NOVA

A Log-Structured File System for Hybrid Volatile/Non-Volatile Main Memories

JIAN XU AND STEVEN SWANSON

Jian Xu is a PhD candidate in the Department of Computer Science and Engineering at the University of California, San Diego. His research interests include operating system and system software design for next-generation storage technologies. He is working together with Professor Steven Swanson in the Non-Volatile Systems Laboratory. jix024@cs.ucsd.edu

Steven Swanson is a Full Professor in the Department of Computer Science and Engineering at the University of California, San Diego and the Director of the Non-Volatile Systems Laboratory. His research interests include systems, architecture, security, and reliability issues surrounding non-volatile, solid-state memories. He has also co-led projects to develop low-power co-processors for irregular applications and to devise software techniques for using multiple processors to speed up single-threaded computations. In previous lives he has worked on scalable dataflow architectures, ubiquitous computing, and simultaneous multithreading. He received his PhD from the University of Washington in 2006. swanson@cs.ucsd.edu

NOVA is a new kind of log-structured file system designed for emerging non-volatile main memory (NVMM) technologies, like Intel's 3D XPoint memory. NOVA provides better performance and stronger consistency guarantees than either conventional block-based file systems or other, recently proposed NVMM file systems. NOVA's focus on NVMMs leads to three key design decisions: NOVA maintains a separate metadata log for each file and directory, uses copy-on-write to provide write atomicity, and applies lightweight journaling to make complex operations atomic. These techniques allow NOVA to improve performance by a factor of between 3.1 and 13.5 without jeopardizing crash consistency.

Emerging non-volatile memory (NVM) technologies such as spin-torque transfer, phase change, resistive memories [2, 8], and Intel and Micron's 3D Xpoint [1] technology promise to revolutionize I/O performance. Researchers have proposed placing NVMs on the processor's memory bus alongside conventional DRAM, leading to hybrid volatile/non-volatile main memory systems [12]. Combining faster, volatile DRAM with slightly slower, denser non-volatile main memories (NVMMs) offers the possibility of storage systems that combine the best characteristics of both technologies.

Hybrid DRAM/NVMM storage systems present a host of opportunities and challenges for system designers. These systems should improve conventional file access performance and allow applications to abandon slow read/write file interfaces in favor of faster memory-mapped, load/store access interfaces. They will also allow for increased concurrency and efficiently support more flexible access patterns. File systems must realize these advantages while still providing the strong consistency guarantees that applications require and respecting the limitations of emerging memories (e.g., limited program cycles).

Disk-based file systems are not suitable for hybrid memory systems because NVMM has different characteristics from disks in both performance and consistency guarantees. As a result, naively running disk-based file systems on NVMM cannot fully exploit NVMM's high performance, and performance optimizations compromise consistency on system failure.

Existing NVMM file systems such as BPFS [3], PMFS [4], and ext4-DAX [10] also fail to provide the combination of performance and consistency NVMM should deliver. They use shadow paging and journaling to provide metadata atomicity, but these mechanisms incur high overheads that limit performance. PMFS and ext4-DAX avoid some of these costs by sacrificing data atomicity.

To provide consistency *and* high performance in an NVMM file system, we have created the *NOn-Volatile memory Accelerated (NOVA)* file system. NOVA rethinks conventional log-structured file system techniques to exploit the fast random access that hybrid memory systems provide. The result is that NOVA supports massive concurrency, keeps log sizes small, and minimizes garbage collection costs while providing strong consistency guarantees and very high performance.

Several aspects of NOVA set it apart from previous log-structured file systems. NOVA assigns each inode a separate log to maximize concurrency. NOVA stores the logs as linked lists, so they do not need to be contiguous in memory, and it uses atomic updates to a log's tail pointer to provide atomic log append. For operations that span multiple inodes, NOVA uses lightweight journaling.

NOVA uses copy-on-write for file data instead of storing it in the log. The resulting logs are compact, so the recovery process only needs to scan a small fraction of the NVMM. This also significantly reduces garbage collection overhead, allowing NOVA to sustain good performance even when the file system is nearly full. Finally, NOVA provides an `atomic mmap` interface that simplifies the tasks of writing programs that directly access NVMM via load and store instructions.

