Study of HACC-IO Benchmark

Chen Wang

HACC-IO Benchmark

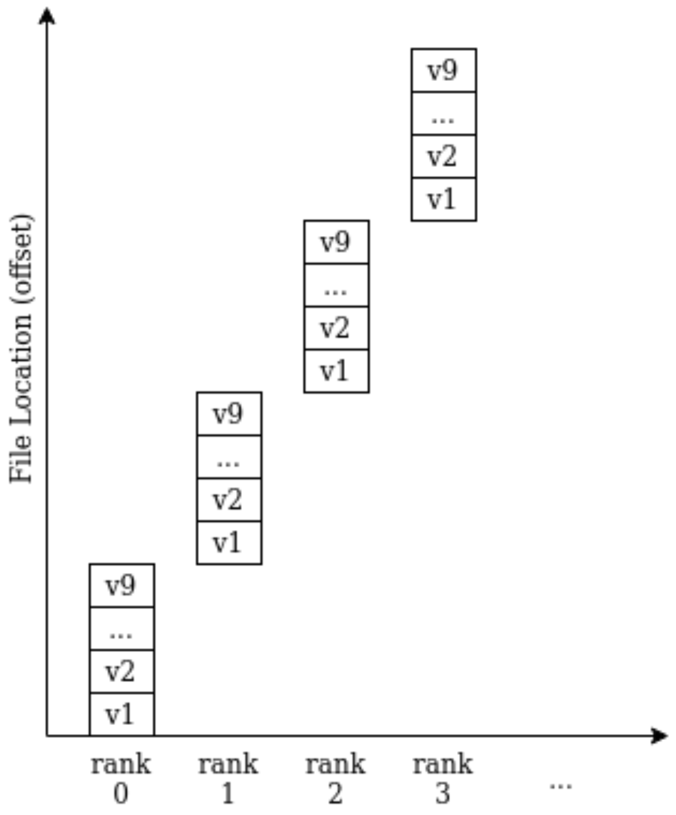
- Write 9 variables. All variables have the same size.
- Use one 1D array to store all 9 variables (except HDF5 compound)
- In our experiments the problem size is fixed:
  - Each variable is 8GB, i.e., 1M doubles.
  - Total file size is 72GB
  - Scale: 1024 Procs: 32nodes x 32 procs/node.
  - Each process get 8M per variable

HACC-IO Benchmark

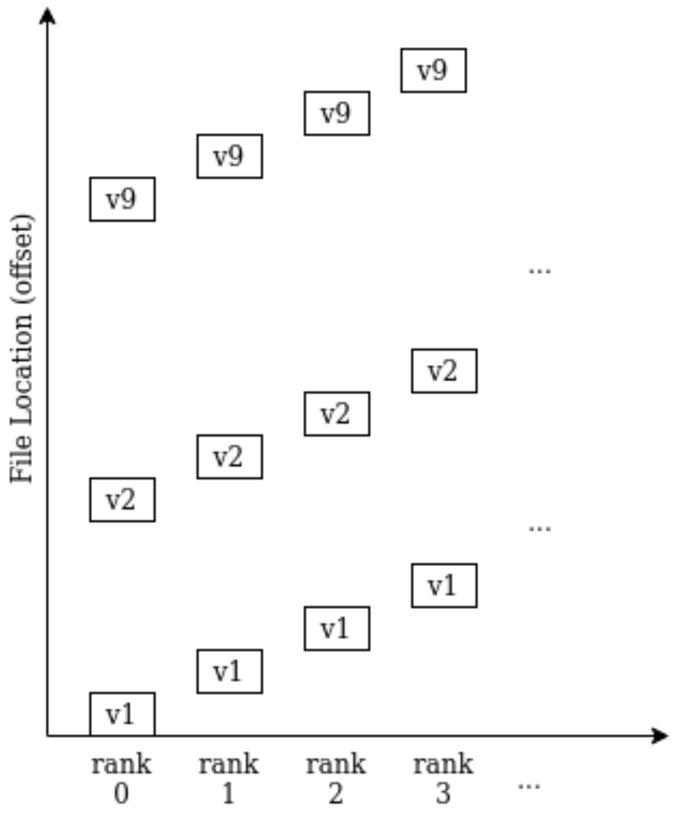
- Write 9 variables. All variables have the same size.
- Use one 1D array to store all 9 variables (except HDF5 compound)
- 2 MPI Benchmarks:
  - MPI Interleaved, MPI Contiguous

```
double *writedata = (double*) malloc(BUF_SIZE_PER_VAR/mpi_size*NUM_VARS);

// Benchmark 1. MPI Interleaved
for (i=0; i < NUM_VARS; i++) {
    MPI_File_write_at(fh, mpi_off, &writedata[i*NUM_DOUBLES_PER_VAR_PER_RANK], NUM_DOUBLES_PER_VAR_PER_RANK,
                      MPI_DOUBLE, &mpi_stat);
    mpi_off += BUF_SIZE_PER_VAR/mpi_size;     // Move to the end of the current write.
}

// Benchmark 2. MPI Contiguous
for (i=0; i < NUM_VARS; i++) {
    MPI_File_write_at(fh, mpi_off, &writedata[i*NUM_DOUBLES_PER_VAR_PER_RANK], NUM_DOUBLES_PER_VAR_PER_RANK,
                      MPI_DOUBLE, &mpi_stat);
    mpi_off += BUF_SIZE_PER_VAR;              // Move to the position of next variable
}
```

