PiBench Online: Interactive Benchmarking of Persistent Memory Indexes

Xiangpeng Hao
Simon Fraser University
xha62@sfu.ca

Lucas Lersch
TU Dresden & SAP SE
lucas.lersch@sap.com

Tianzheng Wang
Simon Fraser University
tzwang@sfu.ca

Ismail Oukid
Snowflake Computing
ismail.oukid@snowflake.com

ABSTRACT

The emergence of persistent memory (PM), such as Intel Optane DC Persistent Memory Modules (DCPMM), opened up many opportunities for building high-performance indexes directly on PM. However, the many PM indexes proposed by prior work had their evaluation based on PM emulation using DRAM and therefore it was not clear how they would perform on real PM hardware. Moreover, they typically used ad hoc, in-house benchmarks and did not collect PM-specific hardware metrics that are key performance indicators and are instrumental for users and developers to understand the performance behavior of PM indexes. These issues call for a systematic, fair and reproducible approach for evaluating PM indexes.

This demonstration highlights the principles and lessons learned from our recent evaluation of PM indexes on real DCPMM and showcases PiBench, a unified benchmarking framework that enables fair and reproducible evaluation of PM indexes. In addition to common metrics, PiBench uniquely integrates monitoring tools to collect PM-specific hardware counters, allowing in-depth performance analysis. Our demonstration is enabled by PiBench Online, a new interactive system built on top of PiBench. Using PiBench Online, users can upload their own index implementations, run preset or customized workloads, and analyze results interactively, all through an easy-to-use web interface. PiBench is open-source and PiBench Online is deployed at https://pibench.org. We hope PiBench Online can promote fair comparison and reproducibility in database and systems communities.

PVLDB Reference Format:
Xiangpeng Hao, Lucas Lersch, Tianzheng Wang, Ismail Oukid. PiBench Online: Interactive Benchmarking of Persistent Memory Indexes. *PVLDB*, 13(12): 2817-2820, 2020.
DOI: https://doi.org/10.14778/3415478.3415483

1. INTRODUCTION

Next-generation, scalable and high-performance persistent memory (PM) offers both byte-addressability and persistence on the memory bus. PM blurs the boundary between storage and memory and creates opportunities for a new generation of fast index data structures that persist data and operate directly on PM, without extra, complex layers of persistence using HDDs or SSDs. These PM indexes become very attractive in the context of high-performance database systems. Many new designs were proposed [1, 2, 3, 7, 8, 9, 10], however, most of them were based on emulation using DRAM, since PM hardware was unavailable. Real PM exhibits unique performance characteristics such as asymmetric read/write speeds, limited bandwidth and longer access latency [5]. Therefore, it was not clear whether previously proposed algorithms and data structures would work as expected on real PM devices.

With the recently released Intel Optane DC Persistent Memory Modules (DCPMM), for the first time we explored how PM indexes perform on real devices, and distilled insightful principles to guide the design of future PM indexes [6]. Our results show that surprisingly, certain "recommended" design principles (e.g., copy-on-write) turned out to be harmful on real PM. Furthermore, we noticed the absence of a benchmarking framework for PM indexes that (1) allows fair and reproducible evaluation of PM indexes, and (2) collects PM-specific hardware metrics such as bytes transferred to PM, cache misses, and bandwidth consumption. PM indexes are usually benchmarked using ad hoc in-house tools and workloads, on different hardware platforms (emulated or real PM). Even the same workload may be implemented in very different ways by different researchers. Consequently, it becomes unreliable for researchers and practitioners to compare and reason about results reported by different papers. These issues call for a unified, easy-to-use benchmarking framework for PM indexes that allows researchers to compare and analyze different designs reliably and improve reproducibility.

