# OrderMergeDedup: Efficient, Failure-Consistent Deduplication on Flash

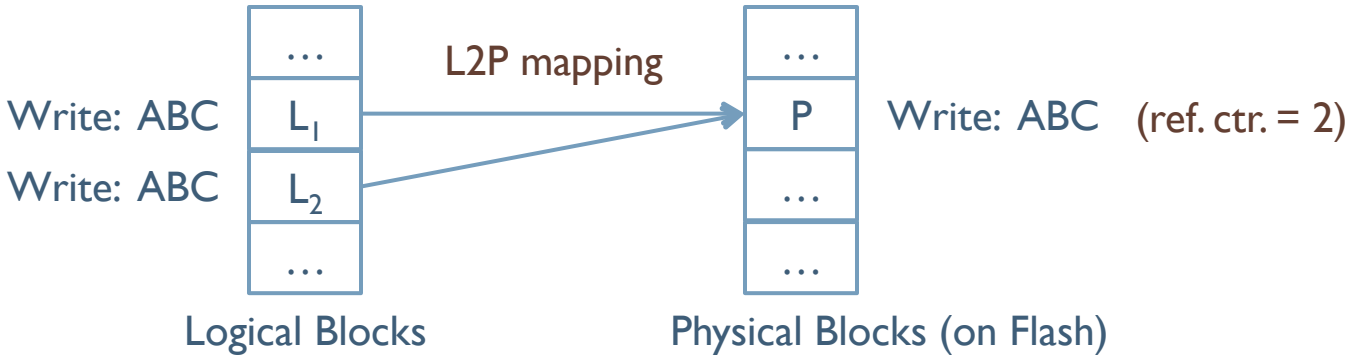
Zhuan Chen and Kai Shen
*University of Rochester*

# Background

- I/O deduplication
  - Eliminate I/O writes with redundant content
  - Reduce the storage space usage
  - Write reduction: reduce the Flash wear, improve performance
  - Broad usage in data centers, personal computers, data-driven sensing

# Motivation

- I/O deduplication is not free: metadata maintenance
  - (1) Logical-physical block mapping
    (2) Physical block fingerprints
    (3) Physical block reference counts
  - Need to maintain failure-consistency for data and metadata



# Challenge

- Existing approaches for failure-consistency
  - Rely on non-volatile RAM or supercapacitors/batteries [Srinivasan etal. 2012; Chen et al. 2011; Gupta et al. 2011]
  - Checking/repair tools [Quinlan et al. 2002]
  - Redo logging [Meister et al. 2010] (additional I/O for logging writes)
  - Shadowing [Tarasov et al. 2014] (additional I/O for index block writes)
- Challenge: metadata & failure-consistency-induced I/O cost shouldn't significantly diminish the deduplication I/O saving
- We look into soft updates-style I/O ordering

# I/O Ordering for Failure-Consistency

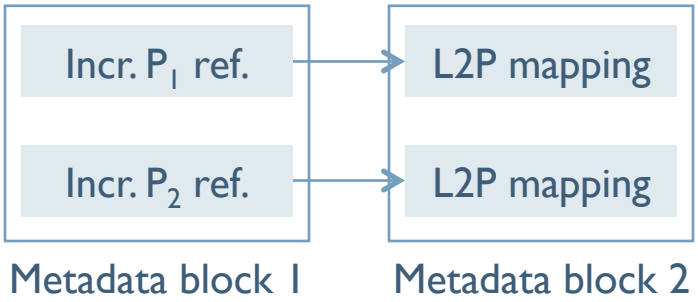
- Define an order for data/metadata writes
  - Ordered writes are committed one by one
  - A failure still keeps a deduplication system consistent
  - A failure can only leave garbage (which can be reclaimed asynchronously)

Example: new write (duplicated content)

