Enabling Research and Discovery
SRCC
Stanford University srcc.stanford.edu

# Research Computing storage systems design

## LUSTRE FEATURES AT WORK


Stéphane Thiell, Kilian Cavalotti, Ruth Marinshaw

Stanford Research Computing Center (SRCC)


LAD18
PARIS, 24.-26.09.18
LUSTRE ADMINS & DEVS WORKSHOP

# Contents

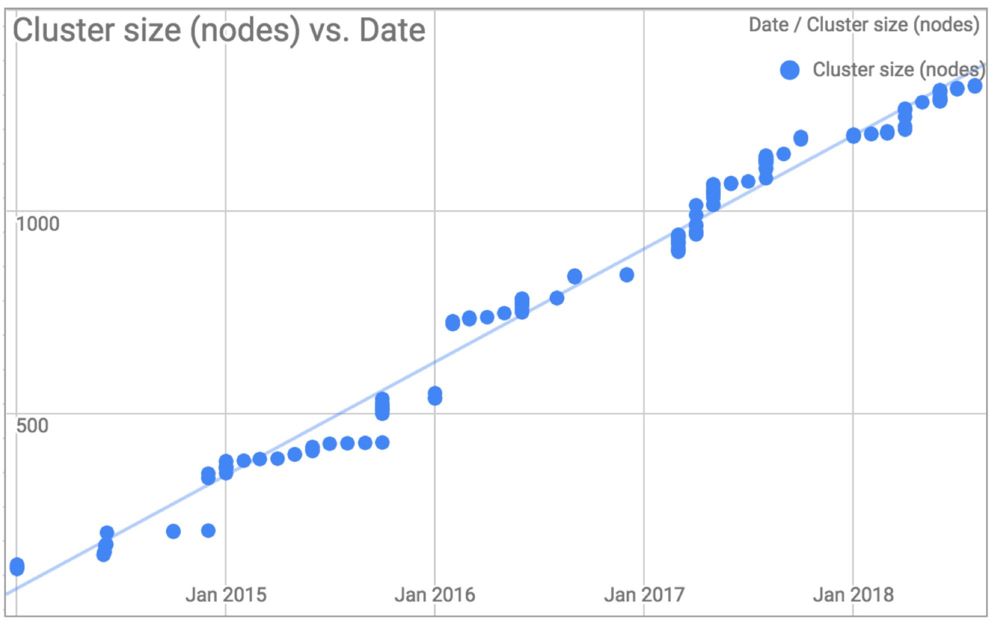
# Sherlock: SRCC’s shared computing cluster

- Support **sponsored or departmental faculty research**
- **Condo** cluster frequently growing
- **3,400 users** representing **577 different faculty research groups** (as of September 2018)
- **https://www.sherlock.stanford.edu/**

New rack (Dell C6420)

# Sherlock: tech specs (as of September 2018)

- **1,317 nodes**
- **23,756 CPU cores** and **700 GPUs**
- Two separate Infiniband fabrics: FDR and EDR
- More than **1.6 PFlops** of computing power
- CentOS 7.5, **Lustre client 2.10.5**



