Lifetime Management of Flash-Based SSDs Using Recovery-Aware Dynamic Throttling

Sungjin Lee, Taejin Kim, Kyungho Kim*, and Jihong Kim

Seoul National University, Korea
{*chamdoo, taejin1999, jihong*}*@davinci.snu.ac.kr*
Samsung Electronics, Korea
kyungho21.kim@samsung.com

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 limited and unpredictable lifetime of SSDs remains to be a serious obstacle to wider adoption of SSDs in enterprise systems. In this paper, we propose a novel recovery-aware dynamic throttling technique, called READY, which guarantees the SSD lifetime required by the enterprise market while exploiting the self-recovery effect of floating-gate transistors. Unlike a static throttling technique, the proposed technique makes throttling decisions dynamically based on the predicted future write demand of a workload so that the required SSD lifetime can be guaranteed with less performance degradation. The proposed READY technique also considers the self-recovery effect of floating-gate transistors which improves the endurance of SSDs, enabling to guarantee the required lifetime with less write throttling. Our experimental results show that the proposed READY technique can improve write performance by 4.4x with less variations on the write time over the existing static throttling technique while guaranteeing the required SSD lifetime.

1 Introduction

NAND flash memory has been widely used in mobile embedded systems like mobile phones, MP3 players, and laptop computers because of its low-power consumption, high mobility, and high performance. Recently, as the price-per-byte of NAND flash memory is falling, NAND flash-based solid-state drives (SSDs) are increasingly popular in enterprise servers as well, replacing hard disk drives. However, the poor write endurance of NAND flash memory is still regarded as a main barrier for a wide adoption of flash-based SSDs in the enterprise market. In order for SSDs to be broadly adopted in the enterprise environment, two key problems on the SSD lifetime need to be addressed properly.

The first problem is that the *endurance* of flash devices is rapidly decreasing. The endurance of flash-based SSDs is directly related to the number of program/erase (P/E) cycles allowed to memory cells, which are made from floating-gate transistors. Due to the charge trapping characteristic of a floating-gate transistor [1, 2], NAND flash memory is gradually impaired as the number of P/E cycles increases and becomes unreliable beyond a maximum number of P/E cycles. As the semiconductor process is scaled down and with multi-level cell (MLC) technology, the endurance of a floating-gate transistor is significantly degraded. For example, the maximum number of P/E cycles of single-level cell (SLC) flash memory fabricated in a 70 nm process is about 100K P/E cycles. For 2-bit MLC flash memory fabricated in the 2x nm process, the maximum number of P/E cycles decreases to 3K P/E cycles [3, 4, 5] while, for 3-bit MLC flash memory, this number is only a few hundred cycles [6].

The second problem is the *unpredictable lifetime* of flash devices. Since the endurance of SSDs is dependent upon the number of P/E cycles, the SSD lifetime is determined by extra data written by garbage collection and wear-leveling as well as by the number of bytes written by applications. This means that, unlike HDDs, the SSD lifetime is a function of a workload. Therefore, even if the endurance of SSDs seems sufficient, the lifetime of SSDs strongly depends on the write intensiveness of the workload. For example, SSDs may achieve the required lifetime if a small number of write requests are required from applications. On the other hand, the same SSDs will fail much earlier if they are used in a write intensive environment. In particular, as cost-effective MLC-based SSDs are becoming popular in the enterprise market where write requests are intensive [7, 8], it is a challenge to guarantee a minimum SSD lifetime of 3-5 years, which enterprise customers often require [9].

In this paper, we overcome these technical difficulties by proposing a **re**covery-**a**ware **dy**namic throttling technique, called READY. A basic concept of READY is to throttle write performance by adding throttling delays to write requests, so as to guarantee the required SSD lifetime. With dynamic throttling, the IOPS and bandwidth of SSDs is reduced to a certain extent. From the application prospective, applications' execution times are increased as if they run on top of a slower device. As a result, the amount of write traffic sent to a storage device is reduced, lessening the wearing-rate of SSDs.

The dynamic throttling technique inevitably reduces the overall write performance. In order to mitigate performance degradation, we carefully determine throttling delay by predicting future write demands and distribute the predicted delay over the entire SSD lifetime so that better write response time can be obtained with less variations on the response time. In addition, the proposed dynamic throttling technique takes into account the self-recovery characteristic of a floating-gate transistor. Because of the physical characteristics of NAND flash memory, the damage caused by repetitive P/E cycles can be partially recovered during the idle period between two consecutive P/E cycles, improving the endurance of a floating-gate transistor [1, 2, 10, 11, 12]. By considering the endurance improvement by the self-recovery effect, the proposed READY technique can be more optimistic on the total number of data written, thus employing a smaller throttling delay. Our evaluation results show that the proposed throttling technique improves the average write response time by 4.4x with less variations over an existing static throttling technique while guaranteeing the SSD lifetime.

This paper is organized as follows. In Section 2, we briefly explain the endurance characteristics of NAND flash memory. Section 3 describes the proposed recovery-aware dynamic throttling technique in detail. In Section 4, we evaluate the effectiveness of the proposed recovery-aware dynamic throttling technique using enterprise benchmarks. Section 5 describes related work on improving the SSD endurance. Finally, Section 6 concludes with summary and directions for future work.

2 Endurance Characteristics of Flash Memory

In NAND flash memory, program/erase (P/E) operations inevitably cause damage to floating-gate transistors, reducing the overall endurance of memory cells. At the device level, memory 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, which indicates a logical bit value of a cell, and the cell becomes unreliable when the threshold voltage is higher than a certain voltage margin, e.g., 0.65V for MLC flash memory [1]. According to [1, 10], the increase, δV_{trap}, in a threshold voltage because of charge trapping approximately scales with P/E cycles in a power-law fashion as follows:

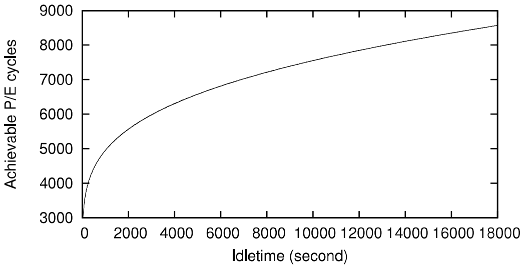


Figure 1: The achievable number of P/E cycles depending on the different idle times.

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

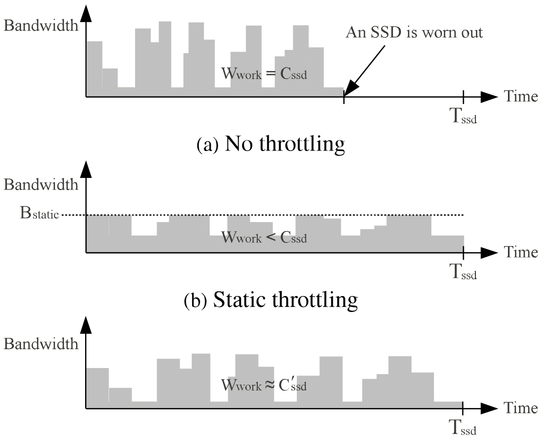
where N is the number of P/E cycles. A_{it} and B_{ot} are constant and set to 2.97×10^{-3} and 2.0×10^{-2}, respectively. Usually, NAND flash memory vendors do not reveal important parameters for their recent products. Thus, in this work, A_{it} and B_{ot} for 20 nm MLC flash memory are obtained by 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.

A floating-gate transistor also 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 the idle time between P/E cycles on the same cell, and its effect in general increases as the logarithm of the idle time, i.e., detrapping $\propto \ln(\mathrm{t})$, where t is the length of the idle time. According to [1, 10, 13], the decrease, δV_{detrap}, in a threshold voltage due to charge detrapping can be expressed as follows:

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

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

Besides the length of the idle time, there are other factors that affect the cell recovery, such as an external temperature and a programmed threshold voltage. In this work, the temperature is assumed to be a room temperature 25°C because the external ambient temperature of a storage device is typically maintained at the room temperature [14]. The programmed threshold voltage is not taken into account in this study because its effect on the damage recovery is relatively negligible.



