Lustre Administrator and Developer Workshop 2016

FUJITSU

shaping tomorrow with you

DL-SNAP and Fujitsu's Lustre Contributions

Shinji Sumimoto

Fujitsu Ltd. a member of OpenSFS

Outline

- DL-SNAP
  - Background: Motivation, Status, Goal and Contribution Plan
  - What is DL-SNAP?
  - Use case and Utility Commands
  - Implementation and Evaluation
- Other Lustre Contributions
  - Directory Quota, IB Channel Bonding

Motivation, Status, Goal and Contribution Plan

FUJITSU

- Motivation:
  - Backup files on large scale file system are an issue to solve. However, existing system level backup requires large storage space and backup time.
- Status:
  - We started to develop a snapshot function, and, we have developed a prototype of the function.
- Goal of This Presentation:
  - To present our snapshot specification and the prototype implementation
  - To discuss its usability and gather user's requirements.
- Contribution Plan:
  - 1Q 2018 to Lustre community

Background of DL-SNAP

FUJITSU

- It is difficult to make backup on large scale file system.
  - PB class file system backup takes long time and requires its backup space.
- To reduce backup storage usage and backup time:
  - Using snapshot to reduce duplicate data
  - Not all file system data, selection of backup area
- Two level of backup: System level and User level

Background: System Level vs. User Level Backup

FUJITSU

- System level backup:
  - System guarantees to backup data and to restore the backup data
  - Therefore, double sized storage space or another backup device is required to guarantee data backup and restore.
  - File Services must be stopped during backup.
- User level backup:
  - User can select backup data
  - File Service does not need to be stopped.

Background: Selected User Level Backup Scheme

FUJITSU

- Customer Requirement:
  - Continuing file system service
  - Difficult to guarantees the backup data to restore in system operation
  - Providing effective backup service with limited storage space

- Therefore, user level backup scheme is selected.
  - We started to develop DL-SNAP which is user and directory level snapshot

What is DL-SNAP?

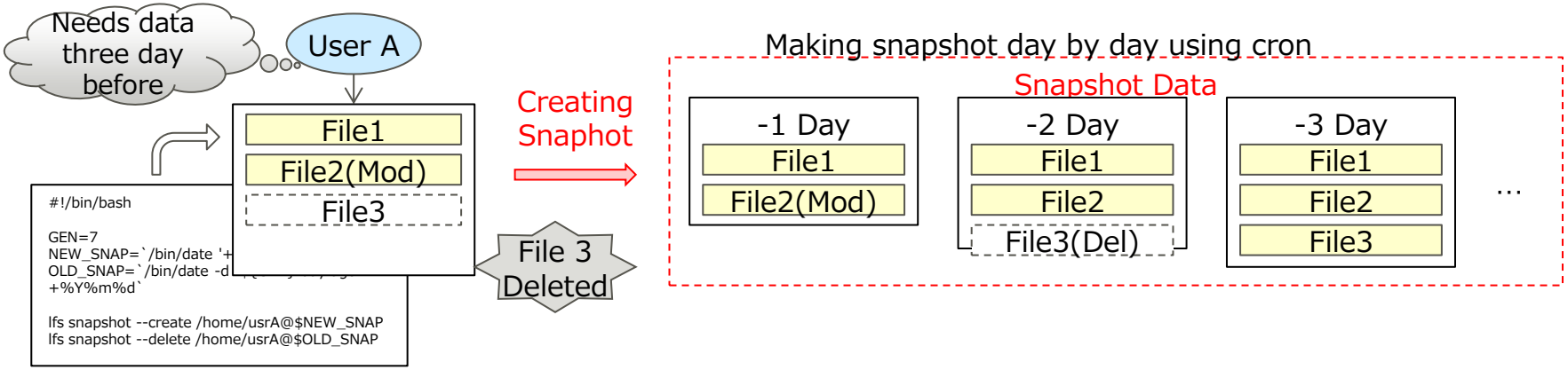
FUJITSU

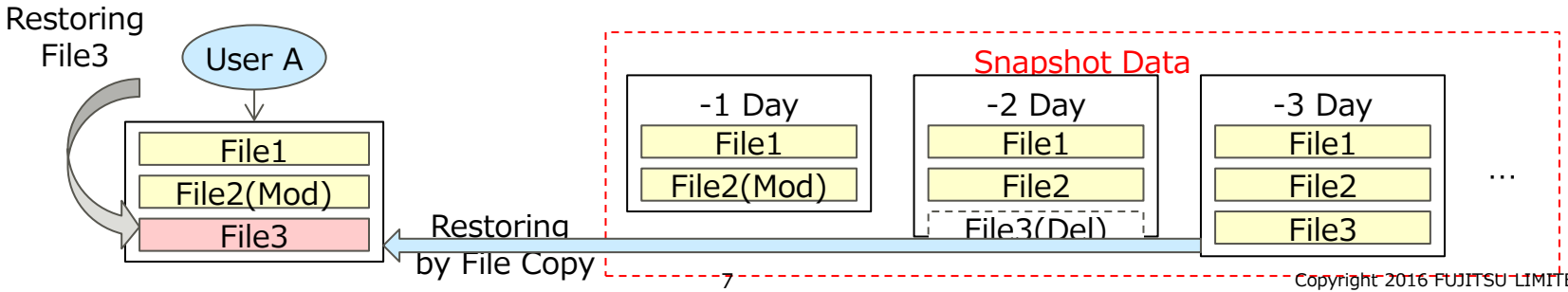
- DL-SNAP is designed for user and directory level file backups.
- Users can create a snapshot of a directory using lfs command with snapshot option and create option like a directory copy.
- The user creates multiple snapshot of the directory and manage the snapshots including merge of the snapshots.
- DL-SNAP also supports quota to limit storage usage of users.

