CCL-BTree: A Crash-Consistent Locality-Aware B+-Tree for Reducing XPBuffer-Induced Write Amplification in Persistent Memory

Zhenxin Li , Shuibing He , Zheng Dang, Peiyi Hong ,
Xuechen Zhang [†], Rui Wang , Fei Wu

浙江大学
Zhejiang University

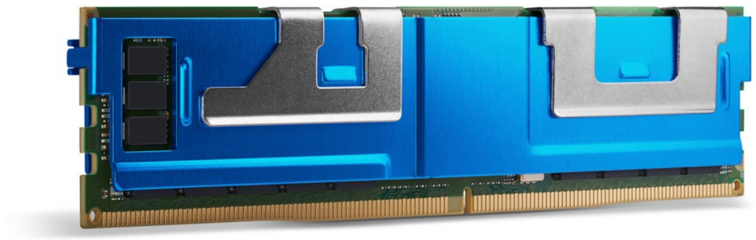
† WASHINGTON STATE UNIVERSITY

EuroSys 2024

Persistent memory (PM)

The first commercially available PM device -- Intel Optane Persistent Memory

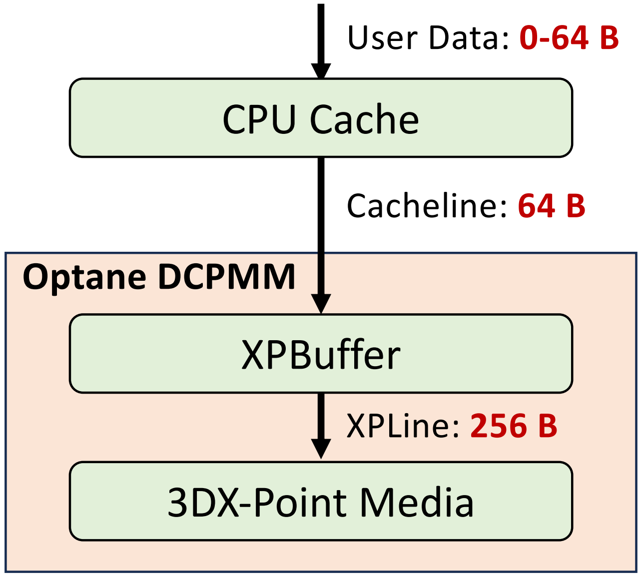
- Memory-like speed and byte addressability
  - Low latency (~100 ns for small I/O)
  - High bandwidth (3 GB/s write and 8 GB/s read per DIMM)
  - Byte-addressable using load/store instructions
- Storage-like capacity and persistence
  - Up to 3TB (6 * 512 GB) per socket
  - Durable storage like SSD



Intel Optane Persistent Memory 200 Series

Two types of write amplification in PM

Small writes suffer from write amplification in two hardware layers!



Cacheline-induced amplification (CLI)

$$\text{CLI} = \frac{\text{the amount of data written to XPBuffer}}{\text{the amount of users' data}}$$

XPBuffer-induced amplification (XBI)

$$\text{XBI} = \frac{\text{the amount of data written to media}}{\text{the amount of users' data}}$$

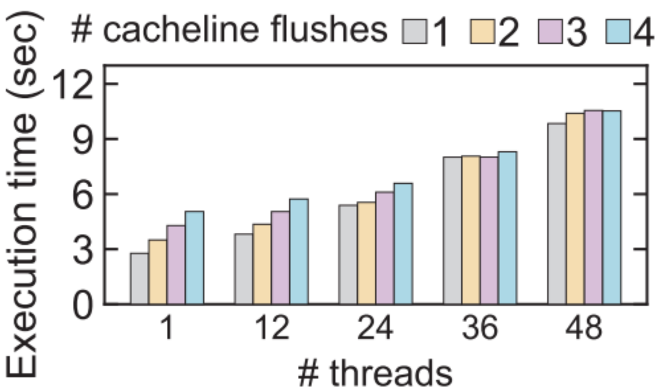
Which one has a greater impact on write performance?

