Fantastic SSD Internals and How to Learn and Use Them

Nanqinqin Li
University of Chicago and
Princeton University

Mingzhe Hao
University of Chicago

Huaicheng Li
University of Chicago and
Carnegie Mellon University

Xing Lin
NetApp

Tim Emami
NetApp

Haryadi S. Gunawi
University of Chicago

ABSTRACT

This work presents (a) Queenie, an application-level tool that can automatically learn 10 internal properties of block-level SSDs, (b) Kelpie, the learning and analysis results of running Queenie on 21 different SSD models from 7 major SSD vendors, and (c) Newt, a set of storage performance optimization examples that use the learned properties. By bringing numerous observations and unique findings, this work exposes substantial improvement spaces for both SSD users and vendors, enlightening possibilities of unleashing more SSD performance potential and highlighting the necessity of further exploring SSD internals.

CCS CONCEPTS

• **General and reference** → **Empirical studies**; *Measurement*; • **Information systems** → **Flash memory**.

KEYWORDS

Solid-State Drive, Performance Characterization

ACM Reference Format:

Nanqinqin Li, Mingzhe Hao, Huaicheng Li, Xing Lin, Tim Emami, and Haryadi S. Gunawi. 2022. Fantastic SSD Internals and How to Learn and Use Them. In *The 15th ACM International Systems and Storage Conference (SYSTOR '22), June 13–15, 2022, Haifa, Israel.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3534056.3534940

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SYSTOR '22, June 13–15, 2022, Haifa, Israel

© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-9380-5/22/06…$15.00
https://doi.org/10.1145/3534056.3534940

1 INTRODUCTION

Solid-State Drives (SSDs) are a cornerstone of modern storage systems because of their competitive performance, reliability, capacity, and cost [2, 13, 19, 33, 41, 46]. However, while fulfilling user's increasing demands on storage, modern SSDs also bring their own challenges: it is difficult to optimally utilize them as most of them show up as black-box devices, with internal complexities such as FTL mapping, write buffer management, and garbage collection mechanisms, hidden and intangible from their users [21, 23, 26, 29, 49, 50].

These complexities, unfortunately, can bring non-negligible side-effects, with performance inconsistency as one of the notorious ramifications. For example, write buffer flush can contend with reads on NAND resources and bring long latency tails [10, 19, 39]; reads with inappropriate alignment can take extra overhead to process as SSDs apply minimal unit of access [31, 32]; some SSDs are designed for certain purposes, and when used inappropriately, can dramatically downgrade the overall system performance [5, 30].

Motivated to resolve these negative impacts, multiple pieces of prior work [12, 28, 30–32] try to extract crucial SSD properties and propose coherent designs based on the observations. They have reasonably argued that probing SSDs can help build more effective solutions and bring significant performance improvement.

Based on these gains, we further ask: is there more knowledge hidden in modern SSDs, waiting to be learned and utilized, especially as modern SSDs have evolved rapidly over the past decade? We found that there are many questions unanswered in prior work. Do modern SSDs have favorable sizes on reads and punish those that do not comply (§4.1, §5.1)? Do components that were prevalent in SSDs previously, such as read buffer, still exist in recent SSD models (§4.5)? Do large-capacity SSDs, which are very common nowadays, have write buffers of appropriate sizes (§4.2) and the capability to handle highly-parallel writes (§4.4)? Do SSDs apply hybrid (externally and internally triggered) buffer flush policies (§4.3) that can be exploited for less contention and better performance (§5.2)? Do SSDs really perform better when they face less "stress" (§4.6)?

To answer those questions, we introduce Queenie, Kelpie, and Newt.[1] **(a) Queenie** is a holistic, application-level tool that *probes* the SSDs with carefully tuned read/write mixed workloads. By just measuring latencies observable at the application level, Queenie learns and extracts 10 SSD internal properties and can run on any block-device SSDs. Queenie is open-sourced [3] as we are not aware of a similar tool publicly available. **(b) Kelpie** represents our analysis results of running Queenie on 21 different SSD models from 7 major SSD vendors, from regular consumer-level ones to latest enterprise-level ones. Kelpie brings numerous observations and 6 unique findings, exposing substantial improvement space for both SSD users and manufacturers. Part of Kelpie's raw data set is made public [3]. Finally, **(c) Newt** is a set of I/O optimization examples that showcase how applications can leverage this knowledge in real-world scenarios such as aligning read sizes and exploiting write buffer knowledge.

2 GOALS

2.1 Properties and Advantages

To design Queenie, we must first decide the important SSD internal properties to learn and extract. Below, we provide the list, the definition of each of the properties, and the advantages of knowing them, as summarized in Table 1.

P_1 **Page size** is the minimal unit of read and write. Knowing the page size is the foundation of subsequent probing techniques. *Advantage*: knowing the internal page size will help applications align I/Os properly to avoid unnecessary alignment overhead such as read-modify-write [31, 32].

P_2 **Page type** represents the NAND cell type (SLC, MLC, TLC, etc.) and how the page offsets are mapped to low/medium/high bits of the MLC/TLC cells. A page offset that is mapped to higher bits tend to have a higher latency. *Advantage*: mapping hot data to low pages (lower latency) can bring performance improvement [16] under a typical static logical-to-physical page offset mapping at the NAND block level [50].

P_3 **Chunk size** reveals the striping unit inside the device (similar to RAID chunk size). For example, if an SSD has 16 chips, it might spread incoming 16 pages evenly among the 16 chips (1-page chunk), or it could also split them into 2 groups to 2 chips with 8 pages each (8-page chunk). *Advantage*: this information can help applications understand the throughput of sequential reads/writes.

P_4 **Stripe width** is the number of chunks that can be parallelized internally without contention at the chip level (akin to RAID stripe width). *Advantage*: knowing this parallelism level allows applications to exploit the internal bandwidth better, *e.g.*, how databases are redesigned to map better to the SSD internal parallelism [12].

[1]In the "Fantastic Beasts and Where to Find Them" movie, **Queenie** is a character who can read other people's minds, **Kelpie** is a shape-shifting creature that can take any form, representing the many forms of our findings which depend on the probed SSD models, and **Newt** is a wizard who help solve crimes such as the crimes of Grindelwald in the second sequel.

ID	Name	Output Format
P_1	Page size	Size (*e.g.*, 4 KB)
P_2	Page type	S/M/TLC + low/high
P_3	Chunk size	Size (*e.g.*, 64 KB)
P_4	Stripe width	A number (*e.g.*, 64)
P_5	Channel/chip layout	#Channels * #Chips
P_6	Read perf. consistency	'Good'(✓) or 'Bad'(✗)
P_7	Read buffer cap	A size or none
P_8	Write buffer cap	A size or none
P_9	Write parallelism	A number (*e.g.*, 4)
P_{10}	Internal flush window	A duration (*e.g.*, 5ms)

Table 1: Properties covered by Queenie. *The description of the 10 SSD properties and their IDs (P_1-P_{10}) covered by our work*

P_5 **Channel/chip layout** represents the number of channels and chips per channel. *Advantage*: though sometimes treated as a "boring" fact, this property can unearth unusual findings (§6), *e.g.*, a channel can exhibit more contention (higher latency) with some channels than with the others.

P_6 **Read performance consistency** is about whether the SSDs have favorable read sizes and "punish" those who do not follow, *e.g.*, whether SSDs can process non-paged, sectored reads with minimal alignment overhead and whether large reads are broken into smaller ones and served simultaneously. *Advantage*: it is worth to double check these standard expectations, as sometimes we find exceptional cases (§4.1) that might require special handling (§5.1).

P_7 **Read buffer capacity** is about determining how much of the internal RAM or SLC is occupied for caching reads (for cost-efficiency, some newer QLC drives use SLC as the buffer instead of RAM). As NAND read speed grows faster, we check whether vendors still employ read cache inside the device. *Advantage*: caching is paramount to performance. Knowing this information can hint of a better cache design at a higher layer.

P_8 **Write buffer capacity** is similarly about speculating how much the internal RAM or SLC is used for buffering user writes before flushing them to the NAND cells. *Advantage*: knowing the buffer size, together with controlling/tracking user write operations, can help predict the timing of flushes and potential garbage collection activities [30].

P_9 **Write parallelism** is the number of parallel writes supported by the device. The issue here is that unlike read operations, writes tend to be absorbed into a centralized buffer first. This probing checks whether SSDs are able to maintain high write parallelism regardless of the centralized buffering. *Advantage*: the result here can bring insights on how to apply SSD-specific optimizations for write workloads, while revealing defects in write handling in some SSDs (§4.4).

	#	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
[31]	4		✓					✓	✓		
[12]	2	✓		✓	✓						
[30]	7								✓		
Q&K	21	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 2: Queenie vs. existing work. *This table compares Queenie and Kelpie (Q&K) with other related SSD probing work. '#' stands for number of SSD models probed by each work. As we can see, Queenie covers more SSDs and properties.*

P_{10} **Internal flush window** is the amount of time the device needs to softly dump the entire write buffer to the underlying NAND without causing severe contention (*e.g.*, background flushes under low load). *Advantage*: as this internal flush usually incurs less contention and shorter blocking time, it can be further exploited to design a solution that mitigates the buffer flush interference (§5.2).

