Practical Erase Suspension for Modern Low-latency SSDs

Shine Kim, *Seoul National University and Samsung Electronics;*
Jonghyun Bae, *Seoul National University;* Hakbeom Jang, *Sungkyunkwan University;*
Wenjing Jin and Jeonghun Gong, *Seoul National University;* Seungyeon Lee,
Samsung Electronics; Tae Jun Ham and Jae W. Lee, *Seoul National University*

https://www.usenix.org/conference/atc19/presentation/kim-shine

This paper is included in the Proceedings of the 2019 USENIX Annual Technical Conference.

July 10–12, 2019 • Renton, WA, USA

ISBN 978-1-939133-03-8

Open access to the Proceedings of the 2019 USENIX Annual Technical Conference is sponsored by USENIX.

Practical Erase Suspension for Modern Low-latency SSDs

Shine Kim[†‡], Jonghyun Bae[†], Hakbeom Jang*, Wenjing Jin[†], Jeonghun Gong[†]
Seungyeon Lee[‡], Tae Jun Ham[†], Jae W. Lee[†]
[†]*Seoul National University,* **Sungkyunkwan University,* [‡]*Samsung Electronics*

Abstract

As NAND flash technology continues to scale, flash-based SSDs have become key components in data-center servers. One of the main design goals for data-center SSDs is low read tail latency, which is crucial for interactive online services as a single query can generate thousands of disk accesses. Towards this goal, many prior works have focused on minimizing the effect of garbage collection on read tail latency. Such advances have made the other, less explored source of long read tails, *block erase operation*, more important. Prior work on erase suspension addresses this problem by allowing a read operation to interrupt an ongoing erase operation, to minimize its effect on read latency. Unfortunately, the erase suspension technique attempts to suspend/resume an erase pulse at an arbitrary point, which incurs additional hardware cost for NAND peripherals and reduces the lifetime of the device. Furthermore, we demonstrate this technique suffers a write starvation problem, using a real, production-grade SSD. To overcome these limitations, we propose alternative *practical* erase suspension mechanisms, leveraging the iterative erase mechanism used in modern SSDs, to suspend/resume erase operation at well-aligned safe points. The resulting design achieves a sub-200μs 99.999th percentile read tail latency for 4KB random I/O workload at queue depth 16 (70% reads and 30% writes). Furthermore, it reduces the read tail latency by about 5$\times$ over the baseline for the two data-center workloads that we evaluated with.

1 Introduction

NAND flash-based SSDs offer superior throughput and average latency compared to those of hard disks and thus have become the de-facto standard for storage devices. However, SSDs have much greater performance variability than that of hard disks [11]. While SSDs can achieve very low average read latency (e.g., under 15μs [5]), their tail latency (e.g., 99.999th percentile) can be very long (e.g., over 10ms). Figure 1 demonstrates this issue with a real low-latency SSD.

One can mistakenly think that long tail read latency is a rare event that affects very few requests. However, such tail latency can play a considerable role in data-center computing because a single query (e.g., web search) may require thousands of disk reads across the data-center [2,7]. In such a case, a single long-latency disk access can lead to an increase in the overall query response time. Also, the chance of a query experiencing long disk tail latency is continuously increasing with the trend of ever-increasing data size.

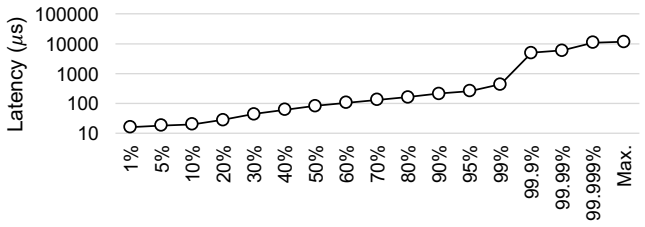

Figure 1: Read latency distribution of a PCIe 3×4 NVMe low-latency SSD [5] running 4KB random reads (70%) and writes (30%) workload with a queue depth of 16.

To avoid such performance degradation in a data-center induced by SSD tail latency, minimizing the tail latency of a read operation is very important. Naturally, this requires tackling two significant sources of long read tails: garbage collection (GC) and erase operations. Recent prior works have already explored ways to minimize the effect of GC on read tail latency [6, 14, 37]. Furthermore, production-level low-latency SSDs — such as Samsung Z-SSD [38], which we base our work on — have already minimized the effect of GC on read operation [38] by allowing the user-initiated read to be processed between operations in a GC (i.e., read and program to copy valid pages from one block to another) [4, 20]. In contrast, much less attention has been paid to the effect of erase operation on read tail latency, primarily because erase latency is relatively small (e.g., 5ms) compared to the effect of GC on tail latency (e.g., can exceed 100ms without any optimization). However, with techniques minimizing this effect, erase latency is now becoming the most dominant component of read tail latency.

To control the effect of erase on tail latency, Wu et al. [35] proposed an erase suspension technique, which suspends an ongoing erase (and verify) pulse when a read request is issued to the same flash die. After processing the read request, the erase pulse resumes from the exact point at which it was suspended. However, the commodity NAND flash business is known to be extremely cost-sensitive, and this technique may increase the cost of NAND peripherals to generate an erase pulse of an arbitrary length and track the exact state of every erase. Furthermore, it can cause a serious NAND reliability problem and write starvation.