(a) No throttling

(b) Static throttling

(c) Recovery-aware dynamic throttling

Figure 2: A comparison of three difference throttling policies: no throttling, static throttling, and recovery-aware dynamic throttling.

According to [1], the effective increase, δV_{th}, in a threshold voltage can be expressed as follows:

$$\delta V_{th} = \delta V_{trap} - \delta V_{detrap}. \tag{3}$$

Based on Eq. (3), we have plotted the achievable P/E cycles of 20 nm MLC flash memory in Figure 1, depending on the average idle times between two consecutive P/E cycles on the same block. The maximum P/E cycles without the recovery effect are 3K. As expected, the achievable P/E cycles are gradually increased in proportional to the length of the idle time. Note that recent studies that measured the effective P/E cycles of real NAND flash parts also reported that the endurance of NAND flash memory is higher than P/E cycles in datasheets [15, 16].

The detrapping phenomenon of a floating-gate transistor has a positive effect on improving the endurance (or increasing P/E cycles) of flash-based SSDs. However, most studies use a fixed number of P/E cycles, e.g., 3K P/E cycles, provided by flash manufacturers as a primary factor to manage the lifetime of SSDs. Therefore, the benefit of the damage recovery is not fully utilized. Unlike other studies, our recovery-aware dynamic throttling technique takes advantage of the self-recovery effect in managing the lifetime of SSDs to lessen the performance penalty caused by write throttling.

3 Recovery-Aware Dynamic Throttling

In this section, we describe the proposed recovery-aware dynamic throttling technique. We first introduce the need for dynamic throttling in flash-based SSDs using a simple motivational example and then explain the main functions of the proposed throttling technique in detail.

3.1 Basic Idea

Figure 2 shows a motivational example of dynamic throttling in SSDs. The maximum number, C_{ssd}, of bytes that can be written to the SSD is proportional to the SSD capacity and the number of P/E cycles allowed to each block. C_{ssd} is thus easily calculated with the following equation: SSD capacity × P/E cycles [17]. For example, if the SSD capacity is 128 GB and the number of P/E cycles is 3K, C_{ssd} becomes 375 TB. Suppose that a lifetime, T_{ssd}, to be guaranteed is 1.5768 $\cdot 10^8$ seconds, i.e., 5 years. In the example of Figure 2(a) which does not use write throttling, the required lifetime cannot be satisfied because the number, W_{work}, of bytes written to the SSD exceeds C_{ssd} before T_{ssd}.

To ensure the lifetime warranty of the SSD, some SSD vendors recently have started to adopt *static throttling* [18, 19], which is shown in Figure 2(b). Static throttling guarantees the required lifetime by limiting the maximum bandwidth of the SSD to a certain fixed value, which is denoted by B_{static}. Static throttling determines the value of B_{static} based on the assumption of the worst case scenario where the number of bytes written per second is always larger than C_{ssd}/T_{ssd}. In this case, B_{static} must be fixed to C_{ssd}/T_{ssd} to ensure the required lifetime. The drawback of this approach is that it is likely to underutilize the maximum endurance of the SSD, i.e., $W_{work} < C_{ssd}$ at T_{ssd}, because of its assumption that the SSD must provide the B_{static} bandwidth although actual workloads may not be that intensive all the time. In addition, due to this conservative assumption, the I/O response time is degraded with static throttling.

In order to overcome the limitation of the static throttling technique, we propose a recover-aware dynamic throttling technique, READY, which is depicted in Figure 2(c). By dynamically throttling write requests according to the characteristics of a workload and the remaining SSD lifetime, the proposed READY technique fully utilizes the given endurance of the SSD up to the maximum, while minimizing performance degradation. READY is also aware of the endurance improvement by the self-recovery characteristic of memory cells. Therefore, the data that can be written to the SSD increase by ΔC_{ssd}, so the maximum number of writable bytes becomes $C^{'}_{ssd}$ (= $C_{ssd} + \Delta C_{ssd}$). This allows us to guarantee the required lifetime with less throttling overheads.

In designing a dynamic throttling policy, we focus on two aspects of the design requirements of SSDs. The first is to determine a throttling delay as low as possible so that W_{work} is close to $C^{'}_{ssd}$ at the time of T_{ssd}. If $W_{work} = C^{'}_{ssd}$ before T_{ssd}, we cannot guarantee the required lifetime as shown in Figure 2(a). If $W_{work} < C^{'}_{ssd}$ at T_{ssd}, write performance significantly deteriorates, underutilizing the available endurance of the SSD

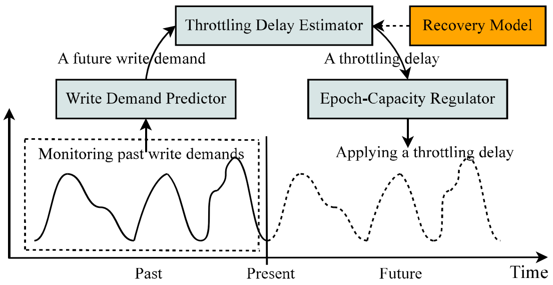


Figure 3: Three main functions of READY.

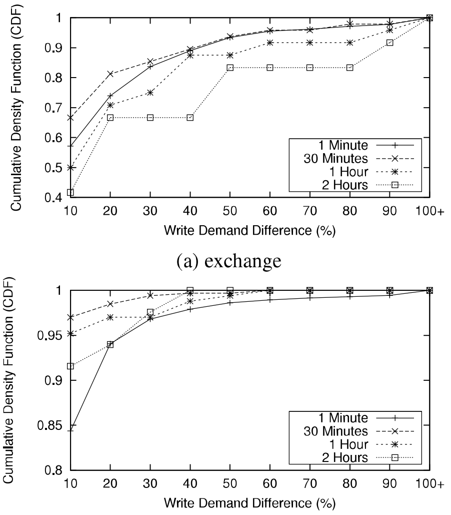
like static throttling as depicted in Figure 2(b). The second is to distribute a throttling delay over every write request as evenly as possible. Otherwise, response time variations can be large, thus lowering the quality of the user experience significantly.

To effectively deal with these design issues, the proposed dynamic throttling technique has been designed with three main functions as shown in Figure 3. The write demand predictor is in charge of predicting future write demands, which indicate the number of bytes that is written to SSDs, by monitoring previous write demands. Once the future demand for writes has been predicted, the throttling delay estimator determines a throttling delay by considering both the future write demand and the remaining lifetime of SSDs. The epoch-capacity regulator throttles write performance by applying a throttling delay to each write request so that the target SSD lifetime will be reached.

3.2 Estimation of Future Write Demands

In designing a dynamic throttling policy, it is important to estimate the number of bytes that will be written to the SSD in advance because the SSD performance must be throttled properly if the write demand is expected to be too high. The role of the write demand predictor is to predict future write demands by monitoring the previous write demands of a workload.

For this purpose, in READY, the entire lifetime, T_{ssd}, of the SSD is divided into *epochs*. At the beginning of each epoch, the write demand predictor estimates the number of bytes that is to be written during the epoch based on the number of bytes actually written to the SSD during the latest epoch. If the data of w_{i-1} have been written during the $(i-1)$-th epoch, the write demand predictor predicts that the same number of bytes will be written to the SSD during the i-th epoch. That is, $w_i \approx w_{i-1}$. This approach is motivated by previous observations [20] that showed that enterprise workloads often exhibit cyclic behavior with periods between several minutes and several days. Although that work did not address I/O demands in storage devices, it showed that a strong cyclical behavior is frequently observed in enterprise applications. This means that if the length of an epoch is properly decided to include the cyclic period of a workload, the write demand observed in the latest epoch can be used as a factor that indicates future write demands.



