Block-level Inline Data Deduplication in ext3

Aaron Brown
Kris Kosmatka

University of Wisconsin - Madison
Department of Computer Sciences

December 18, 2010

Outline

Why SSDs?

Solid state media increasingly used as primary storage, particularly for mobile devices. What is driving this trend?

Pros

- Low power usage
- Resistant to shock/vibration
- Quiet
- Performance! Fast random access

Cons

- Expensive (for now)
- Small (for now)
- Limited life span (read-write cycles)

SSD Technical Details

- Based on NAND flash, non-volatile
- Physical I/O units
  - Page - reads & writes (2-4 KB)
  - Block - erase (64-128 pages)
- Cannot overwrite single page: erase whole block then rewrite page
- Thus, writes much slower than reads
- Limited erase-write cycles, wear leveling required

Flash Translation Layer

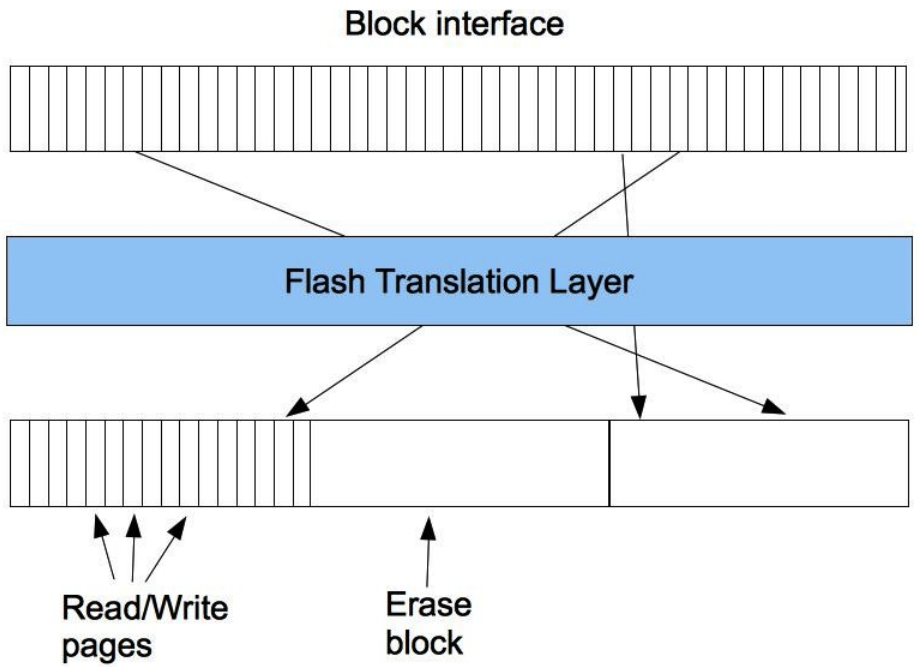


Figure: SSD Vendors present a block interface to the OS. Page allocation, placement, wear leveling, error correction etc. are handled by the FTL. Implementation details of the FTL vary widely by vendor and are often not published.

Outline

The SSD Write Problem

Writes are painful

- relatively slow writes compared to reads
- limited number of erase-write cycles over lifetime of device
- cost induced space constraints

Can we leverage the filesystems we already have to use space more efficiently and limit writes on this new class of hardware?

Outline

The Dedupfs Solution

Reduce overall space usage and number of writes required by identifying opportunities for deduplication on the fly.

- Add dedup layer to existing ext3
- Check for duplicates on block write
- If duplicate found simply point to it and avoid write
- Before deleting check if the block has remaining references

The hash cache

In-memory cache of mappings from data block hash values to block numbers.

- Hash function selecteable at mount time
- Cache size selectable at mount time
- Treated as a hint not ground truth
  - On a hit perform full bytewise comparison
  - Mappings not complete, may miss some dedup opportunities
- Replacement policy, approximate LRU using clock-like algorithm
- Not part of on-disk data structures

Reference counts

Mapping from block numbers to number of currently active references to it.

