This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2349517, IEEE Transactions on Computers

Effective Lifetime-Aware Dynamic Throttling for NAND Flash-Based SSDs

Sungjin Lee and Jihong Kim *Member, IEEE*

Abstract—NAND flash-based solid-state drives (SSDs) are increasingly popular in enterprise server systems because of their advantages over hard disk drives such as higher performance and lower power consumption. However, the decreasing write endurance and the unpredictable lifetime remains to be a serious obstacle to their wider adoption in enterprise systems. In this paper, we propose effective lifetime-aware dynamic throttling, called LADY, which guarantees the required storage lifetime by intentionally throttling the write performance of SSDs with consideration of the effective write endurance of NAND flash memory. Unlike existing static throttling, LADY makes throttling decisions based on the characteristics of a workload so that the required SSD lifetime can be guaranteed with less performance degradation. LADY also exploits the improvement on write endurance depending on the NAND program speed and the recovery effects of floating-gate transistors, thereby maximally utilizing the available write endurance of NAND flash while mitigating the decreasing write endurance problem. Our experimental results show that LADY improves write performance by 4.7x with small write response time variations over existing static throttling while guaranteeing the required SSD lifetime.

Index Terms—NAND Flash Memory, Storage System, Solid-State Drive, Lifetime Management, Performance Throttling

1 Introduction

NAND flash memory has been widely used in mobile systems ranging from smart phones to laptops. NAND flash-based solid-state drives (SSDs) are now becoming popular storage solutions for enterprise servers. Despite the prevalence of SSDs in enterprise markets, the limited lifetime of SSDs is considered a major obstacle that precludes the use of SSDs in enterprise servers.

Enterprise customers typically require a minimum storage lifetime (which is usually 3 or 5 years) because it is essential for designing storage systems as well as for devising storage deployment and maintenance strategies, such as the calculation of the total costs of ownership (TCO) [2, 3, 4]. In spite of the importance of a storage lifetime in enterprise environments, unfortunately, there are only a few studies on managing the lifetime of flash-based SSD.

The lifetime of SSDs depends on the amount of written data, which is decided by the number of program/erase (P/E) cycles and the SSD capacity. As the semiconductor process is scaled down and a multi-level cell (MLC) technology is adopted, the capacity of SSDs is continuously increased; however, at the same time, the number of P/E cycles is more rapidly decreased. For example, MLC flash doubles the capacity of SSDs, but the number of P/E cycles drops to 3K [5], which is much smaller than 100K P/E cycles of SLC flash. The lifetime of SSDs is also strongly dependent upon write-intensiveness of workloads. SSDs can achieve the required lifetime under non-write-intensive environments where a small amount of data is written by applications. On the other hand, the same SSDs are worn out much earlier if they are used in write-intensive environments. Because of the rapidly decreasing P/E cycles and the workload-dependent lifetime characteristic, it is a great challenge for SSDs to satisfy a minimum storage lifetime that enterprise customers demand.

In this paper, we propose effective **l**ifetime-**a**ware **dy**namic throttling, called *LADY*, which resolves the lifetime problems of SSDs. The main idea of LADY is to intentionally throttle the write performance of SSDs to guarantee the required lifetime. In LADY, the amount of data written per unit time is controlled by adjusting the write speed of SSDs. This makes the lifetime of SSDs predictable, allowing enterprise customers to manage SSDs according to their performance/lifetime requirements.

The important design issue of LADY is how to properly throttle write performance. To achieve better write response times without excessive performance throttling, LADY predicts future write traffic and decides an appropriate write speed so that the SSD is worn out at the end of the target lifetime. In particular, LADY carefully controls the write speed to prevent large fluctuations in write response times. LADY also supports priority-aware dynamic throttling that differently throttles write requests depending on their priorities. This helps us to manage write performance and lifetime according to the importance of enterprise services.

LADY exploits the effective wearing characteristics of NAND flash, so as to minimize a performance penalty associated with write throttling. The damage caused by repetitive P/E cycles is lowered by slowing down the program speed of NAND devices [7]. To take advantage of its benefit on endurance improvement, LADY uses a slow NAND program mode in throttling write performance, instead of merely delaying write requests. Moreover, the damage on memory cells is partially recovered during idle times [8, 9, 10]. LADY makes use of endurance improvement by the self-recovery effect to maximally utilize the effective lifetime of NAND flash. By exploiting the effective wearing characteristics, LADY mitigates the decreasing P/E cycles problem, allowing more data to be written to SSDs.

In order to evaluate the effect of LADY on storage performance and lifetime, we carried out a set of evaluations

- *An earlier version of this paper was presented at the USENIX Conference on File and Storage Technologies, February 14-17, 2012 [1].*
- *S. Lee is with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139. J. Kim is with the Department of Computer Science and Engineering, Seoul National University, Gwanak-ro, Gwanak-gu, Seoul 151-742, Korea. E-mail: jihong@davinci.snu.ac.kr.*

0018-9340 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2349517, IEEE Transactions on Computers

with trace-driven simulators using enterprise traces. Our evaluation results showed that LADY improved the average write response time by 4.7x with smaller variations over existing throttling algorithms while guaranteeing the target SSD lifetime. We also implemented the prototype of LADY in the Linux kernel to show its feasibility in real-world applications like TPC-C.

This paper is organized as follows. In Section 2, we explain the effective wearing characteristics of NAND flash. Section 3 introduces the motivation of dynamic throttling and Section 4 formally describes a dynamic throttling problem after illustrating our write traffic model. In Section 5, we explain the proposed LADY technique in detail. Our evaluation results are presented in Section 6. In Section 7, we explain related work, and finally, Section 8 concludes with summary.

2 Effective Wearing of NAND Flash

In NAND flash, program/erase (P/E) operations inevitably cause damage to floating-gate transistors, reducing the write endurance of memory cells. At the device level, cells are gradually worn out as charges get trapped in the interface and oxide layers of a floating-gate transistor during P/E cycles. This charge trapping increases the threshold voltage of a floating-gate, and the cell becomes unreliable when the threshold voltage is higher than a certain voltage margin (e.g., 0.65V for MLC flash) [8]. According to [8, 10], the increase in a threshold voltage δV_{trap} because of charge trapping approximately scales with P/E cycles in a power-law fashion as follows:

$$\delta V_{trap} = A_{it} \times N^{0.62} + B_{ot} \times N^{0.3}, \quad (1)$$

where N is the number of P/E cycles. A_{it} and B_{ot} are constant and set to 2.97×10^{-3} and 2.0×10^{-2}, respectively. Usually, NAND flash vendors do not reveal important parameters for their recent products. For this reason, A_{it} and B_{ot} for 20 nm MLC flash memory are obtained by scaling up values for 90 nm MLC flash memory (which are available to the public) so that the number of P/E cycles approximately matches 3K at the point where δV_{trap} is 0.65V.

The effective wearing of floating-gate transistors (i.e., the amount of damage caused to cells) is greatly reduced depending on the self-recovery effect of memory cells and the voltage level applied for erasing blocks. Therefore, if the effective wearing characteristics of NAND flash are taken into account, it is possible to exploit much larger P/E cycles than fixed P/E cycles in datasheets.

Self-recovery Effect: A floating-gate transistor has a self-recovery property which heals the damage of a cell by detrapping charges captured in the oxide of a cell. This recovery (or detrapping) process occurs during idle times between P/E cycles on the same cell, and its effect in general increases as the logarithm of idle times (i.e., detrapping $\propto \ln(t)$) where t is the length of idle times. According to [8, 10, 11], the decrease in a threshold voltage δV_{detrap} is expressed as follows:

$$\delta V_{detrap} = C_e \times \delta V_{trap} \times ln(\frac{t}{t_0}), \quad (2)$$

where C_e is a recovery efficiency and set to 5.63×10^{-2} according to [9]. t_0 is 1 hour.

The increase in a threshold voltage δV_{th} with the self-recovery effect is expressed as follows [8]:

$$\delta V_{th} = \delta V_{trap} - \delta V_{detrap}. \quad (3)$$

The length of idle times between P/E cycles on the same block is very long. Thus, the number of P/E cycles with the self-recovery effect is much larger than the number of P/E cycles in datasheets.

Performance/Endurance Trade-off: There is a trade-off between an NAND program speed and the number of P/E cycles depending on the level of a voltage for block erasure. To erase a flash block, an NAND chip controller must apply a high erase voltage (e.g., 14 V) to memory cells. A recent study reported that if the erase voltage is reduced, the damage on cells is lowered as well [7]. Thus, the number of P/E cycles that can be performed increases with the reduced erase voltage. To use the reduced erase voltage, however, more precise charge placement is required to program data to cells, which inevitably increases the program time.

The erase voltage level can be configured continuously, but NAND devices with discrete erase voltage levels are more feasible in practice. Thus, in this work, we assume NAND devices that support four program/erase modes $mode$ for 0, ..., 3 depending on the level of the erase voltage (as proposed in [7]). If $mode$ is 0, a block is programmed with the nominal NAND program speed, but the block must be erased later with the nominal erase voltage without any benefits on write endurance. If $mode$ is 3, a block is programmed with the slowest program speed and erasure for the block can be done with the lowest erase voltage. Thus, the write endurance can be improved.

The number $N_{P/E}^{(mode)}$ (for $0 \leq mode \leq 3$) of P/E cycles depending on the program/erase mode $mode$ is expressed as follows:

$$N_{P/E}^{(mode)} = N_{P/E}^{spec} \times (1 + r_{P/E}^{(mode)}), \quad (4)$$

where $N_{P/E}^{spec}$ is the number of P/E cycles in datasheets (e.g., 3K) and $r_{P/E}^{(mode)}$ is the P/E improvement ratio over the nominal erase voltage. Based on the real measurement study [7], $r_{P/E}^{(mode)}$ for $1 \leq mode \leq 3$ is 0.33, 0.4, and 0.49. $N_{P/E}^{(3)}$ is 1.49x larger than $N_{P/E}^{spec}$. $N_{P/E}^{(0)}$ is $N_{P/E}^{spec}$.

The NAND program time $T_{prog}^{(mode)}$ (for $0 \leq mode \leq 3$) depending on the program/erase mode $mode$ is expressed as follows:

$$T_{prog}^{(mode)} = T_{prog} \times (1 + r_{prog}^{(mode)}), \quad (5)$$

where T_{prog} is the nominal NAND program time and $r_{prog}^{(mode)}$ is the performance degradation ratio over the nominal erase voltage. Based on [7], $r_{prog}^{(mode)}$ for $1 \leq mode \leq 3$ is 0.4, 0.75, and 1.3, respectively. For example, $T_{prog}^{(3)}$ is 2.3x slower than T_{prog}. $T_{prog}^{(0)}$ is T_{prog}.

While the self-recovery effect occurs during idle times, the reduced erase voltage affects the NAND program and block erasure processes. Thus, their effects on the effective wearing are nearly orthogonal. As a result, the number $N_{P/E}^{eff}$ of effective P/E cycles is written as

0018-9340 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2349517, IEEE Transactions on Computers

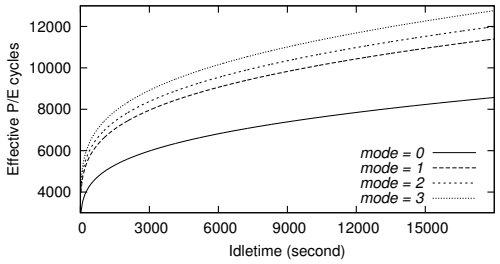


Fig. 1: The number of effective P/E cycles depending on different idle times and program/erase modes

follows:

$$N_{P/E}^{eff} = N_{P/E}^{recov} \times (1 + r_{P/E}^{(mode)}), \quad (6)$$

where $N_{P/E}^{recov}$ is the number of P/E cycles with the self-recovery effect.

