Hardening the NOVA File System

UCSD-CSE Techreport CS2017-1018

Jian Xu, Lu Zhang, Amirsaman Memaripour, Akshatha Gangadharaiah, Amit Borase, Tamires Brito Da Silva, Andy Rudoff*, Steven Swanson

University of California, San Diego, Intel*

{jix024,luzh,amemarip,agangadh,aborase,tbritodasilva}@eng.ucsd.edu, andy.rudoff@intel.com, swanson@eng.ucsd.edu

Emerging fast, persistent memories will enable systems that combine conventional DRAM with large amounts of non-volatile main memory (NVMM) and provide huge increases in storage performance. Fully realizing this potential requires fundamental changes in how system software manages, protects, and provides access to data that resides in NVMM. We address these needs by describing a NVMM-optimized file system called NOVA that is both fast and resilient in the face of corruption due to media errors and software bugs. We identify and propose solutions for the unique challenges in hardening an NVMM file system, adapt state-of-the-art reliability techniques to an NVMM file system, and quantify the performance and storage overheads of these techniques. We find that NOVA's reliability features increase file system size system size by 14.9% and reduce application-level performance by between 2% and 38%.

If you cite this report in your research please use the this bibtex entry:

```
@TECHREPORT{HARDENING-NOVA,
  AUTHOR        = {Jian Xu and Lu Zhang and Amirsaman Memaripour and Akshatha
                   Gangadharaiah and Amit Borase and Tamires Brito Da Silva and
                   Andy Rudoff and Steven Swanson},
  TITLE         = {Hardening the NOVA File System},
  NUMBER        = {CS2017-1018},
  INSTITUTION   = {Department of Computer Science \& Engineering,
                   University of California, San Diego},
  NOTE          = {http://nvsl.ucsd.edu},
  MONTH         = {May},
  YEAR          = {2017},
  AUTHOR1_URL   = {http://nvsl.ucsd.edu},
  AUTHOR1_EMAIL = {swanson@cs.ucsd.edu},
  URL           = {http://nvsl.ucsd.edu},
}
```

Abstract

Emerging fast, persistent memories will enable systems that combine conventional DRAM with large amounts of non-volatile main memory (NVMM) and provide huge increases in storage performance. Fully realizing this potential requires fundamental changes in how system software manages, protects, and provides access to data that resides in NVMM. We address these needs by describing a NVMM-optimized file system called NOVA that is both fast and resilient in the face of corruption due to media errors and software bugs. We identify and propose solutions for the unique challenges in hardening an NVMM file system, adapt state-of-the-art reliability techniques to an NVMM file system, and quantify the performance and storage overheads of these techniques. We find that NOVA's reliability features increase file system size system size by 14.9% and reduce application-level performance by between 2% and 38%.

1. Introduction

Fast, persistent memory technologies (e.g., battery-backed NVDIMMs [52] or Intel and Micron's 3D XPoint [3]) will enable computer systems with expansive memory systems that combine volatile and non-volatile memories. These hybrid memory systems offer the promise of dramatic increases in storage performance.

Integrating NVMMs into computer systems presents a host of interesting challenges. The most pressing of these focus on how we should redesign existing software components (e.g., file systems) to accommodate and exploit the different performance characteristics, interfaces, and semantics that NVMMs provide.

Several groups have proposed new file systems [19, 23, 92, 97] designed specifically for NVMMs and several Windows and Linux file systems now include at least rudimentary support for them [16, 32, 95]. These file systems provide significant performance gains for data access and support "direct access" (or DAX-style) mmap() that allows applications to access a file's contents directly using load and store instructions, a likely "killer app" for NVMMs.

Despite these NVMM-centric performance improvements, none of these file systems provide the data protection features necessary to detect and correct media errors, protect against data corruption due to misbehaving code, or perform consistent backups of the NVMM's contents. File system stacks in wide use (e.g., ext4 running atop LVM, Btrfs, and ZFS) provide some or all of these capabilities for block-based storage. If users are to trust NVMM file systems with critical data, they will need these features as well.

From a reliability perspective, there are four key differences between conventional block-based file systems and NVMM file systems.

First, the memory controller reports persistent memory media errors as high-priority exceptions rather than error codes from a block driver. Further, the granularity of errors is smaller – a cache line rather than an entire block.

Second, persistent memory file systems must support DAX-style memory mapping that maps persistent memory pages directly into the application's address space. DAX is the fastest way to access persistent memory since it eliminates all operating and file system code from the access path. However, it means a file's contents can change without the file system's knowledge, something that is not possible in a block-based file system.

Third, the entire file system resides in the kernel's address space, vastly increasing vulnerability to "scribbles" – errant stores from misbehaving kernel code.

Fourth, persistent memories are vastly faster than block-based storage devices. This means that the trade-offs that block-based file systems make between reliability and performance need a thorough re-evaluation.

We explore the impact of these differences on reliability mechanisms by building *NOVA*, a hardened NVMM file system. NOVA extends the NOVA [97] NVMM file system with a full suite of data integrity protection features. We quantify the performance and storage overhead of these mechanisms and evaluate their effectiveness at preventing corruption of both file system metadata and file data.

In particular, we make the following contributions:

1. We use the error-reporting and management interface that existing NVMM-enabled systems provide to identify file system reliability challenges that are unique to NVMM file systems.
2. We identify a key challenge to taking consistent file system snapshots while using DAX-style `mmap()` and develop an snapshot algorithm that resolves it.
3. We describe a fast replication algorithm called *Tick-Tock* for NVMM data structures that combines atomic update with error detection and recovery.
4. We adapt state-of-the-art techniques for data protection to work in NOVA and to accommodate DAX-style `mmap()`.
5. We quantify NOVA's vulnerability to scribbles and develop techniques to reduce this vulnerability.
6. We quantify the performance and storage overheads of NOVA's data protection mechanisms.
7. We describe additional steps NOVA and other NVMM file systems can take to further improve reliability.

We find that providing strong reliability guarantees in NOVA increases storage requirements by 14.9% and reduces application-level performance by between 2% and 38%.

To describe hardened NOVA, we start by providing a brief primer on NVMM's implications for system designers, existing NVMM file systems, key issues in file system reliability, and the original NOVA [97] filesystem (Section 2). Then, we describe NOVA's snapshot and (meta)data protection mechanisms (Sections 3 and 4). Section 5 evaluates these mechanisms, and Section 6 presents our conclusions.

2. Background

NOVA targets memory systems that include emerging non-volatile memory technologies along with DRAM. This section first provides a brief survey of NVMM technologies and the opportunities and challenges they present. Then we describe recent work on NVMM file systems and discuss key issues in file system reliability.

2.1 Non-volatile memory technologies

Modern server platforms have support of NVMM in form of NVDIMMs [37, 52] and the Linux kernel includes low-level drivers for identifying physical address regions that are non-volatile, etc. NVDIMMs are commercially available from several vendors in form of DRAM DIMMs that can store their contents to an on-board flash-memory chip in case of power failure with the help of super-capacitors.

NVDIMMs that dispense with flash and battery backup are expected to appear in systems soon. Phase change memory (PCM) [12, 17, 46, 67] and resistive RAM (ReRAM) [26, 84], and 3D XPoint memory technology [3] are denser than DRAM, and may enable very large, non-volatile main memories. Their latencies are longer than DRAM, however, making it unlikely that they will fully replace DRAM as main memory. Other technologies, such as spin-torque transfer RAM (STT-RAM) [40, 58] are faster, but less dense and may find other roles in future systems (e.g., as non-volatile caches [104]). These technologies are all under active development and what we understand about their reliability and performance are evolving rapidly [87].

Commercial availability of these technologies appears to be close at hand. The 3D XPoint memory technology recently announced by Intel and Micron is rumored to offer performance up to 1,000 times faster than NAND flash [3], has already appeared in SSDs [35], and is expected to appear on the processor memory bus shortly [2]. In addition all major memory manufacturers have candidate technologies that could compete with 3D XPoint. Consequently, we expect hybrid volatile/non-volatile memory hierarchies to become common in large systems.

Allowing programmers to build useful data structures with NVMMs requires CPUs to make new guarantees about when stores become persistent that programmers can use to guarantee consistency after a system crash [5, 9]. Without these guarantees it is impossible to build data structures in NVMM that are reliable in the face of power failures [18, 48, 55, 60, 90, 93, 101].

NVMM-aware systems provide some form of "persist barrier" that allows programmers to ensure that earlier stores become persistent before later stores. Researchers have proposed several different kinds of persist barriers [19, 43, 63]. Under x86 a persist barrier comprises a `clflush` or `clwb` [36] instructions to force cache lines into the system's "persistence domain" and a conventional memory fence to enforce ordering. Once a store reaches the persistence domain, the system guarantees it will reach NVMM, even in the case of crash. NOVA and other NVMM file systems assume these or similar instructions are available.

2.2 NVMM File Systems

Several groups have designed NVMM file systems [19, 23, 92, 95, 96, 97] that address the unique challenges that NVMMs' performance and byte-addressible interface present. One of these, NOVA [97], is the basis for NOVA, and we describe it in more detail in Section 2.4.

NVMMs' low latencies make software efficiency much more important than in block-based storage devices [10, 13, 78, 94, 98, 103].

NVMM-aware CPUs provide a load/store interface with atomic 8-byte [1] operations rather than a block-based interface with block- or sector-based atomicity. NVMM file systems can use these atomic updates to implement features such as complex atomic data and metadata updates, but doing so requires different data structures and algorithms than block-based file systems have employed.

Since NVMMs reside on the processor's memory bus, applications should be able to access them directly via loads and stores. NVMM file systems [23, 96, 97] bypass the DRAM page cache and access NVMM directly using a technique called *Direct Access (DAX)*, avoiding extra copies between NVMM and DRAM in the storage stack. DAX has been deployed in Windows [32] and Linux file systems [16, 95].

Most NVMM file systems also provide a DAX version of `mmap()` that allows applications to perform loads and stores directly to NVMM.

2.3 File System Consistency and Reliability

Apart from their core function of storing and retrieving data, file systems also provide facilities to protect the data they

[1] NVMM-aware Intel CPUs provide 8-byte atomic operations. Other architectures may provide different atomicity semantics.

hold from corruption due to system failures, media errors, and software bugs (both in the file system and elsewhere).

