Avoiding the Streetlight Effect: I/O Workload Analysis with SSDs in Mind

Gala Yadgar and Moshe Gabel
Computer Science Department, Technion
{*gala,mgabel*}*@cs.technion.ac.il*

Abstract

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'. Our initial results reveal nontrivial aspects that can significantly influence the design and performance of SSD-based systems.

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.

Analyses of file-system workloads examine the distributions of file size, age, and functional lifetime, correlating them with file extension and directory structure [3, 12, 16]. On the other hand, 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 [6, 7, 10, 15, 19]. 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 [11].

Existing 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 flash translation layer (FTL).

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 [4, 20]. At the same time, FTL design allows 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, 17] 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 is a first attempt to characterize I/O workloads with SSDs in mind. We analyze over one hundred traces from four public repositories [1, 7, 9, 13], with the goal of identifying characteristics relevant to SSD design and performance. We consider *temperature range* (rather than just the amount of "hot" data they contain), *logical locality* (rather than simple spatial locality or sequentiality), and their sensitivity to increasing SSD page sizes. Our results expose nontrivial characteristics with substantial implications for SSD design and performance.

Repository:	UMass	MSR	MS-Prod	FIU
Number of traces	36	34	43	9
Write requests (M)	0.01–0.4	0.01–58	0.01–9	1–410
Median write size	1–48KB	4–64KB	1–64KB	4KB
Duration	12 hours	1 week	6–24 hours	3 weeks
Volume size (GB)	0.07–650	8–820	9–3200	8–400
Server categories	DB	dev, file web	DB, dev, file mail,web	file, mail web

Table 1: Trace repositories. Additional details are available in the original publications [1, 7, 9, 13].

2 Workloads

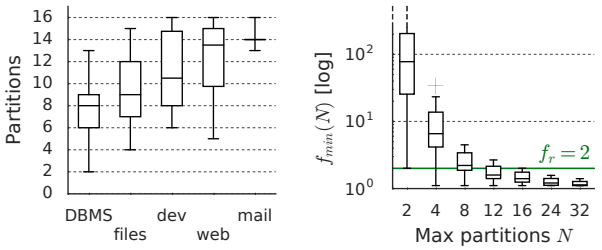
We analyze block-level traces of storage workloads from four online repositories: the University of Massachusetts (SPC traces) [1], Microsoft Research at Cambridge [13], Microsoft production servers [7] and Florida International University [9]. We focus on standard attributes available in all these traces: request arrival time (relative to the beginning of the trace), volume, offset (in bytes relative to the beginning of the volume), request size (in bytes), and I/O operation (either read or write).

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, 122 traces in all. 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.

Motivation. 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 [5]. Such separation is also useful for other optimizations such as wear leveling and page reuse [21]. Consequently, many FTLs separate data into two partitions. Stoica and Ailamaki [18] 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 focus on this analysis and characterize workloads according to the minimum number of partitions they require. We also



(a) Required partitions (b) Lowest ratio

Figure 1: (a) Required number of partitions for $f_r \leq 2$. (b) $f_{min}(N)$ when restricting the number of partitions.

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

Definitions. 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 the 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 p logical partitions requires p active blocks in each plane.

Following the notation of Stoica and Ailamaki [18], 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 [2]. 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. Ensuring $f_r \leq 2$ is sufficient for minimizing garbage collection overhead [18].

Methodology and results. To see whether this optimization is realistic given modern chip organization, we calculate the 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.

Our results show that the minimal number of required partitions is evenly distributed 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. For example, 9 partitions are sufficient for 75% of DBMS workloads, 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

Page:	0	1	2	3	⇒	Page:	0'	1'
A	0.4	0.3	0.2	0.1		A'	0.7	0.3
B	0.4	0.1	0.3	0.2		B'	0.5	0.5

Table 2: Example workloads A and B with access frequencies to four logical pages and alignment to 2-page boundaries.

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 $f_r \leq 3$.

To understand the implications of these findings, 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 in order 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(b) 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 1000[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.

Implications. Previous studies have shown that garbage collection overheads are similar for a ratio of 2 and for uniform accesses, and demonstrated that these overheads increase considerably with higher ratios [18]. Though the exact impact of values as high as 5 or 77 are yet to be determined, it is clear that separating data into only two partitions is far from optimal. Our results show that workloads and categories differ greatly in the number of partitions they require and in the cost of suboptimal partitioning. This lays the groundwork for more detailed analysis of workload skew.

