From Flash to 3D XPoint: Performance Bottlenecks and Potentials in RocksDB with Storage Evolution

Yichen Jia
Computer Science and Engineering
Louisiana State University
yjia@csc.lsu.edu

Feng Chen
Computer Science and Engineering
Louisiana State University
fchen@csc.lsu.edu

***Abstract*—Storage technologies have undergone continuous innovations in the past decade. The latest technical advancement in this domain is *3D XPoint memory*. As a type of Non-volatile Memory (NVM), 3D XPoint memory promises great improvement in performance, density, and endurance over NAND flash memory. Compared to flash based SSDs, 3D XPoint based SSDs, such as Intel's Optane SSD, can deliver unprecedented low latency and high throughput. These properties are particularly appealing to I/O intensive applications. Key-value store is such an important application in data center systems.**

This paper presents the first, in-depth performance study on the impact of the aforesaid storage hardware evolution to RocksDB, a highly popular key-value store based on Log-structured Merge tree (LSM-tree). We have conducted extensive experiments for quantitative measurements on three types of SSD devices. Besides confirming the performance gain of RocksDB on 3D XPoint SSD, our study also reveals several unexpected bottlenecks in the current key-value store design, which hinder us from fully exploiting the great performance potential of the new storage hardware. Based on our findings, we also present three exemplary case studies to showcase the efficacy of removing these bottlenecks with simple methods, achieving a performance improvement by up to 18.8%. We further discuss the implications of our findings for system designers and users to develop schemes in future optimizations. Our study shows that many of the current LSM-tree based key-value store designs need to be carefully revisited to effectively incorporate the new-generation hardware for realizing high-speed data processing.

I. Introduction

With the rapid growth of Internet services, data center systems need to handle a huge volume of data at a very high processing rate [31]. This grand trend demands a non-stopping, fast-pace evolution of the storage subsystems, incorporating cutting-edge hardware and software technologies and optimizing the entire system in a cohesive manner.

On the hardware side, NAND flash based SSDs have already been widely adopted in today's data centers [32]. Although compared to conventional disk drives, flash SSDs can deliver higher performance and better power efficiency, the well-known slow random write and limited lifetime issues still remain a non-negligible concern in large-scale systems.

More recently, Intel's Optane SSD [16], which is built on 3D XPoint [15], a type of Non-volatile Memory (NVM) technology, has received increasingly high interests in the industry. Unlike flash SSD, 3D XPoint SSD uses resistance-based recording material to store bits, enabling it to provide much lower latency and higher throughput. Most importantly, 3D XPoint SSD significantly alleviates many long-existing concerns on flash SSDs, such as the read-write speed disparity, slow random write, and endurance problems [26], [42]. Thus 3D XPoint SSD is widely regarded as a pivotal technology for building the next-generation storage system in the future.

On the software side, in order to accelerate I/O-intensive services in data centers, RocksDB [11], which is the state-of-art Log-structured Merge tree (LSM-tree) based key-value store, has been widely used in various major data storage and processing systems, such as graph databases [8], stream processing engine [1], and event tracking systems [29]. They all rely on RocksDB as the storage engine to provide high-speed queries for key-value workloads.

However, since RocksDB is particularly optimized for flash-based SSDs, new challenges would naturally emerge as we transit to 3D XPoint SSD in the future. Considering the architectural differences between flash SSD and 3D XPoint SSD, it is a highly interesting and practical question—*Is the current design of LSM-tree based key-value store readily applicable to the new 3D XPoint SSD?*

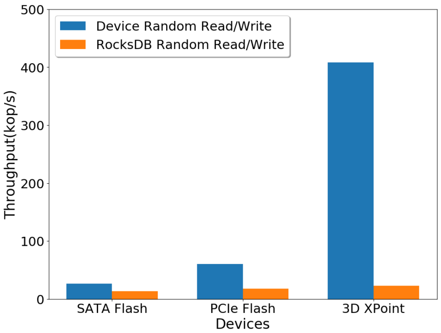

Fig. 1: A motivating example—performance improvement of RocksDB workloads on Intel Optane SSD

To have a glimpse of the potential problem, we show a motivating example in Figure 1. We use Intel Open Storage Toolkit [25] to generate 4KB random requests with 8 threads and read/write ratio being 1:1 to access the first 10GB storage space on a 280GB Intel Optane 900P SSD. The raw I/O throughput increases from 26 kop/s on an Intel 530 SATA SSD to 408 kop/s on the 3D XPoint SSD, which is a speedup of 15.7 times. Then we benchmark RocksDB with 4KB requests following the *randomreadrandomwrite* distribution and read/write ratio being 1:1. The key-value I/O throughput on RocksDB increases from 13 kop/s to 23 kop/s, which is an increase of only 76.9%. It indicates that impediments exist in the current design, hindering us from effectively exploiting the high processing capacity of the new hardware.

In this paper, we present a set of comprehensive experimental studies to quantitatively measure the performance impact of 3D XPoint SSD on RocksDB. In the experiments, we have identified several unexpected performance bottlenecks and also gained important insight for future deployment of RocksDB in data center systems. To our best knowledge, this paper is the first study on the performance behavior of RocksDB on 3D XPoint SSD.

For our experiments, we use *db_bench* to generate various types of key-value workloads to conduct extensive experiments and characterize RocksDB. Our purpose is not to compare the absolute performance difference of RocksDB running on different storage devices. Rather, we desire to obtain insightful understanding on the effect of the unique properties of the new-generation storage hardware on RocksDB performance, and to gain system implications for designers to effectively integrate RocksDB into data center systems. We have made the following contributions in this paper:

- This is the first work studying the performance behavior of RocksDB on 3D XPoint SSD. We have identified several important bottlenecks in RocksDB through experiments and analysis, such as the throttling mechanism, the Level-0 file query overhead, the read/write interference, etc.
- Leveraging our findings, we have also designed and implemented three exemplary case studies to showcase the efficacy of removing the bottlenecks with simple methods on RocksDB, such as mitigating the near-stop situation for workloads with periodic write bursts, dynamic Level-0 file management that improves the throughput by up to 13%, and a simulated NVM logging approach that reduces the 90th percentile write tail latency by up to 18.8%.
- Based on our observations, we have also discussed the related system implications as future guidance to optimize RocksDB on 3D XPoint SSD and to integrate the new hardware into large-scale systems and date centers.

The rest of this paper is organized as follows. Section II introduces the background. Section III and Section IV present the experimental setup and results. Section V gives three case studies. In Section VI, we summarize our key findings and discuss the important implications to users and system designers. Related work is presented in Section VII. The final section concludes the paper.

II. Background

Flash Memory vs. 3D XPoint Memory. NAND flash is a type of EEPROM devices. A flash memory chip typically consists of several *planes*. Each plane is composed of thousands of *blocks*, each of which is further divided into *pages*. There are three major technical constraints for flash memory. First, a read operation is typically fast (e.g., 50 μs), while a write is relatively slow (e.g., 500 μs). Second, a programmed (written) page cannot be overwritten again until the whole block is erased. An erase is a time-consuming operation (e.g., 2.5 ms) and conducted in the unit of blocks. Third, the flash pages of a block must be programmed in a sequential manner. More details can be found in prior studies [3], [6], [18].

