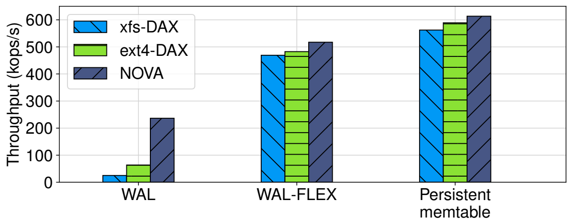

Figure 4. RocksDB SET throughput. Appends to the write-ahead log (WAL) file limit RocksDB throughput on NVMM file systems. Using FLEX writes improves performance by 2.2× to 18.7×. Replacing the skip-list and the log with a crash-consistent, persistent skip-list improves throughput by another 19% on average.

hashtable does not support resizing as it requires journaling mechanism to guarantee consistency.

RocksDB RocksDB [22] is a high-performance embedded key-value store based on log-structured merge trees (LSM-trees). When applications write data to a LSM-tree, RocksDB inserts the data to a skip-list in DRAM, and appends the data to a write-ahead log (WAL) file. When the skip-list is full, RocksDB writes it to disk and discards the log file.

Figure 4 measures RocksDB SET throughput with 20-byte keys and 100-byte values. RocksDB's default settings perform poorly on xfs-DAX and ext4-DAX, because each append requires journaling for those file systems. NOVA performs better because it avoids this cost.

RocksDB benefits from FLEX as well. It improves throughput by 2.2× - 18.7× and eliminates the performance gap between file systems.

Since the skip-list contains the same information as the WAL file, we eliminate the WAL file by making the skip-list a persistent data structure, similar to NoveLSM [32] based on LevelDB. The final bars in Figure 4 measure the performance of RocksDB with a crash-consistent skip-list in NVMM. Performance improves by 11× compared to the RocksDB baseline but just 19% compared to optimizing WAL with FLEX.

3.4 Evaluating FLEX

In general, FLEX involves replacing conventional file operations with similar DAX-based operations to avoid entering the kernel. We have applied FLEX techniques by hand to the SQLite, RocksDB, and Kyoto Cabinet, but they could easily be encapsulated in a simple library.

FLEX replaces `open()` with `open()` followed by DAX-`mmap()` to map the file into the application's address space. Then, the application can replace `read()` and `write()` system calls with userspace operations.

A FLEX write first checks if the file will grow as a result of the write. If so, the application can expand the file using `fallocate()` and `mmap()` or `mremap()` to expand the mapping. To amortize the cost of `fallocate()`, the application can extend the file by more than the write requires.

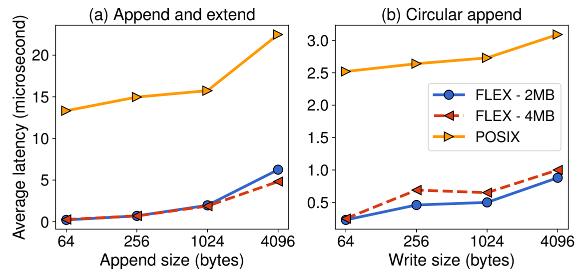

Figure 5. The Impact of FLEX File Operations. Emulating file accesses in user space can improve performance for a wide range of access patterns. Note that the Y axes have different scales. "-2 MB" and "-4 MB" denote different `fallocate()` sizes.

Once space is available, a FLEX write uses non-temporal stores to copy data into the file. If the write needs to be synchronous the application issues an `sfence` instruction to ensure the stores have completed. FLEX also uses an `sfence` instruction to replace `fsync()`.

FLEX reads are simpler: They simply translate to `memcpy()`.

FLEX requires the application to track a small amount of extra state about the file, including its location in memory, its current write point, and its current allocated size.

FLEX operations provide semantics that are similar to POSIX, but there are important and potentially subtle differences. First, operations are not atomic. Second, POSIX semantics for shared file descriptors are lost. We have not found these differences to be relevant for the performance-critical file operations in the workloads we have studied. We elaborate this point in Section 3.4.1.

To understand when FLEX improves performance, we constructed a simple microbenchmark that opens a file, and performs a series of reads or writes each followed by `fsync()`. We vary the size and number of operations and the amount of file space we pre-allocate with `fallocate()`. Figure 5 shows results for two different cases: "Append and extend" uses FLEX to emulate append operations that always cause the file to grow. "Circular append" reuses the same file area and avoids the need to allocate more space. The applications we studied use both models to implement logging: RocksDB uses "append and extend" whereas SQLite and Kyoto Cabinet use "circular append."

The data show that FLEX outperforms normal write operations by up to 61× for append and extend and up to 11× for circular append. The larger speedup for append and extend is due to the NVMM allocation overhead. Performance gains are especially large for small writes, a common case in the applications we studied.

For use cases that must extend the file, minimizing the cost of space allocation is critical. The results in the figure use 2 MB pages to minimize paging overheads. With 4 KB pages, FLEX only provides speedups for transfers under 4 KB.

Our experience with applying FLEX to RocksDB, SQLite, and Kyoto Cabinet shows that it can provide substantial performance benefits for very little effort. In contrast to re-implementing data structures to be crash-consistent, FLEX requires little to no changes to application logic and requires no additional logging or locking protocols. The only subtleties lie in determining that strict POSIX semantics are not necessary.

These results show that FLEX can provide an easy, incremental, and high-value path for developers creating new applications for NVMM or migrating existing code. It also reduces the importance of using a native NVMM file system, further easing migration, since FLEX performance depends little on the underlying file system.

