CSE 5449: Intermediate Studies in Scientific Data Management

Lecture 14: How to apply I/O tuning

Dr. Suren Byna

The Ohio State University

E-mail: byna.1@osu.edu

https://sbyna.github.io

02/23/2023

Today's class

- Any questions?
- Class presentation topic
- Class project progress
- Today's class –
  - How to apply I/O tuning?

Factors that impact parallel I/O performance

Applications

- Number of MPI ranks
- Number of I/O requests
- Size of I/O requests
- Number of files
- Number of metadata calls
  - File open and close requests
- Number of seek operations
- Contiguous / non-contiguous requests
  - Number of seeks
- Alignment of I/O request with
  - File block
  - Sub-files
- Shared file or multiple files
- …

High-level I/O library

- Metadata operations for self-describing property
- Location of metadata
- How many processes are participating in metadata or data operations
- Alignment in file offsets
- Hyperslab selections
  - contiguous / non-contiguous?
  - complex hyperslabs construction cost
- Chunking
  - Chunk size
  - Number of chunks
- Sub-files
  - How many? How's the data aggregated?
- Compression used or not?
  - What's the compression / decompression cost?
  - Where is compression / decompression executed?
- Does a file need to be exact size of data or can it have some gaps?
- Cache metadata or not?

MPI-IO

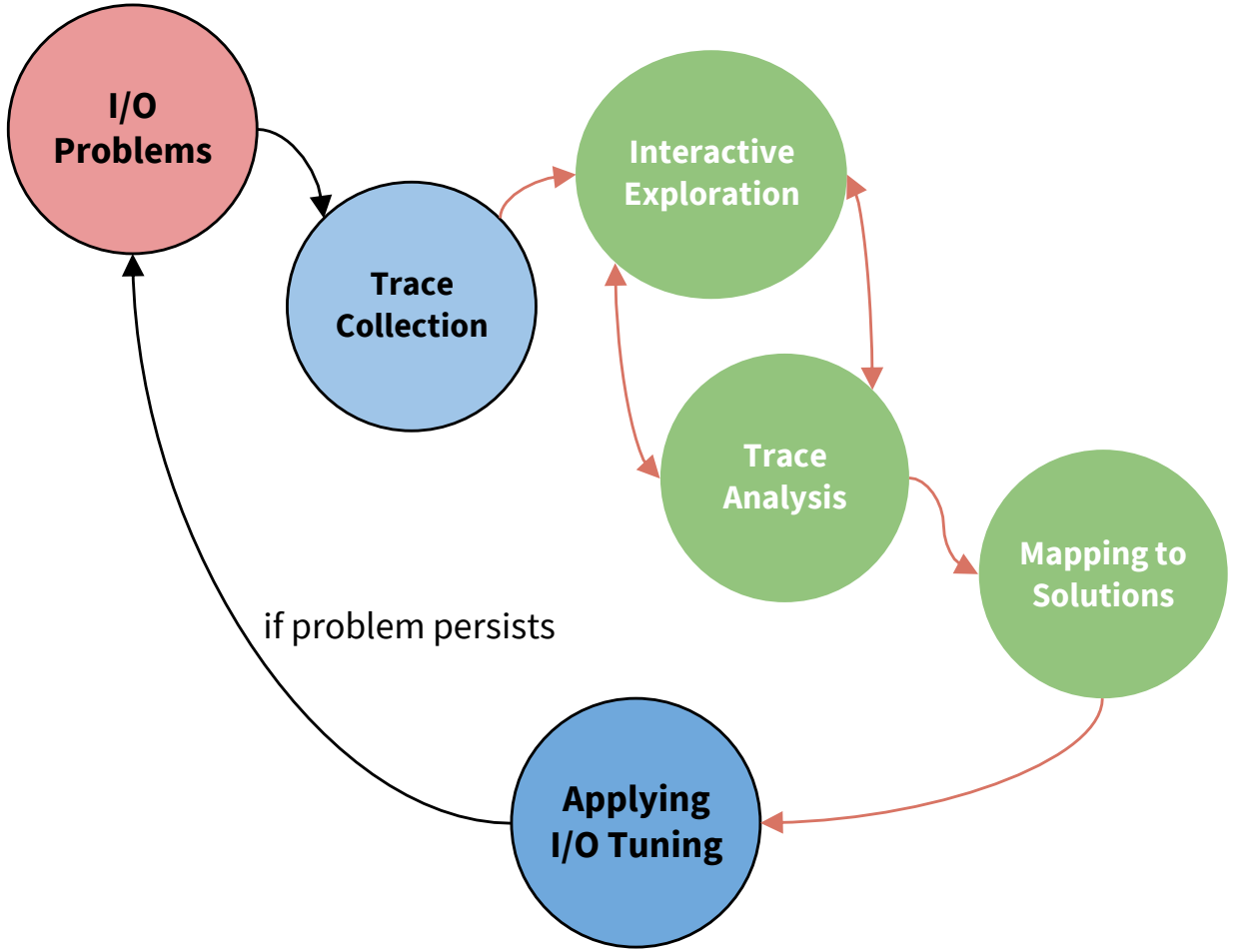
- Contiguous / non-contiguous accesses
- Number of I/O requests
- Size of I/O requests
- POSIX consistency semantics
- Synchronous / Asynchronous I/O calls
- Collective or independent
- If collective:
  - Number of aggregators
  - Aggregator placement
  - Aggregation buffer size
  - Aggregator to file system mapping – network connections and block sizes

