Storage Gardening: Using a Virtualization Layer for Efficient Defragmentation in the WAFL File System

Ram Kesavan, Matthew Curtis-Maury, Vinay Devadas, Kesari Mishra

NetApp Inc.

Outline

- Some WAFL Background
- Defragmentation techniques in WAFL
  - Multi-core systems, multiple storage devices of different media
  - Trade-offs for enterprise-grade system
  - Multiple applications and use-cases
  - All features (eg. snapshots) must be preserved
- Evaluation & some history

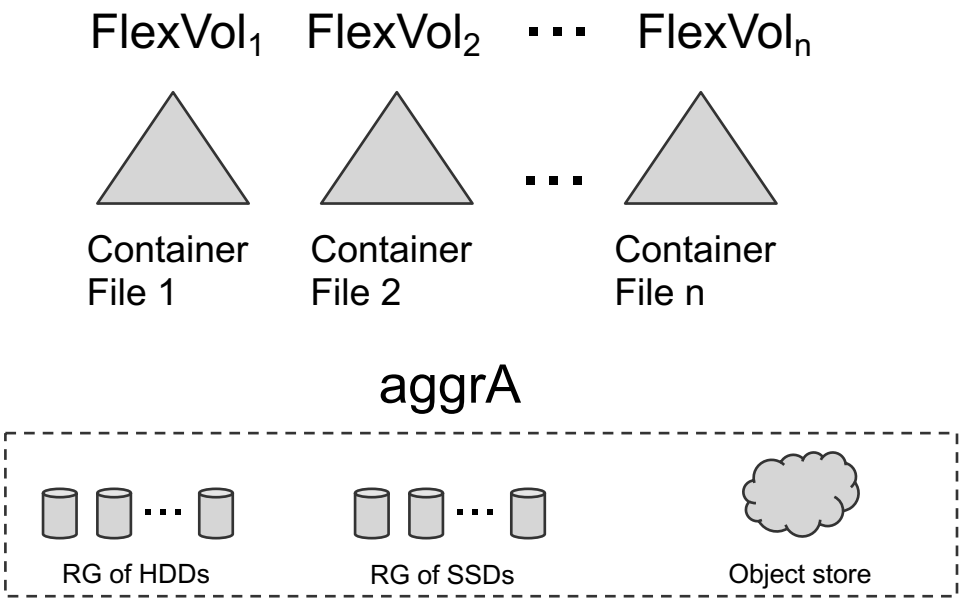
Forms of Fragmentation

- File layout fragmentation
  - Impacts sequential read performance
  - Mitigation: predictive/speculative readahead algorithms
  - Unavoidable: more reads IOs
- Free space fragmentation
  - Impacts write throughput of file/storage system
  - Impacts file layout
- Intra-block fragmentation
  - WAFL supports sub-block compaction & addressing
  - Impacts achieved storage efficiency
- Briefly discussed: Objects in an object tier