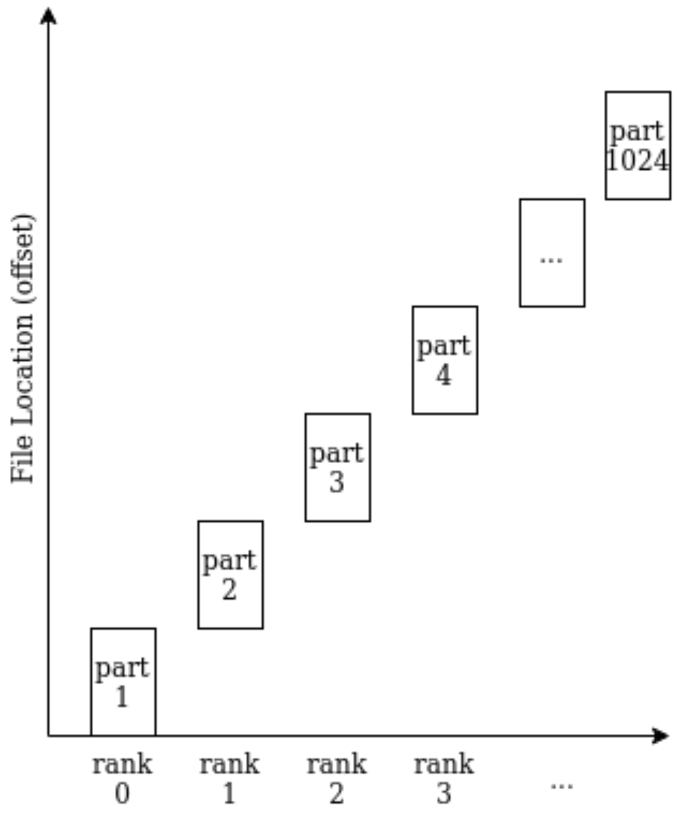
5 Benchmarks, 3 Access Patterns



MPI Interleaved = HDF5 Multi



MPI Contiguous = HDF5 Individual



HDF5 Compound

HACC-IO Benchmark

- Write 9 variables. All variables have the same size.
- Use one big 1D array to store all 9 variables (except HDF5 compound)
- 3 HDF5 Benchmarks:
    - HDF5 individual, HDF5 Multi, HDF5 Contiguous

```
// Benchmark 3, 4: HDF5 Individual and HDF5 Multi
double *writedata = (double*) malloc(BUF_SIZE_PER_VAR/mpi_size*NUM_VARS);
for (i = 0; i < NUM_VARS; i++) {
    dset_ids[i] = H5Dopen(file_id, DATASETNAME[i], H5P_DEFAULT);
    mem_space_ids[i] = H5Screate_simple(1, mem_dims, NULL);
    file_space_ids[i] = H5Screate_simple(1, file_dims, NULL);

    // Select column of elements in the file dataset
    file_start[0] = mpi_rank * NUM_DOUBLES_PER_VAR_PER_RANK;
    file_count[0] = NUM_DOUBLES_PER_VAR_PER_RANK;
    H5Sselect_hyperslab(file_space_ids[i], H5S_SELECT_SET, file_start, NULL, file_count, NULL);

    // Select elements in the memory buffer
    mem_start[0] = i * NUM_DOUBLES_PER_VAR_PER_RANK;
    mem_count[0] = NUM_DOUBLES_PER_VAR_PER_RANK;
    H5Sselect_hyperslab(mem_space_ids[i], H5S_SELECT_SET, mem_start, NULL, mem_count, NULL);

    H5Dwrite(dset_ids[i], H5T_NATIVE_DOUBLE, mem_space_ids[i], file_space_ids[i], dxfer_plist_id, writedata);
}
```

HACC-IO Benchmark

- Write 9 variables. All variables have the same size.
- Use one big 1D array to store all 9 variables (except HDF5 compound)
- 3 HDF5 Benchmarks:
    - HDF5 individual, HDF5 Multi, HDF5 Contiguous

```
typedef struct {
    double id;
    double mask;
    double x;
    double y;
    double z;
    double vx;
    double vy;
    double vz;;
    double phi;
} hacc_t;
```

```
// Benchmark 5: HDF5 Compound datatype
Hmemtype = H5Tcreate (H5T_COMPOUND, sizeof (hacc_t));
           H5Tinsert (Hmemtype, "id", HOFFSET (hacc_t, id), H5T_NATIVE_DOUBLE);
           H5Tinsert (Hmemtype, "mask", HOFFSET (hacc_t, mask), H5T_NATIVE_DOUBLE);
           H5Tinsert (Hmemtype, "x", HOFFSET (hacc_t, x), H5T_NATIVE_DOUBLE);
           H5Tinsert (Hmemtype, "y", HOFFSET (hacc_t, y), H5T_NATIVE_DOUBLE);
           H5Tinsert (Hmemtype, "z", HOFFSET (hacc_t, z), H5T_NATIVE_DOUBLE);
           H5Tinsert (Hmemtype, "vx", HOFFSET (hacc_t, vx), H5T_NATIVE_DOUBLE);
           H5Tinsert (Hmemtype, "vy", HOFFSET (hacc_t, vy), H5T_NATIVE_DOUBLE);
           H5Tinsert (Hmemtype, "vz", HOFFSET (hacc_t, vz), H5T_NATIVE_DOUBLE);
           H5Tinsert (Hmemtype, "phi", HOFFSET (hacc_t, phi), H5T_NATIVE_DOUBLE);

mem_dims[0] = NUM_HDATA_PER_RANK;
file_dims[0] = mem_dims[0]*mpi_size;
file_space_id = H5Screate_simple(1, file_dims, NULL);
dset_id = H5Dcreate2(file_id, "ALLVAR", Hmemtype, file_space_id, H5P_DEFAULT, dcpl_id,
H5P_DEFAULT);
```

HACC-IO Benchmark

- To narrow down the analysis:
  - Timing code includes only the write functions
    - *MPI_File_write_at()* vs. *H5Dwrite()*
  - *open/close/flush* are not included.
  - For HDF5, metadata writes are also not included.

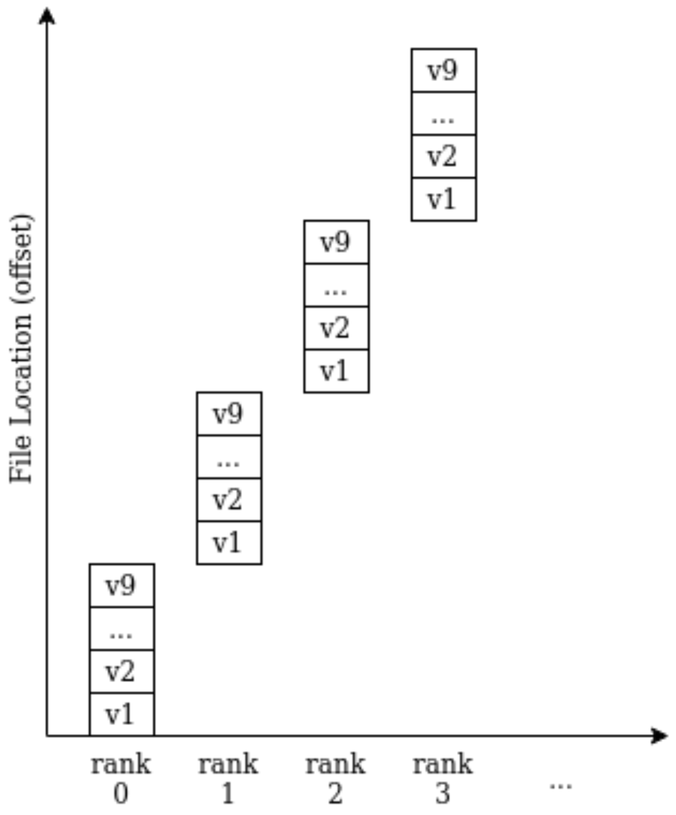
Timing code - HDF5

```
for (i = 0; i < NUM_VARS; i++) {
    // open, set mem_space_id, file_space_id
}

write_tstart = MPI_Wtime();
for (i = 0; i < NUM_VARS; i++) {
    H5Dwrite(dset_id, H5T_NATIVE_DOUBLE, mem_space_id, file_space_id, dxfer_plist_id,
writedata);
}
write_tend = MPI_Wtime();

for (i = 0; i < NUM_VARS; i++) {
    // close
}
```

