STORAGE DEVELOPER CONFERENCE
SDC 22
Fremont, CA
September 12-15, 2022
BY Developers FOR Developers
A SNIA Event

Design Modern Object Store Server for Lustre File System

in the Era of Solid State Storage and Persistent Memory

Yong Chen, Principal Software Engineer Lead and Architect
SAIT, Samsung Electronics

Agenda

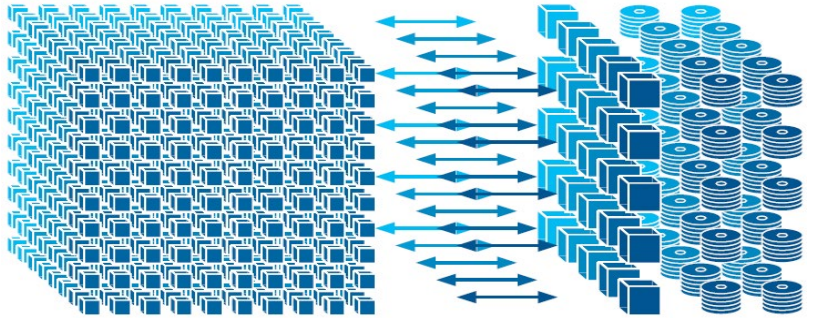
- Introduction
  - to Lustre architecture
- Background
  - on HPC storage
- Challenging Issues
  - in existing Lustre design
- Design Proposal
  - to solve these issues, and modernize Lustre
- Potential Benefits and Future Impact
  - improved I/O performance from full potential of hardware
  - future Lustre innovations and re-invigorating developer community

Lustre Architecture

A closer look

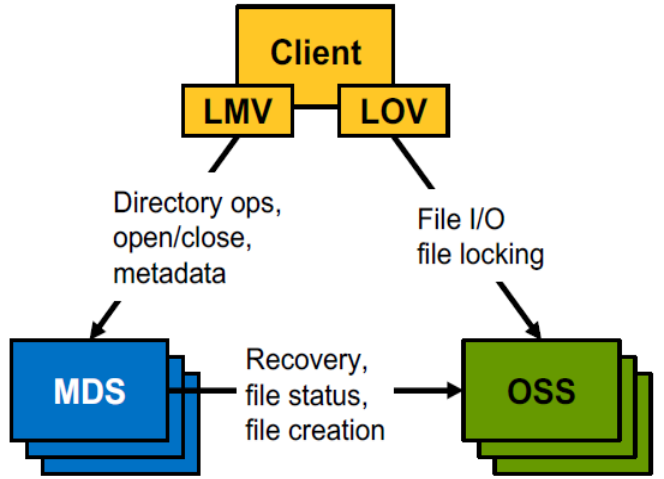
The Lustre File System

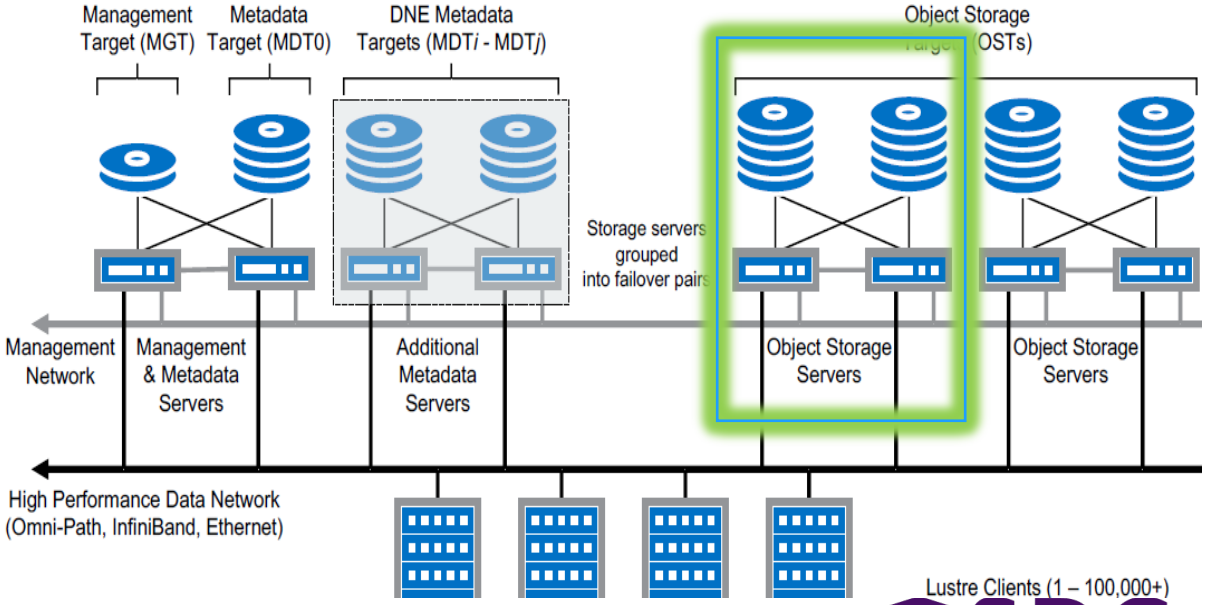
- Open-source distributed parallel file system
  - designed for extreme scalability and high performance
  - used to support most demanding data-intensive HPC workloads, e.g. weather, molecule
    - provides massively parallel data access for 10s of K clients, the dense computing node
  - also being used in emerging Big Data analytics and Deep Learning applications
- Most trusted open source storage solution for supercomputers: **stable**
  - widely adopted by supercomputers such as those ranked in **TOP500** list, e.g.
    - #1 ranked **Fugaku** (June **2020)** in Japan by Fujitsu
    - #1 ranked **Frontier** (June **2022) at Oak Ridge NL**, the first/only known **exascale** supercomputer
- Lustre server components + targets
  - MGS(MGT), global configuration and registration of all servers
  - MDS(MDT), Metadata Server, central brain for the whole system
    - inode map for client to look up where user files are stored
  - OSS(ODT), Object Store Sever
    - target is where the user data are stored
  - Infrastructure layer
    - LNet, Portal RPC, used to communicate between clients and servers



Lustre System Architecture

- Lustre client
  - the heart to carry out parallel file system logic
  - combines metadata and object storage to present coherent POSIX file system tree
  - the active driver to achieve parallel I/O by communicating with servers thru RPC calls
  - basic I/O flow
    - retrieves metadata from MDS => LMV
      - including file Layout information
    - builds LOV to map file
      - data chunks to objects on OSS
    - issues Rd/Wr requests directly to OSS
      - coordinate to acquire proper locks to maintain consistency



