Toward a Better Understanding and Evaluation of Tree Structures on Flash SSDs

Diego Didona, Nikolas Ioannou, Radu Stoica
IBM Research Zurich
Rüschlikon, Switzerland
ddi,nio,rst@zurich.ibm.com

Kornilios Kourtis*
Cilium
Zurich, Switzerland
kkourt@kkourt.io

ABSTRACT

Solid-state drives (SSDs) are extensively used to deploy persistent data stores, as they provide low latency random access, high write throughput, high data density, and low cost. Tree-based data structures are widely used to build persistent data stores, and indeed they lie at the backbone of many of the data management systems used in production and research today.

We show that benchmarking a persistent tree-based data structure on an SSD is a complex process, which may easily incur subtle pitfalls that can lead to an inaccurate performance assessment. At a high-level, these pitfalls stem from the interaction of complex software running on complex hardware. On the one hand, tree structures implement internal operations that have non-trivial effects on performance. On the other hand, SSDs employ firmware logic to deal with the idiosyncrasies of the underlying flash memory, which are well known to also lead to complex performance dynamics.

We identify seven benchmarking pitfalls using RocksDB and WiredTiger, two widespread implementations of an LSM-Tree and a B+Tree, respectively. We show that such pitfalls can lead to incorrect measurements of key performance indicators, hinder the reproducibility and the representativeness of the results, and lead to suboptimal deployments in production environments. We also provide guidelines on how to avoid these pitfalls to obtain more reliable performance measurements, and to perform more thorough and fair comparisons among different design points.

PVLDB Reference Format:
Diego Didona, Nikolas Ioannou, Radu Stoica and Kornilios Kourtis. Toward a Better Understanding and Evaluation of Tree Structures on Flash SSDs. PVLDB, 14(3): 364 - 377, 2021.
doi:10.14778/3430915.3430926

1 INTRODUCTION

Flash solid-state drives (SSDs) are widely used to deploy persistent data storage in data centers, both in bare-metal and cloud deployments [25, 27, 29, 37, 63, 72, 74], while also being an integral part of public cloud offerings [2, 12, 13]. While SSDs with novel technologies such as 3D Xpoint [18, 49] offer significant advantages [38, 39, 79], they are not expected to replace flash-based SSDs anytime soon. These newer technologies are not yet as mature from a technology density standpoint, and have a high cost per GB, which significantly hinders their adoption. Hence, flash SSDs are expected to be the storage medium of choice for many applications in the short and medium term [19].

Persistent tree data structures (PTSes) are widely used to index large datasets. PTSes are particularly appealing, because, as opposed to hash-based structures, they allow data to be stored in sorted order, thus enabling efficient implementations of range queries and iterators over the dataset. Examples of PTSes are the log structured merge (LSM) tree [55], used, e.g., by RocksDB [27] and BigTable [10]; the B+Tree [14], used, e.g., by Db2 [35] and WiredTiger [52] (which is the default storage engine in MongoDB [51]); the Bw-tree [40], used, e.g., by Hekaton [24] and Peloton [30, 58]; the B-ϵ tree [8], used, e.g., by Tucana [57] and TokuMX [59].

Over the last decade, due to the reduction in the price of flash memory, PTSes have been increasingly deployed over flash SSDs [50, 69]. Not only do PTSes use flash SSDs as a drop-in replacement for hard disk drives, but new PTS designs are specifically tailored to exploit the capabilities of flash SSDs and their internal architectures [43, 65, 71, 75, 76].

Benchmarking PTSes on flash SSDs. Evaluating accurately and fairly the performance of PTSes on flash SSDs is a task of paramount importance for both industry and research in order to be able to compare alternative designs. Unfortunately, as we show in this paper, evaluating performance is a complex process, which may easily incur subtle pitfalls that can lead to non-reproducible performance results and inaccurate conclusions.

The reason for this complexity is the intertwined effects of the internal dynamics of flash SSDs and of the PTS implementations. On the one hand, flash SSDs employ firmware logic to deal with the idiosyncrasies of the underlying flash memory, which results in highly non-linear performance dynamics [23, 33, 66, 67]. On the other hand, PTSes implement complex operations, (e.g., compactions in LSM-Trees and rebalancing in B+Trees) and update policies (e.g., in a log-structured fashion vs. in-place). These design choices make performance hard to analyze [22]. They also result in widely different access patterns towards the underlying SSDs, thus leading to complex, intertwined performance dynamics.

Contributions. We identify seven benchmarking pitfalls which relate to different aspects of the evaluation of a PTS deployed on a flash SSD, and we provide guidelines on how to avoid these pitfalls. We provide specific suggestions both to academic researchers, to improve the fairness, completeness and reproducibility of their results, and to performance engineers, to help them identify the most efficient and cost-effective PTS for their workload and deployment.

In brief, the pitfalls we describe and their consequences on the PTS benchmarking process are the following:

*Work done while at IBM Research Zurich

Proceedings of the VLDB Endowment, Vol. 14, No. 3 ISSN 2150-8097.
doi:10.14778/3430915.3430926

(1) **Running short tests.** Flash SSDs have time-evolving performance dynamics. Short tests lead to results that are *not* representative of the long-term application performance.
(2) **Ignoring the device write amplification (WA-D).** SSDs perform internal garbage collection that leads to write amplification. Ignoring this metric leads to inaccurate measurements of the I/O efficiency of a PTS.
(3) **Ignoring the internal state of the SSD.** Experimental results may significantly vary depending on the initial state of the drive. This pitfall leads to unfair and non-reproducible performance measurements.
(4) **Ignoring the dataset size.** SSDs exhibit different performance depending on the amount of data stored. This pitfall leads to biased evaluations.
(5) **Ignoring the extra storage capacity a PTS needs to manage data and store additional meta-data.** This pitfall leads to ignoring the storage allocation versus performance trade-off of a PTS, which is methodologically wrong and can result in sub-optimal deployments in production systems.
(6) **Ignoring SSD over-provisioning.** Over-provisioning an SSD (i.e., reserving part of its capacity for internal operations) can improve the performance of a PTS, at the cost of reducing the amount of data that the PTS can index. This pitfall leads to the capacity versus performance trade-off achievable by a PTS being ignored.
(7) **Ignoring the effect of the underlying storage technology on performance.** This pitfall leads to drawing quantitative and qualitative conclusions that do not hold across different SSD types.

We present experimental evidence of these pitfalls using an LSM-Tree and a B+Tree, the two most widely used PTSes, which are also at the basis of several other PTS designs [7, 45, 62]. We consider the LSM-Tree implementation of RocksDB, and the B+Tree implementation of WiredTiger. We use these two systems since they are widely adopted in production systems and research studies.

The storage community has studied the performance of flash SSDs extensively, focusing on understanding, modeling, and benchmarking their performance [23, 36, 66, 68]. Despite this, we have found that many works from the systems and databases communities are not aware of the pitfalls in evaluating PTSes on SSDs. Our work offers a list of factors to consider, and bridges this gap between these communities by illustrating the intertwined effects of the complex dynamics of PTSes and flash SSDs. Ultimately, we hope that our work paves the way for a more rigorous, fair, and reproducible benchmarking of PTSes on flash SSDs.

2 BACKGROUND

Section 2.1 provides an overview of the PTSes that we use to demonstrate the benchmarking pitfalls, namely LSM-Trees and B+Trees. Section 2.2 provides background on the key characteristics of flash-based SSDs that are related to the benchmarking pitfalls we describe. Figure 1 shows the flow of write operations from an application deployed over a PTS to the flash memory of an SSD.

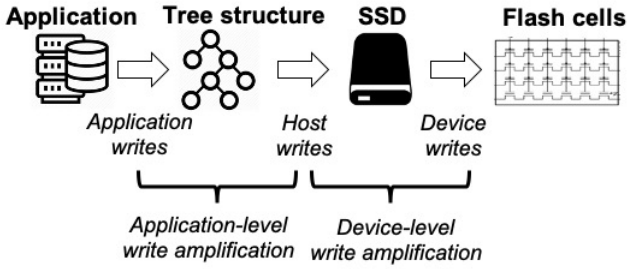


Figure 1: Write amplification in PTSes. The PTS issues additional writes to the SSD to maintain its internal tree structure, which lead to application-level write-amplification. The SSD firmware performs additional writes to overcome the lack of in-place update capabilities of flash memory, which lead to device-level write amplification.

2.1 Persistent Tree Data Structures

We start by introducing the two PTSes used in our experimental study, and two key performance metrics of a PTS.

2.1.1 LSM-Trees. LSM-Trees [55] have two main components: an in-memory component, called *memtable*, and a disk component. Incoming writes are buffered in the memtable. Upon reaching its maximum size, the memtable is flushed onto the disk component, and a new, empty memtable is set up. The disk component is organized in levels $L_1, \cdots, L_N$, with progressively increasing sizes. L_1 stores the memtables that have been flushed. Each level $L_i, i > 1$, organizes data in sorted files that store disjoint key ranges. When L_i is full, part of its data is pushed to L_{i+1} through an operation called *compaction*. Compaction merges data from one level to the next, and discards older versions of duplicate keys, to maintain disjoint key ranges in each level.

2.1.2 B+Trees. B+Trees [14] are composed of internal nodes and leaf nodes. Leaf nodes store key-value data. Internal nodes contain information needed to route the request for a target key to the corresponding leaf. Writing a key-value pair entails writing to the appropriate leaf node, potentially splitting it in two. A key-value write may also involve modifying the internal nodes to update routing information, or to perform re-balancing of the tree.

2.1.3 Application-level write amplification. Block size constraints and internal operations, such as flushing and compactions in LSM-Trees and internal node updating in B+Trees, result in extra writes to persistent storage. These extra writes are detrimental to performance because they reduce the useful SSD bandwidth, resulting in lower overall throughput and higher latencies [3, 39, 45, 64]. We define application-level write amplification (WA-A) as the ratio between the overall data written by the PTS (which considers both application data and internal operations) and the amount of application data written. WA-A is depicted in the left part of Figure 1. WA-A has a minimum value of 1, corresponding to the ideal case where the PTS writes no extra data in addition to the user key-value pairs.

