DAOS Tutorial

NDA Considerations

- This is a public talk with all public material
- DAOS components are all announced hardware and open source software
- DAOS elements of Aurora are all public
- No NDA components of Aurora/Sunspot will be covered
- No GPU discussions
- No SDK discussions

- Focus on usage of DAOS

DAOS on Aurora

Argonne NATIONAL LABORATORY
U.S. DEPARTMENT OF ENERGY
intel
Hewlett Packard Enterprise
Aurora

Aurora

Leadership Computing Facility
Exascale Supercomputer

Peak Performance
≧ 1 Exaflops DP

Intel GPU
Intel® Data Center GPU Max

Intel Xeon Processor
Intel® Xeon® CPU Max

Platform
HPE Cray-Ex

Compute Node
2 Xeon Intel® Xeon® CPU Max processors
6 Intel® Data Center GPU Max
Node Unified Memory Architecture
8 fabric endpoints

GPU Architecture
Intel XeHPC architecture
High Bandwidth Memory Stacks

Node Performance
>130 TF

System Size
> 10,624 nodes

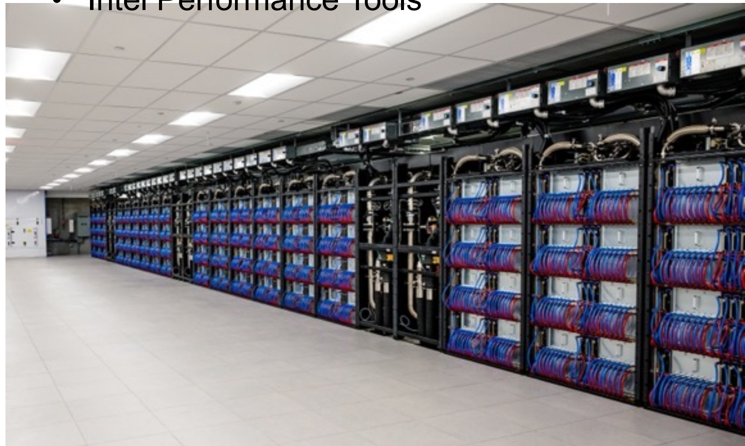
Aggregate System Memory
>10 PB aggregate System Memory

System Interconnect
HPE Slingshot 11
Dragonfly topology with adaptive routing

Network Switch
25.6 Tb/s per switch (64 200 Gb/s ports)
Links with 25 GB/s per direction

High-Performance Storage
220 PB
≧25 TB/s DAOS bandwidth

Software Environment
- C/C++
- Fortran
- SYCL/DPC++
- OpenMP offload
- Kokkos
- RAJA
- Intel Performance Tools

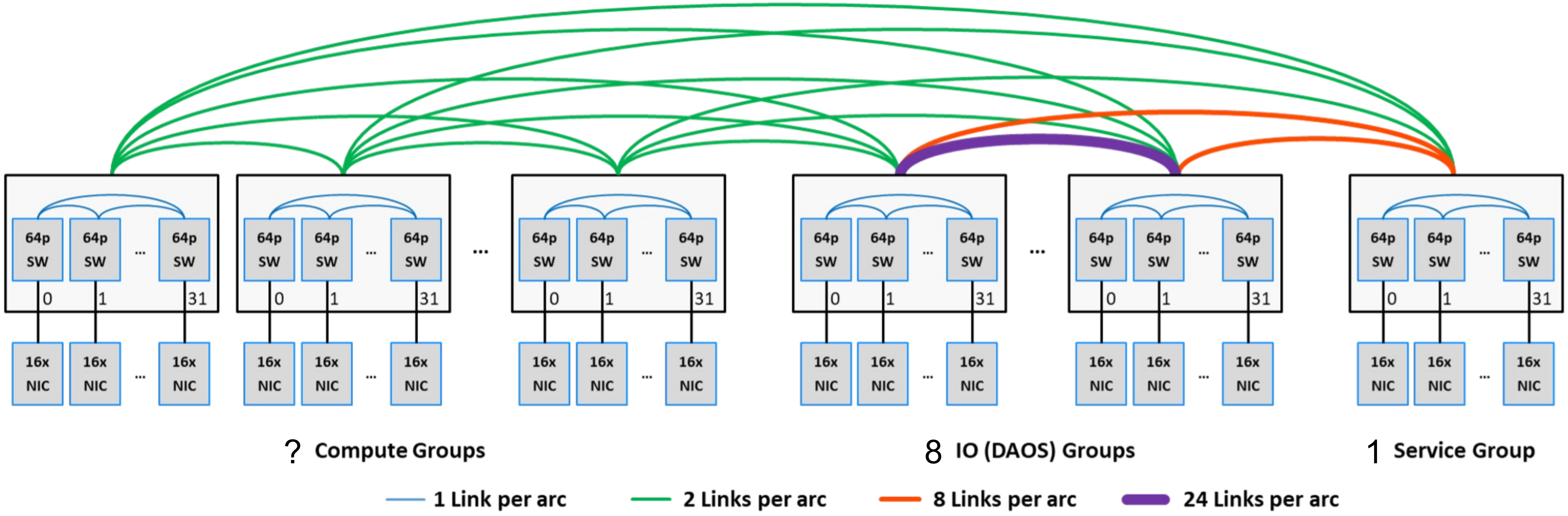


Aurora Storage Systems

- DAOS provides Aurora’s main “platform” high performance storage system
- Aurora leverages existing Lustre storage systems, Flare and Eagle, for center-wide data access and data sharing

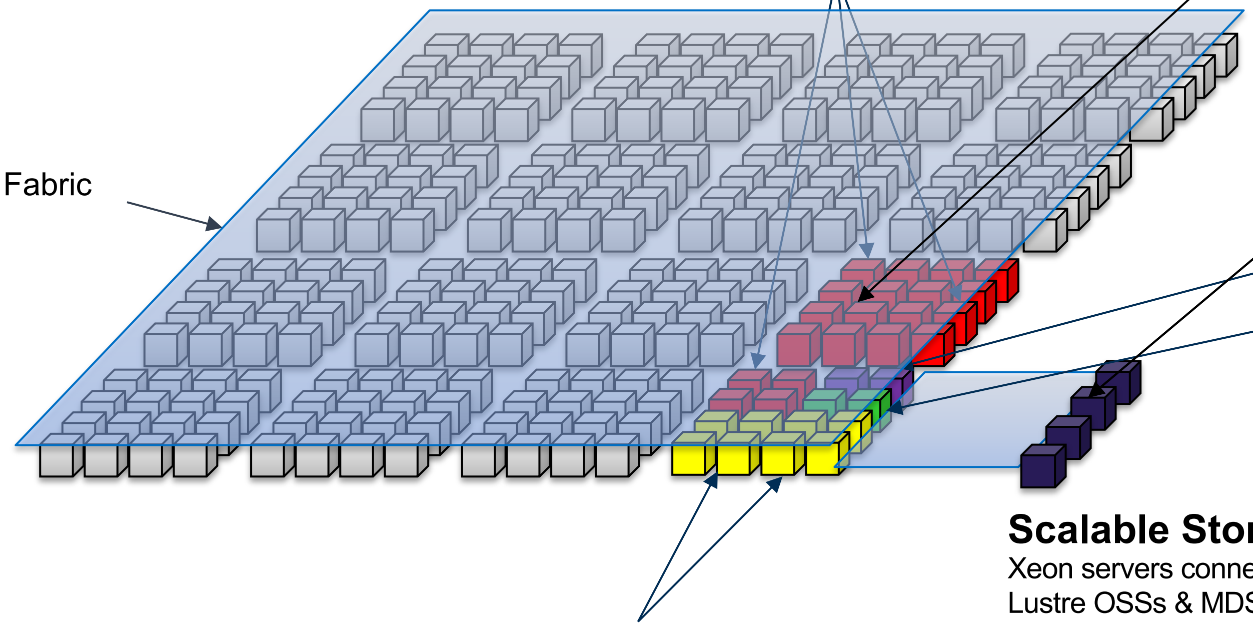
System	Capacity	Performance
Aurora DAOS	220 PB @ EC16+2 ▪ 250 PB NVMe ▪ 8 PB Optane PMEM	≥ 25 TB/s Read & Write
Eagle Not yet mounted on Aurora	100 PB @ RAID6 ▪ 8480 HDD ▪ 40 Lustre MDT	> 650 GB/s Read & Write
Flare	100 PB @ RAID6 ▪ 8480 HDD ▪ 40 Lustre MDT	> 650 GB/s Read & Write

Aurora Network Architecture



- Increased DAOS inter-group bandwidth
  - Support rebuilding and inter-server communication
  - Prevent DAOS server traffic interfering with application communication
- Increased bandwidth to service group
  - Support off-cluster access and data-movement

Aurora Overview



DAOS Performance
220 PB capacity @ EC16+2
≥ 25 TB/s

DAOS Nodes (DNs)
1024 Xeon servers
(16) 512GB NVRAM
(16) 15TB NVMe attached storage
DAOS service

Lustre Performance
Grand – 100 PB @ 650 GB/s
Eagle – 100 PB @ 650 GB/s

Slingshot Fabric

System Service Nodes (SSNs)

User Access Nodes (UANs)

Scalable Storage Cluster (SSC)
Xeon servers connected to JBOD
Lustre OSSs & MDSs

Gateway Nodes (GNs)
100 Xeon servers with no local storage
Access to external storage

DAOS Node Details

- Intel Coyote Pass System
  - (2) Xeon 5320 CPU (Ice Lake)
  - (16) 32GB DDR4 DIMMs
  - (16) 512GB Intel Optane Persistent Memory 200
  - (16) 15.3TB Samsung PM1733
  - (2) HPE Slingshot NIC

- 1024 Total Servers
  - Each node will run 2 DAOS engines/ranks
  - 2048 DAOS engines

IO-500 Results SC23

- Using 642 DAOS servers
- 300 nodes used as clients
  - Run configured for maximum performance using no data protection
- IOR
  - Easy: 20 TiB/s (write) and 11.6 TiB/s (read)
  - Hard: 10.2 TiB/s (write) and 6.2 TiB/s (read)
- mdtest
  - Write: 147 M (easy) and 100M (hard)
  - Stat: 230M (easy) and 218M (hard)
- Overall Score
  - Bandwidth 11,362 GiB/s
  - IOPS 164,391.73 kiops
  - TOTAL 43,218.80
- https://io500.org/submissions/view/695

https://io500.org

#3 in the IO500 Bandwidth Score

Certificate
IO500 Performance Certification
This Certificate is awarded to:
Argonne National Laboratory (Aurora Stor
#1 in the IO500 Bandwidth Score
IO500
Nov 2022
https://io500.org/list/SC22/io500

DAOS Overview

Distributed Asynchronous Object Storage

DAOS is an open-source software-defined scale-out object store that provides high bandwidth and high IOPS storage containers.

DAOS is architected from the ground up to exploit new NVMe technologies and is extremely lightweight since it operates End-to-End (E2E) in user space with full OS bypass.

DAOS offers a shift away from an I/O model designed for block-based and high-latency storage to one that inherently supports fine-grained data access and unlocks the performance of the next-generation storage technologies.

- Design Objectives
  - High throughput and IOPS at arbitrary alignment and size
  - Advanced data placement taking into account fault domains
  - Software-managed redundancy supporting both replication and erasure code with an online rebuild
  - End-to-end data integrity
  - Security framework to manage access control to storage pools
  - Software-defined storage management to provision, configure, modify and monitor storage pools over COTS hardware
  - Seamless integration with the Lustre parallel filesystem

Excellent architecture overview: https://docs.daos.io/v2.6/overview/architecture/

DAOS Features

- DAOS relies on Open Fabric Interface (OFI) for low-latency communications and stores data on both storage-class memory (SCM) and NVMe storage.
- DAOS presents a native key-array-value storage interface that offers a unified storage model over which domain-specific data models are ported, such as HDF5, MPI-IO, and Apache Hadoop. A POSIX I/O emulation layer implementing files and directories over the native DAOS API is also available.
  - —The POSIX emulation layer is accessed by using the DAOS POSIX container type.
- VOS (Versioning Object Store) is a key technology
  - —The Versioning Object Store (VOS) is responsible for providing and maintaining a persistent object store that supports byte-granular access and versioning for a single shard in a DAOS pool.
  - —DAOS I/O operations are logged and then inserted into a persistent index maintained in SCM. Each I/O is tagged with a particular timestamp called epoch.
  - —https://github.com/daos-stack/daos/blob/master/src/vos/README.md

https://docs.daos.io/v2.6/overview/architecture/

DAOS System

- —A DAOS *system* consists of a set of DAOS *storage nodes* connected to the same network
- —The Aurora DAOS storage nodes (which are dual socket) run two DAOS *server* instances per node, Each server process in turn starts one DAOS *Engine* process per physical socket.
- —Membership of the DAOS servers is recorded into the system map, that assigns a unique integer *rank* to each *Engine* process.
- —Its *Engine* sub-processes export the locally-attached SCM and NVM storage through the network.
- —Within a DAOS engine, the storage is statically partitioned across multiple *targets* to optimize concurrency. To avoid contention, each target has its private storage, its own pool of service threads, and its dedicated network context that can be directly addressed over the fabric independently of the other targets hosted on the same storage node.
  - ▪ On Aurora 16 targets per engine are configured, each target is using *1/16* of the capacity of the SCM capacity of that socket, independently of the other targets.
  - ▪ Each target is also using a fraction of the NVMe capacity of the NVMe drives that are attached to this socket. On Aurora each engine has 8 NVMe disks and 16 targets – 2 targets per NVMe drive.