Garbage collection (GC) is another crucial factor affecting SSD performance. If the timing of GC is known, it can be avoided by re-routing reads to other less-busy replicas [18, 20]. We are not aware of prior work that successfully and affirmatively unveiled this information. Some model GC as a statistical distribution of #writes between adjacent GC occurrences [30]. We attempted probing GC by issuing writes to the same NAND chip, while performing background read on all chips to examine when GC happened and which chips are activated to accommodate the GC. However, ensuring that writes are sent to a particular chip can be extremely tricky, and frequent GC wore out our drives too quickly.

2.2 SSD Models

Our goal is to probe a wide variety of SSDs from different major vendors such that we can contrast the probing results from different devices. From our labs and our industry partners, we have amassed 21 SSD models from 7 vendors, from consumer-level ones (8 pieces) to enterprise-level ones (13). The release year of these models ranges from 2008 to 2018. Their interfaces vary: SATA (7), SAS (6) and NVMe (8). Their sizes range from 64 GB to 2 TB, covering different NAND technologies such as SLC (6), MLC (12), and TLC (3). The full list of our devices can be found later in Table 4 in Section 6. As mentioned, we are not aware of any other probing papers with this scale of hardware being probed.

2.3 Related Work

We briefly discuss related work on SSD and storage probing, SSD performance modeling, and host-managed SSDs.

SSD probing. There are existing papers that attempt to probe SSD internals [12, 30, 31], but as shown in Table 2, our work provides more contributions in terms of SSD probing. First, Queenie and Kelpie cover more properties (P_1 $-P_{10}$), catching more "fantastic" facts of modern SSD models. Second, Queenie has been tested on 21 SSD models including enterprise ones, while other papers only probe up to 7 consumer models. Table 2 does not cite [28, 32] as they are shorter workshop versions. We also do not directly compare against [48] since their main focus is Intel Optane SSDs, though they also performed some probing on flash-based SSDs such as stripe width (P_4) to contrast with Optane SSDs.

Storage probing. Pulling up one level higher, storage probing in general has been a common area for decades such as probing HDDs [43, 47], memory hierarchy [51], RAID [14], SMR [4], and USB drives [8]. These areas of work produce positive outcomes of better understanding the hardware internals. However, their mechanisms cannot be easily ported to SSDs given the fundamentally different physical nature.

SSD performance modeling. A different, complementary way of direct probing is black-box performance modeling [9, 15, 24, 37, 38, 44]. Prior work in this area demonstrates that modeling SSDs is feasible by just collecting external performance metrics, but some do warn that the performance models could be error-prone as they "may not necessarily apply to other SSD models" [53].

Host-managed SSDs. We acknowledge the rise of software-defined SSDs to increase controllability either via extended interfaces [27, 34, 36] or full exposure of SSD internals [6, 7, 17, 40, 52]. Probing is only useful for black-box commodity SSDs, which is our focus in this paper given the larger scale of deployment of such SSDs.

3 QUEENIE (THE "MIND READER")

3.1 Probing Methods

Queenie probes the 10 properties (P_1 $-P_{10}$) as follows. We release the source code and pseudo code of Queenie [3].

Probing precondition. Properties P_1–P_7 require the SSD drives to be fully erased (with secure erase) and then populated with a full sequential write (we call this a "refill" operation); For P_8–P_{10}, drives need to be properly erased such that all writes can be sent to empty pages (*i.e.*, no overwriting).

P_1 **Page size.** Queenie extracts the smallest unit of read by reading continuous 0.5KB sectors to detect the interval between page boundaries. For example, assume an SSD has a page size of 4KB and page boundaries at 0, 4 and 8KB. Then a read of 2 sectors at offset 3KB would require the SSD to read only one page, while the same read at 3.5KB will read 2 pages with *higher* latency (if the 2 pages are parallelized inside the SSD, the latency would still be slightly higher due to channel-level contention), indicating a page boundary at 4KB. By repeating this read at larger offsets, Queenie will see the adjacent boundary at 8KB, confirming the 4KB page size. The probing function is `F1_pushRead` (Alg. 1 in [3]).

P_2 **Page type.** Queenie sends page reads *one page at a time* and compares the read latency of each page. To eliminate the side effects of internal readahead (if any), Queenie reads from higher to lower page offsets (readahead usually begins caching when seeing monotonically increasing back-to-back offsets). For M/TLC drives, we will observe *different* latencies as the offsets *vary*, as pages are mapped to different lower/medium/higher bits of the cell. With this, Queenie can retrieve the LPN (logical page number) positions of low/medium/high pages. This page-wise layout can be further divided by the stripe width (P_4) to get the layout inside a flash chip, assuming pages are evenly distributed to all chips. The function is `F2_rangeRead` (Alg. 2 in [3]).

P_3 **Chunk size.** Queenie reuses `F1_pushRead` to detect chunk boundaries by reading consecutive pages. With an SSD of a 16-page chunk size and chunk boundaries at 0, 16, and 32 pages, a 2-page read at page 14 will go to only one chip, while reading at page 15 would be served by two chips in parallel with *lower* latency, indicating a boundary at page 16. Similarly, at larger page offsets, Queenie will see another boundary at page 32 and infer a chunk size of 16 pages.

P_4 **Stripe width.** Queenie issues concurrent chunk reads with incremental offsets. For instance, for an SSD with a stripe width of 16 chunks, issuing 4 reads with an offset increment of exactly the stripe width would cause these 4 reads to be sent to the same chip, resulting in heavier contention than those with smaller offset increments. The function is `F3_strideRead` (Alg. 3 in [3]).

P_5 **Channel/chip layout.** `F3_strideRead` also helps reveal channel-level contention when reads are sent to the same channel but to different chips, with a specific offset increment smaller than the stripe width. Such an offset increment hints the number of channels.

P_6 **Read performance consistency.** Queenie issues random reads of increasing sizes with sector-level increments, checks whether larger reads experience higher average latencies than smaller ones, and then identifies "problematic" size ranges. The function is `F4_incRead` (Alg. 4 in [3]).

P_7 **Read buffer capacity.** Queenie issues a large read first and then re-reads the very first page of the previous large read. If the device has a read buffer large enough to contain the large read, then the re-read latency should be much lower than that of a regular page read because it is buffered. The function is `F5_reRead` (Alg. 5 in [3]).

P_8 **Write buffer capacity.** Queenie issues concurrent non-overlapping writes. When the write buffer is flushed (near full), it will cause a write-latency spike. The amount of data written between the latency spikes hints the capacity of the write buffer. The function is `F6_conSeqW` (Alg. 6 in [3]).

P_9 **Write parallelism.** While running `F6_conSeqW`, the distribution of write latencies reflects the number of writes that can be supported at once. As a simple example, if 4 of 8 concurrent writes (issued at the exact same time, μs-level) observe almost a 2x latency compared to the other 4 exactly concurrent writes, then we can conclude the write buffer can absorb 4 concurrent writes at a time (*not* per second).

Results	4 KB	1 KB	8 KB
Confidence	0.99	0.70	0.39

Table 3: Automated analysis example output for page size (P_1). *This table shows an example where there are 3 possible answers for P_1, with 4 KB as the final answer (0.99 confidence).*

P_{10} **Internal flush window.** After identifying the write buffer capacity, function `F7_seqwSleep` (also Alg. 6 in [3]) slowly populates the entire write buffer by injecting `sleep` (2ms to 5s in a "binary search" manner) to determine the minimal sleep length that eliminates the flush spikes completely. This minimal length represents the internal flush window.

3.2 Probing Time and Automated Analysis

To ensure that our conjectures are highly consistent, for every measurement, we repeat the I/Os for 5,000–10,000 times and use the average, requiring *1-8 hours* for each probing experiment to finish (for all 21 SSD drives in our collection). For read-only properties P_1–P_7, the drive only needs to be "refilled" once at the beginning of the probing to minimize erasure times and prevent the drive from wearing out [21, 25].

We also developed a tool that automatically analyzes the outputs of these probing functions and generates a final result for the properties. The key to the automated analysis is identifying *latency abnormalities.* For example, in probing write buffer capacity (P_8), *latency spikes* are considered an indication of flush occurrences since their latencies are much higher than the others; in probing stripe width (P_4), latency will be significantly higher if those concurrent reads are sent to the same flash chip.

The tool uses the *Jenks natural breaks algorithm* [45], a classification method optimized for one-dimensional data (the latencies) to distinguish abnormal from normal latencies. We configure Jenks to output two classes and take the one with fewer data points as the abnormalities. Then, the most common interval between neighboring abnormalities is calculated to represent the probed property. For instance, the interval between two neighboring spikes in P_8 (see Figure 2 later) is the amount of data needed to trigger a buffer flush, indicating the write buffer capacity.