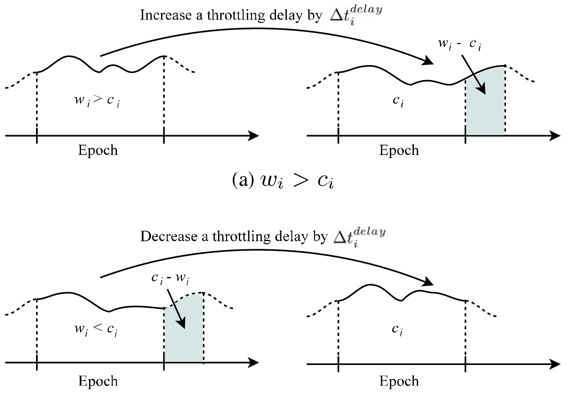
(a) exchange

(b) proxy

Figure 4: Write demand differences with different epoch lengths for `exchange` and `proxy`.

To confirm our hypothesis, we have analyzed the characteristic of write demands using enterprise traces. We have compared the difference in write demands between two consecutive epochs while varying the length of an epoch from 1 minute to 2 hours. Our analysis has been performed with several enterprise traces from the MSR-Cambridge and MS-Production traces [21, 22]. Figure 4 shows our investigation results for the two traces, `proxy` and `exchange`. Here, the X-axis represents the write demand difference between the predicted write demand and the actual one in percentage. For example, if the predicted demand is 100 MB and the actual one is 95 MB, the write demand difference between them is 5%. The Y-axis is the cumulative density function (CDF) of the write demand difference of the epochs. The smaller the difference, the better the accuracy of future write demand prediction is.

As shown in Figure 4, when the length of an epoch is decided properly, it is possible to achieve high accuracy in predicting future write demands. In the case of `exchange`, for about 85% of the epochs, the write demand difference of less than 30% is obtained with the epoch length of 30 minutes. For `proxy`, the epoch length of 30 minutes shows the best accuracy in estimating future write demands. This result clearly shows that the epoch-based write demand prediction is useful to estimate future write demands. Note that other methods, such as a moving average, are also applicable for esti-


Increase a throttling delay by Δt_i^{delay}
w_i - c_i
$w_i > c_i$
c_i
Epoch
Epoch
Decrease a throttling delay by Δt_i^{delay}
c_i - w_i
$w_i < c_i$
c_i
Epoch
Epoch

(a) $w_i > c_i$

(b) $w_i < c_i$

Figure 5: A change in a throttling delay.

mating future write demands.

Since the best epoch length may be different depending on a workload and its characteristic (which changes with time), the proposed READY technique selects the epoch length dynamically adapting to a changing workload. We will discuss this issue in Section 3.5.

3.3 Calculation of Throttling Delay

The throttling delay estimator adaptively changes a throttling delay at every epoch by monitoring the write demand and the remaining SSD lifetime. At the first epoch, i.e., the 0-th epoch, a throttling delay, t_0^{delay}, is set to 0. Then, at the beginning of each i-th epoch, the delay estimator increases or decreases a throttling delay, t_i^{delay}, based on the expected write demand and the capacity of each epoch. The expected write demand indicates the number, w_i, of bytes that is supposed to be written during the i-th epoch. The capacity of an epoch is the number, c_i, of bytes allowed to be written during the i-th epoch. In this work, w_i is equal to the number, w_{i-1}, of bytes written during the $(i-1)$-th epoch under the assumption of $w_i \approx w_{i-1}$. The capacity, c_i, of the i-th epoch is determined by dividing the remaining capacity, C_r, of the SSD by the number of remaining epochs. Here, the remaining capacity, C_r, represents the number of bytes that can be written to the SSD until it becomes unreliable.

If w_i is equal to c_i, we don't need to change a throttling delay for the i-th epoch. Therefore, t_i^{delay} is the same as t_{i-1}^{delay}, which is the throttling delay of the $(i-1)$-th epoch. However, if w_i is larger than c_i as shown in Figure 5(a), it is necessary to increase a throttling delay because the data to be written to the SSD are expected to be larger than the capacity allocated to the epoch. The increase, Δt_i^{delay}, in a throttling delay can be expressed as follows:

$$\Delta t_i^{delay} = t_{epoch} \cdot \left(\frac{w_i}{c_i} - 1\right) \Big/ n \quad \text{if } w_i > c_i, \qquad (4)$$

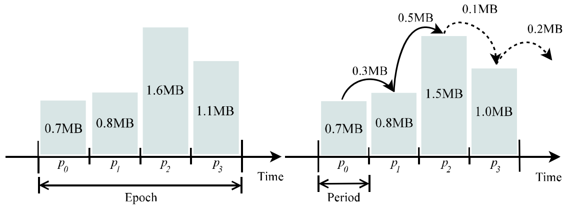
where n is the number of pages allowed to be written to the SSD during the i-th epoch, i.e., c_i/page size, and t_{epoch} is the epoch length. To make the data written during the i-th epoch equal to c_i, $(w_i - c_i)$ of the data must be delayed to the next epoch as shown in Figure 5(a). The total time required to delay $(w_i - c_i)$ of the data can be approximated as $t_{epoch} \cdot (w_i/c_i - 1)$. In our dynamic throttling policy, a throttling delay is equally distributed to each page write (refer to Section 3.4), so Δt_i^{delay} can be obtained by dividing the total throttling delay by n. Finally, a throttling delay, t_i^{delay}, for the i-th epoch is determined as follows: $t_i^{delay} = t_{i-1}^{delay} + \Delta t_i^{delay}$.

If w_i is smaller than c_i as shown in Figure 5(b), it means that the write requests were not intensive enough to wear out the device before the required lifetime or they were too throttled during the previous epoch. Therefore, the throttling delay may be reduced so that more data can be written to the SSD. The decrease, Δt_i^{delay}, in a throttling delay can be expressed as follows:

$$\Delta t_i^{delay} = t_{epoch} \cdot \left(\frac{c_i}{w_i} - 1\right) \Big/ n \quad \text{if } w_i < c_i. \qquad (5)$$

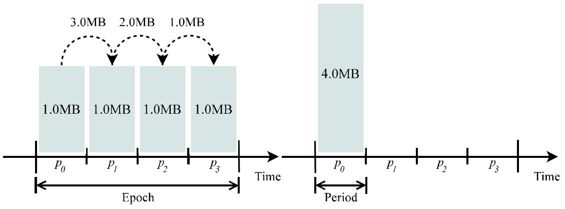
To increase the number of bytes to be written to the SSD by $(c_i - w_i)$ during the i-th epoch, a throttling delay, t_i^{delay}, for the i-th epoch is reduced by Δt_i^{delay} as follows: $t_i^{delay} = t_{i-1}^{delay} - \Delta t_i^{delay}$. In the case of $t_{i-1}^{delay} < \Delta t_i^{delay}$, t_i^{delay} is 0. This means that it is not necessary to apply a throttling delay because the required lifetime can be guaranteed without write throttling.

Until now, we assumed that the number of P/E cycles is fixed to a certain number. The achievable P/E cycles, however, can be increased depending on the amount of the idle time between two consecutive P/E cycles in a certain block because of the self-recovery effect of memory cells. In order to exploit this endurance improvement, the throttling delay estimator first estimates the number of achievable P/E cycles at the beginning of each epoch, using the damage and recovery model mentioned in Section 2. In this work, the number of achievable P/E cycles is estimated, using the average idle time of every block in the SSD. The idle time is actually somewhat different among blocks. However, this difference is not significant because the wear-leveler of the SSD makes the P/E cycles of all available blocks evenly distributed. Therefore, the average idle time can be used as a useful parameter to estimate the overall endurance improvement of the SSD. The estimator then calculates the remaining capacity, C_r, based on the achievable P/E cycles and distributes it to the remaining epochs. Since the number of P/E cycles is increased due to the recovery effect, the capacity, c_i, of each epoch is also increased, allowing more data to be written to the SSD with less throttling delays.



