Improving Performance of Flash Based Key-Value Stores Using Storage Class Memory as a Volatile Memory Extension

Hiwot Tadese Kassa, *University of Michigan;* Jason Akers, Mrinmoy Ghosh, and Zhichao Cao, *Facebook Inc.;* Vaibhav Gogte and Ronald Dreslinski, *University of Michigan*

https://www.usenix.org/conference/atc21/presentation/kassa

This paper is included in the Proceedings of the 2021 USENIX Annual Technical Conference.

July 14–16, 2021

978-1-939133-23-6

Improving Performance of Flash Based Key-Value Stores Using Storage Class Memory as a Volatile Memory Extension

Hiwot Tadese Kassa
University of Michigan

Jason Akers
Facebook, Inc.

Mrinmoy Ghosh
Facebook, Inc.

Zhichao Cao
Facebook, Inc.

Vaibhav Gogte
University of Michigan

Ronald Dreslinski
University of Michigan

Abstract

High-performance flash-based key-value stores in data-centers utilize large amounts of DRAM to cache hot data. However, motivated by the high cost and power consumption of DRAM, server designs with lower DRAM per compute ratios are becoming popular. These low-cost servers enable scale-out services by reducing server workload densities. This results in improvements to overall service reliability, leading to a decrease in the total cost of ownership (TCO) for scalable workloads. Nevertheless, for key-value stores with large memory footprints these reduced DRAM servers degrade performance due to an increase in both IO utilization and data access latency. In this scenario a standard practice to improve performance for sharded databases is to reduce the number of shards per machine, which degrades the TCO benefits of reduced DRAM low-cost servers. In this work, we explore a practical solution to improve performance and reduce costs of key-value stores running on DRAM constrained servers by using Storage Class Memories (SCM).

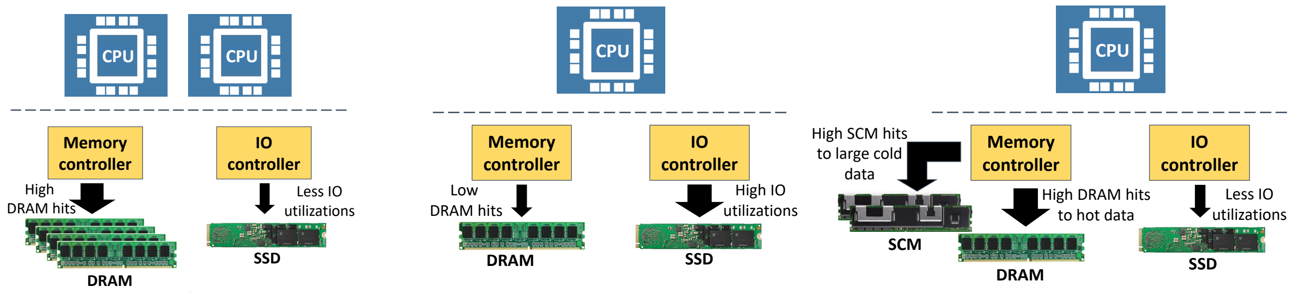
SCM in a DIMM form factor, although slower than DRAM, are sufficiently faster than flash when serving as a large extension to DRAM. In this paper, we use Intel® Optane™ PMem 100 Series SCMs (DCPMM) in AppDirect mode to extend the available memory of RocksDB, one of the largest key-value stores at Facebook. We first designed hybrid cache in RocksDB to harness both DRAM and SCM hierarchically. We then characterized the performance of the hybrid cache for 3 of the largest RocksDB use cases at Facebook (WhatsApp, Tectonic Metadata, and Laser). Our results demonstrate that we can achieve up to 80% improvement in throughput and 20% improvement in P95 latency over the existing small DRAM single-socket platform, while maintaining a 43-48% cost improvement over the large DRAM dual socket platform. To the best of our knowledge, this is the first study of the DCPMM platform in a commercial data center.

1 Introduction

High-performance storage servers at Facebook come in two flavors. The first, *2P server*, has two sockets of compute and a large DRAM capacity as shown in Figure 1a and provides excellent performance at the expense of high power and cost. In contrast, *1P server* (Figure 1b), has one socket of compute and the DRAM-to-compute ratio is half of the *2P server*. The advantages of *1P server* are reduced cost, power, and increased rack density [1]. For services with a small DRAM footprint, *1P server* is the obvious choice. A large number of services in Facebook fit in this category.

However, a class of workloads that may not perform adequately on a reduced DRAM server and take advantage of the cost benefits of *1P server* at Facebook are flash-based key-value stores. Many of these workloads use RocksDB [2] as their underlying storage engine. RocksDB utilizes DRAM for caching frequently referenced data for faster access. A low DRAM to storage capacity ratio for these workloads will lead to high DRAM cache misses, resulting in increased flash IO pressure, longer data access latency, and reduced overall application throughput. Flash-based key-value stores in Facebook are organized into shards. An approach to improve the performance of each shard on DRAM constrained servers is to reduce the number of shards per server. However, this approach can lead to an increase in the total number of servers required, lower storage utilization per server, and dilutes the TCO benefits of the *1P server*. This leaves us with the difficult decision between *1P server*, which is cost-effective while sacrificing performance, or *2P server* with great performance at high cost and power. An alternative solution that we explore in this paper is utilizing recent Intel® Optane™ PMem 100 Series SCMs (DCPMM) [3] to efficiently expand the volatile memory capacity for *1P server* platforms. We use SCM to build new variants of the *1P server* platforms as shown in Figure 1c. In *1P server variants*, the memory capacity of *1P server* is extended by providing large SCM DIMMs alongside DRAM on the same DDR and bus attached to the CPU memory controller.

Storage Class Memory (SCM) is a technology with the properties of both DRAM and storage. SCMs in DIMM form factor have been studied extensively in the past because of their attractive benefits including byte-addressability, data persistence, cheaper cost/GB than DRAM, high density, and their relatively low power consumption. This led to abundant research focusing on the use cases of SCM as memory



(a) **2P server.** (b) **1P server.** (c) **1P server variant.**

Figure 1: Server configurations with different DRAM sizes: (a) Server with 256GB memory, high hit rate to DRAM and low IO utilization (b) Server with reduced (64GB) memory, lower DRAM hit rate and increased in IO utilization and (c) Server with reduced memory and SCM added to the memory controller, high hit rate to DRAM & SCM due to optimized data placement, which decreases IO utilization.

Table 1: Example of memory characteristics of DRAM, SCM and flash per module taken from product specifications.

Characteristics	**DRAM**	**SCM**	**Flash**
Idle read latency (ns)	75	170	85,000
Read bandwidth (GB/s)	15	2.4	1.6
Power (mW / GB)	375	98	5.7
DRAM Relative Cost per GB	1	0.38	0.017
Granularity	byte addressable	byte addressable	block based
Device Capacity (GB)	32	128	2048

and persistent storage. The works range from optimizations with varying memory hierarchy configurations [4–8], novel programming models and libraries [9–11], and file system designs [12–14] to adopt this emerging technology. Past research was focused primarily on theoretical or simulated systems, but the recent release of DCPMM-enabled platforms from Intel motivates studies based on production-ready platforms [15–24]. The memory characteristics of DRAM, DCPMM, and flash are shown in Table 1. Even though DCPMM has higher access latency and lower bandwidth than DRAM it has a much larger density and lower cost, and its access latency is two orders of magnitude lower than flash. Currently, DCPMM modules come in 128GB, 256GB, and 512GB capacities, much larger than DRAM that typically ranges from 4GB to 32GB in a data-center environment. Hence we can get a tremendously larger density with DCPMM. If we efficiently (cost and performance) use this memory as an extension to DRAM, this would enable us to build dense, flexible, servers with large memory and storage, while using fewer DIMMs and lowering the total cost of ownership (TCO).

Although recent works demonstrated the characteristics of SCM [15, 18], the performance gain achieved in large commercial data-centers by utilizing SCM remains unanswered. There are open questions on how to efficiently configure DRAM and SCM to benefit large scale service deployments in terms of cost/performance. Discovering the use cases within a large scale deployment that profit from SCM has also been challenging. To address these challenges for RocksDB, we first profiled all flashed-based KV store deployments at Facebook to identify where SCM fits in our environment. These studies revealed that we have abundant read-dominated workloads, which focused our design efforts on better read performance. This has also been established in previous work [25–27] where faster reads improved overall performance for workloads serving billions of reads every second. Then, we identified the largest memory consuming component of RocksDB, the block cache used for serving read requests, and redesigned it to implement a hybrid tiered cache that leverages the latency difference of DRAM and SCM. In the hybrid cache, DRAM serves as the first tier cache accommodating frequently accessed data for fastest read access, while SCM serves as a large second tier cache to store less frequently accessed data. Then, we implemented cache admission and memory allocation policies that manage the data transfer between DRAM and SCM. To evaluate the tiered cache implementations we characterize three large production RocksDB use cases at Facebook using the methods described in [28] and distilled the data into new benchmark profiles for db_bench [29]. Our results show that we can achieve 80% improvement to throughput, 20% improvement in P95 latency, and 43-48% reduction in cost for these workloads when we add SCM to existing server configurations. In summary, we make the following contributions:

- We characterized real production workloads, identified the most benefiting SCM use case in our environment, and developed new db_bench profiles for accurately benchmarking RocksDB performance improvement.
- We designed and implemented a new hybrid tiered cache module in RocksDB that can manage DRAM and SCM based caches hierarchically, based on the characteristics of these memories. We implemented three admission policies for handling data transfer between DRAM and SCM cache to efficiently utilize both memories. This implementation will enable any application that uses RocksDB as its KV Store back-end to be able to easily use DCPMM.
- We evaluated our cache implementations on a newly released DCPMM platform, using commercial data center workloads. We compared different DRAM/SCM size server configurations and determined the cost and per-

formance of each configuration compared to existing production platforms.

