On the Scalability of Testing the Crash Consistency of PM Systems

Duo Zhang†, Om Rameshwar Gatla†, Abdullah Al Raqibul Islam‡, Dong Dai‡, Mai Zheng†
†Iowa State University ‡University of North Carolina at Charlotte

1 Motivation

Persistent memory (PM) technologies [4, 8] can provide durability with latencies comparable to DRAM. Such characteristics bridge the gap between traditional memory and storage, and have inspired many PM-based optimizations in both user-level applications [5, 19] and operating system (OS) kernels [17, 21]. While Intel is winding down its Optane PM business, it is expected that vendor-neutral CXL-based PMs will continuously evolve and trigger new system optimizations [1, 6, 3].

Unfortunately, building correct PM-based systems is challenging: writes to PM need to be carefully ordered and persisted to avoid inconsistent or unrecoverable states upon crashes, which is non-trivial given the subtle behavior of modern cache and memory subsystem [14, 15, 7, 16, 10, 20]. To address the challenge, many testing tools have been proposed [12, 11, 13]. Nevertheless, most of them focus on user-level PM applications, leaving the OS kernel and the potential dependencies unexplored. In fact, one recent study shows that there are various PM-related issues at the kernel level [18], which calls for tool support for full systems.

Notably, Kalbfleisch et al. proposed the VINTER [11] framework recently which can support full-system testing and have been applied to test multiple kernel-level PM file systems (e.g., NOVA [17], PMFS [2]). While promising, our experiments on VINTER exposed a scalability challenge in (at least) three aspects as follows:

First, writes to the target PM file system can easily overwhelm VINTER. As shown in Table 1, we apply three small workloads on NOVA which generates three different sizes of writes (i.e., 256, 256*4, and 256*20 bytes). It took VINTER about 1 hour 45 minutes (‘1h 45m’) to complete the test under the 256*4 workload; and with 256*20 bytes of write, VINTER cannot finish within 12 hours. Further analysis shows that among the three core phases of VINTER (i.e., *record*, *replay*, *test*), both the *replay* and *test* phases are the bottlenecks.

Second, VINTER cannot scale to typical PM application scenarios which require important libraries (e.g., PMDK) as it only support minimal-built OS. Moreover, we found that the size of the emulated PM device in VINTER is too small to be used as the memory pool for PMDK. Since VINTER relies on the PM device size internally (i.e., the address range is used for tracing), extending VINTER to support large PM devices (and PMDK) would require substantial efforts.

Writes (Bytes)	Total Runtime	Runtime for Each Phase		
		Record	Replay	Test
256	6m 47s	8s	5m	1m 39s
256*4	1h 45m	16s	55m 42s	48m 30s
256*20	-	21s	>12h	-

Table 1: **Scalability Issue Observed.**

PM Software Stack	VINTER	Ours
Application (e.g., B-tree)	×	✓
Library (e.g., PMDK)	×	✓
File System (e.g., NOVA)	✓	✓
Driver (e.g., NVDIMM)	✓	✓

Table 2: **Comparison of PM Layers Supported.**

Last but not the least, VINTER has little diagnosis support to help understand complicated crash issues.

2 Our Approach

We are exploring a new method based on PANDA [9] to support scalable testing of realistic full PM system stack. As shown in Table 2, the prototype can support full PM software stack (e.g., PMDK-based B-tree on PM kernel modules). We are able to record all PM instructions to generate crash images under the full system stack. Moreover, we record snapshots and non-determinism logs to facilitate understanding full-system behavior. In addition, we are investigating a parallelism mechanism to generate crash states at different barrier points in parallel, which is expected to improve the scalability effectively.

References

[1] Compute express link. https://www.computeexpresslink.org/.

[2] Pmfs introduction. https://github.com/linux-pmfs/pmfs.

[3] Update on pmdk and our long term support strategy. https://pmem.io/blog/2022/11/update-on-pmdk-and-our-long-term-support-strategy/.