File systems have devised a variety of different techniques to guarantee system-wide consistency of file system data structures, including journaling [23, 89], Copy-on-Write [11, 19, 69] and log-structuring [61, 70, 71]. Transactional file systems [15, 53, 59, 62, 65, 81, 91] provide stronger consistency guarantee that application data updates are atomic.

The most reliable file systems provide two key features that significantly improved their reliability: The ability to take snapshots of the file system (to facilitate backups) and set of mechanisms to detect and recover from data corruption due to media errors and other causes.

Existing NVMM file systems provide neither of these features, limiting their usefulness in mission-critical applications. Below, we discuss the importance of each feature and existing approaches.

2.3.1 Snapshots

Snapshots provide a consistent image of the file system at a moment in time. Their most important application is facilitating consistent backups without unmounting the file system, affording protection against catastrophic system failures and the accidental deletion or modification of files.

Many modern file systems have built-in support for snapshots [11, 42, 44, 69]. In other systems the underlying storage stack (e.g., LVM in Linux) can take snapshots of the underlying block device. Other work [22, 64, 73, 80, 85] tries to improve the efficiency and reduce the space consumption of snapshots.

Neither existing DAX-enabled NVMM file systems nor current low-level NVMM drivers support snapshots, making consistent online backups impossible. Several file systems for NVMM do provide snapshots [47, 90, 105], but none of them support DAX-style `mmap()`. Ext4-DAX [95] and xfs-DAX [38] do not support snapshots.

2.3.2 Data Corruption

File systems are subject to a wide array of data corruption mechanisms. These include both media errors that cause storage media to return incorrect values and software errors that store incorrect data to the storage media. Data corruption and software errors in the storage stack have been thoroughly studied for hard disks [6, 7, 74, 75], SSDs [50, 57, 76] and DRAM-based memories [77, 82, 83]. The results of DRAM-based studies apply to DRAM-based NVDIMMs, but there have been no (publicly-available) studies of error behaviors in emerging NVMM technologies.

Storage hardware, including NVMMs, uses error-correcting codes (ECC) to provide some protection against media errors. Errors that ECC detects but cannot correct result in an error. For block-based storage, this error appears as read or write failure from the storage driver. Intel NVMM-based system report these media errors via a high-priority exception (see Section 4.1).

Software errors can also cause data corruption. If the file system is buggy, it may write data in the wrong place or fail to write at all. Other code in the kernel can corrupt file system data by "scribbling" [45] on file system data structures or data buffers.

Scribbles are an especially critical problem for NVMM file systems, since the NVMM is mapped into the kernel's address space. As a result, all of file system's data and metadata are always vulnerable to scribbles.

We discuss other prior work on file system reliability as it relates to NOVA in Section 4.7.

2.4 The NOVA File System

NOVA's initial design focused on two goals: Fully exposing the performance that NVMMs offer and providing very strong consistency guarantees – all operations in NOVA are atomic. Below, we describe the features of NOVA that are most relevant to our description of NOVA.

NOVA divides NVMM into five regions. NOVA's 512 B superblock contains global file system information and the recovery inode. The recovery inode represents a special file that stores recovery information (e.g., the list of unallocated NVMM pages). NOVA divides its inode tables into per-CPU stripes. It also provides per-CPU journals for complex file operation (see below). The rest of the available NVMM stores logs and file data.

NOVA is log-structured and stores a separate log for each inode to maximize concurrency and provide atomicity for operations that affect a single file. The logs only store metadata and comprise a linked list of 4 KB pages. Log entries are small – between 32 and 64 bytes. Logs are generally non-contiguous, and log pages may reside anywhere in NVMM.

NOVA keeps read-only copies of most file metadata in DRAM during normal operations, eliminating the need to access metadata in NVMM during reads.

NOVA uses copy-on-write to provide atomic updates for file data and appends metadata about the write to the log. For operations that affect multiple inodes NOVA uses lightweight, fixed-length journals – one per core.

NOVA divides the allocatable NVMM into multiple regions, one region per CPU core. A per-core allocator manages each of the regions, minimizing contention during memory allocation.

After a system crash, NOVA must scan all the logs to rebuild the memory allocator state. Since, there are many logs, NOVA aggressively parallelizes the scan. Recovering a 50 GB NOVA file system takes just over 1/10th of second [97].

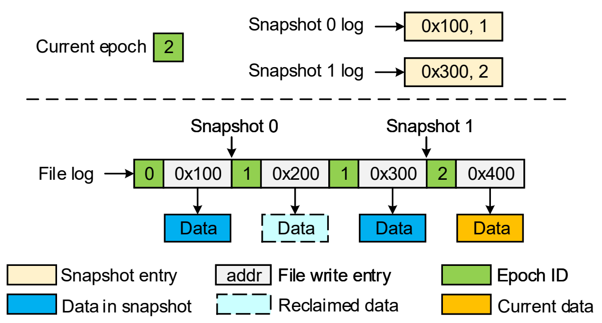

Figure 1: Taking snapshot in NOVA NOVA takes a snapshot by incrementing the epoch ID to start a new epoch. When a data block or log entry becomes dead in the current epoch, NOVA records its death in the snapshot log for most recent snapshot it survives in.

3. Snapshots

NOVA's snapshot support lets system administrators take consistent snapshots of the file systems while applications are running. The system can mount a snapshot as a read-only file system or roll the file system back to a previous snapshot. NOVA supports an unlimited number of snapshots, and snapshots can be deleted in any order. NOVA is the first NVMM file system that supports taking consistent snapshots when applications modify file data via DAX mmap().

Below, we described the three central challenges that NOVA's snapshot mechanisms addresses: Taking snapshots, managing storage for snapshots, and taking usable snapshots of DAX `mmap()`'d files.

3.1 Taking Snapshots

NOVA implements snapshots by maintaining a current *epoch number* for the whole file system, and storing the epoch number in each new log entry. To create a snapshot, NOVA increments the file system's epoch number and records the old epoch number in a list of snapshots.

Figure 1 shows an example in which NOVA creates two snapshots and the current epoch number is 2. Snapshot creation does not block file system write operations as it does in some other file systems [33, 85, 105].

3.2 Snapshot Storage Management

Supporting snapshots requires NOVA to preserve file contents from previous snapshots while also being able to recover the space a snapshot occupied after its deletion.

Preserving file contents requires a small change to how NOVA implements write operations. To perform a write, NOVA appends a write log entry to the file's log. The log entry includes pointers to newly-allocated and populated NVMM pages that hold the written data. If the write overwrites existing data, NOVA locates the previous write log entry for that portion of the file, and performs an *epoch check* that compares the old log entry's epoch ID to the file system's current epoch ID. If the comparison matches, the old write log and the file data blocks it points to no longer belong to any snapshot, and NOVA reclaims the data blocks. In Figure 1, an application overwrites the same block of the file for four times. The epoch check for the third write reclaims the second write's data block, since they belong to the same epoch.

If the epoch IDs do not match, then the data in the old log entry belongs to an earlier snapshot and NOVA leaves the old entry and data alone.

Determining when to reclaim data belonging to deleted snapshots requires additional bookkeeping. For each snapshot, NOVA maintains a *snapshot log* that records the inodes and blocks that belong to that snapshot, but are not part of the current file system image.

NOVA populates the snapshot log during the epoch check: If the epoch IDs for the new and old log entries do not match, it appends a tuple to the snapshot log that the old log entry belongs to. The tuple contains the old log entry, and the current epoch ID, called the *delete epoch ID*. In Figure 1, the log entry for snapshot 0 shows that page 0x100 is part of the snapshot 0 and that a write in epoch 1 overwrote it.

To delete a snapshot, NOVA removes the snapshot from the list of live snapshots and appends its log to the following snapshot's log. Then, a background thread traverses the combined log and reclaims dead inode/data based on the delete epoch ID: If the delete epoch ID for an entry in the log is less than or equal to the snapshot's epoch ID, it means the log entry and/or the associated data blocks are now dead.

NOVA keeps the list of live snapshots in NVMM, but it keeps the contents of the snapshot logs in DRAM. On a clean unmount, it writes the snapshot logs to NVMM in the recovery inode. This is safe since NOVA can reconstruct the snapshot logs while it scans the inode logs during recovery after an unclean unmount.

3.3 Snapshots for DAX-mmap'd Files

Taking consistent snapshots while applications are modifying files using DAX-style mmap requires NOVA to reckon with the order in which stores to NVMM become persistent (i.e., reach physical NVMM so they will survive a system failure). These applications rely on the processor's "memory persistence model" [63] to make guarantees about when and in what order stores become persistent. These guarantees allow the application to restore their data to a consistent state during recovery from a system failure.

From the application's perspective, reading a snapshot is equivalent to recovering from a system failure. In both cases, the contents of the memory-mapped file reflect its state at a moment when application operations might be

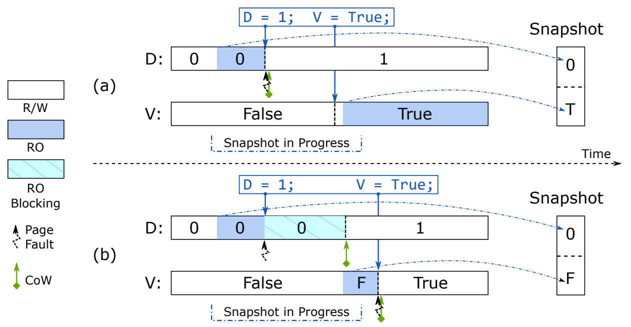

Figure 2: Snapshots of `mmap()`'d Data NOVA marks `mmap()`'d pages as read-only to capture their contents in the snapshot. Naively marking pages one at a time, can result in an inconsistent snapshot (a). Instead, NOVA marks all the pages read-only before satisfying any subsequent page faults, guaranteeing consistency (b).

in-flight and when the application had no chance to shut down cleanly.

A naive approach to checkpointing `mmap()`'d files in NOVA would simply mark each of the read/write mapped pages as read-only and then do copy-on-write when a store occurs to preserve the old pages as part of the snapshot.