See: https://docs.daos.io/v2.6/overview/architecture/#daos-system

DAOS Pools

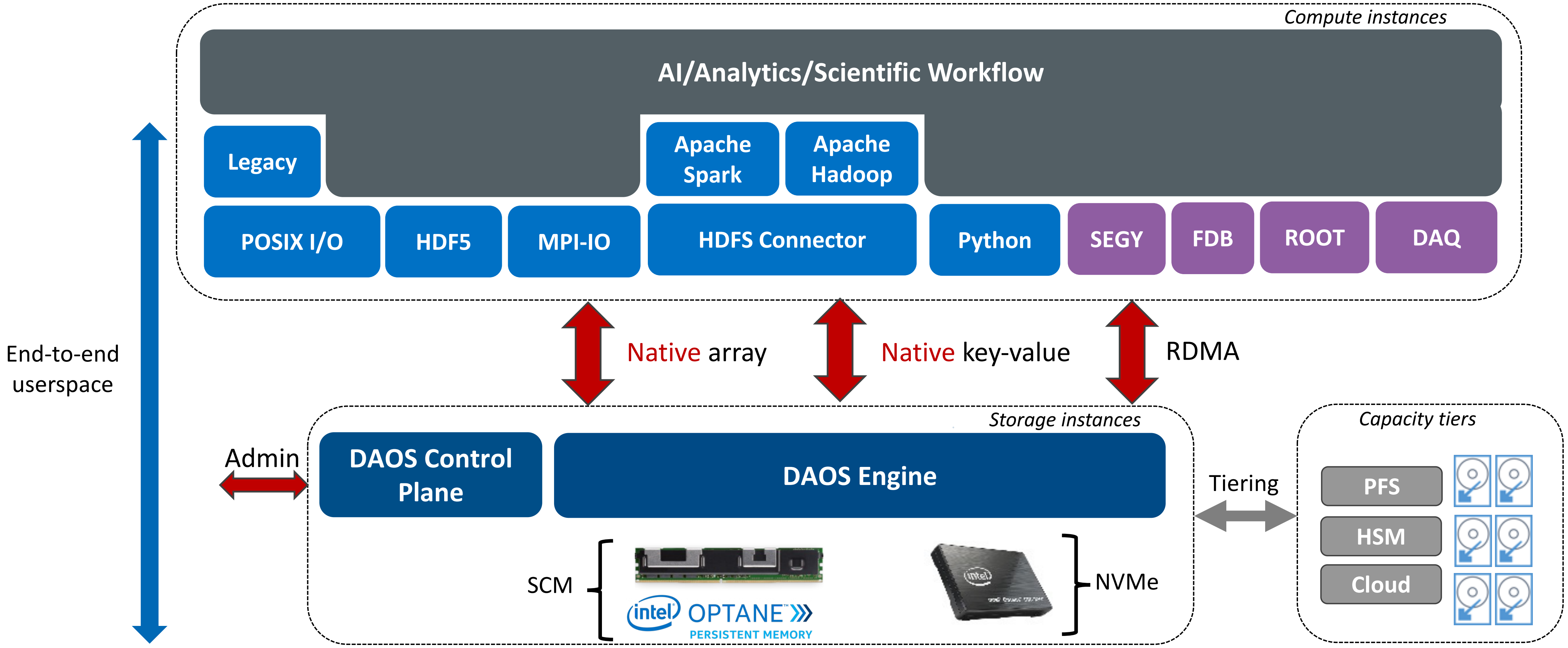
- A pool is a set of targets spread across different storage nodes over which data and metadata are distributed to achieve horizontal scalability, and replicated or erasure-coded to ensure durability and availability.
- A pool is identified by a unique *pool UUID* and maintains target memberships in a persistent versioned list called the *pool map*.
- The pool map records the list of active targets, it also contains the storage topology in the form of a tree that is used to identify targets sharing common hardware components. This topology informs object placement decisions that respect data redundancy settings.
- Upon target failure and exclusion from the pool map, data redundancy inside the pool is automatically restored online.

Excellent architecture overview: https://docs.daos.io/v2.6/overview/architecture/

DAOS Ecosystem

🟦 Generic I/O middleware supported today

🟪 Domain-specific data models under development in co-design with partners



DFS API

POSIX	DFS
mkdir(), rmdir()	dfs_mkdir(), dfs_rmdir()
open(), close(), access()	dfs_open(), dfs_release(),dfs_lookup()
pwritev(), preadv()	dfs_read/write()
{set,get,list,remove}xattr()	dfs_{set,get,list,remove}xattr
stat(), fstat()	dfs_stat(),ostat()
readdir() …	dfs_readdir() …

- Mostly 1-1 mapping from POSIX API to DFS API.
- Instead of File & Directory descriptors, use DFS objects.
- All calls need the DFS mount which is usually done once initialization.

DFUSE

- To mount an existing POSIX container with dfuse, run the following command:
  - `dfuse --pool mypool --container mycont -m /mnt/dfuse`
  - No one can access your container / mountpoint unless access is provided on the pool and container (through ACLs).
- Now you have a parallel file system under /mnt/dfuse on all nodes where that is mounted
  - Access files / directories as any namespace in the container, and applications can run without any modifications (the easy path).
- Interception Library:
  - This library works in conjunction with dfuse and allow to interception of POSIX I/O calls and issue the I/O operations directly from the application context through libdaos without any application changes.
  - This provides kernel-bypass for I/O. To use this set the LD_PRELOAD to point to the shared library in the DOAS install dir
    - LD_PRELOAD=/usr/lib64/libpil4dfs.so

Unified Namespace

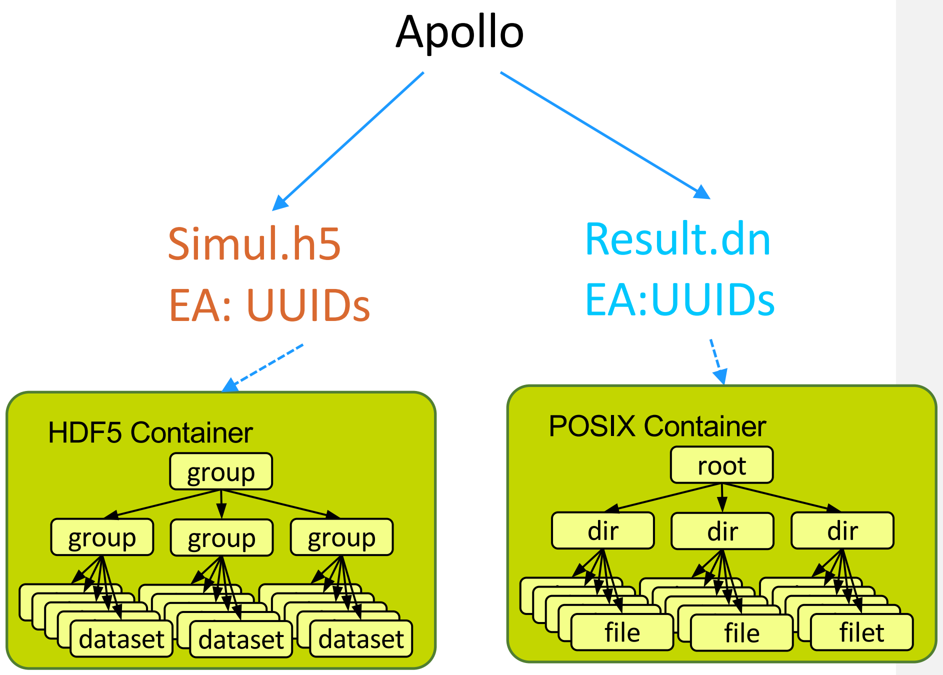
DAOS pool & container identified by labels/uuids

Concept: store those uuids in POSIX XATTR

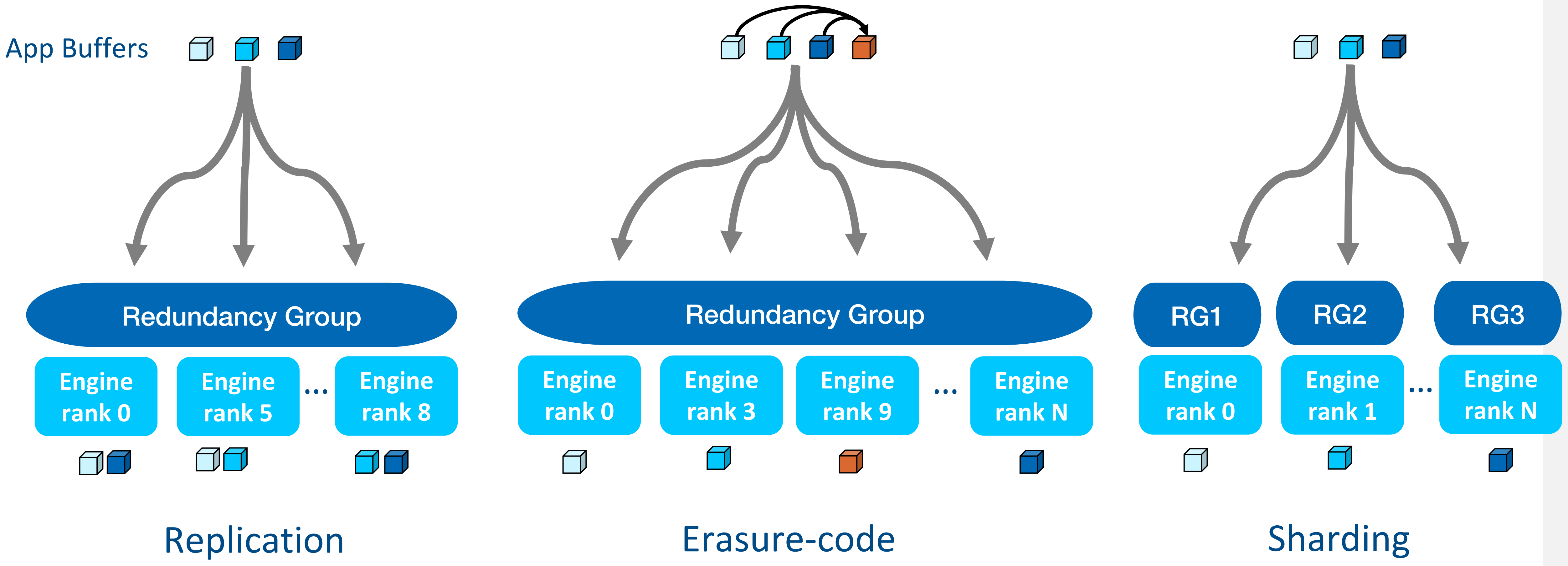
- Take path instead of pool/container labels
- Libraries/tools can extract them from POSIX XATTR
- libduns

Path created is a special file or directory pointing at the container

- Directories for POSIX containers, files otherwise
- Special file/directory is effectively empty in the hosting filesystem (just one XATTR)



DAOS Data Model: Distribution & Fault Tolerance



Container Level Redundancy Setting

- On Container create, one can set the redundancy factor property (`--properties=rd_fac:n`):
  - 0 (default): no redundancy
    - One can create objects with higher redundancy after
  - 1: tolerate 1 failure domain
    - At least RP2 or ECxP1
  - Up to 4
- DAOS automatically selects an object class for files and directories using the redundancy factor.
  - Widely striped, EC oclass for files (SX, EC_xP1GX, EC_xP2GX, etc.)
  - Single stripe, RP oclass for directories (S1, RP_2GX, RP_3GX, etc.)

Using DAOS on Aurora

Scheduler

- Scheduler is PBS Pro
  - Other ALCF resources used Cobalt
- qstat – list running jobs
- qsub – submit a job
  - Note the script is copied, so changes to the script once submitted have no effect
  - Common flags:
    - -A <project>
    - -lselect=<n> (how many nodes)
    - -lwalltime=30:00 (thirty minutes)
    - -I (interactive)
    - -ldaos=default (tells the ALCF prologue script which DAOS cluster to job intends to access)
- Queues
  - lustre_apps – default, run large jobs here
  - debug – single node jobs can run here
- Job script will use your $HOME as the working directory, cd to $PBS_O_WORKDIR to run from submission directory

Modules

- Aurora uses the module environment
- module list
  - Describes current list of modules loaded
- module avail
  - List of available modules
- Load the daos/base module
  - Provides certain environment variables
  - Provides support scripts
  - Not loading results in ‘daos’ commands not working
  - `module load daos/base`

Pool

- Please request a pool for your project via help@alcf.anl.gov
- Be reasonably accurate in your request for the capacity
- Tell us:
  - Project
  - Owner username
  - Group username
  - Size in TB
- Once the pool has been created it will be accessible for your use from a UAN or compute node.

Pool Query Example

```
gmcpheet@aurora-uan-0014:~> module use /soft/modulefiles/
gmcpheet@aurora-uan-0014:~> module load daos/base
gmcpheet@aurora-uan-0014:~> daos pool query Storage_test
Pool 1866d225-a851-4098-838a-01d9f049610b, ntarget=736, disabled=0, leader=1, version=537,
state=Ready
Pool health info:
- Rebuild done, 0 objs, 0 recs
Pool space info:
- Target(VOS) count:736
- Storage tier 0 (SCM):
  Total size: 30 TB
  Free: 28 TB, min:39 GB, max:39 GB, mean:39 GB
- Storage tier 1 (NVMe):
  Total size: 970 TB
  Free: 970 TB, min:1.3 TB, max:1.3 TB, mean:1.3 TB
gmcpheet@aurora-uan-0014:~>
```

Container (1)

- You can make a container
- You only *need* one container, but you can make several
  - —If you make 1000 containers, you are doing something wrong.

