Cache VOL: Efficient parallel I/O through caching data on fast storage

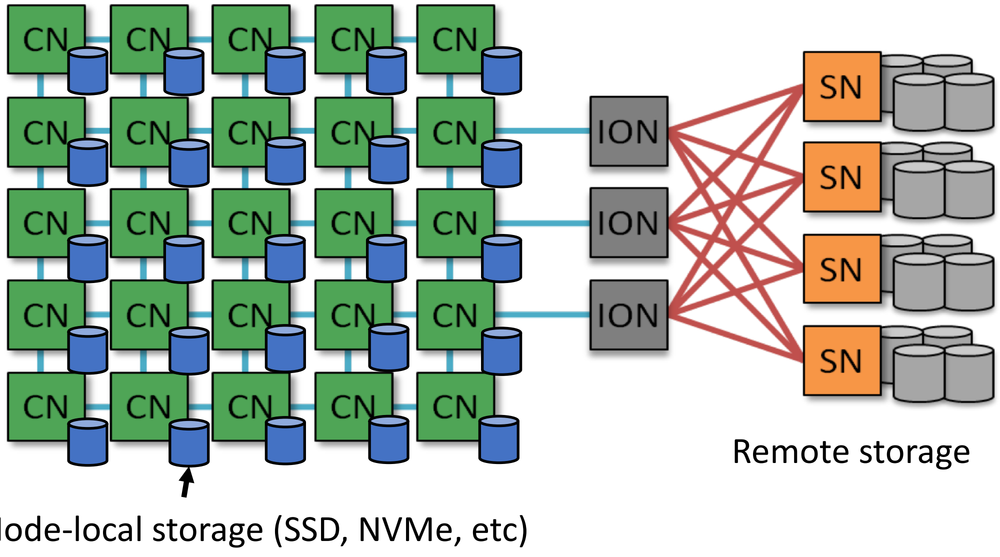
Huihuo Zheng (ANL), Venkatram Vishwanath (ANL), Quincey Koziol (LBL),
Houjun Tang (LBL), Suren Byna (LBL)

10/13/2021

huihuo.zheng@anl.gov

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

Integrating node-local storage into parallel I/O



Typical HPC storage hierarchy: node-local storage (NLS) + global parallel file system (PFS)

Theta @ ALCF: Lustre + SSD (128 GB / node),
ThetaGPU (DGX-3) @ ALCF: NVMe (15.4 TB / node)
Summit @ OLCF: GPFS + NVMe (1.6 TB / node)

Node-local storage

- Local to the compute node, does not need to go through the network (stable)
- Larger aggregate bandwidth compared to the parallel file systems

Theta (w) – Lustre: 200 GB/s, SSD: 3TB/s
Summit (w) – GPFS: 2.5 TB/s, NVMe: 9.7 TB/s

Cache VOL

- Caching / staging data on node-local storage
- Data movement in the background (*through Async VOL*)
- All complexity is hidden from the users

Applications that will benefit from Cache VOL

- Heavy checking pointing I/O
- Intensive repetitive read

BERKELEY LAB Lawrence Berkeley National Laboratory The HDF Group Argonne NATIONAL LABORATORY

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

ECP EXASCALE COMPUTING PROJECT

How to build Cache VOL

1) Building HDF5 (post_open_fix branch)

```
$ git clone -b post_open_fix https://github.com/hpc-io/hdf5.git; cd hdf5;
$ ./autogen.sh
$ ./configure --prefix=${HDF5_ROOT} --enable-parallel --enable-threadsafe --enable-unsupported CC=mpicc
$ make all install –j4
```

2) Building argobots and Async VOL

```
$ git clone --recursive https://github.com/hpc-io/vol-async.git; cd vol-async
$ cd argobots; ./autogen.sh; CC=gcc; ./configure --prefix=${ABT_DIR}
$ make all install
$ cd ../src; make all; # Remember to edit the HDF5_DIR and ABT_DIR in the Makefile
$ cp *.h ${HDF5_VOL_DIR}/include/
$ cp lib* ${HDF5_VOL_DIR}/lib/
```

3) Building Cache VOL

```
$ git clone https://github.com/hpc-io/vol-cache.git; cd vol-cache/src;
$ make all install # libh5cache_vol.so will be installed in ${HDF5_VOL_DIR}/lib
```

