HDF5 VOL Status Report

Scot Breitenfeld, Hyo-Kyung Lee and Larry Knox
The HDF Group

Version History

Feb 2022 – Initial Document Released

April 2022 – Changes: (1) Updated Section 2 testing results, (2) added Section 3.4, (3) added new LOG-VOL results to Section 5.1, and (4) added Sections 6 and 7.

September 2022 – Changes: (1) Added Polaris results for hdf5-iotest with the log-based VOL, Section 3.2.1 and (2) Updated Section 2 testing results.

1 Introduction

This report provides an overview of the HDF5 VOL connectors created for ECP [1,2,3]. Building and testing of VOLs were added to Spack and are discussed in Section 2. Finally, Sections 3-7 further investigated the async-vol and the log-vol with applications. The adios-vol was not tested for this group of applications since it does not support hyperslab selections[4]. Section 8 discusses the next steps in VOL connectors evaluation. Section 9 documents the status of VOL connectors testing.
This report is a living document. As such, it will be continually edited and updated as new VOLs are developed, the current VOLs have new developments or features, and new systems become available.

2 Spack for HDF5 VOL Tests

A Spack package to build and test various VOLs using vol-tests was developed for the project. Currently, the Spack VOL building and testing option are in a Spack fork at:

https://github.com/hyoklee/spack.git

The vol-tests require the latest HDF5 from develop branch, or version 1.13.2. All versions less than this will fail. Furthermore, the szip library should be enabled in HDF5. Additionally, the proper **HDF5_VOL_CONNECTOR** and **HDF5_PLUGIN_PATH** environment variables need to be set.

The HDF5 VOL packages consist of *hdf5-vol-async, hdf5-vol-cache* and *hdf5-vol-external-passthrough*. In Spack, the option to enable each VOL are:

- +vol-async: Use hdf5-vol-async.
- +vol-cache: Use hdf5-vol-cache.
- +vol-external-passthrough: Use hdf5-vol-external-passthrough.

The options to enable vol-tests are +async, +parallel and +part. Also, adding "**--test all"** option during installation will test all dependencies. Using "--**test root"** only tests the vol-tests. For example,

$spack install --test root hdf5-vol-tests+vol-async

will install HDF5 Async VOL first and then run the vol-tests using the HDF5 Async VOL library.

[1] This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
[2] This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 using NERSC award ASCR-ERCAP0020245.
[3] This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
[4] https://adios2.readthedocs.io/en/latest/ecosystem/h5vol.html

The results of the vol-tests are in Appendix 6. Unfortunately, most of the VOLs fail the vol-tests. The vol-tests is a comprehensive test and checks a wide range of HDF5 APIs functionality. However, most of the VOLs do not support the depth of API coverage in the vol-tests, so it is expected that they will fail. Currently, the vol-tests are not designed in a modular fashion, allowing specific unsupported API groupings to be skipped. The VOL tests developed by the VOL developers fare better since they contain only API tests that the VOL supports, Table **1**.

VOL	Compiler	Installs	Tests	vol-tests
log	gcc 10.1.0	Y	Y	N
	intel 19.1.3.304	Y	N	N
	cce 11.0.1	Y	Y	N
async	gcc 10.1.0	Y	N	N
	intel 19.1.3.304	Y	N	N
	cce 11.0.1	Y	N	N
adios2[1,2]	gcc 10.1.0	Y	Y	N
	intel 19.1.3.304	Y	Y	N
	cce 11.0.1	N	N	N

Table 1: VOL testing on Cori. [1] adios2 test fails when Python is built due to *libbsd* package error. *libbsd* is required for the *expat* library. Hence, an external Python was used to run the test. [2] adios2 build fails for *cce* because it fails to build *zstd*, which is required for *sz* and *blosc*. Building and testing only work when they are turned off.

Recent work has focused on using Spack to build four hdf5-vol packages, i.e., hdf5-vol-async, hdf5-vol-cache, hdf5-vol-log and hdf5-vol-external-passthrough on HPC and EA machines at ALCF, OLCF and NERSC. The Spack defaults for installing these packages were sufficient to build them successfully on Arcticus, Cori, Crusher, Perlmutter, Spock, and Summit. Theta required only the addition of one additional parameter to specify a newer version of CMake than the system provided.

For other compilers available in Spack, some of the dependency packages will fail to build. For several compilers, the failed package can be replaced with one built by the Spack default gcc, thus allowing the build to finish. Examples of successful commands with additional Spack parameters are listed in Appendix 9.1.

3 HDF5-IOTEST

Historically, parallel I/O tuning of the HDF5 library has suffered from the uncertainty of how the HDF5 usage schema (data layout, HDF5 tuning parameters, etc.) affects parallel I/O. Hence, an I/O HDF5 performance assessing tool, hdf5-iotest, has been introduced to quantify the *performance variability* of a *set*, **HDFspace**, of logically equivalent HDF5 representations of common data layout patterns. *hdf5-iotest* repeatedly writes (and reads) over several time-steps a configurable number of 2D array variables in a tiled fashion, Figure 1.

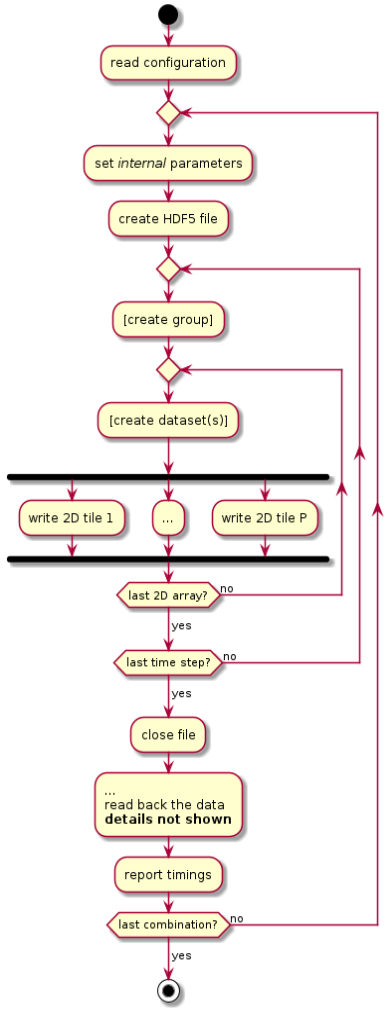

Figure 1 Flow diagram of hdf5-iotest

3.1 HDF5-IOTEST Configuration

In *hdf5-iotest*, each run of the program performs 192 (for parallel) different elements of set **HDFspace**, and are combinations of seven internal parameters used to configure the HDF5 library for write and read operations, Figure 2.

For the experiments, a fixed number of ranks was used, where the total number of rows and columns per MPI processes per 2D array variable were both 42 on Summit and 32 on Cori. The number of 2D array variables, *iarray*, was 10, and the number of repetitions, *istep*, of each element in the set was 10. An alignment increment of 16 MiB was used, and all objects were aligned. The metadata block allocation size was 2048 bytes.

To clarify the meaning of the internal parameter "*rank*" in the tests, take, for example, the size of a 2D array as (168,42) and the slowest dimension being the "array." When *irank* is 2, there will be ten groups (*iarray*=10), each having two datasets (*isteps*=2) of size (168,42). If *irank* is 3, "*istep*" is made into a third dimension, such that there will be ten datasets (*iarray*=10) of size (2,168,42). When *irank* is 4, "*iarray*" is made into a fourth dimension, and there will only be one dataset of size (10,2,168,42).
When the slowest dimension is "step" instead of "*array*." When *irank* is 2, there will be two groups (*istep*=2), each having ten datasets (*iarray*=10) of size (168,42). If *irank* is 3, "*iarray*" is made into a third dimension, such that there will be two datasets (*istep*=2) of size (10,168,42). When *irank* is 4, "*istep*" is made into a fourth dimension, and there will only be one dataset of size (2,10,168,42).

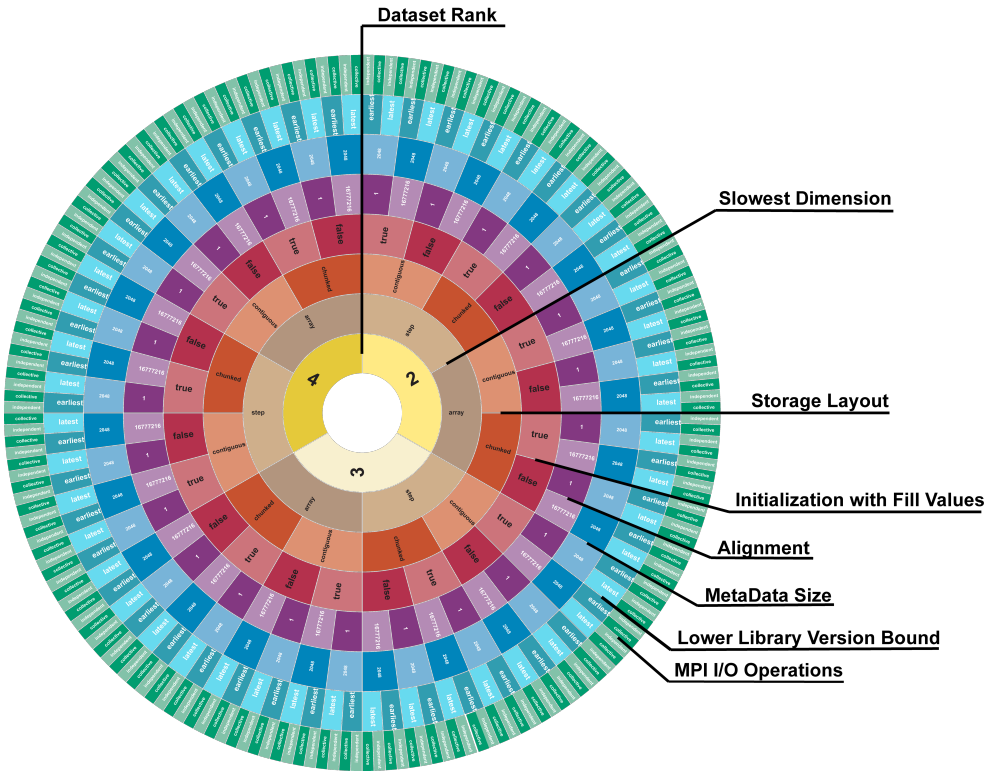

Figure 2 Overview of all seven parameter combinations per execution of the program.