- Two techniques to create containers – using an ALCF wrapper or the native DAOS command:
  - —mkcont --type POSIX --pool <name> --user <name> --group <name> <container name> --dry-run
  - —daos container create –help
- For more information on container architecture and container creation options:
  - —https://docs.daos.io/v2.6/overview/storage/#daos-container
  - —https://docs.daos.io/v2.6/user/container/

- To query a container:
  - —daos cont query <pool name> <container name>

Container (2)

- POSIX containers can be mounted on UAN (login nodes) and CN (compute nodes)
  - —Mounted via FUSE

- UAN
  - —mkdir –p /tmp/$USER/<cont name>
  - —start-dfuse.sh --pool $POOL --cont $CONT --m /tmp/$USER/$CONT
  - —Full caching is enabled by default, this is great for logins or read-only workloads
  - —To unmount
    - ▪ fusermount3 –u /tmp/$USER/<cont name>

- CN
  - —launch-dfuse.sh <pool>:<cont>
    - ▪ Will start dfuse on all compute nodes mounting the directory at the following location
      - o /tmp/$POOL/$CONT
  - —By default, all caching is turned off which is best for expected consistency
  - —No cleanup is required on CN. Epilogue scripts will cleanup any dfuse mounts.

Container Example

- Create a new POXIX container in the pool Storage_test with the

```
gmcpheet@aurora-uan-0014:~> daos container create --type POSIX --properties rf:2 Storage_test new_container
Successfully created container 1eed7813-d2b9-40a8-a969-a2192cab0b9f
Container UUID : 1eed7813-d2b9-40a8-a969-a2192cab0b9f
Container Label: new_container
Container Type : POSIX
gmcpheet@aurora-uan-0014:~> daos pool list-containers Storage_test
UUID                                 Label
----                                 -----
1eed7813-d2b9-40a8-a969-a2192cab0b9f new_container      <<<< New container
655754da-4433-402f-8ccc-c2b63df7bf8c intel_recreate_cont
14a8b420-a60e-42c2-a622-38c65cb09adf no_red_container
beac13f5-eb12-429f-af74-5bdf8b908e43 default_ior_cont_name
gmcpheet@aurora-uan-0014:~>
```

Link to DAOS container documentation: https://docs.daos.io/v2.6/user/container/

Container Example – dfuse mount container and create file

```
gmcpheet@aurora-uan-0014:~> ls -ld mnt                                    <<<< empty directory for use as mount point for container
drwxr-xr-x 2 gmcpheet systems 4096 Sep 27  2023 mnt
gmcpheet@aurora-uan-0014:~> dfuse mnt Storage_test new_container          <<< This command will mount the POSIX container for use
gmcpheet@aurora-uan-0014:~> findmnt -t fuse.daos                          <<< This command will show all DAOS fuse mounts
TARGET            SOURCE FSTYPE    OPTIONS
/home/gmcpheet/mnt dfuse  fuse.daos rw,nosuid,nodev,noatime,user_id=32202,group_id=1010,default_permissions
gmcpheet@aurora-uan-0014:~> cd mnt                                        <<< change directories into the container
gmcpheet@aurora-uan-0014:~/mnt> vi new_file                               <<< create a new file
gmcpheet@aurora-uan-0014:~/mnt> ls -l new_file
-rw-r--r-- 1 gmcpheet systems 64 Oct 22 19:12 new_file
gmcpheet@aurora-uan-0014:~/mnt> cat new_file
Add a new line to the new file in the new DAOS POSIX container.
gmcpheet@aurora-uan-0014:~/mnt> cd ..
gmcpheet@aurora-uan-0014:~> fusermount3 -u mnt
gmcpheet@aurora-uan-0014:~> findmnt -t fuse.daos
gmcpheet@aurora-uan-0014:~>
```

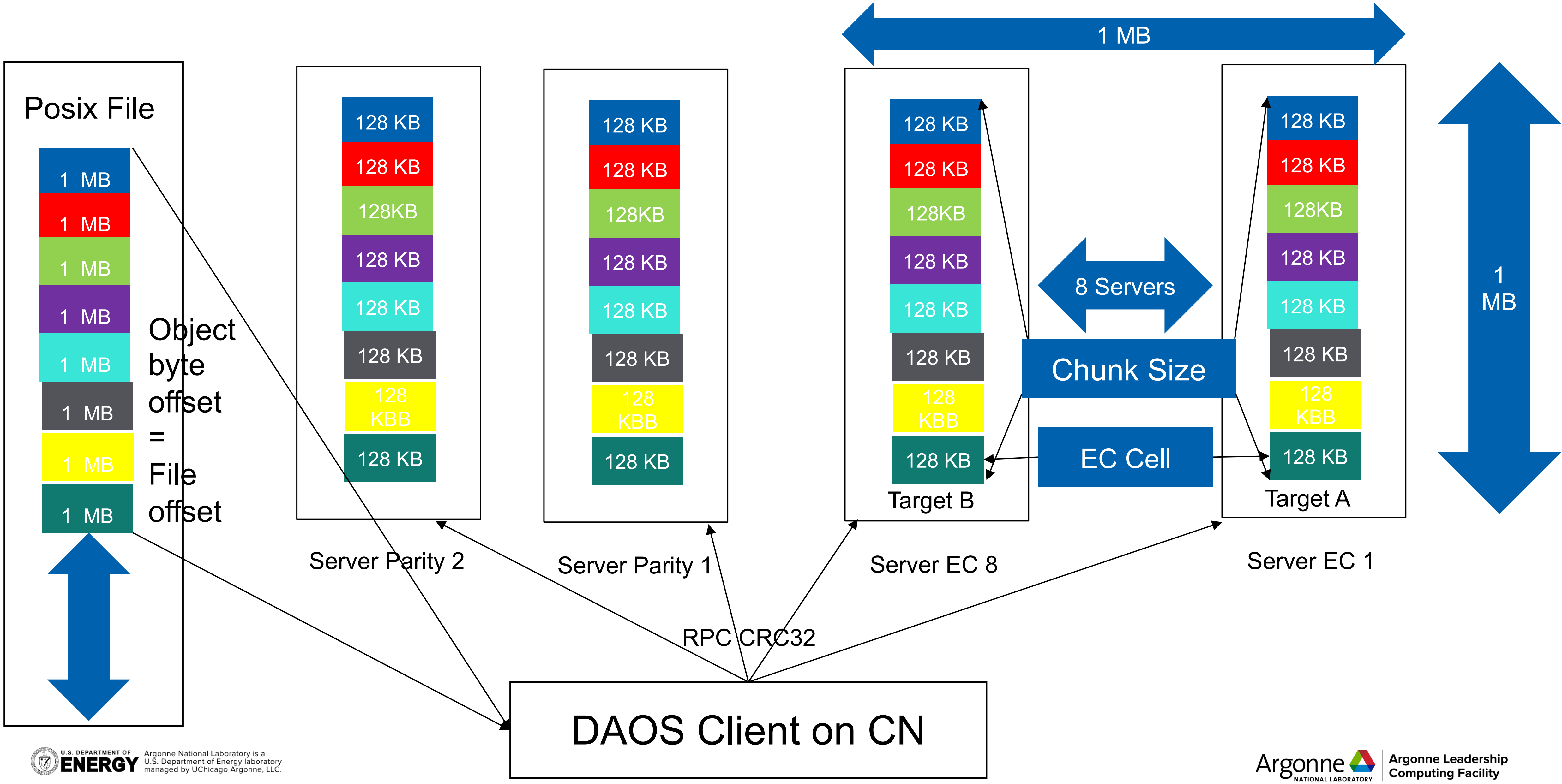
MPI-IO Driver for DAOS

- The DAOS MPI-IO driver is implemented within the I/O library in MPICH (ROMIO).
  - Added as an ADIO driver
  - Available in Intel MPI
  - https://github.com/pmodels/mpich
- MPI Files use the same DFS mapping to the DAOS Object Model
  - MPI Files can be accessed through the DFS API
  - MPI Files can be accessed through regular POSIX with a dfuse mount over the container.
- How to use this driver?
  - MPICH: append “daos:” to the file name/path or set env variable:
    - `export ROMIO_FSTYPE_FORCE="daos:‟ especially for HDF5 and PNetCDF`
      - `Eventually MPICH will auto-detect, not in current build on aurora`

DARSHAN support for DAOS

- DFS and DAOS modules displayed with darshan-parser
- mpiexec –genv LD_PRELOAD=/lus/flare/projects/Aurora_deployment/pkcoff/lib/darshan/lib/libdarshan.so:/opt/aurora/24.180.0/spack/unified/0.8.0/install/linux-sles15-x86_64/oneapi-2024.07.30.002/hdf5-1.14.3-h3fkw32wnijfekubu37szfdki2tsvcm4/lib/libhdf5.so:/opt/aurora/24.180.0/spack/unified/0.8.0/install/linux-sles15-x86_64/oneapi-2024.07.30.002/parallel-netcdf-1.12.3-sali72sqrarwsm5uyuaxb3sniubou7q7/lib/libpnetcdf.so:/usr/lib64/libpil4dfs.so
- MPI-IO programs will write darshan log using MPI-IO daos driver, if you use ROMIO_FSTYPE_FORCE=daos: then logfile written with daos driver, so:
    - export DARSHAN_LOGFILE=/tmp/<pool_name>/<container_name>/<logfilename>
    - mv /tmp/<pool_name>/<container_name>/<logfilename> /lus/flare/<project_path>
    - /lus/flare/projects/Aurora_deployment/pkcoff/lib/darshan/bin/darshan-parser /lus/flare/<project_path>/<logfilename> > <darshan_analysis_file>
- Will eventually be incorporated into modules
- Send me your darshan logfile to aid in your analysis: pcoffman@anl.gov

EC_8P2GX cell size 128K and 1MB chunk SSF



DATA Replication and Error Detection

- System is scoped for 2 Parity erasure coding on files and replication on directories – currently an EC stripe width of 8 - EC_8P2GX
  - About 33% bandwidth degradation over SX (no replication)
- Recommended checksum crc32
  - DER_CSUM error on write, retry on read
    - export DD_STDERR=ERR
- Stay with default chunk_size of 1 MB redundancy factor 2
  - daos container create --type=POSIX --dir-oclass=RP_3G1 --file-oclass=EC_8P2GX --properties=rd_fac:2,ec_cell_sz:131072,cksum:crc32,srv_cksum:on <pool_name> <container_name>
    - ALCF mkcont wrapper will do this for you
  - File erasure coding will eventually be EC_16P2GX

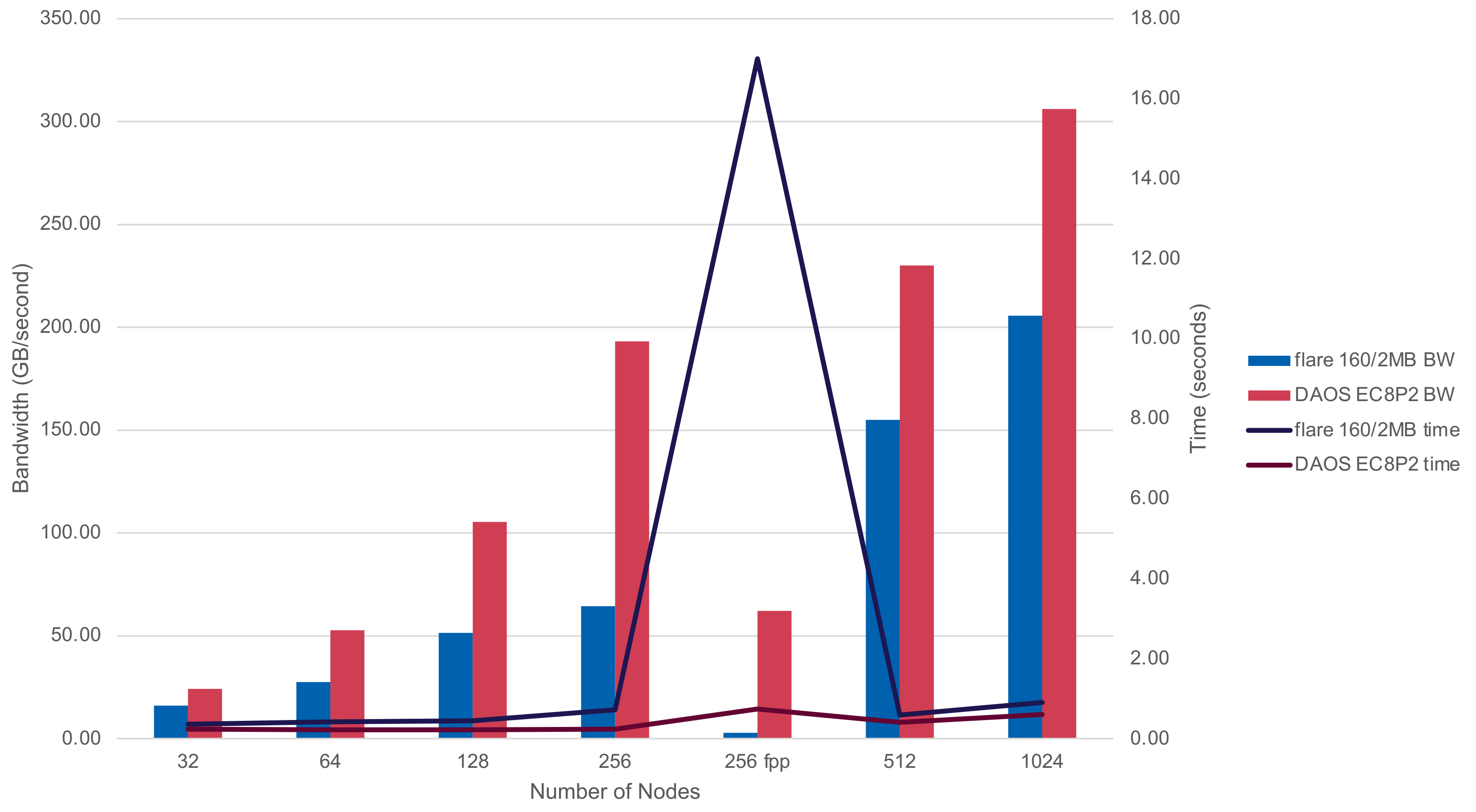
DAOS utilization versus Lustre/Spectrum Scale

