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

Huihuo Zheng[1], Venkatram Vishwanath[1], Quincey Koziol[2], Houjun Tang[2], John Ravi[2], John Mainzer[3], and Suren Byna[2]

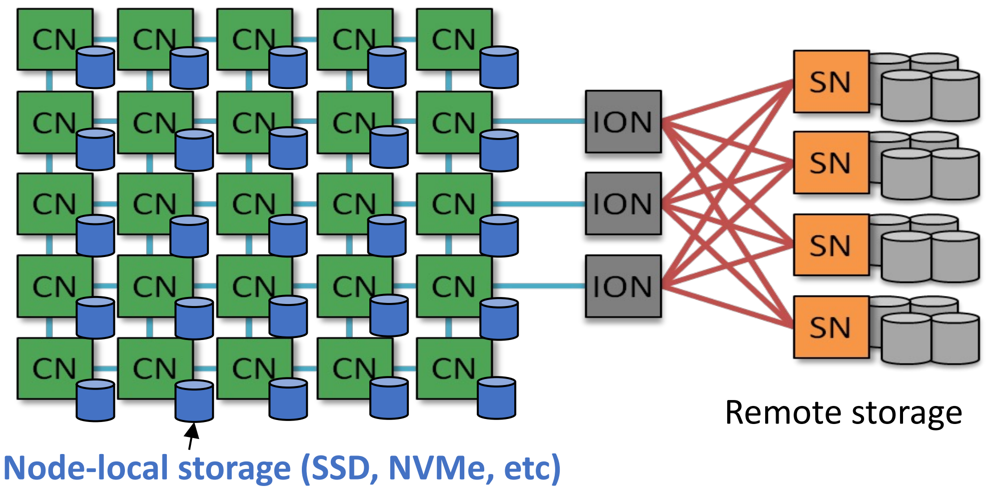
[1]Argonne National Laboratory, [2]Berkeley National Laboratory, [3]The HDF Group

huihuo.zheng@anl.gov

Nov 16th, 2022

Transparently integrating node-local storage into parallel I/O workflows

Typical HPC storage hierarchy



Polaris @ ALCF: NVMe (7.68 TB / node)
Summit @ OLCF: GPFS + NVMe (1.6 TB / node)
Fugaku @ RIKEN: Lustre + NVMe (1.6 TB / 16 nodes)
Frontier @ OLCF: Lustre + NVMe (37PB total)

Node-local storage

- Local & private; no contention or job interference
  → more stable and scalable IO;
- Faster (larger aggregate bandwidth).
  Theta (w) – Lustre: 650 GB/s, SSD: 3TB/s
  Summit (w) – GPFS: 2.5 TB/s, NVMe: 9.7 TB/s

Challenges

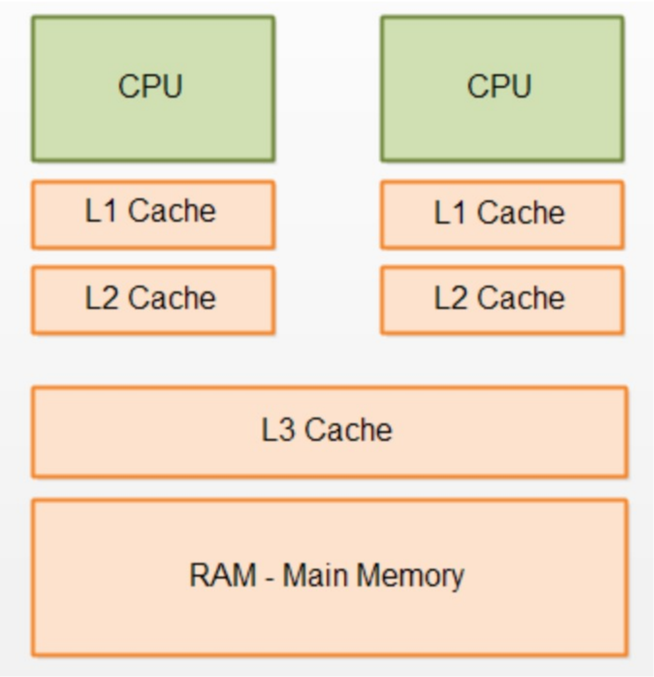
- No global namespace;
- Accessible only during job running;
- Limited system software support.