Sections 3.2 and 3.3 report the LOG-VOL and ASYNC-VOL performance results, respectively.

3.2 HDF5-IOTEST and the LOG-based VOL

Invoking the LOG-VOL by setting environment variables avoids making changes in HDF5-iotest. The first experiment was run on Cori using 512 ranks (16 Haswell nodes). The Lustre stripe and count sizes were 8MiB and 48, respectively. Over the HDF5 parameter space, the file sizes ranged from 400.0 MiB to 818.0 GiB for NATIVE-VOL and 403.0 MiB to 404.5 MiB for LOG-VOL. The minimum metadata block allocation size was fixed at 2048 bytes. The LOG-VOL was less affected by the parameter space compared to NATIVE-VOL for total (Figure 3), write (Figure 4) and read (Figure 5), and for most of the parameters, the I/O times were magnitudes less with LOG-VOL.

Figure 3 Total time (read and write) for all elements in the HDFspace set for Cori on 512 ranks

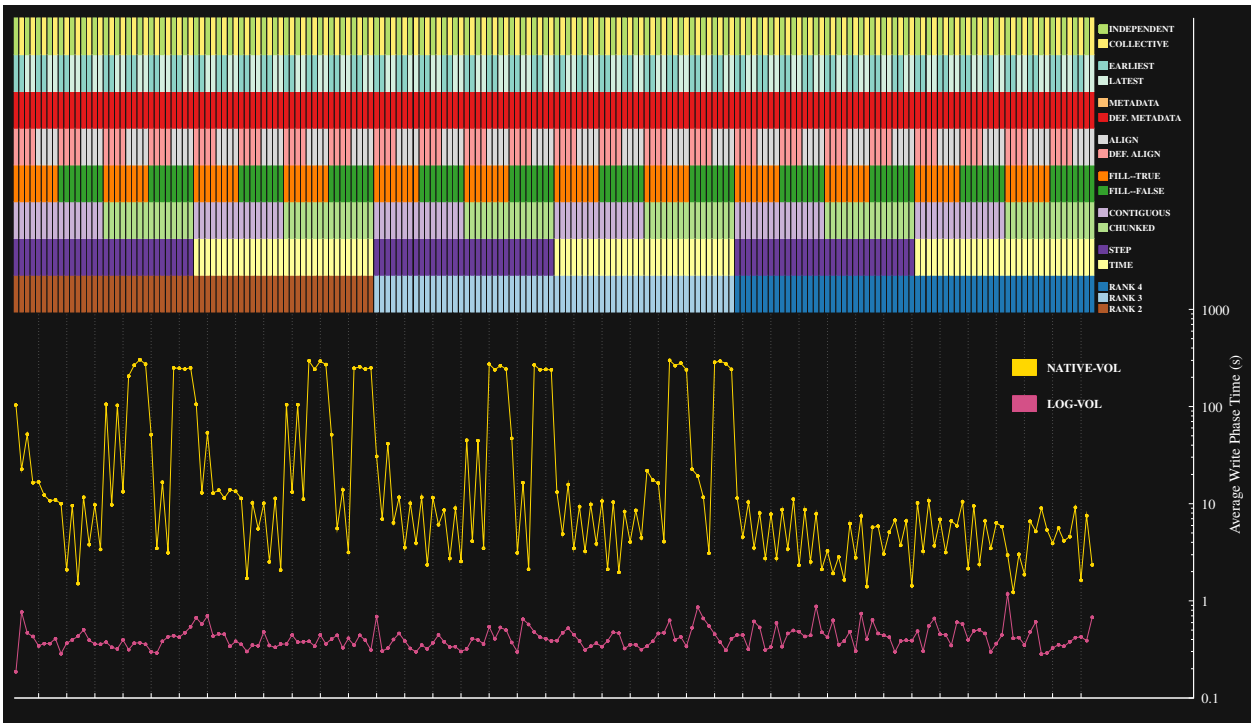

Figure 4 Write phase for all elements in the HDFspace set on Cori for 512 ranks

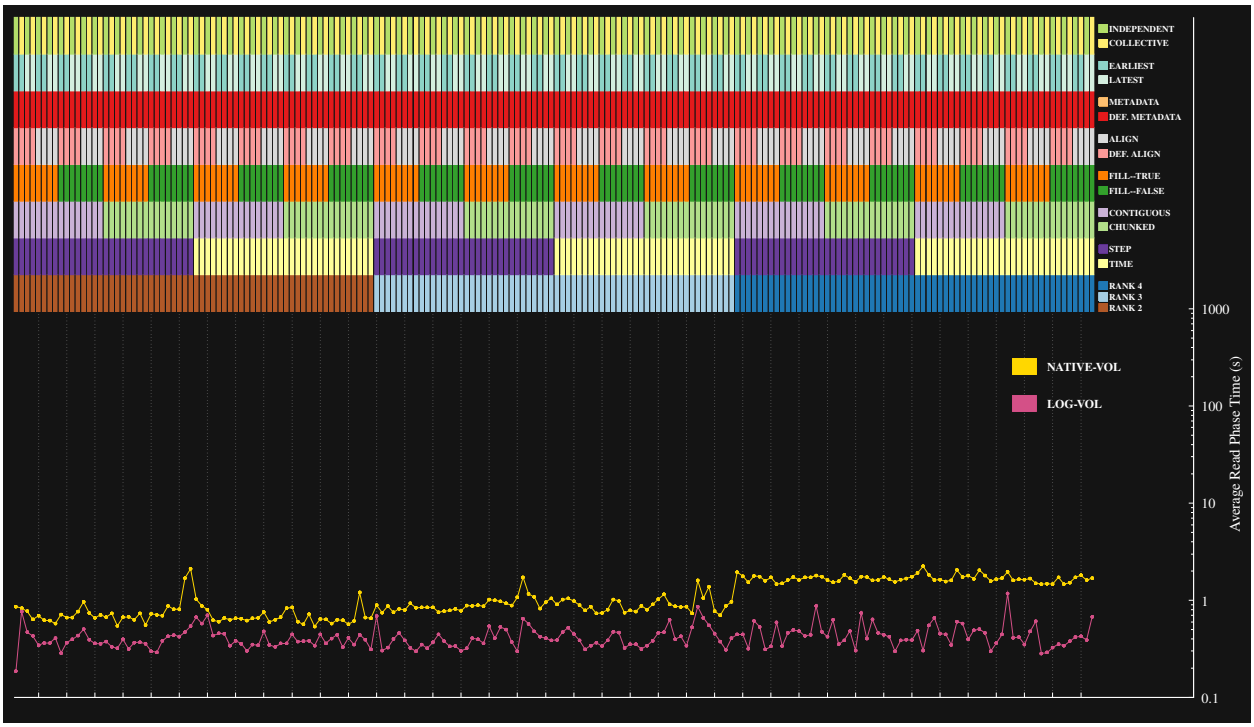

Figure 5 Read phase for all elements in the HDFspace set on Cori for 512 ranks

On Summit (GPFS), hdf5-iotest was run on 336 ranks, and the resulting file sizes varied from 454.2 MiB – 455.2 MiB for LOG-based VOL and 452.2 MiB to 538.4 GiB for NATIVE-VOL. The LOG-based VOL shows better total (Figure 6) and write (Figure 7) performance by orders of magnitude. However, the trend for

reading showed general improvement for NATIVE-LOG for ranks of arrays 2 and 3 but showed comparable performance for arrays of rank 4 (Figure 8).

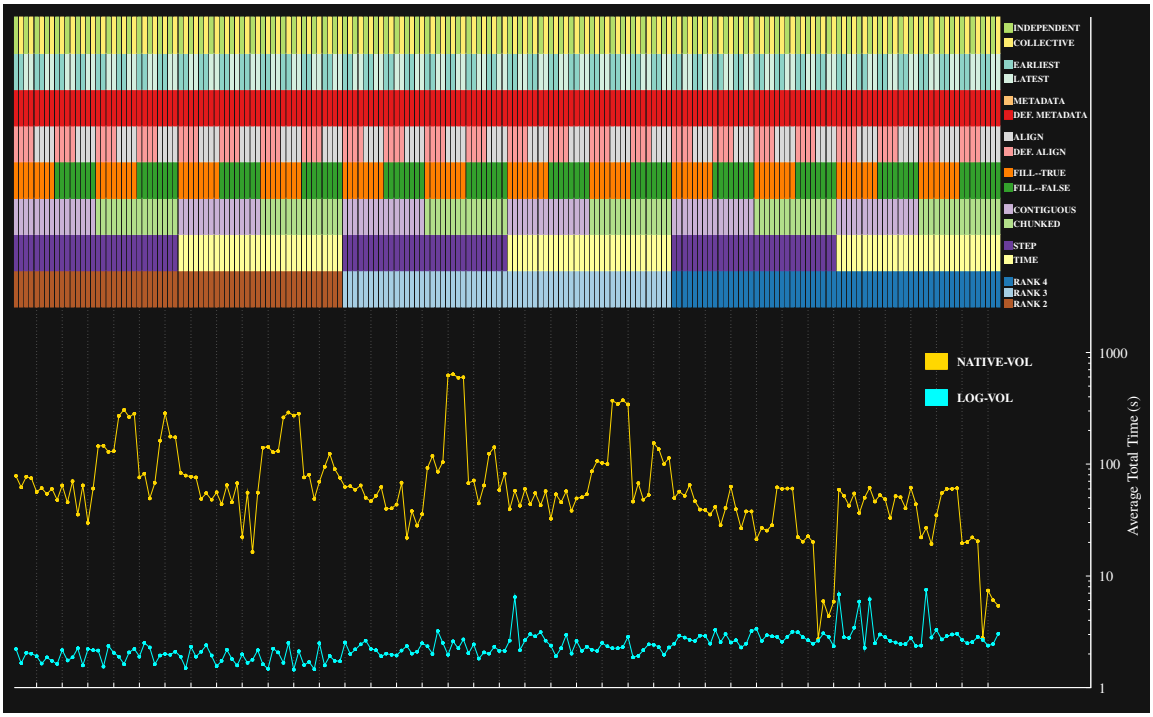

Figure 6 Total time (read and write) for all elements in the HDFspace set for Summit on 336 ranks

