WAFL Iron: Repairing Live Enterprise File Systems

Ram Kesavan, Harendra Kumar, Sushrut Bhowmik

NetApp Inc.

USENIX Conference on FAST, 2018

WAFL Background

- ONTAP available in various configurations
  - “Typical” ONTAP node: 100’s of FlexVol volumes in a few aggregates
  - Datasets of several apps hosted on a node
  - FlexVol and aggregate are WAFL file systems, each up to 800 TiB
- WAFL is a log-structured COW file system
  - Guaranteed consistency after panic/power failures
  - Journal of recent client ops in NVRAM
  - All file system state reconstructed during journal replay
  - Corruption of journal cannot corrupt file system
    - at worst, loss of some recent client ops

Protection For File System Blocks

- Each 4KB block persisted with checksum + file system context
  - Checksum protects against subsequent media damage
  - Context protects against lost/mis-directed writes
  - Mismatch during read is detected below WAFL
- WAFL layered on software RAID
  - 0+1, dual parity, and triple parity
  - Damaged blocks reconstructed on-the-fly by RAID
- Reconstruction may be impossible; in rare cases
  - Multi-device failure (past RAID's limit)
  - Block corrupted *before* checksum, context, parity computed
    - Typically software bugs: scribbles or bugs in logic

WAFL Check & Repair

- On-the-fly handling of damage to user data
  - Persistently tag as bad, return protocol-specific error
- On-the-fly handling of damage to most auxiliary metadata
  - Rebuild in background
  - At worst, temporal loss of associated feature or performance
- Aggregate marked corrupt and taken offline
  - Only if WAFL code-path cannot navigate past corrupt metadata
  - NetApp customer support case raised
  - Repair invoked

Online vs Offline Repair

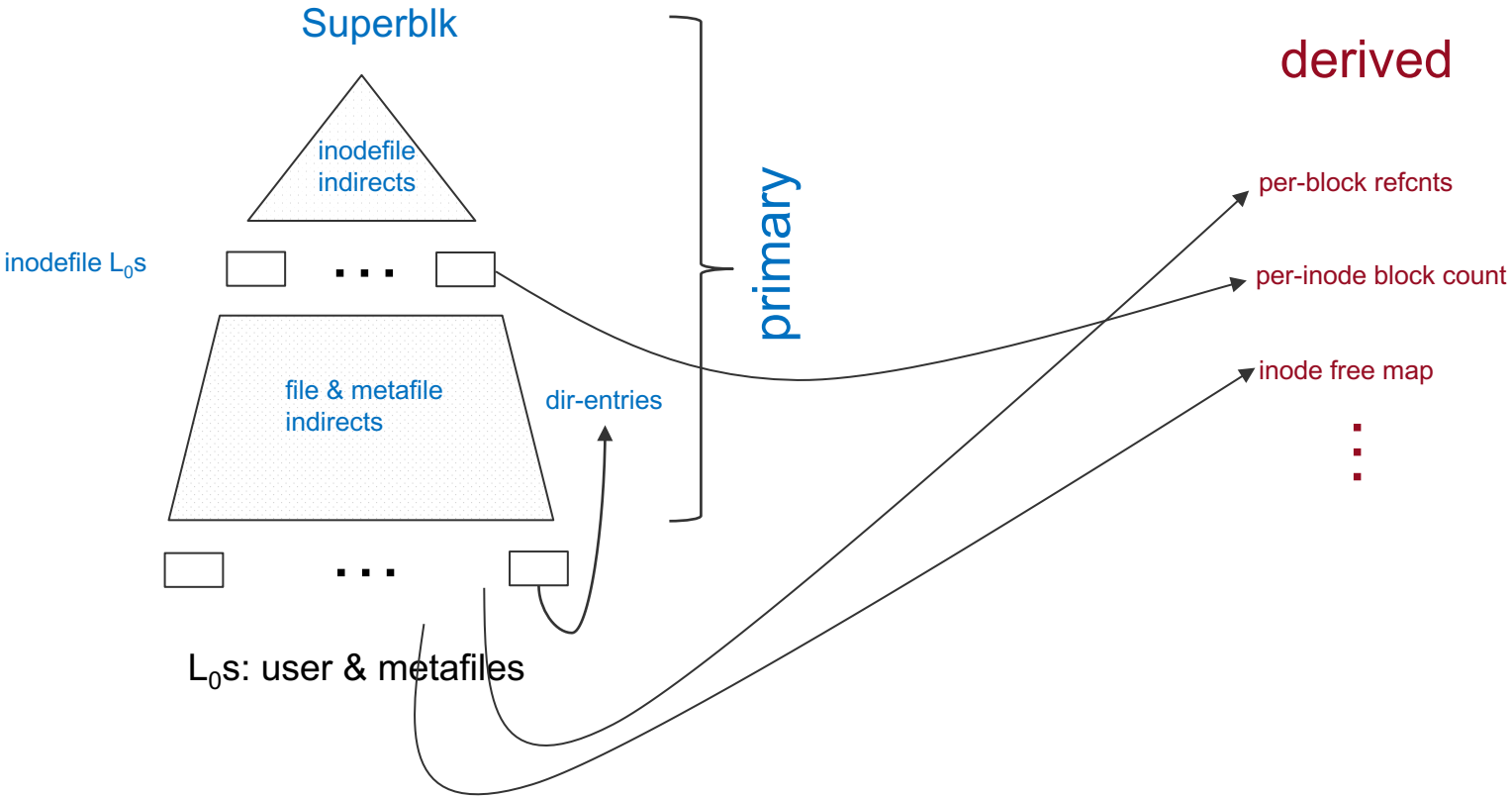
- WAFL Check (circa1996): classic fsck-like behavior
  - Memory:Metadata ratio scaling issues
  - File system unavailable to clients
  - Time to complete ~linear with file system size
- WAFL Iron (circa 2003): focus on above drawbacks
  - Flag to “aggr online” command
  - Unconditional commit; offline mode of Iron provides conditional commit
  - Repair speed controllable: can trade-off client performance
- Other examples: ReFS and recent Linux check-ins for XFS

Iron provides practically the same assurances as WAFL Check did.

Rules of Engagement for Iron

- Similar treatment for all ops (protocol & internal)
  1. Ensure all state is checked on-demand at access
  2. Don’t reverse state afterwards
     - Can do so conservatively for internal ops
  3. Only check each metadata block once
     - Monotonically expand portion of self-consistent metadata
     - Guarantee convergence even with new incoming ops
  4. Iron status metadata must scale with memory, file system size
     - Stored in (meta)files; all metadata in WAFL are stored in files
     - Pages in/out of buffer cache based on access

Metadata in WAFL: Primary vs Derived



Claiming of Resources: Iron Status Metadata

1. Shadow derived metafiles
   - Re-computes derived metadata
   - Eg. claimed refcnt: i^{th} number counts #refs to i^{th} block seen thus far
   - Replaces derived metadata
     - If count adjusted down to 0: reclaims lost block
   - A block is considered free iff claimed refcnt == refcnt == 0
     - WAFL block allocator consults (and increments) both counts
     - "Free"-path *conditionally* decrements the claimed refcnt
2. Progress indicator metafiles
   - Ensures no wasted work; metafile blocks are scavenged and re-loaded
   - Eg. checked bitmap: i^{th} bit indicates i^{th} block has been checked

Phases of WAFL Iron

- Mount: client access not allowed yet
  - Scan subset of primary & derived metadata
  - Reduction of that subset over releases: hour+ to <1min
  - Create Iron status metafiles
  - Mount aborts if this step does not complete
- Enable full client access
  - On-demand check + repair work based on client access
  - Background scan to check + repair all metadata
- Completion: no change for clients
  - Final book-keeping activity
  - Delete Iron status metafiles
  - Aggregate marked clean

Corruption: Manifest vs Latent

- Manifest: localized to the block
  - Invalid signatures, out-of-bound values, etc.
  - Mostly caused by hardware errors or memory scribbles
  - Detected on first load/use of metadata in the block
- Latent: each block appears correct
  - Violates distributed property of the file system
  - Mostly caused by logics bugs
  - Detected when invariants in the code are tripped
- Both forms of corruption can impact primary or derived metadata
  - Paper walks thru' each permutation

Claiming of Resources: On-Demand vs Lazy

- Via background scan: eg. claimed refcnt metafile
- On-demand claiming protects against latent corruption
  - i.e., avoid incorrect re-allocation of a used resource
- Metadata integrity (circa 2011)
  - Described in one of the FAST ‘17 WAFL papers
  - Avoids latent corruption due to memory scribbles or logic bugs
  - Avoids corruption to Iron status metadata
- Quarantining & metadata integrity ensure no incorrect re-allocation
  - On-demand claiming of resources became unnecessary
  - Random IOs (claimed refcnt metafile) avoided when servicing client ops

Performance

- Two main metrics: decade+ continual improvement
- Time to first-client access
  - Hours down to secs in almost all configurations
- Interference to client performance
  - Apps in the ”ironing” aggregate and other aggregates in the node
  - Up to 25% on mid to low-end range
  - Extra IO overhead due to Iron less on all-SSD nodes
    - But, interference (as a %age) is higher; baseline op latencies are <1ms
  - Other improvements in the dev pipeline
- With reduced interference, time to completion becoming less critical

Conclusion

- Practical online repair is necessary for enterprises
- WAFL Iron first shipped in 2003
  - Provides practically same assurances as offline repair
  - Continued improvement over time
    - Lower time to first access
    - Lower interference to client ops
- Recent changes and improvement (not presented in paper)
  - Autoheal; first shipped with FlexGroups
  - WAFL Iron parallelism enhancements