A brief primer on Persistent Memory Objects [4]

Derrick Greenspan, Naveed Ul Mustafa, Zoran Kolega, Mark Heinrich, Yan Solihin
University of Central Florida
{derrick.greenspan, unknown.naveedulmustafa, heinrich, yan.solihin}@ucf.edu
kolegazoran@knights.ucf.edu

I. Motivation

(*Note: please click here for the original paper.*)

DIMM-Compatible Persistent Memory (PM), such as Intel Optane PMem combines byte-addressability with non-volatility, providing an opportunity to host byte-addressable data persistently, and may augment or replace DRAM as main memory due to its higher density and lower cost per byte. PM may be managed as *memory-mapped files* [2], [5], [10], or as a collection of *persistent memory objects* (PMOs). The memory-mapped file approach limits the use of system calls, but keeps data as an array of bytes and requires keeping two systems (filesystem and virtual memory) and their distinct metadata and semantics consistent for the same underlying data. The PMO approach organizes PM as a collection of persistent memory objects (PMOs) holding pointer-rich data structures without the backing of a filesystem, which is a more intuitive design.

This paper presents and evaluates a crash-consistent and secure design of PMO abstraction. Crash consistency is the property that allows data to be recovered to a consistent state after a crash, and is an important requirement for storing persistent data structures, otherwise, in the event of a failure, data could be in an unrecoverable state. Prior work in PMOs [1], [6], [11], [12] do not provide crash consistency as an intrinsic feature of the abstraction design. Instead, they put burden on the programmer to use the low-level primitives of flushing and fencing in the proper order, any mistake can be difficult to debug. An alternative design for crash-consistency is to use transactional memory, relying on the tight coupling of concurrency management and crash consistency. This forces crash consistency management to use the same small code granularity preferred by a transaction. This conflicts with the preferred crash consistency granularity, which is large since crashes are much rarer events than thread conflicts.

Since data structures often contain pointers, persistent data structures present an enticing target for security attacks. A pointer corrupted by the attacker in one run becomes persistent, effecting future runs of the same, or even different, applications [9]. A PMO may be exploited for security attack while it is *In-use* (attached to the address space of a user process) or *At-rest* (not attached to the address space of any process). While previous work [11], [12] propose protections for *in-use* PMOs, there have been no defense proposed for *at-rest* PMOs.

II. Threat Moodel

We consider a threat model where PMO is *at-rest*. The goal of attacker is to disclose or overwrite private data of a user process held in an *at-rest* PMO. We do not trust system software except for a subset, specifically the Linux Kernel Crypto API and PMO subsystem. We also assume that the attacker knows the location of the PMO in persistent memory.

III. Design

Our PMO design addresses the crash-consistency and security challenges of PMOs, and provides three fundamental system calls: **attach**, **detach**, and **psync**. Attach maps a specified PMO into the address space of the calling process. Detach renders the PMO inaccessible to the calling process.

a) Crash-consistency: We address crash-consistency by introducing *psync*, a system call that decouples *when* data in a PMO reaches a crash-consistent state from *how* it happens. With psync, the programmer specifies points in the code where data are to be rendered crash consistent. The system then ensures that all stores prior to the psync are crash consistent prior to any subsequent one. The **key idea** behind psync is that we utilize a shadow PMO for updates, and introduce an invariant that at least one of either the primary or the shadow PMO are valid at all times that can be used for crash recovery. When psync is invoked on a PMO, it copies all the pages with modified data from the shadow into the primary copy.

b) Security: We protect against attacks on at-rest PMOs by extending our PMO design to support integrity verification and encryption. Integrity verification is performed by computing the checksum of the PMO at detach time, and storing it within the PMO metadata. A future attach on the same PMO causes the checksum to be computed again, and compared with the stored checksum. If it has not been modified, the attach succeeds, otherwise it fails. Encryption is performed by invoking the Kernel Crypto API to decrypt the PMO when in use, and immediately encrypt it at rest. To ensure that the design is still crash consistent, we repurpose the shadowing approach so that there is always either a valid encrypted or valid decrypted copy of the PMO.