File systems

- Number of storage servers
- Number of metadata servers
- Number of storage targets (stripe count)
- Block size on storage server
- Page size on storage target
- Amount of contiguous data stored on a storage target (stripe size)
- Traffic on storage targets
- Fullness of storage targets
- Fragmentation on storage targets

I/O tuning process

- Collect logs / traces
- If performance bottlenecks are there,
  - Identify the time when bottleneck happened
  - Know why it happened – root cause analysis
- Find tuning options
- **Apply tuning options**



File system – Lustre architecture

- Lustre
- Main components
  - Metadata server (MDS)
  - Object storage servers (OSS)
  - Object storage targets (OSTs)

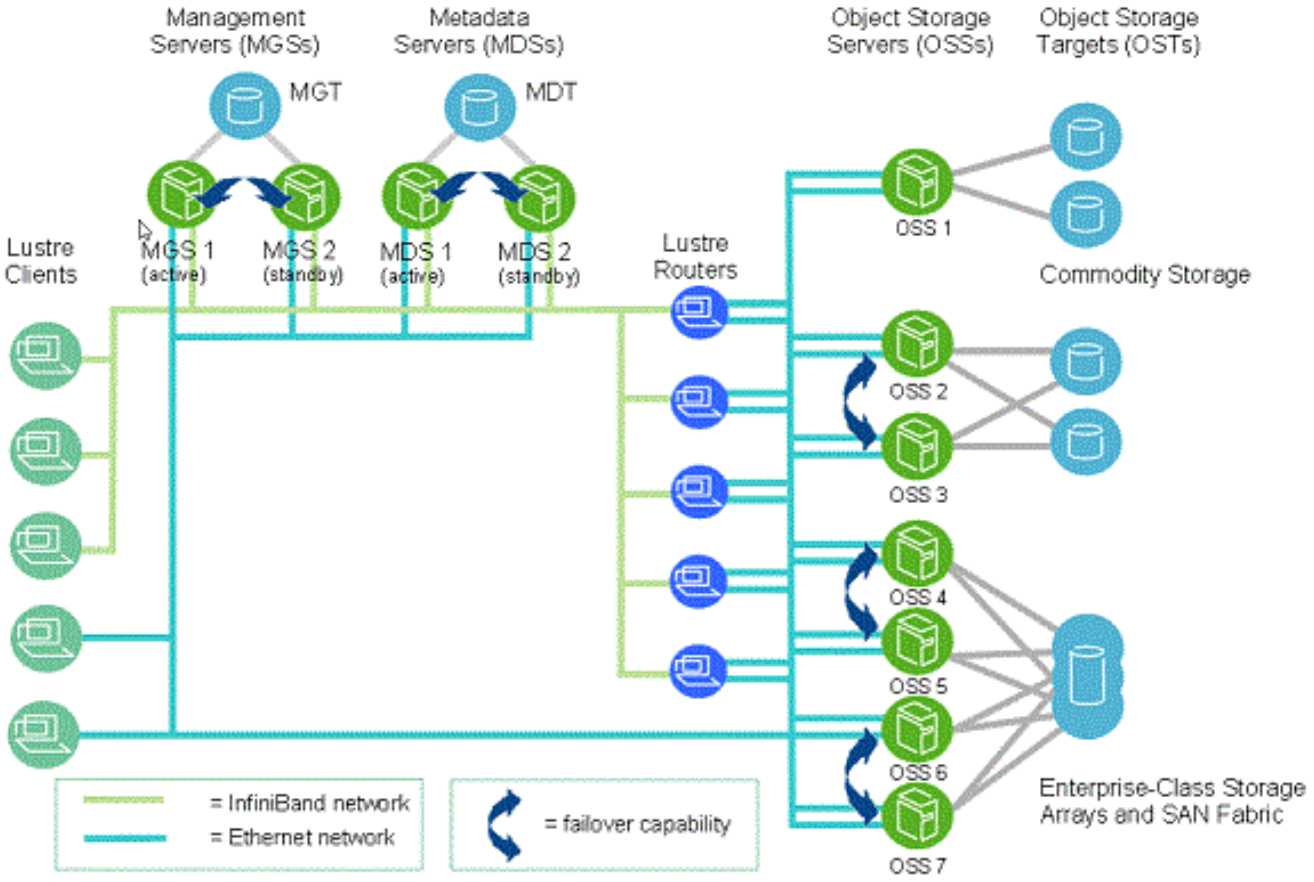


Image from: https://www.weka.io/learn/lustre/lustre-file-system-explained

User-level tuning of Lustre

- $ lfs setstripe --stripe-size [stripe-size] --stripe-index [OST-start-index] --stripe-count [stripe-count] filename

stripe-size	Number of bytes write on one OST before cycling to the next. Use multiples of 1MB. Default has been most successful.
stripe-count	Number of OSTs a file exists on
OST-start-index	Starting OST. Default highly recommended

