
whamcloud

whamcloud

EOFS Workshop
Paris
Sept, 2011

Lustre at exascale

- Eric Barton
  CTO
  Whamcloud, Inc.
  eeb@whamcloud.com

Agenda

- Forces at work in exascale I/O
  - Technology drivers
  - I/O requirements
  - Software engineering issues
- Proposed exascale I/O model
  - Filesystem
  - Application I/O
  - Components

Exascale I/O technology drivers

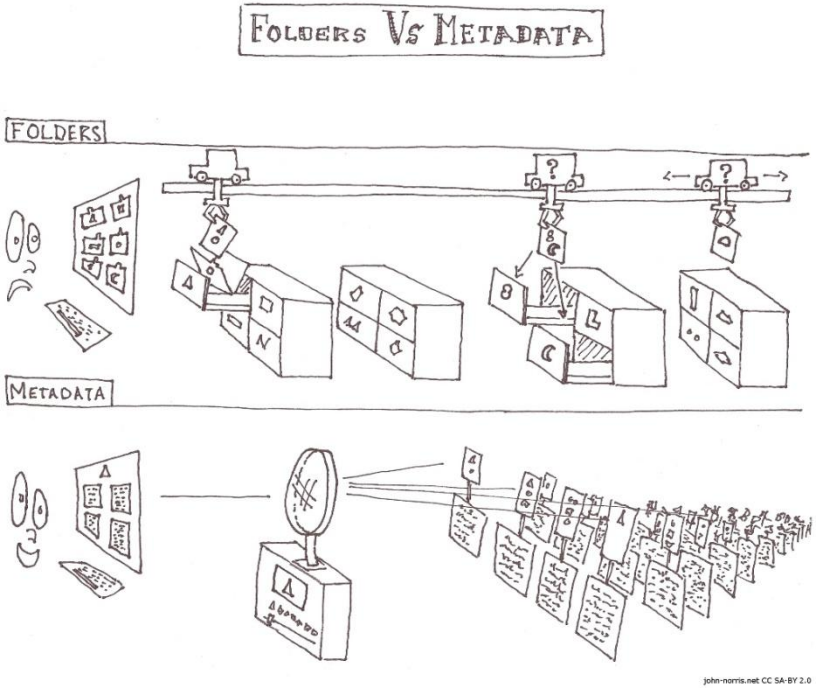
- Node
  - Concurrency: O(1,000)
  - Memory bandwidth: ~500GB/s
  - Comms bandwidth: ~50GB/s
- System
  - Compute nodes: O(100,000)
  - Total concurrency: O(100,000,000)
  - Total memory: ~10PB
  - System MTTI: O(1 day)
  - Storage nodes: O(1,000)
  - Total storage: ~500PB
  - Total I/O bandwidth: ~50TB/s

Exascale I/O technology drivers

- Local storage
  - Solid state – e.g. NVRAM on all compute nodes
  - High cross-sectional bandwidth to compute nodes
  - Low capacity
- Global storage
  - Predominantly disk
  - Relatively low network bandwidth available site-wide
  - Data
    - High capacity
    - Streaming I/O
  - Metadata
    - Low capacity
    - Random I/O

Exascale I/O technology drivers

- (Meta)data explosion
  - Many billions of entities
    - Mesh elements
    - Graph nodes
    - Timesteps
  - Complex relationships
  - Uncertainty quantification (UQ)
    - 1,000s of simulation runs
- OODB
  - Query / Search
    - What did my simulation(s) discover?
  - Schemas / Inheritance
    - Data provenance / quality
- Storage Management
  - Migration / Archive



Exascale I/O requirements

- Concurrency
  - 100,000s of concurrent I/O streams
    - Dictated by # of compute nodes
  - Object I/O
    - "Laminar" data flow by definition
  - I/O forwarders
    - Limit server fan-out to O(1,000)
- Multiple complex schemas
  - Multiple application domains
    - Different schemas
  - Not suitable for kernel development
    - Much harder to debug
- Search
  - Non-resident index traversal
    - Random reads – IOPS limited
  - Move search closer to the storage