[4] What is persistent memory? https://www.snia.org/education/what-is-persistent-memory.

[5] Pmem-redis. https://github.com/pmem/pmem-redis, 2021.

[6] Ankit Bhardwaj, Todd Thornley, Vinita Pawar, Reto Achermann, Gerd Zellweger, and Ryan Stutsman. Cache-coherent accelerators for persistent memory crash consistency. In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*, 2022.

[7] Vijay Chidambaram, Thanumalayan Sankaranarayana Pillai, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Optimistic Crash Consistency. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP '13)*, Farmington, PA, November 2013.

[8] Jeremy Condit, Edmund B. Nightingale, Christopher Frost, Engin Ipek, Benjamin Lee, Doug Burger, and Derrick Coetzee. Better i/o through byte-addressable, persistent memory. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, 2009.

[9] Brendan Dolan-Gavitt, Josh Hodosh, Patrick Hulin, Tim Leek, and Ryan Whelan. Repeatable reverse engineering with panda. In *Proceedings of the 5th Program Protection and Reverse Engineering Workshop*, 2015.

[10] Om Rameshwar Gatla, Muhammad Hameed, Mai Zheng, Viacheslav Dubeyko, Adam Manzanares, Filip Blagojević, Cyril Guyot, and Robert Mateescu. Towards robust file system checkers. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST)*, 2018.

[11] Samuel Kalbfleisch, Lukas Werling, and Frank Bellosa. Vinter: Automatic non-volatile memory crash consistency testing for full systems. In *Proceedings of the 2022 USENIX Annual Technical Conference*, 2022.

[12] Hayley LeBlanc, Shankara Pailoor, Om Saran K R E, Isil Dillig, James Bornholt, and Vijay Chidambaram. Chipmunk: Investigating crash-consistency in persistent-memory file systems. In *Proceedings of the Eightteenth European Conference on Computer Systems*, 2023.

[13] Sihang Liu, Yizhou Wei, Jishen Zhao, Aasheesh Kolli, and Samira Khan. PMTest: A Fast and Flexible Testing Framework for Persistent Memory Programs. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[14] Ashlie Martinez and Vijay Chidambaram. CrashMonkey: A Framework to Systematically Test File-System Crash Consistency. In *Proceedings of the 9th USENIX Conference on Hot Topics in Storage and File Systems (HotStorage)*, 2017.

[15] Jayashree Mohan, Ashlie Martinez, Soujanya Ponnapalli, Pandian Raju, and Vijay Chidambaram. Finding crash-consistency bugs with bounded black-box crash testing. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 33–50, 2018.

[16] Helgi Sigurbjarnarson, James Bornholt, Emina Torlak, and Xi Wang. Push-button verification of file systems via crash refinement. In *Proceedings of 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*, 2016.

[17] Jian Xu and Steven Swanson. Nova: A log-structured file system for hybrid volatile/non-volatile main memories. In *Proceedings of 14th USENIX Conference on File and Storage Technologies*, 2016.

[18] Duo Zhang, Om Rameshwar Gatla, Wei Xu, and Mai Zheng. A study of persistent memory bugs in the linux kernel. In *Proceedings of the 14th ACM International Conference on Systems and Storage*, 2021.

[19] Jishen Zhao, Onur Mutlu, and Yuan Xie. Firm: Fair and high-performance memory control for persistent memory systems. In *47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

[20] Mai Zheng, Joseph Tucek, Dachuan Huang, Feng Qin, Mark Lillibridge, Elizabeth S. Yang, Bill W Zhao, and Shashank Singh. Torturing Databases for Fun and Profit. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 449–464, 2014.

[21] Bohong Zhu, Youmin Chen, Qing Wang, Youyou Lu, and Jiwu Shu. Octopus+: An rdma-enabled distributed persistent memory file system. In *ACM Transactions on Storage*, 2021.