OFWAR: Reducing SSD Response Time Using On-Demand Fast-Write-and-Rewrite

Qi Wu and Tong Zhang, *Senior Member*, *IEEE*

Abstract—This paper presents a cross-layer design strategy to reduce SSD response time and its variation. The key is to cohesively exploit system-level run-time data access workload variation and temporal locality and device-level NAND flash memory write latency versus data retention time trade-off. The basic idea is simple: once write intensity of the workload increases and begins to degrade SSD response time, we speed up memory programming at the penalty of shorter data retention time, and rewrite these short-lifetime data later, if necessary, when workload write intensity drops. A scheduling solution is developed to effectively implement this design strategy. Simulations over various workloads were carried out and the results demonstrate that the cross-layer design strategy can reduce the average SSD response time by up to 52.3%.

Index Terms—Solid-state drive, data retention, workload variation, write latency, average response time

1 Introduction

The steady bit cost reduction over the past decade has enabled NAND flash memory enter increasingly diverse applications. In particular, it is now economically viable to implement solid-state drives (SSDs) using NAND flash memory. Modern SSDs, especially high-end enterprise SSDs, can deliver very high IOPS (Input/Output Operations per Second), e.g., up to 1,180,000 IOPS of the latest Fusion-IO PCIe SSDs [1]. This is realized by implementing many internal SSD channels and aggressively applying architectural techniques such as interleaving and stripping. IOPS has become the most widely cited metric in advertising commercial SSDs. However, as pointed out by recent studies (e.g., see [2]), it is at least equally important to reduce SSD response time and its variation for enterprise applications. Unfortunately, with the technology scaling down, NAND flash memory programming speed continues to drop, which tends to significantly degrade SSD response time.

This paper presents a cross-layer design approach to reduce both SSD response time and its variation. First, it is well known that most real-life workloads exhibit certain degree of data access intensity variation, and write requests during high-intensity period play a critical role in determining SSD response time. Since it becomes increasingly infeasible to make NAND flash memory chips provide a *sustained* high programming speed, it can be highly desirable if NAND flash memory chips can *temporally* boost programming speed once write requests during high-intensity period begin to noticeably degrade system speed performance. Because of the progressive nature of NAND flash memory programming procedure, memory cell operational noise margin is proportional to memory programming latency. Given the fixed memory program/erase (P/E) cycling endurance target, a smaller memory cell operational noise margin will directly result in a shorter data retention time. Therefore, there is a fundamental trade-off between NAND flash memory programming speed and data retention time. This makes it possible to temporally boost memory programming speed at the cost of data retention time, which has already been used in latest commercial products [3]. The above discussion suggests an on-demand fast-write-and-rewrite design strategy: Whenever necessary, we temporally boost NAND flash memory programming speed to ensure the desired SSD response time. Since those fast-written data have very short data retention time (e.g., few weeks or days instead of several months or years), we should rewrite those data in time to ensure data integrity. We develop a specific strategy to efficiently implement this simple design concept, which can further exploit workload temporal locality to reduce the overhead. In summary, our contributions include:

1) We for the first time propose to cohesively exploit system-level workload variation and device-level programming speed vs. data retention time trade-off to reduce SSD response time. This is realized by a simple on-demand fast-write-and-rewrite design approach.
2) We present a practical strategy to implement the proposed design concept, which can further effectively exploit workload temporal locality. We also discuss various incurred implementation overheads and analyze their impacts.
3) With the case study of real-world disk traces and using 2 bits/cell NAND flash memory, we demonstrate that the proposed cross-layer design approach can reduce the average SSD response time by up to 52.3%. We also extensively evaluate the involved design trade-offs.

The rest of this paper is organized as follows: The basics of NAND flash memory are presented in Section 2. Section 3 presents the key underlying motivations and basic concept,

• *The authors are with the Electrical, Computer and Systems Engineering Department, Rensselaer Polytechnic Institute (RPI), Troy, NY 12180. E-mail: seaflywu@gmail.com; tong.zhang@ieee.org.*

Manuscript received 14 Dec. 2012; revised 28 Mar. 2013; accepted 24 Apr. 2013. Date of publication 12 May 2013; date of current version 12 Sep. 2014. Recommended for acceptance by L. Wang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2013.114

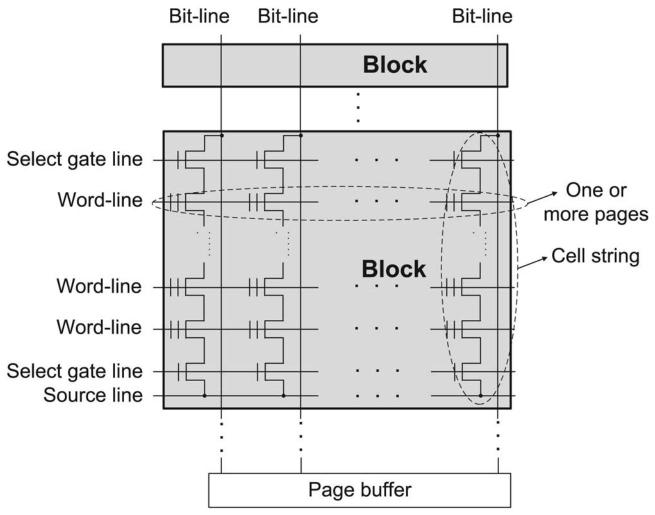


Fig. 1. Illustration of NAND flash memory array structure.

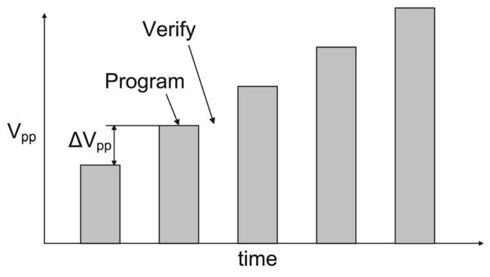


Fig. 2. Illustration of the incremental step pulse program (ISPP) scheme being used to realize programming in NAND flash memory.

and proposes an implementation strategy. The effectiveness and involved trade-offs are quantitatively evaluated by extensive simulations, for which the simulation setup is described in Section 4, and basic results and sensitivity analysis are presented in Section 5 and Section 6, respectively. Related work are surveyed in Section 7, and conclusions are drawn in Section 8.

2 Basics of NAND Flash Memory

Each NAND flash memory cell is a floating gate transistor whose threshold voltage can be programmed by injecting certain amount of charges into the floating gate. Hence, flash memory data storage is realized by programming the threshold voltage of each memory cell into two or more non-overlapping voltage windows. Before one memory cell can be programmed, it must be erased (i.e., remove the charges in the floating gate, which sets its threshold voltage to the lowest voltage window). NAND flash memory cells are organized in an $\mathrm{array} \rightarrow \mathrm{block} \rightarrow \mathrm{page}$ hierarchy, as illustrated in Fig. 1, where one memory array is partitioned into blocks, and each block contains a certain number of pages. Within one block, each memory cell string typically contains 16 to 64 memory cells, and all the memory cells driven by the same word-line are programmed and sensed at the same time. All the memory cells within the same block must be erased at the same time. Data are programmed and fetched in the unit of page, where the page size ranges from 512B to 8KB user data.

Regarding memory programming, threshold voltage control is typically realized by using incremental step pulse program (ISPP) [4], [5], i.e., all the memory cells on the same word-line are recursively programmed using a program-and-verify approach with a stair case program voltage V_{pp} with a step increment of ΔV_{pp}, as illustrated in Fig. 2. Such a recursive program-and-verify strategy is necessary to accommodate the inevitable fabrication process variation that makes the threshold voltage of different memory cells increments differently even under the same amount of programming voltage. Under such a recursive program-and-verify strategy, each programmed state associates with a verify voltage that is used in the verify operation and determines the target position of each programmed state threshold voltage window. Denote the verify voltage of the target programmed state as V_p. During each program-and-verify cycle, the threshold voltage of each memory cell is first boosted by up to ΔV_{pp} and then compared with the corresponding verify voltage V_p. If the memory cell threshold voltage is lower than V_p, the program-and-verify recursion will continue, otherwise the corresponding bit-line will be configured so that further programming of this memory cell is disabled.

3 Proposed Design Strategy

3.1 Motivation and Basic Concept

Our proposed on-demand fast-write-and-rewrite design strategy aims to reduce SSD response time, especially for applications with high write intensities. The basic idea is simple and directly motivated by the following three well-known facts:

3.1.1 NAND Flash Memory Programming Speed vs. Data Retention Time Trade-Off

As discussed in Section 2, NAND flash memory programming employs a recursive program-and-verify procedure that sweeps the entire memory cell threshold voltage region with a step increment ΔV_{pp}. As we increase ΔV_{pp}, the flash memory programming will incur a less number of program-and-verify cycles, leading to a higher programming speed. Nevertheless, a bigger ΔV_{pp} will increase the width of each programmed level and hence reduce the noise margin between adjacent programmed levels, leading to data retention time degradation. Modern flash memory chips can support dynamic configuration of the programming speed vs. data retention time trade-off [3].