WAFL Background

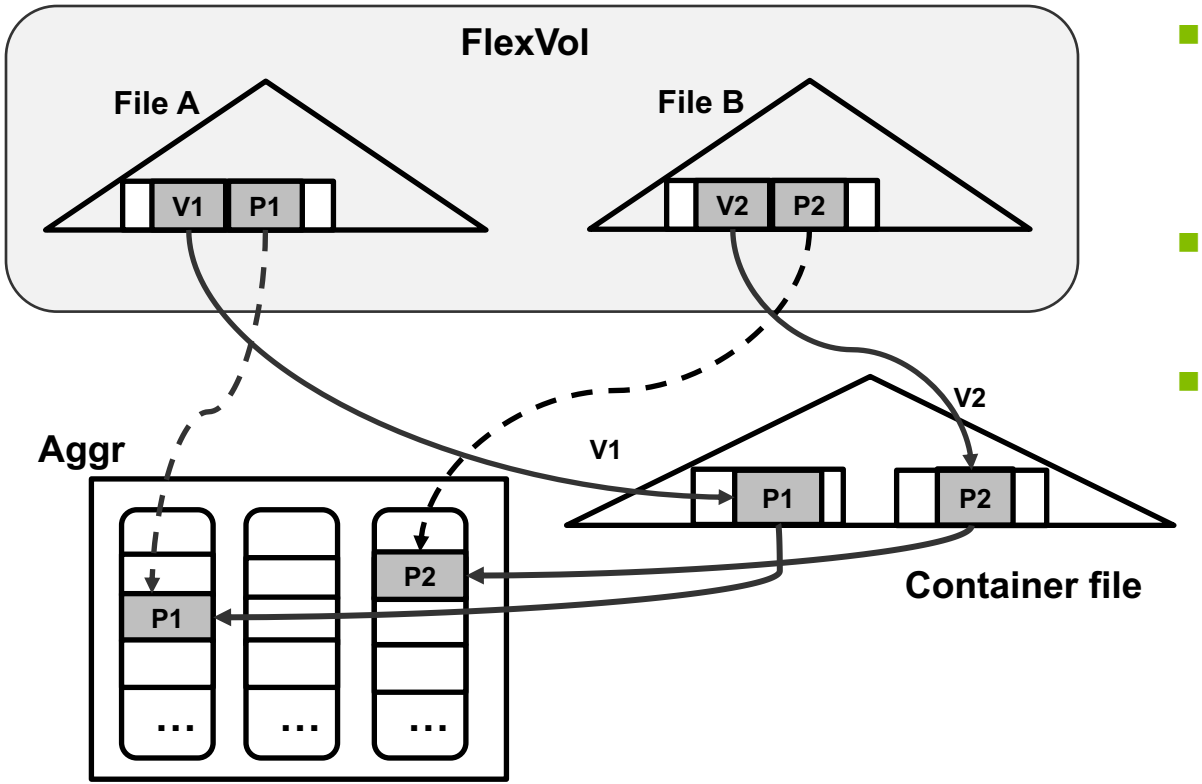
- Runs in the kernel space of proprietary OS: ONTAP
- Multi-core, multi-protocol (NFS, SMB, SCSI, NVMe, etc.), multi-workload
- Multi-media: HDDs, SSDs, object store, cloud, flash devices, and more
- WAFL is feature-rich
- Typical deployment:
  - One node: 100's of FlexVols (file systems), 100's of TiB, 1000's of LUNs, with dozens of applications
  - Several features enabled: snapshots, file cloning, file system cloning, replication, dedupe, compression, mobility, encryption, etc.
- Copy-On-Write: so has the potential to fragment
  - Storage gardening techniques

FlexVols & Aggregates



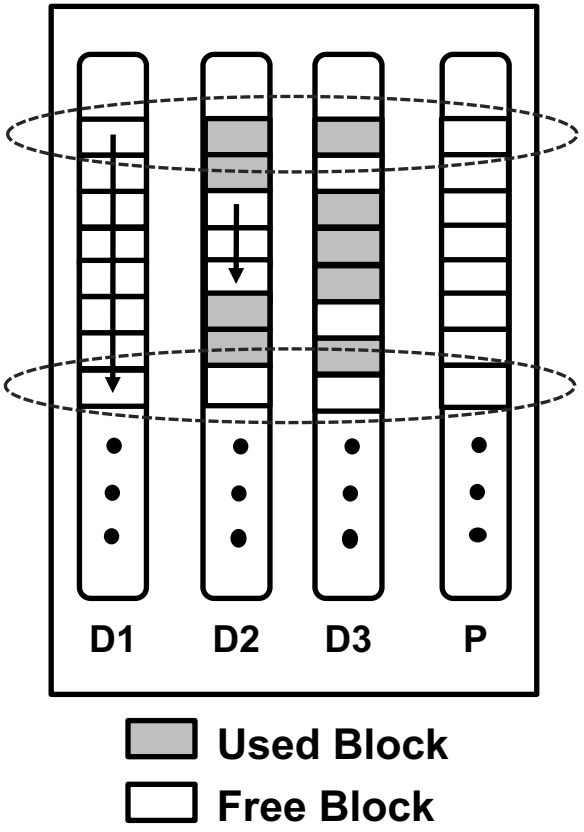
- Aggregate: pool of storage
- FlexVol: namespace exported to clients—files, dirs, LUNs
- Each FlexVol & Aggr is a WAFL file system
  - tree of blocks rooted at superblock, leaves of tree contain data of user files & metafiles
- Each FlexVol stored as a container file

FlexVols & Aggregates



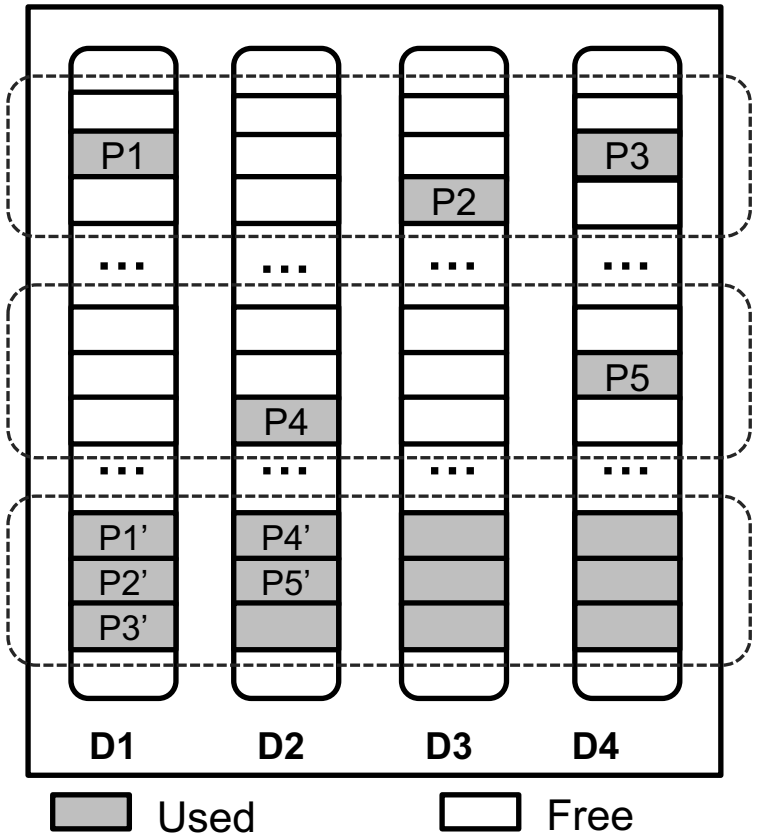
- 4KB block number spaces:
  - V in FlexVol
  - P in aggregate
- FlexVol structures cache P
  - For performance
- V->P indirection used for several features in WAFL
  - Also for storage gardening techniques

Free Space Fragmentation



- Results in "partial" stripe writes
  - More reads & compute for xor/parity, more writeIOs, poor file layout
- Latency of write-op not directly affected
  - Ack'ed after logging to fast NVRAM
- WAFL checkpoints ~GBs of dirty content
  - Should complete in few seconds
  - Lower device write throughput => longer checkpoint
  - Impacts op latency & IOPS throughput

WAFL: Segment Cleaning



- Each block stored with WAFL context
  - For protection against lost writes
  - Identifies file + offset
  - FlexVol block => container file
- Loaded as dirty container leaf block
- Rewritten (moved) by next CP
  - Preserves snapshots in FlexVol
  - More efficient
- Lazy fixup of stale cached P
  - WAFL context used to catch stale P
  - Read redirected to container file
- CSC: JIT cleaning of segments

CSC Evaluation: Summary

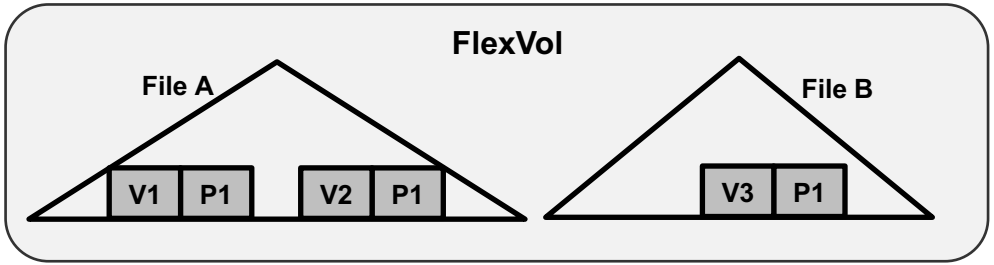
- All-HDD:
  - With CSC: Op latency & write chain length stabilize after 35 days
- SSD+HDD:
  - Hot spots of working set stay in SSD tier; SSDs fragment quickly
  - HDD write chain length stabilizes after 22 days
  - CSC in SSD-tier: Ameliorates flash wear-out (early gen. SSDs)
- All-SSD:
  - Expectation of high & consistent performance: CPU is the bottleneck
  - Disk bandwidth & wear-out is less of a concern (modern gen. SSDs)
  - CSC improves write chain lengths, but without op latency improvement
  - Pure random overwrites: op latency regresses (0.7ms → 1.3ms)

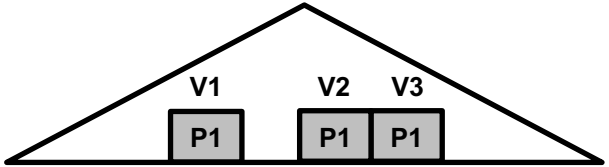
File Layout Defragmentation

- Re-dirtying file blocks can trivially fix contiguity in P space
  - But that impacts efficiency of diff'ing-snapshots of the FlexVol
- Special *fake-dirty* that retains its V
  - Diff'ing of snapshots finds V unchanged
  - Stale P handled by consulting the container
- Performance: pre-fragmented data
  - All-HDD: results in lower latency, higher throughput
  - All-SSD:
    - Beneficial for pure read workloads
    - But, op latency increases with read/write op mix (1.7ms → 2.5ms)

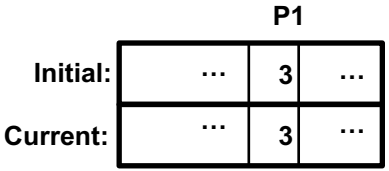
Sub-block Compaction in WAFL

- Tail-end of files & of compression groups
- Compacted into leaves of the container file
- Benefits:
  - No fixed sizes for sub-blocks; potential for workload-aware compaction
  - Compaction-related metadata overhead proportional to savings
  - Crunching/re-compaction of FlexVol snapshots

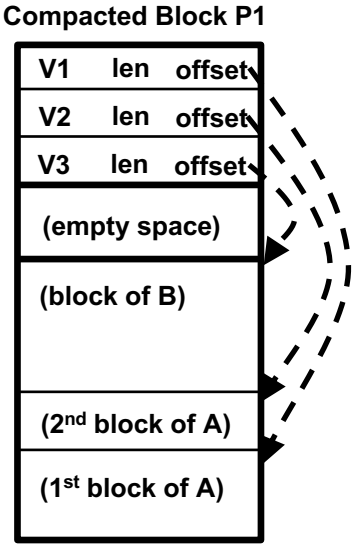




Container File for FlexVol



RefCount Metafile



Re-Compaction

- Background scan recovers wasted fragments
- Walks container file
  - Compares ref_i with ref_{curr} per P; if worth re-compacting…
  - …each fragment loaded as a dirty container leaf
  - Re-compacted (moved) by next CP
- Same handling of stale cached P
- Future work:
  - Make re-compaction less intrusive
  - Make re-compaction autonomous

Conclusion

- Fourth form of fragmentation
  - WAFL aggregate can include (on-prem/remote) object tier
  - Cold blocks packaged into large objects
  - Objects can get fragmented; defragmentation based on cost-metrics
- Defragmentation techniques help in all-HDD or mixed-media systems
  - All-SSD systems are sensitive to CPU consumption
  - Ideally, defragmentation should be autonomous and based on system load
- Customer data show
  - All-SSD systems have sufficient CPU headroom
  - Autonomous defragmentation is feasible
  - Consistent & predictable performance