Fujitsu’s Contribution to the Lustre Community

Sep.24 2014
Kenichiro Sakai, Shinji Sumimoto
Fujitsu Limited, a member of OpenSFS
Outline of This Talk

- Fujitsu’s Development and Contribution Policies
  - Fujitsu’s Lustre Contribution Policy
  - Contribution plan
  - Roadmap

- Introduction of Contribution Features
  - IB Multi-Rail
  - Automated evict recovery
  - Directory Quota
  - Improving single process I/O performance
  - Client QoS

- Challenges Toward Exascale Era
  - Concerns for exascale file system
Fujitsu’s Development and Contribution Policies

- Fujitsu’s Lustre Contribution Policy
- Contribution plan
- Roadmap
Fujitsu’s Lustre Contribution Policy

- Fujitsu will open its development plan and feed back its enhancement to Lustre community
  - LAD is the most suitable place to present and discuss.

- Fujitsu’s basic contribution policy:
  - Opening development plan
  - Feeding back its enhancement to Lustre community no later than after a certain period when our product is shipped.

![Diagram of Fujitsu’s Lustre Contribution Policy](image-url)
Contribution Plan

- Fujitsu’s now porting our enhancements into Lustre 2.x
  - These features were implemented in FEFS based on Lustre 1.8
  - They’ve been used in our customer’s HPC system, including K computer
- We’ll start submitting patches for Lustre in 2015
  - Lustre 2.6 bugs are found during porting → We’ll submit their patches too

<table>
<thead>
<tr>
<th>Functions</th>
<th>Submitting Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>IB multi-rail</td>
<td>Jan. 2015</td>
</tr>
<tr>
<td>Automated Evict Recovery</td>
<td>Apr. 2015</td>
</tr>
<tr>
<td>Directory Quota</td>
<td>2nd half of 2015</td>
</tr>
<tr>
<td>Improving Single Process I/O Performance</td>
<td>2nd half of 2015</td>
</tr>
<tr>
<td>Client QoS</td>
<td>2nd half of 2015</td>
</tr>
<tr>
<td>Server QoS</td>
<td>TBD</td>
</tr>
<tr>
<td>Memory Usage Management</td>
<td>TBD</td>
</tr>
</tbody>
</table>

Details are described in later slides
## Roadmap

- Fujitsu’s development and community feedback plan
  - Schedule may change by Fujitsu’s development/marketing strategy

<table>
<thead>
<tr>
<th></th>
<th>CY2014</th>
<th>CY2015</th>
<th>CY2016</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fujitsu</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Porting FEFS features</td>
<td>Enhancement (TBD: Snapshot, etc.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>into Lustre2.x</td>
<td></td>
</tr>
<tr>
<td><strong>OpenSFS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feed-back</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lustre</td>
<td>2.6</td>
<td>2.7</td>
<td>2.8</td>
</tr>
<tr>
<td></td>
<td>- IB Multi-rail</td>
<td>- Directory Quota</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Evict recovery</td>
<td>- Single process I/O</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Client QoS</td>
<td></td>
</tr>
</tbody>
</table>
Introduction of Contribution
Features

- IB Multi-Rail
- Automated evict recovery
- Directory Quota
- Improving single process I/O performance
- Client QoS
IB Multi-Rail

- Improves LNET throughput and redundancy using multiple InfiniBand (IB) interfaces

- Improving LNET throughput
  - Using multiple IB interfaces as single Lustre NID
  - LNET B/W improves in proportion to the number of IBs on single Lustre node

- Improving Redundancy
  - LNET can continue communicating unless all IBs fail
  - MDS/OSS failover is not necessary when a single point IB failure occurs

---

**Client**

- HCA0 ib0(192.168.0.10)
- HCA1 ib1(192.168.0.11)

**Server (MDS/OSS)**

- HCA0 ib0(192.168.0.12)
- HCA1 ib1(192.168.0.13)

**Single LNET**

(network=o2ib0)

NID=192.168.0.10@o2ib0

NID=192.168.0.12@o2ib0
IB Multi-Rail: Related Work (OFED level)

- **IPoIB bonding**
  - OFED has this function already
  - → RDMA isn’t supported

- **RDMA bonding**
  - Ongoing work by Mellanox
  - OFED will support RDMA bonding (I’m not sure when...)
  - → Our IB multi-rail function might be unnecessary in the future

