PiBench Online: Interactive Benchmarking of Persistent Memory Indexes

Xiangpeng Hao
xha62@sfu.ca
Simon Fraser University

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

Ismail Oukid
ismail.oukid@snowflake.com
Snowflake Computing

Tianzheng Wang
tzwang@sfu.ca
Simon Fraser University

ABSTRACT

The recent emergence of scalable persistent memory (PM) such as Intel Optane DC opened up many opportunities for building index structures (trees and hash tables) that are fast and recovers instantly by persisting and working directly in PM. Many new PM indexes have been proposed, however, evaluating them in a fair, reproducible manner becomes hard. Much prior work used relied on DRAM-based emulation and in-house microbenchmarks. It is both time-consuming and error-prone to measure accurately interested metrics (e.g. throughput, memory consumption and tail latency) without a unified benchmarking framework.

We demonstrate PiBench Online, an interactive system for benchmarking PM indexes in a fair and reproducible manner. PiBench Online is an online service based on PiBench, the first unified, highly customizable benchmarking framework for PM indexes. PiBench Online enables push-button evaluation of PM indexes: the user can upload a shared library that implements the index, set parameters to run customized benchmarks, and analyze results interactively, all through an easy-to-use web interface. We have made PiBench Online available at http://pibench.org and hope it can promote fair comparison and reproducibility in future PM index research.

ACM Reference Format:
Xiangpeng Hao, Lucas Lersch, Ismail Oukid, and Tianzheng Wang. 2020. PiBench Online: Interactive Benchmarking of Persistent Memory Indexes. In *SIGMOD '20: ACM SIGMOD/PODS International Conference on Management of Data, June 14–19, 2020, Portland, OR.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Next-generation, scalable persistent memory (PM) promises both byte-addressability and persistence on the memory bus. This blurs the boundary between storage and memory, and offers much potential for a new generation of high-performance index structures (trees, tries and hash tables) that persist data and operate directly in PM, without extra, complex layers of persistence using disks or SSDs. However, since PM exhibits unique performance characteristics [4], it is necessary to re-design index structures to fit the idiosyncrasies of PM, as seen by many recent proposals from databases [1, 2, 6], computer systems [7, 8] and even computer architecture [3] communities.

Fairly and reliably evaluating different PM indexes in a reproducible manner has become an important problem. Different researchers may use different hardware (emulation based or using real PM devices like Intel Optane DC), making it hard to compare and interpret the reported numbers. Even using the same hardware, unlike evaluations of full database systems where standard benchmarks (such as TPC-C) are widely used, index structures are often benchmarked using in-house tools and workloads. As a result, even the same workload may be implemented in very different ways by different researchers making the comparison. Therefore, it becomes particularly hard and time-consuming to reproduce and verify experimental results for PM indexes. We believe that a unified, easy-to-use benchmarking framework for PM indexes would allow researchers to compare and analyze different designs with ease and improve reproducibility within and beyond the database community.

1.1 PiBench: a Unified Persistent Index Benchmarking Framework

To solve this problem, we developed PiBench [5], a unified benchmarking framework for PM indexes. PiBench defines a set of common interfaces (insert, lookup, delete, update, scan) supported by index structures and implements highly customizable workloads. To evaluate an index structure, one only needs to implement these interfaces and compile it as a shared library to link with PiBench, which will then issue the specified workloads against the index structure under evaluation and report various metrics and results. PiBench is highly customizable, allowing the user to specify parameters such as operations to perform, the number of threads to use, key/value sizes and distributions. Behind the scenes, PiBench collects various statistics including throughput, tail latency and the associated standard deviation. We also integrated PiBench with the processor counter monitor (PCM) and `ipmwatch` which expose values of low-level CPU counters to allow in-depth performance analysis (e.g., cache misses, PM/DRAM bandwidth consumption). More details of PiBench can be found elsewhere [5]; here we focus on its demonstration, described next.