- We were able to match the performance of large DRAM footprint servers using small DRAM and additional SCM while decreasing the TCO of read dominated services in production environment.

The rest of the paper proceeds as follows. In Section 2 we provide a background of RocksDB, the DCPMM hardware platforms, and brief description of our workloads. Sections 3 and 4 explain the designs and implementation of the hybrid cache we developed. In Section 5 we explain the configurations of our systems and the experimental setup. Our experimental evaluations and results are provided in Section 6. We then discuss future directions and related works in Section 7 and Section 8 respectively, and conclude in Section 9.

2 Background

2.1 RocksDB architecture

A Key-value database is a storage mechanism that uses key-value pairs to store data where the key uniquely identifies values stored in the database. The high performance and scalability of key-value databases promote their widespread use in large data centers [2, 30–32]. RocksDB is a log-structured-merge [33] key-value store engine developed based on the implementation of LevelDB [30]. RocksDB is an industry standard for high performance key-value stores [34]. At Facebook RocksDB is used as the storage engine for several data storage services.

2.1.1 RocksDB components and memory usage

RocksDB stores key-value pairs in a Sorted String Table (SST) format. Adjacent key-value data in SST files are partitioned into data blocks. Other than the data block, each SST files contains Index and Filter blocks that help to facilitate efficient lookups in the database. SST files are organized in levels, for example, Level0 - LevelN, where each level comprises multiple SST files. Write operations in RocksDB first go to an in-memory write buffer residing in DRAM called the memtable. When the buffered data size in the memtable reaches a preset size limit RocksDB flushes recent writes to SST files in the lowest level (Level0). Similarly, when Level0 exhausts its size limit, its SST files are merged with SST files with overlapping key-values in the next level and so on. This process is called compaction. Data blocks, and optionally, index and filter blocks are cached (typically uncompressed) in an in-memory component called the Block Cache that serves read requests from RocksDB. The size of the Block Cache is managed by global RocksDB parameters. Reads from the database are attempted to be serviced first from the memtable, then next from the Block Cache(s) in DRAM, and finally from the SST files if the key is not found in memory. Further details about Rocksdb are found in [2]. The largest use of memory in RocksDB comes from the Block Cache, used for reads.

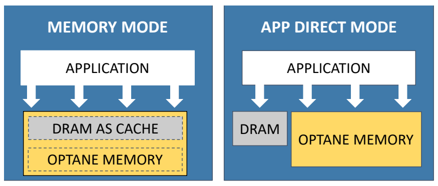


Figure 2: Intel® Optane™ memory operation modes overview.

Therefore, in this work we optimize the RocksDB using SCM as volatile memory for the Block Cache.

2.1.2 Benchmarking RocksDB

One of the main tools to benchmark RocksDB is db_bench [29]. Db_bench allows us to mock production RocksDB runs by providing features such as multiple databases, multiple readers, and different key-value distributions. Recent work [28] has shown how we can create realistic db_bench workloads from production workloads. To create evaluation benchmarks for SCM we followed the procedures given in [28, 35].

2.2 Intel® Optane™ DC Persistent Memory

Intel® Optane™ DC Persistent Memory based on 3D XPoint technology [36, 37], is the first commercially available non-volatile memory in the DIMM form factor and resides on the same DDR bus as DRAM [3]. DCPMM provides byte-addressable access granularity which differentiates it from similar technologies which were limited to larger block-based accesses. This creates new opportunities for low latency SCM usage in data centers as either a volatile memory extension to DRAM or as a low latency persistent storage media.

2.2.1 Operation mode overview

DCPMM can be configured to operate in one of two different modes: Memory Mode and App Direct Mode [38]. Illustrations of the modes are shown in Figure 2.

In **Memory Mode**, as shown in Figure 2, the DRAM capacity is hidden from applications and serves as a cache for the most frequently accessed addresses, while DCPMM capacity is exposed as a single large volatile memory region. Management of the DRAM cache and access to the DCPMM is handled exclusively by the CPU's memory controller. In this mode applications have no control of where their memory allocations are physically placed (DRAM cache or DCPMM).

In **App Direct Mode**, DRAM and DCPMM will be configured as two distinct memories in the system and are exposed separately to the application and operating system. In this case, the application and OS have full control of read and write accesses to each media. In this mode, DCPMM can be configured as block-based storage with legacy file systems or can be directly accessed (via DAX) by applications using memory-mapped files.

2.3 Facebook RocksDB workloads

For our experiments and evaluation we chose the largest RocksDB use cases at Facebook, which demonstrate typical uses of key-value storage ranging from messaging services to

large storage for processing realtime data and metadata. Note than these are not the only workloads that benefit from our designs. The descriptions of the services are as follows:
WhatsApp: With over a billion active users, WhatsApp is one of the most popular messaging applications in the world [39]. WhatsApp utilizes ZippyDB as its remote data store. ZippyDB [40] is a distributed KV-store that implements Paxos on top of RocksDB to achieve data reliability and persistence.
Tectonic Metadata: The Tectonic Metadata databases are also stored in ZippyDB and are an integral part of the large blob storage service of Facebook that serves billions of photos, videos, documents, traces, heap dumps, and source code [41, 42]. Tectonic Metadata maintains the mappings between file names, data blocks and parity blocks, and the storage nodes that hold the actual blocks. These databases are distributed and fault-tolerant.
Laser: Laser is a high query throughput, low (millisecond) latency, peta-byte scale key-value storage service built on top of RocksDB [43]. Laser reads from any category of Facebook's real-time data aggregation services [44] in real-time or from a Hadoop Distributed File System [45] table daily.

3 Hybrid cache design choices

3.1 The challenges of SCM deployment

The first challenge of introducing SCM in RocksDB is identifying which of its components to map to SCM. We chose the uncompressed block cache because it has the largest memory usage in our RocksDB workloads and because our studies reveal that a number of our production workloads, which are read-dominated, benefit from optimizing read operations by block cache extension. We also focused on the uncompressed block cache instead of the compressed ones, so that we can minimize CPU utilization increase when performing compression/decompression. This allowed us to increase the size of SCM (block cache) without requiring additional CPU resources. We also chose block cache over memtable because SCM provides better read bandwidth than writes, hence helping our read-demanding workloads. We then expanded the block cache size by utilizing SCM as volatile memory. We chose this approach because extending the memory capacity while reducing the size of DRAM and the cost of our servers is the primary goal. Although we can benefit from persisting block cache and memtable in SCM for fast cache warmup and fast write access, we left this for future work.

The next challenge is, how we should configure SCM to get the best performance. We have the options of using memory mode, that does not require software architecture changes or app-direct mode that necessitates modification in RocksDB but provides control of DRAM and SCM usage. Figure 3 demonstrates how memory-mode compares to our optimized app-direct mode. Optimized app-direct mode with various DRAM and SCM sizes, renders 20-60% throughput improvement and 14-49% lower latency compared with memory mode. This insight supports that our optimized implementation has

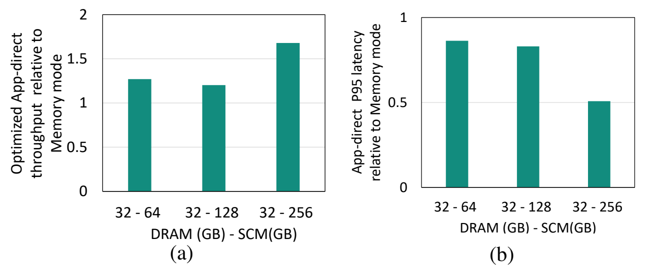


Figure 3: Throughput and latency comparison for memory mode and our optimized hybrid-cache in app-direct mode for WhatsApp.

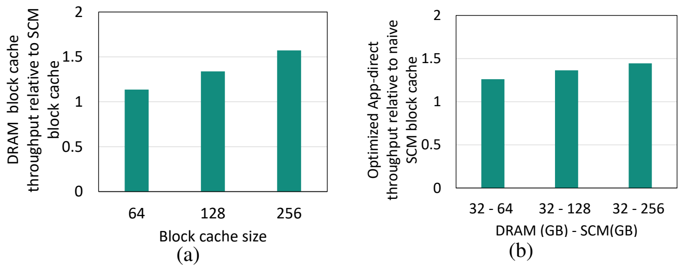


Figure 4: (a) Throughput of DRAM vs SCM based block cache. (b) Throughput of naive SCM vs optimized hybrid-cache for WhatsApp.

a better caching mechanism than memory mode, hence we focused our analysis on app-direct mode.

With app-direct we can manage the allocation of RocksDB's components (memtable, data, filter, and index blocks) to DRAM or SCM. But since we know the data access latency of SCM is slower than DRAM (see Table 1), we have to consider its effect. We compared the throughput of allocating the block cache to DRAM or SCM in app-direct mode in vanilla RocksDB to understand the impact of the higher SCM access latency. As seen in Figure 4a, the slower SCM latency creates 13%-57% difference in throughput when we compare DRAM based block cache to a naive SCM block cache using app-direct mode. This result guided us to carefully utilize DRAM and SCM in our designs. In single-socket machines such as *1P servers*, we have one CPU and 32GB - 64GB DRAM capacity. Out of this DRAM, memtable, index, and filter blocks consume 10-15 GBs. The rest of DRAM and the additional SCM can be allocated for block cache. We compared the naive SCM block cache implementation (all block cache allocated to SCM using app-direct) to a smarter and optimized hybrid cache, where highly accessed data is allocated in DRAM and the least frequently access in SCM. The results in Figure 4b show with optimized app-direct we achieve up to 45% better throughput compared to a naive SCM block cache. From this, we can determine that implementing a hybrid cache compensates for the performance loss due to the higher SCM access latency. These results together with the high temporal locality of our workloads (as discussed below) motivated us to investigate a hybrid cache.

