Improving HDF5 write performance with log-based storage layout

Team members:

Kai-yuan Hou, Qiao Kang, Sunwoo Lee, Alok, Choudhary, Wei-keng Liao

Northwestern University

Rob Latham, Rob Ross

Argonne National Laboratory

HDF5 User Group meeting

Oct. 14 2021

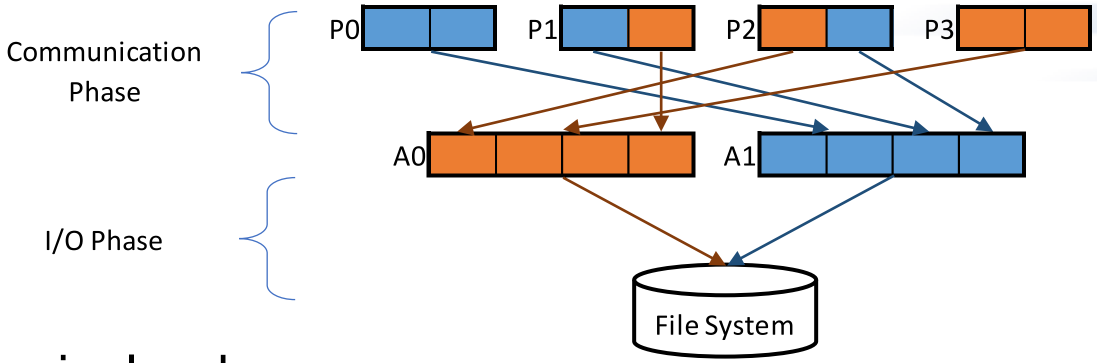
Outlines

- **Backgrounds**
- Design of Log-based VOL
- Case Studies

Parallel I/O performance bottlenecks

- Array canonical data storage layout
  - Requires reorganizing data into row/column-major order
  - Often performs slow for I/O containing a large number of non-contiguous requests
    - Due to a high inter-process communication overhead in the two-phase I/O in MPI collective I/O, which is to organize requests among processes
- HDF5 solution
  - Chunked storage layout
    - Can alleviate but not eliminate the issue
    - Data in each chunk must still be stored in the canonical order

