On the Recoverability of Persistent Memory Systems

Ryan Chartier[†], *Om Rameshwar Gatla*[*], *Mai Zheng*[†*], *Henry Duwe*[*]
[†]*New Mexico State University,* [*]*Iowa State University*

1 Motivation

Persistent memory (PM) technologies [2, 6, 10] are expected to provide durability similar to the flash memory with latencies comparable to DRAM. These unique characteristics bring new challenges for system design. Among others, PM-based systems have to carefully order and persist writes to the memory so that the system states are always recoverable upon failures. This is non-trivial due to the subtle behavior of modern cache and memory subsystem [7, 8].

To make PM easier to use, great efforts have been made to optimize different layers/components of systems (e.g., file systems [5, 11, 12], libraries [4, 9], databases [3]). These PM-aware systems generally include sophisticated designs to achieve high performance while maintaining high recoverability. Nevertheless, the evaluation of these systems is unbalanced: while various benchmarks have been used to demonstrate the performance gain (including the performance of recovery), there is little measurement of the recoverability guarantee, largely because of a lack of effective methodology.

2 Methodology

We believe the evaluation of PM systems should be comprehensive, i.e., not only measuring the performance but also testing the recoverability. To this end, we design a fault injection framework to systematically testing the recoverability of PM systems, which includes four steps: **(1) record:** we instrument the target system to captures memory operations as well as instructions important for recovery (e.g., `clflush`); **(2) analysis:** traces obtained from the record phase are used to identify the most vulnerable points for fault injection (e.g., between a double `clflush`); **(3) replay:** based on the trace recorded, we replay the target system up to a specific execution point, which essentially emulates the PM state of the system as interrupted by a failure event; **(4) recovery:** we invoke the recovery component of the target system and examine the recoverability. In addition, we define two types faults based on the granularity:

- *Macro Faults:* faults that occur at the boundaries of PM library functions (e.g., after a `pmalloc` finishes); this type of coarse-grained faults test the application-level recovery protocol (i.e., assuming the PM library functions are atomic)
- *Micro Faults:* faults that occur *during* PM library functions (e.g., inside `pmalloc`); this type of fine-grained faults test the PM management thoroughly.

3 Preliminary Results

Fault Type	Description	R?
Macro-1	after a transaction commit	Y
Macro-2	within a transaction	Y
Micro-1	in `pmalloc` (b/w two `clflush`)	Y
Micro-2	in `pmalloc` (before any `clflush`)	**N**

Table 1: **Recoverability of N-Store under 4 faults**. *The last column shows whether N-Store recovered sucessfully (Y) or not (N).*

We evaluate our preliminary prototype using N-Store [3], which is a PM-aware database with a PM library. We use PIN [1] to record memory instructions with timestamps, and identify the boundaries of functions such as `pmalloc`. We save N-Store's memory-mapped PM file at each dynamic fault point and attempt to restore from each stored PM file.

As shown in Table 1, we inject 2 macro faults and 2 micro faults. N-Store can successfully recover from Macro-1, Macro-2 and Micro-1. However, we observe a segmentation fault when recovering N-Store from Micro-2, which may imply that the PM library is unable to restore the PM to a clean state. We are investigating the root cause and studying whether other PM systems suffer from similar issues.

References

[1] PIN — a dynamic binary instrumentation tool. https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool/.

[2] Hiroyuki Akinaga and Hisashi Shima. Resistive Random Access Memory (ReRAM) Based on Metal Oxides. *Proceedings of the IEEE*, 98(12):2237–2251, 2010.

[3] Joy Arulraj, Andrew Pavlo, and Subramanya R Dulloor. Let's Talk about Storage & Recovery Methods for Non-Volatile Memory Database Systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 707–722. ACM, 2015.

[4] Joel Coburn, Adrian M. Caulfield, Ameen Akel, Laura M. Grupp, Rajesh K. Gupta, Ranjit Jhala, and Steven Swanson. NV-Heaps: Making Persistent Objects Fast and Safe with Next-Generation, Non-Volatile Memories. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11)*. ACM, 2011.

[5] Subramanya R Dulloor, Sanjay Kumar, Anil Keshavamurthy, Philip Lantz, Dheeraj Reddy, Rajesh Sankaran, and Jeff Jackson. System Software for Persistent Memory. In *Proceedings of the 9th European Conference on Computer Systems*, page 15. ACM, 2014.

