Polling sanitization to balance I/O latency and data security of high-density SSDs

Jiaojiao Wu, Zhigang Cai, Fan Yang, Jun Li, François Trahay, Zheng Yang, Chao Wang, Jianwei Liao

▶ **To cite this version:**

Jiaojiao Wu, Zhigang Cai, Fan Yang, Jun Li, François Trahay, et al.. Polling sanitization to balance I/O latency and data security of high-density SSDs. Transactions on Storage, 2024, 20 (2), pp.1-23. 10.1145/3639826 . hal-04378830

HAL Id: hal-04378830
https://hal.science/hal-04378830

Submitted on 8 Jan 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Polling Sanitization to Balance I/O Latency and Data Security of High-density SSDs

JIAOJIAO WU, ZHIGANG CAI, FAN YANG, and JUN LI, Southwest University, China
FRANCOIS TRAHAY, Telecom SudParis, France
ZHENG YANG, CHAO WANG, and JIANWEI LIAO, Southwest University, China

Sanitization is an effective approach for ensuring data security through scrubbing invalid but sensitive data pages, with the cost of impacts on storage performance due to moving out valid pages from the sanitization-required wordline, which is a logical read/write unit and consists of multiple pages in high-density SSDs. To minimize the impacts on I/O latency and data security, this paper proposes a polling-based scheduling approach for data sanitization in high-density SSDs. Our method polls a specific SSD channel for completing data sanitization at the block granularity, meanwhile other channels can still service I/O requests. Furthermore, our method assigns a low priority to the blocks that are more likely to have future *adjacent page* invalidations inside sanitization-required wordlines, while selecting the sanitization block, to minimize the negative impacts of moving valid pages. Through a series of emulation experiments on several disk traces of real-world applications, we show that our proposal can decrease the negative effects of data sanitization in terms of the risk-performance index, which is a united time metric of I/O responsiveness and the unsafe time interval, by 16.34% on average, compared to related sanitization methods.

CCS Concepts: • **Computer systems organization → Embedded software**.

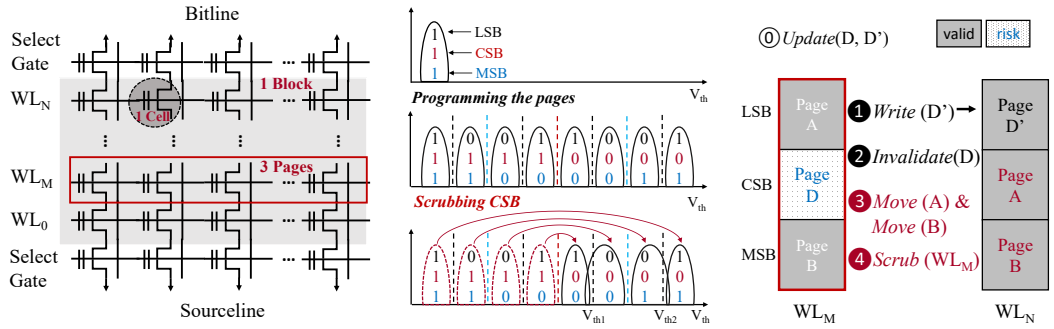
Additional Key Words and Phrases: High-density SSDs, I/O Latency, Data Security, Sanitization, Polling, Scheduling.

1 INTRODUCTION

Solid-state drives (SSDs) have replaced traditional hard-disk drives (HDDs), as the mainstream storage devices for resource-constrained environments such as embedded systems, due to their advantages of high performance and low power consumption [1–3]. To further enhance I/O performance, modern SSD products incorporate and leverage channel-level parallelism in the hardware [4]. Each channel is composed of many blocks, and each block consists of several pages. *Read* and *Write* operations take place at the page-level granularity, whereas *Erase* happens at the block-level granularity [5]. Specially, high-density SSDs offer a competitive per-byte price advantage compared to conventional hard disk drives and have thus become mainstream in the storage market [6]. For example, a modern Trinary-Level Cell (TLC) can store up to 3-bits of information per cell [7] and this paper uses it as the default high-density SSD device. As shown in Figure 1 (a), all TLC cells belonging to a wordline (WL) can store up to three bits per cell, which are defined as Lower Significant Bit (LSB), Central Significant Bit (CSB) and Most Significant Bit (MSB). The same type bits of all TLC cells in the wordline make up an SSD page. In other words, each TLC wordline corresponds to 3-type pages on the flash, including an LSB page, a CSB page, and an MSB page.

Due to the features of *out-of-place update* and *erase-before-program* in SSDs, data pages of the deleted files and obsolete data pages after updating are merely marked as invalidated, which means their contents remain in the flash memory. Such invalid data can be recovered through digital forensics unless their relevant block has been erased [8]. According to *the UK Data Protection Act 2018*, however, the deletion of information must be formal, implying that the contents should not be

Corresponding author: J. Liao, Email: liaojianwei@il.is.s.u-tokyo.ac.jp. He works for College of Computer and Information Science, Southwest University, P.R. China.
Authors' addresses: Jiaojiao Wu; Zhigang Cai; Fan Yang; Jun Li, Southwest University, Chongqing, China, 400715; Francois Trahay, Telecom SudParis, France; Zheng Yang; Chao Wang; Jianwei Liao, Southwest University, Chongqing, China, 400715.



(a) Organization of a TLC SSD block (b) Programming and scrubbing in TLC (c)Wordline-level scrubbing

Fig. 1. Data sanitization in TLC SSDs, caused by an update request. (a) The organization of a TLC SSD block, consisting of multiple wordlines, and each wordline includes three pages. (b) The process of programming and page-level scrubbing in the wordline (assuming the CSB page is sanitized, and each page holds only one-bit data). (c) The process of wordline-level scrubbing.

recoverable in any way [9, 10]. Securely removing obsolete data pages (so-called invalid pages) from SSDs becomes critical for protecting owners' privacy. For example, embedded systems are being deployed in a wide range of application scenarios, including the control of safety-critical systems and data collection in hostile environments [11, 12]. That is to say, SSD devices deployed in such systems are likely to be more vulnerable to both open environmental operations and intentional attacks due to their embedded nature [12, 13].

To address this security issue, the scrubbing-based technique was proposed to efficiently fulfill the sanitization of invalid data inside SSDs [8, 14]. Specifically, scrubbing any one page will introduce serious disturbances to adjacent valid pages in the same WL of high-density SSDs [14, 15]. In other words, during the process of scrubbing-based sanitization, it requires raising the sanitization voltage(V_{th}) to all flash cells in the same WL and causes some states to change, thus rendering it unfeasible to discern the initial data with the reference voltage. In the example shown in Figure 1 (b), after sanitizing the CSB page, the reference voltage of V_{th1} cannot confirm whether the valid data of MSB and LSB should be `10` or `00`, and the reference voltage of V_{th2} fails to confirm whther the valid data of MSB and LSB should be `01` or `11`. To eliminate scrubbing disturbances in high-density SSDs, the WL-level sanitization approach involves coping out the valid data pages (refer to as *page moves*) and then scrubbing the whole WL, as shown in Figure 1 (c). However, such operations affect I/O processing and the lifetime of SSDs due to the additional reads and writes caused by scrubbing [15]. Immediate sanitization (*ISan*) [8, 16] and delayed sanitization (e.g., periodical sanitization, *PSan*) [17] are two typical sanitization schemes. *ISan* intermittently carries out sanitization operations on the obsolete data pages as soon as they have been invalidated. Conversely, *PSan* performs sanitization in a periodical manner, and the time frame for sanitization is commonly determined by the data owners according to their assessment of system confidentiality [18]. Since *PSan* collects many pieces of deleted data and securely deletes them simultaneously, it can avoid considerable *page moves* in a round of sanitization [17].

On the other side, *Dwell Time* is a critical security success indicator in the domain of information security, which is defined as "*the time from the point a threat successfully enters your environment to when the threat is completely remediated* [19]". We have defined a similar measure of unsafe time interval (UTI) in our context, as the time interval between data invalidation (i.e., the threat of data

breach enters) and the completion of sanitization (i.e., the threat of data breach can be remediated), to represent the leakage risk of invalid data. A small value of UTI means less risk of data leakage in SSD devices. This suggests that *ISan* can ensure the shortest values of UTI, while *PSan* can provide better I/O responsiveness.

It is true that SSD vendors often prioritize I/O performance over ensuring a small value of UTI, while users may expect a better guarantee of data security. To address the issue of balancing I/O performance and data security in high-density SSDs, this paper proposes polling-based sanitization scheduling. In summary, this paper makes the following three contributions:

- We propose a generic polling-based sanitization scheduling method for high-density SSDs, named *PollSan*. Our method requires a polling cycle to work on a specific SSD channel, for completing data sanitization tasks at the block granularity. It can noticeably decrease the negative impacts of data sanitization on integrated metrics, including I/O responsiveness and data security when running user applications.
- We build an artificial neural network-based model to estimate the number of future *adjacent page* invalidations on SSD blocks. This model helps us to delay sanitization on the block if its pages inside sanitization-required wordlines (referred as *adjacent pages*) are likely to be invalidated with a high probability in the future, for achieving the goal of minimizing sanitization overhead. Our model analyzes the factors of write count, invalidation count, and the data source feature (i.e., flushed by normal writes, wear leveling or garbage collection) with respect to the given block. After that, it forecasts the number of future *adjacent page* invalidations inside sanitization-required wordlines of the block. Furthermore, we consider both time and space constraints on the SSD channel to decide the number of sanitization blocks in the current polling cycle.
- We conduct comprehensive evaluation tests by replaying the selected I/O traces from real-world applications on a simulated SSD device. Our measurements indicate that the proposed approach effectively mitigates the negative effects of data sanitization on the metric of unifying I/O responsiveness and UTI by `16.34%` on average, compared to existing sanitization schemes for high-density SSDs.

