during the evaluation. If the newly produced concrete write pattern w'_{best} shows a better performance than the existing best-performed one w_{best} (line 8), w'_{best} becomes the best concrete write pattern w_{best} (line 9). After evaluating W, the algorithm generates new n different offspring abstract write patterns for the next iteration (lines 10-11). The loop ends if the age of the given training storage s spends all its lifespan ($\texttt{Age}(s) \geq \texttt{Lifespan}(s)$). Note that the loop is guaranteed to be finished as the age of the storage s (e.g., $\texttt{Age}(s)$) monotonically increases whenever $\text{EVALUATE}(W, S, \mathit{score})$ is invoked. The age of the storage s eventually reaches the lifespan ($\texttt{Lifespan}(s)$). Then, the algorithm returns w_{best} as a learned concrete write pattern.

EVALUATE in Algorithm 1 presents how abstract write patterns in $W = \{\hat{\hat{w}}_1, \hat{\hat{w}}_2, \ldots, \hat{\hat{w}}_n\}$ are evaluated. In lines 4-9, the algorithm iteratively produces a concrete write pattern from each abstract write pattern. *ChooseConcrete* chooses a concrete write pattern belongs to the given abstract write pattern $\hat{\hat{w}}$:

$$\mathit{ChooseConcrete}(\hat{\hat{w}}) \in \gamma(\hat{\hat{w}}).$$

At line 6, the score of each abstract write pattern $\hat{\hat{w}}_i$ is measured by the fitness of the chosen belonging concrete write pattern w_i (i.e., $\mathit{Fitness}(w_i, s)$). It updates w_{best} if the produced concrete write pattern shows better performance (lines 7-10) than the existing best-scored one. After the iterations, it returns the best concrete write pattern w_{best}, produced during the iterations, and the score of it score.

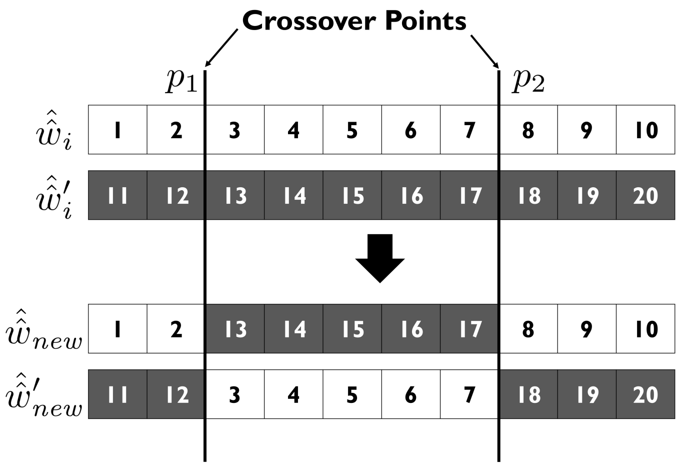


Fig. 3: How 2-point crossover [18] operation works.

GENERATEOFFSPRING presents how the learning procedure generates n different offspring abstract write patterns from the previously generated (parents) abstract write patterns. It first chooses $\frac{n}{2}$ pairs of parent abstract write patterns with *Choose-Old* (line 2) which chooses top-n qualified (e.g., elitism [19]) abstract write patterns as follows:

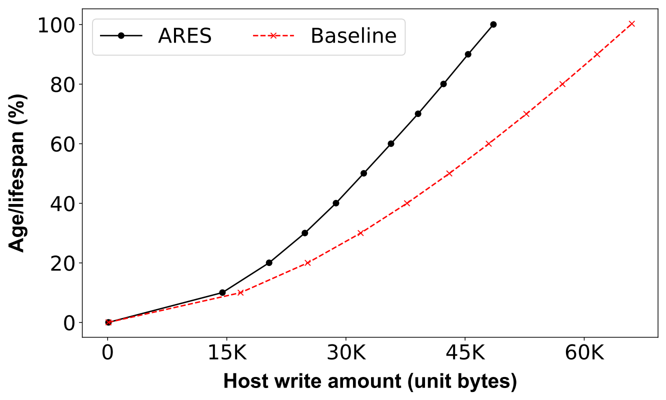
$$\{\hat{\hat{w}} \mid |\{\hat{\hat{w}}' \mid \mathit{score}(\hat{\hat{w}}') \geq \mathit{score}(\hat{\hat{w}})\}| \leq n\}.$$