The Strata file system [37] provides some of the same advantages as FLEX through userspace logging through a library that communicates with the in-kernel file system. Their results show that coupling the user space interface to the underlying file system leads to good performance. Their interface makes strong atomicity guarantees while FLEX lets the application enforce the semantics it requires.

3.4.1 Correctness

Since FLEX is not atomic, applying it to applications that assume atomic writes is likely to cause a correctness problem. To our knowledge, SQLite, RocksDB, and Kyoto Cabinet do not assume the atomicity of write system calls [51], thereby applying FLEX does not break their application logic. Only LMDB assumes that 512 bytes sector writes are atomic [13]. Therefore, running it on NVMM file systems introduces the correctness problem since only 8 bytes are atomic on NVMM. To solve this problem, we added a checksum for the LMDB metadata: When a checksum error is detected, LMDB falls back to the previous header.

3.5 Best Practices

Based on our experiences with these five applications, we can draw some useful conclusions about how applications can profitably exploit NVMMs.

Use FLEX Emulating file operations in user space provides large performance gains for very little programmer effort.

Use fine-grained cache flushing instead of `msync` Applications that already use `mmap` and `msync` to access data and ensure consistency, can improve performance significantly by flushing cache lines rather than `msync`'ing pages. However, ensuring that all updated cache lines are flushed correctly can be a challenge.

Use complex persistent data structure judiciously For both of the DRAM data structures we made persistent, the programming effort required was significant and likely performance gains were relatively small relative to FLEX.

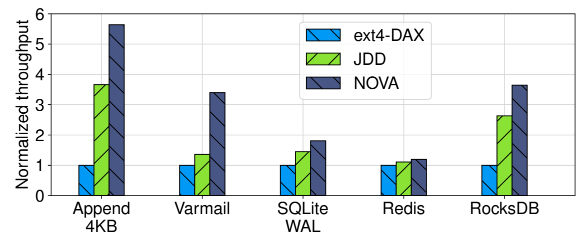

Figure 6. JDD performance. Fine-grained, DAX-optimized journaling on NVMM improves performance for metadata-intensive applications.

This finding leads us to two conclusions: First, it is critical to make building persistent data structures in NVMM as easy as possible. Second, it is wise to estimate the potential performance impact the persistent data structure will have before investing a large amount of programmer effort in developing it [41].

Preallocate files on adapted NVMM file systems Several of the performance problems we found with adapted NVMM file systems stemmed from storage allocation overheads. Using `fallocate` to pre-allocate file space eliminated them.

Avoid meta-data operations Directory operations (e.g., deleting files) and storage allocation incurred journaling overheads in both xfs and ext4. Avoiding them improves performance, but this is not always possible.

3.6 Reducing journaling overhead

Several of the best practices we identify above focus on avoiding metadata operations since they are often slow. This can be awkward and some metadata operations are unavoidable, so improving their performance would make adapting to NVMMs easier and improve performance.

NOVA's mechanism for performing consistent metadata updates is tailored specifically for NVMMs, but ext4 and xfs' journaling mechanisms were built for disk, and this legacy is evident in their poorer metadata performance.

Ext4 uses the journaling block device (JBD2) to perform consistent metadata updates. To ensure atomicity, it always writes entire 4 KB pages, even if the metadata change affects a single byte. Transactions often involve multiple metadata pages. For instance, appending 4 KB data to a file and then calling `fsync` writes one data page and eight journal pages: a header, a commit block, and up to six pages for inode, inode bitmap, and allocator.

JDB2 also allows no concurrency between journaled operations, so concurrent threads must synchronize to join the same running transaction, making the journaling a scalability bottleneck [56]. Son *et al.* [56] and iJournaling [50] have tried to fix ext4's scalability issues by reducing lock contention and adding per-core journal areas to JBD2.

Previous works [9, 10] has identified the inefficiencies of coarse-grain logging and proposed solutions in the context

	Native techniques		Optimizations (Lines changed)		
	WAL	mmap+msync	FLEX	CLWB+fence	Persistent Objects
SQLite	×	-	266	-	-
Kyoto Cabinet	×	×	133	48	-
LMDB	-	×	-	101	-
Redis	×	-	-	-	1326
RocksDB	×	-	56	-	380

Table 1. Application Optimization Summary The applications we studied used a variety of techniques to reliably store persistent state. All the optimizations we applied improved performance, but the amount of programmer effort varied widely. The data Figures 1, 2, 3, and 4 show that programmer effort does not correlate with performance gains.