To address these limitations, we present *practical* erase suspension schemes for modern low-latency SSDs. Instead

of suspending/resuming an erase pulse at an arbitrary point, our work focuses on suspending/resuming the erase operation at well-aligned safe points by either i) aborting an ongoing erase operation immediately and resuming from the last safe point or ii) deferring the suspension of erase operation until the next safe point. Exploiting their trade-offs, we also introduce a timeout-based switching mechanism between the two mechanisms, to adapt to workload changes dynamically. This scheme enables modern low-latency SSDs to offer extremely low read tail latency on a wide range of workloads without causing any NAND reliability problem or write starvation.

Our contributions are summarized as follows:

- We are the first to identify the problems of NAND reliability and write starvation in the existing erase suspension scheme and demonstrate the latter on a real, production-grade SSD.
- We propose two practical erase suspension mechanisms, *immediate* erase suspension and *deferred* erase suspension. We also analyze the trade-offs between the two mechanisms and introduce a timeout-based switching policy between the two, to take the best of both as the workload changes.
- We demonstrate a significant reduction in read tail latency with the proposed erase suspension mechanisms on various workloads including Aerospike Certification Tool (ACT) [1] and TPC-C benchmark workloads [31].

2 Practical Erase Suspension

2.1 Motivation

To perform a NAND block erase, the incremental step pulse erasing scheme is a standard feature in modern SSDs [18]. Instead of utilizing a single, very high voltage pulse (e.g., 14V) for an erase, which has negative impact on NAND lifetime [12, 25, 29], this scheme performs an erase operation with several, discrete pulses (typically 5 or fewer), and each pulse has a higher nominal voltage than the previous one. Figure 2 illustrates this scheme. By verifying the set of erased cells between erase pulses and by applying higher voltage pulses to cells that are not erased yet, this scheme minimizes damage on NAND cells [12]. A single erase pulse consists of the following 3 stages: ① *voltage ramping* stage in which the erase pulse reaches the desired voltage, ② *erase execution* stage during which the voltage is stabilized and maintained, and ③ *voltage recovery* stage in which the erase voltage is reset for the erase-verify operation.

The erase suspension mechanism has been proposed to provide tight read tail latency by suspending an ongoing erase pulse at the arrival of a read request to be resumed later [35]. While effective in reducing read tail latency, this mechanism poses several implementation challenges in terms of NAND reliability and cost. First, an erase suspension adds to the cost of an erase an extra pair of voltage recovery (③) and ramping (①) stages for suspending and resuming the erase pulse. These additional stresses caused by ramping up and down the voltage degrade the endurance of NAND [12, 25, 29]. For example, a recent case study using sub-20nm TLC NAND shows that the *raw bit error rate (RBER)* is increased by 14.4% with only 1000 erase suspensions [26]. The increased RBER leads to an increase in *uncorrectable bit error rate (UBER)* even with error correction, to eventually reduce the lifetime of SSD [3].

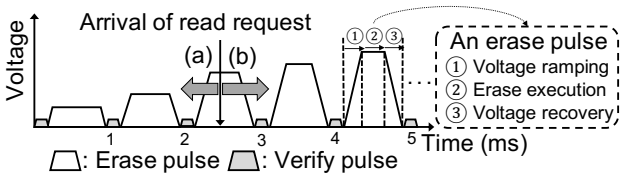

Figure 2: Incremental step pulse erase and practical erase suspension mechanisms: (a) Immediate erase suspension (I-ES) and (b) Deferred erase suspension (D-ES).

Another limitation of the existing erase suspension scheme is that it requires the capability to suspend and resume an erase pulse at an arbitrary point. This increases the cost for NAND peripherals to generate a pulse of an arbitrary length, track the exact state of each suspended erase, and recover the peripheral state to resume. As NAND cells continue to scale with the introduction of 3D NAND, NAND peripherals are becoming a scalability bottleneck to a greater extent [19, 24, 27]. Considering the extreme cost sensitivity of the commodity NAND business, this is a significant drawback.

Instead, our erase suspension scheme allows erase suspension only at the beginning or the end of each erase step. When a read request arrives while an erase pulse is still asserted, our scheme asserts an erase suspension command that either i) aborts the ongoing erase pulse and forfeits the current erase step progress to serve the read request immediately (named *immediate erase suspension*) or ii) finishes the ongoing erase step and serves the read request (named *deferred erase suspension*). The next subsections discuss each option in detail and also present an adaptive switching scheme between our two proposed mechanisms.

2.2 Immediate Erase Suspension (I-ES)

One way to serve an incoming read request during the erase pulse is to immediately terminate the ongoing erase step and to forfeit the progress (Figure 2(a)). Then, after serving the read request(s), the erase operation can resume from the beginning point of the current erase step. We call this scheme *Immediate Erase Suspension (I-ES)* since it immediately cancels the ongoing erase step.