- DAOS metadata stored in persistent memory on same server as data with low latency
- DAOS can handle smaller, discontiguous reads and writes much better than traditional PFS
    - No concept of file blocks (Spectrum scale) or stripes (Lustre) where optimal IO centers on separate block or same stripe
        - Costly MPI-IO collective buffering to align data often not needed, usually much better performance turning it off except for extreme cases
        - Use MPI-IO backend in PNetCDF and HDF5, set ROMIO hints to turn off collective buffering
            - export ROMIO_HINTS=<fullpathfilename>
                - romio_cb_write disable
                - romio_cb_read disable
        - OR set MPI-IO collective buffering hints and cpu binding for optimum data movement
            - cb_buffer_size 16777216
            - cb_config_list *:8
            - eg up to first 64 ranks: --cpu-bind=verbose,list:4:56:5:57:6:58:7:59:8:60:9:61:10:62:11:63:12:64:13:65:14:66:15:67:16:68:17:69:18:70:19:71:20:72:21:73:22:74:23:75:24:76:25:77:26:78:27:79:28:80:29:81:30:82:31:83:32:84:33:85:34:86:35:87
        - File-per-process handled much better
- POSIX application calls can be kept in user-space with DFS interception - libraries for low-latency as opposed to kernel-space
    - LD_PRELOAD=/usr/lib64/libpil4dfs.so

MPI-IO ROMIO One-sided aggregation speedup

- Alternate algorithm using one-sided communication in aggregation portion for two-phase collective buffering in MPI-IO
  - Can offer significant speedup in some cases
  - Enabled through environment variables
    - ROMIO_WRITE_AGGMETHOD=2
    - ROMIO_READ_AGGMETHOD=2
  - https://github.com/pmodels/mpich/blob/main/src/mpi/romio/adio/common/ad_tuning.c
    - Run with ROMIO_ONESIDED_INFORM_RMW=1 to determine if read-modify-write is needed then for better performance either;
      - ROMIO_ONESIDED_ALWAYS_RMW=1
        - Use this for PNetCDF and HDF5
    - Or
      - ROMIO_ONESIDED_NO_RMW=1

LAMMPS flare Lustre vs DAOS checkpoint 2M atoms per node and file per node



Acknowledgements

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

DAOS

Distributed Asynchronous Object Store

Overview of DAOS and Best Practices (Part II)

Kaushik Velusamy
ALCF Developer Session Oct 23, 2024

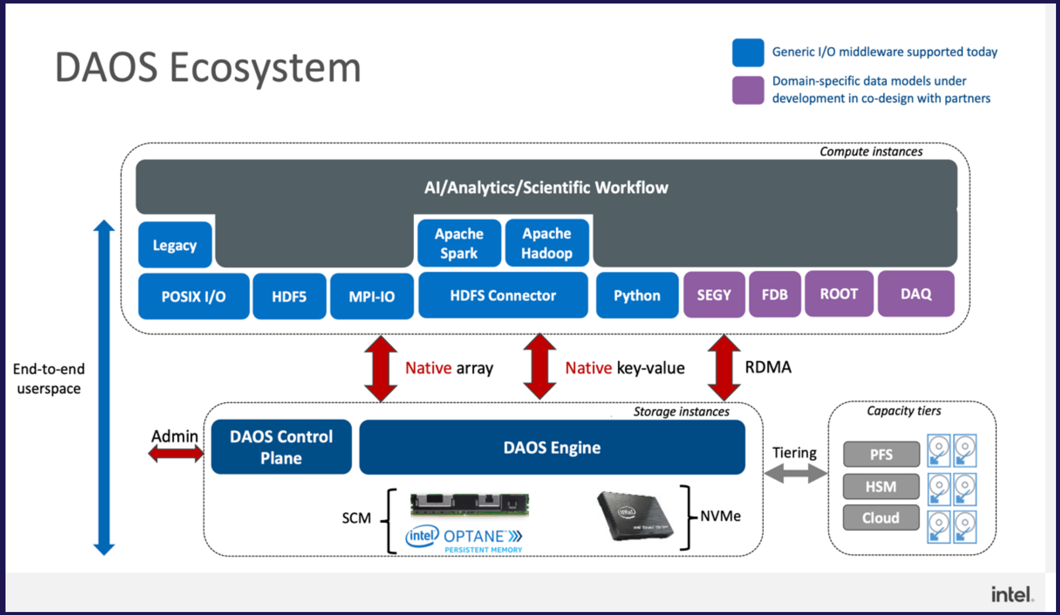
1. Why DAOS

- DAOS is a major file system in Aurora : 230 PB, >30 TB/s, 1024 DAOS Nodes
- Open-source software-defined **object store**
- Designed for massively **distributed** Non Volatile **Memory** (NVM) and NVMe **SSD**
- DAOS presents a unified storage model with a native Key-array Value storage interface – POSIX, MPIO, HDF5 etc
- Storage and retrieval of objects in a distributed, parallel, and **asynchronous** manner.
- Advanced data protection, self-healing, redundancy, versioning, distribution and fine-grained data control.
- Efficient for unstructured data.
- Efficient for accessing small data.
- High bandwidth, low latency, and IOPS

Aurora DAOS : #1 in the IO500 Bandwidth with 300 / 1024 DAOS nodes

Easy Write 20 TB/s	Easy Read 12 TB/s
Hard Write 4 TB/s	Hard Read 9 TB/s

System	Capacity	Performance
DAOS	230 PB	>=30 TB/s
Flare	91 PB 160 OSTs 40 MDTs	>=650 GB/s
Gecko Home	12 PB 32 OSTs 12 MDTs	
/tmp DRAM	1 TB usable ~<0.5TB	



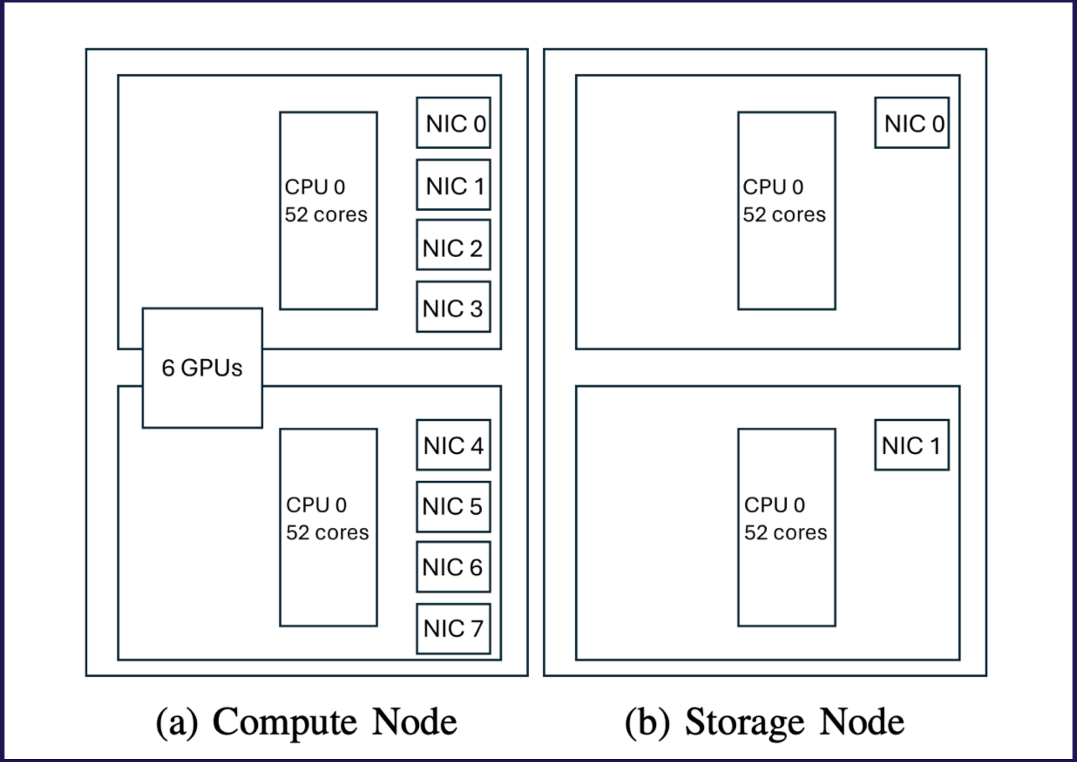
Argonne NATIONAL LABORATORY

Compute nodes

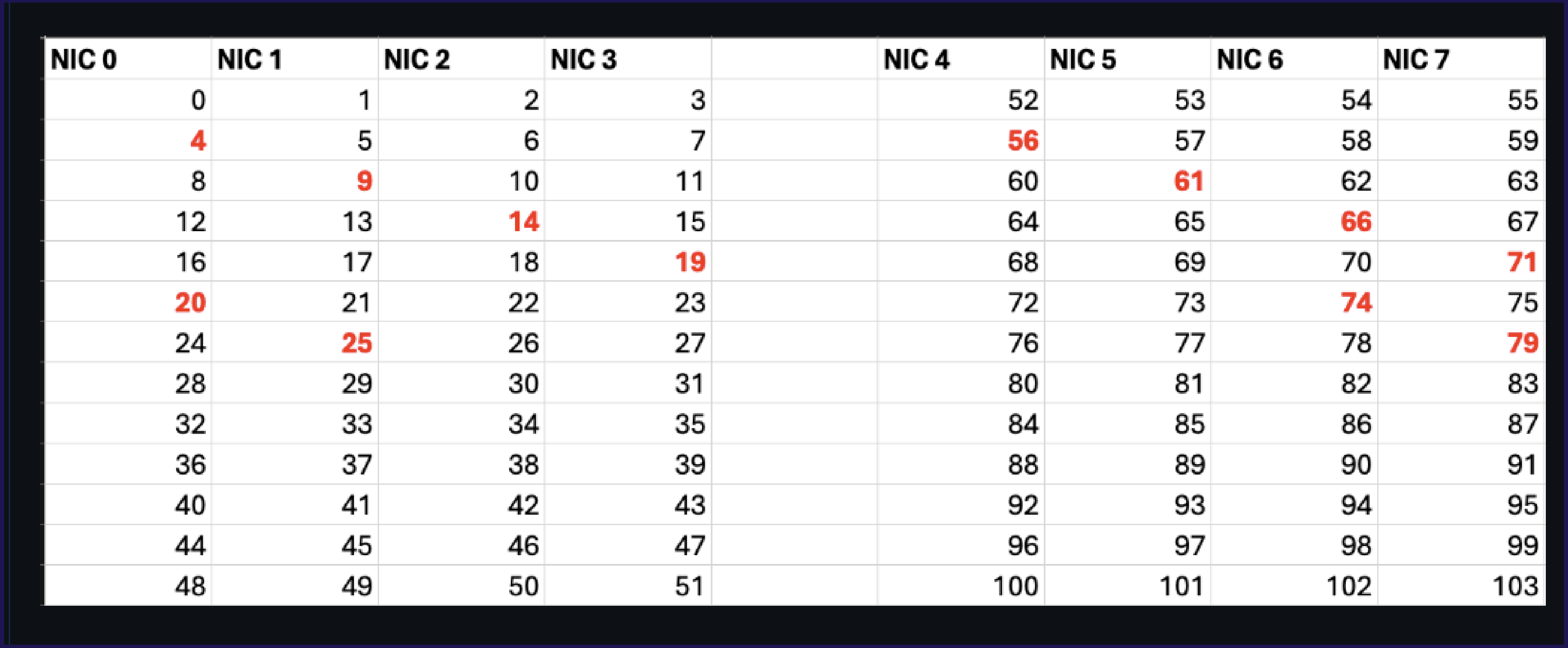
- 10,624 Compute Nodes
- 8 NICs / Compute Node
- (25 ~ 20) GB/s X 8 NICs = 200 GB/s

Daos nodes

- 1024 Total DAOS Server Nodes
- 2 NICs / Daos Node
- (25 ~ 20) GB/s X 2 NICs = 40GB/s



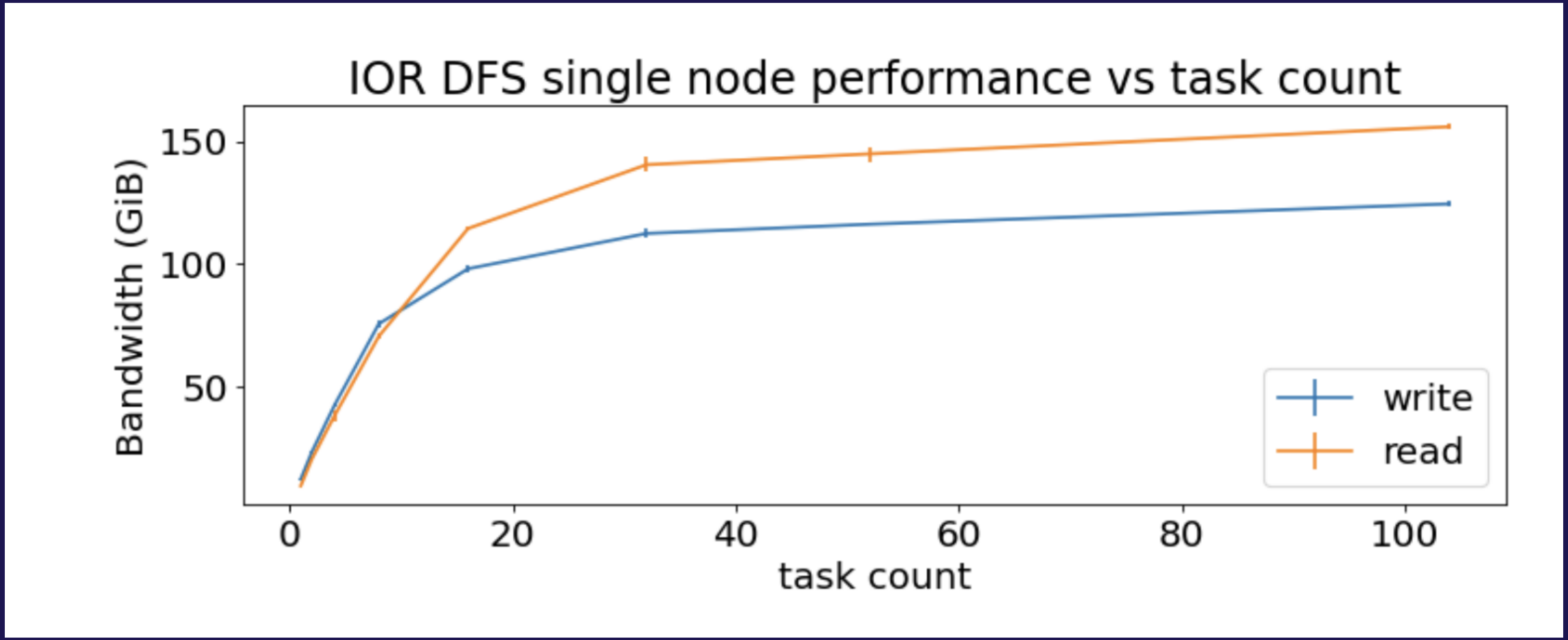
Every read and write goes over the network. Hence NIC binding is the key