1.2 Demonstration: PiBench Online

We have used PiBench to evaluate classic PM indexes and distilled useful principles [5]. This demonstration introduces PiBench Online, an interactive benchmarking and analysis system using PiBench. Through the demonstration, we (1) showcase PiBench's capability and (2) highlight the key findings and pinciples that were distilled from our evaluation of representative PM indexes. We also make PiBench Online publicly available so it is easily accessible by the community (databases and beyond), and promote the use of a unified, fair benchmarking framework for future work in PM indexes,

PiBench Online is an online service that allows the user to submit their index benchmarking request via a browser. As Figure 1 shows, using PiBench Online's front-end (web interface), the user simply (1) uploads a shared library that implements the index, and (2) sets the desirable parameters (e.g., number of concurrent threads, operation types, runtime) and metrics (e.g., throughput, tail latency). PiBench Online's backend then links with the library and issues the specified workloads and returns results. The user can analyze and adjust the presentation of results interactively via a web interface. The results can be exported to aid presentation (e.g., to be included in a paper).

Our demonstration will exhibit that PiBench Online enables push-button evaluation and analysis of index structures running on real Intel Optane DC Persistent Memory Modules. We deployed the server side on a machine with a 24-core (48 hyperthreads) Intel Xeon Gold 6252 CPU (2.1 GHz) with 1.5 TB of Intel Optane DCPMM. The client side is a highly responsive in-browser GUI built using JavaScript. We have pre-built several recent PM indexes (FPTree [6], BzTree [1] and a new extendible PM hash table). The user will be able to

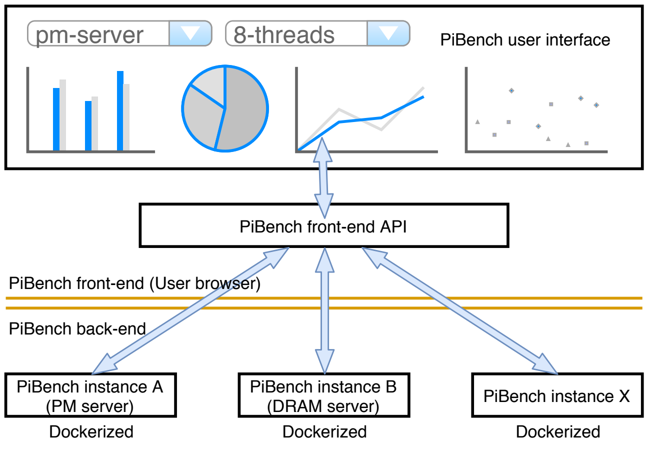


Figure 1: PiBench Online system architecture.

directly upload them (or their own) and follow instructions on the web interface to finish the benchmarking and analysis process. In the case of connectivity problems, we will still be able to conduct the demonstration with local, emulated PM based on DRAM.

Next, we describe the design of PiBench Online, followed by a walkthrough of our planned demonstration scenarios.

2 SYSTEM DESIGN AND COMPONENTS

We design PiBench Online to enable (1) customized benchmarks and (2) a unified analysis experience, through a client and a server component, described next.

