2022 ECP Community BOF Days

Sharing Experience on HDF5 VOL Connectors Development and Maintenance

Approved for public release

Jordan Henderson (The HDF Group)

Jerome Soumagne (The HDF Group)

Neil Fortner (The HDF Group)

Houjun Tang (LBNL)

Huihuo Zheng (ANL)

Suren Byna (LBNL)

Scot Breitenfeld (The HDF Group)

Larry Knox (The HDF Group)

Kai-yuan Hou (Northwestern University)

Wei-keng Liao (Northwestern University)

Wednesday, May 11, 2022
1:00 PM – 2:00 PM ET

EXASCALE COMPUTING PROJECT

National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

Sharing Experience on HDF5 VOL Connectors Development and Maintenance

Scot Breitenfeld

- Suren Byna, Houjun Tang, Jean Luca Bez, Bin Dong (Lawrence Berkeley National Laboratory)
- Scot Breitenfeld, Dana Robinson, Jerome Soumagne, Jordan Henderson, Neil Fortner, and Neelam Bagha (The HDF Group)
- Venkat Vishwanath and Huihuo Zheng (Argonne National Laboratory)
- Wei-keng Liao, Kai-Yuan Hou (Northwestern University)

BERKELEY LAB Lawrence Berkeley National Laboratory · The HDF Group · Argonne NATIONAL LABORATORY · Northwestern University · ECP EXASCALE COMPUTING PROJECT

Overview

- HDF5 VOL developers will share their experiences creating and maintaining HDF5 VOL connectors.
- Each talk will address
  - Short description and references
  - How will HDF5 apps benefit?
  - What is the major success?
  - What were the main obstacles to overcome?
  - Lessons learned.

Agenda

Topic	Presenter
Introduction	Scot Breitenfeld
VOL framework, VOL toolkit, and Spack	Dana Robinson, Larry Knox
VOL Connectors and experiences	
• DAOS	Jerome Soumagne
• Asynchronous I/O	Houjun Tang
• Cache	Huihuo Zheng
• Log-based	Wei-keng Liao
Question and Answers	

Contacts and Useful links

- **Contacts**
  - Suren Byna (LBNL) SByna@lbl.gov
  - Scot Breitenfeld (The HDF Group) brtnfld@hdfgroup.org

HDF5 User Support:
HDF Helpdesk: help@hdfgroup.org
HDF Forum: https://forum.hdfgroup.org/

HDF5 GitHub: https://github.com/HDFGroup/hdf5

The HDF5 Virtual Object Layer (VOL) *and* VOL Connectors in spack

May 11, 2022

Dana Robinson
Larry Knox
The HDF Group

*** Important HDF5 Version Note ***

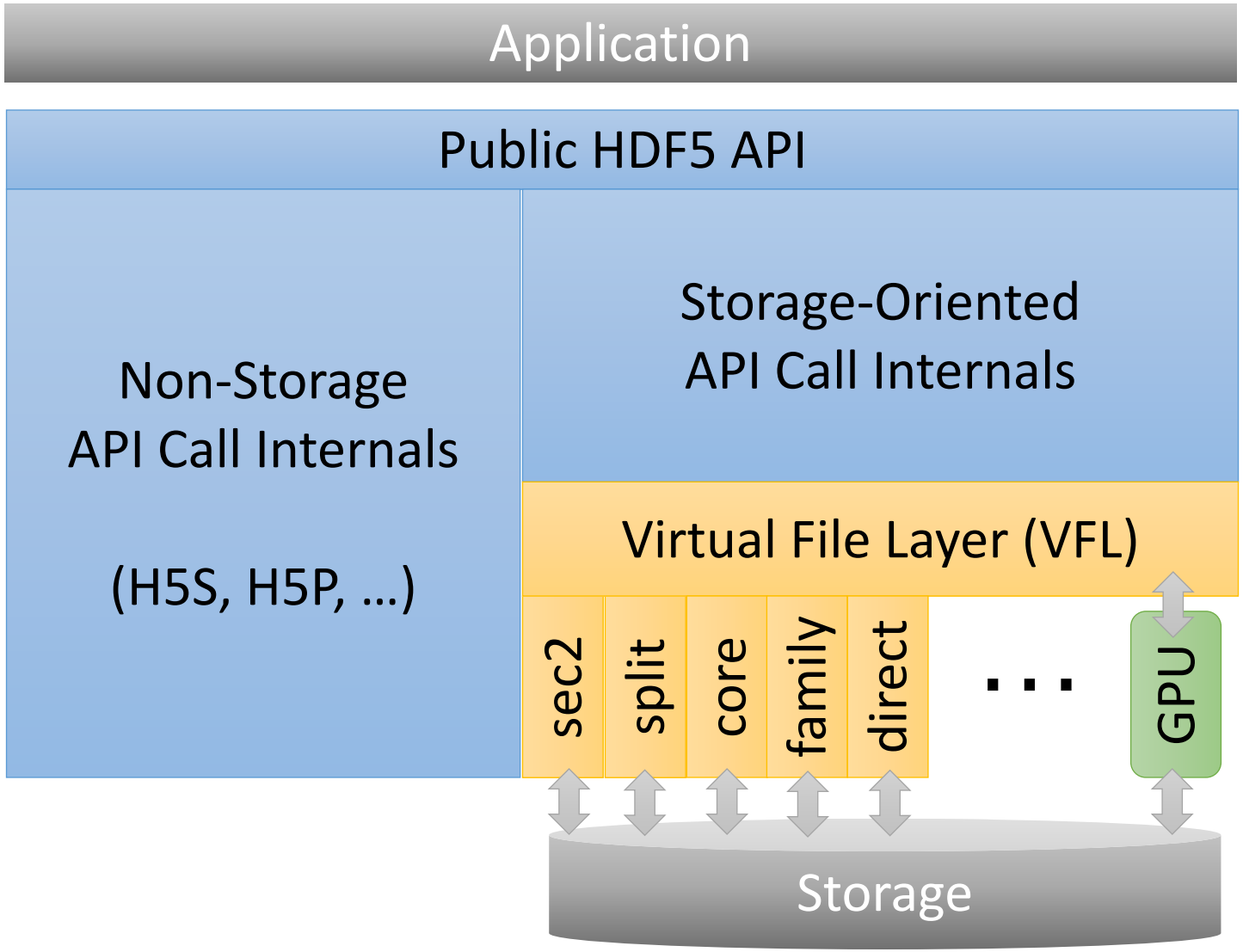
The HDF Group

- **ALL VOL CONNECTOR DEVELOPMENT SHOULD TARGET HDF5 1.13.x**
- Do **NOT** use HDF5 1.12.x
- There were important changes to the VOL interface in HDF5 1.13.0 that could not be moved to 1.12.x without breaking binary compatibility
- Note that HDF5 1.13.0 is an experimental branch
  - It is possible that the VOL interface could be changed in the 1.13.x versions that will be released before HDF5 1.14.0
  - Should be fairly stable, though

What is the Virtual Object Layer?

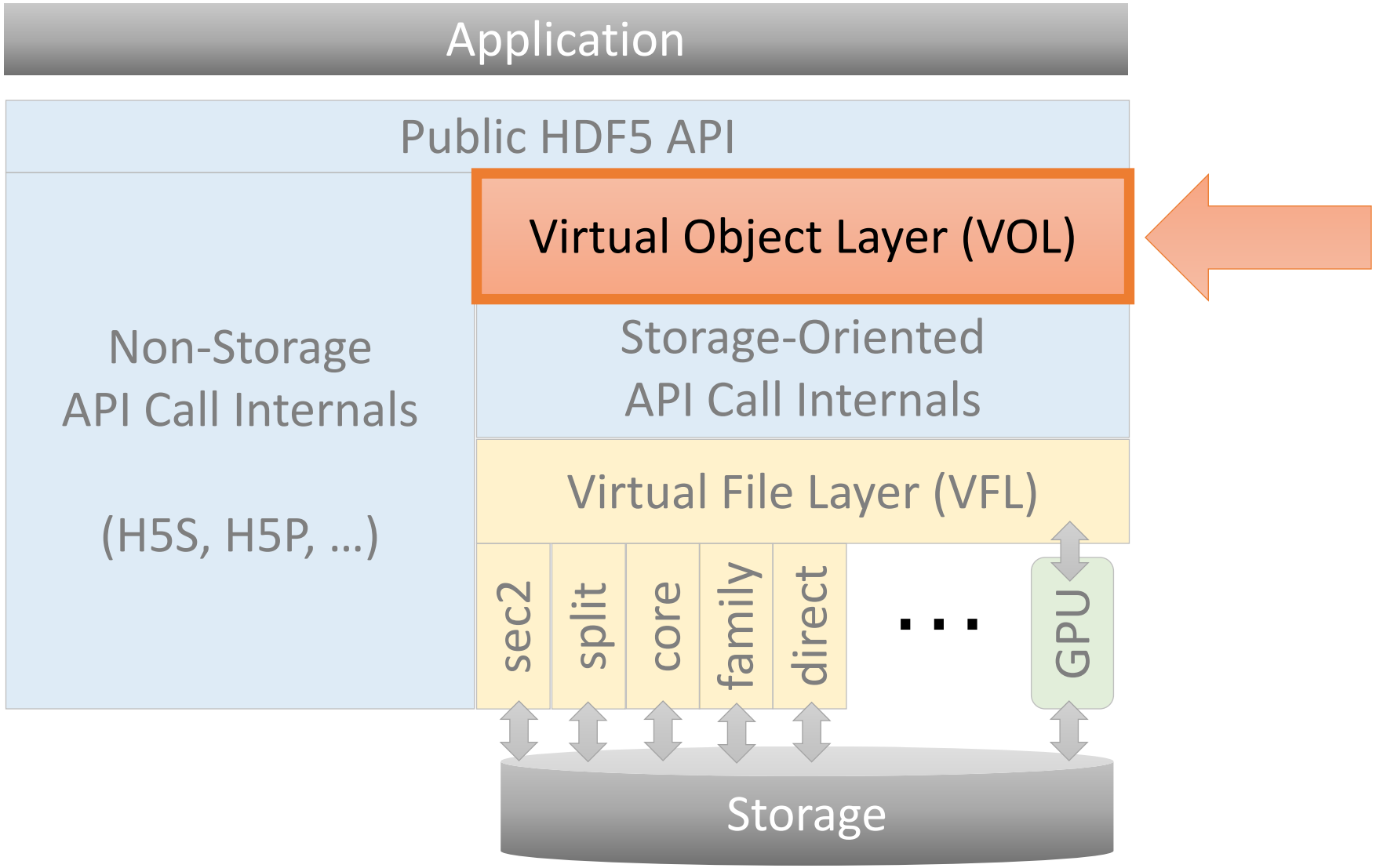
Original HDF5 Architecture (pre-1.12.0)