In effect, I-ES is a practical variant of the original erase suspension [35] with the following two changes. To improve NAND reliability, a verify pulse is applied before resuming a suspended erase pulse, exploiting the incremental step

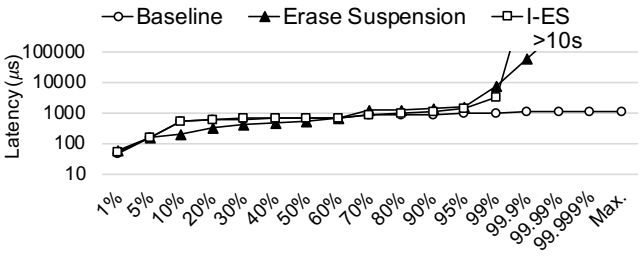

Figure 3: Write starvation experiment results. This workload consists of two threads where one thread continuously generates 128KB read requests, while another thread continuously generates 128KB write requests (QD 1 for both). Baseline and I-ES use real low-latency SSD [5], and Erase Suspension [35] uses an SSD simulator [33].

pulse erasing scheme in Section 2.1. As each NAND cell in a NAND block has different erase timing, some NAND cells get erased more quickly than others [12,25,26,29]. Therefore, applying the same pulse to all NAND cells when resuming the erase pulse results in over-erasure of some cells, which puts unnecessary stress on already erased cells and harms the reliability of NAND. A verify pulse, before resuming the erase pulse, detects already erased cells to not inflict unnecessary stress on these cells in a NAND block. Furthermore, to keep the cost of NAND peripherals low, a whole step pulse is asserted at every resumption, instead of the remaining step pulse time. To generate an erase pulse of an arbitrary length, a variable-length pulse generator with fine-grained control must be placed on a NAND device. However, today's commodity NAND does not provide such a mechanism. In contrast, I-ES does not require such changes keeping the cost of NAND low.

Advantages and Disadvantages. Since the erase step is canceled immediately, the read command can always be guaranteed the highest priority. The read command does not experience any other delay caused by an erase operation except for a fixed latency (e.g., ~100μs) that tries to cancel the ongoing erase pulse (③ in Figure 2).

However, this scheme is problematic. With the continuous incoming read requests, an erase step may be repeatedly canceled, preventing write requests (which are dependent on the erase requests) from completion. To confirm this behavior, we modified the firmware of the real, production-grade low-latency SSD [5] to implement this scheme. Figure 3 shows the outcome of this experiment. As expected, a few write operations cannot finish for a long time because the preceding erase operation is continuously canceled by incoming read requests. As a result, its tail latency keeps increasing until the end of the workload.

The erase (and write) starvation problem occurs because the latency of erase suspension/resuming operation is longer than the incoming rate of read requests on the NAND chip. For example, if the throughput of PCIe Gen 3×4 NVMe interface is 3.2GB/s (i.e., the incoming read rate is 1.19μs/4KB) and the erase suspension/resuming overhead is 100μs, it is possible for a read request to arrive while the erase suspension/resumption is happening. In such cases, the erase will immediately suspend again without making any forward progress, and thus the erase (and write) operation will starve. Note that the same problem manifests with the original erase suspension scheme [35], which differs from I-ES only for the duration of the asserted pulse at resumption, as shown in Figure 3. NVMe and PCIe specifications require the host to reset both hardware and software at write starvation (i.e., user's write requests timeout) [23, 28], which adversely affects system performance [22] and may eventually lead to a fatal system failure if repeated.

2.3 Deferred Erase Suspension (D-ES)

Another way to serve an incoming read request during the erase pulse is to let the read request wait until the current erase step finishes (Figure 2(b)). After that, in lieu of proceeding to the next erase step, the read request(s) is (are) served. Then, the erase operation resumes and continues to the next erase step. We call this *Deferred Erase Suspension (D-ES)* as it defers erase suspension until the end of the current erase step.

Advantages and Disadvantages. This mechanism lets the erase operation finish without incurring the write starvation problem. By guaranteeing a single erase step to be performed once the erase step is initiated, this mechanism guarantees forward progress for the erase operation.

Although this mechanism does not incur erase (and write) starvation, it can harm read tail latency by making read requests wait until the end of the current erase step (i.e., 1ms in our low-latency NAND). This scheme can also show worse read latency for bursty reads as D-ES adds extra latency to reads. However, such situations could be avoided by batching backlogged erases after serving the bursty reads first. This issue is addressed by T-ES, which switches adaptively between I-ES and D-ES.

2.4 Timeout-based Erase Suspension (T-ES)

Timeout-based Erase Suspension Policy. If it is possible to know the request pattern of an application *a priori*, one should use I-ES when the application is expected to have a phase with sparse read requests as in this case erase (and write) starvation does not occur as an erase is likely to make progress during the upcoming sparse read period. On the other hand, if an application is expected to have a steady stream of incoming read requests, the user should use D-ES as in this case, employing I-ES leads to an exponential increase in write tail latency (i.e., write starvation).