2.1.4 Space amplification. A PTS maintains extra data in addition to the bare key-value pairs of the dataset. LSM-Trees, for example, store multiple values of the same key in different levels. B+Trees store routing information in internal nodes, and may reserve some buffers to implement particular update policies [9]. *Space amplification* captures the storage overhead incurred by a PTS to store such additional data and metadata. Space amplification is defined as the ratio between the amount of bytes that the PTS occupies on the drive and the size of the raw key-value dataset. Space amplification is greater than or equal to 1, where a value of 1 indicates the ideal case where the PTS storage overhead is 0. Space amplification is a function of the design of the PTS and of its parameterization (e.g., number of levels in a LSM-Tree), and is independent of the underlying storage medium.

2.2 Flash SSDs

In this section we describe the internal architecture of flash SSDs, as well as key concepts relevant to their performance dynamics.

2.2.1 Architecture. Flash-based SSDs organize data in *pages*, which are combined in *blocks*. A prominent characteristic of flash memory is that pages do not support in-place updates of the pages. A page needs to be erased before it can be programmed (i.e., set to a new value). The erase operation is performed at the block level, so as to amortize its cost. Flash translation layers (FTLs) [46] hide such idiosyncrasies of the medium. In general, an FTL performs writes out-of-place, in a log-structured fashion, and maintains a mapping between logical and physical addresses. When space runs out, a garbage collection process selects some memory blocks, relocates their valid data, and then erases the blocks.

In the remainder of the paper, for the sake of brevity, we use the term SSD to refer to a flash SSD.

2.2.2 Over-provisioning. Over-provisioning is a key technique used to enable garbage collection and to reduce the amount of data that an SSD relocates internally. Over-provisioning reserves part of the physical storage capacity of the SSD to store blocks that are used in the garbage collection process. SSD manufacturers always implement *hardware* over-provisioning in an SSD, by exposing only part of the physical capacity of the SSD to the upper layers, and reserving the rest as spare blocks used for garbage collection. Users can further implement a *software* over-provisioning, by ensuring that a portion of the logical block address (LBA) space is never written. This is typically achieved by erasing all the blocks of an SSD, and by creating at least one partition that is never written to. The over-provisioning of an SSD is, hence, independent of the PTS that is deployed on it. However, as we shall see, the *implementation* of a PTS may lead to subtle dynamics that are comparable to implementing software over-provisioning. This is achieved by programmatically restricting the LBAs that the PTS writes to.

2.2.3 Device-level write amplification. Garbage collection reduces the performance of the SSD as it leads to internal re-writing of data in an SSD. We define device-level write amplification (WA-D) as the ratio between the amount of data written to flash memory (including the writes induced by garbage collection) and the amount of host data sent to the device. WA-D has a minimum value of 1, which corresponds to an application that performs sequential writes (excluding the cases where the SSD firmware performs compression, which can lead to WA-D values lower than 1). In our setting, a purely random write workloads achieves a WA-D of approximately 3.3.WA-D is depicted in the right part of Figure 1.

3 EXPERIMENTAL SETUP

This section describes our experimental setup, which includes the PTS systems we benchmark, the hardware on which we deploy them, and the workloads we consider.

3.1 Systems

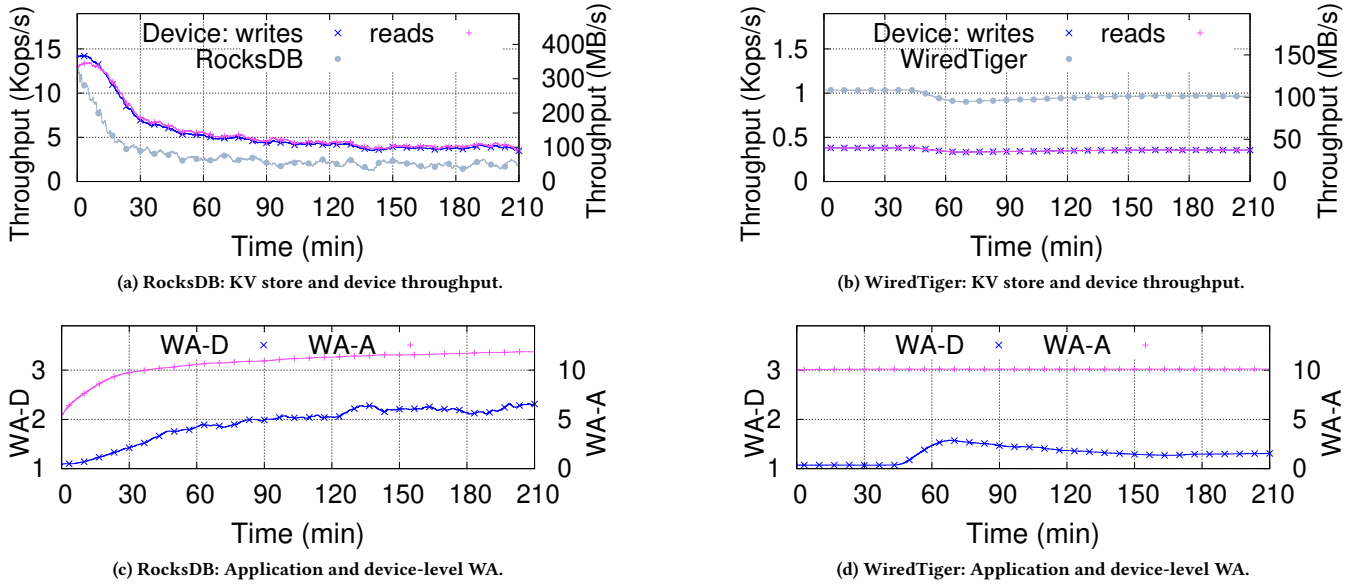
We consider the RocksDB [27] and WiredTiger [52] key-value (KV) stores, which implement, respectively, an LSM-Tree and B+Tree. Both are mature systems widely used on their own and as building blocks of other data management systems. We configure RocksDB and WiredTiger to use small (10 MB) in-memory page caches and direct I/O, so that the dataset *does not fit* into RAM, and both KV and internal operations are served from secondary storage.

3.2 Workloads

We consider different workloads with different read/write mixes (write-only and 50/50) and different sizes of the values of the keys (4000B and 128B). We use a default workload for most of the experiments, and then we consider variations by changing one of its parameters. The default dataset is composed of 50M KV pairs, with 16 byte keys and 4000 byte values. The size of the dataset is ≈200 GB, which represent ≈50% of the capacity of the storage device. Before each experiment we ingest all KV pairs in sequential order. The default workload we consider is a write-only workload, where one user thread updates existing KV pairs according to a uniform random distribution. Despite the fact that read operations also induce nontrivial performance dynamics [4], we primarily consider a write-only workload to focus as much as possible on the performance dynamics that stem from writing data, which is the operation that modifies the internal state of the SSD and of the PTSes. We use a single user thread to avoid the performance dynamics caused by concurrent data accesses. Concurrency control techniques in PTSes are a complex topic that is outside the scope of this paper [40].

3.3 Metrics

To demonstrate the pitfalls, we analyze several application, system and hardware performance metrics.
i) KV store throughput, i.e., the number of operations per second completed by the KV store.
ii) Device throughput, i.e., the amount of data written per second to the drive as observed by the OS. The device throughput is often used to measure the capability of a system to utilize the available I/O resources [39]. We measure device throughput using `iostat`.
iii) User-level write amplification measured as the ratio between the device write throughput and the product of the KV store's throughput times the size of a KV pair. The device write throughput also includes additional metadata writes performed by the filesystem, which we find to be negligible with respect to the writes of the PTS.



(a) RocksDB: KV store and device throughput.

(b) WiredTiger: KV store and device throughput.

(c) RocksDB: Application and device-level WA.

(d) WiredTiger: Application and device-level WA.

Figure 2: Difference between steady state and bursty performance (on a trimmed SSD) in RocksDB (left) and WiredTiger (right). Steady-state performance differs from the initial one (top) because of the change in write amplification (bottom).

iv) Application-level write amplification measured via the SMART attributes of the device. Specifically, we take the ratio of the attributes `nand_bytes_written` and `host_bytes_written` that we obtain using the `nvme intel smart-log-add` utility.
v) Space amplification, which we obtain by taking the ratio of the disk total utilization and the cumulative size of the KV pairs in the dataset. Also this metric factors in the overhead posed by the filesystem, which is negligible with respect to the several GB datasets that we consider.

For the sake of readability of the plots, unless stated otherwise, we report 10-minutes average values when plotting the evolution of a performance metric over time.

3.4 State of the drive

We experiment with two different initial conditions of the internal state of the drive.
• *Trimmed.* All blocks of the device are erased (using the `blkdiscard` utility). Hence, initial writes are stored directly into free blocks and do not incur additional overhead (no WA-D occurs), while updates after the free blocks are exhausted incur internal garbage collection. A trimmed device exhibits performance dynamics close (i.e., modulo the wear of the storage medium) to the ones of a mint factory-fresh device. This setting is representative of bare-metal standalone deployments, where a system administrator can reset the state of the drive before deploying the KV store, and the drive is not shared with other applications.
• *Preconditioned.* The device undergoes a preliminary writing phase so that its internal state resembles the state of a device that has been in use. To this end, we first write the whole drive sequentially, to ensure that all logical addresses have associated data. Then, we issue random writes for an amount of bytes that is twice the size of the disk, so as to trigger garbage collection. In this setting even the first write operation issued by an application towards any page is effectively an over-write operation. This setting is representative of *i*) consolidated deployments, e.g., public clouds, where multiple applications share the same physical device, or *ii*) standalone deployments with an aged filesystem.

These two configurations represent the two endpoints of the spectrum of the initial conditions of the drive, concerning the state of the drive's block. In a real-world deployment the initial conditions of the drive would be somewhere in-between these two endpoints.

3.5 Hardware

We use a machine equipped with an Intel Xeon CPU E5-2630 v4 @ 2.20GHz (20 physical cores, without hyper-threading) and 126 GB of RAM. The machine runs Ubuntu 18.04 with a 4.15 generic Linux kernel[1]. The machine's persistent storage device is a 400 GB Intel p3600 enterprise-class SSD [17]. Unless stated otherwise, we setup a single partition on the drive, which takes the whole available space. We mount an ext4 filesystem configured with the `nodiscard` parameter [15].

4 BENCHMARKING PITFALLS

This section discusses the benchmarking pitfalls in detail. For each pitfall we *i*) first give a brief description; *ii*) then discuss the pitfall in depth, by providing experimental evidence that demonstrates the pitfall itself and its implications on the performance evaluation

[1]Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on ibm.com/trademark.