Based on Eq. (6), we plot the effective P/E cycles of 20 nm MLC flash memory in Fig. 1 depending on the length of idle times with different program/erase modes. The number of effective P/E cycles is increased in proportional to the length of idle times. Similarly, as the slower NAND program mode is used, the effective lifetime is improved as well.

3 Motivation

Fig. 2 shows our motivational example for dynamic throttling. Based on the specification of the SSD, the maximum amount of data that can be written is proportional to the SSD capacity and the number of P/E cycles on datasheets. For example, if the number of P/E cycles is 3K and the SSD capacity is 128 GB, the data of 375 TB can be written. Suppose that a target SSD lifetime is 5 years. In the example of Fig. 2(a) with no throttling, the SSD is worn out before the target lifetime.

In order to ensure a lifetime warranty, some SSD vendors recently adopt *static throttling* [12, 13]. As shown in Fig. 2(b), static throttling guarantees the required lifetime by statically limiting the maximum write throughput under the assumption that incoming write traffic is always heavy. In practice, actual workloads are not intensive all the time, so static throttling often slows down the write speed of SSDs uselessly, underutilizing the available SSD endurance. For example, in Fig. 2(b), the SSD is still not worn out at 5 years. Static throttling also incurs large fluctuations in write response times; it throttles the write speed so much when a workload is intensive; on the other hand, it never throttles write performance when a workload is not intensive.

The proposed effective lifetime-aware dynamic throttling technique, LADY, overcomes the limitations of static throttling. As depicted in Fig. 2(c), LADY dynamically changes the write speed of the SSD according to the characteristics of a workload so that the write endurance is maximally utilized without excessive write throttling. In particular, LADY exploits the performance/endurance trade-off by adaptively changing the program/erase mode and considers the endurance improvement by the self-recovery effect. This increases the number of P/E cycles, allowing more data to be written to the SSD.

LADY is designed with the following objectives in mind to properly control the write speed of the SSD.

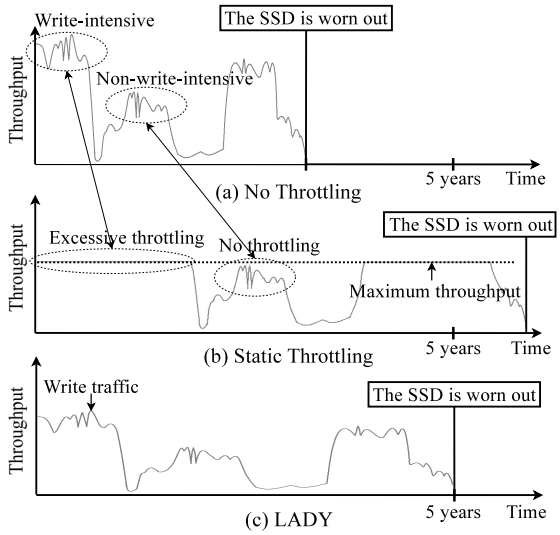


Fig. 2: A comparison of throttling policies: no throttling, static throttling, and LADY

First, the write speed must be properly decided so that the SSD is worn out at the end of the target lifetime. If the SSD is less throttled, the required lifetime cannot be guaranteed (like the SSD without write throttling). If the write speed is excessively throttled, the write performance could significantly deteriorate, underutilizing available write endurance (like the SSD with static throttling). Second, the write speed must be properly decided so that response time variations are minimized. If the write speed is too throttled in a certain time-period while it is less throttled over another time-period, I/O response times are greatly fluctuated, resulting in the degradation of the user experience.

4 Problem Formulation

4.1 Write Traffic Model

We first present a write traffic model used throughout this paper and formally define a dynamic throttling problem using our model [1]. Our write traffic model includes all of the write requests from the host system as well as from FTL modules, including garbage collection and wear-leveling. The target SSD lifetime is denoted by T_{ssd}. The total amount of data that can be written until the SSD is worn out is denoted by C_{ssd}. Note that C_{ssd} changes depending on the length of idle times and the program/erase mode.

The top figure of Fig. 3 shows original write traffic represented by our write traffic model. We define a *write element* $\mathcal{W}_i$ as the basic unit of write traffic and associate it with the size of requested data and properties related to time. The size of $\mathcal{W}_i$ is fixed to a page which is the smallest unit for writing data. The size of $\mathcal{W}_i$ is denoted by $\mathcal{F}_S(\mathcal{W}_i)$. $\mathcal{W}_i$ has two time properties, T_{prog} and τ_i^{it}. Here, T_{prog} is the time taken to program $\mathcal{W}_i$. τ_i^{it} is the length of idle times until the next write element $\mathcal{W}_{i+1}$ arrives after the data of $\mathcal{W}_i$ are completely served. The total length of the time for $\mathcal{W}_i$ is denoted by $\mathcal{F}_T(\mathcal{W}_i)$, and it is $T_{prog} + \tau_i^{it}$.

1. For readers' convenience, we summarize mathematical terms frequently used throughout this paper in Section 1 of Appendix.

0018-9340 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2349517, IEEE Transactions on Computers

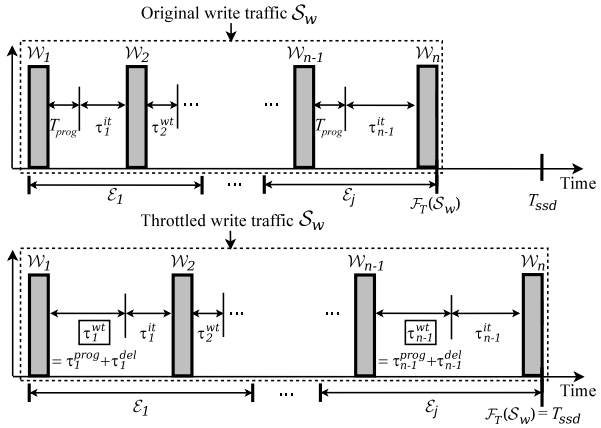


Fig. 3: An illustration of original write traffic (top) and throttled write traffic (bottom) in our write traffic model

Using $\mathcal{W}_i$ defined above, we represent the write traffic sent to the SSD until it becomes unreliable as a sequence $\mathcal{S}_w$ of write elements i.e., $\mathcal{S}_w = \langle \mathcal{W}_1, \ldots, \mathcal{W}_n \rangle$, where $\mathcal{W}_i$ occurs before $\mathcal{W}_q$ if $i < q$. The total size of data written by $\mathcal{S}_w$ is denoted by $\mathcal{F}_S(\mathcal{S}_w)$. Similarly, $\mathcal{F}_T(\mathcal{S}_w)$ is the total length of time until the data of $\mathcal{F}_S(\mathcal{S}_w)$ are written to the SSD. $\mathcal{F}_S(\mathcal{S}_w) = \mathcal{F}_S(\mathcal{W}_1) + \cdots + \mathcal{F}_S(\mathcal{W}_n)$ and $\mathcal{F}_T(\mathcal{S}_w) = \mathcal{F}_T(\mathcal{W}_1) + \cdots + \mathcal{F}_T(\mathcal{W}_n)$. $\mathcal{F}_S(\mathcal{S}_w)$ is equal to C_{ssd} because the SSD is worn out after the data of C_{ssd} are written. If write traffic is heavy and dynamic throttling is not used, $\mathcal{F}_T(\mathcal{S}_w)$ could be smaller than T_{ssd}.

In our write traffic model, time is divided into *time-epochs*, simply called *epochs*. Here, an epoch is a unit period of time for predicting future write traffic and deciding a write speed (see Section 5). Given the sequence $\mathcal{S}_w$, we construct a sequence $\mathcal{E}_k$ of write elements for an epoch k, i.e., $\mathcal{E}_k = \langle \mathcal{W}_{k_1}, \ldots, \mathcal{W}_{k_m} \rangle$ where $\mathcal{W}_{k_1}$ is a write element in $\mathcal{S}_w$ that first arrives at the SSD after $\mathcal{E}_k$ begins and $\mathcal{W}_{k_m}$ is the last one before $\mathcal{E}_k$ ends. The amount of data written during $\mathcal{E}_k$ is denoted by $\mathcal{F}_S(\mathcal{E}_k)$ and the length of $\mathcal{E}_k$ is denoted by $\mathcal{F}_T(\mathcal{E}_k)$. As expected, $\mathcal{F}_S(\mathcal{E}_k) = \mathcal{F}_S(\mathcal{W}_{k_1}) + \cdots + \mathcal{F}_S(\mathcal{W}_{k_m})$ and $\mathcal{F}_T(\mathcal{E}_k) = \mathcal{F}_T(\mathcal{W}_{k_1}) + \cdots + \mathcal{F}_T(\mathcal{W}_{k_m})$. The write traffic $\mathcal{S}_w$ is also represented as a sequence of epochs, i.e., $\mathcal{S}_w = \langle \mathcal{E}_1, \ldots, \mathcal{E}_j \rangle$ where j is the number of epochs in $\mathcal{S}_w$.

4.2 Dynamic Throttling Problem

LADY delays individual write elements $\mathcal{W}_i$ so that the write traffic $\mathcal{S}_w$ is properly regulated to offer a lifetime guarantee. In LADY, the time taken to write the data of $\mathcal{W}_i$ is variable. As illustrated in Fig. 3, this *variable write time* τ_i^{wt} is decided by two delay factors, τ_i^{prog} and τ_i^{del}, where τ_i^{prog} is the NAND program time $T_{prog}^{(mode)}$ depending on the program/erase mode and τ_i^{del} is the length of an *artificial delay*. This artificial delay is needed because τ_i^{prog} could not be long enough to sufficiently delay $\mathcal{W}_i$. If τ_i^{wt} is longer than T_{prog}, the time taken to write $\mathcal{W}_i$ is increased by $(\tau_i^{wt} - T_{prog})$. This increased write time reduces the SSD write performance, creating the illusion that the SSD operates slowly.

The main objective of LADY is to decide τ_i^{wt} for $\mathcal{W}_i$ so that $\mathcal{F}_S(\mathcal{S}_w)$ is equal to C_{ssd} at T_{ssd}. To minimize response time variations, τ_i^{wt} must be distributed across $\mathcal{W}_i$ as evenly as possible. Consequently, the problem of

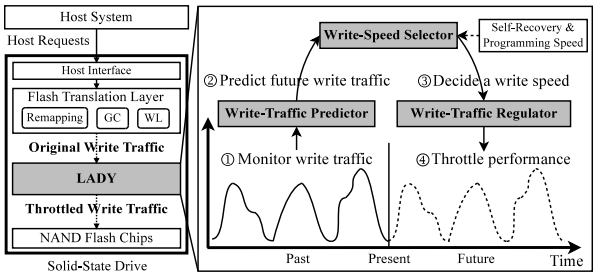


Fig. 4: Three main functions of LADY

dynamic throttling can be expressed as follows:

$$\begin{aligned}&\text{Decide } \tau_i^{wt} \text{ for } \mathcal{W}_i \text{ in } \mathcal{S}_w \text{ if } \mathcal{F}_T(\mathcal{S}_w) < T_{ssd}\\ &\quad \text{subject to}\\ &\qquad \mathcal{F}_S(\mathcal{S}_w) = C_{ssd} \text{ at } T_{ssd} \text{ and } \tau_1^{wt} = \cdots = \tau_{n-1}^{wt}. \quad (7)\\ &\text{Otherwise,}\\ &\quad \tau_1^{wt} = \cdots = \tau_{n-1}^{wt} = T_{prog}.\end{aligned}$$

In Eq. (7), τ_i^{wt} is decided by LADY when $\mathcal{W}_i$ arrives and the SSD lifetime T_{ssd} is determined by enterprise customers. However, the write traffic $\mathcal{S}_w$, including $\mathcal{F}_S(\mathcal{S}_w)$ and $\mathcal{F}_T(\mathcal{S}_w)$, is unknown when a decision on τ_i^{wt} is made. Moreover, C_{ssd} changes according to the length of idle times as well as the program/erase mode chosen. Thus, the future write traffic $\mathcal{S}_w$ and the effective write endurance C_{ssd} must be carefully estimated for write throttling. LADY is designed to properly decide τ_i^{wt} in real-world environments where no knowledge on $\mathcal{S}_w$ and C_{ssd} is available a priori.