3.2 RocksDB workload characteristics

Below we scrutinize the characteristics of our largest RocksDB workloads that guided our hybrid cache design.

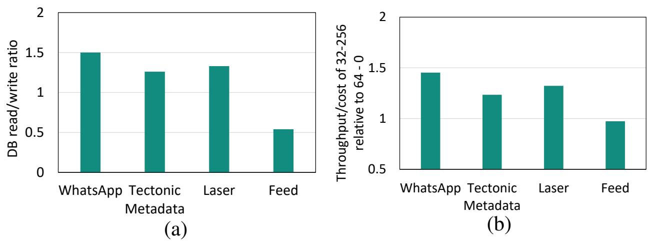


Figure 5: (a) Read to write ratio to DB. (b) Key-value throughput/-cost compare for large block cache friendly workloads and Feed with higher writes than reads.

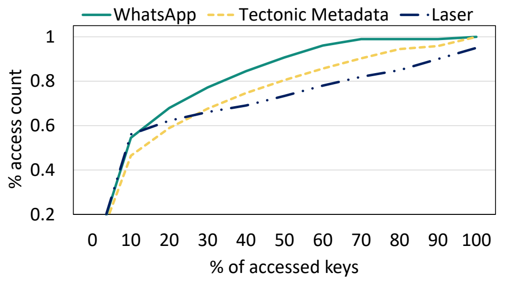


Figure 6: Key-value locality.

Reads and writes to DB: As we discussed earlier, prior work showed that optimizing reads provides large impact in commercial data center workloads [25–27]. Our studies also show that we have a large number of read-dominated workloads, therefore optimizing the block cache, used for storing data for fast read access, will benefit a number of our workloads. In RocksDB, when a key is updated in memtable it will be invalid in the block cache. Hence, if the workload has more write queries than reads, then the data in the cache will become stale. Note that write-dominated workloads won't be affected by our hybrid cache designs because we did not reduce any DRAM buffer (memtable) in the write path. In our studies, we profiled deployed RocksDB workloads for 24 hours using an internal metrics collection tool to comprehend the read and write characteristics. Figure 5a shows the workloads described in Section 2.3 reads more bytes from the DB than it writes. To contrast, we evaluated one of our write-dominated workloads, Feed, also seen in Figure 5a. In Figure 5b, we calculated the throughput per cost of *1P server variants* with 32 GB DRAM and 256 GB SCM capacity normalized to throughput/cost of *1P server* with 64 GB DRAM capacity. The throughput/cost improvement of Feed for our largest DRAM-SCM system cannot offset the additional cost due of SCM. Hence, we focus on exporting read-dominated workloads to our hybrid systems.

Key-value temporal locality: Locality determines the cacheability of block data given a limited cache size. A hybrid cache with a small DRAM size will only benefit us if we have a high temporal locality in the workloads. In this case, significant access to the block cache will come from the DRAM cache, and SCM will hold the bulk of less frequently accessed blocks. We used RocksDB trace analyzer [35] to investigate up to 24 hours query statistics of workloads running on production *2P server* and evaluate locality as the distribution of the total database access counts to the total keys accessed per database. Figure 6 shows that our workloads possess a power-law relationship [46] between the number of key-value pair access counts and the number of keys accessed. We can observe in the figure that 10% of the key-value pairs carry ~50% of the key-value accesses. This makes a hybrid cache design with small DRAM practical for deployment.

DB and cache sizes: The desirable DRAM and SCM cache sizes required to capture workload's locality is proportional to the size of the DB. Workloads with high key-value locality and large DB sizes can achieve a high cache hit rate with limited cache sizes. But as the locality decreases for large DB sizes, the required cache sizes will grow. In the extreme case of random key-value accesses, all blocks will have similar heat, diluting the value of the DRAM cache and reducing overall hybrid cache performance asymptotically toward that of the SCM-only block cache. For small DBs, locality might not play a significant role because the majority of the DB accesses fit in a small cache. Such types of workloads will not be severely affected by DRAM size reduction, and choosing the *1P server variants* with large SCM capacity will be a waste of resources. In our studies, after looking at various workloads in production, we choose our hybrid DRAM-SCM cache configuration to accommodate several workloads with larger DB sizes ($\sim$2TB total KVstorage per server).

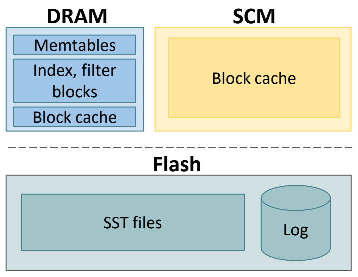
4 DRAM-SCM hybrid cache module

In our RocksDB deployment, we placed the memtables, index blocks, and filter blocks in DRAM. We then designed a new hybrid cache module that allocates the block cache in DRAM and SCM. The database SST files and logs are located in Flash. The overview of RocksDB components allocation in the memory system is shown in Figure 7a. Our goal in designing the new hybrid cache module is to utilize DRAM and SCM hierarchically based on their read access latency and bandwidth characteristics. In our design, we aim to place hot blocks in DRAM for the lowest latency data access, and colder blocks in SCM as a second tier. The dense SCM based hybrid block cache provides a larger effective capacity than practical with DRAM alone leading to higher cache hit rates. This dramatically decreases IO bandwidth requirements to the SST files on slower underlying flash media.

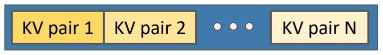
The block cache is an integral data structure that is completely managed by RocksDB. Similarly, in our implementations, the new hybrid cache module is fully managed by RocksDB. This module then is an interface between RocksDB and the DRAM and SCM block caches, and fully manages the caches' operations. The overall architecture of the hybrid cache is shown in Figure 7c. The details of its internal components are as follows:

4.1 Block cache lists

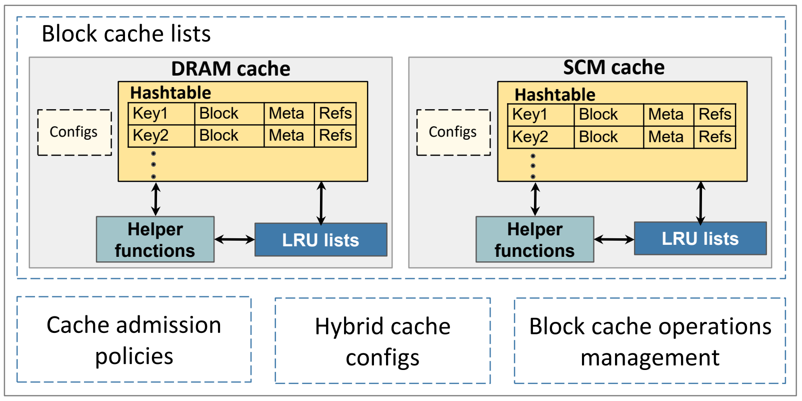
The hybrid cache is a new top-level module in RocksDB that maintains a list of underlying block caches in different tiers.



(a) Memory allocation of RocksDB components to hybrid DRAM-SCM cache.



(b) Data block structure.



(c) Hybrid tier cache component and architecture.

Figure 7: RocksDB components and memory allocation.

The list of caches are extended from the existing RocksDB block cache with LRU replacement policy. Note that in our implementations we have DRAM and SCM cache, but the module can manage more than these two caches such as multiple DRAM and SCM caches in a complex hierarchy.

4.1.1 Block cache architecture and components

The internal structures of DRAM and SCM caches, which are both derived from the block cache, are shown in Figure 7c. The block cache storage is divided into cache entries and tracked in a **hashtable**. Each cache entry holds a key, data block, metadata such as key size, hash, current cache usage, and a reference count of the cache entry outside of the block cache. The data block is composed of multiple key-value pairs as shown in Figure 7b. Binary searches are performed to find a key-value pair in a data block. The data block size is configurable in RocksDB. In our case, the optimal size was 16KB. As the number of index blocks decreases we can increase the data block size. As a result, with 16KB we were able to reduce the number of index blocks making room for data blocks within our limited DRAM capacity. Every block cache has **configs** that are configured externally. This includes size, a threshold for moving data, a pointer to all other caches for data movement, and the memory allocator for the cache. The cache maintains an **LRU list** that tracks cache entries in order from most to least recently used. The **helper functions** are used for incrementing references, checking against the reference threshold, transferring blocks from one cache to another, checking size limits, and so on. For the components listed above we extended and modified RocksDB to support tiered structure, different kinds of admission policies and we designed new methodologies to enable data movement between different caches and to support memory allocation to different memory types.

4.1.2 Data access in the block cache

A block is accessed by a number of external components to the block cache, such as multiple reader clients of the RocksDB database. The number of external referencers is tracked by the reference count. Mapping to a block is created when it is referenced externally, this will increment the reference count. Whereas when the referencer no longer needs a block, mapping is released, and the reference count is decremented. If a block has zero external references, it will be in the hashtable and tracked by the LRU list. If a block gets referenced again, then it will be removed from the LRU list. Note that in the LRU list, newly released blocks with no external references are on the top of the LRU list as the most recently used blocks, and when blocks are evicted, the bottom least recently used blocks are evicted first. The block cache is used for read-only data, hence it doesn't deal with any dirty data management. Therefore, when transferring data between DRAM and SCM we do not have to deal with dirty data.

4.2 Cache admission policies

Identifying and retaining blocks in DRAM/SCM based on their access frequencies requires proactive management of data transfer between DRAM, SCM, and flash. Hence, we developed the following block cache admission policies.

4.2.1 DRAM first admission policy

In this admission policy, new blocks read from flash are first inserted into the hashtable of the DRAM cache. The block cache data structures are size limited. Hence when the size of the blocks allocated in the DRAM cache exceeds the size limits, the oldest entries tracked by the DRAM LRU list are moved to the next tier cache (SCM cache) by the data mover function of the DRAM cache, using the SCM cache's memory allocator. On lookups, both the DRAM and SCM caches are searched until the cache block is found. If it is not found, it will initiate a flash read. Similar to the DRAM cache when the capacity of the SCM cache exceeds the limit, the oldest entries in the LRU list of the SCM cache are freed to accommodate new cache blocks evicted from the DRAM cache.