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 512B 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 4KB pages, which is the new standard size for sectors and file-system pages alike.

Motivation. While hard disk sector sizes have remained stable for many years, SSD page size has increased rapidly,

[1]We did not continue the search beyond f_r=1000 because this value was encountered in only a few extreme cases.

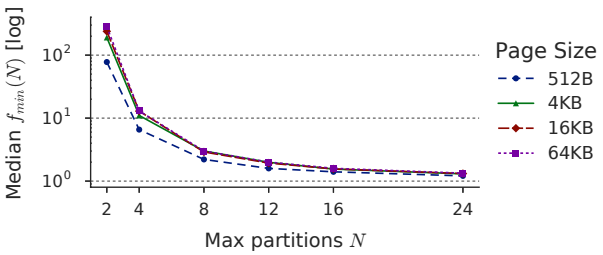


Figure 2: Median $f_{min}(N)$ with different page sizes when restricting the number of partitions.

from 2KB to 16KB in a few years [2]. 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 4KB granularity, to allow fine-grained allocation and access. Mapping these 4KB 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.

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 Table 2. Workloads A and B have the same access distribution and CDF, 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 skew, alignment increased the skew of A, yet it decreased that of B.

Methodology and results. We study the effect of increasing device access granularity: for each workload and each physical page size from 4KB to 64KB, 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 2 shows the median access frequency ratio when the number of partitions is restricted and the page size varies from 512B to 64KB (the figure of the maximum looks similar). Interestingly, increasing the page size from 512B to 4KB 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 (omitted for lack of space) show that for individual workloads, increasing the page sizes beyond 4KB resulted in a difference of only 1 or 2 par-

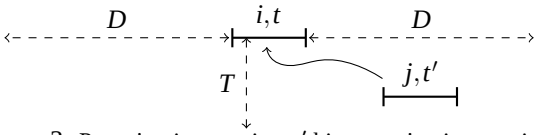


Figure 3: Page j written at time t' hits page i written at time t.

titions. Some workloads behave like example workload A for some page sizes and like B for others.

Implications. The implications of changes in workload skew are well understood when it comes to caching, tiering, partitioning, and other optimizations. Our initial results indicate that increasing the page size beyond the standard operating system size of 4KB has little effect on workload skew, and thus on most optimizations. Analyzing the effect of increased page size on lifetime and on read access patterns is part of our future work.

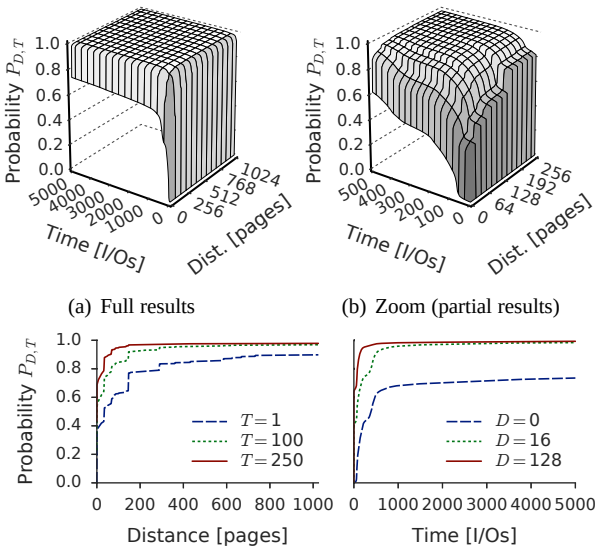
5 Spatial Locality

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 [7, 10, 15, 16]. 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 [6, 7, 10, 15]. 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.

Motivation. 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 [8]. 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?"

As another example, consider 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 address. In this context, the expected probability of writing to the same logical *block* can help optimize garbage collection.

Definitions. 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, $|i-j| \leq D$, will be written at time $t' \leq t+T$. The logical locality is given as a



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

(c) Cross-sections

Figure 4: Logical locality of the 'casa' workload (FIU).

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. 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 3). 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.