However, this approach can leave the snapshot in an inconsistent state: Setting the page to read-only captures its contents for the snapshot, and the kernel requires NOVA to set the pages as read-only one at a time. So, if the order in which NOVA marks pages as read-only is incompatible with ordering that the application requires, the snapshot will contain an inconsistent version of the file.

Figure 2 (a) shows an example: Consider a data value D, and a flag V that is `true` when D contains valid data. To enforce this invariant, a thread could issue a persist barrier between the stores to D and V.

In the figure, D and V reside in different pages, and NOVA marks D's page as read-only *before* the program updates D and marks V's page as read-only *after* setting V. As a result, the snapshot has V equal to `true`, but D with its old, incorrect value.

To resolve this problem, when NOVA starts marking pages as read-only, it blocks page faults to the read-only `mmap()`'d pages until it has marked all the pages read-only and finished taking the snapshot. Figure 2 (b) shows how this solves the problem: D's page fault will not complete (and the store to V will not occur) until all the pages (including V's) have been marked read-only and the snapshot is complete.

This solution is also correct for multi-threaded programs. If thread 1 updates D and, later, thread 2 should update V, the threads must use some synchronization primitive to enforce that ordering. For instance, thread 1 may release a lock after storing to D. If D's page is marked read-only before D changes, the unlock will not occur until all the page have been marked read-only, ensuring that the store to V will not appear in the snapshot.

This approach guarantees that, for each thread, the snapshot will reflect a prefix of the program order-based sequence of NVMM store operations the thread has performed. This model is equivalent to strict persistence [63], the most restrictive model for how NVMM memory operations should behave (i.e., in what order updates can become persistent) in a multi-processor system. CPUs may implement more aggressive, relaxed models for memory persistence, but strict persistence is strictly more conservative than all proposed models, so NOVA's approach is correct under those models as well [2].

3.4 Design Decisions and Evaluation

There are several ways we could have addressed the problems of creating snapshots, managing the data they contain, and correctly capturing `mmap()`'d data. Other file systems have leveraged reference counts [68, 69], tracked the relationships between blocks [22], relied on an SSD's flash translation layer [85], or maintains a bitmap for all blocks per snapshot [42].

None of these approaches is a good fit for NOVA: It does not rely on reference counts or bitmap for space management, and adding them just to support snapshots would penalize all operations that modify data or metedata. There is also no FTL to rely on.

Our approach stores snapshot logs in DRAM. The maximum total size of the snapshot logs is proportional to the size of the file system rather than the number of checkpoints, since each data block and log entry would be referred to by at most one snapshot log.

In practice, the size of the logs is manageable. We ran the fileserver workloads described in Section 5 while taking snapshots every 10 seconds. The size of the logs ranged from 0.02% to 0.026% of the space in use in the file system.

Taking snapshots has different effects on the performance of applications that use file-based access and those that use DAX-style `mmap()`. Figure 1 measures the impact on both groups. On the right, it shows results for the WHISPER [56] suite of eight applications optimized for DAX-`mmap()`. On the left, are results for four Filebench [88] workloads.

For all the applications, the figure compares performance without snapshots to performance while taking a snapshot every 10 seconds. For the WHISPER applications, taking periodic snapshots only reduces performance by 6.2% on average. For filebench workloads, the impact is negligible – 0.6% on average.

[2] This is directly analogous to sequential memory consistency being a valid implementation of any more relaxed memory consistency model. Indeed, strict persistence is a natural extension of sequential consistency to include a notion of persistence.

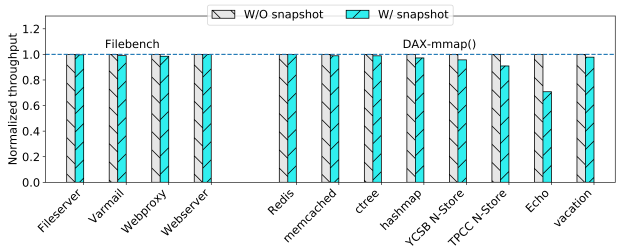

Figure 3: Snapshots' Performance Impact For file-based application ("Filebench"), taking a snapshot every 10s reduces performance by just 0.6% on average. Frequent snapshots are more costly for the DAX-mmap()-optimized applications ("DAX-mmap()"): Performance drops by 6.2% on average.

4. Handling Data Corruption in NOVA

Like all storage media, NVMM is subject to data and metadata corruption from media failures and software bugs. To prevent, detect, and recover from data corruption, NOVA relies on the capabilities of the system hardware and operating system as well as its own error detection and recovery mechanisms.

This section describes the interfaces that NOVA expects from the memory system hardware and the OS and how it leverages them to detect and recover from corruption. We also discuss a technique that prevents data corruption in many cases, and NOVA's ability to trade reliability for performance. Finally, we discuss NOVA's protection mechanisms in the context of recent work on file system reliability.

4.1 Media Errors

NOVA detects NVMM media errors with the same mechanisms that processors provide to detect DRAM errors. The details of these mechanisms determine how NOVA and other NVMM file systems can protect themselves from media errors.

This section describes the interface that recent Linux kernels (e.g., Linux 4.10) and Intel processors provide via the PMEM low-level NVDIMM driver. Porting NOVA to other architectures or operating systems may require NOVA to adopt a different approach to error detection.

NOVA assumes the memory system provides ECC for NVMM that is similar to (or perhaps stronger than) the single-error-correction/double-error-detection (SEC-DED) scheme that conventional DRAM uses. We assume the controller transparently corrects correctable errors, and silently returns invalid data for undetectable errors.

For detectable but uncorrectable errors, Intel's Machine Check Architecture (MCA) [34] raises a *machine check exception (MCE)* in response to uncorrectable memory errors. After the exception, MCA registers hold information that allows the OS to identify the memory address and instruction responsible for the exception.

The size of the ECC code word depends on the memory used in the system. DRAM-based NVMMs uses a 9-byte code word for every 8-byte data word, so an uncorrectable error destroys at least 64-bits. Other memories may use a larger or smaller code word size. The memory controller and/or processor may track errors at a coarser granularity (e.g., the size of a cache line). We refer to this granularity as the *poison radius* (PR) of an error. We assume that PRs are a power of two in size and aligned to that size. We say that an uncorrectable error in a PR "poisons" the entire PR.

The system can clear a poisoned PR by issuing a "Clear Uncorrectable Error" [1] through the kernel's Advanced Configuration and Power Interface (ACPI) driver. The poisoned status of PR persists across system failures and the PMEM driver collects a list of poisoned PRs at boot.

The default response to an MCE in the kernel is a kernel panic. However, recent Linux kernels include a version of `memcpy()`, called `memcpy_mcsafe()` [41], that returns an error to the caller instead of crashing in response to most MCEs. NOVA always uses this function when reading from NVMM. In rare cases, MCEs are not recoverable because the processor's state is corrupted and a kernel panic is inevitable.

4.2 Tick-Tock Metadata Protection

NOVA use replication and checksums to protect its metadata from media errors and corruption. NOVA protects each data structure by keeping two copies – a *primary* and a *replica* – and adding a CRC32 checksum to each.

To update a metadata structure, NOVA first copies the contents of the data structure into the primary (the *tick*), and issues a persist barrier to ensure that data is written to NVMM. Then it does the same for the replica (*tock*). This scheme ensures that, at any moment, one of the two copies is correctly updated and has a consistent checksum.

To access a metadata structure and check for corruption NOVA copies the primary and replica into DRAM buffers using `memcpy_mcsafe()` to detect media errors. Then it confirms the checksums for each copy. If either of them has suffered a media error or has an inconsistent checksum, NOVA recovers using the other copy. If both copies are consistent but not identical, the system failed between the tick and tock phases, and NOVA copies the primary to the replica.

If both copies suffered a media error or are inconsistent, the metadata is lost, and NOVA returns an error.

Design Decisions and Alternatives Tick-tock metadata protection provides strong protection against both media errors and scribbles and is easy to implement. We considered several other designs and evaluated alternative in the details of our implementation.

Using checksums to detect data corruption and relying on replication for error correction more than double the storage required for metadata storage. We could rely on error correction codes instead to handle detection and correction and avoid full replication. For instance, using Reed Solomon codes to protect a 32-byte log entry would require storing two bytes of ECC information, a 6.2% overhead, but it could only correct a single byte error. Computing Reed Solomon (and other ECC codes) is much slower than CRC. Intel's CRC32 acceleration instructions can compute a 32-byte CRC in 13 ns compared to more than 1000 ns to compute the Reed Solomon code word using the implementation in Linux.

Replication also provides better resilience to scribbles. Though we know of no systematic study of scribble size in real systems, we expect that some of them may arise from misdirected `memcpy()`-like operations. Since NOVA's metadata structures are small, it is likely that such a scribble will obliterate an entire data structure. Full replication can recover from these, but ECC cannot.

We could adopt a log-based mechanism for committing changes, but that would require at least as many stores and persist barriers as tick-tock. It would also eliminate full replication, increasing vulnerability to scribbles.

4.3 Protecting File Data

NOVA adapts RAID-style parity protection and checksums to protect file data and it includes features to maximize protection for files that applications access via DAX-style `mmap()`.

RAID Parity and Checksums NOVA makes each file data page its own stripe, and divides each page into equally sized stripe segments, or strips. The strip size is configurable, but it must be larger than 512 B (the default) and smaller than a page. It also must be a least one PR. NOVA stores an additional parity strip for each page and it maintains two separate copies of CRC32 checksums for each data strip. Figure 4 shows the checksum and parity layouts in the NVMM space.

When NOVA performs a read it checks the target strip's checksum. If the strip is corrupted, NOVA performs RAID-5-style recovery to restore the lost data. If more than one of the strips is corrupted, the page is lost.

Writes are simple since NOVA uses copy-on-write by default: For each file page write, NOVA allocates new pages, populates them with the written data, computes the checksums and parity strip, and finally commits the write with an atomic log appending operation.