[6] M Hosomi, H Yamagishi, T Yamamoto, K Bessho, Y Higo, K Yamane, H Yamada, M Shoji, H Hachino, C Fukumoto, et al. A Novel Non Volatile Memory with Spin Torque Transfer Magnetization Switching: Spin-RAM. In *Proceedings of IEEE International Electron Devices Meeting (IEDM)*, pages 459–462. IEEE, 2005.

[7] Sanketh Nalli, Swapnil Haria, Mark D Hill, Michael M Swift, Haris Volos, and Kimberly Keeton. An analysis of persistent memory use with whisper. In *ACM SIGARCH Computer Architecture News*, volume 45, pages 135–148. ACM, 2017.

[8] Andy Rudoff. Persistent Memory Programming. *;login: The USENIX Magazine*, 42(2):34?40, May 2017.

[9] Haris Volos, Andres Jaan Tack, and Michael M Swift. Mnemosyne: Lightweight persistent memory. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11)*. ACM, 2011.

[10] H-S Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E Goodson. Phase Change Memory. *Proceedings of the IEEE*, 98(12):2201–2227, 2010.

[11] Jian Xu and Steven Swanson. NOVA: A Log-Structured File System for Hybrid Volatile/Non-volatile Main Memories. In *Proceedings of the 14th USENIX Conference of File and Storage Technologies (FAST,16)*, 2016.

[12] 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. ACM, 2017.

On the Recoverability of Persistent Memory Systems

Ryan Chartier[1], Om Rameshwar Gatla[1,2], Mai Zheng[1,2] and Henry Duwe[2]

New Mexico State University[1] *Iowa State University*[2]

FAST'19
17th USENIX Conference on File and Storage Technologies
FEBRUARY 25–28, 2019
BOSTON, MA, USA
www.usenix.org/fast19

Motivation

- Persistent Memory (PM) technologies offers durability similar to flash memory with latencies comparable to DRAM

	DRAM	PCM	RRAM	MRAM	SSD	HDD
Read Latency	60 ns	50 ns	100 ns	20 ns	25 µs	10 ms
Write Latency	60 ns	150 ns	100 ns	20 ns	300 µs	10 ms
Addressability	Byte	Byte	Byte	Byte	Block	Block
Volatile	Yes	No	No	No	No	No
Energy/bit access	2 pJ	2 pJ	100 pJ	0.02 pJ	10 nJ	0.1 J
Endurance	$>10^{16}$	10^{10}	10^{8}	10^{15}	10^{5}	$>10^{16}$

Table 1: Characteristics of memory/storage technologies[1]

- Various PM systems have been proposed , e.g.:
  - NV-Heaps, Mnemosyne, DudeTM, NOVA, …
  - designed for achieving high performance while maintaining consistency and high recoverability
  - but the evaluation is unbalanced

Various benchmarks have been used to demonstrate the performance gain, but there is ***little measurement of recoverability guarantee***, largely because of a lack of effective methodology