In this demonstration, we highlight the insights and important guidelines obtained in our experimental study for building future PM indexes and database systems, and showcase *PiBench*, our new benchmarking framework used to obtain these results. The basis of our demonstration is the online deployment, *PiBench Online*,[1] which extends PiBench by introducing the following contributions:

- Intuitive GUI allowing interactive benchmarking and analysis.
- Generate and export plots to be directly included in papers.
- Facilitate the reproducibility of experiments across papers.
- Ready to be publicly deployed and made available for researchers to have easier access newly release PM hardware in an effort to democratize access to PM and obviate the need of emulation.

1.1 Introducing PiBench

PiBench is a unified framework uniquely designed for benchmarking PM indexes, with the following features.

Flexible and Easy-to-Use. PiBench defines a set of common interfaces (insert, lookup, delete, update, scan) supported by index structures and implements representative workloads. As Figure 1

[1] Live system deployed at https://pibench.org. Code available at https://github.com/sfu-dis/pibench-online.

Proceedings of the VLDB Endowment, Vol. 13, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3415478.3415483

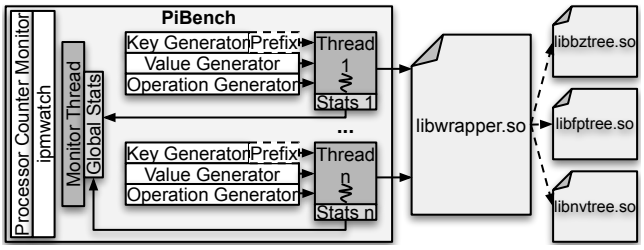

Figure 1: Overview of PiBench. User-provided indexes (shared libraries) implement a common set of interfaces and are linked with PiBench for benchmarking. Workload generation and metrics (including PM-specific ones) are done by PiBench for reproducibility.

shows, the user only needs to implement these interfaces and compile it as a shared library (e.g., `libbztree.so` for BzTree [1]) to link with PiBench. These index structures are typically implemented using C/C++, however, PiBench does not restrict it and a binding (e.g., `Cgo`[2] that allows Go code to call C code) can be used. PiBench will then issue the specified workloads against the index under evaluation and report results.

Highly Customizable. PiBench allows the user to specify a magnitude of parameters such as operations to perform, the number of threads to use, the metrics to collect, etc. Workloads are also fully customizable: the user can set different key/value sizes and distributions (e.g., zipfian/uniform), or use a preset YCSB [4] workload.

Tailored for PM. Behind the scenes, PiBench collects metrics such as the amount of completed operations and latency of individual operations. These metrics are collected at specified time intervals and sampling rates, allowing for more meaningful statistics, such as standard deviation and percentiles, in addition to common metrics such as average throughput and total runtime. We also integrate *Processor Counter Monitor*[3] and *IPMWatch*[4] in PiBench. This makes it unique in enabling in-depth analysis using PM-specific metrics like cache misses, PM/DRAM bandwidth consumption and the traffic between PM media and memory controller.

Lightweight. It is imperative for a benchmark tool to not introduce a significant overhead. We verified that PiBench exhibits only 2–5% of overhead when running a lookup-only workload (the fastest operation) in a highly optimized index (FPTree, arguably one of the fastest [7]), while collecting metrics such as throughput, tail latency and various hardware performance counters.

It is worth noting that PiBench can also be used to evaluate non-PM indexes (e.g., in-memory or disk based) as long as they implement the common interfaces (natively or through a wrapper).

1.2 Insights for Indexes on PM

Using PiBench, we conduct a comprehensive evaluation of representative PM index structures [1, 2, 7, 9] that cover a wide range of techniques used for designing PM indexes, such as unsorted leaf nodes, lock-free concurrency and copy-on-write designs. The results revealed several important and useful design principles and lessons learned, summarized below:

- PM indexes need to be designed to not exhaust the available PM bandwidth. While bandwidth was rarely an issue for DRAM trees, our results show for multiple tree structures, bandwidth indeed can be a bottleneck, especially if the memory channels are not fully populated with enough DCPMM.