Protecting DAX-mmap'd data By design, DAX-style `mmap()` lets application modify file data without involving the file system, so it is impossible for NOVA to keep the checksums and parity for read/write mapped pages up-to-date. Instead, NOVA provides the following guarantee: The checksums and parity for data pages are up-to-date at all times, except when those pages are mapped read/write into an application's address space.

We believe this is the strongest guarantee that NOVA can provide on current hardware, and it raises several challenges. First users that use DAX-`mmap()` take on responsibility for detecting and recovering from both media errors (which appear as `SIGBUS` in user space) and scribbles. This is an interesting problem but beyond the scope of this paper. Second, NOVA must be able to tell when a data page's checksums and parity should match the data it contains and when they may not.

To accomplish this, when a portion a file is `mmap()`'d, NOVA records this fact in the log, signifying that the checksums and parity for the affected pages are no longer valid. NOVA only recomputes the checksums and parity for dirty pages on `msync()` and `munmap()`. On `munmap()`, it adds a log entry that restores protection for these pages when the last mapping for the page is removed. If the system crashes while pages are mapped, the recovery process will identify these pages while scanning the logs, recompute checksums and parity, and add a log entry to mark them as valid.

Design Decisions and Alternatives Using RAID parity and checksums to protect file data is similar to the approach that ZFS [11] and IRON ext3 [66] take, but we store parity for each page rather than one parity page per file. The approach incurs storage overheads for both the parity strip (which grows with strip size) and the checksums (which shrinks with strip size, because the number of strips drops). For 512 B strips the total overhead is about 13.4%. Increasing the NVMM page size reduces the storage overhead, but it would lead to more fragmentation in log pages or require a more complex NVMM page allocator.

We also considered a simpler scheme that would allow users to mark files as RAID-protected or not. This eliminates the need for tracking which parts of which files are protected, but places an administrative burden on the file system's user and raises policy questions without obvious answers. For instance, if a user copies an unprotected file, should the copy also be unprotected? and What should happen if an application `mmap()`'s a protected file? On balance, the benefits of a clearer, simpler rule about when file data is protected outweighs the costs of implementing our more sophisticated scheme.

4.4 Minimizing Vulnerability to Scribbles

Scribbles pose significant risk to NOVA's data and metadata, since a scribble can impact large, continuous regions of memory. We are not aware of any systematic study of the prevalence of these errors, but scribbles, lost, and misdirected writes are well-known culprits for file system corruption [23, 29, 45, 102]. In practice, we expect that smaller scribbles are more likely that large ones, in part

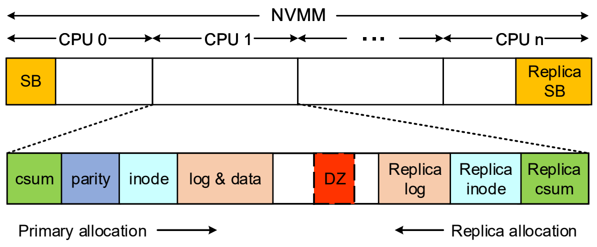

Figure 4: NOVA space layout NOVA's per-core allocators satisfy requests for primary and replica storage from different directions. They also store data pages and their checksum and parity pages separately. The virtual "dead zone" (DZ) for metadata allocation guarantees a gap between primary and replica pages.

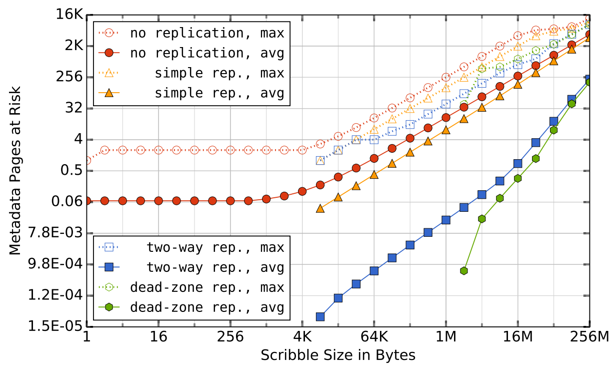

Figure 5: Scribble size and metadata bytes at risk Replicating metadata pages and taking care to allocate the replicas separately improves resilience to scribbles. The most effective technique enforces a 1 MB "dead zone" between replicas and eliminates the threat of a single scribble smaller than 1 MB. The graph omits zero values due to the vertical log scale.

since the bugs that result in larger scribbles would be more severe and more likely to be found and fixed.

To quantify the risk that these errors pose, we define *bytes-at-risk (BAR)* for a scribble as the number of bytes it may render irretrievable.

NOVA packs log entries in to log pages, and it must scan the page to recognizing each entry. Without protection, losing a single byte can corrupt a whole page. For replicated log pages, a scribble that spans both copies of a byte will corrupt the page. To measure the BAR for a scribble of size N we measure the number of pages each possible N-byte scribble would destroy in an aged NOVA file system.

Figure 5 shows the maximum and average metadata BAR for a 64 GB NOVA file system with four different metadata protection schemes: "no replication" does not replicate metadata; "simple replication" allocates the primary and replicas naively and tends to allocate lower addresses before higher address, so the primary and replica are often close; "two-way replication" separates the primary and replica by preferring low addresses for the primary and high addresses for the replica; and "dead-zone replication" extends "two-way" by enforcing a 1 MB "dead zone" between the primary and replica. Figure 4 shows an example of NOVA two-way allocator with dead zone separation. For each pair of mirrored pages, the dead zone forbids the primary and replica from becoming too close, but data pages can reside between them.

We aged the file system by spawning multiple, multi-threaded, filebench workloads. When each workload finishes, we remove about half of its files, and then restart the workload. We continue until the file system is 99% full.

The data show that even for the smallest 1-byte scribble, the unprotected version will lose up to a whole page (4 kB) of metadata and an average of 0.06 pages. With simple replication, scribbles smaller than 4 kB have zero BAR. Under simple replication, an 8 kB scribble can corrupt up to 4 kB, but affects only 0.04 pages on average.

Two-way replication tries to allocate the primary and replica farther apart, and it reduces the average bytes at risk with an 8 kB scribble to 2.9×10^{-5} pages, but the worst case remains the same.

Enforcing the dead zone further improves protection: A 1 MB dead zone can eliminate corruption for scribbles smaller than 1 MB. The dead zone size is configurable, so NOVA can increase the 1 MB threshold for scribble vulnerability if larger scribbles are a concern.

Scribbles also place data pages at risk. Since NOVA stores the stripes of data pages contiguously, scribbles that are larger than the strip size may causes data loss, but smaller scribbles do not. NOVA could tolerate larger scribbles to data pages by interleaving strips from different pages, but this would make disallow DAX-style `mmap()`. Increasing the strip size can also improve scribble tolerance, but at the cost of increased storage overhead for the parity strip.

4.5 Preventing Scribbles

The mechanisms described above let NOVA detect and recover from data corruption. NOVA also includes a mechanism that leverages the memory protection system in Intel processors to prevent most scribbles.

By default, NVMM is mapped read/write into the kernel's address space. This provide easy access for NOVA, but leaves the NVMM exposed to scribbles from other parts of the kernel [14]. We can reduce this vulnerability by mapping NVMM as read-only, and then using the WriteProtect Enable (WP) bit in the per-core CR0 control register to disable write protection on *all* kernel memory when NOVA needs to modify NVMM. WAFL [45] and PMFS [23] both use this mechanism to protect data as well.

The approach also requires disabling local interrupts to prevent a context switch while write protection is disabled.

Clearing and restoring the WP bit and disabling and re-enabling local interrupts takes 400 ns on our machines.

Using the WP bit means that NOVA no longer needs to trust the rest of the kernel, but it also requires that the rest of the kernel trust NOVA to not accidently modify other data in the kernel that is read-only.

We are careful to limit the risk that NOVA poses to other kernel data. We have implemented a version of `memcpy()` that we use for writing to NVMM, called `memcpy_to_pmem()`. It includes the code to clear and restore the WP bit, disable and enable interrupts, and a sanity check to ensure that the target address range is in NVMM.

The WP approach will only protect against scribbles from other kernel code. It cannot prevent NOVA from corrupting its own data by performing "misdirected writes," a common source of data corruption in file systems [8].

4.6 Relaxing Data and Metadata Protection

Many existing file systems can trade off reliability for improved performance (e.g., the data journaling option in Ext4). NOVA can do the same: It provides a *relaxed mode* that relaxes atomicity constraints on file data and metadata.

In relaxed mode, write operations modify existing data directly rather than using copy-on-write, and metadata operations modify the most recent log entry for an inode directly rather than appending a new entry. Relaxed mode guarantees metadata atomicity by journaling the modified pieces of metadata. These changes improve performance and we evaluate their impact in Section 5.

4.7 Related and Future Work

File system reliability has been the focus of a great deal of research driven by the file systems' importance, their complexity, and the complex (and error-prone) behavior of other parts of the storage stack. Below, we describe proposed "best practices" for file system design and how NOVA addresses them. Then, we describe other tools, techniques, and studies that would further improve NOVA's reliability.

4.7.1 Is NOVA Ferrous?

InteRnally cONistent (IRON) file systems [66] provide a set of principles to that lead to improved reliability. We designed NOVA to embody these principles:

Check error codes Uncorrectable ECC errors are the only errors that the NVMM memory system delivers to software (i.e., via `memcpy_mcsafe()`). NOVA uses `memcpy_mcsafe()` for all NVMM access and triggers recovery if it detects an MCE. NOVA also interacts with the PMEM driver that provides low-level management of NVDIMMs. For these calls, we check and respond to error codes appropriately.

Report errors and limit the scope of failures NOVA reports all unrecoverable errors as `-EIO`, and never calls `panic()`.

Use redundancy for integrity checks and distribute redundancy information NOVA's tick-tock replication scheme stores the checksum for each replica with the replica, but it is careful to allocate the primary and replica copies far from one another. Likewise, NOVA stores the parity and checksum information for data pages separately from the pages themselves.

Type-aware fault injection For testing, we built a NOVA-specific error injection tool that can corrupt data and metadata structures in specific, targeted ways, allowing us to test all of NOVA's detection and recovery mechanisms.