Our experiments show that NOVA is significantly faster than existing file systems in a wide range of applications and outperforms file systems that provide the same data consistency guarantees by 3.1x–13.5x. And when measuring garbage collection and recovery overheads, we find that NOVA provides stable performance under high NVMM utilization levels and fast recovery in case of system failure.

More details and results are available in our FAST '16 paper [11]. NOVA is open source and available at https://github.com/NVSL/NOVA.

NOVA Design Overview

NOVA is a log-structured POSIX file system. However, since it targets a different storage technology, NOVA looks very different from conventional log-structured file systems [9] that are built to maximize disk bandwidth.

We designed NOVA based on three observations. First, since NVMMs support fast, highly concurrent random accesses, using multiple logs does not negatively impact performance. Second, logs do not need to be contiguous, because random access is cheap. Third, data structures that support fast search (e.g., tree structures) are more difficult to implement correctly and efficiently in NVMM than in DRAM. Based on these observations, we made the following design decisions in NOVA.

Give each inode its own log: Unlike conventional log-structured file systems, each inode in NOVA has its own log, allowing concurrent updates across files without synchronization. It also means recovery is very fast, since NOVA can replay many logs simultaneously.

Keep logs in NVMM and indexes in DRAM: NOVA keeps log and file data in NVMM and builds highly optimized radix trees in DRAM to quickly locate file data. This means that searching the in-NVMM data structures is not usually necessary, so they can remain simple, easy to verify, and compact.

Implement the log as a singly linked list: The locality benefits of sequential logs are less important in NVMM-based storage, so NOVA uses a linked list of 4 KB NVMM pages to hold the log and stores the next page pointer in the end of each log page. As a result, NOVA can perform log cleaning at fine-grained, page-size granularity, and reclaiming log pages that contain only stale entries requires just a few pointer assignments.

Use logging to provide atomicity for simple, common-case operations: NOVA is log-structured because this provides cheaper atomicity for simple updates than journaling or shadow paging. To atomically write data to a log, NOVA first appends data to the log and then atomically updates the log tail to commit the updates, thus avoiding both the duplicate writes of journaling and the cascading updates of shadow paging.

Use lightweight journaling for more complex operations: Some directory operations, such as a move between directories, span multiple inodes. For these, NOVA uses journaling to atomically update multiple logs: NOVA first writes data at the end of each inode's log, and then journals the log tail updates to update them atomically. NOVA journaling is lightweight since it only involves log tails (as opposed to file data or metadata). The journals are very small—less than 64 bytes—since the most complex POSIX operation (`rename()`) involves up to four inodes, and each journal entry consists of eight bytes for the address of the log tail pointer and eight bytes for the updated value.

Use shadow paging for file data: NOVA uses copy-on-write for modified file data and appends metadata for the write to the log. The metadata describe the update and point to the data pages.

Using copy-on-write for file data results in shorter logs, accelerating the recovery process. It also makes garbage collection simpler and more efficient, since NOVA never has to copy file data out of the log to reclaim a log page. Finally, since it can reclaim stale data pages immediately, NOVA can sustain performance even under heavy write loads and high NVMM utilization levels.

Implementing NOVA

NOVA's core data structures focus on making file and directory operations fast and efficient. They also provide support for an mmap interface that makes using raw NVMM easier for programmers, while providing for efficient garbage collection and fast recovery in the case of a system failure.

