Algorithms and Data Structures for Efficient Free Space Reclamation in WAFL

Ram Kesavan

Technical Director, WAFL

NetApp, Inc.

SDC 2017

Outline

- Garbage collection in WAFL
  - Usenix FAST 2017
  - ACM Transactions on Storage (coming soon)
- Selected topics in WAFL
  - recently published papers

WAFL: 25+ Years!

- Technology trends
  - MHz -> GHz cores, 1 core -> 40+ cores
  - HDDs (5k->15k rpm, MB->TB), SSDs (100GiB->32TiB+), Cloud, SCM
  - Max file system size: 28 GiB -> 400 TiB
- Applications
  - NFS/SMB file sharing -> LUNs with FC/iSCSI access
  - Virtualization over block & file access
- Many data management features
  - snapshots, clones, replication, dedupe, compression, mobility, encryption, etc.
- Mixed workload deployment
  - ONTAP node: 100’s of WAFL file systems, 100’s TiB, 1000’s of LUNs, etc.

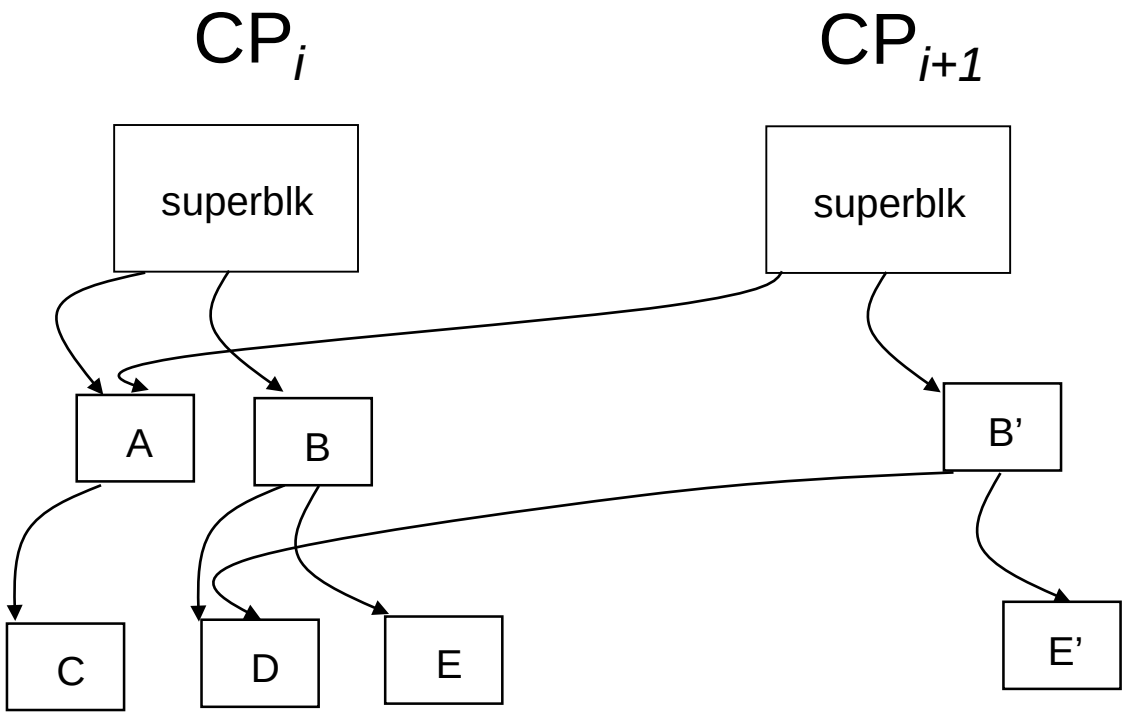
Free Space Metadata

- Reasons for tracking free space
  - find & allocate blocks for new writes
  - report usage for provisioning/purchasing decisions
- Design dimensions:
  - in-memory vs persistent
  - lazy vs contemporaneous
- Consistent performance is key: balanced resource usage
  - user ops vs free space management (& other backend work)
  - CPU, I/O, memory, etc.