4.7.2 Other Reliability Techniques

Several groups have proposed techniques and built tools that help verify file systems. SQCK [30] provides a declarative languages for specifying file system invariants, and FiSC [100] provide a model checking framework for the same. Crash refinement [79] and EXPLODE [99] can automatically check a file system's implementation against its specification or existing consistency checks. EDP [31] would identify any error codes NOVA mishandles. All these techniques would likely find bugs in NOVA. Other techniques, such as Recon's consistency invariants [28] and Membrane's restartability [86] could be applied to NOVA to help it survive bugs that remain.

Other works highlight areas where NOVA's (and most other file systems') reliability could be improved. Corruption of DRAM data structures can result in file system corruption [27, 102] and some file systems (e.g., WAFL [45] and HARDFS [21]) protect DRAM structures with checksums.

4.7.3 Areas for Improvement

There are several additional steps NOVA could take to further improve reliability. All of them are the subject of ongoing development, and we do not expect any of them to have a large impact or performance or storage overheads.

NOVA protects some, but not all, DRAM data structures. Most DRAM structures that NOVA does not protect are short lived (e.g., the DRAM copies we create of metadata structures) or are not written back to NVMM. However, the snapshot logs and allocator state are exceptions and they are vulnerable to corruption. The allocator protects the address and length of each free region with checksums, but it does not protect the pointers that make up the red-black tree that holds them, since we use the kernel's generic red-black tree implementation. The snapshot logs are currently unprotected.

Sector or block failures in disks are not randomly distributed [6], and errors in NVMM are also likely to exhibit complex patterns of locality [51, 82, 83]. For instance, a NVMM chip may suffer from a faulty bank, row, or column, leading to a non-uniform error distribution. Or, an entire NVDIMM may fail.

NOVA's allocator actively separates the primary and replica copies of metadata structures to eliminate *logical* locality, but it does not account for how the memory system maps physical addresses onto the physical memory. A layout-aware allocator could, for instance, ensure that replicas reside in different banks or different NVDIMMs.

NOVA cannot keep running after an unrecoverable MCE (since they cause a `panic()`), but it could recover any corrupted data during recovery. The PMEM driver provides a list of poisoned PRs on boot, and NOVA can use this information to locate and recover corrupted file data during mount. Without this step, NOVA will still detect poisoned metadata, since reading from a poisoned PR results in a normal MCE, NOVA reads all metadata during recovery. Poisoned file data, however, could accumulate over multiple unrecoverable MCEs, increasing the chances of data loss.

Finally, NOVA does not scrub data or metadata. PMEM detects media errors on reboot, but if a NOVA file system ran continuously for a long time, undetected media errors could accumulate. Undetected scribbles to data and metadata can accumulate during normal operation and across reboots.

5. Performance Trade-offs

NOVA's reliability features improve its resilience but also incur overhead in terms of performance and storage space. This section quantifies these overheads and explores the trade-offs they allow.

5.1 Experimental setup

We use the Intel Persistent Memory Emulation Platform (PMEP) [23] to emulate different types of NVMM and study their effects on NVMM file systems. PMEP supports configurable latencies and bandwidth for the emulated NVMM, and emulates `clwb` instruction with microcode. In our tests we configure the PMEP with 32 GB of DRAM and 64 GB of NVMM, and choose two configurations for PMEP's memory emulation system: We use the same read latency and bandwidth as DRAM to emulate fast NVDIMM-N [72], and set read latency to 300 ns and reduce the write bandwidth to 1/8th of DRAM to emulate slower PCM. For both configurations we set `clwb` instruction latency to 40 ns.

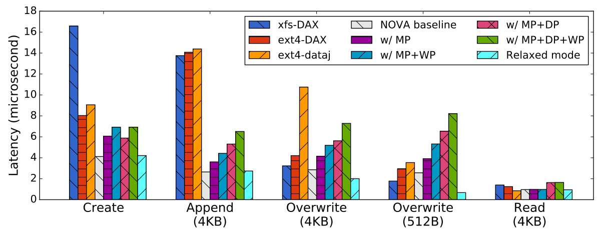

Figure 6: File operation latency In most cases, NOVA's basic file operations are faster other available file systems. The exceptions (e.g., writes and overwrites vs. Ext4-DAX and XFS-DAX) are cases where NOVA provide stronger consistency guarantees and/or data protections. Relaxing these protections ("Relaxed mode") dramatically improves performance.

We compare NOVA against three other file systems. Ext4-DAX and xfs-DAX are the two DAX-enabled file systems currently supported available in Linux (PMFS [23] has been deprecated). Neither of them provides strong consistency guarantees (i.e., they do not guarantee that all operations are atomic), while our baseline filesystem does provide these guarantees. To compare to a file system with stronger guarantees, we also compare to ext4 in data journaling (ext4-dataj) mode running on the NVMM block device that exposes block device interface to an NVMM region. Ext4 and xfs have checksums for metadata, but they do not provide any recovery mechanisms for NVMM media errors or protection against stray writes.

We perform all our experiments on Linux x86-64 kernel 4.10.

5.2 Performance Impacts

To understand the impact of NOVA's reliability mechanisms, we begin by measuring the performance of individual mechanisms and basic file operations. Then measure their impact on application-level performance.

We compare several version of NOVA: We start with "Baseline", which includes snapshot support but no metadata or data protection, and add metadata protection ("MP"), data protection ("DP"), and CR0-based write protection ("WP"). "Relaxed mode" weakens consistency guarantees to improve performance and provides no data protection (Section 4.6).

5.3 Microbenchmarks

We evaluate basic file system operations: `create`, 4 KB `append`, 4 KB `write`, 512 B `write`, and 4 KB `read`. Figure 6 measures the latency for these operations with NVDIMM-N configuration. Data for PCM has similar trends. `Create` is a metadata-only operation and adding metadata protection increases the latency by 47%. `Append` affects metadata and data updates. The baseline append operation is much more efficient than the existing file systems (5.2× to 5.4× times faster). Adding metadata and

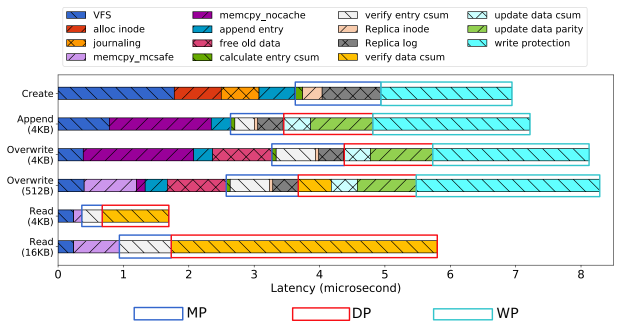

Figure 7: NOVA Latencies for NVDIMM-N For most operations, NOVA's file data protection mechanisms have a larger impact on operation latencies that metadata protection, especially since the cost of verifying checksums scales with read and write size. For reads, in particular, verifying checksums increases latency by 70%.

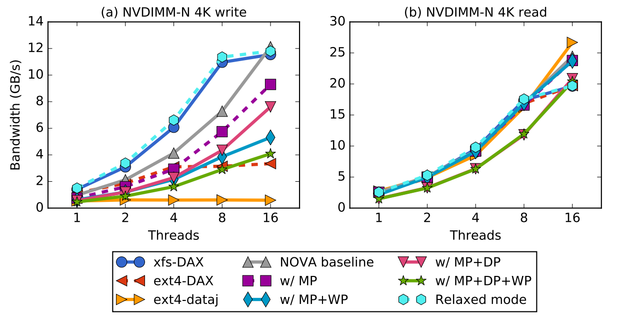

Figure 8: NOVA random read/write bandwidth on NVDIMM-N Read bandwidth is similar across all the file systems we tested, but NOVA's reliability mechanisms reduces throughput by between 14% and 19%. For writes the cost of reliability is higher – up to 66%, and the benefits of relaxing write atomicity ("relaxed mode" and "XFS-DAX") are greater (up to 37%).

data protection increase the latency by 36% and 100%, respectively, and write protection increases the latency by 22% on average. With the maximum level of protection NOVA is 146% slower than the baseline version of NOVA and 2.1× faster than the existing alternatives.

For overwrite, NOVA performs copy-on-write for file data to provide data atomicity guarantees, and the latency is close to that of `append`. For 512 B overwrite, NOVA has higher latency than other DAX file systems since its copy-on-write amplifies the 512 B write to 4 kB. Full protection increases the latency by 2.2×. Relaxed mode is 2.6× faster than xfs-DAX and 3.8× faster than baseline for 512 B overwrite, since it performs in-place updates. For `read` operations, data protection adds 70% overhead because it verifies the data checksum before returning to the user.

Figure 7 breaks down the latency for NOVA with full protection ("w/ MP+DP+WP"). For create, inode allocation and appending to the log consumes 48% of latency, due to inode/log replication and checksum calculation. For 4 kB append and overwrite, data protection has almost the same latency as memory copy (`memcpy_nocache`), and it accounts for 31% of the total latency in 512 B overwrite.

Figure 8 shows FIO [4] measurements for the multi-threaded read/write bandwidth of the file systems. For writes, NOVA's relaxed mode achieves the highest bandwidth. With sixteen threads, Metadata protection reduces NOVA bandwidth by 24%, data protection impacts throughtput by 37%, and full protection on NOVA reduces bandwidth by 66%. For read, all the file systems scale well, while NOVA data protection incurs 14% overhead on 16 threads, due to checksum verification.

5.4 Macrobenchmarks

We use eight application-level workloads to evaluate NOVA: Four Filebench [88] workloads (fileserver, varmail, webproxy and webserver), two key-value stores (RocksDB [25] and MongoDB [54]), the Exim email server [24], and TPC-C running on Shore-MT [39]. Table 1 summarizes the workloads.

Figure 9 measures their performance on our three comparison file systems and several NOVA configurations, normalized to the NOVA baseline throughput on NVDIMM-N. NOVA outperforms DAX file systems by between 3% and 4.4×, and outperforms ext4-dataj by between 20% and 17.5×. NOVA achieves larger improvement on metadata-intensive workloads, such as varmail and Exim, confirming NOVA's efficiency on metadata operations.