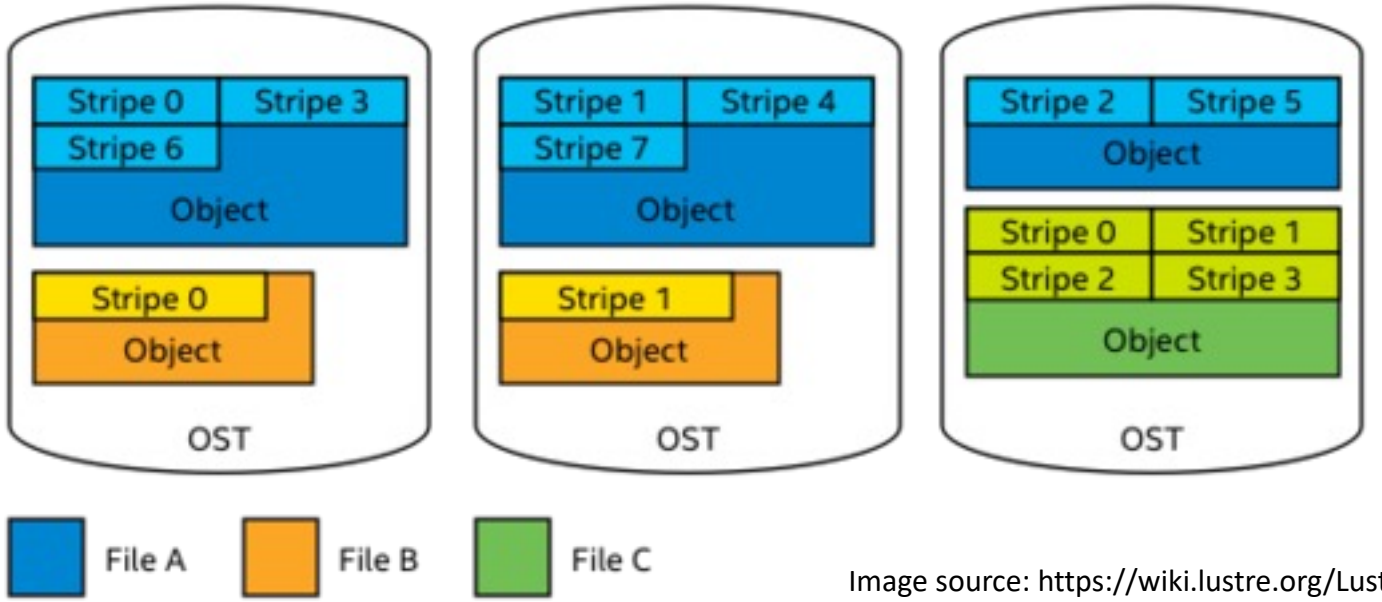


Image source: https://wiki.lustre.org/Lustre_Object_Storage_Service_%28OSS%29

Lustre - getstripe

- lfs getstripe [--quietl-q] [--verbosel-v] [--stripe-countl-c] [--stripe-indexl-i] [--stripe-sizel-S] [--directoryl-d] [--recursivel-r] filename