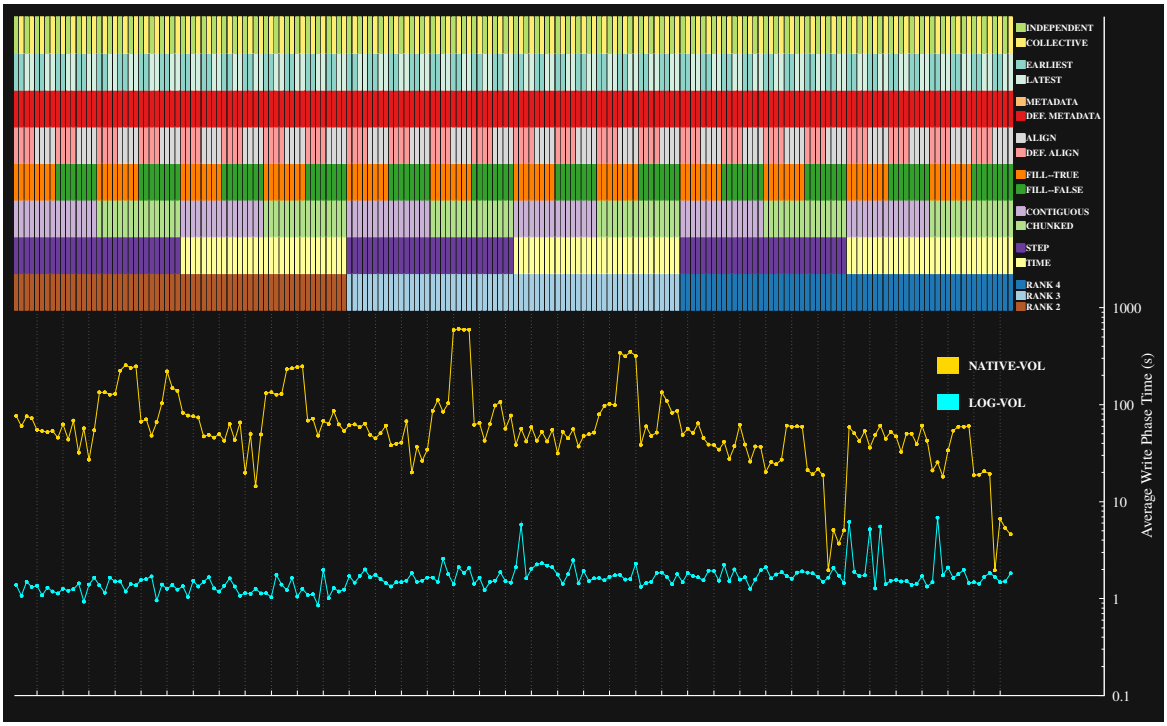

Figure 7 Write phase for all elements in the HDFspace set on Summit for 336 ranks

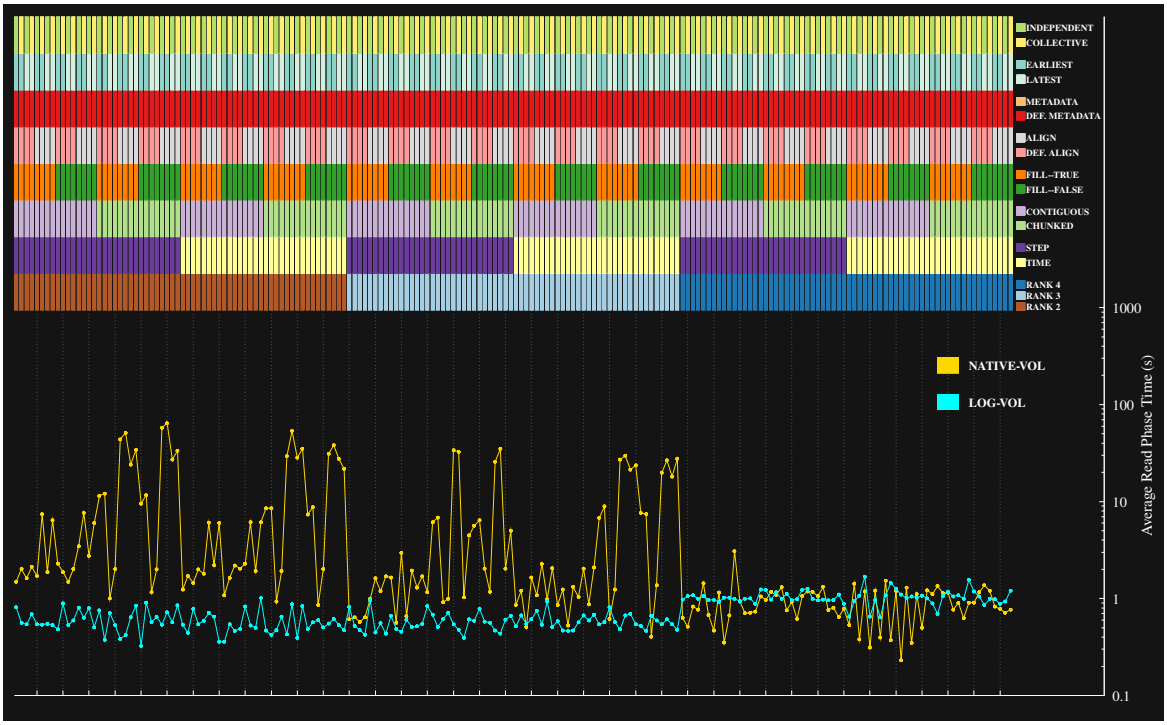

Figure 8 Read phase for all elements in the HDFspace set on Summit for 336 ranks

3.2.1 HDF5-IOTEST on Polaris with log-based VOL

The ANL system *Polaris* was used to study the performance benefits of using the log-based VOL. The number of ranks was 8192, and the general trend of improving the performance over a wide range of the parameter space is clearly evident, Figure 9.

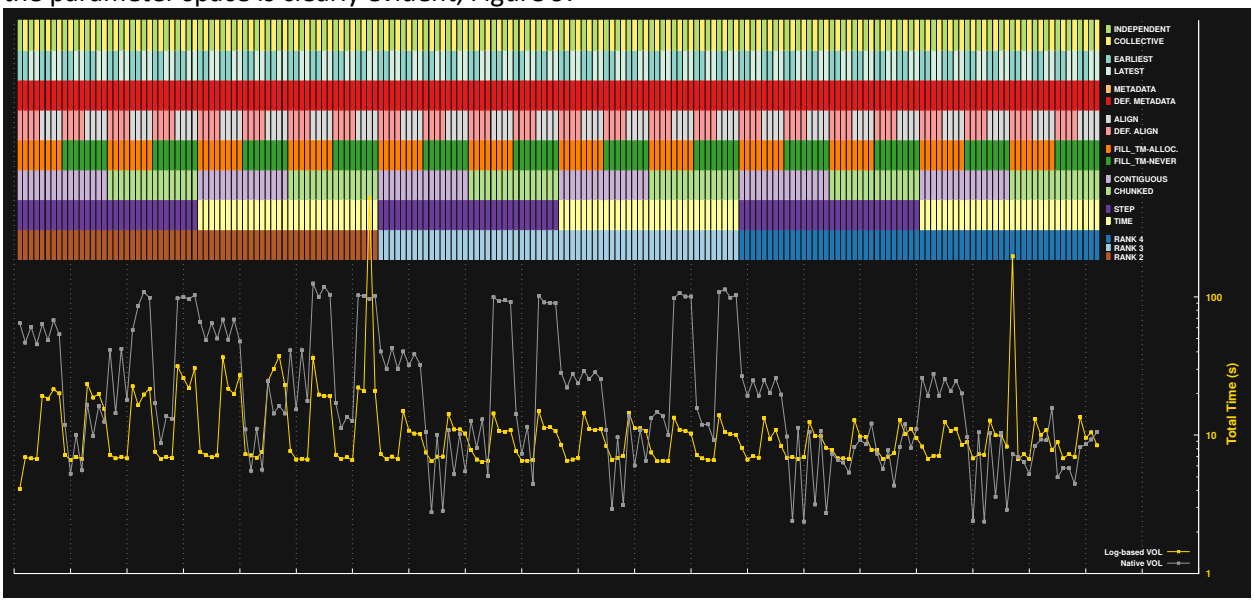

Figure 9 Total time (read and write) for all elements in the HDFspace set for Polaris on 8192 ranks.

3.3 HDF5-IOTEST and the async VOL

Explicit async was added to HDF5-iotest using the new async APIs introduced in HDF5 1.13.0. The program can introduce a pseudo *compute phase* delay between time steps via the "*delay*" keyword in the control deck. Optionally, the implicit async is enabled by setting the async environment variable and calling the non-async APIs.

The study used ten time steps and ten arrays (i.e., datasets), the number of rows and columns were 42 and 42 for Summit, and 32 and 32 for Cori as a starting point of weak scaling, respectively. The file size ranged from 400 MiB to 818 GiB for Cori runs, depending on the alignment.

3.3.1 Cori

The first study compares the sync and async HDF5 APIs with no computational phase. Unlike the LOG VOL, the ASYNC VOL follows the same peak trends as the NATIVE VOL. For the total, write and read times, the sync HDF5 APIs outperformed the async APIs (with the ASYNC VOL) for much of the parameter space, Figure 10-Figure 12. The overhead associated with the async operations can be seen by looking at one parameter event, 179, and running it using the native HDF5 API, the async APIs (*_async) with no ASYNC VOL, and the async APIs with the ASYNC VOL, Table 2. The difference highlights the computational workload needed to see the benefits of using async I/O. The next step is to verify the variance of the async VOL by running multiple instances of hdf5-iotest to see if the trends observed remain the same.