(a) Without epoch-capacity enforcement (b) With pessimistic epoch-capacity enforcement

Figure 6: An example of epoch-capacity enforcement. A solid line indicates the unused capacity forwarded to the next period and a dashed line represents the data delayed to the next period or epoch.



(a) With pessimistic epoch-capacity enforcement (b) With optimistic epoch-capacity enforcement

Figure 7: A comparison of the pessimistic and optimistic epoch-capacity enforcement policies when the 4 MB data are written during the period p_0.

3.4 Enforcement of Epoch Capacity

Once a throttling delay is decided, we throttle SSD performance by distributing throttling delays across every write as evenly as possible. This regulation policy is beneficial in minimizing response time variations, but it cannot guarantee the required lifetime if write demand prediction fails and unexpectedly high write traffic comes from the host. To resolve this problem, it is necessary to adopt an epoch-capacity enforcement policy, which prevents more data than the epoch capacity from being written to the SSD.

One of the easiest ways to enforce the epoch capacity is to stop writing if the epoch capacity is likely to be exhausted before the epoch ends. We call such a regulation strategy the *pessimistic* epoch-capacity enforcement policy. The pessimistic policy divides one epoch into *periods* whose lengths are 1 second each and then distributes the capacity of an epoch to all periods evenly. If more data than the period capacity were requested to write, the epoch-capacity regulator stops writing so that overflowed requests are to be written in the next period. If there is an unused capacity in the current period, the regulator reallocates it to the next period so that it can be used during the next period. This period-based capacity regulation allows us to maintain the minimum write throughput when there is unexpectedly high write traffic. If we stop writing after the epoch capacity is exhausted, the SSD cannot write any data until the epoch ends with significant performance degradation. Figure 6 compares the situations where no epoch-capacity enforcement policy is used and the pessimistic policy is used. Here, we assume that the epoch capacity is 4 MB and the number of periods is 4. As shown in Figure 6(a), the 4.2 MB data are written to the SSD without epoch-capacity enforcement. With pessimistic epoch-capacity enforcement, the maximum number of bytes written to the SSD is limited to 4.0 MB as shown in Figure 6(b).

The weakness of the pessimistic policy is that it does not efficiently handle a bursty I/O pattern which writes a large number of data within a relatively short period. Figure 7(a) shows how the pessimistic policy behaves under a bursty write request. We assume that the capacity of an epoch is 4 MB and the number of periods is 4. Consider that the 4 MB data are requested during the period p_0 while no write requests are issued during the periods $p1$, $p2$, and $p3$. In this example, the pessimistic policy throttles write requests for every period except for p_3 because the requested data always exceed the maximum capacity of the period. However, since the total number of bytes written during the epoch is equal to 4 MB, throttling for the periods, p_0, p_1, and p_2, is, in fact, unnecessary.

We resolve this overly restrictive throttling behavior for bursty write requests by proposing the *optimistic* epoch-capacity enforcement policy. Our optimistic policy maintains a relatively small amount of the spare capacity for each epoch and forcibly throttles write performance only when both the capacity of a period and the spare capacity are exhausted. Figure 7(b) shows an example of the optimistic policy with the same scenario shown in Figure 7(a). Here, we assume that the spare capacity is set to 4 MB. As shown in Figure 7(b), unnecessary throttling can be completely avoided with the optimistic epoch-capacity enforcement policy.

The spare capacity must be carefully chosen. Suppose that the spare capacity is unlimited and there is unexpectedly high write traffic. In that case, READY borrows as much capacity as possible from future epochs without limitation and then uses it up. If unexpected write demands frequently occur and write demands are gradually increasing, the SSD is worn out before the required lifetime. On the other hand, if the spare capacity is too small, unnecessary throttling with a bursty I/O pattern would be frequently observed due to the lack of spare capacity. In this work, the spare capacity is empirically set to 10% of the remaining capacity, C_r, of the SSD. This capacity is sufficient enough to avoid unnecessary throttling in real-world traces. Furthermore, since the spare capacity is limited to 10% of C_r, the worn-out of SSDs before the

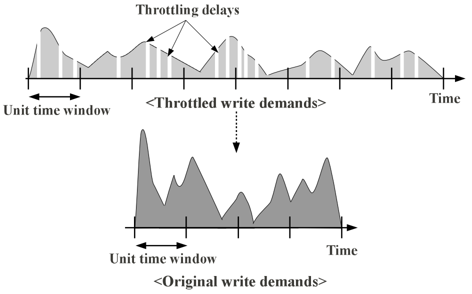


Figure 8: Reconstruction of write demand distribution.

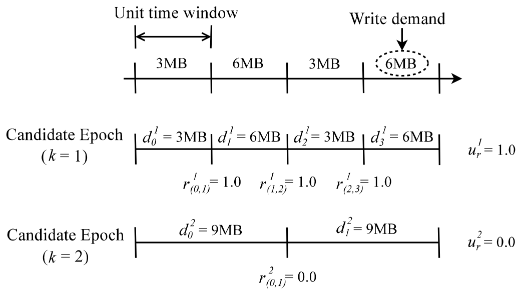


Figure 9: An overall procedure of epoch length selection.

target lifetime never occurs.

Suppose that the spare capacity is 10% and there are n epochs. The capacity of each epoch is c_0, ..., c_{n-1}, respectively. Note that c_0 = ... = c_{n-1} = C_r/n as mentioned in Section 3.3. The spare capacity for the 0-th epoch is $(c_1 + ... + c_{n-1}) \cdot 0.1$, and thus the total capacity that can be written during the 0-th epoch is $c_0 + (c_1 + ... + c_{n-1}) \cdot 0.1$. If n is 3 and C_r is 3 MB, c_0 is 1 MB and the spare capacity is 0.2 MB. If the data of less than c_0 have been written during the 0-th epoch, the remaining capacity, C_r, of the SSD after the 0-th epoch is equally distributed to the remaining epochs and then the spare capacity is determined by $(c_2 + ... + c_{n-1}) \cdot 0.1$ for the 1-st epoch. If the spare capacity, however, is partially used during the 0-th epoch, then c_1, ... , c_{n-1} are reduced to 90% of their original capacities and only the unallocated capacity is used as the spare capacity. For example, in the above example, if the data of 1.1 MB have been written during the 0-th epoch, c_1 and c_2 are 0.9 MB and the spare capacity becomes 0.1 MB in the 1-st epoch. This capacity assignment policy makes the throttling delay estimator slightly increase a throttling delay with a smaller epoch capacity. The overused capacity is accordingly reclaimed during the remaining epochs. If the spare capacity is used up during the 0-th epoch, the pessimistic policy is used with the reduced epoch capacity, i.e., 0.9 MB, and no spare capacity. This means that performance degradation caused by the depletion of the spare capacity is 10% in the worst case.

3.5 Epoch Length Selection

The length of an epoch must be carefully decided. If the epoch length is chosen improperly, the difference in write demands between epochs becomes large. Since a throttling delay is determined by the write demand of the previous epoch, the incorrect epoch length can make a large fluctuation in the overall I/O response time of the SSD. To determine the proper epoch length, we monitor write demands of a workload and find repeated cycles that show similar write demands. We then choose that cycle as the epoch length.

