Leveraging Lustre to address I/O Challenges of 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

<table>
<thead>
<tr>
<th></th>
<th>2012</th>
<th>2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>10-100K</td>
<td>100K-1M</td>
</tr>
<tr>
<td>Threads/node</td>
<td>~10</td>
<td>~1000</td>
</tr>
<tr>
<td>Total concurrency</td>
<td>100K-1M</td>
<td>100M-1B</td>
</tr>
<tr>
<td>Memory</td>
<td>1-4PB</td>
<td>30-60PB</td>
</tr>
<tr>
<td>FS Size</td>
<td>10-100PB</td>
<td>600-3000PB</td>
</tr>
<tr>
<td>MTTI</td>
<td>1-5 Days</td>
<td>6 Hours</td>
</tr>
<tr>
<td>Memory Dump</td>
<td>&lt; 2000s</td>
<td>&lt; 300s</td>
</tr>
<tr>
<td>Peak I/O BW</td>
<td>1-2TB/s</td>
<td>100-200TB/s</td>
</tr>
<tr>
<td>Sustained I/O BW</td>
<td>10-200GB/s</td>
<td>20TB/s</td>
</tr>
<tr>
<td>Object create</td>
<td>100K/s</td>
<td>100M/s</td>
</tr>
</tbody>
</table>
Exascale I/O technology drivers

• (Meta)data explosion
  – Many billions of entities
    • Mesh elements
    • Graph nodes
    • Timesteps
  – Complex relationships
  – UQ ensemble runs

• OODB
  – Read/Write -> Instantiate/Persist
  – Index / Search
    • Where’s the 100 year wave
  – Data provenance + quality

• Storage Management
  – Migration / Archive
Exascale I/O Architecture

Exascale Machine

- Compute Nodes
- Exascale Network
- I/O Nodes
- Burst buffer NVRAM
- Storage Servers
- Metadata NVRAM

Site Storage Network

Shared Storage

- Disk
Exascale I/O requirements

- **Concurrency**
  - Death by 1,000M cuts
    - Scattered un-aligned variable size data structures
    - Asynchronous I/O
  - I/O Staging
    - Aggregate ~100 compute nodes x ~100-1000 threads
    - Burst buffer / pre-staging
    - “Laminar” data flow to global file system
      - Object-per-staging process

- **Search & Analysis**
  - Multiple indexes
  - Ad-hoc index creation
  - Pre-stage data for analysis
    - Subset determined by ad-hoc query
Exascale I/O requirements

- (Meta)data consistency + integrity
  - Metadata at one level is data in the level below
  - Foundational component of system resilience
  - Required end-to-end

- Balanced recovery strategies
  - Transactional models
    - Fast cleanup up failure
    - Filesystem always available
    - Filesystem always exists in a defined state
  - Scrubbing
    - Repair / resource recovery that may take days-weeks
Exascale I/O requirements

- **Global v. local storage**
  - Global looks like a filesystem
  - Local looks like
    - Cache / storage tier
      - How transparent?
    - Something more specific?

- **Automated / policy / scheduler driven migration**
  - Pre-staging from global F/S
  - Post-writeback to global F/S

- **Fault isolation**
  - Massive I/O node failure cannot affect shared global F/S

- **Performance isolation**
  - I/O staging nodes allocated per job
  - Qos requirements for shared global F/S
Software engineering

- Stabilization effort required non-trivial
  - Expensive/scarcce 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)
- Clean stack
  - Common base features in lower layers
  - Application-domain-specific features in higher layers
  - APIs that enable concurrent development
Exascale shared filesystem

• Conventional namespace
  - Works at human scale
  - Administration
    • Security & accounting
    • Legacy data and applications

• DAOS Containers
  - Work at exascale
  - Embedded in conventional namespace
  - Scalable storage objects
  - App/middleware determined object namespace

• Storage pools
  - Quota
  - Streaming v. IOPS
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
I/O stack

- **Userspace**
  - Easier development and debug
  - Low latency / OS bypass

- **Middleware**
  - Domain-specific API style
    - Collective / independent
    - Transaction model
    - OODB, Hadoop, HDF5, Posix...
  - I/O staging / burst buffers

- **Applications and tools**
  - Backup and restore
  - Query, search and analysis
  - Data browsers, visualisers and editors
  - General purpose or application specific according to target APIs
Thank You

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