CLI vs. XBI amplification

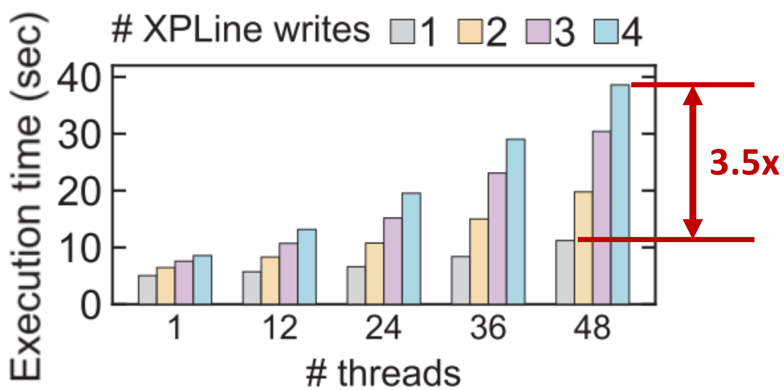
- One socket with 4 * 128 GB Intel Optane DCPMMs 200 series

(a): Fix the value of XBI and increase the CLI by four times

(b): Fix the value of CLI and increase the XBI by four times



(a) The impact of CLI.



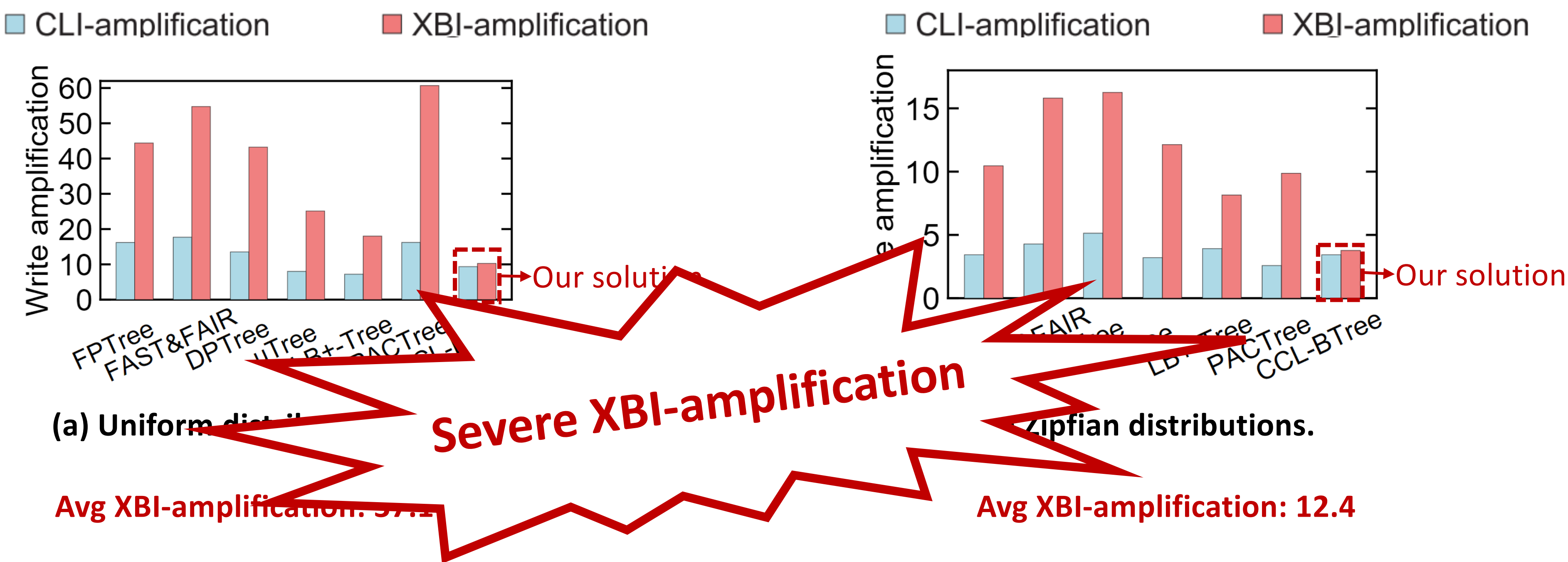
(b) The impact of XBI.