[2] https://blog.golang.org/cgo

[3] https://github.com/opcm/pcm

[4] Available as part of Intel VTune Amplifier 2019 since Update 5.

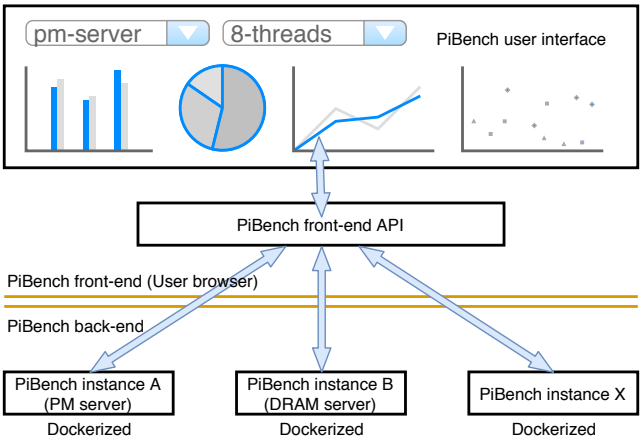

Figure 2: PiBench Online system architecture. The user interacts with a browser-based GUI to send benchmarking requests, which will be handled by a PM server backend based on PiBench.

- PM programming frameworks impose non-trivial overheads, leading to a significant slowdown on real PM compared to the originally reported numbers using emulations on DRAM. The interactions between data structures and PM libraries (such as PMDK[5]) must be carefully coordinated for both correctness and efficiency.
- There are several effective key building blocks and principles that should be followed when designing indexing structures for PM, such as fingerprinting for reducing unnecessary PM accesses, indirections to speed range queries and judicious use of DRAM for better performance. Though not proposed as individual designs, they are largely orthogonal and can be applied individually depending on the need.

More details can be found in our evaluation study [6] and will be highlighted in our demonstration, described next.

2. DEMONSTRATION: PiBench Online

Through this demonstration, we (1) showcase PiBench's capability, (2) highlight the key findings and principles that were distilled from our evaluation of representative PM indexes, and (3) promote the use of a unified, fair benchmark framework for future work in PM indexes. We achieve these goals with PiBench Online, an interactive benchmark and analysis system built on top of PiBench.

2.1 Overview

PiBench Online is an online service allowing users to submit their index implementation and run benchmarks via a web browser. It can be deployed on premise or in a cloud environment. Particularly, if deployed and shared publicly (in a commercial or academic cloud), researchers are able to benchmark their work on a common platform and make it easy to compare results across different publications.

As Figure 2 shows, using the frontend (web interface), the user simply needs to (1) upload a shared library that implements the index, and (2) set the desirable parameters (e.g., number of threads, operation types, number of operations) and metrics (e.g., throughput, tail latency) to start benchmarking. The backend running on a remote server then links with the index, executes workloads and returns the results. The user can interactively analyze, adjust the presentation of results and export them via the web interface.

Our demonstration will exhibit that PiBench Online enables push-button evaluation and analysis of index structures running on real