DL-SNAP Use-case 1

FUJITSU

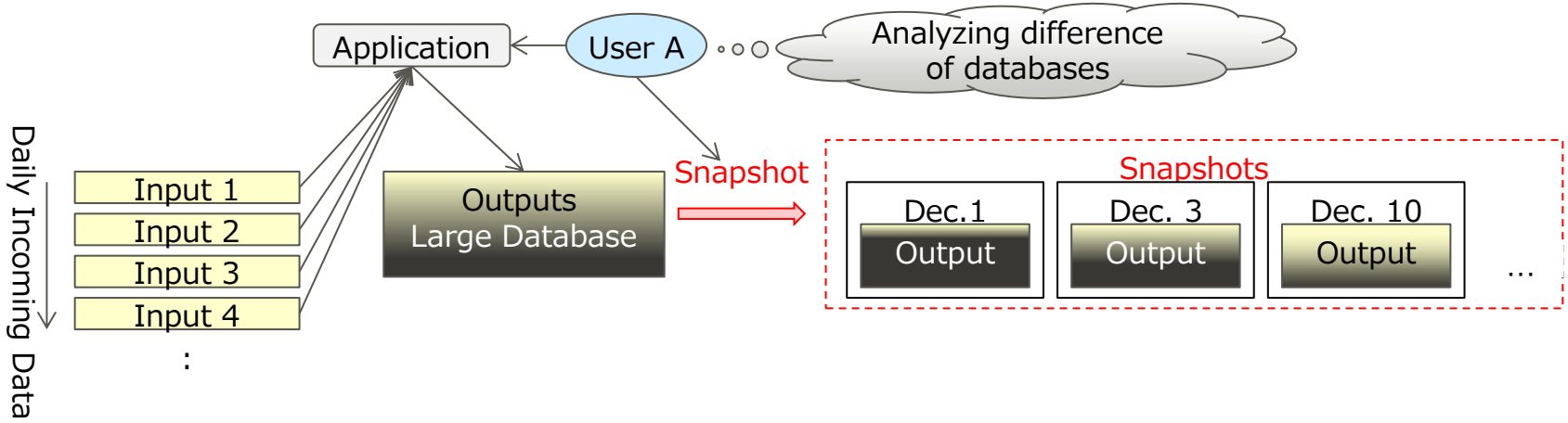
Avoiding file deletion or corruption by file operation





DL-SNAP Use-case 2

- Maintaining large database with partially different data
  - Updating database by an application using DL-SNAP



DL-SNAP: Quota Support and Utility Commands

FUJITSU

- Quota function is also provided to manage storage usage of users
  - a little bit complicate when the owner of the snapshot is different among the original and some snapshot generations.
- Utility Commands: lfs snapshot, lctl snapshot
  - Enabling Snapshot: lctl snapshot on <fsname>
  - Getting Status of Snapshot: lctl snapshot status <fsname>
  - Creating a snapshot: lfs snapshot --create [-s <snapshot>] [-d <directory>]
  - Listing snapshot: lfs snapshot --list [-R] [-d <directory>]
  - Deleting snapshot: lfs snapshot --delete [-f] -s <snapshot> [-d <directory>]

DL-SNAP Implementation

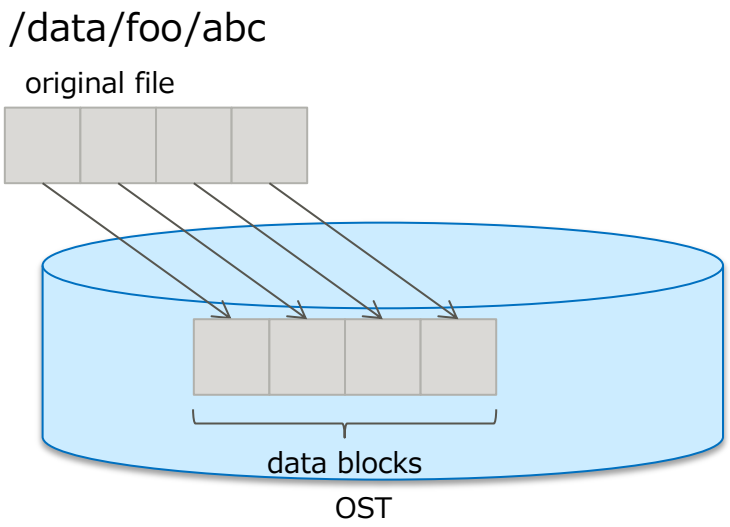
FUJITSU

- The implementation of DL-SNAP is copy on write base
  - Implemented on top of current Lustre ldiskfs and limited in OST level modification
  - Without modification of ext4 disk format
  - Adding special function to create snapshot to MDS.
- OST level modification (more detail on next page):
  - Add Function which creates extra-references on OSTs.
  - Add Copy-on-Write capability to the backend-fs.
- Two Methods to Manage Copy-on-Write Region Blocks
  - Block Bitmap Method
  - Extent Region Method (Our Method)

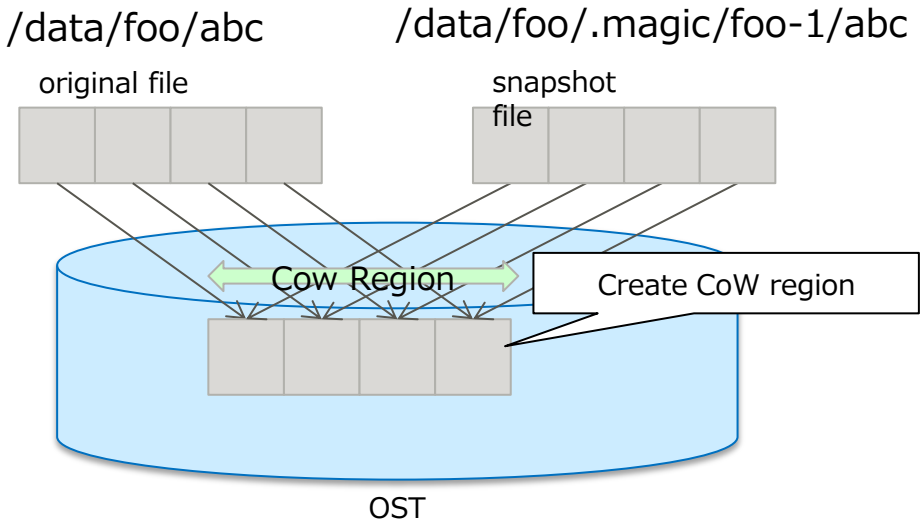
Basic Mechanism of DL-SNAP by Extent Region (1)

