SSD-based Workload Characteristics and Their Performance Implications

GALA YADGAR, Computer Science Department, Technion, Israel
MOSHE GABEL, SHEHBAZ JAFFER, and BIANCA SCHROEDER, Department of Computer Science, University of Toronto, Canada

Storage systems are designed and optimized relying on wisdom derived from analysis studies of file-system and block-level workloads. However, while SSDs are becoming a dominant building block in many storage systems, their design continues to build on knowledge derived from analysis targeted at hard disk optimization. Though still valuable, it does not cover important aspects relevant for SSD performance. In a sense, we are "searching under the streetlight," possibly missing important opportunities for optimizing storage system design.

We present the first I/O workload analysis designed with SSDs in mind. We characterize traces from four repositories and examine their "temperature" ranges, sensitivity to page size, and "logical locality." We then take the first step towards correlating these characteristics with three standard performance metrics: write amplification, read amplification, and flash read costs. Our results show that SSD-specific characteristics strongly affect performance, often in surprising ways.

CCS Concepts: • **Information systems** → **Flash memory**; **Storage management**; *Storage architectures*; *Hierarchical storage management*;

Additional Key Words and Phrases: SSD, I/O workload analysis, locality, workload characterization

ACM Reference format:
Gala Yadgar, Moshe Gabel, Shehbaz Jaffer, and Bianca Schroeder. 2021. SSD-based Workload Characteristics and Their Performance Implications. *ACM Trans. Storage* 17, 1, Article 8 (January 2021), 26 pages.
https://doi.org/10.1145/3423137

1 INTRODUCTION

Characterizations of file-system and block-level workloads are often used in the design and optimization of various levels of the storage stack. Aspects such as file and object placement, metadata management, replication, caching, array configuration, and power management are optimized according to our understanding of the target workloads. The storage community has invested considerable effort in studying storage workloads. These studies naturally focus on those aspects of the workload relevant to optimization of the underlying storage component.

Authors' addresses: G. Yadgar, Computer Science Department, CS Taub Building, Technion, Haifa 3200003; email: gala@cs.technion.ac.il; M. Gabel, S. Jaffer, and B. Schroeder, Department of Computer Science, University of Toronto, 10 King's College Road, Toronto, ON M5S 3G4; emails: mgabel@cs.toronto.edu, shehbaz.jaffer@mail.utoronto.ca, bianca@cs.toronto.edu.

1553-3077/2021/01-ART8 $15.00
https://doi.org/10.1145/3423137

Analyses of file-system workloads examine the distributions of file size, age, and functional lifetime, correlating them with file extension and directory structure [3, 40, 49]. However, I/O workload analyses focus on factors crucial for managing low-level devices and servers: inter-reference gaps, working set sizes, and access skew are relevant for cache management, while request sizes, idle and inter-arrival times, and read/write ratio are important for hard disk performance [10, 17, 27, 28, 33, 48, 60]. Sequentiality—the portion and nature of sequential accesses—is relevant for both: Sequential accesses can be an order of magnitude faster than random accesses on hard disks, but they can also pollute the cache and render it utterly useless. Thus, they often receive special attention in workload analysis [34].

Workload analysis studies have been focusing on the same attributes and characteristics for many years. The storage landscape is changing, however, as flash-based solid-state drives (SSDs) are gradually replacing hard disks in many data centers. Although they present a similar block interface, SSDs differ from hard disks in several ways. The most notable are their fast random access, asymmetric read and write performance, and out-of-place updates, which require dynamic mapping between logical and physical locations as well as garbage collection, both implemented in the SSD firmware.

Consequently, SSDs have different optimization goals than hard disks. For example, rather than organizing data to optimize for fast sequential reads, data is organized on SSDs with the objective of maximizing parallelism and minimizing garbage collection overheads. The need for dynamic mapping leads SSD optimizations to target data movement, rather than data placement [7, 65]. At the same time, they allow for complex optimizations and fast adjustment to workload changes.

This shift in storage device optimization calls for a new understanding of I/O workloads. Although existing analyses are still useful, they were designed for optimizing hard disks. Even analyses and discussions made in the context of SSD optimization [14, 42, 44, 56, 73] consider traditional metrics such as working set size, request rate, and inter-reference gaps. Relying only on aspects relevant for hard disk optimizations means we might be "searching under the streetlight," missing interesting attributes or phenomena relevant for SSD performance, as well as opportunities for optimizing the entire storage system.

This work attempts to characterize I/O workloads with SSDs in mind. We analyze over 100 traces from four public repositories [28, 31, 43, 58], with the goal of identifying characteristics relevant to SSD design and performance. We consider *temperature ranges* (rather than just the amount of "hot" data they contain), *logical locality* (which generalizes both simple spatial locality and sequentiality), and their sensitivity to increasing SSD page sizes. Our initial characterization results were presented in a previous workshop publication [64].

We then correlate these characteristics with SSD performance. We focus our analysis on three flash performance measures. (1) *Write amplification*: the ratio between the amount of data written on the device and that written by the application is correlated with high tail latencies and reduced durability. (2) *Read amplification*: the ratio between the amount of data read from the SSD and the amount of data requested by the application. (3) Flash read costs, which include the low-level latency of reading the data from the flash media and transferring it to the controller.

Our findings confirm the relevance of our suggested characteristics to SSD design and optimization:

- Increasing the number of logical partitions within the SSD reduces write amplification. However, this requires substantial overhead and carries diminishing returns. The optimal number of partitions is smaller than that suggested in previous studies.
- Read amplification increases with page size and depends mainly on the distribution of request sizes. However, repeated accesses to an address range significantly reduce read amplification. The extent of this reduction can be estimated by the logical spatial locality.

- Flash read costs can be efficiently reduced by selectively reading partial byte ranges from large pages. Our simple greedy algorithm is sufficient for this purpose, demonstrating the potential benefit of exporting a "partial read" interface to hosts.

The rest of our article is organized as follows: Section 2 gives necessary background on flash-based SSDs and describes the workloads used in our analysis. Section 3 presents our metrics for characterizing a workload's skew by its temperature ranges, and correlates these metrics with write amplification. In Section 4, we characterize the alignment costs incurred by the device's page size and show how it affects write amplification and read costs. Our new locality metric is presented in Section 5, where we also show how it affects a workload's read amplification. We discuss some limitations and implications of our analysis in the context of related work in Section 6, and we conclude in Section 7.

2 BACKGROUND

2.1 Flash-based SSDs

Data on a NAND flash chip is stored in *cells*, whose voltage level indicates their bit value. Cells are organized into *pages*, which are equivalent to hard disk sectors and are the read and write units. Once written, flash pages must be erased before they can be written again. The erase unit is an entire *block*, which typically consists of 64–256 pages. A flash-based SSD is composed of multiple flash *chips*, which can be accessed in parallel through multiple *channels* to increase read and write throughput. To read a page, the SSD controller issues a READ command to the flash controller. This initiates the sensing of the content of the flash cells and the transfer of data to a dedicated register. The time required for this transfer, tR, is typically 25–100 μsecs. Once the entire page's content is in the register, the SSD controller may fetch any byte range within that page into its RAM, at a throughput of 16–40 nsecs per byte, referred to as the *cycle time*, or *serial access speed*.

SSD controllers implement out-of-place writes: When the user issues a write command, the controller writes new data on a clean physical flash page and marks the old copy of this data as invalid. *Overprovisioning* is the extra physical capacity allocated to accommodate this update scheme. The mapping between logical to physical pages is managed by the *flash translation layer (FTL)*, which is also in charge of *garbage collection*—selecting a victim block, copying its valid pages to another block, and erasing it to generate space for additional writes. The extra page writes generated by garbage collection are referred to as *write amplification*, which is defined as the ratio between the amount of data written on the device and that written by the application. To minimize write amplification, garbage collection is typically performed greedily by choosing as victim the block with the least valid pages. Previous studies showed that separating data into multiple logical partitions according to their update frequency (*temperature*) further reduces write amplification [16, 57].

Flash-based SSD technology has gradually shifted from *single level cells (SLC)* that store a single bit value to *multi-level cells (MLC)* and *three-level cells (TLC)* that store two or three bits, respectively. This transition is accompanied by a continuous increase in flash page size, from 2–4 KB to 8–16 KB [15, 36, 50]. This trend continues with the introduction of TLC and QLC (*four-level cells*) 3D flash chips, which use 16 KB pages as well [54, 68]. As a result, write amplification might increase considerably if the application or file system update data in finer granularity than the page size. Read costs may increase as well, as page-read latency also depends on the amount of data transferred to the SSD controller and then to the host.

2.2 Workloads

We analyze block-level traces of storage workloads from four online repositories: the University of Massachusetts (SPC traces) [58], Microsoft Research at Cambridge [43], Microsoft production

Table 1. Trace Repositories

Repository:	UMass [58]	MSR [43]	MS-Prod [28]	FIU [31]	RocksDB [55]
Number of traces	36	34	43	9	1
Write requests (M)	0.01–0.4	0.01–58	0.01–9	1–410	9.6
Median write size	1–48 KB	4–64 KB	1–64 KB	4 KB	1,024 KB
Duration	12 hours	1 week	6–24 hours	3 weeks	9.7 hours
Volume size (GB)	0.07–650	8–820	9–3200	8–400	512
Server categories	database	development, file, web	database, file, mail, development, web	file, mail, web	key-value store

Additional details are available in the original publications.

servers [28], and Florida International University [31]. We focus on common attributes available in all these traces: request arrival time (relative to the beginning of the trace), volume number, offset (in bytes relative to the beginning of the volume), request size (in bytes), and I/O operation (either read or write).

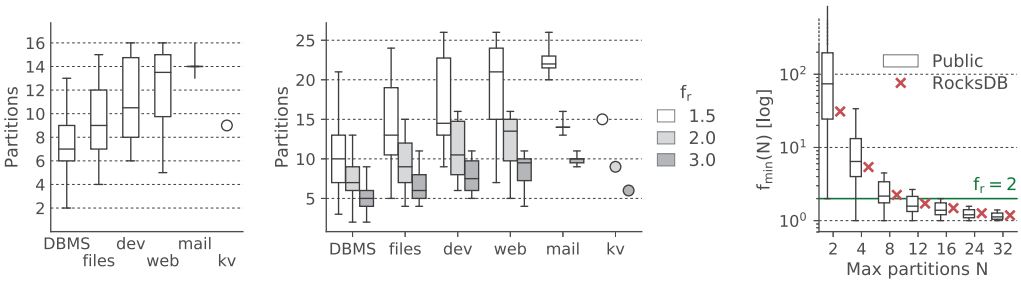
In addition, we analyze a new trace collected in our lab on a 28-core, 128 GB host with two 512 GB Samsung 970 Pro NVMe SSD Drives, running Ubuntu 18.04 LTS with Linux 5.0.0 kernel. We ran the YCSB-0.15.0 [13] benchmark with RocksDB in embedded mode, and used blktrace version 2.0 [8] to record the trace. Both SSDs are formatted with ext4 with default journaling mode (data=ordered). To avoid I/O interference from data collection, the first SSD stores all blktrace output, while the second hosts YCSB and RocksDB. We chose YCSB Workload A: 50% reads and 50% updates of 1B operations on 250M records. We run this setup for 9.7 hours, which results in generating over 352M block I/O requests at the file system level writing a total of 6.8 TB to the disk, with a read throughput of 90 MBps and a write throughput of 196 MBps. Our new trace is also available online [55].