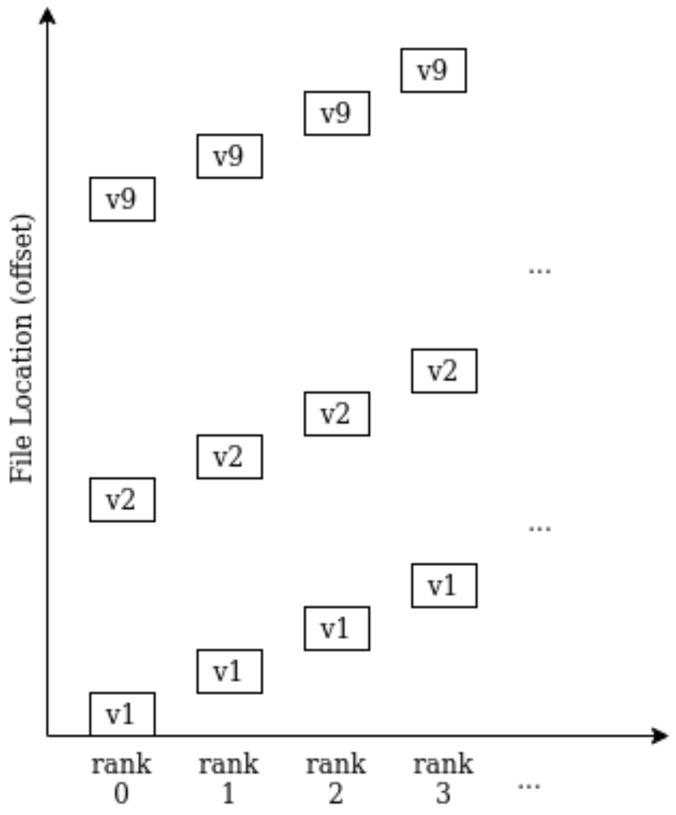
Timing code - MPI

```
double write_tstart = MPI_Wtime();
for (i=0; i < NUM_VARS; i++) {
    if(collective) {
        MPI_File_write_at_all(fh, mpi_off, &writedata[i*NUM_DOUBLES_PER_VAR_PER_RANK],
                              NUM_DOUBLES_PER_VAR_PER_RANK, MPI_DOUBLE, &mpi_stat);
    } else {
        MPI_File_write_at(fh, mpi_off, &writedata[i*NUM_DOUBLES_PER_VAR_PER_RANK],
                              NUM_DOUBLES_PER_VAR_PER_RANK, MPI_DOUBLE, &mpi_stat);
}

mpi_off += BUF_SIZE_PER_VAR;
}
double write_tend = MPI_Wtime();
```

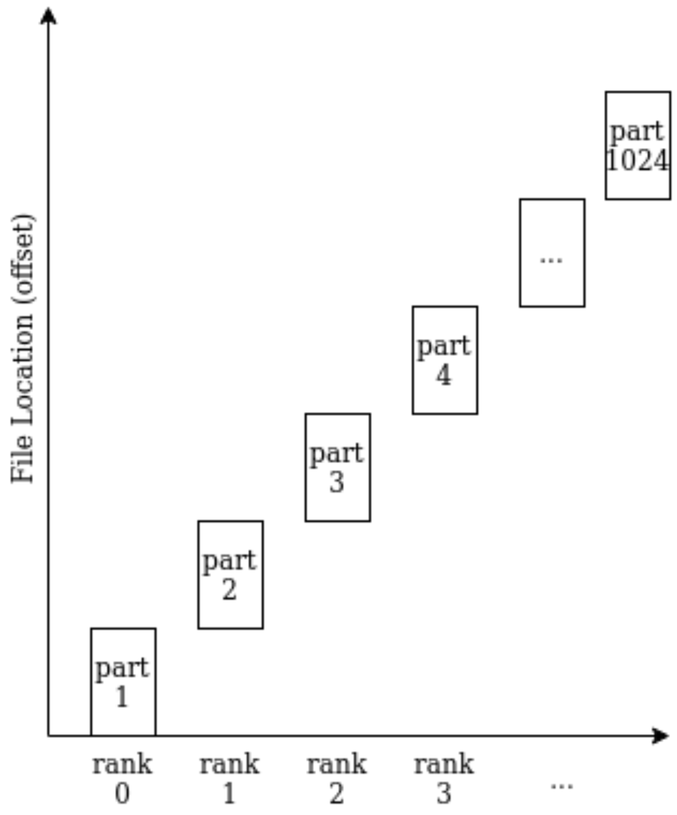
5 Benchmarks, 3 Access Patterns



MPI Interleaved = HDF5 Multi



MPI Contiguous = HDF5 Individual



HDF5 Compound

What we have done so far

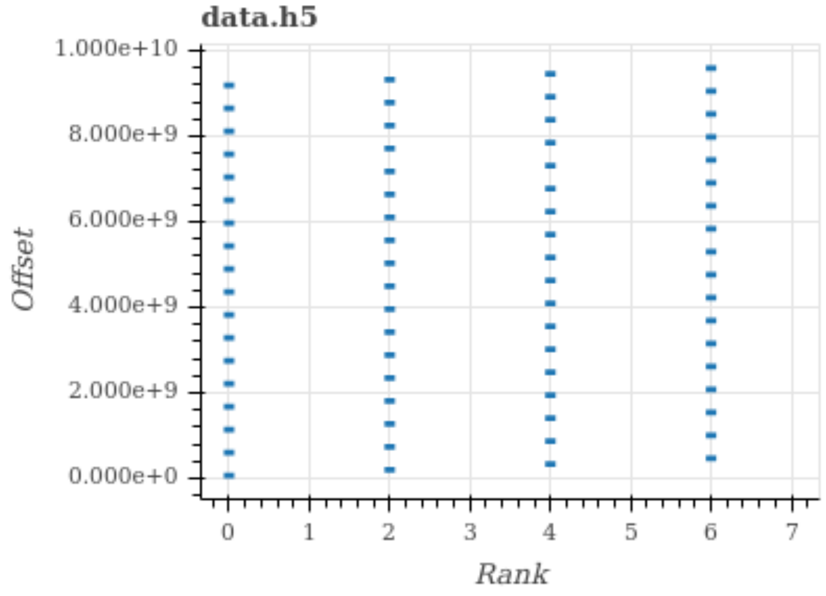
- Evaluated Pure MPI-IO implementation vs. HDF5 implementation
  - Lustre (Quartz) and GPFS (Lassen)
  - Different Lustre stripe size and stripe count
  - Different HDF5 versions
  - Alignment size
  - Metadata block size
  - …
- Tested with different data layouts:
  - MPI-Interleaved = HDF5-Multi
  - MPI-Contiguous = HDF5-Individual dataset
  - HDF5 Compound datatype

Conclusions

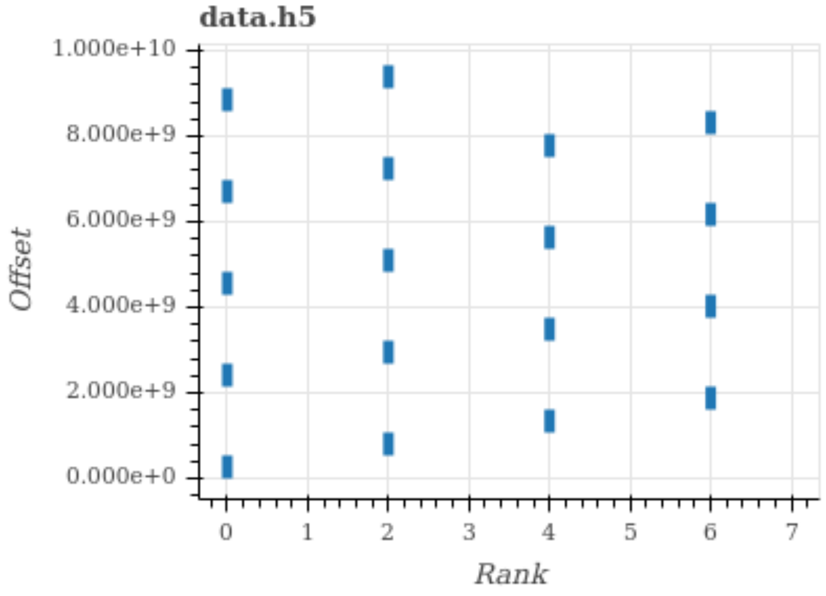
- Current version of HDF5 doesn’t have a way to match the MPI-Interleaved access pattern. HDF5_Multi achieves the same pattern.
- Collective I/O does not help since the request size is already very big.
- MPI_Interleaved (HDF5_Multi) is better on GPFS.
  - Because the stripe size is small and users can not change it.
- MPI_Contiguous (HDF5_Individual) is better on Lustre.
  - Because we use a big stripe size (128M)
  - For smaller stripe sizes (e.g. < 36M) MPI_Interleaved is better.