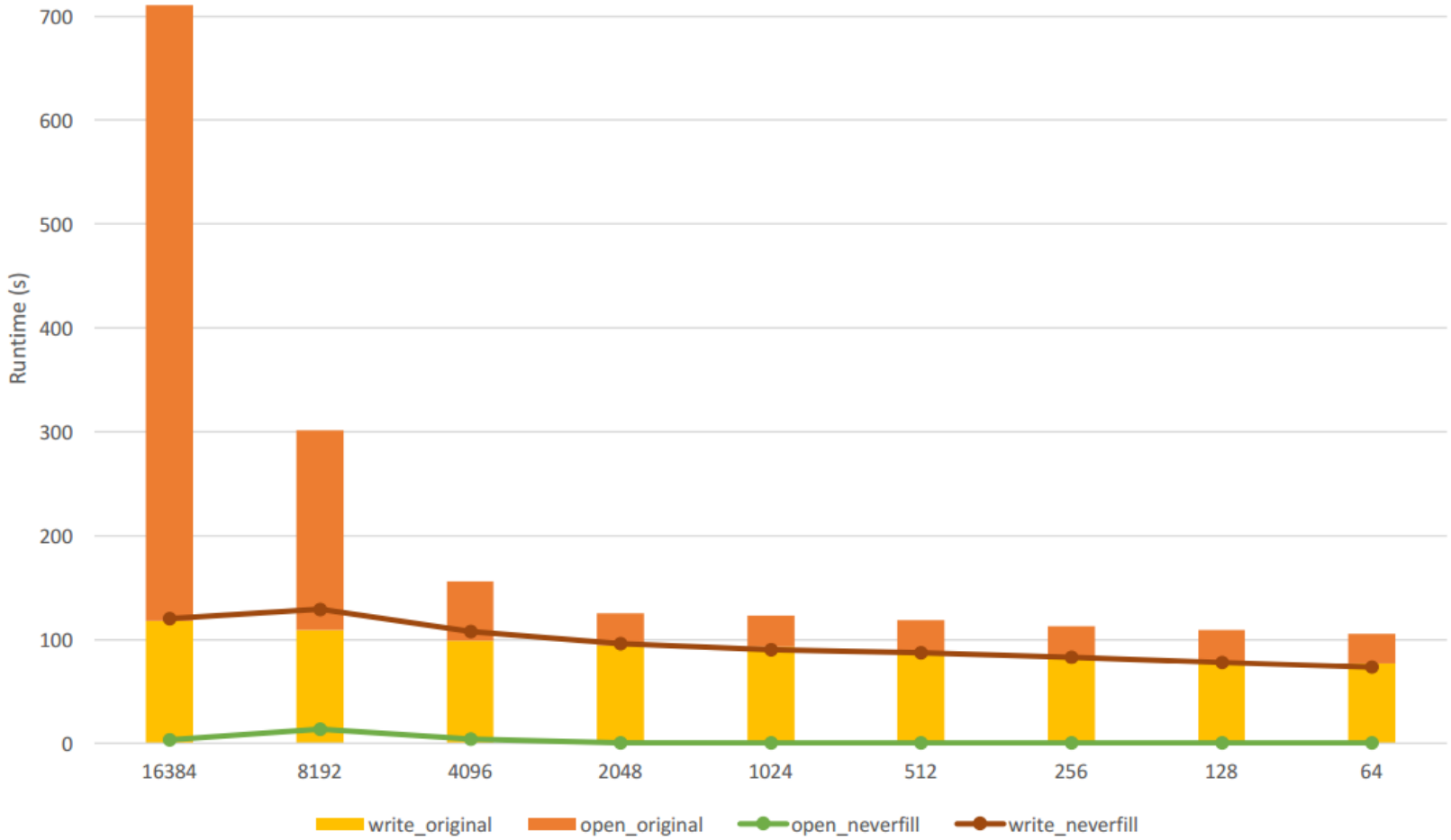
> lfs setstripe --stripe-size 2M --stripe-count 4 temp-file

> lfs getstripe temp-file
temp-file
lmm_stripe_count: 4
lmm_stripe_size: 2097152
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: 59

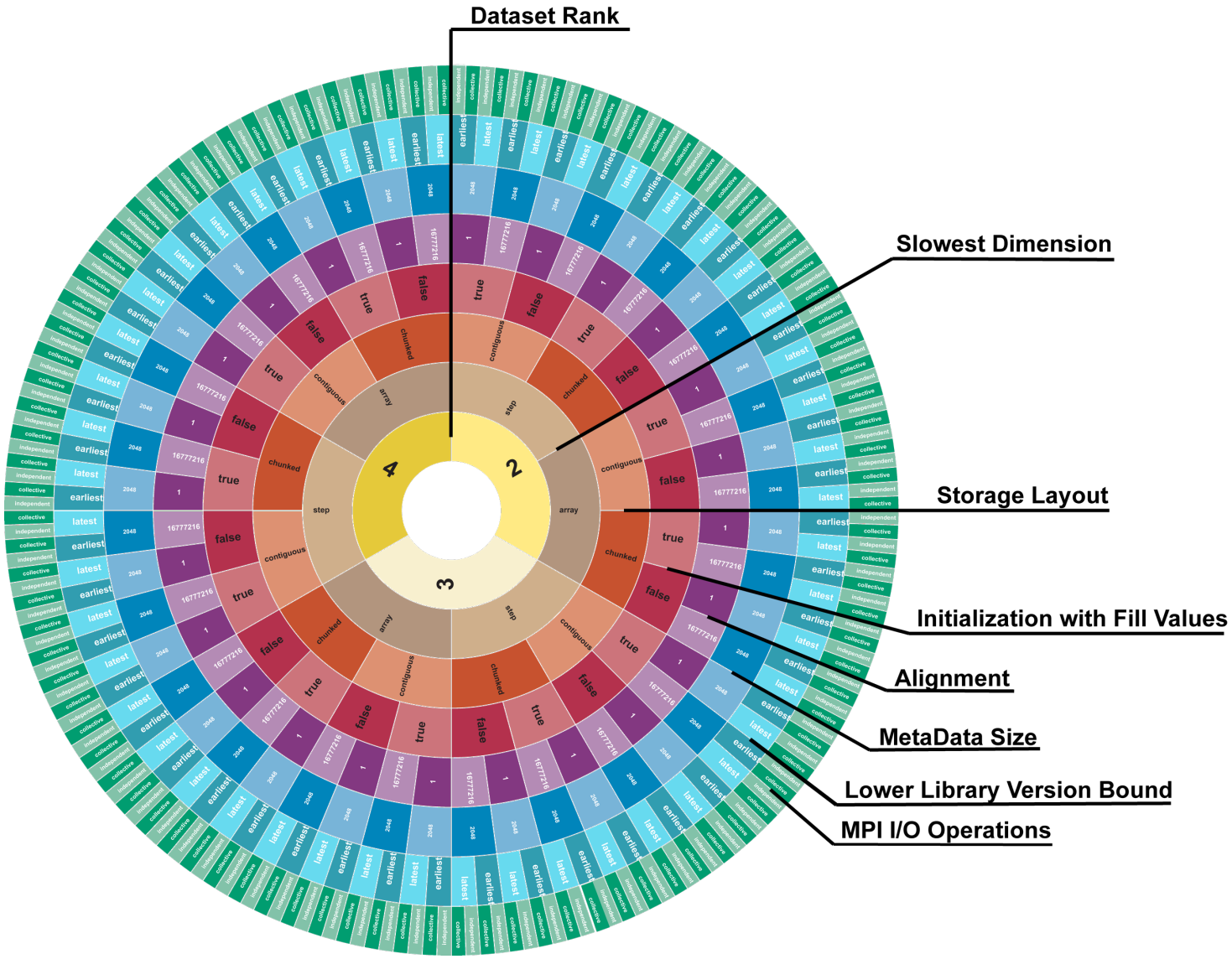
obdidx	objid	objid	group
59	3077476	0x2ef564	0
60	3085794	0x2f15e2	0
61	3050337	0x2e8b61	0
62	3061633	0x2eb781	0