The focus of our analysis is the block-device level. Thus, in workloads that access multiple volumes attached to the same server, we analyzed each volume as a separate trace. Our analysis includes only traces with at least 10,000 requests and at least 5,000 write requests, 123 traces in total. As part of our analysis, we categorize the workloads according to the server's function. Table 1 summarizes the characteristics of each repository.

3 TEMPERATURE RANGES

Workload skew is traditionally leveraged to optimize performance by use of caches. If the cache is large enough to store the application's *working set*, it can serve most requests without accessing the underlying storage. In this context, rules of thumb such as the 80-20 rule—which states that 80% of requests access 20% of the data—are often used to estimate the required cache size for a workload. Thus, traditional analyses characterize the skew of a workload by assessing its working set size or amount of *hot* (frequently accessed) data. While those are sufficient for optimizing hard disk and cache performance, SSD optimization can benefit from more detailed analysis.

In the context of FTL design, separating hot and cold data into different logical partitions has been shown to minimize write amplification (caused by out-of-place updates) and, respectively, garbage collection costs and cell wear [16, 70]. Such separation is also useful for other optimizations such as wear leveling and page reuse [67]. Consequently, many FTLs separate data into two partitions. Stoica and Ailamaki [57] have shown that several *temperatures* can be grouped into the same partition without increasing write amplification, as long as the access skew within each partition is sufficiently small. We build on this analysis and characterize workloads according to the minimum number of partitions they require. We then quantify the cost of allocating fewer



(a) Required partitions, $f_r \leq 2$ (b) Partitions required for different f_r values (c) Lowest ratio

Fig. 1. Required number of partitions for $f_r \leq 2$ (a) and for additional f_r values (b) . (c) $f_{min}(N)$ when restricting the number of partitions.

partitions when the optimal number is too high. We are specifically interested in the common case of two partitions, for hot and cold data.

3.1 Characterizing Temperature Ranges

When a logical page is written to an SSD, the FTL marks its previous physical location as invalid and chooses a chip and a plane to write the page to, according to its striping and load-balancing schemes. It then writes the page to an *active block* in the plane—a block that has been erased and was not fully written yet. Separating data into N logical partitions requires N active blocks in each plane.

Following the notation of Stoica and Ailamaki [57], let f_i be the *update frequency* of page i. Ideally, each partition would contain pages with the same update frequency. This is clearly unrealistic, as modern chips have as few as 512 blocks per plane [1]. Instead, logical pages with several access frequencies are grouped into each partition. Let $f_r(p)$ be the *update frequency ratio* in partition p, which is the ratio between the maximum and minimum update frequencies of pages stored in p. f_r denotes the maximal ratio across *all* partitions. The theoretical analysis in Reference [57] suggests that ensuring $f_r \leq 2$ is sufficient for minimizing garbage collection overhead. This result, however, does not take into account the overhead of maintaining the required number of partitions.

To see whether this optimization is realistic given modern chip organization, we calculate the minimal number of partitions required to ensure $f_r \leq 2$. We rank the logical sectors in each workload according to their update frequency and greedily assign them to partitions: We start with the first partition, assigning to it the page i with maximal f_i, as well as all pages $\{j\}$ with $f_j \geq f_i/2$. We then assign the next page to the second partition, and so on. We used sector sizes of 512 bytes and 4 KB for the analysis of the HDD-based traces and the RocksDB SSD-based trace, respectively, to reflect the access granularity observed in these traces. We discuss access granularity in more detail in Section 4.

Our results show that the minimal number of required partitions ranges between 2 and 16 and that it varies between categories. Figure 1 (a) shows the 25th, 50th, and 75th percentiles for each category, with the vertical lines showing the minimum and maximum required number of partitions for each category (the key-value category includes the single RocksDB trace and is thus represented by a small circle). For example, 9 partitions are sufficient for 75% of DBMS workloads and for the key-value workload, while 75% of mail server workloads require at least 14 partitions. We repeated this experiment with varying restrictions on f_r. As expected, more partitions are required to maintain a lower ratio. The minimum number of required partitions ranges from 3 to 26 for $f_r \leq 1.5$, and from 2 to 11 for $f_r \leq 3$ (Figure 1(b)).

⟹ **Finding 1:** Achieving the theoretical lower bound on write amplification requires a large number of partitions.

To understand the implications of these results, consider a state-of-the-art enterprise SSD with 512 blocks per plane and 28% overprovisioning. Each plane will have 400 available logical blocks. Thus, to maintain 16 partitions, 4% of the blocks must be allocated as active blocks, and on average 2% of the logical capacity will be unutilized. In addition to this overhead, one must consider the cost of maintaining and classifying pages into several partitions in the SSD's RAM.

SSD designs might limit the number of partitions to exploit specialized hardware, or to minimize partitioning overhead. To estimate the effect of sub-optimal partitioning, we repeated the procedure described above to exhaustively find $f_{min}(N)$—the lowest ratio for which the greedy partitioning scheme results in N—a predetermined number of partitions. Figure 1(c) shows the distribution of $f_{min}(N)$ across all traces, for N between 2 and 32. With a limit of 2 and 4 partitions, the lowest ratio can be as high as 1,000[1] and 33.8, respectively, with respective medians 77.7 and 6.6. However, when the number of partitions is limited to a modest 8, $f_{min}(N)$ does not exceed 5, and is 2 or less for 28% of the traces. We highlight $f_{min}(N)$ of the SSD-based RocksDB workload, demonstrating that it lies well within the range of $f_{min}(N)$ values of the HDD-based public workloads. In the rest of our analysis, we include the results for this workload within the aggregate statistics and discuss them explicitly only if they exhibit special characteristics. In the following, we evaluate the performance implications of f_r by examining its effect on write amplification.

3.2 Evaluation Setup for Write Amplification

To evaluate the performance implications of f_r, we analyze the write amplification of the different workloads, varying the number of partitions. Although the write amplification of an SSD is not directly observed by the application running at the host, previous studies correlated it with observable performance metrics, such as latency and throughput [21, 39, 67], and others use it as their main performance optimization goal [4, 15, 57].

Analyzing write amplification in the context of workload characterization presented several unique tradeoffs that are different from traditional experimental evaluations. Our evaluation setup was designed with the following goal in mind: produce results that would be as platform-independent as possible and comparable across workloads, obtained from different systems across many years, to allow us to make general observations. We describe the challenges in designing such a setup in detail in this section. We discuss additional aspects and limitations of our design choices in Section 6.

Available evaluation platforms. Experimentally evaluating the effect of the number of partitions on write amplification is challenging, because off-the-shelf drives do not have a mechanism for externally controlling the number of partitions used by the FTL. OpenChannel SSDs [45, 53] address this problem by allowing the host to manage page mapping and garbage collection directly. During the course of this research, we were unable to obtain such a physical experimental SSD. Unfortunately, we found that currently available OpenChannel SSD simulators [35] do not provide the same flexibility that would be provided by the hardware. Namely, these simulators are tightly coupled with the Linux LightNVM subsystem [6], even in their "white box" mode of operation.

OpenChannel SSDs organize groups of blocks into parallel units called *LUN*s, and LightNVM treats these LUNs as a RAID-0 array. It abstracts multiple LUNs as a single *line*, which is equivalent to a RAID stripe: Writes are striped across the LUNs in a line, while garbage collection happens for the entire line at the same time. There is no support for independent control of individual LUNs. Furthermore, LightNVM supports only up to four open lines, which effectively limits the

[1]We did not continue the search beyond f_r = 1,000, because this value was encountered in only a few extreme cases.

ALGORITHM 1: Write logical page ℓ to partition P

1: Let p be the physical page that ℓ is currently mapped to
2: Invalidate p
3: Write ℓ to p'—the next physical page in the active block of P—and mark it as valid
4: Remap logical page ℓ to p'
5: **if** the active block of P is full **then**
6: Run Garbage Collection (GC, Algorithm 2) for P

FTL to four partitions (with huge page sizes). Finally, simulating unequal-sized LUNs or partitions requires restructuring the scheduling mechanism of garbage collection and I/O threads; these are currently designed for striped LUNs. Thus, we cannot divide the logical capacity of the device into an arbitrary number of independent partitions, which is the heart of our analysis.

Simulation is a standard technique for addressing such challenges. Indeed, previous studies used open-source or proprietary simulators for evaluating proposed FTL designs. We decided not to use any of the available open-source simulators because of their high runtime overheads: A large portion of their logic is dedicated to simulating queuing, device latency, and other hardware delays [4, 9, 35], which are orthogonal to write amplification. At the same time, the performance implications of the garbage collection process in real systems depend on the device's parallelism and scheduling mechanisms. These are device-specific design choices that are also hidden from the host, and we thus intentionally preclude them from our analysis.

Write-amplification simulator (WA-simulator). We use a simplified SSD simulator that we have constructed for the purpose of this study. To eliminate channel allocation and mapping update schemes from our analysis, we simulate an SSD with one channel and an FTL whose page mapping table fits entirely in memory. For N partitions, our FTL uses N active blocks (one for each partition) and one extra clean block called the *standby block*. We preprocess each trace to classify its pages into N partitions as described above. Thus, our simulations represent an optimal classification of pages to temperatures.

WA-simulator does not handle read requests, as they have no effect on write amplification. Instead, it only simulates out-of-place updates of the logical pages written in the input workload (Algorithm 1) and garbage collection for generating clean blocks (Algorithm 2). I/O requests larger than the page size are handled by splitting the request into pages and processing each individual page in increasing order of their LBAs. The simulation begins with an empty SSD, with one active block allocated to each partition. Logical pages are written to the active block of their partition, invalidating their previous location, if necessary.

We implement greedy garbage collection with adaptive partition resizing, i.e., blocks are reallocated between partitions dynamically whenever the active block in a partition becomes full. As long as there are available clean blocks, a new clean block is allocated to this partition and becomes its active block. If there are no clean blocks in the SSD, the block with the lowest number of valid pages in the SSD is chosen as victim (line 9), erased, and reallocated as this partition's active block (line 19). Valid pages from the victim block are moved to the active block of the victim's original partition (line 15) to maintain the separation between pages of different temperatures. This requirement might cause the active block in the victim's partition to become full. In such scenarios, we allocate the standby block to the victim's partition and complete the garbage collection process (lines 16–18). Finally, we ensure that there is always one clean standby block by selecting and erasing another victim block, which is not allocated to any partition. This continues until an extra clean block becomes available for the standby (lines 23–31). The full details are provided in Algorithms 1 and 2.