Collective vs. Independent

- Collective is beneficial for small writes --> merge into bigger writes.
- In our case, write size is big enough(8MB) to amortize disk seek time. The overhead of aggregation outweighs the I/O improvement.



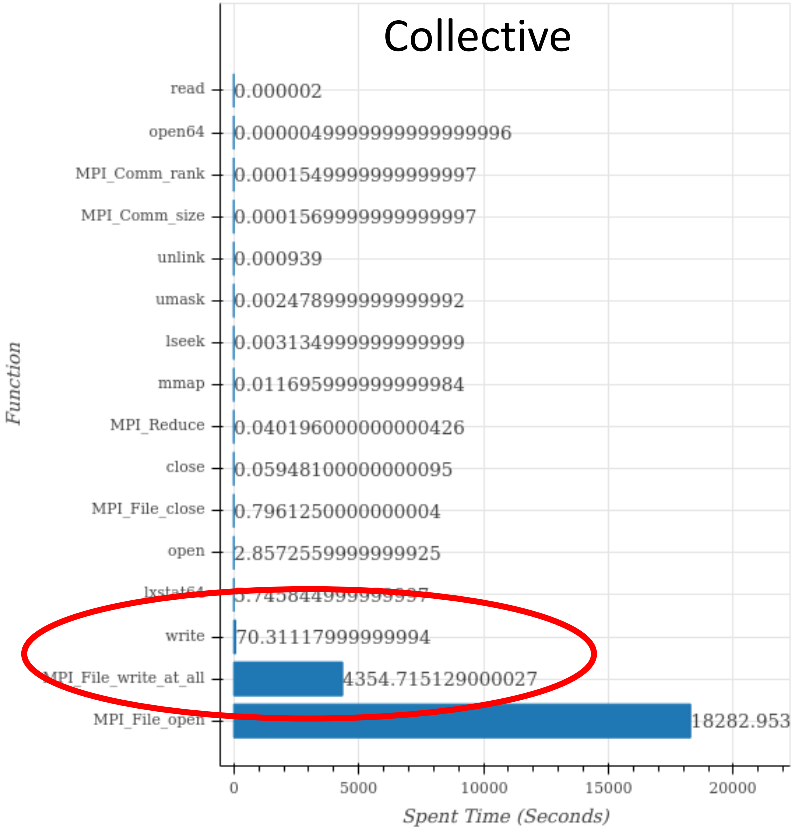
Stripe Size 128M

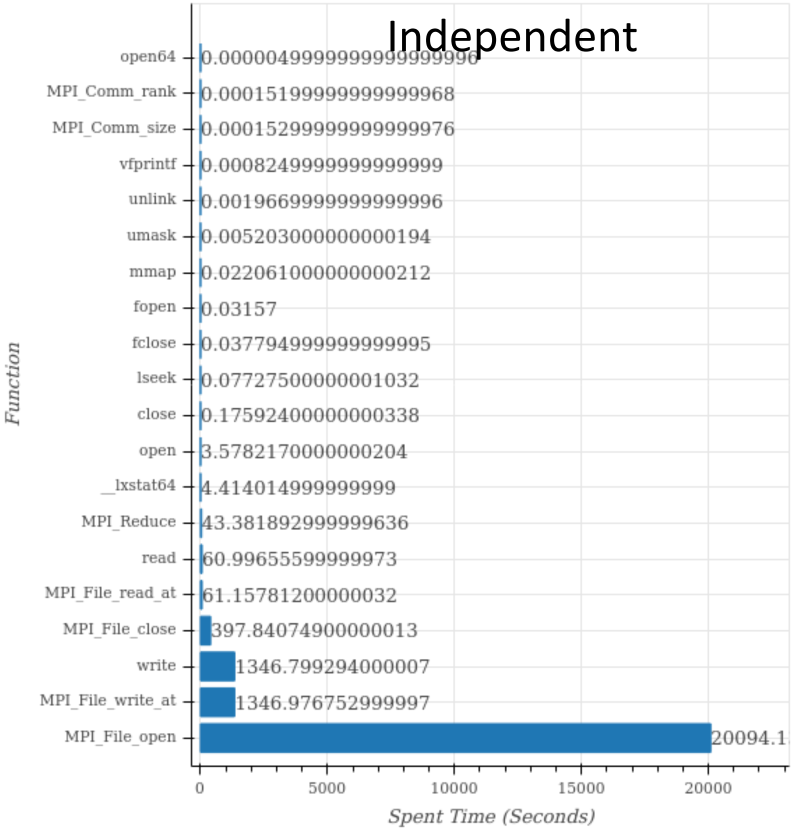


Stripe 512M

Collective vs. Independent

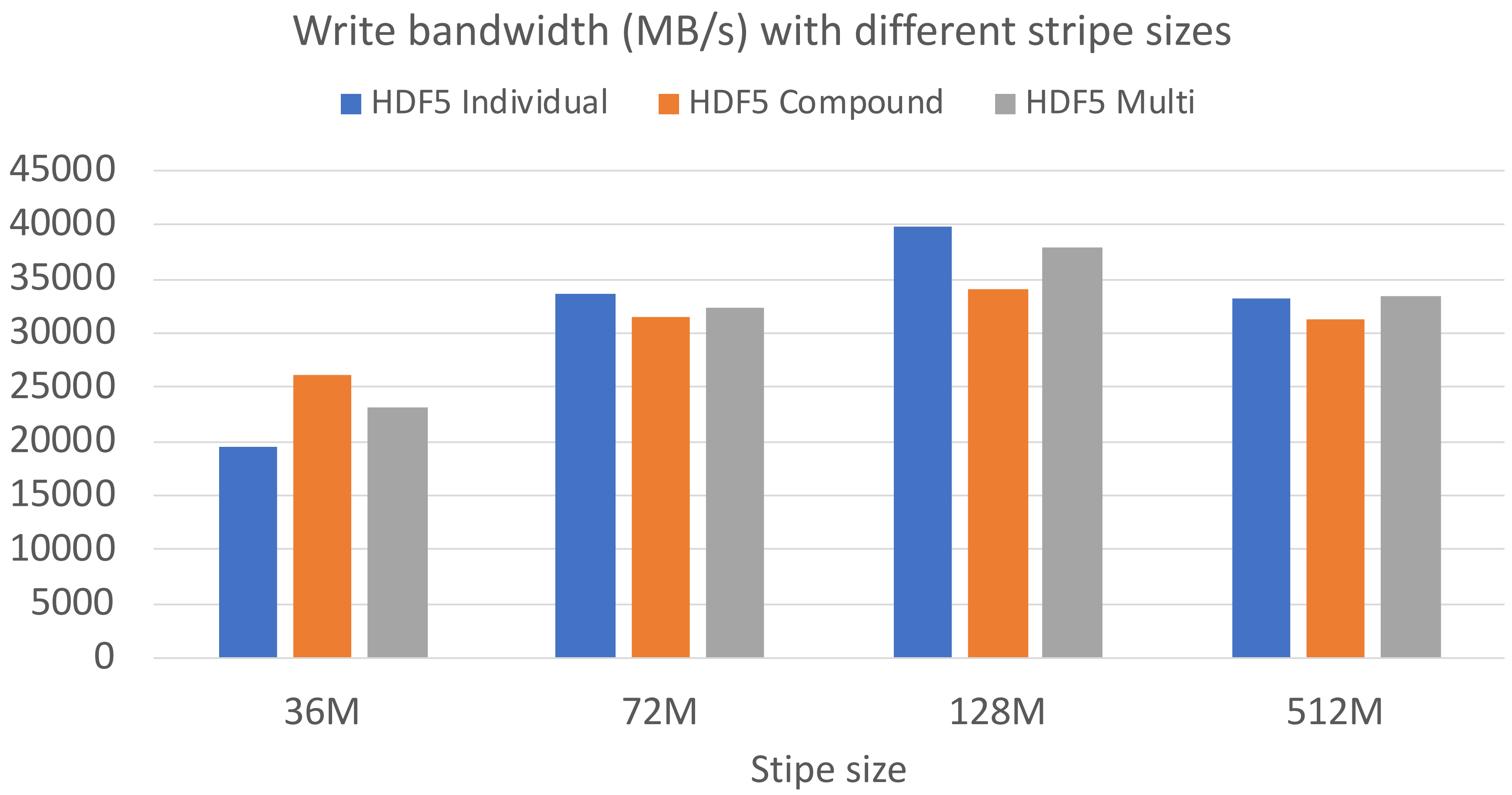
- Collective is beneficial for small writes --> merge into bigger writes.
- In our case, write size is big enough(8MB) to amortize disk seek time. The overhead of aggregation outweighs the I/O improvement.