For properties where different levels of abnormalities might exist, specifying more than two classes might give more accurate results (*e.g.*, in P_4 stripe width, latency spikes could be lower when only half of the concurrent reads go to the same chip). For such properties, we run Jenks several times from two to five classes and use Silhouette score (range from 0 to 1) to select the best classification result for generating the final probing results (Table 3). In other words, the Silhouette score is considered the confidence score for the analysis results generated from a certain number of classes.

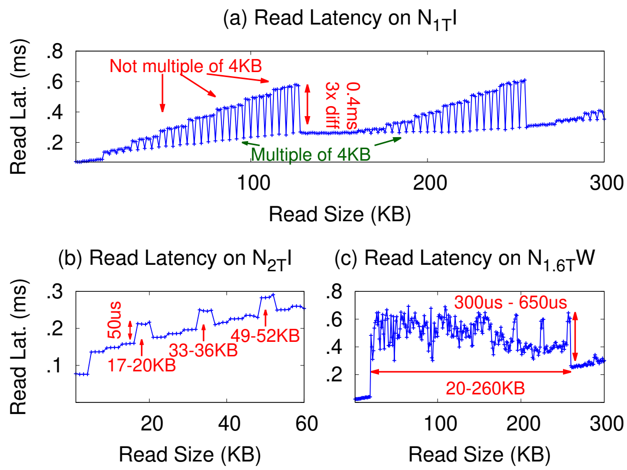

Figure 1: Read size vs. performance (§4.1). *Read latencies (y-axis) observed when the read size is set to different values (x-axis). In 3 models, read I/Os with different sizes (non-page-aligned sizes) can result in up to 400µs or 3x higher latencies. Each dot is the average latency of 5000 random reads.*

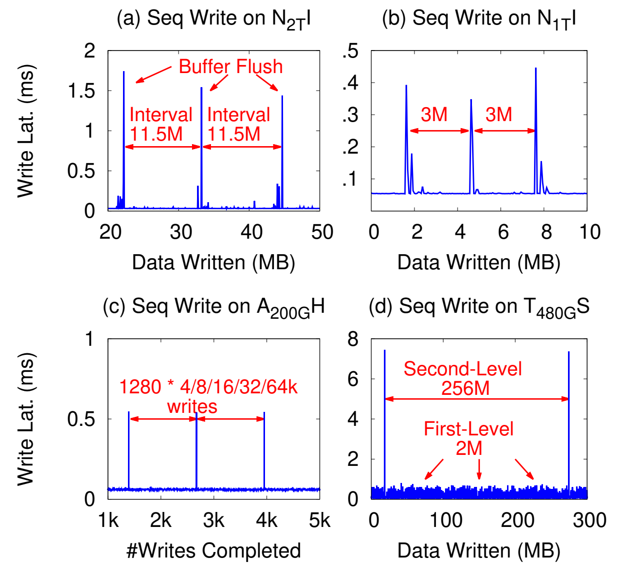

Figure 2: Small write buffers (§4.2). *Write latency (in y-axis) after a certain size of data has been written (x-axis) to the SSD. Buffered writes only take tens of µs, but the ms-level spikes suggest buffer flushing to the NAND cells (programming time). The distance between the two spikes hint on the write buffer size.*

4 KELPIE (MANY "FORMS OF FINDINGS")

While later Section 6 shows all the findings, here we highlight 6 main ones that we consider both "interesting" and "unique." Each subsection starts with a summary of the finding.

Labeling: These SSD models (symbols) are composed of three parts: **protocol** (NVMe, SATA, SAS), **size** (*e.g.*, $_{100G}$, $_{1T}$), and **vendor code** (a letter between *A..Z*). For the last item, a character represents a vendor, but the letter-to-vendor mapping is not revealed for hiding the actual vendor names. For example, "$N_{1T}I$" is a 1 TB NVMe drive from vendor *I*, "$T_{480G}S$" is a 480 GB SATA drive from vendor *S*, and "$A_{800G}P$" is a 800 GB SAS drive from vendor *P*.

4.1 Read Sizes vs. Performance

Observation: *Enterprise-level drives, $N_{1T}I$, $N_{2T}I$, and $N_{1.6T}W$, show higher latencies (50-400µs or up to 3x higher than the average) when the read size (a) is not a multiple of the page size or (b) lies within certain size ranges.*

SSDs have a minimal unit of read/write ("page"). Making an I/O size multiple of the page size will avoid paying the alignment overhead [31]. Modern SSDs are mostly well-optimized to minimize this overhead to single-digit µs. However, this is not always the case even for the latest drives. One *recent* enterprise-level SSD, $N_{1T}I$, exhibits worse latencies when the read size does not fit its expectation.

Figure 1a shows that $N_{1T}I$ returns up to 400µs or 3x higher latency (y-axis) when the read sizes (x-axis) are not multiple of 4KB. To emphasize, the read offsets are page-aligned but not the sizes, and the offsets are random. With this, $N_{1T}I$ would have a downgraded performance under real industrial SSD traces, including a database workload where 65% of the reads are not multiples of 4KBs (§5.1).

Another interesting anomaly is latency spikes within certain size ranges. Figure 1b shows that drive $N_{2T}I$ responds with 50µs higher latency (20-30% higher) when the read sizes fall within certain size ranges (17-20KB, 33-36KB, and 49-52KB). With this knowledge, by just increasing the read size slightly to fall outside these ranges (*e.g.*, change a 20KB read to a 24KB read), one can gain a substantial improvement (§5.1). Figure 1c shows a similar anomaly in drive $N_{1.6T}W$ with a 2-3x latency (350-600µs) when read sizes fall within the 20-260KB range but nowhere else up to 1 MB (the maximum read size in our experiment).

4.2 Small Write Buffer

Observation: *13 SSDs (mostly enterprise-level) have a relatively small write buffer (≤ 64MB), while some older SSDs can employ a large buffer as high as 800MB.*

Surprisingly, most SSDs only employ a write buffer of tens of MBs. Figure 2a hints that $N_{2T}I$, a 2TB enterprise-level drive, only uses 11.5MB for the write buffer; every 11.5MB of write traffic results in a latency spike of up to 1.5ms, which reflects the NAND programming time. Assuming the SSD is handling a reasonable write workload of 100 MB/s, the user would see roughly ten occurrences of a latency spike. Similarly, Figure 2b shows that another 1TB drive, $N_{1T}I$, can employ a "partial" write buffer of only 3MB (see §6).

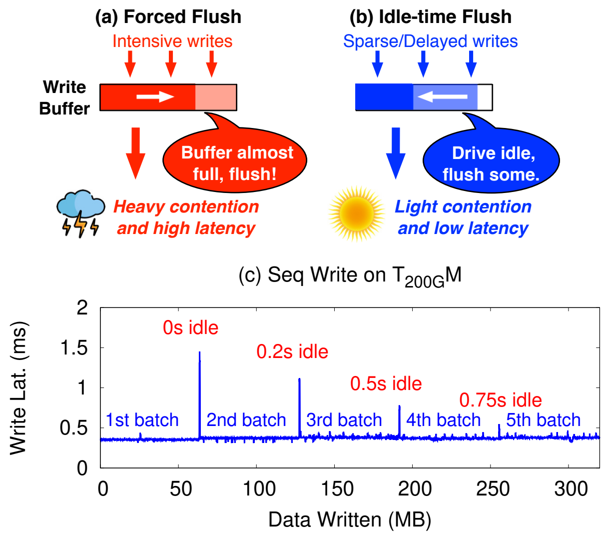

Figure 3: Idle-time buffer flush (§4.3). *(a-b) Buffer flush triggered due to heavy incoming writes or idle time. (c) The longer the sleep time in between the write batches, the lower the latency spike.*

Another interesting observation is that write buffers on some drives are not capped by MBs but rather the number of write I/Os, *e.g.*, in Figure 2c, $A_{200G}H$ issues a flush for every 1280 writes regardless of whether the size is 4/8/16/32/64KB. We also observed that 5 models employ a two-level buffer (*i.e.*, RAM as the first level and SLC as the second [11, 22]). In $T_{480G}S$ drive in Figure 2d, we see a write spike of almost 1ms every 2MB of write and another 7ms spike every 256MB of write, while the latency of buffered writes is only 90µs.

4.3 Idle-Time Buffer Flush

Observation: For 14 out of the 21 SSDs in our collection, internal buffer flush is triggered during idle time, which can be exploited by making the host send sparse/delayed writes.

Internal buffer flush can be triggered during highly intensive writes (Figure 3a), which will cause a write backlog, or during idle time (Figure 3b), which can be exploited to reduce write delays by having the higher storage layers send sparse/delayed writes.