Unfortunately, in reality, it is not easy to predict future I/O access patterns without profiling runs. Thus, we propose a *Timeout-based Erase Suspension (T-ES)* scheme, which performs I-ES until the erase operation is delayed for *N*ms. If the erase operation is delayed for *N*ms (i.e., timeout happens), this scheme switches to D-ES mode to avoid potential erase (and write) starvation. Hopefully, this scheme finds a period to perform an erase operation without being interrupted by a

Table 1: Parameters used to model our low-latency SSD [5].

PCIe Gen 3×4 Lane, 240GB, NVMe SSD Device	
NAND Configurations	4 channels, 4 chips/channel, 1 die/chip
DRAM, Flash Speed Rate	1600MT/s, 1200MT/s (MT/s: Mega Transfers per Second [9])
FTL Schemes	Page Mapping, Preemptible GC [20]
Over-provisioning Ratio	7%
NAND Structure	
128Gb die capacity, 8 planes per die, 683 blocks per plane, 768 pages per block, 4KB page	
NAND Latency	
Read: 3μs, Program: 100μs, Block Erase: 1ms per step (5 steps), Erase Suspension Penalty: 100μs	

Table 2: Throughput and average latency of low-latency SSD prototype and MQSim running various I/O pattern workloads.

	Low-Latency SSD / MQSim	
Seq. read (256KB)	3300 / 3250 MiB/s	
Seq. write (256KB)	2700 / 3100 MiB/s	
	Kilo I/O per seconds	Average latency
Rnd. read (4KB, QD 1)	59 / 65 KIOPS	16.9 / 15.3μs
Rnd. write (4KB, QD 1)	61 / 66 KIOPS	16.4 / 15.2μs
Rnd. read (4KB, QD 32)	790 / 790 KIOPS	40.5 / 40.5μs
Rnd. write (4KB, QD 32)	61 / 66 KIOPS	524 / 484μs

read operation.

Choice of Erase Timeout Delay. T-ES covers a spectrum of policies between I-ES and D-ES, controlled by the value of N. If we set it to 0, this is equivalent to the base D-ES scheme. In contrast, if we set it to infinite, this is equivalent to I-ES scheme prone to erase (and write) starvation problem. In general, choosing a higher value offers more chance to provide better read tail latency by delaying an erase operation, but this choice leads to an increase in maximum write tail latency. On the other hand, choosing a smaller N makes it behave more like a D-ES, which provides smaller maximum write tail latency at the expense of an increased number of reads experiencing erase latency. T-ES provides a knob for the user to choose N based on her willingness to trade-off the maximum write tail latency for potential improvement in read tail latency. For example, if the user wants to achieve sub-100ms write tail latency and the maximum write delay occurring from a GC scheme (e.g., GC policy [6, 13–15, 17, 37], over-provisioning ratio [30]) is 35ms, the user should set N to be 64ms so that the total maximum write latency remains under 100ms. By default, we set N to 64ms for our experiments.

3 Evaluation

3.1 Methodology

Evaluation Framework. Although we utilized a real, prototype low-latency SSD [5] with modified firmware to perform some experiments (Figure 1 and Figure 3), it is not possible to utilize the real device to evaluate our presented schemes such as D-ES and T-ES since implementations of such policy require an extension to the interface between the SSD controller and NAND flash chips (e.g., new commands). For this reason, we use MQSim [33] for our experiments, which we extend so that it can accurately model low-latency NAND flash chips. In particular, we i) allow data cache manager and FTL to use strict 8 plane program so that it can achieve higher write throughput, ii) enable I/O scheduler to process the user read request as the highest priority, and iii) modify NAND flash controller and memory logic to model our practical erase suspension mechanisms. Table 1 summarizes the parameters used for our MQSim, while Table 2 compares the simulation results against those of a real low-latency SSD [5] for various I/O patterns as validation of the simulator. The average read/write latency and throughput from the simulator demonstrate only a 6% error on average (with 13% in the worst case) compared to measurements from the real device.

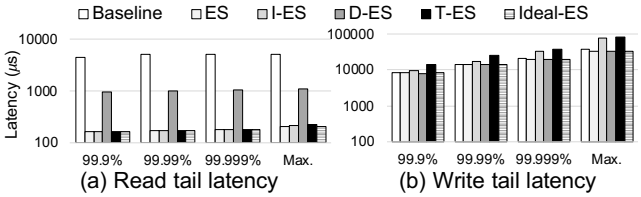

Figure 4: Read/write tail latency on 4KB random reads (70%) and writes (30%) workload.

Evaluated Configurations. Throughout this section, we present evaluation results for the following configurations across different benchmarks (i.e., random 4KB access, ACT [1], and TPC-C [34]). Also, we use for all experiments the steady state pre-condition [32] in which the space of an SSD including the over-provisioning (OP) [30] area is full; this pre-condition helps evaluate the latency behaviors that can happen under the worst-case condition of read and write requests. Section 2 discusses three practical erase suspension mechanisms: I-ES, D-ES, and T-ES. We add the following three configurations to our evaluation for comparison:

- Baseline: Erase operations do not get preempted by an incoming read request.
- Erase Suspension (*ES*): The scheme can suspend and resume an erase pulse from an arbitrary point as proposed by Wu et al. [35].
- Ideal Erase Suspension (*Ideal-ES*): This scheme can suspend and resume an erase operation from any arbitrary point with *zero* erase suspension penalty.