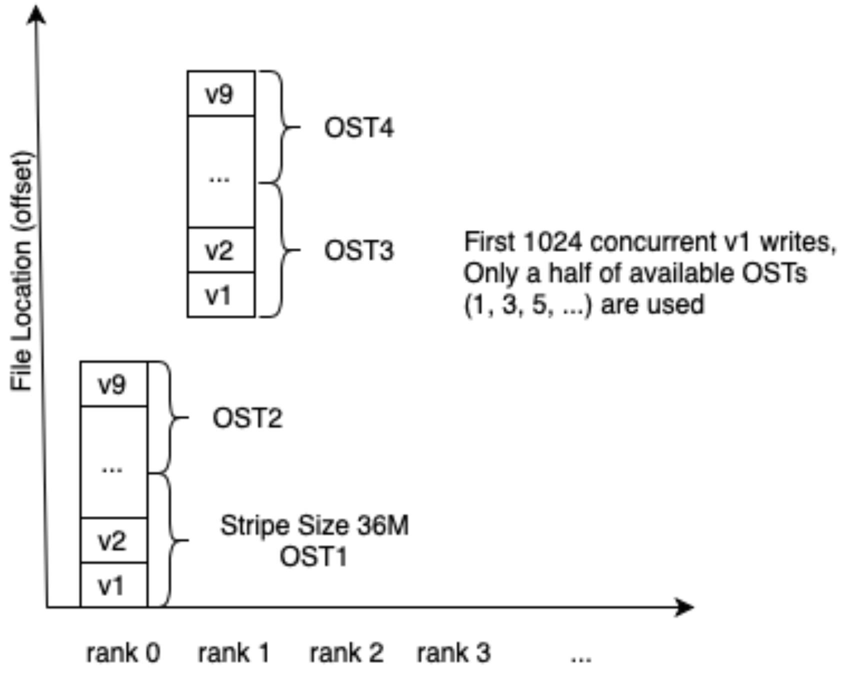
Optimal striping settings depend on access patterns

- On Lustre(Quartz) the best combination is HDF5 Individual + S128M tripe size



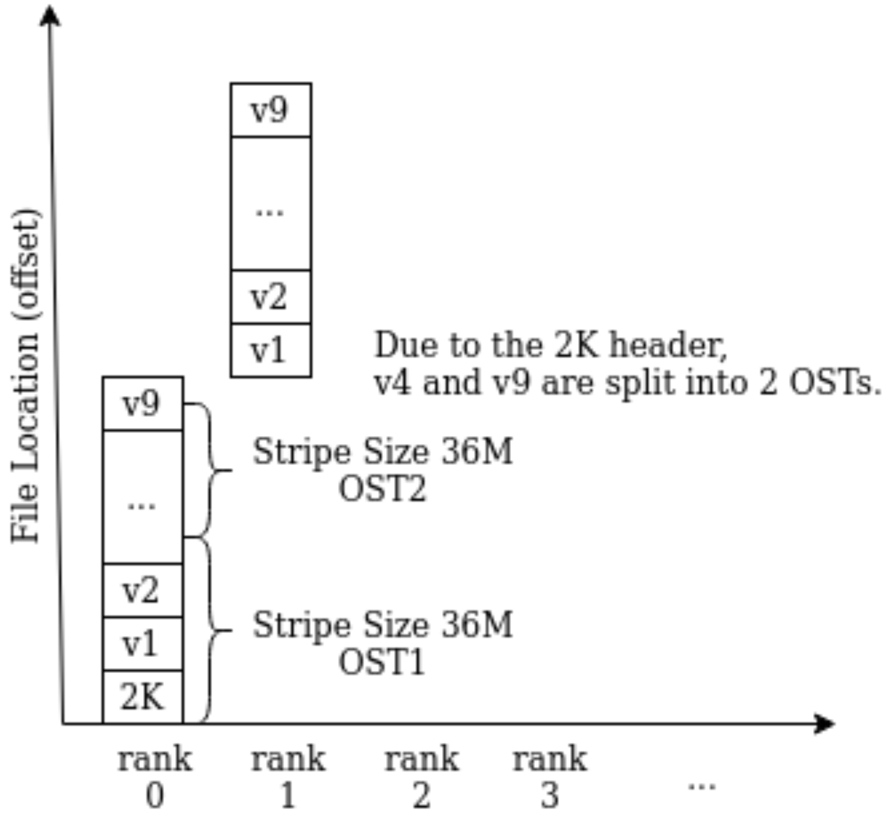
Why 36M Stripe size is bad for MPI Interleaved

- Wrong stripe size could hurt the performance
- 8GB/variable, 1024 ranks → each ranks has 8MB/variable, 72MB total.