Table 2 Overhead between native and async APIs

	Time (sec)		
	Total	Write	Read
Native API	6.65	3.51	3.08
Async API, no ASYNC VOL	8.30	5.78	2.28
Async API, with ASYNC VOL	11.26	5.85	5.29

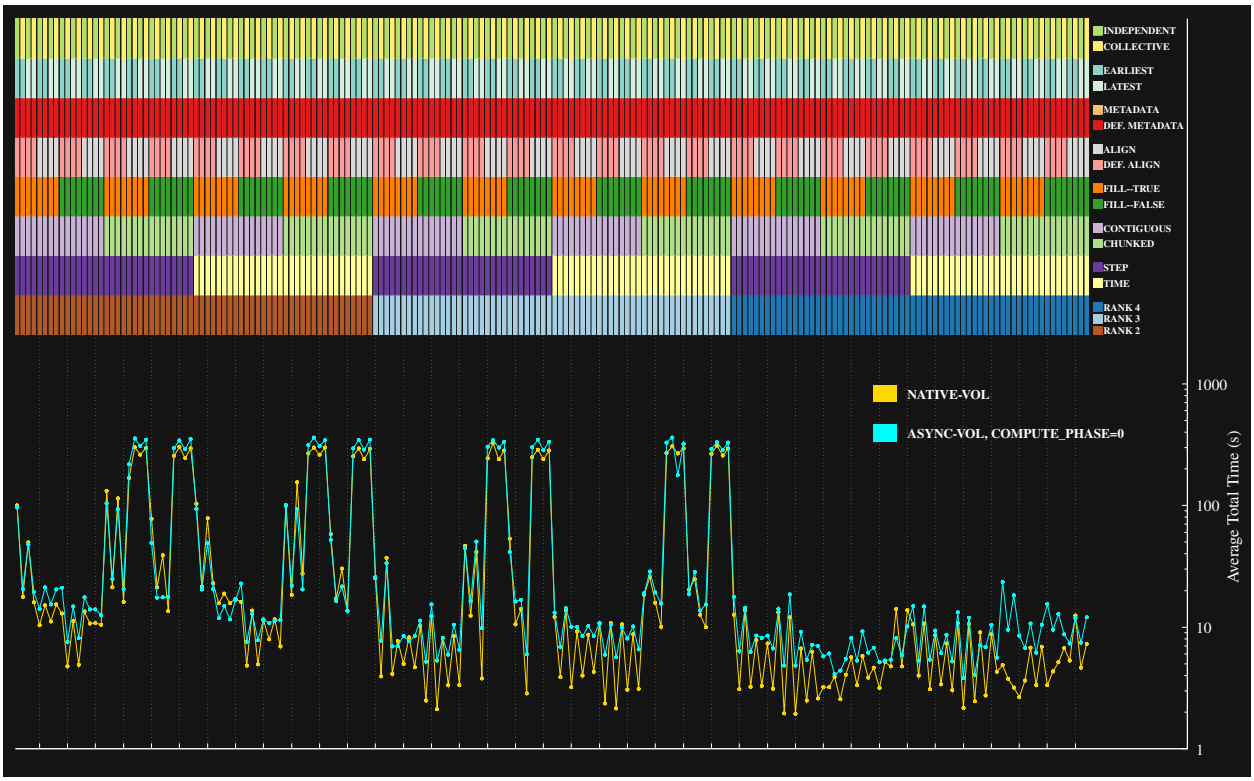

Figure 10 Total time (read and write) for all elements in the HDFspace set for Cori on 512 ranks, with no delay for a compute phase.

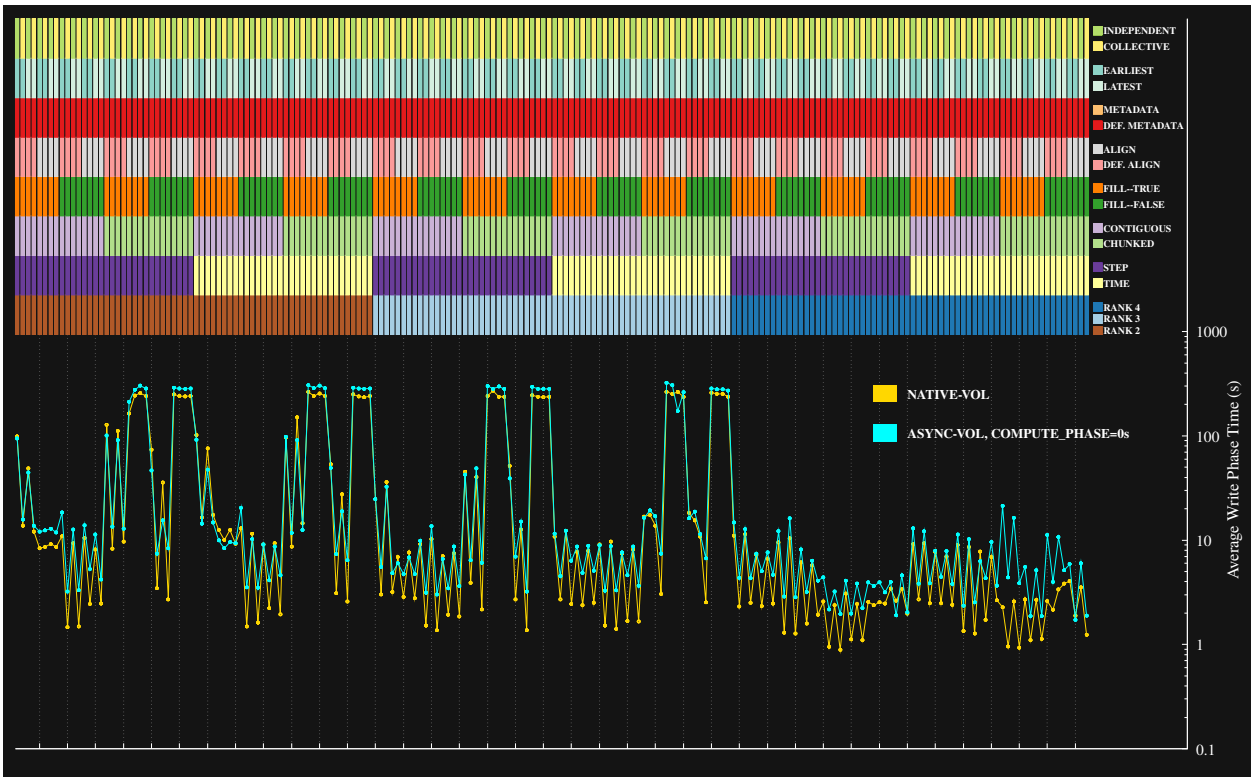

Figure 11 Write phase for all elements in the HDFspace set on Cori for 512 ranks, with no delay between writes.

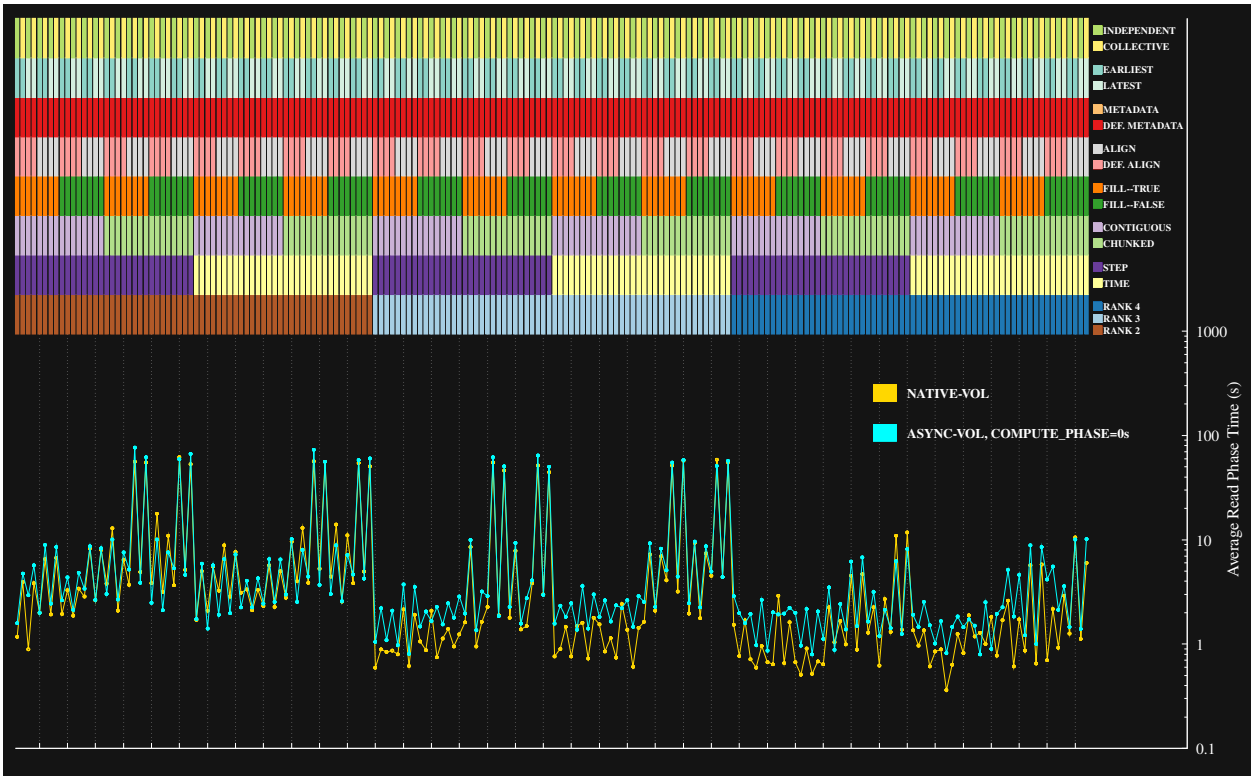

Figure 12 Read phase for all elements in the HDFspace set on Cori for 512 ranks, with no delay between reads.

3.4 HDF5-IOTEST with the DAOS-VOL

DAOS (version 2.0) with the DAOS-VOL and HDF5 13.0 is used with *hdf5-iotest* to study the performance of ANL's testbed system. The experiments used four nodes for a total of 144 ranks and 4 DAOS servers. Comparing synchronous and asynchronous calls and assuming no compute phase delay, the write, read, and total times are remarkably increased when the data sets are contiguous, Figure 13-Figure 15. However, in contrast to the native VOL, the extreme performance variability is not evident with the DAOS VOL. Adding a one-second compute phase delay results in the async implementation hiding most of the performance penalty of using contiguous datasets.