The rest of the paper is organized as follows: Section 2 depicts the background and motivation. The approach of polling sanitization is presented in Section 3. Section 4 describes the evaluation experiments and relevant discussions. Section 5 presents the related work of sanitization. At last, the paper is concluded in Section 6.

2 BACKGROUND AND MOTIVATION

2.1 Flash-based SSD Preliminaries

Flash memory is organized as two-dimensional arrays of floating-gate transistors. A number of cells electrically connected to a WL (with multiple pages in high-density flash memory) and multiple WLs form a block. Specifically, flash-based SSDs usually consist of one or more planes, and each plane contains several blocks. A block is a basic unit for erasing, and is composed of many pages that are the basic units of write/read operations.

SSD devices have a special write feature of *out-of-place update*, indicating that any updated page cannot be directly overwritten on the original page until its residing block has been erased. In other words, SSDs require invalidating the original data page, and flushing the updated data to another free page, for completing an update request. On the other hand, the main firmware layer of SSDs is the flash translation layer (FTL), which takes charge of garbage collection (GC) and wear leveling [20]. In particular, the GC module periodically reclaims the space of invalid pages in the block to free up flash memory for new data by erasing the SSD block. The wear leveling

module manages the limited endurance of the flash memory by ensuring uniform program-erase distribution on all the blocks, to extend the lifetime of SSD devices.

2.2 Sanitization in SSDs

In conventional hard disk drives (HDDs), obsolete data can be overwritten directly, or securely deleted through commands for ATA and SCSI protocols. However, SSDs exhibit certain unique challenges for the secure deletion of obsolete data, due to their special characteristics of *out-of-place update* and *erase-before-program* [10]. The invalid data pages of SSDs still hold the obsolete data, which directly leads to a risk of data leakage, and this risk exists until their blocks have been erased via GC operations. To meet the requirement of secure deletion of obsolete data for SSD devices, many advanced methods have been proposed, such as encryption with ephemeral key-based [17, 21, 22], physical erase-based [23, 24], and scrubbing-based methods [8, 14, 15, 26].

The primary concept behind ephemeral key-based methods is to eliminate unused encryption keys for sensitive data encryption to guarantee that the sensitive data can never be retrieved [17, 21, 22]. If the secure data becomes outdated, it will be securely deleted along with its corresponding small-sized key. This process ensures that the encrypted secure data is rendered irrecoverable. Nevertheless, attackers may still attempt to decipher the outdated secure data before initiating the actual erase operation on the target block containing the sensitive data [27].

The erase-based methods erase the block that has invalid sensitive data by proactively triggering GC or the self-defined erase operations once the data pages have been invalidated [23, 24]. The main drawback of erasure-based methods is the use of flash blocks as the basic secure deletion unit. This leads to significant live data migration overhead, which in turn results in poor I/O performance and short flash memory lifespan.

To alleviate the overhead caused by the erased-based method, Wei et al [8] proposed a technique of scrubbing a page with all zeros, which supports secure deletion at the page level. This technique was later improved by other researchers [14, 15, 25]. The basic idea of page-level sanitization is to create an all-zero page with an overwrite operation, thus making it is impossible to retrieve the original data of invalid pages. More exactly, the scrubbing technique increases the voltage of the flash cell in the wordline for mixing the different states, to fulfill page-level data sanitization. However, implementing such a technique in high-density SSDs is challenging, as it may damage other data pages in the same wordline. To address this issue, it requires migrating valid pages in the scrubbed WL, causing considerable performance penalties. As the example previously shown in Figure 1 (c), each WL in the TLC NAND flash memory consists of three pages. If one of the three pages needs to be sanitized, the remaining valid pages will be moved to other WL (s) before scrubbing the WL, indicating two additional read operations and two additional write operations.

2.3 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a widely used model in machine learning. It is inspired by biological neural networks and consist of many neurons and connections between them. Each connection has a weight that is used to calculate the output of each neuron. ANN represents a promising modeling technique, especially for datasets having non-linear relationships. It can learn rules from input-output pairs and use these rules to predict outputs for new inputs, making them perform well in various tasks [28, 29].

Because ANNs can work effectively in the offline mode, they have been widely applied in various aspects of SSD optimization, including SSD storage management, garbage collection, and designing error correction codes [30–32]. It is worth mentioning that when employing ANNs to optimize SSD performance, they are typically trained offline, since online training requires a significant

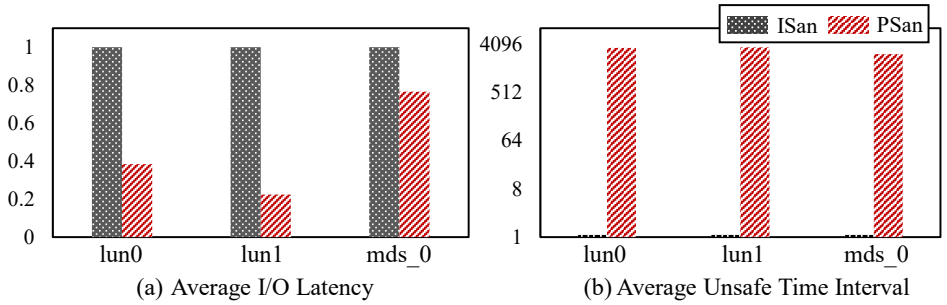


Fig. 2. Performance comparison of two sanitization schemes of *ISan* and *PSan*. Results are normalized to *ISan*, indicating the values of all traces with *ISan* are `1.0`.

amount of computational resources, which can negatively impact the overall performance of the SSD device.

2.4 Motivation

Securely deleting electronic data once they are not actually needed is important for both individuals and organizations. Regulations related to data protection, such as the EU General Data Protection Regulation (GDPR) [33] and the Health Insurance Portability and Accountability Act (HIPAA) [34], require consistent procedures for data protection [35]. More importantly, securely deleting invalid data in SSD devices is bound to incur costs, and WL-level scrubbing is an effective data sanitization method for SSDs when comparing to conventional ephemeral key-based and physical erase-based deletion methods. Therefore, the negative impacts of data sanitization in high-density solid-state drives become more obvious, due to the compacted structure of flash cells. As discussed in Section 1, *ISan* [8] can ensure the shortest values of UTI because it performs sanitization immediately after data becomes invalid. Meanwhile, *PSan* [17] can guarantee the best I/O performance because it generates fewer adjacent page interference and leads to fewer page migrations.

In order to further quantify the strength and limitations of conventional WL-level sanitization schemes, we carried out an experimental study by employing *ISan* and *PSan* to complete data sanitization on high-density TLC SSDs. Section 4.1 describes the details of the experimental platform and benchmarks. Figure 2 shows the experimental results of two conventional sanitization policies after running three selected benchmarks.

Figure 2 (a) shows that *PSan* can noticeably reduce the I/O latency, compared to *ISan*. High-density SSDs commonly have multiple pages in each wordline, and sanitizing a data page with *ISan* always deduces one or more migrations of adjacent pages, which damages I/O responsiveness. On the other side, *PSan* can minimize the number of *page moves* in the sanitization, since *adjacent pages* might be invalidated within a sanitization round. Thus, it can contribute to better I/O performance. Figure 2 (b) presents the results of UTI. As seen, *ISan* performs extremely well on the measure of UTI because it carries out instant data sanitization when the data become invalid. Meanhwile, *PSan* has large UTI values, indicating weak guarantees of data security, this is because it sanitizes the invalid data pages with a peridical manner.

In summary, the security of data and I/O responsiveness are two crucial indicators for user applications, and it is not expected to completely sacrifice either aspect [35, 36]. Such observations motivate us to propose a novel generic sanitization scheme, to further minimize the impacts on I/O latency and data security of high-density SSDs, where each wordline of the SSD block consists of multiple pages.

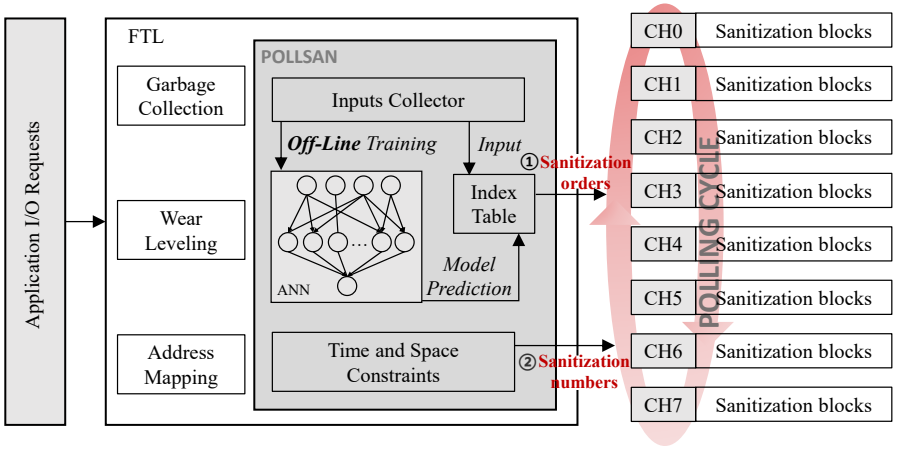


Fig. 3. High-level overview of *PollSan*, working at Flash Translation Layer (FTL) in 8-channel SSDs, for sanitizing blocks with a polling manner over all channels.