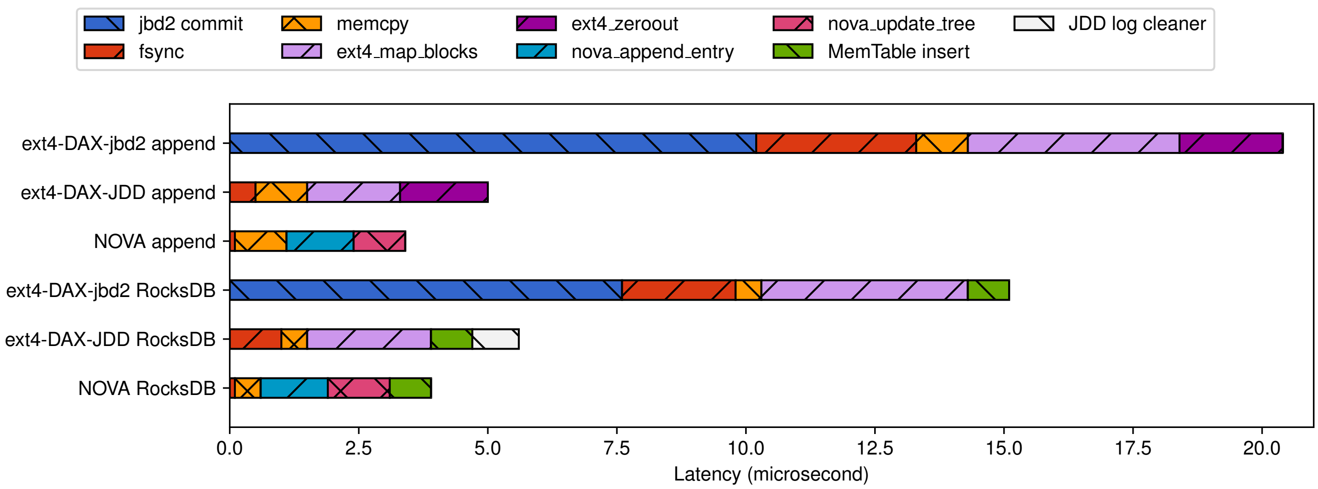

Figure 7. Latency break for 4KB append and RocksDB SET. JDD significantly reduces journaling overhead by eliminating JBD2 transaction commit, but still has higher latency than NOVA's metadata update mechanism.

of block-based file systems. FSMAC [10] maintains data in disk/SSD and metadata in NVMM, and uses undo log journaling for metadata consistency. The work in [9] journals redo log records of individual metadata fields to NVMM during transaction commit, and applies them to storage during checkpointing.

To understand how much of ext4's poor metadata performance is due to coarse-grain logging, we apply these fine-grain logging techniques to develop a journaling DAX device (JDD) for ext4 which performs DAX-style journaling on NVMM and provides improved scalability.

JDD makes three key improvements to JBD2. First, it journals individual metadata fields rather than entire pages. Second, it provides pre-allocated, per-CPU journaling areas so CPUs can perform journaled operations in parallel. Third, it uses undo logging in the journals: It copies the old values into the journal and performs updates directly to the metadata structures in NVMM. To commit an update it marks the journal as invalid. During recovery from a crash, the file system rolls back partial updates using the journaled data. These changes provide for very lightweight transaction commit and make checkpointing unnecessary.

JDD differs from the previous works by focusing on NVMM file systems. FSMAC aims to accelerate metadata updates for disk-based file systems by putting the metadata separately in NVMM. To handle the performance gap between NVMM and disk, FSMAC maintains multiple versions of metadata. The work in [9] optimizes ext4 using fine-grained redo logging on NVMM journal. We built JDD to improve the performance of adapted NVMM file systems using fine-grained undo logging, avoiding the complexity of previous works – managing versions in FSMAC or transaction committing and checkpointing in [9].

Strata [37] and Aerie [63] take a more aggressive approach and log updates in userspace under the control of file system-specific libraries. Metadata updates occur later and off the critical path. This approach should offer better performance than the techniques described above since it avoids entering the kernel for metadata updates. However, it also involves more extensive changes to the application.

Figure 6 shows JDD's impact on a microbenchmark that performs random 4 KB writes followed by `fsync`, Filebench [60] Varmail (which is metadata-intensive), and the three databases and key value stores we evaluated earlier that perform frequent metadata operations as part of WAL. The JDD improves the microbenchmark performance by 3.7× and varmail by 40%. For applications that use write-ahead logging, the benefits range from 11% to 2.6×.

We further analyze the latency of JDD for 4 KB appends and RocksDB SET operation and show the latency breakdown in Figure 7. In ext4-DAX, JBD2 transaction commit (jbd2_commit) occupies 50% of the total latency. JDD eliminates this overhead by performing undo logging. JDD also reduces ext4 overheads such as block allocation (ext4_map_blocks). The remaining performance gap between ext4 and NOVA (46%) is due to ext4's more complex design and its need to keep more persistent states in storage media. In particular (as discussed in Section 3.1) ext4 keeps its data block and inode allocator state continually up-to-date on disk.

The performance improvement on Redis and SQLite are smaller, because they have higher internal overheads. Redis spends most of its time on TCP transfers between the Redis server and the benchmark application, and SQLite spends over 40% of execution time parsing SQL and performing B-tree operations.

4 File System Scalability

We expect NVMM file systems to be subject to more onerous scalability demands than block-based filesystems due to the higher performance of the underlying media and the large amount of parallelism that modern memory hierarchies can support [4]. Further, since NVMMs attach to the CPU memory bus, the capacity of NVMM file systems will tend to scale with the number sockets (and cores) in the systems.

Many-core scalability is also a concern for conventional block-based file systems, and researchers have proposed potential solutions. SpanFS [31] shards file and directories across cores at a coarse granularity, requiring developers to distribute the files and directories carefully. ScaleFS [4] decouples the in-memory file system from the on-disk file system, and uses per-core operation logs to achieve high concurrency. ScaleFS was built on xv6, a research prototype kernel, which makes impossible to perform a good head-to-head comparison with our changes. However, we expect that applying its techniques and the Scalable Commutativity Rule [14] systematically to NVMM file systems (and the VFS layer) might yield further scaling improvements.

This section first describes the FxMark [46] benchmark suite. Then, we identify several operations that have scalability limitations and propose solutions.

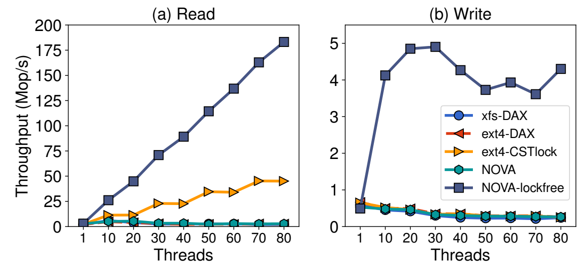

Figure 8. Concurrent 4KB read and write throughput. By default, Linux uses a non-scalable reader/writer lock to coordinate access to files. Using finer-grain, more scalable locks improves read and write scalability.