5 Design and Implementation of LADY

Fig. 4 shows the overall architecture of LADY, which is composed of three modules: a *write-traffic predictor*, a *write-speed selector*, and a *write-traffic regulator*. The write-traffic predictor analyzes the history of previous write traffic and estimates the future write traffic (see Section 5.1). The write-speed selector decides the write speed based on the predicted write traffic and the remaining write endurance. The remaining write endurance is estimated by taking into account the program/erase mode and the self-recovery effect (see Section 5.2). Finally, the write-traffic regulator throttles the write performance so that the target SSD lifetime will be reached with small response time variations (see Section 5.3).

LADY is implemented based on a modular design concept; it runs below the FTL without any functional dependencies. All of the I/O requests sent from both the host system and the FTL (e.g., GC and WL) are sent to LADY. LADY creates the illusion that the FTL runs on slower NAND flash; it monitors write traffic from the FTL, decides a write speed, and throttles individual write elements. Thanks to this modular design, any kinds of FTL schemes can run on top of LADY.

5.1 Estimation of Future Write Traffic

The write-traffic predictor of LADY uses an epoch-based approach that estimates future write traffic on an epoch-by-epoch basis. This is based on our observation that even though it is difficult to exactly predict the whole of the future write traffic S_w in advance, the write traffic in the near future can be accurately estimated by referring

0018-9340 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2349517, IEEE Transactions on Computers

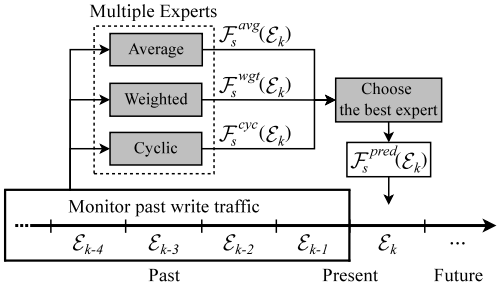


Fig. 5: Prediction of future write traffic

to the previous history. The write-traffic predictor also adopts a multiple-expert system [14, 15, 16] that runs simple multiple workload-prediction policies at the same time and chooses the best one with the highest accuracy. The multiple-expert system not only effectively deals with various workloads [15], but also is useful in a resource-constrained environment like SSDs [16].

Fig. 5 shows how the write-traffic predictor estimates the future write traffic $\mathcal{F}_S^{pred}(\mathcal{E}_k)$ for the next epochs $\mathcal{E}_k$. At the beginning of each epoch, it evaluates the accuracy of individual experts. For each expert, the write-traffic predictor calculates the difference between the predicted write traffic and the actual write traffic. Then, it chooses the expert that shows the best accuracy for the past epoch. The write-traffic predictor employs three experts for workload prediction: *Average*, *Weighted*, and *Cyclic*.

• *Average* is a global average of write traffic observed in all of the previous epochs. *Average* is effective when the long-term behavior of a workload is stable and is not changed greatly. In *Average*, the future write traffic $\mathcal{F}_S^{avg}(\mathcal{E}_k)$ for $\mathcal{E}_k$ is estimated as follows:

$$\mathcal{F}_S^{avg}(\mathcal{E}_k) = \frac{\mathcal{F}_S(\mathcal{E}_{k-1}) + \mathcal{F}_S(\mathcal{E}_{k-2}) + ... + \mathcal{F}_S(\mathcal{E}_1)}{k}. \quad (8)$$

• *Weighted* is an extension of *Average*. It gives a higher weight to recent three epochs, so as to reflect the recency of a workload. In *Weighted*, the future write traffic $\mathcal{F}_S^{wgt}(\mathcal{E}_k)$ for $\mathcal{E}_k$ is estimated as follows:

$$\mathcal{F}_S^{wgt}(\mathcal{E}_k) = \frac{\mathcal{F}_S(\mathcal{E}_{k-1}) + \mathcal{F}_S(\mathcal{E}_{k-2}) + \mathcal{F}_S(\mathcal{E}_{k-3})}{3} \times 0.8 + \frac{\mathcal{F}_S(\mathcal{E}_{k-4}) + ... + \mathcal{F}_S(\mathcal{E}_1)}{k-3} \times 0.2. \quad (9)$$

• *Cyclic* is motivated by the previous observations [17] that enterprise workloads often exhibit cyclic behaviors with periods between several minutes or several days. *Cyclic* detects the repeated pattern of a workload and adjusts the length of an epoch so that it includes the entire repeated pattern [1]. Then, it assumes that the write traffic observed in the latest epoch will be repeated again in the next epoch. In *Cyclic*, the future write traffic $\mathcal{F}_S^{cyc}(\mathcal{E}_k)$ for $\mathcal{E}_k$ is estimated as follows:

$$\mathcal{F}_S^{cyc}(\mathcal{E}_k) = \mathcal{F}_S(\mathcal{E}_{k-1}). \quad (10)$$

In Fig. 5, if $\mathcal{F}_S^{cyc}(\mathcal{E}_k)$ exhibits the smallest write-traffic difference, $\mathcal{F}_S^{pred}(\mathcal{E}_k)$ is set to $\mathcal{F}_S^{cyc}(\mathcal{E}_k)$.

5.2 Determination of Write Speed

The write-speed selector decides the write speed for the next epoch based on the future write traffic and the available write endurance. In this subsection, we first explain our mechanism for deciding the write speed and then describe how the available write endurance is estimated with consideration of the self-recovery effect as well as the program/erase mode.

```
Decide_Write_Speed (mode_{k-1}, τ^wt_{k-1}) {
1:  if (F^pred_S(E_k) == c_k) { /* with the same program mode
                                  used in the previous epoch */
2:      return (mode_{k-1}, τ^wt_{k-1});
3:  }
4:  mode := mode_{k-1};
5:  τ^wt_k := ∞;
6:  while (1) {
      /*(1) Decide a change Δτ^wt in the previous write time τ^wt_{k-1}*/
7:    Change the NAND program speed to T^(mode)_prog;
8:    if (F^pred_S(E_k) > c_k) {
9:      calculate Δτ^wt using Eq. (11);
10:     τ^wt_tmp := τ^wt_{k-1} + Δτ^wt;
11:   } else if (F^pred_S(E_k) < c_k) {
12:     calculate Δτ^wt using Eq. (12);
13:     τ^wt_tmp := max (T^(mode)_prog, τ^wt_{k-1} - Δτ^wt);
14:   } else { /* F^pred_S(E_k) = c_k */
15:     τ^wt_tmp := T^(mode)_prog;
16    }
17:   if (τ^wt_tmp < τ^wt_k) {
18:     mode_k := mode;
19:     τ^wt_k := τ^wt_tmp;
20:   } else break;
      /*(2) Decide a proper program/erase mode*/
21:   if (mode < 3 and T^(mode+1)_prog ≤ τ^wt_k) {
        /* increase the NAND program speed */
22:     mode := mode + 1;
23:     continue;
24:   }
25:   if (mode > 0 and T^(mode)_prog == τ^wt_k) {
        /* decrease the NAND program speed */
26:     mode := mode - 1;
27:     continue;
28:   }
29:   break;
30: }
31: return (mode_k, τ^wt_k); /* τ^del_k = τ^wt_k − T^(mode_k)_prog */
};
```

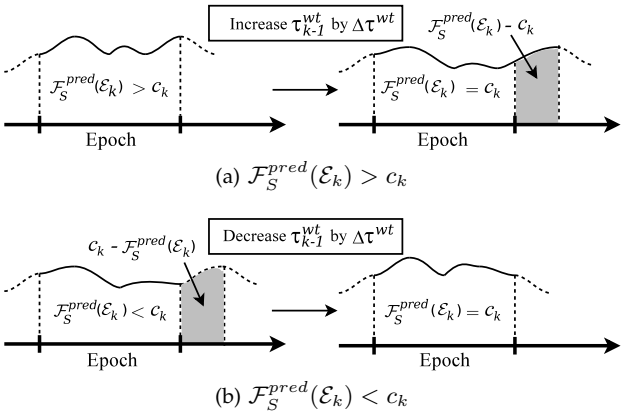
Fig. 6: A write speed decision algorithm

Write-Speed Decision Overview: Whenever a new epoch $\mathcal{E}_k$ begins, the write-speed selector decides a write time τ_k^{wt} for $\mathcal{E}_k$ by *increasing or decreasing a previous write time* τ_{k-1}^{wt} used for a previous epoch $\mathcal{E}_{k-1}$. The newly decided write time is equally applied to all write elements in the epoch (i.e., for $\mathcal{W}_{k_1}, \ldots, \mathcal{W}_{k_m}$ in $\mathcal{E}_k$, $\tau_{k_1}^{wt}, \ldots, \tau_{k_m}^{wt}$ are equal to τ_k^{wt}). After the epoch ends, the write-speed selector decides a write time τ_{k+1}^{wt} for a next epoch $\mathcal{E}_{k+1}$ by updating τ_k^{wt}. In LADY, the changes of the write time occur only at the beginning of each epoch. This is not only useful to avoid response time variations caused by the frequent write speed changes, but also allows us to decide the write speed according to changing workloads.

Write-Speed Decision Algorithm: We now detail how the write-speed selector decides the write speed. The write time τ_k^{wt} is decided by two factors: the program/erase mode τ_k^{prog} and the length of an artificial delay τ_k^{del}. The write-speed selector starts with the nominal program/erase mode with no artificial delays. Then, at the beginning of the k-th epoch $\mathcal{E}_k$ ($k > 1$), it decides the write speed based on the expected future write traffic $\mathcal{F}_S^{pred}(\mathcal{E}_k)$ and an epoch capacity c_k. The epoch capacity

0018-9340 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

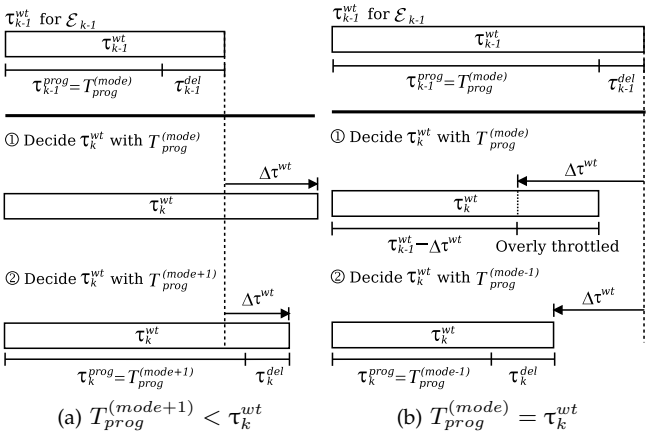
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2349517, IEEE Transactions on Computers



(a) $\mathcal{F}_S^{pred}(\mathcal{E}_k) > c_k$

(b) $\mathcal{F}_S^{pred}(\mathcal{E}_k) < c_k$

Fig. 7: A change in a write time



(a) $T^{(mode+1)}_{prog} < \tau^{wt}_k$ (b) $T^{(mode)}_{prog} = \tau^{wt}_k$

Fig. 8: A change in a program mode

c_k is the number of writable bytes assigned to $\mathcal{E}_k$. The epoch capacity changes depending on the length of idle times and the NAND program mode. We explain how the epoch capacity is estimated later.

