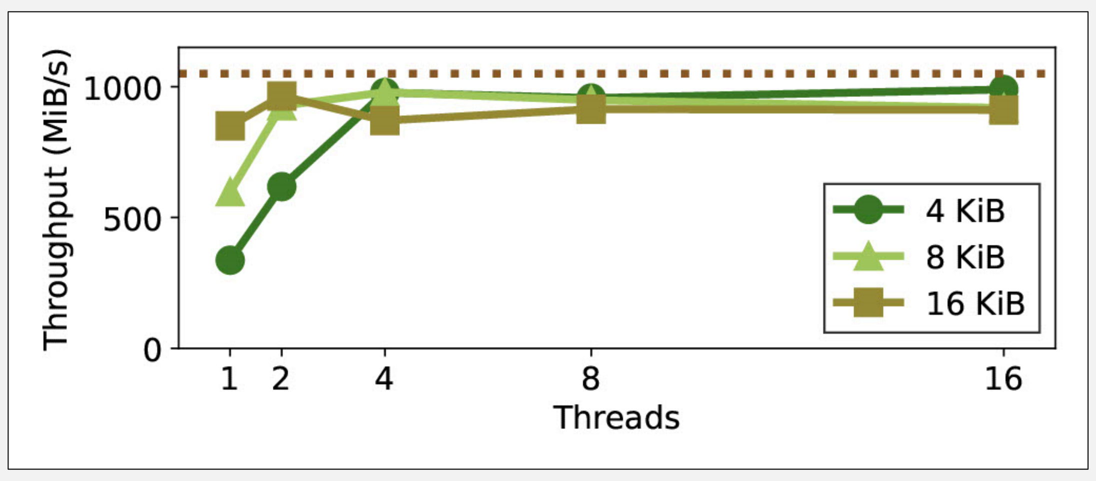
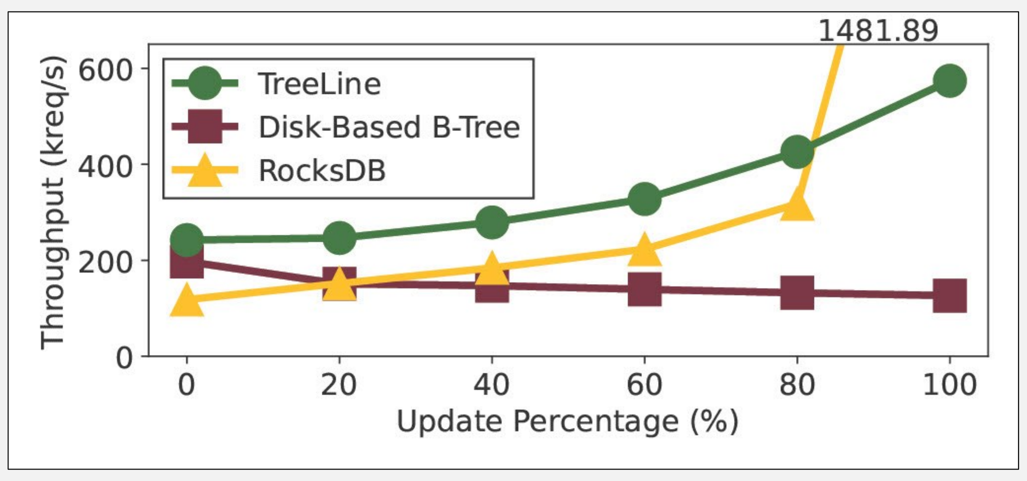
TreeLine: An Update-In-Place Key-Value Store for Modern Storage

Geoffrey Yu*, Markos Markakis*, Andreas Kipf*,
Per-Åke Larson, Umar Farooq Minhas, Tim Kraska

MIT CSAIL

Motivation





Both experiments on an Intel DC P4510 NVMe SSD.

- Modern persistent key-value stores, such as RocksDB [1] and LevelDB [2], typically use log-structured merge trees (LSMs) [3].
- Stellar write performance: large sequential writes exploit the high sequential throughput.
- Slow read performance: need caches, Bloom filters [4,5], compaction strategies [6,7,8]—complex and hard-to-tune. [9]

Random Writes ≈ Sequential Writes on NVMe SSDs

- Random write throughput across (i) request sizes, and (ii) number of writing threads.
- With high parallelism, can reach advertised peak sequential write throughput [10].

LSMs Leave Read Performance on the Table

- Zipfian-distributed (θ = 0.79) workload of reads, updates, and scans on 64 B records.
- Of the requests that are not updates, 10% are range scans and the rest are point reads.
- For read-heavy, the disk-based B-tree outperforms/is competitive against RocksDB.
- TreeLine can outperform both systems all the way up to 80% updates.