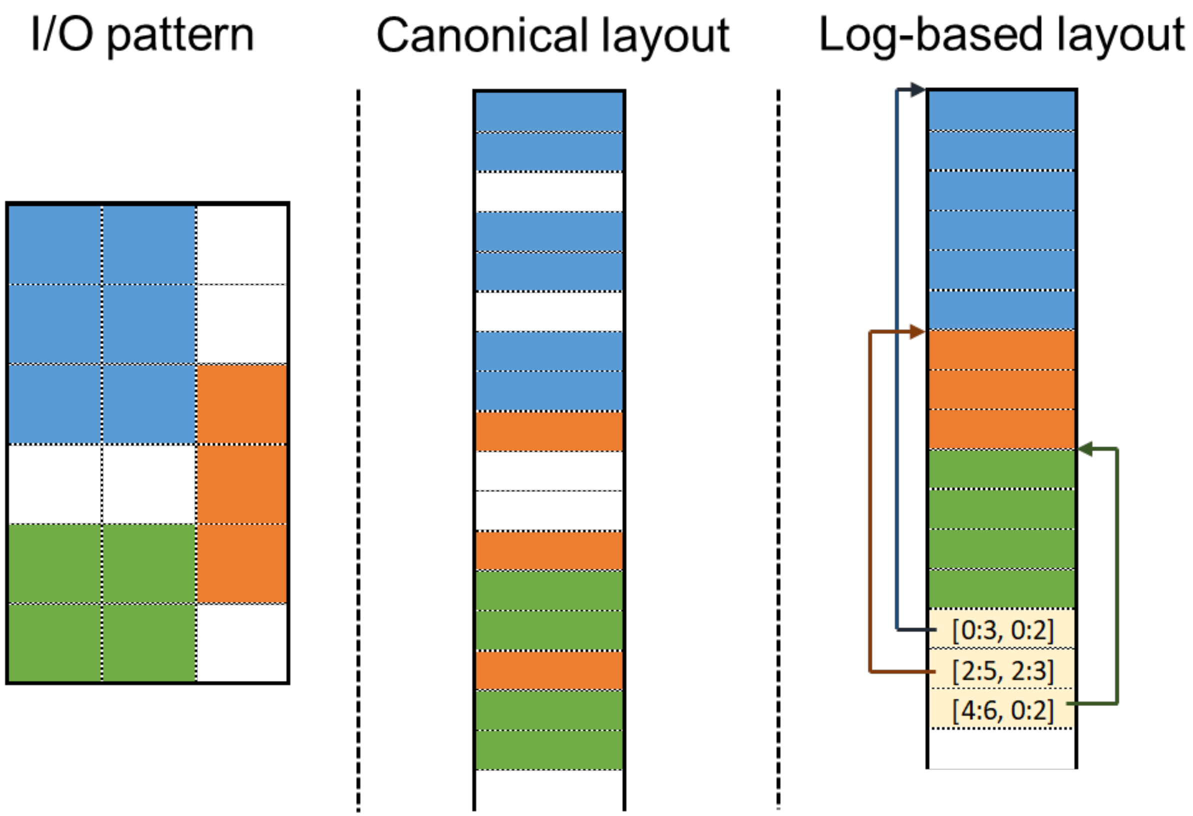
Two phase I/O example



Log-based storage layout

- Append the data in the file based on the order of the requests
  - Disregard the data's canonical order
  - Require additional space to store metadata describing the canonical location of the request
  - Files in the canonical layout can be reconstructed later, on line or off line
  - Kimpe, Dries, et al. "Transparent log-based data storage in MPI-IO applications." *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, Berlin, Heidelberg, 2007.
- Pros
  - Requests are contiguous
  - Avoid expensive communication cost to reorganize the requests
- Cons
  - Metadata size can be large when the number of noncontiguous requests is large
  - Read log-structured data requires communication, similar to the two-phase I/O