Fig. 6 shows our write speed decision algorithm. The write-seed selector first sees if the expected future write traffic is equal to the epoch capacity (i.e., $\mathcal{F}_S^{pred}(\mathcal{E}_k) = c_k$) with the program mode used before. If it is, the write time becomes the same as the previous one under the assumption that the similar traffic would be repeated in future (`lines 1-3`). Otherwise, the write-speed selector attempts to decide the new write time based on the expected future write traffic and the epoch capacity. The write-speed selector uses the same program/erase mode that was used for the previous epoch (i.e., $mode$:= $mode_{k-1}$). The write time is initially set to ∞ (i.e., τ^{wt}_k := ∞) (`lines 4-5`).

If the expected future write traffic is larger than the epoch capacity as shown in Fig. 7(a) (i.e., $\mathcal{F}_S^{pred}(\mathcal{E}_k) > c_k$), the write time must be longer than the previous write time. A change $\Delta\tau^{wt}$ in the previous write time is obtained as follows:

$$\Delta\tau^{wt}_k = \frac{\left\{\mathcal{F}_T^{pred}(\mathcal{E}_k) \times \left(\frac{\mathcal{F}_S^{pred}(\mathcal{E}_k)}{c_k} - 1\right)\right\}}{m} \quad \text{if } \mathcal{F}_S^{pred}(\mathcal{E}_k) > c_k, \quad (11)$$

where m is the number of write elements allowed to be written during $\mathcal{E}_k$ and $\mathcal{F}_T^{pred}(\mathcal{E}_k)$ is the length of $\mathcal{E}_k$. To make the data written during $\mathcal{E}_k$ equal to c_k, $\mathcal{F}_S^{pred}(\mathcal{E}_k) - c_k$ of the data must be delayed to the next epoch as shown in Fig. 7(a). The total time required to delay those data is approximated as $\mathcal{F}_T^{pred}(\mathcal{E}_k) \times (\mathcal{F}_S^{pred}(\mathcal{E}_k)/c_k - 1)$. Since write elements are equally delayed, $\Delta\tau^{wt}_k$ is obtained by dividing the total delay by m. τ^{wt}_{k-1} + $\Delta\tau^{wt}$ is chosen as a temporary write time τ^{wt}_{tmp} (`lines 8-10`).

If the expected future write traffic is smaller than the epoch capacity as shown in Fig. 7(b) (i.e., $\mathcal{F}_S^{pred}(\mathcal{E}_k) < c_k$), it means that write requests were not intensive enough to wear out the SSD before a target lifetime or they were too throttled in the previous epoch. The write time must be shorter than the previous write time so that more data are to be written. A change $\Delta\tau^{wt}$ in the previous write time is as follows:

$$\Delta\tau^{wt}_k = \frac{\left\{\mathcal{F}_T^{pred}(\mathcal{E}_k) \times \left(\frac{c_k}{\mathcal{F}_S^{pred}(\mathcal{E}_k)} - 1\right)\right\}}{m} \quad \text{if } \mathcal{F}_S^{pred}(\mathcal{E}_k) < c_k. \quad (12)$$

To increase data to be written by $c_k - \mathcal{F}_S^{pred}(\mathcal{E}_k)$, the temporary write time is set to $\tau^{wt}_{k-1} - \Delta\tau^{wt}$. If $\tau^{wt}_{k-1} - \Delta\tau^{wt} < T^{(mode)}_{prog}$, it is set to $T^{(mode)}_{prog}$ because the program time is fixed to $T^{(mode)}_{prog}$ (`lines 11-13`).

If the temporary write time is shorter than the previously decided write time (i.e., $\tau^{wt}_{tmp} < \tau^{wt}_k$), the write-speed selector chooses the temporary one along with the corresponding NAND program mode (i.e., τ^{wt}_k = τ^{wt}_{tmp} and $mode_k$ = $mode$). Otherwise, the previously decided write time is maintained and the write speed decision is finished (`lines 17-20`). Note that the artificial delay τ^{del}_k is $\tau^{wt}_k - T^{(mode_k)}_{prog}$ (`line 31`).

The write-speed selector attempts to choose a proper program/erase mode. If $mode$ is smaller than 3 and the write time is longer than the next slower NAND program speed (i.e., $T^{(mode+1)}_{prog} \leq \tau^{wt}_k$), the write-speed selector selects the slower one (`lines 21-24`). Then, the write time is calculated again. The slower program mode improves the write endurance, allowing us to write more data. Thus, as depicted in Fig. 8(a), the write time decreases with the slower mode.

If $mode$ is larger than 0 and the current NAND program time is equal to the write time (i.e., $T^{(mode)}_{prog} = \tau^{wt}_k$), the NAND program speed may be set too slow (i.e., $T^{(mode)}_{prog} > \tau^{wt}_{k-1} - \Delta\tau^{wt}$). This would overly throttle write performance. The write-speed selector chooses the next faster mode and calculates the write time again (`lines 25-28`). As illustrated in Fig. 8(b), the over-throttling problem can be avoided with the faster program mode. As a result, the write time with the faster mode is shorter than that with the slower one.

The expected future write traffic could be the same as the epoch capacity after changing the program mode (i.e., $\mathcal{F}_S^{pred}(\mathcal{E}_k) == c_k$). In that case, the temporary write time is the same as the newly chosen program time because future write traffic would be properly regulated with no artificial delays (`lines 14-16`).

Finally, the write-speed selector changes the program/erase mode until the above two conditions are not met (`line 29`) or the write time with the new program/erase mode is longer than the previously decided one (`line 20`).

Estimating Epoch Capacity: In order to know the epoch capacity c_k, the write-speed selector first estimates

0018-9340 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2349517, IEEE Transactions on Computers

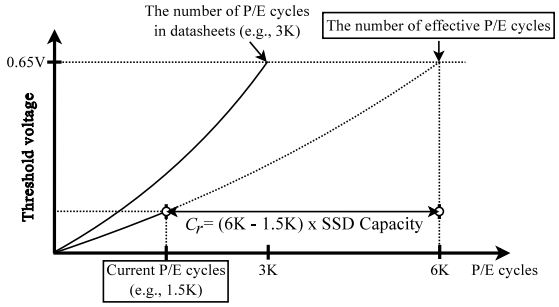


Fig. 9: Estimation of the remaining writable bytes

the number C_r of remaining bytes that can be written until the SSD becomes unreliable. It then decides the epoch capacity by equally distributing the number of remaining bytes to future epochs. The wearing-rate of NAND flash depends on the length of idle times and the program/erase mode. The number of remaining writable bytes is thus a function of the current P/E cycles, the length of idle times between consecutive P/E cycles T_{idle}, and the NAND program mode $T_{prog}^{(mode)}$ chosen for writing data.

$$C_r = f_{c_r}(\text{current P/E cycles},\ T_{idle},\ T_{prog}^{(mode)}). \qquad (13)$$

The SSD is composed of several thousands of flash blocks, so the current P/E cycles and T_{idle} of individual blocks are different from each block. In this paper, we estimate remaining bytes in a conservative way. The highest number of P/E cycles among all of the flash blocks is chosen as current P/E cycles and the shortest idle time is used as T_{idle}.

Fig. 9 illustrates how C_r is estimated. The write-speed selector estimates the maximum number of effective P/E cycles using Eq. (6) in Section 2 under the assumption that T_{idle} and $T_{prog}^{(mode)}$ will be the same in the future. In Fig. 9, the number of current P/E cycles and the maximum number of effective P/E cycles are 1.5K and 6K, respectively. The number of remaining effective P/E cycles is 4.5K (= 6K - 1.5K), so C_r is 4.5K$\times$SSD Capacity. Finally, the epoch capacity c_k is obtained by dividing C_r by the number of remaining epochs.

5.3 Write Performance Regulation

Once the write speed is decided, the write-traffic regulator of LADY throttles write performance by equally delaying write elements in the same epoch. This write-traffic regulation is effective in minimizing response time variations, but it may not guarantee the required lifetime all the time – if unexpectedly heavy write traffic comes, more data than the epoch capacity could be written. For this reason, the write-traffic regulator adopts the epoch-capacity regulation policy that prevents more data than our expectation from being written.

One of the easiest ways to enforce the epoch capacity is to stop writing if the epoch capacity is exhausted. We call it *strict capacity regulation*. For example, suppose that the epoch length is 4 seconds and the epoch capacity is 1 MB. Further suppose that the epoch capacity of 1 MB runs out at the end of the 3rd second. The write-traffic regulator prevents the overuse of the epoch capacity by stopping writing data for the remaining 1 second. This strict capacity regulation incurs great response time variations because write requests arriving after 3 seconds are delayed until the next epoch begins.

We resolve this problem by introducing the concept of a *default capacity* and a *spare capacity*. The default capacity is some of the epoch capacity, which is evenly assigned to every second by default. The default capacity is useful to offer the minimum write throughput. The spare capacity is an additional capacity borrowed from future epochs. If the spare capacity is 10%, the write-traffic regulator borrows 10% of the capacities of all the future epochs. If more data than the default capacity are written, the write-traffic regulator temporarily uses the spare capacity for writing data, avoiding strict capacity regulation. After the epoch ends, the write speed for future epochs is slightly reduced to reclaim the epoch capacity overly used by previous epochs.

We give a detailed explanation about how the write-traffic regulator behaves with the spare capacity. Suppose that the spare capacity is 10% and there are j epochs. The epoch capacity is denoted by c_1, ..., c_j, respectively, and c_1 = ... = c_j = C_r/j. If the length of the epoch is n seconds, the default capacity for the 1st epoch is c_1/n. The spare capacity for the 1st epoch is $(c_2 + ... + c_j) \times 0.1$. The total capacity assigned to the 1st epoch is $c_1 + (c_2 + ... + c_j) \times 0.1$. For example, if j is 4 and C_r is 4 MB, c_1 is 1 MB (= 4 MB/4) and the spare capacity is 0.3 MB (= 3 MB$\times$0.1). If the data smaller than c_1 have been written, the remaining capacity C_r after the 1st epoch is equally distributed to the remaining epochs and the spare capacity is decided by $(c_3+...+c_j)\times 0.1$ for the 2nd epoch. For instance, if 1.0 MB of data are written in the 1st epoch, the spare capacity is not used and C_r is reduced to 3.0 MB [2]. c_2, c_3, and c_4 are 1.0 MB (= 3.0 MB/3) and the spare capacity is 0.2 MB (= 2.0 MB$\times$0.1).

If the spare capacity is partially used, c_2, ... , c_j are reduced to 90% of the original capacity and only the unallocated capacity is used as the spare capacity. In the above example, if the data of 1.2 MB are written in the 1st epoch, C_r is reduced to 2.8 MB at the beginning of the 2nd epoch. c_2, c_3, and c_4 are 0.9 MB and the spare capacity becomes 0.1 MB. This capacity assignment makes the write-speed selector slightly reduce write performance with a smaller epoch capacity (i.e., the epoch capacity is 0.9 MB instead of 1.0 MB), helping us to reclaim the overused capacity in the future epochs.

