Exploring the evolution of storage devices on hardware utilization by benchmarking key-value stores

Rahul Sai, Shulin Pan, Simran Sandhu, Eva Liu
University of Illinois at Urbana-Champaign

Abstract

In this paper, we investigate the performance and resource utilization of key-value stores SplinterDB, RocksDB, and WiredTiger on both NVMe and SATA SSDs. Our primary objective is to reproduce previous studies' results, demonstrating the performance gap between SplinterDB and RocksDB on NVMe, and explore if this behavior persists with SATA-based SSDs. Our secondary objective is to compare the hardware utilization of these key-value stores on different types of SSDs, examining how the evolution of storage devices and interfaces has shifted the bottleneck from disks to CPUs. Our analysis of CPU and disk utilization reveals varying computational demands and optimizations of each key-value store on different storage technologies, highlighting the importance of selecting the appropriate key-value store for specific hardware configurations. Our findings provide valuable insights for future research and development of key-value stores and storage systems, enabling developers to make well-informed decisions in designing storage systems optimized for their specific hardware and use cases.

1. Introduction

With the improvements in storage-media and CPUs in the past years, the benchmarks of popular databases have shown varying results in terms of latency, R/W throughput, both due to implementation details and the underlying hardware. The performance bottlenecks have slowly started shifting from the storage media to the CPUs and NICs. Benchmarking studies have commonly focused on comparing performance of databases and/or storage interfaces and have ignored the overhead on hardware resources. Our primary objective is to explore this resource utilization shift, on different storage interfaces such as SATA and NVMe to CPU, by running various types of workloads on popular KV stores. We are trying to reproduce the result of [1] where SplinterDB outperforms RocksDB significantly on NVMe SSDs. We are also trying to explore if this behavior remains unaffected on SATA based SSDs. We will also compare the hardware utilization of Write optimized DBs on various type of SSDs i.e., SATA and NVMe, to analyze how the evolution of storage devices and interfaces have shifted the bottleneck from disks to CPUs. Finally, we want to compare the hardware utilization (CPU and Disk) of LSM-Based KV Stores vs. B+ Trees KV Stores.

Our analysis highlights that SplinterDB outperforms the other two on both types of SSDs, with optimized performance on NVMe leading to higher throughput. SplinterDB has moderate CPU utilization on both NVMe and SATA SSDs, while WiredTiger has slightly higher CPU utilization, and RocksDB has the highest CPU utilization on both storage devices. A higher CPU utilization of a database suggests that it requires more computational resources to handle its operations effectively. This report begins with an introduction to the KV-stores we're using for the analysis, and then gives details about our experimental setup, results and subsequently delves into a comprehensive analysis of the results.

2. Background and Motivation

Storage devices have come a long way, from disk-based storage to the now widely used Solid State Drives (SSDs), which have begun to be widely used since late 2000s for their efficient data accesses using NAND based flash memory. SSDs usually use one of the two popular interfaces. SATA was introduced in 2000 and has a maximum transfer speed of 6 Gbps, or 750 MBps. It uses Advanced Host Controller Interface) AHCI protocol which allows HDDs to achieve higher levels of throughput and performance. However, it also creates a performance bottleneck for newer SSDs. It is suitable for HDDs and budget SSDs and is overall lower-cost compared to NVMe SSDs.

Non-Volatile Memory Express (NVMe) SSDs are relatively newer; introduced in 2011. The max transfer speed is 32 Gbps, or around 4 GBps. NVMe is optimized for the low-latency, high-bandwidth characteristics of SSDs. It also, however, comes at a higher cost. But, since the mid-2010s, NVMe SSDs have become more affordable and have started to replace SATA SSDs in many applications, especially in high-performance computing and gaming. Since the advent of newer storage interfaces, the bottlenecks have shifted more towards CPU and networks.

RocksDB and SplinterDB are write-optimized key value stores, with RocksDB (LSM-Based) optimized for flash-

based storage and SplinterDB (STBε-Tree based) optimized for NVMe respectively. SplinterDB, was introduced by VMWare Research in 2020. It is built on top of STBε trees, a novel data structure that merges concepts from log structured merge trees and B-epsilon trees. SplinterDB is specifically designed and optimized to leverage the capabilities of NVMe solid-state drives. By minimizing write amplification, SplinterDB achieves exceptional performance, enabling efficient and speedy data storage and retrieval operations.