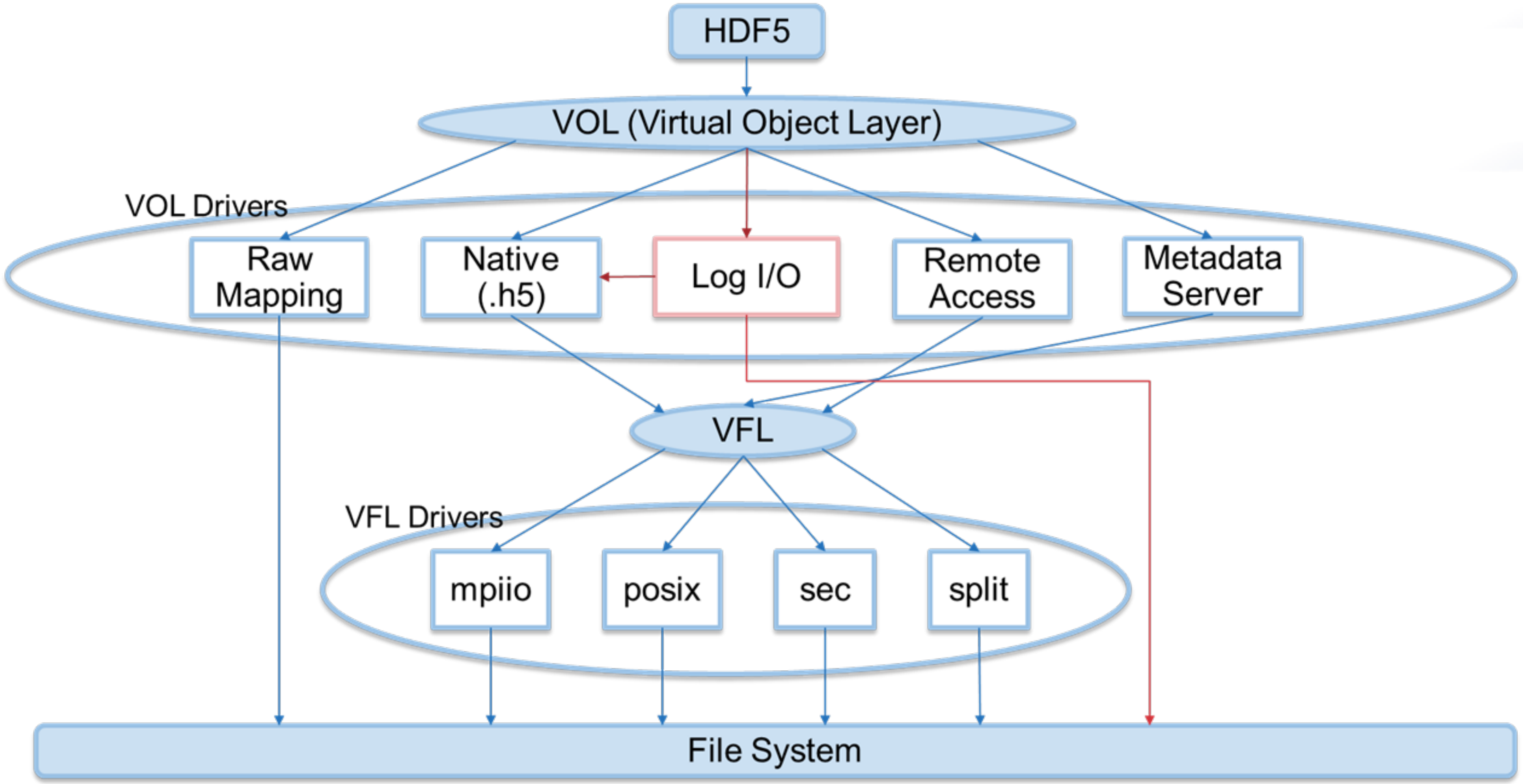
Log-based storage layout



Log-based HDF5 VOL

- Backgrounds
- **Design of Log-based VOL**
- E3SM I/O Case study

Bringing log-based storage layout to HDF5



Comply with HDF5 file format specification

- Log-based data structure within an HDF5 file
  - Files are in legit HDF5 format
  - Datasets are stored in log layout
- Metadata operations through the native VOL
  - File, group, datasets operations are still through the native VOL
- Use MPI-IO to write datasets
  - Write data and metadata to files using MPI-IO directly
  - Bypass overhead in the native VOL
  - Avoid creating data space

Metadata

- Creates a group to store the log data structure
  - 1 dimensional byte (H5T_STD_B8LE) datasets
    - Data datasets: raw data of each I/O requests
    - Metadata datasets: Describe the canonical location and other property of the data entries
- Attributes for additional file metadata (number of metadata datasets … etc.)
- All datasets are scalar
  - Data are stored elsewhere, no need to allocate space
  - Original shape is stored as attribute
  - Given a unique ID for identification in the log data structure

Dataset I/O

- Log-based VOL always aggregates dataset I/O requests
  - H5Dwrite and H5Dread only “stages” the write operation
  - No communication or actual I/O
  - I/O requests are aggregated independently on each process
- Data are flushed to the log dataset on file close or when user calls H5Fflush
- Aggregate metadata across the entire file open session
  - Flushed to the metadata dataset only on file close
- Simulate the effect of blocking I/O by copy the data into internal buffer
  - Allow user buffer to be modified after return from H5Dwrite calls

Example of Log VOL output file

```c
int main (int argc, char **argv) {
    int buf[2];
    hid_t log_vlid, faplid, fid, sid, did;
    hsize_t dim = 2, one = 1, count = 1, start;
    MPI_Init (&argc, &argv);
    // Register LOG VOL plugin
    log_vlid = H5VLregister_connector (&H5VL_log_g, H5P_DEFAULT);
    faplid   = H5Pcreate (H5P_FILE_ACCESS);
    H5Pset_fapl_mpio(faplid, MPI_COMM_WORLD, MPI_INFO_NULL);
    H5Pset_all_coll_metadata_ops(faplid, 1);
    H5Pset_vol (faplid, log_vlid, NULL);
    // Create file
    fid = H5Fcreate ("example.h5", H5F_ACC_TRUNC, H5P_DEFAULT, faplid);
    // Create 1-D dataset
    sid = H5Screate_simple (1, &dim, &dim);
    did = H5Dcreate2 (fid, "D", H5T_STD_I32LE, sid, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
    // Write first element
    start = 0;
    H5Sselect_hyperslab (sid, H5S_SELECT_SET, &start, NULL, &one, &count);
    H5Dwrite (did, H5T_NATIVE_INT, H5S_ALL, sid, H5P_DEFAULT, buf);
    // Flush the data
    H5Fflush (fid, H5F_SCOPE_GLOBAL);
    // Write second element
    start = 1;
    H5Sselect_hyperslab (sid, H5S_SELECT_SET, &start, NULL, &one, &count);
    H5Dwrite (did, H5T_NATIVE_INT, H5S_ALL, sid, H5P_DEFAULT, buf);
    // Close the file
    H5Sclose (sid); H5Dclose (did); H5Pclose (faplid); H5Fclose (fid);
    return 0;
}
```

```
HDF5 "example.h5" {
GROUP "/" {
  ATTRIBUTE "_int_att" {
    DATATYPE  H5T_STD_I32LE
    DATASPACE  SIMPLE { ( 4 ) / ( 4 ) }
  }
  DATASET "D" {
    DATATYPE  H5T_STD_I32LE
    DATASPACE  SCALAR
    ATTRIBUTE "_ID" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
    }
    ATTRIBUTE "_dims" {
      DATATYPE  H5T_STD_I64LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
    }
    ATTRIBUTE "_mdims" {
      DATATYPE  H5T_STD_I64LE
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
    }
  }
  GROUP "_LOG" {
    DATASET "_ld_0" {
      DATATYPE  H5T_STD_B8LE
      DATASPACE  SIMPLE { ( 4 ) / ( 4 ) }
    }
    DATASET "_ld_1" {
      DATATYPE  H5T_STD_B8LE
      DATASPACE  SIMPLE { ( 4 ) / ( 4 ) }
    }
    DATASET "_md_0" {
      DATATYPE  H5T_STD_B8LE
      DATASPACE  SIMPLE { ( 104 ) / ( 104 ) }
    }
  }
}
}
```