Design Overview

Key Idea A: Record Caching

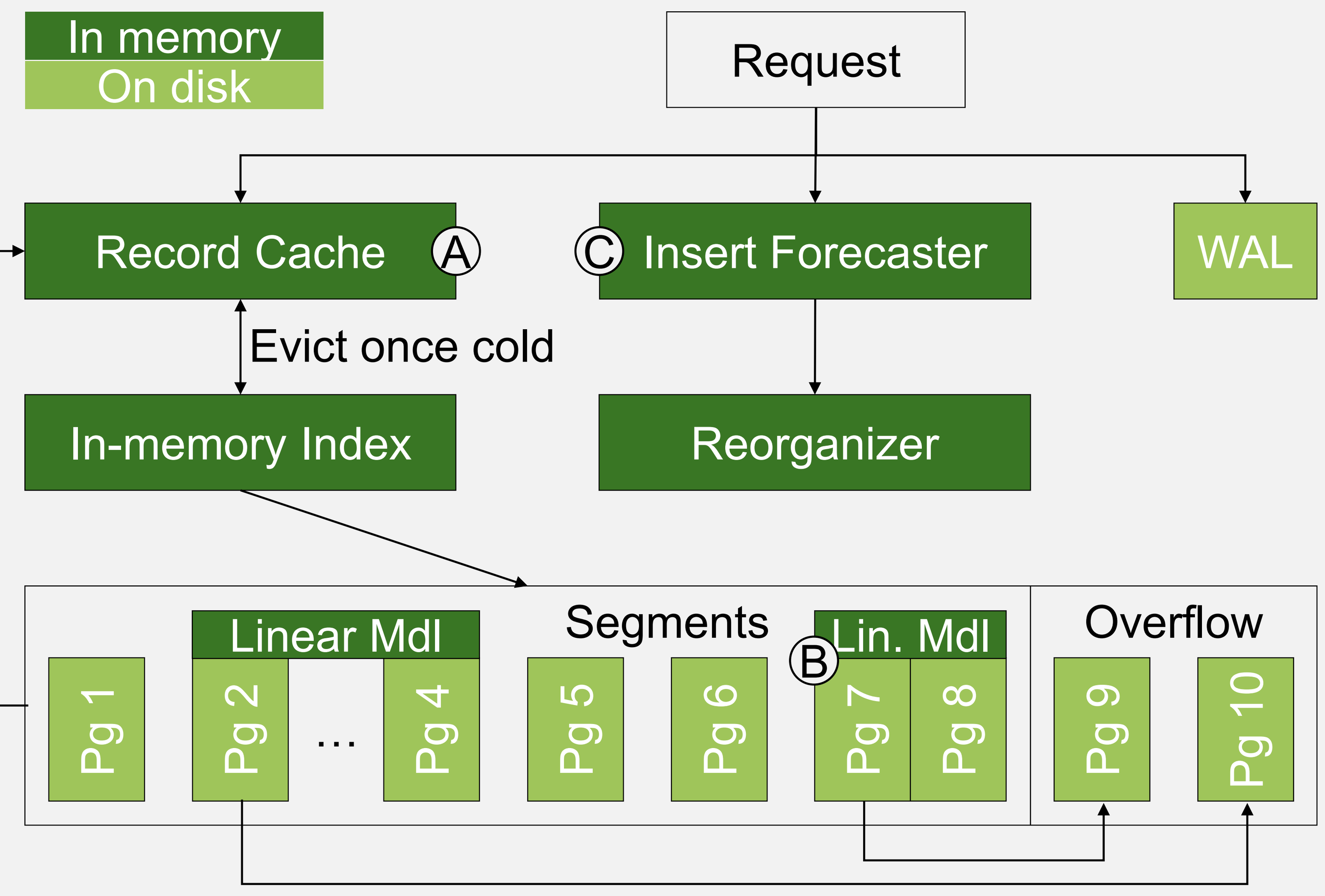
- Key-to-page mapping is expensive to change in update-in-place design.
- But variable hotness among records on the same page.
- **Solution:** Only cache records, to increase memory efficiency.