Cache VOL: using node-local storage as a cache

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

Using caching to improve data access

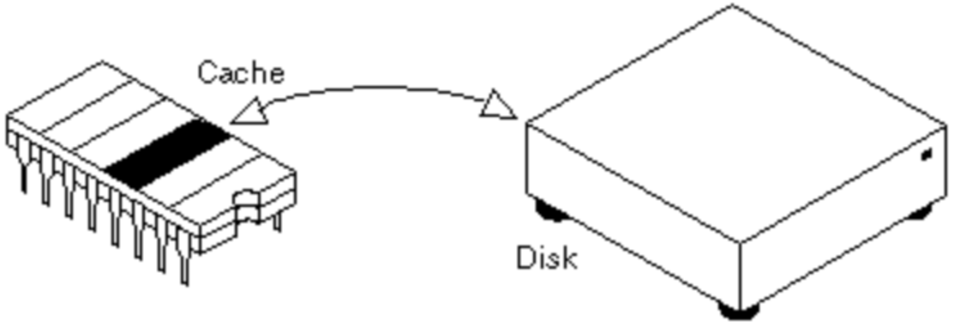
Caching in memory hierarchy



~TB/s

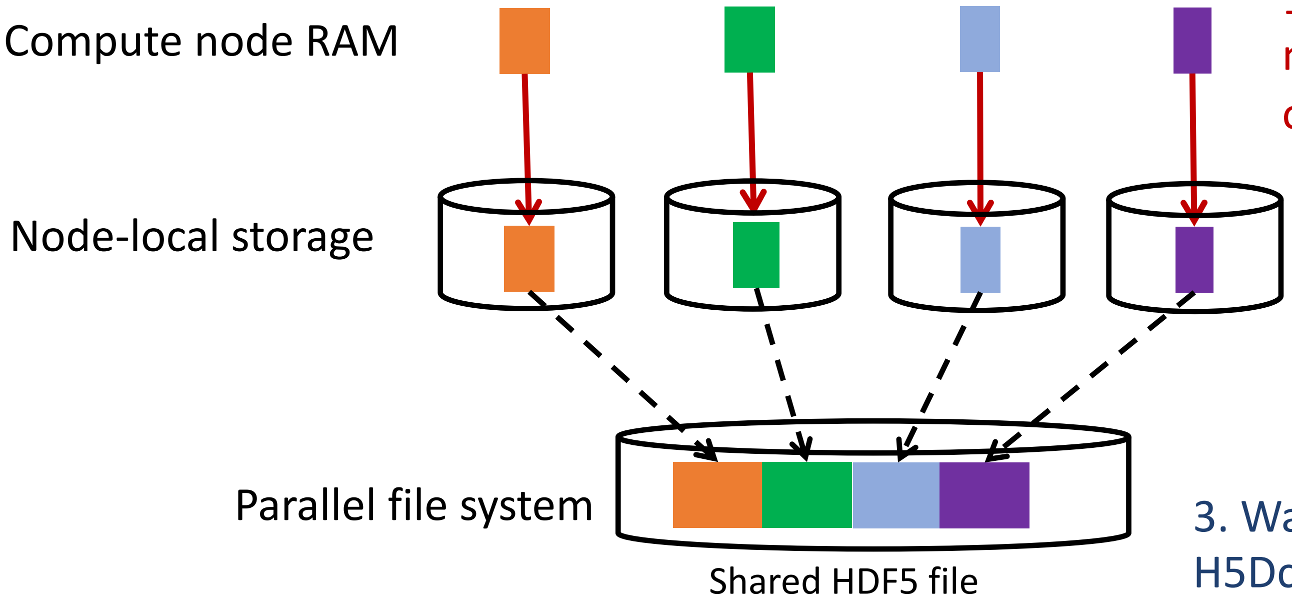
~100GB/s

Page caching in I/O



- **Write:** the data is copied from the user's buffer into the page cache in DRAM. The actual writes to disk are done later.
- **Read:** data is read directly from the page cache in DRAM if it is cached there.

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 through *Async VOL* stacked below the Cache VOL

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

w/o caching	Compute	I/O (RAM→PFS)	Compute
w/ caching	Compute	RAM->NLS	Compute
			I/O: NLS->PFS