# I/O Ordering for Failure-Consistency

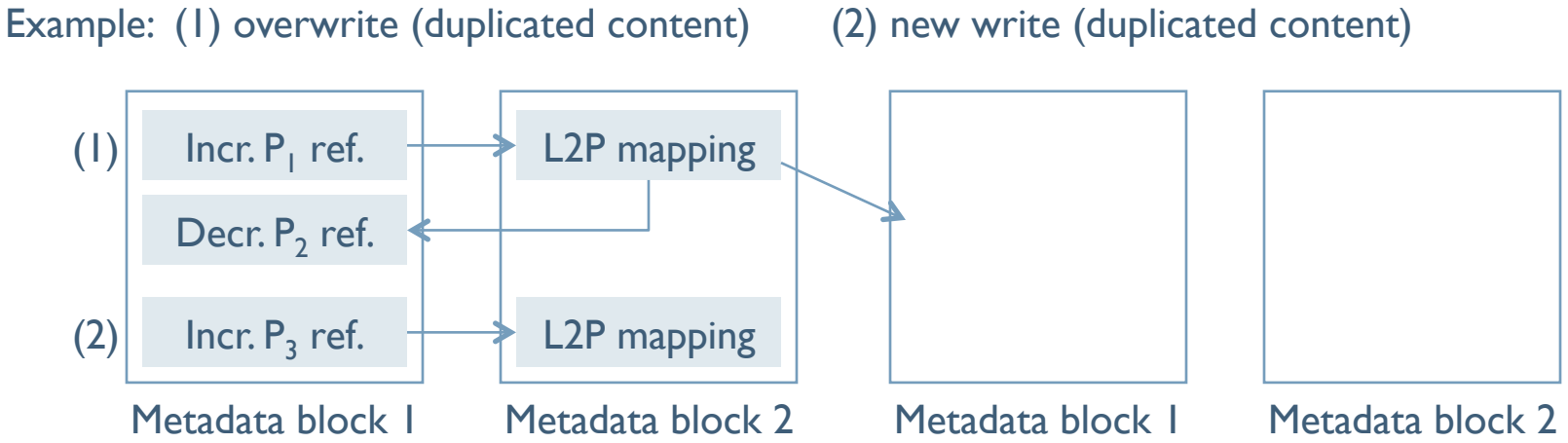
- I/O efficiency
  - No consistency-induced additional I/O
  - We can merge metadata writes residing on the same metadata block as long as they are not subject to any ordering constraint

Example: new write (duplicated content)



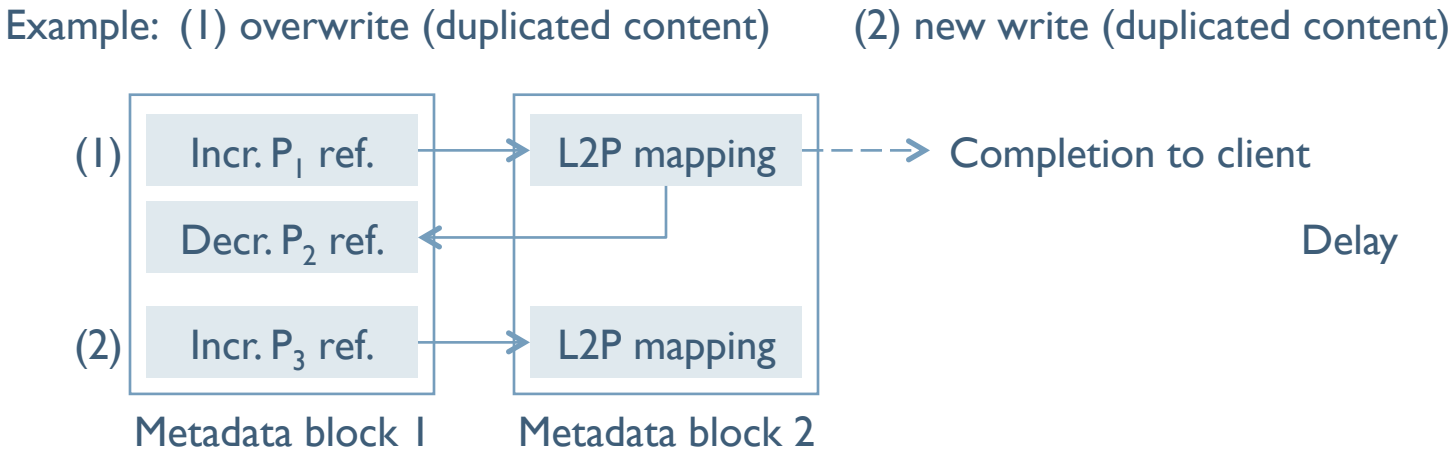
# I/O Ordering for Failure-Consistency

- Cyclic dependencies
  - Prevent metadata I/O merging & complicate the implementation
  - Make soft updates costly for file systems [Seltzer et al. 2000]



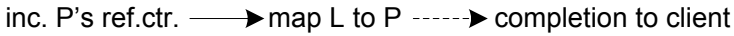
# I/O Ordering for Failure-Consistency

- Resolve cyclic dependencies
  - We carefully design all deduplication I/O paths
  - Delay non-critical metadata I/O (the completion signal doesn't depend on)