4.2.2 SCM first admission policy

In this admission policy, new blocks read from flash are first inserted in the hashtable of the SCM cache. Unlike the DRAM first admission policy, this policy has a configurable threshold for moving data from the SCM cache to the DRAM cache.

When the external references of cache entries in the SCM cache surpasses the reference threshold, blocks are considered to be hot and will be migrated to the DRAM cache for faster access. The data movement, in this case, is handled by the data mover function of the SCM cache. When the capacity of both the DRAM and SCM caches are full, the oldest LRU blocks are evicted from both caches. In the DRAM cache, LRU entries are moved back to the SCM cache, whereas in the SCM cache, the LRU entries are freed to accommodate new block insertions. On lookup, both the DRAM and SCM caches are searched until the cache block is found.

4.2.3 Bidirectional admission policy

In Bidirectional admission policy, similar to the DRAM first admission policy, new data blocks are inserted into the DRAM cache. As the capacity of DRAM and SCM cache reach the limit, the oldest LRU entries are evicted to SCM cache from the DRAM cache and are freed for the case of SCM cache. The difference between DRAM-first and Bidirectional cache is, after the oldest LRU entries are evicted from DRAM to SCM cache, if the external reference to an entry surpasses a preset threshold it is transferred back to the DRAM cache. This property allows us to re-capture fast access performance for blocks with inconsistent temporal access patterns.

In the hybrid cache, we can set the three of the admission policies or we can easily extend a new policy by configuring how to insert, lookup, and move data in the list of block caches. These configs are global parameters in the top-level hybrid cache and are used by the block cache operations manager and list of block caches. Optionally the thresholds for moving data in SCM first and Bidirectional policies can be set to change values based on the current usage of the caches. But in our experiments, we didn't see benefit with changing values. We also performed an analysis with different sizes of cache thresholds and we show the optimal threshold for SCM first and Bidirectional in our evaluations.

4.3 Hybrid cache configs

The hybrid cache configurations are set outside of the module by RocksDB, and include pointers to configs of all block caches, the number of block caches, ids, tier numbers of the caches, and admission policy to use. Configs are used during instantiation and at run time to manage database operations.

4.4 Block cache operation management

This unit redirects external RocksDB operations such as insert, lookup, update, and so on to the target block cache based on the admission policy. For example, it decides if an incoming insert request should go to the DRAM or SCM cache.

5 Systems setup and implementation

5.1 DRAM-SCM cache implementation

We configured Intel DCPMMs in App Direct mode using the IPMCTL [47] tool in our experiments. We used Linux Kernel 5.2 that brings support for a volatile use of DCPMM by configuring a hot-pluggable memory region called KMEM DAX. We then used NDCTL 6.7 [48], a utility for managing SCM in the Linux Kernel, to create namespaces in DCPMM in devdax mode. This mode provides direct access to DCPMM and is faster than filesystem-based access. We then used DAXCTL [48] utility for configuring DCPMM in system-ram mode so that DCPMM will be available in its own volatile memory NUMA node. To implement a hybrid DRAM-SCM cache we used memkind library [49], which enables partitioning of the heap between multiple kinds of memories such as DRAM and SCM in the application space. After the system is configured with DRAM and DCPMM memory types, we modified RocksDB block cache to take two types of memory allocators using memkind. We use MEM_KIND_DAX_KMEM kind for SCM cache and a regular memory allocator for DRAM. The overview of our implementation is shown in Figure 8.

Table 2: System setup: hardware specs and software configs.

Specification	System config
Application version	RocksDB 6.10
OS	CentOS-8
Linux kernel version	5.2.9
CPU model	Intel(R) Xeon(R) Gold 6252 @ 2.10GHz
Socket/Cores per socket	2/24
Threads total	96
L1I/L1D cache	32 KB
L2/L3 cache	1 MB/35.75 MB
Memory controllers total	4
Channels per controller	3
Slots per channel	2
DRAM size	DDR4 192 GB (16 GB X 12 DIMM slots)
SCM size	DDR-T 1.5 TB (128 GB X 12 DIMM slots)
SSD model	Samsung 983 DCT M.2 NVMe SSD
SSD size/filesystem	2 TB / xfs

5.2 Evaluation hardware description

In our evaluations we used a Intel Wolf Pass [50] based system, utilizing 2 CPU sockets populated with Intel Cascade Lake processors [51]. Each CPU has 2 memory controllers with 3 channels each, and 2 DIMM slots per channel, for a total of 24 DIMM slots. DIMM slots were populated by default in a 2-2-2 configuration, with 12 total *16GB PC4-23400 ECC Registered DDR4 DRAM DIMMs* and 12 total *128GB Intel® Optane™ PMem 100 DIMMs*. This makes up a total of 192GB DRAM and 1.5TB of SCM per system. Backing storage for the RocksDB database was a *Samsung 983 DCT M.2 SSD*. The detailed specification is listed in Table 2.

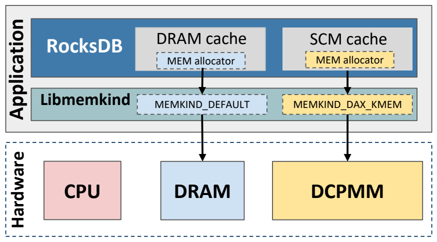


Figure 8: DRAM-SCM cache configuration with libmemkind.

Table 3: OCP Tioga Pass (2P server) and OCP Twin Lakes Platforms (1P server) details.

Specification	**OCP Tioga Pass Config**	**OCP Twin Lakes Config**
CPU model	Xeon(R) Gold 6138	Xeon(R) D-2191
Sockets/cores per socket	2/20	1/18
Threads total	80	36
Memory controllers total	4	2
Channels per controller	3	4
DRAM size	DDR4 256 GB	DDR4 64 GB
SSD size	2 TB	2 TB

5.3 Facebook server designs

The Facebook platforms used for TCO analysis are the *2P server* platform based on the OCP Tioga Pass specification [52], and the *1P server* platform based on the OCP Twin Lakes specification [53]. In addition, we propose several *1P server* variants utilizing differing capacities of DRAM and SCM. The detailed specifications and relative costs of these platforms compared to the baseline *2P server* platform are listed in Table 3 and 4. Platform costs are calculated based on current OCP solution provider data [54,55] and DCPMM cost relative to DRAM provided in [56,57]. The relative cost of DRAM and DCPMM are predicted to maintain similar trends over time [56]. We calculated the cost by adding the cost of individual modules. For DRAM and SCM, we used 16GB and 128GB modules, respectively. In the TCO calculations although introducing SCM adds additional power cost, when we consider the overall power of the system including CPU, NIC, and SSD, the increase of power even for our largest 1P server becomes minimal. We have a power budget for a rack of servers with some power slack and the slight rise in power for SCM is sufficiently below our rack power budget.

Table 4: DRAM and SCM cache in server configuration and memory sizes used block cache in our experiments.

Configuration	**2P serv.**	**1P serv.**	**64-128**	**32-128**	**32-256**
DRAM size	256	64	64	32	32
SCM size	0	0	128	128	256
DRAM Block cache size	150	40	40	12	12
SCM Block cache size	0	0	100	100	200
Other Rocksdb components	10-15	10 -15	10 - 15	10 -15	10 -15
Codebase	5	5	5	5	5
2P rel. cost	**1**	**0.43**	**0.5**	**0.46**	**0.53**

5.4 Cache and memory configurations

To experiment with SCM benefits when added to existing configurations we examined server configurations with different sizes of DRAM and SCM. The configurations we used are shown in Table 4. Our experiments verified that *1P server* has significant performance loss compared to *2P server*. The prominent questions here are how much benefit can we achieve by adding SCM to *1P servers*, and can we still maintain the TCO benefits of *1P server* platform. Hence we used the *1P server* with 64 DRAM and no SCM as the baseline for our evaluations. We then evaluated how much performance we can gain by adding 128 GB SCM to the baseline. Since we are interested in DRAM constrained server configurations, we also evaluate the performance gain when we further reduce DRAM to 32 GB while adding 128 or 256 GB SCM. This gives us the four server configurations provided in Table 4. Although we experimented with 64GB of SCM, as seen in Section 3, to understand the performance of different SCM sizes, DCPMM is not available as a 64GB module, so we didn't consider it in the evaluation below. To run all of the server configurations, we limited the memory usage of the DRAM and SCM for the workloads to the sizes given in Table 4. We also use 1 CPU in our experiments because *1P servers* are single-socket machines. We aimed to maximize block cache allocation to evaluate our DRAM-SCM hybrid cache. To do this we studied the memory usage of other components in RocksDB when it runs in our production servers. We then subtracted these usages from total memory and assigned the rest of the memory for the Block cache. The block cache sizes we used in our evaluations are illustrated in Table 4.

5.5 Workload generation

To generate workloads, we first selected random sample hosts running WhatsApp, Tectonic Metadata, and Laser in production deployments to collect query statistics traces, being careful to select leaders for use cases relying on Paxos. Note that hosts running the same services manifest similar statistics. Then we followed the methodology in [28,35] to model the workloads. We also extended these methodologies [28,35] to scale workloads to match the production deployments. The db_bench workloads we developed mirror the following characteristics of production RocksDB workloads.

KV-pair locality: This is characterized by fitting real workload trace profiles to a probability cumulative function using power distributions in MATLAB [58] based on the power-law characteristics of the workloads (see Figure 6).