- **IB partition method**
  - Mr.Ihara (DDN) presented at LUG 2013
  - Multiple bond interfaces are enabled with IPoIB child interfaces
  - → Requiring multiple LNET, configurations are complex

- **At the moment, our approach seems to be better**
IB Multi-Rail: Implementation

- Implemented in LND (ko2iblnd)
  - Other Lustre modules are not changed
  - Keep compatibility with old version of Lustre

- Multiple IB HCAs are handled as single NID
  - Enable constructing single LNET network

- All IBs are active
  - ko2iblnd selects transmission path by round-robin order
  - Multiple LNET requests are transmitted by using all IB paths in parallel
IB Multi-Rail: How to Use

- Combining single NID width multiple IB interfaces

Client

<table>
<thead>
<tr>
<th>HCA0</th>
<th>ib0(192.168.0.10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCA1</td>
<td>ib1(192.168.0.11)</td>
</tr>
</tbody>
</table>

Single LNET (network=o2ib0)

Server (MDS/OSS)

<table>
<thead>
<tr>
<th>HCA0</th>
<th>ib0(192.168.0.12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCA1</td>
<td>ib1(192.168.0.13)</td>
</tr>
</tbody>
</table>

NID=192.168.0.10@o2ib0

NID=192.168.0.12@o2ib0

- LNET setting (modprobe.conf)

options lnet networks=o2ib0(ib0,ib1)

- NID/IPoIB definition

# lctl --net o2ib0 add_o2ibs 192.168.0.10@o2ib0 192.168.0.10 192.168.0.11 → Client
# lctl --net o2ib0 add_o2ibs 192.168.0.12@o2ib0 192.168.0.12 192.168.0.13 → Server

- Display multi-rail information

# lctl --net o2ib0 show_o2ibs
192.168.0.10@o2ib0 192.168.0.10 192.168.0.11
192.168.0.12@o2ib0 192.168.0.12 192.168.0.13
IB Multi-Rail: Path Selection

- Transmission path is selected in round-robin order
- Source and destination interfaces are selected cyclically when each LNET function (LNetPut/LNetGet) is executed
IB Multi-Rail: Error Handling

Path error
- Ptlrpc resends the request that got an error
  → ko2iblnd selects next transmission path in round-robin order and sends it

Port down
- ko2iblnd removes the transmission path that uses the failed port
  → No error occurs when sending the request

![Normal Case Diagram](chart1)

![Path Error Diagram](chart2)
IB Multi-Rail: LNET Throughput

- **Server**
  - CPU: Xeon E5520 2.27GHz x2
  - IB: QDR x2 or FDR x2

- **Result**
  - B/W almost scales by #IBs
  - Achieves nearly HW performance

---

**LNET Self-Test QDR**

- 2.9 GB/s (read)
- 3.0 GB/s (write)
- 5.2 GB/s (read)
- 5.7 GB/s (write)

**LNET Self-Test FDR**

- 6.0 GB/s (read)
- 6.1 GB/s (write)
- 10.8 GB/s (read)
- 11.4 GB/s (write)

(Concurrency=32)
IB Multi-Rail: I/O Throughput of Single OSS

**OSS/Client**
- CPU: Xeon E5520 2.27GHz x2
- IB: QDR x2

**OST**
- ramdisk x8 (> 6GB/s)

**IOR**
- 32-process (8client x4)

**Result**
- Throughput almost scales by #IBs
- Measurement of FDR is planned

---

![Diagram](image_url)
Directory Quota (DQ for short)

- Manages maximum files and disk usages for each directory
  - All files/subdirectories under DQ-enabled directory are under control
  - Can not be set to subdirectories under DQ-enabled directory
    - Because of simplicity of implementation and performance

- Implemented on top of the Lustre’s Quota framework
  - UID/GID Quota can be used along with DQ
  - Keep compatibility with current Lustre
    - mkfs isn’t needed to upgrade PKG
    - Old version of clients can access DQ-enabled directory
      - DQ is not effective to the old version of clients
Directory Quota: How to Use

- Operations are same as Lustre’s UID/GID Quota
  - Only “quotacheck” operation differs

- Set DQ on target directory (=DQ-directory)
  - `# lfs quotacheck -d <target dir>`
    - Counts the number of inodes&blocks of existing files under DQ-directory

- Set limits of inodes and blocks
  - `# lfs setquota -d <target dir> -B <#blk> -I <#inode> <mountpoint>`

- Enable limiting by DQ
  - `# lctl conf_param <fsname>.quota.<ost|mdt>=<ugd>`
  - `# lctl set_param -P <fsname>.quota.<ost|mdt>= <ugd>`