RocksDB, a widely adopted open-source key-value storage engine, was initially developed by Facebook and has been available to the public since 2012. Built upon the principles of Log-Structured Merge Trees (LSM-Trees), RocksDB is engineered to deliver exceptional performance and efficiency. It is specifically optimized for high-speed storage mediums like flash drives and high-speed disk drives, enabling fast and low-latency operations. Due to its impressive capabilities, RocksDB has gained significant popularity and is extensively used by prominent services such as Uber, Netflix, Facebook, and Airbnb.

WiredTiger, an open-source NoSQL storage engine, was released in 2012. It is a powerful solution for high-performance and scalable data storage. It provides support for both B-trees and Log-Structured Merge (LSM) trees, allowing for flexible data access patterns and optimal performance based on the specific workload requirements. Notably, WiredTiger has gained significant recognition for serving as the default storage engine for MongoDB, a widely adopted NoSQL document-oriented database system.

Certain papers like [2] found that in certain workloads, the CPU becomes the primary performance bottleneck when using high-speed storage devices like NVMe SSDs. The researchers conducted extensive experiments using the FIO benchmark tool, comparing the performance of SATA and NVMe SSDs with different CPU configurations. They found that, as the storage performance increased, the CPU became the limiting factor in several scenarios, particularly in high queue depth workloads. Understanding this shift of the performance bottleneck from storage media to the CPU is critical in the present computing environment. This is largely due to the rapid development and increasing adoption of high-speed storage technologies like NVMe SSDs. With such storage technologies delivering faster data transfer rates, the CPU's ability to process the incoming data has become the limiting factor. The performance of the overall system is now primarily dependent on the CPU's processing power, and this shift necessitates a new approach to system optimization.

3. Design

3.1. Machine Configuration

CloudLab [3] provided readily available profiles for both NVMe and SATA disk drives, which were essential for our experiments. This allows us to set up the required configurations quickly and easily without the need for extensive manual adjustments. Secondly, CloudLab operates on a bare-metal hypervisor, which reduces overhead and delivers better performance compared to traditional virtual machines. This ensured a more accurate representation of the performance of our KV stores in real-world scenarios.

To carry out our experiments, we used these CloudLab machines with the following configurations:

m510: This machine is equipped with an 8-core Intel Xeon D-1548 processor operating at 2.6 GHz, 64 GB of RAM, and a 256GB Toshiba M.2 XG3 NVMe SSD with a sequential write throughput of 1572 MB/s.

xl170: This machine features a 10-core Intel Xeon E5-2640v4 processor running at 2.4 GHz, 64 GB of RAM, and a 480GB Intel DC S3520 SATA SSD with a sequential write throughput of 380 MB/s.

3.2. Key-Value Store Configuration and Setup

To obtain accurate results for our tests, it was essential to limit the cache size for both SplinterDB and RocksDB. However, the process for doing so differed between the two databases. While SplinterDB allowed us to programmatically set the cache size, RocksDB deprecated this option, requiring us to use Cgroups to limit the cache size instead. Linux Cgroups, or control groups, help in restricting specific hardware resources, allowing for better control and management of system performance. For SplinterDB, we set the cache size to 1.25 GB using a Cgroup and Splinter's config object (anything less, and it threw a “trunk_room_to_flush” exception), while for RocksDB, we needed to set the cache size to 1 GB. It is also worth mentioning that SplinterDB relies on DIRECT IO to disable OS write caches, while RocksDB utilizes the OS cache. Furthermore, we encountered a specific issue with SplinterDB, as it could only run on an ext4 filesystem. To overcome this limitation, we mounted a new partition with an ext4 filesystem to ensure compatibility and accurate test results.

3.3. YCSB Configuration

We used Yahoo Cloud Serving Benchmark (YCSB) [4] to generate workloads for our experiments. YCSB is a widely-used tool for benchmarking cloud-based databases. The YCSB configurations that we used were 24-byte keys, 100-byte values, and a dataset of 50 million records, resulting in a 7.42 GB text file.

3.4. Data Collection

To accurately measure the time elapsed and calculate the throughput, we implemented code to capture timestamps at two critical points: when the threads began writing and when they completed their writes. We could then calculate the total time elapsed by simply subtracting these timestamps. thereby allowing us to precisely calculate the throughput and assess performance.

