Fast, Flexible and Comprehensive Bug Detection for Persistent Memory Programs

Bang Di
dibang@hnu.edu.cn
Hunan University

Jiawen Liu
jliu265@ucmerced.edu
University of California, Merced

Hao Chen
haochen@hnu.edu.cn
Hunan University

Dong Li
dli35@ucmerced.edu
University of California, Merced

1. Introduction

Programming persistent memory (PM) has been widely studied in many software systems. Those systems are expected to recover to a consistent state and be able to resume execution in the event of a failure. However, programming to build a crash-consistent application for PM is challenging, because it must enforce data to reliably reach persistence for *data durability* and consider the order in which writes become persistent to provide certain *ordering guarantee*.

Debugging PM programs to detect violation of data durability and order guarantee often comes with large performance overhead, which makes PM debugging too time-consuming to comprehensively detect bugs (especially for those complicated applications). For example, Pmemcheck [1] (an industry-quality bug detector) and XFDetector [4] (a state-of-the-art bug detector) introduce 218x slowdown (two hours) and 1000x slowdown (nine hours) respectively to application execution, when debugging a PM-aware real workload, memcached with 1M persist operations (i.e., the acts of making cache lines persistent). Such large performance overhead not only comes from instrumentation of memory store, cache writeback and memory fence, in order to reason about the durability and ordering of persist operations; The overhead also comes from bookkeeping and updating persistency status of PM locations, which often dominates total overhead. In particular, whenever there is a store instruction, the debugger tool records which PM location has been modified; Whenever there is a cache writeback or fence, the debugger searches the records of PM locations to update the persistency status. Given a program with a large number of store, cache writeback and fence, frequent bookkeeping and expensive searching is the main reason accounting for time-consuming PM debugging.

To reduce performance overhead, some debuggers such as PMTest [5] avoid comprehensive examination of persist operations. They heavily rely on the programmer to intensively add assertion-like checkers into the program to selectively test durability and ordering guarantee. Furthermore, to support debugging for a persistency model, this method requires the programmer to introduce new checkers into each program and re-annotate it. Adding checkers requires the programmer have deep understanding on application semantics, persistency model, and hardware primitives employed in the model, which imposes heavy burden on the programmer. As a result, this method has limited bug coverage (or comprehensiveness), which means some bugs cannot be detected because of the lack of programmer-added checkers.

In conclusion, debugging PM programs faces a fundamental tradeoff between performance overhead and comprehensiveness. Large performance overhead or limited bug coverage makes debugging ineffective or even infeasible for PM programs. In this paper, we propose PMDebugger, a tool to detect crash consistency bugs. By considering PM program characterization, PMDebugger enables high-performance debugging without losing bug coverage. PMDebugger is fast, flexible and comprehensive for bug detection, discussed in Section 3.

Evaluation. Our evaluation results show that PMDebugger leads to 49.3x and 3.4x speedup on average, compared with state-of-the-art solutions, XFDetector and Pmemcheck. Compared with another state-of-the-art detector (PMTest) which is optimized for high performance, PMDebugger achieves comparable performance. PMDebugger identifies 78 synthetic or reproduced bugs (ten bug types in total), while XFDetector, Pmemcheck and PMTest identify 65 (six bug types), 55 (four bug types) and 61 bugs (five bug types) respectively. More importantly, PMDebugger detects 19 new bugs in a real application (memcached) and two new bugs from Intel PMDK (the two bugs are confirmed by Intel.

The paper is accepted into the prestigious conference ASPLOS'21 [1], and will be open-sourced.

2. Characterization of PM Programs

The existing PM debuggers largely ignore PM program characterizations, and hence have a mismatch between the design of data structures and algorithms for debugging, and PM program patterns. Such a mismatch leads to inefficient debugging mechanisms. We abstract three fundamental components from the PM program: memory store, cache writeback (e.g., cache

[1] The full paper was accepted in the 26th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2021. http://pasalabs.org/papers/2021/asplos21_pmdebugger_full_paper.pdf

line flushing or CLF) to enforce durability, and memory fence to provide ordering guarantee. We partition the stream of the three components collected from PM programs, and define that stores between two neighbouring CLFs form a *CLF interval*. We characterize how the three components are interleaved and distributed in typical PM programs, which motivates our debugger. We refer to the interleaving and distribution of the three components in a PM program, as the *PM program pattern*. We find three patterns.

- **Pattern 1**: For most stores, the data durability is guaranteed by the nearest fence;
- **Pattern 2**: Memory locations updated in a CLF interval are highly likely to be persisted together by the same single CLF;
- **Pattern 3**: Store happens more frequently than CLF and fence.

