Toward a better understanding and benchmarking of tree data structures on flash SSDs

Diego Didona, Nikolas Ioannou, Radu Stoica, Kornilios Kourtis

IBM Research - Zurich

Per3S 2022

This talk in a nutshell

We discuss several widespread pitfalls of benchmarking **persistent tree data structures** on **flash solid-state drives (SSD)** and their impact.

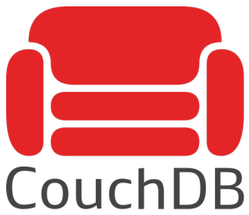
– Non-representative and non-reproducible results
– Suboptimal deployments
– Biased evaluations

We analyze the low-level causes of the pitfalls and suggest solutions

Improve understanding and enable more rigorous benchmarking of the performance of tree data structures on flash SSDs

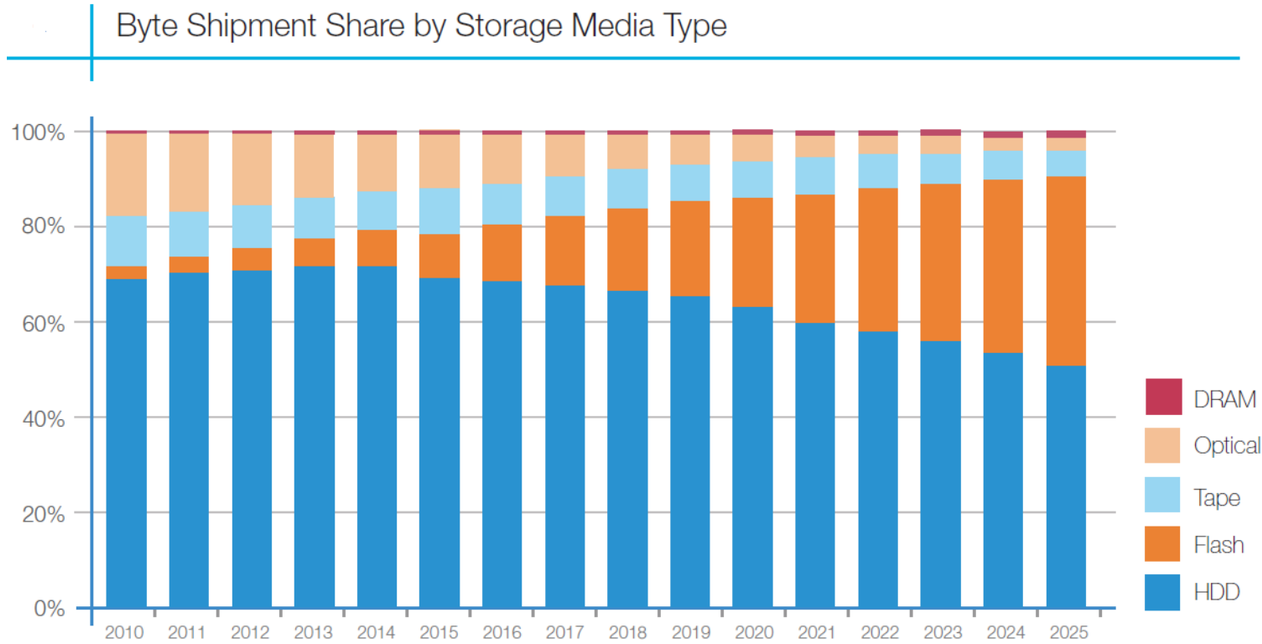
Tree data structures

- Fundamental building block of many production systems
- Efficient indexing of large amount of data