To realize this in READY, we collect information about write demands, i.e., the number of bytes written per unit-time window, at runtime. The write demands collected here, however, include throttling delays that distort the actual write demands of applications. Therefore, it is necessary to reconstruct write demand distribution when throttling is not applied. We estimate this original write demands by rebuilding unit-time windows without throttling delays as shown in Figure 8.

To find the proper epoch length, we use a simple approach that attempts to find the best epoch candidate, which exhibits the smallest fluctuation in write demands, by creating and evaluating several candidate epochs with different lengths. Figure 9 shows our approach in choosing an epoch length. We first create a candidate epoch whose length, k, is one unit-time window, i.e., k = 1. We then calculate the write-demand difference ratio of two consecutive epochs i and $i+1$ with the same length. The write-demand difference ratio, $r^k_{(i,i+1)}$, is defined as follows:

$$r^k_{(i,i+1)} = \frac{|d^k_{i+1} - d^k_i|}{d^k_i}, \qquad (6)$$

where d^k_i and d^k_{i+1} are the number of bytes written during the epochs i and $i+1$, respectively. For example, in the example of Figure 9, d^1_0 and d^1_1 are 3 MB and 6 MB, respectively, and thus $r^1_{(0,1)}$ = 1.0. We calculate the average write-demand difference ratio, μ^k_r, for all available pairs of two consecutive epochs. In the example of Figure 9, μ^1_r for $r^1_{(0,1)}$, ..., $r^1_{(2,3)}$ becomes 1.0.

We then increase the length of a candidate epoch by one unit-time window and calculate μ^{k+1}_r. We repeat this until the number of epochs with the same length becomes one. Finally, the length of a candidate epoch whose average write-demand difference ratio is the smallest is chosen as the epoch length, t_{epoch}. For example, in Figure 9, μ^k_r is the smallest when k is 2, and thus the new epoch length becomes the length of two unit-time windows. After choosing the new epoch length, READY recalculates a throttling delay using Eq.(4) if dynamic throttling is necessary. The new epoch length is determined under the assumption that there are no throttling delays. The epoch length, t_{epoch}, is thus increased to $t_{epoch} \cdot (w_i/c_i)$

to include delays caused by throttling.

Finding the epoch length may take a relatively long time. To mitigate the computational overhead caused by epoch length selection, the epoch length is recalculated when the write-demand difference ratio between the predicted write demand and the actual one is higher than 0.25 and it occurs three times successively. The length of a unit-time window is also set to 10 minutes to further reduce the computational overhead. In this work, 0.25 is chosen empirically by considering both computational overhead and the accuracy of write demand prediction. However, this number can be further optimized in several ways. For example, the write-demand difference ratio that triggers epoch length recalculation can be adaptively changed depending on the characteristics of a workload. If the difference ratio is always smaller than 0.25, we can reduce this number, e.g., 0.15, to find a better epoch length. On the other hand, if the difference ratio is much larger than 0.25 all the time, it may be better to reduce this number, e.g., 0.35, to avoid useless computational overhead.

4 Experimental Results

In this section, we first describe our experimental settings and explain enterprise benchmarks used for the evaluations in detail. We then analyze the benefit of the proposed READY technique over the static throttling technique in terms of SSD lifetime, response time, and response time variations.

4.1 Experimental Settings

To evaluate the effectiveness of the proposed READY technique, we have performed our evaluations using the DiskSim-based SSD simulator [23]. The flash memory used for the evaluations was based on 2-bit MLC NAND flash memory, and each block was composed of 64 4 KB pages. The page read time and the page write time were 50 μs and 600 μs, respectively, and the block erasure time was 2 ms. The number of P/E cycles allowed to a block was initially set to 3K, but it was changed depending on the length of the idle time based on our recovery model. The target lifetime of the SSD was set to 5 years.

We have implemented the static and dynamic throttling techniques in the SSD simulator, along with the damage and recovery model described in Section 2. The throttling module was implemented between the host interface, e.g., SATA, and the flash translation layer (FTL). The throttling module intercepted write requests destined for the FTL and then applied a throttling delay if it was required for the lifetime guarantee. The FTL employed a page-level address mapping algorithm with a greedy garbage collection policy and used a hot-cold swapping algorithm for wear-leveling [23]. Note that there were no changes at the FTL level for throttling because the throttling module has been designed to operate independently regardless of the underlying FTL algorithms.

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 1: A summary of traces used for evaluations.

We compared the performance of five SSD configurations: *NT*, *ST*, *DT*, *READY*$_{\text{PES}}$, and *READY*$_{\text{OPT}}$. *NT* does not use write throttling, so it cannot guarantee the target SSD lifetime if write traffic is very intensive. *ST* and *DT* use static throttling and dynamic throttling, respectively. Note that *DT* uses the optimistic epoch-capacity enforcement policy by default. Both *READY*$_{\text{PES}}$ and *READY*$_{\text{OPT}}$ are different from other configurations in that they take into account the self-recovery effect of memory cells. *READY*$_{\text{PES}}$ uses the pessimistic epoch-capacity enforcement policy, whereas *READY*$_{\text{OPT}}$ employs the optimistic policy.

4.2 Benchmarks

We have chosen two enterprise traces, `proxy` and `proj` from the MSR-Cambridge benchmark [21] and have used three production traces, `exchange`, `map`, and `msnfs`, from the MS-Production benchmark [22]. Table 1 summarizes the traces used for our evaluations. `proxy` and `proj` were recorded for one week. `exchange` and `map` contains 24-hour I/O activities, while `msnfs` was collected for 6 hours. Because of the limited duration of the traces, it was impossible to assess the lifetime guarantee of 5 years with them. For this reason, we performed our evaluations under the assumption that the same I/O pattern is repeated for 5 years.

The write demand is very different depending on the traces. `proxy` and `proj` exhibit a low write demand in comparison with `exchange`, `map`, and `msnfs`. The write amplification factor (WAF), which has a great effect on the write demand, ranges from 1.62 to 2.26 according to the characteristic of I/O references [24]. For the evaluations, the SSD capacity was configured differently depending on the traces so that the lifetime of the SSD is to be a problem. For `proxy` and `proj` with a low write demand, the SSD capacity was set to 32 GB. For `exchange`, `map`, and `msnfs` with a high write demand, the capacity of the SSD was set to 128 GB.

In practice, this capacity planning is carefully decided by customers' requirements. If customers are ready to pay money to obtain a long lifetime and high perfor-

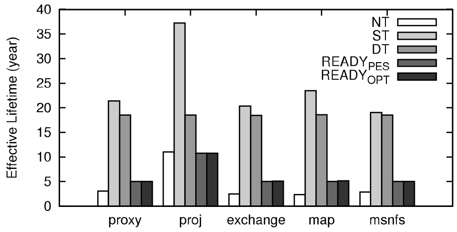


Figure 10: A comparison of effective SSD lifetimes for five traces with different SSD configurations.

mance, an over-provisioned configuration with a larger capacity SSD is the best choice. If customers require a low initial cost, but can manage a relatively high operational cost, a smaller capacity SSD with a shorter target lifetime, i.e., 3 years, is preferred. For customers who want reasonable performance with relatively lower cost and a longer target lifetime, e.g., 5 years, the settings shown in Table 1 may be a better choice.

All the traces used in this work were collected from hard disk drives (HDDs) which exhibit much lower I/O performance than SSDs. Since SSDs increase the overall I/O rate of the storage subsystem by several times [25, 26], the number of bytes written to a storage device during the same time period will be largely increased in comparison with HDDs. That is, 'data written per hour (GB)' shown in Table 1 becomes larger, and thus READY more aggressively throttles write performance because of increased write traffic. Therefore, the SSD capacity in Table 1 is set relatively conservative for systems that use SSDs as a secondary storage device.