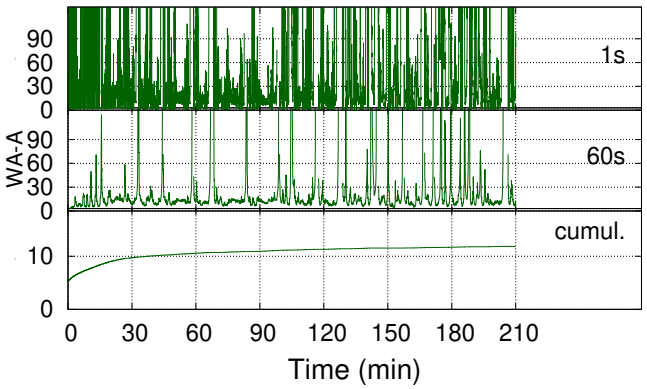


Figure 3: WA-A in RocksDB over time. Computing WA-A with 1-second (top) or 60-second (mid) averages leads to noisy results, due to the bursty nature of the LSM-Tree compaction process. We suggest computing WA-A using cumulative values to obtain more stable results (bottom).

process; and *iii*) finally outline guidelines on how to avoid the pitfall in research studies and production systems.

4.1 Steady-state vs. bursty performance

Pitfall 1: Running short tests. *Because both the PTS and SSD performance vary over time, short-lived tests are unable to capture how the systems will behave under a continuous (non-bursty) workload.*

Discussion. Figure 2 shows the KV store and device throughput (top), and the WA-D and WA-A (bottom) over time, for RocksDB (left) and WiredTiger (right). These results refer to running the two systems on a trimmed SSD. The plots *do not* show the performance of the systems during the initial loading phase. The pictures show that the two PTSes exhibit an initial transient phase during which the dynamics and the performance indicators may widely differ from the ones observed at steady-state. Hence, taking early measurements of a data store's performance may lead to a substantially wrong performance assessment.

Figure 2a shows that measuring the performance of RocksDB in the first 15 minutes would report a throughput of 11-8 KOps/s, which is 3.6-2.6 times higher than the 3 KOps/s that RocksDB is able to sustain at steady-state. In the first 15 minutes, the device throughput of RocksDB is between 375 and 300 MB/s, which is more than 3 times the throughput sustained at steady-state.

Figure 2c sheds light on the causes of such performance degradation. The performance of RocksDB decreases over time for the effect of the increased write amplification, both at the PTS and device level. WA-A increases over time while the levels of the LSM-Tree fills up, and its curve flattens once the layout of the LSM tree has stabilized. WA-D increases over time because of the effect of garbage collection. The initial value of WA-D is close to one, because the SSD is initially trimmed, and keys are ingested in order during the initial data loading, which results in RocksDB issuing sequential writes to the drive. The compaction operations triggered over time, instead, do not result in sequential writes to the SSD flash cells, which ultimately lead to a WA-D slightly higher than 2.

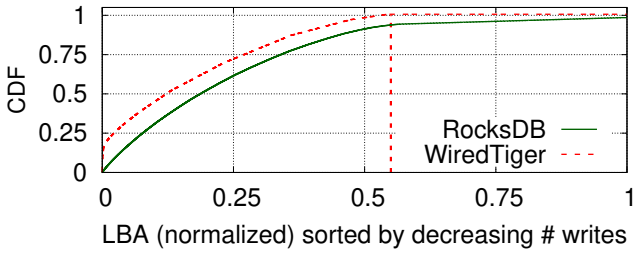


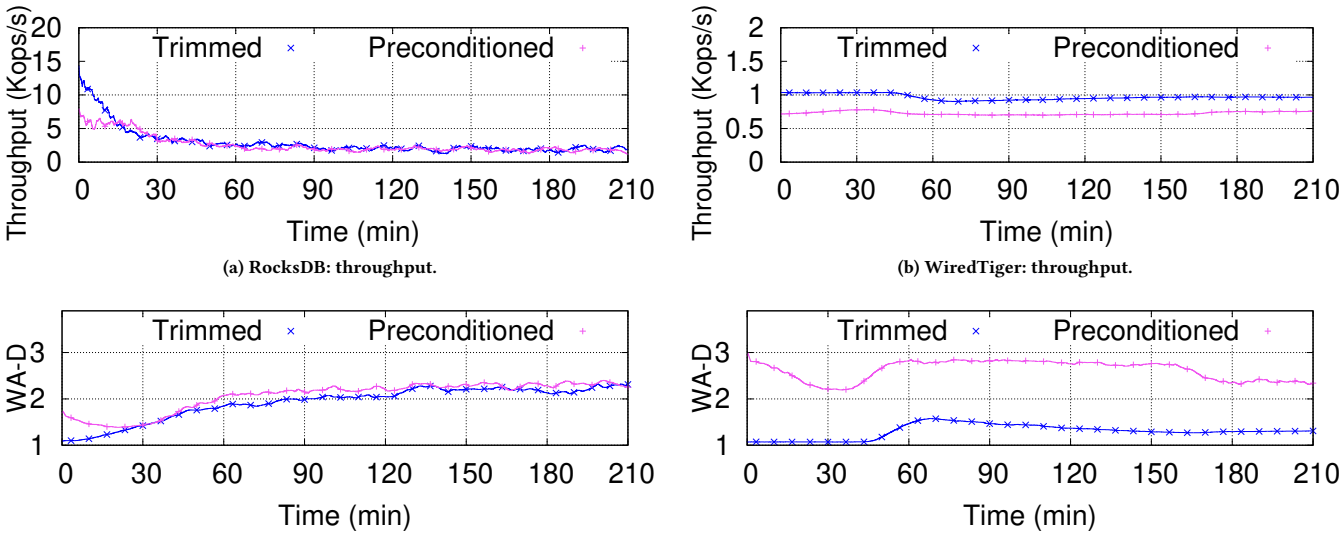
Figure 4: CDF of the LBA write probability in RocksDB and WiredTiger. The vertical dotted line indicates where the CDF corresponding to WiredTiger reaches value 1, and indicates that WiredTiger does not write to ≈ 45% of the LBAs. This has the equivalent effect of increasing the over-provisioning by ≈ 55% of the LBAs.

WiredTiger exhibits performance degradation as well, as shown in Figure 2b. The performance reduction in WiredTiger is lower than in RocksDB for three reasons. *i*) WA-A is stable over time, because updating the B+Tree to accommodate new application writes incurs an amount of extra writes that does not change over time. *ii*) The increase in WA-D is lower than in RocksDB: WA-D reaches at most the value of 1.7, and converges to 1.5. We analyze in more detail the WA-D of WiredTiger in the next section. *iii*) WiredTiger is less sensitive to the performance of the underlying device because of synchronization and CPU overheads [39].

Guidelines. Researchers and practitioners should distinguish between steady-state and bursty performance, and prioritize reporting the former. In light of the results portrayed by Figure 2, we advocate that, to detect steady-state behavior, one should implement a holistic approach that encompasses application-level throughput, WA-A, and WA-D. Techniques such as CUSUM [56] can be used to detect that the values of these metrics do not change significantly for a long enough period of time.

To measure throughput we suggest using an average over long periods of time, e.g., in the order of ten minutes. In fact, it is well known that PTSes are prone to exhibit large performance variations over short period of time [39, 44, 64] –and our experiments on RocksDB confirm these results. Furthermore, we suggest expressing the WA-A at time t as the ratio of the *cumulative* application writes up to time t and the *cumulative* host writes up to time t. This is aimed at avoiding oscillations that would be obtained if measuring the WA-D over small time windows, as we show in Figure 3.

Finally, if WA-D cannot be computed directly from SMART attributes, then we suggest, as a rule of thumb, considering the SSD as having reached steady-state after the cumulative host writes accrue to at least 3 times the capacity of the drive. The first device write ensures that the drive is filled once, so that each block has data associated with it. The second write triggers garbage collection, which overwrites the block layout induced by the initial filling of the dataset. The third write ensures that the garbage collection process approaches steady state, and is also needed to account for the (possibly unknown) amount of hardware over-provisioning of the SSD.



(a) RocksDB: throughput.

(b) WiredTiger: throughput.

(c) RocksDB: device-level WA.

(d) WiredTiger: device-level WA.

Figure 5: Performance achieved over time by RocksDB (left) and WiredTiger (right) depending on the initial conditions of the SSD (trimmed versus preconditioned). The initial conditions of the drive affect throughput (top), potentially even at steady-state, because they affect the SSD garbage collection dynamics and the corresponding WA-D (bottom).

4.2 Analysis of WA-D

Pitfall 2: Not analyzing WA-D. *Overlooking WA-D leads to partial or even inaccurate performance analysis.*

Discussion. Many performance analyses only consider WA-A, which can lead to inaccurate conclusions. We advocate considering WA-D when discussing the performance of a PTS for three main reasons (in addition to being fundamental to identifying steady state, as discussed previously).

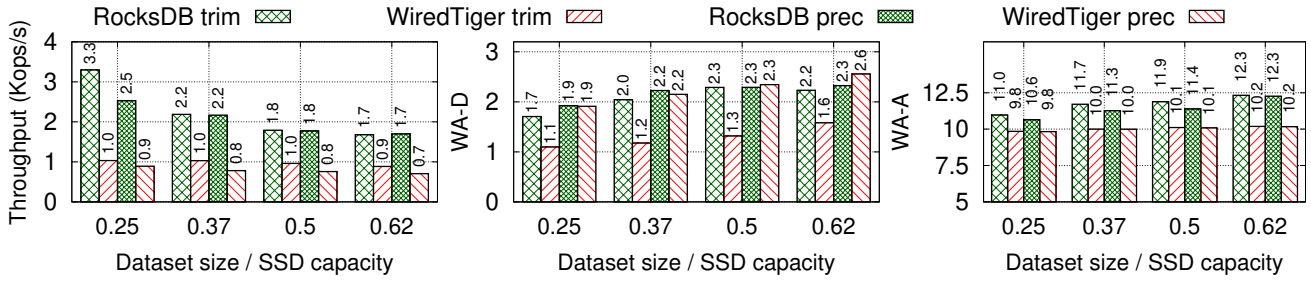
i) WA-D directly affects the throughput of the device, which strongly correlates with the application-level throughput. Analyzing WA-D explains performance variations that cannot be inferred from the analysis of the WA-A alone. Figure 2b shows that WiredTiger exhibits a throughput drop at around the 50th minute mark, despite the fact Figure 2d shows no variations in WA-A. Figure 2d shows that at the 50th minute mark WA-D increases from its initial value of 1, indicating that the SSD garbage collection process has started. This increase in WA-D explains the reduction in SSD throughput, which ultimately determines the drop in the throughput achieved by WiredTiger.