How to use Cache VOL

1) Setting VOL connectors

```
export HDF5_PLUGIN_PATH=$HDF5_VOL_DIR/lib
export HDF5_VOL_CONNECTOR="cache_ext config=SSD.cfg;under_vol=512;under_info={under_vol=0;under_info={}}"
export LD_LIBRARY_PATH=$HDF5_PLUGIN_PATH :$LD_LIBRARY_PATH
```

```
#content 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) Inserting compute work between write/read and close.

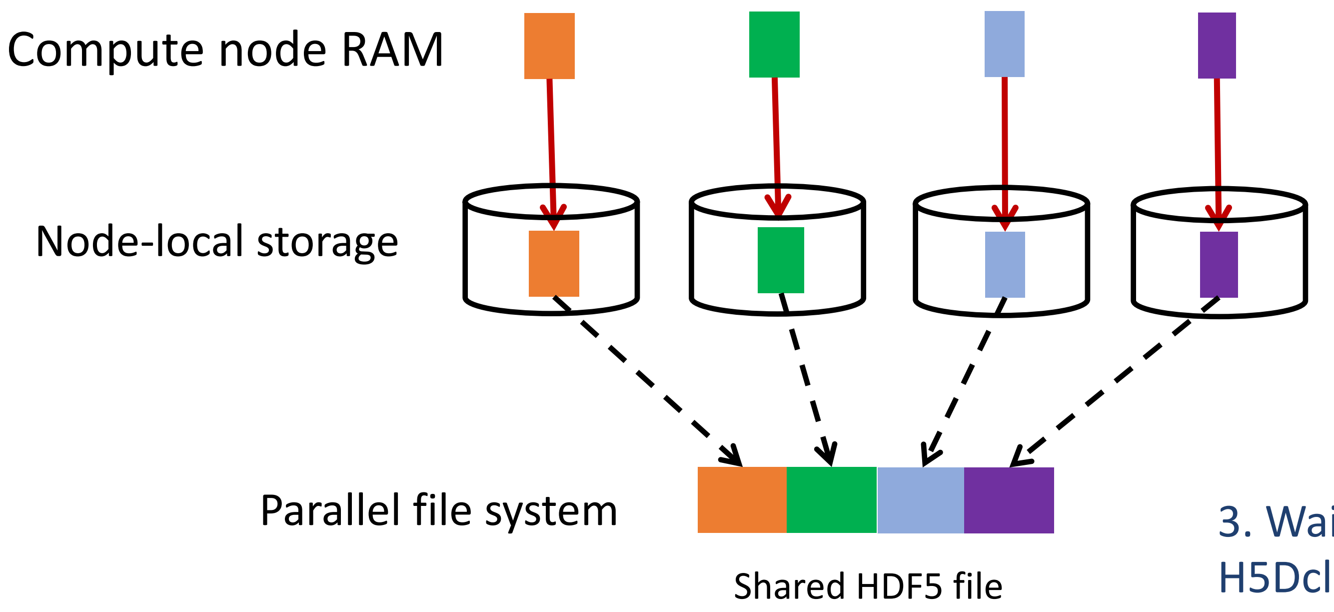
```
MPI_Init_thread(…, MPI_THREAD_MULTIPLE…)
H5Dopen()
H5Dread()
…# compute
H5Dclose()
```