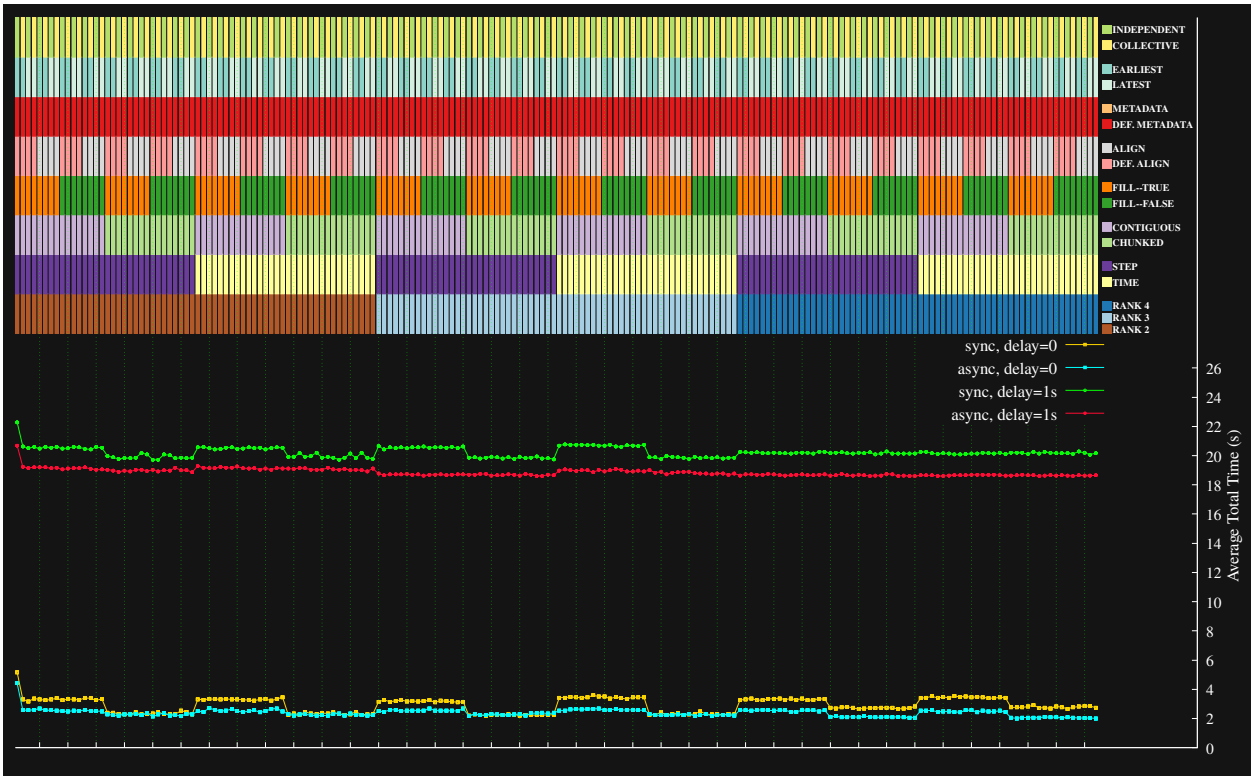

Figure 13 Total time (read and write) for all elements in the HDFspace set for ANL 144 ranks, with no delay and a one-second delay for a compute phase. The performance is degraded for contiguous datasets but can be hidden for a long enough compute phase.

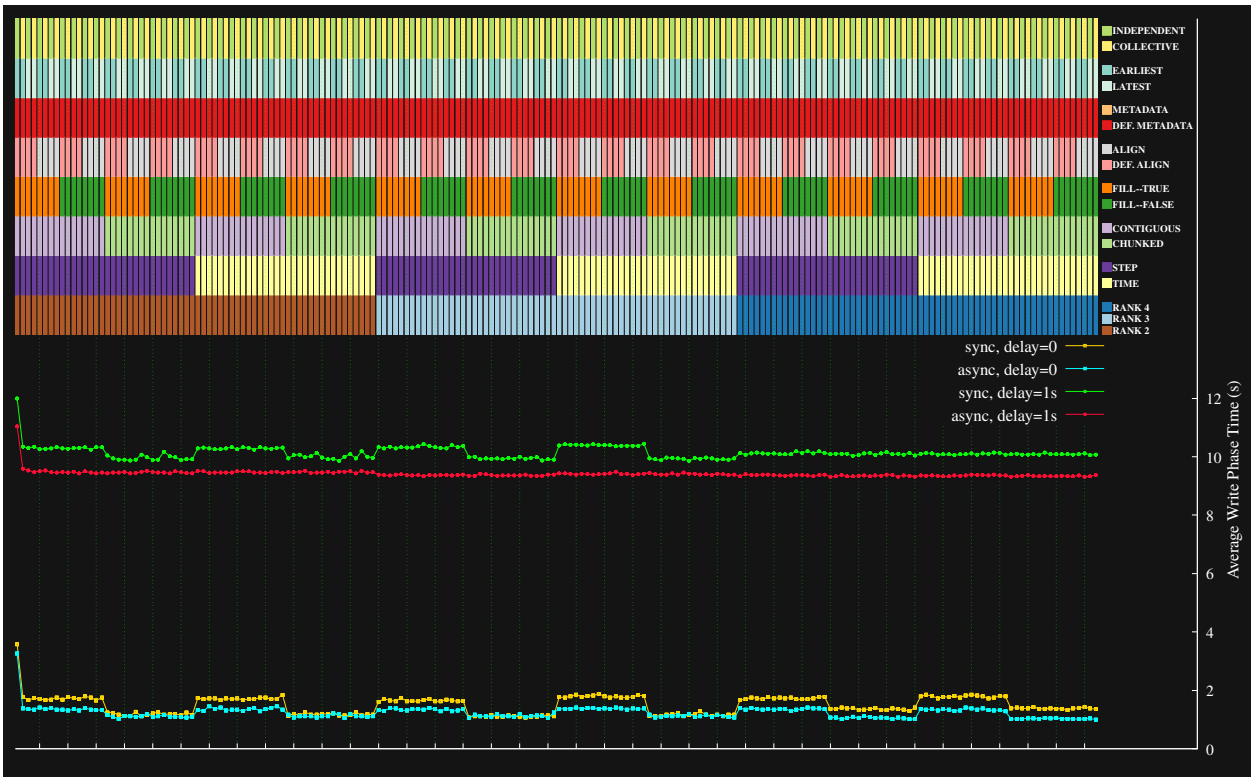

Figure 14 Write time for all elements in the HDFspace set for ANL 144 ranks, with no delay and a one-second delay for a compute phase.

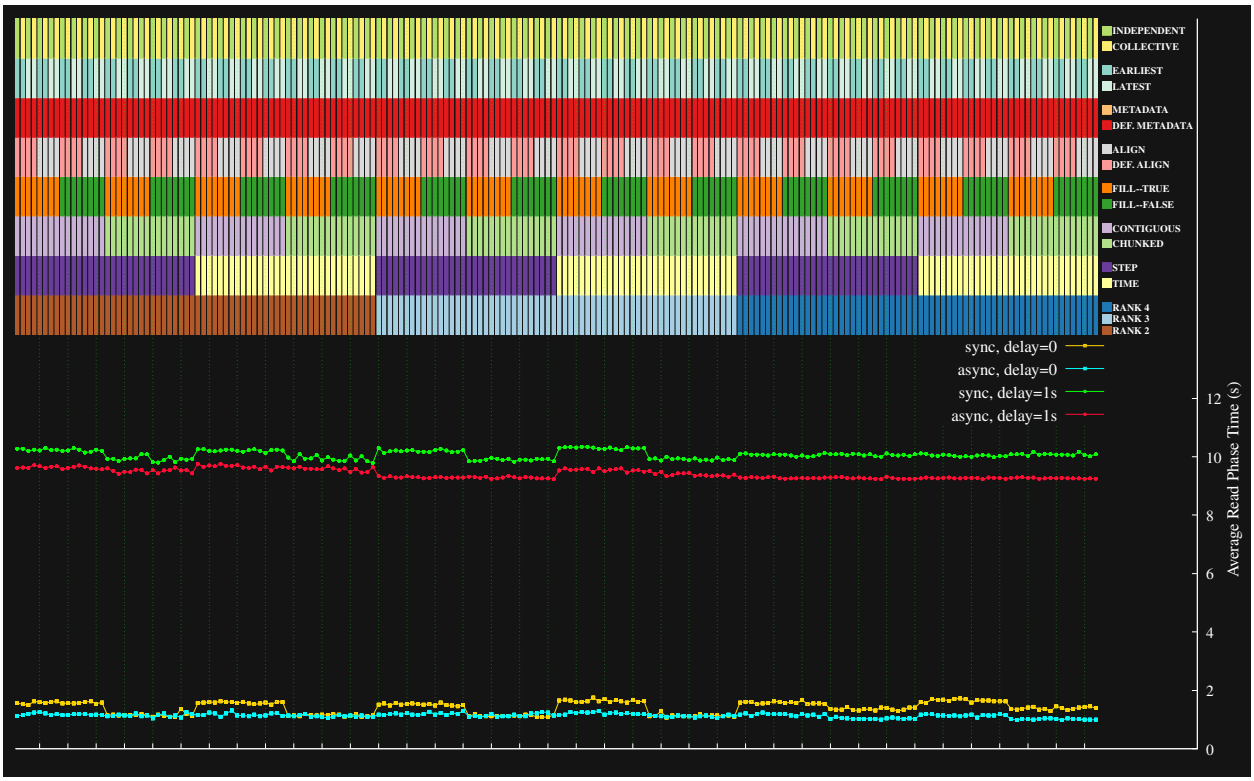

Figure 15 Read time for all elements in the HDFspace set for ANL 144 ranks, with no delay and a one-second delay for a compute phase.

The behavior is expected for contiguous datasets[5]:

In the connector, each chunk is stored in a different DAOS dkey, and data in a single dkey is stored in a single DAOS storage target. Splitting the data into different chunks, therefore, stripes the data across different dkeys and different storage targets. This improves I/O performance by allowing DAOS to read from or write to multiple storage targets at once. The chunk dimensions can be set in the same manner as with native HDF5, with H5Pset_chunk, and application developers are encouraged to consider this I/O parallelism when deciding on the chunk dimensions.

The bandwidth improvement from using different storage targets is so vital that if H5Pset_chunk() is not used, i.e. contiguous datasets, the connector will automatically set a chunk size. The connector, by default, tries to size these chunks to approximately 1 MiB. This automatic chunk target size can be set using the environment variable *HDF5_DAOS_CHUNK_TARGET_SIZE* (in bytes). Setting this variable to 0 disables automatic chunking, and contiguous datasets will stay contiguous (and will therefore only be stored on a single storage target). The connector shapes the automatic chunks to always be contiguous within the full dataset extent, so they are broken up first along the slowest changing dimension, and other dimensions are only split once the chunk size in the slowest changing dimension has been set to 1.