FUJITSU

- Initial state:
  - The original file points to the data blocks on OSTs



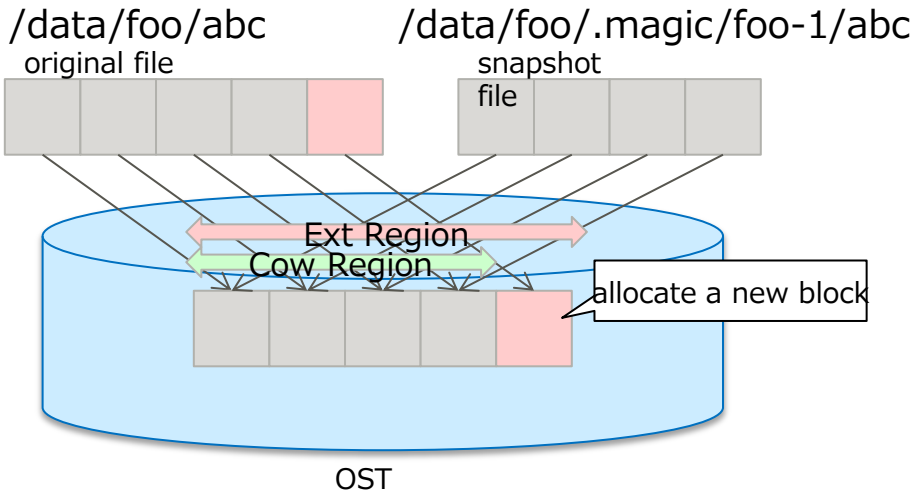
- Taking snapshot:
  - Adds another reference and it points the blocks the original file points to.



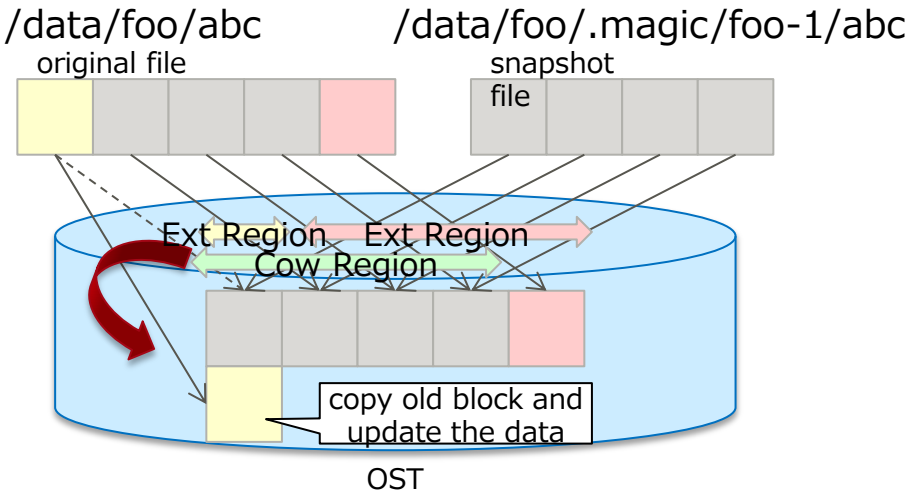
Basic Mechanism of DL-SNAP by Extent Region(2)

FUJITSU

- Append-writing the original file:
  - Allocates a new data block on the OST and writes the data to the data block. Also, creating the original file modification extent of the data block.