Then, it randomly combines the chosen qualified abstract write patterns into $\frac{n}{2}$ pairs of parent abstract write patterns $\{(\hat{\hat{w}}_1, \hat{\hat{w}}'_1), (\hat{\hat{w}}_2, \hat{\hat{w}}'_2), \ldots, (\hat{\hat{w}}_{\frac{n}{2}}, \hat{\hat{w}}'_{\frac{n}{2}})\}$. In lines 4-6, the algorithm iteratively generates two offspring write patterns from a pair of parent write patterns with $\mathit{CrossOver}(\hat{\hat{w}}_i, \hat{\hat{w}}'_i)$. Figure 3 presents how $\mathit{CrossOver}(\hat{\hat{w}}_i, \hat{\hat{w}}'_i)$ generates offsprings from the two parents. It first randomly pick two addresses $p_1, p_2 \in [1, l]$ ($p_1 < p_2$) where l is the length of abstract write patterns. Then, it swaps the abstract relative addresses between p_1 and p_2 of the two parent abstract write patterns; the two changed abstract write patterns become the new offspring abstract write patterns. From the $\frac{n}{2}$ pairs of parent abstract write patterns, the algorithm generates n new offspring abstract write patterns.

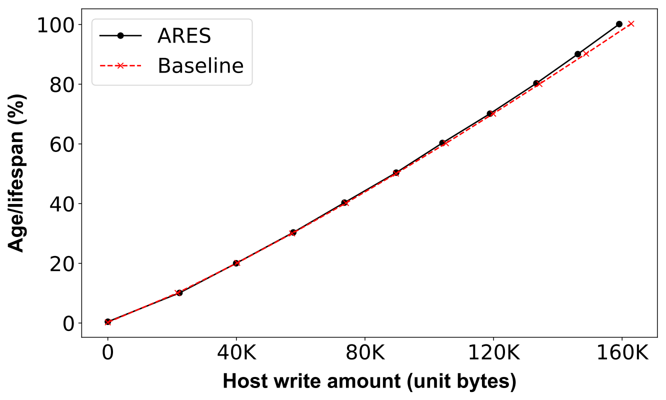
We use 2-point crossover to figure out and combine key subpatterns. Suppose a subpattern $\langle \hat{\Delta a}_i, \hat{\Delta a}_{i+1}, \ldots, \hat{\Delta a}_j\rangle$ of an abstract relative write pattern $\hat{\hat{w}} = \langle \hat{\Delta a}_1, \ldots, \hat{\Delta a}_l\rangle$ ($1 \leq i \leq j \leq l$) is a key subpattern that makes the belonging concrete write patterns (i.e., $\gamma(\hat{\hat{w}})$) effective. Then, two points p_1 and p_2 satisfying $p_1 \leq i \wedge j \leq p_2$ keep the key sub pattern and produce a new abstract relative write pattern. The produced write pattern, including the key subpattern, will have a high score, and the algorithm will preserve such key subpatterns. Eventually, the algorithm can combine such suitable subpatterns into an ideal one.