3D XPoint is a type of Non-volatile Memory (NVM) technology [15], [26]. Unlike NAND flash memory, 3D XPoint memory is byte addressable, meaning that it can be accessed in a fine granularity similar to DRAM. Since 3D XPoint memory does not need an erase operation before write, the read/write speed disparity problem is significantly alleviated. According to Micron, compared to NAND flash, 3D XPoint provides up to 1,000 times lower latency and multiple orders of magnitude greater endurance [26]. Intel's Optane SSD, which is recently available on the market, is built on 3D XPoint memory. In this paper, we use Intel's Optane 900P SSD in our experiments.

LSM-tree based Key-value Store. Many modern key-value data stores are built on *Log-structured Merge tree* (LSM-tree) [2], [9], [11], [13], which is optimized for handling intensive update operations. Typically, the key-value data are stored in both memory and storage devices. A set of *Memtables* are maintained in memory to collect incoming writes first and then flush to the storage device (e.g., a disk or an SSD) in *Sorted Sequence Table* (SST) data files. The related metadata information about SSTs is stored in *Manifest* files. SSTs are logically organized in multiple levels of increasing size, from Level 0 (L0) to Level N (LN) (see Figure 2). Except at Level 0, where the SSTs can have overlapping key ranges, the other SSTs at each level (L1 to LN) must have non-overlapping key ranges in a sorted manner.

A background merging process, called *compaction*, routinely runs to remove the deleted and obsolete items. In particular, when the number of L0 files exceeds a predefined threshold, multiple L0 files merge with the L1 files that have overlapping key ranges, generating new L1 files and discarding the input L0 and L1 files. The compaction processes at the other levels are similar. Involving heavy I/O and computation overhead, compaction is considered as the main performance bottleneck in LSM-tree based key-value stores.

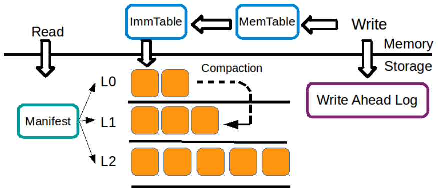

Fig. 2: An illustration of the LSM tree structure

Upon arrival of a write request, the update is first written into a *Write-ahead Log* (WAL) for crash recovery. Then the update is accommodated in a buffer in memory, called *Memtable*. If the Memtable is filled up, it is switched to a read-only *Immutable Memtable*. A new Memtable is allocated for holding the subsequent requests. A background thread periodically flushes the Immutable Memtables to persistent storage.

For handling a read request, we first check the Memtables, then the Immutable Memtables, and finally look up the key in the on-storage SSTs, starting from L0 to LN, until the requested item is found. To reduce the I/O cost, techniques, such as *Bloom Filters*, are used to speed up the queries. The block cache in RocksDB and the OS buffer cache can also eliminate unnecessary storage I/O operations.

RocksDB. Compared to LevelDB [13], RocksDB is a popular LSM-tree based key-value store particularly optimized for flash SSDs. It employs multiple schemes to exploit the properties of flash SSD for better performance, such as its rich internal parallelism resources [7]. For example, RocksDB adds multiple *column families* to logically partition the database and group associated keys together; multiple Immutable Memtables are used to avoid write stalls; the compaction process is multi-threaded; separate thread pools with different priorities are used for flushing and compaction. After years of tuning, RocksDB has optimized its performance with flash storage, making it highly popular in the industry [4], [9], [27].

III. Methodology

Our experiment system is a two-socket Intel W2600CR server. It is equipped with 16 cores on two Intel(R) Xeon(R) E5-2690 2.90GHz processors and 128 GB memory. An Intel 530 Series SATA Flash SSD, an Intel 750 PCIe Flash SSD, and an Intel Optane PCIe 3D XPoint SSD are used as three representative storage devices. We use Ubuntu 14.04 with Linux Kernel 4.4.0, Ext4 file system, and RocksDB 5.17.0 in our experiments.

The data set used in our experiments is around 100 GB. We accordingly set the available physical memory space to be 8 GB during the system boot time, which is about 8% of the data set size. Based on prior study about I/O characterization in large-scale data centers [12], we configure the workload with different read/write ratios and the value size being 1 KB. We use *db_bench*, RocksDB's default benchmarking tool, to generate workloads following the *randomreadrandomwrite* distribution in each experiment. If not otherwise specified, each experiment runs for 300 seconds, which is reasonably long enough to show the performance trend.

IV. Bottleneck Identification

As reported by Yahoo! [31], the percentage of write operations in emerging workloads, such as cloud computing, mobile devices, and social networks, has significantly increased at an unprecedented pace. In 2010, the workloads contained about 10-20% writes, which increased to nearly 50% in 2012. The high insertion ratio contributes to the quickly increasing popularity of RocksDB [5], [38], since its multi-level, append-only structure is highly suitable for handling intensive traffic of updates on flash device. In this work, we conduct a comprehensive measurement to study the RocksDB performance with 3D XPoint SSD, particularly to identify the corresponding bottlenecks on the new-generation hardware.

A. Throttling Mechanism

The memory component of LSM-tree is for two purposes. First, an entry can be inserted (SET) into the memory-resident memtable without involving any I/O cost. Only large, batched I/Os can be seen at the storage level. Second, GET requests to the entries in memory can be served quickly without incurring an I/O to the storage device.

Although the in-memory memtable structure can buffer I/O requests and eliminate data retrievals from the slow on-storage component, it raises cost and power consumption concerns for a large-scale deployment. Thus, in practice, users and system administrators often impose a limit on the number of in-memory Memtables (2 by default) and on-disk Level-0 files (36 by default) in RocksDB. When the number of Level-0 files hits a threshold, a *write throttling* mechanism is triggered to purposefully slow down the incoming request traffic to save the memory used for buffering the I/Os, until the background process makes enough space in Level-0 by merging and deleting Level-0 files. As the insertion ratio of data center workloads keeps increasing, the throttling mechanism would be more frequently triggered, posing huge overhead on the system performance. In this section, we analyze the throttling mechanism and its impact by increasing the insertion ratio from 0% to 100% with 4 parallel processes.

Finding #1. On both SATA and PCIe flash SSD, as Figure 3 shows, the throughput of RocksDB increases as the insertion ratio increases, because of the reduced number of expensive READ requests. For example, on the PCIe flash SSD, the throughput increases from 32 kop/s to 41.3 kop/s, when the insertion ratio increases from zero to 100%. As Figure 6 and Figure 7 show, READ latency is significantly longer on flash SSDs than 3D XPoint SSD. In particular, with a high insertion ratio (90% write), the 90th percentile read tail latency on 3D XPoint SSD is only 251 μs, in contrast to that on the SATA flash SSD (839 μs); the 90th percentile write tail latency is 26 μs, which is close to that on the SATA flash SSD (28 μs). Thus, the reduced expensive READ operations bring more benefits to the throughput on the flash SSDs.