3.1.2 Data Access Intensity Variation

Most real-world workloads exhibit noticeable variability in terms of data access intensity. During the period with workload intensity, many write I/O transactions can overlap and hence must be buffered in a write request queue. The response time of those overlapped transactions has bigger impacts on overall system performance than the response time of those non-overlapped transactions, which can be illustrated by a simple example shown in Fig. 3. On the other hand, during the period with low workload intensity, the storage systems become largely idle and its bandwidth is largely under-utilized.

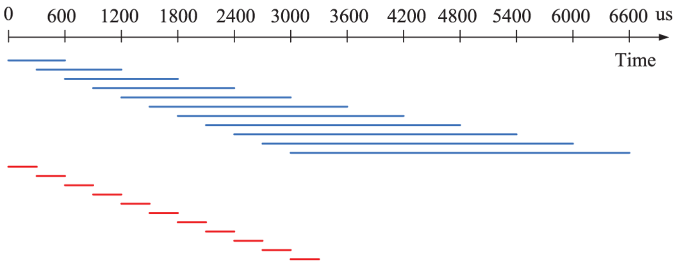


Fig. 3. An illustrative example that shows the impacts of overlapped requests on average response time. Assume every $300\ \mu s$ a write request arrives and the normal flash memory programming latency is $600\ \mu s$. As shown in the upper part of the figure, the average response time for total 11 requests is $3600\ \mu s$. For simplicity, we ignore all the other factors in the response time (e.g., data transfer time, ECC delay, etc.). If the flash programming latency is reduced to $300\ \mu s$, the average response time for total 11 requests will be reduced to $300\ \mu s$ as shown in the lower part of the figure, which is only 8.3% of the original average response time.

3.1.3 Data Access Temporal Locality

Most real-world workloads exhibit a certain degree of data access temporal locality, which has been extensively exploited in modern computing systems. Clearly, due to such temporal locality, lifetime of some data may be (much) less than the SSD system data retention time in current practice (e.g., several months and longer). This naturally makes it favorable to trade data retention time for programming speed.

Motivated by the above three facts, the proposed design strategy can be described as follows: In normal operations of SSDs, all the NAND flash memory chips employ a normal program step voltage $\Delta V_{pp}^{(norm)}$ that can guarantee the system specified data retention time $\tau^{(norm)}$. Once the write request queue contains too many overlapped write transactions, the SSD controller will make NAND flash memory chips use a bigger program step voltage $\Delta V_{pp}^{(large)}$ ($> \Delta V_{pp}^{(norm)}$), which can increase memory programming speed but meanwhile can only ensure a (much) shorter data retention time $\tau^{(short)}$ ($<\tau^{(norm)}$). Because those fast-written data cannot meet the system specified data retention time, they must be rewritten within $\tau^{(short)}$ to prevent data loss. Due to the data access intensity variation, the data rewrite operations may be carried out during SSD idle time. In addition, repeated data write due to data access temporal locality may obviate the rewrite operations for some fast-written data.

Very intuitively, this simple design approach can reduce the average SSD response time, which will be quantitatively demonstrated in Section 5. Nevertheless, this design concept could increase write amplification, which further degrades P/E cycling endurance and potentially invokes garbage collection (GC) more frequently. At the first glance, one may expect that every fast write operation will increase the write amplification. Fortunately, it is not necessarily true. In fact, under the ideal case, those operations may not increase write amplification at all. Let us consider an example as shown in Fig. 4, although there are six fast-writes and two rewrites, the total number of writes with or without fast write are same. The reason that the rewrite operations on logical page number 3 & 4 do not increase the write amplification is that no further requests with logical page number 3 & 4 occur between rewrite operations and GC. The reason that the fast-write operations on logical page number 7 & 8 do not increase the write amplification is that they are rewritten by subsequent

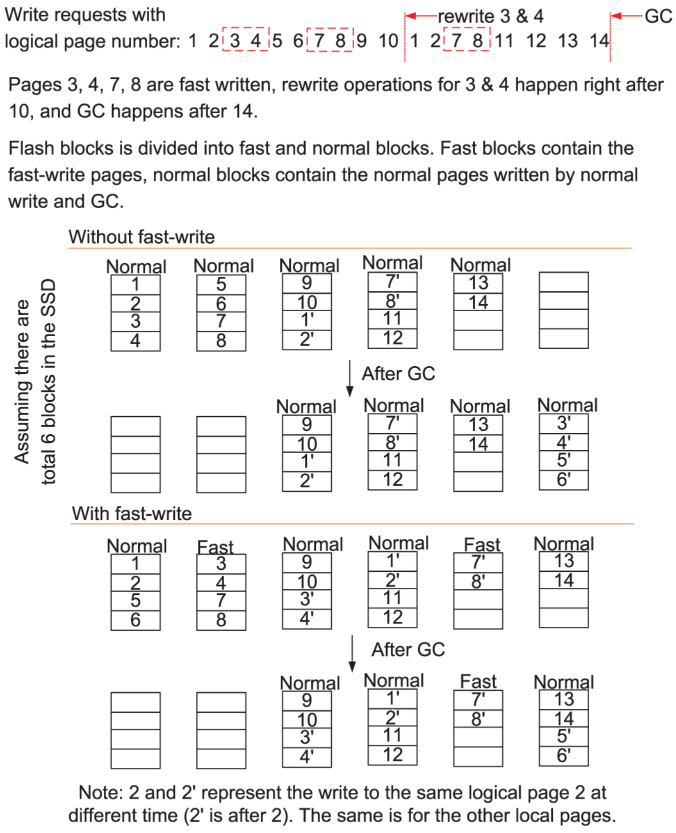
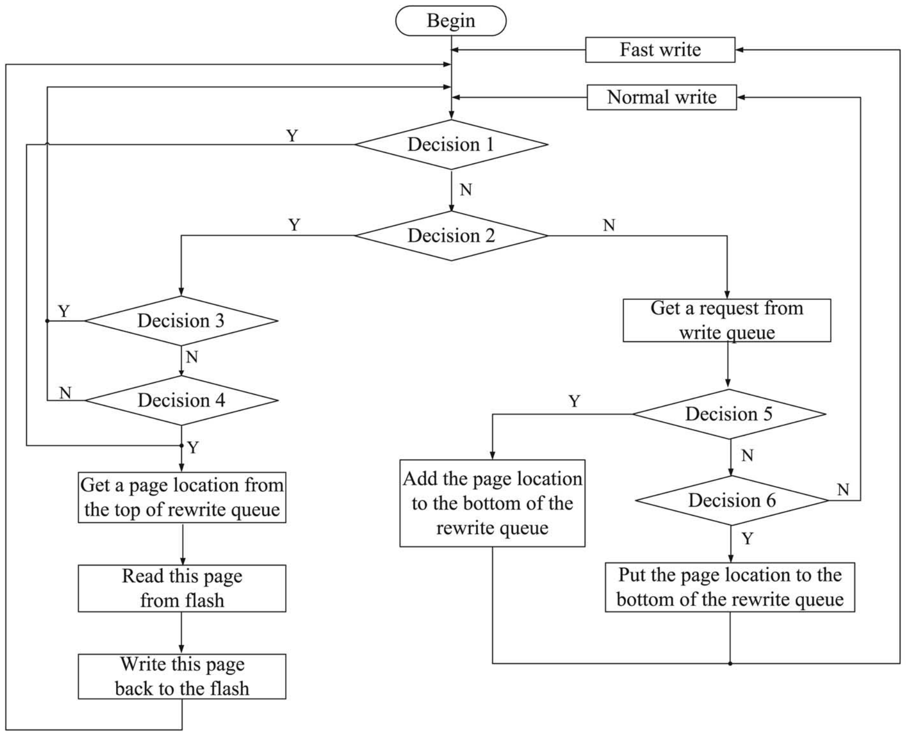

Write requests with
logical page number: 1 2 3 4 5 6 7 8 9 10 1 2 7 8 11 12 13 14
rewrite 3 & 4
GC
Pages 3, 4, 7, 8 are fast written, rewrite operations for 3 & 4 happen right after 10, and GC happens after 14.
Flash blocks is divided into fast and normal blocks. Fast blocks contain the fast-write pages, normal blocks contain the normal pages written by normal write and GC.
Without fast-write
Assuming there are total 6 blocks in the SSD
Normal
1
2
3
4
Normal
5
6
7
8
Normal
9
10
1'
2'
Normal
7'
8'
11
12
Normal
13
14
After GC
Normal
9
10
1'
2'
Normal
7'
8'
11
12
Normal
13
14
Normal
3'
4'
5'
6'
With fast-write
Normal
1
2
5
6
Fast
3
4
7
8
Normal
9
10
3'
4'
Normal
1'
2'
11
12
Fast
7'
8'
Normal
13
14
After GC
Normal
9
10
3'
4'
Normal
1'
2'
11
12
Fast
7'
8'
Normal
13
14
5'
6'
Note: 2 and 2' represent the write to the same logical page 2 at different time (2' is after 2). The same is for the other local pages.