IV. EVALUATION

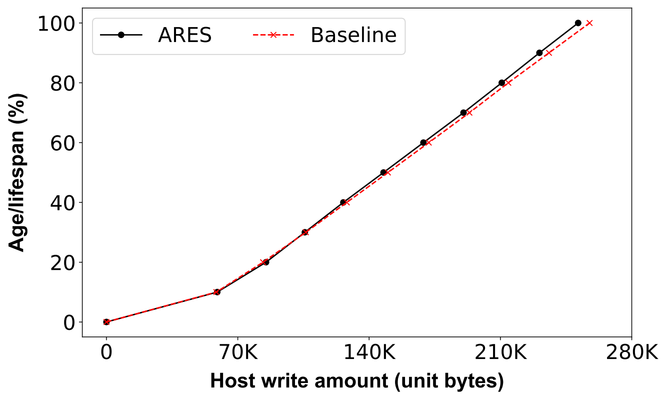
In this section, we evaluate ARES. This section aims to answer the following three research questions:



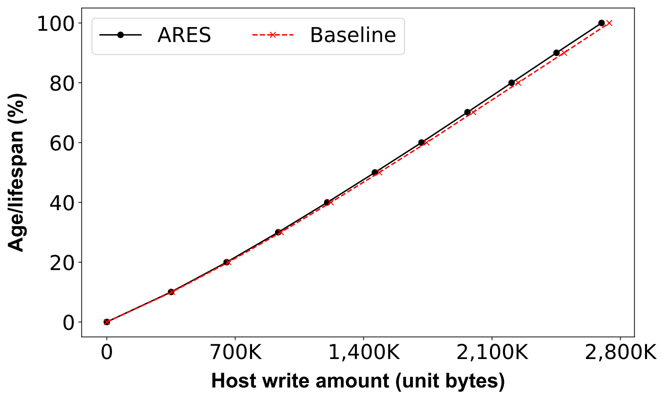
(a) 32GB USB flash drive workload-free



(b) 32GB USB flash drive workload-based



(c) 256GB micro SD Card workload-free



(d) 256GB micro SD Card workload-based

Fig. 4: Performance comparison of the write patterns produced from ARES against the baselines developed by domain experts

- **Effectiveness of ARES**: Does ARES produce competitive write patterns compared to the existing state-of-the-art write patterns manually designed by experts in the industry?
- **Necessity of our abstraction schemes**: Are our abstractions (relative write patterns and abstract relative write patterns) necessary to generate qualified write patterns in the learning?
- **Learned write pattern**: What are the learned abstract relative write patterns? Can we get insight from them?

A. *Effectiveness of* ARES

Now, we experimentally demonstrate the effectiveness of ARES by comparing the performance of the learned write patterns, produced by ARES, against baselines.

Baseline concrete write patterns. As baselines, we use the write patterns designed by test engineers in Samsung Electronics. Note that the baselines have been actually used in Samsung for write endurance testing. As benchmarks, we use 32GB USB flash drive and 256GB micro SD card, which are two popular types of flash-based storage devices in practice.

Settings. We implemented ARES on top of our write endurance test tool used in Samsung. In the write endurance test, we directly propagate the concrete write patterns to the flash-based storage through USB 3.0 ports without using file systems to remove the effects of file systems in the endurance test. Input storages are initialized as a full dirty state where all the sectors are written with valid data, which is a standard setting in write endurance test [3], [12]. All experiments were done on Intel Core i7-8750H CPU and 16 GB RAM on Windows 10. The learning took about 4 days for 32GB USB flash drives and 15 days for 256GB micro SD cards. Note that ARES is an offline approach, and we believe the learning cost (e.g., 15 days) is reasonable as ARES automatically generates write patterns without human effort.

In our evaluation, we consider the following two types of write endurance test:

- **Workload-based:** It considers only the write patterns belonging to enterprise endurance test workloads in JESD219 [12]. It assumes the storage device will be used for specific purposes, such as servers and data centers. The workloads contain write patterns that likely appear in the specific scenarios. ARES produced only the write patterns that belong to JESD219 in the learning procedure.
- **Workload-free:** It can use all the write patterns without considering specific scenarios. ARES considers all the write patterns in the learning procedure. Note that this

endurance test also has been actively used for testing various storage devices in practice.

TABLE II: Performance comparison between write patterns produced from ARES and the baselines.