```
MPI_Init_thread(…, MPI_THREAD_MULTIPLE…)
H5Dcreate()
H5Dwrite()
… # compute
H5Dclose()
```

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

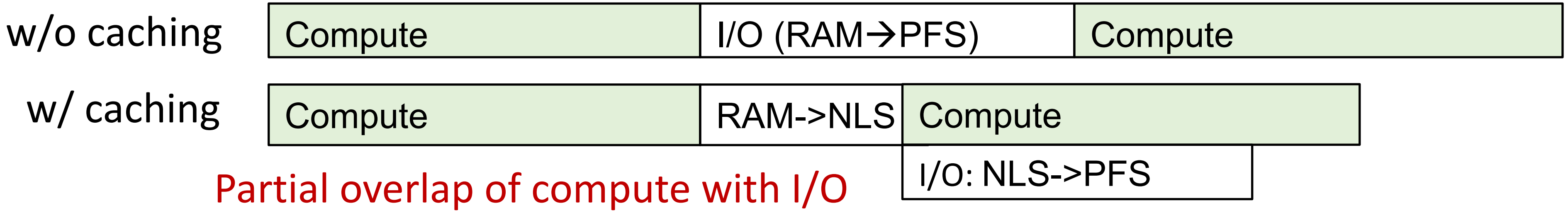
Parallel Write (H5Dwrite) w/ node-local storage

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 the *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

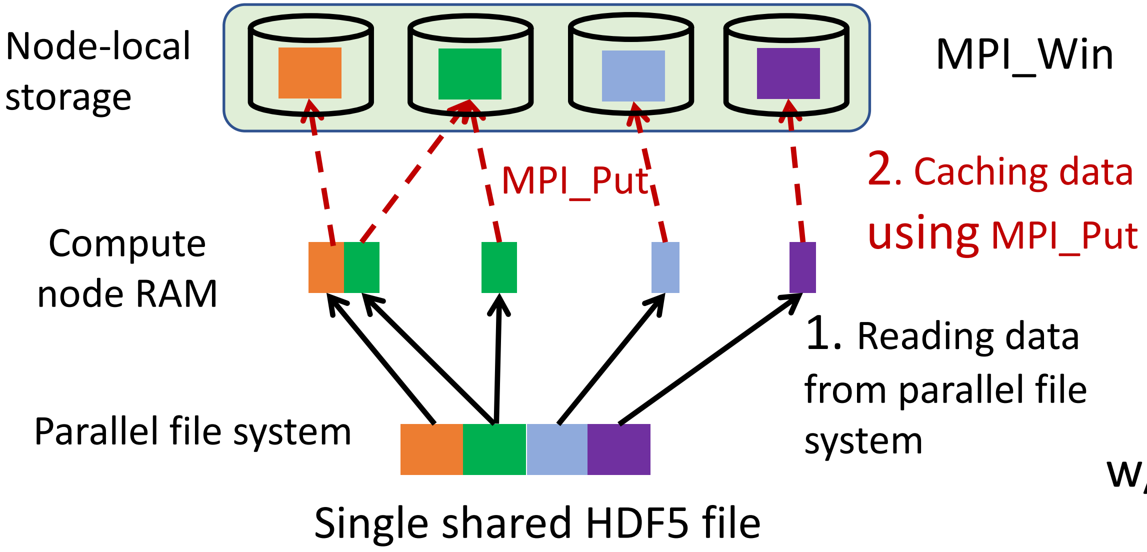
Details are hidden from the application developers.

Parallel Read (H5Dread) w/ node-local storage

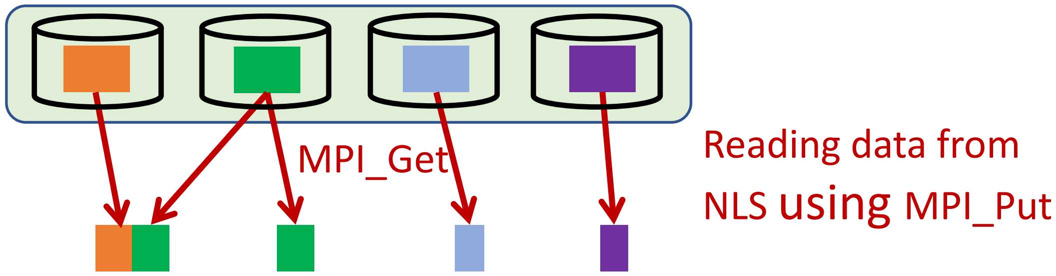