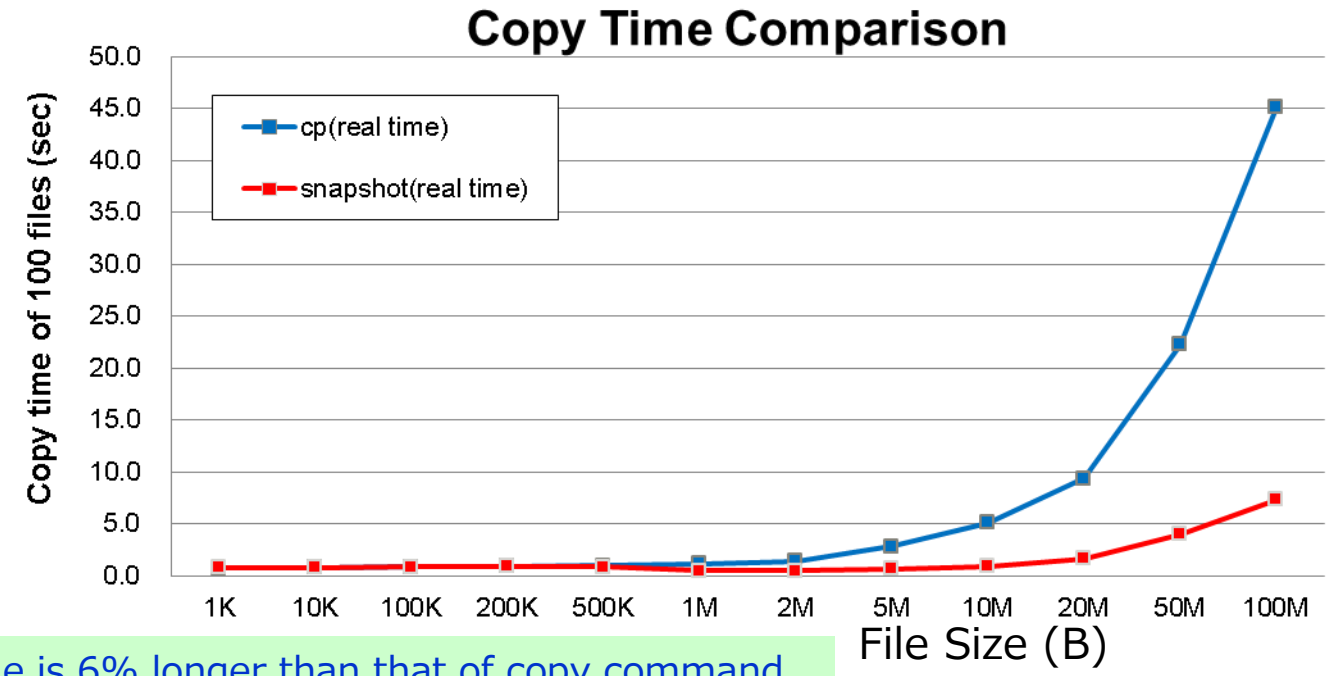


- Over-writing the original file:
  - Allocates a new data block on the OST and copy the original data block. Then, the file point the data block.



Evaluation of DL-SNAP

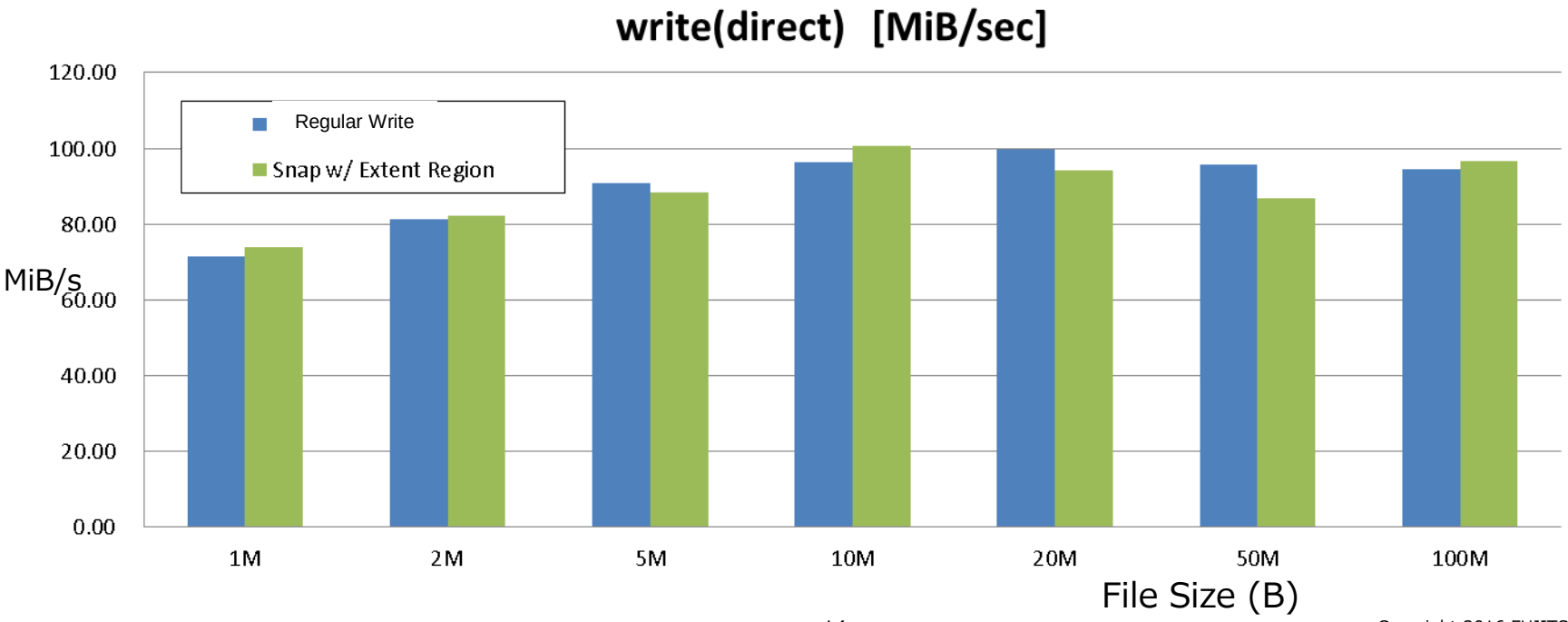
- DL-SNAP is faster than normal copy



1K byte file is 6% longer than that of copy command, but the time on 100 MB file is over 5 times faster