- On a delete, block only freed if its ref count is zero
- Stored as a unix file with a regular inode
- Counts are persistent, survives remounts/reboots/crashes
- Journaled along with other metadata
- backward compatibility

ext3 write flow

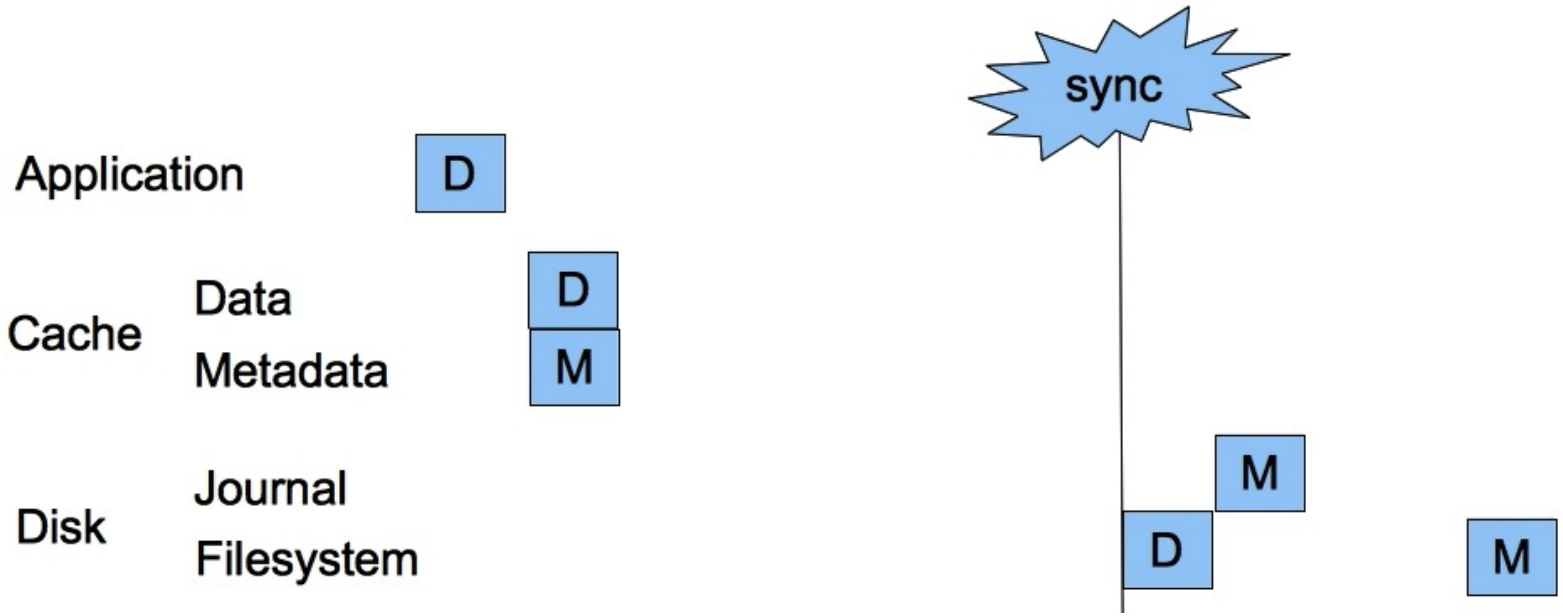


Figure: The normal flow of events when writing a block in ext3. Important: ordered journaling promises that the data block must be on stable storage before the journal transaction is committed. Metadata may be written to its final location some time later.