HDF5 level tuning - Object Creation (SEISM-IO, Blue Waters—NCSA)

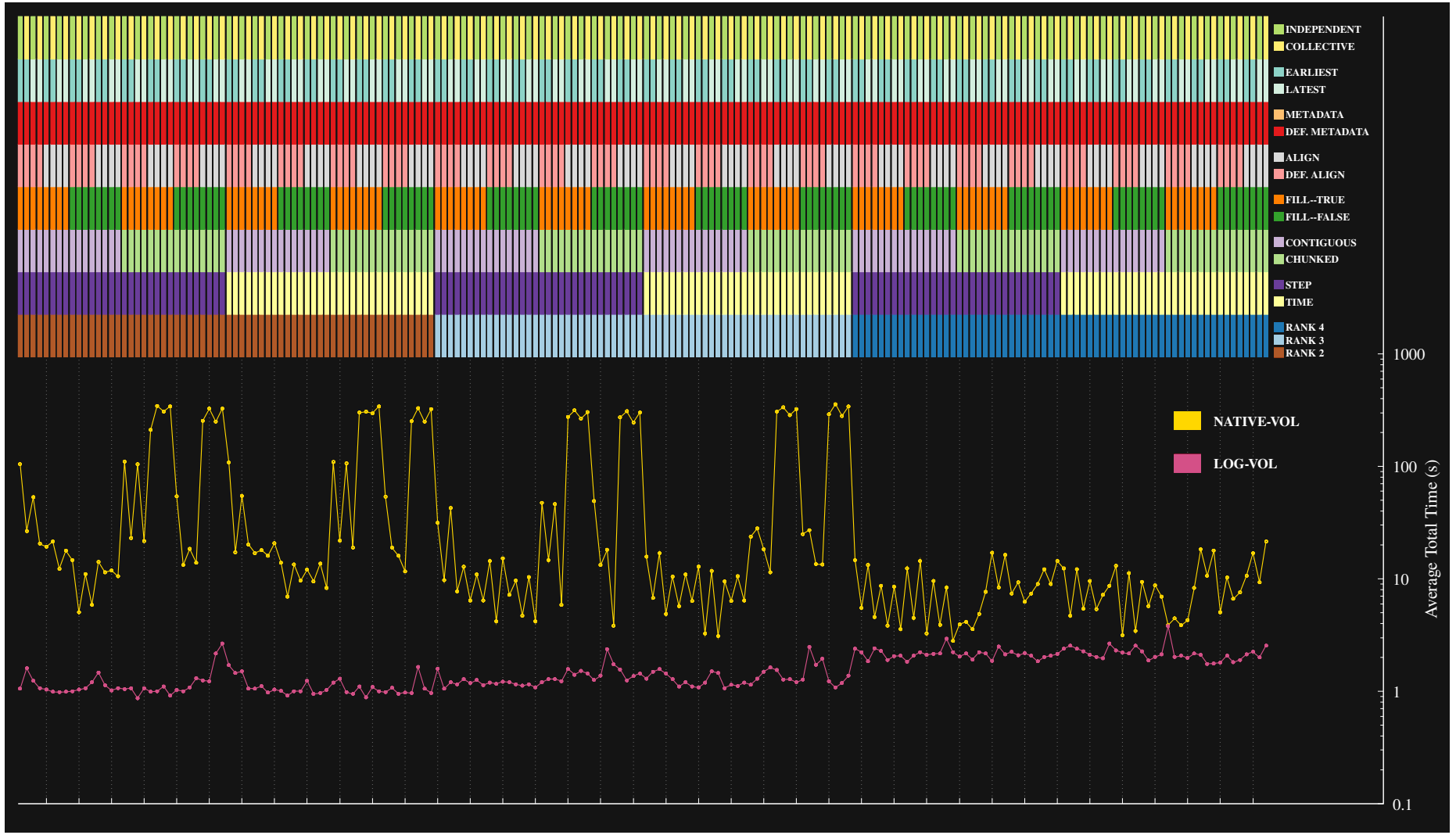
Set HDF5 to never fill chunks (H5Pset_fill_time with H5D_FILL_TIME_NEVER)