3.2 Random Access Benchmark

Workload. We first evaluate our erase suspension policies using a microbenchmark that generates a mixture of 4KB random reads (70%) and writes (30%) at queue depth 16. We utilize Flexible I/O Tester (FIO) [8] to generate such disk access patterns. This workload is widely used to evaluate an SSD's latency performance [10, 38]. Figure 4 shows the results of this experiment.

Read Tail Latency. In the baseline, when an erase happens,

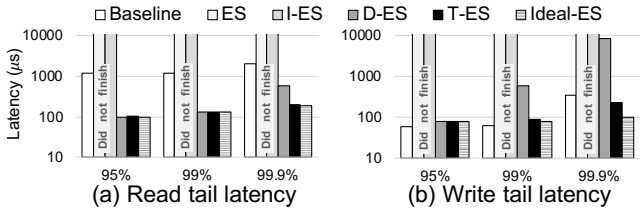

Figure 5: Read/Write tail latency on ACT workload (30× workload multiplier).

Table 3: Performance and stress test results using ACT.

	Baseline	ES	I-ES	D-ES	T-ES	Ideal-ES
Stress	32×	22×	22×	30×	30×	32×
Performance	14×	22×	22×	30×	30×	32×

the subsequent read requests are simply delayed for the duration of the remaining erase time (up to 5ms). As a result, baseline policy shows around 5ms read tail latency. With ES and I-ES such an erase operation is repeatedly aborted by the incoming read requests. As a result, incoming read requests do not experience an extra delay caused by the ongoing erase operation except for the 100μs latency required for erase suspension/resumption. For this reason, both ES and I-ES policies have very low read tail latency. Note that they are prone to experience erase (and write) starvation as pointed out in Section 2.2. However, it does not happen here because this workload has a period when read requests are not sent out for some time (i.e., all 16 outstanding requests in the queue are write requests waiting for an erase to proceed). During this period, erase operation successfully finishes. The read tail latency of D-ES is around 1ms because it always lets the ongoing erase step (not the whole erase) finish, making a read request wait. T-ES behaves similarly to I-ES in this workload as the timeout is rarely triggered.

Write Tail Latency. Baseline, ES, D-ES, and Ideal-ES have a lower write tail latency because their erase operation is finished in a relatively short period of time without canceling an existing erase pulse. Notably, the write tail latency of the baseline policy is the maximum GC latency (35ms) of our model without erase suspension. On the other hand, both I-ES and T-ES abort the erase multiple times, and thus delays write operations significantly. As a result, they tend to have a longer write tail latency.

3.3 Database Benchmark

Workload. ACT models the Aerospike database servers' real-time I/O access patterns. In essence, ACT consists of three threads: one issuing 2K small (1.5KB) read requests per second, another issuing 24 large (128KB) read requests per second, and the third one issuing 24 large (128KB) write requests per second. ACT gradually increases this rate in integer multiples (e.g., ACT 4× workload means 8K small read requests/s, 96 large read requests/s, 96 large write requests/s) and considers that the device has passed the *performance test* if it meets the following conditions for a long time (e.g., 24hrs): i) 95% of transactions finish in 1ms, ii) 99% of transactions finish in 8ms, iii) 99.9% of transactions finish in 64ms, and iv) average transaction time for each type of requests is smaller than ACT workload's I/O request period. If a device satisfies only the fourth condition but fails to satisfy one of the first three conditions, that device is said to pass the *stress test* but not the *performance test*. The maximum ACT multiplier that an SSD can satisfy is that particular SSD's performance rating.

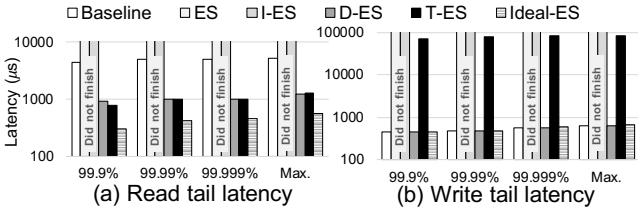

Figure 6: Read/Write tail latency on TPC-C.

ACT Results. Table 3 shows the maximum multiplier in which each configuration passed the performance and stress tests. Figure 5 shows the tail latency of 95%, 99%, and 99.9% accesses at 30× workload multiplier. As shown in Table 3, the baseline has a good average response time and thus can successfully run ACT with 32× multiplier. On the other hand, it has the worst read tail latency since the part of erase latency (up to 5ms) is exposed to read requests. This leads to a relatively poor performance test result for the baseline. ES and I-ES have good read tail latency behavior. However, continuous read requests cause erase (and write) starvation, which leads to a failure in the stress test (fourth condition) at the workload multiplier whose value is above 22×. On the other hand, both D-ES and T-ES demonstrate strong results for both stress and performance tests. Both D-ES and T-ES can maintain low read tail latency while avoiding erase (and write) starvation.

3.4 Transaction Processing Benchmark

Workload. TPC-C [34] is a popular benchmark for online transaction processing frameworks. We utilize a published disk trace of the system running TPC-C from SNIA [31], to evaluate our erase suspension mechanisms.