NVMM Data Structures and Space Management

Figure 1 shows the high-level layout of NOVA data structures in a region of NVMM that it manages. NOVA divides the NVMM into four parts: the superblock and recovery inode, the inode

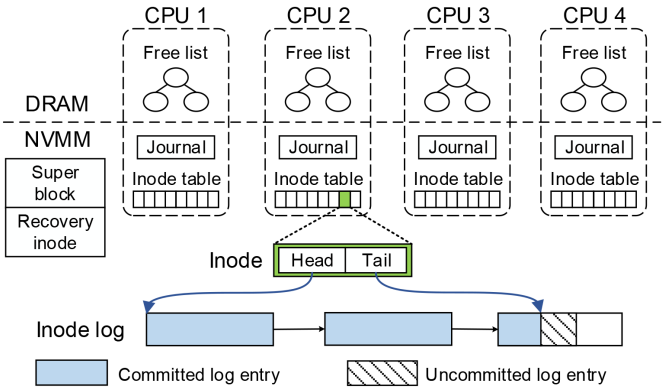

Figure 1: NOVA data structure layout. NOVA has per-CPU free lists, journals, and inode tables to ensure good scalability. Each inode has a separate log consisting of a singly linked list of 4 KB log pages; the tail pointer in the inode points to the latest committed entry in the log.

tables, the journals, and log/data pages. The superblock contains global file system information, the recovery inode stores recovery information that accelerates NOVA remount after a clean shutdown, the inode tables contain inodes, the journals provide atomicity to directory operations, and the remaining area contains NVMM log and data pages.

To ensure high scalability, NOVA maintains an inode table, journal, and NVMM free page list at each CPU to avoid global locking and scalability bottlenecks: partitioning the inode table across CPUs avoids inode allocation contention and allows for parallel scanning in failure recovery. Per-CPU journals allow for concurrent transactions, and per-CPU NVMM free list provides concurrent NVMM allocation and deallocation. NOVA puts the free lists in DRAM to reduce consistency overheads.

Figure 2 shows the structure of a NOVA file. A NOVA inode contains pointers to the head and tail of its log. The log is a linked list of 4 KB pages, and the tail always points to the latest committed log entry. A file inode's log contains two kinds of log entries: inode update entries for metadata modifications and file write entries for writes. Each open file has a radix tree in DRAM to locate data in the file by the file offset. NOVA scans the log from head to tail to rebuild the DRAM data structures when the system accesses the inode for the first time.

NOVA provides fast atomicity for metadata, data, and mmap updates using a technique that combines log structuring and journaling. This technique uses three mechanisms:

64-bit atomic updates: NOVA uses 64-bit in-place writes to directly modify metadata for some operations (e.g., the file's `atime` for reads) and uses them to commit updates to the log by updating the inode's log tail pointer.

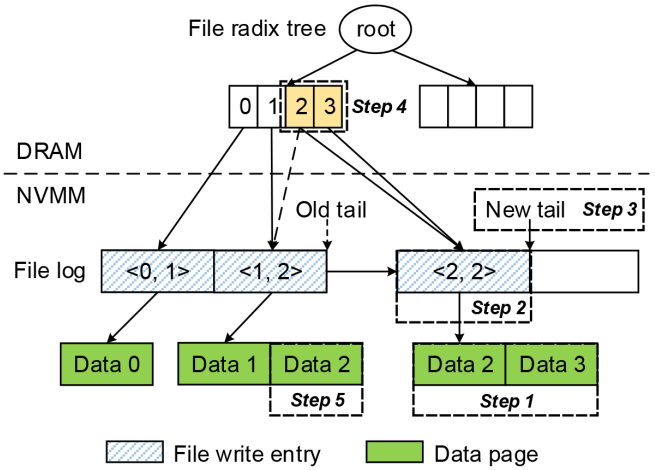

Figure 2: NOVA file structure. An 8 KB (i.e., two-page) write to page 2 (<2, 2>) of a file requires five steps. NOVA first writes a copy of the data to new pages (step 1) and appends the file write entry (step 2). Then it updates the log tail (step 3) and the radix tree (step 4). Finally, NOVA returns the old version of the data to the allocator (step 5).

Logging: NOVA uses the inode's log to record operations that modify a single inode. These include operations such as `write`, `msync`, and `chmod`. The logs are independent of one another.

Lightweight journaling: For directory operations that require changes to multiple inodes (e.g., `create`, `unlink`, and `rename`), NOVA uses lightweight journaling to provide atomicity.

File Operations

NOVA uses copy-on-write for file data. On each write, NOVA writes the new version of the modified pages to newly allocated NVMM. Then it appends a write entry to the inode's log that describes the write and points to those pages.