- Check status
  - `# lctl get_param osd-*.*.quota_slave.info`
Directory Quota: Implementation

- Existing processes of UID/GID Quota are used as far as possible
  - Add some data structures that stores DQ information
  - Keep compatibility with Idiskfs disk layout
- Introduce new ID for DQ (=DID)
  - DID = inode number of DQ-enable directory
  - DID is stored in Idiskfs inode of MDT/OST object files
- Index/account files for DQ are added
  - Usages/limits of the number of inodes/blocks are managed
    - index file: created at first mount
    - account file: created at mkfs
      - Upgrading from no DQ PKG, execute “tunefs.lustre --dirquota”
- ZFS is not supported
  - We don’t have plan to implement DQ in ZFS
Directory Quota: DQ Information

- DID is stored in unused area of ldiskfs inode
  - i_ctime_extra and i_mtime_extra are used
- DQ’s index/account files are created on MDTs/OSTs
- Some flags to identify DQ are added
Evict Recovery

- Recovers from evicted-state automatically while disabling periodical pinging (in Lustre 2.4 or later)

**Issue**
- While disabling periodical pinging, clients cannot notice it’s eviction
- First I/O request from the client to the server gets an error (EIO)

**Approach**
- Reconnect automatically when an eviction occurred
- Server make evicted client send ping request to the server

**Effect**
- Evicted period is shorten → Frequency of I/O error is minimized
Evict Recovery: Basic Mechanism

Evict recovery process:
1. When a server evicts a client, the server notifies MGS
2. MGS notifies the evicted client to connect the server
3. The client sends ping request to the server
Evict Recovery: Sequences (W/O periodic ping)

**BEFORE**

**WITHOUT Automated Evict Recovery**

- MDS/OSS: Lock cancel, Evict client
- Client: LDLM_BL_CALLBACK, No-response

**AFTER**

**WITH Automated Evict Recovery**

- MDS/OSS: Lock cancel, LDLM_BL_CALLBACK
- MGS: Ping, Evict, OST_CONNECT, Connect, Reconnected
- Client: Additional Sequence, Evict client, Reconnected

- Client doesn’t notice it’s evicted

**Client status**
- FULL
- DISCON
- CONNECTING

**1st I/O request**
- Error!!
- Success!!
Improving Single Process I/O Performance

- Important for clients to write a large amount of data such as checkpoint files

- Issue
  - Striping isn’t effective to improve single process I/O performance
    - There’re some bottlenecks in Lustre’s cache method using dirty buffer for each OST

- Our Approach
  - `write()` returns immediately after copying user data to kernel buffer
  - Dedicated I/O threads transfer data from the buffer to OSS/OSTs in parallel
    → write throughput dramatically improves from user perspective
Improving Single Process I/O Performance

- Lustre 2.6.0 vs. prototype (Lustre 1.8 base)
  - We're re-designing implementation suitable for Lustre 2.x

- OSS/Client
  - CPU: Xeon E5520 2.27GHz x2
  - IB: QDR x1

- OST
  - ramdisk x4

- IOR
  - 1-process

Result

- Lustre 2.6.0 0.9~1.0GB/s
- Prototype 2.2~2.9GB/s
Client QoS (Quality of Service)

- Provides fair-share access among users on a single Lustre client

- Issue
  - I/O heavy user degrades I/O performance of other users on the same node

- Approach
  - Request Control: Restricts the max. number of requests issued by each user
    - Prevents a single user occupies requests issued by the client
  - Cache Control: Restricts the max. amount of client cache used by each user
    - Prevents a single user occupies client cache and write from other users are blocked

**WITHOUT QoS**

- Login node
  - User A
  - User B

- Client cache
  - Data
  - Data
  - Data

- Lustre Servers

**WITH QoS**

- Login node
  - User A
  - User B

- QoS for request
- QoS for cache

- Client cache
  - Data
  - Data

- Lustre Servers
Parameters for client QoS are specified by mount option

Parameters for request control
- `qos`
  - Enables request control
- `{m|r|w}usermax=n (1~16)`
  - Maximum number of meta/read/write requests that each user can issue at the same time

Parameter for cache control
- `qos_cache`
  - Enables cache control
- `dpusermax=n (1~100%)`
  - Maximum amount of client cache(*) each user can use in the client
    *per OSC (max_dirty_mb) and per client (obd_max_dirty_pages)
Client QoS: Example of Effectiveness