Fig. 4. An illustrative example of the ideal case under which rewrite operations do not increase write amplification.

requests, therefore there are no extra rewrite operations triggered. This example suggests that we should try to utilize workload temporal locality and GC to reduce the write amplification. Accordingly, we develop the following two techniques: (i) In order to increase the utilization of workload temporal locality, we trigger rewrite operations only when the number of requests in the rewrite queue reaches a threshold value. (ii) We give fast-write blocks higher priority to be recollected in the garbage collection. We will further elaborate on these two techniques in Section 3.2 and their effectiveness will be quantitatively demonstrated in Section 5.

3.2 Implementation Strategy

This subsection presents a strategy to practically implement the above proposed design concept to reduce SSD system response time with minimized cycling endurance overhead. Fig. 5 shows the operational flow of the proposed implementation strategy. The SSD controller contains two buffer queues to handle write requests, including (i) write queue that stores the data of write requests from the host and (ii) rewrite queue that stores the information regarding data to be rewritten. In order to reduce memory consumption, the rewrite queue only holds the logical and physical address of those fast-written pages. When a data page should be rewritten, according to its location, we first read the data page from NAND flash memory and then rewrite it to another physical location with normal memory programming speed. Since the read latency of NAND flash memory is much less than its write latency, this design approach will not incur noticeable extra


Begin
Fast write
Normal write
Decision 1
Y
N
Decision 2
Y
N
Decision 3
Y
N
Decision 4
N
Y
Get a page location from the top of rewrite queue
Read this page from flash
Write this page back to the flash
Get a request from write queue
Decision 5
Y
N
Add the page location to the bottom of the rewrite queue
Decision 6
N
Y
Put the page location to the bottom of the rewrite queue

Decision 1: Is the lifetime of the top entry in the rewrite queue longer than a pre-set threshold value?
Decision 2: Is SSD channel idle?
Decision 3: Is rewrite queue empty?
Decision 4: Is the number of entries in the rewrite queue larger than a pre-set threshold value?
Decision 5: Is the target page location of this write request in the rewrite queue?
Decision 6: Is the number of write requests being held in the write queue is larger than a pre-set threshold value?

Fig. 5. Illustration of the fast-write strategy.

latency and meanwhile can significantly reduce the memory resources used by the rewrite queue.

As shown in Fig. 5, whenever the lifetime of the top entry in the rewrite queue exceeds a pre-set threshold value, the SSD controller interrupts all the other operations and start the rewrite operation (i.e., fetch this rewrite queue top entry, read the corresponding page data from the NAND flash memory, and rewrite this data page back with normal memory programming speed). Otherwise, when the SSD is idle, the controller first checks that whether the number of entries in the rewrite queue is bigger than a pre-set threshold value, and if yes it will execute a rewrite operation. When we fetch one host write request from the write queue, if the target logical page location of this write request also resides in the rewrite queue, this is referred to as *rewrite queue hit*. A rewrite queue hit suggests that the corresponding data page could have a higher probability of experiencing temporal locality. Therefore, when rewrite queue hit occurs, we write the corresponding data page with the fast memory programming speed, and meanwhile add the corresponding entry in the rewrite queue to the bottom of the rewrite queue. The old entry in the rewrite queue corresponding to this logical page does not need to move. When the number of write requests being held in the write queue is bigger than a pre-set threshold value, we always process write requests with the fast memory programming speed, regardless of whether rewrite queue hit occurs or not. When the number of write requests being held in the write queue is less than the pre-set threshold value and rewrite queue hit does not occur, we simply process write requests with normal memory programming speed.

Compared with current design practice, the rewrite queue is an unique hardware component demanded by implementing this proposed on-demand fast-write-and-rewrite design concept. Hence, we further elaborate on the rewrite queue management as follows. We propose to use the simple least recently used (LRU) replacement algorithm in the rewrite queue management for two main reasons: (i) LRU-based queue management can ensure that the entries in the rewrite queue are still ordered based on their lifetime, i.e., the entry on the top of the rewrite queue always corresponds to the data page with the longest lifetime. Since we need to ensure that those fast-written data pages cannot exceed their short data retention time limit, such an ordered queue makes it easy to meet this requirement. (ii) The entries corresponding to those relatively hot data pages could have a bigger probability to be kept in the rewrite queue. This can further reduce the actual number of rewritten operations. In order to avoid expensive rewrite queue search, both the logical page number and physical page number need to be put into the rewrite queue. We also need to add a flag bit in the mapping table to indicate whether the logical page accessed by the incoming write request is in the rewrite queue. If any write request hits the rewrite queue, we will do fast-write on this request. At the same time, the logical page number and its new physical page number will be put into the bottom of the rewrite queue. The old entry in the rewrite queue corresponding to this

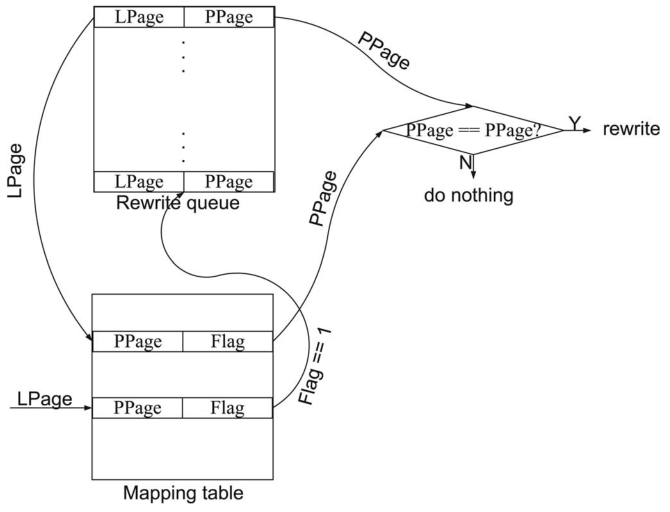


Fig. 6. Rewrite queue operation scheme.

logical page does not need to be moved. When the old entry is flushed from the rewrite queue, we will compare its physical page number with the physical page number of the same logical page number in the mapping table. If they are same, this physical page needs to be rewritten. Otherwise, we do not need to rewrite this physical page since it is rewritten already. Therefore, the only cost is the mapping table lookup which is also needed even without fast-write. Fig. 6 shows the process discussed above.

In MLC SSDs, pages are divided into most significant bit (MSB) pages and least significant bit (LSB) pages. The use of a higher program step voltage ΔV_{pp} indeed will lead to a shorter retention time of both MSB page and LSB page on the same word-line. Therefore, when using the proposed design method to MLC NAND flash memory, the MSB and LSB pages written to the same word-line should be both normal-write or fast-write. Therefore, we categorize all the flash memory blocks into fast-write blocks and normal-write blocks. The fast-write pages are always written to the fast-write blocks and the normal-write pages are always written to the normal-write blocks. The garbage collection will always proceed with normal-writes, since garbage collected pages are most likely cold pages. In order to reduce the degradation of effective P/E cycling endurance, the garbage collection algorithm should give fast-write blocks a higher priority than normal-write blocks to be recollected.

Finally, we note that, as shown in Fig. 5 and discussed above, if the number of entries in the rewrite queue is less than the pre-set threshold value, the SSD controller does not execute a rewrite operation even when the SSD is idle. At the first glance, one may suggest that a better alternative is to remove this pre-set threshold constraint and rewrite fast-written data pages as early as possible. Although this alternative can better prevent rewrite queue overflow for highly intensive workloads, it may significantly reduce the rewrite queue hit rate and increase the actual number of rewrite operations. In another word, this alternative tends to under-utilize the temporal locality in real-life workloads. Therefore, we choose to add a threshold constraint on the number of entries in the rewrite queue before a rewrite operation can be executed. The impact of such a threshold will be quantitatively evaluated in Section 6.