Adding metadata protection reduces performance by between 0 and 9% and using the WP bit to reduce scribbles costs an additional 0.1% to 13.4%. Full protection reduces performance by 2% to 38%, with write-intensive workloads seeing the larger drops. The figure also shows that the performance benefits of giving up atomicity in file operations ("Relaxed mode") are modest – no more than 6.4%.

RocksDB sees the biggest performance loss because RocksDB uses LevelDB [20] which has large write amplification [49]. It also issues many non-page-aligned writes that trigger copy-on-writes under NOVA. Relaxed modes avoids the copies, so RocksDB benefits from relaxed mode more than other workloads.

On the PCM configuration, fileserver, webserver and RocksDB show the largest performance drop. This is because fileserver and RocksDB are write-intensive and saturate PCM's write bandwidth, and webserver is read-intensive and PCM's read latency limits file systems' performance.

Compared to other file systems, NOVA is more sensitive to NVMM performance, because it has lower software overhead and reveals the underlying NVMM performance more directly. Overall, the NOVA baseline outperforms other DAX file systems by 1.9× on average, and adding

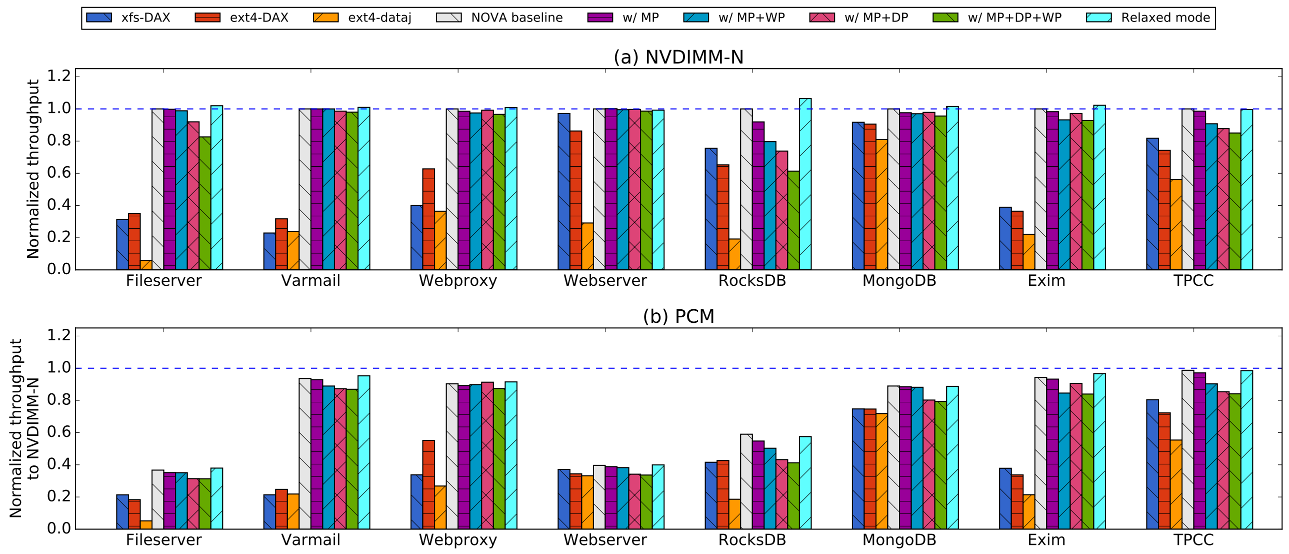

Figure 9: Application performance on NOVA The costs of reliability and the benefits of relaxed mode are both smaller for applications. Read-intensive workloads (e.g., web-proxy) show little change. Reliability impacts the performance of databases and key-value stores more.

Application	Data size	Notes
Filebench-fileserver	64 GB	R/W ratio: 1:2
Filebench-varmail	32 GB	R/W ratio: 1:1
Filebench-webproxy	32 GB	R/W ratio: 5:1
Filebench-webserver	32 GB	R/W ratio: 10:1
RocksDB	8 GB	db_bench's overwrite test
MongoDB	10 GB	YCSB's 50/50-read/write
Exim	4 GB	Mail server
TPC-C	26 GB	The 'Payment' query

Table 1: Application Benchmarks

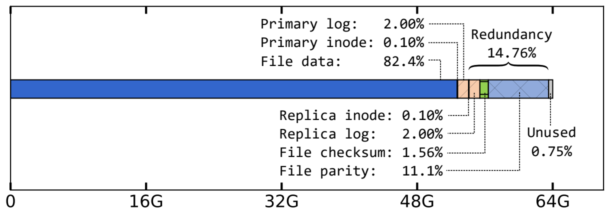

Figure 10: NVMM storage utilization. NOVA file system utilization by a large file server workload.

full protection reduces performance by 11% on average compared to the baseline.

5.5 NVMM storage utilization

Protecting data integrity via redundancy inevitably introduces storage overheads. Figure 10 shows the break down of space that different (meta)data structures occupy in an aged, 64 GB NOVA file system. Overall, NOVA devotes 14.8% of storage space to improving reliability. Of this, metadata redundancy accounts for 2.1% and data redundancy occupies 12.7%.

6. Conclusion

We have used NOVA to explore the unique challenges that improving NVMM file system reliability presents. The solutions that NOVA implements facilitate backups by taking consistent snapshots of the file system and provide significant protection against media errors and corruption due to software errors.

The extra storage required to implement these changes is modest, but their performance impact is significant for some applications. In particular, the cost of checking and maintaining checksums and parity for file data incurs a steep cost for both reads and writes, despite our use of very fast (XOR parity) and hardware accelerated (CRC) mechanisms. Providing atomicity for unaligned writes is also a performance bottleneck.

These costs suggest that NVMM file systems should provide users with a range of protection options that trade off performance against the level of protection and consistency. For instance, NOVA can selectively disable checksum based file data protection and the CR0-based write protection mechanism. Relaxed mode disables copy-on-write.

Making these policy decisions rationally is currently difficult due to a lack of two pieces of information. First, the rate of uncorrectable media errors in emerging NVMM technologies is not publicly known. Second, the frequency and size of scribbles has not been studied in detail. Without a better understanding in these areas, it is hard to determine whether the costs of these techniques are worth the benefits they provide.

Despite these uncertainties, NOVA demonstrates that NVMM file system can provide strong reliability guarantees while providing high performance and supporting DAX-style `mmap()`. It also makes a clear case for developing special file systems and reliability mechanisms for NVMM rather than adapting existing schemes: The challenges NVMMs presents are different, different solutions are appropriate, and the systems built with these differences in mind can be very fast and highly reliable.

References

[1] Advanced configuration and power interface specification. http://www.uefi.org/sites/default/files/resources/ACPI_6_1.pdf.

[2] Intel ships first Optane memory modules for testing. http://www.pcworld.com/article/3162177/storage/intel-ships-first-optane-memory-modules-for-testing.html.

[3] Intel and Micron produce breakthrough memory technology. http://newsroom.intel.com/community/intel_newsroom/blog/2015/07/28/intel-and-micron-produce-breakthrough-memory-technology.

[4] AXBOE, J. Flexible I/O Tester. https://github.com/axboe/fio.

[5] BAILEY, K., CEZE, L., GRIBBLE, S. D., AND LEVY, H. M. Operating system implications of fast, cheap, non-volatile memory. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems* (Berkeley, CA, USA, 2011), HotOS'13, USENIX Association, pp. 2–2.

[6] BAIRAVASUNDARAM, L. N., ARPACI-DUSSEAU, A. C., ARPACI-DUSSEAU, R. H., GOODSON, G. R., AND SCHROEDER, B. An Analysis of Data Corruption in the Storage Stack. *Trans. Storage 4*, 3 (Nov. 2008), 8:1–8:28.

[7] BAIRAVASUNDARAM, L. N., GOODSON, G. R., PASUPATHY, S., AND SCHINDLER, J. An Analysis of Latent Sector Errors in Disk Drives. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (New York, NY, USA, 2007), SIGMETRICS '07, ACM, pp. 289–300.

[8] BAIRAVASUNDARAM, L. N., RUNGTA, M., AGRAWA, N., ARPACI-DUSSEAU, A. C., ARPACI-DUSSEAU, R. H., AND SWIFT, M. M. Analyzing the effects of disk-pointer corruption. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)* (June 2008), pp. 502–511.

[9] BHANDARI, K., CHAKRABARTI, D. R., AND BOEHM, H.-J. Implications of CPU Caching on Byte-addressable Non-volatile Memory Programming. Tech. rep., HP Technical Report HPL-2012-236, 2012.

[10] BHASKARAN, M. S., XU, J., AND SWANSON, S. 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* (New York, NY, USA, 2013), INFLOW '13, ACM, pp. 1:1–1:9.

[11] BONWICK, J., AND MOORE, B. ZFS: The Last Word in File Systems, 2007.

[12] BREITWISCH, M. J. Phase change memory. *Interconnect Technology Conference, 2008. IITC 2008. International* (June 2008), 219–221.

[13] CAULFIELD, A. M., MOLLOV, T. I., EISNER, L. A., DE, A., COBURN, J., AND SWANSON, S. 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* (New York, NY, USA, 2012), ASPLOS XVII, ACM, pp. 387–400.

[14] CHEN, F., MESNIER, M. P., AND HAHN, S. A Protected Block Device for Persistent Memory. In *Mass Storage Systems and Technologies (MSST), 2014 30th Symposium on* (2014), IEEE, pp. 1–12.

[15] CHIDAMBARAM, V., PILLAI, T. S., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. Optimistic crash consistency. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 228–243.

[16] CHINNER, D. xfs: updates for 4.2-rc1. http://oss.sgi.com/archives/xfs/2015-06/msg00478.html.

[17] CHOI, Y., SONG, I., PARK, M.-H., CHUNG, H., CHANG, S., CHO, B., KIM, J., OH, Y., KWON, D., SUNWOO, J., SHIN, J., RHO, Y., LEE, C., KANG, M. G., LEE, J., KWON, Y., KIM, S., KIM, J., LEE, Y.-J., WANG, Q., CHA, S., AHN, S., HORII, H., LEE, J., KIM, K., JOO, H., LEE, K., LEE, Y.-T., YOO, J., AND JEONG, G. A 20nm 1.8V 8Gb PRAM with 40MB/s program bandwidth. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International* (Feb 2012), pp. 46–48.