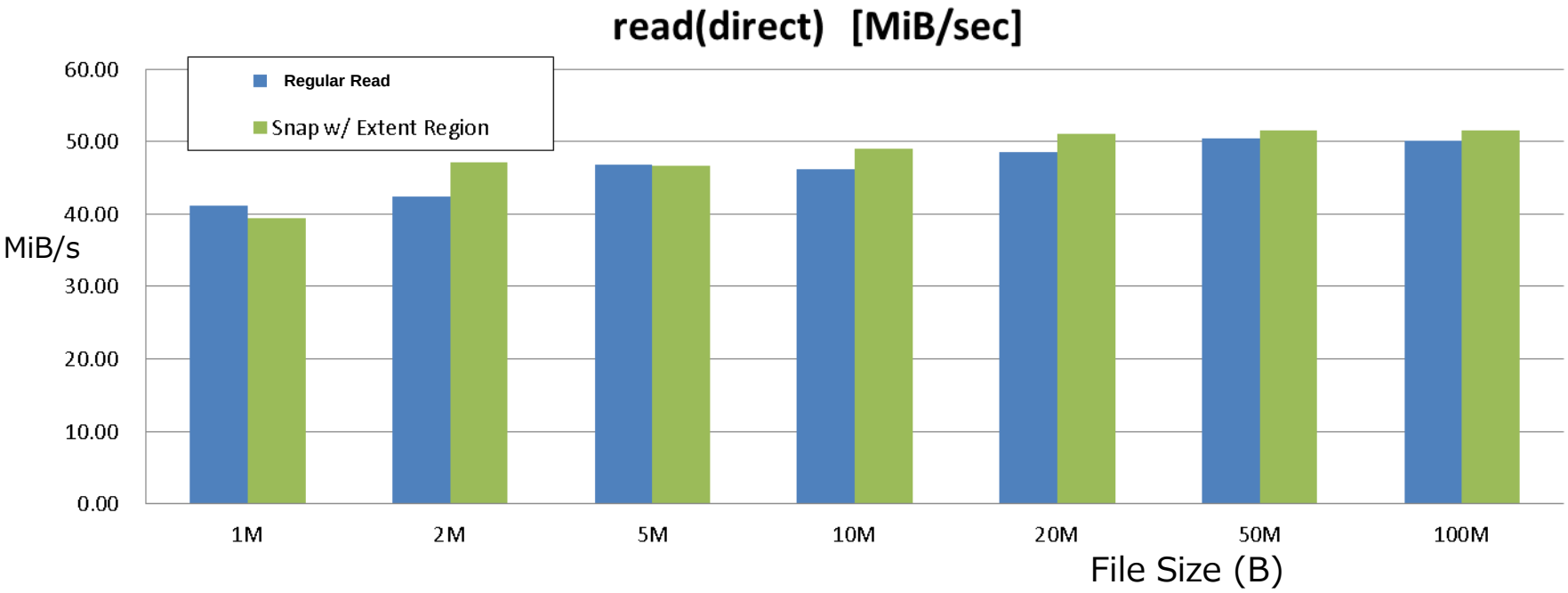
DL-SNAP: Write Performance by IOR

- Comparable performance to regular write



DL-SNAP: Read Performance by IOR

- Comparable performance to regular read



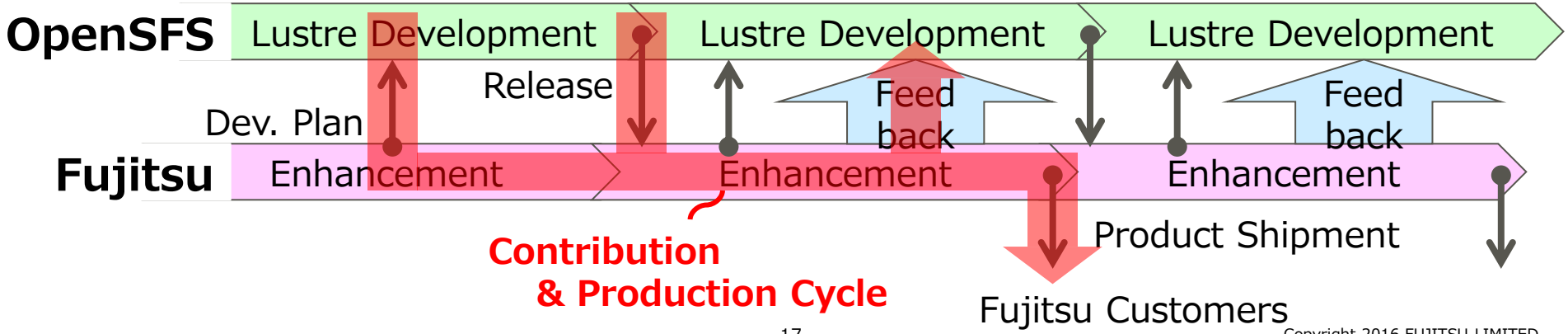
DL-SNAP: Contribution Plan and Vendor neutrality

FUJITSU

- Contribution Plan:
  - 1Q 2018 to Lustre community, several months after shipping as a product
- Vendor Neutrality:
  - The implementation of DL-SNAP is absolutely vendor-neutral because no special hardware is required and based on standard Lustre code based implementation.

Fujitsu' Lustre Contribution Policy (Presented as LAD 14)

- Fujitsu will open its development plan and feed back it's enhancement to Lustre community
- Fujitsu's basic contribution policy:
  - Opening development plan and Contributing Production Level Code
  - Feeding back its enhancement to Lustre community no later than after a certain period when our product is shipped.



DL-SNAP: Current Development Status

FUJITSU

- We are now developing DL-SNAP and evaluated its performance. The performance results show that the creating snapshot time is much better than that using copy command in longer files.
  - Creating snapshot time on 1K byte file is 6% longer than that of copy command, but the time on 100 MB file is over 5 times faster than that of copy.
- Our contribution of DL-SNAP will be planned in 1Q 2018.