TPC-C Results. Figure 6 shows the results from this experiment. The baseline policy generally observes a noticeably higher read tail latency than the other schemes since latency is exposed to many read requests queued behind the erase operation. Neither ES or I-ES runs to completion as they suffer from a severe erase (and write) starvation and cause the simulator to exit prematurely after failing to serve many write requests for a long time. As in the ACT workload, this is because continuous read requests prevent the completion of an erase operation. Both T-ES and D-ES achieve much lower tail latency than the baseline. In particular, both achieve around 1ms max tail latency, which indicates that a read request only experiences about a single erase step delay at most.

The write tail latency of the baseline, D-ES, and Ideal-ES is similar to each other. In contrast, ES and I-ES suffer write starvation, and T-ES records the longest write tail latency. The

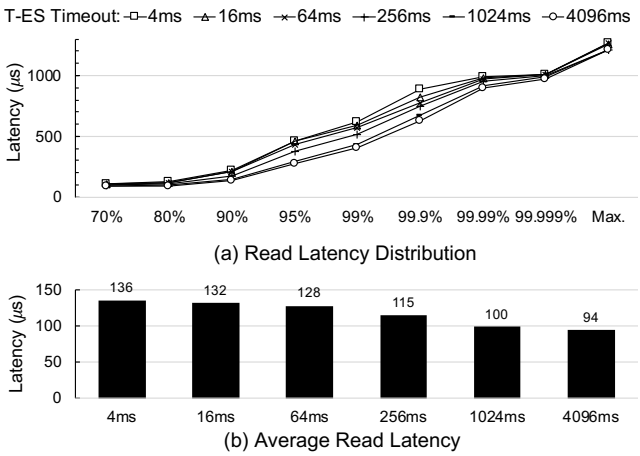

Figure 7: (a) Read latency distribution and (b) average read latency with varying T-ES timeout values for TPC-C.

Table 4: Maximum write latency for TPC-C.

T-ES Timeout	4ms	16ms	64ms	256ms	1.02s	4.09s
Write Latency (Maximum)	27ms	40ms	86ms	280ms	1.04s	4.1s

T-ES scheme delays erase (and following writes) for up to the timeout value hoping to find a period in which it can perform an erase without blocking reads. In this case, T-ES does not find such a period and hence ends up triggering the D-ES mechanism after delaying the write requests; as expected, the maximum write latency converges within 90ms (i.e., the sum of GC latency (24ms) and the T-ES timeout value (64ms)).

3.5 Sensitivity to T-ES Timeout Threshold (*N*)

If an erase operation is delayed for *N*ms, T-ES switches from I-ES to D-ES to avoid erase (and hence write) starvation. We perform a sensitivity study, using TPC-C with varying *N* to provide an insight into the tradeoff with selection of this parameter.

Figure 7(a) shows the read latency distribution with different values of *N* from 4ms to 4096ms. At around the 80th percentile we start to observe a gradual transition of read latency from more of I-ES (about 200μs) to D-ES (about 1ms). Increasing *N* generally i) lowers frequency of both high-latency read (i.e., over-200μs) and ii) average read latency, but iii) increases the maximum write latency. As *N* increases, T-ES has a greater chance of running in I-ES mode to lower the chance for a read request to experience a 1ms delay for finishing an ongoing erase pulse. Fewer long-latency reads lead to a lower average read latency as shown in Figure 7(b).

However, increasing *N* negatively affects the maximum write latency. Table 4 summarizes the maximum write latency with varying *N*. As discussed in Section 2.4, the maximum write latency of T-ES is the sum of GC latency and the T-ES timeout value. This is because GC operations, which need an erase operation to produce a free block for user data write, may be interfered while running in I-ES mode. Once T-ES timeout is triggered, it switches to D-ES to allow GC operations to produce the free blocks to write user's data. The maximum GC latency of the TPC-C workload (with steady state pre-condition) is 24ms. Thus, the measured maximum write latency is not far off from the estimated value (i.e., GC latency plus *N*).

4 Related Work

Garbage Collection Optimization. There are several prior works on alleviating the effect of GC on tail latency [6, 13–15, 17, 37]. While these optimizations can effectively reduce the effect of coarse-grained GCs on read tail latency, they do not address the effect of long erase operation on read tail latency, which becomes more important with optimized GC. Thus, these techniques are orthogonal or complementary to our presented erase suspension schemes.

I/O Scheduling Optimization. Another way to optimize tail latency is to schedule a read/write request in an intelligent way [16, 20, 21, 36, 39, 40]. These proposals are effective when there are multiple devices available. On the other hand, our work focuses on tail latency reduction without requiring multiple devices. Still, if multiple devices are available, our technique can be applied jointly with these optimizations.