Key Idea B: Page Grouping

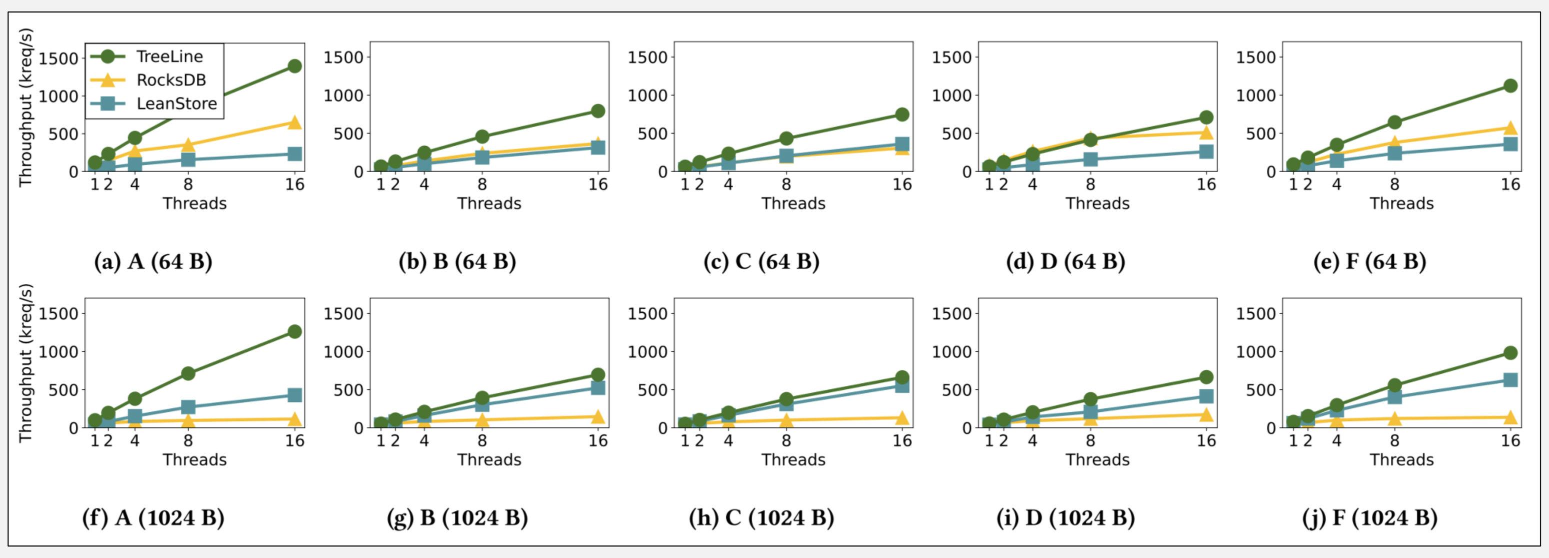
- Unlike writes, sequential reads still faster vs. random reads on modern SSDs.
- Pages must be sequential to make scans fast.
- **Solution:** Group pages into contiguous segments. Use linear models to index records within segments.

Key Idea C: Insert Forecasting

- To avoid constant reorganization, pages should have some empty space.
- But too much empty space increases I/O amplification.
- **Solution:** Leave empty space based on epoch-based insert forecast.