Directory Quota

- Implemented for K computer 2010, and has been used on several sites both 1.8 and 2.x based Lustre.
- No need to projection ID and implemented on top of original Lustre Quota scheme.
- Planed to be contributed in 2018

Directory Quota (DQ for short)

FUJITSU

- 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

- Implemented on top of the Lustre's Quota framework
  - UID/GID Quota can be used along with DQ
  - Keep compatibility with current Lustre

    Upgrade rpm without mkfs

    Old version of clients can access DQ enabled directory

Directory Quota: How to Use

FUJITSU

- Operations are same as Lustre's UID/GID Quota
- 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>=<ug**d**>
  - # lctl set_param -P <fsname>.quota.<ost|mdt>= <ug**d**>
- Check status
  - # lctl get_param osd-*.*.quota_slave.info

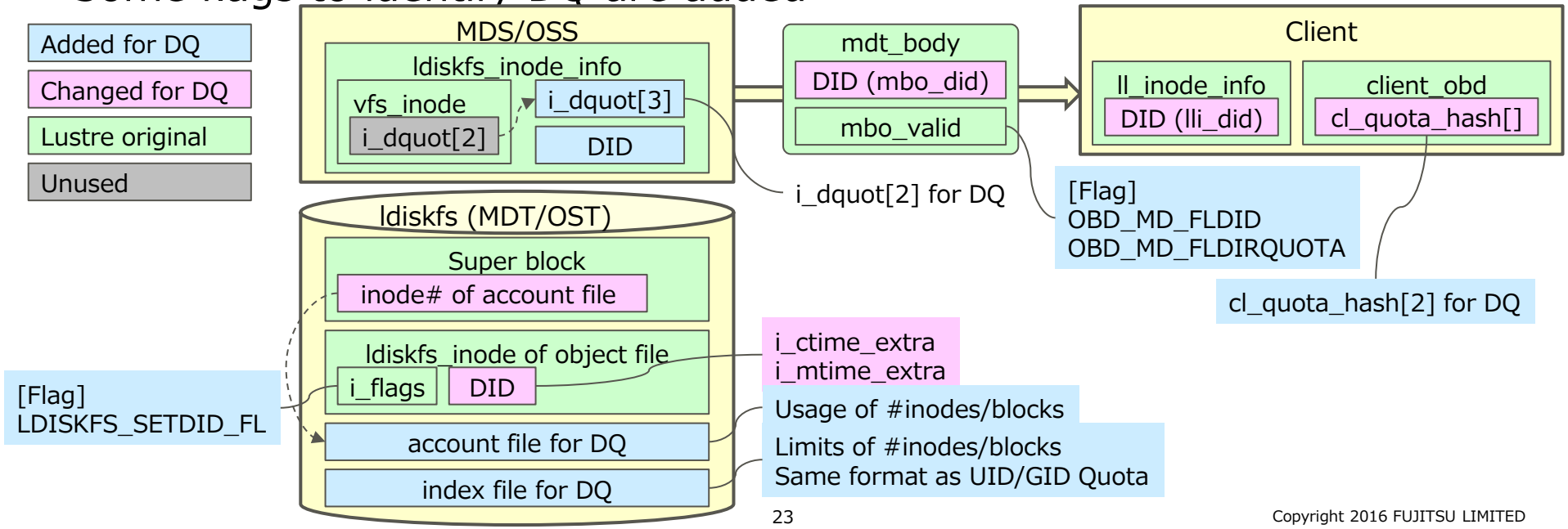
Directory Quota: Implementation

FUJITSU

- Existing processes of UID/GID Quota are used almost as it is
  - Some data structures that stores DQ information are added
  - Disk layout keeps unchanged → mkfs isn't needed to upgrade PKG
- Introduce new ID for DQ (=DID)
  - DID = inode number of DQ enable directory
  - DID is stored in ldiskfs inode of MDT/OST object files
- Index/account files for DQ are added
  - Usages/Limits of the number of inodes/blocks are managed

Directory Quota: Management 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



Current Status of Directory Quota

FUJITSU

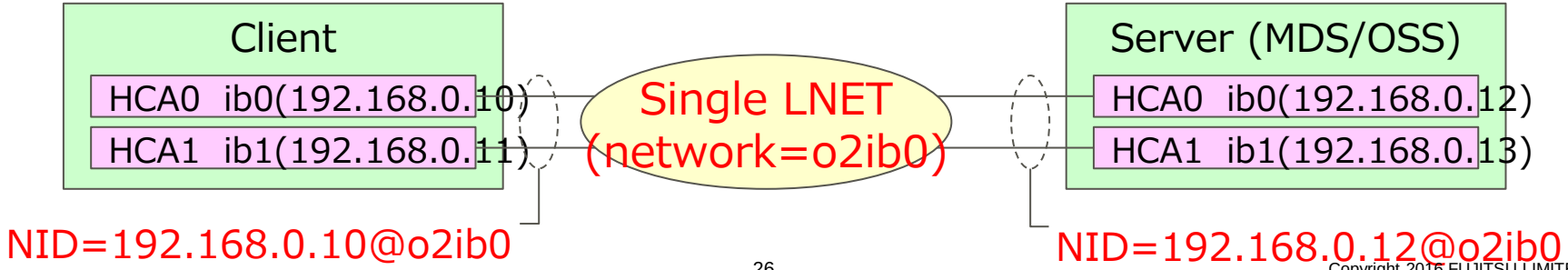
- Lustre 1.8 & 2.6 based DQ on FEFS has been providing as a product
  - Our customers use DQ function on their system operation

IB Multi-Rail

- Implemented for K computer 2010, and contributed our code to OpenSFS (LU-6531) 2015, so you can use this feature now!
- Contributed patch is independent driver, so users can easily use the patch for current Lustre.

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 occurrs



IB Multi-Rail: Related Work

FUJITSU

- OFED level
  - IPoIB bonding: OFED has this function already, but RDMA isn't supported
  - RDMA bonding: Ongoing work by Mellanox: OFED will support RDMA bonding (I'm not sure when…).
  - 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
- LNET Level
  - SGI presented LNET level multi-rail at Lustre Developer Summit 2015.
- Our approach is better in the point of having a real code to work perfectly for many years.