3 POLLING-BASED SANITIZATION SCHEDULING

3.1 Architectural Overview

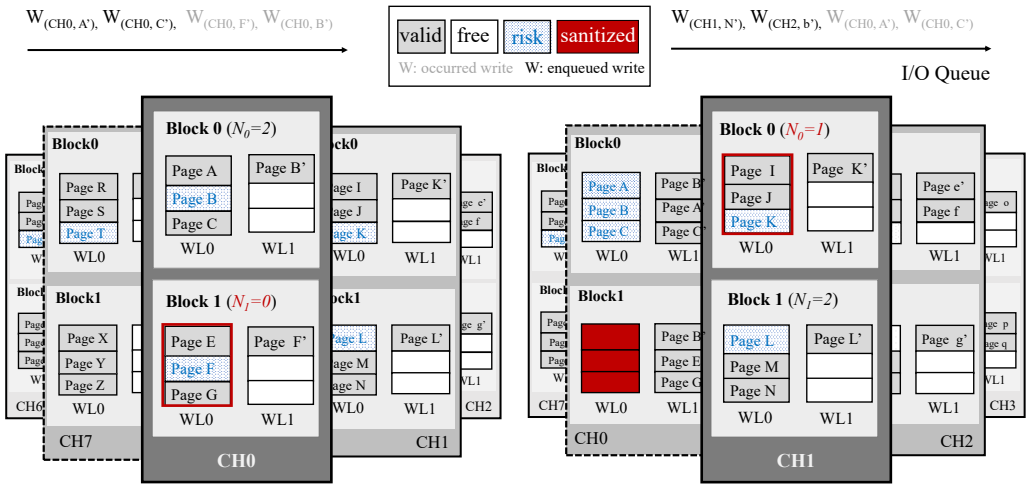
We propose a scheme of polling-based sanitization scheduling, called *PollSan*, which requires a polling cycle[1] to work on a specific SSD channel, for completing data sanitization at block granularity. Figure 3 shows the architectural overview of *PollSan*, that works at the Flash Translation Layer (FTL) of SSDs. As seen, our proposed method of polling-based sanitization mainly addresses two issues: ①***how*** to rate the priorities of sanitization blocks (cf. Section 3.2), and ②***how*** to decide the quantity of block sanitization on the channel in the current polling cycle (cf. Section 3.3).

To be specific, in the polling cycle, *PollSan* preferentially performs sanitization on the blocks if their *adjacent pages* are not likely to be invalided in the future, by resorting to the outputs of our proposed pre-trained ANN model. Consequently, it has time to accumulate more *adjacent page* invalidations, to eventually reduce the number of *page moves* after all sanitization processes. Note that we collected a large amount of data as the dataset after running various workloads of real-world applications. Subsequently, we trained the ANN model on the dataset to predict future *adjacent page* invalidations. We emphasize that the ANN model was trained offline by considering SSDs commonly have limited computation and memory resources. In other words, we generated an index table after multi-round training, and embedded it into the *PollSan* module in the FTL fireware.

Figure 4 illustrates an example of our proposed sanitization approach, polling at SSD channels. Note that we support sanitizing multiple blocks in each polling cycle, see Section 3.3. As seen in Figure 4 (a), both *Page B* in *Block 0* and *Page F* in *Block 1* were updated and thus require sanitization in the polling cycle on *CH0* at T_0, and *PollSan* selects *Block 1* as the sanitization target block. This is because our prediction model suggests that *Block 1* may not have any *adjacent page* invalidations ($N_1 = 0$) in the future, whereas *Block 0* probably has `two` *adjacent page* invalidations ($N_0 = 2$). As discussed, delaying sanitization on the blocks having future *adjacent page* invalidations, contributes to the reduction of moving valid data pages.

After completing data sanitization of the current SSD channel, the *PollSan* scheme will move on to the next channel for data sanitization. Figure 4 (b) demonstrates the sanitization selection

[1]A polling cycle indicates one time slot in which a specific SSD channel has been polled for data sanitization in our context, and the length of each slot may be different.



(a) Polling on CH0 at T_0 (b) Polling on CH1 at T_1

Fig. 4. Polling-based block sanitization in TLC SSDs with the proposed *PollSan* scheme. For illustration simplicity, we assume each channel consists of 2 blocks and each block has 2 wordlines.

on *CH1* at the polling cycle of T_1, which is similar to the case on *CH0* at the polling cycle of T_0. *PollSan* will choose *Block 0* on *CH1* for sanitization because the predictive model indicates that it will generate fewer *adjacent page* invalidations ($N_0 = 1$) in the future. The noticeable information is that *WL0* of *Block 1* on *CH0* was sanitized at T_0, while the data pages of *WL0* of *Block 0* on the same channel are all invalidated at T_1. Then, we can sanitize the wordline of *WL0* of *Block 0* without any *page moves* in the next polling cycle on *CH0*, and this is the primary goal of delaying sanitization on such blocks. As a consequence, we can yield better I/O performance and SSD endurance, comparing to existing work of data sanitization.

In order to ensure *PollSan* performing not worse than *PSan* on the measure of UTI, we set *PollSan* switching to *PSan* when the time frame (i.e., the size of time window with *PSan*) runs out but the SSD device still has unsanitized data pages.

3.2 Predicting Future Adjacent Page Invalidations

Pollsan leverages ANN to predict the number of *adjacent page* invalidation of sanitization block, by considering our dataset has non-linear relationships. This section describes the specifications of our prediction model on the basis of ANN.

3.2.1 Inputs and Output. We employ the notations defined in Table 1 and make use of an artificial neural network to predict the number of future *adjacent page* invalidations, that is the base of block sanitization priority. To minimize modeling overhead, our model analyzes relevant features at the block level, by regarding all wordlines in the block that have similar modeling features. *PollSan* delays the sanitization on the SSD blocks if they will have more *adjacent page* invalidations in the future, which will consequently result in fewer *page moves* after all data sanitization, thereby reducing the overhead of data sanitization. That is, the blocks having considerable *adjacent page* invalidations in the future will be assigned a low priority when scheduling block sanitization.

Table 1. Notation descriptions used in the model

Symbol	Explanation
Y	Number of future adjacent page invalidations
X_{GC}	Whether there are pages with GC
X_{Wear}	Whether there are pages with wear leveling
X_{Wr}	Number of occurred write requests
X_{Inv}	Number of occurred invalid pages

To estimate the number of future *adjacent page* invalidations (i.e., Y), we considered 4 impact features related to the given block as the inputs of ANNs, including whether there is a migrated page caused by GC, whether there is a migrated page induced by wear leveling, the number of write requests, and the number of invalidated data pages. Specifically, both migrated data pages caused by GC and wear leveling might be cold update pages, and they are not likely to be invalidated in the future [37]. The number of write requests reflects how many data pages are occupied with valid/invalid data in the block, corresponding to the chances to have *adjacent page* invalidations in the block. The number of invalidated data pages indicates how many data pages are filled with invalid data pages that are needed to be sanitized. A larger value means that it is more likely to have *adjacent page* invalidations in the future.

In summary, we train the neural network by inputting the block features and the characteristics of data in the block. As a result, the output layer of our model can predict the future *adjacent page* invalidations in target blocks. A higher value of future *adjacent page* invalidations implies that the block will generate more invalid pages in the future, and it should postpone the sanitization operation on that block, by assigning a lower sanitization priority.

3.2.2 Dataset. For training the model to cope with varying workloads, we collect a large amount of data entries of $(Y, X_{GC}, X_{Wear}, X_{Wr}, X_{Inv})$. Specifically, we replay the workloads in the trace collection of Microsoft Research Cambridge (MSRC) [38], and the trace collection of enterprise virtual desktop infrastructure (VDI) [39] (except 4 traces for model testing), and collect a total of 120,635 entries. All variables in the entries collected from the traces are normalized in the range of 0–1, which can accelerate the convergence speed of training and make the model more robust. We use 80% of the data as the training set to train the parameters of the neural network, and 20% of the data as the validation set to select the optimal model and hyperparameters. Since SSDs are resource-limited devices, we train the artificial neural network with an offline mode.

3.2.3 Model Architecture and Deployment. For predicting the future *adjacent page* invalidations on the given SSD block, we selected a three-layer artificial neural network which is a relatively wide design [40, 41]. The neural network consists of three layers, including the input layer, the hidden layer, and the output layer. The input layer receives data related to the features of blocks. The hidden layer comprises three layers, and each hidden layer having 128 neurons. The output layer is the future *adjacent page* invalidations in target blocks. Furthermore, we utilized Rectified Linear Units (*ReLU*) as the activation function throughout the network. The *cross-entropy loss* function and the Adam optimizer were used in our model. The learning rate of ANN was 0.001, which determines the magnitude of weight updates for each sample during the training process. The training loss consistently decreases and gradually approaches a lower value, indicating that the pre-training model is suitable for the future *adjacent page* invalidations prediction.

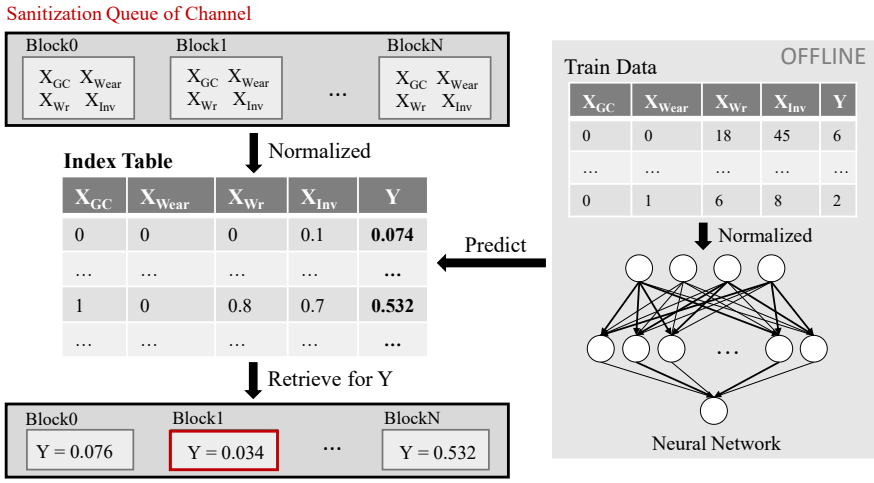


Fig. 5. ANN-based sanitization priority prediction on the granularity of SSD block. The index table is the output of ANN by inputting the collection data items.