On 3D XPoint SSD, interestingly, Figure 3 shows an opposite trend—the throughput decreases from 115 kop/s to 45 kop/s, which is close to the PCIe flash SSD. When insertion ratio continues increasing, the throughput difference becomes less significant, despite the hardware difference. To explain the results, we show more details in Figure 4 and 5. When WRITE operations are dominant, the triggered write throttling mechanism would introduce delays for insertion operations, causing performance fluctuation. As shown in Figure 4 and 5, the throughput variation on 3D XPoint SSD is clearly evident. The throttling periodically happens, pulling down the throughput. For example, when insertion ratio is 90%, the throughput drops from 169 kop/s to as low as 3 kop/s when throttling happens on 3D XPoint SSD.

Thus, opposite to our common expectation that 3D XPoint SSD can always provide significantly better performance than flash SSD, we find that such a performance benefit is workload

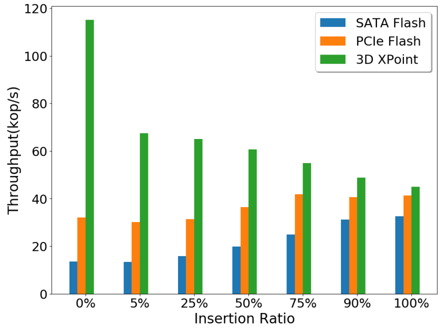

Fig. 3: Throughput vs. Insertion Ratio

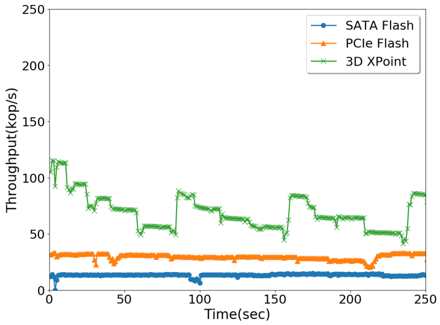

Fig. 4: Throughput (5% Write)

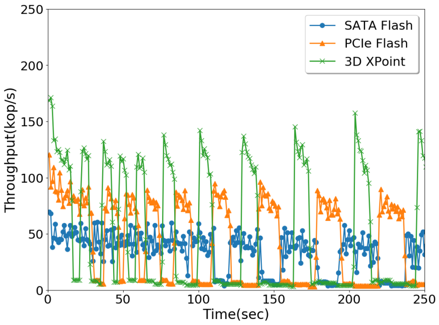

Fig. 5: Throughput (90% Write)

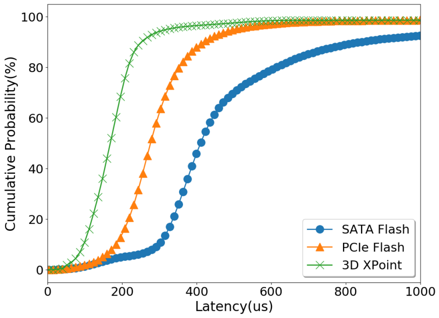

Fig. 6: Read (90% Write)

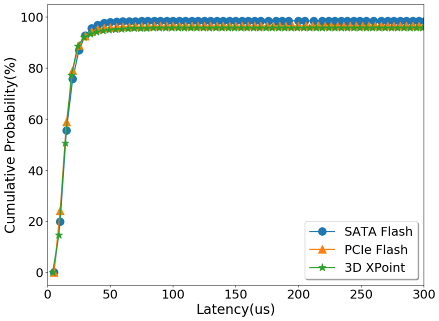

Fig. 7: Write (90% Write)

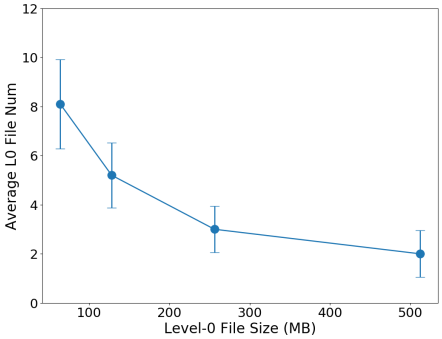

Fig. 8: Num. of Level-0 Files vs. File Size

dependent (more significant with a lower insertion ratio). For a write-intensive workload, the current throttling mechanism in RocksDB, which is heavily optimized for flash memory, drags down the performance of RocksDB on 3D XPoint SSD to the level of running on a regular flash SSD.

Analysis #1. Algorithm 1 briefly describes the write control process in RocksDB. The write process is put to sleep when a delay is needed (e.g., Level-0 slowdown threshold is hit). The $delayed\_write\_rate$ is adjusted to make sure the amount of data being processed at each interval during compaction roughly constant and stable (i.e., ideally, $Prev\_Bytes$ equals $Esti\_Bytes$).

We use a simple model to explain and estimate the decrease of throughput. Let λ_a and λ_s, respectively, denote the application level throughput, which is the performance perceived by applications, and the system level throughput, which is the processing capacity of the key-value system; $refill\_interval$ denotes the minimum of an injected delay period. From Algorithm 1, we can see λ_a converges to $delayed\_write\_rate$ to guarantee that the request arrival rate roughly equals to the processing rate. For a time period t during which one write finishes (i.e., the median latency), we have:

$$\lambda_a \times (refill\_interval + t) = \lambda s \times t \tag{1}$$

Based on Equation 1, we can have:

$$\lambda_a = \frac{t}{refill\_interval + t} \times \lambda_s \tag{2}$$

According to our measurement on the 3D XPoint SSD, λs, the background processing capacity of RocksDB when compaction happens, is 190 kop/s, and t, the median write latency, is 15 μs, so we have:

$$\lambda_a = \frac{15}{1024 + 15} \times 190kop/s = 2.74kop/s$$

For the SATA SSD, λs and t are measured to be 130 kop/s and 15 μs, respectively, so we have:

$$\lambda_a = \frac{15}{1024 + 15} \times 130kop/s = 1.88kop/s$$

The above calculated results are close to our measurement data in Figure 5. We can see that when the throttling process is triggered, the application level throughput would degrade to a similarly low level, disregarding the hardware differences.

Discussion #1. Due to the throttling mechanism, the achievable application level throughput is much lower than expected, even though the underlying storage hardware is highly capable. The throttling process is such a case that exactly showcases the potential problem—if we blindly operated the same throttling strategy on 3D XPoint SSD in the same way as that on slower flash devices, the hardware performance benefit enabled by the new technology would become negligible. As we evolve to the new-generation storage technology, many such optimizations should be carefully reconsidered.

B. Level-0 File Query Overhead

In the LSM-tree structure, the keys in a Level-0 SST files are not sorted. Thus the key ranges could be overlapped. Typically, it requires to search multiple Level-0 SST files and possibly other-level files to find the key-value item. This results in a significant *read amplification* problem on flash based devices.