Dedupfs write flow

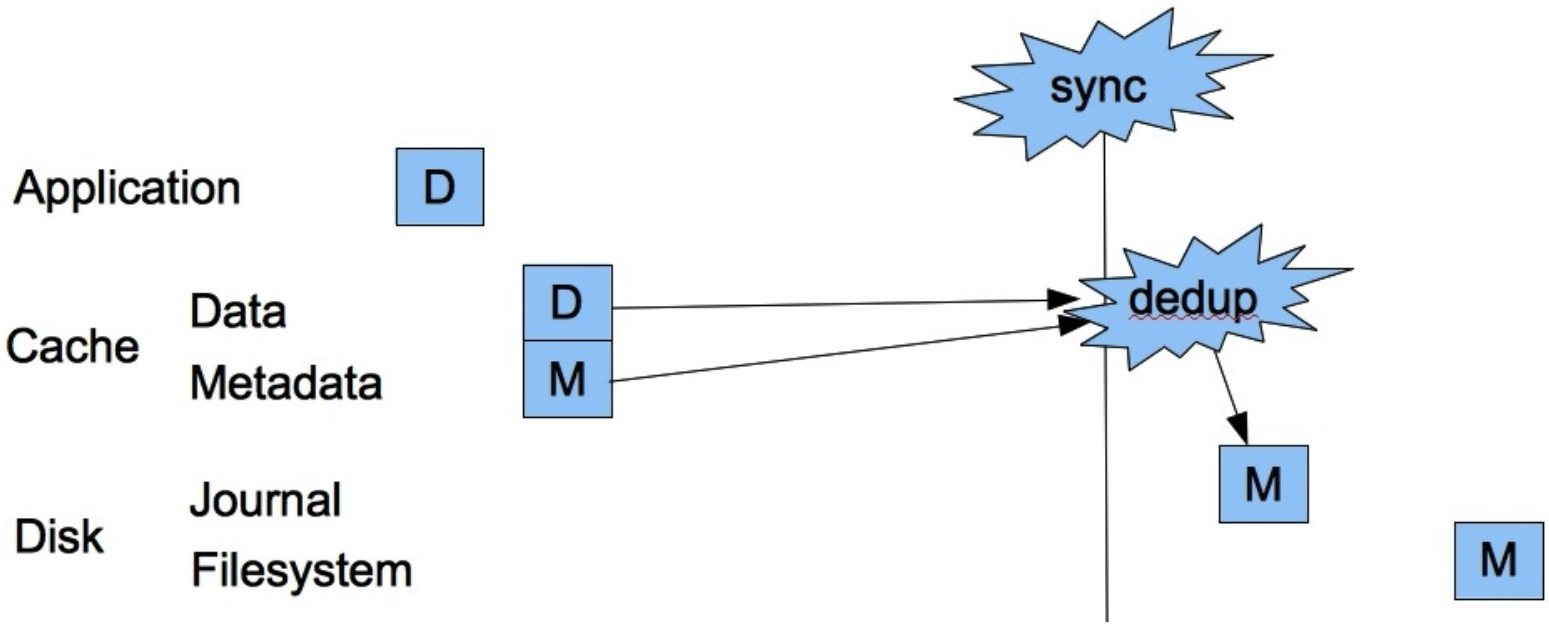