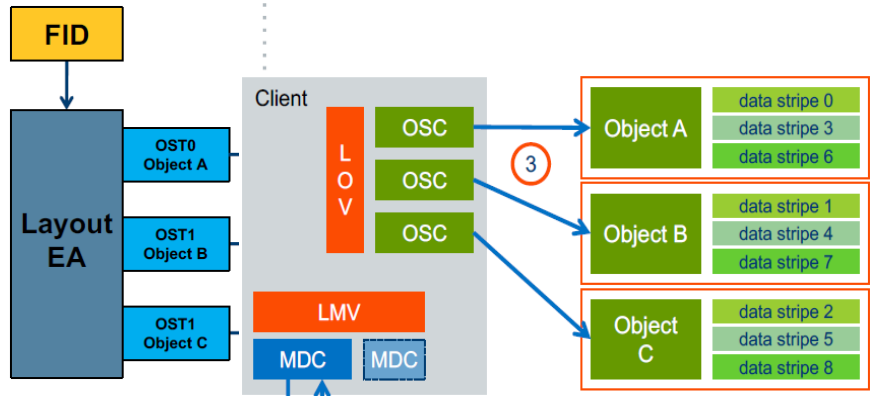


Lustre File vs Object Store Servers/Targets

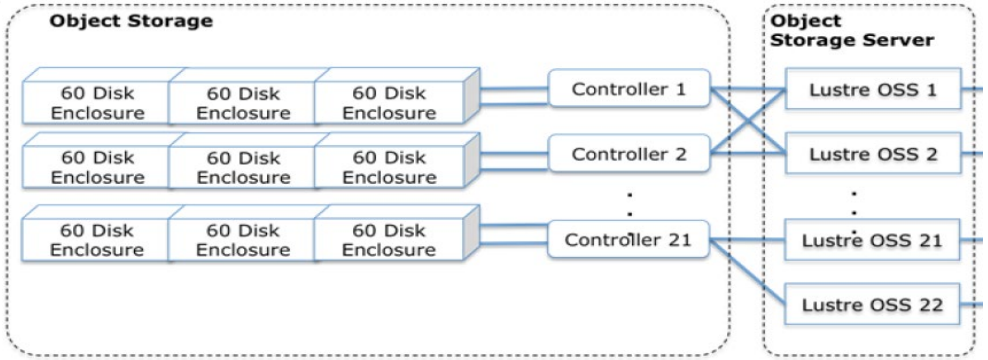
- Lustre POSIX File with 128b FID
  - 64b Sequence #., unique across all OSTs/MDTs
  - 32b Object Index #: reference to objects within OST

u64	u32	u32
sequence	**obj idx**	**version**

- global inode file metadata descriptor on MDT
  - POSIX Attributes: uid, gid, perm, timestamp, size
  - Extended Attributes: e.g. Layout EA
  - LOV: Logic Object View
    - data blocks are in striped across up to 2000 OSTs



- OSS/OST, building blocks of I/O's & storage
  - the bulk of Lustre server nodes, a few MDS
    - attached with multiple large storage array enclosures
  - key enabling factor for massively parallel I/O