3.3 Data Protection during Power Shutdown

Since fast-write pages have much shorter retention time, data may be lost without proper protection during power shutdown period. There are two different scenarios of power shutdown, one is graceful power shutdown and the other one is power failure (or sudden power loss). Graceful power shutdown is an under-control event during which the power source is still providing the power to the system. Under this circumstance, the fast-write pages will be rewritten using normal programming speed before system shutdown. For example, if the rewrite queue depth is 64 K, at the worst case 256 MB data need to be rewritten. It can be finished with around 1 second using latest SSDs.

Power failure means that the system suddenly loses power. Under this circumstance, the system usually has a backup power source (e.g., super capacitor) for saving critical information to non-volatile memory. Usually, the backup power cannot last for a very long time, therefore the information can be saved using backup power is very limited. As a result, it may not be always feasible to rewrite all the fast-write pages. However, at least for our interested enterprise applications, it is reasonable to expect that the main power will resume in a short time after a power failure. Therefore, as long as the data retention time of the fast-write pages (e.g., 4 days in our case study presented later on) is significantly longer than power downtime, the fast-write page data will not be lost. However, we still need to save rewrite queue from volatile memory (e.g., DRAM) to non-volatile memory or simply use non-volatile memory to implement rewrite queue. The designer should determine the rewrite queue depth according to the stable power period provided by backup power resources if the rewrite queue is implemented by volatile memory.

4 Simulation Setup

Quantitative evaluation of the proposed design strategy demands a simulation environment that can quantitatively model the flash memory cell device characteristics and SSD system performance. This section discusses this simulation environment and associated configurations.

4.1 Modeling of Memory Device Characteristics

We use the model presented in [6] to simulate the flash memory device characteristics including memory cell programming and erase behavior, RTN and Data Retention, and cell-to-cell interference.

4.1.1 Memory Erase and Programming

NAND flash memory uses Fowler-Nordheim tunneling (FN) to realize both erase and program, and the threshold voltage of FN-erased memory cells tends to have a wide Gaussian-like distribution. Hence, we can approximately model the threshold voltage distribution of erased state as

$$p_e(x) = \frac{1}{\sigma_e\sqrt{2\pi}} e^{-\frac{(x-\mu_e)^2}{2\sigma_e^2}}, \tag{1}$$

where μ_e and σ_e are the mean and standard deviation of the erased state threshold voltage.

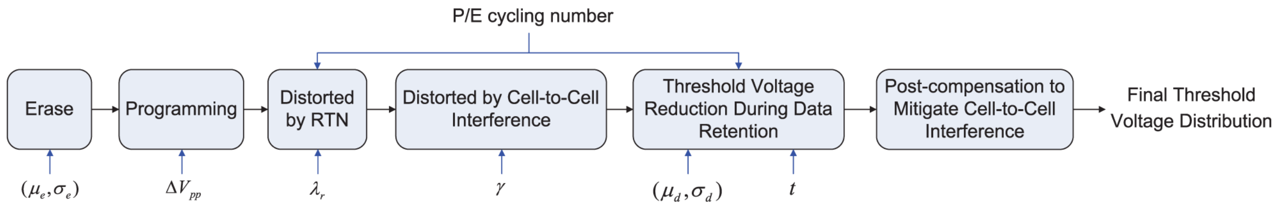

P/E cycling number
Erase
Programming
Distorted by RTN
Distorted by Cell-to-Cell Interference
Threshold Voltage Reduction During Data Retention
Post-compensation to Mitigate Cell-to-Cell Interference
Final Threshold Voltage Distribution
(μ_e,σ_e)
ΔV_{pp}
λ_r
γ
(μ_d,σ_d)
t

Fig. 7. Illustration of the approximate NAND flash memory device model that incorporates major threshold voltage distortion sources and applies post-compensation to partially mitigate cell-to-cell interference.

Regarding ISPP memory programming, the threshold voltage of the programmed state tends to have a uniform distribution over $[V_p,\, V_p + \Delta V_{pp}]$ with the width of ΔV_{pp}:

$$p_p^{(k)}(x) = \begin{cases} \frac{1}{\Delta V_{pp}}, & \text{if } V_p^{(k)} \le x \le \ V_p^{(k)} + \Delta V_{pp} \\ 0, & \text{else.} \end{cases} \tag{2}$$

4.1.2 RTN and Data Retention

RTN causes random fluctuation of memory cell threshold voltage, and we can model the probability density function $p_r(x)$ of RTN-induced threshold voltage fluctuation as a symmetric exponential function:

$$p_r(x) = \frac{1}{2\lambda_r} e^{-\frac{|x|}{\lambda_r}}. \tag{3}$$

Interface state trap generation grows with the P/E cycle number N_{cyc} in a power law fashion, with the exponent as 0.62. In this work, we set the mean of RTN follows $C_r + A_r \cdot N_{cyc}^{0.62}$, where the constant A_r =1.80E-4 and we use another constant $C_r = 1.26\text{E} - 3$ to incorporate the effects of electron injection statistics.

Memory cell threshold voltage reduction during data retention is mainly due to interface state trap recovery and electron detrapping, which approximately follow Poisson statistics. Hence, the threshold voltage reduction during data retention can be approximately modeled as a Gaussian distribution $\mathcal{N}(\mu_d, \sigma_d^2)$. The mean value of threshold voltage shift is proportional to the mean of sum of interface state traps and oxide traps, i.e., $\mu_d = A_t \cdot N_{cyc}^{0.62} + B_t \cdot N_{cyc}^{0.3}$. We set $A_t = 7\text{E} - 4$, $B_t = 4.76\text{E} - 3$, and $\sigma_d = 0.3\mu_d$ in our simulation based on the results presented in [7]. Moreover, the significance of threshold voltage reduction during data retention is also proportional to the initial threshold voltage magnitude. We set that the threshold voltage reduction during data retention approximately scales with $K_s(x - x_0)$, where x is the initial threshold voltage, and the constant x_0 and K_s are set as 1.4 and 0.38 respectively in our simulation based on the results presented in [8].

4.1.3 Cell-to-Cell Interference

Threshold voltage shift of a victim cell caused by cell-to-cell interference can be estimated as

$$F = \sum_k (\Delta V_t^{(k)} \cdot \gamma^{(k)}), \tag{4}$$

where $\Delta V_t^{(k)}$ represents the threshold voltage shift of one interfering cell which is programmed after the victim cell, and the coupling ratio $\gamma^{(k)}$ is defined as

$$\gamma^{(k)} = \frac{C^{(k)}}{C_{total}}, \tag{5}$$

where $C^{(k)}$ is the parasitic capacitance between the interfering cell and the victim cell, and C_{total} is the total capacitance of the victim cell.

4.1.4 An Approximate Memory Device Model

Based on the above discussions, we can approximately model NAND flash memory device characteristics as shown in Fig. 7, using which we can simulate memory cell threshold voltage distribution and hence obtain memory cell raw storage reliability. Based upon (1) and (2), we can obtain the distortion-less threshold voltage distribution function $p_p(x)$. Recall that $p_{pr}(x)$ denotes the RTN distribution function (see (3)), and let $p_{ar}(x)$ denote the threshold voltage distribution after incorporating RTN, which is obtained by convoluting $p_p(x)$ and $p_r(x)$:

$$p_{ar}(x) = p_p(x) \otimes_{p_r}(x). \tag{6}$$

Cell-to-cell interference is further incorporated based on (4). To capture inevitable process variability, we set both the vertical coupling ratio γ_y and diagonal coupling ratio γ_{xy} as random variables with bounded Gaussian distributions:

$$p_c(x) = \begin{cases} \frac{c_c}{\sigma_c\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu_c)^2}{2\sigma_c^2}}, & \text{if } |x - \mu_c| \ \le \ w_c \\ 0, & \text{else,} \end{cases} \tag{7}$$

where μ_c and σ_c are the mean and standard deviation, and c_c is chosen to ensure the integration of this bounded Gaussian distribution equals to 1. In this study, we set $w_c = 0.1\mu_c$ and $\sigma_c = 0.4\mu_c$, and set the mean μ of γ_y and γ_{xy} as 0.096 and 0.0072, respectively.