Setting the HDF5_DAOS_CHUNK_TARGET_SIZE to a larger value than the default improves the performance, making it comparable, or better, than chunked datasets, Figure 16-Figure 18.

[5] Internal unpublished report.

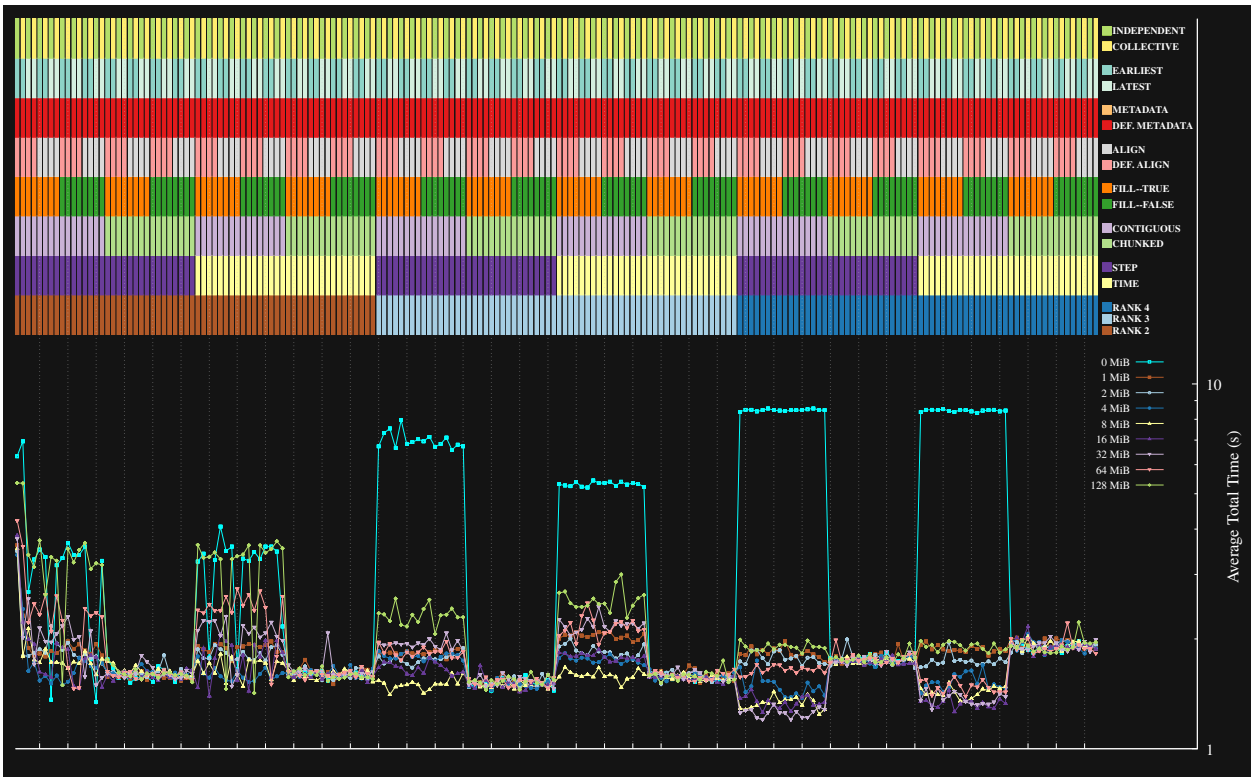

Figure 16 Total time (read and write) for all elements in the HDFspace set for ANL 144 ranks, with no delay for a compute phase and various sizes of automatic chunking. The performance degrades when turning off or making the chunk size too large.

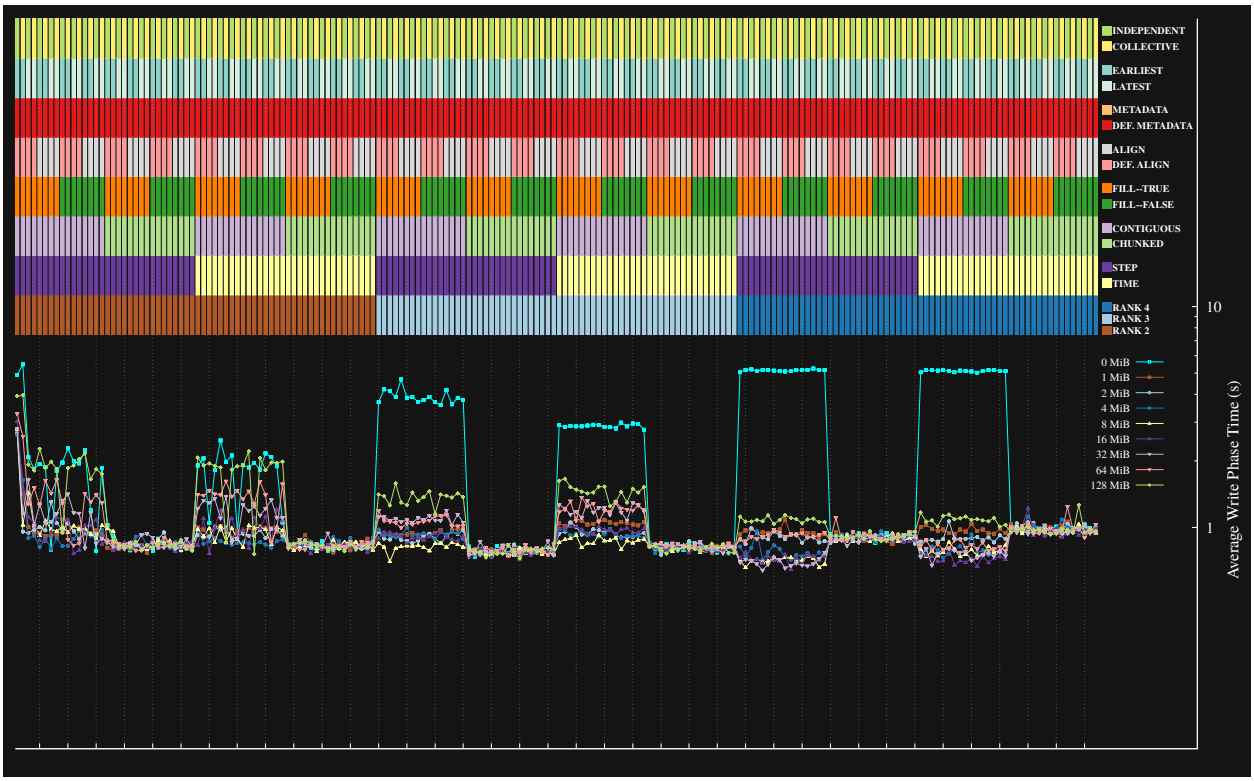

Figure 17 Total write phase time for all elements in the HDFspace set for ANL 144 ranks, with no delay for a compute phase and various sizes of automatic chunking. The performance degrades when turning off or making the chunk size too large.

Figure 18 Total read phase time for all elements in the HDFspace set for ANL 144 ranks, with no delay for a compute phase and various sizes of automatic chunking. The performance degrades when turning off or making the chunk size too large.

4 E3SM

Scorpio is the I/O backend for the *E3SM* application and is an alternative to the complete E3SM application. Scorpio uses *NetCDF*, including the C and Fortran interfaces, as the I/O format library. NetCDF is a set of software libraries to facilitate the creation, access, and sharing of array-oriented scientific data in self-describing machine-independent data formats. The following sections cover the functionality and performance of various ECP VOLs with E3SM.

The LOG VOL is not compatible with NetCDF4 due to the use of *H5*iterate* APIs, which are currently not supported by the LOG VOL. NetCDF4 makes use of *H5Literate*, *H5Aiterate* and *H5DSiterate_scales*. Other unsupported API usages might also be an issue but were not investigated further. Similarly, the async VOL does not support HDF5's iterate APIs and is incompatible with NetCDF4. On the other hand, the DAOS VOL is compatible with NetCDF4 due to HDF5 iterate APIs being supported.

An alternative implementation is to bypass NetCDF4 and directly call HDF5. As part of the ExaIO and Datalib projects, an E3SM option to skip NetCDF4 by calling the HDF5 APIs directly instead was developed and is available in an *E3SM I/O* kernel program on GitHub. All the VOL studies for E3SM use this new implementation.

On Summit, LOG VOL showed improvement over the NATIVE VOL for both f_case_866x72_16p.nc and f_case_48602x72_512p.nc. For the latter case, the NATIVE-VOL did not complete after an hour on Summit, whereas the LOG VOL was completed in a few minutes.

	Output to h0 file (MiB/s)		Output to h1 file (MiB/s)	
	NATIVE-VOL	LOG-VOL	NATIVE-VOL	LOG-VOL
f_case_866x72_16p.nc	942	1892	257	722
f_case_48602x72_512p.nc	N/A*	12695	N/A*	260

Table 3 Summit comparison of E3SM-IO using ten time steps (-r 10). *Exceeded 1 hour on Summit without completing

The async VOL was not tested, as it would use the schema from the NATIVE-VOL implementation, and there is no expectation that it would perform any better.

5 QMCPACK

QMCPACK [6], an open-source Quantum Monte Carlo (QMC) simulation code for "electronic structure calculations of molecular, quasi-2D and solid-state systems" was used to study the various HDF5 VOL

[6] J. Kim, A. Baczewski, T. D. Beaudet, A. Benali, M. C. Bennett, M. A. Berrill, S. Blunt, E. Josu, L. Borda, M. Casula, D. M. Ceperley, S. Chiesa, B. K. Clark, R. C. C. Iii, K. T. Delaney, M. Dewing, M. G. Lopez, Y. Luo, F. D. Malone, M. Richard, A. Mathuriya, J. Mcminis, A. Cody, W. D. Parker, S. D. Pineda, N. A. Romero, B. M. Rubenstein, A. Tillack, J. P. Townsend, N. M. Tubman, QMCPACK: An open-source ab initio Quantum Monte Carlo package for the electronic structure of atoms, molecules, and solids, J. Phys.: Condens. Matter 30 (195901) (2018). arXiv:1802.06922v2, doi:10.1088/1361-648X/aab9c3.

implementations. In addition, the LOG VOL and the Async VOL were investigated using the QMCPACK application.