Device	workload-based		workload-free	
	ARES	Baseline	ARES	Baseline
32GB USB flash drive	159K	162K	47K	64K
256GB micro SD Card	2,718K	2,739K	251K	257K

Table II compares the performance of the write patterns produced from ARES against the baselines. The number in the table presents the amount of written data until the storage s (32GB USB flash drive or 256GB micro SD Card) uses all its lifespan (e.g., $\texttt{Age}(s)/\texttt{Lifespan}(s) \geq 1$). In the workload-free test for a 32GB USB flash drive, for example, the write pattern produced from ARES made the storage retire after the amount of written data exceeded about $47K$ unit bytes, while the baseline write pattern made the storage retire after $64K$ unit bytes. For the metric, the lower is the better because an effective write pattern makes the storage retire with small amount of data write.

Results. Table II shows that ARES successfully generated better write patterns than the baselines developed by domain experts. For all cases, the write patterns produced from ARES shows better performance than the baselines. Especially for the workload-free endurance test for the 32GB USB flash drive, the write pattern produced by ARES is about 26% (47k vs 64k) more effective than the baseline. ARES is significantly more effective than the baselines for this case because the search space is not limited to specific workloads (user scenarios); ARES successfully found endurance-critical write patterns that the domain experts missed.

Figure 4 presents how effectively the baselines and the learned write patterns increase the age of the storages. In the figures, the black and red lines correspond to the learned write patterns and the baselines, respectively. The X-axis presents the amount of written data during the endurance test, and Y-axis presents how much lifespan is spent.

B. Necessity of Our Abstraction Schemes

Now, we demonstrate the necessity of our abstraction schemes. Figure 5 shows our abstract relative write patterns ($\widehat{RelWritePattern}$) are essential for learning qualified wrtie patterns (in 32GB USB flash drive workload-free). In Figure 5, X-axis presents the learning progress, and Y-axis presents the performance of the learned best write pattern. The red, orange, and grey-colored lines present how the performance (fitness) of the learned best write pattern (w_{best} in Algorithm 1) changes over the learning process. In the figure, the red-colored line corresponds to our approach that uses abstract relative write patterns ($\widehat{RelWritePattern}$) in the learning procedure. Grey and orange-colored lines present learning that uses relative write patterns (*RelWritePattern*) and concrete write patterns (*WritePattern*), respectively. The blue dotted line corresponds to the baseline (existing manually designed state-of-the-art concrete write pattern).

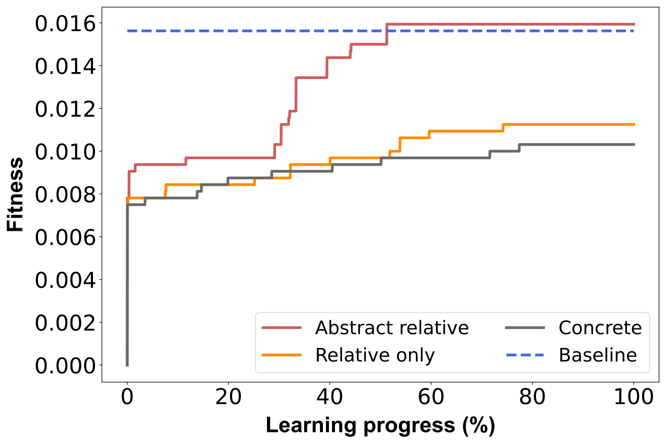


Fig. 5: How the performance of the learned write patterns changes over the learning