Figure 3c shows this opportunity. Here, we use $T_{200G}M$ (known to have a 64 MB write buffer) and send batches of writes, where a batch is 64MB. In between the write batches, we insert an increasing user-level sleep time that ranges between 0 to 0.75 seconds. Without an idle window (x=64MB), the figure shows a high latency spike around the boundary of the 1st and 2nd write batches, as expected. However, as we increase the user sleep time between the subsequent batches, we see a reduced backlog. For example, as we insert 0.75s idle time between the 4th and 5th batch (x=256MB), the internal write buffer is likely being flushed, and the user writes can be absorbed by the buffer, resulting in low latencies.

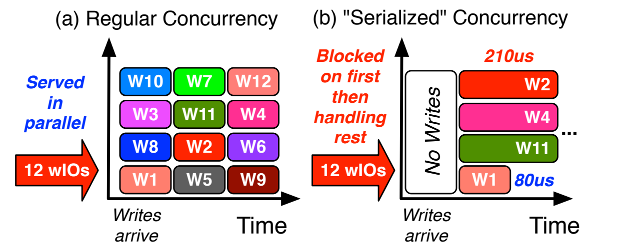

Figure 4: "Serialized" concurrent writes (§4.4). *(a) An expected behavior of write parallelism with 4 concurrent completions at a time, and (b) an anomalous behavior of write parallelism where the concurrent I/Os of the same batch are serialized when the write batch starts with an empty device queue.*

Outcomes from the other 13 drives also show this prevalent design choice. For example, $N_{1.6T}S$ can empty its buffer (40.25MB) in a idle window of 50ms and $T_{480G}S$ is capable of doing the same for its second-level buffer (256MB) in 5s (more details in §6).

4.4 "Serialized" Concurrent Writes

Observation: 4 out of 21 drives, $A_{960G}P_T$, $A_{960G}P_S$, $A_{1.6T}P$, and $N_{1.6T}W$, exhibit an anomalous concurrent write behavior where the concurrent I/Os in a given batch are serialized when the device queue is empty of pending writes.

Regarding property P_9 (write parallelism), one of Queenie's functions sends continuous concurrent 32KB writes, say `w1...wz` where `1` and `z` represent the first and last write of this long-running experiment, respectively. For device $A_{960G}P_S$, Figure 4a shows that this device returns 4 write completions at a time, *i.e.*, a write parallelism (P_9) of 4.

However, we notice a consistent anomalous behavior in the beginning of the experiment, which prompts us to configure Queenie to run batches of concurrent writes and *pause* in between until the device queue has no outstanding writes. Figure 4b shows this anomalous behavior of "serialized" concurrent writes. Whenever concurrent writes are sent when the device queue is empty, we notice that the first write (`w1`) completes first after 80µs and the other concurrent writes (`w2-w4`) of the same batch complete after 210µs. In other words, if users send periodic concurrent writes, these writes will be serialized as illustrated in Figure 4b.

4.5 The Disappearing Read Cache

Observation: Only 1 SSD model, $A_{800G}P$, employs an internal read cache. Disappearing read cache is likely due to the increasing NAND speed and DRAM buffer in higher layers.

An internal read cache, which was an essential part of SSD years ago [31], is hardly seen in modern SSDs. As recent NVMe drives deliver low read latency (30-150µs), using internal RAM for data caching might be deemed unnecessary.

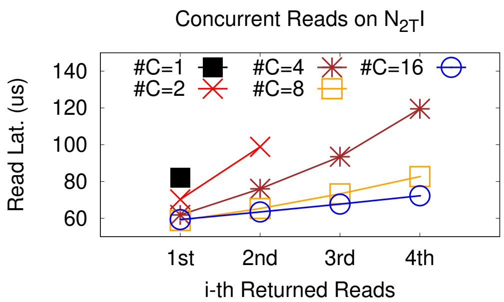

Figure 5: Less load, higher latency (§4.6). *In drive $N_{2T}I$, less concurrent reads lead to higher latencies than higher concurrent reads (e.g., compare the red × line for C=2 concurrent reads vs. the blue ○ line for C=16). The y-axis presents the average latency of the i^{th} returned I/O in every batch of concurrent reads.*

Nevertheless, drives with high NAND latency can still benefit from caching. $A_{800G}P$, for example, employs a 16MB read cache and reduces re-read latency from 320 to 60µs, an 80% reduction. This encouraging speed-up may motivate more SATA/SAS flash storage to adopt read caching as these older models still exhibit high read latency from 150 to 400µs.

4.6 Less Load but Higher Latency

$\boxed{\textit{Observation:}}$ *In $N_{2T}I$, fewer concurrent reads surprisingly lead to higher latency: some drives may prioritize throughput.*

Usually, less concurrency/load results in lower latencies. However, we observe the opposite behavior in drive $N_{2T}I$. In Figure 5, we perform five additional experiments, each with a different level of concurrency C from 1 to 16. Each experiment sends C concurrent random page reads (as a batch), waits until the completion of the batch, and then sends another batch of C concurrent reads. This loop repeats for thousands of times. The y-axis shows the average latency of the i^{th} returned I/O within every batch.

Let us start with the single ■ point, which indicates that there is only one I/O (1^{st}) in the batches of "1-concurrent" I/O, and the average latency is 82µs. However, in the 16-concurrent I/Os experiment (the blue ○ line), we see that the average latency of the 1^{st} I/O (x=1) and the 4^{th} one (x=4) is only 59 and 72µs, respectively, significantly *lower* than the ■'s 82µs value. Put simply, less-loaded experiments (less concurrency) result in higher latencies than the more-loaded ones. The root cause of this phenomenon remains a mystery. Perhaps this drive is optimized for throughput.

5 NEWT (A "CRIME SOLVER")

This section illustrates the benefits of Queenie and Kelpie for storage designs and policies, that is, how storage architects or users can leverage the extracted information to improve storage performance. Please note that the case studies in this section are just initial proofs of concept. We believe that subsequent studies can use the probed information extensively. Below, we use real industrial block-level SSD traces that represent a large company's database (`DB`), search engine (`SearchEng`), and cloud storage (`CloudStore`) workloads.

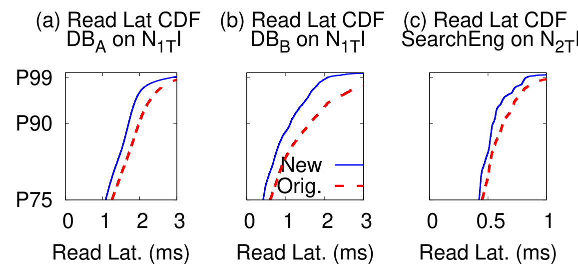

Figure 6: Adjusting read sizes (§5.1). *The CDF graphs show a potential latency improvement when the I/O stack knows the SSD's read-size oddities, e.g., by mitigating the sub-optimal latency (the red dashed "Orig." CDF lines) by simply altering the read sizes, resulting in lower latencies (x-axis) in high percentiles (y-axis), as shown by the "New" blue lines. DB_A, DB_B, and SearchEng represent the industrial SSD traces we use.*

5.1 Read Size Alignment

Section §4.1 shows that some data-center SSDs interestingly exhibit higher latencies when the read sizes fall into certain ranges. In Figure 1, $N_{1T}I$ delivers higher latencies when read sizes are not multiples of 4KB, $N_{2T}I$ has inexplicable 4KB-size ranges (*e.g.*, 17-20, 33-36, and 49-52KB sizes) that will result in higher latencies, and $N_{1.6T}W$'s speed drops when read sizes are in between 20-260KB.

Conceivably, such behaviors are ill-suited for real user workloads. For example, in the industrial traces we use, specifically the database traces, around 65% of the read sizes are *not* multiples of 4KB but rather multiples of 512 bytes. The latency-increasing size range on $N_{1.6T}W$ (20-260KB) will also be an issue since even page-aligned I/O sizes can easily fall within this problematic range (*e.g.*, 24KB).

A straightforward rearrangement that higher storage layers can employ to mitigate this issue is to avoid those problematic size ranges by adjusting the read sizes. For example, for $N_{1T}I$, the OS can increase the read size to the next page-size boundary (*e.g.*, change a 7KB read to 8KB). Figures 6a-b show that this very simple approach can speed up two database workloads (DB_A and DB_B), specifically, 14-31% read latency improvement at the 90^{th} percentile, and 18-32% and 13-48% at the $95^{th}$and 99^{th}percentiles, respectively.