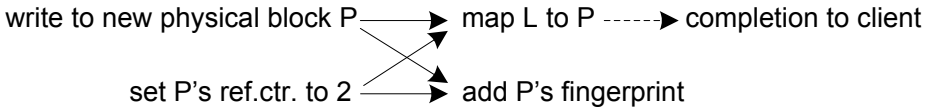


# I/O Ordering for Failure-Consistency

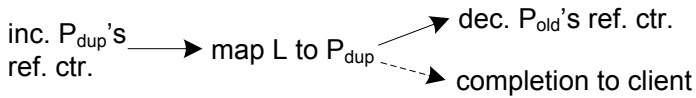
1. Write new block L; duplicating existing physical block P



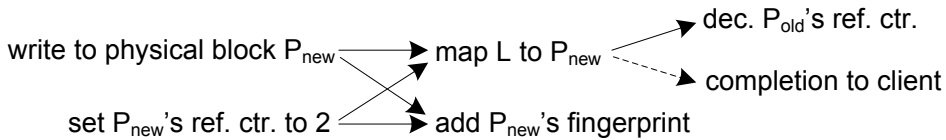

2. Write new block L; no duplicate




3. Overwrite block L mapped to physical block $P_{old}$; duplicating physical block $P_{dup}$




4. Overwrite block L mapped to physical block $P_{old}$; no duplicate



# Metadata I/O Merging for Efficiency

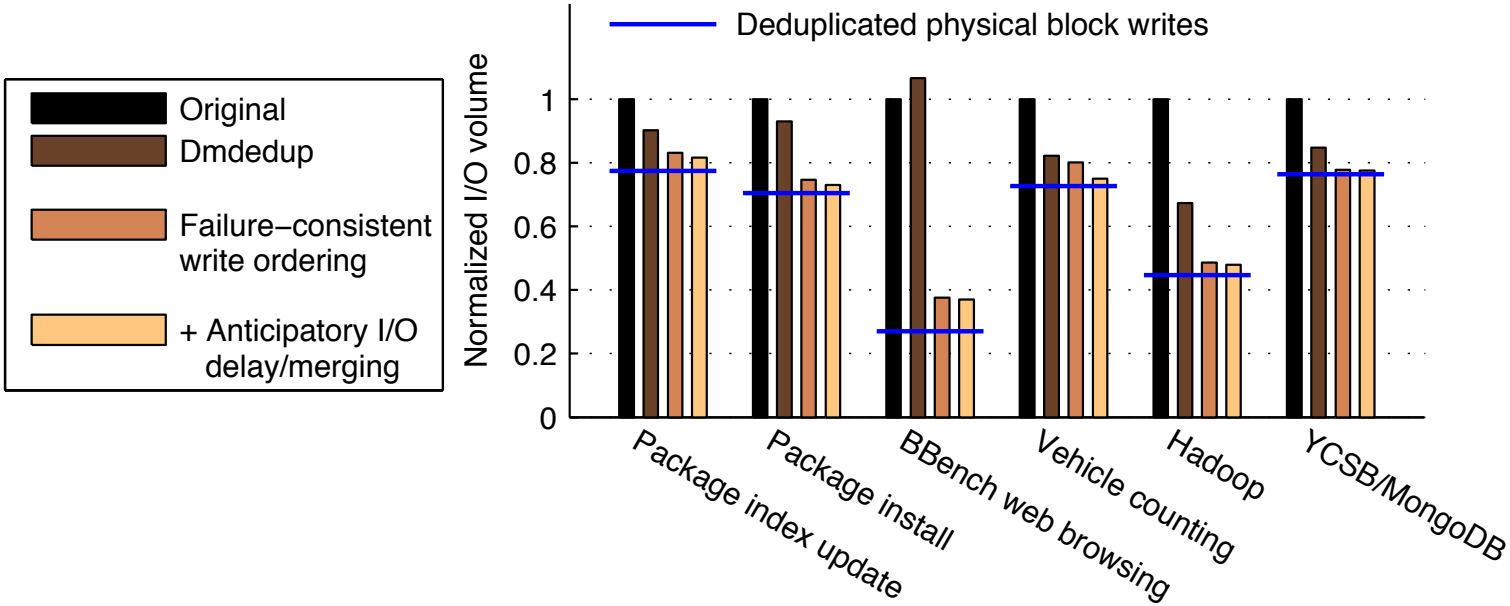
- Anticipatory I/O delay and merging
  - Delay a metadata write in anticipation for near-future merging opportunities
  - Limited delay duration (e.g., 1 millisecond), slight performance impact