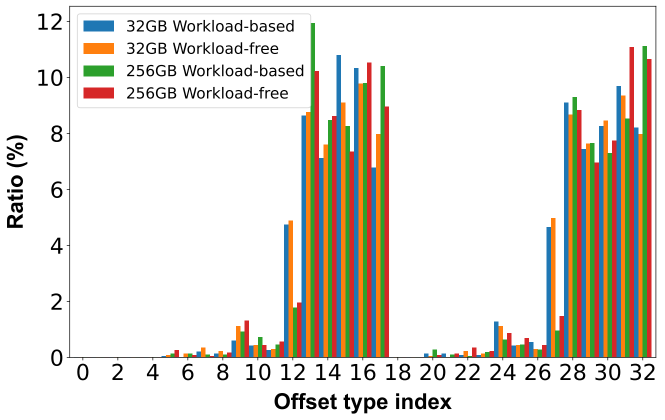


Fig. 6: Distribution of the offset type indices in the learned abstract relative write patterns.

As Figure 5 shows, only ARES with abstract relative write pattern (red line) successfully learned a write pattern that performs better than the baseline (blue dotted line). However, learning without our abstract relative write patterns (orange and grey lines) produced far less effective write patterns than the baseline. It shows our abstract relative write patterns effectively reduced the search space; the learning found a high-quality write pattern.

C. Learned Write Patterns

Now, we discuss what the learned abstract write patterns look like, and the insight obtained. Figure 6 presents distributions of $\hat{\Delta a}$ in the learned best scored (e.g., *Score*($\hat{w}$) in Algorithm 1) abstract relative write patterns. For all cases, a majority of $\hat{\Delta a}$ in the best abstract relative write patterns are in $\{13, 14, 15, 16, 17\}$ or $\{28, 29, 30, 31, 32\}$ where such $\hat{\Delta a}$ corresponds to $\Delta a > 16384$ or $\Delta a \leq -16384$ in Table I. It shows that qualified write patterns consist of many long (e.g., $|\Delta a| \geq 16384$) jumps.

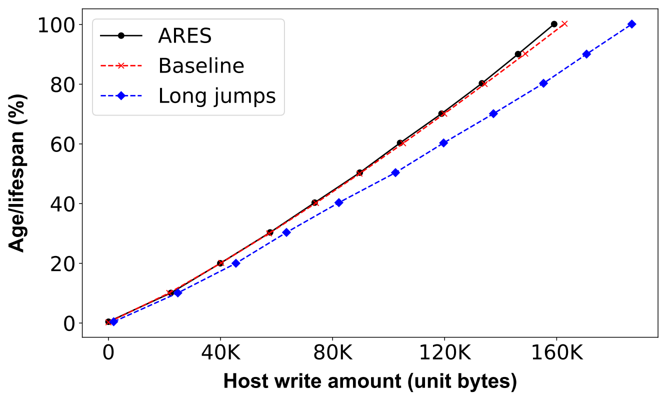


Fig. 7: Performance comparison of a write pattern consists of only long jumps against the others

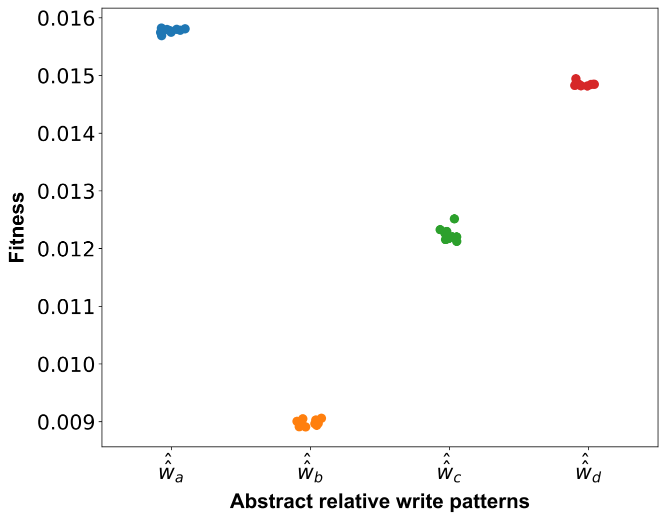


Fig. 8: Fitness by abstract relative write pattern

From the observation, we manually designed a write pattern that consists of only long jumps. Blue-colored line in Figure 7 presents the performance of the write pattern. As Figure 7 shows, however, the long jump write pattern (blue-colored) is less effective than the baseline (red-colored). It implies that the short jumps in the learned write pattern (black-colored) also play important roles; various types of jumps should be combined to become better than the baseline. As future work, our next step will be to determine which combinations are most crucial.