Figure 2 illustrates a `write` operation. The notation *<file page offset, number of pages>* denotes the page offset and number of pages a `write` affects. The first two entries in the log describe two `writes`, <0, 1> and <1, 2>, of 4 KB and 8 KB (i.e., one and two pages), respectively. A third, 8 KB `write`, <2, 2>, is in flight.

To perform the <2, 2> `write`, NOVA fills data pages and then appends the <2, 2> entry to the file's inode log. Then NOVA atomically updates the log tail to commit the write, and updates the radix tree in DRAM, so that offset "2" points to the new entry. The NVMM page that holds the old contents of page 2 returns to the free list immediately. During the operation, a per-inode lock protects the log and the radix tree from concurrent updates. When the `write` system call returns, all the updates are persistent in NVMM.

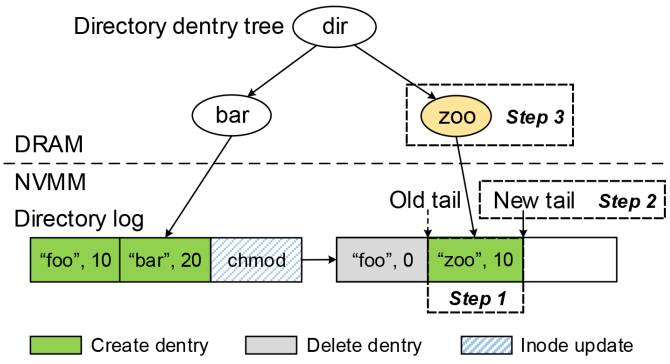

Figure 3: NOVA directory structure. Dentry is shown in <name, inode_number> format. To create a file, NOVA first appends the dentry to the directory's log (step 1), updates the log tail as part of a transaction (step 2), and updates the radix tree (step 3).

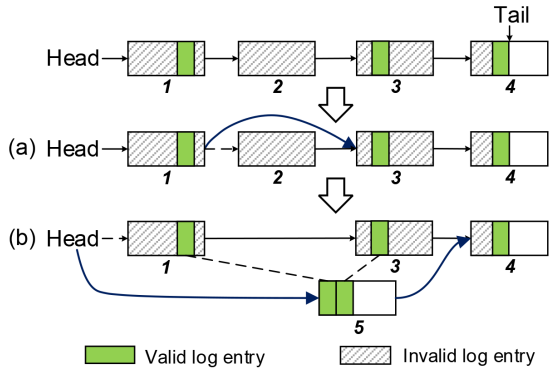

Figure 4: NOVA log cleaning. The linked list structure of log provides simple and efficient garbage collection. Fast GC reclaims invalid log pages by deleting them from the linked list (a), while thorough GC copies live log entries to a new version of the log (b).

If NOVA cannot find a contiguous region of NVMM big enough for the write, it will require multiple log entries. In this case NOVA breaks the `write` into multiple write entries and appends them all to the log to satisfy the request. To maintain atomicity, NOVA commits all the entries with a single update to the log tail pointer.

Directory Operations

Each directory inode's log holds two kinds of entries: directory entries (dentry) and inode update entries. Dentries include the name of the child file/directory, its inode number, and timestamp. NOVA uses the timestamp to atomically update the directory inode's *mtime* and *ctime* with the operation. NOVA appends a dentry to the log when it creates, deletes, or renames a file or subdirectory under that directory.

NOVA adds inode update entries to the directory's log to record updates to the directory's inode (e.g., for `chmod` and `chown`). These operations modify multiple fields of the inode, and the inode update entry provides atomicity.

Figure 3 illustrates the creation of file *zoo* in a directory that already contains file *bar*. The directory has recently undergone a `chmod` operation and used to contain another file, *foo*. The log entries for those operations are visible in the figure. NOVA first selects and initializes an unused inode in the inode table for *zoo* and appends a create dentry of *zoo* to the directory's log. Then NOVA uses the current CPU's journal to atomically update the directory's log tail and set the valid bit of the new inode. Finally, NOVA adds the file to the directory's radix tree in DRAM.

