RFC: H5FD_MIRROR

Virtual File Driver

Matthew Dougherty

The functional objective of the *Virtual File Driver (VFD)* **H5FD_MIRROR** is to relay local host **H5FD_SPLITTER VFD** calls to a remote host file system, resulting in identical and simultaneous creation of HDF5 files on both hosts. **H5FD_MIRROR** consists of three software components each executing as different processes: **mirror_S**, **mirror_R**, and **scheduler**. When an **H5FDopen** function call occurs, **mirror_S** sends a service request through the network to **scheduler** using a pre-defined static network port. This will cause **scheduler** to assign a new and dedicated network port for **mirror_S** and **mirror_R** to communicate on. For every **H5FDopen** request, **scheduler** will start one **mirror_R** process, instructing **mirror_R** to connect to **mirror_S**, and perform all subsequent **H5FD** functions relating to the open HDF5 file on the dedicated network port. Subsequently, **mirror_R** will invoke **H5FD_SEC2 VFD** to create the remote HDF5 file.

1) Introduction

In figure one below, the **H5FD_SPLITTER VFD** has two channels leading to the final **H5FD_SEC2** VFDs: a local **H5FD_SEC2** and a remote **H5FD_SEC2**. The purpose and restriction of **H5FD_MIRROR VFD** is to pass *write-only* **H5FD** functions across a network using **Sockets/TCP**.

The remainder of this RFC expands upon the requirements, design analysis, and describes issues involved in the implementation.

It should be understood that,

1) every **H5FDopen** causes **scheduler** to create a dedicated **mirror_R** process at the remote host, which will manage all **H5FD** functions subsequent to the specific **H5FDopen.**
2) multiple **mirror_R** processes may exist at the same time operating on different HDF5 files, by different users or applications.
3) a user application will normally provide the IP address of the **scheduler** and its dedicated static port in order to request remote services. Therefore, it is possible that file operations in the **remote abstraction** of figure one, can occur concurrently on different remote hosts by the same local host application.