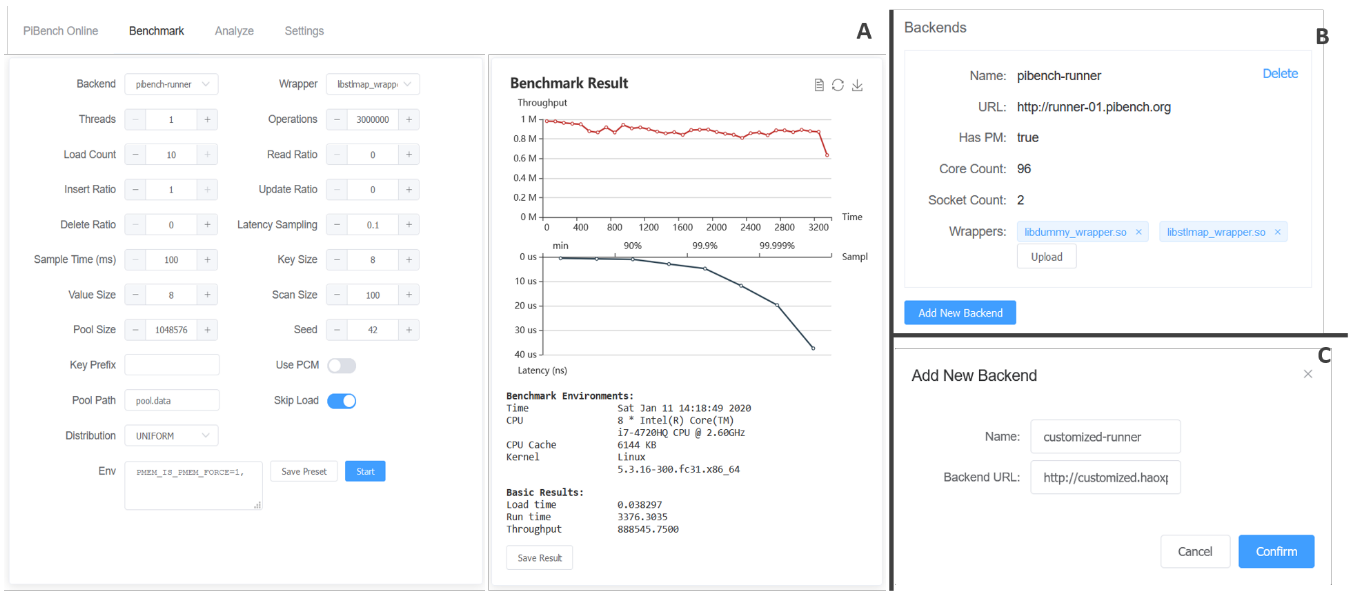
2.1 Client

The user interacts with PiBench through the front-end of PiBench Online. Figure 2 and Figure 3 shows screenshots of it, consisting of three components, described below.

Workload Management (Figure 2-A). After selecting the desired backend runners and indexes, the user can adjust the configurations and send the workload to the backend runner. Once the runner finished the benchmark, it will send back the collected performance statistics. The user can save the performance data for later analysis.

Backend Management (Figure 2-B,C). PiBench Online enables users to benchmark on multiple platforms and compare the results with different indexes. The users can deploy their backend runner on any supported machines and connect these runners on the PiBench online website. This management panel allows users to connect new runners and upload new indexes to a runner.

Performance Analyses (Figure 3). Once the user finished benchmarking, s/he can interactively analyze the interested data. The analysis panel allows the user to select multiple benchmark results and visualize these results by selecting the



Backend	pibench-runner	Wrapper	libstlmap_wrappe
Threads	1	Operations	3000000
Load Count	10	Read Ratio	0
Insert Ratio	1	Update Ratio	0
Delete Ratio	0	Latency Sampling	0.1
Sample Time (ms)	100	Key Size	8
Value Size	8	Scan Size	100
Pool Size	1048576	Seed	42
Key Prefix		Use PCM	
Pool Path	pool.data	Skip Load	
Distribution	UNIFORM		
Env	PMEM_IS_PMEM_FORCE=1,		