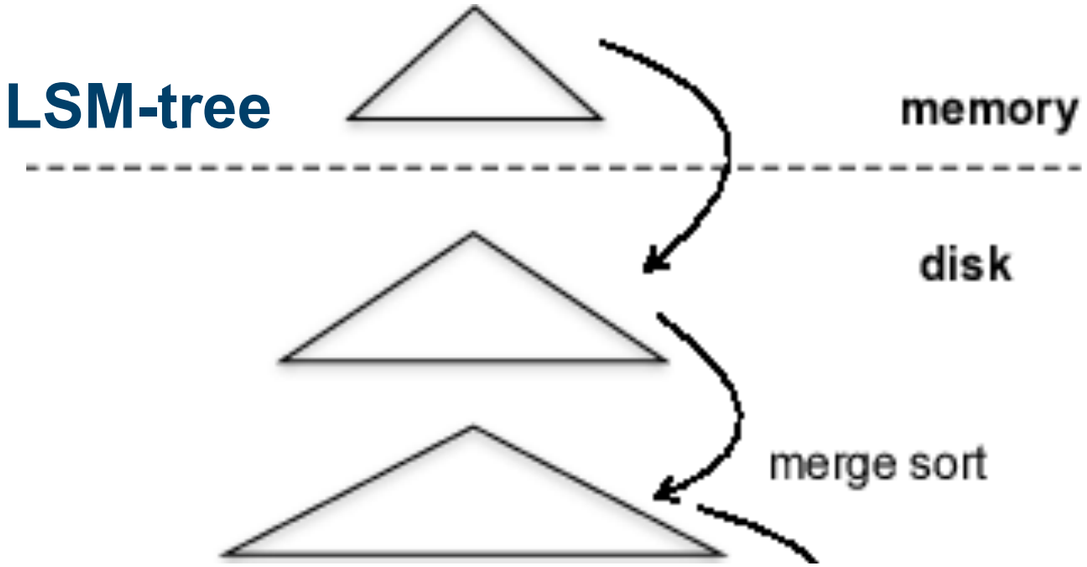
Flash SSDs

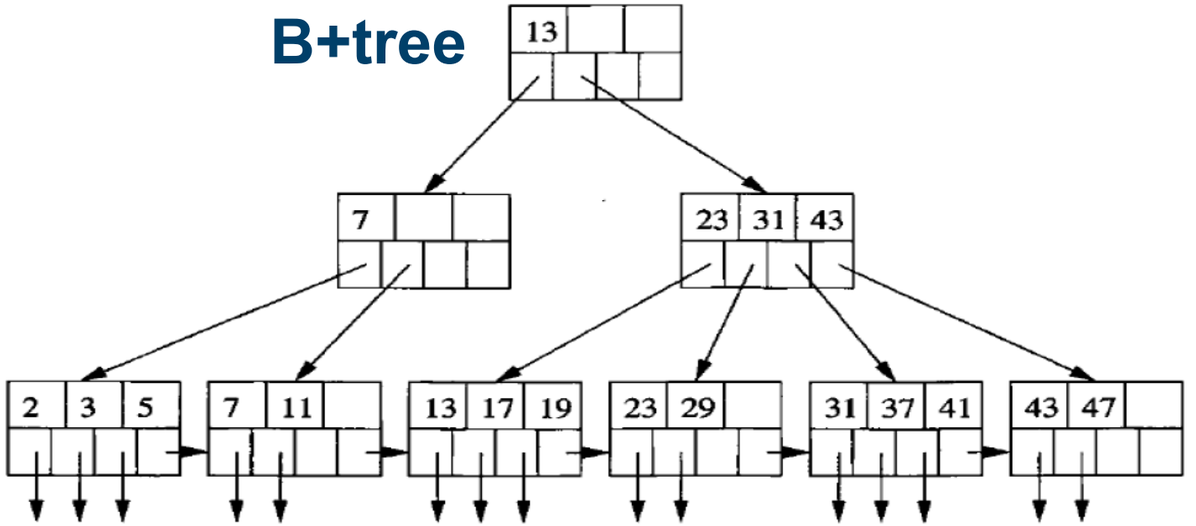
- Medium of choice to store massive data sets in data centers
- Persistency with excellent performance / cost tradeoff



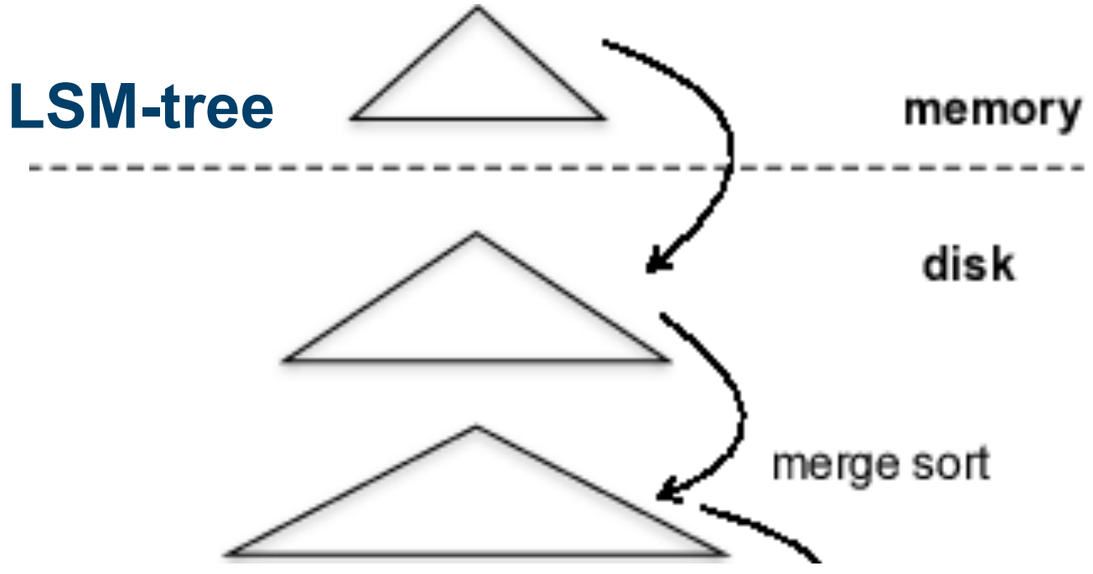
Benchmarking tree data structures on flash SSDs is challenging

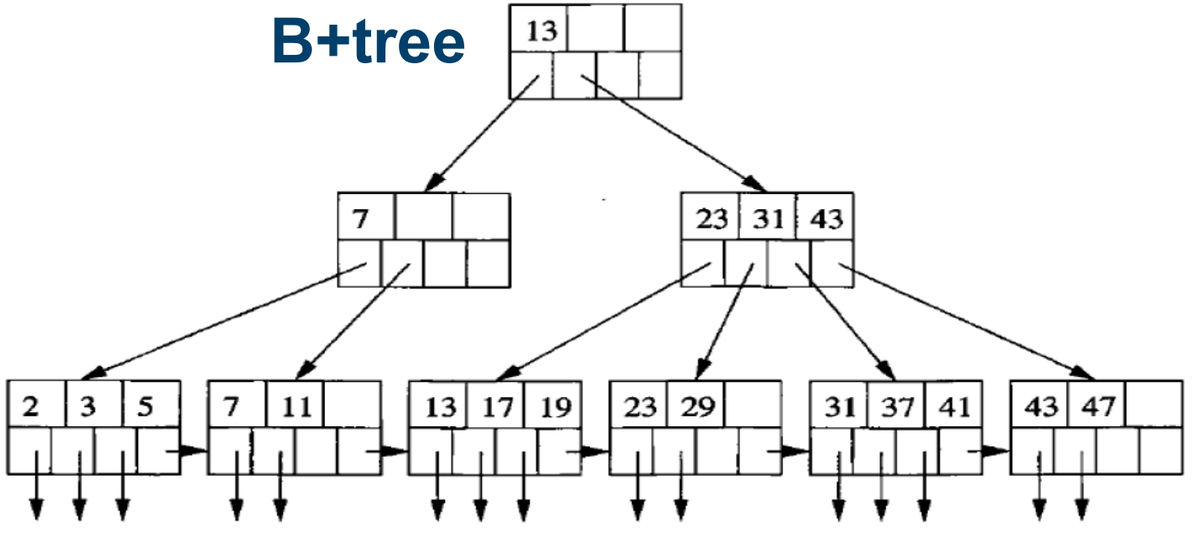
Tree data structures: complex internal operations