Let p_{ac} denote the threshold voltage distribution after incorporating cell-to-cell interference and $p_t(x)$ denote the distribution of threshold voltage reduction during data retention, we have the threshold voltage distribution after incorporating major distortion sources as

$$p_f(x) = p_{ac}(x) \otimes_{p_t}(x). \tag{8}$$

Finally, we use the post-compensation technique [9] to partially mitigate the cell-to-cell interference. In our simulation, we assume using MLC flash and set its memory read access time as $50\ \mu$s and program time as $600\ \mu$s [10] for normal speed write with retention time is 10 years. We set the data retention time for fast-write as 4 days. According

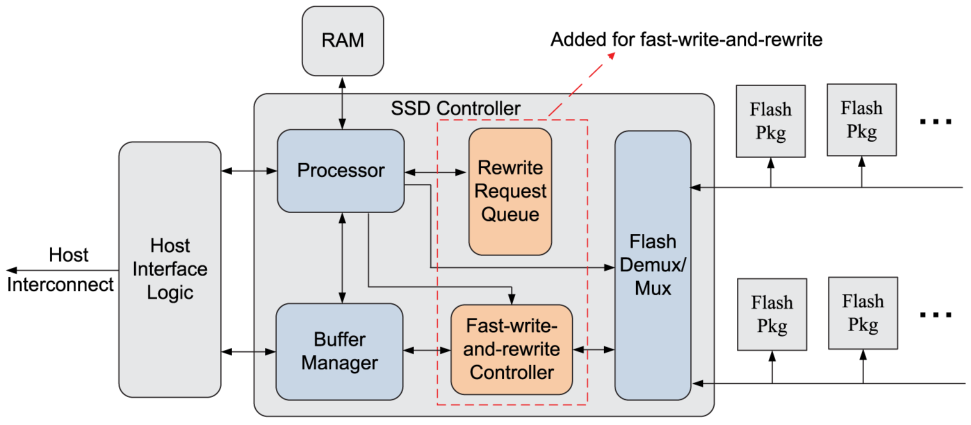


Fig. 8. Illustration of SSD structure when using the proposed design approach.

to the simulation based on our flash device model, the V_{pp} could be reduced half when the data retention time requirement decreases from 10 years to 4 days. Therefore, we set flash memory programming time for fast-write as 300 μs.

4.2 SSD System Performance Modeling

We carry out SSD system performance simulation using the DiskSim Simulator [11] with the SSD model patch [12] from Microsoft Research Lab. The SSD configuration is described as follows. Each chip contains 2 dies that share an 8-bit I/O bus and a number of common control signals, each die contains 2 planes, and each plane contains 2048 blocks. Each block contains 64 pages, each of which consists of eight 512-byte sectors. The SSD contains 2 channels (gangs), and each channel harbors 16 flash memory chips. We also configure the simulator to support data stripping over 2 channels. Following the version 2.0 of the Open NAND Flash Interface (ONFI) [13], we set the bus bandwidth as 133 MB/s. It employs a modified greedy strategy GC which keeps tracking the remaining lifetime of any block and gives them different GC rates so as to delay the expire time of any single block. The details of this algorithm can be found in [12]. As we discussed earlier, we modify the GC algorithm used in the simulation tool in order to give the fast-write blocks a higher priority than the normal-write blocks to be recollected. In our simulation, GC will be triggered if the number of free blocks is less than 20% of the number of total physical blocks. The unit size of data that corresponds to one entry on the logical-to-physical address translation table (mapping table) is 4KB and the pages in the write queue is sequentially written to available free blocks, and meanwhile the mapping table is accordingly updated. We set the system provisioning as 28%, which means the logical volume is 50 GB on a physical volume of 64 GB SSD, in order to reduce the write amplification and improve the garbage collection efficiency.

In order to evaluate our proposed design approach, we also modify the original DiskSim to support fast-write and rewrite operations as shown in Fig. 8. We use 10 different I/O traces including Iozone and Postmark from [12], Finance1, and Finance2 from [14], and Trace 1-6 from [15]. In our simulation, those benchmarks are run hundreds of times in order to invoke enough GC operations.

5 Basic Simulation Results

Before presenting the simulation results, we define the following abbreviations:

1) Nth_rw: the threshold value of the number of requests in the rewrite queue for triggering rewrite;
2) Nth_w: the threshold value of the number of requests in the write queue for triggering fast-write;
3) D_rw: the rewrite queue depth.

Fig. 9 shows the SSD response time standard deviation, which is normalized to the SSD response time standard deviation without using the proposed design approach. The average SSD response time standard deviation reduction is 39.9% over the 10 traces. Reduction of response time standard deviation comes from the average SSD response time reduction as shown in Fig. 10a. As shown in Fig. 10a, most traces can achieve a significant average response time reduction except traces Financial1 and Financial2. The average response time reduction after using the proposed design approach over the 10 different traces is 52.3%. Fig. 10b shows the number of total rewrite pages normalized to the number of total requested pages in the original trace files. The average normalized number of total rewrite pages is 14.1% over the 10 traces. Fig. 10b only shows the total rewrite operations. However, as we discussed in Section 3.1, not every rewrite will result in an

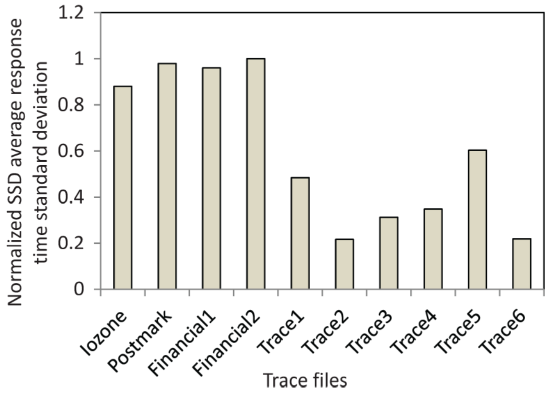


Fig. 9. SSD response time standard deviation normalized to the SSD response time standard deviation without using the proposed design approach. Nth_rw, Nth_w, and D_rw are set to 128, 128, and 64 K respectively.

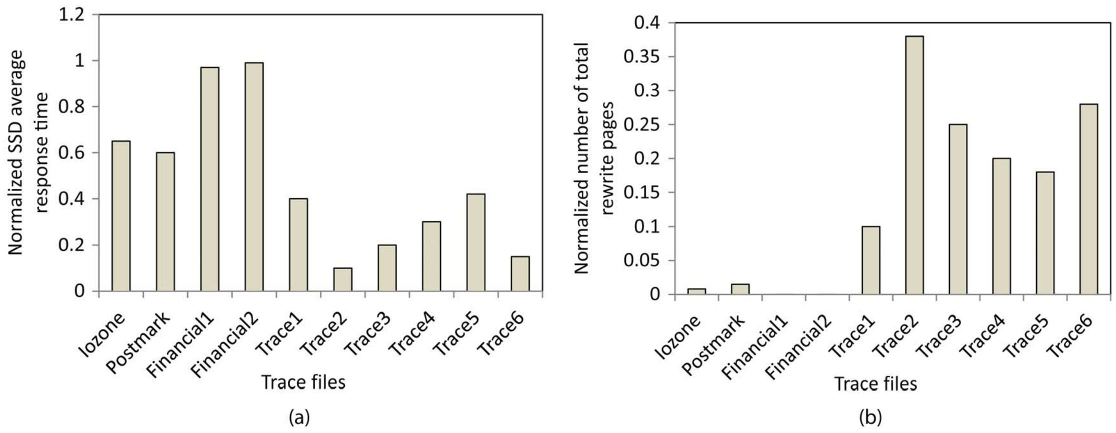


Fig. 10. (a) Average response time normalized to the average response time without using the proposed design approach and (b) number of total rewrite pages normalized to the number of total request pages in the trace files. Nth_rw, Nth_w, and D_rw are set to 128, 128, and 64 K respectively.

extra write operation to the SSD compared to the scenario without using the proposed design approach. Hence, Fig. 11 shows the normalized number of extra write pages when GC is taken into consideration. The average number of extra write pages is 6.1% over the 10 traces. Compared with 52.3% average response time reduction and 39.9% average response time standard deviation reduction, 6.1% extra write pages appears to be justifiable. In addition, the number of extra write pages can be further reduced using the trade-off between the average response time reduction and the number of total rewrite pages, which can be adjusted by changing D_rw, Nth_w, and/or Nth_rw. We will further quantitatively illustrate these trade-offs in Section 6.

Intuitively, the extra rewrite operations may have negative impacts on the average response time. However, all the above real workloads do not reveal this effect. Therefore, we use six synthetic workloads generated in DiskSim to more clearly illustrate this effect. These six synthetic traces are all composed of 10 billion random write (RW10B) I/O requests with the mean of inter-arrival time varying from 10 ms to 0.02 ms and the deviation is fixed at 0. Simulation results shown in Fig. 12a suggest that, when the inter-arrival time is long (e.g., 10 ms, 5 ms, and 1 ms), the average response time degradation is negligible. This is because very few rewrite operations are triggered, since the number of requests in the write queue is always very small. On the other hand, when the inter-arrival time is very short and the requests are evenly distributed (deviation of inter-arrival time is set to 0), the average response time marginally increases due to the negative impacts of the rewrite operations. Nevertheless, the read response time degradation is so small that it is almost negligible. The reason is that the rewrite queue is only 64 K, therefore, after rewrite queue is full no more rewrite operations will be triggered until the rewrite queue is forced cleaned after the retention time limit (i.e., 4 days) is approaching.

