Accelerating HDF5's Parallel I/O for Exascale using DAOS

October 12, 2021

Jerome Soumagne, Jordan Henderson, Neil Fortner, Scot Breitenfeld, Raymond Lu, Dana Robinson, Neelam Bagha and Elena Pourmal
The HDF Group

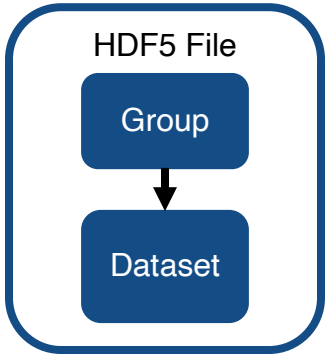
Mohamad Chaarawi, Ira Lewis and Johann Lombardi
Intel Corporation

The HDF Group

Outline

- **What's new in this presentation**
  - Full paper: "Accelerating HDF5 I/O for Exascale using DAOS," in IEEE Transactions on Parallel and Distributed Systems, doi: 10.1109/TPDS.2021.3097884.
- **HDF5 Parallel I/O w/Native File Format**
- **Introduction to Intel ® DAOS**
- **HDF5 DAOS VOL Connector and File Format**
- **New Features**
- **Evaluation and Application Example**
- **Conclusion**

HDF5 Parallel I/O w/Native File Format

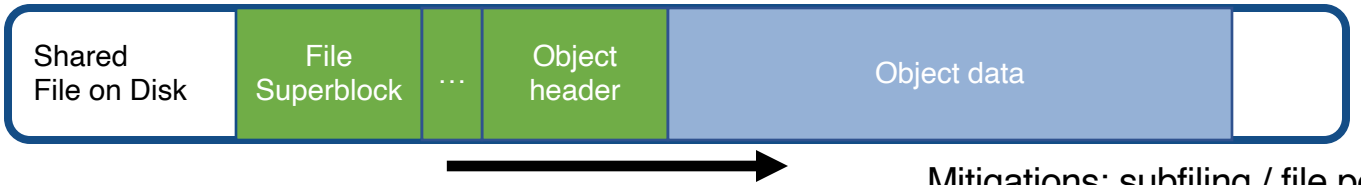
- 


```
H5Pset_fapl_mpio()
H5Fcreate()
H5Gcreate()
H5Dcreate()
H5Dwrite()
H5Dread()
```

Collective metadata operations
(not an API or data model restriction)

Collective or Independent raw data operations

- **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)

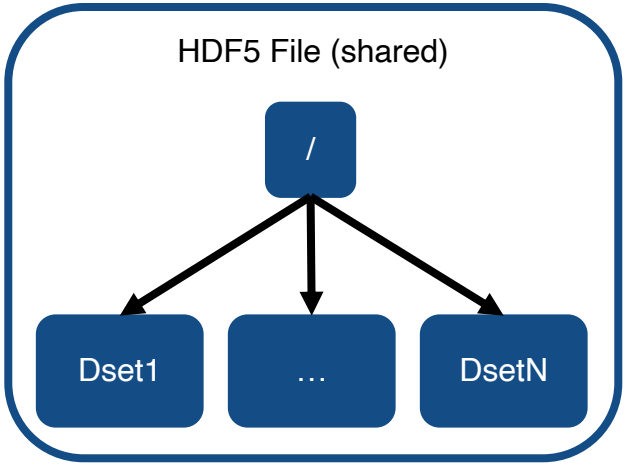


Mitigations: subfiling / file per process I/O
Added complexity, always limited by POSIX
Object-based model of HDF5 lost in storage

HDF5 Parallel I/O w/Native File Format

- **In Theory**

```
sprintf(dset_name, “Dset%d”, my_rank + 1);
fapl = H5Pset_fapl_mpio(…);
file = H5Fcreate(…);
dset = H5Dcreate(file, dset_name, …);
H5Dwrite(dset, …);
```