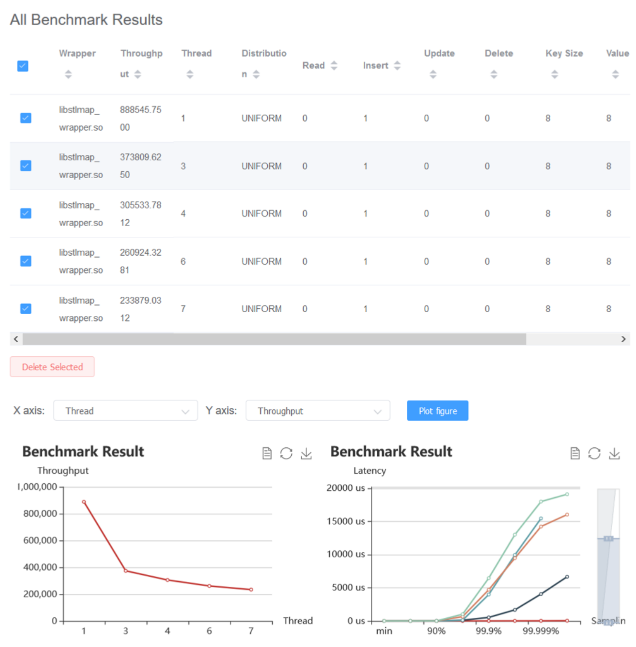
```
Benchmark Environments:
Time                    Sat Jan 11 14:18:49 2020
CPU                     8 * Intel(R) Core(TM)
                        i7-4720HQ CPU @ 2.60GHz
CPU Cache               6144 KB
Kernel                  Linux
                        5.3.16-300.fc31.x86_64

Basic Results:
Load time               0.038297
Run time                3376.3035
Throughput              888545.7500
```

Name:	pibench-runner
URL:	http://runner-01.pibench.org
Has PM:	true
Core Count:	96
Socket Count:	2
Wrappers:	libdummy_wrapper.so ×, libstlmap_wrapper.so ×

Name:	customized-runner
Backend URL:	http://customized.haoxi

Figure 2: PiBench Online user interface (A) and some selected UI components (BC).



	Wrapper	Throughput	Thread	Distribution	Read	Insert	Update	Delete	Key Size	Value
✓	libstlmap_wrapper.so	888545.7500	1	UNIFORM	0	1	0	0	8	8
✓	libstlmap_wrapper.so	373809.6250	3	UNIFORM	0	1	0	0	8	8
✓	libstlmap_wrapper.so	305533.7812	4	UNIFORM	0	1	0	0	8	8
✓	libstlmap_wrapper.so	260924.3281	6	UNIFORM	0	1	0	0	8	8
✓	libstlmap_wrapper.so	233879.0312	7	UNIFORM	0	1	0	0	8	8

X axis: Thread Y axis: Throughput

Figure 3: PiBench Online interactive performance analysis panel.

desired legends. All the benchmark results are automatically saved in the user browser, and this data persistency allows the user to restore the session without re-run the benchmarks. We also maintain a leaderboard in PiBench Online to rank the performance of different indexes tested; these results can serve as a repository for users to quickly learn about the performance of various indexes in the future.

2.2 Server

PiBench backend consists of two components: 1. a `rust` implemented HTTP server that communicates with the PiBench frontend and translates the user input into structured PiBench parameters. 2. a PiBench binary which generates the workload, spawns new processes, executes the benchmark and collects the performance data.

The PiBench binary is a benchmark framework specifically designed for indexes benchmarking. Users can implement a simple wrapper on top of their indexes and benefit from the framework's comprehensive yet uniform benchmark workload. PiBench allows users to configure the key/payload size, thread count, persistent pool size, warmup size etc. It also optionally supports the Intel PCM module and the Intel `ipmwatch`. The Intel PCM module collects more detailed performance data such as cache reference and memory bus bandwidth. The `ipmwatch` is a new performance analysis tool designed to monitor persistent memory statistics, once enabled, it observes the internal states of Intel DCPMM and reports the persistent memory reference and internal buffer hit rate. Users can easily and optionally enable these two tools to collect more fine-grained performance statistics.

The HTTP server is designed to be as efficient as possible so that it will not slow down the rest of the benchmark. Every backend runner by default has a C++ `std::map` index, the runner on initialization will perform a standard benchmark using this index, and use this benchmark result as the machine's reference performance. To support a wider variety of indexes, the runner accepts user-uploaded indexes, but requires the index to solve the dependencies and ensure the compatibilities of the running platform.