IV. Evaluation Methodology

We compare our PMO system implementation to a baseline scheme with no crash consistency (NCC) and a state-of-the-art crash consistent filesystem design (NOVA-Fortis) [10]. For our evaluation, we use a real system with Intel Optane PMem. We use several Microbenchmarks: an OpenMP version of LU decomposition [7], a 2D-Convolution (2DConv) benchmark, and a Tiled Matrix Multiplication (TMM) benchmark taken from a PM study [3]. We run LU, 2DConv, and TMM with matrix sizes of 3584×3584 doubles, 4096×128 integers, and 3072×3072 integers, respectively. We ported these benchmarks by replacing their dynamic memory allocation calls with a pair of `pcreate` (that creates a PMO of given size) and `attach`, and with `pmem_map_file` for NOVA-Fortis. At the end of each iteration of the performance critical loop in the benchmarks, we insert a synchronization point, if a specified amount of time has elapsed. We also rely on FileBench benchmarks [8], which represent I/O intensive real-world applications. When evaluating PMOs, we replace files with PMOs of respective sizes, and synchronize after every update. We use four different workloads, each has a different percentage of write operations: FileServer (FS) is 67% writes, VarMail (VM) is 50%, WebProxy (WP) is 16%, and WebServer (WS) is 9%. We also evaluate the impact of encryption on Filebench by adding new `attach`/`detach` calls between each I/O operation.

V. Results

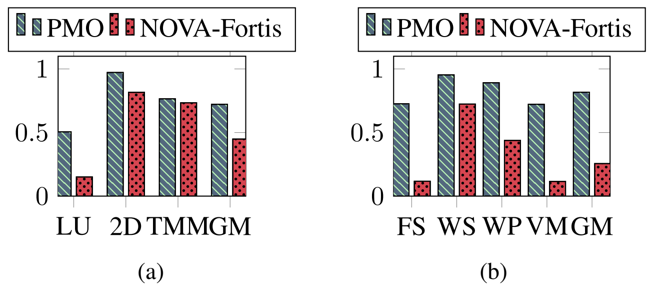

Fig. 1: (a) Microbenchmark performance. (b) Filebench bandwidth.

We find that for the microbenchmarks (Figure 1a), PMOs are only $\approx$ 27.8% slower than a system with no crash consistency, and is $\approx 1.61\times$ faster than NOVA-Fortis. For Filebench (Figure 1b), PMOs and Nova-Fortis provide crash consistency at the expense of losing 18.3% and 74.4% bandwidth, respectively. This means that our PMO system achieves bandwidth $\approx 3.2\times$ higher than NOVA-Fortis for this set of benchmarks.

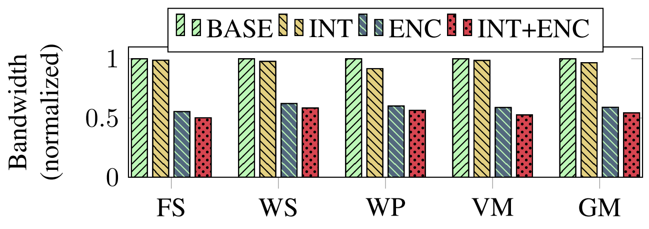

Fig. 2: Bandwidth comparison of attach/detach PMO, with different modes: baseline (BASE), Integrity (INT), Encryption (ENC), and both (ENC+INT).

We find that integrity and encryption together (Figure 2) lower the bandwidth by 46%, while integrity checking alone incurs a small overhead of only 3%.

Finally, we design an experiment to verify that our approach prevents unauthorized disclosure of data and protects against integrity violations. We create 1000 PMOs and write a secret into a random selection of them; an attacker selects multiple PMOs at random and attempts to read the data within the PMO, or attempts to modify it. We find that attempts to read the data within the PMO are 100% successful without using encryption, but completely unsuccessful when encryption is used. Similarly, with integrity verification, the kernel detects data corruption in 100% of cases.