XBI-amplification determines the performance when the PM bandwidth is exhausted!

XBI-amplification in persistent B+-trees

Most existing persistent B+-trees focus on reducing CLI amplification, not XBI amplification

- One socket with 4 * 128 GB Intel Optane DCPMMs 200 series
- 48 threads, 8B key, 8B value, warm up 50M KVs & upsert 50M KVs



(a) Uniform **Zipfian distributions.**

Avg XBI-amplification: 37.1 **Avg XBI-amplification: 12.4**

Our Solution: CCL-BTree

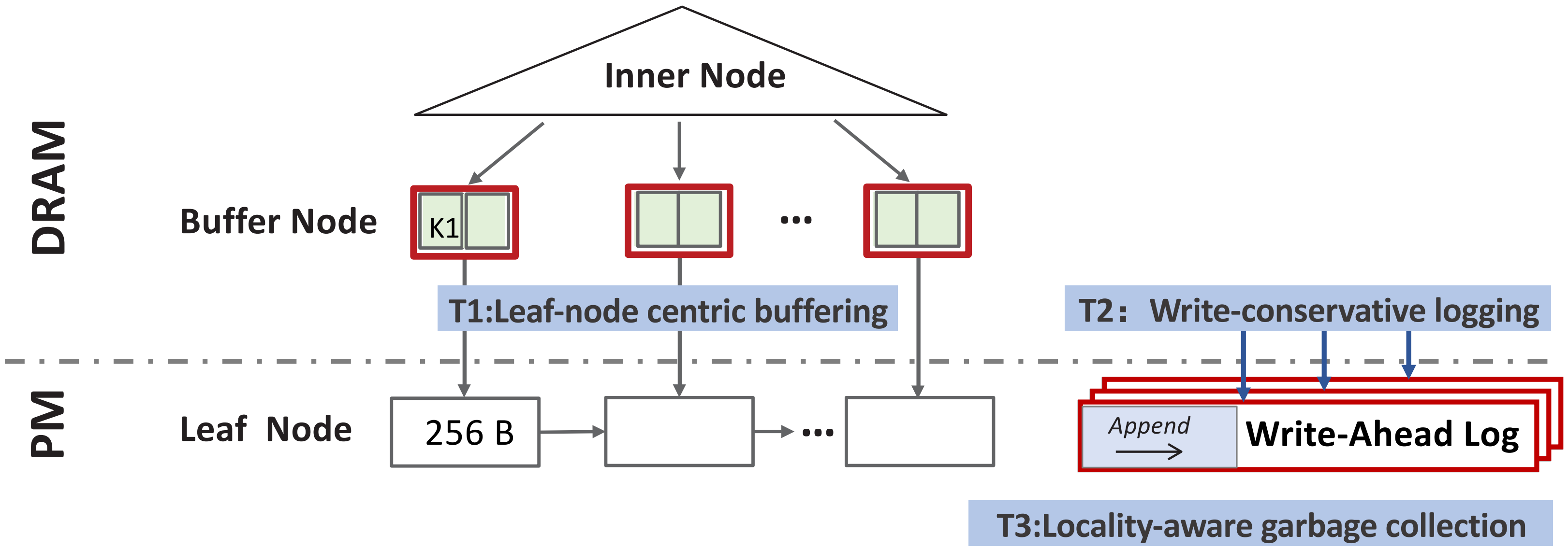
Main components:

- Buffer node

  Cache multiple small KVs in buffer nodes and flush them to PM in batch

- Write ahead log

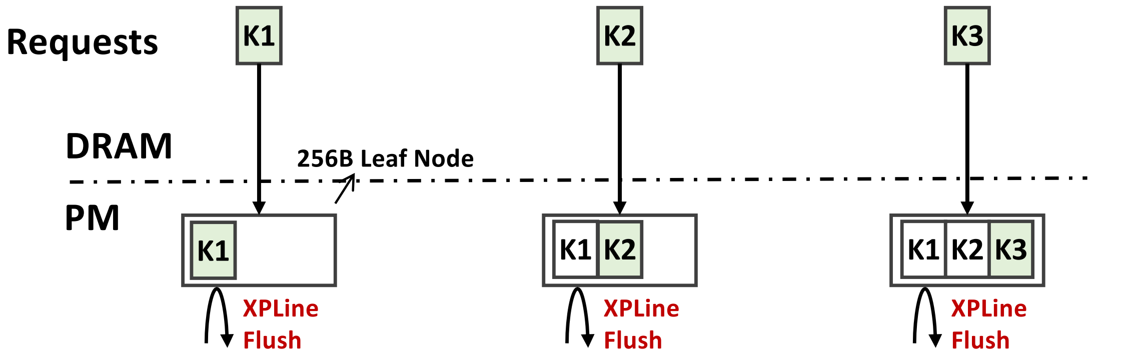
  Maintain the crash consistency of data in buffer nodes



Technique 1: leaf-node centric buffering

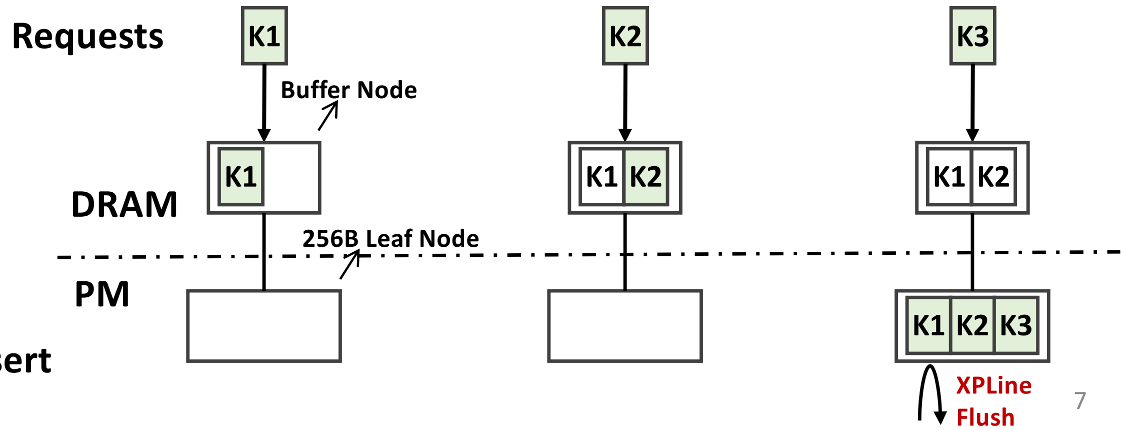
Merge contiguous small writes and flush them to leaf nodes in batch

- ☐ Directly insert into the leaf node
  - ☹ Trigger one XPLine flush for each KV
  - ☹ Severe XBI-amplification