The HDF Group



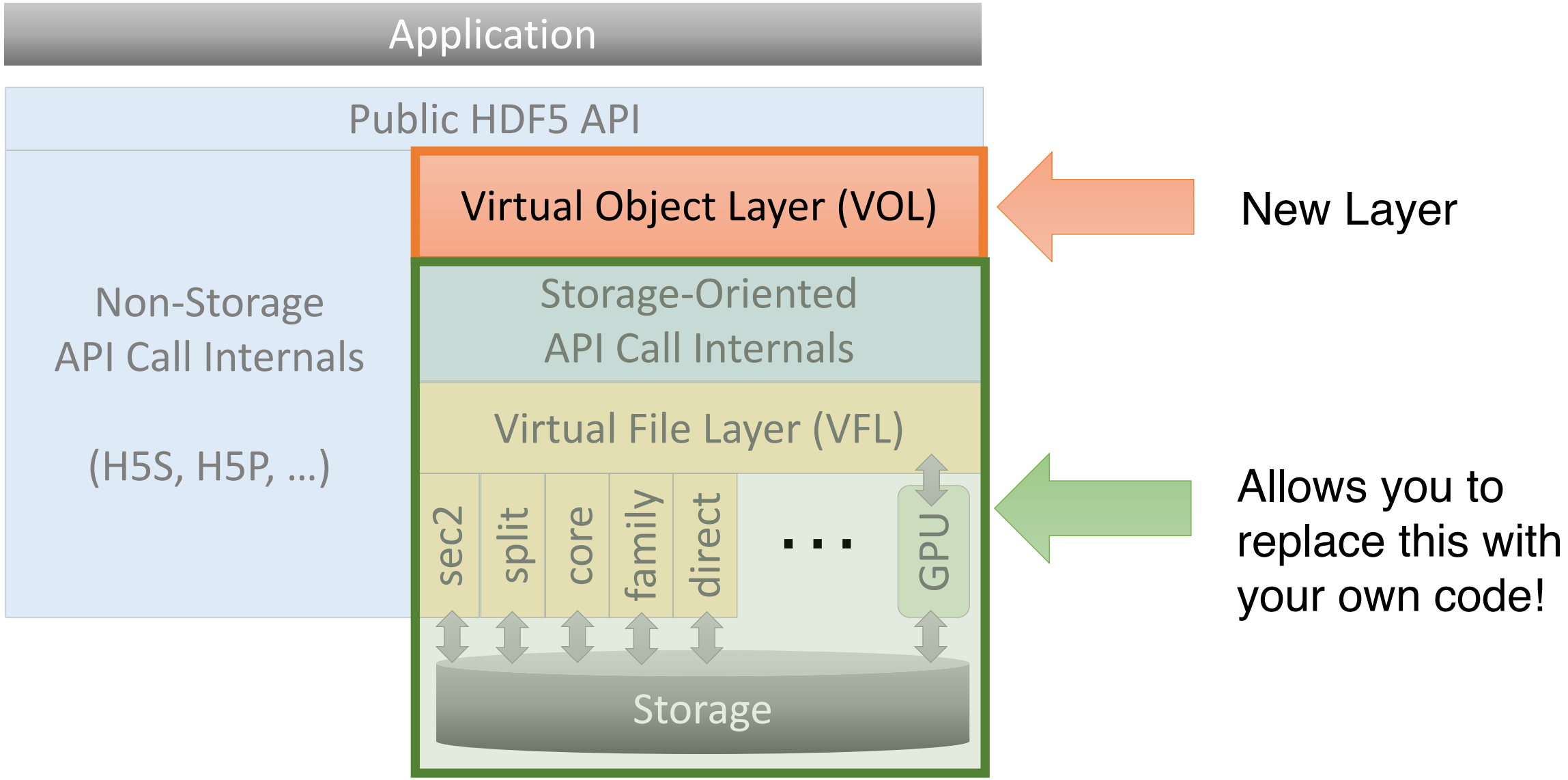
Current HDF5 Architecture (1.12.0+)

The HDF Group

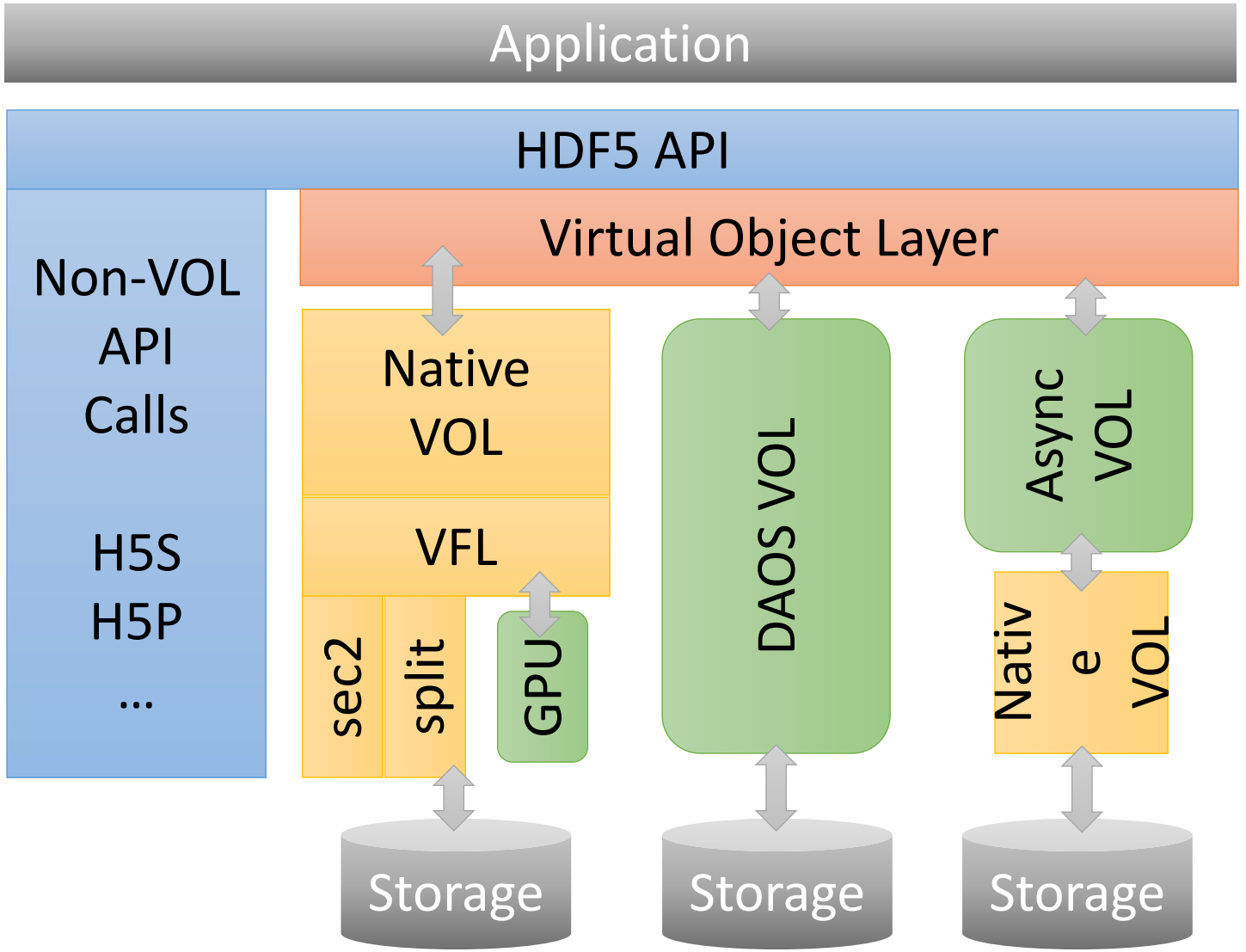


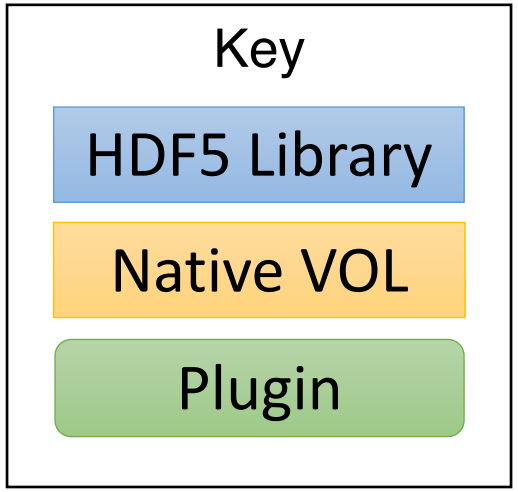
Current HDF5 Architecture (1.12.0+)

The HDF Group



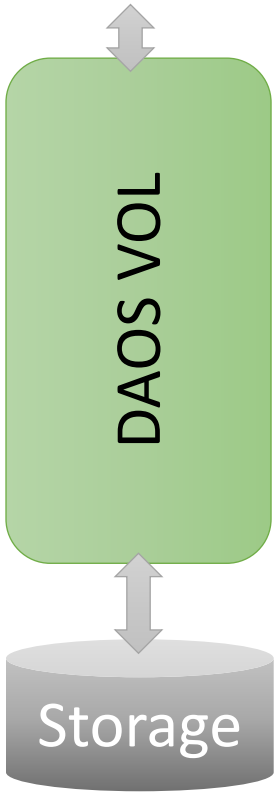
Current HDF5 Architecture (1.12.0+)





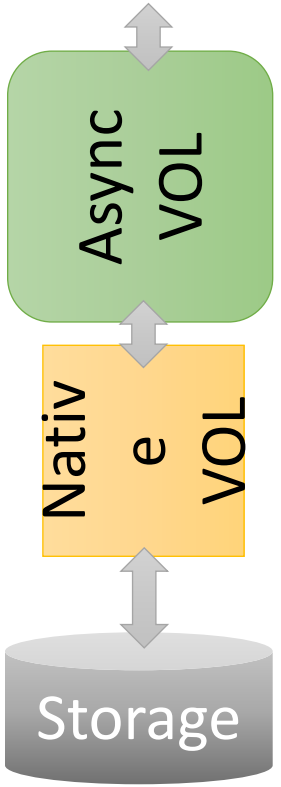
Two Kinds of VOL Connector

The HDF Group



Terminal

Maps HDF5 objects to arbitrary storage schemes



Pass-Through

Perform operations (e.g. caching, logging) before passing the data on to the next connector.

Terminal VOL Connectors

The HDF Group

```
H5Fcreate("foo.h5", flags, fcpl , fapl)
```

Public API

VOL Connector

"foo.h5"