HDF5 Parameter Space



Log-structured data write can help reduce variability



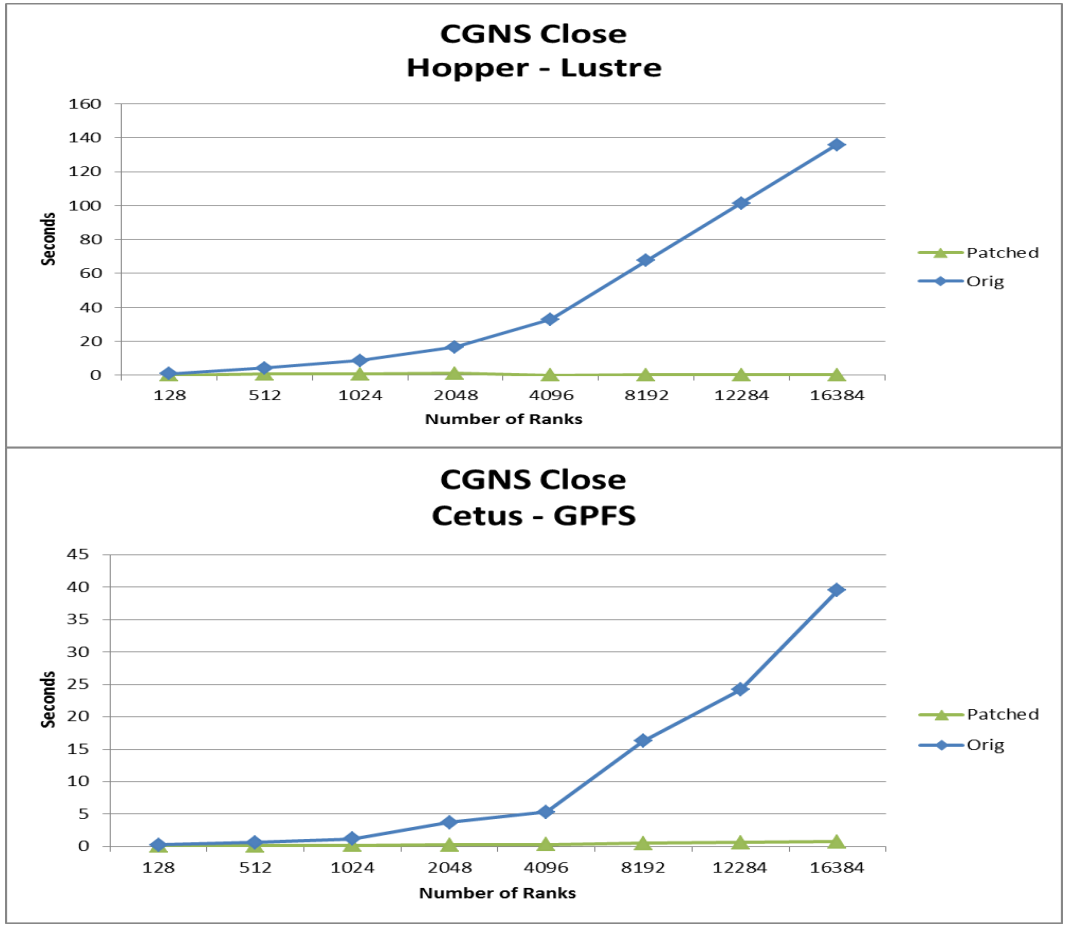
Total time (read & write) in the HDFspace set for Cori on 512 ranks, LOG-BASED VOL

Write Metadata Collectively

- **Symptoms:** Many users reported that `H5Fclose()` is very slow and doesn't scale well on parallel file systems.
- **Diagnosis:** HDF5 metadata cache issues very small accesses (one write per entry). We know that parallel file systems don't do well with small I/O accesses.
- **Solution:** Gather up all the entries of an epoch, create an MPI-derived datatype, and issue a single collective MPI write.