NIC 0	NIC 1	NIC 2	NIC 3		NIC 4	NIC 5	NIC 6	NIC 7
0	1	2	3		52	53	54	55
4	5	6	7		**56**	57	58	59
8	**9**	10	11		60	**61**	62	63
12	13	**14**	15		64	65	**66**	67
16	17	18	**19**		68	69	70	**71**
20	21	22	23		72	73	**74**	75
24	**25**	26	27		76	77	78	**79**
28	29	30	31		80	81	82	83
32	33	34	35		84	85	86	87
36	37	38	39		88	89	90	91
40	41	42	43		92	93	94	95
44	45	46	47		96	97	98	99
48	49	50	51		100	101	102	103

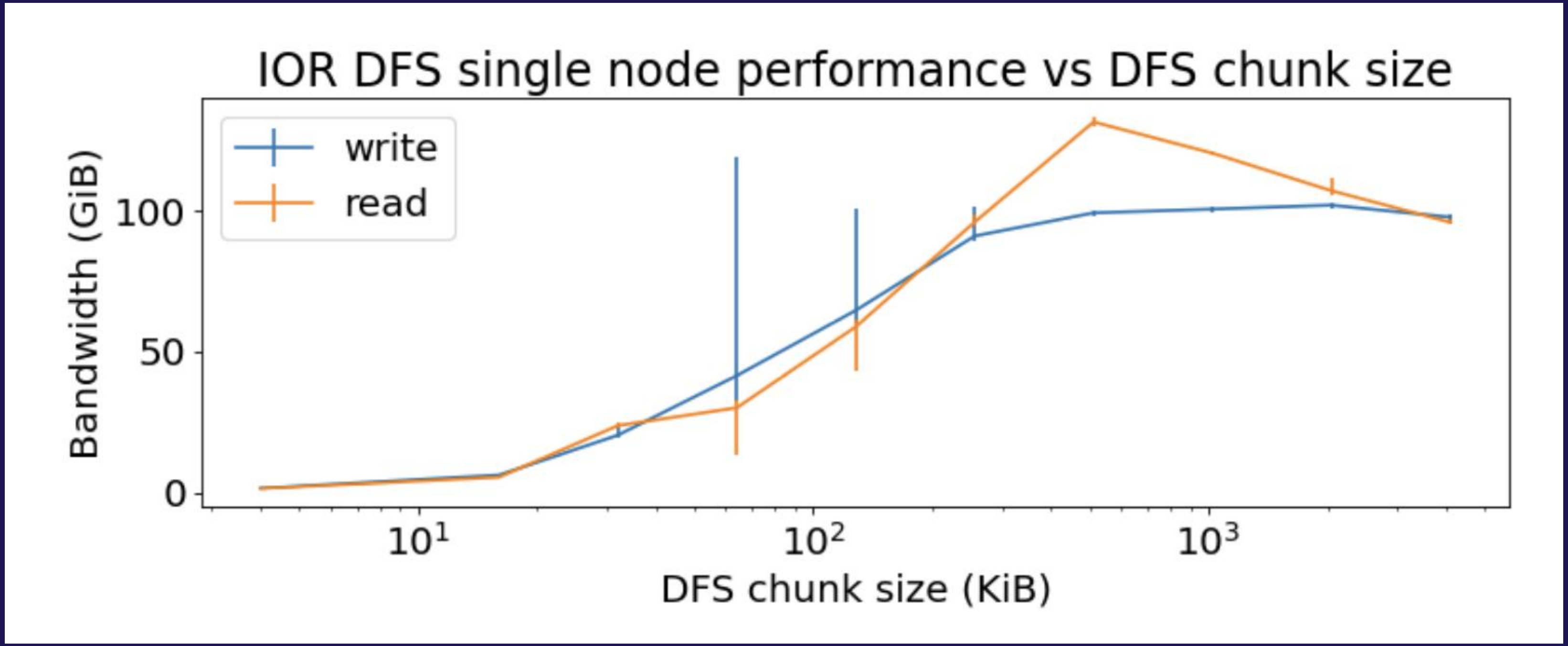
CPU_BINDING1=list:4:9:14:19:20:25:56:61:66:71:74:79

32 processes (4 processes per network card)



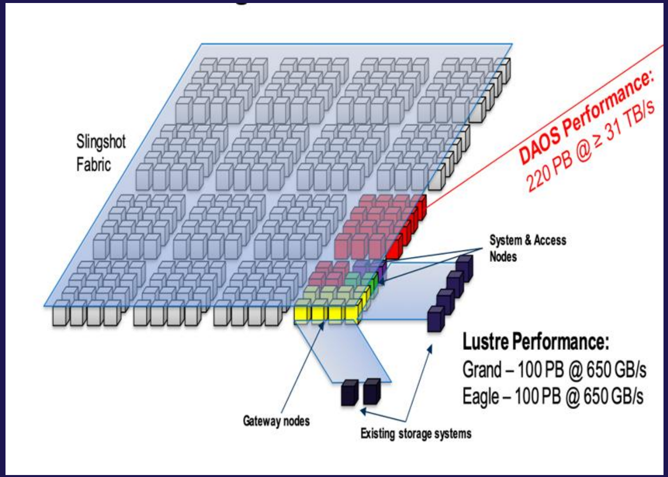
Rob Latham, Robert Ross, Philip Carns, Shane Snyder, Kevin Harms, Kaushik Velusamy, Paul Coffman, Gordon McPheeters, "Initial Experiences with DAOS Object Storage on Aurora" ", 9th International Parallel Data Systems Workshop held in conjunction with the SC 2024

512 KB injection

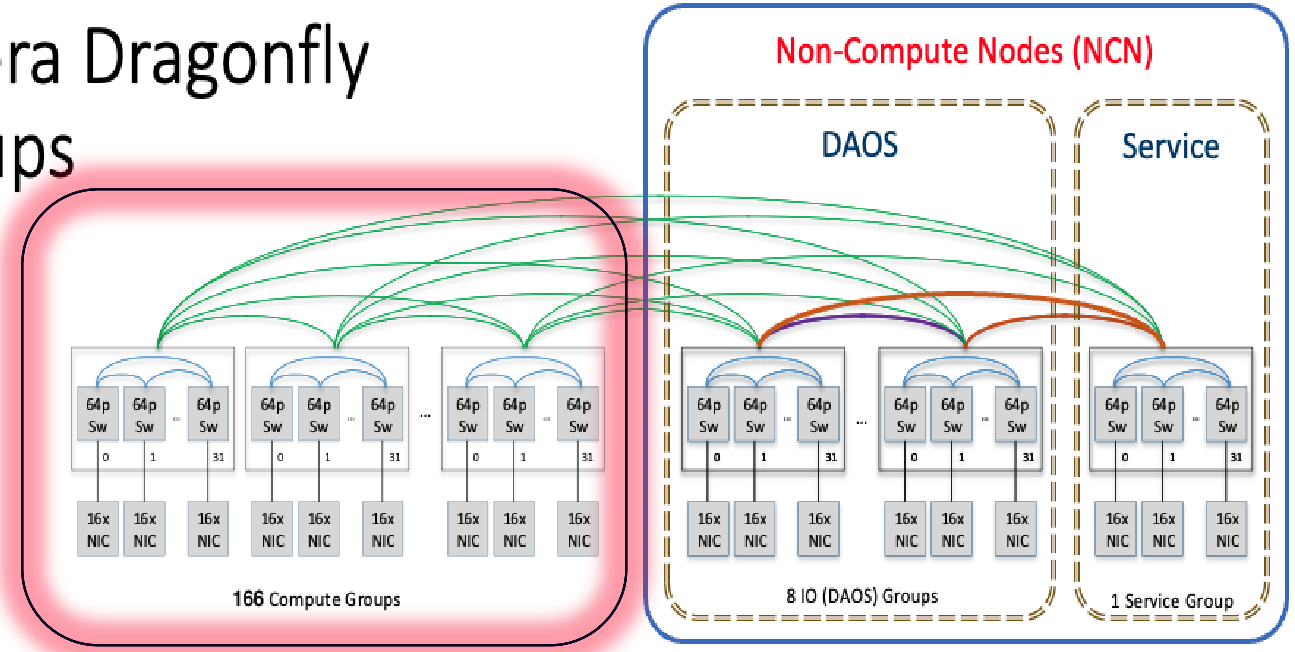


Rob Latham, Robert Ross, Philip Carns, Shane Snyder, Kevin Harms, Kaushik Velusamy, Paul Coffman, Gordon McPheeters, "Initial Experiences with DAOS Object Storage on Aurora" ", 9th International Parallel Data Systems Workshop held in conjunction with the SC 2024

Network Architecture – slingshot fabric - Dragonfly



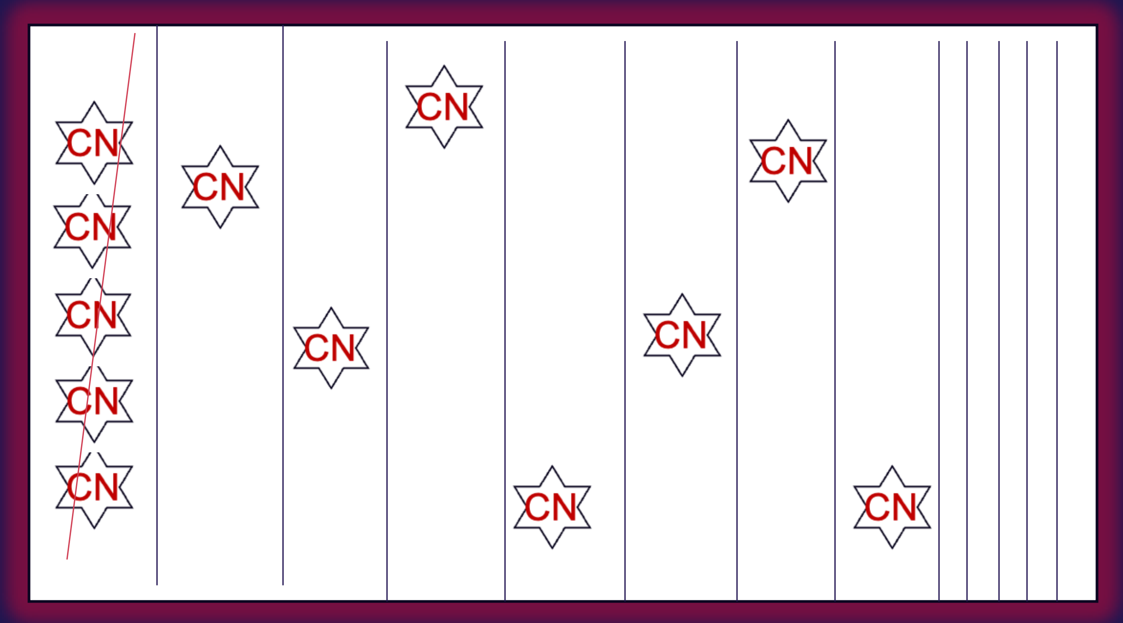
Aurora Dragonfly Groups

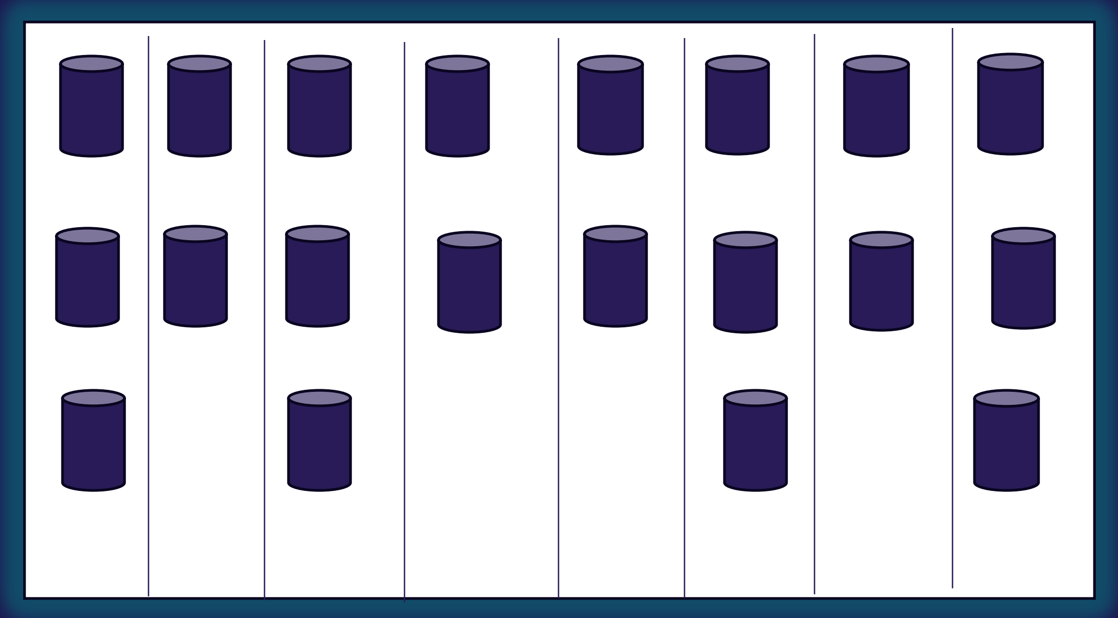


Each Link is 50GB/s bidirectional, 25GB/s unidirectional: ▬ 1 link per arc ▬ 2 links per arc ▬ 8 links per arc ▬ 22 links per arc

- 1-D Dragonfly Topology - 175 total groups (166 compute + 8 IO + 1 Service)
  - All the global links are optical, all the local links are electrical
  - 2 global links between any two compute groups
  - 22 links between any two IO groups, 8 links between the Service group and each IO group
- All NCN will be racked in HPE cabinets and under HPE System Management.
  - The DAOS cluster is 64 DAOS racks of 1024 DAOS nodes organized in 8 Dragonfly groups.
  - The Service cluster is a single Dragonfly group composed of 6 racks of I/O gateway nodes and 6 racks of HPE Cray front-end nodes (FENs).