4.3 Lifetime Analysis

We first analyze the lifetime of the SSD for five respective traces. Figure 10 shows the effective lifetime of the SSD with different SSD configurations. Here, the effective lifetime is the lifetime which is estimated based on the assumption that the I/O activities of the traces are repeated for 5 years. Note that the self-recovery effect of memory cells is taken into account in estimating the SSD lifetime. As shown in Figure 10, *NT* cannot guarantee the required lifetime of the SSD for all the traces, except for `proj`. In our observation, the write demand of `proj` is not intensive, and thus the SSD can achieve the lifetime more than 5 years without write throttling.

ST and *DT* do not consider the self-recovery effect of floating-gate transistors, and therefore they throttle write performance based on the fixed 3K P/E cycles. Since the P/E cycles of the SSD are increased due to the effect of self-recovery, the effective SSD lifetimes with *ST* and *DT* are much longer than the required lifetime. This means that *ST* and *DT* excessively throttle write performance, underutilizing available P/E cycles of the SSD. This excessive throttling results in poor write performance in comparison with *READY*$_{\text{PES}}$ and *READY*$_{\text{OPT}}$. In particular, *DT* dynamically decides a throttling delay in response to a changing workload. Therefore, *DT* maximizes the utilization of P/E cycles within 3K unlike *ST*. We will discuss this issue in detail with Table 2.

Trace	SSD configuration	C_{ssd}(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
	READY$_{\text{PES}}$		312.8	141.0
	READY$_{\text{OPT}}$		312.8	141.0
`exchange`	*NT*	375	949.3	1918.8
	ST		1415.3	348.2
	DT		1387.3	374.4
	READY$_{\text{PES}}$		1077.8	1077.2
	READY$_{\text{OPT}}$		1077.8	1065.6

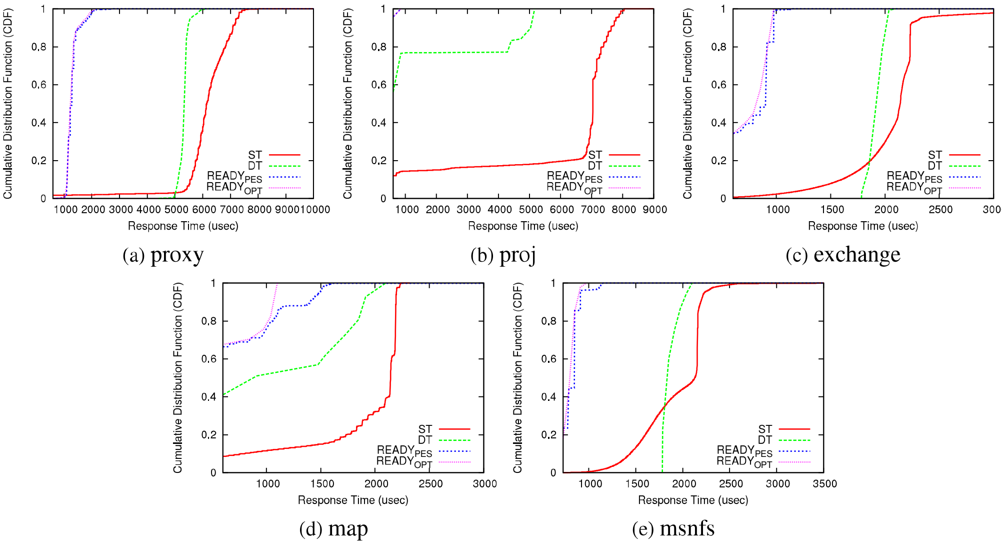
Table 2: The amount of data written for 5 years for two traces, `proj` and `exchange`.

READY$_{\text{PES}}$ takes advantage of the self-recovery effect of memory cells. Therefore, it throttles write performance so that the effective lifetime of the SSD is close to the required lifetime for all the traces. Figure 10 also shows that *READY*$_{\text{OPT}}$ guarantees the required SSD lifetime even though it uses the capacity borrowed from future epochs in advance. This clearly shows that the optimistic epoch-capacity enforcement policy properly manages overused epoch capacity so that the given lifetime is to be satisfied.

Table 2 analyzes the lifetime of the SSD from the perspective of written data for two traces, `proj` and `exchange`. As mentioned in Section 2, C_{ssd} represents the number of bytes that can be written to the SSD according to the NAND flash memory specification, whereas $C^{'}_{ssd}$ is the total number of writable bytes when the recovery effect is taken into account. W_{work} is the total number of bytes written to the SSD for 5 years.

As expected, *ST* and *DT* throttle write performance so that W_{work} becomes close to C_{ssd}. In particular, in the case of *ST*, W_{work} is about 43% and 8% smaller than C_{ssd} for `proj` and `exchange`, respectively. This is because *ST* excessively throttles write performance, assuming that write requests are always intensive. Unlike *ST*, *DT* dynamically changes a throttling delay according to the write demands of a workload and the remaining lifetime so that W_{work} is close to C_{ssd}, allowing more data to be written to the SSD.

READY$_{\text{PES}}$ and *READY*$_{\text{OPT}}$ fully utilize the endurance improvement offered by the self-recovery effect, making W_{work} close to $C^{'}_{ssd}$ at the target SSD lifetime. In the case of `proj`, since the endurance of the SSD is sufficient enough to guarantee the required 5-year lifetime, throttling is not performed in most cases.



(a) proxy (b) proj (c) exchange

(d) map (e) msnfs

Figure 12: Cumulative distribution functions (CDFs) of write response times.

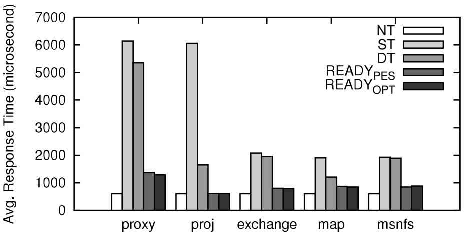


Figure 11: A comparison of average write response times for five traces with different SSD configurations.

4.4 Performance Analysis

To evaluate the performance benefit of the proposed READY technique, we measured the average response time of a page write while running five traces with different SSD configurations. Figure 11 shows the our evaluation results. As expected, *NT* exhibits the best I/O response time among all of the evaluated configurations, but it cannot guarantee the target lifetime as shown in Figure 10 because it does not throttle write performance. The average write response time of *NT* is close to the page access time, i.e., 600 μsec, with little variation.

Both *READY*$_{\text{PES}}$ and *READY*$_{\text{OPT}}$ throttle write requests to meet the required lifetime, so their performance is worse than that of *NT*; they exhibit 1.0x to 2.13x higher write response time than *NT*. In the case of `proj`, *READY*$_{\text{PES}}$ and *READY*$_{\text{OPT}}$ do not reduce write performance because the required lifetime can be satisfied without throttling. Therefore, little performance degradation, which is less than 1.9%, is observed in `proj`. *READY*$_{\text{PES}}$ and *READY*$_{\text{OPT}}$ achieve 2.57x better performance than *DT* on average. This performance benefit mainly comes from the increased P/E cycles of the SSD. Since *READY*$_{\text{PES}}$ and *READY*$_{\text{OPT}}$ are aware of the improvement in SSD endurance, they can assign more capacity to epochs, reducing throttling delays.

DT exhibits 1.7x faster response time over *ST* on average. *DT* determines the epoch capacity periodically based on the remaining lifetime of the SSD and changes a throttling delay so that write requests are properly delayed in response to future write demands. This epoch capacity assignment and throttling delay distribution policy allows us to fully utilize the available endurance of the SSD. On the other hand, *ST* neither considers the remaining lifetime of the SSD nor the characteristic of a workload in making a throttling decision. Instead, *ST* simply throttles write performance by limiting the maximum bandwidth of the SSD. Therefore, *ST* causes many unnecessary throttling delays.