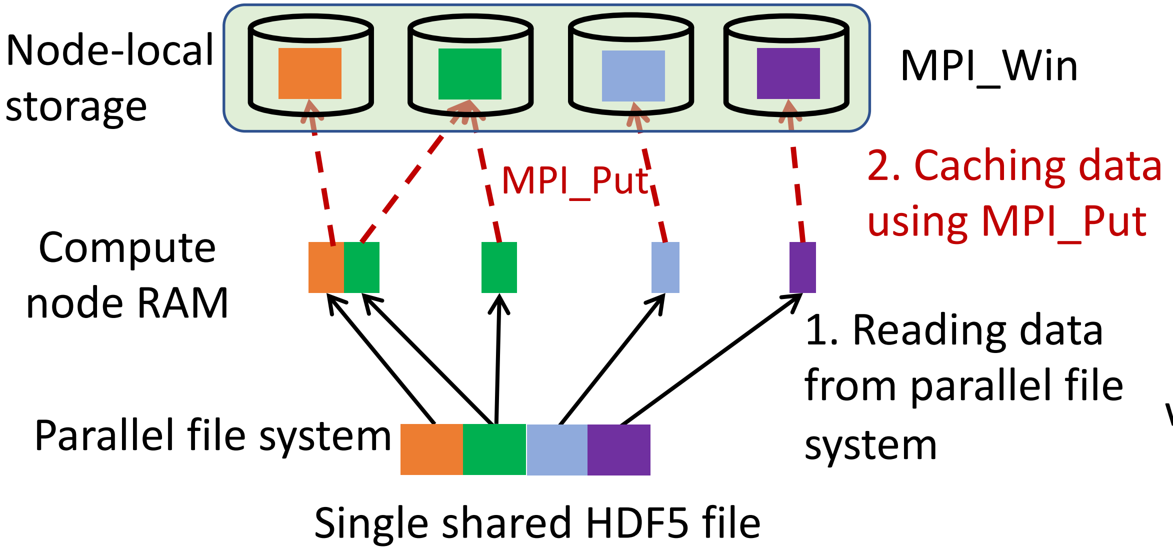
Partial overlap of compute with I/O

Details are hidden from the application developers.

Parallel Read (H5Dread)

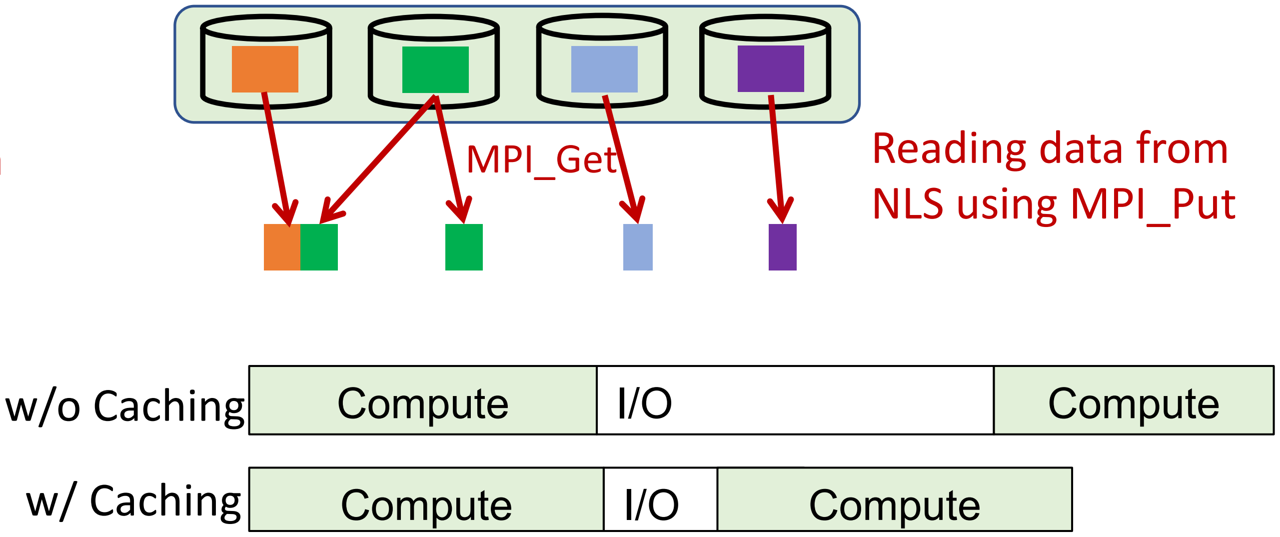
Targeting workloads with repeatedly reading the same dataset multiple times.

Create memory mapped files and attached virtual memory pointer to an MPI window



First time reading the data

Memory-mapped shared file system
- Each process exposes a portion of its storage to other processes through MPI Window
- Other processes read from or write to this shared storage space through MPI_Put, MPI_Get.