In the worst case, the spare capacity could be completely used up before the epoch ends. In this case, the write-traffic regulator strictly regulates write traffic to prevent more data than we assigned from being written. In the example above, the sum of the epoch and spare capacities assigned to the 1st epoch is 1.3 MB. Fig. 10 illustrates what happens when the data of 1.4 MB are requested for writing in the 1st epoch. Here, we assume that the length of the epoch is 4 seconds. The write-traffic regulator assigns the default capacity of 0.25 MB (= 1 MB/4 seconds) to every second. The spare capacity is not assigned and is reserved. Initially, 0.25 MB are written in

2. According to Eq. (13), C_r changes depending on the length of idle times and the program mode. For the sake of simplicity, we assume that these parameters are the same as the previous epoch.

0018-9340 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2349517, IEEE Transactions on Computers

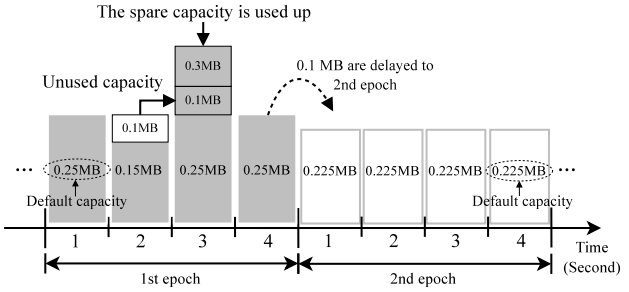


Fig. 10: An example of how LADY handles write traffic when the spare capacity is exhausted

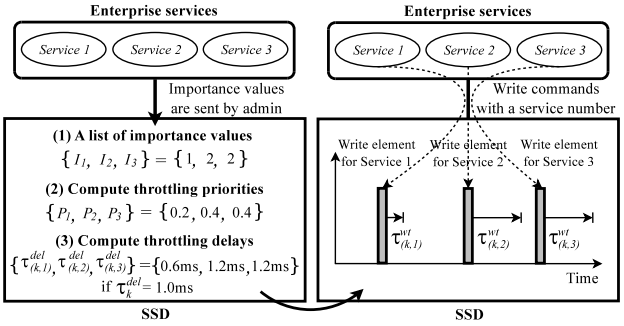


Fig. 11: A procedure of PA-DT in deciding write speeds

the 1st second and 0.15 MB are written in the 2nd second. 0.1 MB of the capacity assigned to the 2nd second are not used. The unused capacity is forwarded to the 3rd second. In the 3rd second, 0.65 MB are written. The total capacity assigned to the 3rd second is completely consumed (= the spare capacity of 0.3 MB + the unused capacity of 0.1 MB + the default capacity of 0.25 MB). In the 4th second, 0.35 MB are requested for writing. Since the spare capacity is exhausted, the write-traffic regulator allows only the data of 0.25 MB to be written, delaying the data of 0.1 MB to the next second. As a result, 1.3 MB of data have been written in the 1st epoch.

After the spare capacity runs out, the write-traffic regulators sets the spare capacity to 0 MB and then throttles incoming write traffic using strict capacity regulation. At the beginning of the 2nd epoch, C_r is 2.7 MB and the capacity of the 2nd epoch becomes 0.9 MB. The spare capacity is 0 MB. Therefore, more data than 0.9 MB cannot be written in the 2nd epoch. This could incur great response time variations, but allows us to guarantee the required lifetime.

The spare capacity must be carefully chosen. If the spare capacity is unlimited, it is equivalent to LADY without epoch capacity regulation in which an unlimited spare capacity can be borrowed from future epochs. On the other hand, if the spare capacity is too small, strict capacity regulation would be frequently observed due to the lack of the spare capacity. In this work, the spare capacity is empirically set to 10% of the remaining capacity. In our observation, it is large enough to avoid strict capacity regulation in real-world applications.

5.4 Priority-Aware Dynamic Throttling

Enterprise services have different performance requirements, so their importance is different from one another [18, 19]. For example, in database management systems (DBMS), transactions having a great effect on end-users' experiences are handled with a higher priority than other ones. Similarly, interactive enterprise services are also served in advance of other services like batch jobs, so as to improve the quality of end-users.

To effectively handle write requests from services with different priorities, we propose priority-aware dynamic throttling, called PA-DT, which is an extension of LADY. The basic idea of PA-DT is to less throttle write requests from important services so that they are more quickly served. The SSD lifetime is more used by important services. This overused lifetime is offset by further throttling write requests from less important services

Fig. 11 illustrates how PA-DT decides write speeds for services with different importance. Initially, PA-DT receives a list of importance values I_l for enterprise services *Service* l from administrators, i.e., *Service 1*, *Service 2*, and *Service 3* as in the example of Fig. 11. The importance value of a service ranges from 1 to 4. In Fig. 11, the importance values I_1, I_2, and I_3 for *Services 1*, *2*, and *3* are 1, 2, and 2, respectively. An enterprise service with a smaller important value is more important. PA-DT computes relative throttling priorities P_l for target services using their importance values. PA-DT obtains a throttling priority P_l for *Service l* as follows: $P_l = I_l/(I_1 + \cdots + I_{N_s})$, where N_s is the number of services with specific importance values. The sum of the throttling priorities is always 1.0. In Fig. 11, P_1, P_2, and P_3 are 0.2, 0.4, and 0.4, respectively. Similarly, a service with a smaller throttling priority has a higher priority.

According to the priorities of services, PA-DT decides priority-aware artificial delays $\tau^{del}_{(k,l)}$ for *Services* l as follows: $\tau^{del}_{(k,l)} = N_s \times \tau^{del}_k \times P_l$. In Fig. 11, $\tau^{del}_{(k,1)}$, $\tau^{del}_{(k,2)}$, and $\tau^{del}_{(k,3)}$ for *Services 1*, *2*, and *3* are 0.6 ms, 1.2 ms, and 1.2 ms if τ^{del}_k is 1.0 ms. $\tau^{del}_{(k,l)}$ is proportional to P_l. For example, $\tau^{del}_{(k,1)}$ is twice shorter than $\tau^{del}_{(k,2)}$ and $\tau^{del}_{(k,3)}$. For services without importance values, τ^{del}_k is used. Finally, PA-DT throttles write elements differently according to their write times $\tau^{wt}_{(k,l)}$.

In order to support PA-DT, the SSD firmware must know (i) importance values for enterprise services and (ii) service numbers to which individual write elements belong. First, a list of important values for services can be delivered by adding a new custom command to the existing bus interface, such as ATA and SATA. The S.M.A.R.T function of the SATA interface that allows device vendors to define their own commands would be useful for this. Second, in order to deliver a service number to which a write element belongs, a SATA WRITE command must be modified to specify a service number of a write request. The modification of a block device driver is also needed to deliver software-side information (i.e., important values and service numbers) via those H/W interfaces. Considering that SSD manufacturers often offer custom device drivers and interfaces, it would not be a serious obstacle to add such custom commands/interfaces to SSD products.

5.5 Discussion

Multi-channel architecture support: For the sake of simplicity, we describe LADY based on the SSD with

0018-9340 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2349517, IEEE Transactions on Computers

single-channel architecture, but LADY works well with multi-channel architecture. Regardless of the underlying channel architecture, LADY controls write performance by changing the write speed of individual NAND devices (e.g., NAND flash chips). This throttling mechanism allows LADY to work independently of organizations of SSD architectures. For a more detailed explanation, please see Section 3 of Appendix.

Throttling issues with multi-threaded and multiple applications: In our write traffic model in Section 3, we assume that if one write element is delayed by t, the arrival times of the following write elements are delayed by the same amount of time. This is not true in some environments like multi-threaded and multiple applications. Suppose that two write elements are issued by two threads running on different CPUs. In that case, even if one write element is delayed by t, the arrival time of the other write element may not be delayed because two threads run independently. Therefore, the write traffic may be not properly throttled and more data than our expectation are written. LADY effectively handles such a situation. If incoming write traffic is not sufficiently throttled, LADY continuously increases a write time, further slowing down the write speed of the SSD, until write traffic is properly regulated. The behaviors of host applications in fact do not affect LADY. A more detailed description of throttling issues with multiple applications is found in Section 4 of Appendix.

Using LADY in the RAID system: The SSD with LADY can be used to comprise a disk array of the RAID system. However, since LADY changes the write speed of the SSD according to the characteristic of input write traffic, if some SSDs receive different write traffic from other ones in the same RAID, then response times across SSDs could be very different. This problem could be solved by implementing LADY in the RAID controller so that it manages multiple SSDs together. Coordinating multiple SSDs in the RAID controller to reduce response time variations is not new and has been studied intensively by [23, 24]. Moreover, since LADY has no dependencies with other SSD firmware modules, it can be easily implemented in the RAID controller.

6 Experimental Results

In order to evaluate LADY, we first carried out a set of evaluations with trace-driven simulators. We also implemented the prototype of LADY in the Linux kernel to assess its feasibility in real-world environments.

6.1 Experiments with Trace-Driven Simulators

Experimental Settings: The NAND flash was based on 2-bit MLC flash, and a block was composed of 64 4 KB pages. The page read and program times were 50 μs and 600 μs, respectively, and the block erasure time was 2 ms. The number of P/E cycles was set to 3K, but it changed depending on idle times and the program/erase mode. The NAND program time was also changed according to the program/erase mode chosen. The SSD lifetime was 5 years.

We performed our evaluations using two types of trace-driven simulators: an SSD simulator and a dynamic throttling (DT) simulator. The SSD simulator was based on a DiskSim-based SSD simulator with four channels [22]. Because of its rich functionalities, however, it exhibited a slow simulation speed. For this reason, we used the SSD simulator to collect firmware-level I/O traces sent to NAND flash, which included all of the I/O requests both from SSD firmware and from a host system. The SSD simulator used the page-level FTL with a greedy garbage collection policy. We used a firmware-level trace as an input for our DT simulator. The DT simulator supported several throttling algorithms. Since the DT simulator did not simulate complicated FTL algorithms, it was much faster than the SSD simulator. Throttling algorithms worked independently of FTL algorithms; they reduced write performance and did not change the behaviors of FTL algorithms. Thus, our simulation method was effective enough to accurately evaluate throttling algorithms in a rapid manner.

Techniques	*DT*	$LADY_{\mathrm{RECOV}}$	$LADY_{\mathrm{ALL}}$
Artificial Throttling Delay	○	○	○
Self-Recovery Effect	×	○	○
Program/Erase Mode	×	×	○

TABLE 1: A summary of evaluated sub-techniques

We compared LADY with two existing SSD configurations, *NT* and *ST*. While *NT* was the SSD without write throttling, *ST* was the SSD with static throttling. We categorized LADY into three sub-techniques: *DT*, $LADY_{\mathrm{RECOV}}$, and $LADY_{\mathrm{ALL}}$, which are summarized in TABLE 1. All the three sub-techniques used the same write-traffic estimation and write performance regulation policies, which were explained in Section 5.1 and 5.3, respectively. *DT* throttled write performance by changing artificial delays without considering the effective lifetime of NAND flash. *DT* used 3K P/E cycles with the nominal NAND program speed. $LADY_{\mathrm{RECOV}}$ was the same as *DT*, except that it considered the self-recovery effect. $LADY_{\mathrm{RECOV}}$ always used the nominal NAND program speed. $LADY_{\mathrm{ALL}}$ was the same as $LADY_{\mathrm{RECOV}}$, but it exploited different program/erase modes. A default epoch length was set to 10 minutes, but for *Cyclic* it changed adapting to workloads.