- This talk will focus on OSS/OST

Storage in Supercomputing

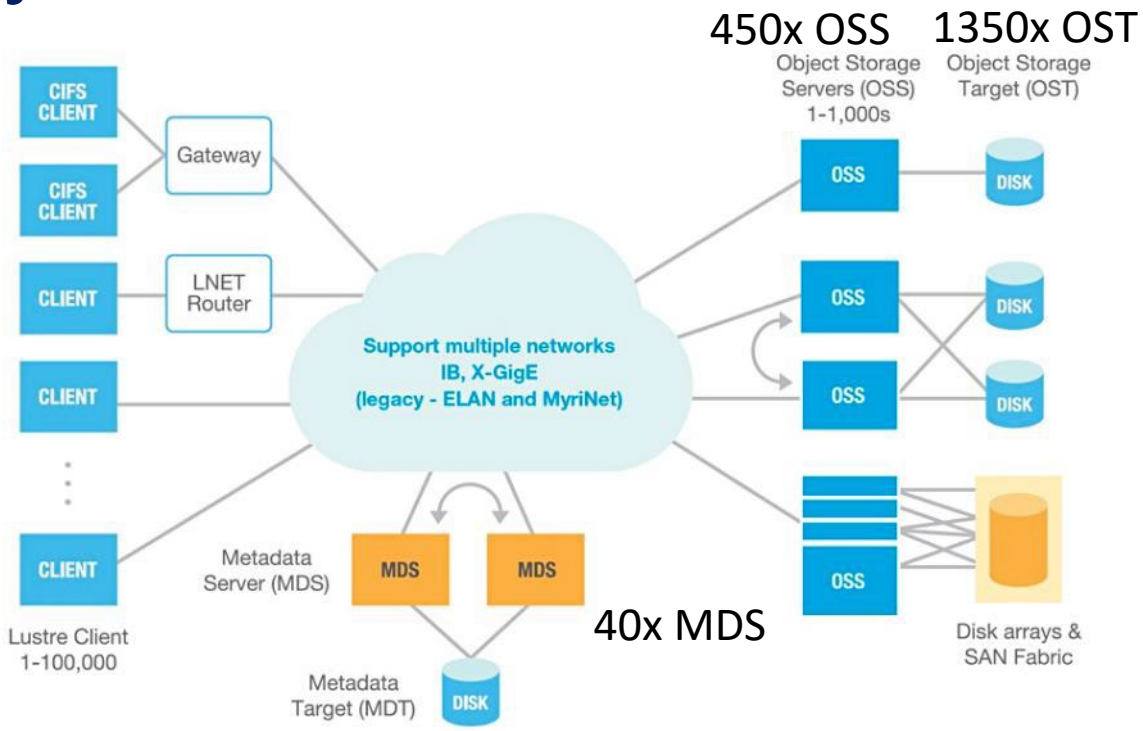
The landscape

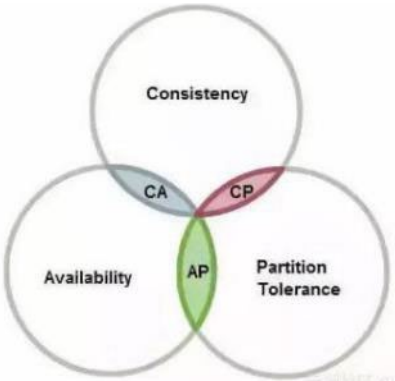
Lustre with All-Flash

- Imminent **challenges** to Lustre after having served us for 20+ years
  - a lot has advancements in Hardware and Software
  - SSDs have taken over, dominating mass storage even in the supercomputer data centers
    - straight-forward HW upgrade to SSDs?
- #5 ranked TOP500 Perlmutter supercomputer at Lawrence Berkeley NL
  - the largest all-flash storage system as of June 2021
  - balancing act to keep $Storage < 15% $Total due to higher SSD $cost
    - 3-4X more powerful computing than previous Cori
    - with **only** 35PB SSD vs 30PB of HDD on older Cori
  - trade-off: performance over capacity
- Return on Investment, per Lockwood blog
  - high-bandwidth jobs: "outstanding"
  - high IOPS & metadata heavy: **only** "good enough"
    - "still quite a bit of work to do to get the most out of big flash investment … Software continue to be the challenge…tradeoffs to make in Lustre…a lot of work in Software"
  - Results promising, but not simple straightforward HW upgrade
    - Bold, right choice in 2018 to move on from HDDs, SSD certainly IS the future!

Challenges in Distributed File Systems

- ORNL Frontier Orion the largest Lustre namespace
  - built on multi-tiered systems with SSDs/HDDs
    - 480x NVMe SSD metadata tier
    - 5.4K NVMe SSD performance tier,11.5PB based on SSU-F
    - 47.7K HDD capacity tier, 700PB based on SSU-D
      - 20x times the storage of Perlmutter 35PB SSD-only
- CAP: Consistency, Availability, Partition Tolerance
  - i.e. correct Rd after Wr; Robust; afford to lose msg
  - dictates distributed systems **cannot** maintain all three
- Lustre: POSIX strict Consistency
  - via centralized MDS default only 1
  - **Availability**: Active/Passive failover, MDS no SPOF
    - no improvement to performance, still potential bottleneck
  - **Scale-out**: DNE with multi-MDS, dedicated MDS for cascading sub-directory nodes,
    - strictly, metadata is **sharded**, not Partition Tolerant
  - primarily **Parallel** file system for concurrent I/O, not distributed
    - active research area to improve metadata performance & scalability
- Alternative?
  - POSIX is performance constrained, can we move away from POSIX?
  - that is exactly the direction Intel's DAOS project has taken

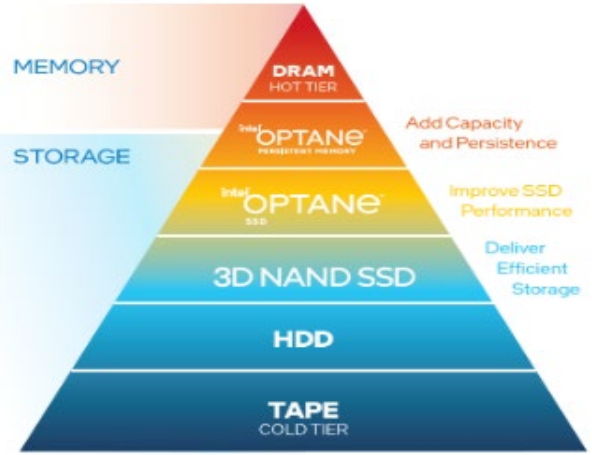




Intel DAOS: Distributed Asynchronous Object Storage

- Aurora at Argonne NL, due to complete in late 2022 after delays
  - 2x exaFLOPS compared to 1.1 in Frontier
  - major leap of technology advancements
    - Sapphire Rapids CPU, PCIe Gen 5, and CXL
    - 10PB memory and 230PB SSD storage
      - vs 11.5PB SSDs & 700PB HDDs in Frontier

- DAOS as storage platform, first discussed at SDC'15
  - native **Key-Value Object Store** to overcome **POSIX** limitations
    - vs POSIX based Parallel File System e.g. Lustre
  - designed from ground up to support **Storage Class Memory**
    - Persistent Memory (3D-XPoint Optane) and NVMe SSDs
  - both Client & Server running in User-Space
    - Polling mode vs kernel Interrupt to bypass Linux kernel at both SQ & CQ time
    - combined with RDMA, to boost bandwidth and lower latency



- DAOS delivers High-IOPS, High-Bandwidth and Low-Latency
  - a step closer to all-in-memory High Performance Computing

The Case of Forward Looking vs Backward Compatibility

- DAOS takes top 11 spots out of 22 in IO500 ISC22 (vs 2 out of 22 for Lustre)
- Historically HPC legacy apps depend on POSIX
  - many lessons on backward compatibility
    - HP/Intel's VLIW Itanium IA-64 vs x86
    - Windows 8 on Surface RT ARM: Marketplace App vs Win32
  - thousands of important apps can't be ignored or re-written
- DAOS's solutions to POSIX vs Object Store interface
  - per Lockwood ISC19 Blog on discussion with Architect Lombardi
  - A: FUSE library approach, but with performance hit
  - B: POSIX intercept/shim using preload library, but with consistency risk
    - performance-sensitive app to bypass FUSE and map POSIX API calls to DAOS native APIs call
- Non-technical concerns in DAOS
  - exclusive dependency on Intel Optane
    - no alternative or 2nd supplier to Intel's, also tied to Intel Xeon CPU
  - persistent memory expensive, chicken-egg issue, need more killer use cases

5	SC21	Huawei Cloud		PDSL	Flashfs
6	ISC21	Intel	Endeavour	Intel	DAOS
7	ISC20	Intel	Wolf	Intel	DAOS
8	ISC22	LRZ	SuperMUC-NG Phase2	Lenovo	DAOS
9	ISC22	University of Cambridge	Cumulus	Dell/Intel	DAOS
10	ISC21	Lenovo	Lenovo-Lenox	Lenovo	DAOS
11	SC21	BPFS Lab	Kongming		BPFS
12	SC19	WekaIO	WekaIO on AWS	WekaIO	WekaIO Matrix
13	ISC20	TACC	Frontera	Intel	DAOS
14	ISC22	Oracle Cloud Infrastructure	Oracle Cloud with WEKA on RDMA	WekaIO	WEKA
15	ISC22	Lenovo	Lenovo-Lenox3	Lenovo	DAOS
16	ISC21	National Supercomputer Center in GuangZhou	Venus2	National Supercomputer Center in GuangZhou	kapok