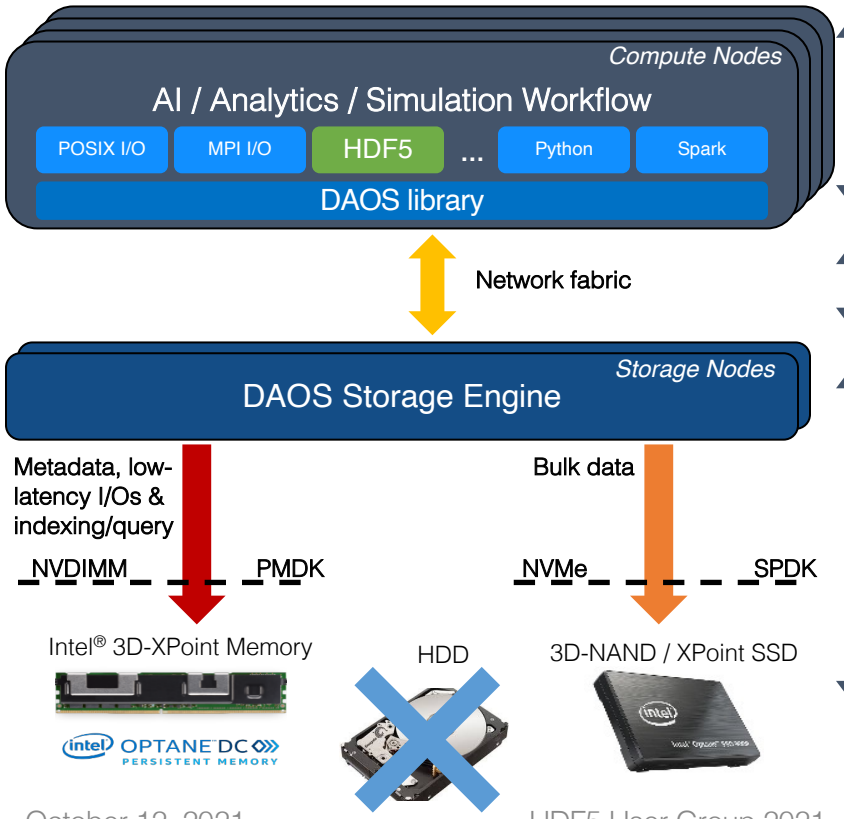


- **In Practice**

```
fapl = H5Pset_fapl_mpio(…);
file = H5Fcreate(…, fapl, …);
for (i = 0; i < n_ranks; i++) {
    sprintf(dset_name, “Dset%d”, i + 1);
    dset[i] = H5Dcreate(file, dset_name, …);
}
H5Dwrite(dset[my_rank], …);
```

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

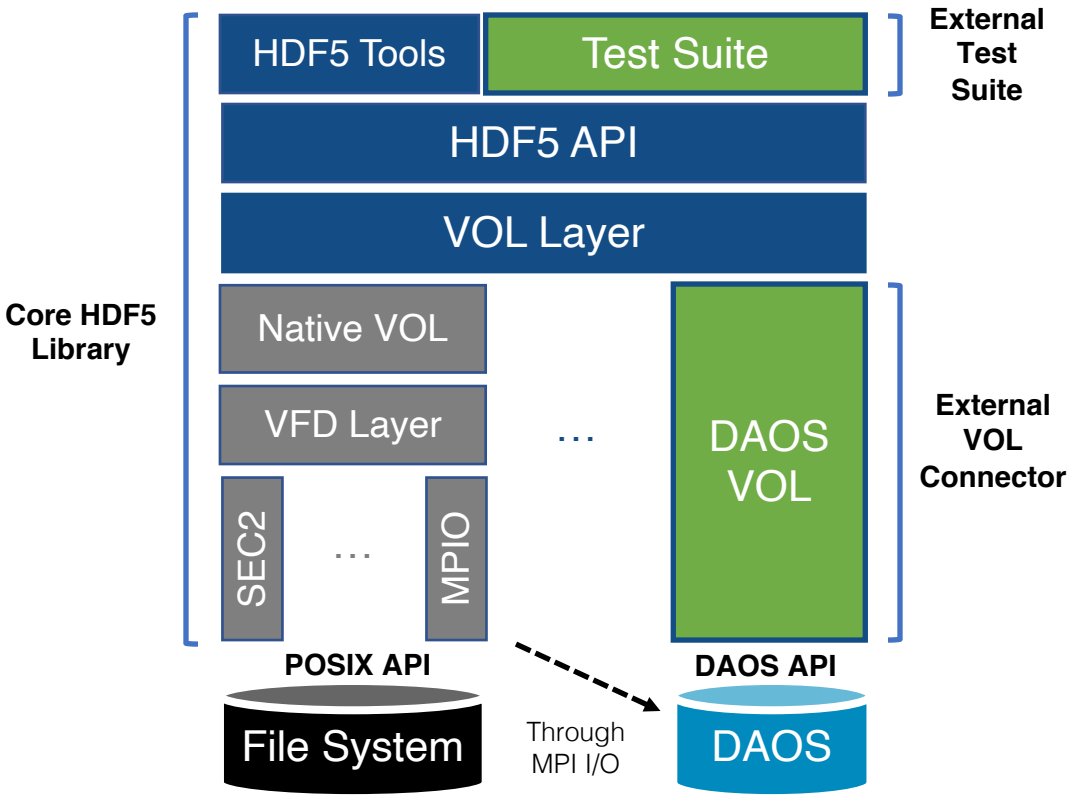
Delivers high-IOPs, high-bandwidth and low-latency storage with advanced features in a single tier