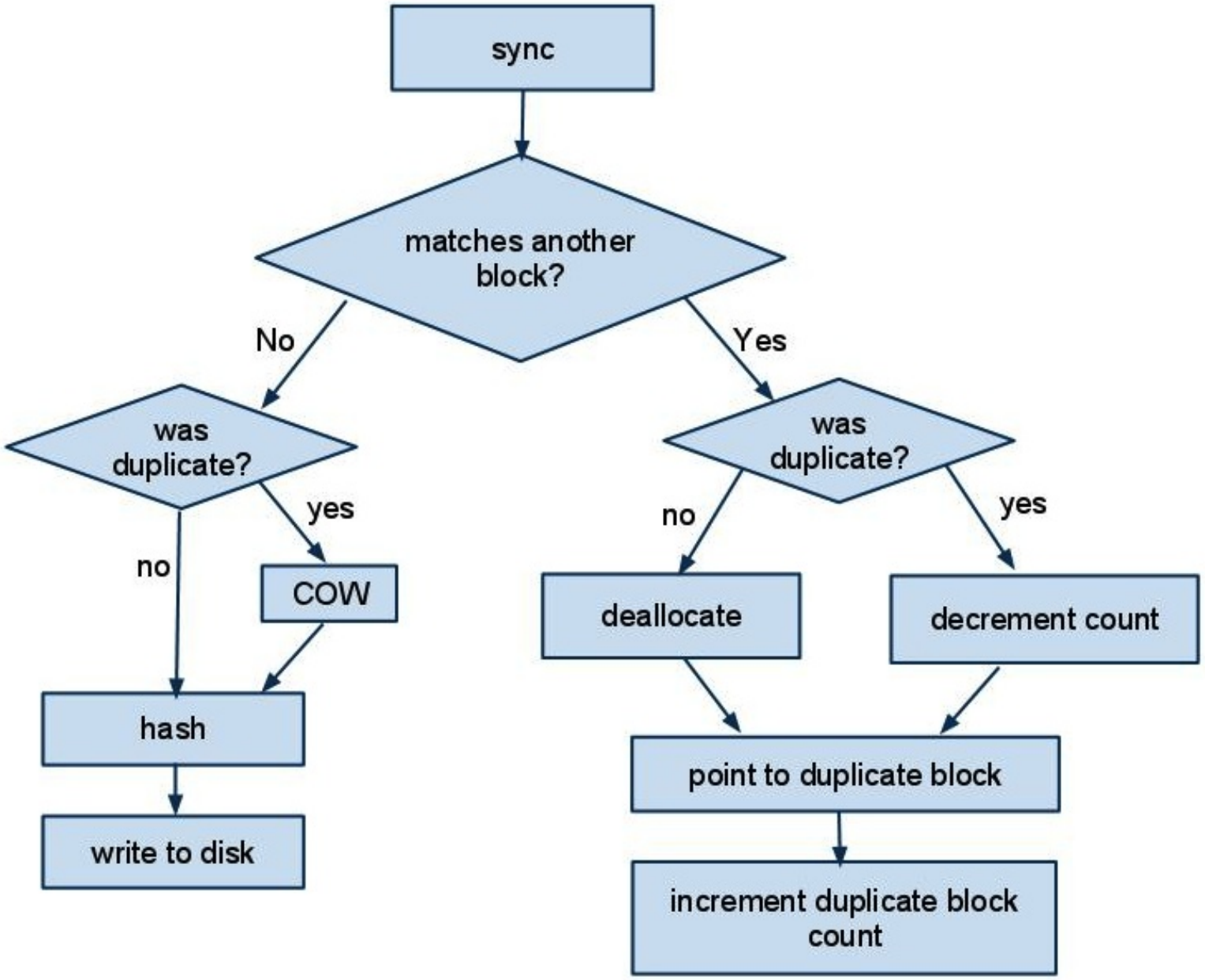