Example Scenario on placement of nodes for effective B/W utilization
This is what is happening





166 compute dragonfly groups in 166 Cabinets

64 CN per rack *166 groups – 10,624 CNs

8 daos dragon fly groups in 64 cabinets
We place daos nodes as equally as possible on all 8 daos groups

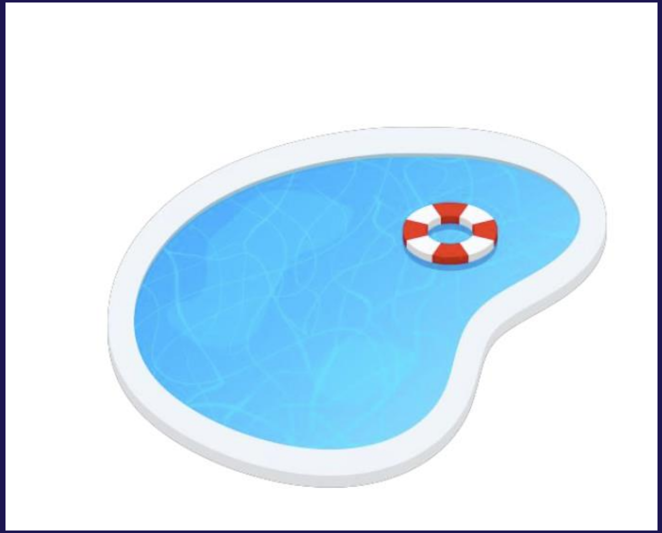
20 /1024 daos nodes
128 /1024 daos nodes
600 /1024 daos nodes

Step 1: What is a DAOS pool ?

```
Mail to : support@alcf.anl.gov

Project Name:
Alcf User Names:
Total Space requested ( 100 TBs ~++)
Justification:
```

```
kaushikvelusamy@aurora-uan-0010:~/> daos pool query datascience
Pool 9ca55e01-16c2-4801-8ff2-304b6307d4d5, ntarget=640, disabled=0, leader=0,
version=3982
Pool space info:
- Target(VOS) count:640
- Storage tier 0 (SCM):
  Total size: 1.5 TB
  Free: 1.4 TB, min:2.1 GB, max:2.1 GB, mean:2.1 GB
- Storage tier 1 (NVMe):
  Total size: 48 TB
  Free: 48 TB, min:76 GB, max:76 GB, mean:76 GB
Rebuild done, 0 objs, 0 recs
```

Physically allocated dedicated storage for your project.



Step 2 : Daos sanity tests

```
module use /soft/modulefiles
module load daos
```

```
env | grep DRPC
ps –ef | grep daos
clush --hostfile ${PBS_NODEFILE} ps –ef | grep agent | grep -v grep'  | dshbak -c
daos pool query ${DAOS_POOL}
```

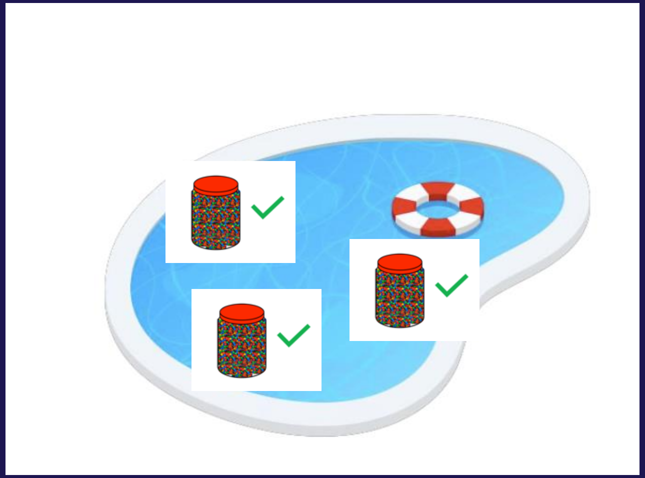
```
kaushikvelusamy@aurora-uan-0010:~/> daos pool query ${DAOS_POOL}
Pool 9ca55e01-16c2-4801-8ff2-304b6307d4d5, ntarget=640, disabled=0, leader=0, version=3982
Pool space info:
- Target(VOS) count:640
- Storage tier 0 (SCM):
  Total size: 1.5 TB
  Free: 1.4 TB, min:2.1 GB, max:2.1 GB, mean:2.1 GB
- Storage tier 1 (NVMe):
  Total size: 48 TB
  Free: 48 TB, min:76 GB, max:76 GB, mean:76 GB
Rebuild done, 0 objs, 0 recs
kaushikvelusamy@aurora-uan-0010:~/>
```

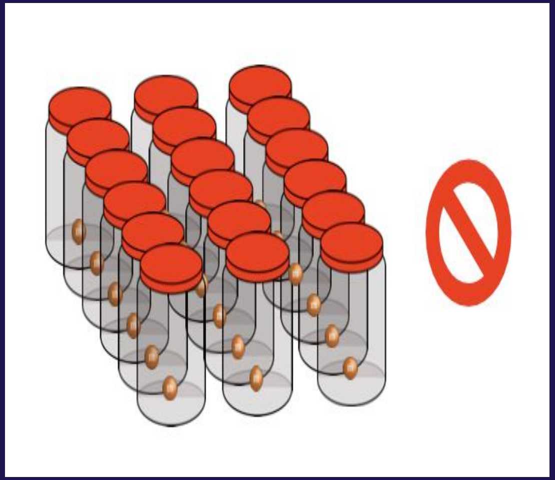
```
kaushikvelusamy@aurora-uan-0010:~/> daos pool query ${DAOS_POOL}


external ERR  # [1628306.082628] mercury->rpc [error] /builddir/build/BUILD/mercury-2.4.0rc5/src/mercury_core.c:4479 hg_core_send_input_cb() NA callback returned error
(NA_HOSTUNREACH)

hg   ERR  src/cart/crt_hg.c:1395 crt_hg_req_send_cb(0x562c74806520) [opc=0xff040001 (CART:CRT_OPC_PROTO_QUERY) rpcid=0x17fa8bf0002be20 rank:tag=93:0] RPC failed; rc:
DER_HG(-1020): 'Transport layer mercury error'
```

What is a DAOS container?

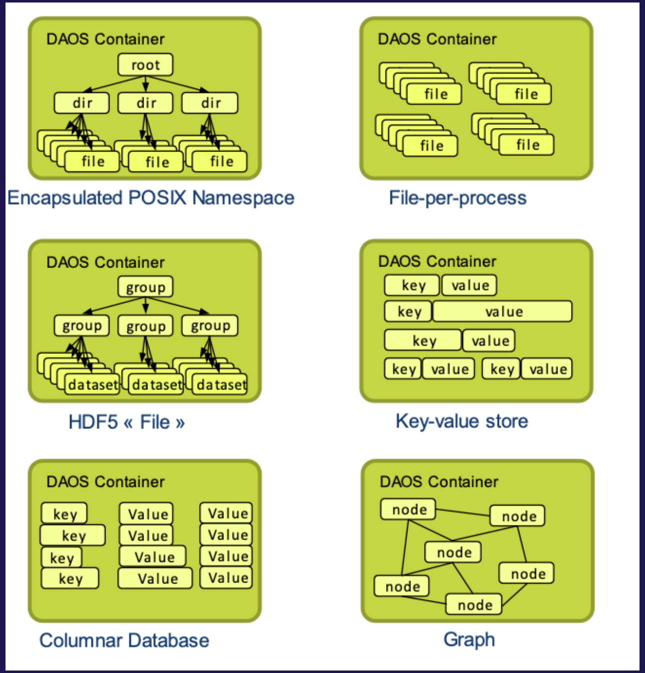
- /tmp
  - /poolname
    - /containername
      - exp1
      - exp2

- A pool contains *thousands* of containers
- Basic unit of storage from user perspective
- Containers can be of type [POSIX, HDF, PYTHON, SPARK]*
- Currently: POSIX container launched through DFUSE





Future DAOS containers

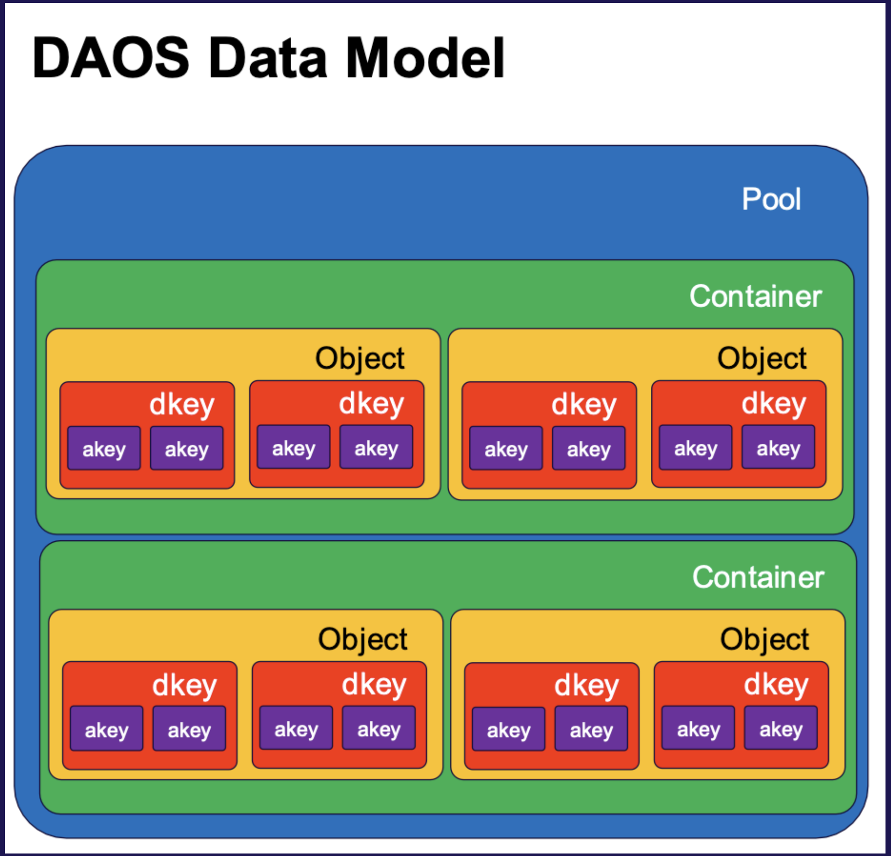


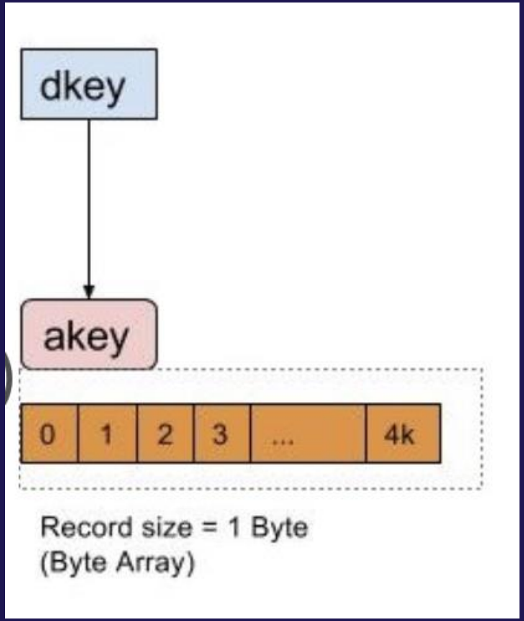
Step 3 : Creating a container

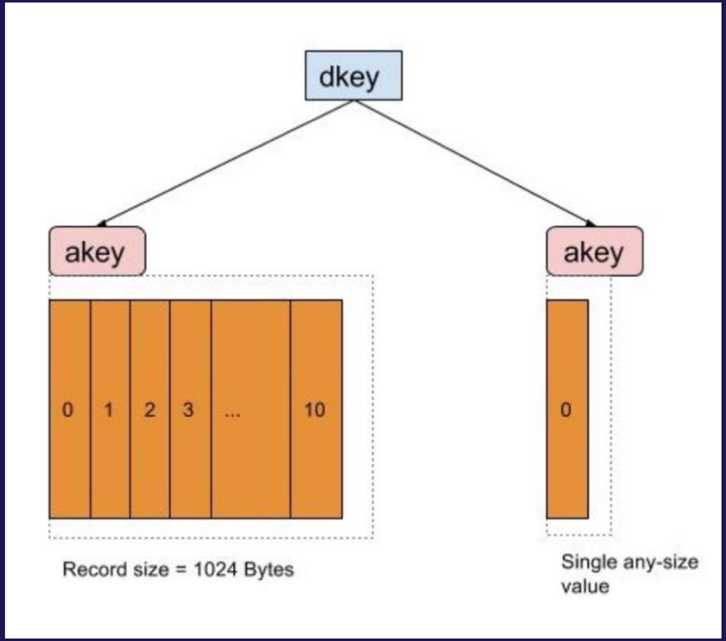
```
daos container create --type POSIX ${DAOS_POOL}  ${DAOS_CONT} --properties rd_fac:1
```