HDF5 VOL Architecture

- New Component
- Enhanced Component
- Native Component

- **Three main components:**
  - HDF5 Library
  - DAOS VOL Connector
  - (External) HDF5 Test Suite

- **Tools support:**
  - h5dump, h5ls, h5diff, h5repack, h5copy, etc



HDF5 DAOS VOL Connector

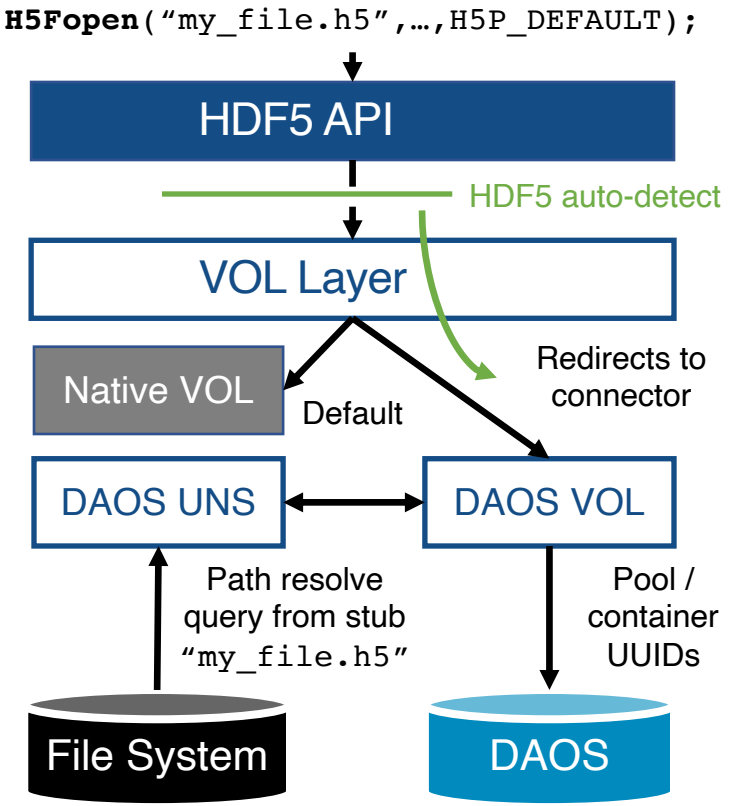
- **Allows the creation and use of HDF5 files in DAOS**
  - Minimal or no code changes for application developer (if only looking for compatibility)
  - Two ways to tell which connector to use
    - HDF5 file access property list (**recommended for new files or when manipulating multiple VOLs**)