# SRCC's new HPC storage portfolio

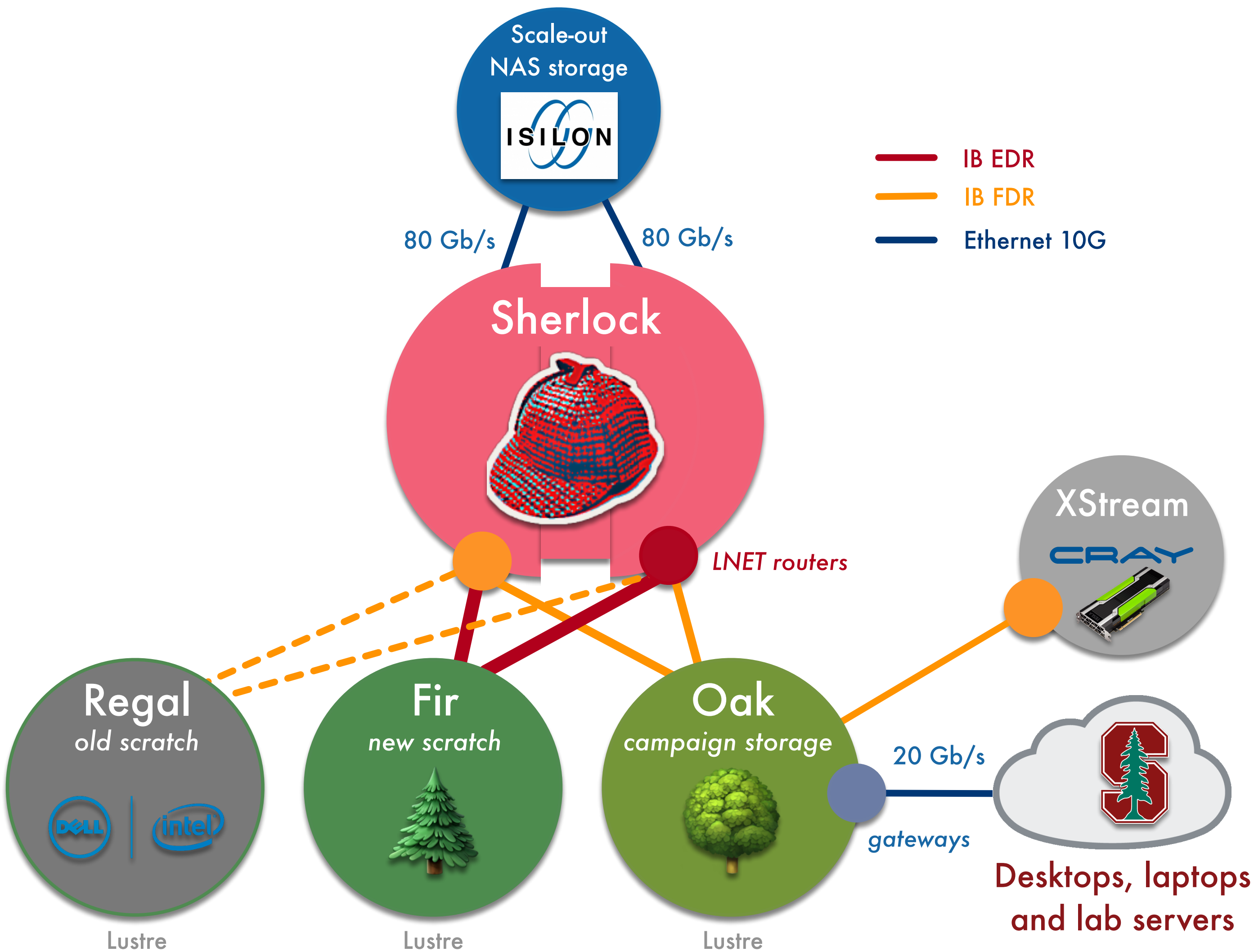
2017+

- ***Oak* longer term** HPC filesystem (Lustre 2.10)
- Mounted on Sherlock and XStream (Lustre) but also accessible through **other protocol/methods**
- Great for **large streaming I/O**

2018+

- ***Fir* scratch** HPC filesystem (targeting Lustre 2.12)
- Mounted on **Sherlock only**
- Special feature: optimized for **small file I/O**
- **High bandwidth,** great for **large streaming I/O**