ALGORITHM 2: Garbage Collection for partition P_{cur}, triggered when its active block becomes full.

```
7:  If there is a clean block, use it as the new active block and finish
8:  Let c be the current standby block
9:  Select victim block b: the non-active block with fewest valid pages in the SSD
10: Let P(b) be the partition that b currently belongs to
11: if P(b) == P_cur then                          ▷ b remains allocated to P_cur
12:     RewriteBlock(b)
13:     Make b the new active block of P_cur
14: else                                           ▷ reallocate b to partition P_cur
15:     Copy and remap valid pages in b (the logical pages mapped to b) to the active block of P(b)
16:     if the active block of P(b) became full during the copy then
17:         Use the clean block c as the new active block of P(b)
18:         Copy and remap the remaining valid pages from b to the new active block
19:     Erase b and set it as the active block of P_cur
20:     if there is no standby block then      ▷ because c was allocated as the active block of P(b)
21:         EnsureCleanBlock
22: function EnsureCleanBlock
23:     while there is no clean standby block do
24:         Select new victim block b' and its current partition P(b') as above
25:         Copy and remap the valid pages from b' to the active block of P(b') as above
26:         if the active block of P(b') is not full then
27:             Erase b' and set it as the new standby block
28:             return
29:         else
30:             RewriteBlock(b')
31:             Set b' as the new active block of P(b')
32: function RewriteBlock(victim block b)
33:     Load valid pages in b (the logical pages mapped to b) to memory
34:     Erase block b
35:     Write back the valid pages with updated mappings.
```

Compared to a real implementation, WA-simulator gives a lower bound of the write amplification: (a) it uses in-memory, direct-mapped FTL that sits entirely in memory; (b) any block within the entire SSD can be dynamically allocated to any partition with no restrictions; (c) it uses the minimum amount of overhead (active and standby blocks) by buffering valid pages in memory during erasures; and (d) it relies on precalculated page temperatures.

We measure write amplification for the workloads that include at least 100K logical page writes, and that write each of the sectors in their working set at least four times, on average, resulting in 66 workloads in total. For each workload, we vary the block size (128 and 256 pages), number of partitions (2—24), and overprovisioning (7% and 28%).

Simulated device size. Determining the size of the simulated device for each workload deserves special attention. The logical size of the original traced volume can be approximated by the range of LBAs in the workload. Thus, a common approach in evaluations of FTL designs is to simulate an SSD whose size is as close as possible to that of the traced volume, and to initialize it by writing its entire logical address space, either sequentially or in random order. This initialization is often followed by a warm-up phase processing sequences of write requests, for the purpose of distributing invalid pages across the device and filling its physical capacity. This simulation approach is effective for evaluating alternative FTL designs by comparing their write amplification or performance on individual traces.

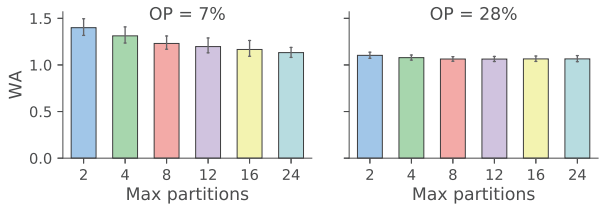


Fig. 2. Average write amplification (WA) of all workloads with different overprovisioning (OP) values. The error bars show 95% confidence intervals.

Unfortunately, it is inapplicable for the purpose of our analysis, whose purpose is workload characterization rather than FTL design. In most of our workloads, the majority of logical addresses are not accessed during the traced period. Thus, determining the device size according to this logical address space would result in an overprovisioning value that is proportional to the logical address space, but unproportionally large with respect to the pages being accessed. More importantly, the different workloads are collected on volumes of different original sizes and access different portions of them, making it impossible to compare their characteristics and to make general observations.

Furthermore, including in our evaluation setup logical addresses that are not accessed during trace collection would result in them being tagged as "frozen," i.e., with the lowest temperature. These logical pages are not written by any I/O request, which means they would have to be preprocessed at the beginning of the simulation. Our FTL, which relies on offline temperature classification, would allocate all frozen pages into a separate partition. This partition would not experience any garbage collection and thus not affect the simulation further. As a result, a simulation of the entire logical address range is equivalent to a simulation that includes only the logical addresses that are included in the trace, but with a much higher overprovisioning value. However, this value would be different for each workload, which is undesirable for our goal of comparing trends across traces.

Instead, we set the logical device capacity to the number of unique pages written in the trace. We determine the number of physical blocks by the logical capacity, block size, and overprovisioning. For example, given a trace that writes to 1,000,000 unique logical addresses, a block size of 128 pages, and an overprovisioning of 7%, we simulate

$$\left\lfloor \left(1 + \left\lfloor \frac{1{,}000{,}000}{128} \right\rfloor\right) \times 1.07 \right\rfloor = 8{,}359 \text{ physical blocks.}$$

This choice of parameters allows us to compare traces collected on devices with a wide range of capacities and eliminate from our evaluation the effect of those portions of the data not accessed during trace collection. These parameters might appear to skew our analysis towards configurations in which the effective overprovisioning value is smaller than what is common in production environments. However, in the following section, we show that the alternative introduces a much larger skew. In addition, we keep in mind that workloads running on SSDs are expected to use larger portions of their logical capacity than those running on HDDs.

3.3 Temperature Ranges and Write Amplification

Figure 2 shows the average write amplification of all the workloads with different numbers of partitions and block size 128 (the figure for block size 256 is very similar). With low overprovisioning (7%), increasing the number of partitions consistently reduces the average write amplification for all workloads, but with diminishing returns. However, when the overprovisioning is sufficiently high (28%), the effect of the number of partitions on the write amplification is negligible.

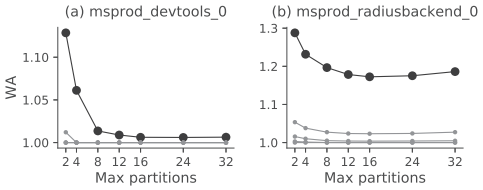


Fig. 3. Write amplification for two traces as a function of max partitions with overprovisioning of 7% and block size of 128. The gray lines show the write amplification for increasing values of overprovisioning, from 28% to 100%.

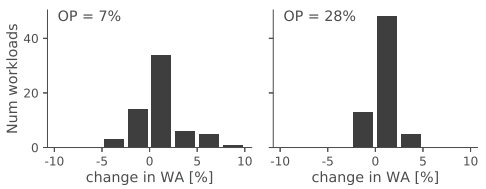


Fig. 4. Histogram of differences in write amplification when requiring $f_r \leq 3$ instead of $f_r \leq 2$.

$\Longrightarrow$ **Finding 2:** increasing the number of partitions reduces write amplification with diminishing returns.

To better understand these aggregate results, we study the write amplification of each separate workload. The dark (top) lines in Figure 3 show the write amplification of two representative workloads with an overprovisioning of 7%. As the number of partitions increases, the write amplification either decreases with diminishing returns (a) or decreases and then begins to increase (b).

The light gray lines in Figure 3 show the write amplification of these workloads with increasing overprovisioning values: 28%, 50%, 75%, and 100%. As the overprovisioning increases beyond 50% and 75% in devtools_0 and radiusbackend_0, respectively, the plots overlap, indicating that the write amplification values converge. For almost 80% of our workloads, the write amplification converged to a value smaller than 1.05 with four partitions and an overprovisioning of 50%. These results confirm that setting the SSD size to the logical address range in the trace would not produce useful results for workload characterization: Write amplification would always be very close to 1.

Figure 4 shows how the write amplification changes when the number of partitions decreases from that required for $f_r \leq 2$ to that required for $f_r \leq 3$. Recall (from Figure1(b)) that $f_r \leq 3$ typically requires 30% fewer partitions than $f_r \leq 2$. Figure 4 shows that this increases the write amplification by less than 5% in almost all cases. In 23% of the workloads, increasing f_r reduces the write amplification, thanks to the overhead savings. In the following section, we address additional characteristics that impact the relationship between f_r and write amplification.

$\Longrightarrow$ **Finding 3:** Due to partitioning overheads, increasing the number of partitions might *increase* write amplification.

4 ACCESS GRANULARITY

I/O workload granularity is determined by the page size of the underlying device and the access granularity of the file system generating the requests. A sector size of 512 B was the default granularity for many years, and is also the access granularity of all the workloads we examined, with one exception: All the requests in workloads from the FIU repository are of 4 KB pages, which is the new standard size for sectors and file-system pages alike.

While hard disk sector sizes have remained stable for many years, SSD page size has increased rapidly, from 2 KB to 16 KB in a few years [1]. This trend is expected to continue as flash feature sizes decrease, to allow more aggressive write optimizations and bit error correction. Nonetheless, file systems continue to operate at 4 KB granularity to allow fine-grained allocation and access. Mapping these 4 KB logical pages to larger physical pages is the responsibility of the FTL, and although many optimizations have been suggested to alleviate the overhead of this mapping, all FTLs employ some form of logical address alignment.

Page:	0	1	2	3
A	0.4	0.3	0.2	0.1
B	0.4	0.1	0.3	0.2

$\Rightarrow$

Page:	0'	1'
A'	0.7	0.3
B'	0.5	0.5

Fig. 5. Example workloads A and B with access frequencies to four logical pages and alignment to two-page boundaries.

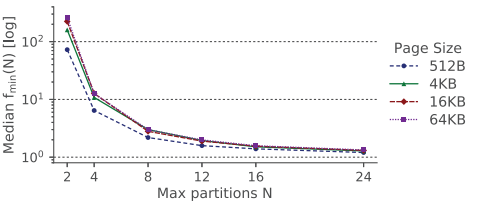


Fig. 6. Median $f_{min}(N)$ with different page sizes when restricting the number of partitions.

Alignment causes accesses to two or more logical pages to refer to the same aligned page, with two major implications. First, page *lifetime*—the time between consecutive writes to the same page—decreases, as pages are updated more frequently than in the original workload. In addition, the access distribution of the workload is modified: Accesses to different pages, with different original access frequencies, are now considered as accesses to the same larger page.

Consider, for example, two workloads that access pages 0–3, as described in Figure 5. Workloads A and B have the same access distribution, although individual pages are ranked differently in each workload: Hot data are clustered in a small logical address space in workload A but not in B. Now consider workloads A' and B', which result from aligning A and B to a 2-page boundary. Although A and B both had the same original skew, alignment increased the skew of A, yet it decreased that of B.

4.1 Characterizing Access Granularity

We study the effect of increasing device access granularity: For each workload and each physical page size from 4 KB to 64 KB, we align the requested logical address to page boundaries and aggregate accesses to the same physical page. We then repeat the experiments in Section 3 for each page size.