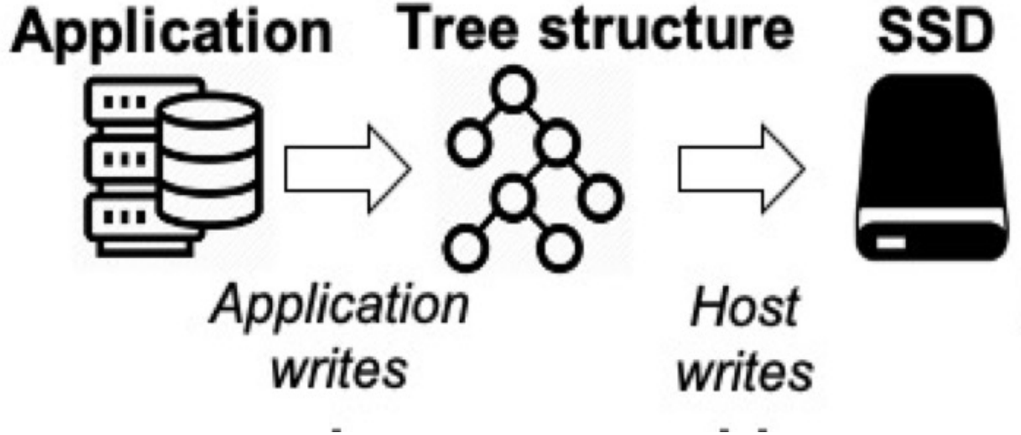
Tree data structures: complex internal operations





Application-level write-amplification:

$$\frac{\text{Bytes written to disk by tree structure}}{\text{Bytes written by user}}$$



Flash SSDs: complex black boxes

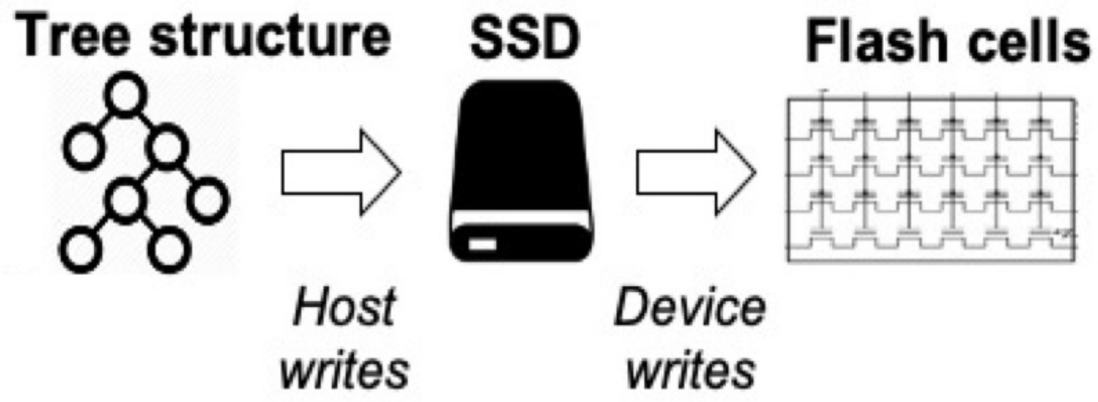
- No in-place updates:
  - Update value in page P → write new value to clean block and invalidate P
  - Pages cannot be erased individually → read large blocks and relocate valid pages
- Error detection, wear levelling, data refresh, metadata IO, journaling, …
- Tiering / multi-bit mode

Flash SSDs: complex black boxes

- No in-place updates:
  - Update value in page P → write new value to clean block and invalidate P
  - Pages cannot be erased individually → read large blocks and relocate valid pages
- Error detection, wear levelling, data refresh, metadata IO, journaling, …
- Tiering / multi-bit mode

Device-level write-amplification:

$$\frac{\text{Bytes written to flash}}{\text{Bytes written to SSD by tree}}$$



Six common benchmarking pitfalls

Pitfalls

1. Running short tests
2. Not analyzing write amplification
3. Ignoring the tree's extra capacity requirements
4. Testing with a singe data-set size
5. Ignoring the initial condition of the SSD
6. Testing with a single type of SSD

Consequences

Obtain non-representative, non-reproducible results

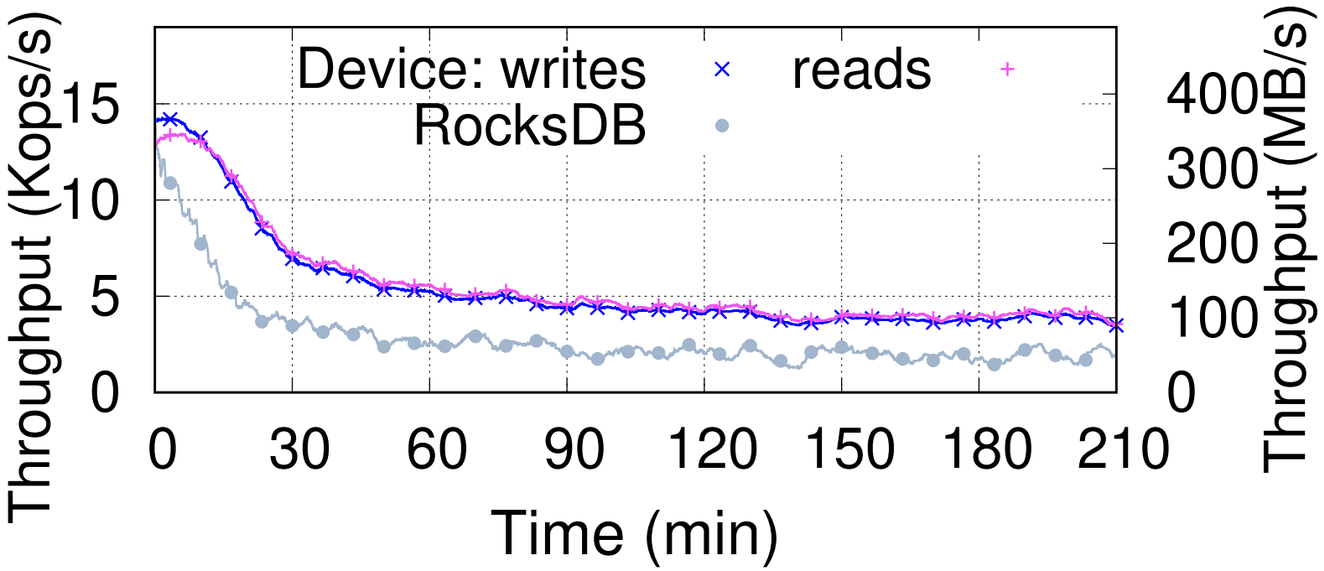
Suboptimal deployment and cost/performance analysis