The response time variation is one of the important design issues that must be taken into account in designing throttling algorithms. We compared response time variations between different SSD configurations. Figure 12 shows the cumulative density functions (CDFs) of write response times for five traces. As shown in Figure 12, *ST* shows significant response time variations for all the traces because it forcibly stops writing if throttling is needed. On the other hand, by distributing throttling delays to write requests as evenly as possible, *NT*, *READY*$_{\text{PES}}$, and *READY*$_{\text{OPT}}$ greatly reduce variations on the write response time.

For `exchange`, `msnfs`, and `map`, *READY*$_{\text{PES}}$ incurs relatively high I/O response time variations in comparison with *READY*$_{\text{OPT}}$. *READY*$_{\text{PES}}$ must stop writing data when there are a large number of writes within a short

Traces	proxy	proj	exchange	map	msnfs
Accuracy of write demand prediction (%)	99.9	33.9	80.5	50.9	100

Table 3: Accuracy of write demand prediction.

period. On the other hand, $\mathit{READY}_{\mathrm{OPT}}$ uses the optimistic epoch-capacity enforcement policy, so they handle a bursty I/O pattern more efficiently without compulsorily write throttling.

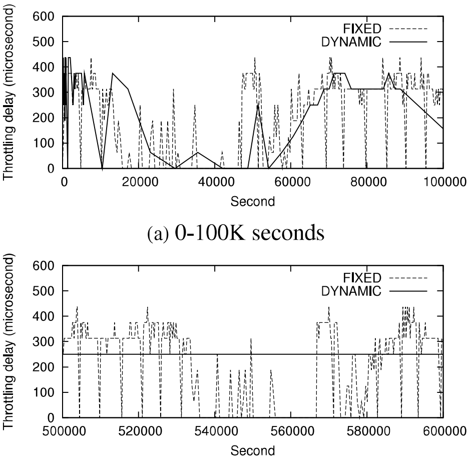
The write response time of *DT* ranges from 600 μsec to several thousand seconds in `map` and `proj` unlike `proxy`, `exchange`, and `msnfs`. The write patterns of `map` and `proj` change greatly with time, and thus the difference in write demands between two consecutive epochs is relatively large. Since a throttling delay for a certain epoch is determined by the write demand of the previous epoch, the difference between throttling delays is accordingly increased in `map` and `proj`. Nevertheless, the response time of *DT* is more stable than *ST*.

We evaluated the accuracy of our epoch length selection method in predicting future write demands. Table 3 shows our evaluation results for five traces. We assume that epoch length detection is accurate if the difference between the prediction write demand and the actual one is smaller than 25%. As shown in Table 3, our method achieves high accuracy for `proxy`, `exchange`, and `msnfs`. The accuracy of write demand prediction, however, is reduced to 50.9% and 33.9% for `map` and `proj`, respectively, due to a high fluctuation in write requests. We expect that the accuracy of epoch length detection may be improved with traces recorded for a longer time.

To evaluate the effect of the epoch length selection method on the SSD response time, we compared the changes in throttling delays when the fixed epoch length is used and the epoch length is dynamically determined according to a workload. For the evaluation, we executed the `exchange` trace, which is a 24-hour trace, repeatedly. Figure 13 shows our evaluation result. In this figure, *FIXED* represents $READY_{OPT}$ with the fixed epoch length and *DYNAMIC* is the SSD configuration when $READY_{OPT}$ uses the proposed epoch length detection method. The fixed epoch length was set to 10 minutes. As shown in Figure 13, even though $READY_{OPT}$ generally works well with `exchange`, some variations on throttling delays are observed with *FIXED*. *DYNAMIC* also exhibits variations on response times at the beginning of the execution, but it becomes stable after repeated write demands are detected as shown in Figure 13(b).

4.5 Detailed Analysis

We performed a detailed analysis of different SSD configurations. Figure 14 represents the throughput of the SSD with the different throttling schemes when intense I/Os are being served. As mentioned several times before, *ST* limits the maximum bandwidth of the SSD by a certain level, 2.49 MB/s in `map`. The overall throughput of the SSD is thus greatly deteriorated with *ST* as shown in Figure 14(a). *DT* works better than *ST*. Due to the limited write endurance of the SSD, however, significant performance degradation cannot be avoided with *DT* as depicted in Figure 14(b). $\mathit{READY}_{\mathrm{PES}}$ and $\mathit{READY}_{\mathrm{OPT}}$ exhibit much higher performance than *ST* and *DT* by exploiting the improved write endurance of the SSD benefited from the self-recovery effect of memory cells. In particular, $\mathit{READY}_{\mathrm{OPT}}$ performs better than $\mathit{READY}_{\mathrm{PES}}$ when a large number of data are being written to the SSD, e.g., a period of 200 to 350 second in Figure 14(d). Even when write requests are intensively issued, $\mathit{READY}_{\mathrm{OPT}}$ writes the requested data to the SSD rather than forcibly throttling the bandwidth of the SSD by using the spare capacity borrowed from future epochs. This allows $\mathit{READY}_{\mathrm{OPT}}$ to exhibit better write response time for the traces like `map` which exhibit a great fluctuation in write requests.



(a) 0-100K seconds

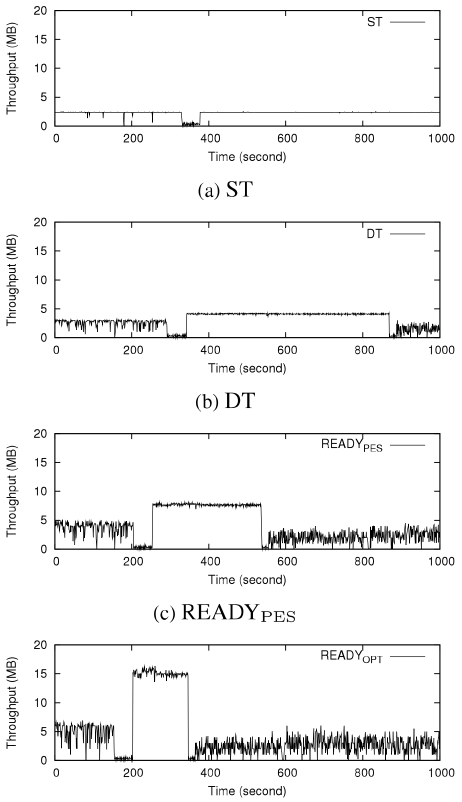
(b) 500K-600K seconds

Figure 13: A comparison of throttling delays when the fixed epoch length is used and the epoch length is dynamically determined in the `exchange` trace.

5 Related Work

There have been a lot of studies on improving the endurance of flash-based SSDs. Many existing garbage collection and wear-leveling techniques [27, 28, 29, 30, 31, 32, 33, 34] are designed to improve the lifetime of SSDs by avoiding useless data migration during a block recycling process or by distributing P/E cycles of flash blocks as evenly as possible.

As the endurance of flash memory continuously dete-



(a) ST

(b) DT

(c) READY$_{\text{PES}}$

(d) READY$_{\text{OPT}}$

Figure 14: A detailed analysis of four SSD configurations with the `map` trace.

riorates, several endurance enhancement techniques that aggressively reduce the number of data written to SSDs have been proposed. Data de-duplication [26, 35, 36] and data compression [37, 38] are representative examples of these policies. Data de-duplication detects duplicate data blocks that already exist in a storage device and then eliminates redundant writes to SSDs for such blocks. Data compression eliminates repeated bit patterns within a data block, reducing writes to SSDs. These techniques are useful in improving the lifetime of SSDs, but they have some limitations in that none of them guarantee the SSD lifetime or make use of the recovery effect of a memory cell.

