# On the Recoverability of Persistent Memory Systems

Ryan Chartier[1], Om Rameshwar Gatla[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 various memory/storage technologies

- Recent research have optimized different layers/components of PM-aware applications
  - NV-Heaps, Mnemosyne, DudeTM, NOVA, etc.
- PM-aware applications achieve high performance while maintaining high recoverability
- But the evaluation of these applications is unbalanced

Various benchmarks demonstrate performance gain but provide ***little measurement*** of recoverability guarantee

## Methodology

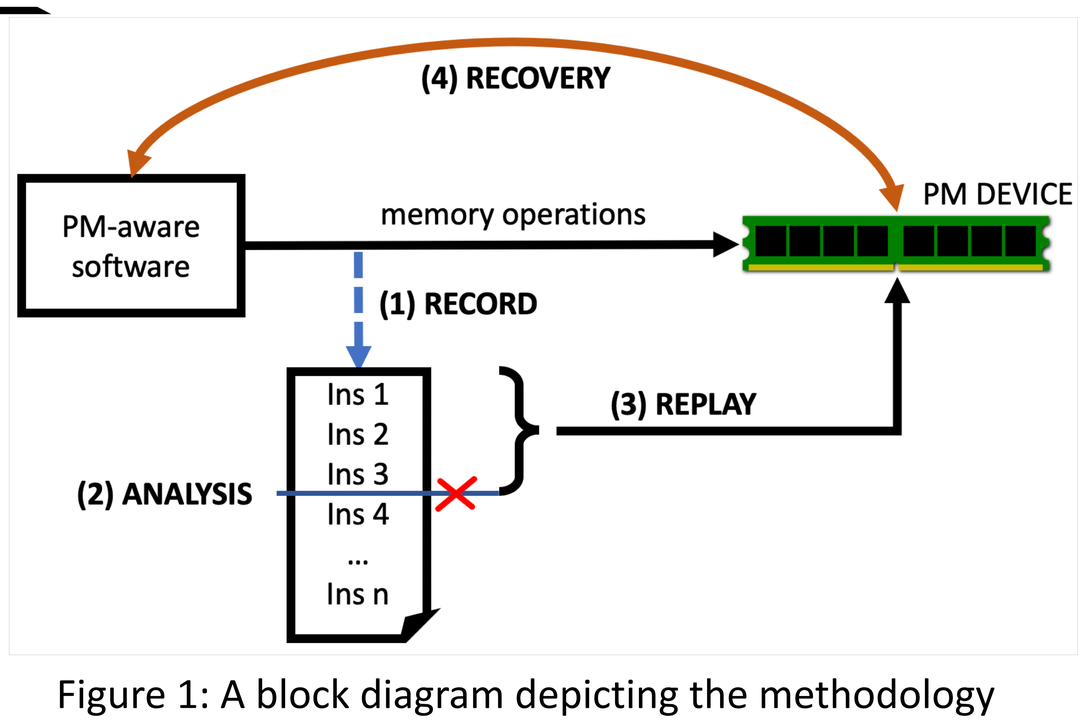


Figure 1: A block diagram depicting the methodology

- 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 the 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

## Preliminary 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 for the segmentation fault
- Identify more vulnerable points in the memory operations
- Extend the prototype to automatically identify the boundaries of PM functions
- Reduce the overhead of memory tracing
- Test more PM applications for recoverability

IOWA STATE UNIVERSITY
Department of Electrical and Computer Engineering