The analysis of WA-D also helps explaining the performance drops in RocksDB, shown in Figure 2a. Throughout the test, the KV throughput drops by a factor of ≈4, from 11 to 2.5 KOps/s. Such a degradation is not consistent with the ≈2 increase of WA-A and the slightly increased CPU overhead caused by internal LSM-Tree operations (most of the CPUs are idle during the test). The doubling of the WA-D explains the device throughput degradation, which contributes to the application-level throughput loss.

ii) WA-D is an essential measure of the I/O efficiency of a PTS. One needs to multiply WA-A by WA-D to obtain the end-to-end write amplification – from application to memory cells– incurred by a PTS *on flash*. This is the write amplification value that should be used to quantify the I/O efficiency of a PTS on flash, and its implications on the lifetime of an SSD. Focusing on WA-A alone, as done in the vast majority of PTS performance studies, may lead to incorrect conclusions. For example, Figure 2 (bottom) shows that RocksDB incurs a steady-state WA-A of 12, which is higher than the WA-A achieved by WiredTiger by a factor of 1.2×. However, the end-to-end write amplification of RocksDB is 25, which is 2.1× higher than WiredTiger's.

iii) WA-D measures the flash-friendliness of a PTS. A low WA-D indicates that a PTS generates a write access that does not incur much garbage collection overhead in the SSD. Hence, measuring WA-D enables the fitness of a PTS for flash SSD deployments to be quantified, and the effectiveness of flash-conscious designs to be verified. For example, LSM-Trees are often regarded as flash-friendly due to their highly sequential writes, while B+Trees are considered less flash-friendly due to their random write access pattern. The direct measurement of WA-D in our tests, however, capsizes this conventional wisdom, showing that RocksDB and WiredTiger achieve a WA-D of around 2.1 and 1.5, respectively. As a reference, a pure random write workload, also targeting 60% of the device capacity, has a WA-D of 1.4 [66].

To understand the causes of this surprising result, we monitor the host write access pattern generated by RocksDB and WiredTiger using `blktrace`. Figure 4 reports the CDF of the page access frequency in the two systems. We observe that in WiredTiger 46% of the pages are not written (0 write accesses). This indicates that WiredTiger only writes to a limited portion of the logical block address (LBA) space, corresponding to the LBAs that initially store the ≈ 200 GB of KV pairs plus some small extra capacity, i.e., ≈ 50% of the SSD's capacity in total. This data access pattern corresponds to having only 50% of the LBAs with associated valid data, since



(a) Throughput. (b) WA-D. (c) WA-A.

Figure 6: Impact of the size of the dataset in RocksDB and WiredTiger, with preconditioned and trimmed device. Larger datasets lead to a lower throughput (a). This is mostly due to an increase in WA-D (b), since the WA-A only increases mildly (c).

the SSD has been initially trimmed. Because the SSD garbage collection only relocates valid data, this corresponds to having ≈ 200 GBs of software over-provisioning (on top of the hardware over-provisioning), which leads to a low WA-D. In the following sections we provide additional insights on the effects of such different LBA access patterns on performance, also taking into account the initial state of the SSD (Section 4.3), the size of the dataset (Section 4.4), and software over-provisioning (Section 4.6).

Guidelines. The analysis of the WA-D should be a standard step in the performance study of any PTS. Such analysis is fundamental to properly measure the flash-friendliness and I/O efficiency of alternative systems. In fact, the analysis of WA-D leads to important insights on the internal dynamics and performance of a PTS, as we also discuss further in the following sections.

4.3 Initial conditions of the drive

Pitfall 3: Overlooking the internal state of the SSD. *Not controlling the initial condition of the SSD may lead to biased and non-reproducible performance results.*

Discussion. Figure 5 shows the performance over time of RocksDB (left) and WiredTiger (right), when running on a trimmed SSD and on a preconditioned one. The top row reports KV throughput, and the bottom one reports WA-D. The plots *do not* show the performance of the systems during the initial loading phase.

The plots show that the initial state of the SSD heavily affects the performance of a PTS and that, crucially, the steady-state performance of a PTS can greatly differ depending on the initial state of the drive. This is surprising, as one would expect the internal state of an SSD to converge to the same configuration, if running the same workload for long enough, and hence to deliver the same steady-state performance regardless of the initial state of the SSD.

This phenomenon is caused by how the LBA access patterns of RocksDB and WiredTiger intertwine with the SSD garbage collection mechanism as a function of the initial state of the drive. As discussed earlier, WiredTiger writes only to roughly 50% of the available LBA space. In a trimmed device, this means that the SSD benefits from an extra software over-provisioning of 50%, which greatly reduces WA-D. In a preconditioned device, on the other hand, all LBAs have associated valid data, which means that the garbage collection process has only the hardware over-provisioning available, and needs to relocate more valid pages when erasing a block, leading to a higher WA-D.

The difference in WA-D, and hence in performance, depending on the initial state of the SSD is much less visible in RocksDB. This is because *i*) the LSM tree utilizes more capacity than a B+Tree and *ii*) RocksDB writes to the whole range of the LBA space. Hence, the initial WA-D for RocksDB depends heavily on the initial device state; however, all LBAs are eventually over-written and thus the WA-D converges to roughly the same value, regardless of the initial state of the drive.

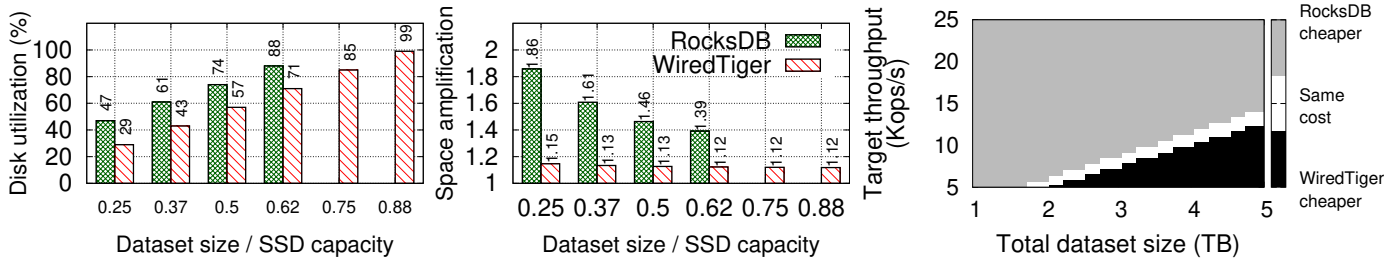
Our results and analysis lead to two important lessons.

i) The I/O efficiency of a PTS on an SSD is not only a matter of the *high level design* of the PTS, but also of its *low-level implementation*. Our experiments show that the benefits on WA-D given by the large sequential writes of the LSM implementation of RocksDB are lower than the benefits of the B+Tree implementation of WiredTiger, despite the more random write access pattern of B+Trees.

ii) Not controlling the initial state of the SSD can potentially jeopardize two key properties of a PTS performance evaluation: fairness and reproducibility. The fairness of a benchmarking process can be compromised by simply running the same workload on two different PTSes back to back. The performance of the second PTS is going to be heavily influenced by the state of the SSD that is determined by the previous test with the other PTS. The lack of fairness can lead a performance engineer to pick a sub-optimal PTS for their workload, or a researcher to report incorrect results.

The reproducibility of a benchmarking process can be compromised because running two independent tests of a PTS with the same workload and on the same hardware may lead to substantially different results. For production engineers this means that the performance study taken on a test machine may differ widely with respect to the performance observed in production. For researchers, it means that it may be impossible to replicate the results published in another work.

Guidelines. We recommend controlling and reporting the initial state of the SSD before *every* test. This state depends on the target deployment of the PTS. For a performance engineer, such a state should be as similar as possible to the one observed in the production environment, which also depends on other applications possibly collocated with the PTS. We suggest researchers precondition the SSD as described in Section 3.4. In this way, they can evaluate the PTS in the most general case possible, thus broadening



(a) Space utilization. (b) Space amplification. (c) Storage cost comparison.

Figure 7: Space amplification in RocksDB and WiredTiger (both for trimmed and preconditioned SSD), and its effects on storage costs (preconditioned SSD). RocksDB runs out of space in the two largest datasets. RocksDB uses more space than WiredTiger to store a dataset of a given size (a), leading to a higher space amplification (b). The heatmap (c) reports the system that requires fewer drives to store a given dataset while achieving a target throughput –hence incurring a lower storage monetary cost.

the scope of their results. To save on the preconditioning time, the SSD can be trimmed, provided that one checks that the steady-state performance of the PTS does not substantially differ from the one observed on a preconditioned drive.

4.4 Dataset size

Pitfall 4: Testing with a single dataset size. *The amount of data stored by the SSD changes its behavior and affects overall performance.*

Discussion. Figure 6 reports the steady-state throughput (left), WA-D (middle), and WA-A (right) of RocksDB and WiredTiger with datasets whose sizes span from 0.25 to 0.88 of the capacity of the SSD (from 100GB to 350GB of KV pairs). We omit the results for RocksDB on the two biggest datasets because it runs out of space. The figure reports results both with a trimmed and with a preconditioned SSD.

Figure 6a shows that the throughput of two systems is affected by the size of the dataset that they manage, although to a different extent and in different ways depending on the initial state of the SSD. By contrasting Figure 6b and Figure 6c we conclude that the performance degradation brought by the larger dataset is primarily due to the idiosyncrasies of the SSD: larger datasets lead to more valid pages in each flash block, which increases the amount of data being relocated upon performing garbage collection, i.e., the WA-D.

Changing the dataset size affects the comparison between the two systems both quantitatively and qualitatively. We also note that the comparison among the two systems is affected by the initial condition of the drive. On a trimmed SSD, RocksDB achieves a throughput that is 3.3× higher than WiredTiger's on the smallest dataset. On the largest dataset, however, this performance improvement shrinks to 1.9×. Moreover, WiredTiger exhibits a lower WA-D across the board, due to the LBA access pattern discussed in the previous section. On a preconditioned SSD, the speedup of RocksDB over WiredTiger still depends on the size of the dataset, but it is lower in absolute values than on a trimmed SSD, ranging from 2.7× on the smallest dataset to 2.57× on the largest one. Moreover, whether RocksDB has a better WA-D than WiredTiger depends on the dataset size. In particular, the WA-D of RocksDB and WiredTiger are approximately equal when storing datasets whose sizes are up to half of the drive's capacity. Past that point, RocksDB's WA-D is sensibly lower than WiredTiger's (2.3 versus 2.6). This happens because the benefits of WiredTiger's LBA access pattern decrease with the size of the dataset (and hence of the range of LBAs storing KV data) and the reduced over-provisioning due to preconditioning.