The Case of Legacy Support: DAOS vs Lustre

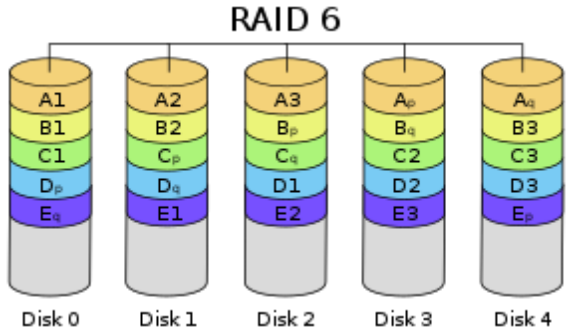
- We agree with the long term direction of Intel DAOS project
  - we applaud Intel achievements in DAOS to showcase what latest technologies capable of
  - we also recognize Intel significant contributions HW/SW to Solid State Storage & PMEM
    - NVMe, DPDK/SPDK/PMDK, 3D-XPoint/Optane, CXL
- We believe Lustre continue to play important role in foreseeable future
    - 1 in 4 (28%) are running Lustre from IO500 top 83
- we like to continue improve & modernize Lustre
  - Our approach is to apply the same set of modern technologies to Lustre project
    - Key-Value Store + Erasure Coding
    - User-mode SPDK/PMDK/RDMA
  - maintain full compatibility
    - existing legacy applications can benefit directly without costly modification or re-development
  - This talk to share what and how we try to achieve this goal

IO500 ISC22 (total 83)	DAOS	Lustre
Top 22	11 (50%)	2 (15%)
Top 45	13 (29%)	12 (26%)
All 83	**13 (16%)**	**23 (28%)**

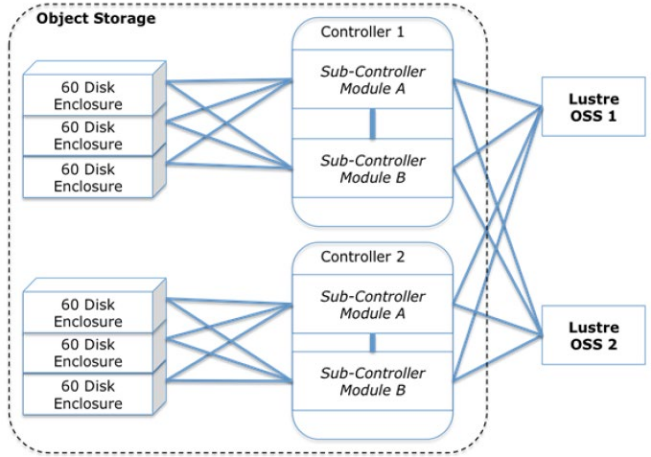
Challenges in Lustre

Issues and proposals

RAID The Storage Workhorse



- Guard against Hard Drive Failure since 1988
  - will continue in the future, even with SSD
  - Block interface: LBA with fixed-length units
- OST Backend options: either LDDISKFS/Ext4 or ZFS
  - both heavily rely on RAID, usually through hardware RAID controller
  - ZFS: unique design with special RAID-Z, per-file basis
- RAID usage in Lustre field deployments
  - Frontier Orion, ClusterStor E1000, All-Flash-Array SSU-F
    - Cray GridRAID declustered-P, 2x OST:12-SSD/each
- Samsung Supercomputer for 30y, from Cray in 1992
  - latest SSC-21 HPE with 30 PetaFLOPS, Ranked 15th on TOP500
  - DDN Lustre, using 8+2 RAID-6 in all-flash-array



RAID The Storage Workhorse, but…

- RAID, **challenged** in the era of Solid State Drives
  - hinders SSD innovations, hard to leverage native features as Zoned Namespaces ZNS
  - not friendly to SSD performance, HBA adds significant overhead to latency
  - only protects drive failure within array, (usually) not across servers
- the worst enemy: lengthy recovery window, longest wait for rebuild to finish
  - the system barely functions during degraded mode
    - pray no other drive fails, or it will get even slower, or lose the whole array
  - the bad new: SSD drive capacity increasingly getting bigger
    - concerns over RAID recovery counter to SSD maker interests, which is to roll out bigger drives
- **Issue #1: need better protection than RAID**
- **Proposal #1**: **Erasure Coding + Object Store**
  - critical change: to un-shackle from the simple matrix mapping restriction in RAID
  - when combined together the two can overcome RAID shortcomings

Erasure Coding vs RAID

- Erasure Coding 101
  - inter-planetary communication for Voyager probe
    - built-in encoding to solve long distance transmission loss
    - round-trip response to/from Earth: 5 hours
  - like RAID, data + encoded chunks, with redundancy
    - rebuild missing chunks, if under threshold
  - certain Erasure Coding equivalent to RAID
    - if using same Reed Solomon algorithm
    - EC provides a lot more protection options and flexibilities