Algorithm 1 WRITE CONTROL PROCESS

```
Dec=0.8, Inc=1.25
$delayed\_write\_rate = usr\_defined\_value$
$refill\_interval$ =1024 $\mu$s
$num\_bytes = last\_batch\_group\_size$
procedure WRITE($num\_bytes$)
    if $need\_delay$ then
        if $Prev\_Bytes <= Esti\_Bytes$ then
            $delayed\_write\_rate* = Dec$
        else
            $delayed\_write\_rate* = Inc$
        end if
        delay = DELAYWRITE($num\_bytes$)
        sleep(delay)
    end if
    Process(WRITE)
end procedure
function DELAYWRITE($num\_bytes$)
    $time\_slice = time\_now - last\_refill\_time$
    $bytes\_refilled = time\_slice * delayed\_write\_rate$
    if $bytes\_refilled > num\_bytes$ then
        if $time\_slice > refill\_interval$ then
            return 0
        end if
    end if
    $single\_ref = refill\_interval * delayed\_write\_rate$
    if $bytes\_refilled + single\_ref > num\_bytes$ then
        return $refill\_interval$
    else
        return $num\_bytes/delayed\_write\_rate$
    end if
end function
```

As the storage becomes dramatically faster with the 3D XPoint technology, it is worth studying if this read amplification problem could become less significant.

We desire to study the performance impact with different number of Level-0 files. In the current RocksDB, a challenge is that the numbers of Level-0 files cannot be statically set. In fact, it dynamically changes during runtime. We find that the number of Level-0 files is largely determined by the file size. To show such a relationship between file size and the number of Level-0 files, we vary the Level-0 file size from 32MB to 512MB with 4 concurrent processes and 1:1 READ/WRITE ratio and show the results in Figure 8. By setting different Level-0 file sizes, we can roughly control the desired numbers of files generated at Level 0.

Finding #2. Figure 9 shows the throughput with different numbers of Level-0 files. Opposite to our expectation, we find that the number of Level-0 files has a significant impact on the performance with 3D XPoint SSD, which is even more pronounced compared to flash SSDs. In specific, the throughput on the PCIe flash SSD decreases by only 12.3%, from 41.5 kop/s to 36.4 kop/s by increasing the number of files from 2 to 8. In contrast, the throughput on 3D-XPoint SSD decreases even more, from 86.4 kop/s to 69.2 kop/s, which is 19.9%, meaning that the negative impact of the read amplification problem is more significant on faster storage.

To investigate this result, a closer look at READ requests further reveals where the overhead originates from. For READ requests, fewer Level-0 files are generally beneficial, on all the three devices. For example, on 3D XPoint SSD, the 90th percentile tail latency of READ requests is 134 μs when the number of Level-0 files is 8, and it drops to 101 μs when the number of Level-0 files decreases to 2, as Figure 10 shows.

This overhead difference can be explained by the lookup process. For illustration, we use Figure 11 to show the different querying processes with large and small Level-0 files, both containing the same number of items. Note that all the small files whose key ranges cover the desired key have to be searched until the key is found. The reason is as follows.

Assume the small file case has k small files, each with size N, while the large file case has a large file with size $k \times N$. Because the files in Level-0 are organized by Skiplist [28], if we split the large file into k small files, the lookup complexity would increase from $O(log(k \times N))$ to $O(k \times log(N))$, which is equal to $O(log(N^k))$. Since a lookup operation in a large Level-0 file on 3D XPoint is not significantly longer (e.g., 9.7 μs for 256MB vs. 8.5 μs for 32MB), the number of involved Level-0 files becomes the dominant factor. As so, maintaining more Level-0 files would lead to a longer READ latency.

Analysis #2. Our results show that it is beneficial to maintain a small number of files with a relatively large size. However, it does not mean that we should maintain one single huge Level-0 file. In the current system design, the mutable memtable becomes immutable when it is full. Then another new mutable memtable is allocated to continue serving incoming requests. The immutable table is further flushed to Level 0. Thus a larger Level-0 file, meaning a larger mutable memtable, will cause a longer latency for WRITE operations.

As shown in Figure 12, the 90th percentile tail latency of WRITE operations increases from 25 μs to 31 μs, when the SST file size increases from 64MB to 256MB on the SATA flash SSD. It is because WRITE operations are first accumulated in the mutable memtable and then it switches to immutable when full and is further flushed to disk. As the complexity of insertion to a skiplist is $O(log(N))$, a larger memtable would cause a longer latency for WRITE operations.

According to the analysis above, fewer but larger Level-0 files would result in smaller READ latencies due to a smaller number of files for search, but a very large Level-0 file size would cause a longer WRITE latency. Such an effect is more evident on 3D XPoint SSD than that on flash based ones, since 3D XPoint SSD is much faster and the overhead involved in Level-0 file queries contributes a more significant portion to the overall request processing time.

Discussion #2. Unlike our expectation, as the underlying storage media becomes faster, the role of the memory components and Level-0 files of RocksDB would become even more important. The memory component management, as well as

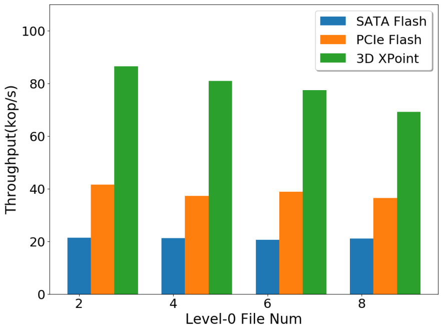

Fig. 9: Throughput vs. Num. of L0 Files

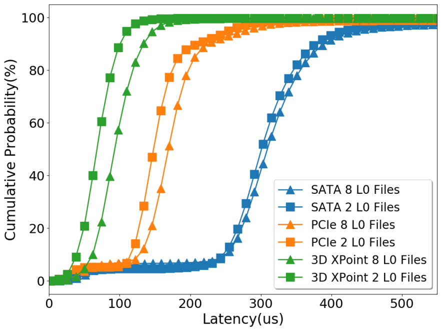

Fig. 10: Read Latency vs. Num. of L0 Files

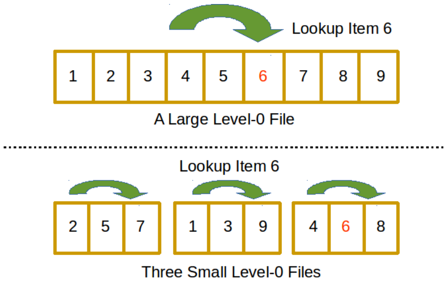

Fig. 11: Large File vs. Small Files

the Level-0 file number setting and the querying mechanism, would have a greater impact on the overall performance.

C. Parallelism and Interference

Due to internal parallelism, flash SSDs are particularly suitable for parallel I/O processing. RocksDB supports concurrent writers and write batching to provide high throughput performance. However, the parallel read and write operations may interfere each other [7]. We benchmark RocksDB by varying the number of processes from 1 to 32 with READ/WRITE ratio being 1:1.

Finding #3. Figure 13 shows that the throughput increases significantly as the number of threads increases on all three devices. For example, the throughput increases from 35.4 kop/s to 79.5 kop/s on 3D XPoint SSD by changing the parallelism degree from 1 to 32.

However, READ and WRITE requests behave differently as storage device evolves. As for READ requests, as Figure 14 shows, read latency on 3D XPoint SSD is significantly smaller than that on the flash SSDs. With 32 threads, the 90th percentile tail latency on the 3D XPoint SSD is 335 μs, which is 76% smaller than that on the SATA flash SSD (1.4 ms).