# SRCC HPC storage architecture (Q4 2018)

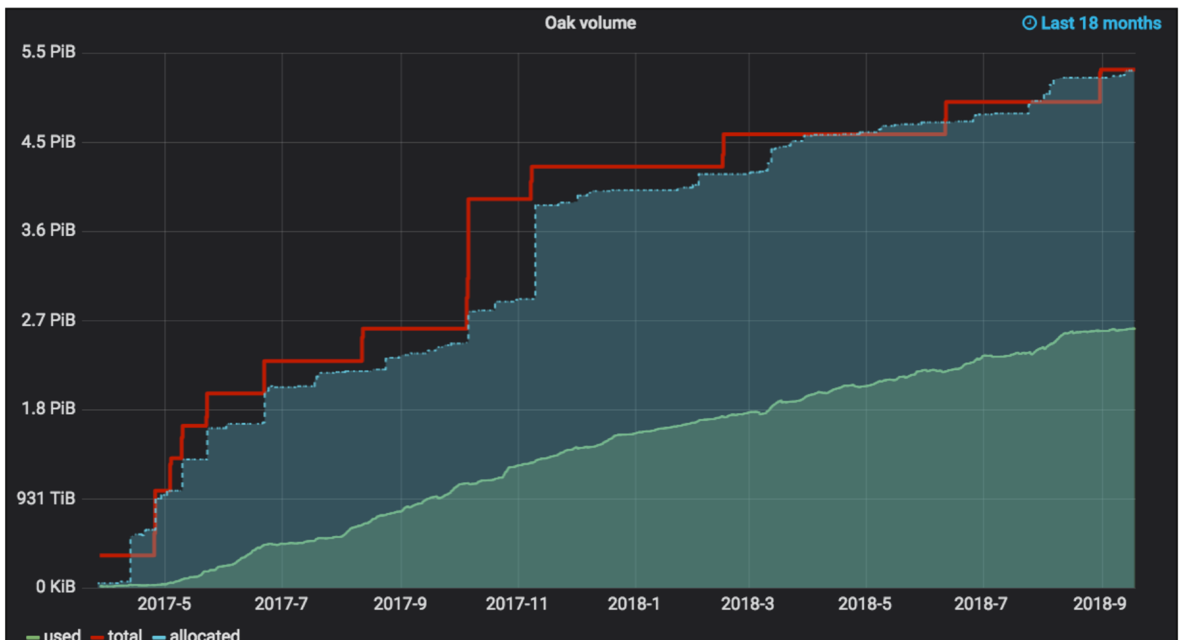


# Oak "cheap & deep" storage: Two-Year Status Report

Oak's architecture is unique and was originally presented at LAD'16

Full cost recovery after 4 years if >6PB (or 5.3PiB or 2 I/O racks)

- Growth beyond expectation with almost no publicity
- 4-year goal (6 PB usable) achieved in ~1.5 years!



# Introducing **Fir**, our next generation scratch filesystem

New parallel, scratch filesystem for our research community

- To answer the needs of **diverse workloads**
  - Traditional HPC
  - **Long-tail of science / Deep Learning**

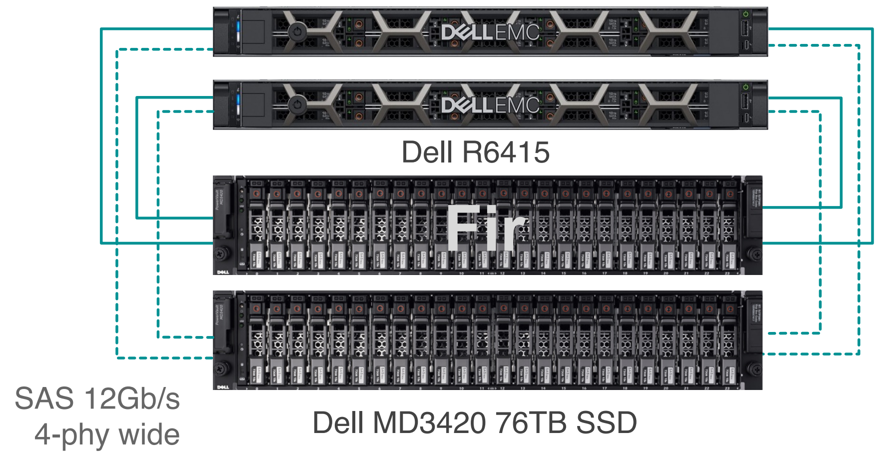
- Sponsored (unlike Oak) but **limited budget**

- Working towards using common tooling, approaches, infrastructure building blocks (when possible)




Stanford University

# Oak vs Fir: MD cell


- Pair of MDS Dell R630 (Intel)
  - 10K SAS drives
  - Infiniband FDR
- DNE ready
  - new MDT added every 6 PB (fixed inodes/volume ratio)

- Pair of MDS Dell R6415 (AMD EPYC)
  - 42 SSDs of 3.84TB each
  - Infiniband EDR
  - New Broadcom SAS 9405W-16e tri-mode HBA (4 x 48Gb/s)
- DNE and DoM ready
  - 4 MDTs of 18TB

# Fir: metadata subsystem design approach

## Main idea

- Try to better handle small files using Data-on-MDT (DoM) and SSDs

## MD cell design approach

1. Study file sizes from our old scratch “Regal” with Robinhood
2. Purchase and deploy an SSD-based MD cell based on previous sizing and budget constraints
3. Enable DoM on new filesystem accordingly