Pattern 1 gives us critical information on how to store and organize information for memory locations. The traditional debugger such as Pmemcheck, PMTest and XFDetector organizes memory locations based on their addresses into a tree-like structure, for the convenience of searching records (for handling CLF) and deleting records (for handling fence). This method, however, comes with the overhead of tree reorganization (e.g., merging and balancing). This overhead must be overweighed by the performance benefits brought by tree reorganization. The performance benefit comes from faster search and deletion. However, the pattern 1 tells us that the bookkeeping mechanism such as tree re-organization cannot be paid off very well for many memory locations, because once the nearest fence happens, the information for the memory locations is deleted, giving few opportunity to gain performance benefit in the long term. On the other hand, we see some memory locations survive multiple fences, showing the potential of using the tree-like structure.

Pattern 2 gives us critical information on whether it is promising to collectively maintain and update persistency status of memory locations. Collective processing enables fast query on status of memory locations, but can bring large performance benefit only when the persistency status of many memory locations *can be* collectively maintained. Pattern 2 shows us such potential.

Pattern 3 highlights the importance of efficiently processing memory store, because of its frequent occurrences.

3. Design

Figure 1 provides a high-level view of PMDebugger.

PMDebugger is fast. It enables high-speed debugging by introducing a highly efficient bookkeeping and updating mechanism. This mechanism is driven by the characterization study results (i.e., the three patterns). This mechanism includes two optimization techniques: (1) collectively managing status of memory locations, and (2) using a hybrid data structure for bookkeeping. Using (1), PMDebugger is able to greatly accelerate deletion of records when processing fence, and updating the records when processing cache writeback. For (2), PMDebugger combines an AVL tree and an array. Leveraging the strength of each data structure, PMDebugger splits and distributes the records into the two, based on the record lifetime, frequency of operations, and overhead of data structure maintenance. With the consideration of the program characterization, PMDebugger is able to break the tradeoff between performance overhead and bug coverage.

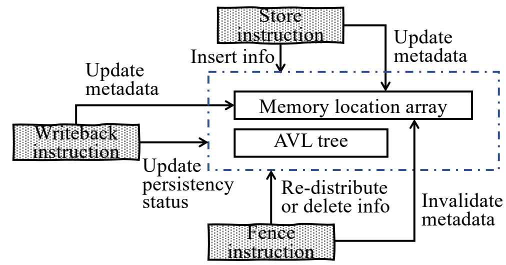

Figure 1: The design of PMDebugger.

PMDebugger is flexible. Built upon the data structures and optimization techniques customized to the PM debugging, PMDebugger introduces highly efficient operations for PM debugging, such as updating persistency status of memory locations and deleting their records. These debugging operations bring foundation to efficiently process the three fundamental components, based on which PMDebugger allows the user to introduce any rule for bug detection and implement high performance debugging. In essence, PMDebugger uses a hierarchical design composed of PM debugging-specific data structures, operations, and bug-detection algorithms (rules).

Given the flexibility provided by PMDebugger, we generalize nine rules to detect bugs for various persistency models. Among the nine, four of them are unique to the emerging relaxed persistency models [3].

PMDebugger is comprehensive for bug detection. Its comprehensiveness comes from its much shorter execution time than the existing PM debugger tools, which allows PMDebugger to thoroughly examine instructions; Its comprehensiveness also comes from its capability to detect various bugs for various persistency models. Using PMDebugger, we are able to identify bugs not identifiable by the existing tools [1,2,4,5].

References

[1] Intel Corporation. An introduction to pmemcheck. https://pmem.io/2015/07/17/pmemcheck-basic.html, 2015.

[2] Intel Corporation. Detect persistent memory programming errors using persistence inspector. https://software.intel.com/content/www/us/en/develop/articles/detect-persistent-memory-programming-errors-with-intel-inspector-persistence-inspector.html, 2020.

[3] Vaibhav Gogte, William Wang, Stephan Diestelhorst, Peter M. Chen, Satish Narayanasamy, and Thomas F. Wenisch. Relaxed persist ordering using strand persistency. In *ISCA*, 2020.

[4] Sihang Liu, Korakit Seemakhupt, Yizhou Wei, Thomas F. Wenisch, Aasheesh Kolli, and Samira Khan. Cross-failure bug detection in persistent memory programs. In *ASPLOS*, 2020.

[5] Sihang Liu, Yizhou Wei, Jishen Zhao, Aasheesh Kolli, and Samira Manabi Khan. Pmtest: A fast and flexible testing framework for persistent memory programs. In *ASPLOS*, 2019.