[5] https://github.com/pmem/pmdk/

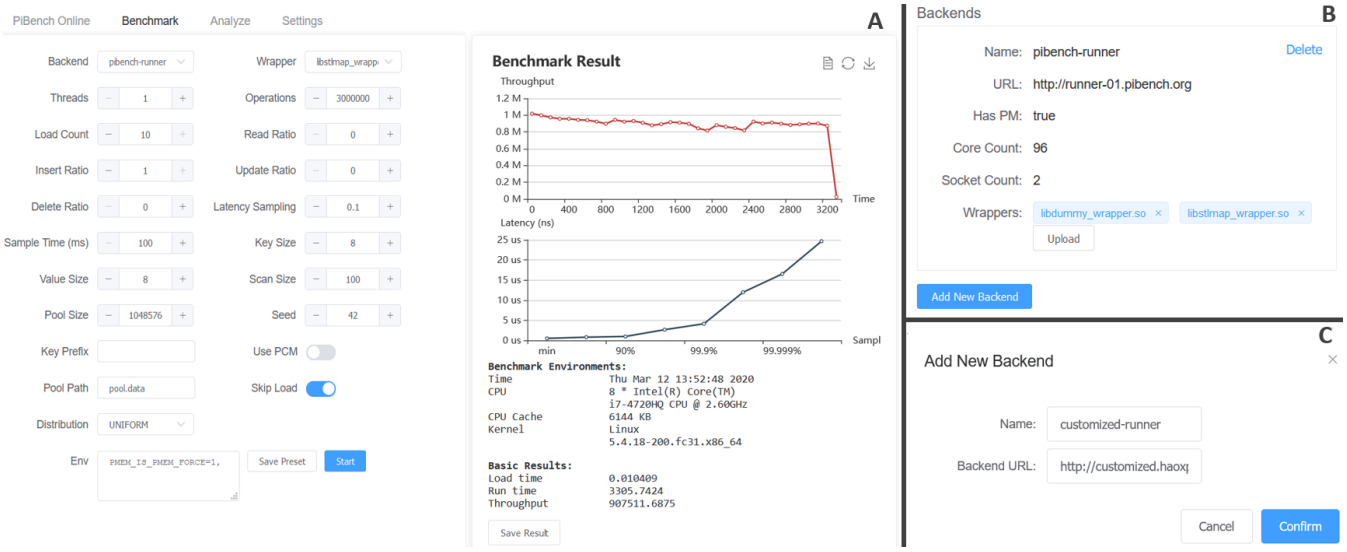

Figure 3: Web-based user interface (A) and UI components (B–C) in PiBench Online frontend.

Intel Optane DC Persistent Memory Modules. We deployed the backend on a dual-socket server with two 24-core Intel Xeon Gold 6252 CPUs clocked at 2.1 GHz (in total 48 cores, 96 hyperthreads). The server is fully populated with 1.5 TB of Intel Optane DCPMM (6×128 GB DCPMM DIMMs per socket) and 384 GB of DRAM (6×32 GB DRAM DIMMs per socket). The client side is a highly responsive in-browser GUI built using JavaScript. We provide recent PM index implementations for the demonstration (FPTree [7], BzTree [1], wbTree [2], NV-Tree [9]). Users can also upload other implementations for benchmarking and analysis. In case of connectivity problems, we plan to conduct the demonstration with local DRAM with delays injected to emulate PM.

2.2 User Interface

The user interacts with PiBench Online via a frontend which consists of three components, as we describe next.

Workload Management (Figure 3A). After selecting a backend (detailed later) and the index to evaluate, the user can set the parameters before starting the benchmark. Once the benchmark finishes, the backend returns the results which can be saved for later analysis.

Backend Management (Figures 3B-C). We allow the user to benchmark on different platforms and compare the results with different indexes. The user can deploy multiple backends on different machines, register them using the management panel and select the preferred backend in the main *"Benchmark"* panel in Figure 3(A).

Performance Analysis (Figure 4). Once the results are returned by the backend, the user can interactively analyze them. The analysis panel allows the user to select multiple benchmark results and visualize them in a single plot by selecting the desired metrics. All the benchmark results can be exported and are automatically saved in the browser, allowing the user to restore the session without re-running benchmarks. We will also maintain a leaderboard in PiBench Online to rank the performance of different indexes tested; these results can serve as a repository for the user to quickly learn about the performance of various indexes in the future.