flags

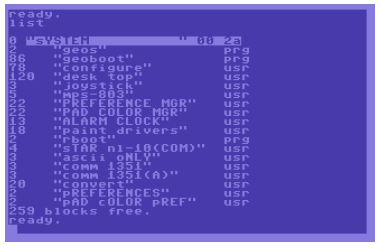
fcpl

fapl

Storage

mongoDB

PostgreSQL



Microsoft Azure
Blob Storage

neo4j

Resources

VOL Toolkit Repository

The HDF Group

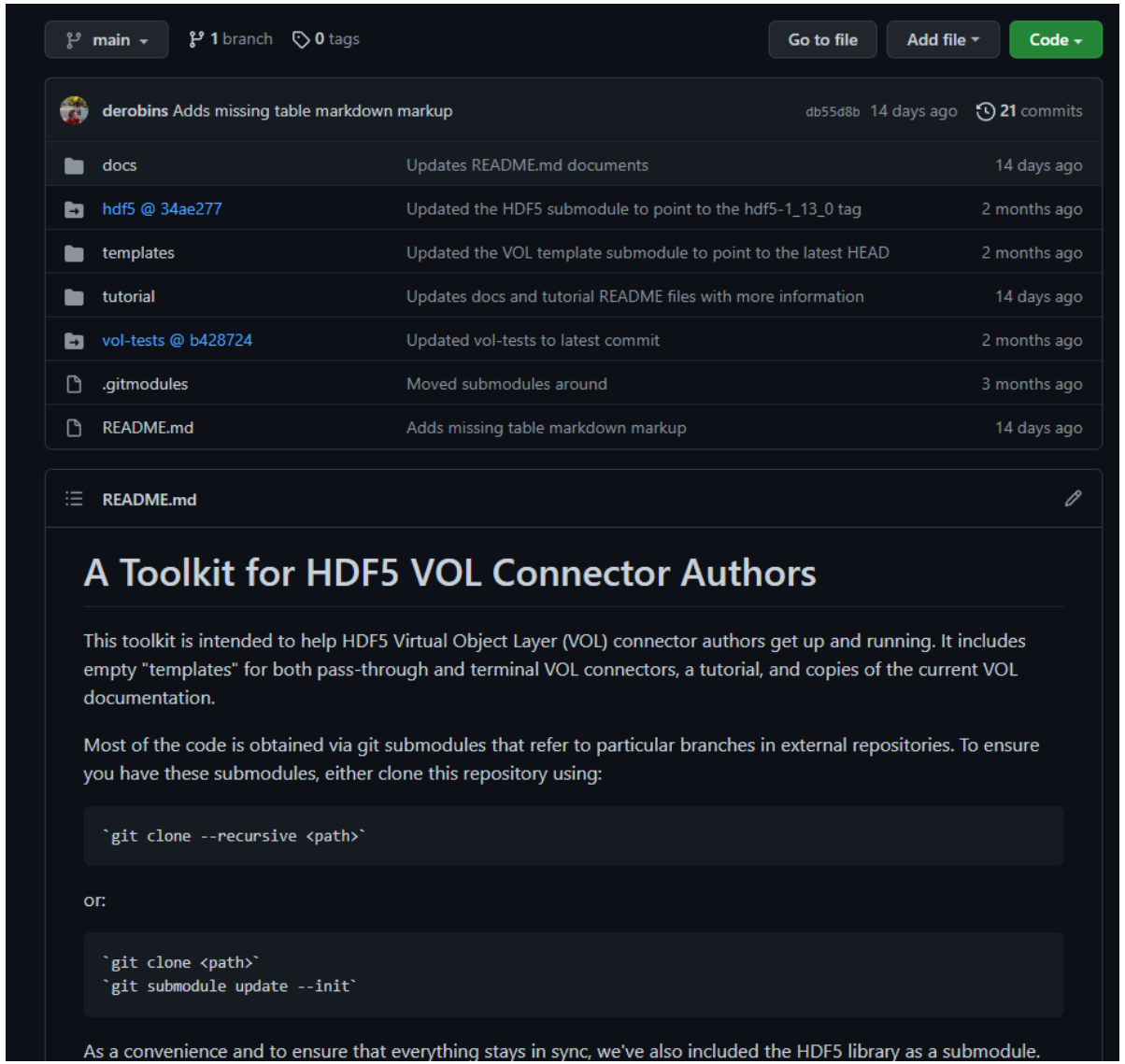
- Location: https://github.com/HDFGroup/vol-toolkit
- All your VOL construction needs in a single location
- Does not contain original content
- Designed to bring important content from other repositories together with consistent versioning
- Content is mainly included as git submodules, though the docs are currently copied in
- Tags will identify "HDF5 1.13.0", etc. versions of the toolkit
- Includes an appropriate version of HDF5

VOL Toolkit Repository

The HDF Group

The toolkit contains:

- VOL documentation
- Connector templates
  - Terminal
  - Passthrough
- Testing suite
- Tutorial
- HDF5



Documentation

The HDF Group

Two documents are included in the toolkit

User's Guide

- Covers basic VOL operations like registration, handling plugin paths, etc.

Connector Author's Guide

- Helpful instructions for constructing VOL connectors
- RM for "connector author" calls that aren't covered in the main HDF5 API docs

Both were copied from the hdf5doc repository (https://github.com/HDFGroup/hdf5doc/tree/master/RFCs/HDF5/VOL)

Templates

The HDF Group

Two template repositories are linked in the toolkit

vol-template (https://github.com/HDFGroup/vol-template)

- Template for building terminal VOL connectors
- Build files + stubs.
- Developed and supported by THG
- Officially a "template repository" on github so you can clone + rename

vol-external-passthrough (https://github.com/hpc-io/vol-external-passthrough)

- Template for constructing pass-through connectors
- Has no-op, pass-through stubs for all callbacks
- Developed and supported by NERSC

Production Connectors (NOT in Toolkit)

The HDF Group

When developing your own connector, it can be VERY helpful to see what others have done

Examples:

vol-daos (https://github.com/HDFGroup/vol-daos)
- Terminal VOL connector based on Intel's DAOS developed by THG
- Largely complete coverage of the HDF5 API
- Supports parallel HDF5 and async I/O

vol-async (https://github.com/hpc-io/vol-async)
vol-cache (https://github.com/hpc-io/vol-cache)
- Pass-through VOL connectors developed by NERSC
- Support parallel HDF5 (both) and async I/O (vol-async)

Find a full list here: https://portal.hdfgroup.org/display/support/Registered+VOL+Connectors

Testing Suite

The HDF Group

A subset of the HDF5 library tests has been collected in a separate repository

vol-tests (https://github.com/HDFGroup/vol-tests)

- Requires CMake
- Supports parallel connectors and async
- No Windows support
- Tests a lot of the HDF5 API
- Tests the HDF5 command-line tools
- Expect a lot of failed tests until you have significant HDF5 API coverage in your connector
- Instructions for use located in the repository's README

Tutorial and Associated VOL Connector

The HDF Group

Feb 2022 VOL tutorial

- Watch here: https://www.youtube.com/watch?v=7XEbm-__QuM
- "Hello, world!" of VOL creation
- Builds a simple connector from scratch using the template terminal VOL connector as a starting point
- Tutorial connector:

  vol-tutorial (https://github.com/HDFGroup/vol-tutorial.git)

Compliance Levels

VOL "Compliance Levels"

The HDF Group

We want to provide a systematic way to determine what features are supported by a VOL connector

- What's a "baseline" that every reasonable VOL connector should provide?
- Will probably set up a flag set to cover everything above that
- Need to determine which flags would be useful for VOL consumers?
  - By API? (e.g. "supports references via H5R")
  - By functionality? ("supports creation order")
  - Something else?
- I will send an email to the VOL developer mailing list (and reach out to others)
- Probably meet a few times over the summer so this gets into HDF5 1.14.0

Spack

Hdf5 & Hdf5 VOL Packages tested with Spack

The HDF Group

- VOL Packages:
  - hdf5-vol-external-passthrough
  - hdf5-vol-async
  - hdf5-vol-log
- HDF5 versions for VOL packages
  - develop-1.13
  - 1.13.1
- Other HDF5 versions tested
  - 1.12.2
  - 1.10.8

Build Status for 3 VOL packages with spack defaults – “spack install <vol package>”

- Cori Success – gcc@10.1.0
- Crusher Success – gcc@11.2.0
- Perlmutter Success – gcc@11.2.0
- Spock Success – gcc@11.2.0
- Summit Success – gcc@8.3.1
- Theta Success – gcc@10.1.0 with ^cmake@3.22.2

Testing Hdf5 & 3 VOL Packages in Spack with other compilers

The HDF Group

HDF5 and the vol packages have also been built with spack with these compilers where available, specified by adding %compiler@version to the "spack install <vol package>" command. : cce@10.0, cce@13.0.1, intel@19.1.0.3.166, intel@19.1.3.304, pgi@20.4, and rocmcc@8.33.

When errors occurred building one of the requisite packages, the package built with gcc was added to the install command. For example, when building perl with intel 19.1.3 failed, perl built with gcc 10.1.0 was added using this command: "spack install <package> %intel@19.1.3.304 ^perl@5.34.1%gcc@10.1.0 ^hdf5@develop-1.13%intel@19.1.3.304".

Additional test work planned

The HDF Group

- Run batch jobs with regression tests: “spack install --test=root <package>”
- Replace with verification tests: “spack test <package>”
- Test additional VOL packages
- Report VOL test results to https://cdash.hdfgroup.org/index.php?project=HDF5.

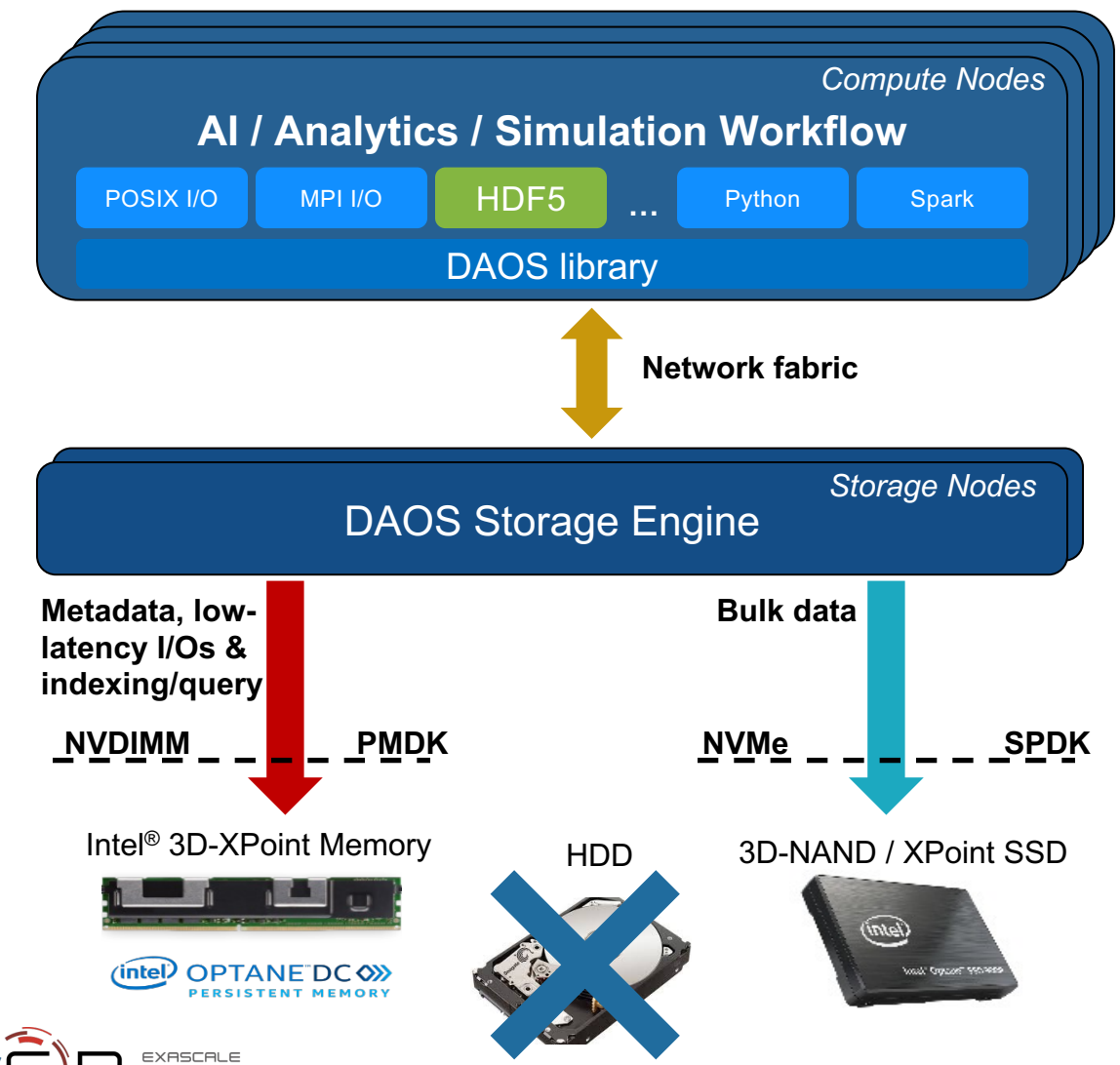
THANK YOU!

Questions & Comments?