For the read-size oddities in $N_{2T}I$, the OS can increase the read size outside its problematic size ranges (*e.g.*, change a 20KB read to 24KB). We use a search engine trace with mostly page-aligned I/Os but now make sure the peculiar ranges are avoided. Figure 6c shows that the latency is improved by

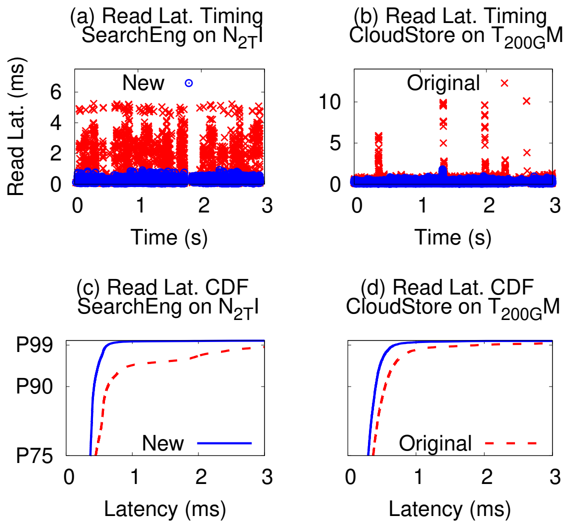

Figure 7: Exploiting write buffer knowledge (§5.2). *The top figures show the read latencies (y-axis) across time (x-axis). With our rate-limiting shim layer the high latency spikes (red dots) now disappear (blue dots). The bottom figures show the corresponding read latency CDF of the same experiments (New-vs-Original lines).*

14–20% between the 90–99thpercentiles. Other than these, open questions remain on what to do with $N_{1.6T}W$ where the under-performing size range is up to 260KB. The root cause could be factors like buggy firmware that seem more appropriate for the manufacturer to fix.

The main side-effect of read alignment is that it incurs more load on the SSD. However, as shown in Figures 1 and 6, reading more data in aligned requests is better than reading less unaligned data, especially for drives like $N_{1T}I$ that has significant overheads with unaligned reads. Other side-effects include cache pollution, read disturbance, etc.

5.2 Exploiting Write Buffer Knowledge

When the internal write buffer is full, an expensive flush is triggered, causing write latency spikes (§4.3). What we did not show is that such an expensive flush also causes a ripple effect to *read* latencies for two reasons. First, large flushes send more writes to the NAND cells and make incoming NAND reads wait behind the longer writes due to the length of the cell programming time. Second, flushing large amounts of data will likely trigger concurrent GCs across many chips, generating more read-write and erase contention compared to periodically flushing smaller amounts of data.

This begs the question: is there a way to mitigate the negative impact of full-buffer flush? This is where the knowledge about the internal write buffering becomes valuable (P_8 write buffer size and P_{10} flush window). This gray-box information can be effectively used by the higher storage layers to

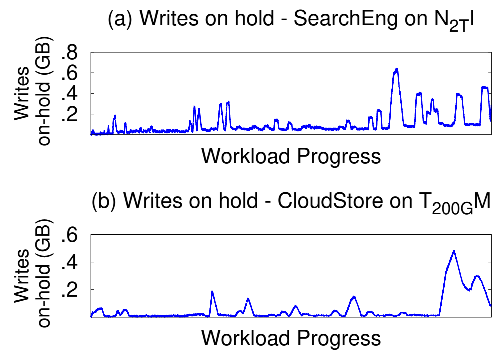

Figure 8: On-hold data in the rate-limiting layer (§5.2). *A reasonable amount of writes (y-axis) on-hold across time (x-axis).*

rate-limit the incoming writes accordingly to gain performance but without delaying writes significantly, specifically, by avoiding a full-buffer flush and allowing the underlying SSDs to flush gradually at the rate of its internal flush speed.

To demonstrate the potential improvement, we design a 4-step algorithm that can be deployed as a block-level rate-limiting shim layer. **(1)** We use Queenie to identify the write buffer capacity and the internal flush window and then divide these two values to get the average "flush speed," *e.g.*, in $N_{2T}I$, we identified a 11.5MB buffer with 200ms flush window, implying a flush speed of 57.5 MB/s (11.5MB/200ms). **(2)** The shim layer monitors the incoming write intensity in a recent time period of configurable length, *e.g.*, 5 MB write data in a monitored period of 100ms implies a 50 MB/s intensity. **(3)** We then introduce the "flush urgency" by dividing this incoming write intensity by the flush speed, indicating how intensive are the incoming writes with respect to the internal digest speed. In this example, the flush urgency is 0.87 (50/57.5). **(4)** If the shim layer observes an urgency less than 0.5, then it will allow all the incoming writes within the current 100ms period to enter the SSD. Otherwise, the incoming writes must be slightly delayed by T ms (*e.g.*, 0.1-1ms). The value of T is calculated by a rate-limiting function that incorporates the proportion of the recent read/write intensity and the flush urgency.

To evaluate this mechanism, we run the `SearchEng` and `CloudStore` traces on $N_{2T}I$ and $T_{200G}M$, respectively. Figures 7a-b show our shim layer results in a dramatic shift from the latency spikes ("Original") to a much stable latency ("New") in both experiments. Figures 7c-d show that our rate-limiter is able to reduce read latencies by 25-33%, 30-58%, and 64-82% at the 90th, 95th, and 99th percentiles. We also show in Figure 8 how much write data the shim layer needs to hold. As shown, less than 64 MB are on-hold (y-axis) in every period on average, with the exception of write bursts (near the end of the time line) which require the shim layer to hold up to 800 MB of writes for 100 seconds.

The main limitation of the shim layer is that rate-limiting can only be applied to writes not called under application-level *sync()* since delaying writes can reduce data durability enforced by such explicit sync. More exploration on the design space including requirements for data persistence and durability is open for future work.

5.3 Other Properties

Kelpie reveals many important SSD performance characteristics, but we acknowledge that some of them may be hard to leverage outside the FTL. For example, properties P_1-P_4 could be leveraged to "pinpoint" user data onto low/high pages (*i.e.*, a sequential write workload issued to a freshly erased drive would be striped into chunks and be spread onto flash chips with certain page layouts as probed by P_2), but would be extremely difficult to keep track of the mapping once GC kicks in. Findings in §4.6 hinted that heavier load can even reduce single I/O latency on $N_{2T}I$ (Figure 5). To leverage this, single user reads can be batched with dummy reads to improve latency, but such a radical scheme requires precise I/O control, might introduce non-trivial side-effects, and may not be generally applicable. Property P_9 and findings in §4.4 reveal that for some drives, writes can be serialized when the device queue is empty, which seems a problem more appropriate for the manufacturers to fix than an opportunity for the users to leverage. We hope that our work can spur more future exploration.

6 KELPIE (ALL FINDINGS)

This last section describes all of our findings. Kelpie's raw data set is 10 GB in size, which contains all results and graphs of running Queenie on the 21 SSD models. We release part of this data set that we can [3]. We now discuss each of the properties probed. Table 4.X denotes column X of Table 4.

P_1 **Page size**. 4KB is a general standard that applies to all non-SLC drives (except $A_{800G}P$). In contrast, SLC drives (except $T_{200G}M$) mostly use a larger page size such as 8 or 16KB. Also, in general, recent drives tend to employ a 4KB page size. We also observed some models (*e.g.*, $N_{1.6T}S$, $N_{128G}S$ and $A_{960G}P_S$) having a dual-plane structure where two parallel pages are mapped to two adjacent planes of the same chip, hence can be simultaneously accessed in parallel. Here, we keep the original page size but recommend to treat the two pages as a whole entity for further probing.

P_2 **Page type**. We categorize the SSDs into 3 classifications: SLC (with low pages only), MLC (low and high), and TLC (low, medium, and high). Low/medium/high means that the page is mapped to low/medium/high bits of the MLC/TLC cells. Based on the latencies observed, we also speculate the page-to-cell mapping pattern. For example, "`4L 4H`" means that the first four pages of each NAND chip are mapped to low pages and the next four to high pages (hence higher latency), and this pattern repeats in subsequent pages. Table 4.P_2 shows that MLC is the most popular type in our set. For MLC models, we further find three patterns of page-to-cell mappings: `1L 1H`, `4L 2H`[2], and `4L 4H`, which are more orderly patterns than non-commodity SSDs such as OpenChannel SSDs that exhibit a complex pattern of `6L 1H 2L 1H 2L 2H...1H 2L 1H` [1] (where "..." repeats the `2L 2H` pattern), as probed by a prior work [18, 4.3]. For TLC drives ($N_{2T}I$, $N_{1T}I$, and $A_{1.6T}P$), the page layout of `L/M/H` patterns are too long to be put in the table. These drives employ a complex "composite" mapping, *e.g.*, on $N_{2T}I$, the entire 0-128KB range is mapped to `L` pages, while the 128KB-256KB range follows an `2M 2H` pattern. Finally, SLC is only seen in the small-capacity, older non-NVMe SSDs (< 200 GB).

P_3 **Chunk size**. For this property, Table 4.P_3 uses KB (instead of pages) as the unit of chunk size. For example, $N_{2T}I$'s 64KB chunk size implies that the drive maps 16 consecutive 4KB pages into a chunk. Our findings show that all SLC drives use one-page chunks. For non-SLC ones, every vendor has its own configuration. For example, vendor S uses one-page chunks, vendor P configures 4-page chunks, and vendor I organizes large chunks of 16-64 pages. Here, the chunk size can also reflects the difference in FTL mapping granularity, *e.g.*, drives with one-page chunks are more likely to use a page-level FTL mapping. As a small note, $T_{64G}S$'s chunk size is marked with a "—" as the drive has only 1 chip.