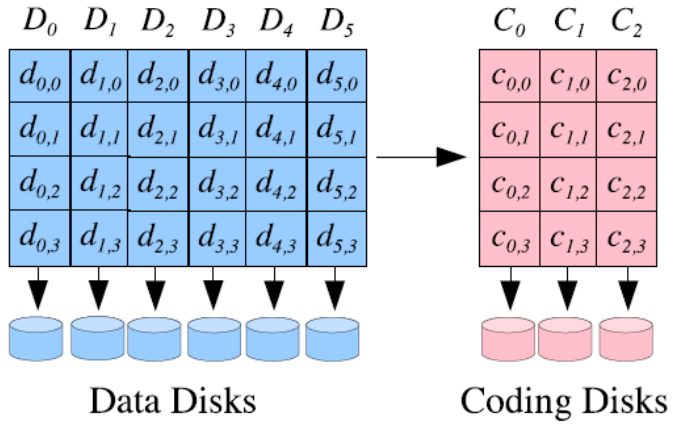
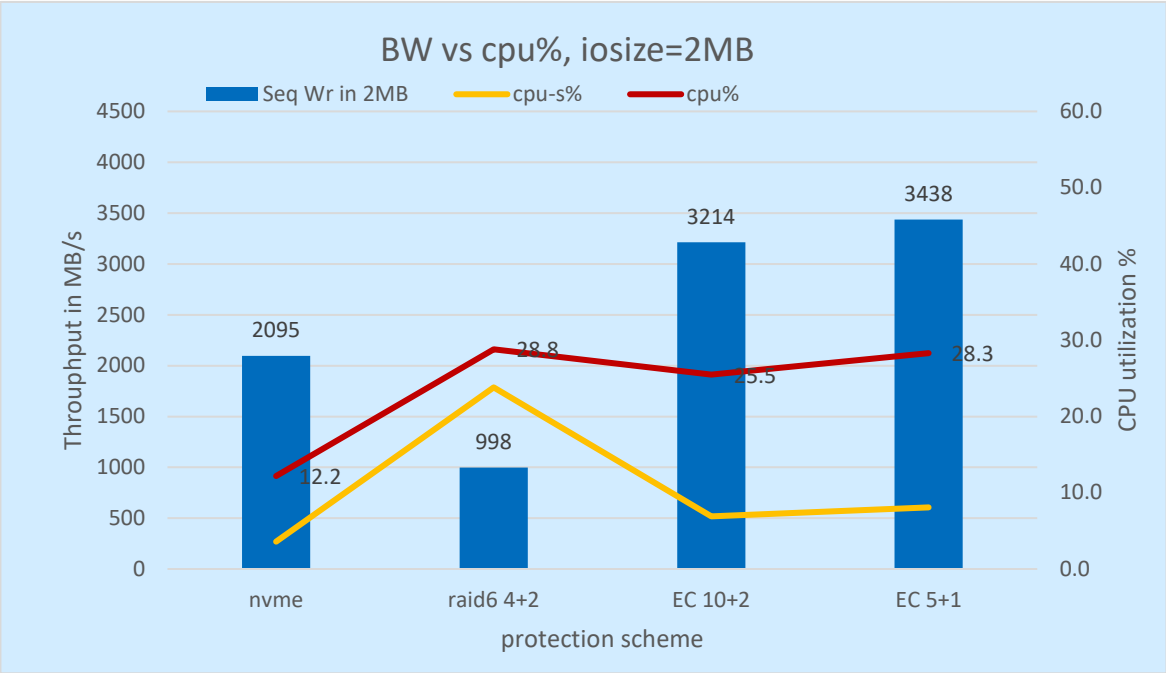


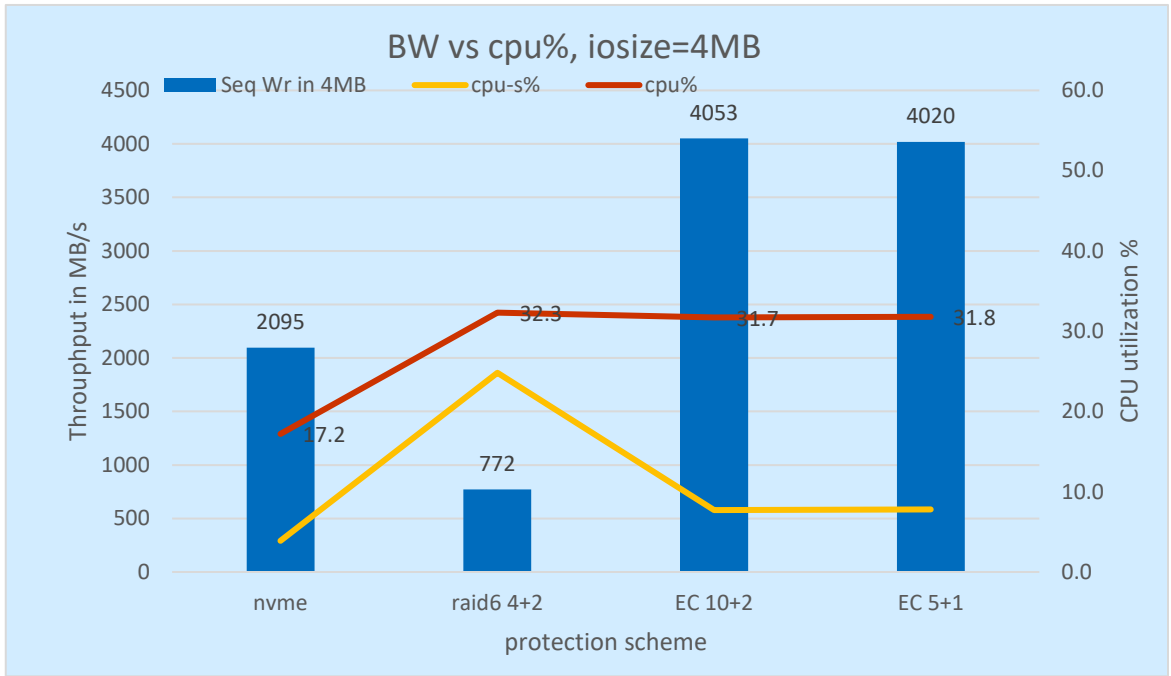
Figure 1: One stripe from an erasure coded storage system. The parameters are $k = 6$, $m = 3$ and $r = 4$.

- Erasure Coding: pros and cons
  - ideal for large I/O block size, fitting Lustre for HPC workload w/ multiple MBs 1-4MB typical
    - small I/O, such as metadata changes, better to use simple replication
  - expensive computational overhead, similar to RAID
    - EC acceleration using Computational Storage Device is active research area
    - Samsung Smart-SSD based effort is ongoing with collaborator at Iowa State University

Compute Overhead: RAID vs Erasure Coding

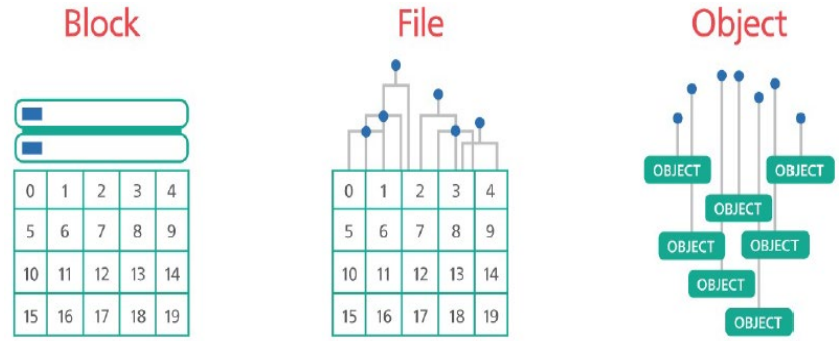
- **Setup**: RAID vs EC + Object Store
  - software RAID (kernel md) vs EC + Object Store using Ceph RBD with Jerasure EC library
  - BW non-comparable: local PCIe Gen3 vs 100GbE Object Store; cpu% Relatively
  - fio: QD:32; Sequential Write BW in 2MB or 4MB; CPU utilization all = (usr% + kernel sys%)
  - quick experiment: **no** configuration optimizing or tuning
- **Results**: similar for RAID and Erasure Coding, compute overhead significant, both need HW acceleration
  - Total cpu% equivalent around 28% for 2MB; around 32% for 4MB
  - Total kernel sys% for Software RAID MD: 82%, while 28% for Jerasure EC
  - SSD Write bandwidth utilization: Software RAID significantly low: 12% (RAID6 4+2) compared to 33% (EC)