[18] COBURN, J., CAULFIELD, A. M., AKEL, A., GRUPP, L. M., GUPTA, R. K., JHALA, R., AND SWANSON, S. 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* (New York, NY, USA, 2011), ASPLOS '11, ACM, pp. 105–118.

[19] CONDIT, J., NIGHTINGALE, E. B., FROST, C., IPEK, E., LEE, B., BURGER, D., AND COETZEE, D. Better I/O through byte-addressable, persistent memory. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles* (New York, NY, USA, 2009), SOSP '09, ACM, pp. 133–146.

[20] DEAN, J., AND GHEMAWAT, S. LevelDB. https://github.com/google/leveldb.

[21] DO, T., HARTER, T., LIU, Y., GUNAWI, H. S., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. HARDFS: Hardening HDFS with Selective and Lightweight Versioning. In *Presented as part of the 11th USENIX Conference on File and Storage Technologies (FAST 13)* (San Jose, CA, 2013), USENIX, pp. 105–118.

[22] DRAGGA, C., AND SANTRY, D. J. GCTrees: Garbage collecting snapshots. *ACM Transactions on Storage (TOS) 12*, 1 (2016), 4.

[23] DULLOOR, S. R., KUMAR, S., KESHAVAMURTHY, A., LANTZ, P., REDDY, D., SANKARAN, R., AND JACKSON, J. System Software for Persistent Memory. In *Proceedings of the Ninth European Conference on Computer Systems* (New York, NY, USA, 2014), EuroSys '14, ACM, pp. 15:1–15:15.

[24] Exim Internet Mailer. http://www.exim.org.

[25] FACEBOOK. RocksDB. http://rocksdb.org.

[26] FACKENTHAL, R., KITAGAWA, M., OTSUKA, W., PRALL, K., MILLS, D., TSUTSUI, K., JAVANIFARD, J., TEDROW, K., TSUSHIMA, T., SHIBAHARA, Y., AND HUSH, G. 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* (Feb 2014), pp. 338–339.

[27] FRYER, D., QIN, M., SUN, J., LEE, K. W., BROWN, A. D., AND GOEL, A. Checking the Integrity of Transactional Mechanisms. *Trans. Storage 10*, 4 (Oct. 2014), 17:1–17:23.

[28] FRYER, D., SUN, K., MAHMOOD, R., CHENG, T., BENJAMIN, S., GOEL, A., AND BROWN, A. D. Recon: Verifying File System Consistency at Runtime. *Trans. Storage 8*, 4 (Dec. 2012), 15:1–15:29.

[29] GANESAN, A., ALAGAPPAN, R., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. Redundancy Does Not Imply Fault Tolerance: Analysis of Distributed Storage Reactions to Single Errors and Corruptions. In *15th USENIX Conference on File and Storage Technologies (FAST 17)* (Santa Clara, CA, 2017), USENIX Association, pp. 149–166.

[30] GUNAWI, H. S., RAJIMWALE, A., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. Sqck: A declarative file system checker. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2008), OSDI'08, USENIX Association, pp. 131–146.

[31] GUNAWI, H. S., RUBIO-GONZÁLEZ, C., ARPACI-DUSSEAU, A. C., ARPACI-DUSSEA, R. H., AND LIBLIT, B. Eio: Error handling is occasionally correct. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2008), FAST'08, USENIX Association, pp. 14:1–14:16.

[32] HARRIS, R. Windows leaps into the NVM revolution. http://www.zdnet.com/article/windows-leaps-into-the-nvm-revolution/.

[33] HITZ, D., LAU, J., AND MALCOLM, M. A. File system design for an NFS file server appliance. In *USENIX Winter* (1994), pp. 235–246.

[34] INTEL. Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3, Chapter 15. https://software.intel.com/sites/default/files/managed/a4/60/325384-sdm-vol-3abcd.pdf, Version December 2016.

[35] INTEL CORPORATION. Intel Optane Memory. http://www.intel.com/content/www/us/en/architecture-and-technology/optane-memory.html.

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

[37] NVDIMM Namespace Specification. http://pmem.io/documents/NVDIMM_Namespace_Spec.pdf.

[38] INTERNATIONAL, S. G. XFS: A High-performance Journaling Filesystem. http://oss.sgi.com/projects/xfs.

[39] JOHNSON, R., PANDIS, I., HARDAVELLAS, N., AILAMAKI, A., AND FALSAFI, B. Shore-MT: A Scalable Storage Manager for the Multicore Era. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology* (New York, NY, USA, 2009), EDBT '09, ACM, pp. 24–35.

[40] KAWAHARA, T. Scalable Spin-Transfer Torque RAM Technology for Normally-Off Computing. *Design & Test of Computers, IEEE 28*, 1 (Jan 2011), 52–63.

[41] KERNEL.ORG. [PATCH v14] x86, mce: Add memcpy_mcsafe(). https://patchwork.kernel.org/patch/8427231.

[42] KESAVAN, R., SINGH, R., GRUSECKI, T., AND PATEL, Y. Algorithms and Data Structures for Efficient Free Space Reclamation in WAFL. In *15th USENIX Conference on File and Storage Technologies (FAST 17)* (2017), USENIX Association.

[43] KOLLI, A., ROSEN, J., DIESTELHORST, S., SAIDI, A., PELLEY, S., LIU, S., CHEN, P. M., AND WENISCH, T. F. Delegated Persist Ordering. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (Oct 2016), pp. 1–13.

[44] KONISHI, R., AMAGAI, Y., SATO, K., HIFUMI, H., KIHARA, S., AND MORIAI, S. The Linux implementation of a log-structured file system. *ACM SIGOPS Operating Systems Review 40*, 3 (2006), 102–107.

[45] KUMAR, H., PATEL, Y., KESAVAN, R., AND MAKAM, S. High Performance Metadata Integrity Protection in the WAFL Copy-on-Write File System. In *15th USENIX Conference on File and Storage Technologies (FAST 17)* (Santa Clara, CA, 2017), USENIX Association, pp. 197–212.

[46] LEE, B. C., IPEK, E., MUTLU, O., AND BURGER, D. Architecting phase change memory as a scalable DRAM alternative. In *ISCA '09: Proceedings of the 36th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2009), ACM, pp. 2–13.

[47] LEE, E., JANG, J. E., KIM, T., AND BAHN, H. On-demand snapshot: An efficient versioning file system for phase-change memory. *IEEE Transactions on Knowledge and Data Engineering 25*, 12 (Dec 2013), 2841–2853.

[48] NVM Library. http://pmem.io/nvml.

[49] LU, L., PILLAI, T. S., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. WiscKey: Separating Keys

from Values in SSD-conscious Storage. In *14th USENIX Conference on File and Storage Technologies (FAST 16)* (Santa Clara, CA, 2016), USENIX Association, pp. 133–148.

[50] MEZA, J., WU, Q., KUMAR, S., AND MUTLU, O. A Large-Scale Study of Flash Memory Failures in the Field. In *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (New York, NY, USA, 2015), SIGMETRICS '15, ACM, pp. 177–190.

[51] MEZA, J., WU, Q., KUMAR, S., AND MUTLU, O. Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (June 2015), pp. 415–426.

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

[53] MIN, C., KANG, W.-H., KIM, T., LEE, S.-W., AND EOM, Y. I. Lightweight Application-Level Crash Consistency on Transactional Flash Storage. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)* (Santa Clara, CA, 2015), USENIX Association, pp. 221–234.

[54] MONGODB, INC. MongoDB. https://www.mongodb.com.

[55] MORARU, I., ANDERSEN, D. G., KAMINSKY, M., TOLIA, N., RANGANATHAN, P., AND BINKERT, N. Consistent, Durable, and Safe Memory Management for Byte-addressable Non Volatile Main Memory. In *Proceedings of the First ACM SIGOPS Conference on Timely Results in Operating Systems* (New York, NY, USA, 2013), TRIOS '13, ACM, pp. 1:1–1:17.

[56] NALLI, S., HARIA, S., HILL, M. D., SWIFT, M. M., VOLOS, H., AND KEETON, K. An Analysis of Persistent Memory Use with WHISPER. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2017), ASPLOS '17, ACM, pp. 135–148.

[57] NARAYANAN, I., WANG, D., JEON, M., SHARMA, B., CAULFIELD, L., SIVASUBRAMANIAM, A., CUTLER, B., LIU, J., KHESSIB, B., AND VAID, K. SSD Failures in Datacenters: What? When? And Why? In *Proceedings of the 9th ACM International on Systems and Storage Conference* (New York, NY, USA, 2016), SYSTOR '16, ACM, pp. 7:1–7:11.

[58] NOGUCHI, H., IKEGAMI, K., KUSHIDA, K., ABE, K., ITAI, S., TAKAYA, S., SHIMOMURA, N., ITO, J., KAWASUMI, A., HARA, H., AND FUJITA, S. 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* (Feb 2015), pp. 1–3.

[59] OU, J., AND SHU, J. Fast and failure-consistent updates of application data in non-volatile main memory file system. In *2016 32nd Symposium on Mass Storage Systems and Technologies (MSST)* (May 2016), pp. 1–15.

[60] OUKID, I., LASPERAS, J., NICA, A., WILLHALM, T., AND LEHNER, W. 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* (New York, NY, USA, 2016), SIGMOD '16, ACM, pp. 371–386.

[61] OUSTERHOUT, J., GOPALAN, A., GUPTA, A., KEJRIWAL, A., LEE, C., MONTAZERI, B., ONGARO, D., PARK, S. J., QIN, H., ROSENBLUM, M., RUMBLE, S., STUTSMAN, R., AND YANG, S. The RAMCloud Storage System. *ACM Trans. Comput. Syst. 33*, 3 (Aug. 2015), 7:1–7:55.

[62] PARK, S., KELLY, T., AND SHEN, K. Failure-atomic Msync(): A Simple and Efficient Mechanism for Preserving the Integrity of Durable Data. In *Proceedings of the 8th ACM European Conference on Computer Systems* (New York, NY, USA, 2013), EuroSys '13, ACM, pp. 225–238.