Create memory mapped files and attached them to a MPI_Win for one-sided remote access

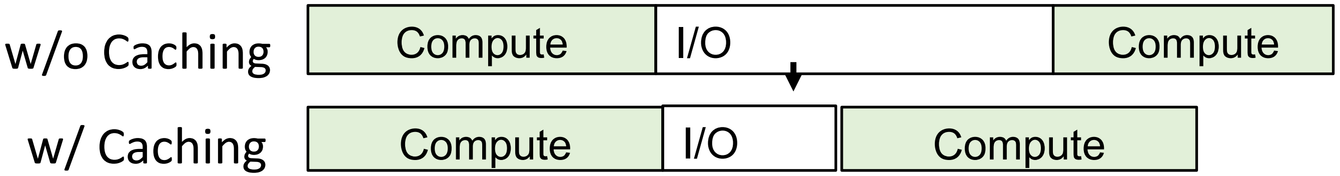
One-sided communication for accessing remote node storage.

- 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)



First time reading the data

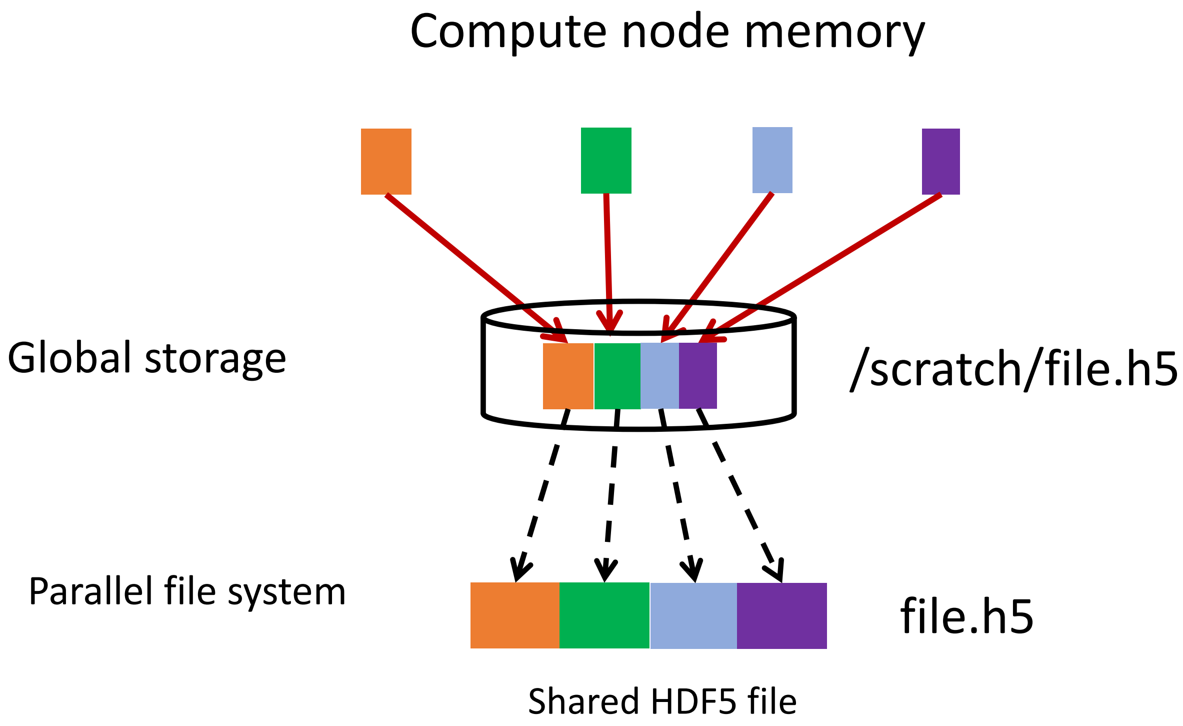




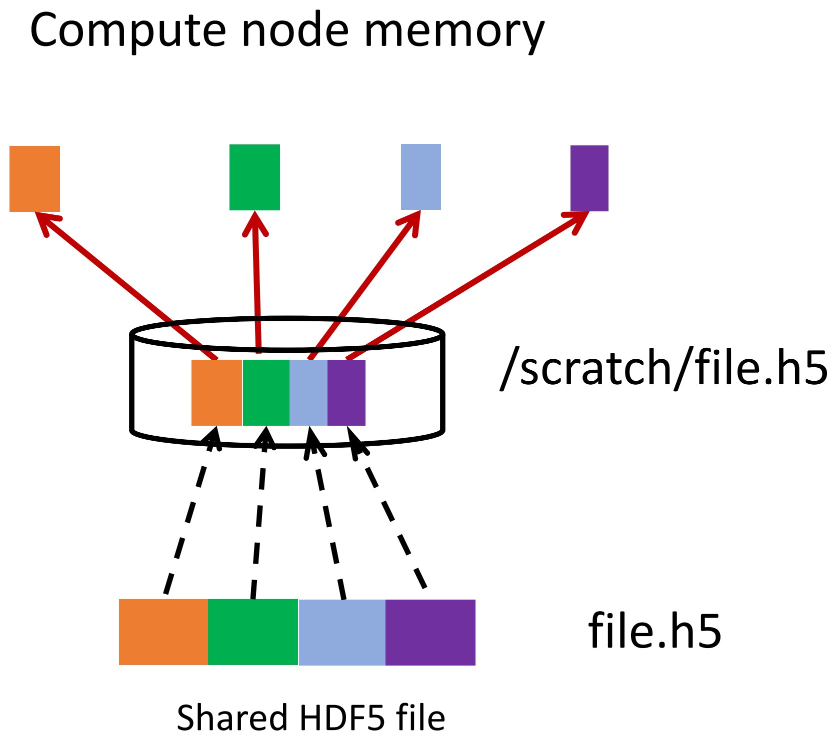
Reading the data directly from node-local storage

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

Caching on global storage



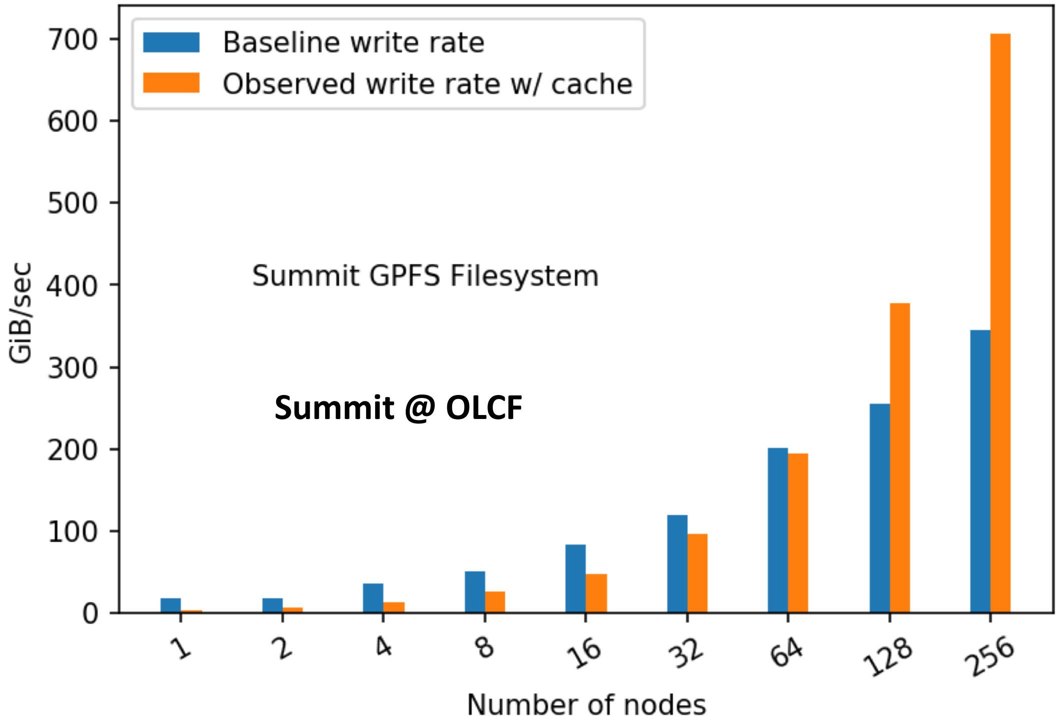
Parallel write

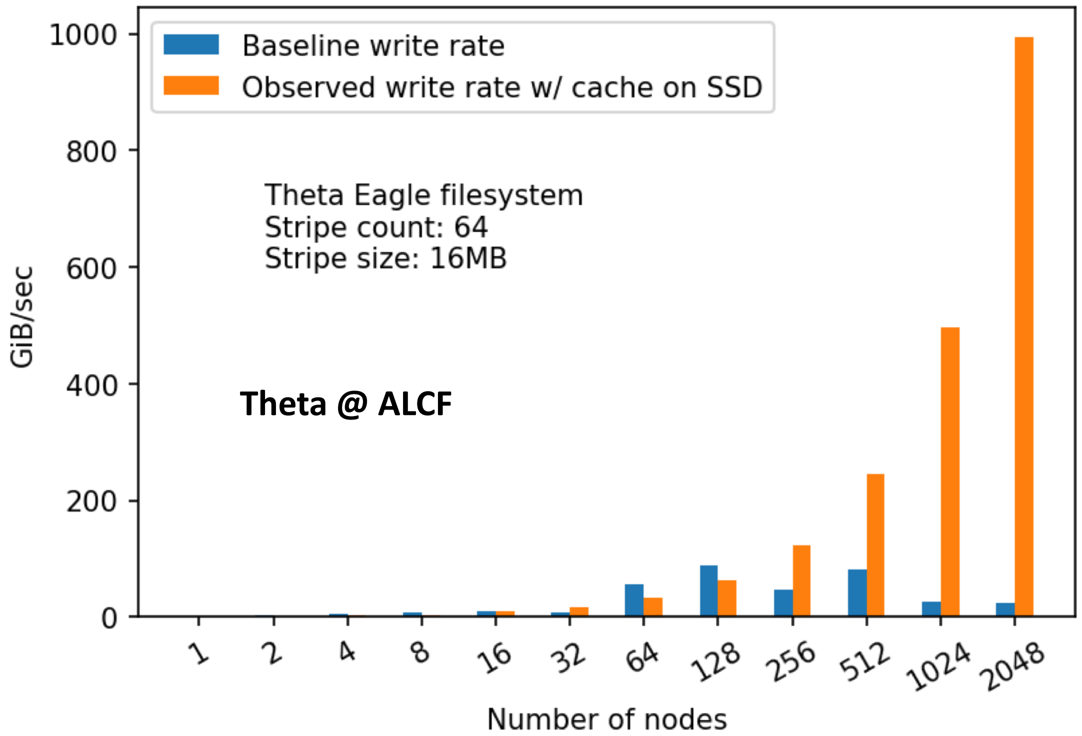