The WRITE requests show opposite results. The 90th percentile tail latency on the 3D XPoint SSD (440 μs) is significantly longer than that on the SATA flash SSD (47 μs) when the thread number is 32, as shown in Figure 15.

The pipelined write process, which is used in RocksDB to improve concurrent write throughput, can explain these phenomena. By default, RocksDB keeps one single write thread queue for concurrent writers. A thread becomes the leader of the write batch group, when it reaches the head of the queue, and is responsible for flushing the updates for the batch group. The whole process is described in Algorithm 2.

3D XPoint SSD provides a high processing speed for READ requests, which unfortunately places a high pressure on the concurrent WRITE requests. A result is that more awaiting writes are accumulated during the same period of time. As shown in Figure 16, the average number of waiting threads on 3D XPoint SSD is evidently larger than that on flash SSDs with 32 threads. Each writing thread has to wait for the previous time-consuming flushing operations to complete, causing a queuing effect. More waiting threads would lead to a longer waiting time for WRITE requests on 3D XPoint SSD than on flash SSD, as Figure 15 shows.

Algorithm 2 PIPELINED WRITE PROCESS

```
procedure PIPELINEDWRITEIMPL
    if writer.state==$Memtable\_Leader$ then
        for writers in Group do
            writer.state=$Memtable\_Writer$
        end for
    else if writer.state==$Memtable\_Writer$ then
        Write()
    else
        Await()
    end if
    PickNewLeader()
end procedure
```

Discussion #3. Opposite to the traditional understanding that 3D XPoint SSD always provides low latency services than flash SSD, WRITE requests could take longer to complete in mixed workloads. With a high arrival rate of requests and intensive READ/WRITE interference, software designs in RocksDB, such as pipelined write process, would play a significant role for request latency. As a result, more accumulated WRITE requests would result in a longer processing time.

D. Logging

LSM-tree based key-value store usually manages a Write Ahead Log (WAL) for data recovery and system consistency. Though being crucial, logging mechanisms can incur high performance overhead. With a high-speed 3D XPoint SSD, it is worth studying if the involved overhead could be weakened. We study their impact by enabling and disabling logging mechanisms, using a workload with READ/WRITE being 1:9.

Finding #4. According to Figure 17, WAL still has a significant impact on write performance, even with a much faster 3D XPoint SSD. The 90% tail latency of WRITE operations is reduced from about 54 μs to about 22 μs, if disabling the WAL mechanism on 3D XPoint SSD. It is because when a write request is issued, the WAL update is first written to the write buffer. Then the real write operation is executed in the in-memory data structure, memtable. The WAL and memtable are flushed to disk asynchronously and

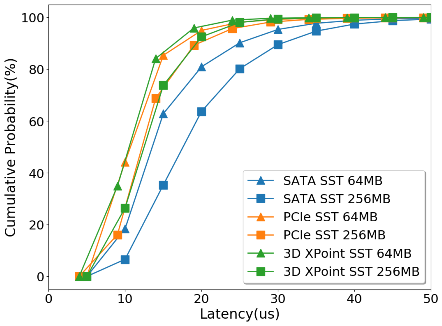

Fig. 12: Write Latency vs. SST File Size

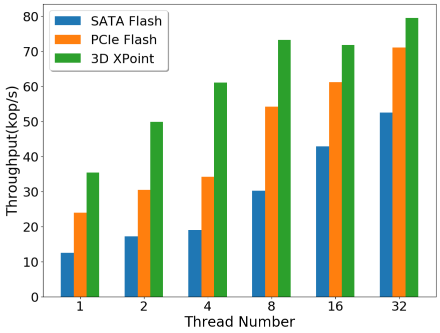

Fig. 13: Throughput vs. Parallelism

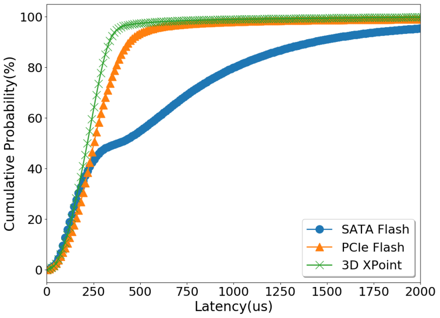

Fig. 14: Read Latency (32 Threads)

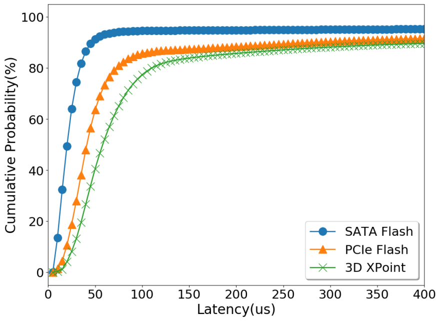

Fig. 15: Write Latency (32 Threads)

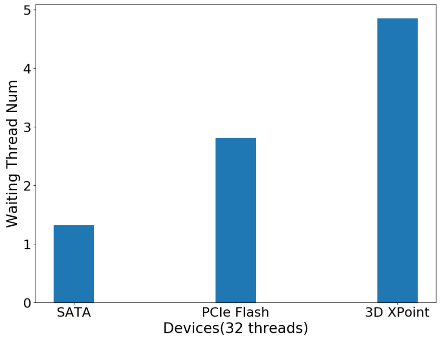

Fig. 16: Waiting Threads vs. Devices

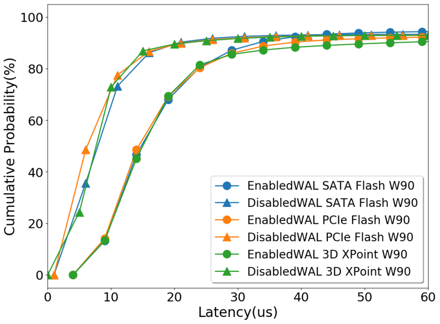

Fig. 17: Write Latency vs. WAL

separately later. Thus the device differences are diminished. Our results show that logging is still an important factor hindering the performance of RocksDB on all devices.

Discussion #4. WAL poses great performance penalty, even on 3D XPoint SSD. We need to consider solutions to speed up this process. Further optimization on this costly process remains an important demand to achieve stable performance, even on high-speed 3D XPoint SSD.

V. Case Studies

In this section, we present three case studies to illustrate how to optimize the performance of RocksDB by overcoming the bottlenecks discussed above. It is worth noting that these pilot solutions are not intended to be fully optimized designs but to serve as examples to showcase the potential optimizations that could be enabled based on our findings.

A. Removing Near-Stop Situation

The original Level-0 file management mechanism works as follows. When the number of Level-0 files reaches a compaction trigger threshold, the background compaction process is launched to merge the files that have overlapping keys in Level-0 and the lower levels. When the number of Level-0 files reaches a slowdown threshold, a throttling process as described in Algorithm 1 will be triggered. In extreme cases, when the number of Level-0 files reaches *stop_threshold*, a stopping process will be launched to constrain the Level-0 size according to the user's requirements.

Such a simple throttling mechanism is particularly detrimental when the system undergoes a periodically appearing "flash of crowd" situation, in which write bursts periodically appear and pull the system into a near-stop (under 10 kop/s) situation. The real-time throughput, as we show in Section IV-A, can be as low as 3 kop/s, rendering the system almost unusable.