Guidelines. We suggest that production engineers benchmark alternative PTSes with a dataset of the size that is expected in production, and refrain from testing with scaled-down datasets for the sake of time. We suggest researchers experiment with different dataset sizes. This suggestion has a twofold goal. First, it allows a researcher to study the sensitivity of their PTS design to different device utilization values. Second, it disallows evaluations that are purposely or accidentally biased in favor of one design over another.

4.5 Space amplification

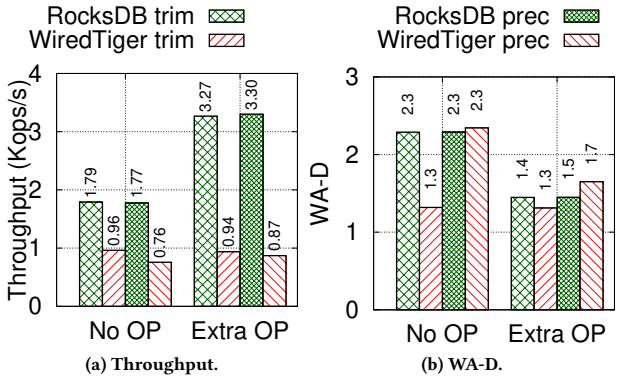
Pitfall 5: Not accounting for space amplification. *The space utilization overhead of a PTS determines its storage requirements and deployment monetary cost.*

Discussion. PTSes frequently trade additional space for improved performance, and understanding their behavior depends on understanding these trade-offs. Figure 7a reports the total disk utilization incurred by RocksDB and WiredTiger depending on the size of the dataset. The disk utilization includes the overhead due to filesystem meta-data. Because RocksDB frequently writes and erases many large files, its disk utilization varies sensibly over time. The value we report is the maximum utilization that RocksDB achieves. Figure 7b reports the space amplification corresponding to the utilization depicted in Figure 7a.

WiredTiger uses an amount of space only slightly higher than the bare space needed to store the dataset, and achieves a space amplification that ranges from 1.15 to 1.12. RocksDB, instead, requires much more additional disk space to store the several levels of its LSM-Tree. Overall, RocksDB achieves an application space amplification ranging between 1.86, with the smallest dataset we consider, to 1.39, with the biggest dataset that it can handle [2].

These results show that space amplification plays a key role in the performance versus storage space trade-off. Such trade-off

[2] The disk utilization in RocksDB depends on internal parameters such as the number and the sizes of the LSM-Tree levels [28]. It is possible to achieve a lower space amplification than the one we report, but with a higher compaction overhead and lower throughput.



(a) Throughput. (b) WA-D.

Figure 8: Impact of extra SSD over-provisioning (OP). Extra OP may improve throughput (a) and WA-D (b), at the cost of reducing the amount of data that the SSD can store.

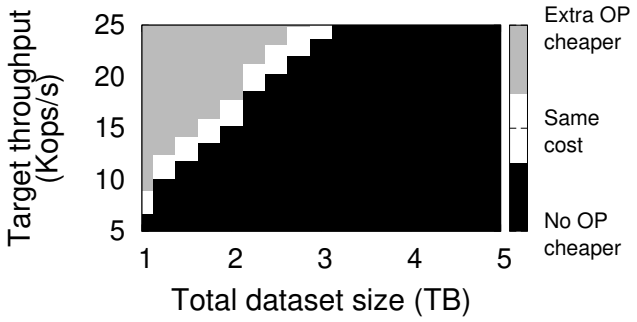


Figure 9: Storage cost comparison using RocksDB with or without extra over-provisioning (OP) on a preconditioned SSD. The heatmap shows the setting that requires fewer drives to store a given dataset while achieving a target throughput –hence incurring a lower storage monetary cost.

affects the total storage cost of a PTS deployment, given an SSD drive model, a target throughput, and total dataset size. In fact, a PTS with a low space amplification may fit the target dataset in a smaller and cheaper drive with respect to another PTS with a higher write amplification, or can index more data given the same capacity, requiring fewer drives to store the whole dataset.

To showcase this last point, we perform a back-of-the-envelope computation to identify which of the two systems require fewer SSDs (and hence incur a lower storage cost) to store a given dataset and at the same time achieve a target throughput. We use the throughput and disk utilization values that we measure for our SSD (see Figure 6a and Figure 7a). For simplicity, we assume one PTS instance per SSD, and that the aggregate throughput of the deployment is the sum of the throughputs of the instances. Figure 7c reports the result of this computation. Despite having a lower per-instance throughput, the higher space efficiency of WiredTiger makes it preferable over RocksDB in scenarios with a large dataset and a relatively low target throughput. This configuration represents an important class of workloads, given that with the ever-increasing amount of data being collected and stored, many applications begin to be storage capacity-bound rather than throughput-bound [11].

Guidelines. The experimental evaluation of a PTS should not focus only on performance, but should include also space amplification. For research works, analyzing space amplification provides additional insights on the performance dynamics and trade-offs of the design choices of a PTS, and allows for a multi-dimensional comparison among designs. For production engineers, analyzing space amplification is key to computing the monetary cost of provisioning the storage for a PTS in production, which is typically more important than maximum performance [53].

As a final remark, we note that this pitfall applies also to PTSes not deployed over an SSD, and hence our considerations apply more broadly to PTSes deployed over any persistent storage medium.

4.6 SSD over-provisioning

Pitfall 6: Overlooking SSD software over-provisioning. *Over-provisioning the SSD may lead to a more favorable capacity versus performance trade-offs.*

Discussion. Figure 8 compares the steady-state throughput (left) and WA-D (right) achieved by RocksDB and WiredTiger in two settings: *i*) the default one in which the whole SSD capacity is assigned to the disk partition accessible by the filesystem underlying the PTS, and *ii*) one in which some SSD space is not made available to the filesystem underlying the PTS, and is instead assigned as extra over-provisioning to the SSD. Specifically, in the second setting we trim the SSD and assign a 300GB partition to the PTS. Hence, the SSD has 100GB of trimmed capacity that is not visible to the PTS. 100GB corresponds to half of the free capacity of the drive once the 200 GB dataset has been loaded. For both settings we consider the case in which the PTS partition remains clean after the initial trimming, and the case in which it is preconditioned.

Extra over-provisioning improves the performance of RocksDB by a factor of 1.83×. This substantial improvement is caused by a drastic reduction of WA-D, that drops from 2.3 to 1.4, and it applies to RocksDB regardless of the initial state of the PTS partition, for the reason discussed in Section 4.3. The impact of extra over-provisioning is much less evident in WiredTiger. In the trimmed device case, the extra over-provisioning has no effect on WiredTiger. This happens because WiredTiger writes only to a certain range of the LBA space (see Figure 4). Hence, all other trimmed blocks act

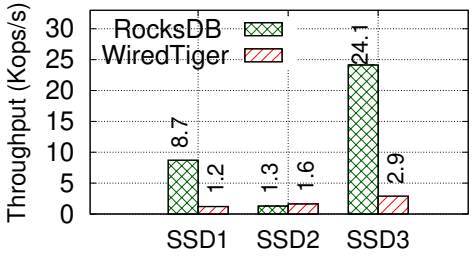
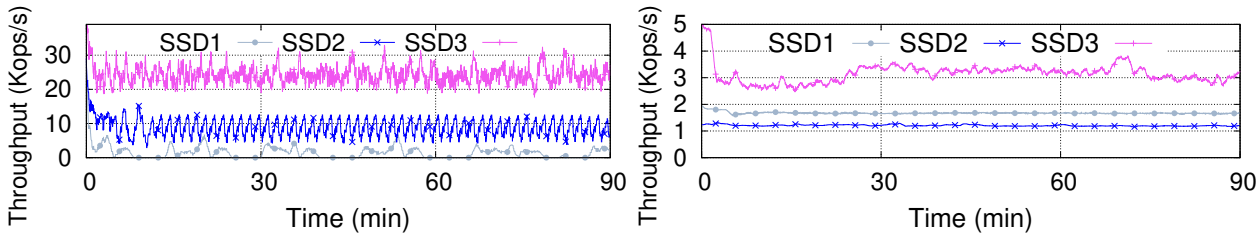


Figure 10: Impact of the SSD type on throughput. The type of SSD significantly affects the absolute performance achieved by RocksDB and WiredTiger, and can even determine which of them achieves the higher throughput.



(a) RocksDB. (b) WiredTiger.

Figure 11: Throughput (1 minute average) of RocksDB (left) and WiredTiger (right) over time using SSDs based on different technologies. The type of SSD influences heavily the throughput variability of RocksDB, and much less that of WiredTiger.

as over-provisioning, regardless of whether they belong to the PTS partition or the extra over-provisioning one. On the preconditioned device, instead, all blocks of the PTS partition have data associated with them, so the only software over-provisioning is given by the trimmed partition. This extra over-provisioning reduces WA-D from 1.7 to 1.3, yielding a throughput improvement of 1.14×.

Allocating extra over-provisioning can be an effective technique to reduce the number of PTS instances needed in a deployment (and hence reduce its storage cost), because it increases the throughput of the PTS without requiring additional hardware resources. However, extra over-provisioning also reduces the amount of data that a single drive can store, which potentially increases the amount of drives needed to store a dataset and the storage deployment cost. To assess in which use cases using extra over-provisioning is the most cost-effective choice, we perform a back-of-the-envelope computation of the number of drives needed to provision a RocksDB deployment given a dataset size and a target throughput value. We perform this study on RocksDB because it benefits the most from extra over-provisioning. We use the same simplifying assumptions that we made for the previous similar analysis. Figure 9 reports the results of our computation. As expected, extra over-provisioning is beneficial for use cases that require high throughput for relatively small datasets. For larger datasets with relatively low performance requirements, it is more convenient to allocate as much of the drive's capacity as possible to RocksDB.

Guidelines. It is known that PTSes have many tuning knobs that affect performance [5, 22, 41]. We suggest considering SSD over-provisioning as an additional, yet first class, tuning knob of a PTS. SSD extra over-provisioning trades capacity for performance, and can reduce the storage cost of a PTS deployment in some use cases.