For measuring the CPU and Disk Utilization, we used collectl which is a versatile Linux performance monitoring tool used by system administrators and developers to gather detailed information about various system resources. It provides a comprehensive view of system metrics, such as CPU usage, memory, network, disk I/O, and process statistics. It also provides options to control the granularity of the time interval for the measurements. We used collectl as a background process to measure CPU and Disk Utilization at a 0.1 second interval and added an auto-shut off mechanism when the original client driver program stopped. collectl stores the data in a gzip tab data format, making it easy to analyze.

3.5. Data Analysis

To automate data analysis, we utilized a Python script that employs the `pandas` library to read tabular data from Collectl. The script then utilizes the `matplotlib` library to generate plots based on the collected data. By automating this process, we streamline the analysis of data from Collectl and enable efficient visualization for further insights.

4. Implementation

To saturate the disk, we needed to implement multithreading for performing the operations. To do this effectively, we used the Linux 'split' command to partition the YCSB workload text file into equal segments based on the number of threads (e.g., 8 threads would result in 8 files). We then wrote a custom parser utilizing regular expressions to extract the operation, key, and value from the YCSB workload files. Each thread reads one file (from the file splits) and performs the specified operation. The three key-value stores – Splinter, RocksDB, and WiredTiger – each have their unique multithreading implementations, which posed challenges during development. Splinter required the program to register each thread explicitly, whereas WiredTiger demanded a separate 'wt_thread' datatype to handle its threads. Implementing multithreading for these key-value stores in both C and C++ presented a considerable challenge due to the intricacies and differences in their respective approaches. This hands-on experience enabled us to gain valuable insights into the practical aspects of working with these key-value stores and their multithreading capabilities.

5. Results

Threads	NVME Splinter	SATA Splinter	NVME Rocks	SATA Rocks
2	72	74	378	309
3	63	82	287	226
4	60	75	220	180
5	57	70	200	161
6	55	78	167	142

Table 1: Number of Threads vs. Time Elapsed (in s)