Figure: Dedupfs intercepts the flush of a block to disk at the last possible opportunity. Check the hashcache for possible duplicates. If a match is found and the bytewise comparison succeeds then the metadata is updated in the journal and the write is cancelled. Metadata makes its way to final disk location as usual.

Dedupfs write flow



Outline

Performance measurement setup

- Minimal Debian distribution compiled with Dedupfs support
- Tests run in User Mode Linux on a dual processor Pentium 4 3.2GHz with 2 GB of RAM
- Filesystem image a Unix file on the host system
- Tests each repeated 10 times

Overhead

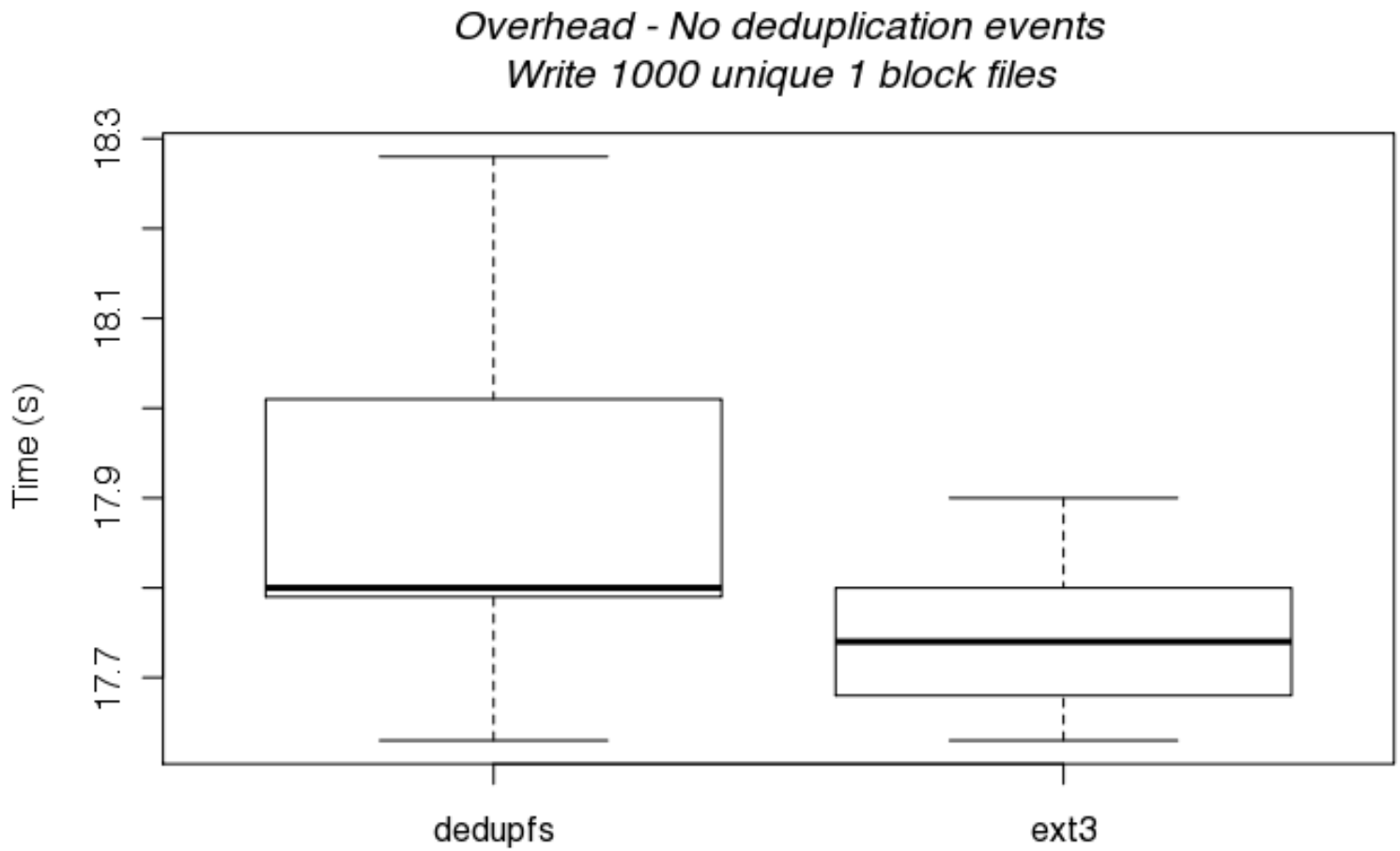


Figure: A purely non duplicate workload. Write 1000 one block files each with unique data, sync, and delete all files.