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

    - Environment variable

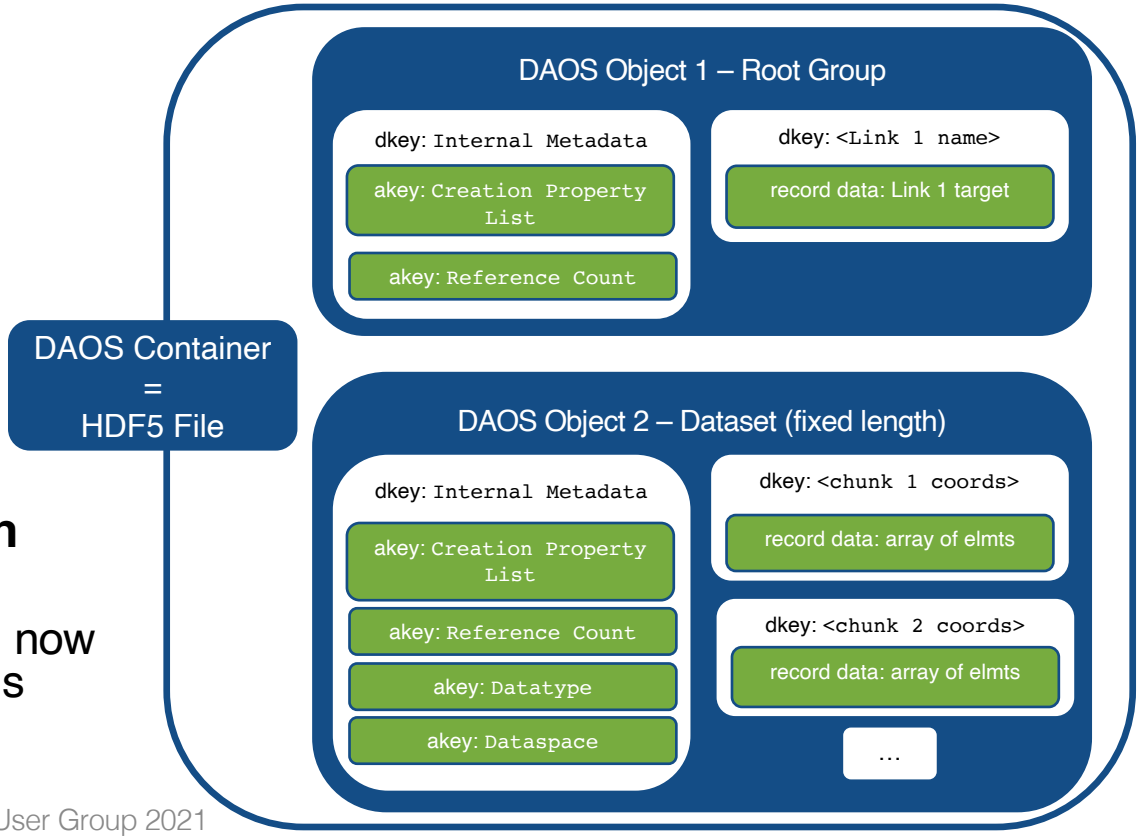
```
HDF5_VOL_CONNECTOR=daos
HDF5_PLUGIN_PATH=/path/to/connector/folder
```

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



HDF5 DAOS VOL "File" Format

- HDF5 File
  - Root Group
  - Dataset



- **Independent I/O through DAOS (K/V objects)**
  - Parallel I/O and chunking now become first-class citizens

Data control and Placement

- **Multiple options**
  - Chunking enabled by default for contiguous datasets, controlled with:

    `H5Pset_chunk()`
  - Set DAOS object class per DAOS object to control number of targets used for storing object (= **stripe count**):

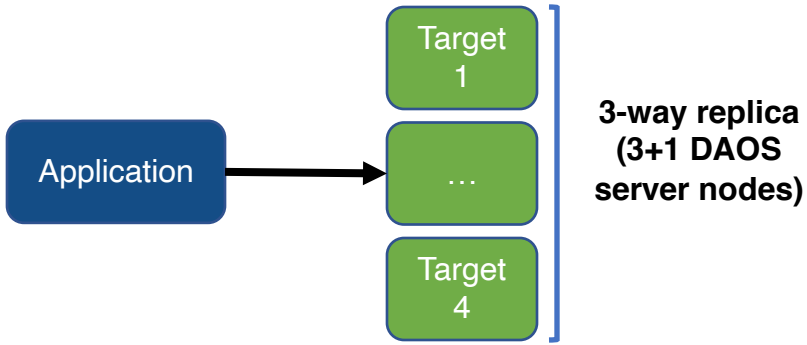
    `H5daos_set_object_class()`

    default uses all targets available
  - Set property to control numbers of **replicas** (for recovery), also controlled through:

    `H5daos_set_object_class()`

    default is no replica
  - Additional container properties (e.g. chksum, acl, rf) controlled with:

    `H5daos_set_prop()`



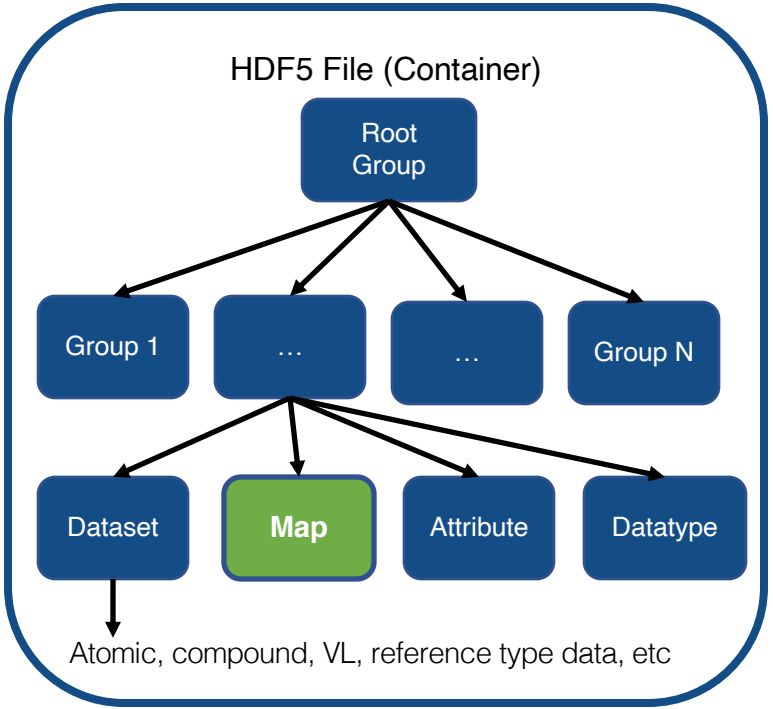
target ≠ storage node:
multiple storage targets per node

Features

- **All HDF5 features are currently supported except features specific to the native file format**
- **Additional features implemented**
  - Map objects (enabled by K/V objects)
  - File deletion
  - Independent metadata
    - HDF5 objects can be created independently
    - Currently enabled with:

```
H5daos_set_all_ind_metadata_ops()
```

    - **May become default behavior in the future**
  - Asynchronous I/O



Asynchronous I/O

- **Enables asynchrony using Event Set API**
  - Implemented at DAOS connector level
  - HDF5 API returns before operation completes, places operation in an “event set”, while tracking dependencies
    - Uses DAOS task engine

- **Asynchrony must be explicitly controlled by application**
  - Similar to existing async APIs, such as MPI non-blocking
  - Use async versions of all routines that may block
  - Beware of dependencies
    - e.g., **`H5Dcreate_async()`** → **`H5Dwrite_async()`** → **`H5Dclose_async()`**
    - **`H5ESwait()`** is responsible for advancing asynchronous operations

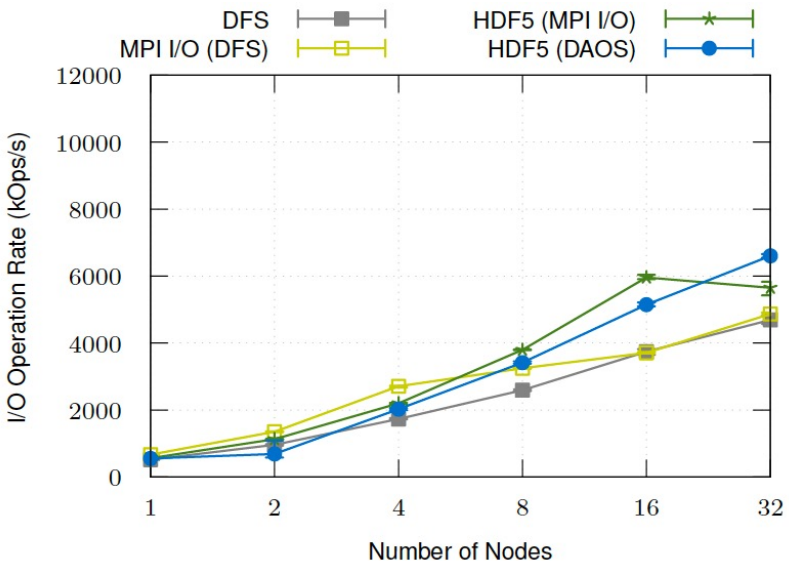
Evaluation – Configuration

- **DAOS system deployed on Frontera (TACC)**
  - 4 DAOS storage nodes
  - 24 Intel® Optane persistent memory DIMMs of 256GB each
  - InfiniBand HDR100 (100 Gb/s) connectivity to the compute nodes
  - Use only 28 cores within same NUMA node as InfiniBand card
  - 2 TB DAOS pool without NVMe backend to make exclusive use of persistent memory