Atomic-mmap

NOVA includes a novel direct NVMM access model with stronger consistency called `atomic-mmap`. When an application uses `atomic-mmap` to map a file into its address space, NOVA allocates *replica pages* from NVMM, copies the file data to the replica pages, and then maps the replicas into the address space. When the application calls `msync` on the replica pages, NOVA handles it as a `write` request described in the previous section, uses `movntq` operation to copy the data from replica pages to data pages directly, and commits the changes atomically. Comparing to PMFS and ext4-DAX that map the NVMM file data pages directly into the application's address space, `atomic-mmap` has higher overhead but provides stronger consistency guarantee.

Garbage Collection

NOVA uses two complementary techniques to reclaim dead log entries: *fast GC* and *thorough GC*. Both use the same criteria to determine whether a log entry is dead: namely, if it is not the last entry in the log and any of the following conditions are met:

- A file write entry is dead if it does not refer to valid data pages.
- An inode update that modifies metadata (e.g., *mode* or *mtime*) is dead if a later inode update modifies the same piece of metadata.
- A create dentry is dead if a corresponding delete dentry is appended to the log. Both dentries become invalid in this case.

Fast GC applies these rules to reclaim log pages that do not contain any live entries by deleting the page from the log's linked list. Figure 4a illustrates this: originally, the log has four pages and all the entries in page 2 are dead. NOVA atomically updates the next page pointer of page 1 to point to page 3, freeing page 2.

Thorough GC compacts log pages by copying live data from several pages into a new page. Figure 4b illustrates thorough GC after fast GC is complete. NOVA allocates a new log page 5 and copies valid log entries in pages 1 and 3 into it. Then NOVA links page 5 to page 4 to create a new log and replace the old one. NOVA does not copy the live entries in page 4 to avoid updating the log tail, so that NOVA can atomically replace the old log by updating the log head pointer.

Workload	Average file size	I/O size (r/w)	Threads	R/W ratio	Number of files (Small/Large)
Fileserver	128 KB	16 KB/16 KB	50	1:2	100K:400K
Webproxy	32 KB	1 MB/16 KB	50	5:1	100K:1M
Webserver	64 KB	1 MB/8 KB	50	10:1	100K:500K
Varmail	32 KB	1 MB/16 KB	50	1:1	100K:1M

Table 1: Filebench workload characteristics. The selected four workloads have different read/write ratios and access patterns.

Mounting, Unmounting, and Recovery

NOVA's design allows for fast mounting and unmounting as well as efficient recovery after a system failure. When NOVA mounts a file system, it must construct two kinds of data structure in DRAM: the per-inode radix trees and the NVMM allocator.

After either a clean unmount or system failure, NOVA rebuilds the radix trees on demand when any application opens a file or directory.

Reconstructing the NVMM allocator state, however, cannot wait. During a normal unmount, NOVA writes the NVMM page allocator state in the recovery inode's log. Then it can quickly rebuild the allocator on remount.

After a system failure, NOVA rebuilds the NVMM allocator by scanning all the inode logs. Fortunately, NOVA can use multiple threads to perform the scan in parallel, and the logs are small since they only contain metadata.

Evaluation

NOVA aims to provide strong consistency, high performance, and fast recovery, and our results show that it achieves these goals.

We have implemented NOVA in the Linux kernel version 4.0 using the existing NVMM hooks that the kernel provides. It currently passes the Linux POSIX file system test suite [7].

We measure NOVA's performance using Intel's Persistent Memory Emulation Platform (PMEP) [4], a dual-socket Xeon platform with special CPU microcode and firmware that allows it to emulate some aspects of NVMM performance with DRAM. PMEP supports configurable latencies and bandwidth for the emulated NVMM, allowing us to explore NOVA's performance on a variety of future memory technologies. PMEP emulates `clflushopt` (efficient cache line flush), `clwb` (cache line write back), and `PCOMMIT` (persistent commit) instructions with processor microcode.