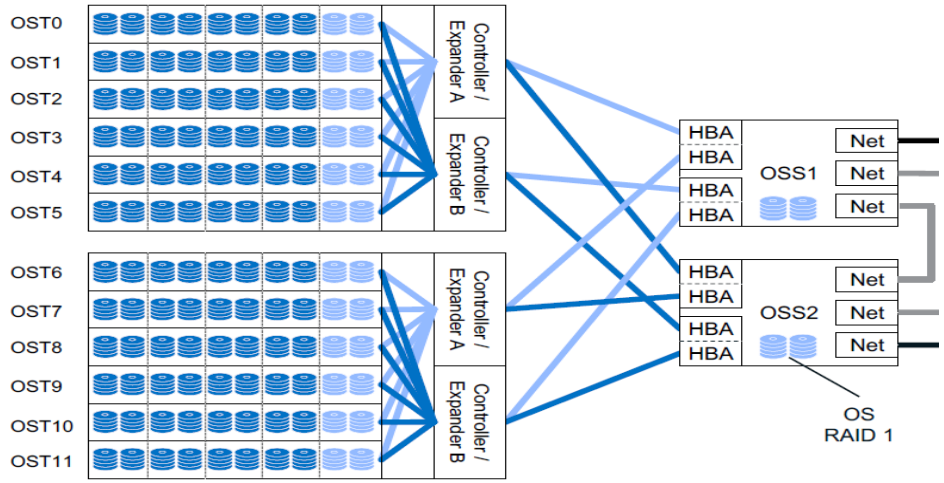


Lustre Object Store: Legacy Backend Storage Target

- Two Object Store Target Reference Designs
  - both OST design s are based on local file system
    - LDISKFS based on EXT4, with patches to kernel driver
    - ZFS is file system combined with LVM + RAID all-in-one

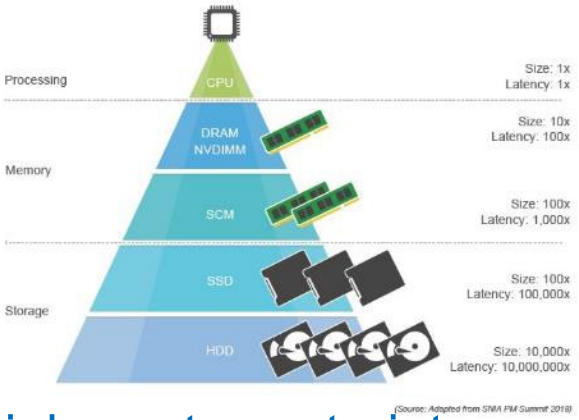


- Storage 101: File System vs Object Storage
  - File System: hierarchical w/ strict relationship to each other, human friendly
  - Object Store: unstructured, independent, scalable, best example: AWS S3
  - use File System as Object Store, not as scalable or performant
- **Issue #2**: both OSTs are based on **legacy LFS**
  - both designed for HDD era, no optimized for SSDs
    - metadata and user data mixed in on-disk LFS format
    - metadata are treated equally, same as regular user data



Modern Object Store Design with Metadata in PMEM

- Emerging SCM Persistent Memory was unimagined two decades ago
  - Optane memory offers persistency over DRAM with10x higher capacity
  - Optane memory is 10x slower than DRAM, but
  - Optane memory is 100x faster than SSDs
- **Proposal #2**: native Object Store + metadata in PMEM
  - OST should be based on truly native Object Store design
    - OSS objects independent, not to suffer from unnecessary File System inherent constraints
- OST Metadata should be treated as first class citizen, differentiated
  - New OSS can enjoy the following benefits from modern SCM
    - Metadata operates at memory speed, instead of that of slower SSDs
    - SSDs be dedicated to large Sequential I/Os, in MBs for most HPC workload
      - SSD sustains at peak performance, without disturbance of small random I/O's
  - Bonus: separate metadata can be replicated to remote backup