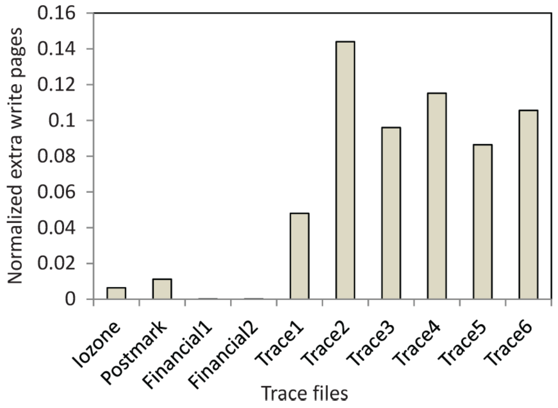


Fig. 11. Normalized number of extra write pages when GC is taken into consideration. Nth_rw, Nth_w, and D_rw are set to 128, 128, and 64 K respectively.

6 Sensitivity Analysis

The previous section shows the simulation results with a single set of configurations, i.e., D_rw is 64 K, Nth_rw is 128, and Nth_w is 128. Clearly, the average response time reduction and number of total rewrite pages could vary with different configurations. In order to show the impact and involved trade-offs, we present sensitivity analysis in this section.

6.1 Rewrite Queue Depth

As we discussed in Section 3.1, the rewrite queue depth could affect the trade-off between the average response time reduction and the number of total rewrite pages. A bigger rewrite queue depth D_rw may bring more fast-write operations and potentially improve the rewrite queue hit rate. Therefore, a bigger D_rw results in more average response time reduction as shown in Fig. 13. At the same time, a bigger D_rw also generates more extra write operations as shown in Fig. 14. All the traces except traces Financial1 and Financial2 show this trade-off when we vary D_rw. We also notice that the average response time of trace1$\sim$ 6 cannot be further reduced when D_rw is bigger than 20 K. The reason is that the dynamic page numbers of rewrite queue for traces1$\sim$ 6 never exceed 20 K according to our simulation results. We further notice that the impact of rewrite operations on response time is not significant, since most of the rewrite operations occur when the SSD is idle.

6.2 Fast-Write Trigger Threshold

As shown in Fig. 10a, the proposed design approach fails to noticeably reduce the average response time for traces Financial1 and Financial2. The reason is that Nth_w, which is set to

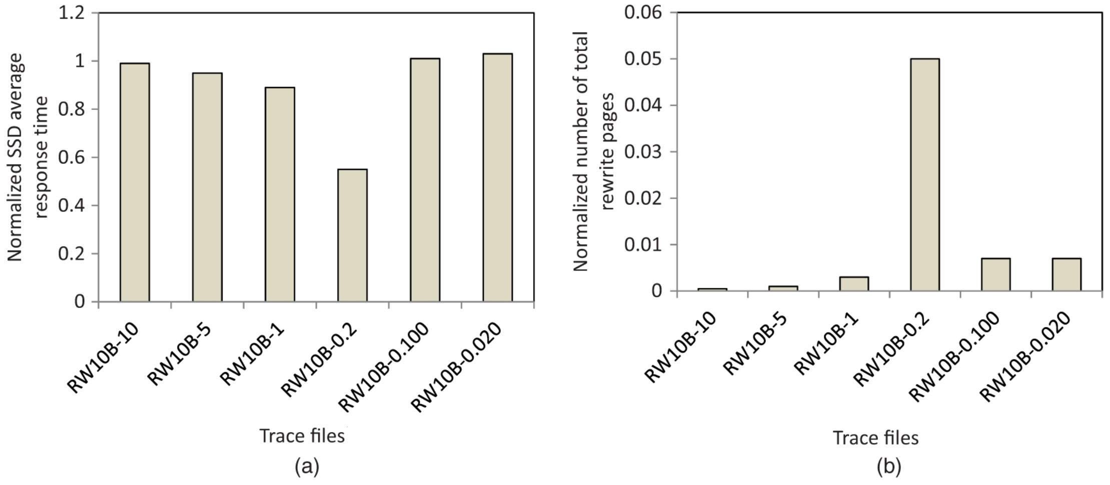


Fig. 12. (a) Average response time normalized to the average response time without using the proposed design approach and (b) Number of total rewrite pages normalized to the number of total request pages in the trace files. Nth_rw, Nth_w, and D_rw are set to 128, 128, and 64 K respectively.

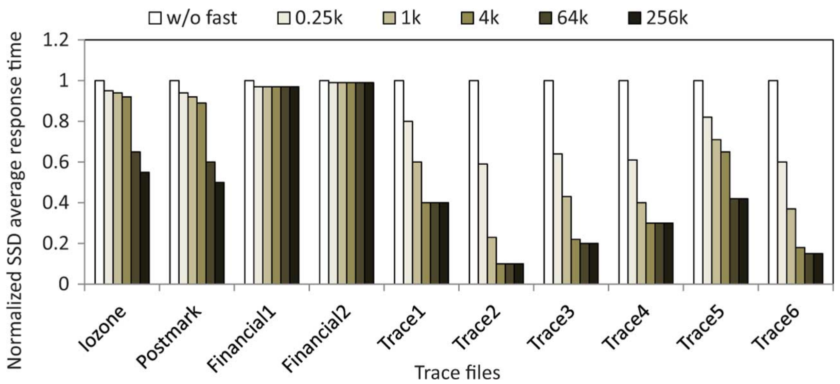


Fig. 13. Normalized average response time under different rewrite queue depth under different trace files. Nth_rw is set to 128. Nth_w is set to 128.

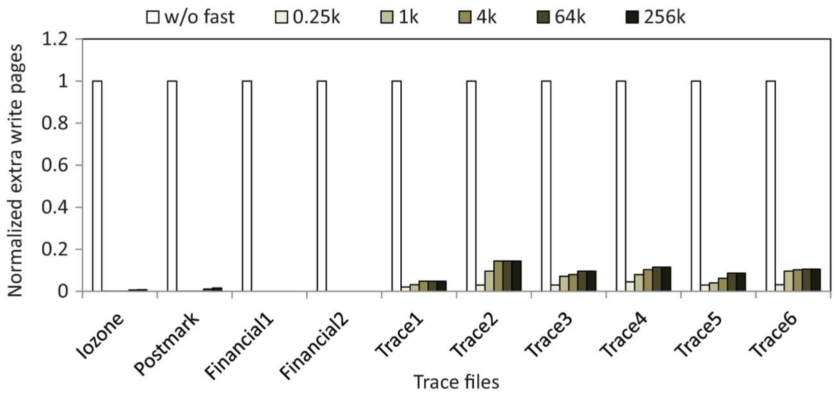


Fig. 14. Normalized number of total extra write pages with different rewrite queue depth under different trace files. Nth_rw is set to 128. Nth_w is set to 128.

128 in the above simulations, is too high for those two traces. Hence, we re-run these two traces under very small Nth_w, and the results are shown in Fig. 15. Now we can observe noticeable average response time reduction for these two traces, which is up to 16.3% and 8% with 6% and 1.2% extra number of rewrite pages for Financial1 and Financial2, respectively.

Fig. 16 shows the normalized average response time when Nth_w varies from 32 to 2048 for all the traces. A smaller Nth_w generates more fast-write operations, which tends to result in a larger average response time reduction as shown in the results of traces1$\sim$ 6. However, the traces Iozone and Postmark show opposite results. The reason is that the rewrite

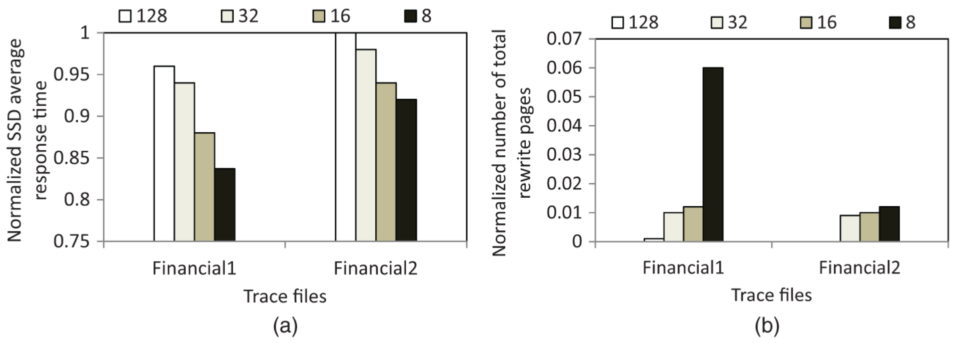


Fig. 15. (a) Average response time normalized to the average response time without using the proposed design approach and (b) number of total extra write pages normalized to the number of total request pages in the trace files. Nth_rw is set to 128. D_rw is set to 64 K and Nth_w is varied from 128 to 8.