Deduplication Performance

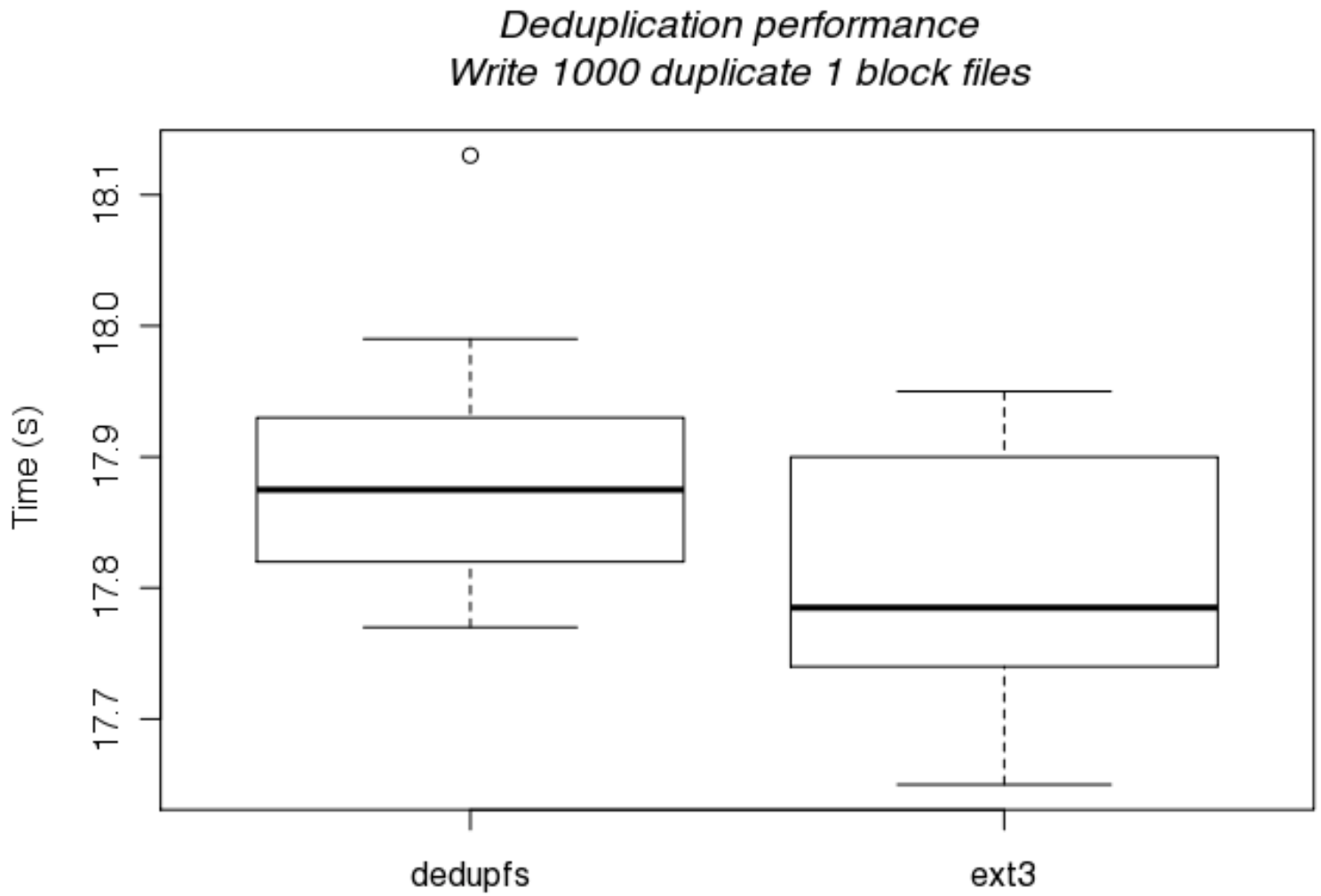


Figure: An ideal workload for deduplication. Write 1000 identical one block files, sync, and delete all files.