In our tests we configure the PMEP with 32 GB of DRAM and 64 GB of NVMM. We choose two configurations for PMEP to emulate different NVMM technologies: for STT-RAM we use the same read latency and bandwidth as DRAM, and configure `PCOMMIT` to take 200 ns (to match projections for STT-RAM write times); for PCM we use 300 ns for the read latency and reduce the write bandwidth to 1/8 of DRAM while increasing `PCOMMIT` time to 500 ns. `clwb` takes 40 ns in both configurations.

We compare NOVA to seven other file systems: Two NVMM file systems, PMFS and ext4-DAX, are journaling file systems. Two others, NILFS2 and F2FS, are log-structured file systems. We also compare to ext4 in default mode (ext4) and in data journal mode (ext4-data), which provides data atomicity. Finally, we compare to btrfs, a copy-on-write Linux file system. PMFS, ext4-DAX, and NOVA are *Direct Access (DAX)* file systems that bypass the operating system page cache and access NVMM directly. Btrfs and ext4-data are the only two file systems in the group that provide the same strong consistency guarantees as NOVA. Ext4-DAX does not currently provide a data journaling option. We add `clwb` and `PCOMMIT` instructions to flush data where necessary in each file system, and use Intel persistent memory driver [6] to emulate an NVMM-based RAMDisk-like device.

Macrobenchmarks

We select four Filebench [5] workloads to evaluate the application-level performance of NOVA (Table 1). For each workload we test two data set sizes by changing the number of files. The *small* data set will fit entirely in DRAM, allowing file systems that use the DRAM page cache to cache the entire data set. The *large* data set is too large to fit in DRAM, so the page cache is less useful.

Figure 5 shows the Filebench throughput. In the fileserver workload, NOVA outperforms other file systems by 1.8x–16.6x on STT-RAM, and between 22% and 9.1x on PCM for the largest data set. NOVA outperforms ext4-data by 11.4x and btrfs by 13.5x on STT-RAM, while providing equivalent consistency guarantees. NOVA on STT-RAM delivers twice the throughput compared to PCM, because of PCM's lower write bandwidth.

Web proxy is a read-intensive workload, and it puts all the test files in one large directory. For the small data set, NOVA performs similarly to ext4 and ext4-DAX, and 2.1x faster than ext4-data. For the large workload, NOVA performs 36%–53% better than F2FS and ext4-DAX. PMFS and NILFS2 perform poorly in this test because their directory designs are not scalable.

Web server is a read-dominated workload and does not involve any directory operations. As a result, non-DAX file systems benefit significantly from the DRAM page cache, and the workload

