Selection I/O in HDF5 Virtual File Drivers

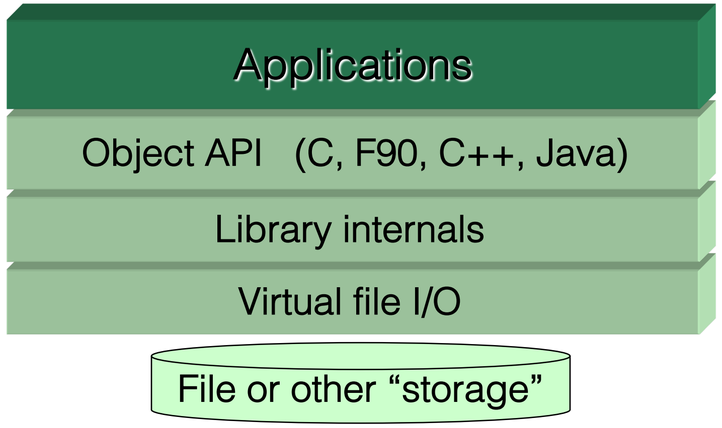
October 20, 2021

Neil Fortner, The HDF Group

The HDF Group

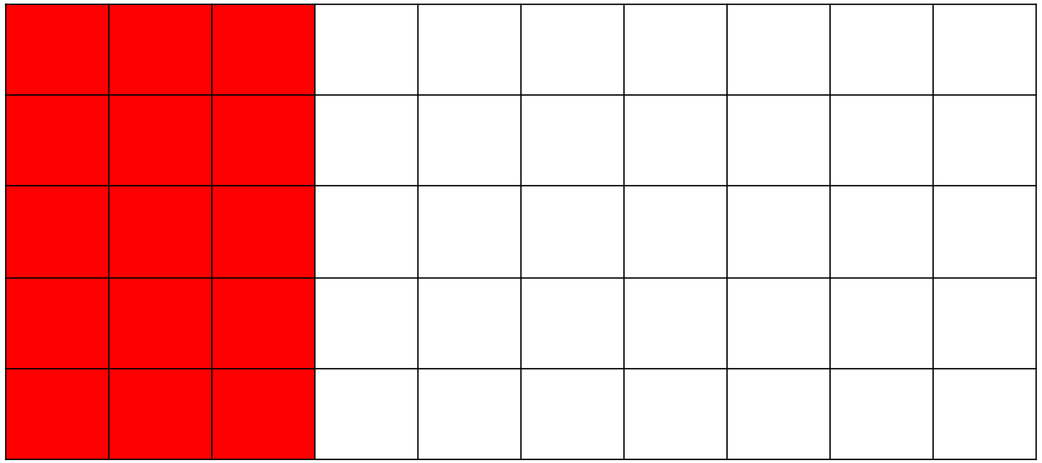
Virtual File Layer

- Sits between HDF5 library and filesystem
- Library provides single offset, length, and buffer for I/O, Virtual File Driver (VFD) is responsible for translating to underlying I/O system
- Direct map to single files on traditional POSIX like file systems



Virtual File Layer

- **Existing read/write interfaces for VFD**
  - `herr_t (*read)(H5FD_t *file, H5FD_mem_t type, hid_t dxpl, haddr_t addr, size_t size, void *buffer);`
  - `herr_t (*write)(H5FD_t *file, H5FD_mem_t type, hid_t dxpl, haddr_t addr, size_t size, const void *buffer);`

read() =

```
read(0, 3);
read(10, 3);
read(20, 3);
read(30, 3);
read(40, 3);
```

Motivation: Non-Contiguous I/O

The HDF Group

- **With the existing scheme, non-contiguous I/O must be broken into a single VFD call for each block of bytes**
  - Simple, effective for traditional file systems
- **More advanced storage systems can take advantage of having knowledge of the whole I/O request**
  - MPI I/O
  - Asynchronous I/O
  - Subfiling
  - Object stores
  - Etc