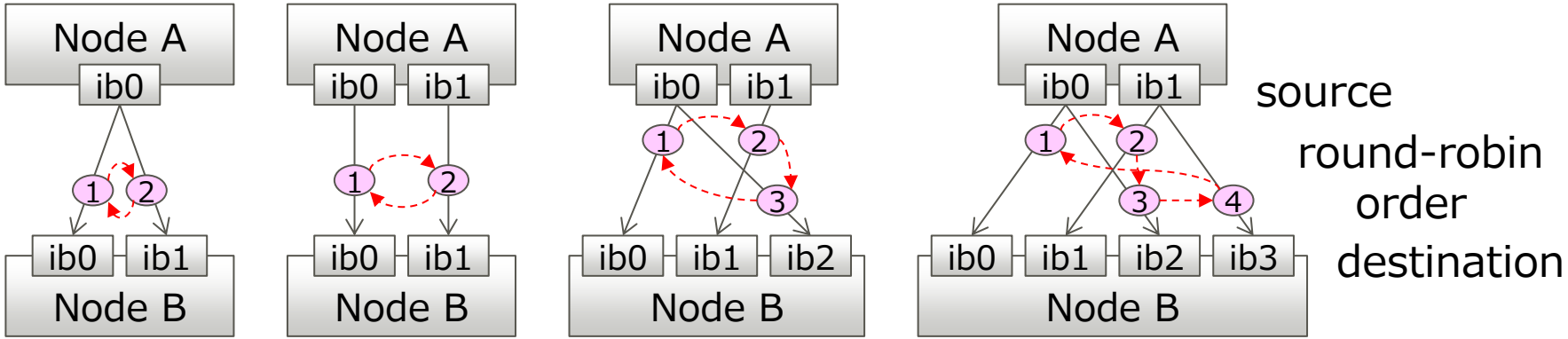
IB Multi-Rail: Implementation

FUJITSU

- Implemented in LND (ko2iblnd)
  - Other Lustre modules are not changed
  - Keep compatibility with old version of Lustre (socklnd)
- Multiple IB HCAs are handled as single NID
  - Enable constructing single LNET network
- All IB HCAs are active
  - ko2iblnd selects transmission path by round-robin order
  - Multiple LNET requests are transmitted by using all IB paths in parallel

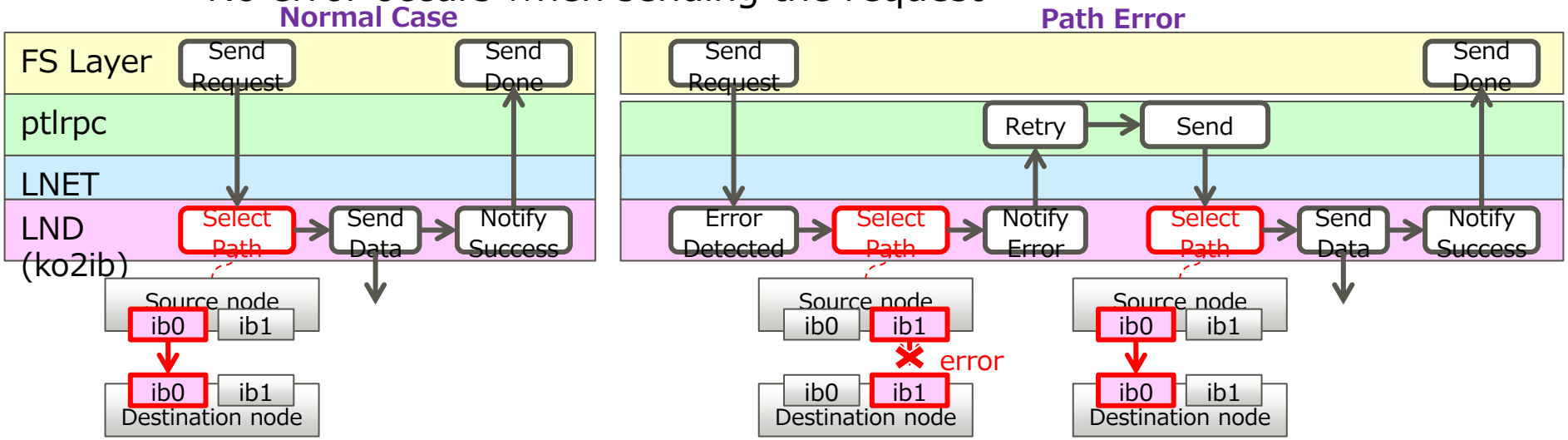
IB Multi-Rail: Path Selection

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



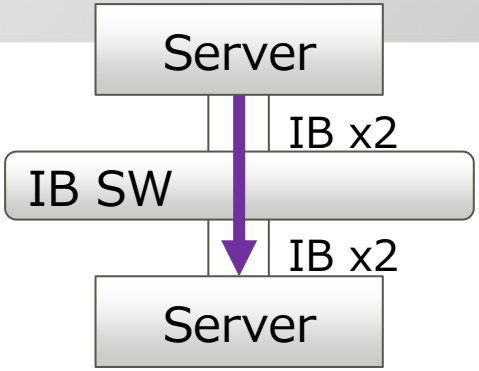
IB Multi-Rail: Error Handling

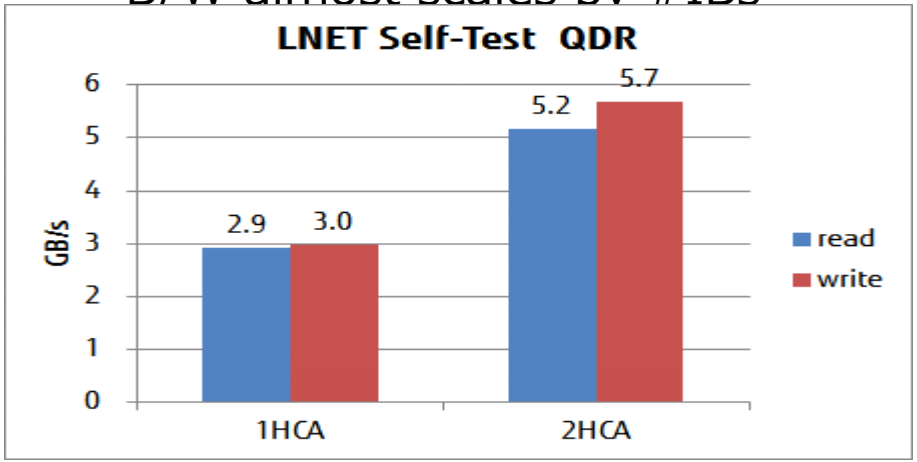
- Path error
  - Ptlrpc resends the request that got an error
    → ko2iblnd selects next transmission path in round-robin order and sends it
- Port down
  - ko2iblnd removes the transmission path that uses the failed port
    → No error occurs when sending the request