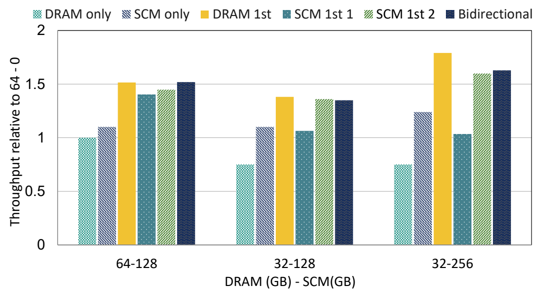
Value distribution: The granularity of value accessed is modeled using Pareto distribution from workload statistics in MATLAB [59].

Query composition: The percentage of get, put, and seek quires are incorporated in the db_bench profiles.

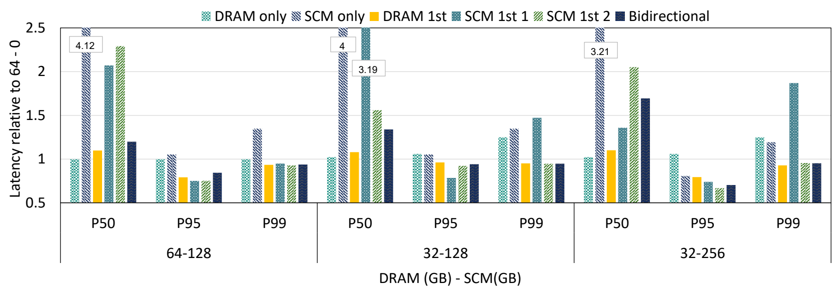
DB, key, and value sizes: We added the average values of the number of keys per database, key, and value size from collecting data from our production servers to create a realistic DB sizes in the db_bench profile.

QPS: The QPS in db_bench is modeled using sine distributions [60] based on trace collected in the production host.

Scaled db_bench profiles: After generating the workloads db_bench profile for a singe database we scaled the workloads by running multiple RocksDB instances, simulating the number of production shards per workload on a single host. These shards share the same block cache. In RocksDB there exists multiple readers and writers to the database. To imitate



(a) Throughput comparison of admission policies.



(b) Latency comparison of admission policies.

Figure 9: Throughput and application read latency comparison using only DRAM for block cache, using only SCM for block cache, and hybrid DRAM-SCM admission policies (DRAM first, SCM first, and Bidirectional) for WhatsApp using all server configurations in Table 4. (a) Throughput comparisons (db operations/sec) for admission policies. Here the baseline is 64 - 0 configs. (b) P50, P95, and P99 latencies for all admission policies and server configs compared to 64 - 0 server.

this property we run multiple threads reading and writing to the set of shards in the db_bench process.

6 Evaluation

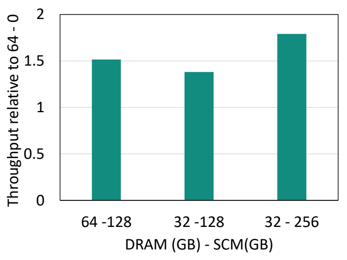
6.1 Throughput and latency comparison for admission policies

In Figure 9 we compare the throughput achieved for 5 different categories: **DRAM Only**: The Block Cache is only allocated in DRAM. **SCM Only**: The Block Cache is only allocated in SCM using app-direct mode. **DRAM First**: The DRAM first policy discussed in Section 4. **SCM First 1 and 2**: The SCM first policy discussed in Section 4 with two threshold values. SCM 1st 1, with threshold value of 2 and SCM 1st 2 with the optimal threshold which is 6. **Bidirectional**: The Bidirectional policy discussed in Section 4 with the optimal threshold value which is 4.

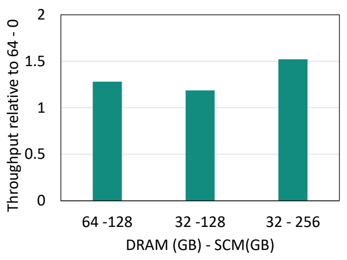
In Figure 9a, we demonstrate the throughput differences of all admission policies for WhatsApp. The results show that using only SCM for block cache provides 15% throughput improvement for the 128 GB SCM configurations and 25% improvement for the 256GB SCM compared to the baseline (a server configuration with 64GB DRAM and no SCM). If we look at Figure 9b, because of latency differences between SCM and DRAM, getting data only from SCM worsens the P50 application read latencies. Therefore we conclude that while the default RocksDB SCM implementation may decrease flash IO utilization, it will have a net negative impact on Facebook application performance due to the 2X - 4X worse P50 latency observed. We can also see in the figure for all the server configurations DRAM first policy achieves the best performance. For 64 - 128 and 32 - 128 configurations, SCM first and Bidirectional get close in throughput benefit to DRAM first, but when there is a large block cache, like in the 32 - 256 configuration DRAM first attains the best result. This is because, in DRAM-first, data transfer between DRAM and SCM cache only occurs once, when DRAM cache evicts data to SCM. But in the case of SCM and Bidirectional policies, data transfer occurs when DRAM evicts data to SCM and when hot blocks are transferred from SCM to DRAM. This creates more bandwidth consumption across the DDR bus resulting in performance degradation, especially for configurations with large block cache sizes. For larger DRAM capacities, SCM first 1, SCM first 2, and Bidirectional policies have comparable throughput because the large DRAM size reduces data evictions to SCM. But as DRAM is reduced (in 32 -128), SCM 1st throughput falls quickly because it will move data from SCM to DRAM with a low activation threshold. When we increase DCPMM capacity in the 32 - 256 case, data transfer increases even for SCM 2 and Bidirectional policies, hence the DRAM first policy is the overall performance winner. If we look at read latencies shown in Figure 9b, the P50 latency remains similar for DRAM first compared to 64 - 0 configurations. The reason for this is that P50 latencies are primarily governed by DRAM accesses. The effect of data transfer from flash instead of SCM can be observed in the P95 and P99 latencies, where the DRAM First policy does significantly better than other policies and the default configuration with no SCM. Tectonic Metadata and Laser (not shown here) also attain the best performance with DRAM first policy.

6.2 Performance comparison of DRAM first policy for all workloads

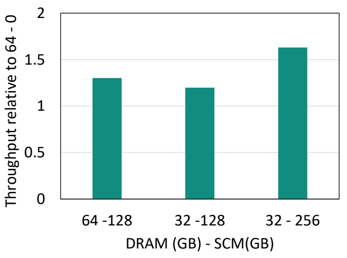
Figure 10 shows the throughput and latency comparison of WhatsApp, Tectonic Metadata, and Laser for DRAM first admission policy (**the best admission policy for all workloads**) for all server configurations. As seen in the figure, our hybrid block cache implementation provides throughput improvement for all the workloads. As seen in Figure 10a, 10b, and 10c throughput is increased up to 50 - 80% compared to the baseline *1P server*'s 64 - 0 due to the addition of SCM. The throughput increase is correlated to the total size of the block cache. Note that increasing the SCM or DRAM capacity further than 256GB will require either more DIMM slots or higher density DIMMs, with different price/performance/reliability considerations. The size of the database also impacts locality and the maximum throughput benefit, as discussed in 3.2. Because Tectonic Metadata has a larger DB size than WhatsApp or Laser the throughput benefit is expected to be smaller for each configuration. Looking at application level



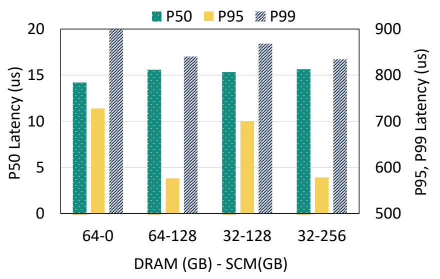
(a) **WhatsApp:** throughput comparison.



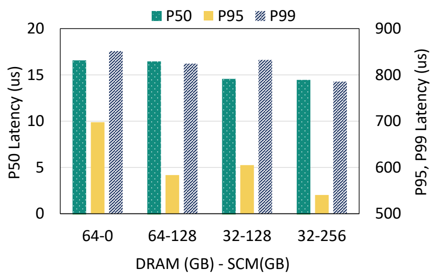
(b) **Tectonic Metadata:** throughput comparison.



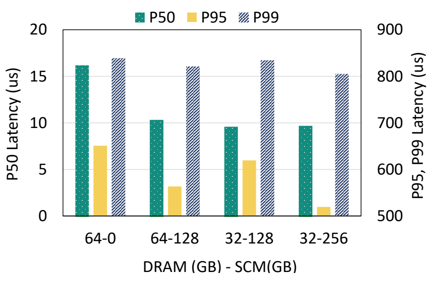
(c) **Laser:** throughput comparison.



(d) **WhatsApp:** latency comparison.

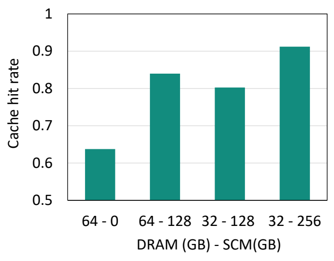


(e) **Tectonic Metadata:** latency comparison.

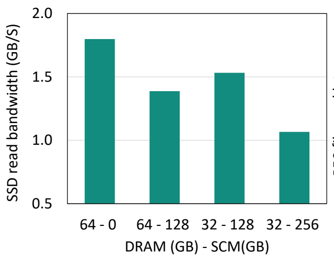


(f) **Laser:** latency comparison.

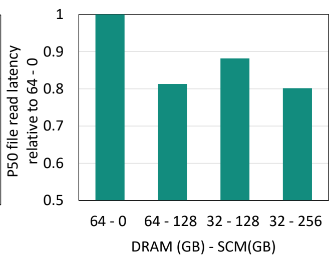
Figure 10: Throughput and application read latency comparison for WhatsApp, Tectonic Metadata and Laser for DRAM first admission policy using all server configurations shown in Table 4. (a,b, and c) Throughput comparisons (db operations/sec). Here the baseline is 64 - 0 configs. (d, e and f) absolute P50, P95, and P99 latencies for all workloads and server configs.