(a) Naïve insert

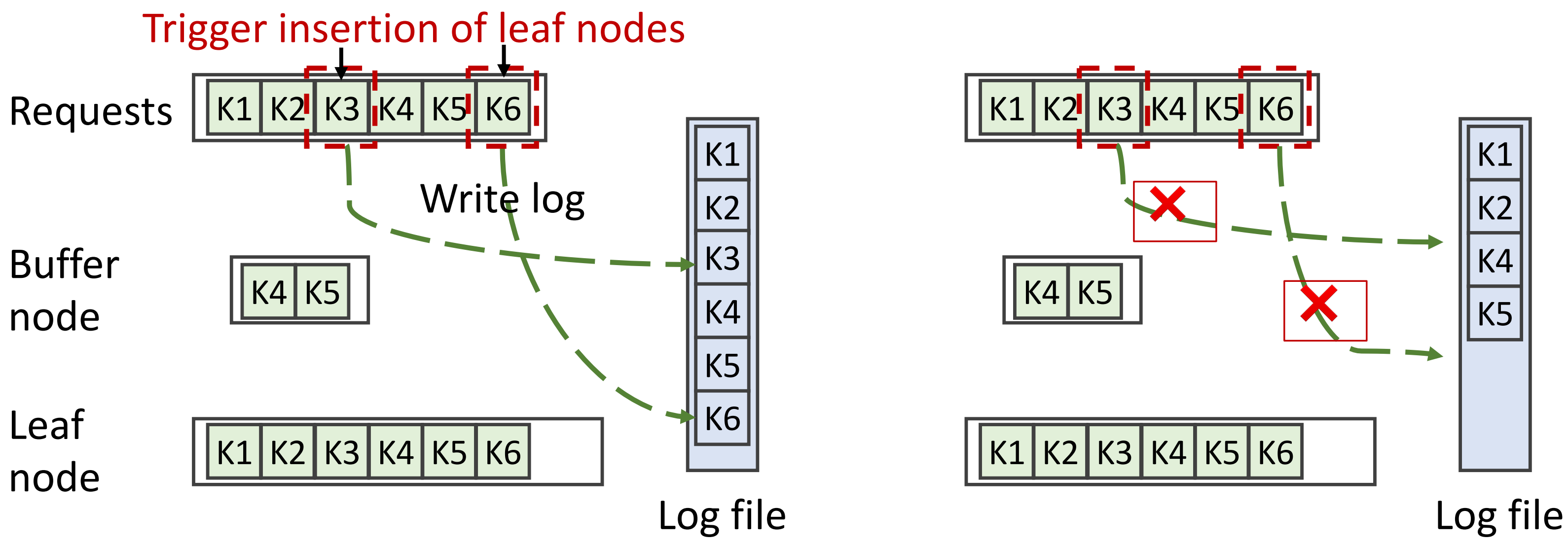
- ☐ Introduce buffer node (# slots = 2)
  - ☺ Trigger one XPLine flush for 3 KVs
  - ☺ Lower XBI-amplification



(b) Buffered insert

Technique 2: write-conservative logging

Skip the unnecessary log operations while ensuring the crash consistency



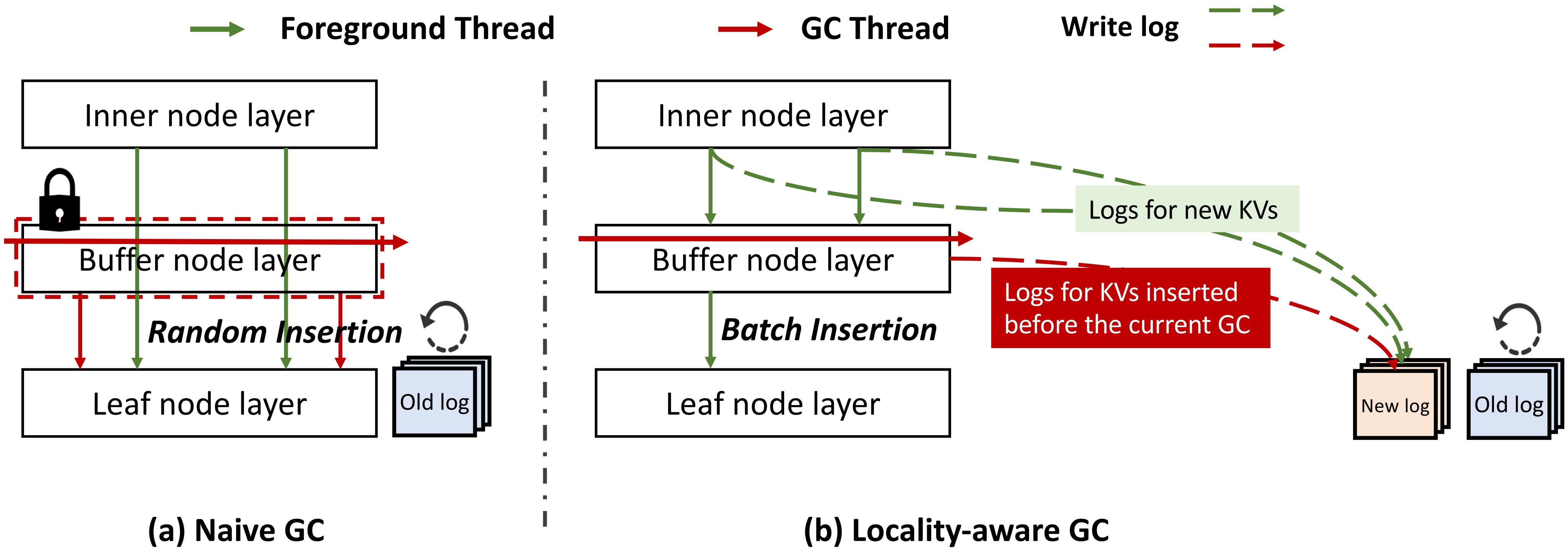
(a) Naive logging **(b) Write-conservative logging**