D. Effectiveness of Abstract Relative Write Patterns

We conducted an additional experiment to check whether abstract relative patterns represent similar concrete write patterns (e.g., $\{w \mid w \in \gamma(\hat{\hat{w}})\}$) that show similar performance (i.e., $Fitness(s, w)$) in the endurance test. We generate four abstract relative write patterns $\hat{\hat{w}}_a$, $\hat{\hat{w}}_b$, $\hat{\hat{w}}_c$, and $\hat{\hat{w}}_d$. For each abstract relative write pattern, we randomly produced ten belonging concrete write patterns (e.g., $\{w_a^1, w_a^2, \ldots, , w_a^{10}\} \subseteq \gamma(\hat{\hat{w}}_a)$). Then, we compare the performance of the belonging concrete write patterns to check whether they show similar performance. Figure 8 show the performance of the concrete write patterns. The x-axis presents the performance ($Fitness(s, w)$). The blue, orange, green, and red-colored dots present concrete write patterns belong to $\hat{\hat{w}}_a$, $\hat{\hat{w}}_b$, $\hat{\hat{w}}_c$, and $\hat{\hat{w}}_d$, respectively. As Figure 8 shows, the four abstract relative write patterns are qualified abstractions that the belonging concrete write patterns show similar performance. For example, the standard deviation of the blue, orange, green, red dots are $3e-5$, $5e-5$, $1e-4$, and $3e-5$, respectively. It shows that the abstraction scheme is well-designed; our abstract relative write patterns effectively reduced the search space without significantly losing a chance of finding good concrete write patterns.

V. Related Works

In this section, we discuss prior works related to ARES. To our knowledge, writing endurance tests for flash-based storage have been conducted manually not only in Samsung but also in other companies, and therefore ARES represents the first attempt to automate the process in an industry context.

Industry standard for write endurance test. In our evaluation, ARES is evaluated under workload-based, which belongs to industry standard, and workload-free. JEDEC (Joint Electron Device Engineering Council), a semiconductor engineering trade organization and standardization organization, presents JESD218 and JESD219 as write endurance test standards for flash-based storages. JESD218 [3] describes the write endurance requirements and test methods, and JESD219 [12] specifies the workload characteristics to be considered in the endurance test. JEDEC standards are widely used in representative manufacturers such as Samsung Electronics, SK Hynix, Micron Technology, Western Digital Technologies, and Kioxia Corporation. The standard, however, is designed to test SSD; the standard is inappropriate for flash-based storages such as USB flash drives. Further, JESD219 describes workloads for only two applications: enterprise and client. Note that the applications of flash-based storage are not limited to these two cases. The usage of flash-based storage is constantly expanding. For example, machine learning processes [20], [21] use flash-based storage devices. The storages also need to be tested under workload-free that does not have specific user scenarios, and our evaluation shows that ARES is very effective in this case (e.g., Figure 4a).

Learning inputs for software testing. Machine learning techniques have been used for generating qualified inputs for testing. Like ARES, GA-Prof [22] uses genetic algorithms to find test inputs that maximize the elapsed execution time of the Application Under Test(AUT). SMARTEST [23] learns language models that guide symbolic execution to effectively find vulnerable inputs (transaction sequences) for smart contracts. QBE [24] uses reinforcement learning to effectively explore test inputs (GUI actions) for Android GUI testing to detect crashes. iPerfXRL [25] uses deep reinforcement learning to

