Snapshots in a Flash with ioSnap™
Sriram Subramanian, Swami Sundararaman, Nisha Talagala, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau
Presented By: Samer Al-Kiswany
Snapshots Overview

- Point-in-time representation of the state of a storage device
- Snapshots are primarily used for backup, disaster recovery
  - Creates/deletes are common operations
  - Accesses/activates are rare operations
- Use Copy-on-Write (CoW) or Redirect-on-Write (RoW)
Why Rethink Snapshots for Flash?

• Flash is revolutionizing storage systems
  • Accelerating data centers, enterprise apps, desktops

• Natural fit for supporting snapshots
  • **Redirect-on-Write**: never overwrite existing data
  • **Log-structuring**: data ordered on their creation time (almost)

• Rate of data-change is higher for Flash devices
  • e.g., multi-threaded 8KB IOs, TB capacity device
    • Flash: 30K IOPS and device fills in ~1 hour
    • HDD: 500 IOPS and device fills in ~3 days
Snapshots in Flash Challenges

• Users are sensitive to performance variability
• Need predictable performance all the time

• NAND flash has low endurance & inefficient in-place writes
  • In-place updates of reference counts not possible

• Cannot waste storage space for storing snapshot metadata
• $/GB is high and need to keep costs low
ioSnap™ Overview

- First flash-aware snapshotting system
- Leverages RoW in FTL to support snapshot operations
- Supports large number of writable snapshots ($2^{16}$)
- Proposes usage of epochs in FTL to maintain log-time ordering
- Embraces rate-limiting to minimize performance implications

Performance Results (prototype built into Fusion-io VSL driver)
- Instantaneous snapshot creation/deletion (~50usec, 4k metadata)
- Matches vanilla read/write performance numbers
- Provides predictable performance for foreground IOs
Outline

• Introduction

• ioSnap™ Design

• Evaluation

• Conclusions
Design Goals

Goals

• Negligible impact on foreground performance
• Predictable foreground performance
• Minimal space overheads for snapshot metadata
• Integrate with existing FTLs
Creating / Deleting Snapshots in Flash

Key concept:

Creation: write a snapshot create note in the log
Deletion: write a snapshot deletion note

Snapshot creation/deletion is fast & negligible (fixed) metadata

Leverage time ordering of data in a log to create snapshots
Well What About Segment Cleaner?

Epoch: notion of period of time

Epochs help preserve log-time ordering
Managing Liveness of blocks

• **Issue:** snapshots indirectly increase the reference count
  • Validity bitmap with a single bit doesn’t work

• Possible solutions:
  • Maintain more bits/block ($2^{16}$ snapshot implies 16x increase in bitmaps)
  • Selectively maintain per sub-segment bitmap for snapshots
    • Only create a bitmap if a snapshot has (or modified) data in it

**Insight:** determine if a given block has at least **one** reference to it
Preserving Liveness Via CoW Validity Bitmaps

Snapshots in a Flash with ioSnap™ (patent pending technology)

![Diagram showing preserving liveness via CoW Validity Bitmaps]

Epoch 1:
- Segment Boundary: 10 20 30 40
- Snapshot: 1 1 1 1
- Bits needed to be flipped: 0 0 0

Epoch 2:
- Segment Boundary: 10 20 30 40
- Snapshot: 1 1 1 1
- Bits needed to be flipped: 0 0 0

Epoch 2 (step1):
- Segment Boundary: 10 20 30 40
- Snapshot: 1 1 1 1
- Bits needed to be flipped: 0 0 0

Epoch 2 (step2):
- Segment Boundary: 10 20 30 40
- Snapshot: 1 1 1 1
- Bits needed to be flipped: 0 0 0

Validity Map CoW
Snapshot-Aware Segment Cleaner

Snapshots in a Flash with ioSnap™ (patent pending technology)

Segment cleaner preserves log-time ordering
Design Summary

• Leverage RoW and implicit time ordering in the Log
• Epochs preserve log-time ordering even with a cleaner
• Sub-segment-level bitmaps to track validity of blocks
• Snapshot-aware cleaner preserves log-time and block validity

• Snapshot management
• Background snapshot activation
• Rate-limiting to provide predictable foreground performance
Outline

• Introduction

• ioSnap™ Design

• Evaluation

• Conclusions
Evaluation

• How does it compare with existing snapshotting systems?
• What’s the impact on user IOs in the absence of snapshots?
• Snapshot creation/deletion time? Implications on user IO?
• Implications of a snapshot-aware segment cleaner?
• How long does it talk to activate a snapshot?
• Implications on the crash recovery mechanism?

Setup: quad core Intel i7 processor, 1.2TB NAND flash, 12GB RAM, Linux 2.6.35, older generation of Fusion-io VSL driver, 4K Block sizes
**Comparison with BTRFS (1)**

Impact on foreground latency upon snapshot creation

Around 8 GB of sequential writes followed by a random workload interspersed by a snapshot every 5 sec

Snapshots in ioSnap does not impact foreground latency
Conclusions

“Make everything as simple as possible, but not simpler.”
– Albert Einstein

• Flash is revolutionizing the storage industry
  • Rethink data services to leverage flash’s capability & performance

**ioSnap**: first flash-aware snapshotting system

• Leverages RoW capability in FTLs to implement snapshots
• Proposes usage of epochs to preserve log-time ordering
• Low-overhead instantaneous snapshots (*performance & storage*)
• Embraces *rate-limiting* to minimize impact to foreground traffic
• Activations are slow & can be 10s of sec for a TB size snapshot
Thank you