To minimize computational costs and memory overhead inside SSDs, we used collected data and trained a neural network model in **an offline manner**, to generate an index table for dispatching block sanitizations in the given polling cycle. As illustrated in Figure 5, we first conduct offline training of the neural network model on the collected training dataset, and obtain a well-trained ANN model. Then, we can utilize the pre-trained neural network model to forecast the number of future *adjacent page* invalidations of the sanitization-required block. This is achieved by inputting a (default) total of `400` data combinations between `0` and `1` to the trained model, resulting in an output index table of different input combinations. Specifically, the ranges of input combinations of the four features are classified as two categories. X_{GC} and X_{WL} are binary variables with a value of `0` or `1`, X_{Wr} and X_{Inv} vary within [0, 1.0], by increments of `0.1`.

When predicting the sanitization priority of a given block, *PollSan* first collects the values of four considered features on the candidate block, and then standardizes them as an input combination. After that, we retrieved the corresponding output in the index table on the basis of the status of blocks that are required for sanitization in the same channel. A larger output implies a greater likelihood of subsequent *adjacent page* invalidations in the block. Consequently, we select the blocks with smaller outputs for carrying out data sanitization.

3.3 Scheduling on Sanitization Blocks

Our approach of *PollSan* polls at the channel level. In each polling cycle on the specific channel, it first identifies blocks that contain sanitization-required wordlines and calculates the sanitization priorities[2], with the pre-trained ANN model. After determining the priority of sanitization, *PollSan* will estimate the number of blocks to be sanitized on the current polling channel, on the basis of the channel's time and space constraints. Then, *PollSan* conducts data sanitization on all invalid data pages in the selected block (s). When all scheduled block sanitizations are completed, *PollSan* will turn to another polling cycle and sanitize sensitive data pages on another channel.

[2]A large sanitization priority indicates a small number of future *adjacent page* invalidations.

In order to determine the number of sanitization blocks in a polling cycle of the given SSD channel, we propose a method to decide the number of blocks that can be scrubbed in the current polling cycle, on the basis of the current I/O workloads and the free space of SSD device. Specifically, the proposed method obeys the following time constraints and space constraints.

3.3.1 Time Constraints. Sanitization may result in certain migration operations, thus causing a rise in the I/O workload. That is to say, choosing more blocks for sanitization on a specific channel will bring about more impacts on normal I/O performance. To characterize the influence level of data sanitization on I/O response time caused by our method, we propose an indicator of T to express the latency time caused by completing data sanitization and the enqueued I/O requests. Assuming that the average cumulative amount of enqueued I/O requests in the case of *ISan* is V, the incoming I/O workloads during the sanitization process per second is L_{in}, the additional I/O loads per second caused by *ISan* is L_{ISan}, and the I/O processing throughput of the SSD device is L_{out}. We believe that the amount of I/O data processed within time T should be equal to the cumulative amount of enqueued I/O requests V plus the generated I/Os during sanitization. Then, Equation 1 holds.

$$V + (L_{\text{in}} + L_{\text{ISan}}) \cdot T = L_{\text{out}} \cdot T \tag{1}$$

Consequently, the time required to complete data sanitization and I/O requests (i.e., the time latency of T) in the proposal of *ISan* can be derived with Equation 2.

$$T = \frac{V}{L_{\text{out}} - (L_{\text{in}} + L_{\text{ISan}})} \tag{2}$$

In the scenario of *PollSan*, the current accumulated amount of enqueued I/O requests in the channel can be represented as V_{PollS}, that the incoming I/O workloads arrive during the sanitization process per second is denoted as L_{in_PollS}. The I/O workload per second of N blocks caused by data sanitization in the channel can be defined as $L_0, L_1 ... L_{N-1}$ respectively. Then, the total I/O workload generated by selecting k block sanitizations per second can be defined as $L_{PollS} = \sum_{i=0}^{k-1} L_i$. Note that the negative impacts of data sanitization caused by *PollSan* should not exceed that caused by *ISan*. In other words, during the same time period of T, the overall I/O workloads with the *PollSan* mechanism should not exceed that with the *ISan* mechanism, ensuring the validity of the following equation:

$$(L_{in\_PollS} + L_{PollS}) \times T + V_{\text{now}} \leq (L_{\text{in}} + L_{\text{ISan}}) \times T + V \tag{3}$$

Subsequently, we can expend Equation 3 by using Equation 2, to obtain the following equation:

$$(L_{in\_PollS} + \sum_{i=0}^{k-1} L_i) \times \frac{V}{L_{\text{out}} - (L_{\text{in}} + L_{\text{ISan}})} + V_{\text{now}} \leq (L_{\text{in}} + L_{\text{ISan}}) \times \frac{V}{L_{\text{out}} - (L_{\text{in}} + L_{\text{ISan}})} + V \tag{4}$$

Finally, we can derive the suitable number of sanitization blocks in the case of using our approach of *PollSan* at the given polling time slot, by selecting the maximum value of k, according to the constraint shown in Equation 5.

$$\sum_{i=0}^{k-1} L_i \leq L_{\text{out}} - \frac{V_{\text{now}}}{V} (L_{\text{out}} - L_{\text{in}} - L_{\text{ISan}}) - L_{\text{in\_PollS}} \quad (\textit{subject to } k < N) \tag{5}$$

3.3.2 Space Constraints. In the process of data sanitization, the valid data pages that are adjacent to the sanitization-required pages must be migrated to other blocks in the same channel. That is, completing data sanitization on more blocks in a specific SSD channel will decrease the available space, thus probably triggering garbage collection (GC) to reclaim space, which will greatly impact I/O responsiveness and lead to an inefficient sanitization process. Hence, we emphasize that it is not recommended to migrate the valid data pages on the sanitization block (s) to a target block

that does not have enough free space. Specifically, the number of page migrations issued by data sanitization on SSD blocks should be limited to avoid triggering an immediate GC operation, by considering the factor of available space on the SSD channel.

Suppose that we have a total of M blocks and that N blocks are expected to be sanitized (termed as *source blocks*) on the given channel, indicating that we have $M - N$ blocks (termed as *destination blocks*) for holding the migrated data pages introduced by data sanitization. We assume that the available space of *destination blocks* in the channel is S, and that the minimum remaining space to trigger GC is S^*. The space of valid move pages of N *source blocks* is S_0, $S_1 ... S_{N-1}$. Subsequently, the additional space required for data sanitization should be less than the currently available space of the given channel. Then, the space constraint defined in Equation 6 must be satisfied, when determining the number of sanitization blocks k in the polling cycle.

$$\sum_{i=0}^{k-1} S_i \leq S - S^* \quad (subject\ to\ \ k < N) \tag{6}$$

In summary, we can unify both the time constraint (Equation 5) and the space constraint (Equation 6), thus setting the smaller value of k as the number of blocks that are supposed to be selected for sanitization, in the current polling cycle.

4 EXPERIMENTS AND DISCUSSIONS

4.1 Experimental Settings

We performed experimental evaluation using the widely studied SSD simulator SSDSim [42], which has been modified to support sanitization. We use a local ARM-based machine as SSD controllers which usually have limited computation power and memory capacity [43]. The machine has an ARM Cortex A7 Dual-Core with 800MHz, 128MB of memory and 32-bit Linux (*ver 3.1*). We simulated a 24GB SSD device and set the performance parameters based on the values used in a recent study [15], and Table 2 presents the specifications of the simulator. To reflect the impacts of GC and wear leveling before replaying traces, the simulated SSD is aged so that valid data and invalid data occupy 62.64% and 7.36% of its capacity, respectively [6], and the program/erase cycle of blocks is initialized with warm-up by exactly referring to [44].

Apart from the newly proposed *PollSan* scheme, we employed the following four sanitization schemes as comparison counterparts in evaluation tests:

- ISan [8], which requires sanitization on invalid pages immediately once the updated operation happens for ensuring the security of data.
- PSan [17], which performs sanitization in a cyclical manner, where the period is determined by the data owner on the basis of data sensitivity. In our scenario, we have tested the time window of sanitization in the range of 1024 and 8192 (**Appendix A**), and selected 4096 requests as the default size of the time window in the evaluation section.
- SAS [14], which is a sophisticated scrubbing-aware secure deletion design. *SAS* aims to minimize the overhead of page migrations by adaptively choosing a proper operation (scrubbing or erase) to delete the selected sensitive data, according to the estimated overhead. Furthermore, *SAS* creates scrubbing-friendly patterns for reducing the overhead of live-page-copying in secure deletion procedures. Specifically, *SAS* conducts sanitization operations according to a *Poisson* distribution, and places the original data pages together with their updated copies.
- IDS [26], which is an advanced instant data sanitization method for NAND-based flash memory, and it can prevent data leakage without any negative effects on valid data in shared pages. Through an analysis of the encoding rules in high-density SSDs, *IDS* proposes

Table 2. Experimental settings of *SSDsim*

Chip parameters	
(*Die, Plane, Block, Page)*	(1, 1, 512, 384)
(Page size, Cell density)	(8KB, TLC)
(Scrubbing latency)	(0.7ms)
(Program latency, Read latency)	(0.7ms,0.075ms)
(Erase latency)	(10ms)
SSD parameters	
(Channel, Chip)	(8,16)
(FTL, GC trigger)	(Page-level, 30%)

Table 3. Specifications on traces

Traces	**Req #**	**Wr R**	**Wr SZ**	**Avg. INT**	**San Page**
ali_0	311756	70.8%	7.2KB	69.2ms	308587
ali_1	395658	73.0%	8.8KB	54.5ms	463262
ali_2	444494	66.6%	11.6KB	48.5ms	581864
ali_3	289746	79.9%	8.6KB	74.5ms	363173
lun0	534529	66.6%	9.3KB	6.7ms	711368
lun1	605819	47.8%	12.2KB	5.9ms	659617
mds_0	1211034	88.1%	7.2KB	4.9ms	1552880
wdev_0	1143261	79.9%	8.2KB	5.3ms	1391426

sanitizing the invalid page by storing a mirrored copy of the adjacent page in some special cases, to reduce the migration operations on flash memory introduced by data sanitization. Specifically, it works for some scenarios of sanitizing the MSB page in TLC SSDs. In which, *IDS* first identifies whether its adjacent CSB page is valid or not. If yes, it reads the valid CSB page data into the cache and inverses the data. Then, it sanitizes the invalid MSB page by flushing the inversed data onto it. Otherwise, *IDS* works like *ISan*.

