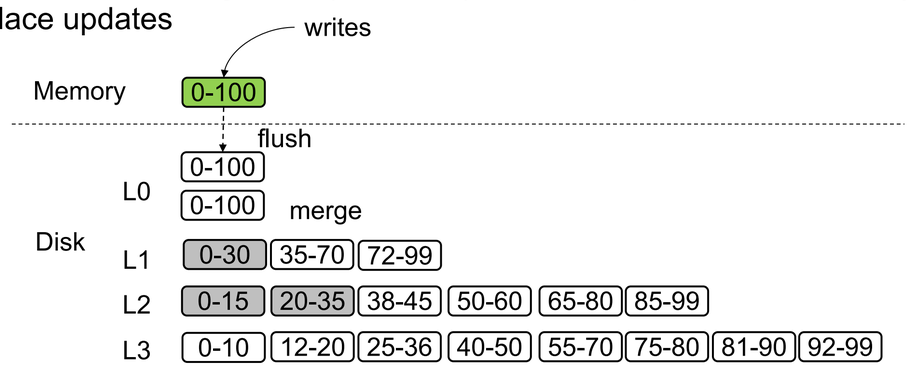
Breaking Down Memory Walls in LSM-based Storage Systems

Chen Luo
University of California Irvine
cluo8@uci.edu
Advisor: Michael J. Carey

Background

- The Log-Structured Merge-tree (LSM-tree) is a write-optimized structure by applying out-of-place updates

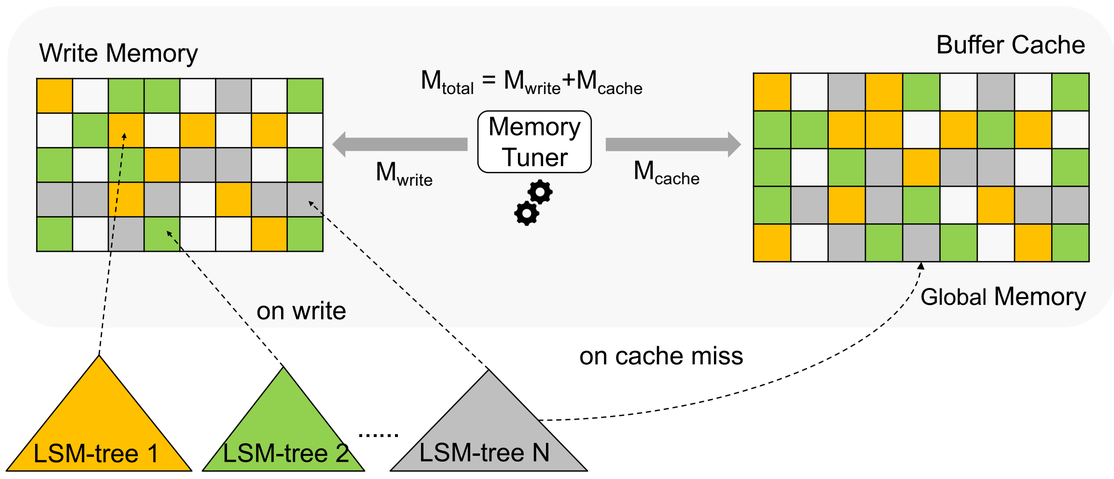


Memory Walls in LSM-trees

- Efficient memory management is critical for storage systems to achieve high performance
- Compared to update-in-place systems, LSM-trees have introduced additional memory walls:
  - Wall 1: static size limit on each LSM memory component
  - Wall 2: no efficient sharing of write memory among multiple LSM-trees
  - Wall 3: static memory allocation between write memory and buffer cache
- Must these break down these memory walls for optimal performance!

Adaptive Memory Management Architecture

- Write Memory: store incoming writes for all LSM-trees
  - All memory components are managed through a shared memory pool
  - Pages are allocated on-demand when an LSM-tree has insufficient memory to store writes
- Buffer Cache: provides caching of immutable on-disk data
  - All disk pages are managed using a predefined cache replacement policy
- Memory Tuner: tunes the memory allocation between Write Memory and Buffer Cache
  - Minimize the overall I/O cost