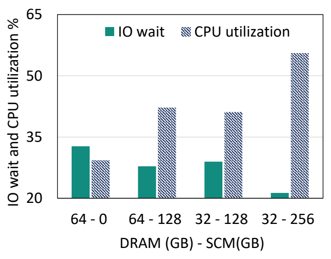
(a) Cache hit rate.



(b) SSD read bandwidth in GB/S.



(c) File read latency.



(d) Io wait and CPU utilization.

Figure 11: Cache utilization, IO read bandwidth, IO read latency, IO wait and CPU utilization of WhatsApp for DRAM 1st admission policy.

read latency in Figure 10d, 10e, and 10f, we observe that P50 latency is relatively stable for WhatsApp and Tectonic Metadata. While P50 latency does improve for Laser, the absolute magnitude of the improvement is less significant than the improvements to tail latency. P95 and P99 show an overall improvement of 20% and 10% respectively for all services. The P50 latencies primarily reflect situations where the data is obtained from DRAM. The benefit of SCM is reflected in P95 and P99 scenarios where in one case the data is in SCM, while the default case the data is in flash storage. Write latencies in all cases stay similar since we are only optimizing the block cache used for reads, while writes always go to the memtables residing in DRAM.

6.3 IO bandwidth, cache and CPU utilization

Figure 11 illustrates the cache hit rate, IO bandwidth, and latency improvement of DRAM first policy for WhatsApp. As seen in Figure 11a, the higher capacity of the block cache (sum of DRAM and SCM cache) leads to a higher cache hit rate. We show in Figure 11a, that for WhatsApp the hit rate increases up to 30% and the increase is correlated to the cache size. Tectonic Metadata and Laser (not shown here) also follow a similar pattern of increasing hit rates.

Another important indicator explaining throughput gain is SSD bandwidth utilization. As the cache hit rate increases for the server configurations with SCM it translates to less demand for read access from the SSD, and therefore decreased IO read bandwidth. Figure 11b shows for WhatsApp adding SCM reduces SSD read bandwidth by up to 0.8 GB/s, or roughly 25% of the SSD's datasheet max read bandwidth. Figure 11c shows the file read latency improvement for all server configurations relative to 64 - 0. Decreased demand for read IO bandwidth improves the P50 latency by up to 20%. Latencies at higher percentiles stay the same because there are still scenarios where the IO queue will be saturated with reads, which drives worst-case latency. But for the majority of the requests, latencies are improved because of the decrease in IO bandwidth. The other workloads also show similar patterns. Figure 11d shows the CPU utilization for all server configurations. We observe that the CPU utilization increases as the block cache size increases. This is due to the increase in CPU activity as we increase amount of data accessed with low IO wait latency from the cache. One thing to note is, even though CPU utilization increase for our new *1P server variants*, we can still safely service the workloads with 1 CPU even in the largest 256 SCM configuration.

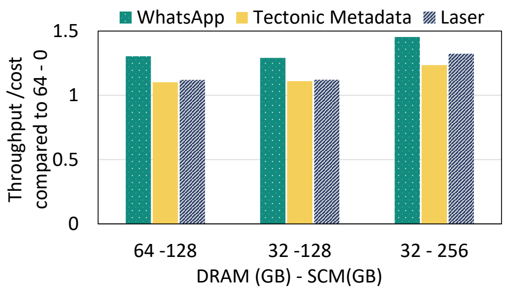


Figure 12: Throughput/cost of WhatsApp, Tectonic Metadata and Laser normalized to 64 - 0 1P servers throughput/cost.

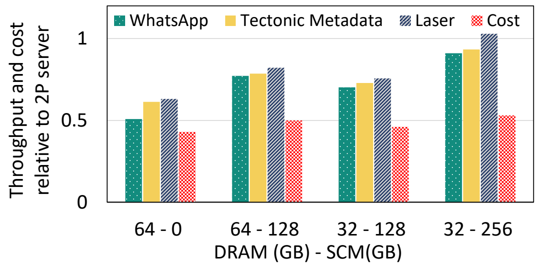


Figure 13: Throughput of 1P and its variants compared to 2P servers.

6.4 Cost and performance

In the above sections, we aim to understand the performance achieved for different DRAM and SCM variations of the *1P server* configuration. The large capacity of SCM per DIMM slot enables us to dramatically increase the memory capacity of the platform without impacting the server motherboard design. As seen in Table 4 adding SCM increases the cost of a server. Figure 12 estimates the performance per cost compared to baseline *1P server* (64 - 0) configuration. We observe in the figure that the 32 - 256 configuration gives the best cost relative to performance across all workloads. In this configuration the 23% cost increase over the 64 - 0 baseline produces a 50% - 80% performance improvement. To a smaller degree the 64-128 and 32-128 configurations also provide performance-relative cost benefits over the standard *1P server*. Notable is the fact that the benefit across these two configurations is nearly identical due to the proportional difference in DRAM cost vs. performance increase. If future DRAM/SCM hardware designs provide additional flexibility across capacity pricing then we may discover new configurations which achieve even larger TCO benefits.

In Figure 13, we present a throughput comparison between the *2P server* and the various *1P server* configurations. The figure shows that increasing the amount of SCM brings throughput closer to parity with the *2P server* for a minimal

Table 5: Performance equivalent TCO relative to 2P.

Server types	2P server	1P server	1P (32 - 256)
Relative TCO	1.0	0.72 - 0.86	0.52 - 0.57

increase in relative cost. While the baseline *1P server* (64 - 0) configuration only achieves 50 - 60% of the performance of a *2P server*, the 32 - 256 *1P* variant raises relative performance to 93 - 102%. By dividing the relative cost of the platforms (Table 4) by their relative performance (Figure 13) for each workload we derive the performance-equivalent TCOs in Table 5. In the case of the 32 - 256 configuration, improving *1P* performance with SCM improves the relative TCO to 0.52 - 0.57 of the *2P server*. Therefore, we demonstrate that deploying SCM configurations of *1P servers* instead of *2P servers* results in an **overall cost savings of 43% - 48%** across some of the largest RocksDB workloads at Facebook. Note that although SCM adds more power per rack (5-10%) the performance gain allows us to deploy fewer racks, hence it's evidently a power win compared to the 64-0 server.

General takeaway: From this project, we have learned that SCM creates cost-effective solutions to increase performance in data centers. We have observed that based on the workload hotness/locality we can utilize DRAM and SCM to serve most workloads efficiently.

Even though the performance and cost of using SCMs are impressive for the chosen workloads, and the performance gain can be translated to other read-dominated workloads in our environment, a discussion on whether we should have a deployment of SCMs in Facebook at scale is outside the scope of this paper. We briefly discuss some additional challenges of mass-scale deployment:

Workloads: Section 3.2 talks about the class of applications that would benefit from SCM. We have also identified a number of write-heavy workloads at Facebook that would not benefit from SCMs.

Reliability: Since SCMs are not widely available in the market, the reliability of SCMs is a concern until they have been proven in mass deployments.

SKU diversification: Adding a new hardware configuration into the fleet requires consideration of other costs like maintaining a separate code path and creating a new validation and sustainability workflow. This complexity and cost will be added to the practical TCO of any new platform deployment.

7 Discussion and future work

In the previous section, we demonstrated that KV Stores based on RocksDB are examples of the potential advantages of SCM in a large data center. In the future, we want to experiment with extending the memory capacity of other large memory footprint workloads using SCM. Some candidate large scale workloads that will profit from large memory capacity are Memcached [61, 62] and graph cache [27]. Memcached is a distributed in-memory key-value store deployed in large data centers. Optimizing Memcached to utilize SCM will enable an extension of memory beyond the capacity of DRAM.

Graph cache is a distributed read optimized storage for social graphs that exploits memory for graph structure-aware caching. These workloads are read-dominated and have random memory accesses that can benefit from the high density and byte-addressable features of SCM. Although in this paper we did not leverage the persistent capability of SCM for RocksDB uncompressed block cache, in the future we want to study the benefits of fast persistent SCM for the memtable and SST files. We also want to explore with SCM is its performance via emerging connectivity technology such as Compute Express Link (CXL) [63]. The workloads we analyzed in this paper are more latency bound than memory bandwidth bound but, for high memory bandwidth-demanding services, sharing DRAM and SCM on the same bus will create interference. In such cases having dedicated SCM access via CXL will avoid contention, but at the same time will potentially increase data access latency, requiring careful design consideration.

8 Related work

Performance analysis and characterization in DCPMM: Recent studies proved the potential of commercially available Intel® Optane™ memory for various application domains. [19, 64] has determined the performance improvement of DCPMM in memory and app-Direct modes for graph applications. [20, 21] evaluated the performance of DCPMM platform as volatile memory compared to DRAM for HPC applications. DCPMM performance for database systems were shown in [16, 65–68] both as a memory extension and for persisting data. Works such as [15, 18, 24, 69] also shows the characteristics and evaluations of DCPMM when working alone or alongside of DRAM. Specifically, [18] has identified the deviation of DCPMM characteristics from earlier simulation assumptions. While these works shed light on the usage of DCPMM for data-intensive applications, in our work, based on the memory characteristic findings of these work, we analyzed the performance of DCPMM for large data datacenter production workloads. Our work focus on utilizing the DCPMM platform to the best of its capability and study its possible usage as a cost-effective memory extension for future data-center system designs.