# Lustre Data-on-MDT (DoM)

Introduced in Lustre 2.11 and optimized in Lustre 2.12

The ideas behind Data-on-MDT

- Allow storing **small files on MDTs**
- Take advantage of **flash storage** often already available on MDTs
- Limit the number of small files stored on OSTs and network requests

Observations

- DoM itself doesn't automatically improve performance
- Filesystems have fewer MDTs than OSTs
- Very easy to be fooled by benchmarks (esp. on idle filesystems)

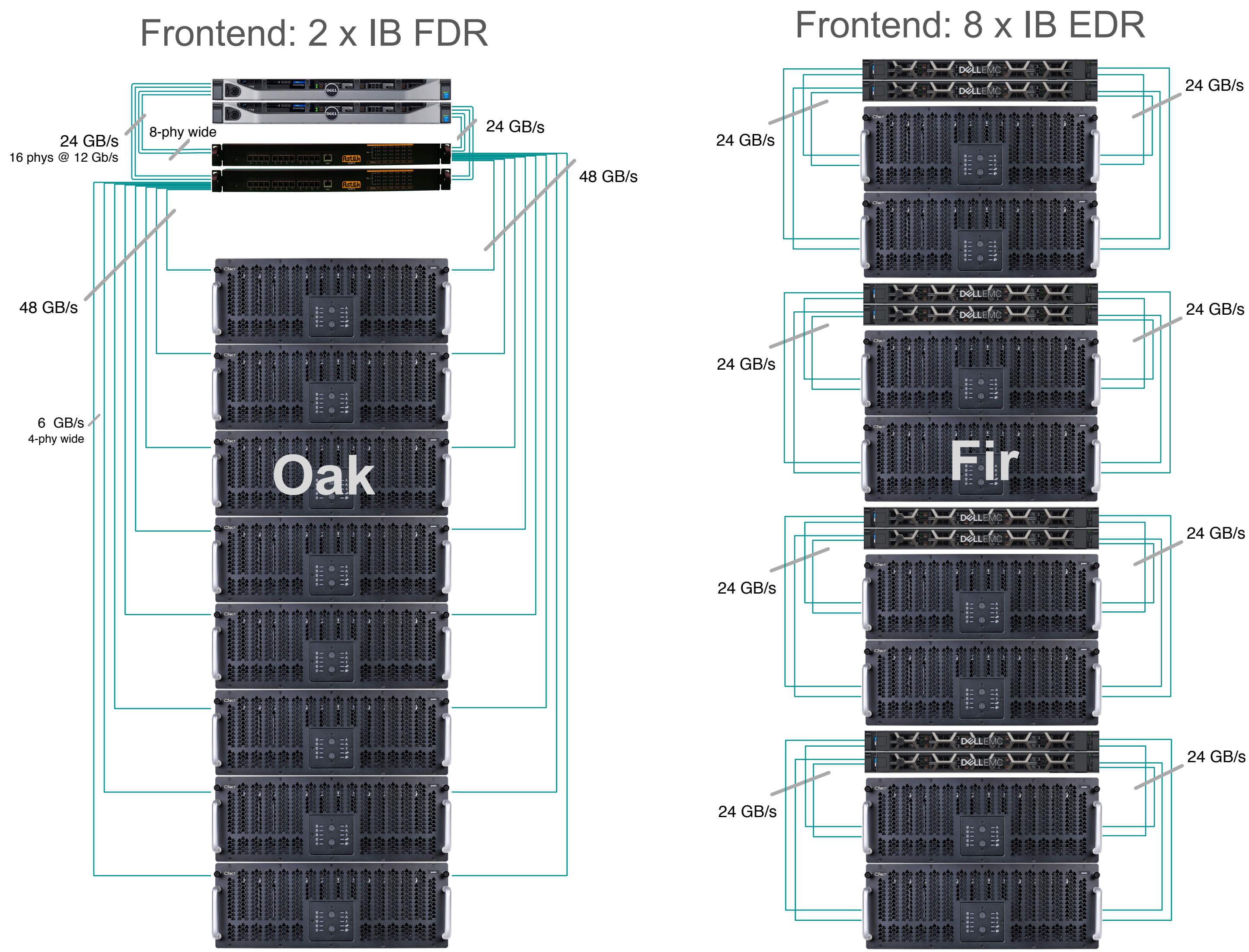
# Fir: metadata subsystem design approach (cont'd)