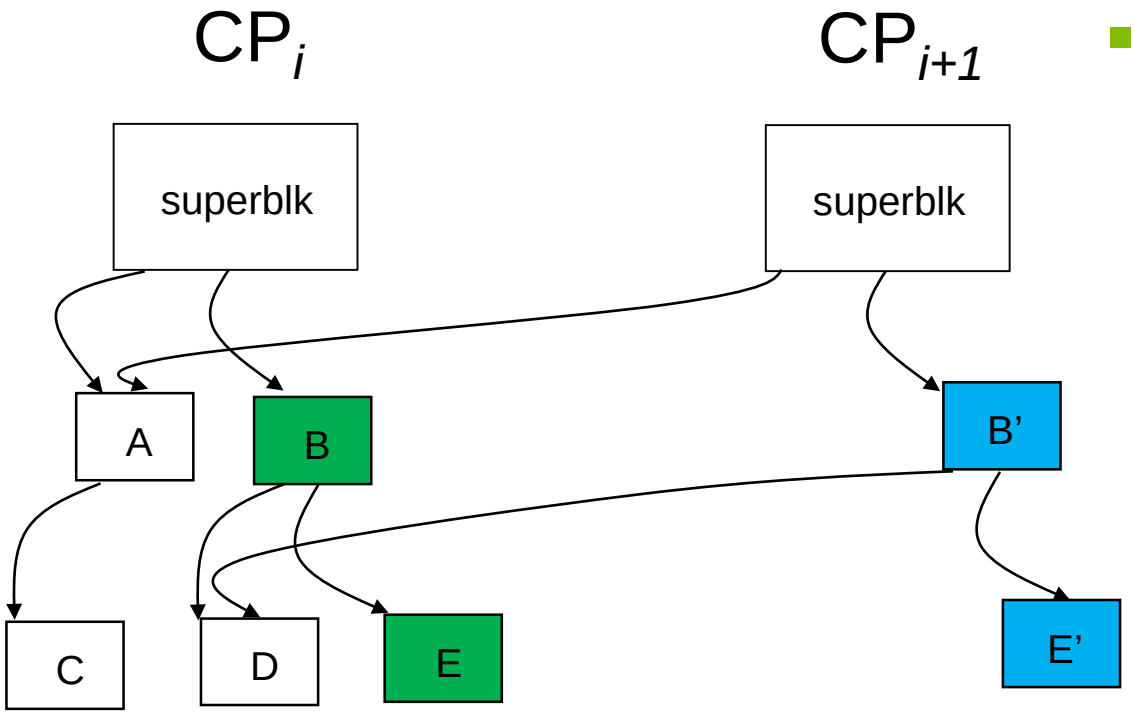
WAFL: Layout & Transactional Model

- File system is a tree of blocks
  - everything is a file, including metadata
- Log-structured & copy-on-write
  - each *dirty* buffer written to a newly allocated block
  - except the superblock
- Large atomic transactions: ~GB worth of dirty buffers
  - ~2s to 5s long
  - persistent file system always self-consistent
  - *consistency point* (CP)