2.3 PiBench Backend

PiBench backend consists of two components: (1) An HTTP server (implemented in `rust`) that communicates with the frontend and parses user input into PiBench parameters. (2) A PiBench binary

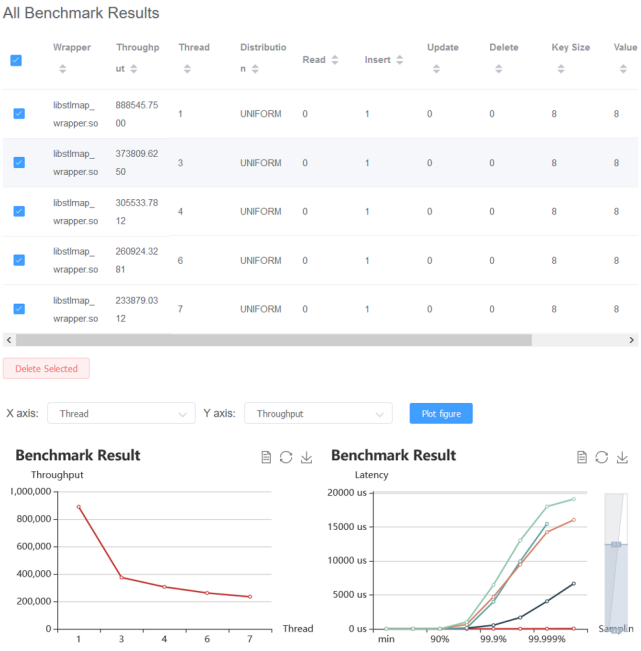

Figure 4: Interactive performance analysis panel.

which is invoked by the HTTP server to generate the workload, spawn new threads, execute the benchmark and collect the results.

Upon initialization, the backend checks basic system parameters (e.g., number of threads, PM capacity) and returns these to the client. As Figure 3(B) shows, PiBench Online can also use DRAM emulation or evaluate non-PM indexes if no PM is detected (`Has PM`). It then (optionally) performs a sample benchmark with a simple `C++ std::map` index for a test run. The results can be queried by the client for the user to compare with other indexes to be uploaded. After an index (shared library) is uploaded, the backend then links it with PiBench binary (using `dlopen`[6]) and starts the

[6] https://man7.org/linux/man-pages/man3/dlopen.3.html

actual benchmark. Indexes may have different dependencies; for easy management, we expect the user to compile the dependencies into a single shared library or upload them together with the index to be evaluated.

3. DEMONSTRATION SCENARIOS

Our demonstration consists of three parts: (1) a brief introduction, (2) two live scenarios that the audience can participate and (3) a leaderboard/takeaway messages presentation.

3.1 Poster/Video: Background Introduction

To set the stage, we start by introducing the background of PM devices, research in PM-based database systems and the challenges of evaluating PM indexes. We also cover the basics about PiBench and PiBench Online. A (virtual) poster will be put up to aid our explanation, and a video/GIF animation will be displayed on a monitor for the audience to get a first glance of how the system works at a high level. The poster will also highlight the key findings described in Section 1.2 and more details from our evaluation work [6].

3.2 Live: Benchmarking

In this scenario, we show the user how to use PiBench Online to evaluate PM indexes under a particular workload. The user may choose from one of our pre-built index libraries, or even upload their own (via a USB stick or by downloading it through the internet). We then allow the user to set hardware parameters (e.g., using actual or emulated PM) and configure the workload. To get the user started, we provide an example configuration, on top of which the user can adjust the parameters using our web-based GUI. The modified configuration can then be saved as a preset for future use. The user can choose to start with a pre-set workload and (optionally) adjust the operation/thread count, key/payload size, distribution and so on. In addition, the user can optionally enable *PCM* and *IPMWatch* to collect hardware counter values.

Once the configuration is set, the user can click the `start` button in Figure 3(A), which will cause the frontend to send a request to the server and start benchmarking. The user can now wait until the server finishes the benchmark and sends back the results. To compare the results among different configurations, the user can save the current benchmark results by clicking the `save` button.

3.3 Live: Interactive Result Analysis

After the benchmark scenario, the frontend parses the results received from the server and displays them with interactive data plots. This allows the user to analyze the results and compare different performance metrics. For example, one may choose to observe throughput (Y axis) over thread count (X axis), or change to see other metrics using the `Plot figure` button shown in Figure 4. The user may begin with selecting previously saved results. The system will then automatically highlight the differences in the configurations and results. We then show that the user can selectively focus on a subset of aspects and visualize the differences by plotting the figures. The figures are fully interactive, allowing the user to zoom in and focus on a particular area or dynamically add new data series to the figure. Finally, we show that the results (figures and raw data) can be exported for further analysis and publication.