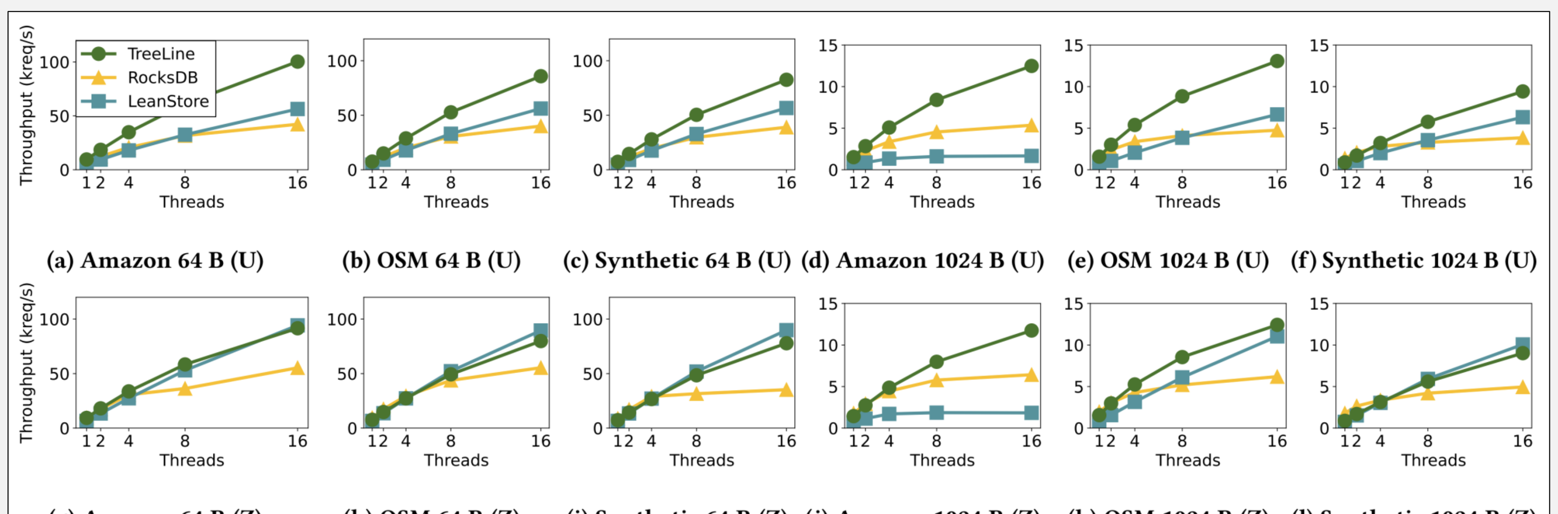


Evaluation



(a) A (64 B) (b) B (64 B) (c) C (64 B) (d) D (64 B) (e) F (64 B)

(f) A (1024 B) (g) B (1024 B) (h) C (1024 B) (i) D (1024 B) (j) F (1024 B)



(a) Amazon 64 B (U) (b) OSM 64 B (U) (c) Synthetic 64 B (U) (d) Amazon 1024 B (U) (e) OSM 1024 B (U) (f) Synthetic 1024 B (U)

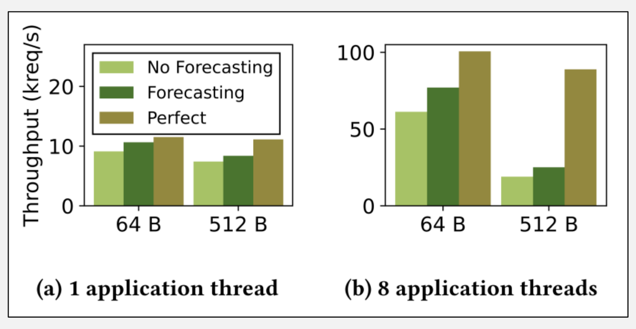
(g) Amazon 64 B (Z) (h) OSM 64 B (Z) (i) Synthetic 64 B (Z) (j) Amazon 1024 B (Z) (k) OSM 1024 B (Z) (l) Synthetic 1024 B (Z)

Point Workloads

- TreeLine outperforms RocksDB (LeanStore) by **1.62× (2.81×)** and **2.99× (1.53×)** on average for **64 B**/**1024 B** records.

Scans

- With 16 request threads, for uniform scans, TreeLine outperforms RocksDB (LeanStore) by **2.21× (1.58×)** and **2.50× (2.80×)** on average for **64 B**/**1024 B** records.
- With 16 request threads, for Zipfian-distributed scans, TreeLine outperforms RocksDB (LeanStore) by **1.74× (0.91×)** and **1.88× (1.86×)** on average for **64 B**/**1024 B** records.



(a) 1 application thread (b) 8 application threads

[1] Facebook, Inc. 2021. RocksDB. https://rocksdb.org.
[2] Google, Inc. 2011. LevelDB https://github.com/google/leveldb.
[3] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. 1996. The log-structured merge-tree (LSM-tree). Acta Informatica 33, 4 (1996), 351–385.
[4] Burton H. Bloom. 1970. Space/Time Trade-Offs in Hash Coding with Allowable Errors. Commun. ACM 13, 7 (jul 1970), 422–426. https://doi.org/10.1145/362686.362692
[5] Peter C. Dillinger and Stefan Walzer. 2021. Ribbon filter: practically smaller than Bloom and Xor. CoRR abs/2103.02515 (2021). arXiv:2103.02515 https://arxiv.org/abs/2103.02515.
[6] Mark Callaghan. 2018. Name that compaction algorithm. https://smalldatum.blogspot.com/2018/08/name-that-compaction-algorithm.html.
[7] Niv Dayan, Manos Athanassoulis, and Stratos Idreos. 2017. Monkey: Optimal Navigable Key-Value Store. In Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 79–94. https://dl.acm.org/doi/10.1145/3035918.3064054.
[8] Facebook, Inc. 2021. Universal Compaction. https://github.com/facebook/rocksdb/wiki/Universal-Compaction.
[9] Facebook, Inc. 2020. RocksDB Tuning Guide. https://github.com/facebook/rocksdb/wiki/RocksDB-Tuning-Guide.
[10] Intel Corporation. 2017. Intel DC P4510. https://ark.intel.com/content/www/us/en/ark/products/122573/intel-ssd-dc-p4510-series-1-0tb-2-5in-pcie-3-1-x4-3d2-tlc.html.