Atomic Update of the Persistent File System



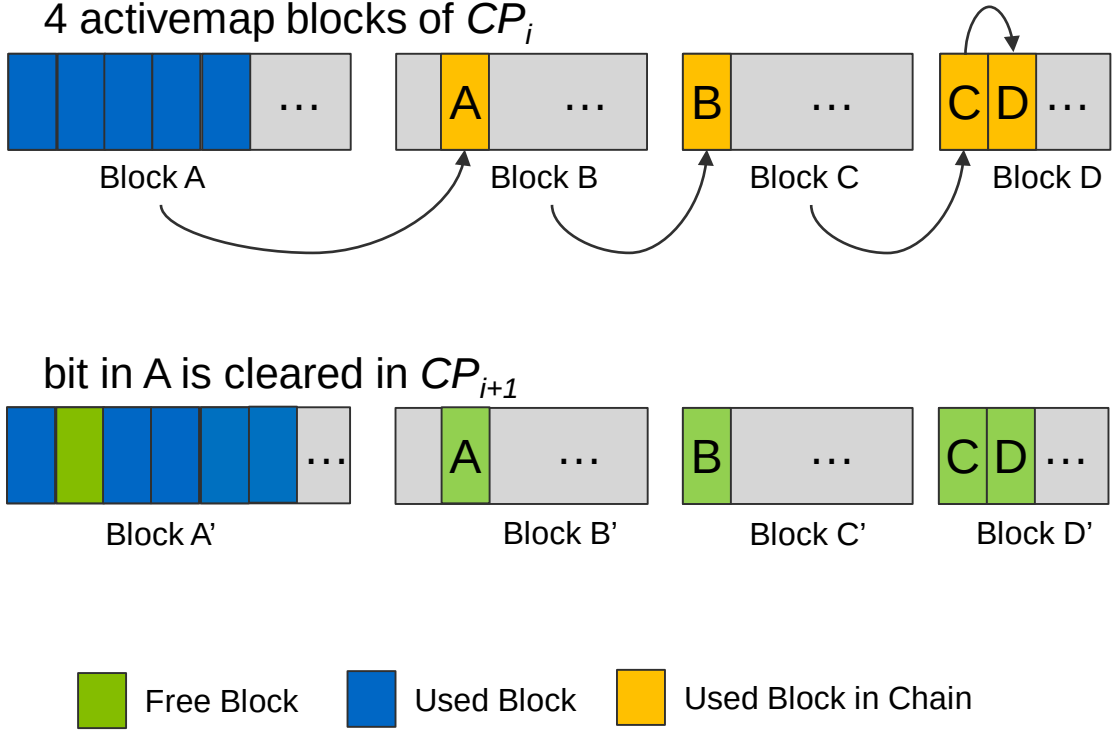
Atomic Update of the Persistent File System



- 1 GB/s writes need
  - 1 GB/s allocations of new blocks
  - 1 GB/s frees of unused blocks

Activemap Bits: used(1) -> free(0)

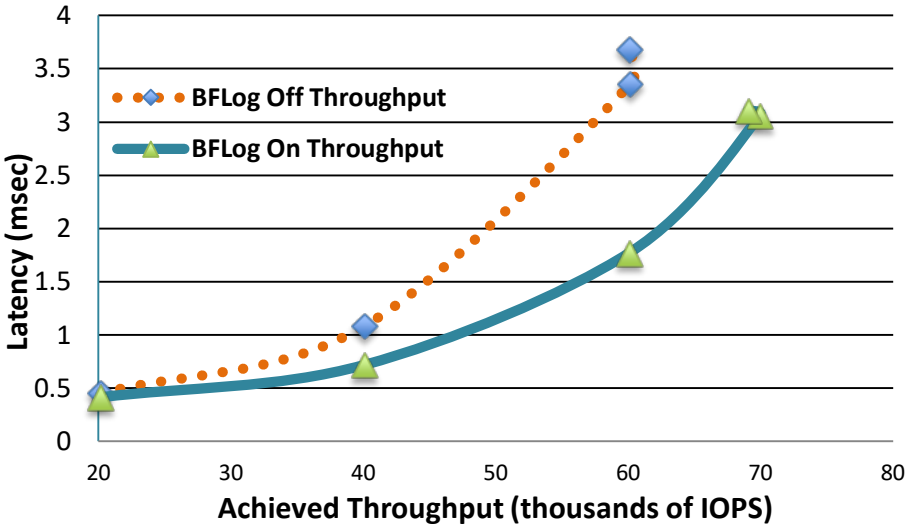
- Long CPs due to:
  - random frees -> more dirty activemap blocks
  - long activemap chains
- Long CPs hurt write throughput