Exascale I/O requirements

- Filesystem integrity
  - Foundational component of system resilience
  - End-to-end storage system data and metadata integrity
  - Requirement – not just a feature
- Balanced fault handling strategies
  - Transactional models
    - Filesystem always exists in a defined state
    - Immediate cleanup on failure
  - Scrubbing
    - Repair minor damage that can persist for days

Exascale I/O requirements

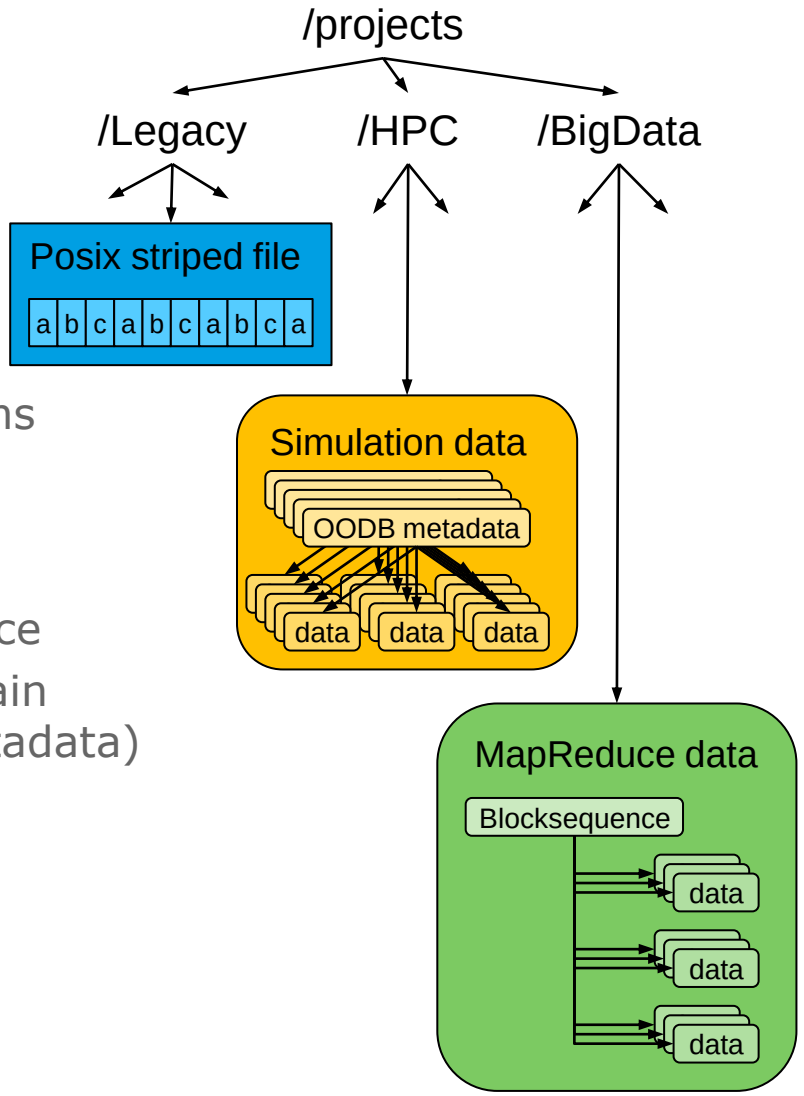
- Global v. local storage
  - Global looks like a filesystem
  - Local looks like
    - Cache / storage tier
      - How transparent?
    - Something more application specific?
- Automated / policy driven migration
  - Pre-staging from global F/S
  - Post-writeback to global F/S
- Fault isolation
  - Transactionality
  - Idempotence
- Performance isolation
  - Qos requirements for global F/S