Log-based VOL

- Backgrounds
- Design of Log-based VOL
- **Case Studies**

E3SM climate simulation model

- E3SM I/O benchmark for PnetCDF, NetCDF4, HDF5
  - https://github.com/Parallel-NetCDF/E3SM-IO.git
- Large amount of non-contiguous file writes
  - Inefficient in contiguous layout
- Large number of datasets to write
  - Write one dataset at a time is slow
- Uneven number of write request per process
  - Need aggregation to perform collective I/O
- Native VOL driver cannot finish in a reasonable time

E3SM simulation framework case study

- Cori KNL nodes, 128 Lustre stripe, 16 MiB stripe size, 64 processes / node
- HDF5 develop branch
- TAM MPI-IO optimization developed by our group
  - Use local request aggregation to improve collective I/O performance
- 5 output files
  - 2 files (H0 and H1) for atmospheric component (F case)
  - 1 file for oceanic component (G case)
  - 2 files (H0 and H1) for land component (I case)

E3SM output file properties

Property\File	F case H0	F case H1	G case	I case H0	I case H1
Number of processes	21600	21600	9600	1344	1344
Total size of data (GiB)	14.09	6.68	79.69	86.11	0.36
Number of fixed sized variables	15	15	11	18	10
Number of record variables	399	36	41	542	542
Number of records	1	25	1	240	1
Number of partitioned vars	25	27	11	14	14
Number of non-partitioned vars	389	24	41	546	538
Number of non-contig. requests (max among processes)	174953	83261	20888	9248875	38650
Number of attributes	1427	148	858	2789	2759

E3SM simulation framework case study

Description \ Dataset	F case H0	F case H1	G case	I case H0	I case H1
Preparing I/O (sec)	0.00	0.00	0.01	0.00	0.00
Opening the file (sec)	0.20	0.12	0.75	0.08	0.09
Creating datasets and attributes (sec)	0.38	0.07	0.29	0.59	0.72
Posting I/O requests (sec)	0.09	0.10	0.01	17.22	0.08
Flushing the data to the log (sec)	2.72	1.17	4.70	8.92	0.22
Closing the file, flushing metadata (sec)	4.32	0.80	2.27	2.34	0.66
End to end time (sec)	**7.70**	**2.27**	**8.03**	**29.15**	**1.78**
E3SM I/O write amount (GiB)	14.09	6.68	79.69	86.11	0.36
Output file size (GiB)	15	9.5	80	93	0.39
Metadata overhead (MiB)	386.70	2825.67	210.33	6969.92	29.39

Summary and Future Work

- Log-based storage layout for HDF5 datasets
  - Built on HDF5 data objects
- Significant performance improvement over existing storage layout in HDF5
  - Especially on complex I/O patterns
- Future work
  - Improve H5Dwrite performance
  - Reduce metadata overhead
  - Investigate scaling issues on Summit GPFS
    - We observed significant performance degradation when scaled beyond certain number of processes

Acknowledgements

- This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.
- This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231.
- 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.
- Quincey Koziol @ LBNL and now in Amazon
- Suren Byna @ LBNL
- Danqing Wu @ ANL

IOR benchmark

- Experiment setup
  - 1 MiB block
  - 1 MiB transfer size
  - 32 Blocks
  - Cori, 8 Haswell nodes, 32 process per node
  - HDF5 1.12.0

Operations	Log-based VOL	Native VOL
File size	8 GiB	8 GiB
I/O time	1.49 sec	1.84 sec
Bandwidth	5.36 GiB/s	4.34 GiB/s

Thank you

- https://github.com/DataLib-ECP/vol-log-based
- Questions?