Methodology and results. We calculate logical locality by defining a 'window size' of D_{max} and T_{max} (1024 and 5000 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 4, 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.

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

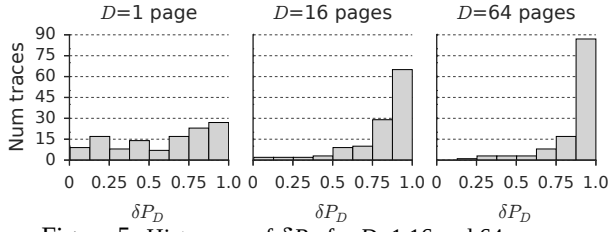


Figure 5: Histogram of δP_D for D=1,16 and 64 pages.

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 4 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, implying 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 5 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 omitted for lack of space). In summary, to capture more hits, it is better to consider a larger distance, than to wait for a longer time.

Implications. Our initial findings expose a variety of behaviors in the logical locality of the workloads. We are currently investigating possible aggregate metrics for representing logical locality, as well as correlations between logical locality and other workload characteristics.

6 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 non-trivial 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.

Acknowledgments

We thank Abdalla Amaria, Khaled Fahom, Baraa Hleihil, Kujan Lauz, Muhammad Musa, Roni Refael, and Ali Tabaja for their help with the traces. We thank Assaf Schuster, Eitan Yaakobi and the anonymous reviewers for their insightful suggestions. This work is partially funded by the Microsoft Economic Center at the Technion.

References

[1] http://traces.cs.umass.edu/.
[2] Intel 64M20C Client Compute NAND Flash Memory. 2013.
[3] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch. A five-year study of file-system metadata. *Trans. Storage*, 3(3), Oct. 2007.
[4] L. Bouganim, B. Jónsson, and P. Bonnet. uFLIP: Understanding flash IO patterns. In *CIDR*, 2009.
[5] P. Desnoyers. Analytic models of SSD write performance. *Trans. Storage*, 10(2):8:1–8:25, Mar. 2014.
[6] A. Gulati, C. Kumar, and I. Ahmad. Storage workload characterization and consolidation in virtualized environments. In *VPACT*, 2009.
[7] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda. Characterization of storage workload traces from production windows servers. In *IISWC*, 2008.
[8] J. Kim, J. Lee, J. Choi, D. Lee, and S. H. Noh. Improving SSD reliability with RAID via elastic striping and anywhere parity. In *DSN*, 2013.
[9] R. Koller and R. Rangaswami. I/O deduplication: Utilizing content similarity to improve I/O performance. *Trans. Storage*, 6(3):13:1–13:26, Sept. 2010.
[10] A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller. Measurement and analysis of large-scale network file system workloads. In *USENIX (ATC)*, 2008.
[11] C. Li, P. Shilane, F. Douglis, D. Sawyer, and H. Shim. Assert(!Defined(Sequential I/O)). In *HotStorage*, 2014.
[12] D. T. Meyer and W. J. Bolosky. A study of practical deduplication. In *FAST*, 2011.
[13] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: Practical power management for enterprise storage. *Trans. Storage*, 4(3):10:1–10:23, Nov. 2008.
[14] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating server storage to SSDs: Analysis of tradeoffs. In *EuroSys*, 2009.
[15] A. Riska and E. Riedel. Disk drive level workload characterization. In *USENIX (ATC)*, 2006.
[16] D. Roselli, J. R. Lorch, and T. E. Anderson. A comparison of file system workloads. In *USENIX (ATC)*, 2000.
[17] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber. Extending SSD lifetimes with disk-based write caches. In *FAST*, 2010.
[18] R. Stoica and A. Ailamaki. Improving flash write performance by using update frequency. *VLDB Endow.*, 6(9):733–744, July 2013.
[19] F. Wang, Q. Xin, B. Hong, S. Brandt, E. Miller, and D. Long. File system workload analysis for large scale scientific computing applications. In *MSST*, 2004.
[20] G. Yadgar, R. Shor, E. Yaakobi, and A. Schuster. It's not where your data is, it's how it got there. In *HotStorage*, 2015.
[21] G. Yadgar, E. Yaakobi, and A. Schuster. Write once, get 50% free: Saving SSD erase costs using WOM codes. In *FAST*, 2015.