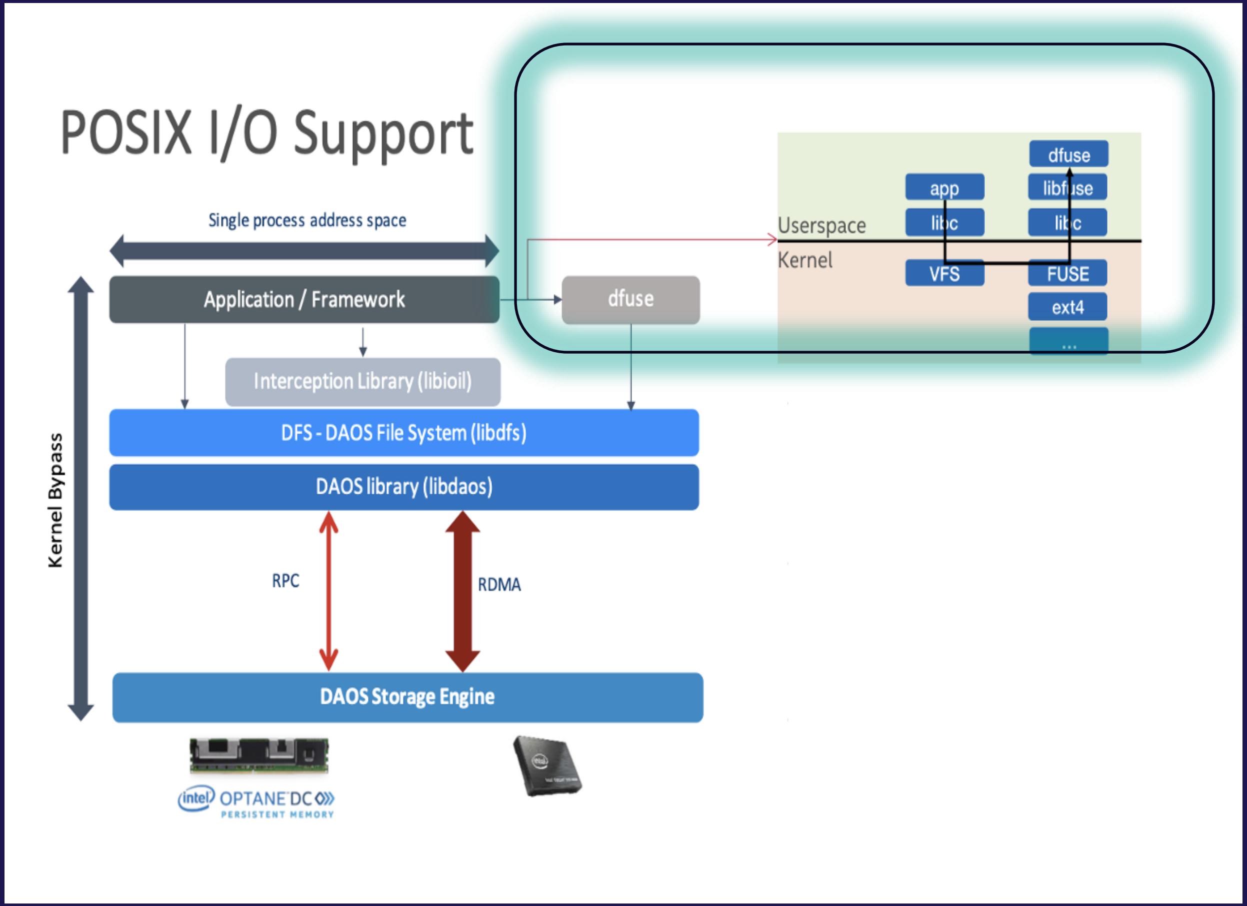
```
kaushikvelusamy@aurora-uan-0010:~/> daos cont list datascience

UUID                                 Label
----                                 -----
d85f4fac-d486-4e76-a46d-a60135b2dc64 kaushik_resnet_dataset_cont_sx
2fe037ee-715d-4186-aa19-250fa1fc2988 posix-s1
c9ffb011-28d1-4356-9afd-feaf8269d2bb posix-s4
```

```
daos container query     $DAOS_POOL_NAME  $DAOS_CONT_NAME
daos container get-prop  $DAOS_POOL_NAME  $DAOS_CONT_NAME
daos container set-prop  $DAOS_POOL_NAME  $DAOS_CONT_NAME
daos container list      $DAOS_POOL_NAME  $DAOS_CONT_NAME
daos pool      autotest  $DAOS_POOL_NAME
daos container destroy   $DAOS_POOL_NAME  $DAOS_CONT_NAME
daos container check --pool=$DAOS_POOL_NAME --cont=$DAOS_CONT_NAME
```


DAOS Data Model
Pool
Container
Object
dkey
akey
Container
Object
dkey
akey


dkey
akey
0
1
2
3
...
4k
Record size = 1 Byte
(Byte Array)


dkey
akey
akey
0
1
2
3
...
10
0
Record size = 1024 Bytes
Single any-size value



1. POSIX Mode
2. MPIIO Mode
3. DFS Mode

Interception library for POSIX mode

```
mpiexec
mpiexec --env LD_PRELOAD=/usr/lib64/libioil.so
mpiexec --env LD_PRELOAD=/usr/lib64/libpil4dfs.so
```

Exercise 2: Interacting with DAOS (Login node)

```
mkdir –p /tmp/${DAOS_POOL}/${DAOS_CONT}
start-dfuse.sh -m /tmp/${DAOS_POOL}/${DAOS_CONT} --pool ${DAOS_POOL} --cont ${DAOS_CONT}
```

```
mount | grep dfuse
ls /tmp/${DAOS_POOL}/${DAOS_CONT}
cd /tmp/datascience/thunder_1/
cp
mv
cat
```

```
fusermount3 -u /tmp/${DAOS_POOL}/${DAOS_CONT}
```

POSIX Container

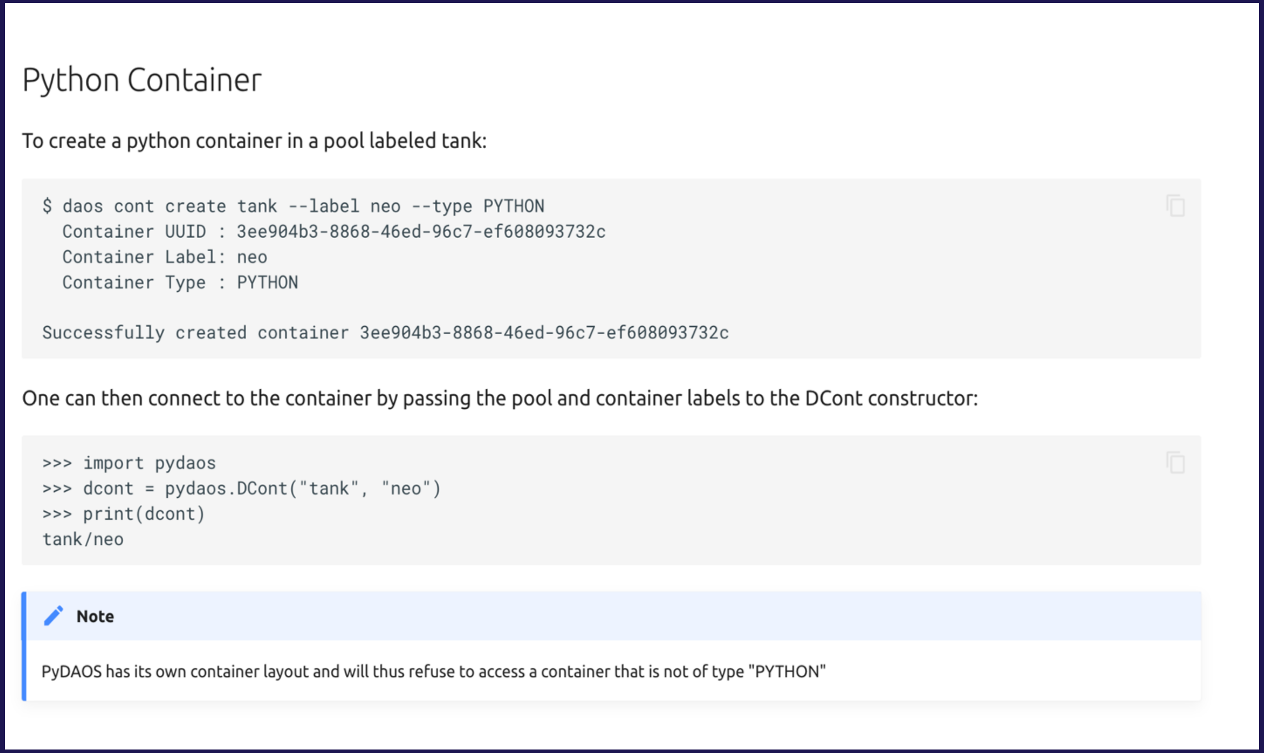
```
>ls /tmp/datascience/kaushik_resnet_dataset_cont/
train-data/ test-data/ val-data/
>
```

Exercise 2: Interacting with DAOS (Compute nodes)

```
# launched using pdsh on all compute nodes mounted at: /tmp/<pool>/<container>

launch-dfuse.sh ${DAOS_POOL_NAME}:${DAOS_CONT_NAME}

mount | grep dfuse
ls /tmp/${DAOS_POOL}/${DAOS_CONT}/

clean-dfuse.sh  ${DAOS_POOL_NAME}:${DAOS_CONT_NAME}
```

- ✓ Verified access to a DAOS POOL
- ✓ Created a DAOS posix container
- ✓ Mounted the container on our nodes
- ✓ Tested cp,mv,cat on the container
- Using the container from MPIEXEC

```
LD_PRELOAD=/usr/lib64/libpil4dfs.so mpiexec -np ${NRANKS} -ppn ${RANKS_PER_NODE} \
                                                                        --cpu-bind ${CPU_BINDING1}  \
                            --no-vni -genvall
                                                                        thunder/svm_mpi/run/aurora/wrapper.sh
                                                                        thunder/bin/thundersvm-train \
                            -s 0 -t 2 -g 1 -c 10 -o 1  /tmp/datascience/thunder_1/real-sim_M100000_K25000_S0.836
```



Python Container

To create a python container in a pool labeled tank:

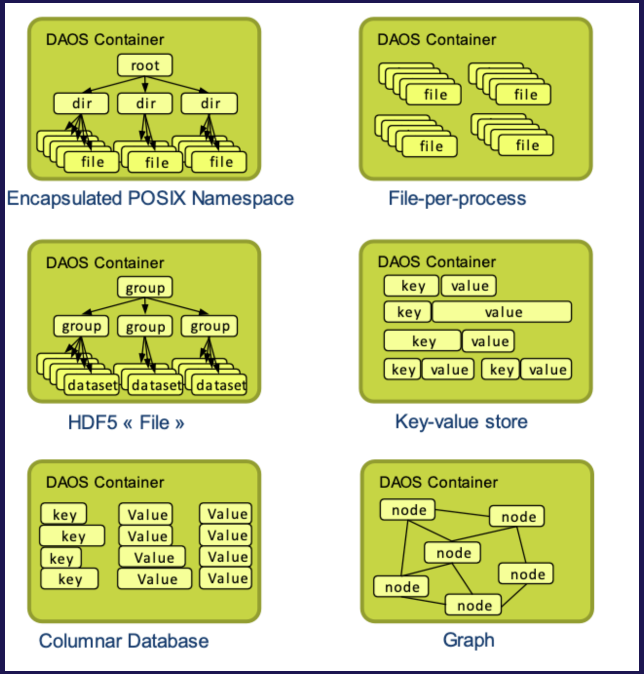
```
$ daos cont create tank --label neo --type PYTHON
  Container UUID : 3ee904b3-8868-46ed-96c7-ef608093732c
  Container Label: neo
  Container Type : PYTHON

Successfully created container 3ee904b3-8868-46ed-96c7-ef608093732c
```

One can then connect to the container by passing the pool and container labels to the DCont constructor:

```
>>> import pydaos
>>> dcont = pydaos.DCont("tank", "neo")
>>> print(dcont)
tank/neo
```

Note

PyDAOS has its own container layout and will thus refuse to access a container that is not of type "PYTHON"



Lustre job script

```
#!/bin/bash -x
# qsub -l select=1 -l walltime=01:00:00 -A Aurora_deployment -k doe -l filesystems=flare -q lustre_scaling ./pbs_script1.sh  or - I


mpiexec -np 16 -ppn 16 -genvall --no-vni ior  -a mpio  -i 5 -t 16M -b 640M  -w  -r -o ./io_1.dat
```