In order to remove the near-stop situation, we propose a *two-stage throttling* method to avoid a sharp performance drop. This is particularly beneficial to workloads with periodic write bursts, which is common in large-scale systems [22], [37].

- *Stage 1: Slight Throttling*. When the number of Level-0 files reaches the user-defined $slowdown\_threshold$, throttling is activated to conservatively reduce the write rate to no lower than the maximum acceptable $delayed\_write\_rate$.
- *Stage 2: Aggressive Throttling*. When the number of Level-0 files continues to grow and reaches the second-level throttling threshold, which is defined as $(slowdown\_threshold + stop\_threshold)/2$, the more aggressive throttling, Algorithm 1, will be applied to slow down the incoming traffic.

To evaluate the effect of our proposed two-stage throttling method, we benchmark RocksDB using a workload with READ/WRITE ratio being 1:1. This workload has a periodic write burst (READ/WRITE ratio being 1:9) lasting for 25 seconds per minute. As Figure 18 shows, the throttling throughput of the original design is about 9 kop/s from 135s to 142s, 12 kop/s from 151s to 163s. In contrast, we can hardly see near-stop periods with our proposed approach, meaning that even such a simple approach can effectively remove the near-stop situation for workloads with periodic write bursts.

B. Dynamic Level-0 Management

As Section IV-B shows, the number and size of Level-0 files have a great impact on the performance of RocksDB. In specific, having fewer Level-0 files would reduce the READ

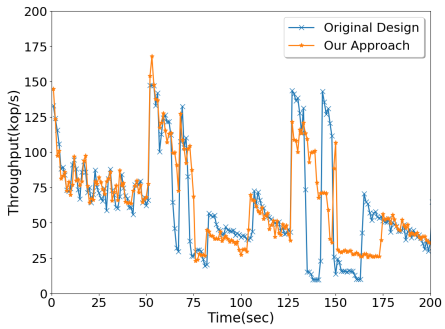

Fig. 18: Throughput vs. Time

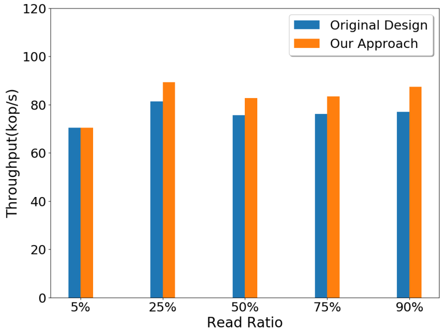

Fig. 19: Throughput vs. Read Ratio

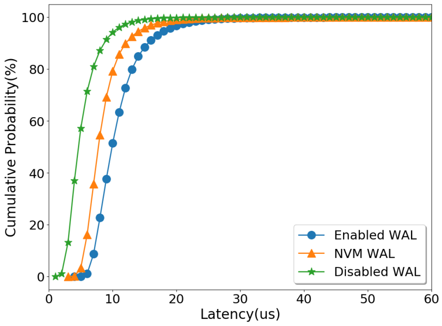

Fig. 20: Latency vs. Logging

latency due to a smaller number of files for searching, while a smaller Level-0 file size would reduce the WRITE latency due to the reduced insertion overhead to a smaller skiplist. Therefore, there exists a tradeoff point between the two factors, depending on the workloads (READ or WRITE intensive).

Assuming the aggregate volume of Level-0 files is constant, we present a simple dynamic Level-0 management method that optimizes the number of Level-0 files and the file size based on the READ/WRITE ratio measured online.

The RocksDB is initialized to throttle writes when the number of files at Level-0 reaches 24. Then we measure the READ/WRITE ratio during runtime. When the workload is observed to be WRITE intensive, we configure the Level-0 to have smaller, yet more (e.g., 24 in our example) files; When the workload is READ intensive, we configure the Level-0 to have larger, yet fewer (e.g., 6 in our example) files.

To evaluate the effect of our proposed dynamic Level-0 management method, we benchmark RocksDB with the default configuration by varying the READ ratio from 5% to 90%. In our example, we empirically tag the workloads to be WRITE intensive, if write operations account for more than 25%. According to Figure 19, our proposed approach can improve the throughput of RocksDB in most cases. For example, when the READ ratio is 90%, our approach can improve the throughput from 77 kop/s to 87 kop/s, which is an improvement of 13%. When the read ratio is 5%, these two approaches achieve similar throughput.

C. Reducing Logging Overhead

As we show in Section IV-D, WAL incurs noticeable overhead to RocksDB. In order to reduce the logging overhead, a potential solution is to separate WAL out and relocate it to a faster device, such as byte-addressable non-volatile memory (NVM). Since the size of WAL is quite small, it is reasonable to deploy a rather small NVM device to accumulate WAL updates very quickly.

To evaluate the effect of our proposed NVM logging method, we emulate NVM by using Linux `tmpfs` in DRAM. We benchmark RocksDB with the default configuration by setting the insertion ratio to be 50%. Figure 20 shows that logging in NVM can effectively reduce the overhead. For example, the 90th percentile write tail latency decreases from about 16 μs to 13 μs, which is an improvement of 18.8%. However, compared with disabling WAL, logging in NVM still incurs noticeable performance penalty. It indicates that the WAL overhead cannot be totally removed by using such a simple approach and a more sophisticated solution is needed to fully address it.

VI. System Implications and Discussions

In this section, we present important implications for system designers to effectively deploy RocksDB and optimize the database management on 3D XPoint SSD.

• **Reads**. For READ operations, the new 3D XPoint SSD provides much better performance than flash SSDs in most cases, which is as expected. We also find that other factors, such as Level-0 file management, can significantly affect READ performance. In fact, fewer Level-0 files means less search overhead, which is proven to be noticeably beneficial for READ performance. In addition, since the number of Level-0 files is greatly affected by in-memory component (e.g., memtable size), the memory management could have non-negligible, indirect performance impact than expected. Thus optimizing data management in memory also becomes important as the storage performance continues to improve.

• **Writes**. For WRITE operations, 3D XPoint SSD can provide better performance than flash SSDs, for workloads with light to moderate amount of writes. With a heavy load of write requests, however, many optimizations in the current key-value store design, such as the throttling process and write pipelining, which are customized and heavily tuned for flash SSDs, can create unexpected negative effect on 3D XPoint SSD. It unfortunately cancels the great performance advantage of 3D XPoint SSD, despite the much faster hardware. As real-world applications are becoming increasingly WRITE intensive, we must carefully reconsider such "obsolete" optimizations and develop new schemes to fit the properties of the new-generation hardware.

• **Memtable**. The memtable in RocksDB has an opposite effect on READ and WRITE operations. On a faster 3D XPoint SSD, a larger memtable in LSM-tree based data store still can bring benefits for READ-intensive workloads. It is because a larger memtable would result in fewer Level-0 files, which reduces the search time at Level 0. Due to the smaller performance gap between memory and underlying storage, such a benefit is more significant on 3D XPoint SSD.