4.1 FxMark scalability test suite

Min *et al.* [46] built a file system scalability test suite called FxMark and used it to identify many scalability problems in both file systems and Linux's VFS layer. It includes nineteen tests of performance for data and metadata operations under varying levels of contention.

Min *et al.* use FxMark to identify scalability bottlenecks across many file systems. Interestingly, it is their analysis of tmpfs, a DRAM-based pseudo-file system that reveals the bottlenecks that are most critical for ext4-DAX, xfs-DAX, and/or NOVA.

We repeat their experiments and then develop solutions to improve scalability. The solutions we identify are sufficient to give good scalability with NVMM, but would probably also help disk-based file systems too.

FxMark includes nineteen workloads. Below, we only discuss those that show poor scalability for at least one the NVMM file systems we consider.

4.2 Concurrent file read/write

Concurrent `read` and `write` operations to a shared file are a well-known sore spot in file system performance. Figure 8 shows scalability problems for both reads and writes across ext4-DAX, xfs-DAX, and NOVA. The root cause of this poor performance is Linux's read/write semaphore implementation [5, 6, 33, 40]: It is not scalable because of the atomic update required to acquire and release it.

The semaphore protects two things: The file data and the metadata that describes the file layout. To remove this bottleneck in NOVA, we use separate mechanisms to protect the data and metadata.

To protect file data, we leverage NOVA's logs. NOVA maintains one log per inode. Many of the log entries correspond to write operations and hold pointers to the file pages that contain the data for the write. Rather than locking the whole inode, we use reader/writer locks on each log entry to protect the pages to which it links. Although this lock resides in NVMM, its state is not necessary for recovery and is cleared before use after a restart, so hot locks will reside in processor caches and not usually be subject to NVMM access latency.

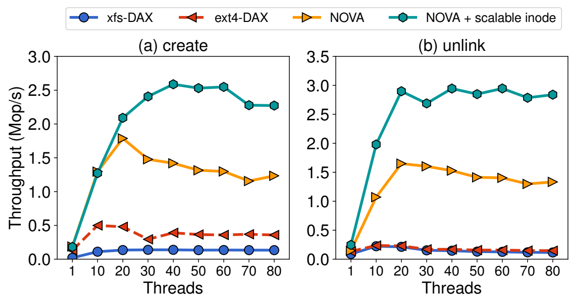

Figure 9. Concurrent `create` and `unlink` throughput. The `create` and `unlink` operations are not scalable even if performed in isolated directories, because Linux protects the global inode lists and inode cache with a single spinlock. Moving to per-cpu structures and fine-grain locks improves scalability above 20 cores.

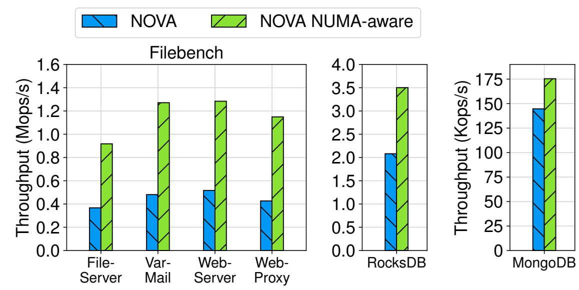

Figure 10. NUMA-awareness in the file system. Since NVMM is memory, NUMA effects impact performance. Providing simple controls over where the file system allocates NVMM for a file lets application run threads near the data they operate on, leading to higher performance.

NOVA's approach to tracking file layout makes protecting it simple. NOVA uses an in-DRAM radix tree to map file offsets to write entries in the log. Write operations update the tree and both reads and writes query it. Instead of using a lock we leverage the Linux radix tree implementation that uses read-copy update [42] to provide more scalable, concurrent access to a file.

Figure 8 shows the results (labeled "NOVA-lockfree") on our 80-core machine. 4 KB read performance scales from 2.9 Mops/s for one thread to 183 Mops/s with 80 threads (63×). The changes improve write performance as well, but write bandwidth saturates at twenty threads because our NVMM is attached to one of four NUMA nodes and each node has twenty threads.

Adding fine-grain locking for ranges of a file is possible for ext4-DAX and xfs-DAX, and it would improve performance when running on any storage device.

Using the radix tree to store file layout information would be more challenging since ext4 and xfs make updates to file layout information immediately persistent in the file's inode and indirect blocks. This is necessary to avoid reading the data from disk when the file is opened, which would be slow on block device. Since NVMM is much faster, NOVA can afford to scan the inode's log on `open` to construct the radix tree in DRAM.

An alternative solution for ext4 and xfs would be to replace VFS's per-inode reader/write semaphore with a CST semaphore [33] (or some other more scalable semaphore). The ext4-CSTlock line in the figure shows the impact on ext4-DAX: Performance scales from 2.1 Mops/s for one thread to 45 Mops/s for eighty threads (21×). The gains are not as large as the approach we implemented in NOVA, and they only apply to reads. Both of these approaches could coexist.

4.3 Directory Accesses

Scalable directory operations are critical in multi-program, data intensive workloads. Figure 9 shows that creating files in private directories only scales to twenty cores. Min *et al.* identify the root cause, but do not offer a solution: VFS takes a spinlock to add the new inode to the superblock's inode list and a global inode cache. The inode list includes all live inodes, and the inode cache provides a mapping from inode number to inode addresses.

