Motivation

- Policies may need to persist their own metadata • Key-Value pair like mapping unmapped Mapping needs to be **consistent** with application state Mapping shouldn't violate durability expectations **Challenge: Consistency** Example Policy: Thin Provisioning in a journaled file system • When bitmap deallocate a block, reclaim it Commit Block 🛛 🖁 🛛 Bitmap Block 🛛 🗸 File System Journal V is allocated as data block again, flushing data first (ordered mode) bitmap block B indicates data block V is freed B' B checkpoint checkpoint: bitmap block B B unmap(V) **Lessons**: ordering must be preserved bitmap block B indicates Block Interface Power failure, data block V is freed Txn not committed unmap(V) B **Lessons**: before txn commit would fail Apply policies for the last txn: bitmap block B indicates unmap(V) data block V is freed B **Block incoming**

- Management features are not built in?
- Storage and Hardware • SSDs (PCIE/SATA) File systems • Disks (SAS/SATA) Remote storage Scientific clusters • Tapes **Management Challenges** ZFS as volume manager Even integrate special hardware Storage virtualization/deeply stacked storage? Users run other applications? Storage hardware is associated with properties (e.g. performance, reliability) Storage system is aware of application semantics Metadata vs data Allocated vs unallocated Flexible policies map application requests to hardware capability Composable policies make it easier to create new solutions. Applications (File Systems/Databases) Storage System I/O Requests with application semantics High Level Storage Policies Composable Provisioning Cache Policy Policy Eviction Writeback Block Mapping Flush Policy

- Workloads and Applications • Databases • NFS servers **Current solutions are** : **Applications manage underlying hardware** Examples: File systems: ZFS/btrfs • Integrated: metadata replication, RAID-Z,... NFS Servers: NetApp Applience **Problems with current solutions: What if ...?** Monolithic solutions are inflexible
- Storage environments are highly heterogeneous **Policy Based Management**

SATA SSD Disk Remote Disk PCIE SSD

Policy Based Storage System in Heterogeneous Environment

Dai Qin, Ashvin Goel, Angela Demke Brown University of Toronto {mike,ashvin}@eecg.toronto.edu demke@cs.toronto.edu

Challenges

Lessons: after txn commit would fail

I/O requests

Logical Blocks

Mapping Table

Persist



Physical Volumes



The Deallocated Block

txn commit: V marked allocated, but unprovisioned

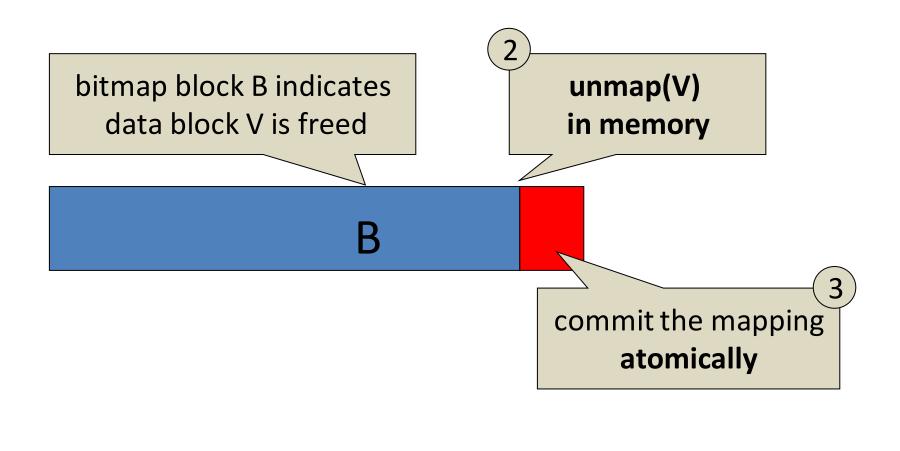


Persist the mapping during checkpoint **Result: unmap(V)** Data lost

Persist the mapping before txn commit **Result: unmap(V)** Data lost

Persist the mapping after txn commit **Result: V might still** be provisioned

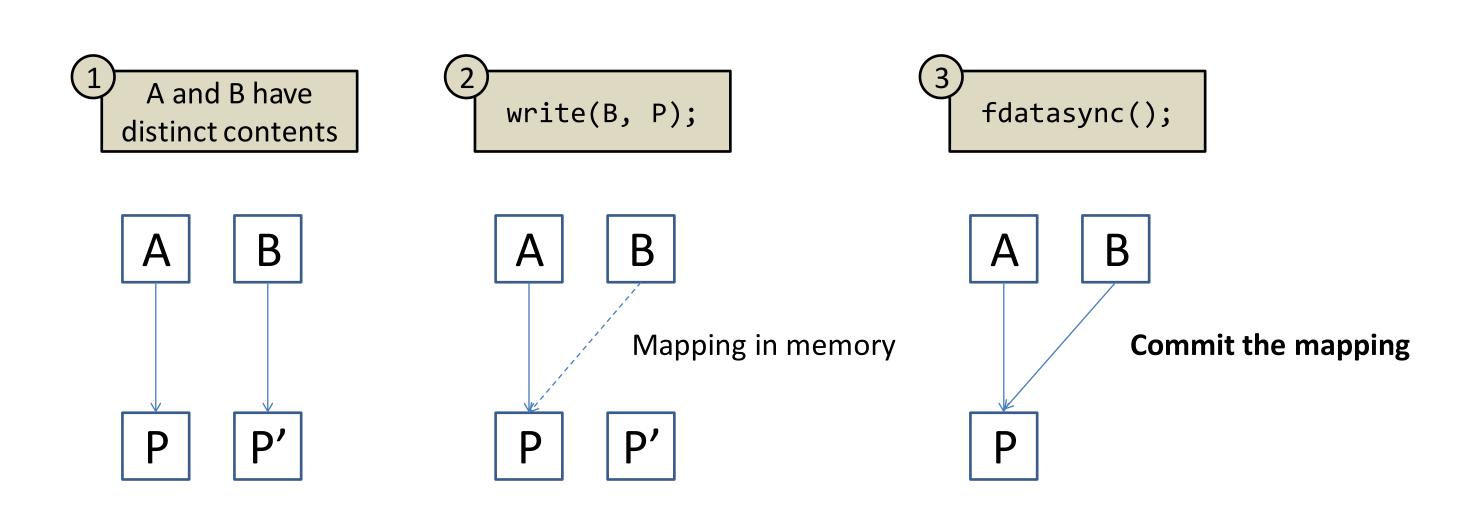
Solution: commit the mapping **atomically** with file System commit record



Challenge: Durability

- Application have durability expectations

 - are durable
- Example Policy: Inline Deduplication



Summary of Solutions

Guarantee	Durability	Consistency
When to create mapping	On data update	After txn finish Before commit
When to commit mapping	On disk barrier	Commit atomically wit txn

Power failure: V->nil might not be done

• After disk barrier applications expect that file data

• Blocks with same content mapped to same location

Need to persist the mapping on fdatasync()