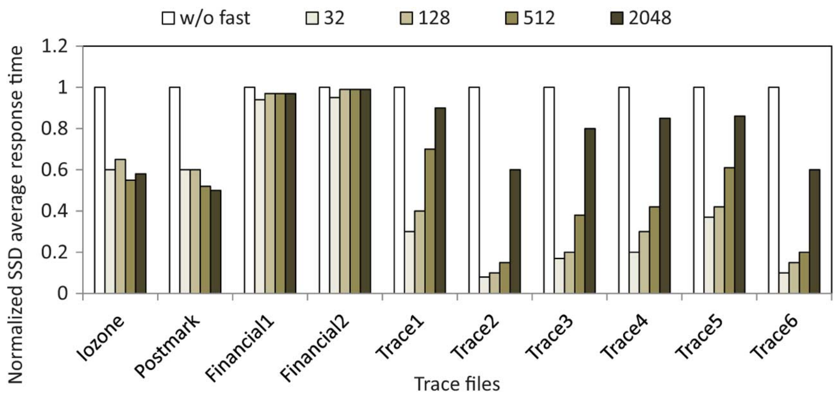


Fig. 16. Normalized average response time under different Nth_w. Nth_rw is set to 128. D_rw is set to 64 K.

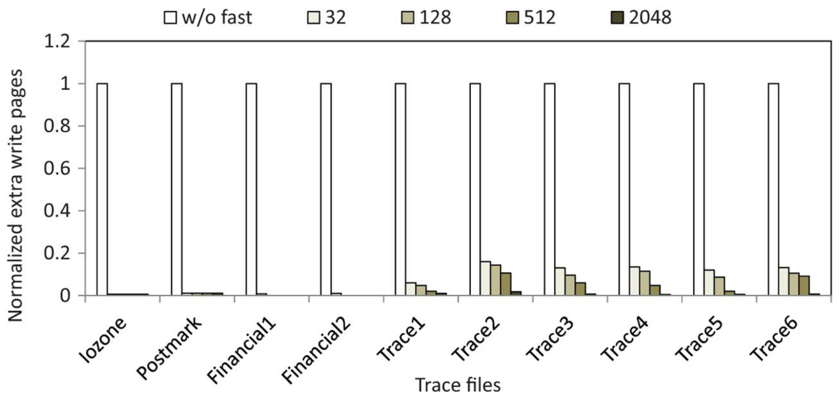


Fig. 17. Normalized number of total extra write pages under different Nth_w. Nth_rw is set to 128. D_rw is set to 64 K.

queue is already full when running these two traces. Therefore, a bigger Nth_w can postpone the overflow of rewrite queue, which can potentially improve the SSD average response time reduction as shown by the traces Iozone and Postmark in Fig. 16. The number of total extra write pages, as shown in Fig. 17, continuously drops as Nth_w increases from 32 to 2048, since less rewrite operations are generated as Nth_w increases.

6.3 Rewrite Trigger Threshold

Fig. 18a shows the rewrite queue hit rate for different traces, for which we assume that the SSD controller can have a page to rewrite whenever a SSD channel is idle. As we discussed in Section 3.2, this could lead to a low rewrite queue hit rate for the traces which are very unbalanced or intensive. As shown in Fig. 18a, only trace Iozone has a high hit rate since it is very intensive and does not have much idle time. Therefore, its rewrite queue cannot be flushed frequently. If we add a new constraint (i.e., read a page location from rewrite queue only when the length of rewrite queue is bigger than a threshold), the rewrite queue hit rate could improve. As shown in Fig. 18b, the hit rate of those trace significantly improves by setting the rewrite trigger threshold value as 2k. Consequently, the number of total extra write pages can be reduced as shown in Fig. 20. The average response time of traces1$\sim$ 6, as shown in Fig. 19, does not have noticeable change as the threshold

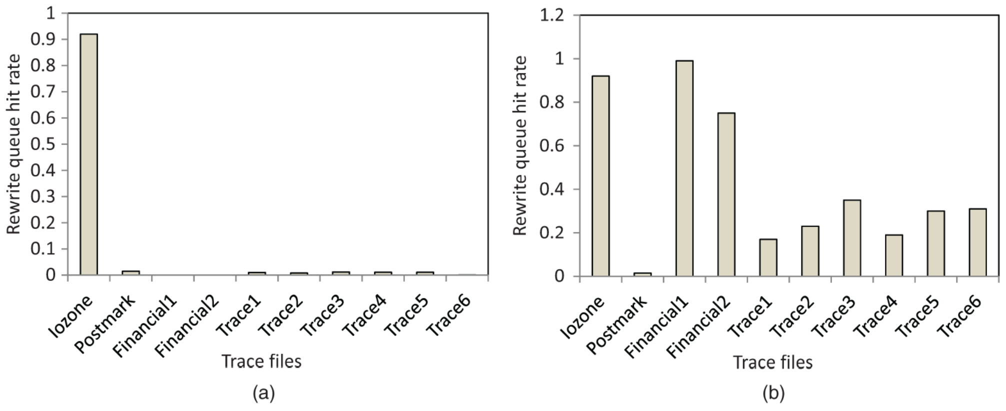


Fig. 18. (a) Rewrite queue hit rate when the SSD controller gets a page number from rewrite queue whenever a SSD channel is idle and (b) rewrite queue hit rate when the SSD controller gets a page number from rewrite queue when a SSD channel is idle and the number of entries in rewrite queue is bigger than 128. D_rw is set to 64 K and the threshold value of Nth_w is set to 128.

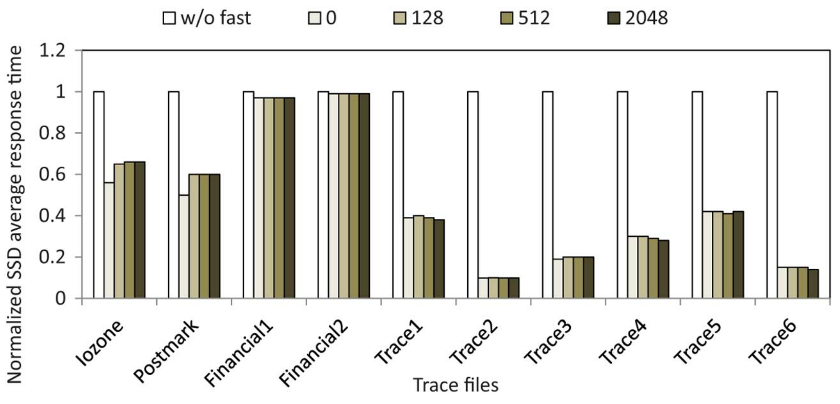


Fig. 19. Normalized average response time under different Nth_rw. D_rw is set to 64 K. Nth_w is set to 128.

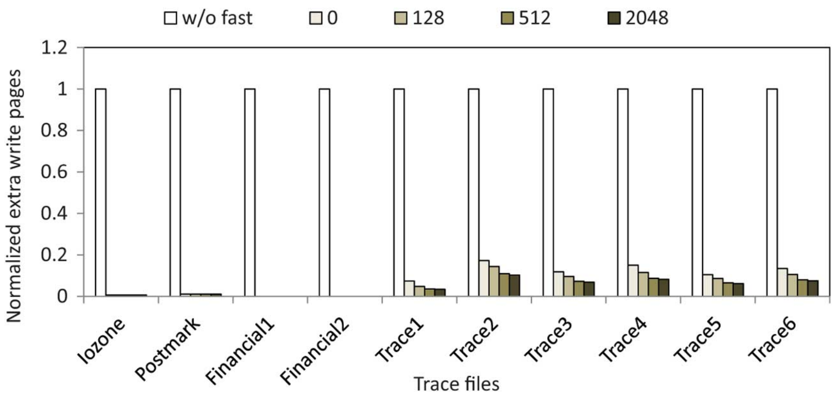


Fig. 20. Normalized number of total extra write pages under different Nth_rw. D_rw is set to 64 K. Nth_w is set to 128.

value varies, since most rewrite operations occur during the SSD idle time. The response time of Iozone and Postmark drops as threshold value decreases, which is due to rewrite queue overflow as we discussed in Section 6.2.