We solve this problem and improve scalability for the inode list by breaking it into per-CPU lists and protecting each with a private lock. The global inode cache is an open-chaining hash table with 1,048,576 slots. We modify NOVA to use a per-core inode cache table. The table is distributed across the cores, each core maintains a radix tree that provides lock-free lookups, and threads on different cores can perform inserts concurrently. In Figure 9, the "NOVA + scalable inode" line shows the resulting improvements in scaling.

Updates to shared directories also scale poorly due to VFS locking. For every directory operation, VFS takes the inode mutexes of all the affected inodes, so operations in the shared directories are serialized. The `rename` operation is globally serialized at a system level in the Linux kernel for consistent updates of the dentry cache. Fixing these problems is beyond the scope of this paper, but recent work has addressed them [4, 61].

4.4 NUMA Scalability

Intelligently allocating memory in NUMA systems is critical to maximizing performance. Since a key task of NVMM file systems is allocating memory, these file systems should be NUMA-aware. Otherwise, poor data placement decisions will lead to poor performance [20].

We have added NUMA-aware features to NOVA to understand the impact they can have. We created a new `ioctl` that can set and query the preferred NUMA node for the file. A NUMA node represents a set of processors and memory regions that are close to one another in terms of memory access latency. The file system will try to use that node to allocate all the metadata and data pages for that file. A thread can use this `ioctl` along with Linux's CPU affinity mechanism to bind itself to the NUMA node where the file's data and metadata reside.

Figure 10(left) shows the result of Filebench workloads running with fifty threads. The NVMM is attached to NUMA node 0. Without the new mechanism, threads are spread across all the NUMA nodes, and some of them are accessing NVMM remotely. Binding threads to the NUMA node that holds the file they are accessing improves performance by 2.6× on average.

The other two graphs in Figure 10 measure the impact on RocksDB and MongoDB [47]. We modified RocksDB to schedule threads on the same NUMA node as the `SSTable` files using our `ioctl`, and ran `db_bench readrandom` benchmark with twenty threads. Similarly, we modified MongoDB to enable NUMA-aware thread scheduling, and ran read-intensive (95% read, 5% update) YCSB benchmark [18] with twenty threads. For both workloads, the data set size is 30 GB. The graphs show the result: NUMA-aware scheduling improves RocksDB and MongoDB performance by 68% and 21%, respectively.

5 Conclusion

We have examined the performance of NVMM storage software stacks to identify the bottlenecks and understand how both applications and the operating system should adapt to exploit NVMM performance.

We examined several applications and identified several simple techniques that provide significant gains. The most widely applicable of these use FLEX to move writes to user space, but implementing `msync` in userspace and assiduously avoiding metadata operations also help, especially on adapted NVMM file systems. Notably, our results show that FLEX can deliver nearly the same level of performance as building crash-consistent data structures in NVMM but with much less effort.

On the file system side, we evaluated solutions to the problems of inefficient logging in adapted NVMM file systems, multicore scaling limitations in file systems and the Linux's VFS layer, and the novel challenge of dealing with NUMA effects in the context of NVMM storage.

Overall, we find that although there are many opportunities for further improvement, the efforts of systems designers over the last several years to prepare systems for NVMM have been largely successful. As a result, there are a range of attractive paths for legacy applications to follow as they migrate to NVMM.

Acknowledgments

This work was supported in part by CRISP, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. We thank the anonymous reviewers for their insightful feedback. We are also thankful to Anton Gavriliuk and Thierry Fevrier from HP for their support and hardware access.

References

[1] Mihnea Andrei, Christian Lemke, Günter Radestock, Robert Schulze, Carsten Thiel, Rolando Blanco, Akanksha Meghlan, Muhammad Sharique, Sebastian Seifert, Surendra Vishnoi, Daniel Booss, Thomas Peh, Ivan Schreter, Werner Thesing, Mehul Wagle, and Thomas Willhalm. 2017. SAP HANA Adoption of Non-volatile Memory. *Proc. VLDB Endow.* 10, 12 (Aug. 2017), 1754–1765. https://doi.org/10.14778/3137765.3137780

[2] Joy Arulraj, Justin Levandoski, Umar Farooq Minhas, and Per-Ake Larson. 2018. Bztree: A High-performance Latch-free Range Index for Non-volatile Memory. *Proc. VLDB Endow.* 11, 5 (Jan. 2018), 553–565. https://doi.org/10.1145/3164135.3164147

[3] Meenakshi Sundaram Bhaskaran, Jian Xu, and Steven Swanson. 2013. Bankshot: Caching Slow Storage in Fast Non-volatile Memory. In *Proceedings of the 1st Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads (INFLOW '13).* ACM, New York, NY, USA, Article 1, 9 pages. https://doi.org/10.1145/2527792.2527793

[4] Srivatsa S. Bhat, Rasha Eqbal, Austin T. Clements, M. Frans Kaashoek, and Nickolai Zeldovich. 2017. Scaling a File System to Many Cores Using an Operation Log. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17).* ACM, New York, NY, USA, 69–86. https://doi.org/10.1145/3132747.3132779

[5] Silas Boyd-Wickizer, Austin T. Clements, Yandong Mao, Aleksey Pesterev, M. Frans Kaashoek, Robert Morris, and Nickolai Zeldovich. 2010. An Analysis of Linux Scalability to Many Cores. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10).* USENIX Association, Berkeley, CA, USA, 1–16. http://dl.acm.org/citation.cfm?id=1924943.1924944

[6] Silas Boyd-Wickizer, M Frans Kaashoek, Robert Morris, and Nickolai Zeldovich. 2012. Non-scalable locks are dangerous. In *Proceedings of the Linux Symposium.* 119–130.