MPI Interleaved, HDF5 Multi

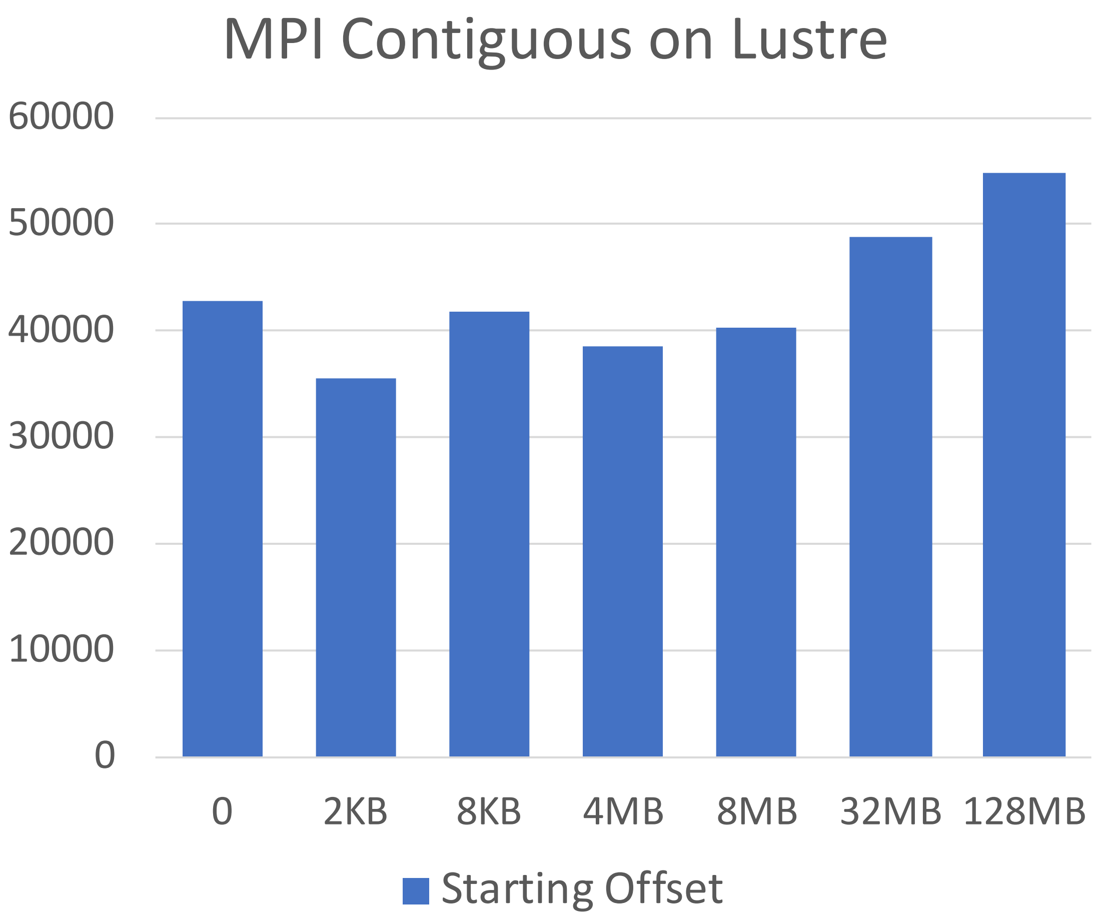
HDF5 with 2KB header, 36M stripe size

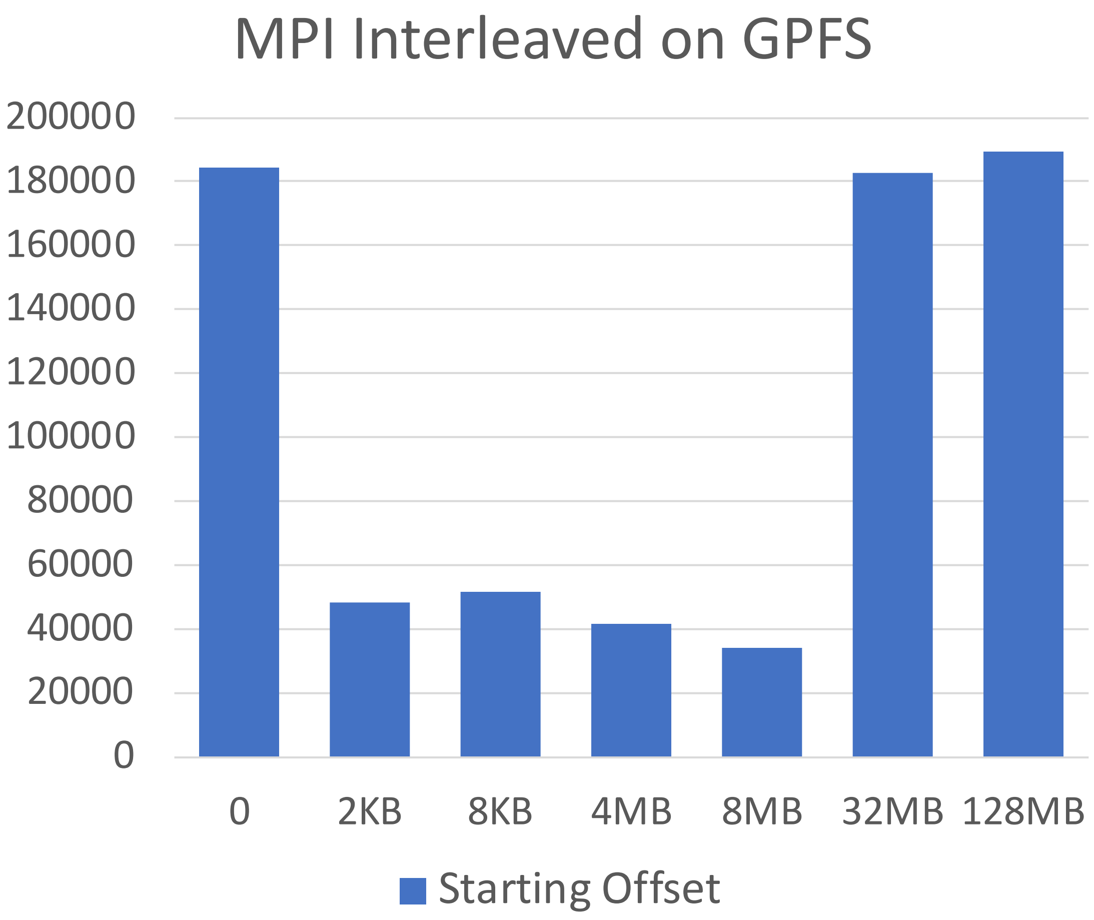


HDF5 Multi

Impact of the header

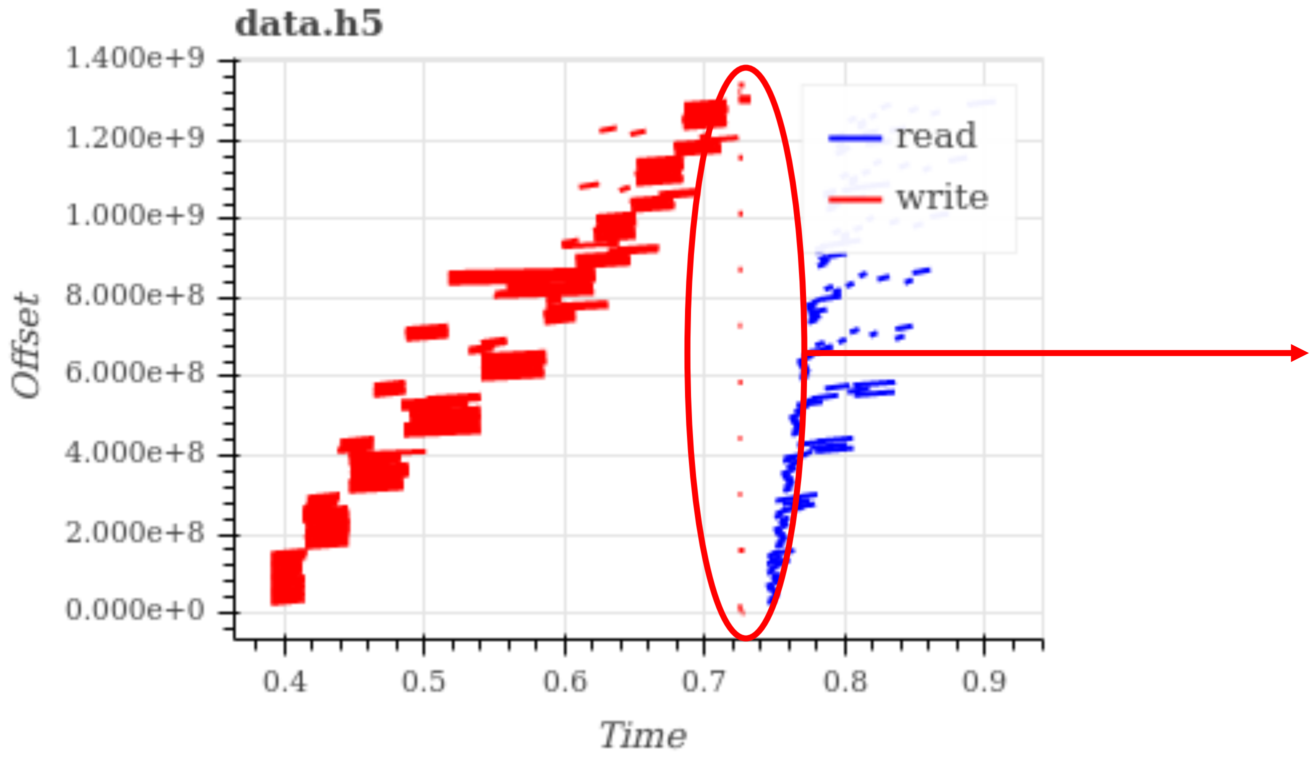
- Pure MPI-IO Implementation starting not from 0 but a specific offset.