H5P_SET_COLL_METADATA_WRITE	Establishes I/O mode property setting, collective or independent, for metadata writes
H5P_GET_COLL_METADATA_WRITE	Retrieves I/O mode property setting for metadata writes
H5P_SET_ALL_COLL_METADATA_OPS	Establishes I/O mode, collective or independent, for metadata read operations
H5P_GET_ALL_COLL_METADATA_OPS	Retrieves I/O mode for metadata read operations

Closing a CGNS File …



HDF5 – Set collective I/O for aggregating

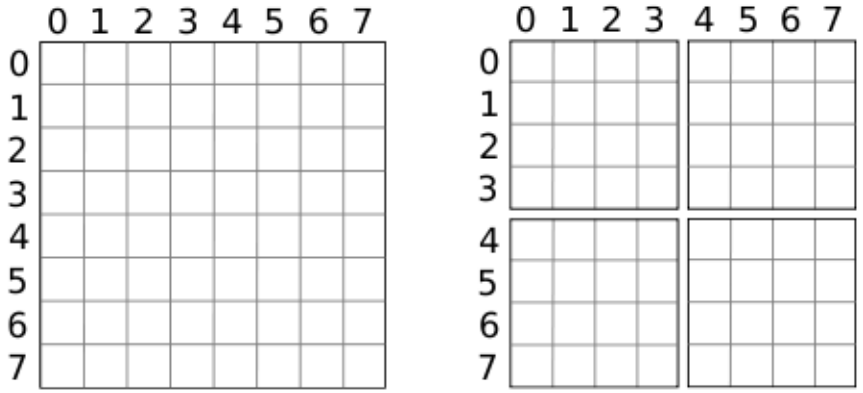
```
/* create the file in parallel */ fapl_id = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpio(fapl_id, mpi_comm, mpi_info); file_id =
H5Fcreate("myparfile.h5", H5F_ACC_TRUNC, H5P_DEFAULT, fapl_id);

dxpl_id = H5Pcreate(H5P_DATASET_XFER);
H5Pset_dxpl_mpio(dxpl_id, H5FD_MPIO_COLLECTIVE); /* describe a 1D array of
elements on this processor */
memspace = H5Screate_simple(1, count, NULL); /* map this processor's elements into
the shared file */
filespace = H5Screate_simple(1, mpi_size*count, NULL);
offset = mpi_rank * count;
H5Sselect_hyperslab(filespace, H5S_SELECT_SET, &offset, NULL, &count, NULL);
H5Dwrite(dset_id, H5T_NATIVE_FLOAT, memspace, filespace, dxpl_id, somedata0);
```

HDF5 Dataset I/O

- Issue large I/O requests
  - At least as large as the file system block size
- Avoid **datatype conversion**
  - Use the same data type in the file as in memory
- Avoid **dataspace conversion**
  - One dimensional buffer in memory to two-dimensional array in the file

Can break collective operations; check what mode was used
H5Pget_mpio_actual_io_mode, and why
H5Pget_mpio_no_collective_cause

Use chunking if chunks of data are used for reading




```
fapl = H5Pcreate(H5P_FILE_ACCESS); H5Pset_alignment(fapl, 0, stripe_size);
file = H5Fcreate("myparfile.h5", H5F_ACC_TRUNC, H5P_DEFAULT, fapl);
dcpl = H5Pcreate(H5P_DATASET_CREATE);
H5Pset_chunk(dcpl, 3, chunk_dims);
H5Dcreate(file, "mydataset", type, filespace, H5P_DEFAULT, dcpl, H5P_DEFAULT);
```

```
btree_ik = (stripe_size - 4096) / 96;
fcpl = H5Pcreate(H5P_FILE_CREATE);
H5Pset_istore_k(fcpl, btree_ik);
file = H5Fcreate("myparfile.h5", H5F_ACC_TRUNC, fcpl, fapl);
```

increasing the default size of the B-tree so that it is roughly the same size as a stripe:

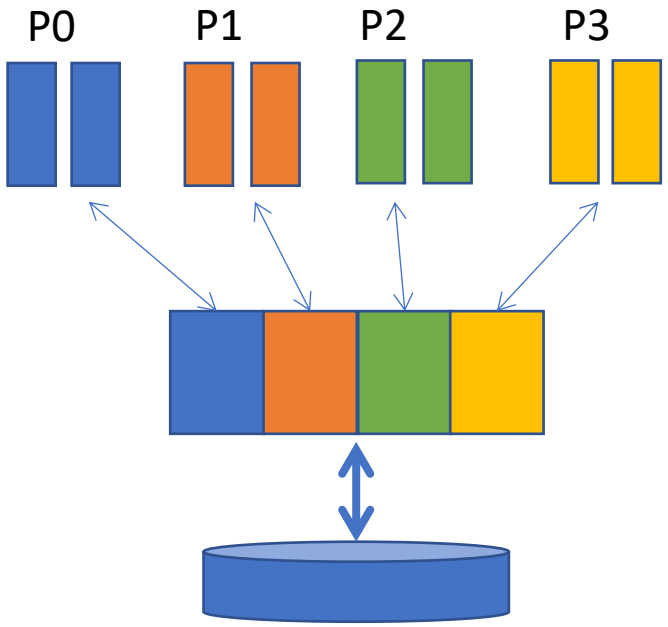
Metadata caching

Disable evictions from the metadata cache, unless H5Fflush is called or the file is closed.

```
mdc_config.version = H5AC__CURR_CACHE_CONFIG_VERSION;
H5Pget_mdc_config(file, &mdc_config)
mdc_config.evictions_enabled = FALSE;
mdc_config.incr_mode = H5C_incr__off;
mdc_config.decr_mode = H5C_decr__off;
H5Pset_mdc_config(file, &mdc_config);
```