Parallel read

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

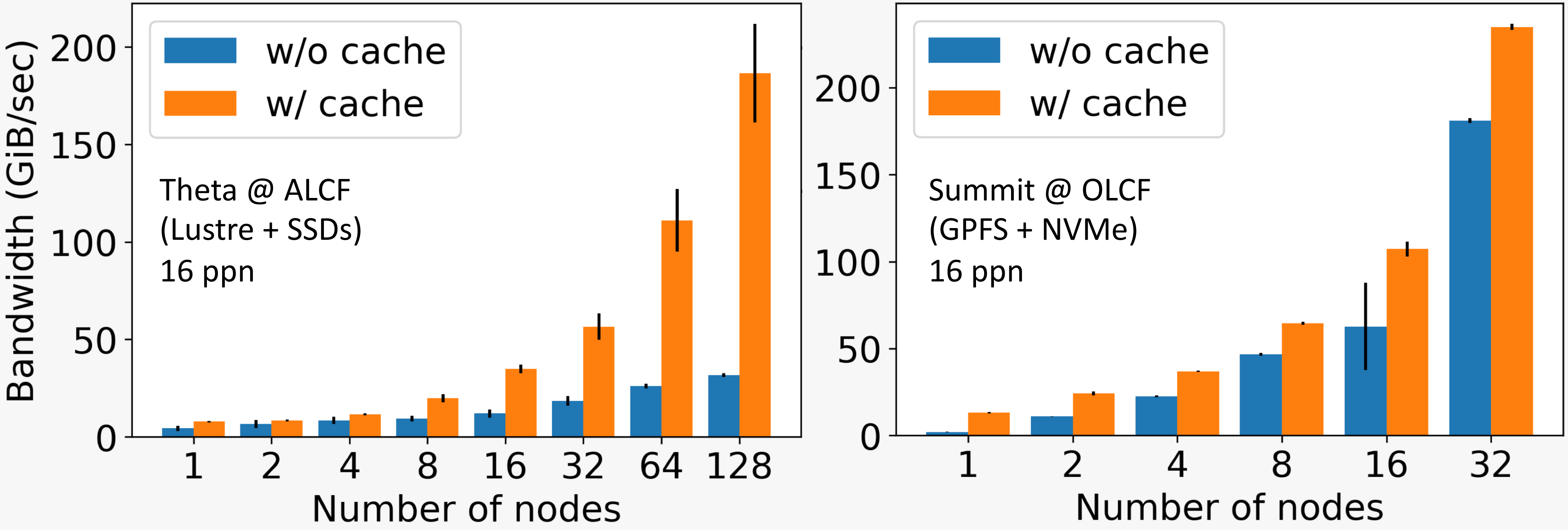
Performance evaluation w/ h5bench (VPIC IO)





Parallel write performance: each process writes 16MB of data to a shared HDF5 file. With caching, the write bandwidth scale linearly with a larger aggregate bandwidth surpassing the Lustre / GPFS write bandwidth.

Parallel read for deep learning applications



Parallel read performance. The bandwidth is averaged over four iterations. At each step, the application reads a random batch (32) of samples (224x224x3) with shuffling. The application reads through the entire dataset in one iteration.

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

Conclusion

- Node-local storage caching / staging achieves faster and more scalable I/O over direct I/O to parallel file system.
- VOL implementation makes it easy to integrate the framework into existing HPC applications and python workloads with minimal code change.

Ongoing work

- Integrate Cache VOL to HPC applications and deep learning applications.

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.