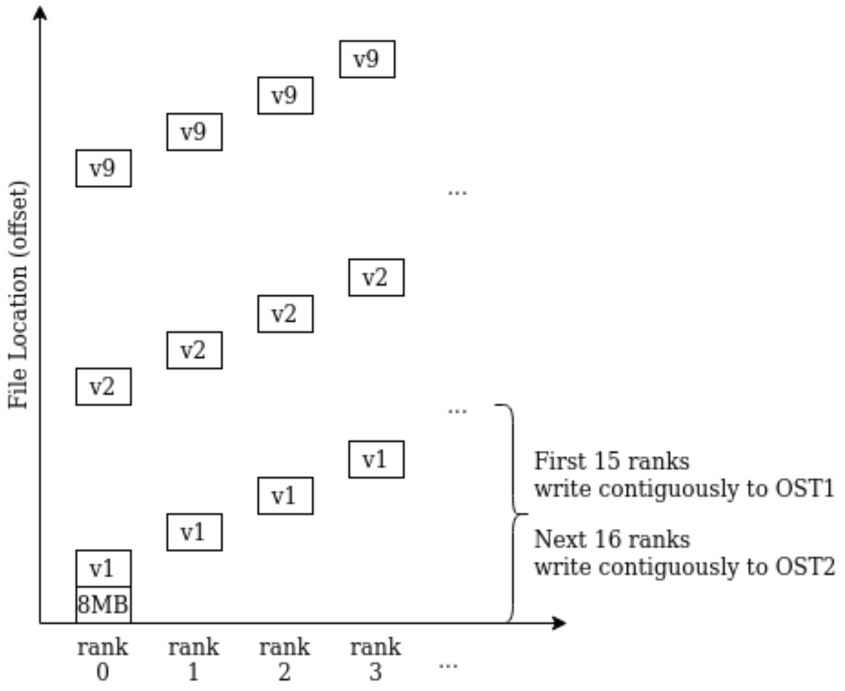
Impact of the header

- How to solve this issue?
  - **H5Pset_meta_block_size**
  - H5Pset_alignment
    - It does not solve the “holes” issue.
  - User split driver

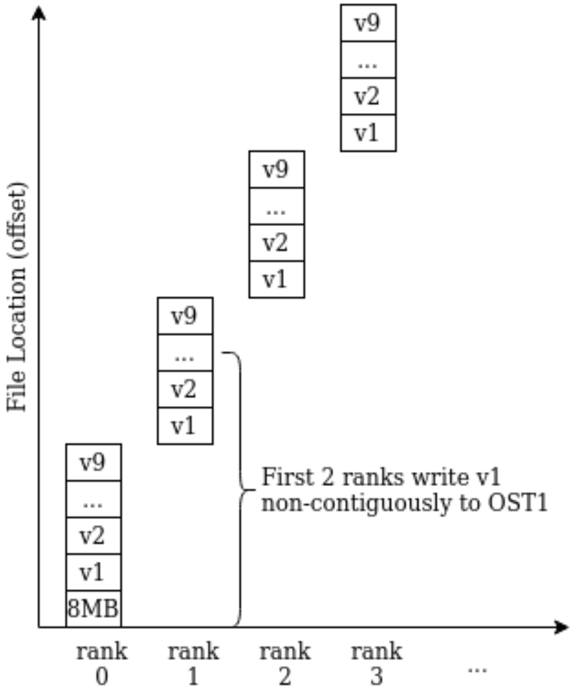


superblock + metadata for 9 datasets are written at the end

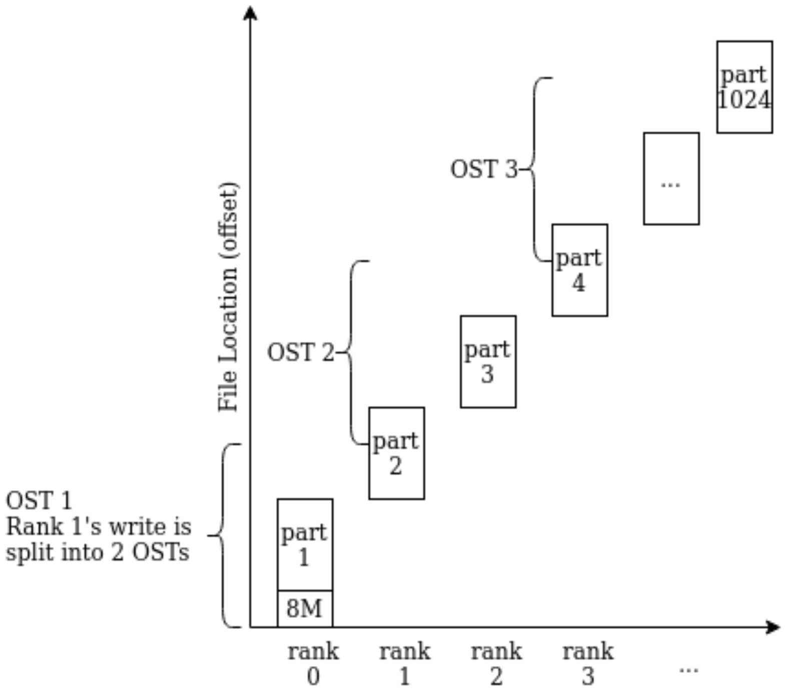
HDF5 with 8MB header, stripe size to 128M