For WRITE-intensive workloads, a large memtable increases the overhead of insertion operations, causing unanticipated performance degradation. The current system design largely ignores this issue, simply assuming a large in-memory memtable would be generally beneficial for both READ and WRITE operations. More sophisticated mechanisms are needed for performance improvement. For example, since memtable size has opposite effects on READ and WRITE requests, system designers may consider dynamically merging multiple small mutable memtables into a large immutable memtable according to workload characteristics.

• **Read/Write Interference**. We have observed significant throughput improvement with a high I/O parallelism on all three SSDs. Although RocksDB on 3D XPoint SSD outperforms flash based SSDs in terms of throughput, Read/Write interference on 3D XPoint SSD has a more pronounced impact on the WRITE request latency than on flash SSDs. It is because 3D XPoint SSD processes READ requests much faster, which in turn results in more WRITE threads being queued up and waiting until the previous batch writing is done.

Many optimizations can be done to alleviate this problem. For example, we may use multiple short write thread queues rather than one single long queue, creating more parallelism and overlapping. We can also associate write requests with different priority-based performance policies. For example, latency sensitive requests can be processed with a high priority. For practitioners, selecting a proper parallelism setting to balance throughput and latency is a wise choice.

• **Logging**. Logging is a necessary mechanism to ensure the reliability of the data store. However, it is also an important factor affecting performance, which is particularly visible on the high-speed 3D XPoint SSD. In fact, we see a non-trivial performance improvement in both throughput and latency by speeding up the WAL accesses. In practice, system designers should consider optimizing the current logging mechanism or proposing novel designs to minimize the overhead. For example, compressing and condensing the data written to the log could help reduce the I/O traffic and correspondingly reduce the time overhead of logging.

The essential difference between flash SSD and 3D XPoint SSD with RocksDB is not only the significant speed improvement of the physical storage media, but also the continuously diminishing performance gap between memory and storage. As a result, although several original designs, such as write throttling, Level-0 file management, and the pipelined writing, work well on flash SSD, the involved overhead would become non-negligible on 3D XPoint SSD. Our work has provided quantitative results and valuable guidance for system designers to optimize the current design and successfully integrating it with 3D XPoint SSD into data center systems.

VII. Related Work

LSM-tree based key-value stores have been extensively studied. Prior work can be roughly divided into two categories.

Improving Key-value Store Performance. Given the importance of write performance, HyperLevelDB [10] aims to increase the write throughput by introducing a fine-grained locking and a new compaction algorithm. bLSM [31] proposes a new merge scheduler to effectively optimize both write latency and throughput. VT-Tree [33] introduces an additional level of indirection for sorting data and also eliminates unnecessary data copy operations. WiscKey [23] takes a different approach to improve performance. It separates keys and values and moves the values out of the LSM structure. LOCS [40] exploits the internal parallelism of Open-channel SSDs by leveraging the exposed low-level hardware details to improve the compaction performance. Similarly, cLSM [14] also aims to increase the concurrency in LSM-tree based stores. LSbM-tree [36] develops an on-disk buffer to mitigate the effect of invalidated system buffer cache to enhance the compaction efficiency. LSM-trie [41] develops a data structure, called *trie*, to organize keys, therefore mitigating the problem of write amplification. PebblesDB [30] proposes a structure, called Fragmented Log-Structured Merge Trees (FLSM), to achieve high write performance. In this paper, we particularly focus on RocksDB, a popular key-value store, and study the potential performance impact of the new storage hardware.

Modeling Key-value Store Performance. Many prior studies have evaluated the performance of the LSM-tree based key-value store designs. Some focus on analytic metrics, such as read/write amplification factor, analyzing the experiment results and studying the design rationales [10], [11], [20], [21], [24], [31]. Some other works use the concept of per-operation costs for analysis [33], [34], [39], [40]. Besides, other factors, such as operating system [19] and hardware [7], [35], also have significant performance impact on the key-value stores. These prior works more focus on the key-value store itself or other specific layers and lack sufficient studies on the interaction between the software design and the underlying hardware. Our work focuses on this aspect, in particular how the current software design of RocksDB interact with the underlying storage devices, which are quickly evolving. Thus this paper is largely orthogonal to these prior works.

Among the related work, NoveLSM [17], which uses a byte-addressable skiplist, directly manipulates the persistent state, and enhances read I/O parallelism on NVM, is the closest to this work. However, our work focuses on identifying the bottlenecks of the start-of-the-art key-value store design on the emerging 3D XPoint SSD, which is fundamentally different from byte-addressable NVM.

VIII. Conclusion

In this paper, we have conducted comprehensive experiments to understand the performance impact of the new-generation storage hardware on RocksDB. Besides confirming the great performance potential of 3D XPoint SSD, our experimental results have also observed several unexpected bottlenecks of RocksDB with this new technology. We have also developed three case studies as examples to showcase possible optimizations based on our findings. Finally, we have discussed the system implications for system designers and practitioners. Our work shows that the rapidly evolving storage technology, such as 3D XPoint SSD, not only presents many challenges but also opens numerous opportunities.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback and insightful comments. This work was partially supported by the U.S. National Science Foundation under Grants CCF-1453705, CCF-1629291, and CCF-1910958.

References

[1] Apache Flink. https://flink.apache.org/.

[2] Apache HBase. https://hbase.apache.org/.

[3] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design Tradeoffs for SSD Performance. In *Proceedings of USENIX Annual Technical Conference (ATC '08)*, Boston,MA, Jun 2008.

[4] ArangoDB. ArangoDB. https://www.arangodb.com/, 2014.

[5] Ceph. Ceph. https://ceph.io/.

[6] F. Chen, D. A. Koufaty, and X. Zhang. Understanding Intrinsic Characteristics and System Implications of Flash Memory Based Solid State Drives. In *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '09)*, pages 181–192, New York, NY, USA, 2009. ACM.

[7] F. Chen, R. Lee, and X. Zhang. Essential Roles of Exploiting Internal Parallelism of Flash Memory based Solid State Drives in High-speed Data Processing. In *Proceedings of 2011 IEEE 17th International Symposium on High Performance Computer Architecture (HPCA '11)*, pages 266–277, Feb 2011.

[8] D. Dai, Y. Chen, P. Carns, J. Jenkins, and R. Ross. Lightweight Provenance Service for High-Performance Computing. In *Proceedings of the 26th International Conference on Parallel Architectures and Compilation Techniques (PACT '17)*, pages 117–129, Sep. 2017.

[9] C. Developer. Apache Cassandra. http://cassandra.apache.org/, 2016.

[10] R. Escriva, B. Wong, and E. G. Sirer. HyperDex. http://hyperdex.org/performance/leveldb/, 2012.

[11] Facebook. RocksDB. https://rocksdb.org/, 2014.

[12] B. F.Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of 2010 ACM Symposium on Cloud Computing (SoCC '10)*, Indianapolis,Indiana, June 2010.

[13] S. Ghemawat and J. Dean. LevelDB. http://leveldb.org/, 2014.

[14] G. Golan-Gueta, E. Bortnikov, E. Hillel, and I. Keidar. Scaling Concurrent Log-structured Data Stores. In *Proceedings of the 10th European Conference on Computer Systems (EuroSys '15)*, pages 32:1–32:14, New York, NY, USA, 2015. ACM.