P_4 **Stripe width**. As expected, SSDs with larger capacity tend to have larger stripe widths of up to 128-256 of parallel chunks (chips or planes) within a stripe, showing a massive parallelism for absorbing intensive workloads. In contrast, smaller drives usually have a stripe width of ≤64 (except $T_{200G}M$ and $A_{200G}H$). Another note is that many numbers in Table 4.P_4 are not in the power of 2, including $N_{1.6T}S$ (124), $N_{2T}I$ (186), $N_{1.6T}I$ (122), $T_{64G}I$ (20), $A_{1.6T}P$ (247), $A_{960G}P_T$ (200), $N_{1.6T}W$ (124), $A_{800G}G$ (30), and $A_{200G}H$ (253). This may indicate a couple of design choices such as parity spaces in RAIN [42, 50] or reserved/back-up chips that are not observable externally. Finally, for TLC drive $N_{1T}I$, we cannot conclusively determine the stripe width due to its complex composite mapping ("?" in Table 4).

P_5 **Channel/chip layout**. We found that larger drives prefer a "wide" setting – more channels but fewer chips per channel. For example, $N_{1.6T}S$ has a 16×8 (#channels × #chips/channel) layout which is efficient for channel parallelization but at the same time reduces the multi-chip bandwidth contention on each channel, and $A_{960G}P_S$ employs a 32×2 one. For smaller drives, the setting can vary. There is a

[2] Models with `4L 2H` actually have a "3 bit" structure (*e.g.*, the 3D NAND technology). Kelpie's results show that, however, two of the bits exhibit very similar latency levels and thus we treat both bits as low pages.

	P_1 PgSz	P_2 PgType	P_3 ChukSz	P_4 StripeW	P_5 Layout	P_6 ReadC	P_7 RBuf	P_8 WBuf	P_9 WrPra	P_{10} FluWin
$N_{1.6T}S$	4K	MLC$_{4L\,2H}$	4K	124	16×8	✓	–	40.25M$_p$	8	50ms
$N_{500G}S$	4K	MLC$_{4L\,2H}$	4K	64	4×16	✓	–	2M	1	4ms
$N_{128G}S$	4K	MLC$_{4L\,2H}$	4K	32	16×2	✓	–	1M	1	2ms
$N_{2T}I$	4K	TLC	64K	186	12×16	✗	–	11.5M	4	200ms
$N_{1.6T}I$	4K	MLC$_{1L\,1H}$	128K	122	32×4	✓	–	11.5M	2	10ms
$N_{1T}I$	4K	TLC	256K	?	?	✗	–	11M$_p$	2	3ms
$N_{1.6T}W$	4K	MLC$_{1L\,1H}$	64K	124	16×8	✗	–	NB+P	4✗	–
$N_{1.6T}M$	4K	MLC$_{1L\,1H}$	128K	128	16×8	✓	–	15M	4	∞
$A_{1.6T}P$	4K	TLC	16K	247	16×16	✓	–	NB+P	4✗	–
$A_{960G}P_S$	4K	MLC$_{4L\,4H}$	4K	64	32×2	✓	–	NB+P	4✗	–
$A_{960G}P_T$	4K	MLC$_{1L\,1H}$	16K	200	20×10	✓	–	11.5M$_p$ | 406M	4✗	200ms | 2.5s
$A_{800G}P$	8K	MLC$_{1L\,1H}$	32K	128	16×8	✓	16M	2M | 512M	4	∞
$A_{800G}G$	4K	MLC$_{1L\,1H}$	64K	30	8×4	✓	–	3.75M	2	30ms
$A_{200G}H$	8K	SLC	8K	253	16×16	✓	–	20M | 126.5M†	1	∞
$T_{480G}S$	4K	MLC$_{4L\,4H}$	4K	16	8×2	✓	–	2M | 256M	1	10ms | 5s
$T_{200G}S$	8K	SLC	8K	64	8×8	✓	–	1M	1	35ms
$T_{128G}S$	4K	MLC$_{4L\,4H}$	4K	32	16×2	✓	–	1M	1	2ms
$T_{100G}S$	8K	SLC	8K	8	8×1	✓	–	512K	1	40ms
$T_{64G}S$	16K	SLC	–	1	1×1	✓	–	4M	2	∞
$T_{64G}I$	4K	SLC	4K	20	10×2	✓	–	10M$_p$ | 810M	1	300ms | ∞
$T_{200G}M$	4K	SLC	4K	128	8×16	✓	–	64M	1	1s

Table 4: All findings in Kelpie (§6). *Every P column is described in the corresponding subsection of §6. In the first column, SSD models are denoted with the* ***protocol*** *(NVMe, SAS, or SATA), size in GB, and the* ***vendor code*** *(S, I, W, M, P, G, and H). For example, "$N_{1.6T}S$" is a 1.6TB NVMe drive from vendor S. We do not reveal the full names of the vendors. Other notes: In some columns, for brevity we omit "B" (bytes), hence "K/M" means "KB/MB." "–" means not applicable. "?" implies unsuccessful probing. For other specific labels in every column (such as ✓, ✗, p, NB, P, †, |, ∞), please consult the corresponding subsection in §6.*

"wide" setting similar as above (the 128GB $N_{128G}S$ with 16×2) and a "deep" setting (the 500GB $N_{500G}S$ with 4×16); the deep setting is prone to many-chip bandwidth contention on every channel. We also see two drives with non-power-of-2 #channels: $T_{64G}I$ (10×2) and $A_{960G}P_T$ (20×10). For $N_{1T}I$, the layout is labeled "?" in Table 4 due to the same reason – the composite mapping issue. We found an interesting anomaly in drive $N_{1.6T}S$ where the I/Os to separate channels seem to be *contending* with one another. Upon further investigation, we revealed a channel "grouping," where the 16 channels on $N_{1.6T}S$ are evenly divided into 4 pools, and I/Os to the *same* pool will contend with one another (with an overhead of 10µs) even if the I/Os target different channels in the pool. This contradicts the traditional view of channel parallelism. The root cause remains unknown.

P_6 **Read performance consistency**. As discussed in Section 4.1, 3 enterprise-level SSDs ($N_{2T}I$, $N_{1T}I$ and $N_{1.6T}W$) exhibit degraded performance under certain read sizes (labeled as ✗ in Table 4.P_6). It could be a fact that applications must live with or a bug/defect inside the SSD. With the former, OS/applications can take remedies such as altering the read sizes (§5.1), but this requires a preliminary probing cost to determine whether the SSDs have this problem. If this is a bug/defect, then SSD vendors might want to adopt this kind of test from Queenie to their device quality tests.

P_7 **Read buffer capacity**: Our results show that read buffers are becoming extinct in modern SSDs (almost all "–" in Table 4.P_7), with the exception of one SAS drive $A_{800G}P$ which has a 16MB read buffer. This phenomenon can be attributed to the increasing speed of NAND and the larger DRAM caches in higher storage layers that make internal read cache obsolete. However, for SATA/SAS drives with higher read latencies (*e.g.*, hundreds of µs), a read buffer can still be beneficial (see §4.5).

P_8 **Write buffer capacity**. **(a)** The first trend observed is that write buffering is still prevalent but new drives only provision a small buffer. TB-scale drives (*e.g.*, $N_{1.6T}S$, $N_{2T}I$, $N_{1.6T}I$, $N_{1T}I$, and $N_{1.6T}M$) use only a ≤64MB buffer, perhaps for lower cost and because a small buffer forces frequent small flushes, more favorable than large flushes that can cause long blocking (§4.2 and §5.2) and trigger large GCs. **(b)** Another trend we see is 2-level buffering (§4.2), found in 5 drives labeled with a pair of first and second level sizes in Table 4.P_8. For example, $A_{800G}P$'s "2M | 512M" implies a two-level buffer with 2MB and 512MB for the first and second levels, respectively. In one drive, $A_{200G}H$, the second-level write has two policies (marked with † in the table): flush either after a threshold of 126.5MB or 1280 I/Os of writes has been exceeded, where an "I/O" can be of any size. **(c)** We found that 4 drives ($N_{1.6T}S$, $N_{1T}I$, $T_{64G}I$, and $A_{960G}P_T$)

can choose to flush their buffers partially even when they are not idle. For example, under sequential writes, we see $N_{1.6T}S$ constantly flushes 5.75MB of data out of its 40.25MB full buffer capacity, $N_{1T}I$ 3MB out of 11MB, $T_{64G}I$ 4MB out of 10MB, and $A_{960G}P_T$ 7.5MB out of 11.5MB, labeled with "p" in Table 4.P_8. For SSDs, periodic partial flushes are a double-edged sword: on the one hand they tone down the blocking impact, on the other hand they can increase write amplification (*e.g.*, new overwrites to the same LPNs just recently flushed). **(d)** Finally, we observed 3 drives ($A_{1.6T}P$, $A_{960G}P_S$, and $N_{1.6T}W$) that rarely show write latency spikes even with a full write bandwidth experiment. Our assumption is that these drives perform partial flushes and are able to optimize them without blocking incoming I/Os. In such an optimized design, we cannot conclusively probe their write buffer capacities and mark them with "NB + P" (non-blocking and partial) in Table 4.P_8.