DAOS VOL

Intel DAOS

Credit: Mohamad Chaarawi (Intel Corporation)



- DAOS library directly linked with the applications
- No need for dedicated cores
- Low memory/CPU footprint
- End-to-end OS bypass
- KV API, non-blocking, lockless, snapshot support

- Low-latency & high-message-rate communications
- Native support for RDMA & scalable collective operations
- Support for Infiniband, Slingshot, etc through OFI libfabric

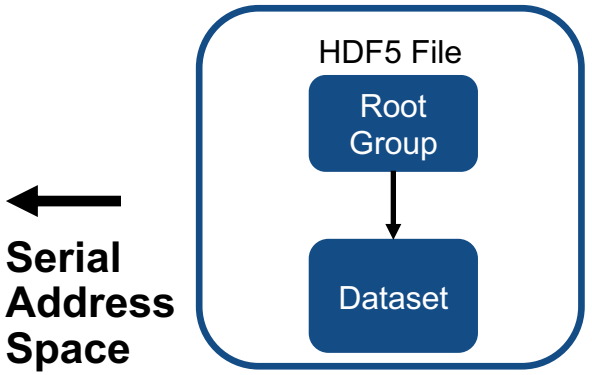
- Fine-grained I/O with media selection strategy
- Only application data on SSD to maximize throughput
- Small I/Os aggregated in pmem & migrated to SSD in large chunks
- Full user space model with no system calls on I/O path
- Built-in storage management infrastructure (control plane)
- NFSv4-like ACL

Delivers high-IOPs, high-bandwidth and low-latency storage with advanced features in a single tier

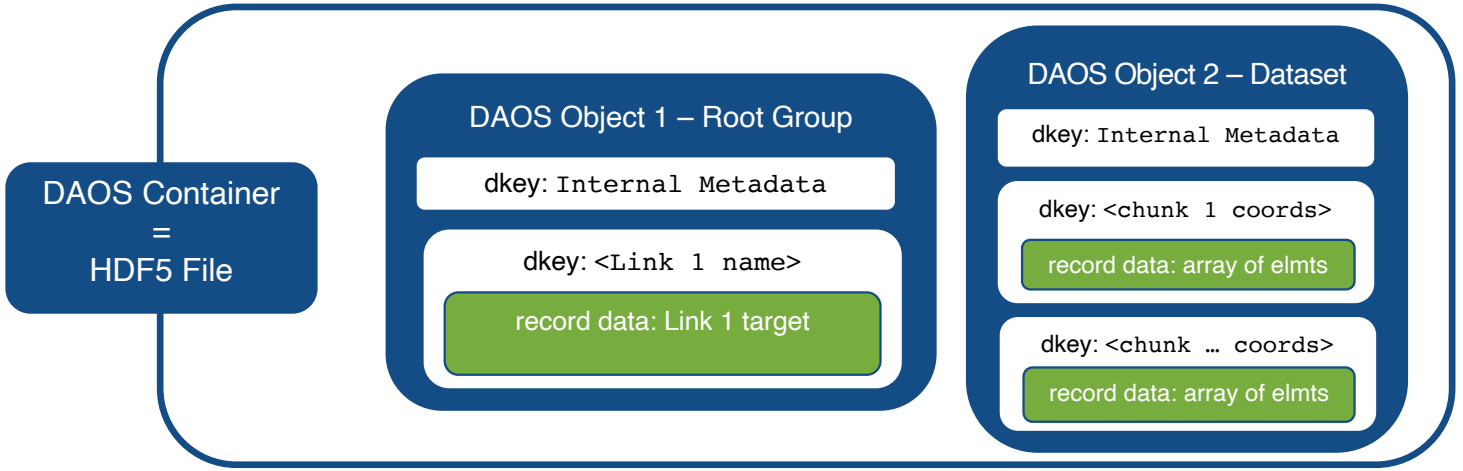
ECP EXASCALE COMPUTING PROJECT

From “Native” to DAOS Representation

- POSIX I/O was designed for **disk-based** storage
  - High-latency to write data at random offsets because of mechanical aspects
  - Current native HDF5 file format inherited POSIX I/O block-based model (serial)



- DAOS is designed for **object-based** storage

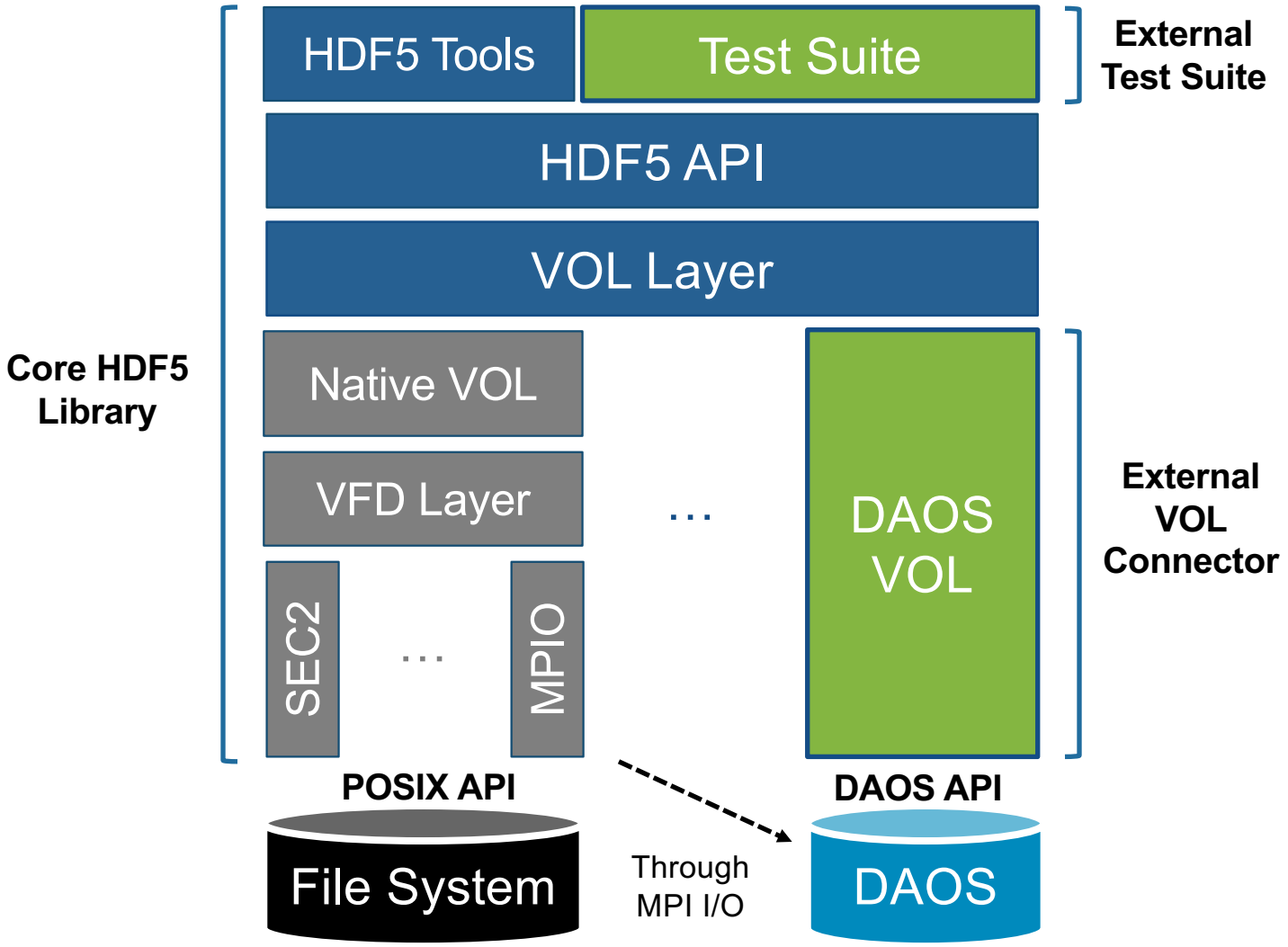


1-to-1 Mapping

- Parallel I/O and chunking first-class citizens
  - Implicit chunking
- Independent object creation

HDF5 VOL Architecture and DAOS VOL

- New Component
- Enhanced Component
- Native Component
- New HDF5 features:
  - Maps (enabled by K/V objects)
  - File deletion
  - Independent metadata
    - HDF5 objects can be created independently
  - Asynchronous I/O
- Tools support:
  - h5dump, h5ls, h5diff, h5repack, h5copy, etc



DAOS VOL Usage

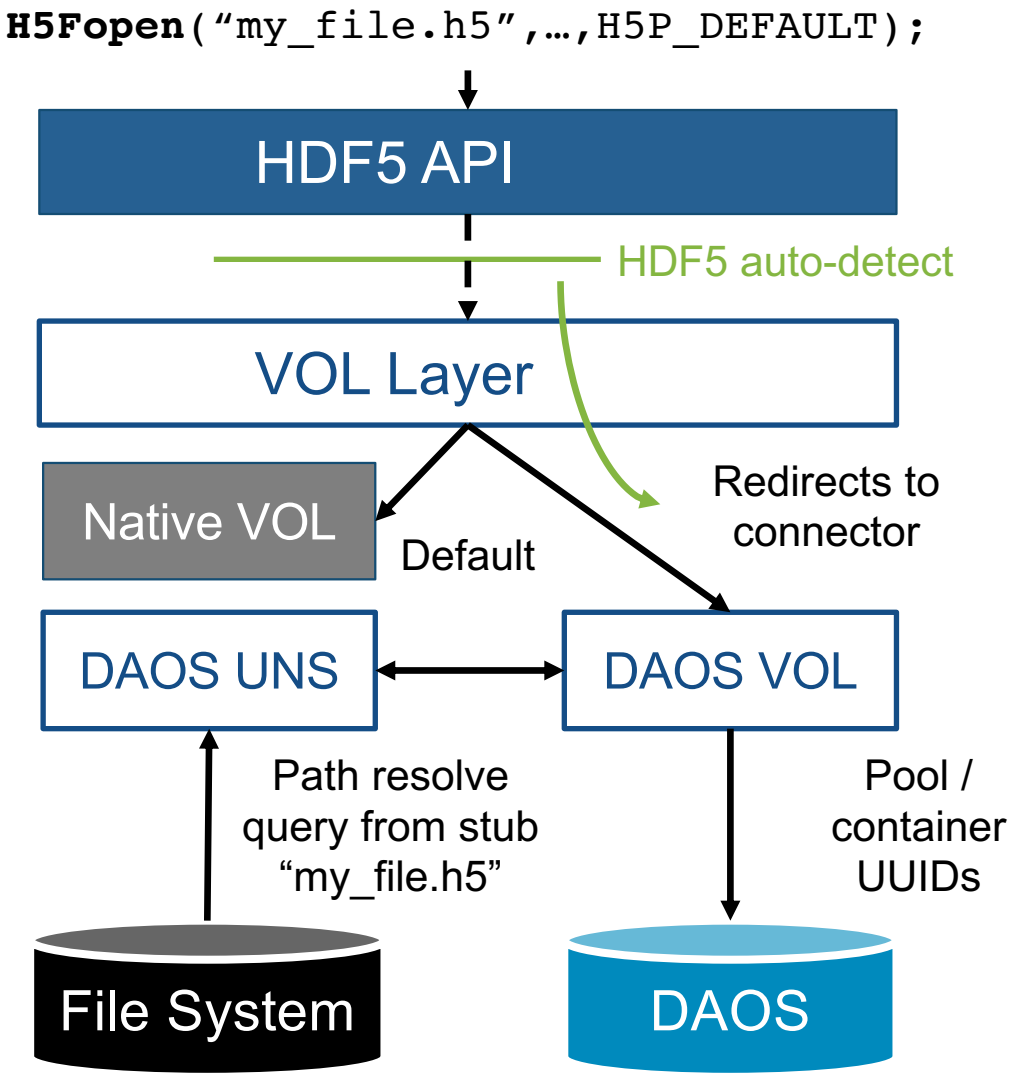
- Minimal or no code changes for application developer (if only looking for compatibility)
- Two ways to tell which connector to use
  - HDF5 file access property list (**recommended for new files or when manipulating multiple VOLs**)

```
herr_t H5Pset_fapl_daos(hid_t fapl_id,
const char *pool, const char *sys_name)
```