1. Study file sizes from our old scratch "Regal" with Robinhood:
   - 511M files, ~3PB
   - 3% of files are empty (!)
   - 44% of files are less than 4KB
   - 69% of files are less than 64KB
   - 76% of files are less than 128KB
   - 91% of files are less than 1MB

2. Purchase and deploy SSD-based disk arrays (budget constrained)
   - 153TB of SSD (raw) or 72TB usable (RAID-10)
     - 1TB reserved for 1B inodes (bytes-per-inode=66560)
     - 71TB reserved for blocks (enough to store 595M x 128KB stripes)

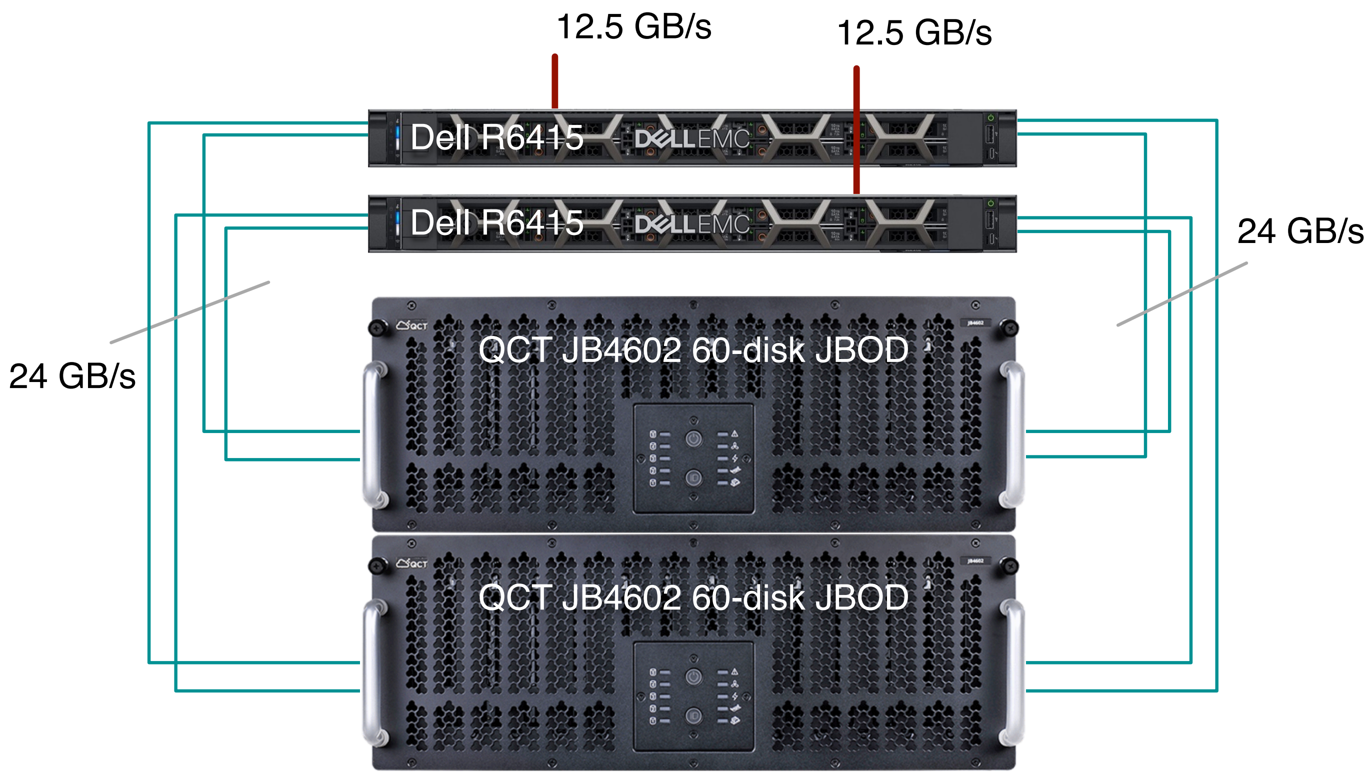
3. Set DoM stripe size to **128KB** (initially)
   - ¾ of future scratch files expected to be stored on MDTs only!