We chose two enterprise traces, `proxy` and `proj` from the MSR-Cambridge benchmark [20] and used three production traces, `exchange`, `map`, and `msnfs`, from the MS-Production benchmark [21]. Table 2 summarizes the traces. Because of the limited duration of the traces, we performed our evaluations under the assumption that the same I/O pattern will be repeated for 5 years. Note that the maximum epoch length for *Cyclic* was smaller than half of the trace duration because we replayed the same traces repeatedly. The amount of written data per hour was different depending on the traces. `proxy` and `proj` exhibited low write traffic compared with `exchange`, `map`, and `msnfs`. The write amplification factor (WAF), which has a great effect on the size of write

Trace	Duration	Data written per hour (GB)	WAF	SSD capacity (GB)
`proxy`	1 week	4.94	1.93	32
`proj`	1 week	2.08	1.62	32
`exchange`	1 day	20.61	2.24	128
`map`	1 day	23.82	1.68	128
`msnfs`	6 hours	18.19	2.26	128

TABLE 2: A summary of traces used for evaluations

0018-9340 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2349517, IEEE Transactions on Computers

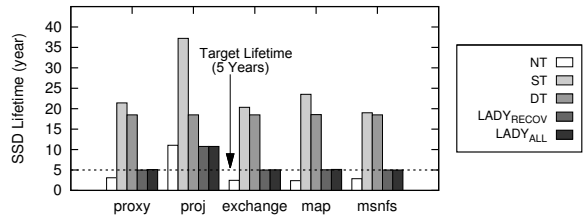


Fig. 12: A comparison of SSD lifetimes for five traces

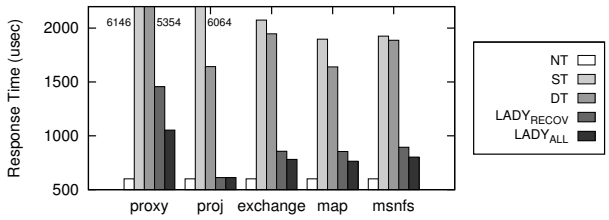


Fig. 13: A comparison of average write response times

traffic, ranged from 1.62 to 2.26. The SSD capacity was configured differently depending on the size of traces. For `proxy` and `proj`, the SSD capacity was 32 GB. For `exchange`, `map`, and `msnfs` with high write traffic, the capacity of the SSD was set to 128 GB.

Lifetime Analysis: Fig. 12 shows evaluation results on SSD lifetimes with five traces. *NT* cannot guarantee the required SSD lifetime for all the traces, except for `proj`. The write traffic of `proj` is not so heavy, so *NT* ensures the 5-year lifetime without throttling. *ST* and *DT* do not consider the effective lifetime of NAND flash, throttling write performance based on 3K P/E cycles. Since the P/E cycles of NAND flash increase because of the self-recovery effect, the effective SSD lifetimes with *ST* and *DT* are much longer than the required lifetime. It means that *ST* and *DT* excessively throttle write performance, which results in poor write performance over *LADY*$_{\mathrm{RECOV}}$ and *LADY*$_{\mathrm{ALL}}$. Unlike *ST*, *DT* dynamically decides the write speed in response to a changing workload, maximally utilizing 3K P/E cycles and exhibiting better performance than *ST*. *LADY*$_{\mathrm{RECOV}}$ takes advantage of the self-recovery effect, thus it throttles write performance so that the SSD lifetime is close to 5 years. *LADY*$_{\mathrm{ALL}}$ considers the self-recovery effect, and furthermore, it exploits the slow NAND program mode. Thus, it can guarantee 5-year lifetime with smaller performance penalties. We discuss the performance benefit of *LADY*$_{\mathrm{ALL}}$ over *LADY*$_{\mathrm{RECOV}}$ in detail later.

Table 3 analyzes the lifetimes of SSDs from the perspective of written data for `proj` and `proxy`. C_{ssd}^{3K} is the number of writable bytes based on 3K P/E cycles, whereas C_{ssd} is the number of writable bytes when the effective wearing of NAND flash is taken into account. W_{work} is the number of bytes written to the SSD for 5 years. *ST* and *DT* throttle write performance so that W_{work} becomes C_{ssd}^{3K} at the end of the target lifetime. In the case of *ST*, however, W_{work} is 43% and 11% smaller than C_{ssd}^{3K} for `proj` and `proxy`, respectively. This is because *ST* excessively throttles write performance under the assumption that write traffic is always heavy.

Trace	SSD configuration	C_{ssd}^{3K}(TB)	C_{ssd}(TB)	W_{work}(TB)
`proj`	*NT*	93.75	312.6	144.4
	ST		403.4	54.2
	DT		346.9	93.7
	LADY$_{\mathrm{RECOV}}$		312.8	141.0
	LADY$_{\mathrm{ALL}}$		312.7	141.3
`proxy`	*NT*	93.75	246.6	399.2
	ST		357.7	83.5
	DT		347.0	93.74
	LADY$_{\mathrm{RECOV}}$		271.9	271.9
	LADY$_{\mathrm{ALL}}$		338.2	331.8

TABLE 3: The amount of data written for 5 years for two traces, `proj` and `proxy`

Unlike *ST*, *DT* dynamically changes the write speed according to a workload, making W_{work} close to C_{ssd}^{3K}. *LADY*$_{\mathrm{RECOV}}$ fully utilizes the endurance improvement offered by the self-recovery effect, making W_{work} close to C_{ssd}. In the case of `proj`, write throttling is not performed in most cases because write traffic is not so heavy. Instead of merely delaying write requests with artificial delays, *LADY*$_{\mathrm{ALL}}$ exploits the slow NAND program mode, allowing us to write more data. For this reason, C_{ssd} of *LADY*$_{\mathrm{ALL}}$ is 24% larger than that of *LADY*$_{\mathrm{RECOV}}$ for `proxy`. In the case of `proj`, C_{ssd} of *LADY*$_{\mathrm{ALL}}$ is almost the same as that of *LADY*$_{\mathrm{RECOV}}$ because the slow program mode is rarely used.

Performance Analysis: To understand the effect of LADY on SSD performance, we measured the response time which was the average elapsed time to write individual pages. Fig. 13 shows our evaluation results. *NT* exhibits the best I/O response time, but it cannot guarantee the target lifetime. Both *LADY*$_{\mathrm{RECOV}}$ and *LADY*$_{\mathrm{ALL}}$ throttle write requests to meet the required lifetime, so their performance is worse than that of *NT*; *LADY*$_{\mathrm{RECOV}}$ and *LADY*$_{\mathrm{ALL}}$ exhibit 1.65x to 1.45x longer write response time than *NT*, on average, respectively. In the case of `proj`, *LADY*$_{\mathrm{RECOV}}$ and *LADY*$_{\mathrm{ALL}}$ do not reduce write performance because the required lifetime can be satisfied without throttling. *LADY*$_{\mathrm{RECOV}}$ achieves 2.32x better performance than *DT* on average. *LADY*$_{\mathrm{ALL}}$ outperforms *LADY*$_{\mathrm{RECOV}}$ and improves the overall write response time by 13.5%. *DT* exhibits 1.4x faster response time over *ST* on average. *DT* decides the

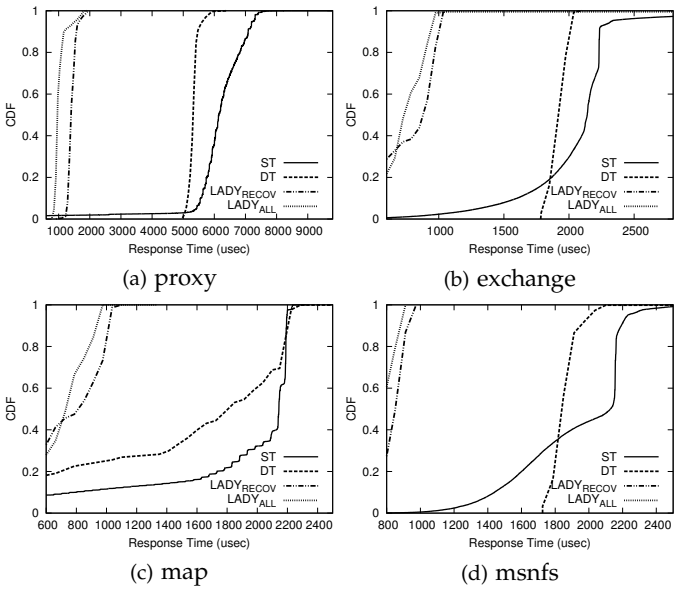


Fig. 14: CDFs of write response times

0018-9340 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2349517, IEEE Transactions on Computers

Prediction Accuracy (%)	proxy	proj	exchange	map	msnfs
Average	65.9	13.7	79.8	45.2	78.8
Weighted	73.0	17.8	74.6	52.5	78.7
Cyclic	76.0	16.1	71.9	59.3	77.5
Multiple-Expert	76.9	27.7	83.1	68.8	75.5

TABLE 4: Accuracy of write traffic prediction

epoch capacity periodically based on the remaining SSD lifetime and changes an artificial delay in response to future write traffic. *ST* neither considers the remaining SSD lifetime nor the characteristic of a workload. *ST* simply regulates write traffic by limiting the maximum write throughput, causing lots of unnecessary throttling.

We compared response time variations between different throttling algorithms. Fig. 14 shows the cumulative density functions (CDFs) of write response times for four traces. *ST* shows significant response time variations for all the traces because it forcibly stops writing if throttling is needed. *DT*, $LADY_{\mathrm{RECOV}}$, and $LADY_{\mathrm{ALL}}$ greatly reduce write response time variations over *ST* by evenly slowing down write performance.

Future Write-Traffic Prediction: We evaluated the accuracy of our future write-traffic prediction policy. Table 4 compares write-traffic prediction accuracies of four prediction policies: *Average*, *Weighted*, *Cyclic*, and *Multiple-Expert*. Here, the prediction accuracy is obtained by comparing the difference between the predicted write traffic and the actual one; the higher the number is the better the prediction accuracy is. The prediction accuracy is different depending on the prediction policies and the workloads, but *Multiple-Expert* exhibits the highest accuracy. Fig. 15 shows the effect of the write-traffic prediction accuracy on response time variations for `map`. If future write traffic is inaccurately estimated, LADY uselessly increases or decreases the write speed according to the mispredicted future write traffic, incurring variations on response times. As expected, *Multiple-Expert* showing the best accuracy exhibits the smallest response time variations.

Effect of Spare Capacity: We evaluated the performance of LADY with various spare capacities, 0%, 5%, 10%, and ∞. Fig. 16 shows the CDFs of write response times for `exchange` and `proxy`. For `exchange`, write response times deteriorate significantly with the spare capacities of 0% and 5%; LADY often stops writing data because of the depletion of the spare capacity. As the spare capacity increases to 10% and ∞, response time variations become smaller, avoiding strict capacity regulation. In the case of `proxy`, the exhaustion of spare capacity rarely occurs, thus the size of the spare capacity

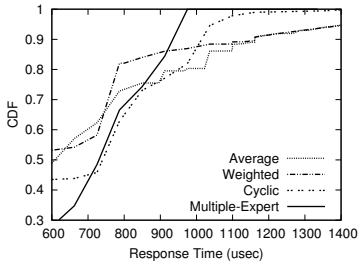
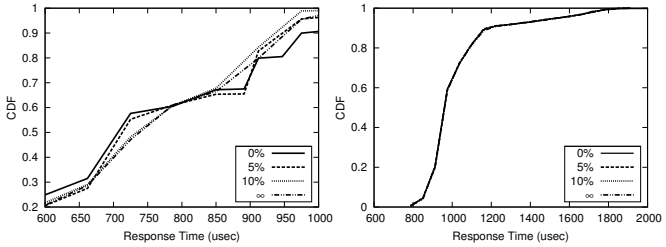


Fig. 15: CDFs of write response times with four different prediction policies for the *map* trace



(a) `exchange` (b) `proxy`

Fig. 16: The CDFs of write response times with various settings of spare capacity

does not affect overall write response times. ∞ shows the most stable write response times by borrowing unlimited spare capacity, but it cannot ensure the required lifetime. For `exchange`, LADY offers about 4.9 years lifetime.