3.4 Leaderboard and Takeaway Messages

Benchmark results (with metrics such as throughput, tail latency and memory consumption) can be saved and ranked in a leaderboard to compare different indexes. The final part of the demonstration exhibits this leaderboard and highlights key takeaway messages distilled from our own experience, including the scarcity and impact of limited PM bandwidth, impact of PM allocator and how copy-on-write (which was thought to be desirable) can be a bad fit for PM indexes. Finally, we present a "wish list" for future PM devices to highlight the desirable features for building database systems, such as DRAM-level performance and durable CPU caches.

4. SUMMARY

Persistent memory brings both opportunities and challenges to the making of database systems, in particular building and evaluating index structures in a fair and reproducible manner. Prior work was proposed based on (inaccurate) emulations using DRAM and evaluation was done in ad hoc ways using in-house benchmark tools and workloads, making it hard for researchers to reasonable about the results reported by different papers. We demonstrate PiBench, a unified benchmark framework specifically designed for evaluating PM indexes. With PiBench, researchers and practitioners only need to implement the common interfaces of indexes and specify the workload to conduct evaluations. All the remaining work (issue index operations and collect metrics) is done systematically by PiBench. Through a new interactive deployment, PiBench Online, our demonstration focuses on (1) presenting PiBench's capabilities and PiBench Online's interactive workflow for testing and analysis, (2) highlighting the important lessons learned from evaluating recent PM indexes and more importantly (3) distilling the critical design principles that should be followed when designing future PM indexes and data structures on real PM hardware. For questions and comments, contact support@pibench.org.

5. REFERENCES

[1] J. Arulraj, J. J. Levandoski, U. F. Minhas, and P. Larson. BzTree: A High-Performance Latch-free Range Index for Non-Volatile Memory. *PVLDB*, 11(5):553–565, 2018.

[2] S. Chen and Q. Jin. Persistent B+-Trees in Non-Volatile Main Memory. *PVLDB*, 8(7):786–797, 2015.

[3] P. Chi, W.-C. Lee, and Y. Xie. Making B+-Tree Efficient in PCM-Based Main Memory. In *ISLPED*, pages 69–74, 2014.

[4] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud Serving Systems with YCSB. In *ACM SoCC*, pages 143–154, 2010.

[5] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor, J. Zhao, and S. Swanson. Basic Performance Measurements of the Intel Optane DC Persistent Memory Module. *CoRR*, abs/1903.05714, 2019.

[6] L. Lersch, X. Hao, I. Oukid, T. Wang, and T. Willhalm. Evaluating Persistent Memory Range Indexes. *PVLDB*, 13(4):574–587, 2019.

[7] I. Oukid, J. Lasperas, A. Nica, T. Willhalm, and W. Lehner. FPTree: A Hybrid SCM-DRAM Persistent and Concurrent B-Tree for Storage Class Memory. In *SIGMOD Conference*, pages 371–386, 2016.

[8] F. Xia, D. Jiang, J. Xiong, and N. Sun. HiKV: A Hybrid Index Key-Value Store for DRAM-NVM Memory Systems. In *USENIX ATC*, pages 349–362, 2017.

[9] J. Yang, Q. Wei, C. Wang, C. Chen, K. L. Yong, and B. He. NV-Tree: A Consistent and Workload-Adaptive Tree Structure for Non-Volatile Memory. *IEEE Trans. Computers*, 65(7):2169–2183, 2016.

[10] P. Zuo, Y. Hua, and J. Wu. Write-Optimized and High-Performance Hashing Index Scheme for Persistent Memory. In *USENIX OSDI*, pages 461–476, 2018.