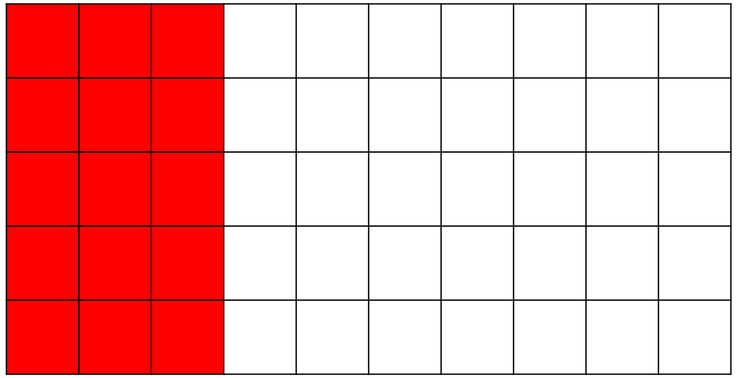
MPIO File Driver

- **The existing MPIO file driver takes advantage of non-contiguous I/O requests**
  - Code written in the library specifically for the MPIO driver packages information on the I/O pattern and passes it to the driver through an undocumented channel
  - Library makes a single call to the file driver with partly fake single block parameters
  - MPIO VFD unpacks information from the undocumented channel to obtain the real I/O pattern, passes it to MPI
  - This pattern undermines the principle motivation for the VFL, that end users can implement their own storage interfaces using public APIs
  - Must develop a generalized scheme for passing non-contiguous I/O requests to the VFD using public APIs

Method 1: Vector I/O

- **Instead of passing a single offset/length/buffer, pass vectors of each**
- **Simple extension, but inefficient for repeating patterns**
- `herr_t (*read_vector)(H5FD_t *file, hid_t dxpl, uint32_t count, H5FD_mem_t types[], haddr_t addrs[], size_t sizes[], void *bufs[]);`
- `herr_t (*write_vector)(H5FD_t *file, hid_t dxpl, uint32_t count, H5FD_mem_t types[], haddr_t addrs[], size_t sizes[], const void *bufs[])`

```
read(
```



```
) = read_vector({0, 10, 20, 30,
                 40}, {3, 3, 3, 3, 3});
```

Method 2: Selection I/O

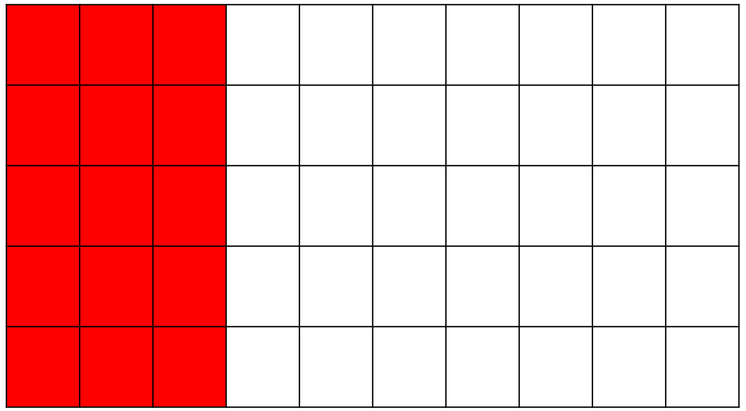
The HDF Group

- Pass HDF5 dataspace selections for file and memory (or a vector of them)
- More efficient for repeating patterns
- Taking full advantage of selection I/O in an external VFD will require new HDF5 API routines, which we are planning to develop

Method 2: Selection I/O

- `herr_t (*read_selection)(H5FD_t *file, H5FD_mem_t type, hid_t dxpl_id, size_t count, hid_t mem_spaces[], hid_t file_spaces[], haddr_t offsets[], size_t element_sizes[], void *bufs[] /*out*/)`
- `herr_t (*write_selection)(H5FD_t *file, H5FD_mem_t type, hid_t dxpl_id, size_t count, hid_t mem_spaces[], hid_t file_spaces[], haddr_t offsets[], size_t element_sizes[], const void *bufs[] /*in*/);`

`read(`



`) = read_selection(hyperslab(`
`(0,0), NULL, (5, 3), NULL));`

Current Status

- **In selection_io branch:**
  - H5FD_t file driver callbacks added
  - H5FD public API calls added
  - Library can pass selection I/O requests for most common I/O use cases
  - H5FD code can translate selection and vector requests to vector or scalar requests if the VFD does not support the I/O mode requested
  - MPIO file driver supports vector I/O, including in collective mode
  - No performance hit seen with selection I/O enabled (haven't finished implementing VFDs that could see improved performance yet)

Future Work

- Support remaining use cases in the library with selection and/or vector I/O
- Remove scalar I/O paths, especially for parallel
- Implement selection I/O in MPIO VFD
- Implement new routines to allow efficient retrieval of selection patterns
- Implement VFDs that benefit from selection I/O (subfiling)