As discussed in Section 3.2, we trained the prediction ANN model by inputting the collected entries after replaying the traces in the MSRC collection and the VDI collection. In order to verify the effectiveness of the trained model in various application scenarios, we tested the practicability of our model on both internal validation dataset and the external validation dataset. In the internal validation dataset, two traces are from the MSRC collection and another two traces are from the VDI collection [39]. Specifically, two VDI traces are additional-01-2016021615-LUN1 (labeled as *lun0*), and additional-01-2016021618-LUN1 (*lun1*). To construct the external validation dataset, we chose four recent traces from *Alibaba Cloud* [45], corresponding to four six-hour trace segments of a 20GB virtual disk (close to the capacity of our emulated 24GB SSD device), labeled as *ali_0* to *ali_3*. Consequently, there are a total of eight disk traces of real-world applications in our evaluation tests.

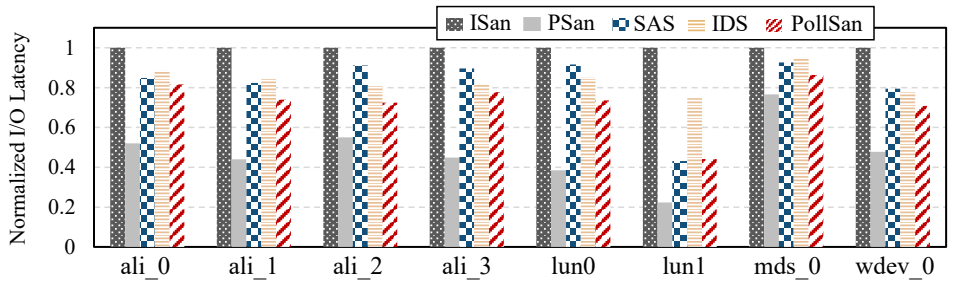


Fig. 6. Normalized average I/O latency achieved by varied sanitization policies.

Table 3 shows the details about the selected traces of internal and external validation sets. In the table, the metric of *Req#* represents the number of requests, the measures of *Wr R* and *Wr SZ* mean the ratio of write requests and the average size of write requests in the workload. The metric of *Avg. INT* implies the average interval time between two requests, indicating the level of I/O intensity in the workload, and *San Page* means the number of data pages requiring sanitization.

4.2 Performance Comparisons and Discussions

To measure the effectiveness of our proposal, we make use of the following metrics in our experiments: ***(a)*** *Average I/O latency*, ***(b)*** *Sanitization time*, ***(c)*** *Average unsafe time interval*, and ***(d)*** *Block erase statistics*.

4.2.1 Average I/O Latency. Reducing the I/O latency to guarantee I/O responsiveness is the primary aim of the proposed sanitization scheduling method, and Figure 6 shows the results of normalized average I/O latency after running the selected traces. As expected, *PSan* outperforms others on the measure of I/O latency. More exactly, it can noticeably reduce the I/O latency by `52.36%`, `42.30%`, `43.59%`, and `35.52%`, in contrast to *ISan*, *SAS*, *IDS*, and our proposal of *PollSan*, respectively. This is because *PSan* conducts sanitization tasks periodically, which can reduce the migration overhead of valid pages and minimize negative effects on I/O responsiveness (see Section 4.2.2). We emphasize that the routine of periodical sanitization in *PSan* greatly threatens the security of sensitive data, even if it has the least impact on I/O performance. Besides, we see that *SAS* can decrease the I/O response time by `18.11%` on average, compared with *ISan*, since it can significantly reduce the number of page migrations, by purposely keeping the same copies of data together. We can observe that *IDS* reduces I/O latency by `16.51%`, in contrast to *ISan*, though both of them are instant sanitization schemes. This is because it fulfills page data sanitization by storing mirrored copies of *adjacent pages* in some cases, which can eliminate the needs of migrating *adjacent pages* to other free pages, leading to a reduction of GC operations.

The most important clue shown in Figure 6 is that, our proposal of *PollSan* results in less I/O latency by `27.38%`, `10.35%` and `13.42%` on average, in contrast to *ISan*, *SAS* and *IDS*. This fact confirms that polling-based sanitization can contribute to a reduction of I/O latency, ensuring a better I/O responsiveness, compared to related work. This is because *PollSan* can select as many sanitization blocks as possible that may not have *adjacent page* invalidations in the future, so that blocks with more *adjacent page* invalidations can be scrubbed later. In addition, our proposal supports data sanitization on multiple blocks in a polling cycle, if the target SSD channel has slight I/O workloads, which can adaptively coordinate sanitization effects and I/O performance.

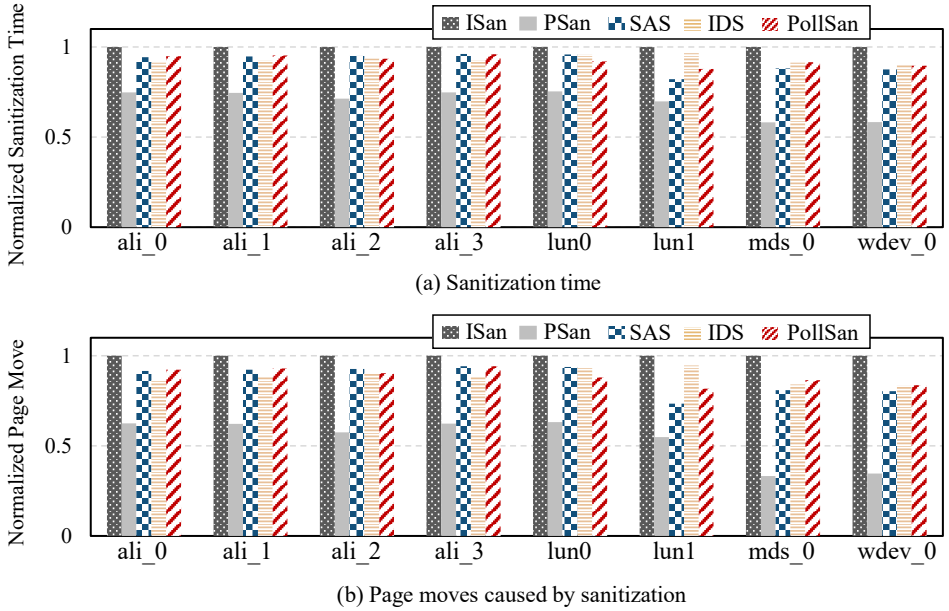


Fig. 7. Comparison of normalized sanitization overhead.

4.2.2 Sanitization Time. Each sanitization process first moves valid pages of the wordline to another free block and then scrubs the wordline. Figures 7 (a) and 7 (b) present the sanitization time and the number of *page moves* caused by sanitization, after replaying the selected traces. As seen, *PSan* induces a smaller sanitization time, because it performs sanitization tasks in a periodical manner, helping to accumulate more *adjacent page* invalidations, and thus results in a smaller umber of *page moves*.

More importantly, our *PollSan* approach results in less sanitization time by 7.45% on average, corresponding to a reduction of 11.32% on *page moves*, in contrast to *ISan*. This fact confirms that our proposal can efficiently decrease the number of *page moves*, thus decreasing the sanitization overhead and extending the lifetime of SSDs. We argue that the proposed model adopted by *PollSan* can predict the number of future *adjacent page* invalidations and choose the appropriate time for sanitization, which can contribute to a better compromise on I/O performance and data security. Note that *PollSan* requires more sanitization time after running some selected traces, comparing to the related work of *SAS* and *IDS*, since *PollSan* introduces more *page moves* onto the free pages of other wordlines in sanitization processes.

4.2.3 Average Unsafe Time Interval. The unsafe time interval (UTI) is a critical factor to measure data security. Ensuring low UTI values of sensitive data pages is another goal of our proposed scheme. Figure 8 shows the comparison of the average UTI of all data pages after replaying the selected traces. As seen, *ISan* and *IDS* work the best on ensuring data security, at the cost of I/O performance degradation. This is because they performs immediate sanitization once the data pages have been invalidated, regardless of the impacts on I/O responsiveness. Our approach of *PollSan* greatly outperforms another comparison scheme of *Psan* by 93.73% on average, in terms of UTI. This is because *PollSan* carries out polling-based sanitization among all SSD channels, which

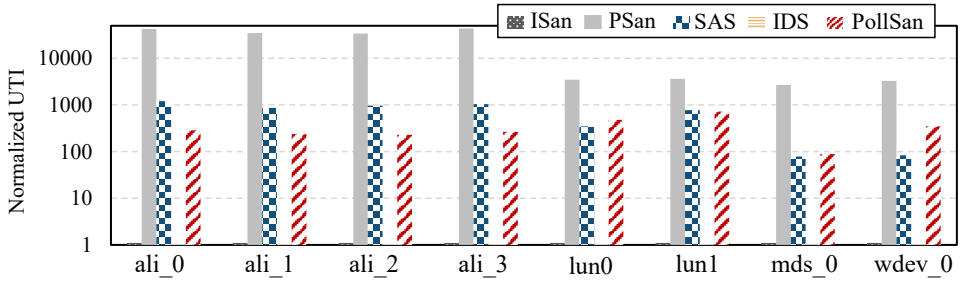


Fig. 8. Normalized average UTI achieved by varied sanitization policies.