Lustre Components: All as Linux Kernel Module

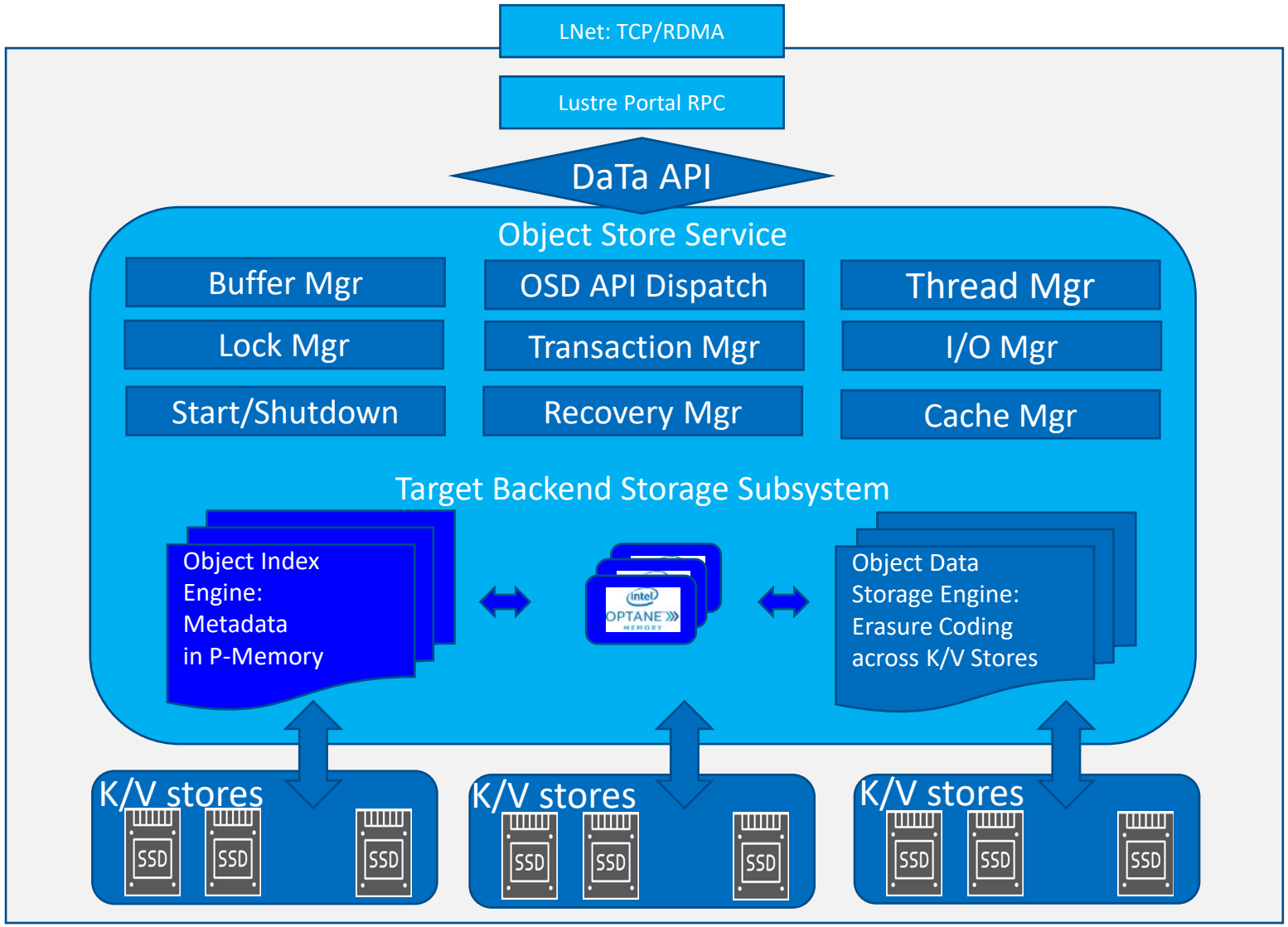
- Major 20+ components implemented in kernel modules
  - fitting choice at the time when Lustre first designed
    - to avoid context switch: I/Os performed all in kernel on OSS/MDS
  - kernel mode disadvantages at modern era
    - kernel I/O SW stack overhead too high for modern SSDs vs SPDK
    - time consuming: inefficient, less productive process than user mode
    - $$$: much more expensive dev resource, much smaller talent pool
    - **Hardest**: LDISKFS is patch to various Ext4 release, a Moving target
- **Issue #3**: the **worst** limiting factor today to Lustre project
  - definitely **not** agile, likely the reason Lustre is losing its appeal
- **Proposal** #3: Object Store Server implementation in User-Space
  - User-mode, prevailing model of programming in era of Solid State & PME
    - SPDK, PMDK, along with RDMA
  - Benefits: Low latency, agile process and higher productivity, larger developer talent pool

Module	
lite	Lustre C
lmv	Lustre L
lod	Lustre L
lov	Lustre L
mdc	Lustre M
mdd	Lustre M
mdt	Lustre M
mgc	Lustre M
mgs	Lustre M
obdclass	"Lustre
ofd	Lustre O
osc	Lustre O
osd-ldiskfs	Lustre O
osd-zfs	Lustre O
osp	"Lustre
ost	Lustre O
gss	Lustre G
ptlrpc	Lustre R

Object Store Server Modern Design

The proposal detail

Object Store Service + Target Backend Storage Subsystem



Engineering Strategy for POC

- Strategy for development work is to **reuse**
  - existing resources as much as possible
- **Porting Infrastructure layer** from existing Lustre codebase
  - Obviously, to keep compatible we have to follow the exact protocol on the wire
  - 00 Network Transport: **LNet** module, starting from TCP and RDMA, using libfabric
  - 01 Lustre Protocol interface: **Portal RPC** to make procedure calls between servers and clients
  - 02 maintain representation identical for common base objects/structures: e.g. obd class
- **Implementing Core Functions** of new Object Store Service & Target in user-mode
  - 2012 Lustre **OSD API** spec by Intel Lustre team (now DAOS) when working on ZFS OST
    - grateful for the extremely helpful API document!
  - The purpose was to “create many possibilities including using Object Store Devices or other new persistent storage technologies”
  - made easy for future OSS project like ours!

No	Module	Functional
00	Networking	LNet: TCP, RDMA
01	Infrastructure: portal RPC	Lustre protocol interface
02	Internal Lustre base:	Lustre device base object Ops: OBD APIs
03	Functional API: OSS/OST	Lustre DaTa APIs: …

Object Store Service Core Functions: DaTa APIs

- **Implementing Core Functions** for new Object Store Service & Target
  - Lustre Object Store **Service** + **Target** in user mode process
  - Third OSS/OST option, after LDISKFS and ZFS
- **Leveraging**: many existing Open Source projects
  - **Metadata: Object Indexing** K/V Store: B+Tree or Hash
  - KV-Store as OST internal object store
    - Samsung KVCS x86 host based, high performance (FMS'22)
    - Samsung K/V SSDs as candidate, if capacity oriented
  - **Data Redundancy**: Jerasure or other EC libraries
  - **PMDK and SPDK**: directly/indirectly
  - **RDMA**: OFED release
  - **Development languages**
    - mix & match with more productive modern choices

No	Module	Functional
03	Functional API: OSS/OST	DaTa APIs: 00/10: Reply/Punch 01/02: GetAttr/SetAttr 03/04: Read/Write 05/06: Create/Destroy 07/17: GetInfo/SetInfo 08/09: Connect/Disconnect 11/12: Open/Close 13/16: StatFS/Sync 18/19/20: Quota 21/22: LAdvise/Fallocate 23: Seek

Object Store Backend Storage Subsystem Requirements

- DaTa Object Types (inode per FID)
  - regular objects: unstructured data (file/objects)
  - index objects: key/values pairs (directories, quota, FLDB)
- DaTa Object Operations
  - Core: create/destroy/manipulate object attributes
  - Data: to access object body: read/write/truncate/punch
  - Index ops: to access index objects as key-value pairs
- Key requirements for storage backend
  - transaction commit
    - atomic, consistent, durable, callback
  - object attributes
    - POSIX attributes
      - user id, group id, access mode, time, size
    - Extended Attributes
  - efficient indexing for index objects
    - efficient retrieval
    - efficient random lookup
    - iteration with ordering, restart from cursor
  - quota