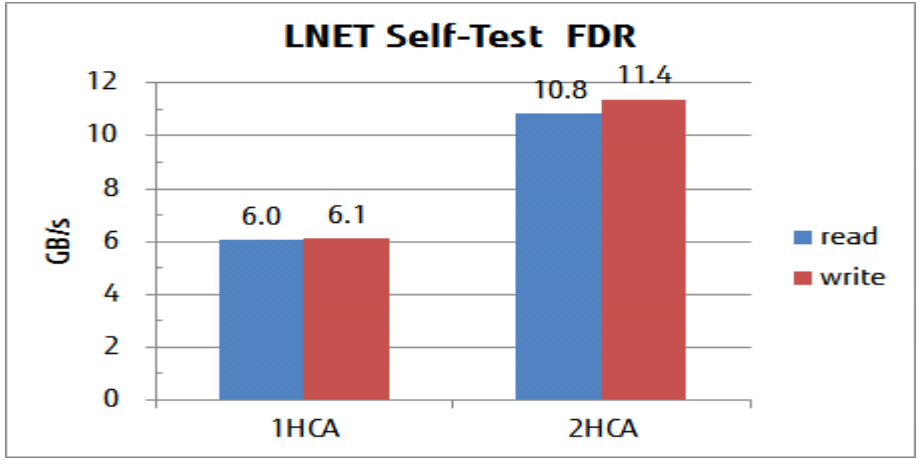


IB Multi-Rail: LNET Throughput

- Server
  - CPU: Xeon E5520 2.27GHz x2
  - IB: QDR x2 or FDR x2
- Result
  - B/W almost scales by #IBs



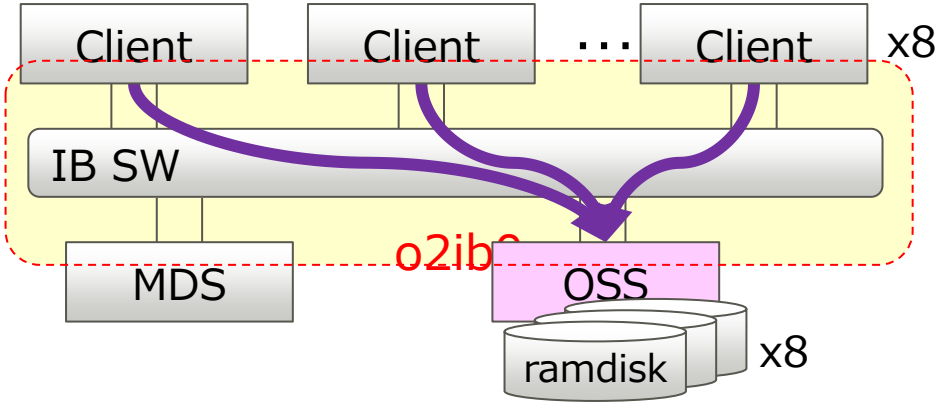




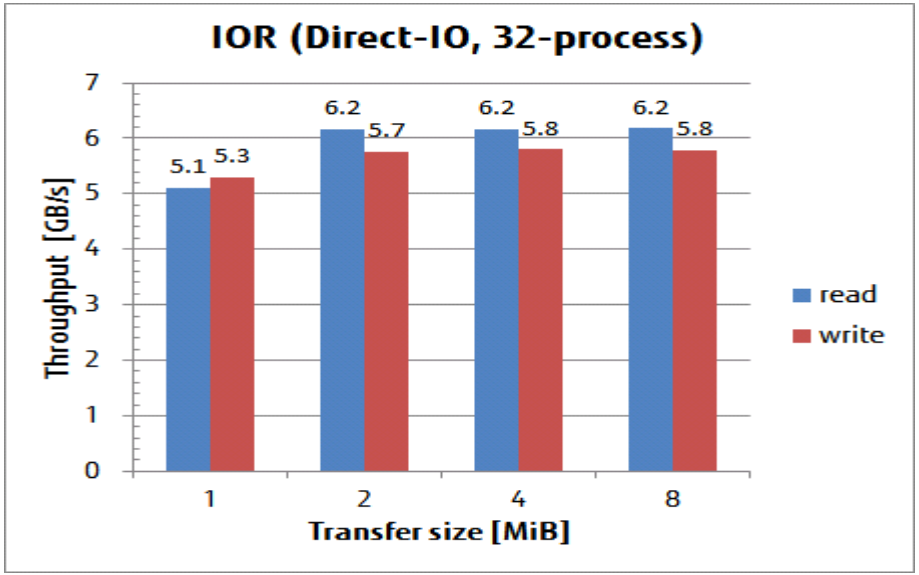
(Concurrency=32)

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



IB Multi-Rail Code Status

FUJITSU

- Our IB Multi-Rail is provided as a commercial product for over 5 years
  - K computer: over 90 OSS, 1000 class clients since 2011
  - Realizing Highly available operation for over 5 years
- OFED based implementation can be widely used for other devices
  - RoCE
  - OmniPath
  - Tofu
- We contributed our code to OpenSFS (LU-6531), 2015, so you can use this feature now!
  - Contributed patch is independent driver, so users can easily use the patch for current Lustre.

FUJITSU

shaping tomorrow with you