Figure 6 shows the median access frequency ratio when the number of partitions is restricted and the page size varies from 512 B to 64 KB (the figure of the maximum looks similar). Interestingly, increasing the page size from 512 B to 4 KB increases the skew considerably, but increasing it further has only a minor effect. We also calculated the number of partitions required for $f_r \leq 2$ with different page sizes. Our results show that for individual workloads, increasing the page sizes beyond 4 KB resulted in a difference of only 1 or 2 partitions. Some workloads behave like example workload A for some page sizes and like B for others.

Our initial analysis indicates that increasing the page size beyond the standard operating system size of 4 KB has little effect on workload skew, and thus on optimizations such as caching and tiering. Below, we analyze the effect of increased page size on write amplification and read costs.

$\Longrightarrow$ **Finding 4:** Increasing the page size beyond 4 KB has little effect on workload skew and thus on typical operating systems optimizations.

4.2 Access Granularity and Write Amplification

We repeated the experiments from Section 3.3 with all requests aligned to page sizes between one and 128 sectors. As a result of this alignment, the *capacity* (in sectors) of the simulated SSD may slightly increase, but its *number of blocks* decreases, and the number of overprovisioned blocks decreases, respectively. Thus, the active blocks (whose number depends on the number of partitions) may occupy a significant portion of the overprovisioned space. We define the *partitioning overhead* as the number of partitions plus one (one active block per partition plus the standby block), divided by the number of overprovisioned blocks.

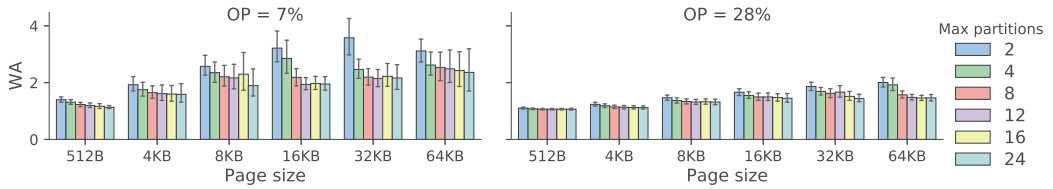


Fig. 7. Average write amplification (WA) of all workloads with different overprovisioning (OP) values. The error bars show 95% confidence intervals.

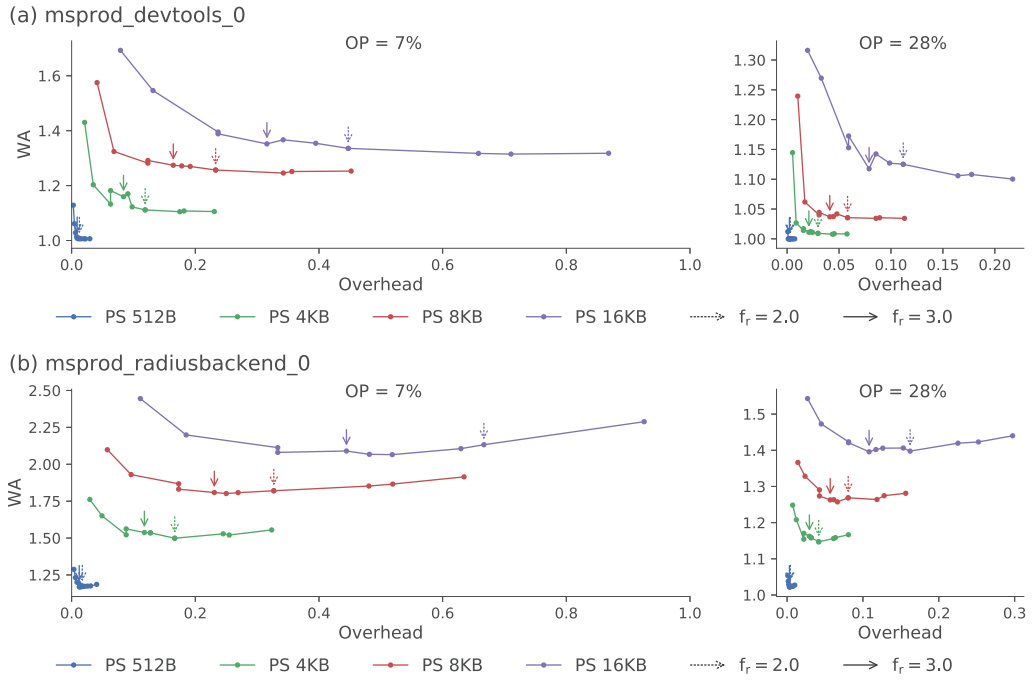


Fig. 8. Write amplification with different page sizes for two representative workloads. Arrows indicate points with $f_r = 2$ (dashed) and $f_r = 3$ (solid). `devtools_0` exhibits diminishing returns (a) while `radiusbackend_0` exhibits a U-shaped curve where the additional partitioning overhead increases write amplification (b).

Figure 7 shows the average write amplification of all the workloads with different numbers of partitions and page sizes (the block size is 128). Our results show that increasing the page size substantially increases write amplification for all overprovisioning values and number of partitions. For example, with 7% overprovisioning, increasing the page size from 4 KB to 8 KB (8 to 16 sectors) increased write amplification for individual traces by as much as 2.17×, 2.51×, and 2.46×, for 2, 8, and 16 partitions, respectively. We observe that increasing the number of partitions above 8 or 12 does not necessarily reduce the write amplification and may even increase it.

⟹ **Finding 5:** Increasing the page size substantially increases write amplification.

To better understand these aggregate results, we study the write amplification of each separate workload with different numbers of partitions and page sizes. Figure 8 shows representative results for two workloads. Increasing the number of partitions (and thus, the partitioning overhead) reduces write amplification. However, for most workloads, curves showing write amplification vs. partitioning overhead resemble those in Figure 8(a). They exhibit a clearly visible "knee" in all page

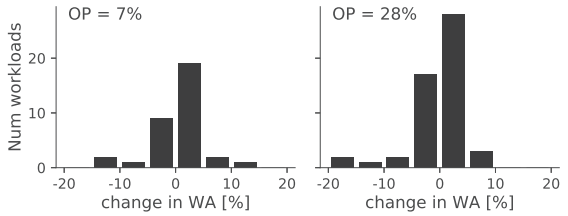


Fig. 9. Histogram of differences in write amplification when requiring $f_r \leq 3$ instead of $f_r \leq 2$ with a page size of 32 sectors. We omit results for workloads whose partitioning overhead is higher than 1, i.e., their number of overprovisioned blocks is smaller than the number of partitions.

sizes, after which adding partitions carries diminishing returns. Some workloads have a U-shaped curve, where further increasing the number of partitions increases write amplification. Figure 8(b) shows one such workload.

We repeated our evaluation with the number of partitions required for $f_r \leq 2.5, 3$, and 5. We found that $f_r \leq 3$ yields the best estimation for the position of the knee. This increase in f_r compared to previous theoretical results is due to the effect of the partitioning overhead, which increases with page size and limits the benefit from fine-grained separation.

$\Longrightarrow$ **Finding 6:** Using additional partitions beyond those required to achieve $f_r \leq 3$ yields diminishing returns.

Figure 9 shows how the write amplification changes when the number of partitions decreases from that required for $f_r \leq 2$ to that required for $f_r \leq 3$, with a page size of 32 sectors (16 KB). While $f_r \leq 3$ typically requires 30% fewer partitions than $f_r \leq 2$, it increases the write amplification by less than 5% in almost all cases. In one-third of the workloads, increasing f_r reduces the write amplification, thanks to the overhead savings.

Our evaluation exposes a tradeoff between the benefits of fine-grained partitioning and the overhead it incurs. Our results indicate that f_r is a useful tool for choosing the best number of partitions for a workload, and that $f_r \leq 3$ correlates with the best practical choice given this tradeoff for all page sizes and overprovisioning values. This value can also be used in SSDs that adjust the number of partitions dynamically [57].

4.3 Access Granularity and Read Amplification

The SSD's page size strongly impacts the workload's read cost: All I/O requests must be aligned to page size boundaries, which may result in reading redundant data from the SSD. This read amplification is determined by three factors: *(1) Small requests increase read amplification:* Every I/O request smaller than the page size will result in reading an entire page. *(2) Misalignment increases read amplification:* Large I/O requests can still generate read amplification if their start and end addresses are not aligned to page boundaries. *(3) Locality reduces read amplification:* Some I/O requests may be eliminated if their data has been read when aligning a previous request.

Consider, for example, a workload (running on a device with a page size of 1) that generates two subsequent requests, to sector 0 and to sector 1. With a page size of two sectors, both requests are aligned to the sector range 0–1. Had this workload been originally run on a device with a page size of 2, the second request would never have reached the SSD—the host would only generate a single 2-sector read. The resulting read amplification would be 1. This effect strongly depends on the locality that "remains" in the workload after it has been processed by the host's cache. We defer the discussion of locality to the next section.

Recall that WA-simulator described in Section 3.2 does not handle read requests. We implement a separate *Read-simulator* for the read amplification and cost calculations presented in this section.

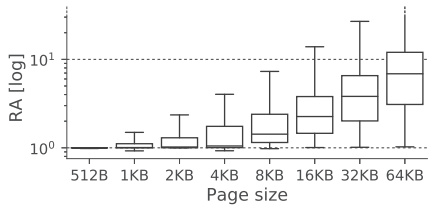


Fig. 10. Read amplification (RA) of all workloads.

Read-simulator calculates the read amplification for a specific workload and page size as follows: The size of data requested by the application is the total number of sectors read by all read requests in the workload. It then aligns all the requests to page-size boundaries. To calculate the number and size of read requests, the simulator runs each workload's trace through a buffer of one page; consecutive requests to the same page are served from the buffer. Finally, it counts the number of sectors read by requests not served by the buffer. The read amplification is the ratio between the number of sectors not served by the buffer and the number of sectors requested by the applications. We include in our analysis workloads with at least 5,000 read requests (116 traces in total).

Our first observation is that, in general, read amplification increases steadily with page size. Figure 10 summarizes the read amplification for all the workloads and page sizes. The maximum read amplification never exceeds half the page size. However, this figure suggests that some workloads are heavily affected by the increase in page size, while others incur only a modest read amplification. We hypothesize that this difference is due to different workload localities, and we verify our hypothesis in Section 5.

$\Longrightarrow$ **Finding 7:** Larger pages always increase read amplification, but the extent of the increase varies across workloads.

4.4 Access Granularity and Flash-level Read Costs

The technological trend of increasing flash page sizes creates a potential bottleneck for applications, such as some key-value stores, that issue small read requests [5]. A possible approach for mitigating this cost is to utilize the flash controller's ability to perform *partial reads*—operations that read an entire page to the flash registers but transfer only the requested byte range to the controller's RAM and on to the host. The Graphene graph processing platform issues its small reads in this manner [37]. 3D-flash chips further support reading only a portion of the page to the flash registers [51, 68].