[7] Matthew J. Breitwisch. 2008. Phase Change Memory. *Interconnect Technology Conference, 2008. IITC 2008. International* (June 2008), 219–221. https://doi.org/10.1109/IITC.2008.4546972

[8] Adrian M. Caulfield, Todor I. Mollov, Louis Alex Eisner, Arup De, Joel Coburn, and Steven Swanson. 2012. Providing safe, user space access to fast, solid state disks. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVII).* ACM, New York, NY, USA, 387–400. https://doi.org/10.1145/2150976.2151017

[9] Cheng Chen, Jun Yang, Qingsong Wei, Chundong Wang, and Mingdi Xue. 2017. Optimizing File Systems with Fine-grained Metadata Journaling on Byte-addressable NVM. *ACM Trans. Storage* 13, 2, Article 13 (May 2017), 25 pages. https://doi.org/10.1145/3060147

[10] Jianxi Chen, Qingsong Wei, Cheng Chen, and Lingkun Wu. 2013. FSMAC: A file system metadata accelerator with non-volatile memory. In *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on.* IEEE, 1–11.

[11] Dave Chinner. 2015. xfs: updates for 4.2-rc1. http://oss.sgi.com/archives/xfs/2015-06/msg00478.html.

[12] Youngdon Choi, Ickhyun Song, Mu-Hui Park, Hoeju Chung, Sanghoan Chang, Beakhyoung Cho, Jinyoung Kim, Younghoon Oh, Duckmin Kwon, Jung Sunwoo, Junho Shin, Yoohwan Rho, Changsoo Lee, Min Gu Kang, Jaeyun Lee, Yongjin Kwon, Soehee Kim, Jaehwan Kim, Yong-Jun Lee, Qi Wang, Sooho Cha, Sujin Ahn, H. Horii, Jaewook Lee, Kisung Kim, Hansung Joo, Kwangjin Lee, Yeong-Taek Lee, Jeihwan Yoo, and G. Jeong. 2012. A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International.* 46–48. https://doi.org/10.1109/ISSCC.2012.6176872

[13] Howard Chu. 2014. LMDB app-level crash consistency. https://www.openldap.org/lists/openldap-devel/201410/msg00004.html.

[14] Austin T. Clements, M. Frans Kaashoek, Nickolai Zeldovich, Robert T. Morris, and Eddie Kohler. 2015. The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors. *ACM Trans. Comput. Syst.* 32, 4, Article 10 (Jan. 2015), 47 pages. https://doi.org/10.1145/2699681

[15] Joel Coburn, Adrian M. Caulfield, Ameen Akel, Laura M. Grupp, Rajesh K. Gupta, Ranjit Jhala, and Steven Swanson. 2011. 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 '11)*. ACM, New York, NY, USA, 105–118. https://doi.org/10.1145/1950365.1950380

[16] Nachshon Cohen, David T. Aksun, and James R. Larus. 2018. Object-oriented Recovery for Non-volatile Memory. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 153 (Oct. 2018), 22 pages. https://doi.org/10.1145/3276523

[17] Jeremy Condit, Edmund B. Nightingale, Christopher Frost, Engin Ipek, Benjamin Lee, Doug Burger, and Derrick Coetzee. 2009. Better I/O through byte-addressable, persistent memory. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09)*. ACM, New York, NY, USA, 133–146. https://doi.org/10.1145/1629575.1629589

[18] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10)*. ACM, New York, NY, USA, 143–154. https://doi.org/10.1145/1807128.1807152

[19] Mingkai Dong and Haibo Chen. 2017. Soft Updates Made Simple and Fast on Non-volatile Memory. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 719–731. https://www.usenix.org/conference/atc17/technical-sessions/presentation/dong

[20] Mingkai Dong, Qianqian Yu, Xiaozhou Zhou, Yang Hong, Haibo Chen, and Binyu Zang. 2016. Rethinking Benchmarking for NVM-based File Systems. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '16)*. ACM, New York, NY, USA, Article 20, 7 pages. https://doi.org/10.1145/2967360.2967379

[21] Subramanya R. Dulloor, Sanjay Kumar, Anil Keshavamurthy, Philip Lantz, Dheeraj Reddy, Rajesh Sankaran, and Jeff Jackson. 2014. System Software for Persistent Memory. In *Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14)*. ACM, New York, NY, USA, Article 15, 15 pages. https://doi.org/10.1145/2592798.2592814

[22] Facebook. 2017. RocksDB. http://rocksdb.org.

[23] R. Fackenthal, M. Kitagawa, W. Otsuka, K. Prall, D. Mills, K. Tsutsui, J. Javanifard, K. Tedrow, T. Tsushima, Y. Shibahara, and G. Hush. 2014. A 16Gb ReRAM with 200MB/s write and 1GB/s read in 27nm technology. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*. 338–339. https://doi.org/10.1109/ISSCC.2014.6757460

[24] FAL Labs. 2010. Kyoto Cabinet: a straightforward implementation of DBM. http://fallabs.com/kyotocabinet/.

[25] Robin Harris. 2016. Windows leaps into the NVM revolution. http://www.zdnet.com/article/windows-leaps-into-the-nvm-revolution/.

[26] Hewlett Packard Enterprise. 2018. HPE Scalable Persistent Memory. https://www.hpe.com/us/en/servers/persistent-memory.html.

[27] Intel. 2015. NVDIMM Namespace Specification. http://pmem.io/documents/NVDIMM_Namespace_Spec.pdf.

[28] Intel. 2017. Intel Architecture Instruction Set Extensions Programming Reference. https://software.intel.com/sites/default/files/managed/0d/53/319433-022.pdf.