MPI-IO performance optimizations – Collective buffering

- Also known as two-phase I/O
- A few processes aggregate data to temporary buffers and the data is then written to file (collective write operations)



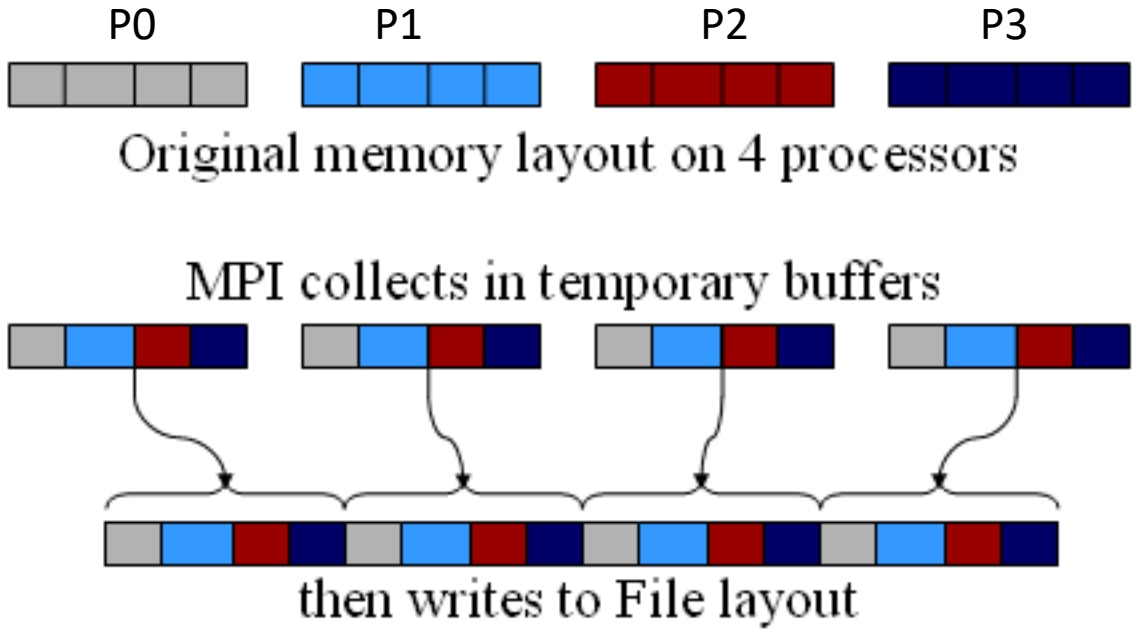


Image from https://cvw.cac.cornell.edu/ParallelIO/choreography

Passing hints with MPI_info

```
MPI_Info info;
MPI_Info_create (&info);
/* no. of I/O devices to be used for file striping */
MPI_Info_set (info, "striping_factor", "4");

/* the striping unit in bytes */
MPI_Info_set (info, "striping_unit", "65536");

MPI_File_open(MPI_COMM_WORLD, ”data.file",
        MPI_MODE_CREATE | MPI_MODE_RDWR, info, &fh);
MPI_Info_free (&info);
```

striping_factor - size of “strips” on I/O servers
striping_unit - number of I/O servers to stripe across
start_iodevice - which I/O server to start with
cb_config_list - list of aggregators
cb_nodes - number of aggregators (upper bound)
romio_cb_read, romio_cb_write - aggregation on/off
romio_ds_read, romio_ds_write - data sieving on/off

Adapted from Prof. Bill Gropp’s “Introduction to MPI-IO” talk

MPI-IO level – Use collective buffering

- setenv MPIIO_MPICH_HINTS "*:romio_cb_write=enable:romio_ds_write=disable”

- All processes must call the collective I/O function
- Aggregating large blocks so that the reads / writes to the I/O system would be large

- `MPI_File_write_at_all ()`
  - _all → all processes in the communicator are participating
  - _at → provides thread-safety and avoids a separate seek
- `MPI_File_seek`
  - `MPI_File_read_all`
  - `MPI_File_write_all`
  - `MPI_File_read_at_all`
  - `MPI_File_write_at_all`

Summary of today's class

- Parallel I/O performance factors and applying tuning options
- Next Class – How to apply tuning options automatically
- Class presentation on March 9th