HDF5 DAOS VOL Plugin

November 14, 2018

Copyright 2018, The HDF Group

Scot Breitenfeld, Neil Fortner,
Jordan Henderson, Jerome Soumagne,
Elena Pourmal

Outline

- Introduction to The HDF Group HDF5 DAOS project
- HDF5 VOL Architecture
- HDF5 DAOS VOL Plugin

Introduction to the HDF5 DAOS Project

The HDF Group has a long history collaborating with the Intel team

- During 2013-2016 we created prototypes for
  - HDF5 VOL architecture
  - HDF5 DAOS VOL plugin

Goal: Provide a seamless way for the HDF5 applications to access data stored in DAOS Object Store

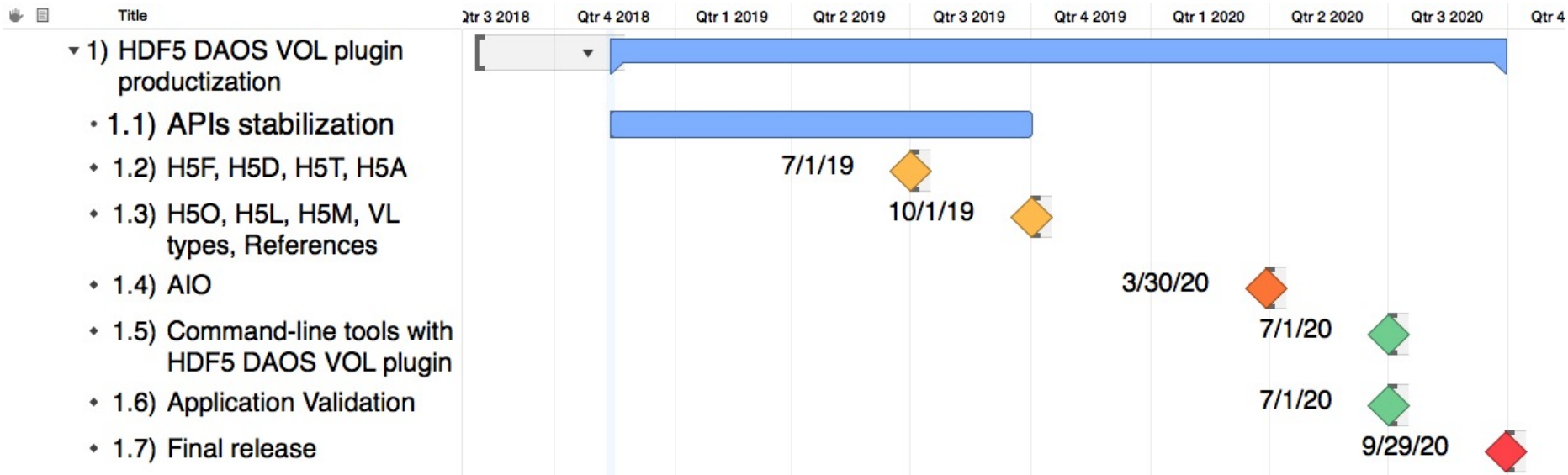
- Minimum changes to the source code
- Support both sequential and parallel I/O
- Get full advantage of the DAOS features
  - AIO
  - Transactions
  - Failure management
  - Map Objects

Introduction to the HDF5 DAOS Project

- **Objectives**
  - Stabilization of
    - HDF5 VOL architecture
    - HDF5 VOL plugin APIs
  - VOL plugin integration with HDF5 regression tests and command-line tools
  - Porting HDF5 applications to DAOS Object Store
  - Performance tuning