# Oak vs Fir: IO racks



# Fir: IO cell details

- Balanced bandwidth vs. capacity with a focus on streaming I/O workload
  - 2 x Dell R6415 AMD EPYC servers (mono socket 7401P)
  - 2 x QCT JB4602 JBOD (SAS 12Gb/s)
  - 120 x 8TB SAS drives



# DIY Lustre systems: lessons learned

Pro's

- Cost of hardware
- Flexibility in terms of system expansion
- Full software stack control, with better upgrade management
- Can deploy new Lustre features ahead of HPC storage vendors
- Do not rely on a single vendor to troubleshoot issues
- Good experiences with hardware vendors (firmware fixes)
- Direct Lustre support on Oak has been super helpful

Con's

- Hardware sourcing issues
  - Purchasing small quantity of units at a time sometimes leads to a long lead time
  - Switching to another hardware vendor is possible but never wanted in practice

# Lustre features at work

## Lustre Changelogs with Robinhood

- Good experience with a two-node setup on Regal (old scratch)
  - node #1: changelog reader with MySQL
  - node #2: performing purges
- Single node setup on Oak and Fir (with newer hardware)

## Distributed namespace (DNE)

- Currently testing DNE on Fir
- Not sure how to really use DNE phase II in practice as performance impact seems significant on small directories; also not user accessible?
- Lustre Documentation could be improved with some recommendations (eg. setdirstripe and lfs mkdir -D are not documented)

## Nodemap

- GID-only mapping in production on Oak with modified l_getidentity because some users are not known by Oak [LU-10884]
- Secondary groups with different members are not always checked server-side (inode cache issue) [LU-10884]

# Lustre features at work (cont’d)

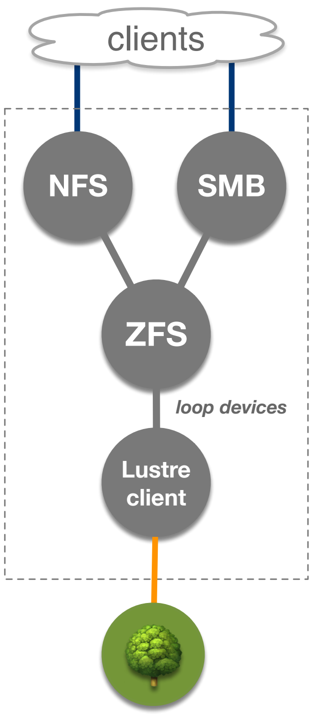
LNet routers

- Great flexibility with Sherlock’s multiple IB fabrics
- We like to use **lctl add_route** from time to time even if marked as “obsolete (DANGEROUS)”
- lnetctl with different IB settings used for Xstream

LFSCK

- Some tests done, planned but not using in production yet

# ZFS over Lustre




What??

- ZFS on Linux 0.7.9 deployed over 1TB loop devices stored in Lustre on Oak (Lustre 2.10).

How?

- Each ZFS filesystem is running in a VM using SR-IOVs for both IB FDR and Ethernet 10G (like all other Oak gateways)
- One VM is used per group.
- ZFS filesystems are then exported using NFSv4 and SMB to various Stanford research labs
- Snapshots from an older ZFS on Linux system were previously imported using zfs send/recv.
- Each ZFS filesystem can use different UID/GID and has no inode quota limit unlike native Oak storage.

# Questions?

Contact: sthiell@stanford.edu

# Extra slides

# Data-on-MDT (DoM) performance on Fir

- Untar of /dev/shm/linux-4.19-rc4.tar into Lustre

| DoM/DNE | default dirstripe=1 | default dirstripe=2 | default dirstripe=4 |
|---|---|---|---|
| **DoM Disabled** | 96.7s | 113s | 108.9s |
| **DoM 128kB** | 53.1s | 74.7s | 84.3s |

› Small sequential writes: our DoM setup helps but DNEv2 doesn't

› DoM helps even more if mdraid data-check is running on the OSTs

Config: Lustre 2.11.55 on CentOS 7.5 patchfull
Test client: Dell R7425 Dual AMD EPYC 7401 (96 threads) with 512GB RAM, 1 x IB EDR, no router
Other notes:

- mdraid data-check limited to 25MB/s
- freed pagecache, dentrie and inode caches on client and servers between all tests

# Data-on-MDT (DoM) performance on Fir (cont'd)

- Kernel compilation: linux-4.19-rc4 (make -j 192)

| DoM/DNE | default dirstripe=1 |
|---|---|
| **DoM Disabled** | 63m26s |
| **DoM 128kB** | 33m35s |