Hybrid DRAM-SCM systems: Previous works studied hybrid DRAM-SCM systems to understand how we can utilize these memories with different characteristics, together and how they influence the existing system and software designs. [70–73] have shown the need for a redesign of existing key-value stores and database systems to take into account the access latency differences between DRAM and SCM. Similarly, by noting the latency differences in these memories, we carefully place hot blocks in DRAM and colder blocks in SCM in our implementations. When deploying hybrid memory, another question that arises is, how to manage data placement between DRAM and SCM. In these aspects, [74] demonstrated efficient page migration between DRAM and SCM based on the memory access pattern observed in the memory controller. In addition, [75–83] perform data/page transfer by profiling and tracking information such as memory access patterns, read/write intensity to a page/data, resource utilization by workloads, and memory features of DRAM and SCM, in hardware, OS, and application level. These works aim to generalize usage of DRAM and SCM to various workloads without involving the application developer, hence requiring hardware and software monitoring that is transparent to the application developers. But in our case, the RocksDB application-level structure exposes separate reads and writes paths and frequency of access of data block. These motivated us to implement our designs in software without requiring any additional overhead in the OS and hardware.

RocksDB performance improvements: [84] demonstrated how to decrease the memory footprint of MyRocks, which is built on top of RocksDB, using block access based nonvolatile memory (NVM) by implementing secondary block cache. While their methods also decrease DRAM size required in the system, the block-based nature of NVM increases read amplification. This is because, the key-value size in RocksDB is significantly less than the size of the block, whereas in our methods, using byte addressable SCM avoids such issues.

9 Conclusion

The increasing cost of DRAM has influenced data centers to design servers with lower DRAM per compute ratio. These servers have shown to decrease the TCO for scalable workloads. Nevertheless, this type of system design diminishes the performance of large memory footprint workloads that relies on DRAM to cache hot data. Key Value stores based on RocksDB is one such class of workloads that is affected by the reduction of DRAM size. In this paper, we propose using Intel® Optane™ PMem 100 Series SCMs (DCPMM) in AppDirect mode to extend the available memory for RocksDB to mitigate performance loss in smaller DRAM designs while still maintaining the desired lower TCO of smaller DRAM systems. We carefully studied and redesigned the block cache to utilize DRAM for the placement of hot blocks and SCM for colder blocks. Our evaluations show up to 80% improvement to throughput and 20% improvement in P95 latency over the existing small DRAM platform when we utilize SCM alongside DRAM, while still reducing the cost by 43-48% compared to large DRAM designs. To our knowledge, this is the first paper that demonstrates practical cost-performance trade-offs for potential deployment of DCPMM in commercial datacenters.

Acknowledgments

We like to thank our shepherd, Tudor David, and the anonymous reviewers for their thorough comments that greatly improved this work. We are also grateful to Yanqin Jin, Jesse Koh, Darryl Gardner, Siying Dong, Pallab Bhattacharya, Shumin Zhao, and many others at Facebook for their support and suggestions in this research project.

References

[1] Facebook. Introducing "yosemite": the first open source modular chassis for high-powered microserver. 2015. https://engineering.fb.com/core-data/introducing-yosemite-the-first-open-source-modular-chassis-for-high-powered-microservers/.

[2] Facebook. Rocksdb. 2020. https://rocksdb.org/.

[3] Intel. Intel® optane™ persistent memory. 2019. https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html.

[4] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, page 24–33, New York, NY, USA, 2009. Association for Computing Machinery.

[5] J. Zhao, S. Li, D. H. Yoon, Y. Xie, and N. P. Jouppi. Kiln: Closing the performance gap between systems with and without persistence support. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 421–432, 2013.

[6] Ju-Young Jung and Sangyeun Cho. Memorage: Emerging persistent ram based malleable main memory and storage architecture. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, page 115–126, New York, NY, USA, 2013. Association for Computing Machinery.

[7] Yonatan Gottesman, Joel Nider, Ronen Kat, Yaron Weinsberg, and Michael Factor. Using storage class memory efficiently for an in-memory database. In *Proceedings of the 9th ACM International on Systems and Storage Conference*, SYSTOR '16, New York, NY, USA, 2016. Association for Computing Machinery.

[8] J. Jeong, J. Hong, S. Maeng, C. Jung, and Y. Kwon. Unbounded hardware transactional memory for a hybrid dram/nvm memory system. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 525–538, 2020.

[9] Haris Volos, Andres Jaan Tack, and Michael M. Swift. Mnemosyne: Lightweight persistent memory. *SIGARCH Comput. Archit. News*, 39(1):91–104, March 2011.

[10] Joel Coburn, Adrian M. Caulfield, Ameen Akel, Laura M. Grupp, Rajesh K. Gupta, Ranjit Jhala, and Steven Swanson. Nv-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, page 105–118, New York, NY, USA, 2011. Association for Computing Machinery.

[11] Lu Zhang and Steven Swanson. Pangolin: A fault-tolerant persistent memory programming library. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 897–912, Renton, WA, July 2019. USENIX Association.

[12] Jeremy Condit, Edmund B. Nightingale, Christopher Frost, Engin Ipek, Benjamin Lee, Doug Burger, and Derrick Coetzee. Better i/o through byte-addressable, persistent memory. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, page 133–146, New York, NY, USA, 2009. Association for Computing Machinery.

[13] Jiaxin Ou, Jiwu Shu, and Youyou Lu. A high performance file system for non-volatile main memory. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, New York, NY, USA, 2016. Association for Computing Machinery.

[14] Subramanya R. Dulloor, Sanjay Kumar, Anil Keshavamurthy, Philip Lantz, Dheeraj Reddy, Rajesh Sankaran, and Jeff Jackson. System software for persistent memory. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, New York, NY, USA, 2014. Association for Computing Machinery.

[15] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R. Dulloor, Jishen Zhao, and Steven Swanson. Basic performance measurements of the intel optane dc persistent memory module, 2019.

[16] Mikhail Zarubin, Patrick Damme, Dirk Habich, and Wolfgang Lehner. Polymorphic compressed replication of columnar data in scale-up hybrid memory systems. In *Proceedings of the 13th ACM International Systems and Storage Conference*, SYSTOR '20, page 98–110, New York, NY, USA, 2020. Association for Computing Machinery.

[17] Kai Wu, Frank Ober, Shari Hamlin, and Dong Li. Early evaluation of intel optane non-volatile memory with hpc i/o workloads, 2017.

[18] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steve Swanson. An empirical guide to the behavior and use of scalable persistent memory. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 169–182, Santa Clara, CA, February 2020. USENIX Association.

[19] Gurbinder Gill, Roshan Dathathri, Loc Hoang, Ramesh Peri, and Keshav Pingali. Single machine graph analytics on massive datasets using intel optane dc persistent memory. *Proceedings of the VLDB Endowment*, 13(10):1304–1318, Jun 2020.

[20] Onkar Patil, Latchesar Ionkov, Jason Lee, Frank Mueller, and Michael Lang. Performance characterization of a dram-nvm hybrid memory architecture for hpc applications using intel optane dc persistent memory modules. In *Proceedings of the International Symposium on Memory Systems*, MEMSYS '19, page 288–303, New York, NY, USA, 2019. Association for Computing Machinery.

[21] Ivy Peng, Kai Wu, Jie Ren, Dong Li, and Maya Gokhale. Demystifying the performance of hpc scientific applications on nvm-based memory systems. *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2020.

[22] Teng Ma, Mingxing Zhang, Kang Chen, Zhuo Song, Yongwei Wu, and Xuehai Qian. Asymnvm: An efficient framework for implementing persistent data structures on asymmetric nvm architecture. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 757–773, New York, NY, USA, 2020. Association for Computing Machinery.

[23] Thomas E. Anderson, Marco Canini, Jongyul Kim, Dejan Kostić, Youngjin Kwon, Simon Peter, Waleed Reda, Henry N. Schuh, and Emmett Witchel. Assise: Performance and availability via client-local NVM in a distributed file system. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1011–1027. USENIX Association, November 2020.

[24] Z. Wang, X. Liu, J. Yang, T. Michailidis, S. Swanson, and J. Zhao. Characterizing and modeling non-volatile memory systems. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 496–508, 2020.

[25] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. Workload analysis of a large-scale key-value store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, page 53–64, New York, NY, USA, 2012. Association for Computing Machinery.

[26] Xiao Shi, Scott Pruett, Kevin Doherty, Jinyu Han, Dmitri Petrov, Jim Carrig, John Hugg, and Nathan Bronson. Flighttracker: Consistency across read-optimized online stores at facebook. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 407–423. USENIX Association, November 2020.

[27] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov, Dmitri Petrov, Lovro Puzar, Yee Jiun Song, and Venkat Venkataramani. TAO: Facebook's distributed data store for the social graph. In *2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pages 49–60, San Jose, CA, June 2013. USENIX Association.

[28] zhichao Cao, Siying Dong, Sagar Vemuri, and David H.C. Du. Characterizing, modeling, and benchmarking rocksdb key-value workloads at facebook. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 209–223, Santa Clara, CA, February 2020. USENIX Association.

[29] Facebook. db_bench. 2020. https://github.com/facebook/rocksdb/wiki/Benchmarking-tools#db_bench.

[30] Google. Leveldb. 2011. https://dbdb.io/db/leveldb.

[31] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, page 205–220, New York, NY, USA, 2007. Association for Computing Machinery.

[32] Redis. Redis. 2020. https://redis.io/.

[33] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. The log-structured merge-tree (lsm-tree). *Acta Inf.*, 33(4):351–385, June 1996.

[34] Facebook. Rocksdb users and use cases. 2020. https://github.com/facebook/rocksdb/wiki/RocksDB-Users-and-Use-Cases.

[35] Facebook. Rocksdb trace, replay, analyzer, and workload generation. 2020. https://github.com/facebook/rocksdb/wiki/RocksDB-Trace%2C-Replay%2C-Analyzer%2C-and-Workload-Generation.

[36] Micron. 3d xpoint technology. 2020. https://www.micron.com/products/advanced-solutions/3d-xpoint-technology.

[37] Intel. 3d xpoint™: A breakthrough in non-volatile memory technology. 2020.

https://www.intel.com/content/www/us/en/architecture-and-technology/intel-micron-3d-xpoint-webcast.html.