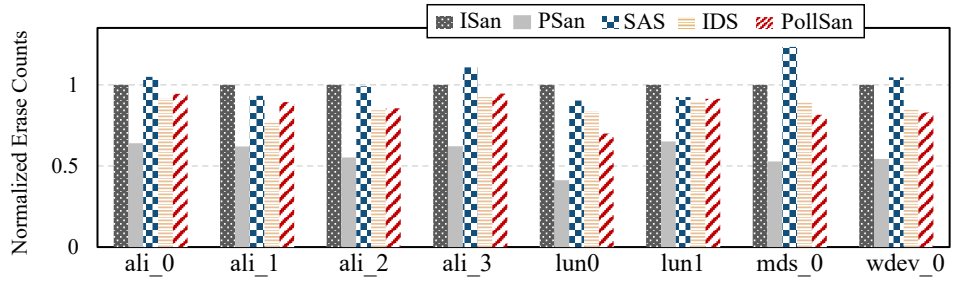


Fig. 9. Normalized block erase counts with varied sanitization policies.

contributes to the quick sanitization on invalid data pages of the polled channel, meanwhile other channels can still keep services for responding to normal I/O requests. In addition, the *SAS* scheme can reduce the UTI by 94.06%, in contrast to *PSan*. This is because SAS carries out sanitization tasks by following the *Poisson* distribution. Specifically, the execution period is divided into three parts: the early stage, the middle stage, and the late stage. *SAS* performs relatively sparse sanitization operations in the early and late stages of benchmark execution, it will perform intensive sanitization tasks in the middle stage, so that a major part of the sanitization operations can be processed quickly.

Another interesting clue shown in Figure 8 is that *PSan* performs significantly worse in the cases of running the selected *ali* traces. We suggest that the sanitization cycle of *PSan* is based on the fixed time window of 4096 requests, and that the average request interval of the *ali* traces is relatively larger, which results in longer sanitization cycles. In addition, we see that our mechanism noticeably outperforms the related work of *SAS* while running the *ali* traces, but performs a little bit worse than *SAS* in the cases of running other traces, on the measure of UTI. This is because, the *ali* traces have relatively large time intervals between I/O requests, and *PollSan* can perform more sanitization operations in every polling cycle, according to the time constraints. On the other hand, the remaining traces have small intervals between I/O requests, which will limit the number of sanitizations in each polling cycle, resulting in the accumulation of block sanitizations, as well as an increase in UTI while using our approach of *PollSan*. But note that our approach outperforms the related work of *SAS*, in terms of I/O latency while running intensive I/O workloads, as previously illustrated in Section 4.2.1.

4.2.4 Block Erase Statistics. The metric of erase counts is used to reflect the endurance of the SSD, so that we record the number of erase operations after running the traces, by using five sanitization schemes. Figure 9 shows the relevant results. As seen, *PSan* and *PollSan* can reduce the number of erase counts by 42.88% and 13.80%, in contrast to *ISan*. This is because they can accumulate invalid pages and sanitize them together, thus reducing the number of erase operations. Another interest information is that the *IDS* method brings about a reduction of 13.53% erase operations comparing to *ISan*. We suggest this is because *IDS* employs the data cache to make mirror copies for ending sanitization on MSB pages if their adjacent CSB pages are valid, which can reduce the number of migrating *adjacent pages* to other free pages, leading to a smaller quantity of erase operations.

Besides, we see the *SAS* method increases the number of erase operations by 19.46% on average, compared with our proposal of *PollSan*. This is because *SAS* will reserve free pages for storing various versions of the same data and delete invalid pages by erase operation, leading to frequent triggering of the garbage collection and an increase in block erasures.

4.3 Analysis of United Impact of I/O Latency and Data Security

To measure the united impact level of I/O performance and data security related to sanitization schemes, we define a time metric of risk-performance index (RPI), on the top of the **unitized** value of UTI and the I/O latency, by referring to the **cost function** [46]. In fact, the **cost function** is widely used for estimating the total cost of production factors in the field of economics. Considering *ISan* can yield the best I/O latency, whereas *PSan* can achieve the best UTI, both of them have a normalized RPI value of 1.0, indicating they have the same total cost (i.e., the united impact) caused by data sanitization. Figure 10 illustrates that *ISan* and *PSan* do have the same value of RPI, and the *isocost line* represents the RPI value as constant for all I/O and UTI combinations. In other words, the RPI values in the lower right quadrant indicate better performance of united I/O latency and data security after data sanitization, in contrast to the baselines of *ISan* and *PSan*.

Equation 7 defines the specifications of obtaining the RPI value, so that we can yield the unitized UTI weights of all traces, and compute the values of RPI for them. We emphasize that a lower PRI value represents a smaller data sanitization impact introduced by data sanitization.

$$RPI = I/O\ Latency + k \times UTI \tag{7}$$

where k is the unitized weight with respect to the given trace and Equation 8 defines the way of obtaining it.

$$k = \frac{T_{\mathrm{ISan}} - T_{\mathrm{PSan}}}{UTI_{\mathrm{PSan}} - UTI_{\mathrm{ISan}}} \tag{8}$$

where T_{ISan} and T_{PSan} represent the I/O time with *ISan* and *Psan* respectively, UTI_{ISan} and UTI_{PSan} mean the UTI values with *ISan* and *Psan*, respectively.

Figure 11 presents the results of RPI using five sanitization schemes. As seen, our method greatly outperforms the other two basic comparison counterparts (i.e., *ISan* and *PSan*), by more than 23.45% on average. This is because our polling sanitization scheme can provide I/O services on the SSD channels that are not currently enduring sanitization. Furthermore, the prediction model helps to minimize the number of *page moves* caused by polling-based sanitization, which can help reduce the sanitization time. Besides, our proposal of *PollSan* can result in a better RPI by 10.04% and 8.42% on average, in contrast to *SAS* and *IDS*. It confirms that our polling-based method of *PollSan* can decrease the impact caused by sanitization on the integrated metrics composed of I/O responsiveness and data security.

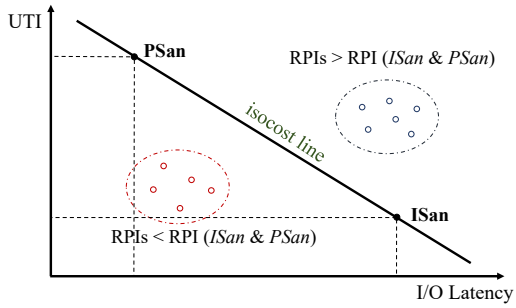


Fig. 10. The illustration of RPI that is defined on the basis of the cost function, where the RPI values for *ISan* and *PSan* are same and located in the same *isocost line*. Note that an *isocost line* shows the different combinations of I/O latency and UTI, having the same value of RPI.

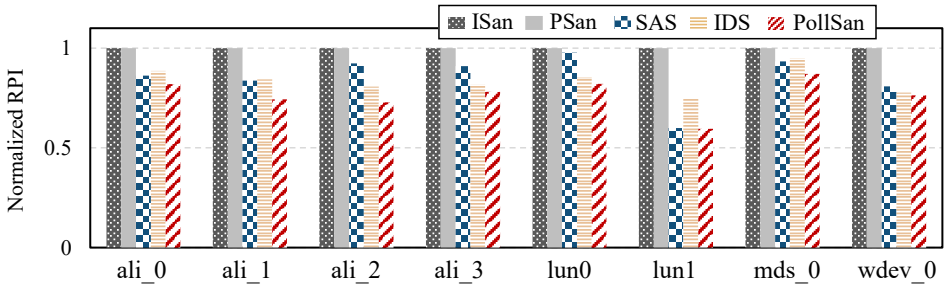


Fig. 11. The normalized RPIs achieved by varied sanitization policies, and the lower the better. Note that both *ISan* and *PSan* maintain the normalized RPI value of `1.0` in all traces.

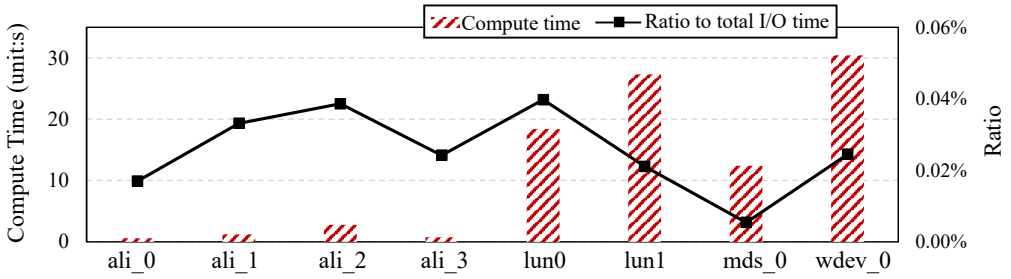


Fig. 12. Overall compute time of *PollSan* after replaying the selected traces. Note that the compute time consists of the retrieving time on the index table and the computing time on deciding the quantity of block sanitizations in each polling cycle.

4.4 Overhead Analysis and Modeling Accuracy

The main memory overhead of our proposal is due to the additional storage required for the parameters used by the prediction model, so that *PollSan* results in memory overhead of about 32KB. More specifically, we need to record the feature information of blocks for predicting their sanitization priorities, so that the spatial overhead is 4byte/block × 1024block/channel × 8channel

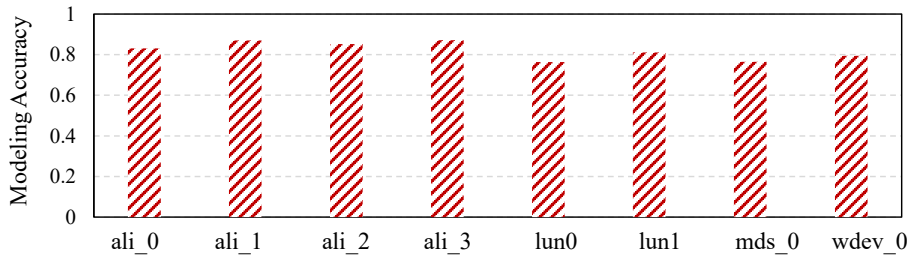


Fig. 13. Modeling accuracy of *PollSan* after replaying the selected traces.