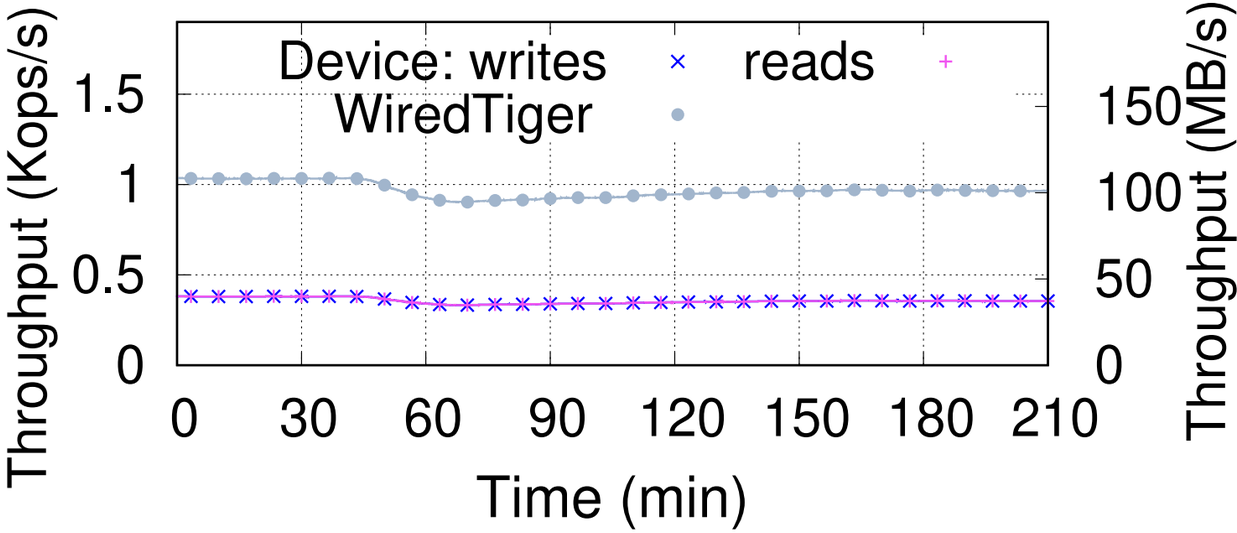
Cannot generalize results

Experimental setup

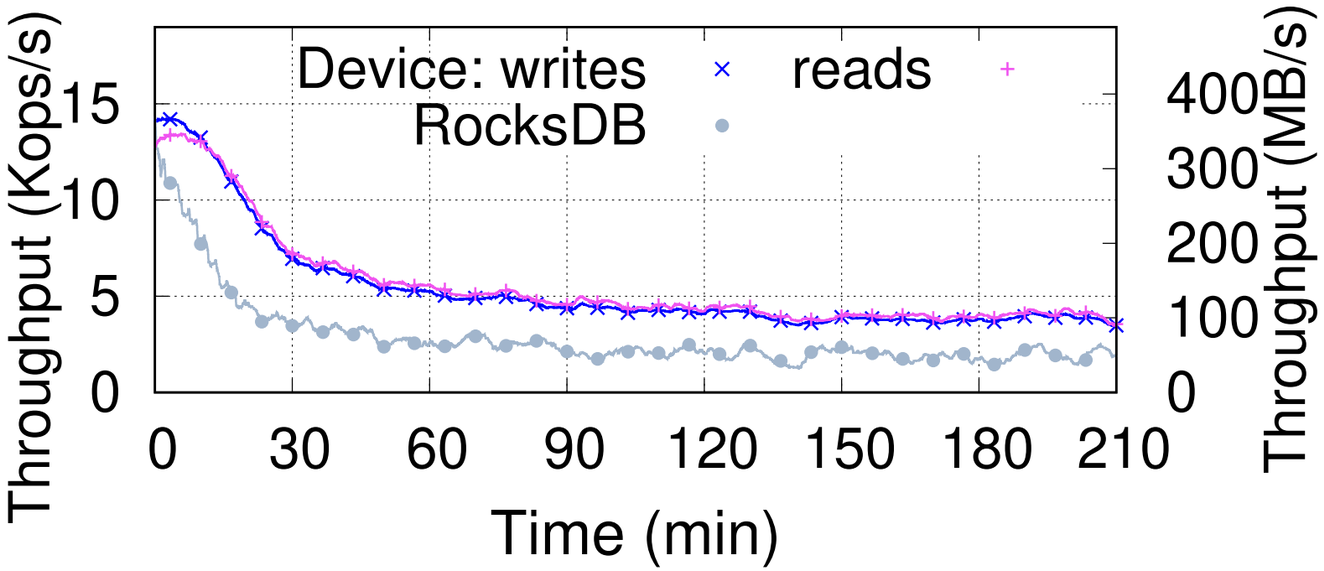
- RocksDB (LSM tree) and WiredTiger (B+tree)
- 400 GB Intel p3600 enterprise-class NVMe SSD
- 200 GB data-set loaded **before** the start of each test
- 100% write workload to stress the internal operations of the trees and the garbage collection dynamics of the SSD (random put operations, Key=8B / Value=128B)
- <100 MB of RAM cache for the systems
  - Data does *not* fit into memory, operations served from disk