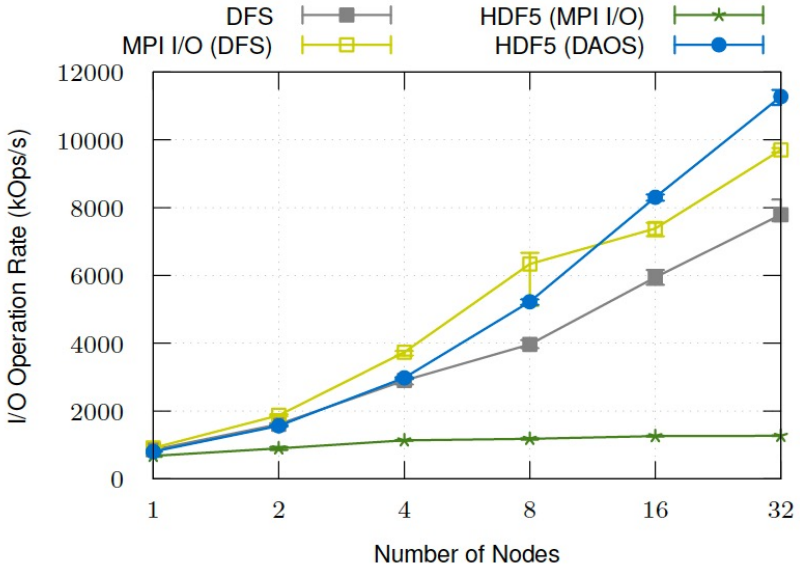
- **Software version used has evolved since then**
  - DAOS version used was 1.1.2.1
  - HDF5 version was 1.13.0rc5 and DAOS VOL version was 1.1.0rc3 (pre-release)

Evaluation – IOR small I/O (1 KB)