DAOS job script

```
#!/bin/bash -x

# qsub -l select=1 -l walltime=00:20:00 -A Aurora_deployment -k doe -l filesystems=flare -q lustre_scaling -l daos=default
                                                                                             ./script1.sh  or - I

module use /soft/modulefiles
module load daos
launch-dfuse.sh datascience:kaus_posix_llm-agpt_sx

LD_PRELOAD=/usr/lib64/libpil4dfs.so mpiexec -np 16 -ppn 16 -genvall
                                     --no-vni ior  -a mpio  -i 5 -t 16M -b 640M  -w  -r
                                     -o /tmp/datascience/kaus_posix_llm-agpt_sx/io_1.dat

clean-dfuse.sh datascience:kaus_posix_llm-agpt_sx
```

Three modes in DAOS

1. POSIX Mode
2. MPIIO Mode
3. DFS Mode

MPI-IO MODE for DAOS

For fine tuning,

```
export ROMIO_PRINT_HINTS=1

echo "cb_nodes 128" >> ${PBS_O_WORKDIR}/romio_hints

mpiexec  --env ROMIO_HINTS = romio_hints_file daos:/YOUR_PROGRAM

mpiexec  --env MPICH_MPIIO_HINTS = path_to_your_file*:cb_config_list=#*:2#
                                                       :romio_cb_read=enable
                                                       :romio_cb_write=enable
                                                       :cb_nodes=32
                                       daos:/YOUR PROGRAM
```

DFS Mode

-a DFS
interception
DAOS vol plugins
DFS APIs

```
#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#include <daos.h>

#include <daos_fs.h>

int main(int argc, char **argv)

{

    dfs_t *dfs;

    d_iov_t global;

    ret = MPI_Init(&argc, &argv);
    ret = MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    ret = dfs_init();
    ret = dfs_connect(getenv("DAOS_POOL"), NULL, getenv("DAOS_CONT"), O_RDWR, NULL, &dfs); }
    ret = dfs_open(dfs, NULL, filename, S_IFREG|S_IRUSR|S_IWUSR,  O_CREAT|O_WRONLY,  obj_class, chunk_size, NULL, &obj);
    ret = dfs_write(dfs, obj, &sgl, off, NULL);
    ret = dfs_read(dfs, obj, &sgl, off, &read, NULL);
    ret = dfs_disconnect(dfs);
    ret = daos_fini();
    ret = MPI_Finalize();

}
```

Interception library for POSIX mode

```
mpiexec
mpiexec --env LD_PRELOAD=/usr/lib64/libioil.so
mpiexec --env LD_PRELOAD=/usr/lib64/libpil4dfs.so
```

Darshan-runtime from aurora

```
module use /soft/modulefiles
module load daos
module list
git clone https://github.com/darshan-hpc/darshan.git
git checkout snyder/dev-daos-module-3.4
./prepare.sh
```

`mkdir /home/kaushikvelusamy/soft/profilers/darshan-daos/darshan-install`

```
./configure --prefix=/home/kaushikvelusamy/soft/profilers/darshan-daos/darshan-install  \
            --with-log-path=/home/kaushikvelusamy/soft/profilers/darshan-daos/darshan-logs \
            --with-jobid-env=PBS_JOBID \
            CC=mpicc --enable-daos-mod

make && make install

chmod 755 ~/soft/profilers/darshan-daos/darshan/darshan-install/darshan-mk-log-dirs.pl
```

`mkdir /home/kaushikvelusamy/soft/profilers/darshan-daos/darshan-logs`

```
cd /home/kaushikvelusamy/soft/profilers/darshan-daos/darshan-logs
~/soft/profilers/darshan-daos/darshan/darshan-install/darshan-mk-log-dirs.pl
~/soft/profilers/darshan-daos/darshan-install/bin/darshan-config  --log-path
```

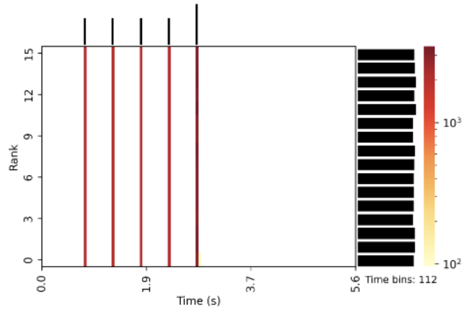
```
mpiexec --env LD_PRELOAD=~/soft/profilers/darshan-daos/darshan-install/lib/libdarshan.so:/usr/lib64/libpil4dfs.so
        -np 32 -ppn 16  --no-vni -genvall \
        ior -a DFS  --dfs.pool=datascience_ops --dfs.cont=ior_test1   \
            -i 5 -t 16M -b 2048M  -w  -r -C -e    -c  -v -o /ior_2.dat
```

darshan-util from laptop

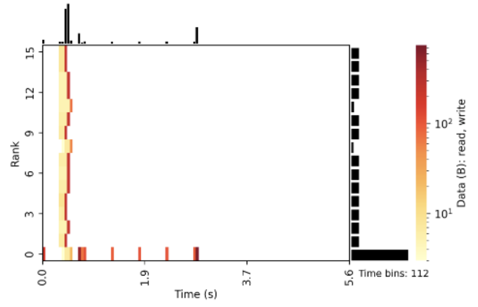
```
conda info –envs

conda activate env-non-mac-darshan-temp

/Users/kvelusamy/Desktop/tools/spack/share/spack/setup-env.sh


spack install darshan darshan-util


export DYLD_FALLBACK_LIBRARY_PATH=/Users/kvelusamy/Desktop/tools/spack/opt/spack/darwin-ventura-
m1/apple-clang-14.0.3/darshan-util-3.4.4-
od752jyfljrrey3d4gjeypdcppho42k2/lib/:$DYLD_FALLBACK_LIBRARY_PATH


darshan-parser ~/Downloads/kaushikv_ior_id917110-44437_10-23-55830-632270104473632905_1.darshan


python3 -m darshan summary
        ~/Downloads/kaushikv_ior_id917110-44437_10-23-55830-632270104473632905_1.darshan
```

DAOS – Darshan – Analyser – From laptop

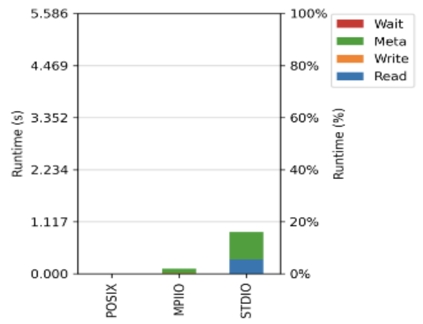
Heat Map: HEATMAP MPIIO



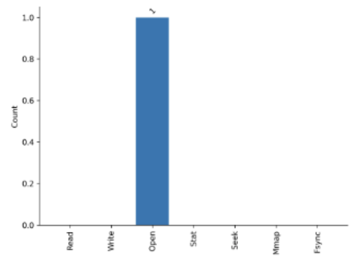
Heat Map: HEATMAP STDIO



I/O Cost



Operation Counts

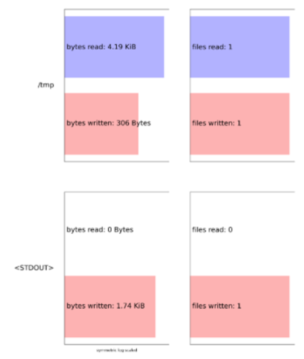


Histogram of I/O operation frequency.

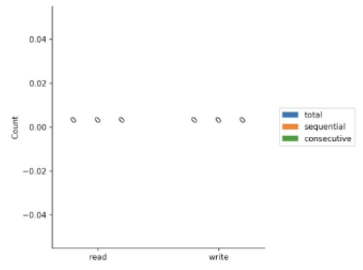
Common Access Sizes

Access Size	Count
256	640
272	39
544	6
328	3

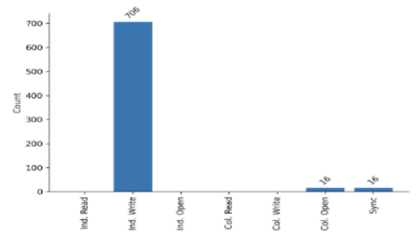
Data Access by Category



Access Pattern

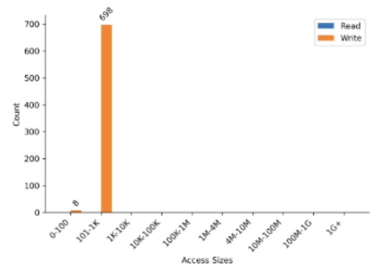


Operation Counts

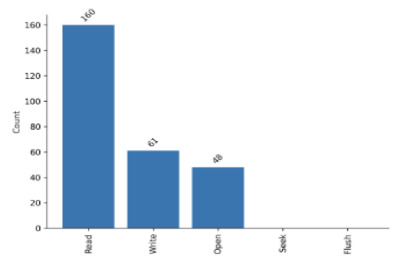


Histogram of I/O operation frequency.

Access Sizes



Operation Counts



Histogram of I/O operation frequency.

DAOS CP – DCP – Data mover vs CP

-
  ```
  cp      /lus/flare/projects/CSC250STDM10_CNDA/kaushik/thundersvm/input_data/real-sim_M100000_K25000_S0.836
          /tmp/${DAOS_POOL}/${DAOS_CONT}
  ```

-
  ```
  daos filesystem copy
          --src /lus/flare/projects/CSC250STDM10_CNDA/kaushik/thundersvm/input_data/real-sim_M100000_K25000_S0.836
          --dst daos://tmp/${DAOS_POOL}/${DAOS_CONT}
  ```

Note: you don't need to add this in your job script – Mount anywhere later

```
daos-soft/mpifileutils/bin> ls

dbcast  dbz2  dchmod  dcmp  dcp  dcp1  ddup  dfilemaker1  dfind  dreln  drm  dstripe  dsync  dtar  dwalk
```

Ref: https://docs.daos.io/v2.4/testing/datamover/

Thunder SVM with and without DAOS jobscript

Different daos cluster with different number of servers currently active

20	/1024 daos nodes	2%	1 TB/s
128	/1024 daos nodes	12.50%	5 TB/s
600	/1024 daos nodes	60%	10 TB/s
800	/1024 daos nodes	78%	20 TB/s
1024	/1024 daos nodes	100%	30 TB/s

HACC results with and without DAOS

```
Lustre Flare:
HACC reaching lustre peak theoretical max 600GB/s
160 OSTs 40 MDTs 48 Gateway nodes



NUM_OF_NODES=512  TOTAL_NUM_RANKS=6144  RANKS_PER_NODE=12

start time: Mon 29 Jul 2024 01:22:53 AM CDT

RecordSize = 38
NpTotal = 646736283406 (646736.283406 million particles)
24575978769428 bytes uncompressed data (24,575,978.769428
MB)

Wrote 2 variables to ./lustre_out_712762.aurora-pbs-
0001.hostmgmt.cm.aurora.alcf.anl.gov_512/lus_pos_test_kaus
_1_ (49952 bytes) in 0.2901s: 0.164212 MB/s

Wrote 9 variables to ./lustre_out_712762.aurora-pbs-
0001.hostmgmt.cm.aurora.alcf.anl.gov_512/lus_pos_test_kaus
_1_ (24575980196884 bytes) in 39.2264s: 597492 MB/s


end time : Mon 29 Jul 2024 01:23:52 AM CDT
```

```
DAOS OPS 23
HACC reaching DAOS OPS peak theoretical max with 23 DAOS
servers:
With 20 servers : 1000 GB/s, 40 NICs * 25GB/s/NIC.
With SX posix daos container

NUM_OF_NODES=512  TOTAL_NUM_RANKS=6144  RANKS_PER_NODE=12

Thu 15 Aug 2024 10:06:02 PM CDT

RecordSize = 38
NpTotal = 646736283406 (646736.283406 million particles)
24,575,978,769,428 bytes uncompressed data
(24,575,978.769428 MB)

Wrote 2 variables to
/tmp/datascience_ops/hacc_1/daos_pos_test_kaus_1_ (49952
bytes) in 0.349028s: 0.136488 MB/s

Wrote 9 variables to
/tmp/datascience_ops/hacc_1/daos_pos_test_kaus_1_
(24575979517492 bytes) in 25.2472s: 928321 MB/s --> 900
GB/s

Thu 15 Aug 2024 10:06:35 PM CDT
```

Best practices

- Check `qsub -l daos=default`
- Did you load DAOS module? `module load daos`
- Do you have your DAOS pool allocated? `daos pool query datascience`
- Is Daos client running on all your nodes? `ps -ef | grep daos`
- Is your container mounted on all nodes? `mount | grep dfuse`
- Can you ls in your container? `ls /tmp/${DAOS_POOL}/${DAOS_CONT}`
- Did your I/O Actually fail?
- What is the health property in your container? `daos container get-prop $DAOS_POOL $CONT`
- Is your space full? Min and max `daos pool query datascience`
- Does your query show failed targets or rebuild in process? `daos pool query datascience`
- `daos pool      autotest`
- `Daos container check`

Acknowledgements

- Gordon Mcpheeters, Paul Coffman, Vitali Morozov, Huihuo Zheng, Mohamad Charawi, Kevin Harms and Venkat Vishwanath.
- Intel DAOS team, HPE cray team & MCS I/O team
- This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.
- This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

Reference

- https://docs.daos.io/v2.6/user/workflow/
- https://docs.daos.io/v2.6/overview/architecture/
- *Rob Latham, Robert Ross, Philip Carns, Shane Snyder, Kevin Harms, Kaushik Velusamy, Paul Coffman, Gordon McPheeters, "Initial Experiences with DAOS Object Storage on Aurora" ", 9th International Parallel Data Systems Workshop held in conjunction with the SC 2024*
- https://docs.alcf.anl.gov/aurora/data-management/daos/daos-overview/ [will be updated soon]