The benefit from partial reads depends on the relative costs of the constant sensing overhead, tR, and the transfer cost, which is linear in the size of the transferred data. However, this benefit is also sensitive to the locality of the workload. Consider the workload described in the beginning of Section 4.3. Aligning the first request to a page size of two sectors would eliminate the second request, thus saving the additional I/O and sensing overheads. To better understand this effect, we quantify the potential benefit from reading partial pages from flash.

We define the following cost model for an SSD that supports partial reads. We assume that the page size, S_{page}, is a multiple of 512 B sectors, and that the SSD uses an error correction code (ECC) of S_{ECC} bytes per sector. Let tR be the sensing overhead (including ECC decoding time) and T_{serial} be the serial access speed, given in units of μsecs and μsecs/byte, respectively. Thus, the total time required for reading S sectors, $S \leq S_{page}$, is

$$T_S = tR + S \times (512 + S_{ECC}) \times T_{serial}.$$

Table 2. Flash Read Cost Model Parameters

Technology	Page size (sectors)	S_{ECC}	tR	T_{serial}
SLC/MLC	1–8	16	25 μsecs	0.025 μsecs/byte
	16–128	64	45 μsecs	0.025 μsecs/byte
TLC	32	7.5	91μ secs	0.020μ secs/byte

TLC S_{ECC} is 120 bits per KB.

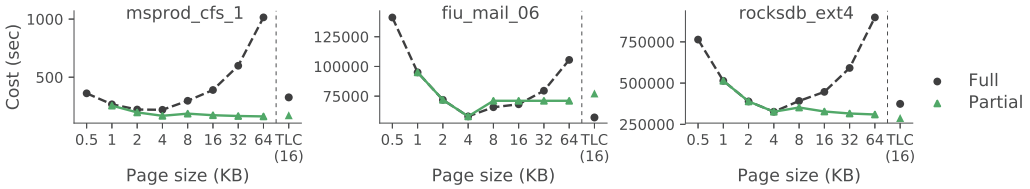


Fig. 11. Total flash read cost for two workloads, with and without partial reads.

This model is intentionally simplified. It ignores I/O and queuing overheads, as well as parallelism in multi-channel SSDs. As a result, the results obtained by this model are insensitive to specific FTL mapping and allocation schemes. Counting all flash operations, regardless of parallel execution, can also be used for estimating the energy consumption of read requests.

We align requests to page boundaries and use the same one-page buffer in Read-simulator as in Section 4.3. Requests that cannot be served from the buffer are split into segments at page boundaries: The first and final segments are issued as partial reads, if possible, and any segments in between are issued as full page reads. T_S gives the time for reading a segment of size S.

We compute the total read cost for each workload based on the parameters in Table 2. The parameters for all page sizes of the SLC/MLC chips were taken from the same vendor's catalog. They capture two trends we observed in publicly available datasheets: (1) tR and S_{ECC} increase considerably for SLC and MLC chips with pages of 8 KB or larger, and (2) T_{serial} does not change dramatically with page size. These trends carry on to TLC and 3D flash chips, which will likely be the technology used for SSDs with pages of 32 KB or larger [36, 50, 51, 68]. The parameters for the TLC chip are taken from a TOSHIBA 3D flash chip [62].

Figure 11 shows the total read cost of three workloads, with and without partial reads. The results for cfs_1 and rocksdb_ext4 are typical. These workloads have very few (cfs_1) or no (rocksdb_ext4) requests smaller than 4 KB (8 sectors); thus, increasing the page size up to 4 KB does not increase its overall read cost. As a matter of fact, increasing the page size reduces the read costs by amortizing the sensing overhead. However, as the page size increases beyond 4 KB, the read cost without partial reads increases rapidly. With partial reads, the cost increases slightly at 8 KB, which is where the model parameter values change, but then decreases slowly: The sensing overhead is amortized for large requests, and the alignment overhead is avoided for small requests, resulting in overall reduction in read cost. The results for the TLC parameter set are similar to those of the SLC/MLC parameters.

$\Longrightarrow$ **Finding 8:** Increasing the page size up to 4 KB reduces the read costs. However, increasing it beyond 4 KB increases read costs.

fiu_mail is less typical, but it demonstrates an interesting phenomenon. All its requests are 4 KB in size. Thus, when the page size is 8 KB or larger, all the requests are performed as partial reads of 4 KB, regardless of the page size, and the total read cost of the workload remains the same. However, many of the requests are a part of short sequences of accesses to consecutive LBAs. When the page size is smaller than the size of the sequence, full reads result in consecutive accesses being served

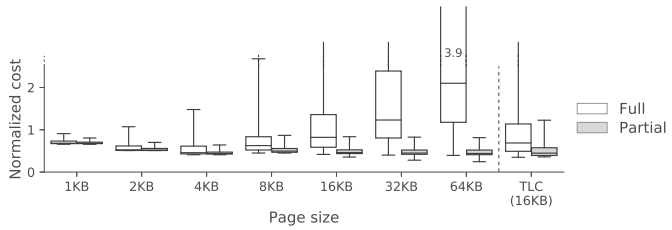


Fig. 12. Read cost for all workloads, with and without partial reads. Boxes show 25th, 50th, and 75th percentiles, and whiskers show minimum and maximum.

from buffer, and the cost of the read is amortized. Thus, for page sizes 8 KB and 16 KB, the total read cost is lower with full reads. The difference between the costs is larger for TLC than for SLC/MLC due to its higher sensing overhead. Although this workload is extreme, we see a similar trend in workloads with many small read requests, such as in the UMass repository.

The results for all the workloads, summarized in Figure 12, are similar to the two presented here. The increase in read cost as the page size increases corresponds to the read amplification results from the previous section. Partial reads reduce the read cost compared to the workloads' original cost (with one-sector pages) by at least 50% for 75% of the workloads.

The benefit from partial reads increases with read amplification. More sophisticated schemes for choosing between full and partial reads can further reduce costs by taking into account queued requests or application-specific "hints." To fully leverage the advantages of partial reads, SSDs should export this capability to the applications via their I/O interface. We did not consider partial writes, as they are not currently supported by off-the-shelf flash controllers [25, 38].

$\Longrightarrow$ **Finding 9:** Partial reads can mitigate the increase in read costs that result from increasing the page size. They should thus be exported via I/O interface of the SSDs.

5 SPATIAL LOCALITY

Locality has always been important in workload analysis. Temporal locality—repeated accesses to the same logical address—has been studied in the form of inter-reference gaps, working set size, and functional lifetime [28, 33, 48, 49]. Spatial locality—accesses to nearby logical addresses—is usually studied in the form of sequential access: The portion of sequential accesses and their "run length." Spatial locality in random accesses is harder to characterize and is usually analyzed by studying the differences between offsets of consecutive requests [17, 28, 33, 48]. This is often referred to as "seek distance," as the motivation for its study is to estimate disk arm movements when serving a given workload.

Although sequential accesses are known to improve SSD throughput, we focus instead on novel characterization of random accesses in SSDs, where spatial locality is critical despite the lack of moving parts. Consider, for example, an SSD that stripes pages across chips in a RAID organization and must update a parity page whenever a data page is written. A possible optimization is to delay parity updates until more pages in the same stripe are updated, to minimize write overhead [29]. To evaluate the efficiency of such optimizations, one might ask, "What is the probability that another page in the same *stripe* will be written in the near future?" We bring additional examples in Section 6.

5.1 Characterizing Spatial Locality

We characterize the *logical spatial locality* (logical locality, for short) of a workload as follows: For a given write to logical page i at time T, we calculate the probability $P_{D,T}$ that a logical page j,

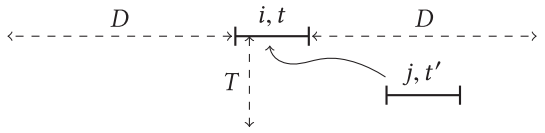
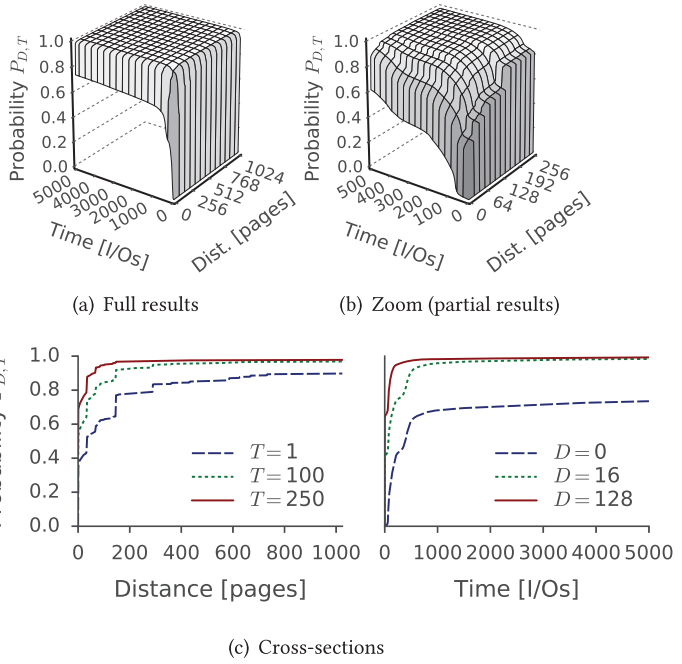


Fig. 13. Page j written at time t' hits page i written at time t.



(a) Full results (b) Zoom (partial results)

(c) Cross-sections

Fig. 14. Logical locality of the "casa" workload (FIU).

$|i - j| \leq D$, will be written at time $t' \leq t + T$. The logical locality is given as a cumulative distribution table, where the value in location (D, T) represents the expected $P_{D,T}$. We currently refer to *virtual time*—a counter incremented on each I/O request. This allows us to compare and analyze results of workloads running on different systems and underlying storage hardware. Analysis of logical locality as a function of absolute time is part of our future work.

For any predetermined D and T, we say that a write request *hits* a previous write request if the time between them is less than T, and we can find a pair of pages, one in each request, whose logical addresses are within D pages from one another (see example in Figure 13). Thus, while traditional measures consider only the last and first logical pages in consecutive requests, logical locality takes into account the entire *address range* of requests within a larger time frame.

We calculate logical locality by defining a "window size" of D_{max} and T_{max} (1,024 and 5,000 in our experiments) and storing a sliding window of the last T_{max} requests in memory while traversing each workload. When a new request hits one of the recorded requests, we update the CDF accordingly. We present the logical locality in a three dimensional graph, where the axes are distance (D) and time (T). Figure 14, discussed below, shows an example. The graph shows how $P_{D,T}$ increases as we increase D and T: A new request hits older requests as we move left on the T axis, or requests at a larger offset as we move right on the D axis.