☹ A naive logging method writes logs for each new KV

☺ CCL-BTree skips logging for KVs that trigger the insertion of leaf nodes when the buffer nodes are full

Technique 3: locality-aware garbage collection

Convert random leaf node access to sequential logging



☹ The naïve GC flushes all KVs in buffer nodes to leaf nodes, which incurs many random accesses.

☺ Our GC only copies a portion of KVs in buffer nodes to new logs in sequential manner.

Theoretical performance analysis

- Insert K key-value pairs with uniform distributions
- N_{batch}: the number of slots in a buffer node.
- The log item size: 24 bytes (16-byte KV and 8-byte timestamp)

☺When $N_{batch} = 2$ (the default value), CCL-BTree reduces 60.4% XPLine writes

# of 256 B XPLine writes	Traditional B+-tree	CCL-BTree
Leaf Node	K	$\frac{1}{N_{batch}+1} * K$
Log	0	$\frac{24}{256} * \frac{N_{batch}}{N_{batch}+1} * K$
Total	K	$\frac{256 + 24 * N_{batch}}{256 * (N_{batch}+1)} * K$

leaf-node centric buffering

write-conservative logging

Experimental setup

☐ Platform

- 2 sockets
- CPU: Intel Xeon Gold 5318Y (24 physical/48 logical cores)
- DRAM: 4 * 16 GB 2666 Mhz
- PM: 4 * 128 GB Intel Optane DCPMMs 200 series

☐ Target comparisons

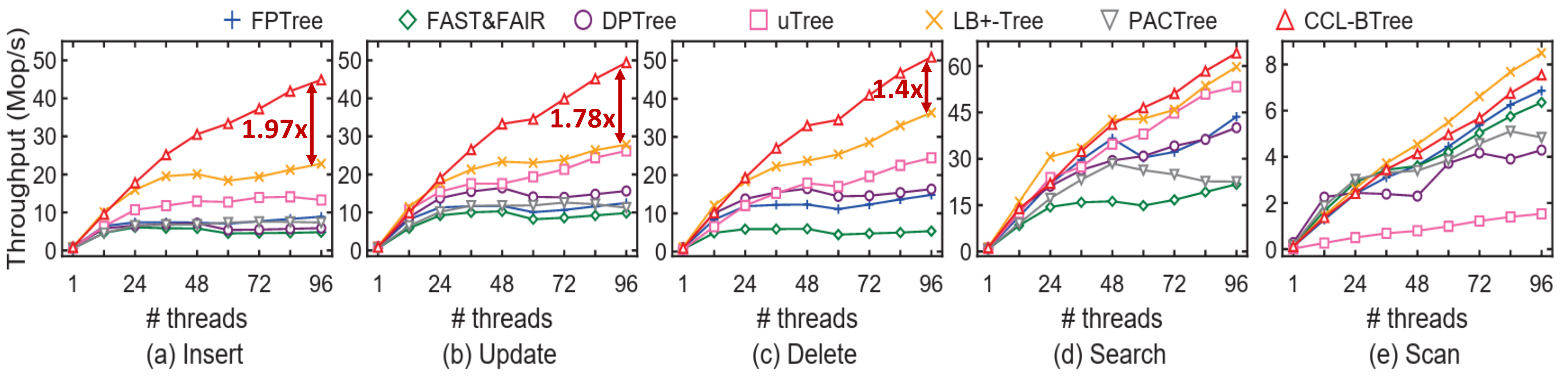
- FPTree [SIGMOD’16]
- FAST&FAIR [FAST’18]
- DPTree [VLDB’19]
- uTree [VLDB’20]
- LBTree [VLDB’20]
- PACTree [SOSP’21]