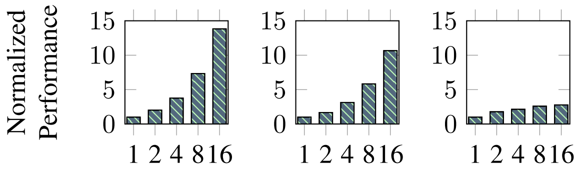

Fig. 3: Thread-scalability of the PMO system. From left to right: 2DConv, TMM, and LU.

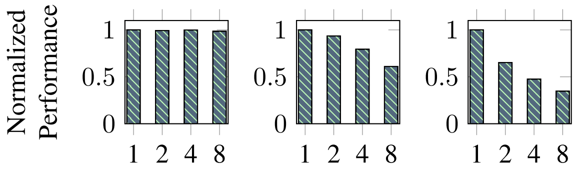

Fig. 4: Synchronization-sensitivity of the PMO system. From left to right: 2DConv, TMM, and LU.

Acknowledgements

This work is supported in part by the ONR under grant N00014-20-1-2750, and by the NSF under grant 1900724.

References

[1] Daniel Bittman, Peter Alvaro, Pankaj Mehra, Darrell DE Long, and Ethan L Miller. Twizzler: a data-centric {OS} for non-volatile memory. In *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, pages 65–80, 2020.

[2] Jungsik Choi, Jaewan Hong, Youngjin Kwon, and Hwansoo Han. Libnvmmio reconstructing software {IO} path with failure-atomic memory-mapped interface. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 1–16, 2020.

[3] H. Elnawawy, M. Alshboul, J. Tuck, and Y. Solihin. Efficient checkpointing of loop-based codes for non-volatile main memory. In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 318–329, 2017.

[4] Derrick Greenspan, Naveed Ul Mustafa, Zoran Kolega, Mark Heinrich, and Yan Solihin. Improving the security and programmability of persistent memory objects. In *IEEE International Symposium on Secure and Private Execution Environment Design (SEED)*, Virtual, 2022. IEEE.

[5] Rohan Kadekodi, Se Kwon Lee, Sanidhya Kashyap, Taesoo Kim, Aasheesh Kolli, and Vijay Chidambaram. Splitfs: Reducing software overhead in file systems for persistent memory. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 494–508, 2019.

[6] Awais Khan, Hyogi Sim, Sudharshan S Vazhkudai, Jinsuk Ma, Myeong-Hoon Oh, and Youngjae Kim. Persistent memory object storage and indexing for scientific computing. In *2020 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*, pages 1–9. IEEE, 2020.

[7] Christian Klauser. Lu decomposition and matrix multiplication with openmp, 2011.

[8] Vasily Tarasov, Erez Zadok, and Spencer Shepler. Filebench: A flexible framework for file system benchmarking. *USENIX; login*, 41(1):6–12, 2016.

[9] Naveed Ul Mustafa, Yuanchao Xu, Xipeng Shen, and Yan Solihin. Seeds of seed: New security challenges of persistent memory. In *IEEE International Symposium on Secure and Private Execution Environment Design (SEED)*, Virtual, 2021. IEEE.

[10] Jian Xu, Lu Zhang, Amirsaman Memaripour, Akshatha Gangadharaiah, Amit Borase, Tamires Brito Da Silva, Steven Swanson, and Andy Rudoff. Nova-fortis: A fault-tolerant non-volatile main memory file system. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 478–496, 2017.

[11] Yuanchao Xu, Yan Solihin, and Xipeng Shen. Merr: Improving security of persistent memory objects via efficient memory exposure reduction and randomization. New York, NY, USA, 2020. Association for Computing Machinery.

[12] Yuanchao Xu, Chencheng Ye, Xipeng Shen, and Yan Solihin. Temporal exposure reduction protection for persistent memory. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 908–924. IEEE, 2022.