HDF5 DAOS Project Roadmap

The HDF Group



HDF5 VOL Architecture

Virtual Object Layer vs. Virtual File Driver

HDF5 VFD vs. HDF5 VOL

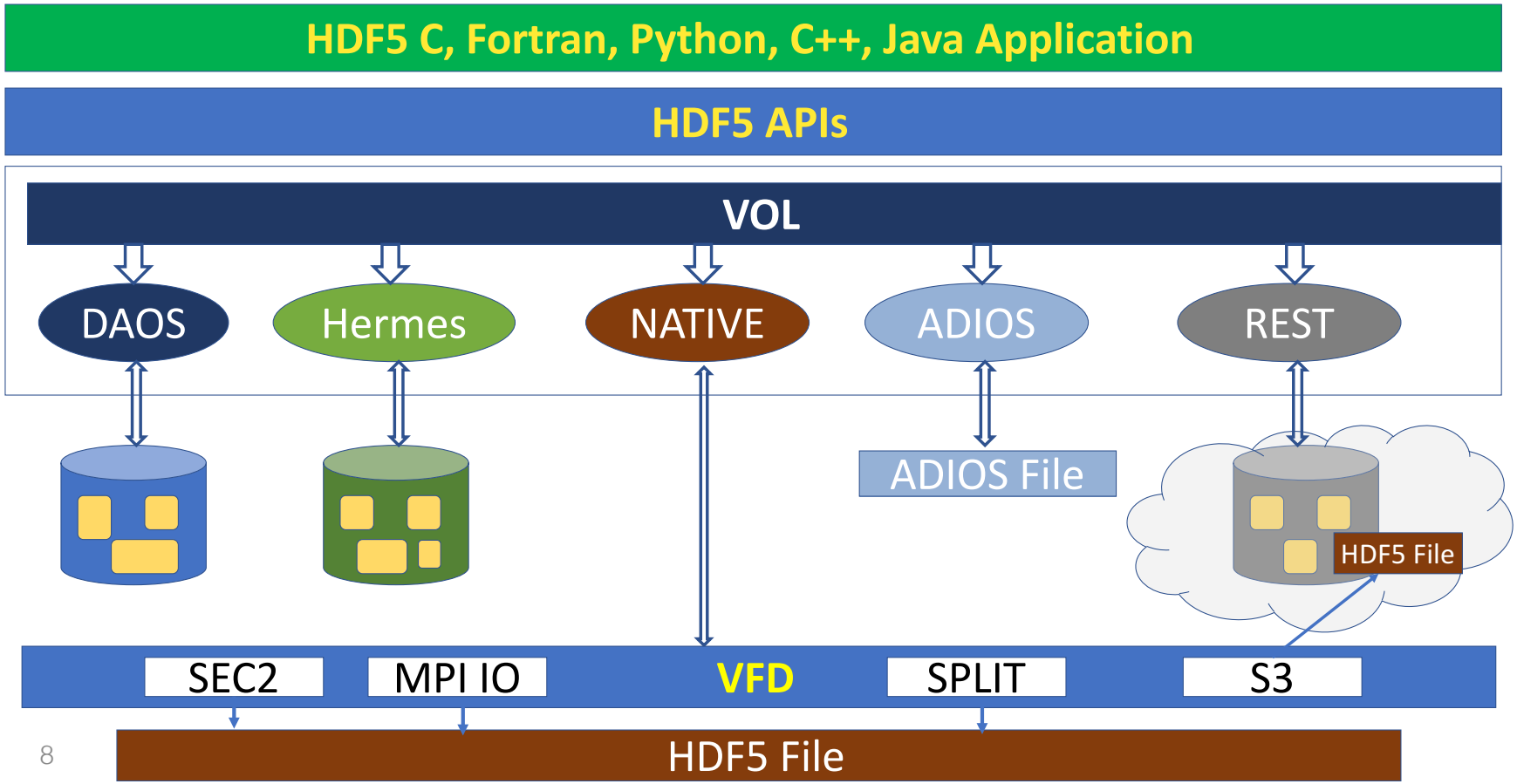
VFD

- HDF5 library I/O public interface
  - Writes/reads offsets and bytes
    - POSIX FS: SEC2, MPI I/O, Split, Family
    - Object Store-like: S3, HSDS
- Extendible and stackable
- *Cannot leverage Object Storage properties easily*