  - Environment variable

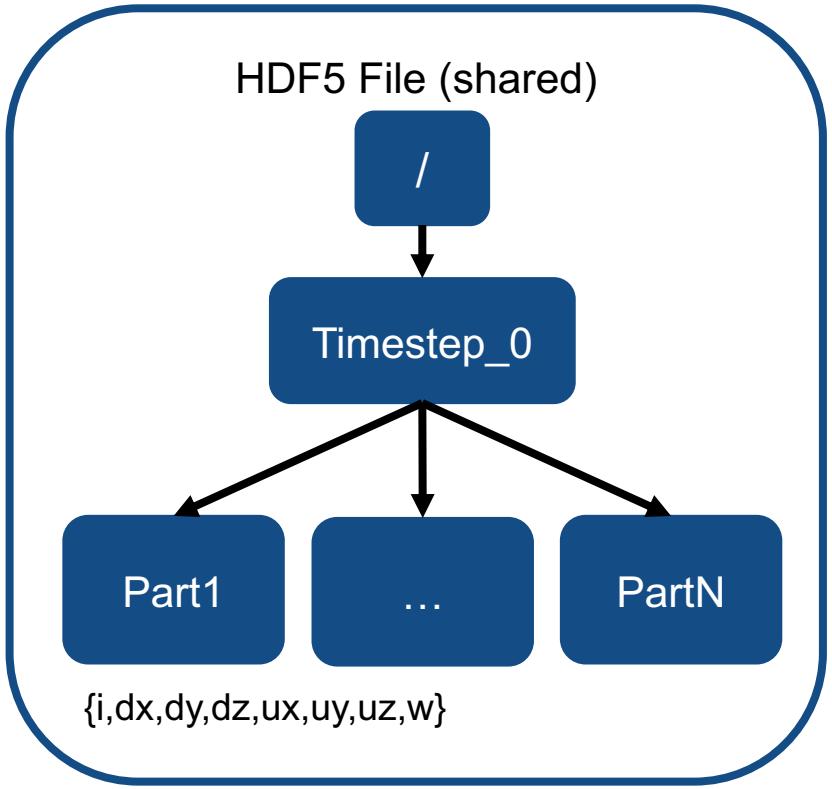
```
HDF5_VOL_CONNECTOR=daos
HDF5_PLUGIN_PATH=/path/to/connector/folder
```

- **Auto-detect and Unified Namespace** component facilitates opening of DAOS files with the DAOS connector (embedded DAOS metadata through extended attributes)

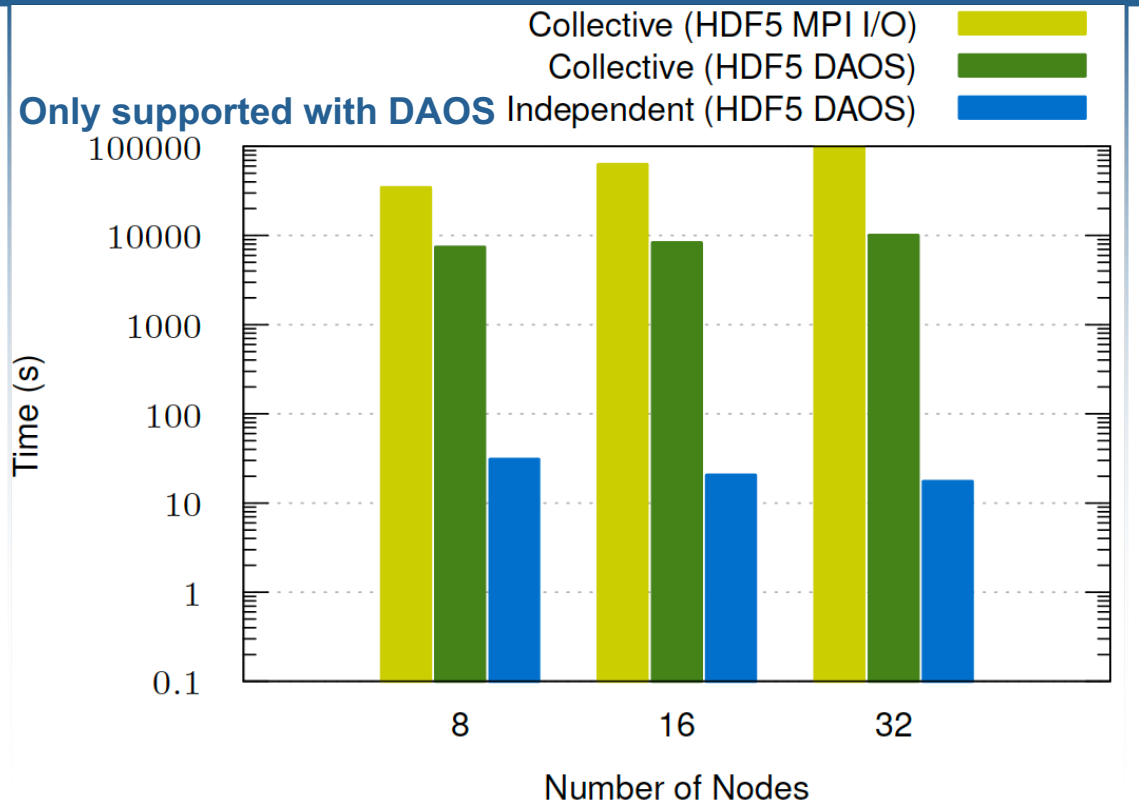


Evaluation – Example w/VPIC (metadata operations)

Re-defined VPIC file structure for electron particle (N particles)



VPIC I/O performance using collective and independent group creation



DAOS VOL – Difficulties and Lessons learned

- *Difficulties*
  - *The DAOS connector was the first connector to fully exercise the entire VOL framework*
  - *Writing a fully featured VOL connector requires a **LOT** of code to cover all the APIs*
    - *Supporting all properties (creation order, fill values)*
    - *Datatype conversion*
    - *Path resolution*
    - *Selections / copy code from chunking module*
  - *Public / private HDF5 APIs*
    - *Private APIs made public*
  - *Asynchronous I/O*
  - *External VOLs and VOL testing*

- *Lessons learned.*
  - *Creating a VOL connector is not an easy task (to cover all APIs)*
    - *Order development by HDF5 hierarchy / files and groups / datasets / etc*
    - *Improvements to create convenience wrappers*
      - *Selections and datatype conversion*
  - *There is still room to improve public / private APIs and their use by VOLs*
    - *Make clear distinction between user APIs and VOL developer APIs*
  - *Design for asynchronous I/O from the start*
  - *Plan and test for external VOL compatibility with specific HDF5 version*

Additional Information

- DAOS VOL Connector repository:
  - https://github.com/HDFGroup/vol-daos
  - **Latest release is v1.1.0**: https://github.com/HDFGroup/vol-daos/releases/tag/v1.1.0
- More results / details in IEEE TPDS paper
  - https://doi.org/10.1109/TPDS.2021.3097884

This material is based upon work supported by the U.S. Department of Energy and Argonne National Laboratory and its Leadership Computing Facility, including under Contract DE-AC02-06CH11357 and Award Number 8F-30005. This work was generated with financial support from the U.S. Government through said Contract and Award Number(s), and as such the U.S. Government retains a paid-up, nonexclusive, irrevocable, world-wide license to reproduce, prepare derivative works, distribute copies to the public, and display publicly, by or on behalf of the Government, this work in whole or in part, or otherwise use the work for Federal purposes.

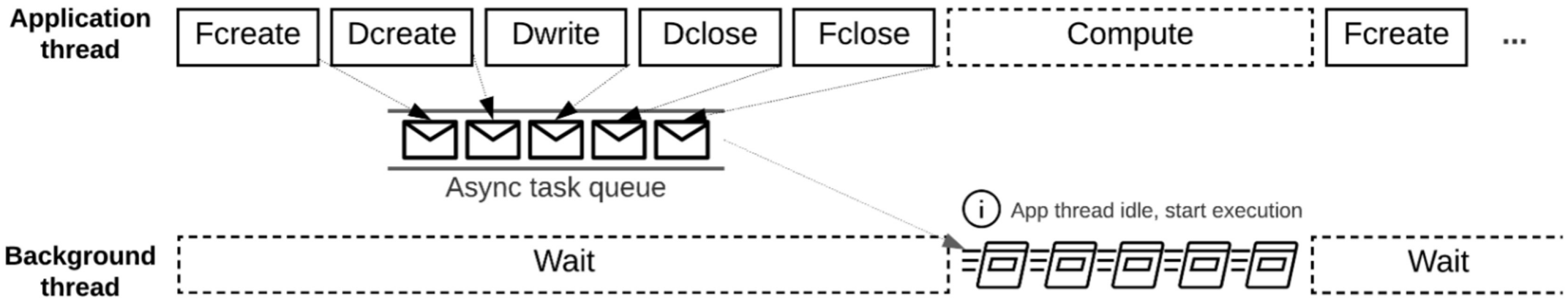
Developing Asynchronous I/O VOL Connector

Houjun Tang

Lawrence Berkeley National Laboratory, USA

Asynchronous I/O VOL Connector

- The async VOL uses **background threads** and supports HDF5 I/O operations, manages data dependencies **transparently and automatically**, provides an interface for **error information retrieval**.
- https://github.com/hpc-io/vol-pdc/



New Async and EventSet APIs

- Async version of HDF5 APIs
  - `H5Fcreate_async(fname, …, es_id);`
  - `H5Dwrite_async(dset, …, es_id);`
  - …
- Track and inspect multiple I/O operations with an ***EventSet ID***
  - `H5EScreate();`
  - `H5ESwait(es_id, timeout, &remaining, &op_failed);`
  - `H5ESget_err_info(es_id, ...);`
  - `H5ESclose(es_id);`

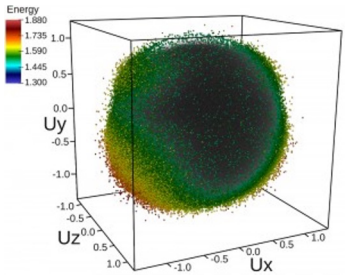
How to use Async VOL

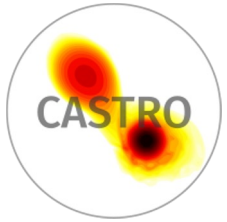
Detailed description in https://hdf5-vol-async.readthedocs.io

spack install hdf5-vol-async

- **Installation**
  - Compile HDF5 (github develop branch or released version 1.13+), with **thread-safety** support
  - Compile Argobots threading library
  - Compile Async VOL connector
    - "-DENABLE_WRITE_MEMCPY" flag to have async vol copy write buffer
- **Set environment variables**
  - `export LD_LIBRARY_PATH=$VOL_DIR/lib:$H5_DIR/lib:$ABT_DIR/lib:$LD_LIBRARY_PATH`
  - `export HDF5_PLUGIN_PATH="$VOL_DIR/lib"`
  - `export HDF5_VOL_CONNECTOR="async under_vol=0;under_info={}"`
  - `(optional) export HDF5_ASYNC_EXE_FCLOSE=1`
  - `(optional) export HDF5_ASYNC_MAX_MEM_MB=67108864`
- **Run the application (using the async and EventSet APIs)**
  - MPI must be initialized with **`MPI_THREAD_MULTIPLE`**