5 Conclusion

This paper introduces practical erase suspension mechanisms to limit the impact of erase operation on long read tail latency. Leveraging the iterative erase mechanism commonly employed by today's flash devices, the proposed mechanisms minimize the impact of erase operation on read tail latency, while requiring only minor extensions to the flash interface. With the proposed erase suspension mechanisms, our proposal enables flash-based SSDs to achieve very low read tail latency, while avoiding erase (and write) starvation and endurance degradation of NAND. For example, our results demonstrate the feasibility of a sub-200μs 99.999th percentile read tail latency for 4KB random access workloads, which is competitive with an emerging non-flash NVM-based SSD [10]. This will harness the full potential of flash-based SSDs as the primary storage platform for future data-centers that will be required to run a variety of latency-sensitive online services.

Acknowledgments

We thank Sam H. Noh for shepherding this paper and the anonymous reviewers for their feedback. We also thank Adel Choi, Heesoo Kim, Donghyeon Kwon, and Daejoong Jung for their support. This work was supported by Research Resettlement Fund for the new faculty of Seoul National University, a research grant from Samsung Electronics, and Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea Government (MSIT) (No. 2014-0-00035, Research on High Performance and Scalable Manycore OS). Jae W. Lee is the corresponding author.

References

[1] Aerospike Certification Tool. https://github.com/aerospike/act.

[2] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. The datacenter as a computer: an introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3):Morgan & Claypool Publishers, 1–154, 2013.

[3] Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman S. Unsal, and Ken Mai. Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime. In *Proceedings of the IEEE 30th International Conference on Computer Design*, ICCD'12, pages 94–101. IEEE, 2012.

[4] Li-Pin Chang, Tei-Wei Kuo, and Shi-Wu Lo. Real-time garbage collection for flash-memory storage systems of real-time embedded systems. *ACM Trans. Embed. Comput. Syst.*, 3(4):ACM, 837–863, November 2004.

[5] Wooseong Cheong, Chanho Yoon, Seonghoon Woo, Kyuwook Han, Daehyun Kim, Chulseung Lee, Youra Choi, Shine Kim, Dongku Kang, Geunyeong Yu, Jaehong Kim, Jaechun Park, Ki-Whan Song, Ki-Tae Park, Sangyeun Cho, Hwaseok Oh, Daniel DG Lee, Jin-Hyeok Choi, and Jaeheon Jeong. A flash memory controller for 15μs ultra-low-latency SSD using high-speed 3D NAND flash with 3μs read time. In *Proceedings of the IEEE International Solid-State Circuits Conference*, ISSCC '18, pages 338–340. IEEE, 2018.

[6] Wonil Choi, Myoungsoo Jung, Mahmut Kandemir, and Chita Das. Parallelizing garbage collection with I/O to improve flash resource utilization. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '18, pages 243–254. ACM, 2018.

[7] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):ACM, 74–80, 2013.

[8] Flexible I/O Tester. https://github.com/axboe/fio.

[9] Alan Freedman. MT/sec. *The Computer Desktop Encyclopedia*. https://www.computerlanguage.com/results.php?definition=MT/sec.

[10] Frank T. Hady, Annie Foong, Bryan Veal, and Dan Williams. Platform storage performance with 3D XPoint technology. *Proceedings of the IEEE*, 105(9):IEEE, 1822–1833, 2017.

[11] Mingzhe Hao, Gokul Soundararajan, Deepak Kenchammana-Hosekote, Andrew A. Chien, and Haryadi S. Gunawi. The tail at store: A revelation from millions of hours of disk and SSD deployments. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies*, FAST '16, pages 263–276. USENIX Association, 2016.

[12] Jaeyong Jeong, Sangwook Shane Hahn, Sungjin Lee, and Jihong Kim. Lifetime improvement of NAND flash-based storage systems using dynamic program and erase scaling. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, FAST '14, pages 61–74. USENIX Association, 2014.

[13] Myoungsoo Jung, Wonil Choi, Shekhar Srikantaiah, Joonhyuk Yoo, and Mahmut T. Kandemir. HIOS: A host interface I/O scheduler for solid state disks. In *Proceedings of the 41st ACM/IEEE International Symposium on Computer Architecture*, ISCA '14, pages 289–300. ACM/IEEE, 2014.

[14] Wonkyung Kang, Dongkun Shin, and Sungjoo Yoo. Reinforcement learning-assisted garbage collection to mitigate long-tail latency in SSD. *ACM Transactions on Embedded Computing Systems*, 16(5s):ACM, 134:1–134:20, 2017.

[15] Wonkyung Kang and Sungjoo Yoo. Dynamic management of key states for reinforcement learning-assisted garbage collection to reduce long tail latency in SSD. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, pages 8:1–8:6. ACM, 2018.

[16] Bryan S. Kim, Hyun Suk Yang, and Sang Lyul Min. AutoSSD: an autonomic SSD architecture. In *Proceedings of the USENIX Annual Technical Conference*, ATC'18, pages 677–690. USENIX Association, 2018.

[17] Jaeho Kim, Donghee Lee, and Sam H. Noh. Towards SLO complying SSDs through OPS isolation. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, FAST'15, pages 183–189. USENIX Association, 2015.

[18] Dong Wook Lee, Sunghoon Cho, Byung Woo Kang, Sukkwang Park, Byoungjun Park, Myoung Kwan Cho, Kun-Ok Ahn, Ye Seok Yang, and Sung Wook Park. The operation algorithm for improving the reliability of TLC (triple level cell) NAND flash characteristics. In *Proceedings of the 3rd IEEE International Memory Workshop*, pages 1–2. IEEE, 2011.