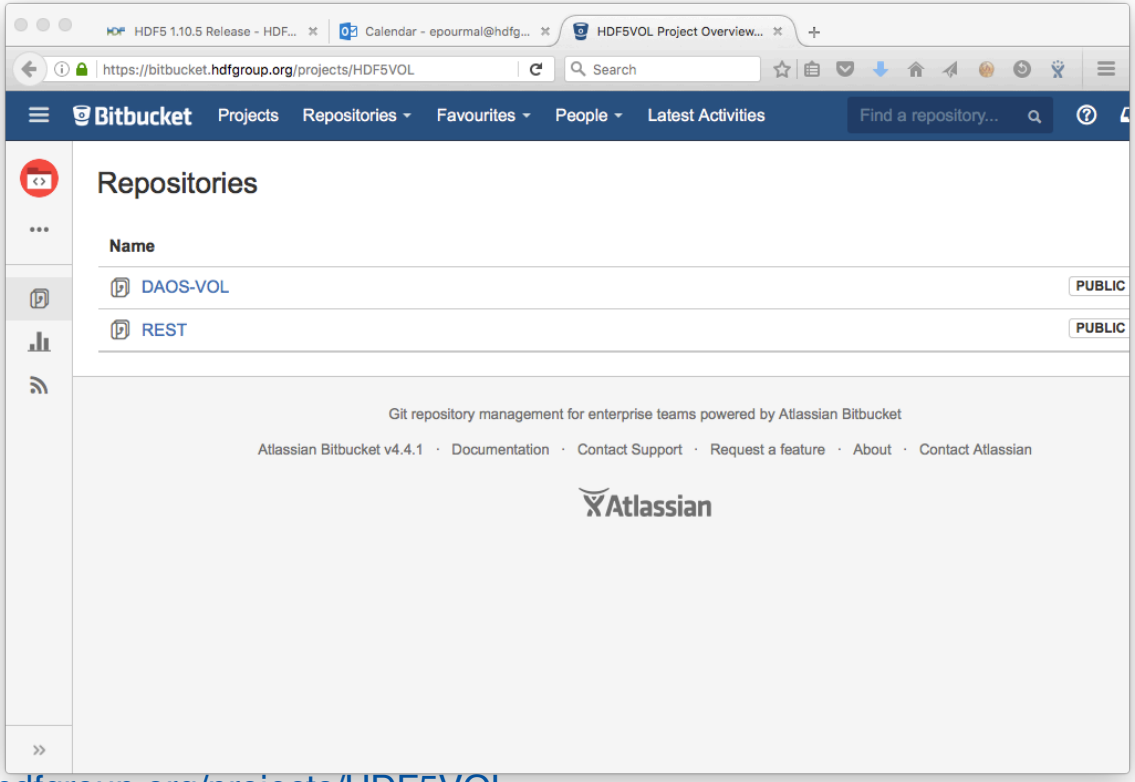
VOL

- HDF5 internal layer
  - Intercepts HDF5 calls and routes them to VOL plugin
    - NATIVE, DAOS, REST, netCDF, ADIOS
- Extendible and stackable
- *Cannot leverage easily HDF5 built-ins*
  - *Datatype conversion, compression, selections*

HDF5 VOL and VFD



HDF5 VOL Plugin and HDF5 VOL Source



https://bitbucket.hdfgroup.org/projects/HDF5VOL
https://bitbucket.hdfgroup.org/projects/HDFFV/repos/hdf5/browse

HDF5 DAOS VOL Plugin

Technical Considerations

HDF5 to DAOS Mapping

HDF5

- **HDF5 File**
- **HDF5 Group**
  - Link name
  - OID or PATH
- **HDF5 Dataset**
  - Chunk coordinates
  - Dataset values in the chunk
- **HDF5 Named Datatype**
- **HDF5 Map**

DAOS

- **DAOS Container**
- **DAOS Object**
  - dkey
  - value
- **DAOS Object**
  - dkey
  - value
- **DAOS Object**
- **DAOS Object**

HDF5 Programming Model for using VOL

- **Currently applications have to use hard-coded VOL specific APIs to initialize and use the plugin!**
  - Burden on application
  - Not portable between different VOL plugins
- **Programming Model with HDF5 DAOS VOL plugin:**
  - Initialize DAOS VOL plugin
    - **H5daos_init will be deprecated**
    - **H5VLregister**
    - **H5VLunregister ?**
  - Set HDF5 access property list to use DAOS VOL plugin
    - **H5Pset_fapl_daos will be deprecated**
    - **H5Pset_vol**
  - Use access property list to create or open DAOS container (HDF5 file)
  - Follow HDF5 programming model to manage objects in the container (in the file)

Unresolved Design Issues

- Finding, loading and initializing a VOL driver
- HDF5 VOL or VOL plugin specific issues
- HDF5 Tools
- HDF5 Testing

Finding, loading and initializing a VOL driver

- **How to tell HDF5 which VOL plugin to use and how to pass required information?**
- **Possible options**
  - Identifier
  - Filename as URI
  - Environment variables
  - Configuration file

Finding, loading and initializing a VOL driver

- **Filename as URI**
  - Similar to HTTP URL
  - Register URI scheme and VOL plugin with The HDF Group
    - URI scheme disctates which VOL plugin to use
  - VOL parameters can be passed as part of URI string using reserved character (TBD) to separate filename and parameters
  - hdf5:///home/user/file.h5**?**vol_param1=abc**?**vol_param2=xyz
- ***Will this approach work with stackable VOLs?***