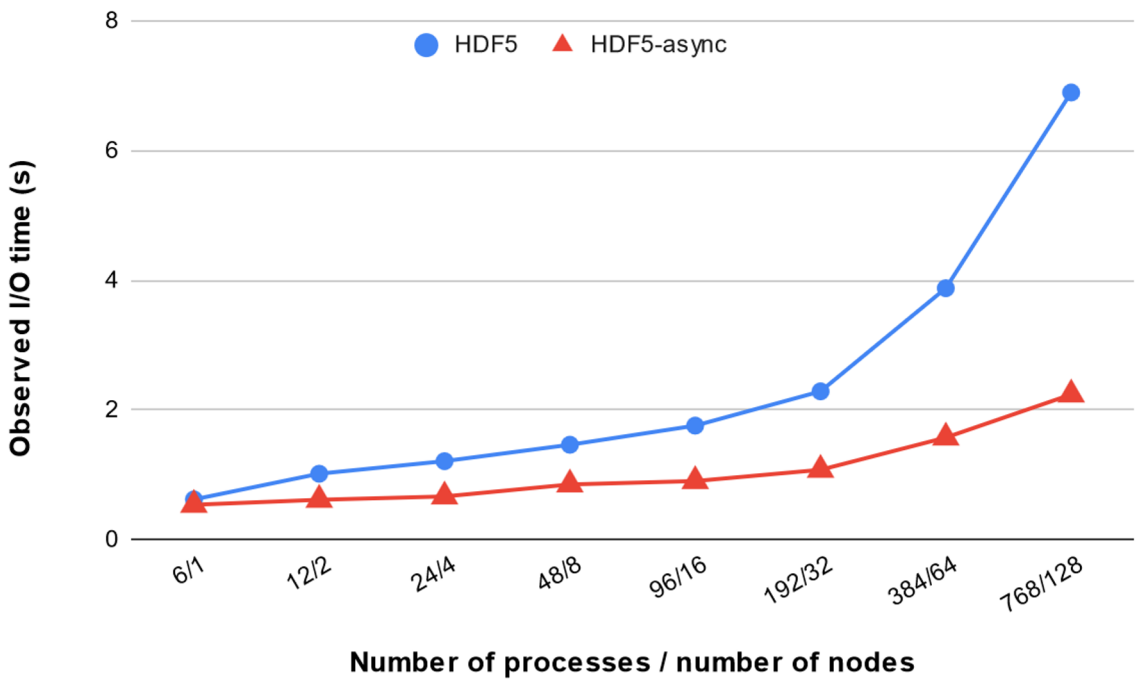
Evaluation Overview



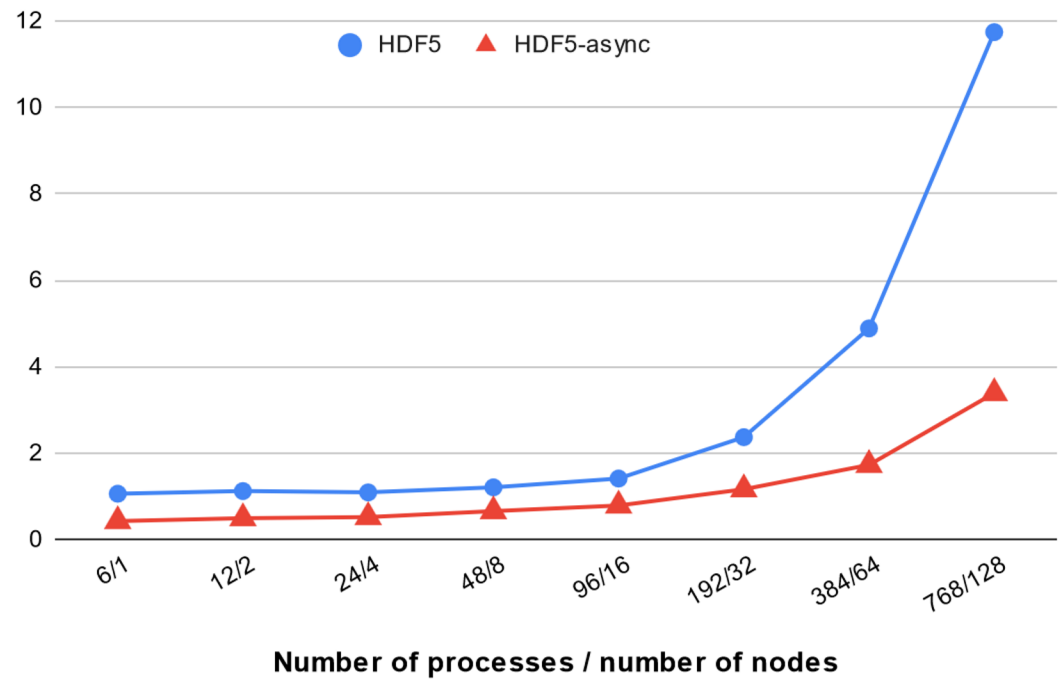


Case	Information	I/O Pattern
VPIC-IO	I/O kernel from VPIC, a plasma physics code that simulates kinetic plasma particles.	**Write**, single file for all steps, 8 variables, 256 MB per process per timestep.
BDCATS-IO	I/O kernel from BDCATS, a parallel clustering algorithm code that analyze VPIC data.	**Read**, single file, 8 variables, 256 MB per process per timestep.
AMReX/Nyx	I/O workload from Nyx, an adaptive mesh cosmological simulation code that solves equations of compressible hydrodynamics flow.	**Write**, one file for each timestep, 6 variables, *single* refinement level, with simulation metadata, 385 GB per timestep
AMReX/Castro	I/O workload from Castro, an adaptive mesh compressible radiation / MHD /hydrodynamics code for astrophysical flows.	**Write**, one file for each timestep, 6 variables, *3* refinement levels, with simulation metadata, 559 GB per timestep

Speedup with VPIC-IO and BDCATS-IO on Summit

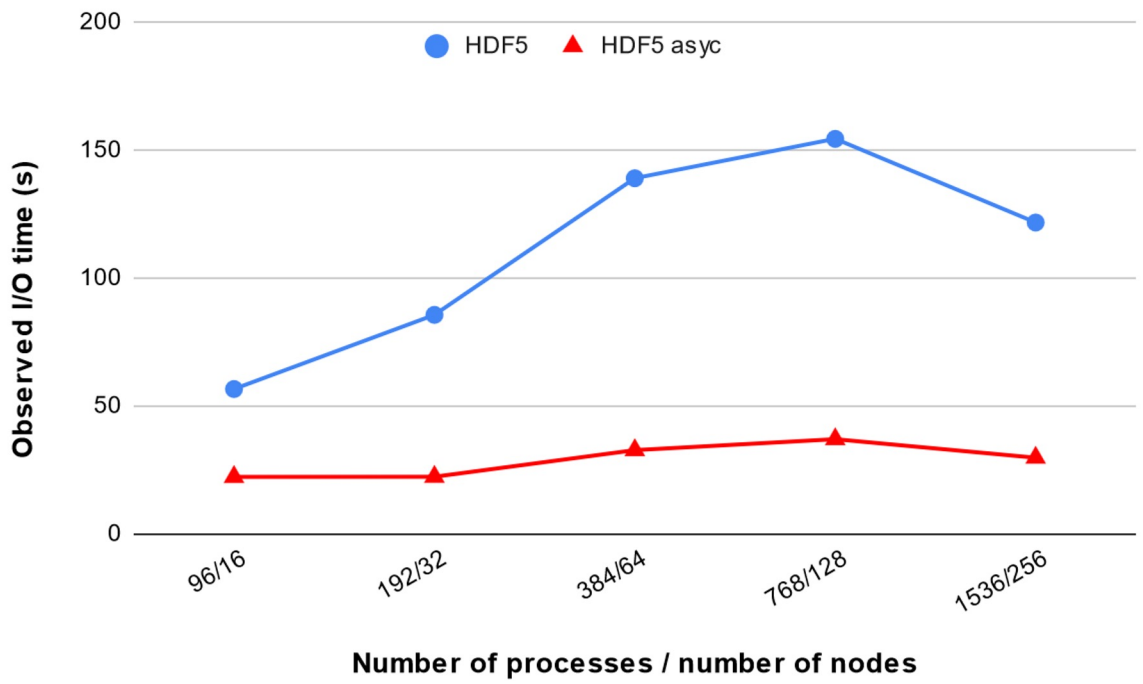


VPIC-IO, writes 256MB per process, 5 steps, emulated compute time.

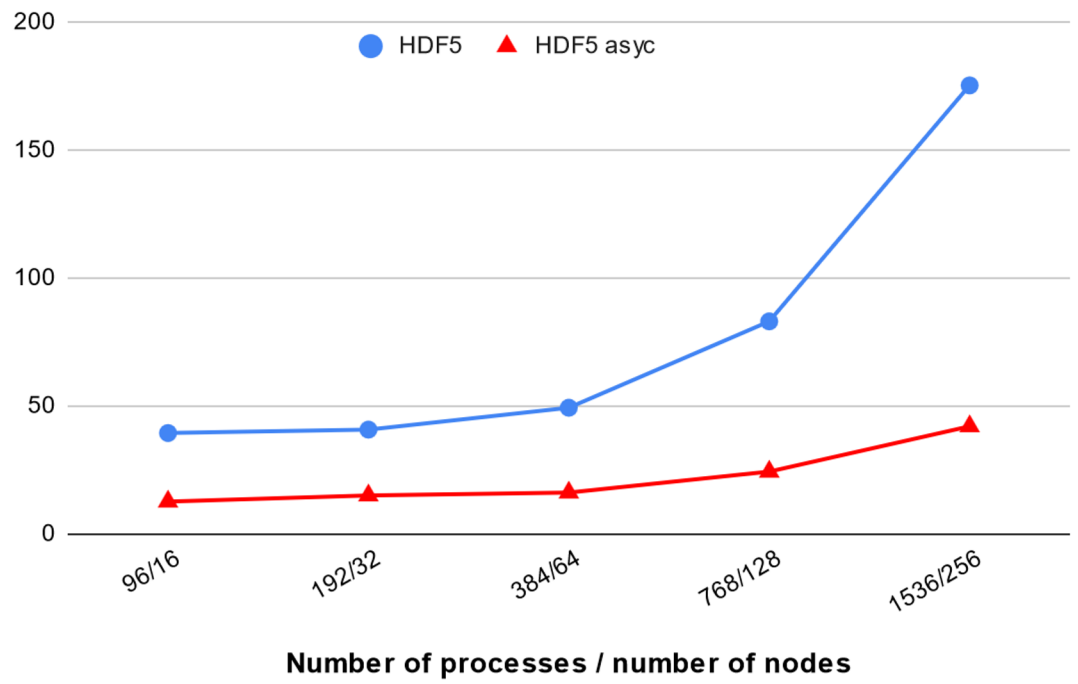


BDCATS-IO, reads 256MB per process, 5 steps, emulated compute time.

Speedup with AMReX Applications on Summit



NyX workload, single refinement level,
writes 385GB x 5 steps, emulated compute time.



Castro workload, 3 refinement levels,
writes 559GB x 5 steps, emulated compute time.

Obstacles During Implementation

- Many HDF5 APIs for implementation and testing.
- HDF5 (thread-safe) global mutex.
- New H5*_async and H5ES* APIs.
- Future ID.
- Error handling and reporting.

Best Practice & Lessons Learned

- Async is effective when I/O time is a significant portion of the total application execution time, and there is enough compute time to overlap with.
- Some operations cannot be done asynchronously, avoid if possible.
  - E.g. The “future ID” obtained from `H5Dget_space` needs to be realized (w/ disk I/O) when the ID is used (e.g. hyperslab selection).
- `MPI_THREAD_MULTIPLE` has overhead.
- Background thread interference.
  - Minimal interference for GPU-accelerated applications.
  - OpenMP applications should leave 1 core/thread for the async background thread.
- Memory allocation needs to be handled properly.
  - Peak memory usage could be higher than sync mode, due to double buffering.
  - Async VOL will switch to sync mode when not enough system memory is available.

Thank you!

Questions?

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.