[29] Sooman Jeong, Kisung Lee, Jungwoo Hwang, Seongjin Lee, and Youjip Won. 2013. AndroStep: Android Storage Performance Analysis Tool. In *Software Engineering (Workshops)*, Vol. 13. 327–340.

[30] Sooman Jeong, Kisung Lee, Seongjin Lee, Seoungbum Son, and Youjip Won. 2013. I/O Stack Optimization for Smartphones. In *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*. USENIX, San Jose, CA, 309–320. https://www.usenix.org/conference/atc13/technical-sessions/presentation/jeong

[31] Junbin Kang, Benlong Zhang, Tianyu Wo, Weiren Yu, Lian Du, Shuai Ma, and Jinpeng Huai. 2015. SpanFS: A Scalable File System on Fast Storage Devices. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. USENIX Association, Santa Clara, CA, 249–261. https://www.usenix.org/conference/atc15/technical-session/presentation/kang

[32] Sudarsun Kannan, Nitish Bhat, Ada Gavrilovska, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. 2018. Redesigning LSMs for Nonvolatile Memory with NoveLSM. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 993–1005. https://www.usenix.org/conference/atc18/presentation/kannan

[33] Sanidhya Kashyap, Changwoo Min, and Taesoo Kim. 2017. Scalable NUMA-aware Blocking Synchronization Primitives. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 603–615. https://www.usenix.org/conference/atc17/technical-sessions/presentation/kashyap

[34] Takayuki Kawahara. 2011. Scalable Spin-Transfer Torque RAM Technology for Normally-Off Computing. *Design & Test of Computers, IEEE* 28, 1 (Jan 2011), 52–63. https://doi.org/10.1109/MDT.2010.97

[35] Wook-Hee Kim, Jinwoong Kim, Woongki Baek, Beomseok Nam, and Youjip Won. 2016. NVWAL: Exploiting NVRAM in Write-Ahead Logging. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*. ACM, New York, NY, USA, 385–398. https://doi.org/10.1145/2872362.2872392

[36] Youngjin Kwon. 2018. Personal communication.

[37] Youngjin Kwon, Henrique Fingler, Tyler Hunt, Simon Peter, Emmett Witchel, and Thomas Anderson. 2017. Strata: A Cross Media File System. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, New York, NY, USA, 460–477. https://doi.org/10.1145/3132747.3132770

[38] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *ISCA '09: Proceedings of the 36th Annual International Symposium on Computer Architecture*. ACM, New York, NY, USA, 2–13. https://doi.org/10.1145/1555754.1555758

[39] Wongun Lee, Keonwoo Lee, Hankeun Son, Wook-Hee Kim, Beomseok Nam, and Youjip Won. 2015. WALDIO: Eliminating the Filesystem Journaling in Resolving the Journaling of Journal Anomaly. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. USENIX Association, Santa Clara, CA, 235–247. https://www.usenix.org/conference/atc15/technical-session/presentation/lee_wongun

[40] Ran Liu, Heng Zhang, and Haibo Chen. 2014. Scalable Read-mostly Synchronization Using Passive Reader-Writer Locks. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. USENIX Association, Philadelphia, PA, 219–230. https://www.usenix.org/conference/atc14/technical-sessions/presentation/liu

[41] Virendra J. Marathe, Margo Seltzer, Steve Byan, and Tim Harris. 2017. Persistent Memcached: Bringing Legacy Code to Byte-Addressable Persistent Memory. In *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*. USENIX Association, Santa Clara, CA. https://www.usenix.org/conference/hotstorage17/program/presentation/marathe

[42] Paul E McKenney, Jonathan Appavoo, Andi Kleen, Orran Krieger, Rusty Russell, Dipankar Sarma, and Maneesh Soni. 2001. Read-copy update. In *AUUG Conference Proceedings*. AUUG, Inc., 175.

[43] Amirsaman Memaripour, Anirudh Badam, Amar Phanishayee, Yanqi Zhou, Ramnatthan Alagappan, Karin Strauss, and Steven Swanson. 2017. Atomic In-place Updates for Non-volatile Main Memories

with Kamino-Tx. In *Proceedings of the Twelfth European Conference on Computer Systems (EuroSys '17)*. ACM, New York, NY, USA, 499–512. https://doi.org/10.1145/3064176.3064215

[44] Micron. 2017. 3D XPoint Technology. http://www.micron.com/products/advanced-solutions/3d-xpoint-technology.

[45] Micron. 2017. Hybrid Memory: Bridging the Gap Between DRAM Speed and NAND Nonvolatility. http://www.micron.com/products/dram-modules/nvdimm.

[46] Changwoo Min, Sanidhya Kashyap, Steffen Maass, and Taesoo Kim. 2016. Understanding Manycore Scalability of File Systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, Denver, CO, 71–85. https://www.usenix.org/conference/atc16/technical-sessions/presentation/min

[47] MongoDB, Inc. 2017. MongoDB. https://www.mongodb.com.

[48] H. Noguchi, K. Ikegami, K. Kushida, K. Abe, S. Itai, S. Takaya, N. Shimomura, J. Ito, A. Kawasumi, H. Hara, and S. Fujita. 2015. A 3.3ns-access-time 71.2uW/MHz 1Mb embedded STT-MRAM using physically eliminated read-disturb scheme and normally-off memory architecture. In *Solid-State Circuits Conference (ISSCC), 2015 IEEE International*. 1–3. https://doi.org/10.1109/ISSCC.2015.7062963

[49] Gihwan Oh, Sangchul Kim, Sang-Won Lee, and Bongki Moon. 2015. SQLite Optimization with Phase Change Memory for Mobile Applications. *Proc. VLDB Endow.* 8, 12 (Aug. 2015), 1454–1465. https://doi.org/10.14778/2824032.2824044

[50] Daejun Park and Dongkun Shin. 2017. iJournaling: Fine-Grained Journaling for Improving the Latency of Fsync System Call. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 787–798. https://www.usenix.org/conference/atc17/technical-sessions/presentation/park

[51] Thanumalayan Sankaranarayana Pillai, Vijay Chidambaram, Ramnatthan Alagappan, Samer Al-Kiswany, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2014. All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 433–448. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/pillai

[52] pmem.io. 2017. Persistent Memory Development Kit. http://pmem.io/pmdk.