P_9 **Write parallelism**. Although many SSDs have a high stripe width (P_4) to support high read parallelism, this is not the case for write parallelism. For example, $N_{1.6T}S$, an enterprise-level drive that is able to handle 124 concurrent chunk reads, can only allow 8 concurrent writes. Indeed, 8 concurrent writes is the highest write parallelism that we observe, while others only reach 2 or 4 as shown in Table 4.P_9. However, we caution that read and write parallelism are not directly comparable, mainly because read I/Os fetch data from the NAND (with the absence of internal read cache) while write I/Os are absorbed by the internal RAM. As presented earlier in Section 4.4, we found anomalies where 4 drives ($A_{960G}P_T$, $A_{960G}P_S$, $A_{1.6T}P$, and $N_{1.6T}W$) exhibit "serialized" concurrent writes when the I/Os are inserted to the device queue without any pending writes. These anomalies are labeled with "✗" in Table 4.P_9.

P_{10} **Internal flush window**. **(a)** Section 5.2 successfully demonstrates that probing the internal *flush speed* can be useful for rate-limiting optimizations. Flush speed is a function of buffer size (P_8) divided by the "flush window" (P_{10}). In earlier experiments in Figure 3 of Section 4.3, the flush window is essentially equal to how much time the OS/user should let the device remain idle to prevent write latency spikes. This window value is shown in Table 4.P_{10} (*e.g.*, mostly around 2 to 300ms). **(b)** The ∞ label in Table 4.P_{10} indicates the worst-case scenario. For 5 devices ($N_{1.6T}M$, $A_{800G}P$, $A_{200G}H$, $T_{64G}S$, and $T_{64G}I$), some writes will experience latency spikes regardless of the length of the idle period. This basically implies that the write flush is never triggered unless the space threshold (*e.g.*, 90% full) has been reached. **(c)** We also found that SSDs from the same vendor have different strategies. $N_{1T}I$, $N_{1.6T}I$, and $N_{2T}I$ are three TB-scale drives from the same vendor I with similar buffer sizes but use dramatically different amount of times to clean their buffers; $N_{2T}I$ does so lazily within a 200ms idle window, $N_{1.6T}I$ requires only 10ms, while $N_{1T}I$ is very aggressive and flushes within 3ms. **(d)** For drives with two-level buffering, likewise, we use a pair of first and second level time windows, which shows that the drives apply different window policies for the two levels. For example, $T_{480G}S$'s "10ms | 5s" window value implies that it aggressively flushes the first level buffer in 10ms but only empties the second level buffer lazily in 5s. **(e)** Finally, "—" in Table 4.P_{10} highlights that the corresponding devices ($N_{1.6T}W$, $A_{960G}P_S$, and $A_{1.6T}P$) perform the optimized non-blocking, partial flushes discussed above. Here, we cannot probe the flush window value.

7 VALIDATION AND CONCLUSION

To validate the probing methods and the analysis results, one direct approach is via confirmation from the SSD vendors. We contacted three major vendors: two were unwilling to cooperate due to the sensitive proprietary nature of these black-box SSDs; one vendor was willing to help, but with strict limitations that make the process extremely slow. We were given only one chance to verify with a simple "right/wrong" feedback with no room for further discussion (because again it is sensitive information). This makes troubleshooting hard: a "wrong" answer could be due to misunderstanding of the properties (*i.e.*, the vendor may have different names and interpretations of these properties) or inaccurate probing methods. After six months, we have only verified one property for three of the SSDs, due to these communication limitations.

Because this method is not scalable, we pursued a different method: we used our (old) open-channel SSD [1] and were able to verify P_1, P_3, and P_4 before it became erratic. We recently moved to SSD emulators such as FEMU [35] and were able to verify P_8 at the time of writing.

Overall, *our findings should be treated as empirically-driven, user-observed conjectures, as the real truth is only known to the vendors.* Nevertheless, we believe this paper makes strong contributions: a comprehensive set of SSD probing techniques that learn black-box information, many interesting forms of deep findings, and case studies that show how the I/O stack can use the learned knowledge.

ACKNOWLEDGMENTS

We thank Gala Yadgar, our shepherd, and the anonymous reviewers for their tremendous feedback and comments. We also thank Anirudh Badam and Abhijeet Gole for their helpful discussions and comments on this paper. This material was supported by funding from NSF (grant Nos. CNS-1526304, CNS-1405959, CCF-2028427, and CCF-2119184) as well as generous donations from Facebook Faculty Research Award, Google Faculty Research Award, NetApp Faculty Fellowship, and CERES Center for Unstoppable Computing.

REFERENCES

[1] 2018. Open-Channel Solid State Drives Specification. http://lightnvm.io/docs/OCSSD-2_0-20180129.pdf.

[2] 2019. SOLID STATE DRIVE (SSD) MARKET - GROWTH, TRENDS, AND FORECAST (2019 - 2024). https://www.mordorintelligence.com/industry-reports/solid-state-drive-market.

[3] 2022. Queenie GitHub Homepage. https://github.com/ucare-uchicago/Queenie.

[4] Abutalib Aghayev and Peter Desnoyers. 2015. Skylight-A Window on Shingled Disk Operation. In *Proceedings of the 13th USENIX Symposium on File and Storage Technologies (FAST)*.

[5] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. 2008. Design Tradeoffs for SSD Performance. In *Proceedings of the USENIX Annual Technical Conference (ATC)*.

[6] Matias Bjørling, Abutalib Aghayev, Hans Holmberg, Aravind Ramesh, Damien Le Moal, Gregory R. Ganger, and George Amvrosiadis. 2021. ZNS: Avoiding the Block Interface Tax for Flash-based SSDs. In *Proceedings of the 2021 USENIX Annual Technical Conference (ATC)*.

[7] Matias Bjørling, Javier González, and Philippe Bonnet. 2017. LightNVM: The Linux Open-Channel SSD Subsystem. In *Proceedings of the 15th Usenix Conference on File and Storage Technologies* (Santa clara, CA, USA) *(FAST'17)*. USENIX Association, USA, 359–373.

[8] Simona Boboila and Peter Desnoyers. 2010. Write Endurance in Flash Drives: Measurements and Analysis. In *Proceedings of the 8th USENIX Symposium on File and Storage Technologies (FAST)*.

[9] Simona Boboila and Peter Desnoyers. 2011. Performance Models of Flash-Based Solid-State Drives for Real Workloads. In *Proceedings of the 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST '11)*. IEEE Computer Society, USA, 1–6.

[10] Zhen Cao, Vasily Tarasov, Hari Prasath Raman, Dean Hildebrand, and Erez Zadok. 2017. On the Performance Variation in Modern Storage Stacks. In *Proceedings of the 15th Usenix Conference on File and Storage Technologies* (Santa clara, CA, USA) *(FAST'17)*. USENIX Association, USA, 329–343.

[11] Li-Pin Chang. 2008. Hybrid Solid-State Disks: Combining Heterogeneous NAND Flash in Large SSDs. In *Proceedings of the 2008 Asia and South Pacific Design Automation Conference* (Seoul, Korea) *(ASP-DAC '08)*. IEEE Computer Society Press, Washington, DC, USA, 428–433.

[12] Feng Chen, Rubao Lee, and Xiaodong Zhang. 2011. Essential Roles of Exploiting Internal Parallelism of Flash Memory Based Solid State Drives in High-speed Data Processing. In *Proceedings of the 17th International Symposium on High Performance Computer Architecture (HPCA-17)*.

[13] John Colgrove, John D. Davis, John Hayes, Ethan L. Miller, Cary Sandvig, Russell Sears, Ari Tamches, Neil Vachharajani, and Feng Wang. 2015. Purity: Building Fast, Highly-Available Enterprise Flash Storage from Commodity Components. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD)*.

[14] Timothy E. Denehy, John Bent, Florentina I. Popovici, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2004. Deconstructing Storage Arrays. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XI)*.

[15] Peter Desnoyers. 2014. Analytic Models of SSD Write Performance. In *ACM Transactions on Storage (TOS)*.

[16] Laura M. Grupp, John D. Davis, and Steven Swanson. 2013. The Harey Tortoise: Managing Heterogeneous Write Performance in SSDs. In *Proceedings of the 2013 USENIX Annual Technical Conference (ATC)*.

[17] Kyuhwa Han, Hyunho Gwak, Dongkun Shin, and Jooyoung Hwang. 2021. ZNS+: Advanced Zoned Namespace Interface for Supporting In-Storage Zone Compaction. In *Proceedings of the 15th Symposium on Operating Systems Design and Implementation (OSDI)*.

[18] Mingzhe Hao, Huaicheng Li, Michael Hao Tong, Chrisma Pakha, Riza O. Suminto, Cesar A. Stuardo, Andrew A. Chien, and Haryadi S. Gunawi. 2017. MittOS: Supporting Millisecond Tail Tolerance with Fast Rejecting SLO-Aware OS Interface. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP)*.