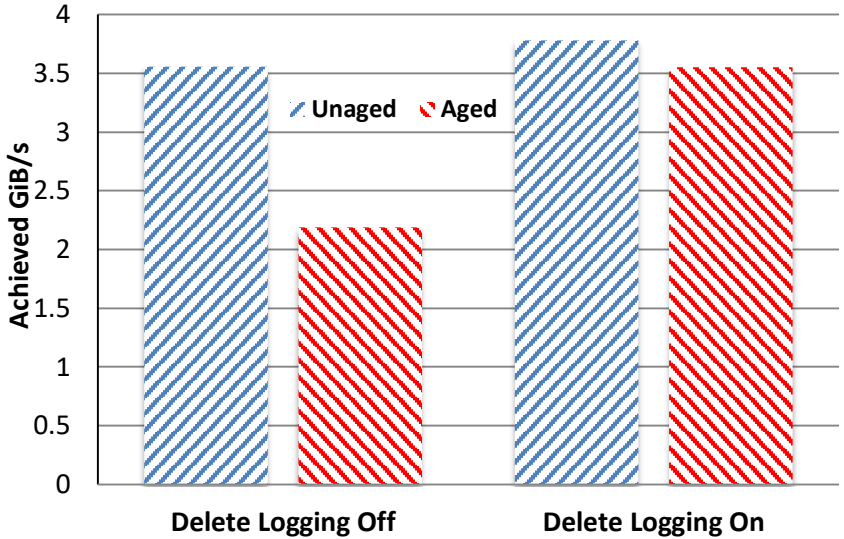


Append Log using L_1s of a File

- Delay updates of activemap to avoid the chaining problem
- Convert several random to few batched
  - TLog(’08): binary sort tree using L_1s of log file
    - dropped due to unpredictable performance
  - BFLog(’12): 3 files - active, inactive (sorting), sorted
    - predictably pace sorting and freeing activity
- Log sized to ~0.5% of file system provides sufficient batching
- Blocks freed by file/LUN deletions, SCSI hole-punching