☐ Other settings

- Use the same 256 B tree node size for each index
- Use pre-allocated PM pools from the local socket for all indexes to minimize the allocation overhead

Overall performance

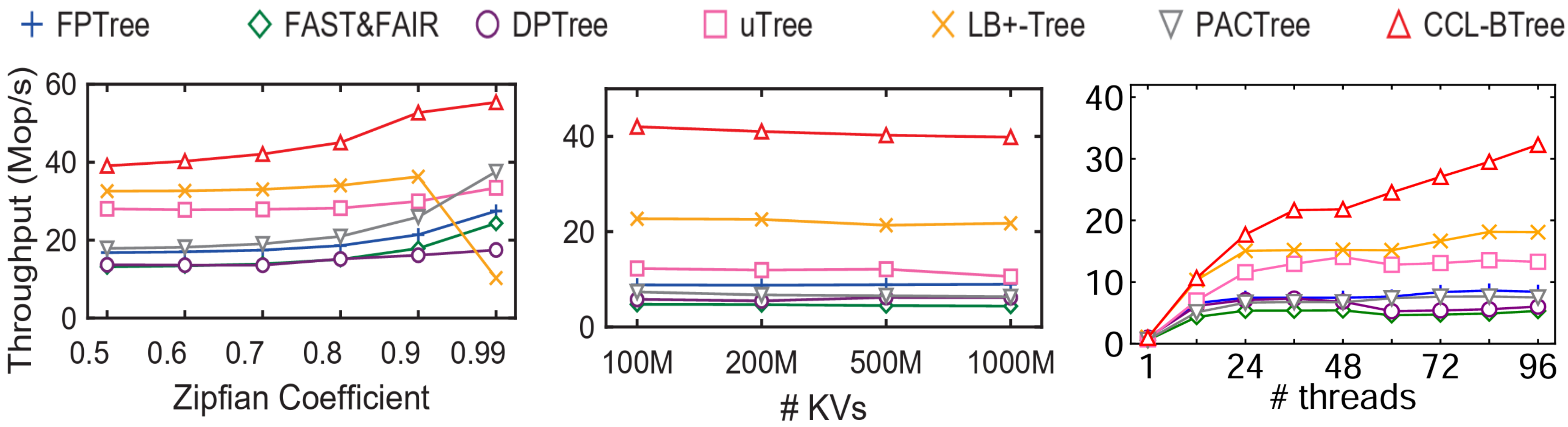
- ☐ 8-byte key and 8-byte value
- ☐ Warm up the index with 50 million KVs and then run each test with 50 million operations



☺ For write workloads, CCL-BTree outperforms other B+-Tree variants by 1.4x at least

☺ For read workloads, CCL-BTree has competitive performance

Other tests



(a) Skew test, 48 threads

(b) Various dataset sizes, 96 threads

(c) eADR test

☺ Demonstrate the high efficiency of CCL-Btree under various workloads

More details

- **System optimizations**
  - NUMA-friendly PM accesses
  - Concurrency control.
  - Variable-size KVs.

- **Evaluation results**
  - Comparison with persistent log structures
  - Realistic datasets test
  - Latency test
  - Recovery
  - …

Please check our paper!