Methodology

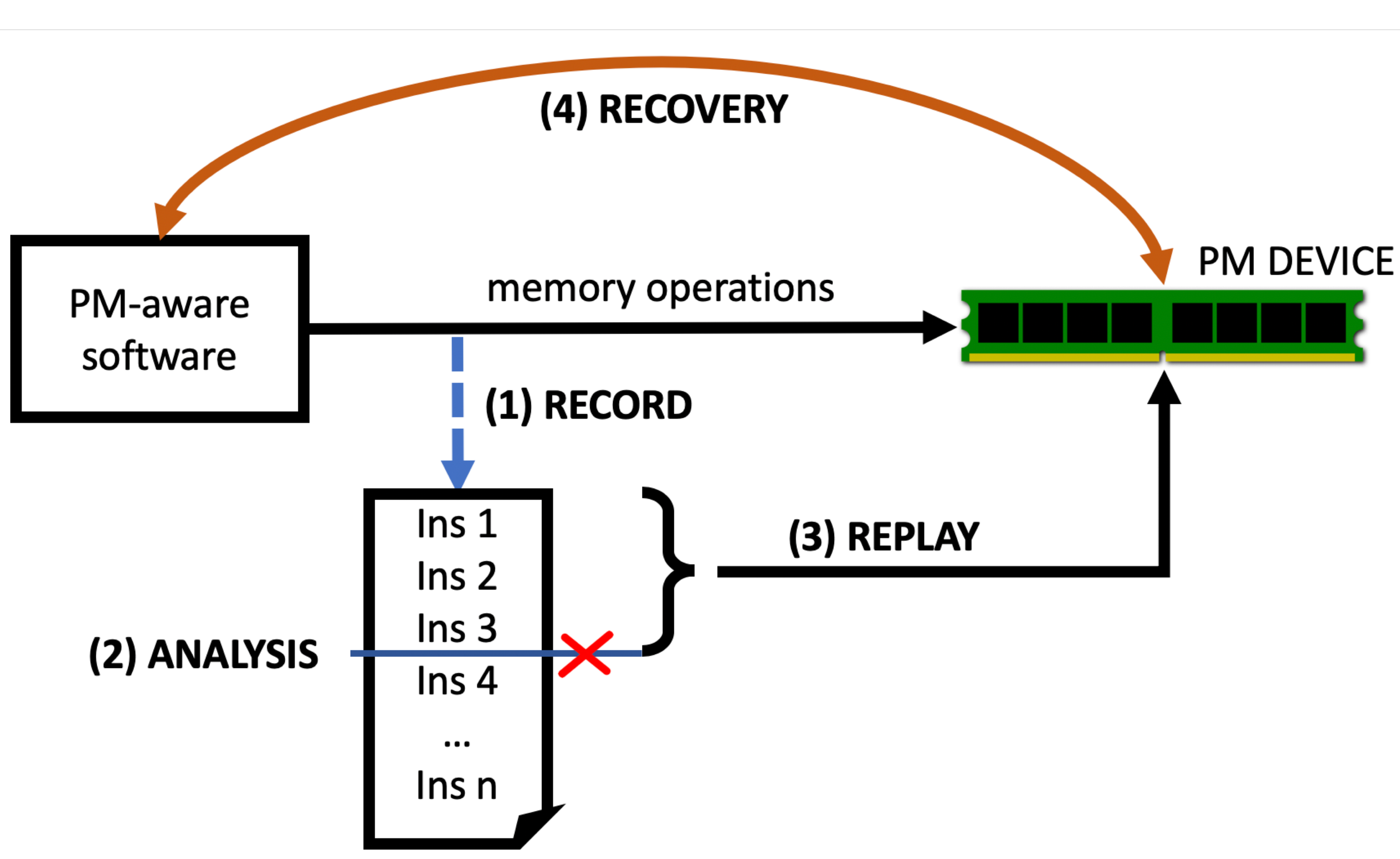

Figure 1: Overview of framework

- A fault injection framework to systematically test recoverability
- Four steps:
  - ***Record:*** instrument target system to capture memory operations, as well as recovery instructions (*clflush*)
  - ***Analysis:*** identify most vulnerable points for fault injection
  - ***Replay:*** replay target system up to a specific execution point
  - ***Recovery:*** invoke recovery component of target system and examine recoverability

- Two types of faults based on granularity: Macro and Micro

Macro (M)	Micro (m)
Faults that occur at the boundaries of PM lib functions	Faults that occur during PM lib functions
Ex.: after *pmalloc* finishes	Ex.: inside *pmalloc*
Coarse-grained fault to test application-level recovery protocol	Fine-grained fault to test PM management thoroughly

Table 2: Two types of faults

Experimental Results

- Used **N-Store** as one case study
- Used Intel's PIN to record memory instructions and identify boundaries of functions (ex.: *pmalloc*)
- Save memory-mapped file at each dynamic fault point and attempt to restore
- Injected 4 faults: Macro-1, Macro-2, Micro-1 and Micro-2
- N-Store successfully recovers from Macro-1, Macro-2 & Micro-1
- ***Segmentation fault*** occurs when recovering from Micro-2
  - PM library was unable to restore PM to clean state

Fault Type	Description	Level
Macro-1	after a transaction commit	Application
Macro-2	within a transaction	Application
Micro-1	in *pmalloc* (b/w two *clflush*)	Library
Micro-2	in *pmalloc* (before any *clflush*)	Library

Table 3: Description for each Macro and Micro faults

Fault Type	Target Component	Recovery Successful?
Macro-1	Undo log of N-Store	Yes
Macro-2	Undo log of N-Store	Yes
Micro-1	*pmemlib*	Yes
Micro-2	*pmemlib*	No

Table 4: Experimental results from fault injection

Future Work

- Investigate the root cause of the segmentation fault
- Automate the framework
- Reduce the overhead of memory tracing
- Analyze more PM systems

Acknowledgements

- This work was supported in part by NSF under grants CNS-1566554/1855565 & CCF-1717630/1853714, and a gift from Western Digital & IDEMA. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] Joy Arulraj, Andrew Pavlo, and Subramanya R. Dulloor. 2015. Let's Talk About Storage & Recovery Methods for Non-Volatile Memory Database Systems. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD 'p15). ACM, New York, NY, USA, 707-722. DOI: https://doi.org/10.1145/2723372.2749441