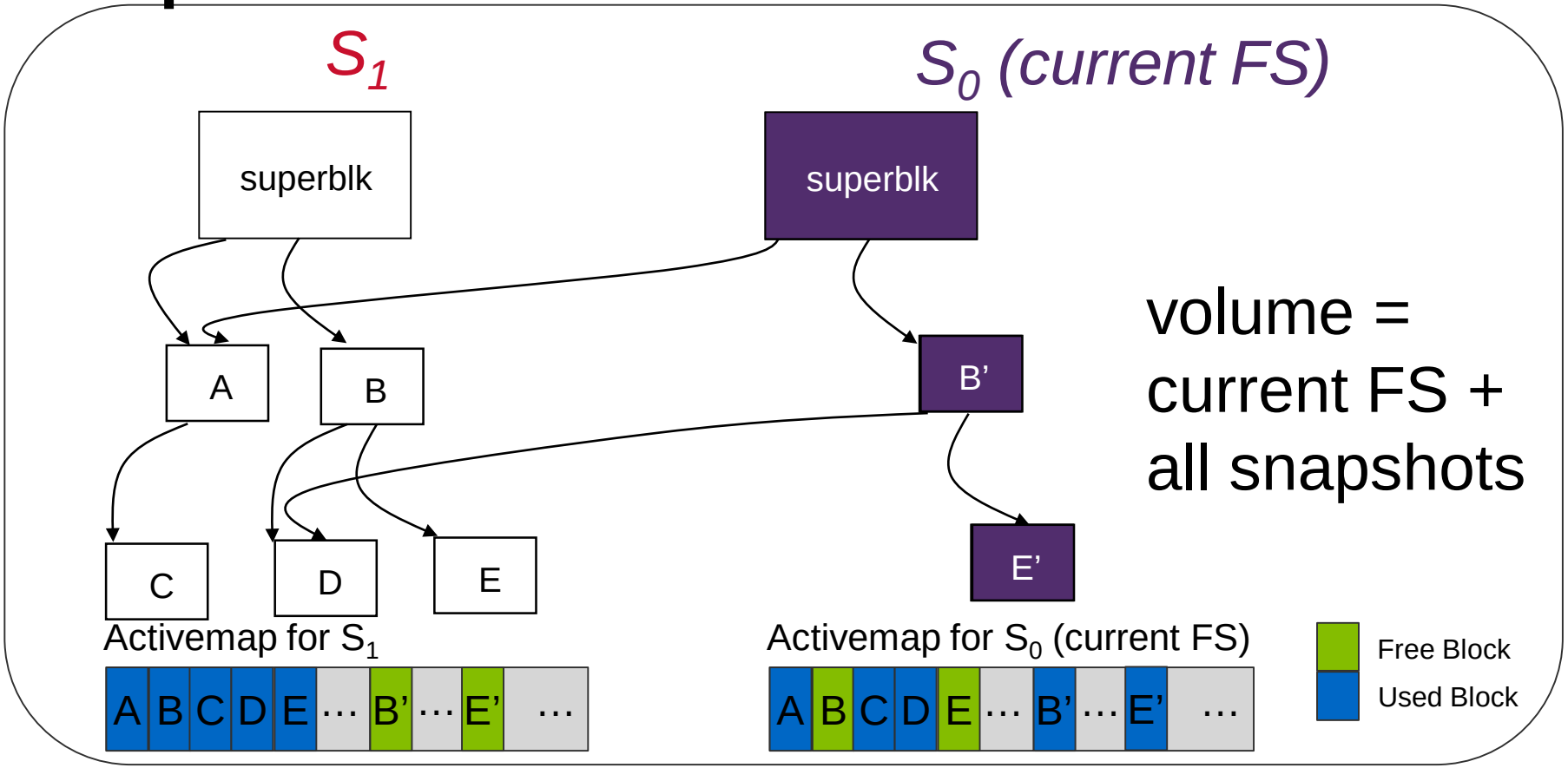
Logging Results





- 17% higher write throughput at 34%-48% lower latency
- 6% to 60% (unaged v aged) raw deletion throughput
- Much higher delete throughput with little interference (not shown)