- We name our approach *OrderMergeDedup*

# Evaluation Setup

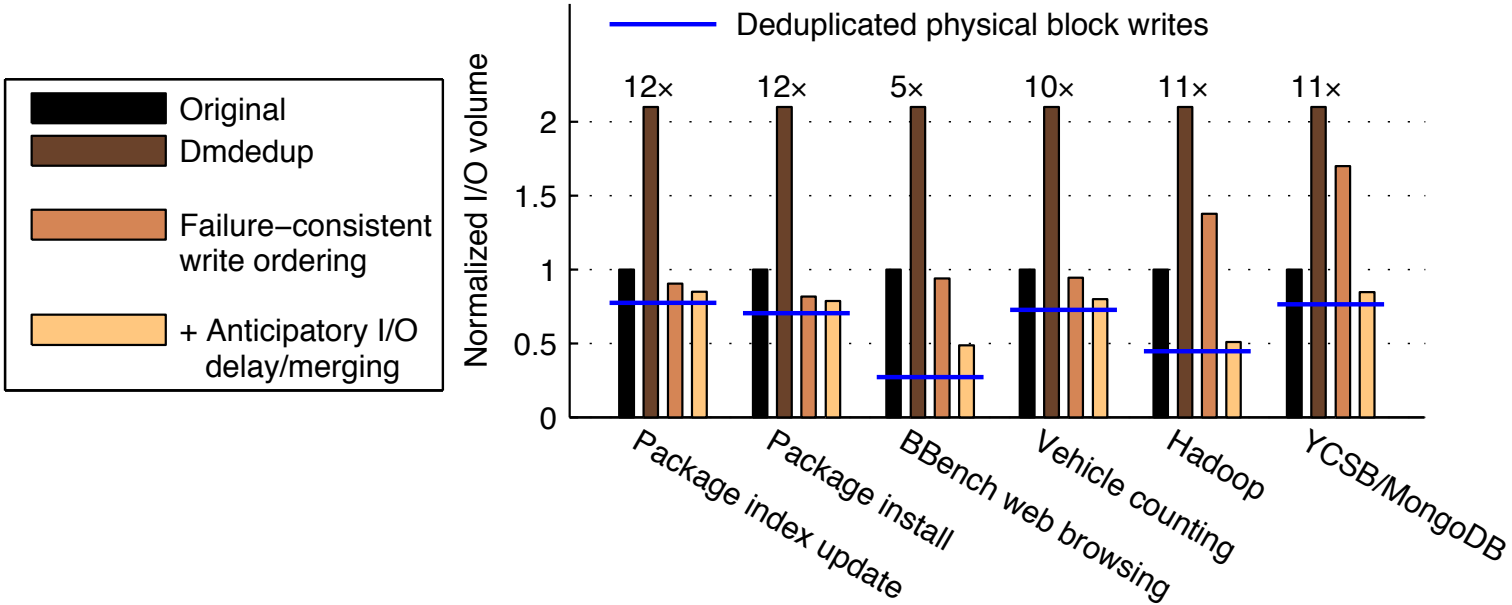
- Prototype of OrderMergeDedup
  - A custom device mapper target of Linux 3.14.29
- Mobile system workloads (Atom-based tablet)
  - Ubuntu package update & installation
  - BBench web browsing
  - Vehicle counting for intelligent traffic sensing
- Server system workloads (Xeon-based server machine)
  - Hadoop
  - YCSB/MongoDB

# Evaluation




- We save 18-63% I/O writes (on workloads with 23-73% write duplication)

# Evaluation (Strong Persistence Model)




- We save 15-51% I/O writes (on workloads with 23-73% write duplication)
- Anticipatory I/O delay/merging is particularly effective

# Conclusion

- OrderMergeDedup
  - Efficient, failure-consistent I/O deduplication on Flash
  - A soft updates-style data/metadata write ordering for failure-consistency (in particular, we resolve all possible cyclic dependencies with carefully designed I/O ordering and by delaying non-critical metadata writes)
  - Anticipatory I/O delay and merging to further reduce metadata I/O writes
  - We save 18-63% I/O writes (on workloads with 23-73% write duplication)
  - Anticipatory I/O delay/merging is particularly effective under the strong persistence model