[19] Mingzhe Hao, Gokul Soundararajan, Deepak Kenchammana-Hosekote, Andrew A. Chien, and Haryadi S. Gunawi. 2016. 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* (Santa Clara, CA) *(FAST'16)*. USENIX Association, USA, 263–276.

[20] Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim, Henry Hoffmann, and Haryadi S. Gunawi. 2020. LinnOS: Predictability on Unpredictable Flash Storage with a Light Neural Network. In *Proceedings of the 14th Symposium on Operating Systems Design and Implementation (OSDI)*.

[21] Jun He, Sudarsun Kannan, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2017. The Unwritten Contract of Solid State Drives. In *Proceedings of the 2017 EuroSys Conference (EuroSys)*.

[22] Seongcheol Hong and Dongkun Shin. 2010. NAND Flash-Based Disk Cache Using SLC/MLC Combined Flash Memory. In *2010 International Workshop on Storage Network Architecture and Parallel I/Os*. 21–30.

[23] Xiao-Yu Hu, Robert Haas, and Eleftheriou Evangelos. 2011. Container Marking: Combining Data Placement, Garbage Collection and Wear Leveling for Flash. In *Proceedings of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*.

[24] H. Howie Huang, Shan Li, Alex Szalay, and Andreas Terzis. 2011. Performance Modeling and Analysis of Flash-Based Storage Devices. In *Proceedings of the 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST '11)*. IEEE Computer Society, USA, 1–11.

[25] Xavier Jimenez, David Novo, and Paolo Ienne. 2014. Wear Unleveling: Improving NAND Flash Lifetime by Balancing Page Endurance. In *Proceedings of the 12th USENIX Symposium on File and Storage Technologies (FAST)*.

[26] Myoungsoo Jung and Mahmut Kandemir. 2013. Revisiting Widely Held SSD Expectations and Rethinking System-Level Implications. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems* (Pittsburgh, PA, USA) *(SIGMETRICS '13)*. Association for Computing Machinery, New York, NY, USA, 203–216.

[27] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, , and Sangyeun Cho. 2014. The Multi-streamed Solid-State Drive. In *the 6th Workshop on Hot Topics in Storage and File Systems (HotStorage)*.

[28] Jihun Kim, Joonsung Kim, Pyeongsu Park, Jong Kim, and Jangwoo Kim. 2018. SSD Performance Modeling Using Bottleneck Analysis. *IEEE Comput. Archit. Lett.* 17, 1 (January 2018), 80–83. https://doi.org/10.1109/LCA.2017.2779122

[29] Jaeho Kim, Kwanghyun Lim, Youngdon Jung, Sungjin Lee, Changwoo Min, and Sam H. Noh. 2019. Alleviating Garbage Collection Interference Through Spatial Separation in All Flash Arrays. In *Proceedings of the 2019 USENIX Annual Technical Conference (ATC)*.

[30] Joonsung Kim, Pyeongsu Park, Jaehyung Ahn, Jihun Kim, Jong Kim, and Jangwoo Kim. 2018. SSDcheck: Timely and Accurate Prediction of Irregular Behaviors in Black-Box SSDs. In *51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-51)*.

[31] Jaehong Kim, Sangwon Seo, Dawoon Jung, Jin-Soo Kim, and Jaehyuk Huh. 2011. Parameter-Aware I/O Management for Solid State Disks (SSDs). In *IEEE Transactions on Computers (TC)*.

[32] Jae-Hong Kim, Dawoon Jung, Jin-Soo Kim, and Jaehyuk Huh. 2009. A Methodology for Extracting Performance Parameters in Solid State Disks (SSDs). In *Proceedings of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*.

[33] Pradeep Kumar and H. Howie Huang. 2017. Falcon: Scaling IO Performance in Multi-SSD Volumes. In *Proceedings of the 2017 USENIX Annual Technical Conference (ATC)*.

[34] Sungjin Lee, Ming Liu, Sangwoo Jun, Shuotao Xu, Jihong Kim, and Arvind Arvind. 2016. Application-Managed Flash. In *Proceedings of the 14th Usenix Conference on File and Storage Technologies* (Santa Clara, CA) *(FAST'16)*. USENIX Association, USA, 339–353.

[35] Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Matias Bjørling, and Haryadi S. Gunawi. 2018. The CASE of FEMU: Cheap, Accurate, Scalable and Extensible Flash Emulator. In *Proceedings of the 16th USENIX Symposium on File and Storage Technologies (FAST)*.

[36] Huaicheng Li, Martin L. Putra, Ronald Shi, Xing Lin, Gregory R. Ganger, and Haryadi S. Gunawi. 2021. IODA: A Host/Device Co-Design for Strong Predictability Contract on Modern Flash Storage. In *Proceedings of the 28th ACM Symposium on Operating Systems Principles (SOSP)*.

[37] Shan Li and H. Howie Huang. 2010. Black-Box Performance Modeling for Solid-State Drives. In *Proceedings of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*.

[38] Yongkun Li, Patrick P. C. Lee, and John C. S. Lui. 2011. Stochastic Modeling of Large-Scale Solid-State Storage Systems: Analysis, Design Tradeoffs and Optimization. In *Proceedings of the 2011 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*.

[39] Heiner Litz, Javier Gonzalez, Ana Klimovic, and Christos Kozyrakis. 2022. RAIL: Predictable, Low Tail Latency for NVMe Flash. *ACM Trans. Storage* 18, 1, Article 5 (jan 2022), 21 pages. https://doi.org/10.1145/3465406

[40] Jian Ouyang, Shiding Lin, Song Jiang, Zhenyu Hou, Yong Wang, and Yuanzheng Wang. 2014. SDF: Software-Defined Flash for Web-Scale Internet Storage System. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[41] Torben Kling Petersen and John Bent. 2017. Hybrid Flash Arrays for HPC Storage Systems: An Alternative to Burst Buffers. In *IEEE High Performance Extreme Computing Conference (HPEC)*.

[42] Sergey Platonov. 2018. RAIN: Reinvention of RAID for the World of NVMe. In *Flash Memory Summit*.

[43] Jiri Schindler, John Linwood Griffin, Christopher R. Lumb, and Gregory R. Ganger. 2002. Track-Aligned Extents: Matching Access Patterns to Disk Drive Characteristics. In *Proceedings of the 1st USENIX Symposium on File and Storage Technologies (FAST)*.

[44] Benny Van Houdt. 2013. A Mean Field Model for a Class of Garbage Collection Algorithms in Flash-Based Solid State Drives. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems* (Pittsburgh, PA, USA) *(SIGMETRICS '13)*. Association for Computing Machinery, New York, NY, USA, 191–202.

[45] Matthieu Viry. [n.d.]. Compute Natural Breaks in Python (Fisher-Jenks algorithm). https://github.com/mthh/jenkspy.

[46] Brent Welch and Geoffrey Noer. 2013. Optimizing a Hybrid SSD/HDD HPC Storage System Based on File Size Distributions. In *Proceedings of the 29th IEEE Symposium on Massive Storage Systems and Technologies (MSST)*.

[47] Bruce L. Worthington, Gregory R. Ganger, Yale N. Patt, and John Wilkes. 1995. On Line Extraction of SCSI Disk Drive Parameters. In *Proceedings of the 1995 ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*.

[48] Kan Wu, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. 2019. Towards an Unwritten Contract of Intel Optane SSD. In *the 11th Workshop on Hot Topics in Storage and File Systems (HotStorage)*.

[49] Suzhen Wu, Yanping Lin, Bo Mao, and Hong Jiang. 2016. Garbage Collection Aware Cache Management with Improved Performance for Flash-based SSDs. In *Proceedings of the 30th International Conference on Supercomputing (ICS)*.

[50] Shiqin Yan, Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Andrew A. Chien, and Haryadi S. Gunawi. 2017. Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs. In *15th USENIX Conference on File and Storage Technologies (FAST 17)*. USENIX Association, Santa Clara, CA, 15–28.

[51] Kamen Yotov, Keshav Pingali, and Paul Stodghill. 2005. Automatic Measurement of Memory Hierarchy Parameters. In *Proceedings of the 2005 ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*.

[52] Jianquan Zhang, Dan Feng, Jianlin Gao, Wei Tong, Jingning Liu, Yu Hua, Yang Gao, Caihua Fang, Wen Xia, Feiling Fu, and Yaqing Li. 2016. Application-Aware and Software-Defined SSD Scheme for Tencent Large-Scale Storage System. In *Proceedings of 22nd IEEE International Conference on Parallel and Distributed Systems (ICPADS)*.

[53] Aviad Zuck, Philipp Guhring, Tao Zhang, Donald E. Porter, and Dan Tsafrir. 2019. Why and How to Increase SSD Performance Transparency. In *The 17th Workshop on Hot Topics in Operating Systems (HotOS XVII)*.