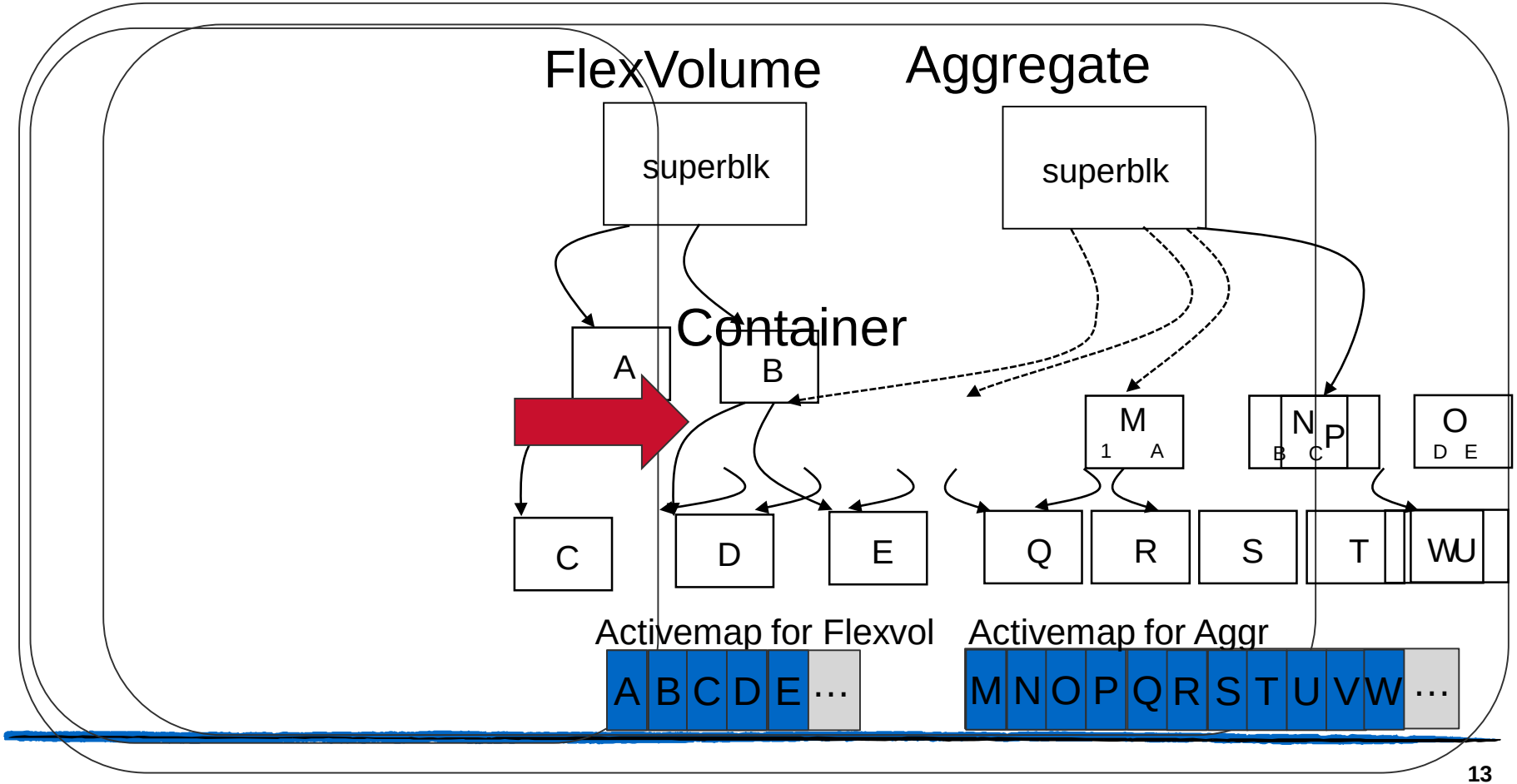
Snapshots


S_1
S_0 (current FS)
superblk
superblk
A
B
B’
C
D
E
E’
volume =
current FS +
all snapshots
Activemap for S_1
A B C D E … B’ … E’ …
Activemap for S_0 (current FS)
A B C D E … B’ … E’ …
Free Block
Used Block