HDF5 Cache VOL: efficient parallel I/O through caching data on node local storage

Huihuo Zheng

Argonne National Laboratory

huihuo.zheng@anl.gov

May 11th, 2022

H. Zheng et al, HDF5 Cache VOL: Efficient and Scalable Parallel I/O through Caching Data on Node-local Storage, CCGrid'22

https://github.com/hpc-io/vol-cache.git

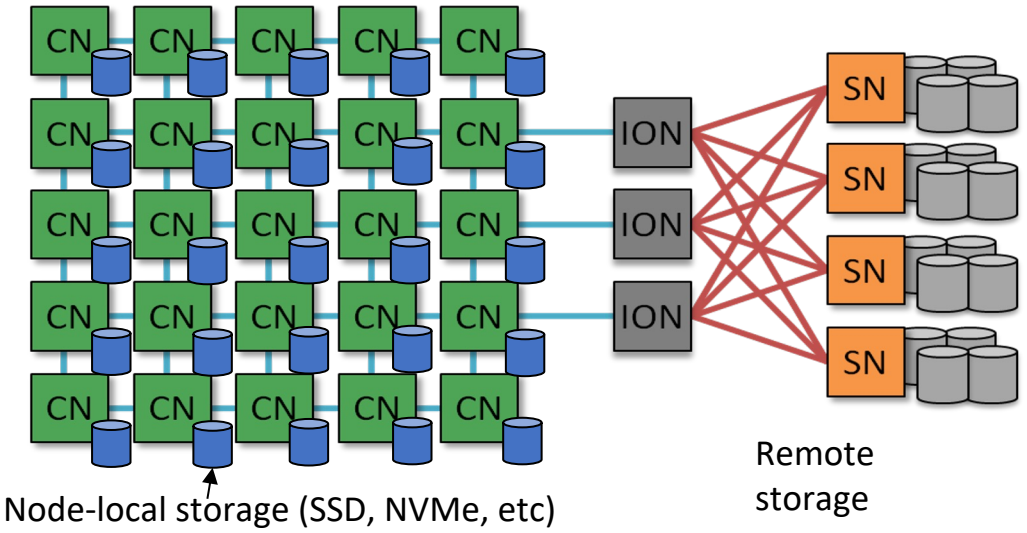
BERKELEY LAB Lawrence Berkeley National Laboratory

The HDF Group

Argonne NATIONAL LABORATORY

ECP EXASCALE COMPUTING PROJECT

Transparently integrating node-local storage into parallel I/O



ThetaGPU (DGX-3) @ ALCF: NVMe (15.4 TB / node)
Summit @ OLCF: GPFS + NVMe (1.6 TB / node)
Perlmutter @ NERSC: Lustre + SSD (960 GB/node)
Frontier @ OLCF: Lustre + NVMe (37PB total)

Node-local storage

- No network / resource contention; larger aggregate bandwidth compared to the parallel file systems

Challenges of using node-local storage

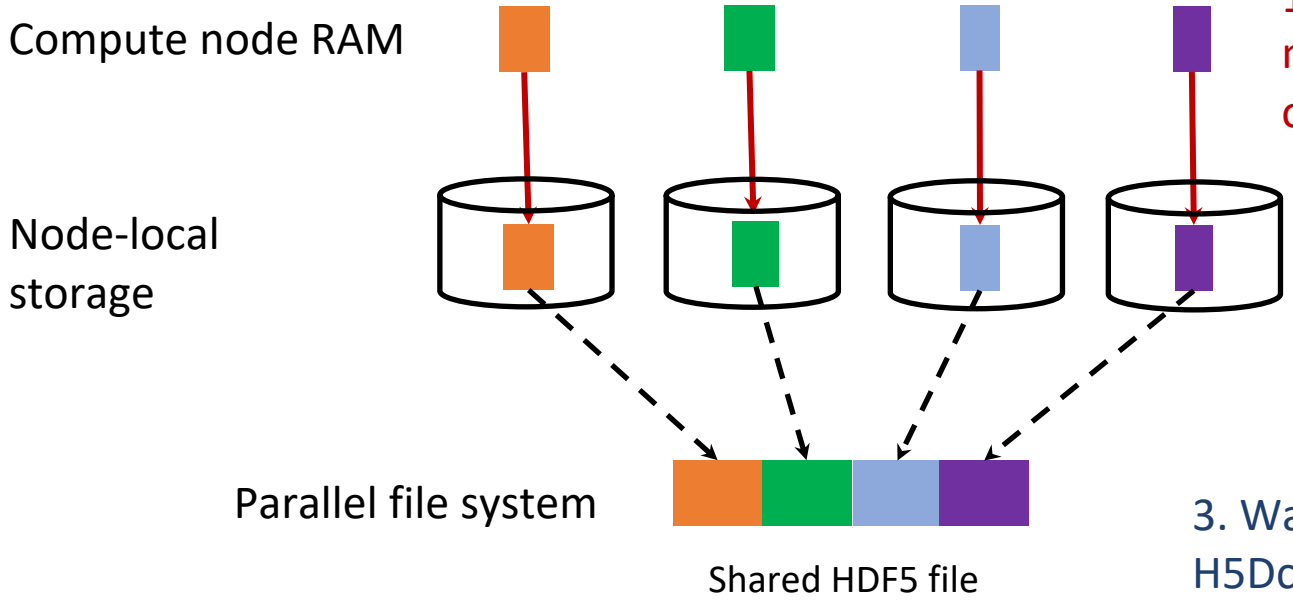
- Distributed storage -> w/o global namespace
- Accessible only during job running

Cache VOL

- Caching / staging data on node-local storage
- Asynchronous data movement to hide overhead
- All complexity hidden in the library
- Easy to integrate to existing HPC applications with minimal code change.

https://github.com/hpc-io/vol-cache.git

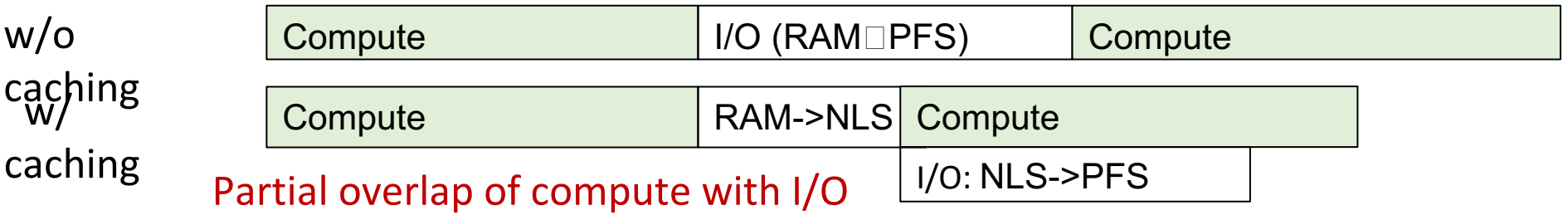
Parallel Write (H5Dwrite)



1. Data is synchronously copied from the memory buffer to memory mapped files on the node-local storage using POSIX I/O.

2. Move data from memory mapped file to the parallel file system asynchronously by calling the dataset write function from *Async VOL* stacked below the Cache VOL

3. Wait for all the tasks to finish in H5Dclose() / H5Fclose()



Partial overlap of compute with I/O

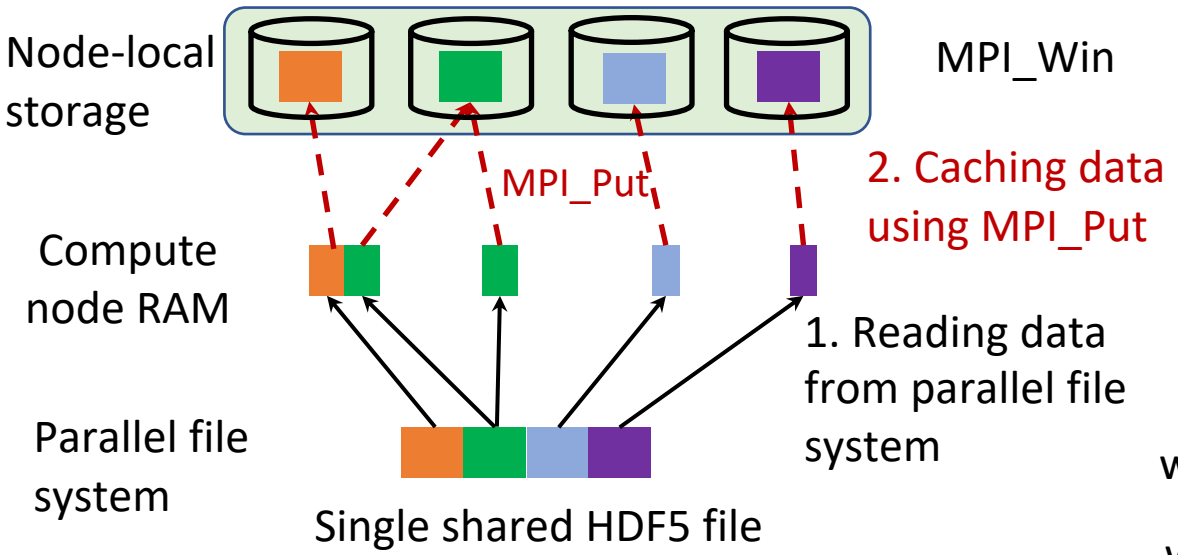
Details are hidden from the application developers.

Parallel Read (H5Dread)

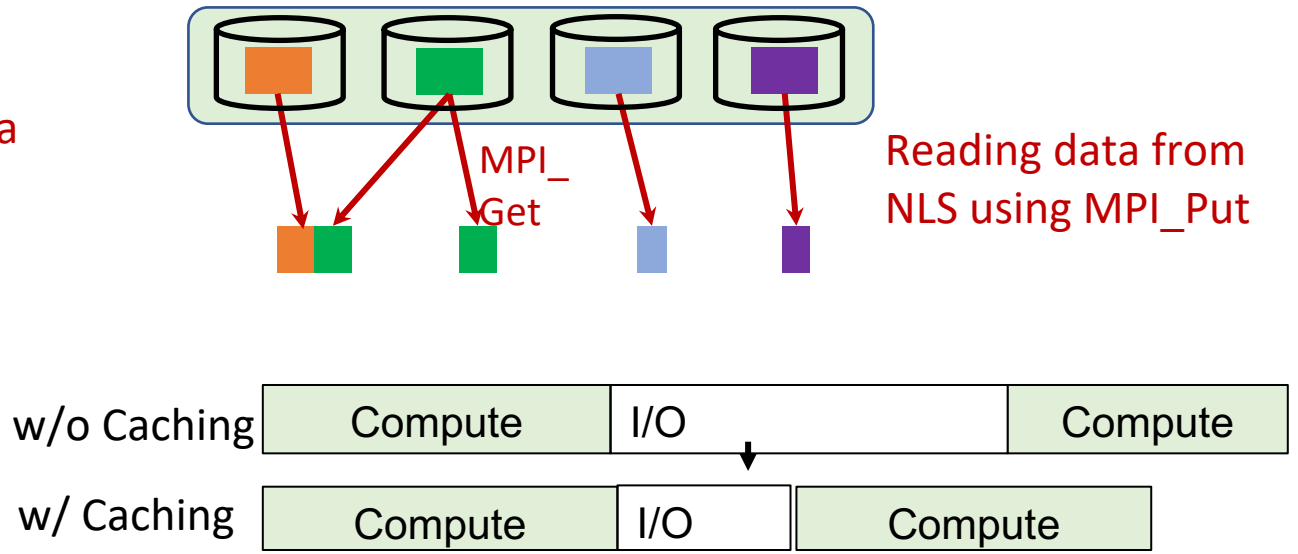
One-sided communication for accessing remote node storage.