  - 1.88x faster with DoM
  - Reference time: 3m2s on /dev/shm

Config: Lustre 2.11.55 on CentOS 7.5 patchfull
Test client: Dell R7425 Dual AMD EPYC 7401 (96 threads) with 512GB RAM, 1 x IB EDR, no router
Other notes:
- mdraid data-check limited to 25MB/s
- freed pagecache, dentrie and inode caches on client and servers between all tests

# Data-on-MDT with DNE: Small file random I/Os with FIO

- Benchmark manyfiles-4k-random-read.fio
  - › all files precreated and then cache dropped on servers and client

| DoM? | 1 MDT | 2 MDTs | 4 MDTs |
|---|---|---|---|
| **Disabled** | 16,994 IOPS | 16,515 IOPS | 16,584 IOPS |
| **128kB** | 95,123 IOPS | 106,928 IOPS | 111,370 IOPS |

manyfiles-4k-rw.fio

| DoM? | 1 MDT | 2 MDTs | 4 MDTs |
|---|---|---|---|
| **Disabled** | 6,379 IOPS | 8,090 IOPS | 8,365 IOPS |
| **128kB** | 13,719 IOPS | 30,620 IOPS | 47,426 IOPS |

# Data-on-MDT with DNE: Small file random I/Os with FIO

- Benchmark manyfiles-4k-random-write.fio

| DoM? | 1 MDT | 2 MDTs | 4 MDTs |
|---|---|---|---|
| **Disabled** | 47,396 IOPS | 107,965 IOPS | 108,256 IOPS |
| **128kB** | 27,645 IOPS | 56,404 IOPS | 110,529 IOPS |

# Data-on-MDT perf improvements: basic experiments

- IO-500 benchmark from a single client and single MDT

| IOPS Phase | DoM disabled | DoM 128kB |
|---|---|---|
| mdtest_easy_write | 23.747 kiops | 23.589 kiops |
| mdtest_hard_write | 3.744 kiops | 4.310 kiops |
| find | 429.420 kiops | 370.490 kiops |
| mdtest_easy_stat | 18.448 kiops | 42.224 kiops |
| mdtest_hard_stat | 7.046 kiops | 4.640 kiops |
| mdtest_easy_delete | 22.334 kiops | 23.348 kiops |
| mdtest_hard_read | 18.387 kiops | 17.122 kiops |
| mdtest_hard_delete | 4.856 kiops | 2.716 kiops |
| IOPS | 17.7597 kiops | 17.2969 kiops |

# Tools that we develop and use: sasutils

sasutils: Serial Attached SCSI (SAS) Linux utilities and Python library

- Display SAS fabric tree and provide aggregated view of devices
- sas_discover, sas_devices, sas_counters, ses_report
- Based on sysfs (and also sg3_utils and smp_utils)
- Support SES Enclosure Nickname

- Available at https://github.com/stanford-rc/sasutils
- Made available in EPEL 7

```
$ sas_discover
oak-io1-s1
|--host19: board: SAS9300-8e 03-25656-02A SV53345573, product: LSISAS3008, bios: 04.00.00.00, fw: 12.00.00.00
|  `---8x--expander-19:0 vendor: ASTEK, product: Switch184, rev: 0004
|          |---1x--end_device-19:0:0 vendor: ASTEK, model: Switch184, rev: 0004
|          `---4x--expander-19:1 vendor: QCT, product: JB4602 SIM 0, rev: 1100
|                  |---1x--end_device-19:1:10 vendor: SEAGATE, model: ST8000NM0075, rev: E002 size 8.0TB
|                  |---1x--end_device-19:1:11 vendor: SEAGATE, model: ST8000NM0075, rev: E002 size 8.0TB
...
```

# Tools that we develop and use: shine

shine: tool to setup and manage Lustre file system(s) on a cluster

Added hooks in shine to assemble/stop MD arrays on target start/stop.

Other shine-related work: High Availability without Pacemaker

- shine already has target failover support (master branch)
- develop a centralized Lustre supervisor with simple policy rules to
  - check servers and possibly also some clients
  - fence non-responsive server in case of a real issue
  - trigger target failover using shine
  - generate notifications (eg. Email, Slack)

- Shine is available at https://github.com/cea-hpc/shine
- Our development branch is available at https://github.com/stanford-rc/shine (oak_ha branch)

# Questions?

Contact: sthiell@stanford.edu