Software engineering

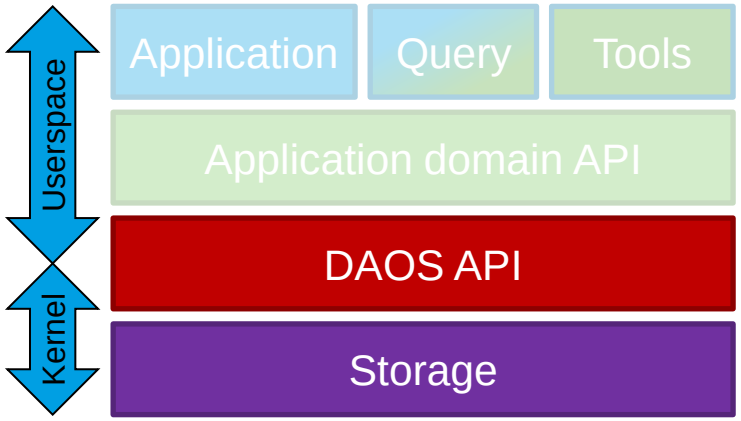
- Stabilization effort required non-trivial
  - Expensive/scarce scale development and test resources
- Build on existing components when possible
  - LNET (network abstraction), OSD API (backend storage abstraction)
- Implement new subsystems when required
  - Distributed Application Object Storage (DAOS)
- Layered model
  - Core features
    - Simple but powerful
  - Middleware
    - Puts complexity where it's easiest to debug
    - Supports different application domains
  - Tools
    - Generic data management and administration
    - Application domain specific browsers and query engines

Exascale filesystem

- Conventional namespace
  - Works at human scale
  - Administration, security, accounting
  - Supports legacy data and applications
- DAOS Containers
  - Work at exascale
  - Embedded in conventional namespace
  - Provide storage for application domain specific object schemas (data + metadata)
- Storage pools
  - Administration and accounting
  - Data management

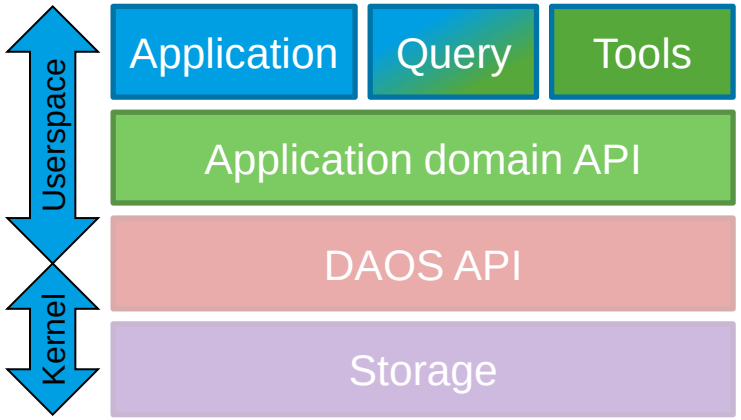


I/O stack



- DAOS Containers
  - Application data and metadata
  - Object resilience
    - N-way mirrors / RAID6
  - Data management
    - Migration over pools / between containers
  - 10s of billions of objects distributed over thousands of OSSs
    - Share-nothing create/destroy, read/write
    - Millions of application threads
  - ACID transactions on objects and containers
    - Defined state on any/all combinations of failures
    - No scanning on recovery
- DAOS API
  - Simple userspace export of DAOS functionality

I/O stack



- Userspace
  - Easier development and debug
- Application domain libraries
  - Domain-specific API style
    - Collective / independent
    - Transaction model
    - OODB, Hadoop, HDF5, Posix…
  - Implementation policy
    - Object aggregation/replication/migration
- Applications and tools
  - Backup and restore
  - Query, search and analysis
  - Data browsers, visualisers and editors
  - General purpose or application specific according to target APIs