2.3 Deployment

3 DEMONSTRATION SCENARIOS

Our demonstration consists of three parts: a brief presentation of the background and two hands-on live scenarios that the audience can participate.

3.1 Poster/Video: Background Introduction

To set the stage, we start with a brief introduction, including the background of PM index research, the challenges of evaluating PM indexes fairly, and the basic idea and functionality of PiBench Online. A poster will be put up to aid our explanation, and a (silent) video/GIF animation will be displayed on a monitor for the user to get a first glance of how the system looks like. The poster will also highlight the key findings from our evaluation work [5] that is presented after a live demonstration.

3.2 Live: Benchmarking

In this scenario, the user wants to know how an index performs with 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 choose the hardware setting (e.g., whether to use actual or emulated PM) and configure the workload. To get the user started, PiBench Online provides an example insert-only, single-thread configuration, on top of which the user can adjust the parameters using our web UI. The modified configuration can then be saved as a preset for future use. The user can adjust the operation/thread count, key/payload size, distribution and so on. In addition to these basic parameters, the user can also optionally enable Processor Counter Monitor and `ipmwatch` to record additional performance statistics.

Once the user finished the configuration, they can click the `start` button, which will cause the frontend to send a request to the server and start a benchmark. 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 additionally save the current benchmark results by clicking the `save` button.

3.3 Live: Interactive Result Analysis

After the benchmarking scenario, the frontend will parse the results sent back by the server and visualize them by plotting interactive figures. This allows the user to analyze the results and compare different performance metrics. For example, the user may choose to observe throughput (y-axis) over threads (x-axis), or change to see other metrics using the `Plot figure` button shown in Figure 2(D). The user may begin with selecting previously saved benchmark 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 difference by plotting the figures. The figures are fully interactive; it allows users to zoom in to focus on a particular data zone or dynamically add new data series to the figure.

3.4 Leaderboard and Takeaways Messages

Benchmark results (with metrics such as throughput, tail latency and memory consumptions) can be saved and ranked in a leaderboard. The final part of the demonstration will show case this leaderboard and highlight several important takeaway messages distilled from our own experience, including the scaracity 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.

4 AVAILABILITY

PiBench Online is publicly available at http://pibench.org. Contact support@pibench.org for questions and comments.

REFERENCES

[1] Joy Arulraj, Justin Levandoski, Umar Farooq Minhas, and Per-Ake Larson. 2018. BzTree: A High-Performance Latch-free Range Index for Non-Volatile Memory. *PVLDB* 11, 5 (2018), 553–565.

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

[3] Ping Chi, Wang-Chien Lee, and Yuan Xie. 2014. Making B+-Tree Efficient in PCM-Based Main Memory. In *Proceedings of the 2014 International Symposium on Low Power Electronics and Design (ISLPED '14)*. 69–74.

[4] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R Dulloor, et al. 2019. Basic Performance Measurements of the Intel Optane DC Persistent Memory Module. *arXiv preprint arXiv:1903.05714* (2019).

[5] Lucas Lersch, Xiangpeng Hao, Ismail Oukid, Tianzheng Wang, and Thomas Willhalm. 2019. *PVLDB* 13, 4, 574–587.

[6] Ismail Oukid, Johan Lasperas, Anisoara Nica, Thomas Willhalm, and Wolfgang Lehner. 2016. FPTree: A Hybrid SCM-DRAM Persistent and Concurrent B-Tree for Storage Class Memory. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, 371–386.

[7] Fei Xia, Dejun Jiang, Jin Xiong, and Ninghui Sun. 2017. HiKV: A Hybrid Index Key-value Store for DRAM-NVM Memory Systems. In *Proceedings of the 2017 USENIX Annual Technical Conference (USENIX ATC '17)*. 349–362.

[8] Pengfei Zuo, Yu Hua, and Jie Wu. 2018. Write-Optimized and High-Performance Hashing Index Scheme for Persistent Memory. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 461–476.