= 32KB, which we believe is an acceptable amount of memory space in the case of a 24GB SSD. With respect to time overhead, our method requires dynamically selecting the sanitization blocks, as well as predicting the values of future *adjacent page* invalidations on blocks by resorting to the index table. We measured the overall compute time of *PollSan*, and Figure 12 shows the results. The time cost is between `0.58` and `30.42` seconds, or less than `0.04%` of the overall I/O time. Thus, we consider that the compute time caused by *PollSan* remains negligible, even though our experiments are completed on the resource-limited platform. Note that the computation overhead affects the I/O latency by postponing dispatches on incoming I/O requests, and relevant I/O time results have been previously reported in Section 4.2.1.

Furthermore, we have recorded the number of prediction hits on the block having fewer *adjacent page* invalidations to show the accuracy of our prediction model. We define a prediction hit as the case of the number of *adjacent page* invalidations is not larger than the average number of *adjacent page* invalidations of all blocks. Figure 13 presents the results of the prediction accuracy, and it shows our method can yield an accuracy of `81.94%` on average with default settings. We suggest that *PollSan* achieves good prediction accuracy, compared to the accuracy of `50.0%` in random predictions.

4.5 Case study on Varied Size of Index Table

PollSan employs a pre-trained index table for mapping the combinations of block features to the sanitization priorities of SSD blocks. The size of the index table output by the pre-trained ANN model may impact prediction accuracy, which in turn affects the performance of the storage system. This section checks the effectiveness of our approach if the SSD has different size of index tables. We suggest that the varied size of index tables require different memory space and retrieval overhead. Therefore, we carry out a case study to verify the prediction accuracy and the lookup time overhead for different index table sizes.

Figures 14 (a) and 14 (b) show the results of the modeling accuracy and compute time, while using our approach of *PollSan* with varied sizes of configured index tables. In the tests, we specifically checked the size of index table with `400`, `4,000`, and `40,000`, labelling as *PollSan-400*, *PollSan-4000*, and *PollSan-40000*. We observe that as the size of the index table becomes larger, there is no noticeable increase in the measure of prediction accuracy after running the benchmarks, but more retrieval time is required. In the case of running the benchmark of *wdev_0*, for example, *PollSan-4000* and *PollSan-40000* have, respectively, `12` and `207` times the retrieval cost of the baseline configuration of *PollSan-400*.

Besides, we emphasize that a larger index table requires more memory space overhead. For example, the configuration of `40,000` entries needs `195` KB of memory for the index table, indicating

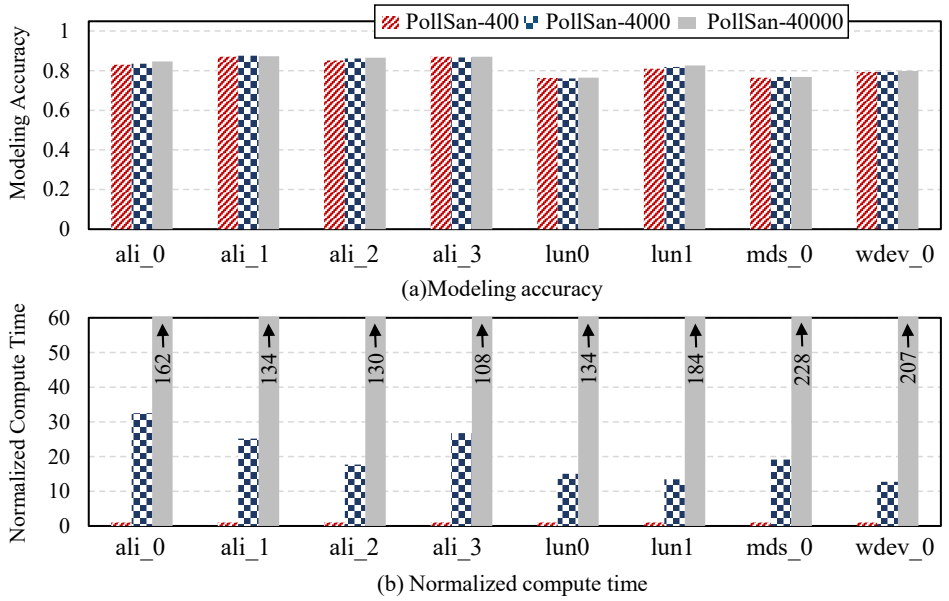


Fig. 14. The comparison of modeling accuracy and compute time with varied size of index table in *PollSan*. The results of compute time are normalized to *PollSan* with the index table size of `400`.

`100` times of space overhead in contrast to the configuration of `400` entries. As a consequence, we decided to use an index table of `400` entries by default, by considering both factors of the prediction accuracy and the time/space overhead.

5 RELATED WORK

This section further describes advanced schemes on data sanitization for SSDs. Specifically, data scrubbing was first proposed by Wei et al [8], which creates an all-zero page with the same page program command. However, this technique is not easy to perform in high-density SSDs (in which, each wordline has multiple pages), as it incurs significant performance overhead for moving out valid pages from the scrubbed wordline. With respect to this problem, Lin et al [25] employed one-shot reprogramming to achieve instant and zero-copy overhead sanitization, which can minimize the disturbance to adjacent valid pages. Wang et al [14] formulated the overhead of secure deletion, to direct the selection of sanitization tasks, as well as created the scrubbing-friendly pattern to reduce the overhead of moving valid pages. Kim et al [47] employed two lock commands to disable accessing the sanitization-required data pages. Hasan and Ray [10] proposed a new analog scrubbing scheme to guarantee that the sensitive data remains unrecoverable after sanitization. Cui et al [15] designed *ADS*, to avoid moving valid pages of approximate data in the process of sanitization, as such data have great error resilience. Consequently, the sanitization overhead in high-density SSDs can be greatly reduced. Moreover, Raquibuzzaman et al [26] presented a novel instant page data sanitization by creating mirror copies of adjacent valid pages and flushing them onto the sanitization pages to fulfil page sanitization. Because it employs the data cache to make mirror copies and directly flushes the copies to the sanitization pages, it can reduce the number of migrating *adjacent pages* onto other free wordlines of flash memory, which can reduce the GC overhead caused by data sanitization in some cases.

We summarize that *PollSan* is the first mechanism that addresses the issue of balancing I/O performance and data security during sanitization in the SSD context. Specifically, the existing sophisticated methods either work for specific scenarios (e.g., approximate data [15]) or expect hardware supports (e.g., the lock functionality [47]). Unlike existing sanitization approaches, *PollSan* does not require application contexts with special data characteristics or hardware supports, which can make guarantee the I/O performance and data security in a wide range of applications.

6 CONCLUSION

This paper has proposed the *PollSan* sanitization scheduling method, which intends to minimze the sanitization impacts on I/O latency and data security of high-density SSDs. To this end, *PollSan* requires each polling cycle to fulfill sanitization on a specific channel, for completing data sanitization on the selected block (s). In the polling cycle on the specific channel, our proposal places low priorities on the blocks that will have more *adjacent page* invalidations in the near future, to minimize the negative impacts of moving valid pages in data sanitization. Moreover, *PollSan* considers both time and space constraint on the SSD channel, for deciding the preferred number of sanitization blocks in the polling cycle of the given channel.

The experimental results show that our approach can efficiently reduce *RPI* by up to `40.40%`, compared to the existing methods, implying that it can better mitigate the negative impacts on I/O responsiveness and data security regarding data sanitization. We suggest that *Pollsan* does not require any hardware modifications, and it can be implemented using standard user-mode flash commands in the FTL firmware.

ACKNOWLEDGMENTS

The authors would like to thank anonymous reviewers for their valuable comments. This work was partially supported by “National Natural Science Foundation of China (No. 61872299), “Natural Science Foundation Project of CQ CSTC (No. cstc2021ycjh-bgzxm0199, 2022NSCQ-MSX0789)". The dataset and codes used for the ANN model development and training are available at: https://github.com/912ssdlab/ANN-model-for-PollSan.

REFERENCES

[1] Mathur G, Desnoyers P, Ganesan D, et al. Capsule: an energy-optimized object storage system for memory-constrained sensor devices. *In Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (**SENSYS**)*. 2006: 195-208.

[2] Lin S, Zeinalipour-Yazti D, Kalogeraki V, et al. Efficient indexing data structures for flash-based sensor devices.*ACM Transactions on Storage (**TOS**)*, 2006, 2 (4): 468-503.

[3] Kang W, Shin D, Yoo S. Reinforcement learning-assisted garbage collection to mitigate long-tail latency in SSD. *ACM Transactions on Embedded Computing Systems (**TECS**)*, 2017, 16 (5s): 1-20.

[4] Kang M, Lee W, Kim J, et al. PR-SSD: Maximizing Partial Read Potential By Exploiting Compression and Channel-Level Parallelism. *IEEE Transactions on Computers (**TC**)*, 2022.

[5] Kang J, Jo H, and Kim J et al. A superblock-based flash translation layer for NAND flash memory. *In Proceedings of ACM International Conference on Embedded Software (**EMSOFT**)*, 2006.

[6] Zhang W, Cao Q, Jiang H, et al. PA-SSD: A page-type aware TLC SSD for improved write/read performance and storage efficiency. *In Proceedings of the 2018 International Conference on Supercomputing (**ICS**)*, 2018: 22-32.

[7] Higuchi T, Kodama T, Kato K, et al. 30.4 a 1Tb 3b/cell 3D-flash memory in a 170+ word-line-layer technology. *In 2021 IEEE International Solid-State Circuits Conference (**ISSCC**)*, 2021, 64: 428-430.

[8] Wei M Y C, Grupp L M, Spada F E, et al. Reliably Erasing Data from Flash-Based Solid State Drives. *In Conference on File and Storage Technologies (**FAST**)*, 2011, 11: 8-8.

[9] Data Protection Act 2018. https://www.legislation.gov.uk/ukpga/contents, 2018.

[10] Hasan M, Ray B. Data recovery from “scrubbed” NAND flash storage: need for analog sanitization. *In Proceedings of 29th USENIX Security Symposium (**USENIX Security**)*, 2020.