[19] Jaeduk Lee, Jaehoon Jang, Junhee Lim, Yu Gyun Shin, Kyupil Lee, and Eunseung Jung. A new ruler on the storage market: 3D-nand flash for high-density memory and its technology evolutions and challenges on the future. In *Proceeding of the 2016 IEEE International Electron Devices Meeting (IEDM)*, pages 11.2.1–11.2.4, Dec 2016.

[20] Junghee Lee, Youngjae Kim, Galen M. Shipman, Sarp Oral, and Jongman Kim. Preemptible I/O scheduling of garbage collection for solid state drives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(2):IEEE, 247–260, 2013.

[21] Christopher R. Lumb, Arif Merchant, and Guillermo A. Alvarez. Façade: virtual storage devices with performance guarantees. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, FAST'03, pages 131–144. USENIX Association, 2003.

[22] Iyswarya Narayanan, Di Wang, Myeongjae Jeon, Bikash Sharma, Laura Caulfield, Anand Sivasubramaniam, Ben Cutler, Jie Liu, Badriddine Khessib, and Kushagra Vaid. SSD failures in datacenters: What? When? and Why? In *Proceedings of the 9th ACM International Systems and Storage Conference*, SYSTOR'16, pages 7:1–7:11. ACM, 2016.

[23] NVM Express 1.3a. http://nvmexpress.org/.

[24] Krishna Parat and Chuck Dennison. A floating gate based 3D NAND technology with CMOS under array. In *Proceeding of the 2015 IEEE International Electron Devices Meeting (IEDM)*, pages 3.3.1–3.3.4, Dec 2015.

[25] Byoungjun Park, Sunghoon Cho, Milim Park, Sukkwang Park, Yunbong Lee, Myoung Kwan Cho, Kun-Ok Ahn, Gihyun Bae, and Sungwook Park. Challenges and limitations of NAND flash memory devices based on floating gates. In *Proceeding of the 2012 IEEE International Symposium on Circuits and Systems*, pages 420–423, 2012.

[26] Jisung Park, Jaehoon Lee, Myungsuk Kim, Myungjun Chun, and Jihong Kim. Reducing read latency fluctuations of flash storage systems using preemptible programs and erases. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies, Work-in-Progress Reports (WiPs)*, FAST '18. USENIX Association, 2018.

[27] Sung-Kye Park. Technology scaling challenge and future prospects of DRAM and NAND flash memory. In *Proceeding of the 2015 IEEE International Memory Workshop (IMW)*, pages 1–4, May 2015.

[28] PCI Express 3.1. https://pcisig.com/specifications/.

[29] C. Sandhya, Apoorva B. Oak, Nihit Chattar, Udayan Ganguly, C. Olsen, S. M. Seutter, L. Date, R. Hung, Juzer Vasi, and Souvik Mahapatra. Study of P/E cycling endurance induced degradation in SANOS memories under NAND (FN/FN) operation. *IEEE Transactions on Electron Devices*, 57(7):1548–1558, July 2010.

[30] Kent Smith. Understainding SSD overprovisioning. In *Proceedings of the Flash Memory Summit (2012)*, Flash Memory Summit'12, 2012.

[31] SNIA IOTTA repository. TPC-C traces. http://iotta.snia.org/traces/131.

[32] SNIA Solid State Storage Performance Test Specification. https://www.snia.org/sites/default/files/HoEasen_SNIA_Solid_State_Storage_Per_Test_1_0.pdf.

[33] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. MQSim: A framework for enabling realistic studies of modern multi-queue SSD devices. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies*, FAST '18, pages 49–66. USENIX Association, 2018.

[34] TPC-C benchmark. http://www.tpc.org/tpcc/.

[35] Guanying Wu and Xubin He. Reducing SSD read latency via NAND flash program and erase suspension. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, pages 117–123. USENIX Association, 2012.

[36] Suzhen Wu, Weidong Zhu, Guixin Liu, Hong Jiang, and Bo Mao. GC-aware request steering with improved performance and reliability for SSD-based RAIDs. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, IPDPS'18, pages 296–305. IEEE, 2018.

[37] Shiqin Yan, Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminatahan Sundararaman, Andrew A. Chien, and Haryadi S. Gunawi. Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies*, FAST'17, pages 15–28. USENIX Association, 2017.

[38] Samsung Z-SSD SZ985. https://www.samsung.com/semiconductor/global.semi.static/Brochure_Samsung_S-ZZD_SZ985_1804.pdf.

[39] Jianyong Zhang, Alma Riska, Anand Sivasubramaniam, Qian Wang, and Erik Riedel. Storage performance virtualization via throughput and latency control. In *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, MASCOTS'05, pages 135–142. IEEE, 2005.

[40] Quan Zhang, Dan Feng, Fang Wang, and Yanwen Xie. An efficient, QoS-aware I/O scheduler for solid state drive. In *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications*, HPCC'13, pages 1408–1415. IEEE, 2013.