4.7 Storage technology

Pitfall 7: Testing on a single SSD type. *The performance of a PTS heavily depends on the type of SSD used. This makes it hard to extrapolate the expected performance when running on other drives and also to reach conclusive results when comparing two systems.*

Discussion. We exemplify this pitfall through an experiment where we keep the workload and the RocksDB and WiredTiger configurations constant and only swap the underlying storage device. We use three SSDs: an Intel p3600 [17] flash SSD, i.e., the drive used for the experiments discussed in previous sections; an Intel 660 [16] flash SSD; and an Intel Optane [18]. We refer to these SSDs as SSD1, SSD2 and SSD3, respectively, in the following discussion. SSD3 is a high-end SSD, based on the newer 3DXP technology that achieves higher performance than flash SSDs. We use SSD3 as an upper bound on performance that a PTS can achieve on a flash block device.

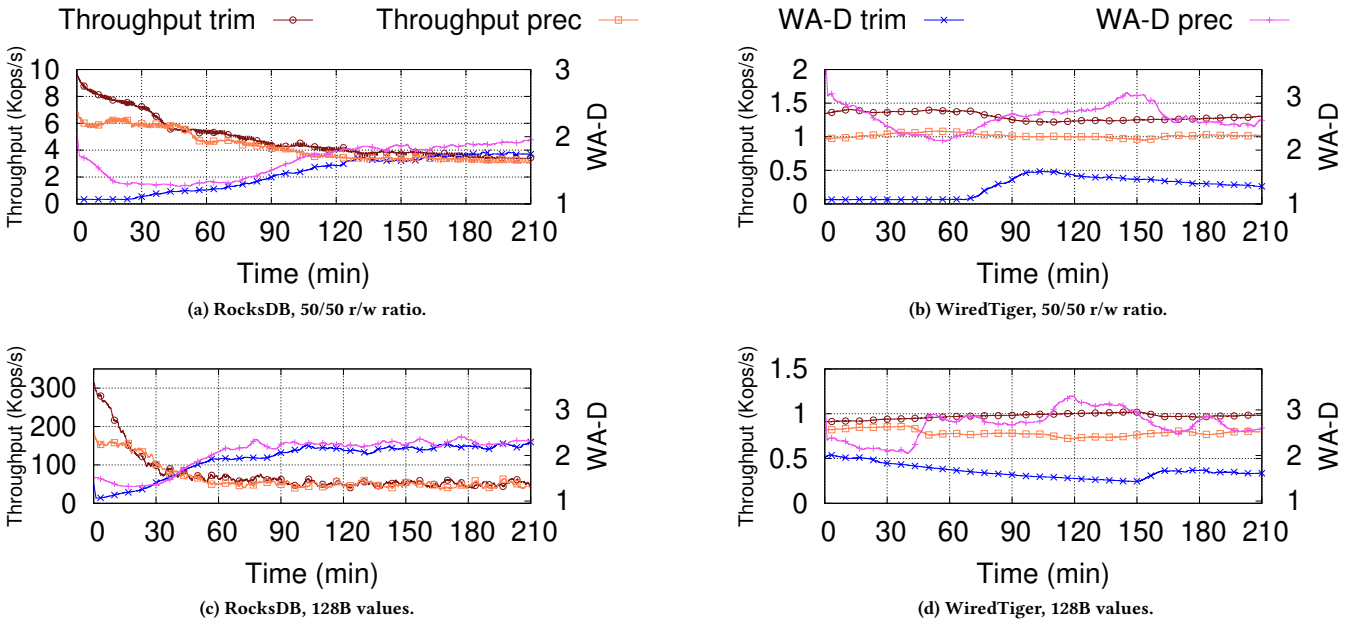
To try and isolate the performance impact due to the SSD technology (i.e., architecture and underlying storage medium performance) in the assessment of a PTS, we eliminate, as much as possible, the other sources of performance variability that we have discussed so far. To this end, we run a workload with a dataset that is 10× smaller than the default one, and we trim the flash SSDs. In this way, the effect of garbage collection in the flash SSDs is minimized, resulting in a WA-D very close to one.

Figure 10 shows the steady-state throughput of RocksDB and WiredTiger when deployed on the three SSDs. As can be depicted from the plot, the performance impact changing the underlying drive varies drastically across the two systems. Explaining these performance variations requires gaining a deeper understanding of the low-level design of the SSDs, which is not always achievable.

RocksDB achieves the highest throughput on SSD3, and a higher throughput on SSD1 than on SSD2. This performance trend is mostly determined by the write latencies of the SSDs, which are the lowest in SSD3, and lower in SSD2 than in SSD1. Also WiredTiger achieves the highest throughput on SSD3 but, surprisingly, it obtains a higher throughput on SSD2 than on SSD1. We argue that the reason for this performance dynamic lies in the fact that SSD2 has a larger internal cache than SSD1. Because WiredTiger performs small writes, uniformly distributed over time, the cache of SSD2 is able to absorb them with very low latency, and destages them in the background. The larger cache of SSD2 does not yield the same beneficial effects to RocksDB because RocksDB performs large bursty writes, which overwhelm the cache, leading to longer write latencies and, hence, lower throughput.

These dynamics also lead to the surprising result that either of the two systems we consider can achieve a higher throughput than the other, just by changing the SSD on which they are deployed.

We also observe very different performance variations for the two systems, when deployed on different SSDs. On the one hand, the best and worst throughputs achieved by RocksDB vary by a factor of almost 20× (SSD2 versus SSD3). On the other hand, they vary only by a factor of 2.4 for WiredTiger. These results indicate that the performance comparison across PTS design points, and the corresponding conclusions that are drawn, are strongly



(a) RocksDB, 50/50 r/w ratio.

(b) WiredTiger, 50/50 r/w ratio.

(c) RocksDB, 128B values.

(d) WiredTiger, 128B values.

Figure 12: Performance of RocksDB (left) and WiredTiger (right) over time, with a preconditioned and trimmed device. The top row refers to a workload with small (128 bytes) values. The bottom row refers to a workload with a 50/50 read/write ratio. The pitfalls we describe apply to a broad set of workloads as long as they have a sustained write component (here we represent pitfalls #1, #2 and #3).

dependent on the SSD employed in the benchmarks, and hence hard to generalize.

The type of SSD also dramatically affects the performance predictability of the two systems. Figure 11 reports the throughput of RocksDB (left) and WiredTiger (right) when deployed over the three SSDs. To highlight the performance variability, we average the throughput over a 1 minute interval (as opposed to the 10 minutes used in previous plots).

The throughput of RocksDB varies widely over time, and the extent of the variability depends on the type of SSD. When using SSD1, RocksDB suffers from throughput swings of 100%. When using SSD2, RocksDB has long periods of time where no application-level writes are executed at all. This happens because the large writes performed by RocksDB overwhelm the cache of SSD2 and lead to long stall times due to internal data destaging. On SSD3, the relative RocksDB throughput variability decreases to 30%. WiredTiger is less prone to performance variability, and exhibits a more steady and predictable performance irrespective of the storage technology.

Guidelines. We recommend testing a PTS on multiple SSD classes, preferably using devices from multiple vendors, and using multiple flash storage technologies. This allows researchers to draw broader and more significant conclusions about the performance of a the design of a PTS, and to assess the "intrinsic" validity of such design, without tying it to specific characteristics of the medium on which the design has been tested. For a performance engineer, testing with multiple drives is essential to identify which one yields the best combination of storage capacity, performance and cost depending on the target workload.

4.8 Additional workloads

In this section, we analyze two additional workloads to show that our pitfalls also apply to workloads with different read/write mixes and value sizes. The first workload generates a 50/50 read/write operation mix (instead of write-only accesses). The second workload generates values whose size is 128B (instead of 4000B). In the second workload, to keep the amount of data stored indexed by the PTS constant to the one used by the previous experiments, we increase accordingly the number of keys. We run these workloads using both a preconditioned and a trimmed drive. For space constraints, we focus on the first three pitfalls. Figure 12 reports the throughput and WA-D over time achieved by RocksDB (left) and WiredTiger (right) for the read/write workload (top) and for the 128B values workload (bottom).

Pitfall 1. The steady-state performance can be very different from the one observed at the beginning of the test. In the mixed read/write workload, throughput takes longer to stabilize than what we have seen in the previous write-only experiments, as writes are less frequent. This is visible by comparing Figure 12a with Figure 2a and Figure 2c, and Figure 12b with Figure 2b and Figure 2d.

Pitfall 2. WA-D is important to explain performance dynamics. We note that the WA-D of WiredTiger in the trimmed case with 128B values (Figure 12d) is different from the one observed for the workload with 4000B values (Figure 2d). With 4000B values the WA-D starts at a value close to 1, whereas with 128B values its starting point is closer to 2. This happens because the initial data loading leads to different SSD states depending on the size of the KV pairs. With 4000B values, one KV pair can fill one filesystem

page, which is written only once to the SSD. With small values, the same page needs to be written multiple times to pack more KV pairs, which leads to higher fragmentation at the SSD level. Such a phenomenon is not visible in RocksDB, because it writes large chunks of data regardless of the size of the individual KV pairs.

Pitfall 3. The initial state of the SSD leads to different transient and steady-state performance. As for the other workloads considered in the paper, this pitfall applies especially to WiredTiger.

These results lead to two main considerations. First, the pitfalls we describe apply to any workload that has a sustained write component that modifies the internal states of the PTS and of the SSD. Second, the extent and timing of the pitfalls vary with the workload. These findings reinforce our message that the performance of PTSes and SSDs are intrinsically intertwined, and must be analyzed carefully and jointly to draw appropriate conclusions.

We finally note that some of our pitfalls are also relevant for read-dominated and even read-only workloads, especially the ones that do not depend strongly on the write intensity of the workload, i.e., the ones regarding the dataset size, the space amplification, and the storage technology.

5 RELATED WORK

Performance analyses of SSD-based storage systems. Benchmarking the performance of PTSes on SSDs is a task frequently performed both in academia [3, 4, 6, 7, 20, 21, 42, 44, 48, 61, 64] and in industry [26, 34, 77, 78].