Finding, loading and initializing a VOL driver

- **Environment variables**
  - Flexible
    - Easy to update, to overwrite the default VOL plugin settings, etc.
    - Used by HDF5 for other features
  - Requires a standard (as with URI) for passing information
    - All VOL plugins will need to follow the standard
    - May not work for some VOL plugins (e.g., username and password are required for accessing file)
- **Configuration file**
  - Flexible
  - Requires a standard that both HDF5 library and VOL plugins should follow

HDF5 VOL and VOL plugin specific issues

- **Non-exposed private routines and structure useful for a VOL plugin**
  - Copying dataspace selection
  - Functions to deal with “set operations” on the selections
  - Datatype conversion functions that need background conversion buffer
- **HDF5 References**
  - References to HDF5 objects and references to sub-regions in datasets
  - Use addresses of the objects in the file
  - Need to be unified into a single “token” that can be implemented by a VOL plugin

HDF5 command-line tools

- **Issues are similar to any other HDF5 application that uses VOL plugin**
  - Rely on HDF5 library to do “right” thing to find, load, and set up VOL plugin
- **Need a common way to specify a VOL plugin using command-line parameter**
- **Build issues**
  - VOL plugin cannot be built before HDF5 library, tools cannot be built before VOL plugin built, but tools are built at the same time when HDF5 library is built
  - Solution: HDF5 command-line tools has to be “VOL plugin” agnostic, i.e., we need “finding, loading, and initializing” process performed by the HDF5 library.

HDF5 Testing

- **How to test HDF5 library with different VOL plugins?**
  - Current HDF5 test suite consist of unit tests for testing library internals, APIs, and features
  - The test suite cannot be easily used with arbitrary settings like property lists, VFDs and VOL plugins
  - Work needs to be done to separate tests for HDF5 “native” specific VOL and general VOL plugins
    - Approach
      - Add test suite that was created for REST VOL and can be used by DAOS VOL
      - Refactor existing HDF5 test suite as we move forward

HDF5 Map Object

- **HDF5 map object (key-value store) is extension to "classical" HDF5 data model**
  - HDF5 uses key-value stores internally but doesn't expose a generic key-value store to the API.
- **Application uses HDF5 map object to index information stored in HDF5 file**
  - Use case (NASA JPSS data processing): find a dataset in a file with an attribute (or attributes) with a particular name and value(s), and run analytics on the dataset array.
  - Currently, one has to open all datasets in the file and read all attributes to find the one that is needed.
  - One can use map object to store ALL attributes (name, <value, object address or path>)
- **New HDF5 Map APIs**
  - Available only with DAOS plugin
  - H5M interface with the following operations
    - Create, open, iterate
    - Set/get key-value pair
    - Check if key exists
    - Get datatypes for keys ad values
    - Get number of key-value pairs store in the map object

Asynchronous I/O

Motivation

- **Time on HPC clusters is very valuable**
- **With traditional I/O, when waiting for write to complete, processors are idle**
  - Wasted time that could be used for computation
- **Asynchronous I/O: perform I/O in the background while processors continue computation**

Programming Model

- **Enable AIO on HDF5 file via file access property list (FAPL)**
  - `H5Pset_async(fapl_id)`
  - All operations are async that do not query file or hold buffer
    - Do not need to rework existing applications
- **Enable AIO on individual operation via dataset transfer property list (DXPL)**
  - `H5Pset_async_op(dxpl_id)`
  - Operations using this DXPL are always asynchronous
    - Application must ensure completion before reusing buffer (for write) or checking result (for read)

Implementation

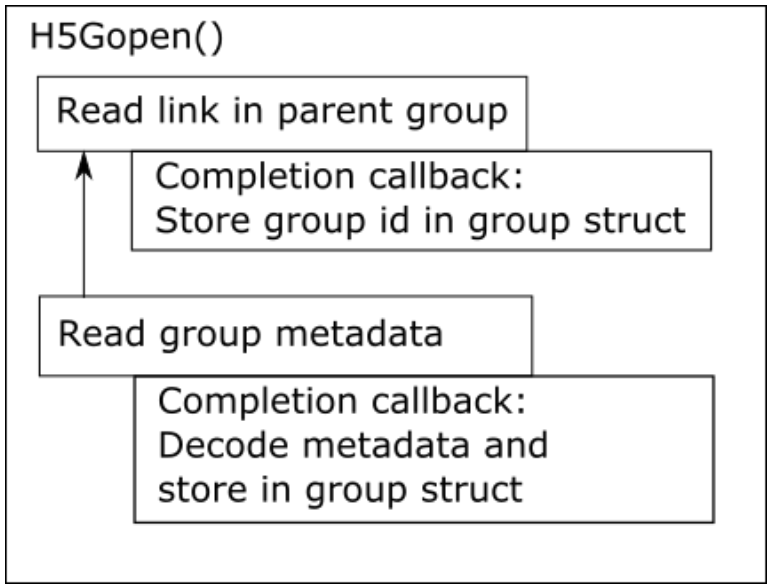
- **Threadless progress engine**
  - To make progress, check for AIO task completion, execute callbacks for completed tasks, and initiate execution of dependency children
  - Make progress whenever entering the HDF5/DAOS layer
- **Use DAOS task API to implement much of the progress engine**
  - Must integrate support for MPI asynchronous operations
- **App may spawn dedicated progress thread**
  - No need to call into HDF5/DAOS layer to make progress