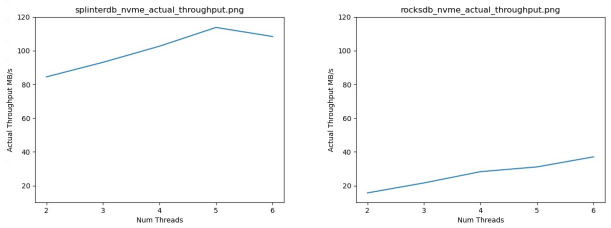

Figure 1: Number of Threads vs Avg. Throughput - SplinterDB, RocksDB on NVMe

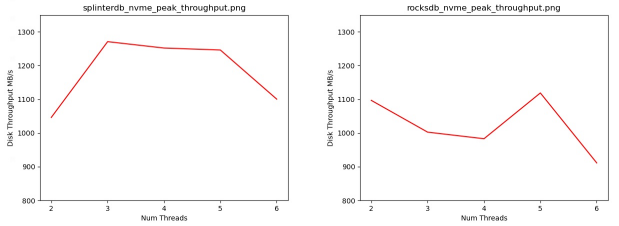

Figure 2: Number of Threads vs Peak Throughput - SplinterDB, RocksDB on NVMe

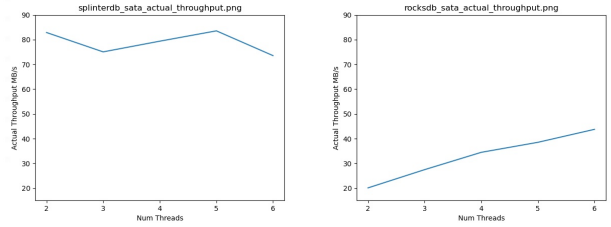

Figure 3: Number of Threads vs Avg. Throughput - SplinterDB, RocksDB on SATA

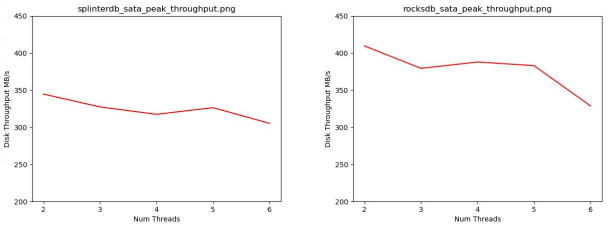

Figure 4: Number of Threads vs Peak Throughput - SplinterDB, RocksDB on SATA

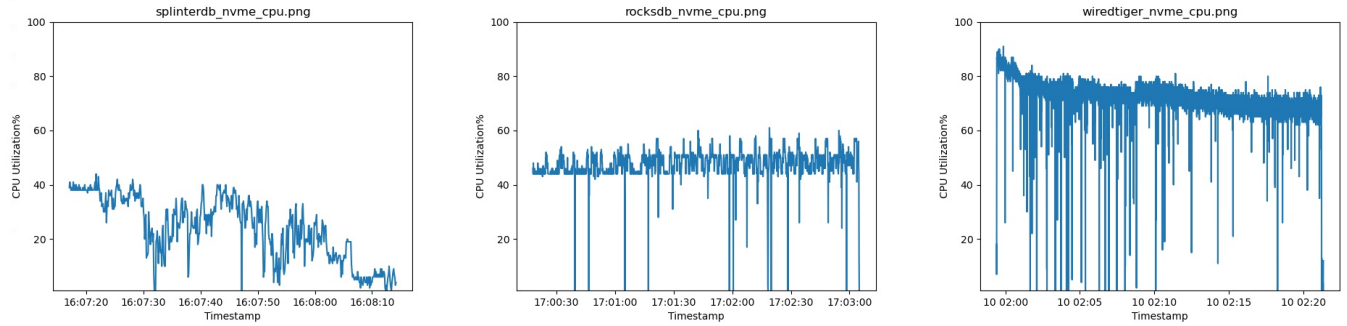

Figure 5: Time vs. CPU Utilization - SplinterDB, RocksDB, WiredTiger

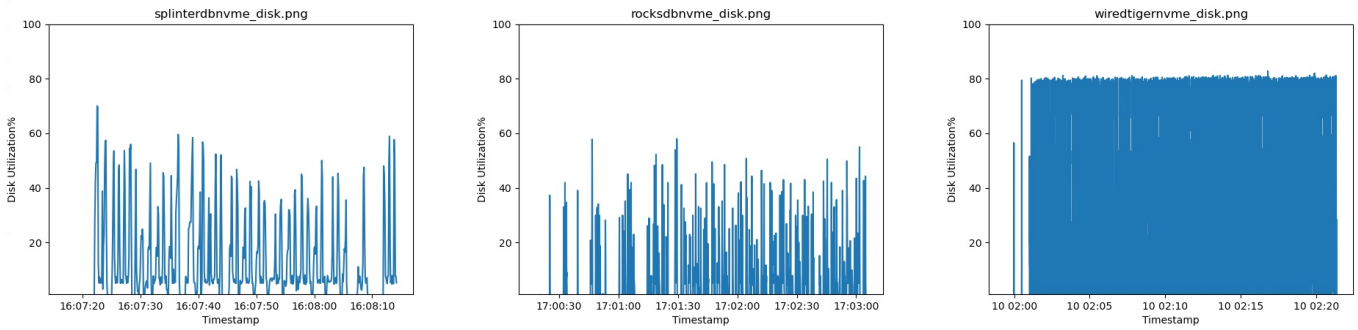

Figure 6: Time vs. Disk Utilization - SplinterDB, RocksDB, WiredTiger

Our primary objective was to reproduce the results from [1] i.e., show the performance gap between SplinterDB and RocksDB on NVMe, and explore if this behavior persists with SATA based SSDs. We made sure to do multiple trials with different thread configurations and average out the results. Table 1 shows the number of threads vs. the time elapsed (for writing the workload) in seconds.

When comparing RocksDB and SplinterDB on NVMe (Figure 1, and Figure 2), we observed that SplinterDB was able to saturate the device bandwidth, achieving an average throughput 3 times higher (113.76 MB/s before factoring in any write amplification) with 5 client threads than RocksDB (37.06 MB/s) with 6 client threads. This suggests that SplinterDB's design is well-suited for NVMe storage, leveraging its parallelism, low-latency, and high bandwidth to deliver superior performance compared to RocksDB on the same storage medium. Similar performance gaps were observed between SplinterDB and RocksDB on SATA (Figure 3 and 4), where RocksDB achieved an average throughput of 43.75 MB/s, while SplinterDB reached an average throughput of 83.56 MB/s, which is approximately twice as high as RocksDB.