In general, these evaluations fall short in considering the benchmarking pitfalls we discuss in this paper. For example, the evaluations of the systems do not report the duration of the experiments, or they do not specify the initial conditions of the SSD on which experiments are run, or consider a single dataset size. As we show in this paper, these aspects are crucial for both the quantitative and the qualitative analysis of the performance of a data store deployed on an SSD. In addition, performance benchmarks of PTSes typically focus on application-level write amplification to analyze the I/O efficiency and flash-friendliness of a system [3, 20, 21, 42, 48, 61]. We show that also device-level write amplification must be taken into account for these purposes.

A few systems distinguish between bursty and sustained performance, by investigating the variations of throughput and latencies over time in LSM-tree key value stores [4, 44, 64]. Balmau et al. [4] additionally show how some optimizations can improve the performance of LSM-Tree key-value stores in the short term, but lead to performance degradation in the long run. These works focus on the high-level, i.e., data-structure specific, causes of such performance dynamics. Our work, instead, investigates the low-level causes of performance variations in PTSes, and correlates them with the idiosyncratic performance dynamics of SSDs.

Yang et al. [80] show that stacking several log-structured data structures may hinder the effectiveness of the log-structured design. Oh et al. [54] investigate the use of SSD over-provisioning to increase the performance of a SSD-based cache. Athanassoulis et al. [1] propose the RUM conjecture, which states that PTSes have an inherent trade-off between performance and storage cost. Our paper touches some of these issues, and complements the findings of these works, by covering in a systematic fashion several pitfalls of benchmarking PTSes on SSDs, and by providing experimental evidence for each of them.

SSD performance modeling and benchmarking. The research and industry storage communities have produced much work on modeling and benchmarking the performance of SSDs. The Storage Networking Industry Association has defined the Solid State Storage Performance Test Specification [70], which contains the guidelines to perform rigorous and reproducible SSD benchmarking. Many analytical models [23, 33, 66, 67] express in closed form the performance and the device-level WA of an SSD as a function of the workload and the SSD internals and parameters. MQSim [68] is a simulator specifically designed to replicate quickly and accurately the behavior of an SSD at steady state.

These works focus on the performance of the bare SSD and the garbage collection process, either using synthetic workloads (e.g., generated using `fio`), or simulations, rather than using real-world systems and workloads. We complement these results by focusing on the effects low-level SSD dynamics have on the end performance of two widely used PTSes. We cover additional dimensions other than the SSD garbage collection process, such as the impact on performance of the dataset size, space amplification, software over-provisioning, and SSD technology. We also provide guidelines on how to conduct a more rigorous and SSD-aware performance benchmarking. By studying the behavior of two of the most widely used PTSes we aim to raise awareness about the SSD performance intricacies in the systems and databases communities.

System benchmarking. Benchmarking a system is a notoriously difficult task, and can incur subtle pitfalls that may undermine its results and conclusions. Such a complexity is epitomized by the list of *benchmarking crimes* [31], a popular collection of benchmarking errors that are frequently found in the evaluation of research papers. Raasveldt et al. [60] provide a similar list with a focus on DB systems. Many research papers target different aspects of the process of benchmarking a system. Mariq et al. [47], Uta et al. [73], and Hoefler and Belli [32] focus on the statistical relevance of the measurements, investigating whether experiments can be repeatable, and how many trials are needed to consider a set of measurements meaningful. Our work is complementary to this body of research, in that it aims to provide guidelines to obtain a more fair and reproducible performance assessment of PTSes deployed on SSDs.

6 CONCLUSION

The complex interaction between a persistent tree data structure and a flash SSD device can easily lead to inaccurate performance measurements. In this paper we show seven pitfalls that one can incur when benchmarking a persistent tree data structure on a flash SSD. We demonstrate these pitfalls using RocksDB and WiredTiger, two of the most widespread implementations of the LSM-tree and of the B+tree persistent data structures, respectively. We also present guidelines to avoid the benchmarking pitfalls, so as to obtain accurate and representative performance measurements. We hope that our work raises awareness about and provides a deeper understanding of some benchmarking pitfalls, and that it paves the way for a more rigorous, fair, and reproducible benchmarking.

REFERENCES

[1] Manos Athanassoulis, Michael Kester, Lukas Maas, Radu Stoica, Stratos Idreos, Anastassia Ailamaki, and Mark Callaghan. 2016. Designing Access Methods: The RUM Conjecture. In *International Conference on Extending Database Technology (EDBT)*. 461–466.

[2] AWS. 2020. SSD Instance Store Volumes. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ssd-instance-store.html. Accessed: 2020-11-09.

[3] Oana Balmau, Diego Didona, Rachid Guerraoui, Willy Zwaenepoel, Huapeng Yuan, Aashray Arora, Karan Gupta, and Pavan Konka. 2017. TRIAD: Creating Synergies Between Memory, Disk and Log in Log Structured Key-Value Stores. In *USENIX Annual Technical Conference (ATC)*. 363–375.

[4] Oana Balmau, Florin Dinu, Willy Zwaenepoel, Karan Gupta, Ravishankar Chandhiramoorthi, and Diego Didona. 2020. SILK+ Preventing Latency Spikes in Log-Structured Merge Key-Value Stores Running Heterogeneous Workloads. *ACM Trans. Comput. Syst.* 36, 4, Article 12 (May 2020), 27 pages.

[5] Nikos Batsaras, Giorgos Saloustros, Anastasios Papagiannis, Panagiota Fatourou, and Angelos Bilas. 2020. VAT: Asymptotic Cost Analysis for Multi-Level Key-Value Stores. *CoRR* abs/2003.00103 (2020). https://arxiv.org/abs/2003.00103

[6] Laurent Bindschaedler, Ashvin Goel, and Willy Zwaenepoel. 2020. Hailstorm: Disaggregated Compute and Storage for Distributed LSM-based Databases. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 301–316.

[7] Edward Bortnikov, Anastasia Braginsky, Eshcar Hillel, Idit Keidar, and Gali Sheffi. 2018. Accordion: Better Memory Organization for LSM Key-Value Stores. *VLDB Endow.* 11, 12 (Aug. 2018), 1863–1875.

[8] Gerth Stolting Brodal and Rolf Fagerberg. 2003. Lower Bounds for External Memory Dictionaries. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 546–554.

[9] Mark Callaghan. 2015. Different kinds of copy-on-write for a b-tree: CoW-R, CoW-S. http://smalldatum.blogspot.com/2015/08/different-kinds-of-copy-on-write-for-b.html. Accessed: 2020-11-09.

[10] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2008. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* 26, 2, Article 4 (June 2008), 26 pages.

[11] Asaf Cidon, Daniel Rushton, Stephen M. Rumble, and Ryan Stutsman. 2017. Memshare: a Dynamic Multi-tenant Key-value Cache. In *USENIX Annual Technical Conference (ATC)*. 321–334.

[12] Google Cloud. 2020. Storage options. https://tinyurl.com/y36r3yxx. Accessed: 2020-11-09.

[13] IBM Cloud. 2020. Storage options. https://cloud.ibm.com/docs/vsi?topic=virtual-servers-storage-options. Accessed: 2020-11-09.

[14] Douglas Comer. 1979. Ubiquitous B-Tree. *ACM Comput. Surv.* 11, 2 (June 1979), 121–137.

[15] Intel Corporation. 2015. Intel Linux NVMe Driver. https://www.intel.com/content/dam/support/us/en/documents/ssdc/data-center-ssds/Intel_Linux_NVMe_Guide_330602-002.pdf. Accessed: 2020-11-09.

[16] Intel Corporation. 2019. Intel 660p SSD. https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/660p-series-brief.pdf Accessed: 2020-11-09.

[17] Intel Corporation. 2020. Intel Solid-State Drive DC P3600 Series. Product specification.

[18] Intel Corporation. 2020. Intel® Optane™ Technology. https://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html. Accessed: 2020-11-09.

[19] Tom Coughlin. 2019. Digital Storage Projections For 2020. https://tinyurl.com/yxu4n98u. Accessed: 2020-11-09.

[20] Niv Dayan, Manos Athanassoulis, and Stratos Idreos. 2018. Optimal Bloom Filters and Adaptive Merging for LSM-Trees. *ACM Trans. Database Syst.* 43, 4, Article 16 (Dec. 2018), 48 pages.

[21] Niv Dayan and Stratos Idreos. 2018. Dostoevsky: Better Space-Time Trade-Offs for LSM-Tree Based Key-Value Stores via Adaptive Removal of Superfluous Merging. In *ACM International Conference on Management of Data (SIGMOD)*. 505–520.

[22] Niv Dayan and Stratos Idreos. 2019. The Log-Structured Merge-Bush & the Wacky Continuum. In *ACM International Conference on Management of Data (SIGMOD)*. 449–466.

[23] Peter Desnoyers. 2014. Analytic Models of SSD Write Performance. *ACM Trans. Storage* 10, 2, Article 8 (March 2014), 25 pages.

[24] Cristian Diaconu, Craig Freedman, Erik Ismert, Per-Ake Larson, Pravin Mittal, Ryan Stonecipher, Nitin Verma, and Mike Zwilling. 2013. Hekaton: SQL Server's Memory-Optimized OLTP Engine. In *ACM International Conference on Management of Data (SIGMOD)*. 1243–1254.

[25] Facebook. 2013. McDipper: A key-value cache for Flash storage. https://www.facebook.com/notes/facebook-engineering/mcdipper-a-key-value-cache-for-flash-storage/10151347090423920. Accessed: 2020-11-09.

[26] Facebook. 2020. RocksDB Performance Benchmarks. https://github.com/facebook/rocksdb/wiki/Performance-Benchmarks. Accessed: 2020-11-09.

[27] Facebook. 2020. The RocksDB persistent key-value store. http://rocksdb.org. Accessed: 2020-11-09.

[28] Facebook. 2020. RocksDB tuning guide. https://github.com/facebook/rocksdb/wiki/RocksDB-Tuning-Guide. Accessed: 2020-11-09.

[29] Google. 2020. LevelDB. https://github.com/google/leveldb. Accessed: 2020-11-09.

[30] Carnegie Mellon University Database Group. 2019. Peloton. https://pelotondb.io/about/. Accessed: 2020-11-09.

[31] Gernot Heiser. 2010. System Benchmarking Crimes. https://www.cse.unsw.edu.au/~gernot/benchmarking-crimes.html. Accessed: 2020-11-09.

[32] Torsten Hoefler and Roberto Belli. 2015. Scientific Benchmarking of Parallel Computing Systems: Twelve Ways to Tell the Masses When Reporting Performance Results. In *ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. Article 73, 12 pages.