[11] Fasano A, Ballo T, and Muench M et al. Sok: Enabling security analyses of embedded systems via rehosting. *In Proceedings of ACM Asia conference on computer and communications security **(ASIA CCS)***, 2021, 687-701.

[12] Mergendahl S, Jero S, Ward B C, et al. The thundering herd: Amplifying kernel interference to attack response times. *In Proceedings of the 2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium **(RTAS)***, 2022: 95-107.

[13] Parameswaran S, and Wolf T. Embedded systems security—an overview. *Design Automation for Embedded Systems*, 2008, 173-183.

[14] Wang W C, Ho C C, Chang Y H, et al. Scrubbing-aware secure deletion for 3-D NAND flash. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **(TCAD)***, 2018, 37 (11): 2790-2801.

[15] Cui J, Liu W, Huang J, et al. ADS: Leveraging approximate data for efficient data sanitization in SSDs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **(TCAD)***, 2021, 41 (6): 1771-1784.

[16] Reardon J, Basin D, Capkun S. Sok: Secure data deletion. *In 2013 IEEE symposium on security and privacy **(IEEE S&P)***, 2013: 301-315.

[17] Reardon J, Capkun S, Basin D A. Data node encrypted file system: Efficient secure deletion for flash memory. *In USENIX Security Symposium **(USENIX Security)***, 2012: 333-348.

[18] Kissel R, Regenscheid A, Scholl M, et al. Guidelines for media sanitization. *US Department of Commerce, National Institute of Standards and Technology*, 2014.

[19] Dwell Time as a Critical Security Success Metric. https://cdn.armor.com/app/uploads/2020/04/Ebook-DwellTime.pdf, Armor White Paper, 2020.

[20] Wu F, Zhou J, Wang S, et al. FastGC: Accelerate garbage collection via an efficient copyback-based data migration in SSDs. In Proceedings of *the 55th Annual Design Automation Conference **(DAC)***, 2018: 1-6.

[21] Lee J, Ganesh K, Lee H J, et al. FESSD: A fast encrypted ssd employing on-chip access-control memory. *IEEE Computer Architecture Letters **(CAL)***, 2017, 16 (2): 115-118.

[22] Jia S, Xia L, Chen B, et al. Deftl: Implementing plausibly deniable encryption in flash translation layer. *In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security **(ACM SAC)***, 2017: 2217-2229.

[23] Reardon J, Marforio C, Capkun S, et al. User-level secure deletion on log-structured file systems. *In Proceedings of the 7th ACM symposium on information, computer and communications security **(ASIACCS)***, 2012: 63-64.

[24] Diesburg S, Meyers C, Stanovich M, et al. Trueerase: Per-file secure deletion for the storage data path. *Proceedings of the 28th annual computer security applications conference **(ACSAC)***, 2012: 439-448.

[25] Lin P H, Chang Y M, Li Y C, et al. Achieving fast sanitization with zero live data copy for MLC flash memory. *In 2018 IEEE/ACM International Conference on Computer-Aided Design **(ICCAD)***, 2018: 1-8.

[26] Raquibuzzaman M, Buddhanoy M, Milenkovic A, et al. Instant data sanitization on multi-level-cell NAND flash memory. In *Proceedings of the 15th ACM International Conference on Systems and Storage **(SYSTOR)***. 2022: 85-95.

[27] Ramsay C, Lohuis J. TEMPEST attacks against AES. Covertly stealing keys for 200 euro. *In Proceedings of the 2017 Hardware Security Conference Training*, 2017: 1-10.

[28] Rumelhart D E, Hinton G E, Williams R J. Learning representations by back-propagating errors. *nature*, 1986, 323 (6088): 533-536.

[29] Basheer I A, Hajmeer M. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, 2000, 43 (1): 3-31.

[30] Liu R, Chen X, Tan Y, et al. SSDKeeper: Self-adapting channel allocation to improve the performance of SSD devices. *In 2020 IEEE International Parallel and Distributed Processing Symposium **(IPDPS)***, 2020: 966-975.

[31] Mizushina K, Nakamura T, Deguchi Y, et al. Layer-by-layer adaptively optimized ECC of NAND flash-based SSD storing convolutional neural network weight for scene recognition. *In 2018 IEEE International Symposium on Circuits and Systems **(ISCAS)***. IEEE, 2018: 1-5.

[32] Liu W, Cui J, Li T, et al. A Space-Efficient Fair Cache Scheme Based on Machine Learning for NVMe SSDs. *In IEEE Transactions on Parallel and Distributed Systems **(TPDS)***, 2022, 34 (1): 383-399.

[33] Voigt, Paul, and Axel Von dem Bussche. The EU general data protection regulation (gdpr). A Practical Guide, 1st Ed., Cham: Springer International Publishing 10.3152676 (2017): 10-5555.

[34] Act, Accountability. Health insurance portability and accountability act of 1996. Public law 104 (1996): 191.

[35] Chen N and Bo C. Duplicates also Matter! Towards Secure Deletion on Flash-based Storage Media by Removing Duplicates.In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security **(ASIACCS)***. 2022.

[36] Li B, and David Du. WAS-Deletion: Workload-Aware Secure Deletion Scheme for Solid-State Drives. *In 2021 IEEE 39th International Conference on Computer Design **(ICCD)***. IEEE, 2021.

[37] Wang S, Zhou Y, Zhou J, et al. An efficient data migration scheme to optimize garbage collection in SSDs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **(TCAD)***, 2020, 40 (3): 430-443.

[38] Narayanan D, Thereska E, Donnelly A, et al. Migrating server storage to SSDs: analysis of tradeoffs. *In Proceedings of the 4th ACM European conference on Computer systems* ***(Eurosys)***, 2009: 145-158.
[39] Lee C, Kumano T, Matsuki T, et al. Understanding storage traffic characteristics on enterprise virtual desktop infrastructure. *In Proceedings of the 10th ACM International Systems and Storage Conference* ***(SYSTOR)***, 2017: 1-11.
[40] Jain A K, Mao J, Mohiuddin K M. Artificial neural networks: A tutorial. *Computer*, 1996, 29 (3): 31-44.
[41] Abiodun O I, Jantan A, Omolara A E, et al. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 2018, 4 (11): e00938.
[42] Hu Y, Jiang H, Feng D, et al. Exploring and exploiting the multilevel parallelism inside SSDs for improved performance and endurance. *IEEE Transactions on Computers* ***(TC)***, 2012, 62 (6): 1141-1155.
[43] Li J, Sha Z, Cai Z, et al. Patch-based data management for dual-copy buffers in RAID-enabled SSDs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* ***(TCAD)***, 2020, 39 (11): 3956-3967.
[44] Liao J, Li J, Zhao M, et al. Read Refresh Scheduling and Data Reallocation against Read Disturb in SSDs. *ACM Transactions on Embedded Computing Systems* ***(TECS)***, 2022, 21 (2): 1-27.
[45] Alibaba Block Traces. https://github.com/alibaba/block-traces.
[46] Varian H R. Intermediate microeconomics (8th Edition): a modern approach. WW Norton & Company, 2010.
[47] Kim M, Park J, Cho G, et al. Evanesco: Architectural support for efficient data sanitization in modern flash-based storage systems. *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* ***(ASPLOS)***, 2020: 1311-1326.

A APPENDIX: PSAN PERFORMANCE WITH VARIED SIZE OF TIME WINDOW

The sanitization interval of *PSan* is determined by the data owners on the basis of their data sensitivity. We have tested the performance of *PSan* while the time window varies from 1024 requests to 8192 requests. Figure 15 presents the results of major performance indicators, showing a larger interval may result in better I/O performance and SSD lifetime, but with the cost of worse data security.

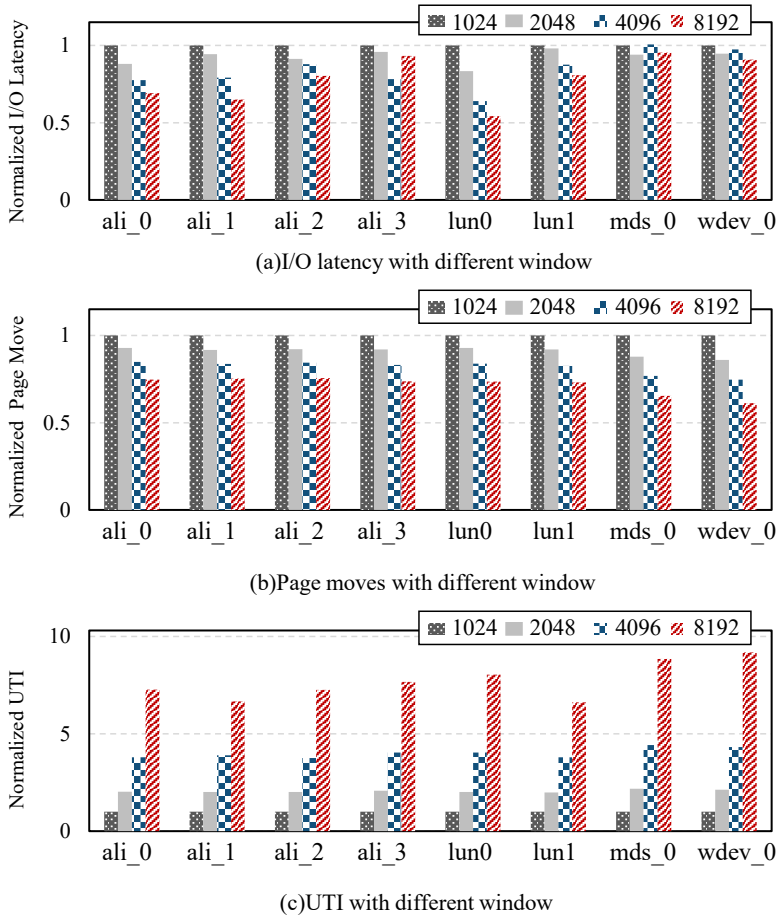


Fig. 15. Normalized *PSan* performance with varied sanitization time window.