Write

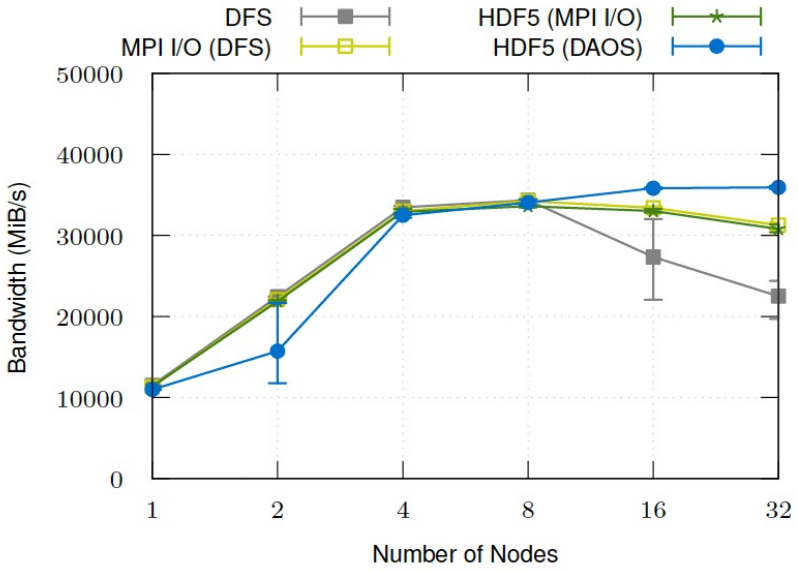


Read

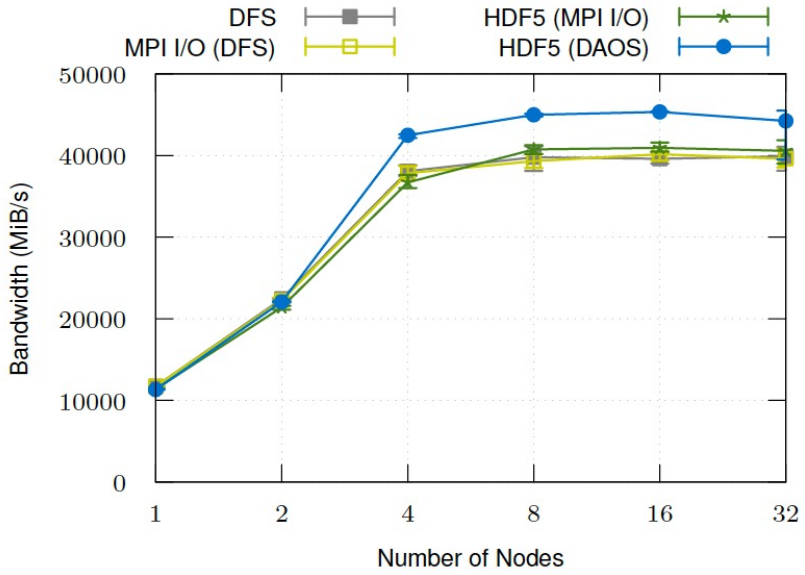


Evaluation - IOR large I/O (1 MB)

Write

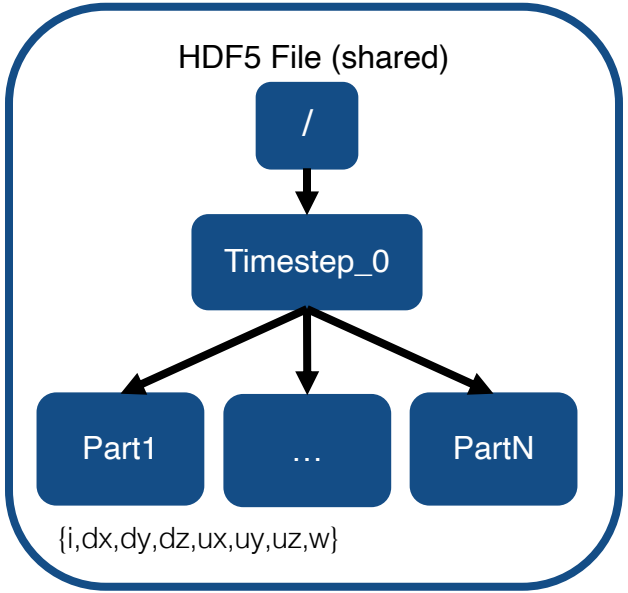


Read

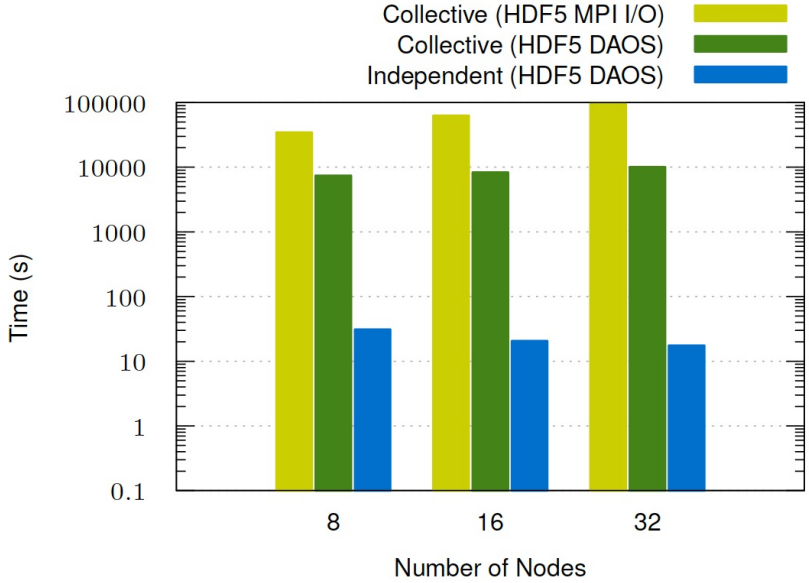


Evaluation – Example w/VPIC

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



VPIC I/O performance using collective and independent group creation



Conclusion

- **Native file format inherited limitations from block-based model**
  - Switching to object-based model is more in line with HDF5 data model
  - New storage models can now be defined without any “parallel” constraints
    - Storage model should map application’s data model

- **Switching to DAOS VOL is a one-line code change**
  - However…
    - New features such as async I/O, maps, fine-grained data control and placement can only be fully utilized with DAOS
    - We expect application I/O kernels to be re-worked based on these new features
- **DAOS VOL reached release candidate status**
  - Will be fully released along with HDF5 1.13.0

Acknowledgments / Questions

- **DAOS VOL Connector repository:**
  - https://github.com/HDFGroup/vol-daos
- **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.