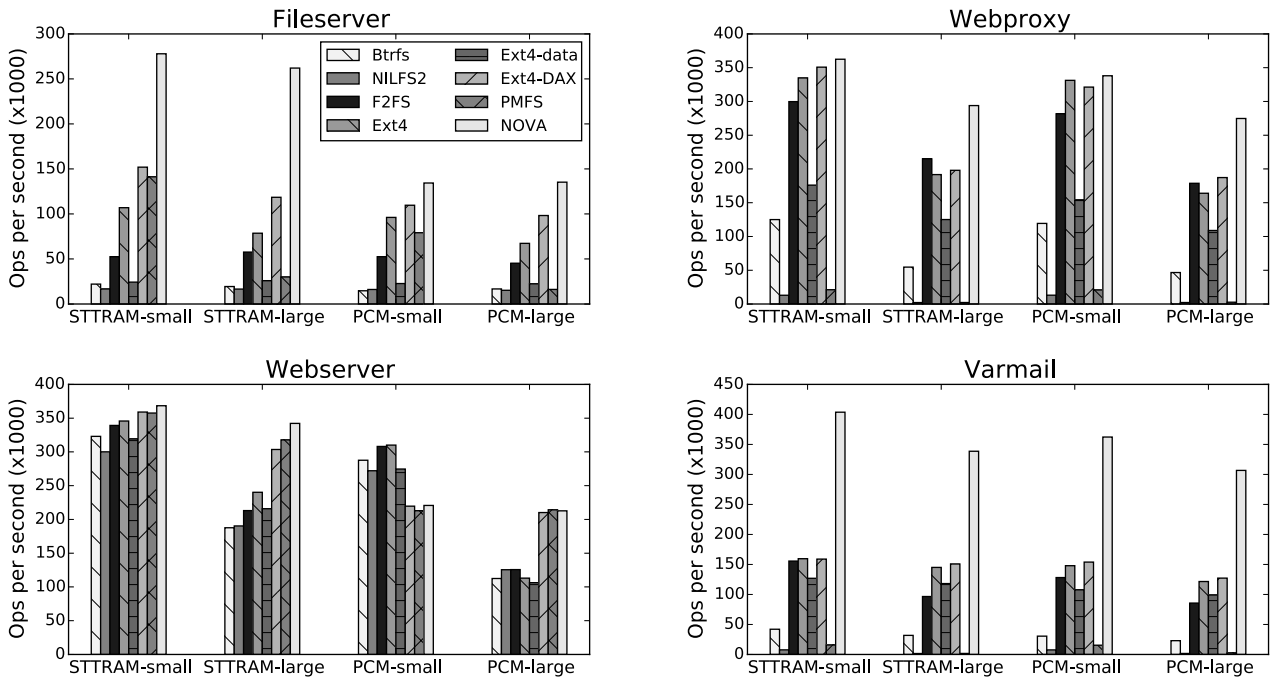

Figure 5: **Filebench throughput with different file system patterns and dataset sizes on STT-RAM and PCM.** Each workload has two data-set sizes so that the small one can fit in DRAM entirely while the large one cannot.

size has a large impact on performance. On STT-RAM, with the large data set, NOVA performs 63% better on average than non-DAX file systems. On PCM, for the small data set, non-DAX file systems are 33% faster on average due to DRAM caching. However, for the large data set, NOVA's performance remains stable while non-DAX performance drops by 60%.

Varmail emulates an email server with a large number of small files and involves both `read` and `write` operations. NOVA outperforms btrfs by 11.1x and ext4-data by 3.1x on average, and outperforms the other file systems by 2.2x–216x. NILFS2 and PMFS still suffer from poor directory operation performance.

Overall, NOVA achieves the best performance in almost all cases and provides data consistency guarantees that are as strong or stronger than the other file systems. The performance advantages of NOVA are largest on write-intensive workloads with large number of files.

Garbage Collection Efficiency

We designed NOVA to perform garbage collection efficiently and maintain stable performance under heavy write loads, even when the file system is nearly full. To verify these characteristics, we ran a 30 GB write-intensive fileserver workload and adjusted the amount of NVMM available to bring utilization to 95%. Then we compared NOVA's behavior with the other log-structured file systems, NILFS2 and F2FS. We ran the test with PMEP configured to emulate STT-RAM.

Figure 6 shows the result. NILFS2 failed after less than 10 seconds since it ran out of space due to garbage collection inefficiencies. F2FS failed after running for 158 seconds after suffering a 60% drop in throughput due to log cleaning overhead. NOVA outperformed F2FS by 12x, and the throughput remained stable over time.

We found that the longer NOVA runs, the more efficient fast GC becomes, eventually accounting for the majority of reclaimed pages.

Recovery Overhead

NOVA provides fast recovery from both normal dismounts and power failures. To measure the recovery overhead, we used the three workloads in Table 2. Each workload represents a different use case for the file systems: Videoserver contains a few large files accessed with large-size requests; mailserver includes a large number of small files and the request size is small; fileserver is in-between. For each workload, we measure the cost of mounting after a normal shutdown and after a power failure.