Reading the data directly from node-local storage

Easy to adopt in the applications

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

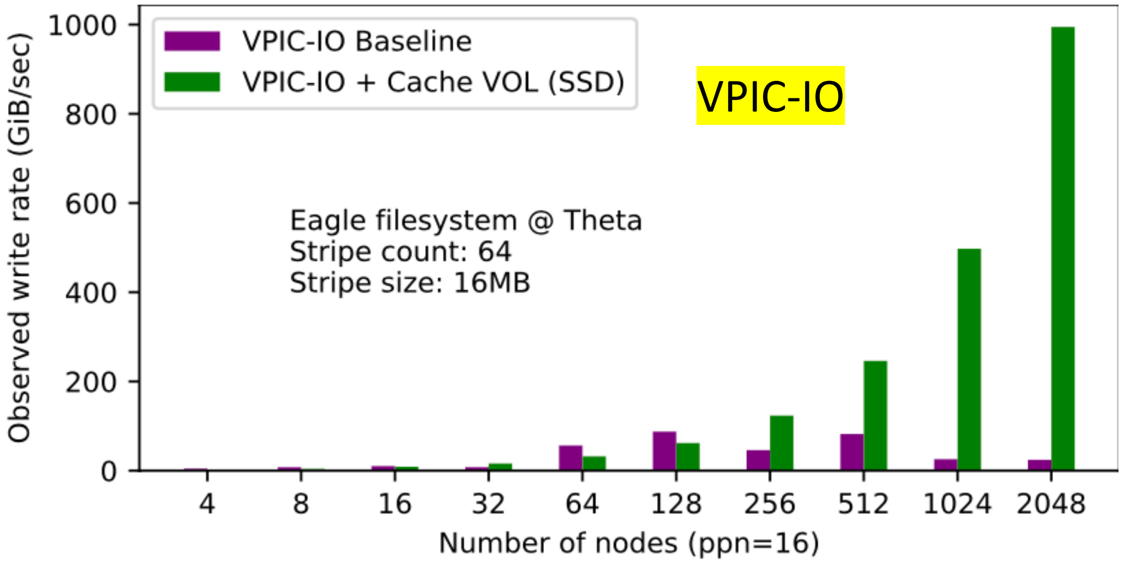
Opt. 1 Through global environment variables (**HDF5_CACHE_RD / HDF5_CACHE_WR [yes|no]**)

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

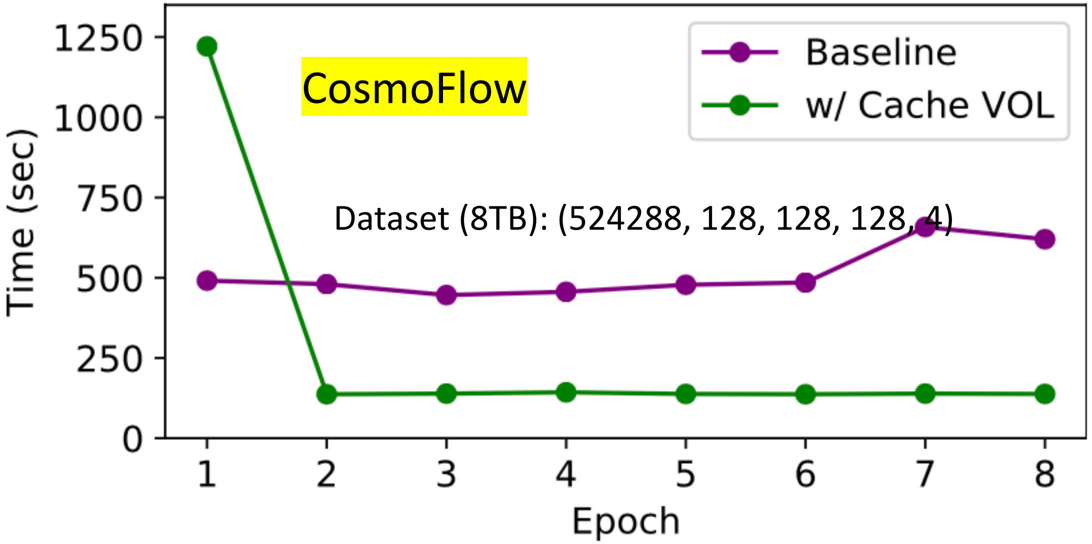
3) Initializing MPI with MPI_Init_thread(…, MPI_THREAD_MULTIPLE…)

4) In some cases, rearranging the function calls to allow the overlap of computation with data migration (check our github repo for the examples and best practices)

Performance evaluation (VPIC-IO & CosmoFlow)



Observed VPIC-IO write rate on Theta and (Right) Summit. The number of time steps is 20. The write rate reported here is the average over the 20 timesteps. The emulated time is 200 seconds per time step on Theta. Each process writes check-points data (32MB x 8) to a shared file at each timestep



Improvement of training throughput by caching data on the node-local storage for CosmoFlow. The training were done on 16 DGX nodes with 128 Nvidia A100 GPUs on ThetaGPU. Each training step randomly read a minibatch of samples from a shared HDF5 file

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

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.