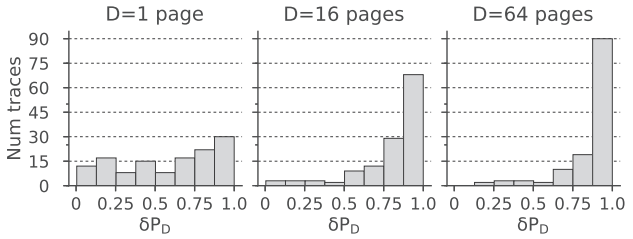


Fig. 15. Histogram of δP_D for $D = 1$, 16, and 64 pages.

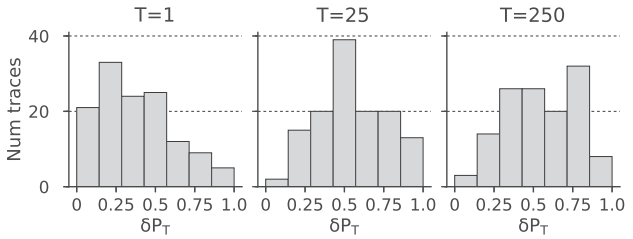


Fig. 16. Histogram of δP_T for $T = 1$, 25, and 250.

Note that in the calculation described above, two or more pages written in the same request will be treated as two requests at the same time, which consequently hit one another: For every one of these pages, i, there is another page ($i - 1$, $i + 1$, or both) written at time $t + 0$. This often causes $P_{D,T}$ to be very close to 1 for all D and T. This result does answer the question of future writes in the vicinity of the current page, but it does not provide meaningful insight about the different workloads.

To prevent large requests from "masking" other forms of locality, we do not consider pages in the same request as hitting one another—we count only hits in previous requests. Although this approach does not capture locality within the same request, it provides a better answer to the question above: We already know the size of the current request, and we can use $P_{D,T}$ to make an educated guess about the future. The cumulative probabilities $P_{D,T}$ can be complemented with the distribution of request sizes for a more complete picture of the locality in the trace.

Figure 14 shows logical locality of the workload of the "casa" server from the FIU repository, which stores home directories. The full results (a) show that when the window is large enough, $P_{D,T}$ approaches 1. A close-up view (b) shows how $P_{D,T}$ increases in each axis, with several noticeable "steps." Comparing the cross-sections in each axis (c) shows that $P_{D,T}$ converges much faster in time (right). This is common in other traces, suggesting that locality does not increase much after a short time. This implies that optimizations such as delayed garbage collection or parity updates are feasible even with only a short wait.

To confirm our hypothesis, we calculate δP_D—the maximal increase in $P_{D,T}$ in a workload for a given distance D. In other words, we measure the potential improvement in $P_{D,T}$ when waiting for the maximal time T. Figure 15 summarizes our results for all workloads. It shows that as we increase the distance from the adjacent page ($D = 1$) to 16 and 64 pages, $P_{D,T}$ can grow more. Conversely, the potential improvement in $P_{D,T}$ remains similar as we increase T (Figure 16). In summary, to capture more hits, it is better to consider a larger distance than to wait for a longer time.

$\Longrightarrow$ **Finding 10:** Locality is better captured by considering a larger distance than by waiting for a longer time.

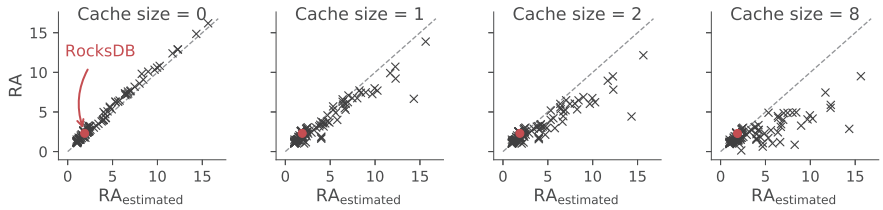


Fig. 17. Simulated and estimated read amplification of all workloads with 16 KB pages. The dashed line depicts $RA = RA_{estimated}$.

5.2 Spatial Locality and Read Amplification

Our initial findings expose a variety of behaviors in the logical locality of the workloads. In this section, we study the effect of spatial locality on the read amplification. We extend the one-page buffer from Section 4.3 to a tiny LRU cache of size 1–16 pages. As discussed in Section 4.3, a host running on a device with the specified page size would not issue certain requests. For example, for a device with a page size of two sectors, a read of sector 1 would not be issued after a previous read of sector 0. The purpose of the tiny LRU cache is to filter such requests even when they are not issued consecutively.

We run the aligned workload through this cache and count the pages that miss in that cache towards the total amount of data read. The size of data requested by the application is the total number of sectors read by all read requests in the workload. We calculate each workload's read amplification separately for each of the cache sizes. Pages in write requests enter the cache but are not counted towards the overall amount of data read. We normalize the read amplification of each workload and page size to the read amplification of this workload with a page size of one sector.

To measure locality, we generate the CDFs for the logical spatial locality of read requests in each workload. We convert this two-dimensional measure into a new scalar metric as follows: We choose a single point (D, T) and use $P_{D,T}$ as our locality measure. In other words, we characterize the locality by the probability that the requested page, i, is "hit" by a request to a nearby page, j, $|j - i| \leq D$, within the subsequent T requests. We set D=page size and T=cache size, which is the degree of locality that can be captured by a given cache. We note that the locality at $D = 1$ can be derived from a workload's reuse distance analysis. However, for larger page sizes, this characterization requires calculating the logical locality CDF.

We have already observed in Section 4.3 that read amplification increases with page size. Figure 10 summarized the read amplification for all the workloads and page sizes with a buffer of one page. In this setting, the maximum read amplification never exceeds half the page size. For a better understanding of the effect of locality, we calculated a naïve estimation of the read amplification of each workload from the distribution of its request sizes. We refer to requests smaller than the page size as *small* and to all other requests as *large*. For a given workload and page size p, let $S = \{s_i | size(s_i) < p\}$ be the set of small requests and $L = \{l_i | size(l_i) \geq p\}$ be the set of large requests. Assuming every small request must be aligned to page-size boundaries, we can estimate the read amplification as

$$RA_{estimated} = \frac{|S| \times p + \sum size(l_i)}{\sum size(s_i) + \sum size(l_i)}.$$

This estimation disregards the effects of locality and of large misaligned requests. Figure 17 shows $RA_{estimated}$ and RA of all the workloads, with a page size of 16 KB, without a cache and with several cache sizes. As in Section 3.1, the results of the RocksDB workload are highlighted and are well within the range of the public workloads. With no cache, the effect of locality is eliminated,

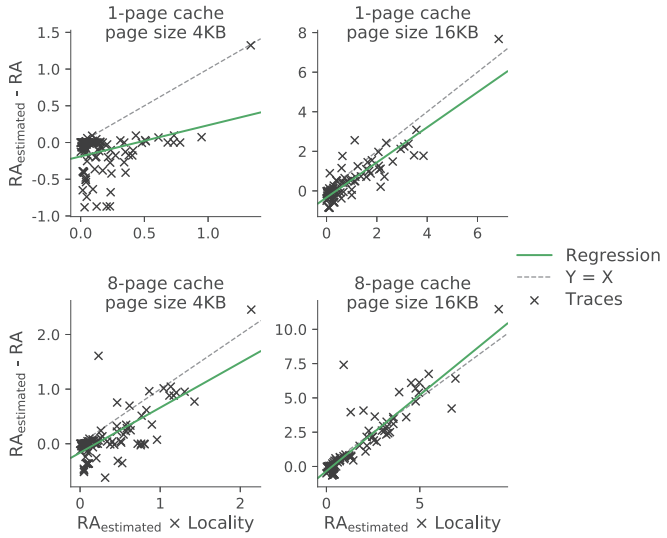


Fig. 18. Read amplification "discount" (X-axis) and difference between simulated and estimated read amplification (Y-axis). The dashed line depicts $RA_{estimated} - RA = RA_{estimated} \times P_{D,T}$. The solid green line depicts the linear regression.

and the difference between $RA_{estimated}$ and RA is due only to misaligned large requests. The cache captures subsequent accesses to sectors that are aligned to the same page, suggesting that locality serves as a "discount," reducing the cost of increasing the page size.

To investigate this correlation, we define the *locality discount* of a workload as $RA_{estimated} \times P_{D,T}$—the expected reduction in read amplification given the probability for a request to hit a page in the cache. We compared the locality discount with the difference between RA and $RA_{estimated}$ for each workload, page size, and cache size. Figure 18 shows the results for 4 KB and 16 KB pages with a cache of one and eight pages, where each dot represents one workload. We observe a linear correlation between $RA_{estimated} - RA$ and the locality discount, demonstrated by the solid green regression line. Moreover, as page size and cache size increase, this correlation approaches our model: $RA_{estimated} - RA = RA_{estimated} \times P_{D,T}$ (the dashed line).

In addition to the expected effect of small and misaligned requests, our results show a strong inverse correlation between the workload's locality and its sensitivity to increased page sizes. Thus, locality characterization can help estimate the portability of applications and file systems to devices with large pages and the effectiveness of caching for improving this portability. It may also be useful for constructing dynamic policies for choosing between regular and partial reads.

$\Longrightarrow$ **Finding 11:** The read amplification of workloads with higher locality are less affected by increasing the page size. A workload's logical locality can indicate the portability of its applications to evolving storage hardware.

6 DISCUSSION AND RELATED WORK

Studies on SSD workloads. The unique physical properties of SSDs have motivated several targeted workload analyses. Studies of smartphone workloads running on eMMC flash storage correlated their sequentiality, access skew, request sizes, and inter-arrival times with the application's performance and energy consumption [14, 42, 73]. However, the conclusions derived from analyzing these workloads are highly specific to smartphone applications.

Another recent study explicitly addresses SSD-specific characteristics of datacenter workloads [21]. It describes desirable characteristics for optimizing SSD performance, such as large request sizes, number of outstanding I/Os, alignment, uniform lifetime, and locality, measured by cache utilization. The authors generate representative workloads by common combinations of applications and file systems, and analyze them with respect to these characteristics. It would be interesting to see a similar analysis of workloads generated in large-scale datacenter environments.

Workload availability. Ideally, SSD-based characterization such as the one presented in this article would be performed on production traces collected on large-scale SSD-based storage servers. Unfortunately, no such traces are publicly available. One possible explanation is that SSD-based environments are relatively new (that is, compared to hard drives). Another is that the relative overhead of tracing is higher for SSDs than for hard drives, due to their lower latencies. With our focus on large-scale datacenter workloads, we chose to use a set of widely used public traces, albeit somewhat old, on which we can apply our new methodology and provide new insight. We are currently working on obtaining and analyzing SSD-based workloads.