Worst Case Analysis: LADY properly manages the SSD lifetime even in the worst-case scenario where the spare capacity is completely used up. To evaluate this, we synthesized the worst-case write traffic; the spare capacity often ran out and the overly-used spare capacity was not fully reclaimed in future epochs. Fig. 17 illustrates how LADY manages incoming write traffic when the spare capacity is nearly exhausted. Fig. 17 shows the sum of the default capacity and the remaining spare capacity, which is denoted by a total capacity. While the default capacity is maintained as 2.2 MB, the spare capacity is changing over time. Fig. 17 also shows the amount of data actually written. The spare capacity is nearly exhausted at around $70th$ second, but LADY properly handles write traffic so that more data than the total capacity are not written. This allows LADY to guarantee the required SSD lifetime. Because of this strict write-traffic regulation, the write performance is inevitably degraded; the write throughput is about 4 MB, but it is reduced to 2-3 MB after the spare capacity is exhausted. We conducted a more comprehensive analysis that included lifetime, performance, and response time variations. For more details, see Section 2 of Appendix.

Adaptability to Write Traffic with Variability: We evaluated how well LADY behaved under write traffic with variability. To this end, we combined I/O traces exhibiting different I/O patterns. We mixed `proxy` and `proj` into one trace (denoted by `proxy+proj`), and combined `exchange`, `map`, and `msnfs` into one trace (denoted by `exchange+map+msnfs`).

Figure 18(a) shows the write throughput (MB/s) and throttling delays (*m*sec) of `proxy+proj` for 0-1,400K seconds. The throttling delay is the length of time in-

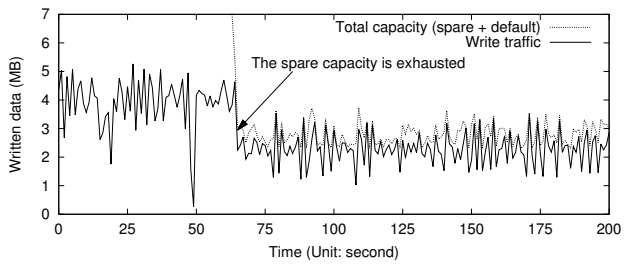
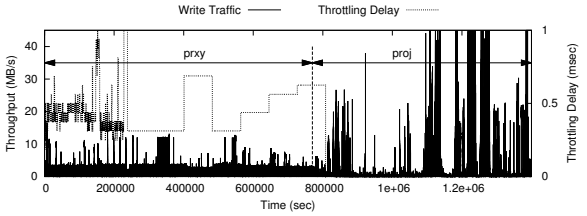


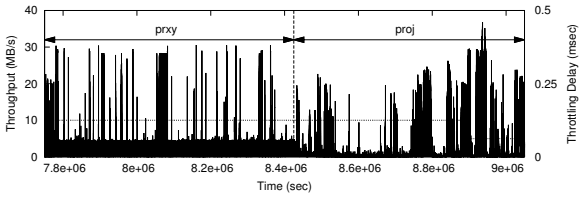
Fig. 17: An illustration of how LADY manages write traffic when the spare capacity is nearly exhausted

0018-9340 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2349517, IEEE Transactions on Computers

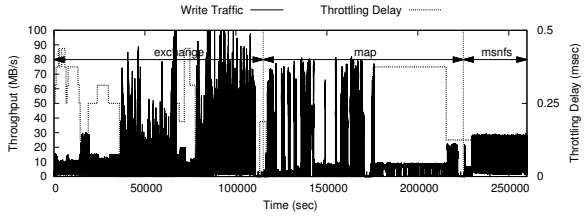


(a) 0-1,400K seconds

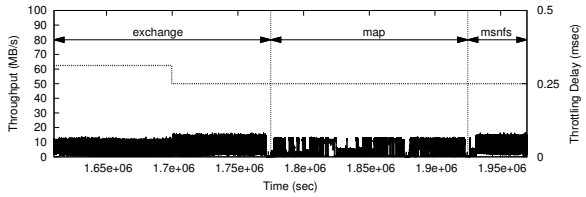


(b) 7,800K-9,000K seconds

Fig. 18: Results with `proxy+proj`



(a) 0-250K seconds



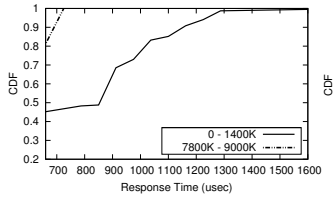
(b) 1610K-1970K seconds

Fig. 19: Results with `exchange+map+msnfs`

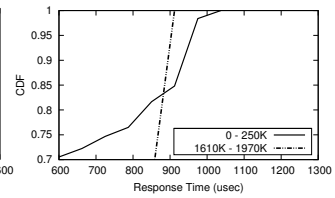
creased by a slower program mode and an artificial delay. The write traffic of `proxy` is much heavier than that of `proj`, while `proj` exhibits large fluctuations in write traffic. Initially, LADY changes throttling delays frequently; it makes wrong decisions because of the lack of the previous history. The write traffic rapidly drops after `proj` starts, so the throttling delay is reduced to 0 $m$sec. LADY is gradually adapted to the changing write traffic over time. Figure 18(b) shows the write throughput and throttling delays for 7,800K-9,000K seconds. The throttling delays become more stable than those for 0-1,400K seconds. Figure 20(a) compares the CDFs of write response times for 0-1,400K and 7,800K-9,000K seconds. For 7,800K-9,000K seconds, fluctuations in response times become smaller. In particular, the average write response time is greatly improved. LADY applies long throttling delays to `proxy` for 0-1,400K seconds, expecting that heavy write traffic would continue in the future. However, the write traffic of `proj` is much lower than that of `proxy`. This allows LADY to apply shorter delays to `proxy`, which improves write performance.

Figures 19 and 20(b) show write throughput, throttling delays, and CDFs for 0-250K and 1,610K-1,970K seconds of `exchange+map+msnfs`. LADY works similarly to `proxy+proj`; LADY often changes the length of throttling delays, but it becomes stable after obtaining sufficient information about previous write traffic.

Priority-Aware Dynamic Throttling: Finally, we evaluated priority-aware dynamic throttling, PA-DT. We executed two traces simultaneously while assigning different importance values. Two I/O trace combinations, `proxy+proj` and `exchange+map`, were used for the evaluation. The SSD capacity for `proxy+proj` was set to 32 GB and the SSD of 256 GB was used for `exchange+map`. The target SSD lifetime was 5 years. Figs. 21 (a) and (b) show the average write response times for `proxy+proj` and `exchange+map`, respectively, with different combinations of importance values ranging from 1 to 4. If two different traces have the same importance values (i.e., (1,1) and (4,4)), they exhibit the same write response times. However, as the difference between two importance values increases, the difference between their write response times increases as well. In



(a) `proxy+proj`



(b) `exchange+map+msnfs`

Fig. 20: A comparison of CDFs for `proxy+proj` and `exchange+map+msnfs`

the case of `proxy+proj`, if `proxy` has higher importance than `proj`, the average response time of `proj` is greatly increased. As listed in Table 2, the amount of data written by `proxy` is about 2.5 times larger than that of `proj`. If a short write time is assigned to `proxy` in spite of its heavy write traffic, `proj` must be throttled more intensively to offset the SSD lifetime excessively consumed by `proxy`. For `exchange+map` where two traces write the similar amount of data, the difference in write response times is not significant in comparison to `proxy+proj`. Finally, regarding the SSD lifetime, PA-DT guarantees the 5-year target lifetime.

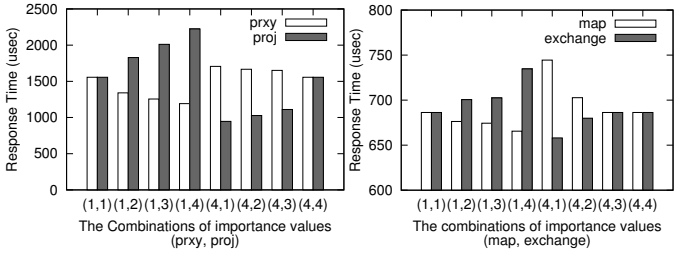
6.2 Experiments with Linux Prototype

In order to evaluate the feasibility of LADY in real-world environments, we implemented a *proof-of-concept* prototype of LADY in a PC server with 3.4 GHz i7 CPU, 12 GB RAM, and Samsung's 840 SSD. The operating system was Ubuntu 10.04 with the Linux kernel 2.6.32.29.

Experimental Settings: Since it was difficult to implement throttling algorithms directly in the firmware of a commercial SSD, we added an intermediate layer, called a throttling layer, between an I/O scheduler and an SSD. Fig. 22 shows a schematic description of our prototype. Throttling algorithms are implemented in the throttling layer. When a host system issues write requests to the SSD, the throttling layer intercepts and delivers them to the SSD. After receiving completion interrupts from the SSD, it puts them into a throttling queue, instead of delivering them to the I/O scheduler. After a throttling delay, it dequeues them from the throttling queue and delivers them to the I/O scheduler.

0018-9340 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2349517, IEEE Transactions on Computers



(a) `proxy+proj` (b) `exchange+map`

Fig. 21: The average response times for `proxy+proj` and `exchange+map`

In our implementation study, the effective wearing properties of NAND flash were not taken into account due to the limited internal information of the SSD. Instead, we focused on evaluating how effectively LADY throttled write performance in real environments. Considering that the effective wearing only had a great effect on the number of effective P/E cycles, the limited internal information was not serious obstacle in assessing the feasibility of our throttling algorithm.

The benchmark programs used for our evaluations included `TPC-C`, `bonnie++`, and `postmark`. Since it was infeasible to run real-world benchmarks for several years, we manually scaled down the target SSD lifetime T_{ssd} and the amount of writable data C_{ssd}. In particular, we executed multiple benchmark instances or threads, so as to understand whether incoming write traffic was effectively throttled or not even when independent write requests were issued. A detailed description of benchmarks and evaluation settings are noted in Table 5.

Performance/Lifetime Analysis: We implemented two throttling algorithms, static throttling and LADY in the throttling layer, which are denoted by *ST* and *LADY*, respectively. We also have evaluated the SSD without write throttling, which is denoted by *NT*.