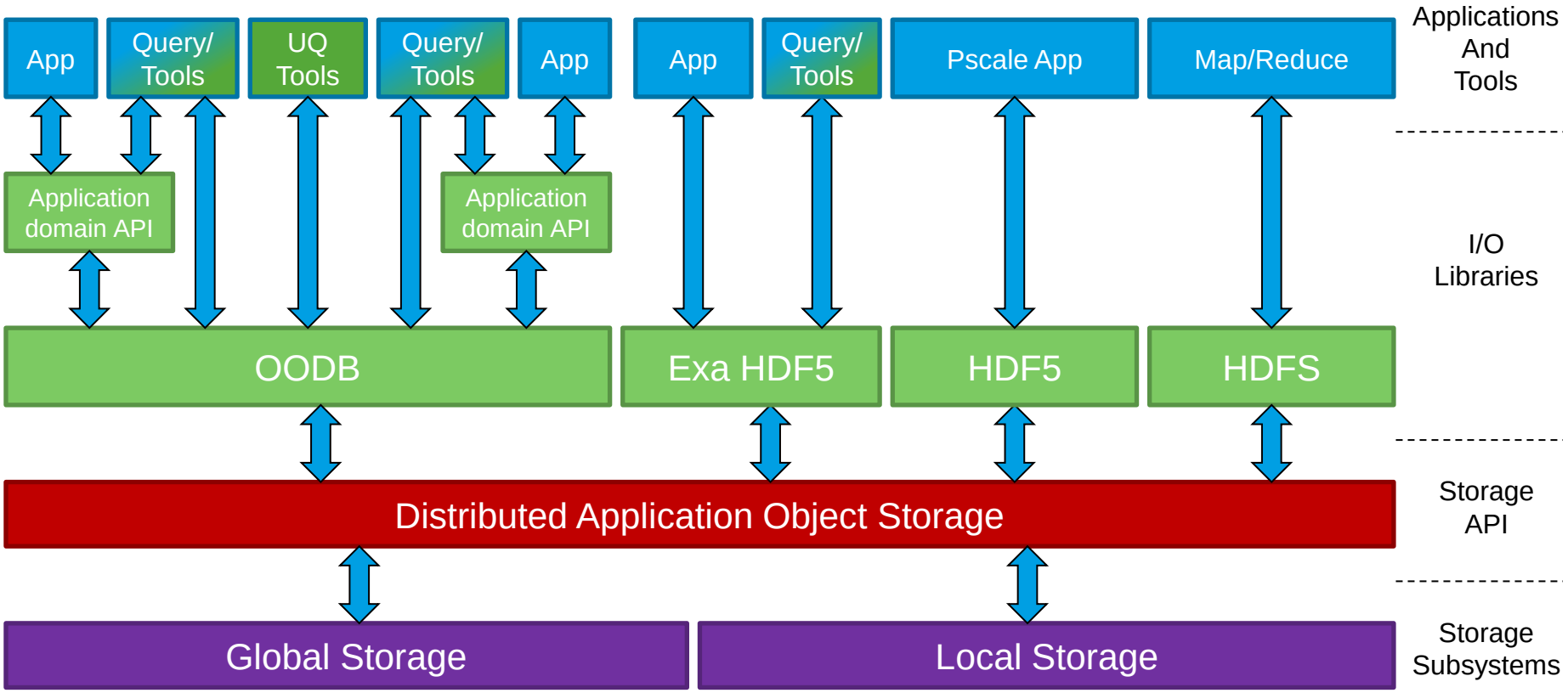
Core components

- Unified storage pools
  - Combine MDT/OST functionality, administer by usage
- Health network
  - Timely failure notification & server collectives
- Userspace DLM API
  - Locking support for schema implementations
- DAOS
  - Sharded object index
  - Accounting stats aggregation
  - Collective open
  - Container merge/split/duplicate/migrate
  - Userspace API

Libraries and tools

- Schemas
  - OODB
    - Generic base model
    - Multiple application domains
      - CFD / genome /
      - Oil and gas
  - Hadoop
  - HDF5 / Exa HDF5
- DAOS browser / editor / debugger
  - Generic + schema-specific
- Server-side query runtime
  - VM sandbox to contain scheduler + query engines

Exascale I/O stack



Thank You

- Eric Barton
  CTO
  Whamcloud, Inc.
  eeb@whamcloud.com