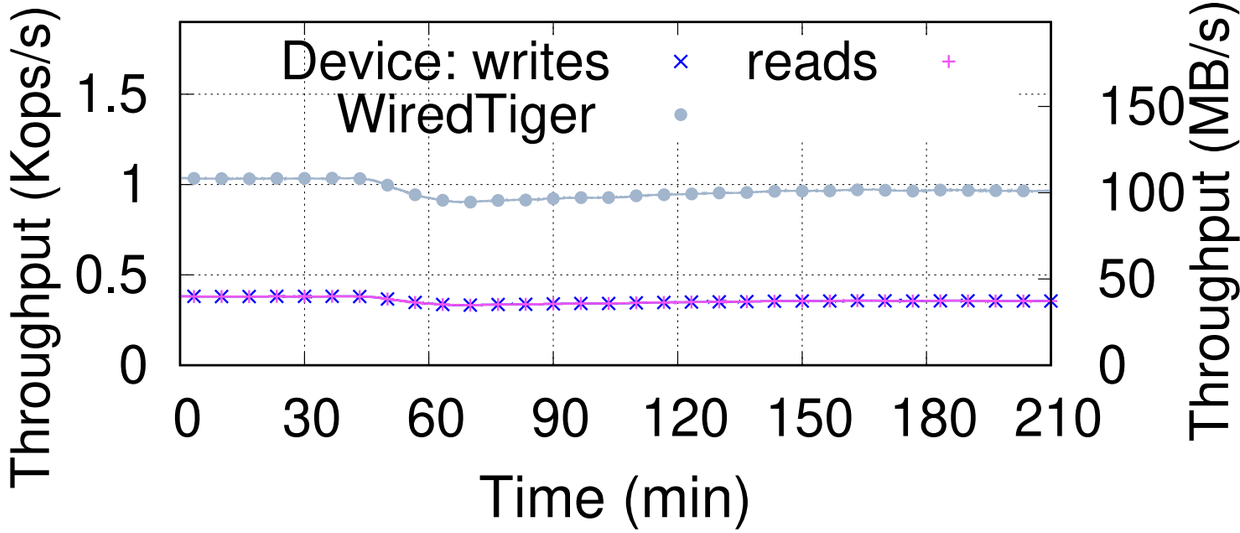
Pitfall 1: running short tests





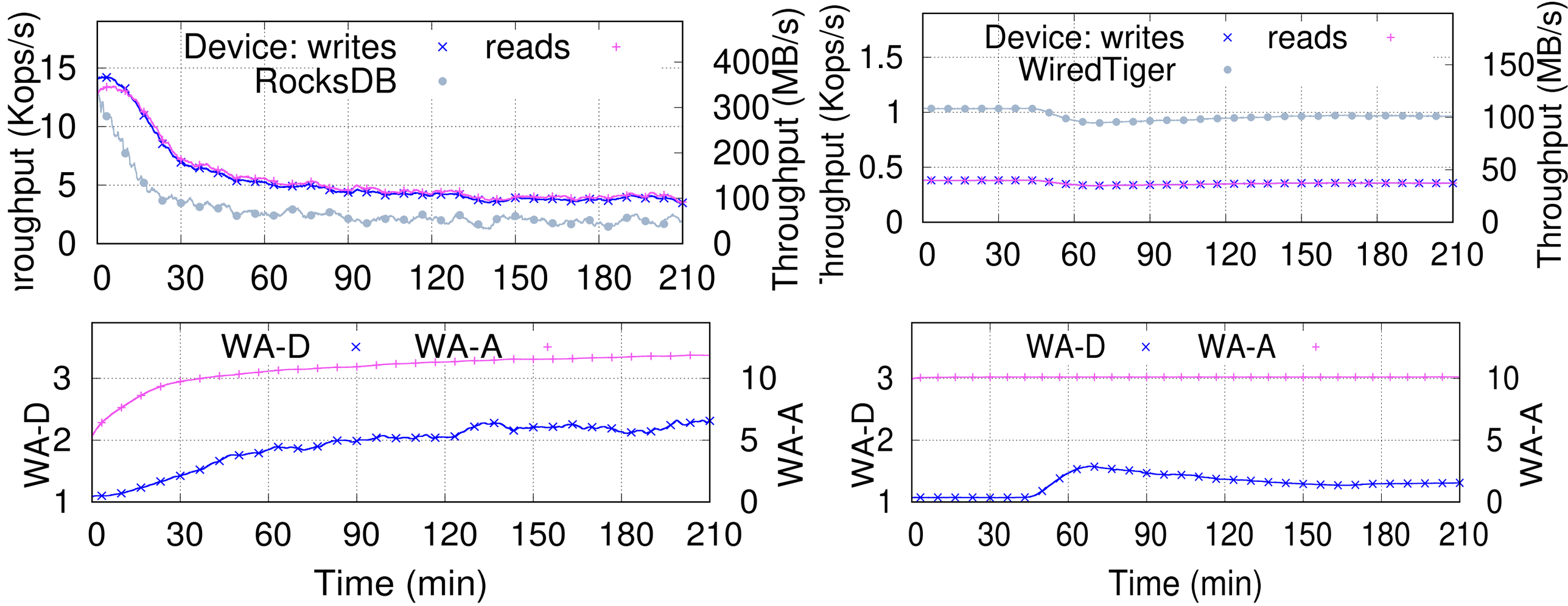
Pitfall 1: running short tests





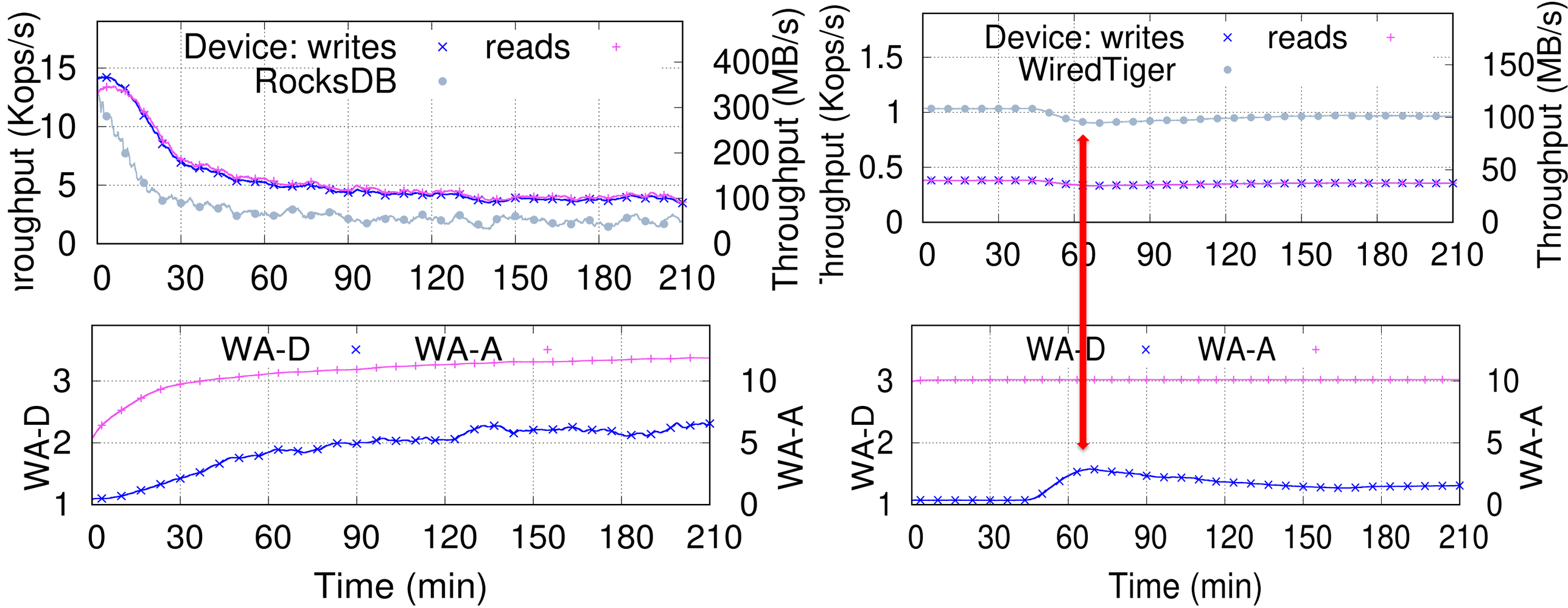
Throughput varies over time:
Reporting performance of a short test is not representative of steady-state performance

Pitfall 2: not considering write amplification



Bottom: application (WA-A) and device (WA-D) write amplifications. WA-A * WA-D determines device lifetime.