7 Related Work

There have been many studies [16]-[27] in the open literature aiming to improve the SSD write performance. Prior work [16]-[21] focused on using a RAM based write buffer to boost SSD write performance. Write requests are first stored in a write buffer. If the write buffer is full, a page selected through a replacement algorithm in the write buffer will be replaced and written to the flash memory. These prior work proposed different buffer management schemes, e.g., different replacement algorithms, to improve the performance of write buffer. Our work is completely orthogonal to their approaches and can be combined together to further improve the SSD write performance. Of course, if the write buffer is large enough to always absorb all the write traffic and there are no overlapped requests to the flash memory, our proposed approach should not be used.

Some prior work [22]-[27] focused on using hybrid SLC/MLC flash memory to improve SSD write performance. Write requests are first stored in SLC region at a higher speed and then migrated to MLC region later on if necessary. Prior work proposed different region partition and data migration strategies. Our work is also completely orthogonal to their approaches and can be combined together to further improve the SSD write performance.

Pan et al. [6] proposed to dynamically adjust flash programming speed according to the run-time P/E cycling number in such a way that the memory raw storage bit error rate is always close to what can be maximally tolerated by the existing redundancy. Our approach also aims to dynamically adjust flash memory programming speed. However, we change the programming speed according to the workload dynamic characteristic instead of run-time P/E cycling number.

8 Conclusion

This paper presents a design approach to improve the SSD write performance by exploiting run-time data access workload variation and temporal locality at the system level and NAND flash memory write latency vs. data retention time trade-off at the device level. The basic idea is to perform unconventional fast write at the penalty of data retention time when the run-time workload is intensive and rewrite those fast-written pages when the SSD is idle. We develop various techniques to facilitate the practical implementation of this design concept to reduce the impact on cycling endurance. The effects of various design parameters in this design approach are also discussed. The simulation and analysis results based on detailed flash memory cell models and SSD system simulator suggest that up to 52.3% SSD average response time reduction can be achieved.

Acknowledgment

The work was supported in part by NSF Grant CNS-1162152.

References

[1] *Fusion-io, ioDrive Octal*, http://www.fusionio.com/products/iodrive-octal/, 2011.

[2] A. Walls, “Solid State Storage Architectures in the Modern Data Center,” *Proc. Flash Summit,* 2011.

[3] Y. Li, “128 gb 3-Bit per Cell NAND Flash Memory on 19 nm Technology,” *Proc. Flash Memory Summit,* Aug. 2012.

[4] K.-D. Suh et al., “A 3.3 V 32 Mb NAND Flash Memory with Incremental Step Pulse Programming Scheme,” *IEEE J. Solid-State Circuits*, vol. 30, no. 11, pp. 1149-1156, Nov. 1995.

[5] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, “Introduction to Flash Memory,” *Proc. IEEE*, vol. 91, no. 4, pp. 489-502, Apr. 2003.

[6] Y. Pan, G. Dong, and T. Zhang, “Exploiting Memory Device Wear-out Dynamics to Improve NAND Flash Memory System Performance,” *Proc. 11th USENIX Conf. File and Storage Technologies (FAST),* Feb. 2011.

[7] N. Mielke et al., “Flash EEPROM Threshold Instabilities Due to Charge Trapping during Program/Erase Cycling,” *IEEE Trans. Device and Materials Reliability*, vol. 4, no. 3, pp. 335-344, Sep. 2004.

[8] J.D. Lee, J.H. Choi, D. Park, and K. Kim, “Effects of Interface Trap Generation and Annihilation on the Data Retention Characteristics of Flash Memory Cells,” *IEEE Trans. Device and Materials Reliability*, vol. 4, no. 1, pp. 110-117, Mar. 2004.

[9] G. Dong, S. Li, and T. Zhang, “Using Data Post-Compensation and Pre-Distortion to Tolerate Cell-to-Cell Interference in MLC NAND Flash Memory,” *IEEE Trans. Circuits and Systems-I: Regular Papers*, vol. 57, no. 10, pp. 2718-2728, Oct. 2010.

[10] R. Micheloni, M. Picca, S. Amato, H. Schwalm, M. Scheppler, and S. Commodaro, “Non-Volatile Memories for Removable Media,” *Proc. IEEE*, vol. 97, no. 1, pp. 148-160, Jan. 2009.

[11] *The DiskSim Simulation Environment Version 3.0 Reference Manual.0*, http://www.pdl.cmu.edu/DiskSim/, 2011.

[12] N. Agrawal, V. Prabhakaran, T. Wobber, J.D. Davis, M. Manasse, and R. Panigrahy, “Design Tradeoffs for SSD Performance,” *Proc. USENIX Ann. Technical Conf.*, pp. 57-70, June 2008.

[13] *Open NAND Flash Interface Specification*, http://onfi.org/specifications/, 2011.

[14] *SPC Trace File Format Specification*, http://traces.cs.umass.edu/index.php/Storage/Storage, 2011.

[15] C. Dirik and B. Jacob, “The Performance of PC Solid-State Disks (SSDs) as a Function of Bandwidth, Concurrency, Device Architecture, and System Organization,” *SIGARCH Computer Architecture News*, vol. 37, pp. 279-289, 2009.

[16] H. Kim and S. Ahn, “BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage,” *Proc. 6th USENIX Conf. File and Storage Technologies (FAST)*, pp. 239-252, Feb. 2008.

[17] I. Koltsidas and S.D. Viglas, “Flashing Up the Storage Layer,” *Proc. 34th Int'l Conf. Very Large Data Bases (VLDB)*, vol. 1, no. 1, pp. 514-525, Aug. 2008.

[18] B. Eckart, G. Wu, and X. He, “BPAC: An Adaptive Write Buffer Management Scheme for Flash-Based Solid State Drives,” *Proc. IEEE 26th Symp. Mass Storage Systems and Technologies (MSST)*, pp. 1-6, May 2010.

[19] T. Xie and J. Koshia, “Boosting Random Write Performance for Enterprise Flash Storage Systems,” *Proc. IEEE 26th Symp. Mass Storage Systems and Technologies (MSST)*, pp. 1-10, May 2011.

[20] S. Jiang, X. Ding, F. Chen, E. Tan, and X. Zhang, “DULO: An Effective Buffer Cache Management Scheme to Exploit Both Temporal and Spatial Locality,” *Proc. 6th USENIX Conf. File and Storage Technologies (FAST)*, pp. 239-252, Feb. 2008.

[21] S.Y. Park, D. Jung, J.U. Kang, J.S. Kim, and J. Lee, “CFLRU: A Replacement Algorithm for Flash Memory,” *Proc. Int'l Conf. Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, pp. 234-241, 2006.

[22] S. Hong and D. Shin, “NAND Flash-Based Disk Cache Using SLC/MLC Combined Flash Memory,” *Proc. Int'l Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, pp. 21-30, May 2010.

[23] L.P. Chang, “Hybrid Solid-State Disks: Combining Heterogeneous Nand Flash in Large SSDs,” *Proc. Asia and South Pacific Design Automation Conf. (ASP-DAC)*, pp. 428-433, 2008.

[24] T. Kgil, D. Roberts, and T. Mudge, “Improving NAND Flash Based Disk Caches,” *Proc. Int'l Symp. Computer Architecture*, pp. 327-338, 2008.

[25] J.M. Jeong, J.H. Kim, S.H. Park, J.W. Park, and S.D. Kim, “A Mixed Flash Translation Layer Structure for SLC-MLC Combined Flash Memory System,” *Proc. SPEED,* 2008.

[26] S. Lee, K. Ha, K. Zhang, J. Kim, and J. Kim, “FlexFS: A Flexible Flash File System for MLC NAND Flash Memory,” *Proc. USENIX Technical Conf.*, p. 9, 2009.

[27] S. Im and D. Shin, “Storage Architecture and Software Support for SLC/MLC Combined Flash Memory,” *Proc. ACM Symp. Applied Computing (SAC)*, pp. 1664-1669, 2009.

Qi Wu (M'06-SM'11) received the BS and MS degrees in electrical engineering from Xian Jiaotong University, China, in 2003 and 2006, respectively. He received the PhD degree in electrical engineering from Rensselaer Polytechnic Institute, Troy, New York, in 2011. Currently he is a principle engineer with Skyera Inc., San Jose, California. He is in charge of defining the architecture of next generation enterprise Solid-State Drive products. His research interests include circuit and architecture design for memory and data storage systems, including 3D DRAM, NAND Flash, and Phase-Change Memory-based SSDs.

Tong Zhang (M'02-SM'08) received the BS and MS degrees in electrical engineering from Xian Jiaotong University, China, in 1995 and 1998, respectively. He received the PhD degree in electrical engineering from the University of Minnesota, Minneapolis, in 2002. Currently he is an associate professor with Electrical, Computer and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, New York. His current research interests include circuits and systems for data storage, signal processing, and computing. Currently, he serves as an associate editor for the *IEEE Transactions on Circuits and Systems*-II and the *IEEE Transactions on Signal Processing*.