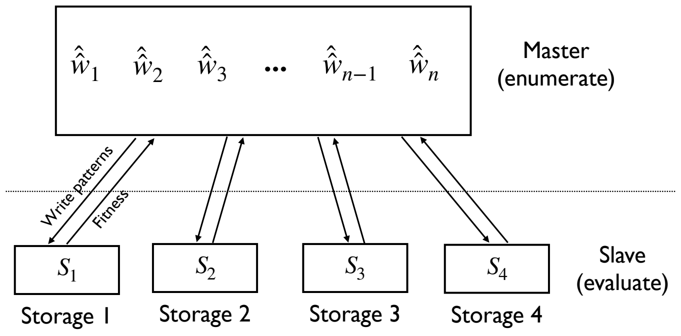


Fig. 9: Example of distributed genetic algorithm (master-slave model with 4-slaves)

identify test inputs for revealing performance bottlenecks of software systems. To our best knowledge, ARES is the first approach to learn test inputs for write endurance testing.

VI. Limitation and Future work

Currently, our algorithm learns write patterns with single storage; the learning is limited to the lifespan of the given storage. Distributed learning [26] using multiple storages can learn better write patterns. For example, we can apply master-slave model [27] described in Figure 9 that uses four storages. As it learns write patterns until the four storages retire, the algorithm enumerates (e.g., three times) more write patterns in the learning. Further, evaluating write patterns can be done in parallel; the learning time will be similar.

VII. Conclusion

In this paper, we presented ARES for automatically generating worse-case write patterns for endurance test of flash-based storage devices. Our approach uses abstract relative write patterns for reducing the search space and exploit their properties to produce concrete write patterns via genetic algorithm. Experimental results confirm that the learned write patterns show better performance than the existing state-of-the art write patterns manually designed by test engineers in Samsung Electronics.

Acknowledgment

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2020-0-01337,(SW STAR LAB) Research on Highly-Practical Automated Software Repair) and the ICT Creative Consilience program(IITP-2023-2020-0-01819). This work was also supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT)(No. 2021R1A5A1021944).

References

[1] B. Eitan and A. Roy, *Binary and Multilevel Flash Cells*. Boston, MA: Springer US, 1999, pp. 91–152. [Online]. Available: https://doi.org/10.1007/978-1-4615-5015-0_3

[2] A. Modelli, A. Visconti, and R. Bez, "Advanced flash memory reliability," in *2004 International Conference on Integrated Circuit Design and Technology (IEEE Cat. No.04EX866)*, May 2004, pp. 211–218.

[3] *Solid-state drive (ssd) requirements and endurance test method*, JEDEC Solid State Technology Association JEDEC Standard No. 218, JESD218, 2010.

[4] R. Micheloni, "Solid-state drive (ssd): A nonvolatile storage system," *Proceedings of the IEEE*, vol. 105, no. 4, pp. 583–588, April 2017.

[5] J. Axelson, *USB mass storage: designing and programming devices and embedded hosts*. lakeview research llc, 2006.

[6] *Embedded Multimedia Card (e-MMC)*, JEDEC Solid State Technology Association JEDEC Standard No. 84-B51, JESD84-B51, 2012.

[7] *Universal Flash Storage (UFS)*, JEDEC Solid State Technology Association JEDEC Standard No. 220C, JESD220C, 2018.

[8] C. Gao, A. Gutierrez, M. Rajan, R. G. Dreslinski, T. Mudge, and C.-J. Wu, "A study of mobile device utilization," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2015, pp. 225–234.

[9] S. Vlad-Andrei, O. M. Andrei, A. Sebastian, and F. T. Alexandru, "An overview on nonvolatile memories used in automotive industry," in *2021 International Conference on Electromechanical and Energy Systems (SIELMEN)*, Oct 2021, pp. 517–520.

[10] L. Tan and N. Wang, "Future internet: The internet of things," in *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, vol. 5, Aug 2010, pp. V5–376–V5–380.

[11] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *Proceedings of the IEEE*, vol. 91, no. 4, pp. 489–502, April 2003.