- Test pattern
  - dd if=/dev/zero of=/mnt/fefs/out.dat bs=1048576 count=2000 (write 2GB)
  - User A: dd x1
  - User B: dd x1~20

- Result
  - Processing time of User A is kept almost constant

 execution time becomes very long

 execution time is almost kept constant

wuserrmax=2
Challenges Toward Exascale File System

- I/O Throughput and Capacity
- Metadata Performance
- System Limits
- Memory Usage
- System Noise
Exascale Concerns: I/O Throughput&Capacity

- **Concern**
  - Requires high throughput (~10TB/s) and huge capacity (~1EB)
    - Single layered storage system won’t be able to satisfy both requirements
    - Device cost, power consumption, footprint

- **Approach**
  - Hierarchical storage system architecture
  - Use appropriate storage devices in each hierarchy

  ![Storage Hierarchy Diagram](image)

  For example:

  1st layer: SSD, fast buffer for job
  2nd layer: HDD, shared area (Lustre)
  3rd layer: Tape, archive area (Lustre-HSM)
Exascale Concerns: Metadata Performance

**Concern**

- Metadata performance will hit the limit
  - Exascale applications create several billions of files in a single job
  - E.g. One of exascale application “NICAM” creates 1.8 billion files per job

**Approach**

- Reduce metadata access to MDS
  - Provide intermediate layer to absorb metadata access between compute node and file system
  - E.g. “File composition library” by RIKEN AICS manages many files as a single file

Exascale Concerns: System Limits

**Concern**

- Capacity of file system must be exabytes class
  - E.g. One of exascale application “COCO” outputs 860PB per job
  - We’ve extended upper limits of Lustre to satisfy requirements of K computer

**Approach**

- Eliminating the restriction of logical upper limits
  - E.g. Eliminating 32-bit restriction, etc...

<table>
<thead>
<tr>
<th>System Limits</th>
<th>FEFS*</th>
<th>Lustre 2.x</th>
<th>Exa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum file system size</td>
<td>8EB</td>
<td>512PB</td>
<td>&gt; 8EB</td>
</tr>
<tr>
<td>Maximum file size</td>
<td>8EB</td>
<td>31.25PB</td>
<td>&gt; 8EB</td>
</tr>
<tr>
<td>Maximum number of files</td>
<td>8E</td>
<td>4G x#MDTs</td>
<td></td>
</tr>
<tr>
<td>Maximum OST size</td>
<td>1PB</td>
<td>128TB</td>
<td>&gt; 1PB</td>
</tr>
<tr>
<td>Maximum stripe counts</td>
<td>20,000</td>
<td>2,000</td>
<td>&gt; 8k</td>
</tr>
<tr>
<td>Maximum number of OSTs</td>
<td>20,000</td>
<td>8,150</td>
<td>&gt; 8k</td>
</tr>
<tr>
<td>Maximum number of MDTs</td>
<td>1</td>
<td>4,096</td>
<td></td>
</tr>
</tbody>
</table>
Exascale Concerns: Memory Usage

**Concern**

- Secure sufficient memory to application programs
  - Compute node of K computer ran out of memory only by mounting file system
  - We reduced memory usage drastically for K computer (2.5GB → 490MB in client)
    (reported at Lustre Developer Summit 2012)

**Approach**

- Controlling memory usage strictly
  - E.e. page cache
- Break away from scale dependency
  - E.g. number of OSTs
Exascale Concerns: System Noise (OS Jitter)

**Concern**

- Eliminating OS jitter to maximize performance of massively parallel applications
  - We took great effort to reduce system noise in K computer
    - Shortening execution time of Lustre daemons; ll_ping, ldlm_poold

**Approach**

- Introducing dedicated cores for system daemons (OS timer, file I/O, MPI, etc)
  - E.g. Fujitsu’s SPARC64 XIfx CPU for Post-FX10 provides with 2-assistant cores
    - Processing cost of daemons to be reduce?

(Reference: Hot Chips 26)
Summary

- **Fujitsu will continue to improve Lustre for exascale systems**
  - Take advantage of experience and technology obtained from development of K computer and consumer supercomputers

- **Fujitsu will open its development plan and feed back its enhancements to Lustre community**
  - Luster Developer Summit is one of the most suitable place to discuss technical matter

- **Several features will be scheduled to be contributed in 2015**
  - InfiniBand Multi-rail, Directory Quota, etc.
shaping tomorrow with you