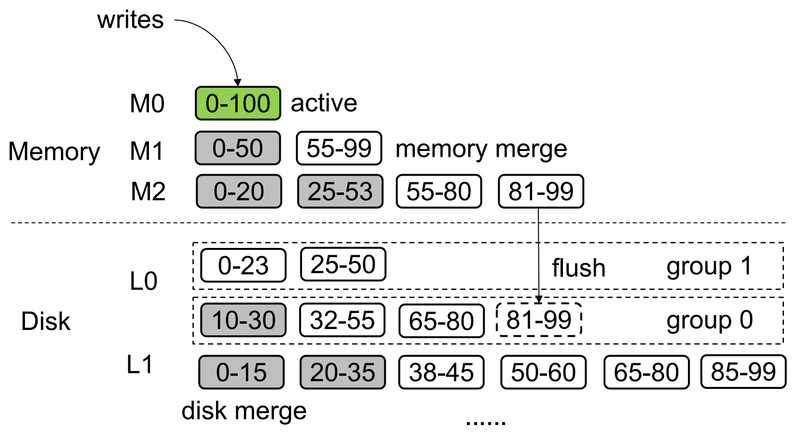
Manage Write Memory

Design Goals

- Write memory directly impacts the merge cost of LSM-trees
  - Important to utilize write memory efficiently for optimal performance
- Question 1: how to maximize the write memory utilization for a single LSM-tree?
  - Existing LSM-tree implementations use B+-trees or skiplists to store incoming writes
  - Low memory utilization because of (1) internal fragmentation and (2) freeing a large chunk of write memory at once
- Question 2: how to allocate write memory to multiple LSM-trees efficiently?
  - These LSM-trees are highly heterogenous with different sizes and write rates

Manage a Single LSM-tree

- To maximize the write memory utilization, we propose to use an in-memory LSM-tree to store incoming writes
  - LSM-trees achieve very high space utilization
  - Trade CPU cycles for minimizing merge I/O cost
- Flushes are performed on a continuous-basis and memory utilization stays high



Manage Multiple LSM-trees

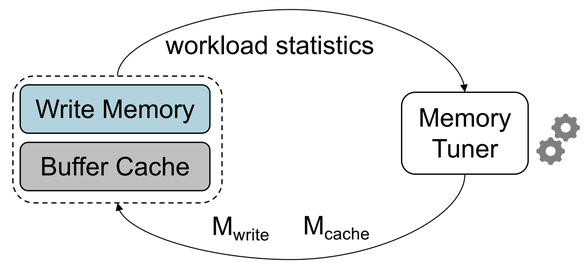
- Write memory is always allocated on-demand to an LSM-tree based on incoming writes
  - When write memory is full, an LSM-tree must be selected for flushing
- Existing LSM-tree implementations use a max-memory flush policy
  - Flush the LSM-tree with the largest memory component
  - Intuition: reclaim the most write memory for future writes
- However, the max-memory flush policy does not work well with the proposed memory component structure
  - Flushing any LSM-tree will reclaim the same amount of memory due to partial flushes
- We propose a min-LSN flush policy that always flushes the LSM-tree with the minimum LSN
  - The flush rate of an LSM-tree will be approximately proportional to its write rate

Memory Tuner

- Given a total memory budget M_{total}, the tuning goal is to find optimal M_{write} and M_{cache} to minimize the weighted I/O cost per operation

$$w \cdot I/O_{write} + r \cdot I/O_{read}$$

- The weights w and r allow the tuning goal to be configured for different use cases
  - On hard disks, w can be set smaller since LSM-trees mainly use sequential I/Os for writes
  - On SSDs, w can be set larger since SSD writes are often more expensive than SSD reads

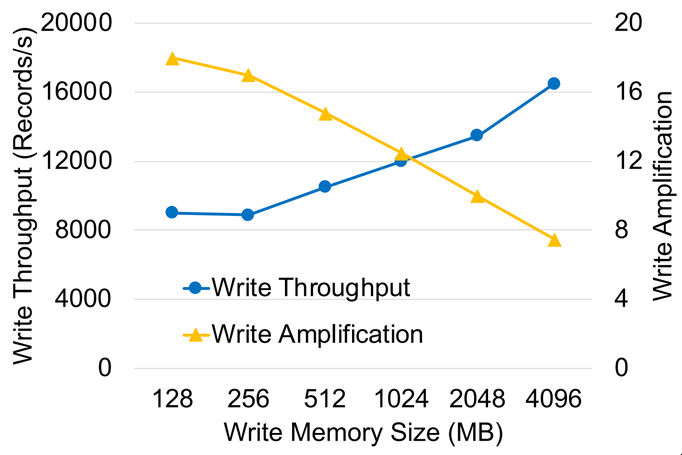


Preliminary Evaluation

Workload

- YCSB benchmark with write-only workloads
- Single LSM-tree with 100M records

Large write memory significantly improves write throughput and reduces write amplification!



Conclusion

- In this work, we break down memory walls of LSM-trees for optimal performance
  - Introduced a new memory management architecture for adaptive memory management
  - Proposed a new memory component structure to better utilize write memory
- Check out our full paper "Breaking Down Memory Walls: Adaptive Memory Management in LSM-based Storage Systems"
  - Full design and implementation of the memory tuner
  - Extensive evaluation on YCSB and TPC-C benchmarks
  - Preprint available at: https://arxiv.org/abs/2004.10360