More recently, the approaches that exploit the recovery effect of flash devices have received considerable attention. This paper is an improved version of our preliminary work [39]. Mohan *et al.* investigated the effect of the damage recovery on the lifetime of SSDs for enterprise servers [1]. They claimed that the endurance of NAND flash memory was durable enough even for I/O intensive enterprise applications because of its recovery ability. However, their investigations were limited to 90 nm SLC and MLC flash memories which exhibit good endurance properties. They also did not exploit the benefit of the recovery effect in ensuring the lifetime of SSDs. Wu *et al.* presented an endurance enhancement technique that boosts recovery speed by heating a flash chip worn out under high temperature [10]. By leveraging the temperature-accelerated recovery, it improved the endurance of SSDs up to five times. However, one of the major drawbacks of this approach is that it requires extra energy consumption to heat flash chips, lowering the energy efficiency of a storage device. Unlike Wu's work, our study considers the endurance improvement of SSDs at the room temperature and exploits this benefit to guarantee the lifetime of SSDs with less throttling overhead.

6 Conclusions

In this paper, we proposed a recovery-aware dynamic throttling technique, READY, to overcome two main problems in realizing the adoption of SSDs in enterprise server systems: the continuously decreasing endurance and unpredictable lifetime problems. READY throttles write performance so that the required lifetime of SSDs is to be satisfied. In order to guarantee the SSD lifetime with less throttling overhead, READY exploits the recovery effect of a floating-gate transistor which effectively increases the number of effective P/E cycles of SSDs. Our evaluation results showed that the proposed throttling technique guarantees a lifetime warranty, while achieving a relatively small reduction in write response time and little response time variation over the static throttling technique.

READY can be improved in several directions. The stress and recovery model of this work is based on the previous studies on the physical characteristics of flash memory [1, 2, 10]. These studies carefully modeled the stress and recovery characteristics of flash memory, but their scopes were limited to NOR or NAND flash memory fabricated in over 90 nm technology. To build a more accurate stress and recovery model, we will perform investigations using real NAND flash parts which are fabricated in less than 30 nm technology. We also plan to implement READY in a real SSD platform to evaluate its effectiveness in real systems.

Acknowledgments

We would like to thank Daniel Peek, our shepherd, and anonymous referees for valuable suggestions that greatly improved the paper. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 20110020426, No. R33-10095, and No. 2011-0020514). Samsung Electronics partially supported our research and the ICT at SNU provided research facilities for this study.

References

[1] 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.

[2] 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.

[3] 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, pp. 121-129, 2011.

[4] N. Sommer, "Signal Processing and the Evolution of NAND Flash Memory," *Embedded Computing Design*, vol. 8, no. 8, 2010.

[5] Y. Koh, "NAND Flash Memory Scaling Beyond 20 nm," in *Proceedings of the IEEE International Memory Workshop*, 2009.

[6] M. Goldman, K. Pangal, G. Naso, and A. Goda, "25nm 64Gb 130mm^2 3bpc NAND Flash Memory," in *Proceedings of the International Memory Workshop*, 2011.

[7] Micron Technology Inc., "An Enterprise-Focused MLC SSD," `http://www.micron.com/products/solid_state_storage/enterprise_ssd/p400e.html`, 2011.

[8] SandForce Inc., "SF-2500 & SF-2600 Enterprise SSD Processors," `http://www.sandforce.com/index.php?id=21&parentId=2`, 2010.

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

[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] R. Yamada, T. Sekiguchi, Y. Okuyama, J. Yugami, and H. Kume, "A Novel Analysis Method of Threshold Voltage Shift due to Detrap in a Multi-Level Flash Memory," in *Proceedings of the Symposium on VLSI Technology*, 2001.

[12] H. Yang and et. al, "Reliability Issues and Models of Sub-90nm NAND Flash Memory Cells," in *Proceedings of the International Conference on Solid-State and Integrated Circuit Technology*, 2006.

[13] 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.

[14] S. Gurumurthi, A. Sivasubramaniam, and V. Natarajan, "Disk Drive Roadmap from the Thermal Perspective: A Case for Dynamic Thermal Management," in *Proceedings of the International Symposium on Computer Architecture*, 2005.

[15] L. Grupp, A. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. Siegel, and J. Wolf, "Characterizing Flash Memory: Anomalies, Observations, and Applications," in *Proceedings of the International Symposium on Microarchitecture*, 2009.

[16] S. Boboila and P. Desnoyers, "Write Endurance in Flash Drives: Measurements and Analysis," in *Proceedings of the USENIX conference on File and Storage Technologies*, 2010.

[17] SanDisk, "Longterm Data Endurance (LDE) for Client SSD," `http://www.sandisk.com/Assets/File/pdf/oem/LDE/WhitePaper.pdf`, 2008.

[18] Silicon Graphics International, "Solid State Disk Solutions," `http://www.sgi.com/products/storage/ssd/endurance.html`.

[19] SMART Modular Technologies, "XceedIOPS SATA SSD," `http://www.smartm.com/products/storage/SSDs.asp`.

[20] 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.

[21] 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.

[22] 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.

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

[24] X. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, "Write Amplification Analysis in Flash-based Solid State Drives," in *Proceedings of the Israeli Experimental Systems Conference*, 2009.

[25] Fusion-io Inc., "MySpace Uses Fusion Powered I/O to Drive Greener and Better Data Centers," `http://www.fusionio.com/case-studies/myspace-case-study.pdf`, 2010.

[26] Q. Yang and J. Ren, "I-CASH: Intelligently Coupled Array of SSD and HDD," in *Proceedings of the International Symposium on High Performance Computer Architecture*, 2011.

[27] J. Kim, J.-M. Kim, S.-H. Noh, S.-L. Min, and Y. Cho, "A Space-Efficient Flash Translation Layer for Compact Flash Systems," *IEEE Transactions on Consumer Electronics*, vol. 48, no. 2, pp. 366-375, 2002.

[28] S.-W. Lee, D.-J. Park, T.-S. Chung, W.-K. Choi, D.-H. Lee, S.-W. Park, and H.-J. Song, "A Log Buffer Based Flash Translation Layer Using Fully Associative Sector Translation," *ACM Transactions on Embedded Computing Systems*, vol. 6, no. 3, 2007.

[29] J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee, "A Superblock-Based Flash Translation Layer for NAND Flash Memory," in *Proceedings of the International Conference on Embedded Software*, 2006.

[30] S. Lee, D. Shin, Y.-J. Kim, and J. Kim, "LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems," *ACM SIGOPS Operating Systems Review*, 2008.

[31] H. Kim and S. Lee, "A New Flash Memory Management for Flash Storage System," in *Proceedings of the Computer Software and Applications Conference*, 1999.

[32] M.-L. Chiang, P.-H. Lee, and R.-C. Chang, "Cleaning Policies in Mobile Computers using Flash Memory," *Journal of Systems and Software*, 1999.

[33] L.-P. Chang, "On Efficient Wear Leveling for Large-Scale Flash-Memory Storage Systems," in *Proceedings of the ACM Symposium on Applied Computing*, 2007.

[34] D. Jung, Y.-H. Chae, H. Jo, J.-S. Kim, and J. Lee, "A Group-Based Wear-Leveling Algorithm for Large-Capacity Flash Memory Storage Systems," in *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 2007.

[35] 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.

[36] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam, "Leveraging Value Locality in Optimizing NAND Flash-Based SSDs," in *Proceedings of the 9th USENIX Conference on File and Storage Technologies*, 2011.

[37] K. Yim, H. Bahn, and K. Koh, "A Flash Compression Layer for Smartmedia Card Systems," *IEEE Transactions on Consumer Electronics*, 2004.

[38] 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.

[39] S. Lee, T. Kim, K. Kim, and J. Kim, "Guaranteeing the Lifetime of SSDs Using Recovery-Aware Dynamic Throttling," in *Proceedings of the International Memory Architecture and Organization Workshop*, 2011.