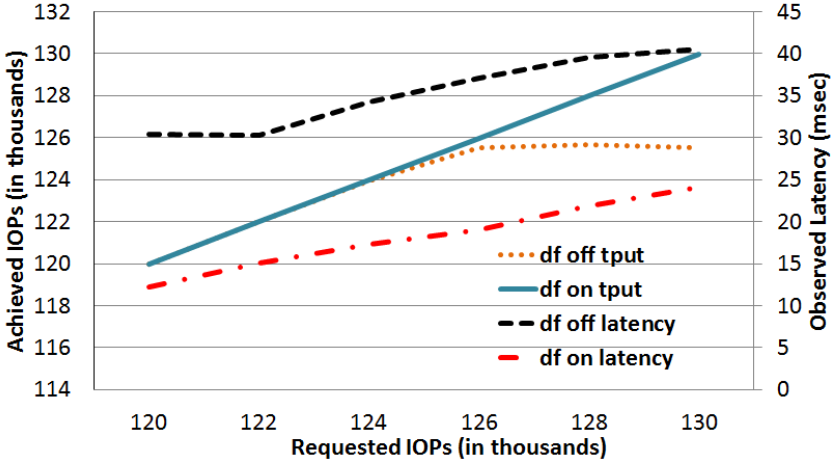
Summary Map

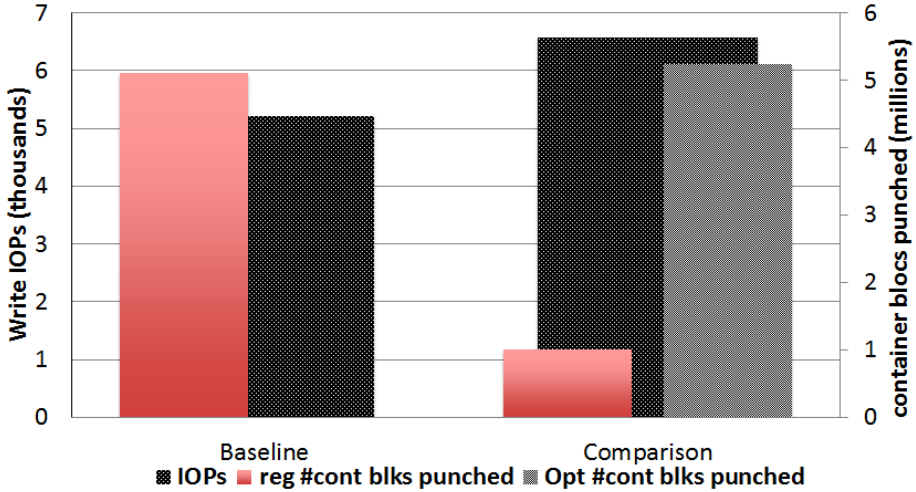
- Block is free iff every activemap says it is free
- Summary Map = Bit-OR of snapshots' activemaps
  - metafile in the current file system
- Block is free iff active & summary bits are both 0
- Summary can become stale:
  - snapshot creation, deletion, etc.
  - sequential scan to fixup summary
  - correctness is preserved while scan is in progress
- Fast reclamation of space without impacting other operations

WAFL Layering: FlexVol & Aggregates



Bunched Delayed Frees & Logging Interaction





- Bunched delayed frees: 60% lower latency at any load & higher saturation point
- Optimized processing via FlexVol Log: 84% of the container blocks punched out in optimized mode -> 26% higher throughput

Other Topics in WAFL

Go further, faster®

WAFL Block Allocation

- WAFL & ONTAP configurations:
  - all-HDD, Flash Pool: SSD+HDD, all-SSD, CloudOntap (AWS/Azure), Select: software defined, Fabric Pool: SSD+ObjectStore (on-prem, AWS/Azure)
  - Allocation policies based on “temperature”
    - Client access patterns, policies, snapshots, replicas, etc.
- Flexible Block Allocator
  - Incorporate new media
  - Test & productize new policies quickly
  - MP: scale-up with #cores

ICPP 2017: Scalable Write Allocation in the WAFL File System

Scaling with CPU cores

- ONTAP was originally single-threaded
  - 2 WAFL threads: client ops vs CP/internal
- WAFL MP model
  - Needed incremental MP-ization of existing code
  - Buffers/inodes: fine-grained locking vs data/metadata partitions
    - Frequent ops first: read, write, lookup, readdir, etc.
  - Hierarchical data partitioning
    - All ops: user & internal
- WAFL thread creation/exit is dynamic

OSDI 2016: To Waffinity and Beyond: A Scalable Architecture for Incremental Parallelization of File System Code

WAFL Buffer Cache

- Mostly LRU buffer cache
  - User data vs metadata; pinning, tracking, etc.
  - Speculative readahead
  - Tricks to read from SSD, HDD, etc.
- MP access: insert, read, write, scavenge

  ICPP 2016: Think Global, Act Local: A Buffer Cache Design for Global Ordering and Parallel Processing in the WAFL File System

WAFL Resiliency

- Storage devices misbehave
  - Media errors, failures, lost/redirected writes
- RAID-DP, checksum + context
- Software/hardware bugs: scribbles vs file system logic bugs
- Prevent corruption from getting persisted
  - Use transactional nature of WAFL
  - Verify delta equations
- And, with negligible performance impact

FAST 2017: High Performance Metadata Integrity Protection in the WAFL Copy-on-Write File System

Other Topics

- Scale-out across an ONTAP cluster: FlexGroups
- Recovery: WAFL Iron for online repair
- Data management topics
  - Storage efficiency: deduplication, compression, compaction
  - Replication, retention, data mobility
  - Instantaneous cloning: file & file system granular
  - Software encryption
- Revert & non-disruptive upgrade
- Upcoming projects: new media, Plexistor integration, etc.

Conclusion

- WAFL has evolved over 2+ decades
  - deployments, media/hardware trends, applications/workloads
- Fundamental designs
  - some stood the test of time
  - some changed with requirements/trends
- Consistent & predictable performance
  - with low latency and high throughput
- Fun & difficult problems
  - some have been solved
  - several new problems