[15] F. T. Hady, A. Foong, B. Veal, and D. Williams. Platform Storage Performance With 3D XPoint Technology. *Proceedings of the IEEE*, 105(9):1822–1833, Sep. 2017.

[16] Intel. Intel Optane SSD. https://www.intel.com/content/www/us/en/products/memory-storage/solid-state-drives/gaming-enthusiast-ssds/optane-900p-series.html, 2018.

[17] S. Kannan, N. Bhat, A. Gavrilovska, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. Redesigning LSMs for Nonvolatile Memory with NoveLSM. In *Proceedings of 2018 USENIX Annual Technical Conference (USENIX ATC '18)*, pages 993–1005, Boston, MA, 2018. USENIX Association.

[18] C. Kim, D. Kim, W. Jeong, H. Kim, I. Park, H. W. Park, J. Lee, J. Park, Y. Ahn, J. Y. Lee, S. Kim, H. Yoon, J. Yu, N. Choi, N. Kim, H. Jang, J. Park, S. Song, Y. Park, J. Bang, S. Hong, Y. Choi, M. Kim, H. Kim, P. Kwak, J. Ihm, D. Byeon, J. Lee, K. Park, and K. Kyung. A 512-Gb 3-b/Cell 64-Stacked WL 3-D-NAND Flash Memory. *J. Solid-State Circuits*, 53(1):124–133, 2018.

[19] G. Lee, S. Shin, W. Song, T. J. Ham, J. W. Lee, and J. Jeong. Asynchronous I/O Stack: A Low-latency Kernel I/O Stack for Ultra-Low Latency SSDs. In *Proceedings of 2019 USENIX Annual Technical Conference (USENIX ATC '19)*, pages 603–616, Renton, WA, July 2019. USENIX Association.

[20] H. Lim, D. G. Andersen, and M. Kaminsky. Towards Accurate and Fast Evaluation of Multi-Stage Log-structured Designs. In *Proceedings of 14th USENIX Conference on File and Storage Technologies (FAST '16)*, pages 149–166, Santa Clara, CA, 2016. USENIX Association.

[21] H. Lim, B. Fan, D. G. Andersen, and M. Kaminsky. SILT: A Memory-Efficient, High-Performance Key-Value Store. In *Proceedings of the 23th Symposium on Operating Systems Principles (SOSP '11)*, Cascais, Portugal, Oct 2011.

[22] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn. On the Role of Burst Buffers in Leadership-class Storage Systems. In *Proceedings of 2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST '12)*, pages 1–11, April 2012.

[23] L. Lu, T. S. Pillai, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. WiscKey: Separating Keys from Values in SSD-conscious Storage. In *Proceedings of 14th USENIX Conference on File and Storage Technologies (FAST '16)*, Santa Clara, Feb 2016.

[24] L. Marmol, S. Sundararaman, N. Talagala, , R. Rangaswami, S. Devendrappa, B. Ramsundar, and S. Ganesan. NVMKV: A Scalable and Lightweight Flash Aware Key-Value Store. In *Proceedings of the 6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '14)*, Philadelphia, PA, Jun 2014.

[25] M. Mesnier. Intel Open Storage Toolkit. https://sourceforge.net/projects/intel-iscsi/, 2016.

[26] Micron. Micron 3DXPoint. https://www.micron.com/products/advanced-solutions/3d-xpoint-technology.

[27] MongoDB. MongoDB. https://www.mongodb.com/, 2007.

[28] T. Papadakis. SkipList. https://en.wikipedia.org/wiki/Skip_list, 1993.

[29] Pinterest. Pinterest. https://www.pinterest.com/, 2010.

[30] P. Raju, R. Kadekodi, V. Chidambaram, and I. Abraham. PebblesDB: Building Key-Value Stores using Fragmented Log-Structured Merge Trees. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*, pages 497–514, 10 2017.

[31] S. Russell and R. Raghu. bLSM: A General Purpose Log Structured Merge Tree. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12)*, pages 217–228, New York, NY, USA, 2012. ACM.

[32] B. Schroeder, R. Lagisetty, and A. Merchant. Flash Reliability in Production: The Expected and the Unexpected. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST '16)*, pages 67–80, Santa Clara, CA, 2016. USENIX Association.

[33] P. J. Shetty, R. P. Spillane, R. R. Malpani, B. Andrews, J. Seyster, and E. Zadok. Building Workload-Independent Storage with VT-Trees. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST '13)*, pages 17–30, San Jose, CA, 2013. USENIX.

[34] R. P. Spillane, P. J. Shetty, E. Zadok, S. Dixit, and S. Archak. An Efficient Multi-tier Tablet Server Storage Architecture. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11)*, pages 1:1–1:14, New York, NY, USA, 2011. ACM.

[35] S. Swanson. Early Measurements of Intel's 3DXPoint Persistent Memory DIMMs. https://www.sigarch.org/early-measurements-of-intels-3dxpoint-persistent-memory-dimms/.

[36] D. Teng, L. Guo, R. Lee, F. Chen, S. Ma, Y. Zhang, and X. Zhang. LSbM-tree: Re-Enabling Buffer Caching in Data Management for Mixed Reads and Writes. In *Proceedings of 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS '17)*, 2017.

[37] E. Thereska, A. Donnelly, and D. Narayanan. Sierra: Practical Power-proportionality for Data Center Storage. In *Proceedings of the 6th Conference on Computer Systems (EuroSys '11)*, pages 169–182, New York, NY, USA, 2011. ACM.

[38] M.-A. Vef, N. Moti, T. Suß, T. Tocci, R. Nou, A. Miranda, T. Cortes, and A. Brinkmann. GekkoFS - A Temporary Distributed File System for HPC Applications. In *Proceedings of 2018 IEEE International Conference on Cluster Computing (CLUSTER '18)*, Sep. 2018.

[39] H. T. Vo, S. Wang, D. Agrawal, G. Chen, and B. C. Ooi. LogBase: A Scalable Log-structured Database System in the Cloud. *Proceedings of the 38th International Conference on Very Large Databases (VLDB '12)*, 5(10):1004–1015, June 2012.

[40] P. Wang, G. Sun, S. Jiang, J. Ouyang, S. Lin, C. Zhang, and J. Cong. An Efficient Design and Implementation of LSM-tree Based Key-value Store on Open-channel SSD. In *Proceedings of the 9th European Conference on Computer Systems (EuroSys '14)*, pages 16:1–16:14, New York, NY, USA, 2014. ACM.

[41] X. Wu, Y. Xu, Z. Shao, and S. Jiang. LSM-trie: An LSM-tree-based Ultra-Large Key-Value Store for Small Data Items. In *Proceedings of 2015 USENIX Annual Technical Conference (USENIX ATC '15)*, July 2015.

[42] J. Zhang, P. Li, B. Liu, T. G. Marbach, X. Liu, and G. Wang. Performance Analysis of 3D XPoint SSDs in Virtualized and Non-Virtualized Environments. In *Proceedings of 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS '18)*, pages 1–10, Dec 2018.