[12] *Solid-state drive (ssd) requirements and endurance test method*, JEDEC Solid State Technology Association JEDEC Standard No. 219, JESD219, 2010.

[13] H. Sun, X. Qin, F. Wu, and C. Xie, "Measuring and analyzing write amplification characteristics of solid state disks," in *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, Aug 2013, pp. 212–221.

[14] G. Wu and X. He, "Delta-ftl: Improving ssd lifetime via exploiting content locality," in *Proceedings of the 7th ACM European Conference on Computer Systems*, ser. EuroSys '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 253–266. [Online]. Available: https://doi.org/10.1145/2168836.2168862

[15] S. Lee, D. Shin, Y.-J. Kim, and J. Kim, "Last: Locality-aware sector translation for nand flash memory-based storage systems," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 6, p. 36–42, oct 2008. [Online]. Available: https://doi.org/10.1145/1453775.1453783

[16] J. Kim, S. Seo, D. Jung, J.-S. Kim, and J. Huh, "Parameter-aware i/o management for solid state disks (ssds)," *IEEE Transactions on Computers*, vol. 61, no. 5, pp. 636–649, May 2012.

[17] O. Kramer, *Genetic Algorithms*. Cham: Springer International Publishing, 2017, pp. 11–19. [Online]. Available: https://doi.org/10.1007/978-3-319-52156-5_2

[18] A. J. Umbarkar and P. D. Sheth, "Crossover operators in genetic algorithms: a review." *ICTACT journal on soft computing*, vol. 6, no. 1, 2015.

[19] C. Chudasama, S. Shah, and M. Panchal, "Comparison of parents selection methods of genetic algorithm for tsp," in *International Conference on Computer Communication and Networks CSI-COMNET-2011, Proceedings*, vol. 85, 2011, p. 87.

[20] J. Bae, J. Lee, Y. Jin, S. Son, S. Kim, H. Jang, T. J. Ham, and J. W. Lee, "FlashNeuron: SSD-Enabled Large-Batch training of very deep neural networks," in *19th USENIX Conference on File and Storage Technologies (FAST 21)*. USENIX Association, Feb. 2021, pp. 387–401. [Online]. Available: https://www.usenix.org/conference/fast21/presentation/bae

[21] H. Choe, S. Lee, H. Nam, S. Park, S. Kim, E.-Y. Chung, and S. Yoon, "Near-data processing for differentiable machine learning models," *arXiv preprint arXiv:1610.02273*, 2016.

[22] D. Shen, Q. Luo, D. Poshyvanyk, and M. Grechanik, "Automating performance bottleneck detection using search-based application profiling," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ser. ISSTA 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 270–281. [Online]. Available: https://doi.org/10.1145/2771783.2771816

[23] S. So, S. Hong, and H. Oh, "SmarTest: Effectively hunting vulnerable transaction sequences in smart contracts through language Model-

Guided symbolic execution," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 1361–1378. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/so

[24] Y. Koroglu, A. Sen, O. Muslu, Y. Mete, C. Ulker, T. Tanriverdi, and Y. Donmez, "Qbe: Qlearning-based exploration of android applications," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, April 2018, pp. 105–115.

[25] T. Ahmad, A. Ashraf, D. Truscan, A. Domi, and I. Porres, "Using deep reinforcement learning for exploratory performance testing of software systems with multi-dimensional input spaces," *IEEE Access*, vol. 8, pp. 195 000–195 020, 2020.

[26] D. V. Carvalho, E. M. Pereira, and J. S. Cardoso, "Machine learning interpretability: A survey on methods and metrics," *Electronics*, vol. 8, no. 8, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/8/832

[27] Y.-J. Gong, W.-N. Chen, Z.-H. Zhan, J. Zhang, Y. Li, Q. Zhang, and J.-J. Li, "Distributed evolutionary algorithms and their models: A survey of the state-of-the-art," *Applied Soft Computing*, vol. 34, pp. 286–300, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494615002987