Our secondary objective was to compare the hardware utilization (CPU and Disk) of Write optimized DBs on various type of SSDs, to analyze how the evolution of storage devices and interfaces have shifted the bottleneck from disks to CPUs. We focus on two key performance metrics: throughput and CPU utilization.

We compared RocksDB's average throughput on NVMe and SATA, finding that RocksDB achieved 43.75 MB/s on SATA and 37.06 MB/s on NVMe. On the other hand, SplinterDB's average throughput was 83.56 MB/s on SATA and 113.76 MB/s on NVMe.

Figure 5 shows the Time vs. CPU Utilization for a 50 million record workload on NVMe SSD using 6 threads, on the three KV stores. On NVMe SSDs, we observed that SplinterDB had a median CPU utilization of 25%, . This indicates that SplinterDB's CPU usage was moderate, suggesting efficient utilization of computing resources. WiredTiger, on the other hand, had a higher median CPU utilization of 48.0% (Figure 2.3), which is twice more inefficient in CPU usage compared to SplinterDB. WiredTiger exhibited the highest CPU utilization among the three on NVMe SSDs, with a median of 71.85% (Figure 2.1). This suggests that WiredTiger placed a heavier load on the CPU, potentially requiring more computational resources to handle its operations effectively.

In summary, SplinterDB showcased the most efficient CPU utilization across both storage mediums, while WiredTiger placed a heavier load on the CPU, and RocksDB maintained a moderate level of CPU usage. These differences highlight the varying computational demands and optimizations of each database engine on different storage technologies.

Finally, we want to compare the throughput and hardware utilization (CPU and Disk) of LSM-Based KV Stores vs. B+ Trees KV Stores. We compare the performance of WiredTiger and SplinterDB on both NVMe and SATA SSDs, as well as the performance of WiredTiger and RocksDB on the same storage devices.

Overall, we observed that WiredTiger exhibited lower average throughput compared to SplinterDB and RocksDB on both NVMe and SATA SSDs. WiredTiger achieved a maximum average throughput of 4.75 MB/s when using 4 client threads on NVMe. On SATA, it was slightly faster as it reached a maximum throughput of 6.23 MB/s with 4 client threads. Comparing the B+tree based WiredTiger with the LSMTree based RocksDB on NVMe, we found that RocksDB achieved 9X higher average throughput (37.06 MB/s before factoring into write amplification) with 6 client threads than WiredTiger. Similar performance differences were observed on SATA, where RocksDB reached an average throughput at 43.75 MB/s with 6 client threads before factoring into write amplification. SplinterDB demonstrated significantly higher throughput compared to WiredTiger on both NVMe and SATA SSDs, as depicted in Figure 1.5 and 1.6. On NVMe, SplinterDB achieved up to 113.76 MB/s throughput with 5 client threads, while on SATA, it reached a maximum throughput of 83.56 MB/s with 5 client threads. These results indicate that SplinterDB outperforms WiredTiger by approximately 24 times in terms of average throughput.

In Figure 6, we can observe the disk utilization of the three KV stores. Compared to the LSM based RocksDB and SplinterDB, we can see consistently high disk utilization on WiredTiger. There might be multiple reasons behind this including in-place updates in case of B+tree, whereas the write buffering on LSMs and compaction is performed in the background and can be scheduled to run during periods of low disk usage. As LSM Trees buffer incoming writes in memory (using data structures like Memtables or in-memory SSTables) and periodically flush them to disk. This approach reduces the number of random writes and converts them into more efficient sequential writes. This helps lower the disk utilization and might also help improving the lifetime of the disk in the long-run.

In summary, our analysis highlights the superior performance of SplinterDB compared to WiredTiger and RocksDB on both NVMe and SATA SSDs. SplinterDB showcases optimized performance for NVMe hardware, leading to significantly higher throughput. The performance of SplinterDB and RocksDB demonstrate the superiority of write-optimized key-value stores, especially on flash-based SSDs.