[63] PELLEY, S., CHEN, P. M., AND WENISCH, T. F. Memory persistency. In *Proceeding of the 41st Annual International Symposium on Computer Architecture* (Piscataway, NJ, USA, 2014), ISCA '14, IEEE Press, pp. 265–276.

[64] PETERSON, Z., AND BURNS, R. Ext3cow: A time-shifting file system for regulatory compliance. *Trans. Storage 1*, 2 (May 2005), 190–212.

[65] PILLAI, T. S., ALAGAPPAN, R., LU, L., CHIDAMBARAM, V., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. Application Crash Consistency and Performance with CCFS. In *15th USENIX Conference on File and Storage Technologies (FAST 17)* (Santa Clara, CA, 2017), USENIX Association, pp. 181–196.

[66] PRABHAKARAN, V., BAIRAVASUNDARAM, L. N., AGRAWAL, N., GUNAWI, H. S., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. IRON File Systems. In *The ACM Symposium on Operating Systems Principles (SOSP)* (2005), ACM.

[67] RAOUX, S., BURR, G., BREITWISCH, M., RETTNER, C., CHEN, Y., SHELBY, R., SALINGA, M., KREBS, D., CHEN, S.-H., LUNG, H. L., AND LAM, C. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development 52*, 4.5 (July 2008), 465–479.

[68] RODEH, O. B-trees, Shadowing, and Clones. *Trans. Storage 3*, 4 (Feb. 2008), 2:1–2:27.

[69] RODEH, O., BACIK, J., AND MASON, C. BTRFS: The Linux B-Tree Filesystem. *Trans. Storage 9*, 3 (Aug. 2013), 9:1–9:32.

[70] ROSENBLUM, M., AND OUSTERHOUT, J. K. The Design and Implementation of a Log-structured File System. In *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 1991), SOSP '91, ACM, pp. 1–15.

[71] RUMBLE, S. M., KEJRIWAL, A., AND OUSTERHOUT, J. Log-structured Memory for DRAM-based Storage. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies* (Santa Clara, CA, 2014), FAST '14, USENIX, pp. 1–16.

[72] SAINIO, A. NVDIMM: Changes are Here So What's Next? In *In-Memory Computing Summit 2016* (2016).

[73] SANTRY, D. S., FEELEY, M. J., HUTCHINSON, N. C., VEITCH, A. C., CARTON, R. W., AND OFIR, J. Deciding when to forget in the elephant file system. In *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 1999), SOSP '99, ACM, pp. 110–123.

[74] SCHROEDER, B., DAMOURAS, S., AND GILL, P. Understanding Latent Sector Errors and How to Protect Against Them. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2010), FAST'10, USENIX Association, pp. 6–6.

[75] SCHROEDER, B., AND GIBSON, G. A. Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you? In *USENIX Conference on File and Storage Technologies (FAST)* (2007).

[76] SCHROEDER, B., LAGISETTY, R., AND MERCHANT, A. Flash Reliability in Production: The Expected and the Unexpected. In *14th USENIX Conference on File and Storage Technologies (FAST 16)* (Santa Clara, CA, 2016), USENIX Association, pp. 67–80.

[77] SCHROEDER, B., PINHEIRO, E., AND WEBER, W.-D. DRAM Errors in the Wild: A Large-scale Field Study. In *ACM SIGMETRICS* (2009).

[78] SEHGAL, P., BASU, S., SRINIVASAN, K., AND VORUGANTI, K. An Empirical Study of File Systems on NVM. In *Proceedings of the 2015 IEEE Symposium on Mass Storage Systems and Technologies (MSST'15)* (2015).

[79] SIGURBJARNARSON, H., BORNHOLT, J., TORLAK, E., AND WANG, X. Push-button verification of file systems via crash refinement. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2016), OSDI'16, USENIX Association, pp. 1–16.

[80] SOULES, C. A. N., GOODSON, G. R., STRUNK, J. D., AND GANGER, G. R. Metadata Efficiency in Versioning File Systems. In *Proceedings of the 2Nd USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2003), FAST '03, USENIX Association, pp. 43–58.

[81] SPILLANE, R. P., GAIKWAD, S., CHINNI, M., ZADOK, E., AND WRIGHT, C. P. Enabling Transactional File Access via Lightweight Kernel Extensions. In *Proccedings of the 7th Conference on File and Storage Technologies* (Berkeley, CA, USA, 2009), FAST '09, USENIX Association, pp. 29–42.

[82] SRIDHARAN, V., DEBARDELEBEN, N., BLANCHARD, S., FERREIRA, K. B., STEARLEY, J., SHALF, J., AND GURUMURTHI, S. Memory Errors in Modern Systems: The Good, The Bad, and The Ugly. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2015), ACM.

[83] SRIDHARAN, V., AND LIBERTY, D. A study of DRAM failures in the field. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for* (Nov 2012), pp. 1–11.

[84] STRUKOV, D. B., SNIDER, G. S., STEWART, D. R., AND WILLIAMS, R. S. The missing memristor found. *Nature 453*, 7191 (2008), 80–83.

[85] SUBRAMANIAN, S., SUNDARARAMAN, S., TALAGALA, N., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. Snapshots in a Flash with ioSnap. In *Proceedings of the Ninth European Conference on Computer Systems* (2014), ACM, p. 23.

[86] SUNDARARAMAN, S., SUBRAMANIAN, S., RAJIMWALE, A., ARPACI-DUSSEAU, A. C., ARPACI-DUSSEAU, R. H., AND SWIFT, M. M. Membrane: Operating System Support for Restartable File Systems. *Trans. Storage 6*, 3 (Sept. 2010), 11:1–11:30.

[87] SUZUKI, K., AND SWANSON, S. The Non-Volatile Memory Technology Database (NVMDB). Tech. Rep. CS2015-1011, Department of Computer Science & Engineering, University of California, San Diego, May 2015. http://nvmdb.ucsd.edu.

[88] TARASOV, V., ZADOK, E., AND SHEPLER, S. Filebench: A flexible framework for file system benchmarking. *USENIX; login 41* (2016).

[89] TWEEDIE, S. C. Journaling the Linux ext2fs Filesystem. In *LinuxExpo'98: Proceedings of The 4th Annual Linux Expo* (1998).

[90] VENKATARAMAN, S., TOLIA, N., RANGANATHAN, P., AND CAMPBELL, R. Consistent and durable data structures for non-volatile byte-addressable memory. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies* (San Jose, CA, USA, February 2011), FAST '11.

[91] VERMA, R., MENDEZ, A. A., PARK, S., MANNARSWAMY, S. S., KELLY, T. P., AND III, C. B. M. Failure-Atomic Updates of Application Data in a Linux File System. In *13th USENIX Conference on File and Storage Technologies (FAST 15)* (Santa Clara, CA, 2015), USENIX Association, pp. 203–211.

[92] VOLOS, H., NALLI, S., PANNEERSELVAM, S., VARADARAJAN, V., SAXENA, P., AND SWIFT, M. M. Aerie: Flexible File-system Interfaces to Storage-class Memory. In *Proceedings of the Ninth European Conference on Computer Systems* (New York, NY, USA, 2014), EuroSys '14, ACM, pp. 14:1–14:14.

[93] VOLOS, H., TACK, A. J., AND SWIFT, M. M. Mnemosyne: Lightweight Persistent Memory. In *ASPLOS '11: Proceeding of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2011), ACM.

[94] VUČINIĆ, D., WANG, Q., GUYOT, C., MATEESCU, R., BLAGOJEVIĆ, F., FRANCA-NETO, L., MOAL, D. L., BUNKER, T., XU, J., SWANSON, S., AND BANDIĆ, Z.

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* (Santa Clara, CA, 2014), FAST '14, USENIX, pp. 309–315.

[95] WILCOX, M. Add support for NV-DIMMs to ext4. https://lwn.net/Articles/613384/.

[96] WU, X., AND REDDY, A. L. N. SCMFS: A File System for Storage Class Memory. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2011), SC '11, ACM, pp. 39:1–39:11.

[97] XU, J., AND SWANSON, S. NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories. In *14th USENIX Conference on File and Storage Technologies (FAST 16)* (Santa Clara, CA, Feb. 2016), USENIX Association, pp. 323–338.

[98] YANG, J., MINTURN, D. B., AND HADY, F. When poll is better than interrupt. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2012), FAST '12, USENIX, pp. 3–3.

[99] YANG, J., SAR, C., AND ENGLER, D. Explode: A lightweight, general system for finding serious storage system errors. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2006), OSDI '06, USENIX Association, pp. 131–146.

[100] YANG, J., TWOHEY, P., ENGLER, D., AND MUSUVATHI, M. Using model checking to find serious file system errors. *ACM Trans. Comput. Syst. 24*, 4 (Nov. 2006), 393–423.

[101] YANG, J., WEI, Q., CHEN, C., WANG, C., YONG, K. L., AND HE, B. NV-Tree: Reducing Consistency Cost for NVM-based Single Level Systems. In *13th USENIX Conference on File and Storage Technologies* (Santa Clara, CA, Feb. 2015), FAST '15, USENIX Association, pp. 167–181.

[102] ZHANG, Y., RAJIMWALE, A., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. End-to-end Data Integrity for File Systems: A ZFS Case Study. In *USENIX Conference on File and Storage Technologies (FAST)* (2010).

[103] ZHANG, Y., AND SWANSON, S. A Study of Application Performance with Non-Volatile Main Memory. In *Proceedings of the 2015 IEEE Symposium on Mass Storage Systems and Technologies (MSST'15)* (2015).

[104] ZHAO, J., LI, S., YOON, D. H., XIE, Y., AND JOUPPI, N. P. Kiln: Closing the Performance Gap Between Systems With and Without Persistence Support. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture* (New York, NY, USA, 2013), MICRO-46, ACM, pp. 421–432.

[105] ZHENG, S., HUANG, L., LIU, H., WU, L., AND ZHA, J. HMVFS: A Hybrid Memory Versioning File System. In *Proceedings of the 32st Symposium on Mass Storage Systems and Technologies (MSST), IEEE* (2016).