CCL-BTree: A Crash-Consistent Locality-Aware B^+-Tree for Reducing XPBuffer-Induced Write Amplification in Persistent Memory

Zhenxin Li, Zhejiang University, Hangzhou, Zhejiang, China, zhenxin@zju.edu.cn
Shuibing He*, Zhejiang University, Hangzhou, Zhejiang, China, heshuibing@zju.edu.cn
Zheng Dang, Zhejiang University, Hangzhou, Zhejiang, China, dangzheng@zju.edu.cn
Peiyi Hong, Zhejiang University, Hangzhou, Zhejiang, China, hongpeiyi@zju.edu.cn
Xuechen Zhang, Washington State University, Vancouver, Washington, USA, xuechen.zhang@wsu.edu
Rui Wang, Zhejiang University, Hangzhou, Zhejiang, China, rwang21@zju.edu.cn
Fei Wu, Zhejiang University, Hangzhou, Zhejiang, China, wufei@zju.edu.cn

Abstract

In persistent B^+-Tree, random updates of small key-value (KV) pairs will cause severe XPBuffer-induced write amplification (XBI-amplification) because CPU cacheline size is smaller than media access granularity in persistent memory (PM). We observe that XBI-amplification directly determines the application performance when the PM bandwidth is exhausted in multi-thread scenarios. However, none of the existing work can efficiently address the XBI-amplification issue while maintaining superior range query performance.

In this paper, we design a novel crash-consistent locality-aware B^+-Tree (CCL-BTree). It preserves the key order between adjacent leaf nodes for efficient range query and proposes a *leaf-node centric buffering* strategy that merges writes and then flushes them together to reduce the number of flushes to the PM media. For crash-consistency, all the buffered KVs are recorded in write-ahead logs. CCL-BTree further devises *write-conservative logging* to skip unnecessary log operations, and *locality-aware garbage collection* to avoid random PM writes in reclaiming log data. Our experiments show that CCL-BTree reduces the XBI-amplification by up to 81%, improves the insert throughput by up to 9.35×, and achieves good range query performance compared to state-of-the-art persistent B^+-Trees.

CCS Concepts: • **Information systems → Data structures**; **Storage class memory**.

Keywords: Persistent Memory, Index Structures, B^+-Tree

ACM Reference Format:
Zhenxin Li, Shuibing He, Zheng Dang, Peiyi Hong, Xuechen Zhang, Rui Wang, and Fei Wu. 2024. CCL-BTree: A Crash-Consistent Locality-Aware B^+-Tree for Reducing XPBuffer-Induced Write Amplification in Persistent Memory. In *European Conference on Computer Systems (EuroSys '24), April 22–25, 2024, Athens, Greece.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3627703.3629582

1 Introduction

B^+-Tree is a widely used index structure in file systems and database systems, because of its better performance than other index structures, e.g., higher range query performance than hash table [34, 52] and radix tree [28, 32], and better data locality than skip list [10, 30]. Persistent B^+-Trees [7, 17, 31, 36, 51] that are built on the emerging byte-addressable persistent memory (PM) attracted wide attention in in-memory databases recently, for its low latency, large capacity, and data persistence.

Like DRAM, PM is packaged on the memory bus and handles requests at the 64 B cacheline granularity. For the convenience of illustration, Figure 1 uses the only commercially available PM, Intel Optane DCPMM [19], as an example to illustrate its architecture. Memory accesses to PM

*Shuibing He is the corresponding author.

EuroSys '24, April 22–25, 2024, Athens, Greece

ACM ISBN 979-8-4007-0437-6/24/04…$15.00
https://doi.org/10.1145/3627703.3629582

Conclusion

- We propose CCL-BTree to address the XPBuffer-induced write amplification issue in persistent B+-trees.
  - Leaf-node centric buffering
  - Write-conservative logging
  - Locality-aware garbage collection

- CCL-BTree improves the insert throughput by 1.97x to 9.35x

- The source code is available at https://github.com/ISCS-ZJU/CCL-BTree

CCL-BTree: A Crash-Consistent Locality-Aware B+-Tree for Reducing XPBuffer-Induced Write Amplification in Persistent Memory

Zhenxin Li , Shuibing He , Zheng Dang, Peiyi Hong ,
Xuechen Zhang [†], Rui Wang , Fei Wu

Thanks & QA

浙江大学
Zhejiang University

Washington State University

Open source: https://github.com/ISCS-ZJU/CCL-BTree

Contact information: zhenxin@zju.edu.cn