HDF5 Individual



HDF5 Multi



HDF5 Compound

How to match the pure MPI performance

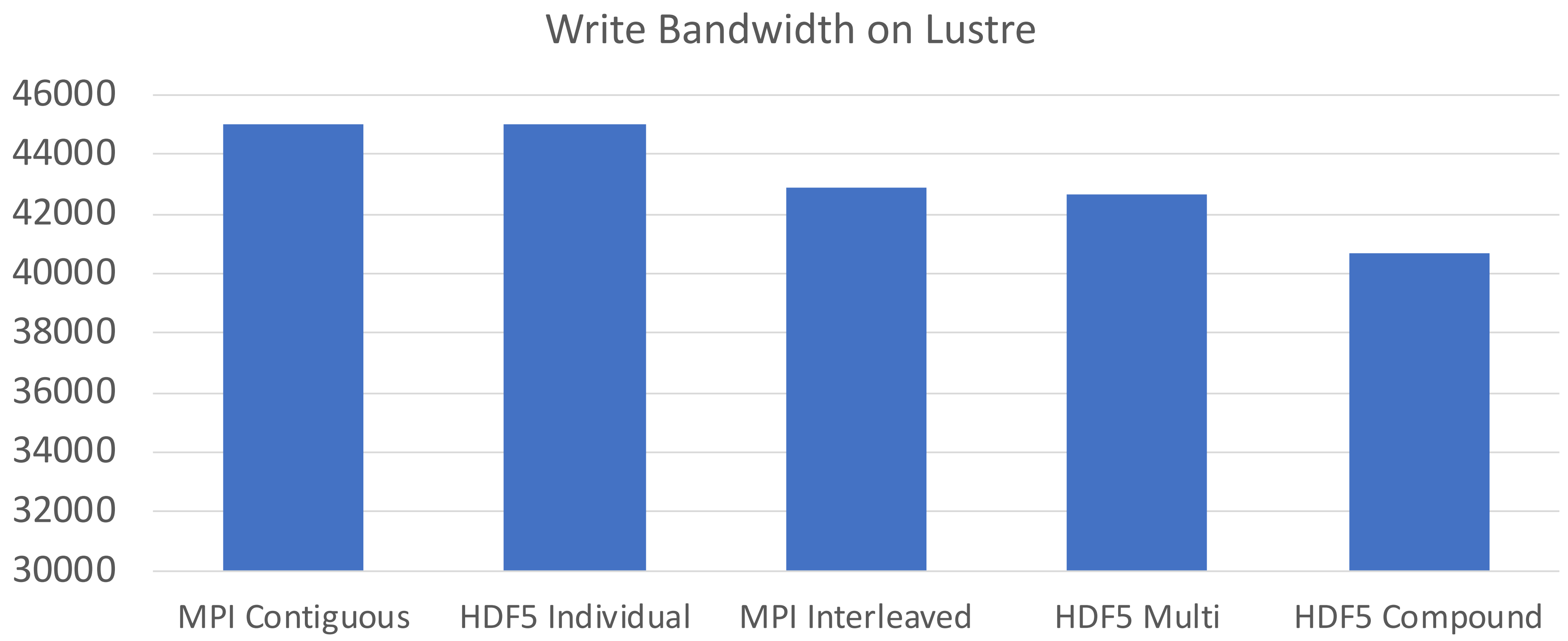
- First find the best stripe size and stripe count using the pure MPI implementation.
- **Decide which data layout gives the best performance**
  - On Lustre, MPI Contiguous (HDF5 Individual) is better
  - On GPFS, MPI Interleavd (HDF5 Multi) is better

How to match the pure MPI performance

- First find the best stripe size and stripe count using the pure MPI implementation.
- Decide which data layout gives the best performance
  - On Lustre, MPI Contiguous (HDF5 Individual) is better
  - On GPFS, MPI Interleavd (HDF5 Multi) is better
- **Tune HDF5 to match the performance of MPI implementation**

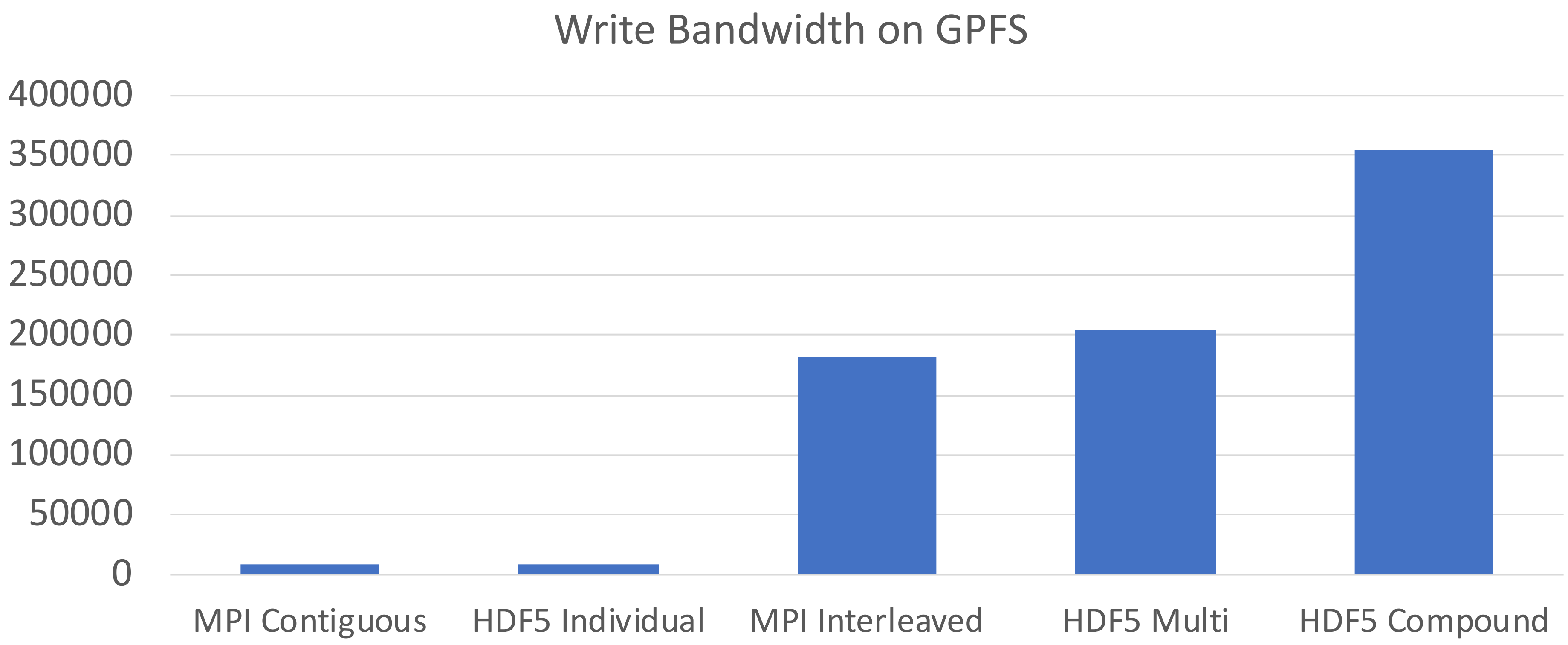
How to match the pure MPI performance

- **Tune HDF5 to match the performance of MPI implementation**
  - **H5Pset_meta_block_size(8MB) on Lustre**



How to match the pure MPI performance

- **Tune HDF5 to match the performance of MPI implementation**
  - **H5Pset_meta_block_size(32MB) on GPFS**



Conclusions

- Current version of HDF5 doesn't have a way to match the MPI-Interleaved access pattern. HDF5_Multi achieves the same pattern.
- Collective I/O does not help since the request size is already very big.
- MPI_Interleaved (HDF5_Multi) is better on GPFS.
  - Because the stripe size is small and users can not change it.
- MPI_Contiguous (HDF5_Individual) is better on Lustre.
  - Because we use a big stripe size (128M)
  - For smaller stripe sizes (e.g. < 36M) MPI_Interleaved is better.