Internal AIO

- **DAOS and MPI operations within an HDF5 operation can always be made asynchronously even if not requested through HDF5 API**
  - Wait at end of HDF5 operation if not asynchronous HDF5 op
- **Create dependencies between DAOS and MPI operations as needed**
- **On metadata write operations, place all operations in a DAOS transaction to make the HDF5 operation atomic**

Internal AIO Example: H5Gopen() (Independent)

- Must read link from parent group to retrieve target group's DAOS ID
- Using DAOS ID, read metadata from group's DAOS object
- Deserialize metadata and assemble in-memory group structure

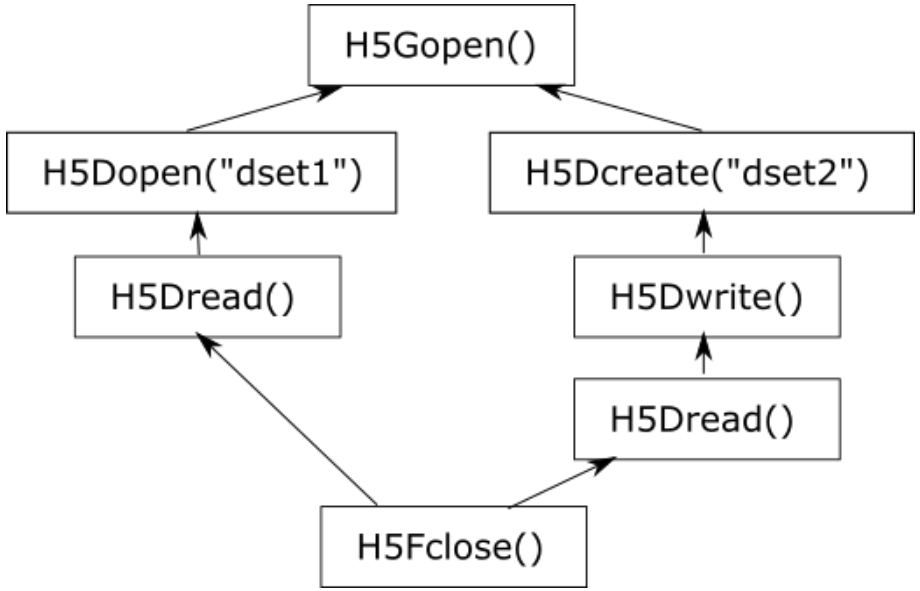


External AIO

- When HDF5 operations depend on each other, do not block on parent operation
- Create graph of DAOS/MPI operations that spans multiple HDF5 operations
- Use the same progress engine to manage the overall AIO graph
- Only applicable when AIO requested on the HDF5 file

External AIO Example

- Open a group
- Open two datasets in the group
- Read from one dataset, write to then read from the other
- Close the file



Possible Issues

- **Error reporting is more difficult – if an operation fails long after it started how do we inform the application which operation failed?**
  - Rollback entire file to state before failed operation?
- **Increased code complexity (both for internal implementation and applications)**
- **Application may not be able to progress if it can't modify the write buffer (general AIO issue)**

Implementation Roadmap

- **Build in AIO support at low level as operations are implemented**
  - Internal AIO
- **Add in support for external AIO**
- **Add support for HDF5 AIO API when available**
- **Add tests as features are implemented**

Acknowledgements

We thank Intel team for feedback and guidance.

Special thanks to

Johann Lombardi, and Mohamad Chaarawi (Intel) for DAOS technical insights.

Quincey Koziol (NERSC) for helping with the VOL feature integration into mainstream HDF5.