[38] Intel. Intel optane dc persistent memory. 2020. https://www.intel.com/content/dam/support/us/en/documents/memory-and-storage/data-center-persistent-mem/Intel-Optane-DC-Persistent-Memory-Quick-Start-Guide.pdf.

[39] IgorsIstocniks. How whatsapp moved 1.5b users across datacenters. 2019. https://docplayer.net/161220289-How-whatsapp-moved-1-5b-users-across-data-senters-igors-istocniks-code-beam-sf-2019.html.

[40] Muthu Annamalai. Zippydb: a modern, distributed key-value data store. 2015. https://www.youtube.com/watch?v=DfiN7pG0D0k.

[41] Satadru Pan, Theano Stavrinos, Yunqiao Zhang, Atul Sikaria, Pavel Zakharov, Abhinav Sharma, Shiva Shankar P, Mike Shuey, Richard Wareing, Monika Gangapuram, Guanglei Cao, Christian Preseau, Pratap Singh, Kestutis Patiejunas, JR Tipton, Ethan Katz-Bassett, and Wyatt Lloyd. Facebook's tectonic filesystem: Efficiency from exascale. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 217–231. USENIX Association, February 2021.

[42] Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang, and Sanjeev Kumar. f4: Facebook's warm BLOB storage system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 383–398, Broomfield, CO, October 2014. USENIX Association.

[43] Guoqiang Jerry Chen, Janet L Wiener, Shridhar Iyer, Anshul Jaiswal, Ran Lei, Nikhil Simha, Wei Wang, Kevin Wilfong, Tim Williamson, and Serhat Yilmaz. Real-time data processing at facebook. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1087–1098, 2016.

[44] Ashish Thusoo, Zheng Shao, Suresh Anthony, Dhruba Borthakur, Namit Jain, Joydeep Sen Sarma, Raghotham Murthy, and Hao Liu. Data warehousing and analytics infrastructure at facebook. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1013–1020, 2010.

[45] Facebook. Hive - a petabyte scale data warehouse using hadoop. 2009. https://www.facebook.com/notes/facebook-engineering/hive-a-petabyte-scale-data-warehouse-using-hadoop/89508453919/.

[46] CK Chow. On optimization of storage hierarchies. *IBM Journal of Research and Development*, 18(3):194–203, 1974.

[47] Intel. Ipmctl. 2020. https://github.com/intel/ipmctl.

[48] Ndctl and daxctl. 2020. https://github.com/pmem/ndctl.

[49] memkind library. 2020. https://github.com/memkind/memkind.

[50] Intel. Intel server board s2600wftr specification. 2019. https://ark.intel.com/content/www/us/en/ark/products/192581/intel-server-board-s2600wftr.html.

[51] Intel. Intel xeon gold 6252 processor specification. 2019. https://ark.intel.com/content/www/us/en/ark/products/192447/intel-xeon-gold-6252-processor-35-75m-cache-2-10-ghz.html.

[52] OCP. Ocp tioga pass 2s server design specification v1.1. 2018. https://www.opencompute.org/documents/open-compute-project-fb-2s-server-tioga-pass-v1p1-1-pdf.

[53] OCP. Ocp twin lakes 1s server design specification v1. 2018. https://www.opencompute.org/documents/facebook-twin-lakes-1s-server-design-specification.

[54] Hyperscalers. Rackgo x yosemite valley. 2019. https://www.hyperscalers.com/Rackgo-X-Yosemite-Valley.

[55] Hyperscalers. Rackgo x leopard cave. 2019. https://www.hyperscalers.com/OCP-Hyperscale-Systems?product_id=194.

[56] Jim Handy. Intel's optane dimm price model. 2019. https://thememoryguy.com/intels-optane-dimm-price-model/#more-2291.

[57] JPaul Alcorn. Intel optane dimm pricing. 2019. https://www.tomshardware.com/news/intel-optane-dimm-pricing-performance,39007.html.

[58] MATLAB. Fit power series models using the fit function. 2020. https://www.mathworks.com/help/curvefit/power.html.

[59] MATLAB. gpfit: Generalized pareto parameter estimates. 2020. https://www.mathworks.com/help/stats/gpfit.htmll.

[60] MATLAB. Fit sine models using the fit function. 2020. https://www.mathworks.com/help/curvefit/sum-of-sine.html.

[61] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, and Venkateshwaran Venkataramani. Scaling memcache at facebook. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 385–398, Lombard, IL, April 2013. USENIX Association.

[62] Brad Fitzpatrick. Distributed caching with memcached. *Linux Journal*, 30(10):2223–2236, 2004.

[63] Compute express link™: The breakthrough cpu-to-device interconnect. 2020. https://www.computeexpresslink.org/.

[64] Ivy B. Peng, Maya B. Gokhale, and Eric W. Green. System evaluation of the intel optane byte-addressable nvm. *Proceedings of the International Symposium on Memory Systems*, Sep 2019.

[65] Alexander van Renen, Lukas Vogel, Viktor Leis, Thomas Neumann, and Alfons Kemper. Persistent memory i/o primitives. *Proceedings of the 15th International Workshop on Data Management on New Hardware - DaMoN'19*, 2019.

[66] Georgios Psaropoulos, Ismail Oukid, Thomas Legler, Norman May, and Anastasia Ailamaki. Bridging the latency gap between nvm and dram for latency-bound operations. In *Proceedings of the 15th International Workshop on Data Management on New Hardware*, DaMoN'19, New York, NY, USA, 2019. Association for Computing Machinery.

[67] Anil Shanbhag, Nesime Tatbul, David Cohen, and Samuel Madden. Large-scale in-memory analytics on intel® optane™ dc persistent memory. In *Proceedings of the 16th International Workshop on Data Management on New Hardware*, DaMoN '20, New York, NY, USA, 2020. Association for Computing Machinery.

[68] Yinjun Wu, Kwanghyun Park, Rathijit Sen, Brian Kroth, and Jaeyoung Do. Lessons learned from the early performance evaluation of intel optane dc persistent memory in dbms. *Proceedings of the 16th International Workshop on Data Management on New Hardware*, Jun 2020.

[69] S. Imamura and E. Yoshida. Fairhym: Improving inter-process fairness on hybrid memory systems. In *2020 9th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, pages 1–6, 2020.

[70] Fei Xia, Dejun Jiang, Jin Xiong, and Ninghui Sun. Hikv: A hybrid index key-value store for dram-nvm memory systems. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 349–362, Santa Clara, CA, July 2017. USENIX Association.

[71] Yihe Huang, Matej Pavlovic, Virendra Marathe, Margo Seltzer, Tim Harris, and Steve Byan. Closing the performance gap between volatile and persistent key-value stores using cross-referencing logs. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 967–979, Boston, MA, July 2018. USENIX Association.

[72] Katelin A. Bailey, Peter Hornyack, Luis Ceze, Steven D. Gribble, and Henry M. Levy. Exploring storage class memory with key value stores. In *Proceedings of the 1st Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads*, INFLOW '13, New York, NY, USA, 2013. Association for Computing Machinery.

[73] Ismail Oukid, Johan Lasperas, Anisoara Nica, Thomas Willhalm, and Wolfgang Lehner. Fptree: A hybrid scm-dram persistent and concurrent b-tree for storage class memory. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 371–386, New York, NY, USA, 2016. Association for Computing Machinery.

[74] Luiz E. Ramos, Eugene Gorbatov, and Ricardo Bianchini. Page placement in hybrid memory systems. In *Proceedings of the International Conference on Supercomputing*, ICS '11, page 85–95, New York, NY, USA, 2011. Association for Computing Machinery.

[75] Z. Zhang, Y. Fu, and G. Hu. Dualstack: A high efficient dynamic page scheduling scheme in hybrid main memory. In *2017 International Conference on Networking, Architecture, and Storage (NAS)*, pages 1–6, 2017.

[76] S. Bock, B. R. Childers, R. Melhem, and D. Mossé. Concurrent migration of multiple pages in software-managed hybrid main memory. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pages 420–423, 2016.

[77] Haikun Liu, Yujie Chen, Xiaofei Liao, Hai Jin, Bingsheng He, Long Zheng, and Rentong Guo. Hardware/software cooperative caching for hybrid dram/nvm memory architectures. In *Proceedings of the International Conference on Supercomputing*, ICS '17, New York, NY, USA, 2017. Association for Computing Machinery.

[78] L. Liu, S. Yang, L. Peng, and X. Li. Hierarchical hybrid memory management in os for tiered memory systems. *IEEE Transactions on Parallel and Distributed Systems*, 30(10):2223–2236, 2019.

[79] K. Wu, J. Ren, and D. Li. Runtime data management on non-volatile memory-based heterogeneous memory for task-parallel programs. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 401–413, 2018.

[80] Ahmad Hassan, Hans Vandierendonck, and Dimitrios S. Nikolopoulos. Software-managed energy-efficient hybrid dram/nvm main memory. In *Proceedings of the 12th ACM International Conference on Computing Frontiers*, CF '15, New York, NY, USA, 2015. Association for Computing Machinery.

[81] H. Chang, Y. Chang, T. Kuo, and H. Li. A light-weighted software-controlled cache for pcm-based main memory systems. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 22–29, 2015.

[82] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu. Utility-based hybrid memory management. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 152–165, 2017.

[83] G. Dhiman, R. Ayoub, and T. Rosing. Pdram: A hybrid pram and dram main memory system. In *2009 46th ACM/IEEE Design Automation Conference*, pages 664–669, 2009.

[84] Assaf Eisenman, Darryl Gardner, Islam AbdelRahman, Jens Axboe, Siying Dong, Kim Hazelwood, Chris Petersen, Asaf Cidon, and Sachin Katti. Reducing dram footprint with nvm in facebook. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery.