
whamcloud

Oct 26, 2010

Lustre Looking Ahead

- Eric Barton
  CTO
  Whamcloud, Inc.

Engineering Principles

- The facts
  - Distributed filesystems are big and complex
  - Moore's law drives requirements relentlessly
  - Instability loses customers *and* halts development
- The strategy
  - Create a *long term* development roadmap
    - Current developments must progress future goals
    - Detours and reverses are prohibitively expensive
  - Develop in baby steps
    - Minimize planning/estimation uncertainty
    - Instability costs much more to fix than to avoid
  - Control technical debt
    - Unnecessary complexity invites instability
    - Restructuring is cheaper in the long run

Agenda

- Far attractors
  - Where Lustre wants to be in the longer term
- Future Technologies
  - Game-changing building blocks
- Development choices
  - What Lustre needs now and in the future

Far Attractors

- Extreme system size
  - Scaling to > many 100,000s clients / 1000s servers
  - Scalable fault management / resilience
- Extreme throughput
  - Coherent load (HPC) / Random load (Scale-out NAS / Cloud)
  - Scalable MD R/W performance
  - Full bandwidth streaming I/O performance
  - Uncontended MD + small I/O performance like local F/S
- Usability
  - Central/Global File System
    - Quality of Service / Service level agreements
    - Security
  - Data management
    - HSM / tiered storage
    - Structured files / datasets
  - Administration
    - Scalable management
    - Scalable analytics

Future technologies: Health Network

- Scalable uncongested collective communications
  - Logically separate network
    - Unaffected by congestion due to O(n) server:client valence
  - Self-healing tree topology
    - Grows from paxos root cluster
  - O(log n) bounded latency
    - Limited traffic levels eliminate congestion
- Uses
  - Prompt / ordered global notifications
    - Fault avoidance
    - Fast recovery
  - Collectives
    - Census
    - Gang scheduling (QoS)
    - Distributed transaction recovery

Future technologies: Epochs

- Non-blocking distributed atomic transactions
  - Global transaction group numbers
    - Lamport clock distributed via DLM captures transaction dependencies
    - Sub-operations on different servers may commit in arbitrary order
  - Client & server recovery logs
    - Prune to oldest volatile epoch
  - Client failure
    - Determine incomplete transactions & roll back/forward
  - Server failure
    - Roll back all volatile epochs on servers & roll forward using VBR
- Uses
  - High performance CMD
  - Generalized RPC aggregation
  - Synchronous replication
    - RAIDed file contents (SNS)
    - Mirrored metadata
  - Transparent migration
  - Global non-blocking snapshots

Future technologies: Caching

- Mechanisms
  - Metadata caching
    - Readdir+ streams dirents & MDT attributes
      - SOM (stable files) / Bulk glimpse (changed files)
    - MD writeback accumulates MD updates and streams back to MDT
    - Subtree locks
      - Minimize shared-dir conflicts
    - MD extent locks
      - Minimize uncontended locking
  - Small I/O aggregation
    - Efficient writeback of random/scattered I/O to OST cache
  - Local SSD
    - Extended caching / Crash recovery
- Policy
  - Streaming / Contention handling / Disconnected operation
- Uses
  - Scale-out general purpose MD & I/O
  - Process/core aggregation
  - Client aggregation via caching proxies

Future technologies: Layouts

- Large, Complex, Dynamic file layouts
  - Layout operated on incrementally
    - Large layouts not penalized for small ops
  - Shared access and update
    - No unnecessary contention
- Uses
  - Data on MDT
    - Small files start on MDT and grow onto OSTs
  - Ultra-wide striping
    - Dynamic layout needed to circumvent Amdahl's law
  - Asynchronous replication & migration
    - Policy – driven tiered storage
    - Fast fault avoidance
  - Online OST & MDT Pool management
  - Non-posix files
    - Uncontended I/O
    - Layout may include complex file metadata (e.g. OODB)

Development: Current Issues

- Lustre 1.8
  - MD performance
  - Failover latency at scale
  - Ldiskfs 16TB OST size limit
- Lustre 2.0
  - Performance regressions / technical debt
  - Administrative shutdown / simplified version interoperation
  - SMP scaling / NUMIOA
  - ZFS / end-to-end integrity
  - Layout locking (HSM)
- Measurement & Test
  - Test automation & results database
  - Improved performance survey toolkit

Development: ZFS alternatives

- BTRFS
  - Potentially solid OSD candidate
    - Ticks many of the same boxes as ZFS
    - Fits Lustre 2 end-to-end data integrity model
  - Evaluation required
    - Design / Code review
      - Transaction / allocation / overcommit / error handling models
      - Fit with OSD API including 0-copy support requirements
    - Benchmarking
      - Full streaming…random I/O surveys
      - Namespace performance - throughput and latency
    - Stress testing
      - Performance and stability under load and error injection
- Ldiskfs
  - Overcome 16TByte limit
    - Kernel ext4 already 64-bit safe
    - Unproven patches available for userspace tools
  - T10DIF
    - Pass data checksums through ext4 (OSD API -> block API)

Development: Technical Debt

- Distributed resource management
  - Quota & Grant fragile
  - Too granular as limits reached
  - Robust generic implementation has wider uses
- Security
  - Client *nodes* authenticated to OST
    - Users causing OST operations not authenticated
    - OST operations not authorized
  - Capabilities required to provide required security
  - Non-trivial capability cache scaling issues

Development: Performance

- Multirail networking
  - Link aggregation
  - Failover
- Server-side request scheduling (NRS)
  - Control # service threads
  - Avoid deadlock
  - Preserve request sequencing guarantees
  - Implement policy
    - Priority / latency bounds
    - Fairness / starvation / QoS
    - Bound total requests-in-flight with varying distributed demand
    - Anticipate health network (gang scheduling)

Development: Sync CMD

- Sequenced updates for distributed operations
  - Write-thru of all but last update guarantees sequence
  - Order ensures worst failure outcome is an orphan
    - Offline lfsck
    - Online checker/scrubber
  - Inodes on same MDT as parent directory entry by default
    - Create scalable namespace using distributed (slower) operations
    - Use scalable namespace with non-distributed (fast) operations
    - Scale aggregate throughput
- Phase 1 - non-local directory create
  - Home/project dirs scattered over all MDTs
  - Home/project subdirs constrained to same MDT
- Phase 2 - striped directories
  - Directory entries hashed over directory stripes
  - O(n) speedup for shared dir ops (e.g. file-per-process create)

Development: Fault Management

- Fault reporting
  - Define canonical fault reporting architecture
    - Vendor neutral core support
    - Vendors integrate into their own management systems
  - Abstract small # of fault classes / severities
    - {Storage, Network, Processor} / {Fatal, Warning}
  - Anticipate health networks
- Online filesystem checker/scrubber
  - Avoid *lfsck* service outage
  - OSD-neutral global consistency checking
    - Bi-directional FID referencing
      - Inodes reference dirents
      - Objects reference inodes
  - OSD-specific resilvering

Development: Administration

- Improved configuration
  - Dynamic LNET configuration
    - Scripted (e.g. ip tools) rather than module parameters
    - Appropriate userspace / kernel code split
  - Simple text F/S configuration
    - Avoid *writeconf* pathologies
    - Explicit LND binding
    - Simpler to use and document
- OST / MDT pool management
  - Permissions
  - Simple space balancing / rebalancing
  - Simplified storage reconfiguration
  - Explicit migration between storage tiers
- Global snapshots
  - Utilize underlying OSD capability
  - Simple blocking model at first
  - Non-blocking snapshots with epochs

Thank You

- Eric Barton
  CTO
  Whamcloud, Inc.