Pitfall 2: not considering write amplification

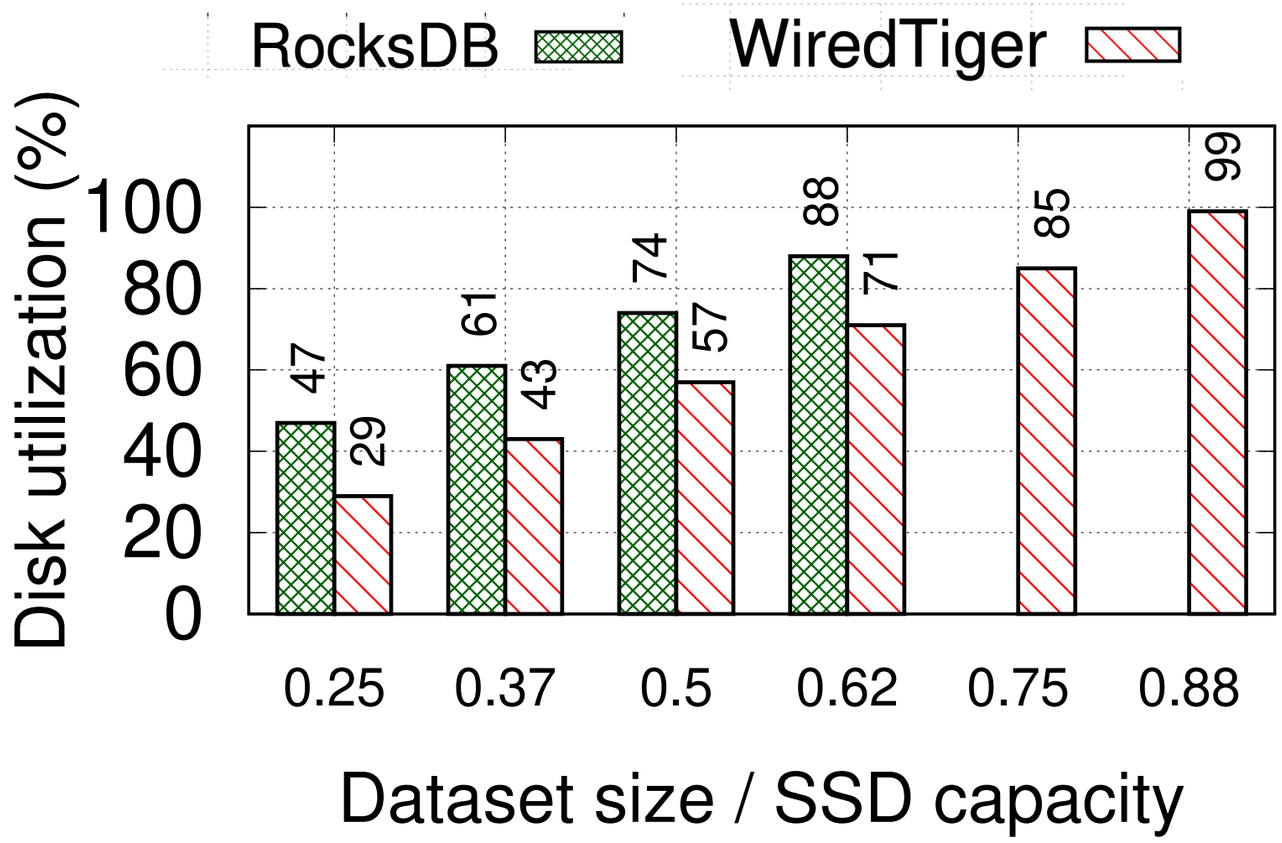


WA-A not sufficient to justify performance variability

WA-D is needed, but is often time neglected

Pitfall 3: neglecting data-structure's space overhead

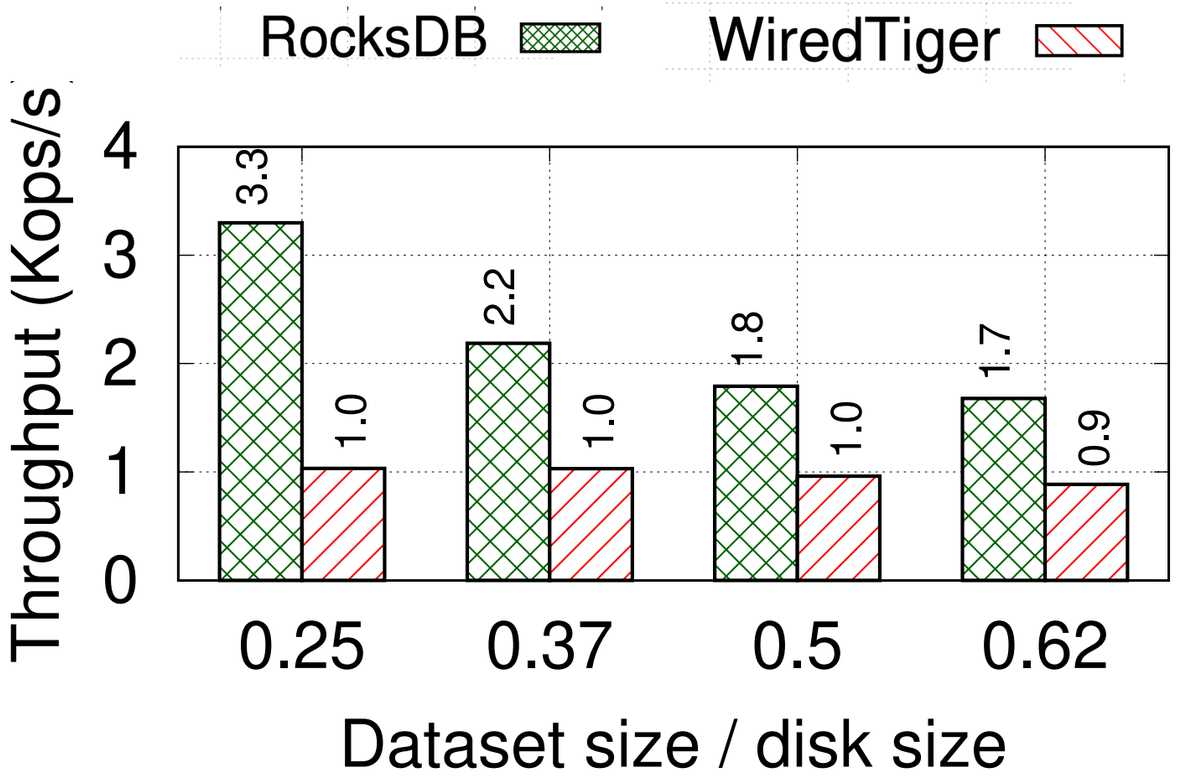
- RocksDB's can index less data
  - Overhead of the several LSM levels
- WiredTiger can be more **cost**-efficient for larger datasets that do not require very high throughput