Temperature classification. Our analysis is performed offline—we count all the accesses to each page before assigning pages to partitions. As a result, our observations are decoupled from specific hardware or software designs, but the characterization itself cannot be directly computable online. An online classification mechanism can also reassign pages to a different partition when their temperature changes. Several online mechanisms were proposed in previous work. For example, ROSE [12] and ComboFTL [24] characterize hot and cold pages based on the size of the I/O request they appear in. WDAC [46] and MHF [23] approximate page access counts using Bloom filters and hash functions, respectively, while FADaC [32] counts these accesses directly. A recent work proposes a neural network for estimating page temperatures based on a collection of features [69]. Additional mechanisms migrate pages dynamically between partitions without explicitly calculating their temperature [11, 57].

Multi-stream SSDs [26] allow hosts to specify which stream a page belongs to according to its process, application type, or expected lifetime. This allows the FTL to aggregate pages according to their stream, rather than characterizing them at the device level. The burden of classification is shifted to the host, which has more resources and information than the device. For example, SFS [41] and FStream [47] derive pages' hotness from their file's type and metadata, while PCStream [30] derives it from the host's program context.

Offline analysis of a workload's update ratio can be combined with existing classification schemes to help identify the best target number of temperatures. We are also exploring ways to extend our analysis to an online classification technique.

Uses for spatial locality. Our extended view of spatial locality can serve as an additional optimization tool for several key FTL functionalities. Consider, for example, a block-mapping FTL, which groups together pages with consecutive logical addresses. Pages are first written to a log and are periodically garbage collected and grouped according to their logical addresses [12]. In this context, the expected probability of writing to the same logical *block* can help optimize garbage collection.

Page-mapping FTLs are too large to fit entirely in the SSD's DRAM. Thus, while the entire map is stored on the flash media, a portion of it is cached for efficient access. The map is organized according to the logical page addresses, and each page of the map stores the physical locations of consecutive logical addresses [18]. Logical locality can help determine the value of cached map pages—it can indicate the probability that the logical addresses whose mapping is stored on a cached page will be accessed in the near future.

Offline analysis of logical locality can further help determine the benefit of mechanisms for fine access granularity. For example, the new 3D-Xpoint technology of Intel's Optane is available in

two modes—byte addressable DIMM or high-end SSDs, with the DIMMs currently available for considerably smaller capacities. The cost of large-capacity DIMMs is expected to be higher than that of SSDs by an order of magnitude. To better understand the tradeoff between these two options, a workload's logical locality can indicate the performance benefit of fine-grained accesses to Optane memory compared to caching of entire pages fetched from Optane SSD. A recent study of Optane performance indicates sensitivity to "crowded accesses" [63], which might also be identified by logical locality. Similar analysis of logical locality and access granularity can also be used for choosing between regular and huge pages for memory mapping [2] or for zone placement and staging in SMR (*shingled magnetic recording*) [22] or IMR (*interlaced magnetic recording*) [20].

Limitations and validity. Our analysis uses traces of workloads that were collected on HDD-based servers, some of which are over a decade old. These traces consist of the largest publicly available collection of datacenter and production environments. Thus, despite their "age," they are still being used for design and evaluation in recently published research in the context of new flash technologies [71] and firmware [72], cache modeling [59], and emerging storage technologies, such as interlaced magnetic recording [61] and STT-MRAM [19].

Several previous studies identified the limitations of applying simulation-based results to real SSDs, and even to hardware evaluation platforms [21, 35, 39, 52]. Factors such as buffering, device-level queuing, and parallelism, and even flash-level errors and programming constraints, can strongly affect write amplification and additional performance metrics. In our characterization analysis, we intentionally exclude all timing-related considerations and simulate only the garbage collection process of an idealized FTL with offline temperature characterization (Section 3.2). Thus, our write amplification results cannot be obtained on real devices or even on realistic simulators. At the same time, they are in line with previous analytical studies of the benefit of separating hot and cold data that were done using some of the same workloads [15, 66].

Our choice to use virtual time (Section 5.1) follows its use in workload analysis in the context of cache optimization and modeling [59, 74] for similar reasons. In the context of garbage collection efficiency, it removes the effect of spikes in the write load, which are addressed in real systems by maintaining a pool of clean blocks, as well as background and preemptive garbage collection. In the context of logical locality, it excludes the effect of the application's I/O rate and the device's internal flush mechanisms. Here, too, our results do not accurately capture the performance characteristics of specific designs or systems. Rather, their goal is to demonstrate the importance of our characterization and to motivate the community to perform similar analysis in the context of new designs and their evaluations.

7 CONCLUSIONS

We believe that device-specific analysis is inevitable—what makes an analysis interesting are the implications of its findings for system design. Thus, workload analyses must evolve with the systems they target and continuously consider additional trace attributes, emerging technologies, and hybrid designs.

We present a first attempt at characterizing I/O workloads with SSDs in mind. Our initial results expose nontrivial insights when examining new characteristics. Some implications of our findings for SSD design and optimization are straightforward, while others are yet to be determined. Additional analysis is clearly due, and we are continuing to refine our metrics and search for additional correlations and insights.

This study is a first step in correlating recently suggested SSD-based workload characteristics with system-level performance metrics. Our results confirm the relevance of these characteristics to SSD design and performance and demonstrate the benefit of exploring new, device-specific

characteristics. We hope this will motivate the research community to explore new workload characteristics in future studies.

REFERENCES

[1] Intel. [n.d.]. Intel 64M20C Client Compute NAND Flash Memory. Retrieved on September 2020.

[2] Neha Agarwal and Thomas F. Wenisch. 2017. Thermostat: Application-transparent page management for two-tiered main memory. In *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'17)*.

[3] Nitin Agrawal, William J. Bolosky, John R. Douceur, and Jacob R. Lorch. 2007. A five-year study of file-system metadata. *ACM Trans. Stor.* 3, 3 (Oct. 2007).

[4] 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 (USENIX ATC'08)*.

[5] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload analysis of a large-scale key-value store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*.

[6] Matias Bjørling, Javier Gonzalez, and Philippe Bonnet. 2017. LightNVM: The Linux open-channel SSD subsystem. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST'17)*.

[7] Luc Bouganim, Björn Jónsson, and Philippe Bonnet. 2009. uFLIP: Understanding flash IO patterns. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR'09)*.

[8] Alan D. Brunelle. 2008. blktrace user guide. Retrieved on September 2020 from https://github.com/efarrer/blktrace/blob/master/doc/blktrace.tex.

[9] John S. Bucy, Jiri Schindler, Steven W. Schlosser, and Gregory R. Ganger. 2008. The DiskSim Simulation Environment Version 4.0 Reference Manual.

[10] Yanpei Chen, Kiran Srinivasan, Garth Goodson, and Randy Katz. 2011. Design implications for enterprise storage systems via multi-dimensional trace analysis. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*.

[11] Mei-Ling Chiang, Paul C. H. Lee, and Ruei-Chuan Chang. 1999. Using data clustering to improve cleaning performance for flash memory. *Softw.: Pract. Exper.* 29, 3 (1999), 267–290.

[12] Mong-Ling Chiao and Da-Wei Chang. 2011. ROSE: A novel flash translation layer for NAND flash memory based on hybrid address translation. *IEEE Trans. Comput.* 60, 6 (2011), 753–766.

[13] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC'10)*.

[14] J. Courville and F. Chen. 2016. Understanding storage I/O behaviors of mobile applications. In *Proceedings of the 32nd Symposium on Mass Storage Systems and Technologies (MSST'16)*.

[15] Peter Desnoyers. 2013. What systems researchers need to know about NAND flash. In *Proceedings of the 9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'13)*.

[16] Peter Desnoyers. 2014. Analytic models of SSD write performance. *ACM Trans. Stor.* 10, 2 (Mar. 2014). DOI: https://doi.org/10.1145/2577384.

[17] Ajay Gulati, Chethan Kumar, and Irfan Ahmad. 2009. Storage workload characterization and consolidation in virtualized environments. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (VPACT'09)*.

[18] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. 2009. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceedings of the ACM International Conference Architecture Support for Programming Languages and Operating Systems (ASPLOS'09)*.

[19] M. Hadizadeh, E. Cheshmikhani, and H. Asadi. 2020. STAIR: High reliable STT-MRAM aware multi-level I/O cache architecture by adaptive ECC allocation. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'20)*.

[20] Mohammad Hossein Hajkazemi, Ajay Narayan Kulkarni, Peter Desnoyers, and Timothy R. Feldman. 2019. Track-based translation layers for interlaced magnetic recording. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'19)*.

[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 12th European Conference on Computer Systems (EuroSys'17)*.

[22] Weiping He and David H. C. Du. 2017. SMaRT: An approach to shingled magnetic recording translation. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST'17)*.

[23] Jen-Wei Hsieh, Tei-Wei Kuo, and Li-Pin Chang. 2006. Efficient identification of hot data for flash memory storage systems. *Trans. Stor.* 2, 1 (Feb. 2006), 22–40.

[24] Soojun Im and Dongkun Shin. 2010. ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer. *J. Syst. Archit.* 56, 12 (Dec. 2010), 641–653.

[25] Jürgen Kaiser, Fabio Margaglia, and André Brinkmann. 2013. Extending SSD lifetime in database applications with page overwrites. In *Proceedings of the 6th International Systems and Storage Conference (SYSTOR'13)*.

[26] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. 2014. The multi-streamed solid-state drive. In *Proceedings of the 6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'14)*.

[27] Anil Kashyap. 2018. Workload characterization for enterprise disk drives. *ACM Trans. Stor.* 14, 2 (Apr. 2018).

[28] S. Kavalanekar, B. Worthington, Qi Zhang, and V. Sharda. 2008. Characterization of storage workload traces from production windows servers. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'08)*.

[29] Jaeho Kim, Jongmin Lee, Jongmoo Choi, Donghee Lee, and Sam H. Noh. 2013. Improving SSD reliability with RAID via elastic striping and anywhere parity. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'13)*.

[30] Taejin Kim, Duwon Hong, Sangwook Shane Hahn, Myoungjun Chun, Sungjin Lee, Jooyoung Hwang, Jongyoul Lee, and Jihong Kim. 2019. Fully automatic stream management for multi-streamed SSDs using program contexts. In *Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST'19)*.

[31] Ricardo Koller and Raju Rangaswami. 2010. I/O deduplication: Utilizing content similarity to improve I/O performance. *ACM Trans. Stor.* 6, 3 (Sept. 2010), 13:1–13:26.

[32] Kevin Kremer and André Brinkmann. 2019. FADaC: A self-adapting data classifier for flash memory. In *Proceedings of the 12th ACM International Conference on Systems and Storage (SYSTOR'19)*.

[33] Andrew W. Leung, Shankar Pasupathy, Garth Goodson, and Ethan L. Miller. 2008. Measurement and analysis of large-scale network file system workloads. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'08)*.