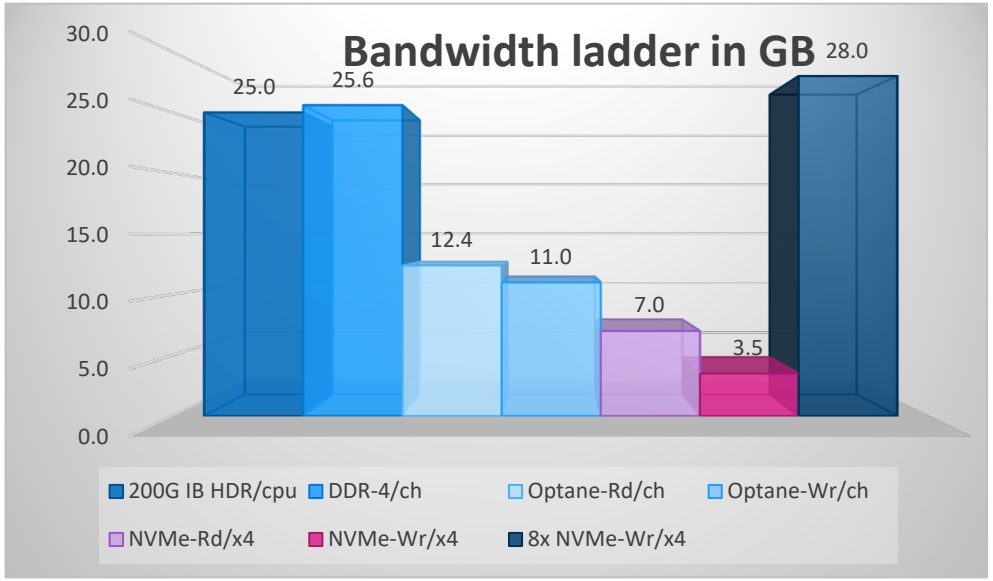
No	Module	Functional
04	OST Object Indexing (inode) (persistent memory)	OST metadata mgmt via K/V Store (B+tree) FID -> object lookup/manipulate
05	Index Object Access (persistent memory)	Index Access: internal key/value mgmt
06	Data Object I/O Access	Data Access: Read/write/truncate/punch
07	Data Redundancy	Data distribution: object mapping: data to Objects sent to K/V stores
08	Algorithm Libraries	CRUSH, Erasure Coding

Benefits and Impact

The Potential

Potential Benefits: Performance and Project Impact

- POC status & goals
  - after high-level designing, entering early phase development
  - **100%** compatible with MDS/client, OSS/OST **drop-in** replacement
  - push HW limit and to fully leverage SSD bandwidth potential
- Expected benefits: improved I/O performance
  - lower latency from metadata in PMEM
  - lower latency from service running in user mode
  - higher bandwidth: without RAID HBA
    - Erasure Coding + Object Store
  - back of envelope estimate on HW potential per cpu



- Future Impact
  - pave the road to improve **critical** Lustre metadata
    - similarly, MDT in PMEM and MDS service in user-mode
  - open door to further innovations
    - e.g. further improve Lustre Metadata scalability
  - lower engineering barrier to attract more talented developers, innovative contributions

Summary

The take-away

Summary

- Introduced the Lustre architecture
- Presented more background of HPC storage
- Discussed the design of existing OSS/OST and
  Identified the challenges & limitations in Lustre and proposed solutions
  - RAID based vs Erasure Coding + Object Stores based
  - HDD era local file system based OST vs native Object Store + metadata all-in-PMEM
  - kernel mode modules vs server service all-in-user-mode
- Previewed the design of the proposed modern OSS/OST
- Discussed potential benefits and future impact
  - to improve I/O performance in both BW and latency to fully utilize SSD potential
  - to lower engineering barrier to attract more innovative contribution
  - to invigorate Lustre developer community

Acknowledgements

- Los Alamos National Laboratory
  - Qing Zheng, Scientist, Ph.D. from PDL of Carnegie Mellon University
- Iowa State University, Data Storage Lab
  - Runzhou Han, Prof. Mai Zheng
- Samsung Electronics, DSA - Product Planning
  - Young Paik
- Samsung Electronics, SAIT - System Architecture Lab
  - Robert Wisniewski, Ph.D., David Lombard, Rolf Riesen, Ph.D.
- Samsung Electronics, SAIT - Neural Processing Lab
  - Joseph Hassoun, James Loo
- Intel project teams on Lustre (then) and DAOS (now)
  - Johann Lombardi

Collaboration, Partnership and More

- Existing collaborators from academia and national labs
  - Iowa State University/DSL, Los Alamos NL, Lawrence Berkeley NL
- Looking for more partners from open-source communities
  - contributing or early adoption, test pilot
- QR-Code (top R) to connect/follow-up on LinkedIn
  - https://www.linkedin.com/in/yong-chen-783317
- QR-Code (below L) We Samsung are **hiring**! for Storage and HPC
  - https://boards.greenhouse.io/samsungsemiconductor/jobs/5298364003 [NPL/SAL]
  - Locations: Silicon Valley, CA preferred; Seattle, WA; Asia or Europe possible

Q & A

The discussion

Please take a moment to rate this session.

Your feedback is important to us.

Backup

More discussion

Object Store vs Key-Value Store

- Take-away: K/V Store is the building blocks (devices) of Object Store
  - Conceptually similarity: Val:= GET(key); PUT(key,val),
    - K/V Store: for developer, internal device level with formal specification
    - Object Store: end products that human end users can use and interact with,
  - Object: modifiable, appendable
  - K/V Value: complete, limited-size, not immutable
    - **sample K/V Stores**: memcached/redis, LevelDB/RockDB, SNIA K/V Spec
    - **device or system:** compliant to SNIA K/V Store API: Samsung **KVCS or** Samsung **K/V SSDs**
- SNIA References
  - Object Storage: Trends, Use Cases, Nov 16, 2021
  - The Key to Value, Understanding the NVMe Key-Value Standard, Sept 1, 2020
  - Key Value Standardization, SDC Sept 22-23, 2020