Table 3 summarizes the results. With a normal shutdown, NOVA recovers the file system in 1.2 ms, since NOVA can restore the allocator state from the a checkpoint. After a power failure, NOVA recovery time increases with the number of inodes and as the I/O operations that created the files become smaller (since small I/O operations result in more log entries). Recovery runs faster on STT-RAM than on PCM because PCM has higher read latency. On both PCM and STT-RAM, NOVA is able to recover 50

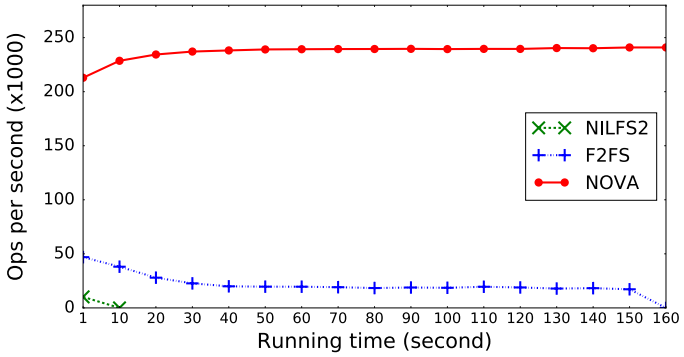

Figure 6: Performance of a full file system. The test runs a 30 GB file-server workload under 95% NVMM utilization with different durations.

GB data in 116 ms, achieving failure recovery bandwidth higher than 400 GB/s.

Conclusion

We have implemented and described NOVA, a log-structured file system designed for hybrid volatile/non-volatile main memories. NOVA extends ideas of LFS to leverage NVMM, yielding a simpler, high-performance file system that supports fast and efficient garbage collection and quick recovery from system failures. Our measurements show that NOVA outperforms existing NVMM file systems by a wide margin on a wide range of applications while providing stronger consistency and atomicity guarantees.

Data set	File size	Number of files	Data-set size	I/O size
Videoserver	128 MB	400	50 GB	1 MB
Fileserver	1 MB	50,000	50 GB	64 KB
Mailserver	128 KB	400,000	50 GB	16 KB

Table 2: Recovery workload characteristics. The number of files and typical I/O size both affect NOVA's recovery performance.

Data set	Videoserver	Fileserver	Mailserver
STTRAM-normal	156 µs	313 µs	918 µs
PCM-normal	311 µs	660 µs	1197 µs
STTRAM-failure	37 ms	39 ms	72 ms
PCM-failure	43 ms	50 ms	116 ms

Table 3: NOVA recovery time on different scenarios. NOVA is able to recover 50 GB data in 116 ms in case of power failure.

Acknowledgments

This work was supported by STARnet, a Semiconductor Research Corporation program, sponsored by MARCO and DARPA. We would like to thank John Ousterhout, Niraj Tolia, Isabella Furth, Rik Farrow, and the anonymous FAST reviewers for their insightful comments and suggestions. We are also thankful to Subramanya R. Dulloor from Intel for his support and hardware access.

References

[1] 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.

[2] A. Akel, A. M. Caulfield, T. I. Mollov, R. K. Gupta, and S. Swanson, "Onyx: A Prototype Phase Change Memory Storage Array," in *Proceedings of the 3rd USENIX Conference on Hot Topics in Storage and File Systems (HotStorage '11*).

[3] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better I/O through Byte-Addressable, Persistent Memory," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09)*, pp. 133–146.

[4] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, "System Software for Persistent Memory," in *Proceedings of the 9th European Conference on Computer Systems (EuroSys '14)*, ACM, pp. 15:1–15:15.

[5] Filebench file system benchmark: http://sourceforge.net/projects/filebench.

[6] PMEM: the persistent memory driver + ext4 direct access (DAX): https://github.com/01org/prd.

[7] Linux POSIX file system test suite: https://lwn.net/Articles/276617/.

[8] S. Raoux, G. Burr, M. Breitwisch, C. Rettner, Y. Chen, R. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H. L. Lung, and C. Lam, "Phase-Change Random Access Memory: A Scalable Technology," *IBM Journal of Research and Development,* vol. 52, no. 4.5, July 2008, pp. 465–479.

[9] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Transactions on Computer Systems (TOCS)*, vol. 10, no. 1, 1992, pp. 26–52.

[10] M. Wilcox, "Add Support for NV-DIMMs to ext4: https://lwn.net/Articles/613384/.

[11] J. Xu and S. Swanson, "NOVA: A Log-Structured File System for Hybrid Volatile/Non-Volatile Main Memories," in *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST '16)*, pp. 323–338.

[12] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," in *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*, ACM, pp. 14–23.