[53] S. Raoux, G.W. Burr, M.J. Breitwisch, C.T. Rettner, Y.C. Chen, R.M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H. L Lung, and C.H. Lam. 2008. Phase-change Random Access Memory: A Scalable Technology. *IBM Journal of Research and Development* 52, 4.5 (July 2008), 465–479. https://doi.org/10.1147/rd.524.0465

[54] redislabs. 2017. Redis. https://redis.io.

[55] Jihye Seo, Wook-Hee Kim, Woongki Baek, Beomseok Nam, and Sam H. Noh. 2017. Failure-Atomic Slotted Paging for Persistent Memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '17)*. ACM, New York, NY, USA, 91–104. https://doi.org/10.1145/3037697.3037737

[56] Yongseok Son, Sunggon Kim, Heon Y. Yeom, and Hyuck Han. 2018. High-Performance Transaction Processing in Journaling File Systems. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*. USENIX Association, Oakland, CA, 227–240. https://www.usenix.org/conference/fast18/presentation/son

[57] SQLite. 2017. SQLite. https://www.sqlite.org.

[58] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. 2008. The Missing Memristor Found. *Nature* 453, 7191 (2008), 80–83.

[59] Symas. 2017. Lightning Memory-Mapped Database (LMDB). https://symas.com/lmdb/.

[60] Vasily Tarasov, Erez Zadok, and Spencer Shepler. 2016. Filebench: A Flexible Framework for File System Benchmarking. *USENIX; login* 41 (2016).

[61] Chia-Che Tsai, Yang Zhan, Jayashree Reddy, Yizheng Jiao, Tao Zhang, and Donald E. Porter. 2015. How to Get More Value from Your File System Directory Cache. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '15)*. ACM, New York, NY, USA, 441–456. https://doi.org/10.1145/2815400.2815405

[62] Shivaram Venkataraman, Niraj Tolia, Parthasarathy Ranganathan, and Roy Campbell. 2011. Consistent and Durable Data Structures for Non-volatile Byte-addressable Memory. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST '11)*. USENIX Association, San Jose, CA, USA, 5–5.

[63] Haris Volos, Sanketh Nalli, Sankarlingam Panneerselvam, Venkatanathan Varadarajan, Prashant Saxena, and Michael M. Swift. 2014. Aerie: Flexible File-system Interfaces to Storage-class Memory. In *Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14)*. ACM, New York, NY, USA, Article 14, 14 pages. https://doi.org/10.1145/2592798.2592810

[64] Haris Volos, Andres Jaan Tack, and Michael M. Swift. 2011. Mnemosyne: Lightweight Persistent Memory. In *ASPLOS '11: Proceeding of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, New York, NY, USA.

[65] Dejan Vučinić, Qingbo Wang, Cyril Guyot, Robert Mateescu, Filip Blagojević, Luiz Franca-Neto, Damien Le Moal, Trevor Bunker, Jian Xu, Steven Swanson, and Zvonimir Bandić. 2014. DC Express: Shortest Latency Protocol for Reading Phase Change Memory over PCI Express. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST '14)*. USENIX, Santa Clara, CA, 309–315. https://www.usenix.org/conference/fast14/technical-sessions/presentation/vucinic

[66] Matthew Wilcox. 2014. Add Support for NV-DIMMs to Ext4. https://lwn.net/Articles/613384/.

[67] Jian Xu and Steven Swanson. 2016. NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*. USENIX Association, Santa Clara, CA, 323–338. https://www.usenix.org/conference/fast16/technical-sessions/presentation/xu

[68] Jian Xu, Lu Zhang, Amirsaman Memaripour, Akshatha Gangadharaiah, Amit Borase, Tamires Brito Da Silva, Steven Swanson, and Andy Rudoff. 2017. NOVA-Fortis: A Fault-Tolerant Non-Volatile Main Memory File System. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, New York, NY, USA, 478–496. https://doi.org/10.1145/3132747.3132761

[69] Jisoo Yang, Dave B. Minturn, and Frank Hady. 2012. When Poll Is Better than Interrupt. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST '12)*. USENIX, Berkeley, CA, USA, 3–3. http://dl.acm.org/citation.cfm?id=2208461.2208464

[70] Jun Yang, Qingsong Wei, Cheng Chen, Chundong Wang, Khai Leong Yong, and Bingsheng He. 2015. NV-Tree: Reducing Consistency Cost for NVM-based Single Level Systems. In *13th USENIX Conference on File and Storage Technologies (FAST '15)*. USENIX Association, Santa Clara, CA, 167–181. https://www.usenix.org/conference/fast15/technical-sessions/presentation/yang

[71] Jishen Zhao, Sheng Li, Doe Hyun Yoon, Yuan Xie, and Norman P. Jouppi. 2013. Kiln: Closing the Performance Gap Between Systems With and Without Persistence Support. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*. ACM, New York, NY, USA, 421–432. https://doi.org/10.1145/2540708.2540744

[72] Ross Zwisler. 2014. Add Support for New Persistent Memory Instructions. https://lwn.net/Articles/619851/.