[33] Xiao-Yu Hu, Evangelos Eleftheriou, Robert Haas, Ilias Iliadis, and Roman Pletka. 2009. Write Amplification Analysis in Flash-Based Solid State Drives. In *ACM International Systems & Storage Conference. (SYSTOR)*. Article 10, 9 pages.

[34] HyperDex. 2020. HyperLevelDB Performance Benchmarks. http://hyperdex.org/performance/leveldb/s. Accessed: 2020-01-31.

[35] IBM. 2020. DB2 Index Structure. https://www.ibm.com/support/knowledgecenter/SSEPGG_11.1.0/com.ibm.db2.luw.admin.perf.doc/doc/c0005300.html. Accessed: 2020-11-09.

[36] Nikolas Ioannou, Kornilios Kourtis, and Ioannis Koltsidas. 2018. Elevating Commodity Storage with the SALSA Host Translation Layer. In *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 277–292.

[37] Steve Knipple. 2017. Leveraging the Latest Flash in the Data Center. Flash Memory Summit. https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2017/20170809_FG21_Knipple.pdf. Accessed: 2020-11-09.

[38] Kornilios Kourtis, Nikolas Ioannou, and Ioannis Koltsidas. 2019. Reaping the Performance of Fast NVM Storage with Udepot. In *USENIX Conference on File and Storage Technologies (FAST)*. 1–15.

[39] Baptiste Lepers, Oana Balmau, Karan Gupta, and Willy Zwaenepoel. 2019. KVell: The Design and Implementation of a Fast Persistent Key-Value Store. In *ACM Symposium on Operating Systems Principles (SOSP)*. 447–461.

[40] Justin J. Levandoski, David B. Lomet, and Sudipta Sengupta. 2013. The Bw-Tree: A B-Tree for New Hardware Platforms. In *IEEE International Conference on Data Engineering (ICDE)*. 302–313.

[41] Hyeontaek Lim, David G. Andersen, and Michael Kaminsky. 2016. Towards Accurate and Fast Evaluation of Multi-Stage Log-Structured Designs. In *USENIX Conference on File and Storage Technologies (FAST)*. 149–166.

[42] Hyeontaek Lim, Bin Fan, David G. Andersen, and Michael Kaminsky. 2011. SILT: A Memory-Efficient, High-Performance Key-Value Store. In *ACM Symposium on Operating Systems Principles (SOSP)*. 1–13.

[43] Lanyue Lu, Thanumalayan Sankaranarayana Pillai, Hariharan Gopalakrishnan, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2017. WiscKey: Separating Keys from Values in SSD-Conscious Storage. *ACM Trans. Storage* 13, 1, Article 5 (March 2017), 28 pages.

[44] Chen Luo and Michael J. Carey. 2019. On Performance Stability in LSM-Based Storage Systems. *VLDB Endow.* 13, 4 (Dec. 2019), 449–462.

[45] Chen Luo and Michael J. Carey. 2020. LSM-based storage techniques: a survey. *The VLDB Journal* 29, 1 (2020), 393–418.

[46] Dongzhe Ma, Jianhua Feng, and Guoliang Li. 2014. A Survey of Address Translation Technologies for Flash Memories. *ACM Comput. Surv.* 46, 3, Article 36 (Jan. 2014), 39 pages.

[47] Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, and Robert Ricci. 2018. Taming Performance Variability. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 409–425.

[48] Leonardo Marmol, Swaminathan Sundararaman, Nisha Talagala, and Raju Rangaswami. 2015. NVMKV: A Scalable, Lightweight, FTL-aware Key-Value Store. In *USENIX Annual Technical Conference (ATC)*. 207–219.

[49] Micron. 2020. 3D XPoint™ Technology. https://www.micron.com/products/advanced-solutions/3d-xpoint-technology. Accessed: 2020-11-09.

[50] minitool.com. 2020. SSD Prices Continue to Fall, Now Upgrade Your Hard Drive! https://www.minitool.com/news/ssd-prices-fall.html. Accessed: 2020-11-09.

[51] MongoDB. 2020. MongoDB's WiredTiger Storage Engine. https://docs.mongodb.com/manual/core/wiredtiger/. Accessed: 2020-11-09.

[52] MongoDB. 2020. The WiredTiger storage engine. http://source.wiredtiger.com. Accessed: 2020-11-09.

[53] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. 2009. Migrating Server Storage to SSDs: Analysis of Tradeoffs. In *ACM European Conference on Computer Systems (EuroSys)*. 145–158.

[54] Yongseok Oh, Jongmoo Choi, Donghee Lee, and Sam H. Noh. 2012. Caching Less for Better Performance: Balancing Cache Size and Update Cost of Flash Memory Cache in Hybrid Storage Systems. In *USENIX Conference on File and Storage Technologies (FAST)*. 25.

[55] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. 1996. The Log-Structured Merge-Tree (LSM-Tree). *Acta Informatica* 33, 4 (June 1996), 351–385.

[56] E. S. Page. 1954. Continuous Inspection schemes. *Biometrika* 41, 1-2 (06 1954), 100–115.

[57] Anastasios Papagiannis, Giorgos Saloustros, Pilar González-Férez, and Angelos Bilas. 2016. Tucana: Design and Implementation of a Fast and Efficient Scale-up Key-value Store. In *USENIX Annual Technical Conference (ATC)*. 537–550.

[58] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. 2017. Self-Driving Database Management Systems. In *Conference on Innovative Data Systems Research (CIDR)*.

[59] Percona. 2020. TokuMX. https://www.percona.com/software/mongo-database/percona-tokumx. Accessed: 2020-11-09.

[60] Mark Raasveldt, Pedro Holanda, Tim Gubner, and Hannes Mühleisen. 2018. Fair Benchmarking Considered Difficult: Common Pitfalls In Database Performance Testing. In *Workshop on Testing Database Systems (DBTest)*. Article 2, 6 pages.

[61] Pandian Raju, Rohan Kadekodi, Vijay Chidambaram, and Ittai Abraham. 2017. PebblesDB: Building Key-Value Stores Using Fragmented Log-Structured Merge Trees. 497–514.

[62] Kai Ren, Qing Zheng, Joy Arulraj, and Garth Gibson. 2017. SlimDB: A Space-Efficient Key-Value Storage Engine for Semi-Sorted Data. *VLDB Endow.* 10, 13 (Sept. 2017), 2037–2048.

[63] Bianca Schroeder, Raghav Lagisetty, and Arif Merchant. 2016. Flash Reliability in Production: The Expected and the Unexpected. In *USENIX Conference on File and Storage Technologies (FAST)*. 67–80.

[64] Russell Sears and Raghu Ramakrishnan. 2012. BLSM: A General Purpose Log Structured Merge Tree. In *ACM International Conference on Management of Data (SIGMOD)*. 217–228.

[65] Zhaoyan Shen, Feng Chen, Yichen Jia, and Zili Shao. 2018. DIDACache: An Integration of Device and Application for Flash-Based Key-Value Caching. *ACM Trans. Storage* 14, 3, Article 26 (Oct. 2018).

[66] Radu Stoica and Anastasia Ailamaki. 2013. Improving Flash Write Performance by Using Update Frequency. *VLDB Journal* 6, 9 (2013), 733–744.

[67] Radu Stoica, Roman Pletka, Nikolas Ioannou, Nikolaos Papandreou, Sasa Tomic, and Haralampos Pozidis. 2019. Understanding the Design Trade-Offs of Hybrid Flash Controllers. In *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 152–164.

[68] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. 2018. MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices. In *USENIX Conference on File and Storage Technologies (FAST)*. 49–66.

[69] techpowerup.com. 2018. SSDs Are Cheaper Than Ever, Hit the Magic 10 Cents Per Gigabyte Threshold. https://www.techpowerup.com/249972/ssds-are-cheaper-than-ever-hit-the-magic-10-cents-per-gigabyte-threshold. Accessed: 2020-11-09.

[70] The Storage Networking Industry Association. 2020. Solid State Storage (SSS) Performance Test Specification (PTS). https://www.snia.org/tech_activities/standards/curr_standards/pts. Accessed: 2020-11-09.

[71] Animesh Trivedi, Nikolas Ioannou, Bernard Metzler, Patrick Stuedi, Jonas Pfefferle, Kornilios Kourtis, Ioannis Koltsidas, and Thomas R. Gross. 2018. FlashNet: Flash/Network Stack Co-Design. *ACM Trans. Storage* 14, 4, Article 30 (Dec. 2018).

[72] Twitter. 2013. Fatcache. https://github.com/twitter/fatcache. Accessed: 2020-11-09.

[73] Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan S. Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. 2020. Is Big Data Performance Reproducible in Modern Cloud Networks?. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 513–527.

[74] Richard L. Villars and Eric Burgener. 2014. IDC: Building data centers for todays data driven economy: The role of flash. Flash Memory Summit. https://www.sandisk.it/content/dam/sandisk-main/en_us/assets/resources/enterprise/white-papers/flash-in-the-data-center-idc.pdf.

[75] Peng Wang, Guangyu Sun, Song Jiang, Jian Ouyang, Shiding Lin, Chen Zhang, and Jason Cong. 2014. An Efficient Design and Implementation of LSM-Tree Based Key-Value Store on Open-Channel SSD. In *European Conference on Computer Systems (EuroSys)*. Article 16, 14 pages.

[76] Ziqi Wang, Andrew Pavlo, Hyeontaek Lim, Viktor Leis, Huanchen Zhang, Michael Kaminsky, and David G. Andersen. 2018. Building a Bw-Tree Takes More Than Just Buzz Words. In *ACM International Conference on Management of Data (SIGMOD)*. 473–488.

[77] WiredTiger. 2013. LevelDB Performance Benchmarks. https://github.com/wiredtiger/wiredtiger/wiki/LevelDB-Benchmark. Accessed: 2020-11-09.

[78] WiredTiger. 2017. WiredTiger Performance Benchmarks. https://github.com/wiredtiger/wiredtiger/wiki/Btree-vs-LSM. Accessed: 2020-11-09.

[79] Kan Wu, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Rathijit Sen, and Kwanghyun Park. 2019. Exploiting Intel Optane SSD for Microsoft SQL Server. In *International Workshop on Data Management on New Hardware (DaMoN)*. Article 15, 3 pages.

[80] Jingpei Yang, Ned Plasson, Greg Gillis, Nisha Talagala, and Swaminathan Sundararaman. 2014. Don't Stack Your Log On My Log. In *Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW)*.