6. Related Work

The SplinterDB paper published in 2020, introduced SplinterDB as a STBε tree. It described the optimizations that SplinterDB implemented for NVMe devices that allowed for reduced write amplification, I/O amplification, and CPU costs. The paper also showed performance measurements using YCSB that compared SplinterDB against RocksDB and PebblesDB. The SSDs used in the paper were all NVMe devices.

Toward a Better Understanding and Evaluation of Tree Structures on Flash SSDs [5], published in 2020 documented the benchmarking process on LSM-tree and B+tree data structures: RocksDB and WiredTiger, respectively. This paper highlights the pitfalls that can lead to incorrect measurements of key performance indicators, some of which were applied in the process of our experimentation. The main pitfalls that were considered were: Running short tests - short tests can lead to results that are not representative of the long-term application performance. We made sure to run longer tests so we can more accurately measure performance; Ignoring dataset size: The amount of data stored can affect performance. We made sure to run our tests with consistent dataset sizes/loads so that it wouldn't lead to biased evaluations.

Interestingly, one of the pitfalls that was also mentioned in the paper was “ignoring the effect of the underlying storage technology on performance”. One of the purposes of our experiment is to explore this topic of the effect of storage technology on performance, and attempt to quantify the effect of NVMe vs. SATA on performance. There have also been other benchmarking papers evaluating WiredTiger, RocksDB, SplinterDB. But most of these papers only run them on one type of SSD (usually NVMe), while our work runs benchmarking evaluation on both NVMe and SATA.

7. Conclusion

In this study, we sought to evaluate the performance of key-value stores SplinterDB, RocksDB, and WiredTiger on both NVMe and SATA SSDs. Our primary objective was to reproduce the results from previous studies demonstrating the performance gap between SplinterDB and RocksDB on NVMe and investigate whether this behavior persists with SATA-based SSDs. Our secondary objective was to compare the hardware utilization of these key-value stores on different types of SSDs, analyzing how the evolution of storage devices and interfaces has shifted the bottleneck from disks to CPUs.

Through our experiments, we successfully reproduced the performance gap between SplinterDB and RocksDB on

NVMe and observed similar gaps on SATA SSDs. We found that SplinterDB showcased significantly higher throughput on both storage mediums, indicating that its design is well-suited for both NVMe and SATA hardware. Additionally, we observed that SplinterDB and RocksDB demonstrated the superiority of write-optimized key-value stores, especially on flash-based SSDs.

From this study, we learned that the choice of key-value store has a significant impact on the performance and resource utilization of storage systems. The varying computational demands and optimizations of each database engine on different storage technologies highlight the importance of selecting the appropriate key-value store for specific hardware configurations. Our work also emphasizes the need to consider the effects of storage technology when evaluating the performance of key-value stores. We also learned the volatility of open-source libraries which are in constant development. A great example is SplinterDB. The documentation for SplinterDB does specify it's not ready for production. It was hard to setup out of the box when compared to RocksDB or WiredTiger. Benchmarking it also was comparatively tougher.

Overall, we think that our findings can provide valuable insights for future research and development of key-value stores and storage systems. By understanding the performance and resource utilization characteristics of different key-value stores on various storage mediums, developers and system architects can make better-informed decisions in designing storage systems optimized for their specific hardware and use cases.

The code for this project is available on a public GitHub Repository: https://github.com/rahulsai1999/cs598SS.

References

[1] A. Conway *et al.*, "SplinterDB: Closing the Bandwidth Gap for NVMe Key-Value Stores," in *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, in USENIX ATC'20. USA: USENIX Association, 2020.

[2] Q. Xu *et al.*, "Performance analysis of NVMe SSDs and their implication on real world databases," in *Proceedings of the 8th ACM International Systems and Storage Conference*, New York, NY, USA: ACM, May 2015, pp. 1–11. doi: 10.1145/2757667.2757684.

[3] R. Ricci, E. Eide, and C. Team, "Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications," ; *login: the magazine of USENIX & SAGE*, vol. 39, no. 6, pp. 36–38, 2014.

[4] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, in SoCC '10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 143–154. doi: 10.1145/1807128.1807152.

[5] D. Didona, N. Ioannou, R. Stoica, and K. Kourtis, "Toward a better understanding and evaluation of tree structures on flash SSDs," *Proceedings of the VLDB Endowment*, vol. 14, no. 3, pp. 364–377, Nov. 2020, doi: 10.14778/3430915.3430926.