Fig. 23(a) shows the SSD lifetimes of *NT*, *ST*, and *LADY*. T_{ssd} is set to 1 hour. *NT* cannot ensure 1-hour lifetime, while *LADY* guarantees the target lifetime. For all the benchmarks, the amount of written data is smaller than C_{ssd}. *ST* also ensures the target lifetime, but it cannot fully utilize the available SSD endurance because of its excessive throttling, which results in great performance degradation. As depicted in Fig. 23(b), *LADY* reduces write response times by 72%, 16%, and 45% over *ST* for `TPC-C`, `Bonnie++`, and `Postmark`, respectively. It must be noted that *LADY* guarantees the target lifetime in the environment where multiple instances of

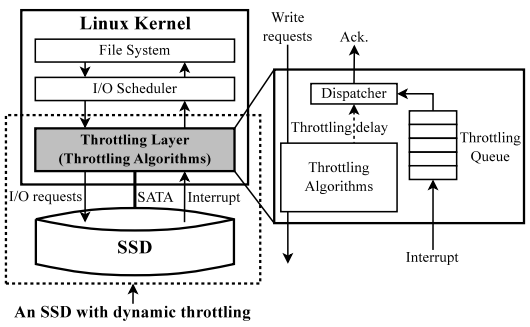


Fig. 22: A schematic description of our prototype

Benchmark	Description	T_{ssd}	C_{ssd}
`TPC-C`	An on-line transaction processing (OLTP) benchmark for transaction processing systems. 40 users run transactions simultaneously.	1 hour	10.8 GB
`Bonnie++`	It creates and deletes files in sequential and random orders, while performing different types of file system operations. Five bonnie++ programs are executed at the same time.	1 hour	72 GB
`Postmark`	It emulates a workload of electronic mail and netnews services. Eight postmark programs are executed concurrently.	1 hour	25 GB

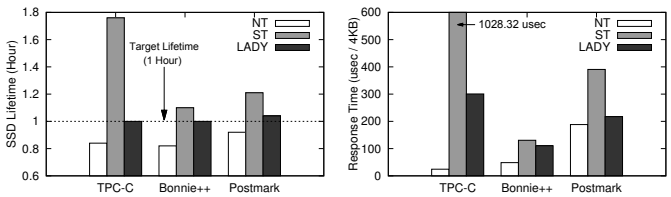
TABLE 5: A summary of benchmark programs

benchmark programs simultaneously access the SSD. As noted in Table. 5, five `Bonnie++` instances and eight `Postmark` instances write data to the SSD at the same time, but *LADY* properly regulates write traffic. The similar results are also observed in `TPC-C` where 40 users run transactions concurrently.

Fig. 24 illustrates the write traffic with three throttling policies, *NT*, *ST*, and *LADY*, when `TPC-C` is running. *NT* exhibits the best performance, but the SSD is completely worn out at 3,024 seconds. By limiting the maximum write throughput of the SSD to 3 MB/s (i.e., 10.8 GB / 3,600 seconds), *ST* extends the SSD lifetime to more than 1 hour. However, the SSD performance is excessively regulated when write requests are intensively issued; on the other hand, it is never throttled if there are only few write requests. For this reason, *ST* incurs significant write response variations, underutilizing the available SSD endurance. Unlike *ST*, *LADY* predicts the future write traffic and then changes write speeds as evenly as possible. Thus, the SSD is worn out at the target lifetime with small variations on write response times.

7 Related Work

As the endurance of flash memory is continuously reduced, several endurance enhancement techniques that aggressively reduce the amount of data written to SSDs have been proposed. Data de-duplication [25] and data compression [26] are representative ones. These techniques are useful in improving the lifetime of SSDs, but they have a limitation in that none of them guarantee the SSD lifetime. J. Guerra *et al*. presented a storage configuration strategy that effectively combines flash-based SSDs with HDDs for multi-tier storage systems [6]. Their technique decided an adequate mix of storage devices that requires the low capital and operating cost, satisfying performance requirements with the minimum power consumption. Although this work considered various aspects required for designing multi-tier storage systems, they did not take into account the SSD lifetime having a great effect on the overall storage cost,



(a) SSD lifetimes (b) Average write response times

Fig. 23: Experimental results with the prototype of LADY in the Linux operating system

0018-9340 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2014.2349517, IEEE Transactions on Computers

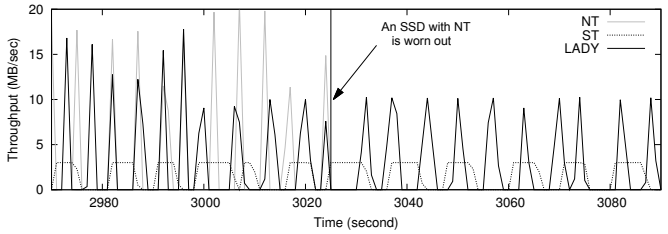


Fig. 24: An illustration of the write traffic of three different throttling policies with TPC-C

along with a trade-off between SSD performance and lifetime. Exploiting the recovery effect of NAND flash has received considerable attention. Mohan *et al*. first investigated the effect of the damage recovery on the SSD lifetime [8]. They showed that the endurance of NAND flash was durable enough even for I/O intensive enterprise servers due to its recovery ability. Their investigations were limited to 90 nm NAND flash which exhibits good write endurance. They also did not exploit the recovery effect in ensuring the SSD lifetime.

Differences from our previous study: We presented the basic idea of dynamic throttling in [1]. Our previously proposed throttling technique was greatly improved in several aspects. First, while our earlier work only considered the self-recovery effect of NAND flash, this study took into account the trade-off between NAND program speed and write endurance in addition to the self-recovery effect. This improved the write performance by 13.5% on average over our previous technique. Second, we improved the write-traffic prediction technique which estimated future write traffic more accurately, lowering response time variations. Third, we proposed priority-aware dynamic throttling which managed the performance and lifetime of SSDs according to the importance of enterprise services. Finally, we implemented a proof-of-concept prototype of LADY in the Linux kernel and evaluated its feasibility using various benchmark programs, including TPC-C, bonnie++, and postmark.

8 Conclusions

In this paper, we proposed the effective lifetime-aware dynamic throttling technique, called LADY, to overcome the lifetime problems of flash-based SSDs in enterprise environments. LADY throttled write performance so that the required lifetime was satisfied. In order to guarantee the SSD lifetime with small throttling penalties, LADY exploited the effective wearing characteristics of NAND flash. Our experimental results showed that LADY guaranteed a lifetime warranty, while achieving better write response times and smaller variations on response times over the static throttling technique.

References

[1] S. Lee, T. Kim, K. Kim, and J. Kim, "Lifetime Management of Flash-Based SSDs Using Recovery-Aware Dynamic Throttling," in *Proceedings of the USENIX conference on File and Storage Technologies*, 2012.

[2] E. Pinheiro, W.-D. Weber, and L.-A Barroso, "Failure Trends in a Large Disk Drive Population," in *Proceedings of the USENIX conference on File and Storage Technologies*, 2007.

[3] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating Server Storage to SSDs: Analysis of Tradeoffs," in *Proceedings of the ACM European conference on Computer systems* , 2009.

[4] A. Opitz, H. König, S. Szamlewska, "What Does Grid Computing Cost?," *Journal of Grid Computing*, 2008.

[5] B. You and et. al, "A High Performance Co-design of 26 nm 64 Gb MLC NAND Flash Memory using the Dedicated NAND Flash Controller," *Journal of Semiconductor Technology and Science*, vol. 11, no. 2, 2011.

[6] C. Black, "24 Months of Intel SSDs... What We've Learned about MLC in the Enterprise...," *Intel Open Port IT Community*, 2011.

[7] J. Jeong, S. Hahn, S. Lee, and J. Kim, "Improving NAND Endurance by Dynamic Program and Erase Scaling," in *Proceedings of the USENIX Workshop on Hot Topics in Storage and File Systems*, 2013.

[8] V. Mohan, T. Siddiqua, S. Gurumurthi, and M. Stan, "How I Learned to Stop Worrying and Love Flash Endurance," in *Proceedings of the Workshop on Hot Topics in Storage and File Systems*, 2010.

[9] N. Mielke, H. Belgal, A. Fazio, Q. Meng, and N. Righos, "Recovery Effects in the Distributed Cycling of Flash Memories," in *Proceedings of the IEEE International Reliability Physics Symposium*, 2006.

[10] Q. Wu, G. Dong, and T. Zhang, "Exploiting Heat-Accelerated Flash Memory Wear-Out Recovery to Enable Self-Healing SSDs," in *Proceedings of the Workshop on Hot Topics in Storage and File Systems*, 2011.

[11] Y. Pan, G. Dong, and T. Zhang, "Exploiting Memory Device Wear-Out Dynamics to Improve NAND Flash Memory System Performance," in *Proceedings of the USENIX Conference on File and Storage Technologies*, 2011.

[12] Dell Inc., "Solid State Drive (SSD) FAQ," 2011.

[13] SMART Modular Technologies, "XceedIOPS SATA SSD," 2012.

[14] N. Cesa-Bianchi, Y. Freund, D. P. Helmhold, D. Haussler, R. E. Schapire, M. K. Warmuth, "How to Use Expert Advice," in *Proceedings of the ACM Symposium on the Theory of Computing*, 1993.

[15] G. Dhiman and T. Rosing, "System-Level Power Management using Online Learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 676-689, 2009.

[16] S. Yoo and C. Park, "Low Power Mobile Storage: SSD Case Study," *Energy-Aware System Design*, pp. 223-246, 2011.

[17] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Workload Analysis and Demand Prediction of Enterprise Data Center Applications," in *Proceedings of the IEEE International Symposium on Workload Characterization*, 2007.

[18] M. Carey, R. Jauhari, and M. Livny, "Priority in DBMS resource scheduling", in *Proceeding of the International Conference on Very Large Data Bases*, pp. 397–410. 1989.

[19] D. McWherter, B. Schroeder, A. Ailamaki, and M. Harchol-Balter, "Improving Preemptive Prioritization via Statistical Characterization of OLTP Locking", in *Proceeding of International Conference on Data Engineering*, pp. 446-457, 2005.

[20] D. Narayanan, A. Donnelly, and A. Rowstron, "Write Off-Loading: Practical Power Management for Enterprise Storage," in *Proceedings of the USENIX Conference on File and Storage Technologies*, 2008.

[21] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of Storage Workload Traces from Production Windows Servers," in *Proceedings of the International Symposium on Workload Characterization*, 2008.

[22] N. Agrawal, V. Prabhakaran, and T. Wobber, "Design Tradeoffs for SSD Performance," in *Proceedings of the USENIX Annual Technical Conference*, 2008.

[23] Y. Kim, J. Lee, S. Oral, D. A. Dillow, F. Wang, and G. M. Shipman, "Coordinating Garbage Collection for Arrays of Solid-state Drives", *IEEE Transactions on Computers*, Vol. 63, No. 4, pp. 888-901, April 2014.

[24] Y. Kim, S. Oral, G. M. Shipman, and J. Lee, "Harmonia: A Globally Coordinated Garbage Collector for Arrays of Solid-state Drives", in *Proceedings of the Symposium on Massive Storage Systems and Technologies*, 2011.

[25] F. Chen, T. Luo, and X. Zhang, "CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory Based Solid State Drives," in *Proceedings of the USENIX Conference on File and Storage Technologies*, 2011.

[26] T. Park and J.-S. Kim, "Compression Support for Flash Translation Layer," in *Proceedings of the International Workshop on Software Support for Portable Storage*, 2010.

Sungjin Lee received the BE degree in electrical engineering from Korea University in 2005 and the MS and PhD degrees in computer science and engineering from the Seoul National University in 2007 and 2013, respectively. He is currently working as a postdoctoral associate in the Computer Science and Artificial Intelligence Laboratory at the Massachusetts Institute of Technology. His research interests include storage systems, operating systems, and embedded software.

Jihong Kim received the BS degree in computer science and statistics from Seoul National University (SNU), Korea, in 1986, and the MS and PhD degrees in computer science and engineering from the University of Washington, Seattle, in 1988 and 1995, respectively. Before joining SNU in 1997, he was a technical staff member at the DSPS R&D Center of Texas Instruments in Dallas, Texas. He is currently a professor at the School of Computer Science and Engineering, Seoul National University. His research interests include embedded software, low-power systems, computer architecture, and storage systems. He is a member of the IEEE.

0018-9340 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.