5.1 QMCPACK performance characteristics using different VOLs

The QMCPACK program '*src/Sandbox/restart.cpp*' is used for the performance benchmark. The program is built by specifying the CMake options

```
-D QMC_BUILD_SANDBOX_ONLY=1 -DENABLE_SOA=0
```

and then "*make restart*" to build the program. The only command-line input is the problem size specified by *-g "n0 n1 n2"*.

```
HDF5 "restart.config.h5" {
GROUP "/" {
   GROUP "state_0" {
      DATASET "block" {
         DATATYPE  H5T_STD_I32LE
         DATASPACE  SCALAR
      }
      DATASET "number_of_walkers" {
         DATATYPE  H5T_STD_I32LE
         DATASPACE  SCALAR
      }
      DATASET "walker_partition" {
         DATATYPE  H5T_STD_I32LE
         DATASPACE  SIMPLE { ( 5 ) / ( 5 ) }
      }
      DATASET "walkers" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 286, 128, 3 ) / ( 286, 128, 3 ) }
      }
   }
   DATASET "version" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
   }
}
}
```

Figure 19: The layout of the restart file restart.config.h5

The HDF5 file structure of *restart.config.h5* is depicted in Figure 19. The dataset *"state_0/walkers"* scales with the number of processes, **Error! Reference source not found.**.

Additionally, *restart.cpp* uses collective as the default I/O mode, and all the datasets in the restart application are contiguous.

Table 4 Restart file sizes.

Number of Processes	File Size (MiB)		
	(2, 2, 2)	(4, 4, 8)	**(8, 8, 16)**
4	0.85	13.51	**108.00**
8	1.70	27.00	**216.00**
16	3.38	54.00	**423.00**
32	6.76	108.00	**864.00**
64	13.51	216.00	**1728.00**
128	**27.00**	**423.00**	**3456.00**

For this study, the size is -g “8 8 8”. On Summit, for 42 ranks, the LOG-VOL could not complete in a reasonable amount of time (< 30 min). According to the LOG VOL developers, “…qmcpack writes datasets in unit size. Instead of writing the dataset in a single block, it writes cell by cell. The result is an extremely large amount of metadata being generated and every H5Dread request must search through it.”

The log-based VOL developers have since substantially improved the LOG-VOL to handle this scenario, Table 5.

Table 5 QMCPACK using HDF5 native VOL and the log-VOL on Summit.

	Total Time (s)	
	NATIVE VOL	LOG VOL
writing random seeds	0.093	0.67
reading random seeds	0.021	0.033
writing walkers	0.14	0.062
reading walkers	0.016	0.058

6 AMReX

The AMReX software framework, which is designed for building massively parallel block-structured adaptive mesh refinement (AMR) applications[7], was used to study the performance of the DAOS VOL. For this study, the AMReX HDF5 benchmark *Tests/HDF5Benchmark/main.cpp* is used with both async enabled and disabled (*AMReX_USE_HDF5_ASYNC=YES/NO*). The only incompatibility with the DAOS VOL was AMReX’s use of *H5Aget_storage_size*, so an alternative solution was provided to the AMReX team.[8] With this change, no other issues with the sync version of DAOS VOL were encountered. The weak scaling average time for writing five plot and particle files, where the number of cells was increased to 128, 256 and 512, are presented in Table 6 using the DAOS-VOL.

Table 6 Weak scaling averaged times for creating five plot and particle dumps.

	Number of ranks		
	32	64	128
Plot files	0.23s	0.32s	0.87s
Particle files	0.19s	0.55s	2.78s

AMReX also has an async implementation associated with the async-VOL. However, the async-VOL HDF5-based implementation assumes the I/O buffers can be modified (or can become out of scope) after the *H5Dwrite_async* is made. This is because, with the *-D ENABLE_WRITE_MEMCPY=1* AMReX option (which is a required option for the async AMReX build), the I/O buffers are copied to an internal buffer in the async-VOL. However, to use the daos-vol async capability, the write buffers are not copied as this option is not available. Hence, when the current AMReX async implementation is used with the

[7] https://github.com/AMReX-Codes/amrex
[8] https://github.com/AMReX-Codes/amrex/pull/2656

DAOS VOL, the program will hang in the *H5ESwait* calls because the I/O buffers are not in scope, and the *H5Dwrite_async* cannot be completed.[9]

7 The HACC Application

HACC (Hardware/Hybrid Accelerated Cosmology Code) is an N-body cosmology code framework where a typical universe simulation demands extreme-scale simulation capabilities. However, a complete simulation of HACC requires terabytes of storage and hundreds of thousands of processors. Consequently, a smaller benchmark I/O code (*GenericIO* by Hal Finkel) was created, which mirrors the I/O calls in HACC without the need to run an entire simulation (http://trac.alcf.anl.gov/projects/genericio). All the "heavyweight" data is handled using POSIX I/O in the benchmark, and the "lightweight" data is handled using collective MPI-IO.

The default I/O strategy in HACC is to have each process write data into a distinct region within a single file using a custom, self-describing file format. Each process writes each variable contiguously within its assigned region. On supercomputers having dedicated I/O nodes (ION), HACC instead uses a single file per ION. The current implementation of HACC provides the option of using MPI I/O (collective or non-collective) or POSIX (non-collective) I/O routines. Additionally, GenericIO implements cyclic redundancy code (CRC) by adding it to the end of the written data array.[10]

In addition to POSIX and MPI-IO, an HDF5 implementation was added to genericIO[11]. HDF5 is a self-describing hierarchal file format, so much of the "metadata" used in the POSIX/MPI-IO implementation of HACC can automatically be handled by HDF5. Thus, using HDF5 greatly reduces the internal bookkeeping and file construction required by HACC compared to using POSIX or MPI-IO.

The use of offsets as pointers to variables (note that these offsets are stored within the HACC file) is eliminated in the HDF5 implementation by using datasets to store variables instead. Currently, there are nine particle variables used in GenericIO programs: *pid, x, y, z, vx, vy, vz, phi*

[9] https://github.com/AMReX-Codes/amrex/issues/2664

[10] Salman Habib, Adrian Pope, Hal Finkel, Nicholas Frontiere, Katrin Heitmann, David Daniel, Patricia Fasel, Vitali Morozov, George Zagaris, Tom Peterka, Venkatram Vishwanath, Zarija Lukic, Saba Sehrish, Wei-keng Liao. "HACC: Simulating Sky Surveys on State-of-the-Art Supercomputing Architectures" New Astronomy, Volume 42, January 2016, Pages 49–65.

[11] https://github.com/brtnfld/genericio.master/tree/hdf5-clean

and *mask*. The HDF5 implementation stores the variables as datasets in the file's root group (/), Figure 20.

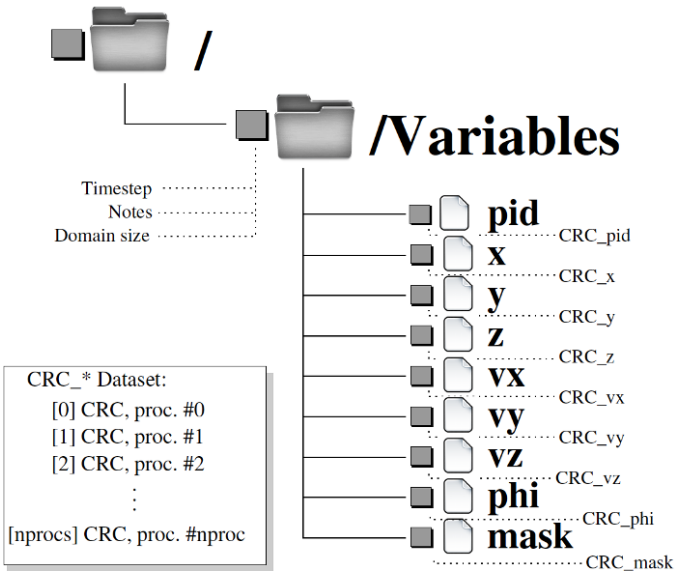

Figure 20 HACC/GenericIO file structure with HDF5.

Attributes at the root level store time step information, such as the time stamp and notes about the simulation parameters. Additional attributes can easily be added if needed. All the datasets are stored under the '*variable'* group.

Associated with each variable's dataset is the cyclic-redundancy check (CRC). The CRC uses the High-Performance CRC64 Library from Argonne National Laboratory. A CRC is computed for each variable, and each processor computes the CRC for the portion of the array residing on that process. The CRC can easily be implemented within the HDF5 file by simply computing a CRC for the array (assuming no partial writes occur) and writing the CRC as a dataset. The reading program can then read the dataset, compute the CRC for the read data and perform a comparison to the values stored in the CRC dataset. The implied restriction is that the layout of the array among the processors is the same for both the writing and the reading of the arrays. Insuring a matching CRC for data written and data being read is essential when creating raw binary files because the file can be transferred to a machine with a different endianness. Therefore, checks must be made to ensure the endianness conversions were implemented correctly. In HDF5, however, the library will convert and verify the byte order automatically, so the use of the CRC may no longer be necessary. Nevertheless, the current implementation includes the CDC dataset schema to remain comparable with the non-HDF5 options.

Table 7 Time to write all nine genericIO variables using DAOS.

	Number of Ranks		
	32	64	128
MPI-IO	4.59s	7.16s	12.41s
HDF5	4.06s	7.17s	13.88s
HDF5, COMPOUND DATATYPE	7.93s	8.07s	12.49s

The subsequent investigation was to test genericIO using the log-VOL. During the experiment, one issue was the CRC values computed by genericIO for each dataset. Namely, only one rank calculates and writes (*H5Dwrite*) the CRC values, while the other ranks still call H5Dwrite but pass a NULL dataspace and write buffer. However, the log-vol would return an error when given a NULL write buffer. The issue has been addressed[12], and log-VOL can now handle a NULL data space.