Using a single data-set size leads to an incomplete cost analysis

Pitfall 4: using a single dataset size

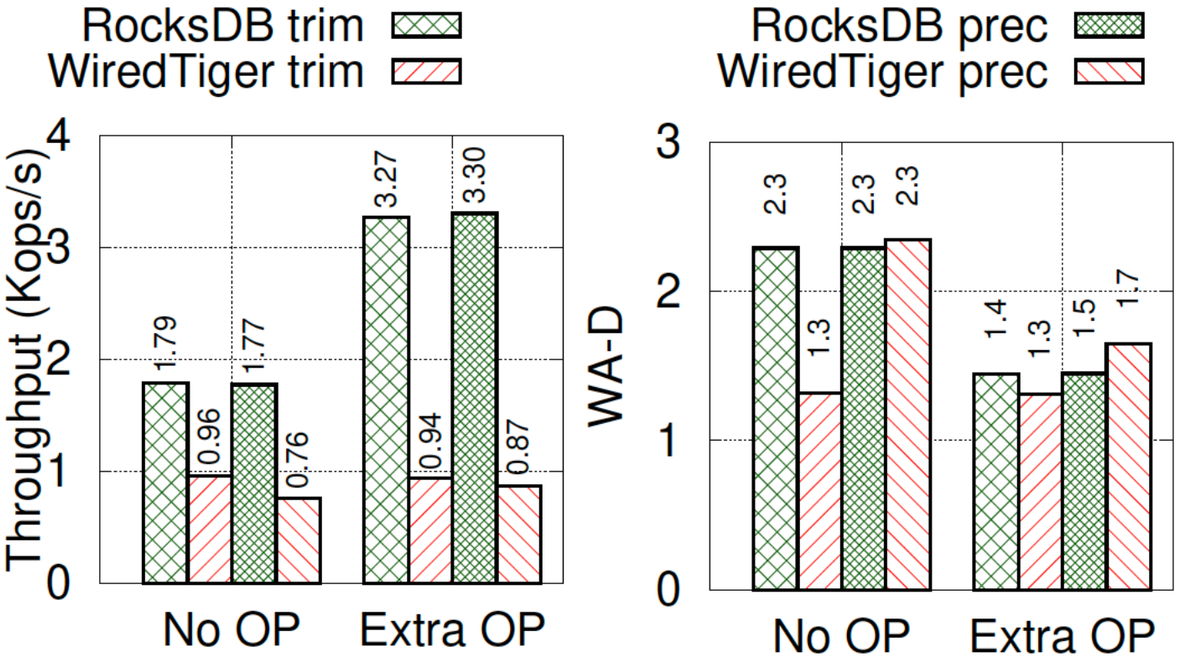
- Larger dataset → lower throughput
  - More valid data → more garbage collection
- Performance advantage of RocksDB shrinks with larger data sets
  - From 3.3X to 1.9X



Using a single data-set leads to biased evaluations and comparisons

Pitfall 5: Ignoring the initial condition of the SSD

- Run similar experiments:
  1. Partition → Trim Partition
  2. Partition → Trim Partition → Pre-cond
  3. Trim SSD → Partition
  4. Trim SSD → Partition → Pre-cond
- The slight differences result in different levels of over-provisioning

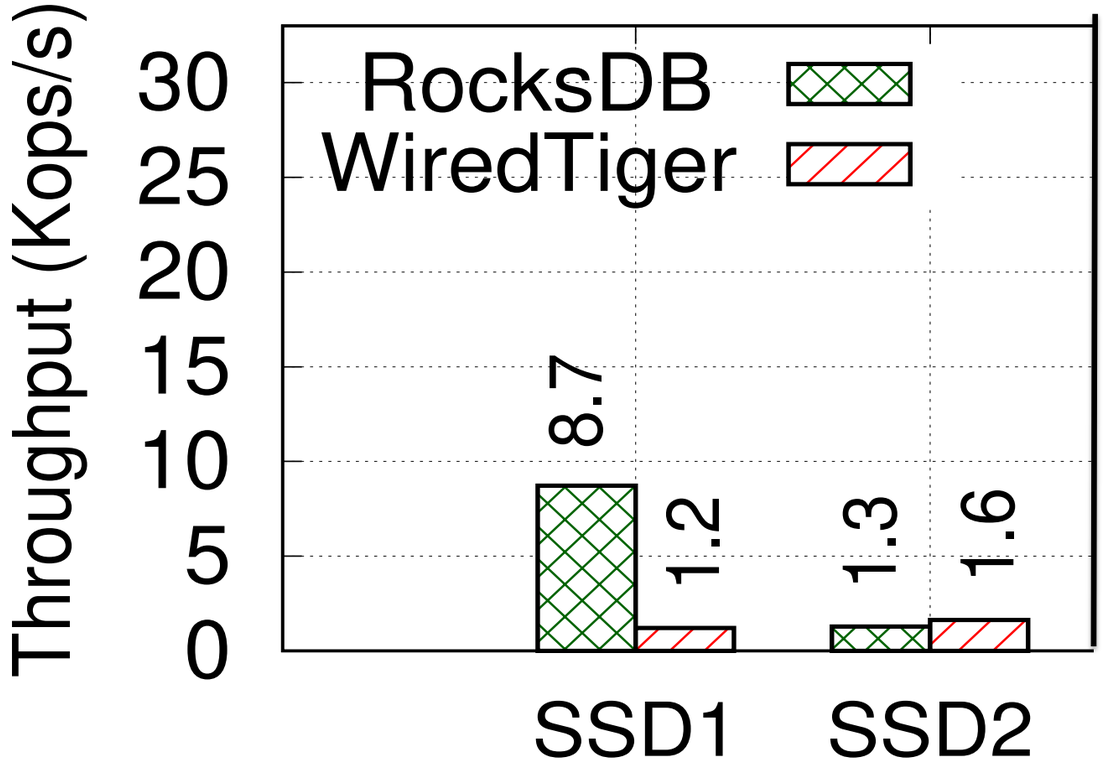


- Performance advantage of RocksDB shrinks when lowering effective OP
  - From 3.5X to 1.9X
- Large change in WA-D and expected lifetime

Pitfall 6: use a single SSD type

- SSD 1: Intel p3600
- SSD 2: Intel 660p

1. RocksDB is better on SSD1
2. WiredTiger is better on SSD2
3. Either can be better than the other depending on the SSD!



Performance results are not directly portable cross-SSDs

We have a paper…

- Additional pitfalls
- More in depth-analysis
- Guidelines to address the pitfalls
- More workloads
- Discussion of related work

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

Conclusion

- Tree data structures on SSDs are first-class citizens in modern storage systems
- Analyzing their performance correctly is hard: complex data structures on complex hardware

- We show experimentally six benchmarking pitfalls that can lead to incomplete, wrong or biased performance and cost evaluations

- We aim to provide better understanding of the performance of these systems and guidelines for a more thorough evaluation

Thank you!

IBM Research Labs > Zurich, Switzerland >

Cloud & AI Systems Research

Research topics:

- **Cloud storage**
- **Non-volatile memories**
- **Accelerators (FPGAs, neuromorphic & analog AI)**
- **Machine learning frameworks**
- **Archival storage**
- **HW design**

www.zurich.ibm.com/cci