[34] Cheng Li, Philip Shilane, Fred Douglis, Darren Sawyer, and Hyong Shim. 2014. Assert(!Defined(Sequential I/O)). In *Proceedings of the 7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'14)*.

[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 Conference on File and Storage Technologies (FAST'18)*.

[36] Y. Li, S. Lee, K. Oowada, H. Nguyen, Q. Nguyen, N. Mokhlesi, C. Hsu, J. Li, V. Ramachandra, T. Kamei, M. Higashitani, T. Pham, M. Honma, Y. Watanabe, K. Ino, B. Le, B. Woo, K. Htoo, T. Y. Tseng, L. Pham, F. Tsai, K. h. Kim, Y. C. Chen, M. She, J. Yuh, A. Chu, C. Chen, R. Puri, H. S. Lin, Y. F. Chen, W. Mak, J. Huynh, J. Chan, M. Watanabe, D. Yang, G. Shah, P. Souriraj, D. Tadepalli, S. Tenugu, R. Gao, V. Popuri, B. Azarbayjani, R. Madpur, J. Lan, E. Yero, F. Pan, P. Hong, J. Y. Kang, F. Moogat, Y. Fong, R. Cernea, S. Huynh, C. Trinh, M. Mofidi, R. Shrivastava, and K. Quader. 2012. 128Gb 3b/cell NAND flash memory in 19nm technology with 18MB/s write rate and 400Mb/s toggle mode. In *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC'12)*.

[37] Hang Liu and H. Howie Huang. 2017. Graphene: Fine-grained IO management for graph computing. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST'17)*.

[38] Fabio Margaglia and André Brinkmann. 2015. Improving MLC flash performance and endurance with extended P/E cycles. In *Proceedings of the IEEE 31st Symposium on Mass Storage Systems and Technologies (MSST'15)*.

[39] Fabio Margaglia, Gala Yadgar, Eitan Yaakobi, Yue Li, Assaf Schuster, and Andre Brinkmann. 2016. The devil is in the details: Implementing flash page reuse with WOM codes. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16)*.

[40] Dutch T. Meyer and William J. Bolosky. 2011. A study of practical deduplication. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST'11)*.

[41] Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. 2012. SFS: Random write considered harmful in solid state drives. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST'12)*.

[42] Jayashree Mohan, Dhathri Purohith, Matthew Halpern, Vijay Chidambaram, and Vijay Janapa Reddi. 2017. Storage on your SmartPhone uses more energy than you think. In *Proceedings of the 9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'17)*.

[43] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. 2008. Write off-loading: Practical power management for enterprise storage. *ACM Trans. Stor.* 4, 3 (Nov. 2008), 10:1–10:23.

[44] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. 2009. Migrating server storage to SSDs: Analysis of tradeoffs. In *Proceedings of the European Conference on Computer Systems (EuroSys'09)*.

[45] Jian Ouyang, Shiding Lin, Song Jiang, Zhenyu Hou, Yong Wang, and Yuanzheng Wang. 2014. SDF: Software-defined flash for web-scale Internet storage systems. In *Proceedings of the ACM International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS'14)*.

[46] Dongchul Park and David H. C. Du. 2011. Hot data identification for flash-based storage systems using multiple Bloom filters. In *Proceedings of the 27th IEEE Symposium on Mass Storage Systems and Technologies (MSST'11)*.

[47] Eunhee Rho, Kanchan Joshi, Seung-Uk Shin, Nitesh Jagadeesh Shetty, Jooyoung Hwang, Sangyeun Cho, Daniel D. G. Lee, and Jaeheon Jeong. 2018. FStream: Managing flash streams in the file system. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST'18)*.

[48] Alma Riska and Erik Riedel. 2006. Disk drive level workload characterization. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'06)*.

[49] Drew Roselli, Jacob R. Lorch, and Thomas E. Anderson. 2000. A comparison of file system workloads. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'00)*.

[50] Samsung Electronics. 2011. *16Gb F-die NAND Flash Multi-Level-Cell (2bit/cell)* (1.1 ed.). Retrieved on September 2020.

[51] Samsung Electronics. 2011. *Samsung V-NAND Technology*. Retrieved on September 2020 from https://www.samsung.com/semiconductor/global.semi.static/2bit_V-NAND_technology_White_Paper-1.pdf.

[52] Mohit Saxena, Yiying Zhang, Michael M. Swift, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2013. Getting real: Lessons in transitioning research simulations into hardware systems. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST'13)*.

[53] Zhaoyan Shen, Feng Chen, Gala Yadgar, and Zili Shao. 2019. One size never fits all: A flexible storage interface for SSDs. In *Proceedings of the 39th IEEE International Conference on Distributed Computing Systems (ICDCS'19)*.

[54] N. Shibata, K. Kanda, T. Shimizu, J. Nakai, O. Nagao, N. Kobayashi, M. Miakashi, Y. Nagadomi, T. Nakano, T. Kawabe, T. Shibuya, M. Sako, K. Yanagidaira, T. Hashimoto, H. Date, M. Sato, T. Nakagawa, H. Takamoto, J. Musha, T. Minamoto, M. Uda, D. Nakamura, K. Sakurai, T. Yamashita, J. Zhou, R. Tachibana, T. Takagiwa, T. Sugimoto, M. Ogawa, Y. Ochi, K. Kawaguchi, M. Kojima, T. Ogawa, T. Hashiguchi, R. Fukuda, M. Masuda, K. Kawakami, T. Someya, Y. Kajitani, Y. Matsumoto, N. Morozumi, J. Sato, N. Raghunathan, Y. L. Koh, S. Chen, J. Lee, H. Nasu, H. Sugawara, K. Hosono, T. Hisada, T. Kaneko, and H. Nakamura. 2019. A 1.33Tb 4-bit/cell 3D-flash memory on a 96-word-line-layer technology. In *Proceedings of the IEEE International Solid-state Circuits Conference (ISSCC'19)*.

[55] SNIA IOTTA Trace Repository. 2020. YCSB RocksDB SSD Traces. Retrieved from http://iotta.snia.org/traces/28568.

[56] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. 2010. Extending SSD lifetimes with disk-based write caches. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*.

[57] Radu Stoica and Anastasia Ailamaki. 2013. Improving flash write performance by using update frequency. *VLDB Endow.* 6, 9 (July 2013), 733–744.

[58] University of Massachusetts Amherst. 2014. *UMass Trace Repository*. University of Massachusetts Amherst. http://traces.cs.umass.edu/index.php/Storage/Storage.

[59] Carl Waldspurger, Trausti Saemundsson, Irfan Ahmad, and Nohhyun Park. 2017. Cache modeling and optimization using miniature simulations. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'17)*.

[60] Feng Wang, Qin Xin, Bo Hong, Scott Brandt, Ethan Miller, and Darrell Long. 2004. File system workload analysis for large scale scientific computing applications. In *Proceedings of the IEEE Symposium on Mass Storage Systems and Technologies (MSST'04)*.

[61] Fenggang Wu, Bingzhe Li, Baoquan Zhang, Zhichao Cao, Jim Diehl, Hao Wen, and David H. C. Du. 2020. TrackLace: Data management for interlaced magnetic recording. *IEEE Trans. Comput. (early access)* (2020), 1–1. DOI: 10.1109/TC.2020.2988257

[62] Fei Wu, Jiaona Zhou, Shunzhuo Wang, Yajuan Du, Chengmo Yang, and Changsheng Xie. 2018. In *Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference (DAC'18)*.

[63] Kan Wu, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. 2019. Towards an unwritten contract of Intel Optane SSD. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'19)*.

[64] Gala Yadgar and Moshe Gabel. 2016. Avoiding the streetlight effect: I/O workload analysis with SSDs in mind. In *Proceedings of the 8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'16)*.

[65] Gala Yadgar, Roman Shor, Eitan Yaakobi, and Assaf Schuster. 2015. It's not where your data is, it's how it got there. In *Proceedings of the USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'15)*.

[66] Gala Yadgar, Eitan Yaakobi, Fabio Margaglia, Yue Li, Alexander Yucovich, Nachum Bundak, Lior Gilon, Nir Yakovi, Assaf Schuster, and André Brinkmann. 2018. An analysis of flash page reuse with WOM codes. *ACM Trans. Stor.* 14, 1 (Feb. 2018).

[67] Gala Yadgar, Eitan Yaakobi, and Assaf Schuster. 2015. Write once, get 50% free: Saving SSD erase costs using WOM codes. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'15)*.

[68] R. Yamashita, S. Magia, T. Higuchi, K. Yoneya, T. Yamamura, H. Mizukoshi, S. Zaitsu, M. Yamashita, S. Toyama, N. Kamae, J. Lee, S. Chen, J. Tao, W. Mak, X. Zhang, Y. Yu, Y. Utsunomiya, Y. Kato, M. Sakai, M. Matsumoto, H. Chibvongodze, N. Ookuma, H. Yabe, S. Taigor, R. Samineni, T. Kodama, Y. Kamata, Y. Namai, J. Huynh, S. E. Wang, Y. He, T. Pham, V. Saraf, A. Petkar, M. Watanabe, K. Hayashi, P. Swarnkar, H. Miwa, A. Pradhan, S. Dey, D. Dwibedy, T. Xavier, M. Balaga, S. Agarwal, S. Kulkarni, Z. Papasaheb, S. Deora, P. Hong, M. Wei, G. Balakrishnan, T. Ariki,

K. Verma, C. Siau, Y. Dong, C. H. Lu, T. Miwa, and F. Moogat. 2017. 11.1 A 512Gb 3b/cell flash memory on 64-word-line-layer BiCS technology. In *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC'17)*.

[69] Pan Yang, Ni Xue, Yuqi Zhang, Yangxu Zhou, Li Sun, Wenwen Chen, Zhonggang Chen, Wei Xia, Junke Li, and Kihyoun Kwon. 2019. Reducing garbage collection overhead in SSD based on workload prediction. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'19)*.

[70] Yue Yang and Jianwen Zhu. 2014. Analytical modeling of garbage collection algorithms in hotness-aware flash-based solid state drives. In *Proceedings of the 30th Symposium on Mass Storage Systems and Technologies (MSST'14)*.

[71] Chun yi Liu, Jagadish Kotra, Myoungsoo Jung, and Mahmut Kandemir. 2018. PEN: Design and evaluation of partial-erase for 3D NAND-based high density SSDs. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST'18)*.

[72] Jie Zhang, Miryeong Kwon, Michael Swift, and Myoungsoo Jung. 2020. Scalable parallel flash firmware for many-core architectures. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20)*.

[73] D. Zhou, W. Pan, W. Wang, and T. Xie. 2015. I/O characteristics of smartphone applications and their implications for eMMC design. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'15)*.

[74] Yuanyuan Zhou, James Philbin, and Kai Li. 2001. The multi-queue replacement algorithm for second level buffer caches. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'01)*.

Received January 2020; revised July 2020; accepted September 2020