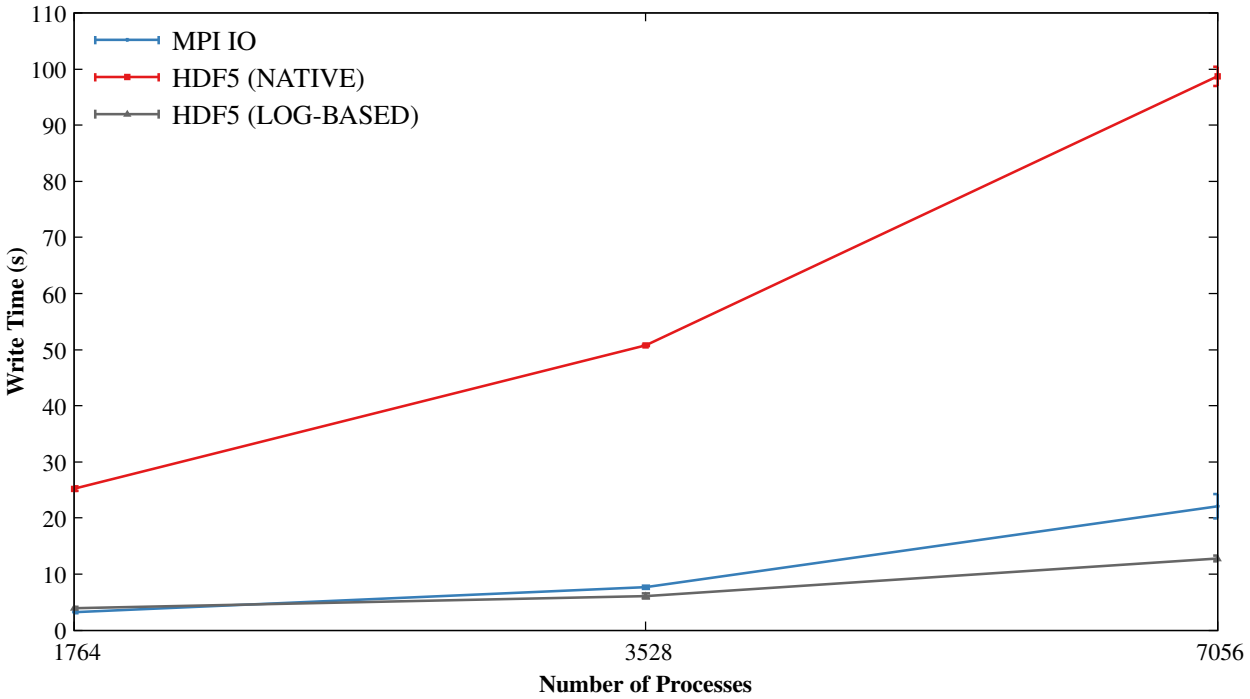

Figure 21 Compares the time to write all nine genericIO variables using MPI-IO, HDF5 with the native-VOL, and HDF5 with the log-based-VOL on Summit.

8 Next steps

A complete investigation of some VOLs requires further runs that were not completed due to working with VOL developers in reporting and addressing VOL bugs and slow turn-around times for some systems' batch jobs. For the DAOS-VOL, system technical issues are currently preventing benchmarking of the applications. The following tasks remain in completing the final report,

- Work with VOL developers in addressing Spack test failures.

[12] https://github.com/DataLib-ECP/vol-log-based/commit/e0bddaa105ddd366350cfb7624c5eb481acf633e

- Complete running the hdf5io-test studies
    - Run hdf5-iotest with the ASYNC-VOL multiple times to determine the performance variance
    - Run hdf5-iotest with the ASYNC-VOL with a compute phase delay.
- E3SM **(**w/HDF5)
    - Run with DAOS-VOL
- QMCPack
    - Run with DAOS-VOL

9 Appendix

9.1 Examples of passing Spack builds needing additional Spack package parameters

- Perlmutter
  - spack install hdf5-vol-async %aocc@3.1.0 (with modules PrgEnv-aocc/8.2.0, aocc/3.1.0)
  - spack install hdf5-vol-log %aocc@3.1.0
  - spack install hdf5-vol-external-passthrough %aocc@3.1.0
  - spack install hdf5 %cce@13.0.1 ^ncurses@6.1%gcc@11.2.0
  - spack install hdf5-vol-async %cce@13.0.1 ^ncurses@6.1%gcc@11.2.0
  - spack install hdf5-vol-log %cce@13.0.1 ^ncurses@6.1%gcc@11.2.0
  - spack install hdf5-vol-external-passthrough %cce@13.0.1 ^ncurses@6.1%gcc@11.2.0
- Spock
  - spack install hdf5-vol-async %cce@13.0.1 ^openssl@1.1.1n%gcc@11.2.0
  - spack install hdf5-vol-log %cce@13.0.1 ^openssl@1.1.1n%gcc@11.2.0 ^openmpi@3.1.6%gcc@11.2.0
  - spack install hdf5-vol-external-passthrough %cce@13.0.1 ^openssl@1.1.1n%gcc@11.2.0 ^openmpi@3.1.6%gcc@11.2.0
- Summit
  - spack install hdf5-vol-async %cce@13.0.1 ^openssl@1.1.1n%gcc@11.2.0
  - spack install hdf5-vol-log %cce@13.0.1 ^openssl@1.1.1n%gcc@11.2.0 ^openmpi@3.1.6%gcc@11.2.0
  - spack install hdf5-vol-external-passthrough %cce@13.0.1 ^openssl@1.1.1n%gcc@11.2.0 ^openmpi@3.1.6%gcc@11.2.0

9.2 GitHub Action (Ubuntu)

The following test uses **hyoklee/vol-tests@develop** and **HDFGroup/hdf5@develop** without using Spack.

VOL	No. of Failures	Note
adios2	212 from ADIOS2 CTest CTest: 0	84% tests passed, 212 tests failed out of 1327 (Last test: 2021/11/08) 4222 All passed. (Last test: 2021/11/09)
async	CTest: 0 bin/h5vl_test: 0	All passed. (Last test: 2021/11/08) ~~See GitHub issue.~~ (Last test: 2021/11/08)
cache	CTest: 0	All passed.
pass-through	CTest: 0	All passed.
log-based	CTest: 4	FAIL: h5_crtgrpd FAIL: h5_group FAIL: h5_select FAIL: h5_subset (Last test: 2021/11/08)

9.3 Jelly (CentOS)

The following test result is from Jelly using gcc 7.1.0 and Spack.

VOL	No. of Failures	Note
async	9	3 - h5vl_test_datatype (Failed) 4 - h5vl_test_file (Failed) 5 - h5vl_test_group (Failed) 7 - h5vl_test_misc (Failed) 10 - h5_test_testhdf5 (SEGFAULT) 12 - h5_partest_t_bigio (Failed) 13 - h5_partest_t_pshutdown (Failed) 14 - h5_partest_t_shapesame (Failed) 15 - h5_partest_testphdf5 (Failed)
aync+mpicc	9	2 - h5vl_test_dataset (Failed) 6 - h5vl_test_link (Failed) 8 - h5vl_test_object (Failed) 9 - h5vl_test_async (Failed) 10 - h5_test_testhdf5 (SEGFAULT) 12 - h5_partest_t_bigio (Failed) 13 - h5_partest_t_pshutdown (Failed) 14 - h5_partest_t_shapesame (Failed) 15 - h5_partest_testphdf5 (Failed)
aync+hpc-io/hdf5	8	5 - h5vl_test_group (Failed) 7 - h5vl_test_misc (Failed) 9 - h5vl_test_async (Failed) 10 - h5_test_testhdf5 (SEGFAULT) 12 - h5_partest_t_bigio (Failed) 13 - h5_partest_t_pshutdown (Failed) 14 - h5_partest_t_shapesame (Failed) 15 - h5_partest_testphdf5 (Failed)
aync+hpc-io/hdf5+mpicc	10	2 - h5vl_test_dataset (Failed) 4 - h5vl_test_file (Failed) 5 - h5vl_test_group (Failed) 6 - h5vl_test_link (SEGFAULT) 8 - h5vl_test_object (Failed) 10 - h5_test_testhdf5 (SEGFAULT) 12 - h5_partest_t_bigio (Failed) 13 - h5_partest_t_pshutdown (Failed) 14 - h5_partest_t_shapesame (Failed) 15 - h5_partest_testphdf5 (Failed)
cache	3	1 - h5vl_test_attribute (Failed) 9 - h5vl_test_async (Failed) 14 - h5_partest_t_shapesame (Timeout)

cache+mpicc	4	7 - h5vl_test_misc (Failed) 8 - h5vl_test_object (Failed) 9 - h5vl_test_async (Failed) 14 - h5_partest_t_shapesame (Timeout)
cache+hpc-io/hdf5	3	2 - h5vl_test_dataset (Failed) 8 - h5vl_test_object (Failed) 14 - h5_partest_t_shapesame (Timeout)
cache+hpc-io/hdf5+mpicc	4	4 - h5vl_test_file (Failed) 6 - h5vl_test_link (Failed) 8 - h5vl_test_object (Failed) 14 - h5_partest_t_shapesame (Timeout)
external-pass-through	n/a	can't compile as of 2021/05/25.

9.4 Cori

Cori has gcc/intel/cce (cray) compilers. Use **spack external** for cmake and perl.

VOL	No. of Failures	Note
cache+hpc-io/hdf5	7	4 - h5vl_test_file (Failed) 5 - h5vl_test_group (Failed) 7 - h5vl_test_misc (Failed) 9 - h5vl_test_async (Failed) 12 - h5_partest_t_bigio (Timeout) 14 - h5_partest_t_shapesame (Timeout) 15 - h5_partest_testphdf5 (Timeout)

9.5 Nene (MacOS Big Sur)

VOL	No. of Failures	Note
adios2	n/a	adios2 unit test fails. vol-tests hang during test.
async+hpc-io	9	1 - h5vl_test_attribute (Failed) 3 - h5vl_test_datatype (Failed) 4 - h5vl_test_file (Failed) 5 - h5vl_test_group (Failed) 6 - h5vl_test_link (Failed) 7 - h5vl_test_misc (Failed) 10 - h5_test_testhdf5 (SEGFAULT) 12 - h5_partest_t_bigio (Failed) 13 - h5_partest_t_pshutdown (Failed) 14 - h5_partest_t_shapesame (Failed) 15 - h5_partest_testphdf5 (Failed)
cache+hpc-io	n/a	>> 46 Undefined symbols for architecture x86_64: 47 "_H5P_LST_DATASET_XFER_ID_g", referenced from: 48 _H5Dmmap_remap in cache_new_h5api.c.o