- Each process exposes a part of its memory to other processes (MPI Window)
- Other processes can directly read from or write to this memory, without requiring that the remote process synchronize (MPI_Put, MPI_Get)

Create memory mapped files and attached them to a MPI_Win for one-sided remote access



First time reading the data



Reading the data directly from node-local storage

https://github.com/hpc-io/vol-cache.git

How to use Cache VOL

1) Setting VOL connectors

```
export HDF5_PLUGIN_PATH=$HDF5_VOL_DIR/lib
export HDF5_VOL_CONNECTOR="cache_ext
config=SSD.cfg;under_vol=518;under_info={under_vol=0;under_info={}}"
export LD_LIBRARY_PATH=$HDF5_PLUGIN_PATH:$LD_LIBRARY_PATH
```

```
#contents of SSD.cfg
HDF5_CACHE_STORAGE_SIZE
137438953472
HDF5_CACHE_STORAGE_TYPE          SSD
HDF5_CACHE_STORAGE_PATH          /local/scratch/
HDF5_CACHE_STORAGE_SCOPE         LOCAL
HDF5_CACHE_WRITE_BUFFER_SIZE     102457690
HDF5_CACHE_REPLACEMENT_POLICY    LRU
```

2) Enabling caching VOL

Opt. 1 Through global environment variables (**HDF5_CACHE_RD / HDF5_CACHE_WR [yes|no]**)

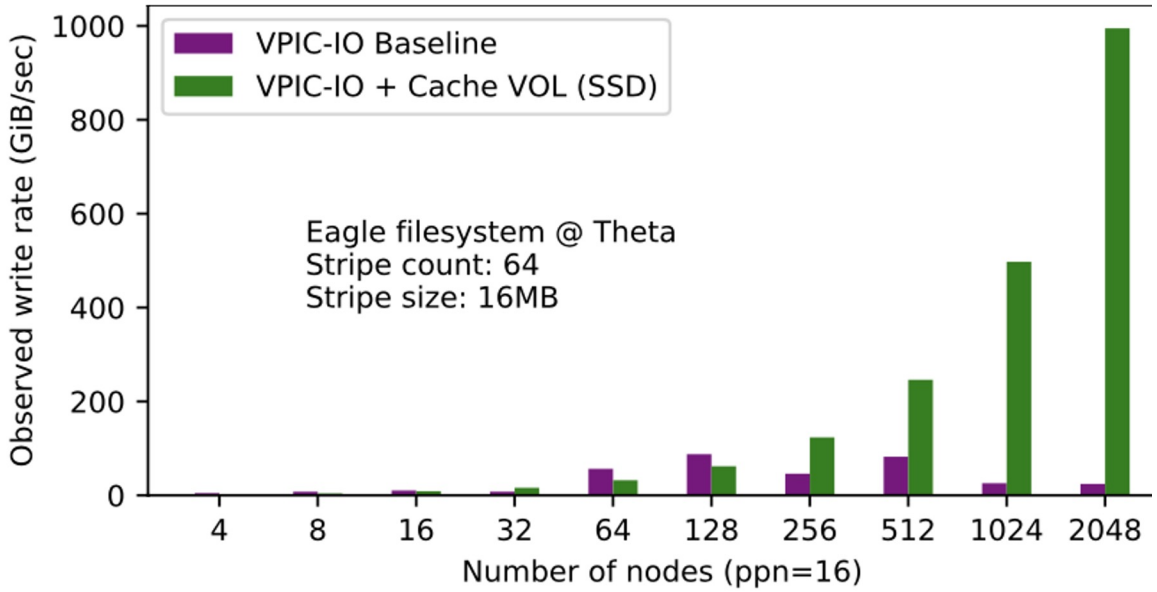
Opt. 2 Through setting file access property: **`H5Pset_fapl_plist('HDF5_CACHE_RD', true)`**

3) Initializing MPI with MPI_Init_thread(…, MPI_THREAD_MULTIPLE…)

4) In some cases, rearranging the function calls to allow the overlap of computation with data migration (see examples later)

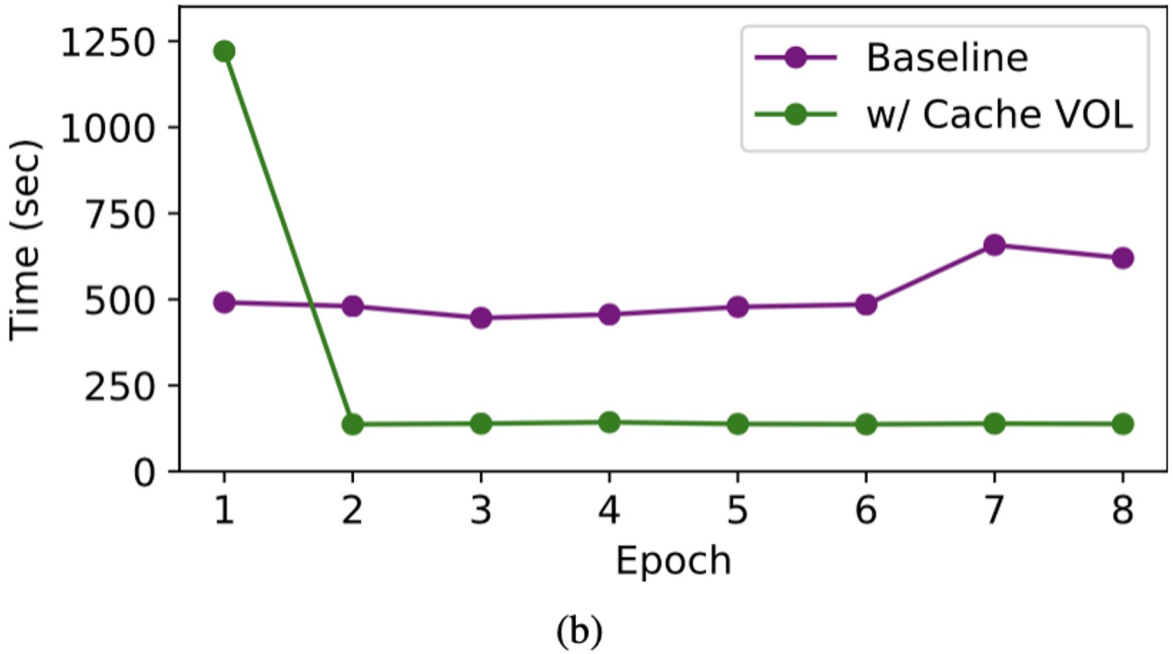
How do HDF5 apps benefit

Hiding the check-pointing I/O overhead behind the computation to improve observed I/O performance



Minimal code changes is needed

Staging data to the node-local storage for deep learning applications to improve the training throughput (work with h5py)



Success and obstacles

Success

- Improved scaling efficiency and performance of write I/O (VPIC-IO)
- Improved the training throughput for deep learning applications

Main obstacles to overcome

- Thread mutex and HDF5 library locking
- Async tasks blocking issues
  - Internal blocking of data migration due to thread mutex
  - Blocking by H5Dclose and H5Fclose
- Async tasks dependence issue
- Integrating to h5py
- Combining write and read

Lessons learned

- External passthrough connector is a very good template to start:

  https://github.com/hpc-io/vol-external-passthrough.git

- Working closely with the THG team and for new functionalities development, bug fixes.
- Taking advantage of existing works without reinventing the wheels. (Cache VOL uses Async VOL for data migration). I also learned a lot from Async VOL.

Acknowledgment

- This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC02-05CH11231 (Project: Exascale Computing Project [ECP] - ExaHDF5 project).
- 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 used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

Log-based Data Layout VOL

Presenters:

Wei-keng Liao, Kai-yuan Hou

DataLib Team members:

Kai-yuan Hou, Alok, Choudhary, Wei-keng Liao, Northwestern University

Rob Latham, Rob Ross, Argonne National Laboratory

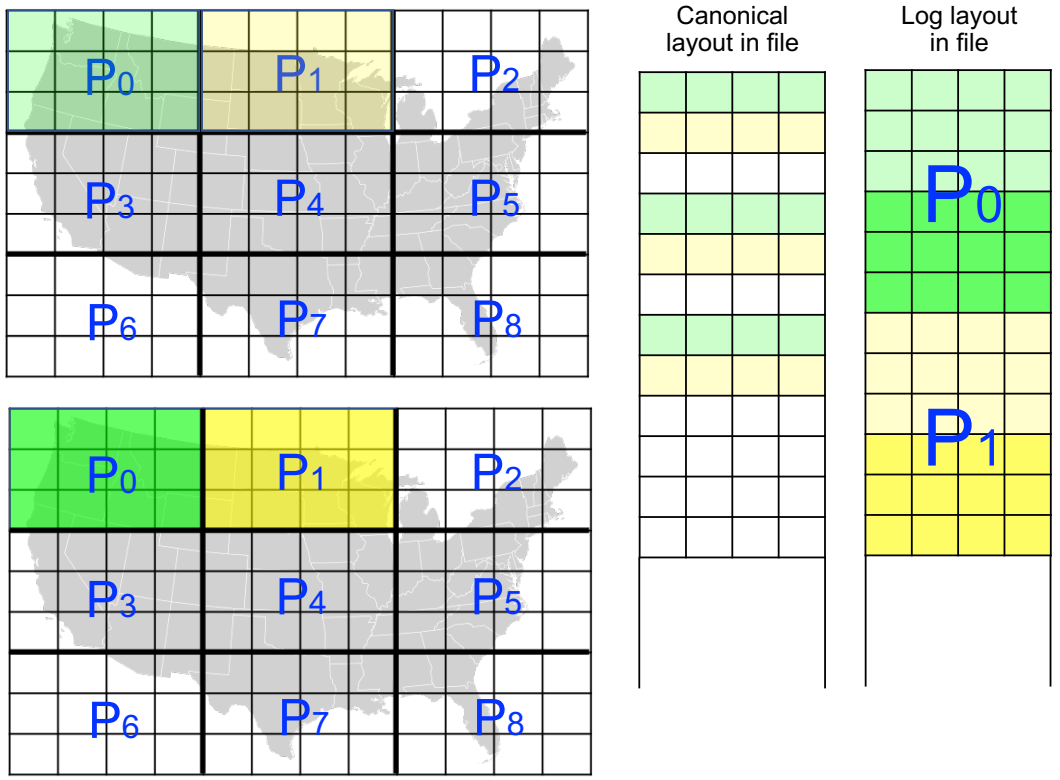
Galen Shipman, Los Alamos National Laboratory

BOF: Sharing Experience on HDF5 VOL Connectors Development and Maintenance

ECP annual meeting, May 11, 2022

Log File Layout Based VOL

- To store write data contiguously in the file, like time logs
  - Multi-dimensional arrays are flattened into 1D dataset objects
  - Write data is appended one after another in files
  - Keeps files conforming with HDF5 format
  - Makes use of native VOL to manage HDF objects
- VOL connector identifier: 514
- https://github.com/DataLib-ECP/vol-log-based



Benefit to Application Users

- High performance for write operations
  - Avoid expensive inter-process communication to rearrange data in canonical data layout
  - Aggregation of small requests into large ones
- Enabled through VOL environment variables
  - No change to the applications
- Same HDF5 utility programs
    - E.g., h5ls, h5dump to view the metadata and data

Case study – E3SM

- Irregular noncontiguous I/O pattern
  - A separate E3SM I/O benchmark
  - https://github.com/Parallel-NetCDF/E3SM-IO
  - Production run cases
    - F case: 21.3 GiB, 414 variables
    - G case: 80 GiB, 52 variables
    - I case: 86.1 GiB, 560 variables
- Experiment settings
  - Cori KNL nodes
  - PnetCDF nonblocking I/O is used
  - HDF5 multi-dataset APIs coming soon

I/O method	F case	G case	I case
HDF5 – native VOL (canonical layout)	can't finish	can't finish	can't finish
PnetCDF (canonical layout)	15.60	8.04	111.57
HDF5 – log-based VOL (log layout)	3.91	4.26	28.42

Time in seconds

Obstacles

- Read performance can be poor
  - Reading a subarray of a dataset must sweep through all metadata logs to find the interactions
- Log metadata size can be large for fragment requests
  - File size can be larger than the one in canonical layout
- Using native VOL to write logs to file is slower
  - Log VOL queries the file offsets of datasets and use MPI I/O directly to write

Lessons Learned

- One can use a data layout different from the default
  - Any high-level I/O libraries, such as HDF5 and PnetCDF, can be used to support this idea
  - Experimental examples can be found in the E3SM-IO benchmark repo
  - https://github.com/Parallel-NetCDF/E3SM-IO