Recursive copy performance

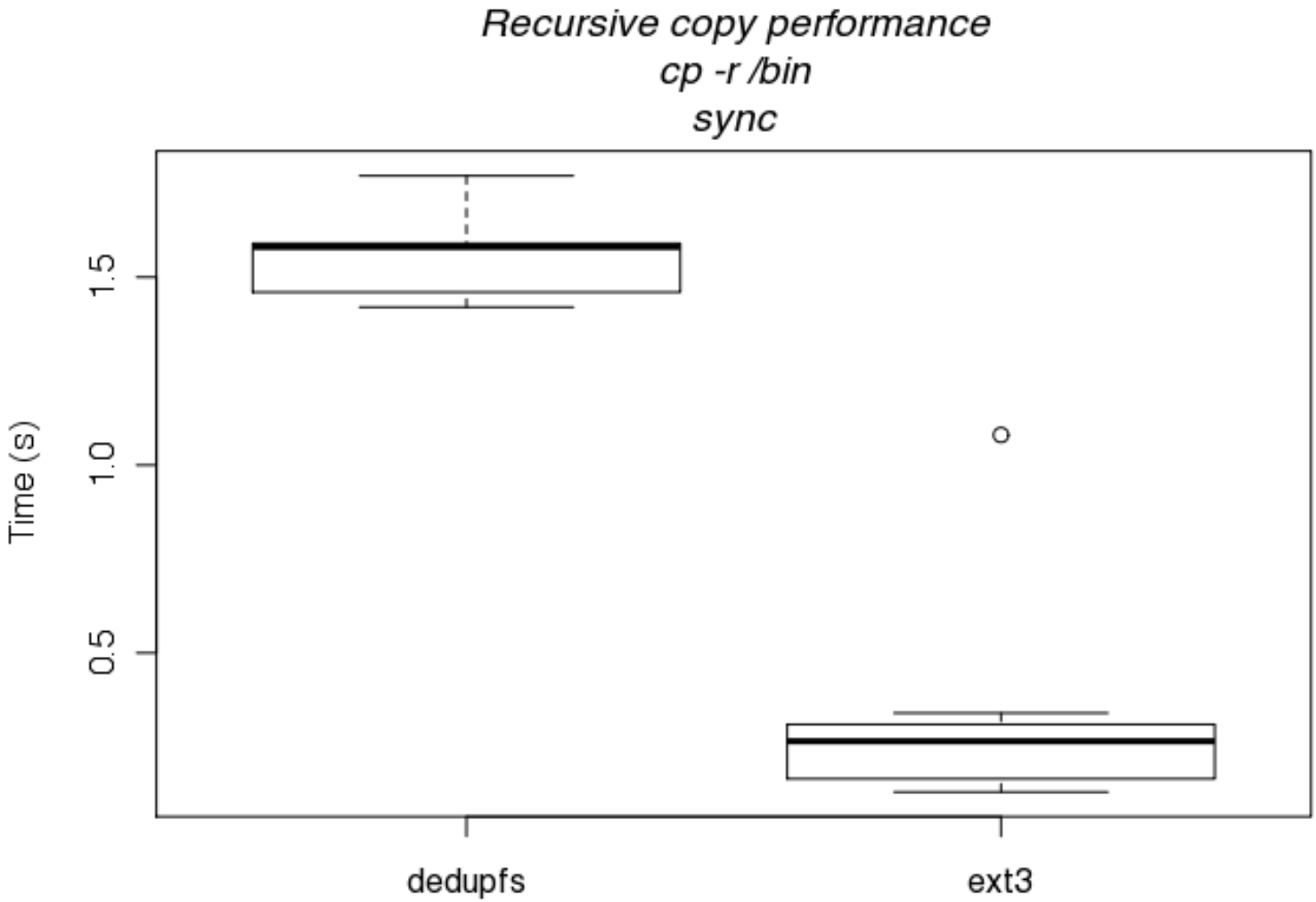


Figure: A real world workload. Recursively copy the the /bin directory and all its contents to Dedupfs, sync, delete all files.

Opportunities for deduplication

The extent of duplicate data on a system varies widely by workload. Dedupfs was used to identify latent duplication within several directories common to *nix platforms

directory	size (MB)	duplicate blocks
/bin	3.5	147
/lib	6.4	53
/usr/bin	14.0	0
/usr/lib	29.0	139

Outline

Dedupfs Features

- Cache of block hash hint mappings
- Block reference counts, persitent and journaled
- Latest possible time for write process interposition
- Low overhead for varying workloads
- Latent deduplication opportunities exist

Outline

Deduplication for SSDs

- Viable method for efficient use of limited space
- Reduces erase-writes cycles
- Extends usable life of device

Dedupfs in ext3

- small addition to code base
- backward (and forward) compatibility

Thanks for Coming!

Our thanks to Remzi for his valuable guidance and advice