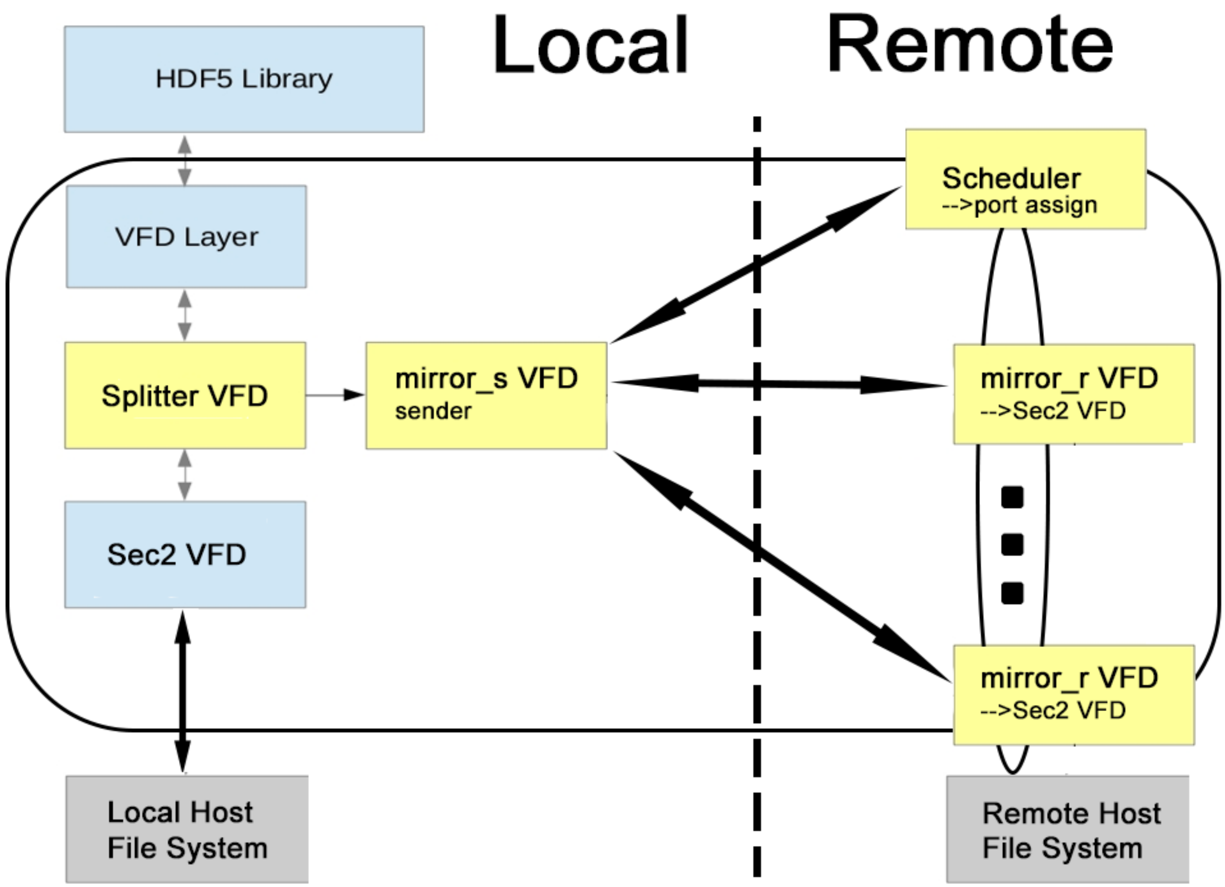

Figure 1

Block diagram illustrating the Mirror_VFD design. Blue boxes represent existing code, and yellow boxes code that must be designed and implemented.

2) Operational Methods

The **H5FD** functions supported by **Mirror_VFD** are:

1) H5FD__init_package
2) Configuration of the **Mirror_VFD**,
3) H5FDopen
4) H5FDwrite
5) H5FDflush
6) H5FDset_eoa
7) H5FDalloc
8) H5FDtruncate
9) H5FDclose

As a rule, all **H5FD** functions passed by **H5FD_SPLITTER,** are passed to the remote **H5FD_SEC2** without modification. Remedial error management is planned: any errors are either ignored, logged & ignored, or reported as errors requiring manual intervention.

2.1) H5FD__init_package(void)

This generic function is used by any **H5FD VFD** to register with **H5FD**. Through its execution, it initializes the **H5FD_MIRROR** software and provides internal **mirror_S** callbacks to **H5FD**.

2.2) Configuration of the **Mirror_VFD**: **H5Pset_fapl_mirror**(*char ipa, int np);

H5Pset_fapl_mirror allows a user to change the IP address and network port that **mirror_S** requires to establish **scheduler** communications. If the "**ipa"** is null, the default IP address is used (local loopback 127.0.0.1). If "**np"** is zero, the default network port is used (port 3000). It should be emphasized this does not change any configuration within **scheduler**, or cause **scheduler** to execute on any particular remote host. Changing the network port of Scheduler must manually be made within the **scheduler**. Running **scheduler** on a remote host must be started manually, which establishes its IP address.

2.3) H5FDopen: H5FD_t *(***open**)(const char *name, unsigned flags, hid_t fapl,haddr_t maxaddr);

The **H5FDopen** function causes the **mirror_S** to initiate a connection to **scheduler**. **Scheduler** assigns a dedicated port that all future **H5FD** functions relating to this particular **H5FDopen** will occur. **Scheduler** starts the remote **mirror_R** process instructing the process to connect to the prescribed network port, and begin servicing **mirror_S** beginning with a new **H5FDopen,** and subsequent H5FD functions for this HDF5 file. After the network connection is established between **mirror_S** and **mirror_R**, **mirror_R** opens the HDF5 file on the remote host file system using the **H5FD_SEC2 VFD**. Upon success, **H5FD_SEC2 VFD** returns the **H5FD_t** data structure, which is only maintained by **mirror_R**; that is, the **H5FD_t** data structure is not sent back to **mirror_S.**

2.4) H5FDwrite: herr_t (***write**)(H5FD_t *file, H5FD_mem_t type, hid_t dxpl_id, haddr_t addr, size_t size, const void *buffer);

The **H5FDwrite** function causes the remote **H5FD_SEC2 VFD** to write "**size"** number of bytes from data from "**buffer"** to the **H5FD_t** "**file",** beginning at the HDF5 offset address "**addr,"** in concert with the data transfer properties defined by **"dxp_id"**.

Since the Sec2 VFD doesn't use it, the prototype sender will not transfer the dxpl to the receiver.

2.5) H5FDflush: herr_t (***flush**)(H5FD_t *file, hid_t dxpl_id, hbool_t closing);

H5FD_SEC2 VFD performs no flush function. **mirror_R** performs no other action.

2.6) H5FDset_eoa: herr_t (***set_eoa**)(H5FD_t *file, haddr t)

The **H5FDset_eoa** function causes the remote **H5FD_SEC2 VFD** to set its **E**nd **of A**ddress space as defined by **"t".**

2.7) H5FDalloc: (***alloc**)(H5FD_t *file, H5FD_mem_t type, hsize_t size)

The **H5FDalloc** function causes the remote **H5FD_SEC2 VFD** to allocate space in the HDF5 file as defined by **"size"** and memory **"type".**

Note that the H5FDalloc() call raises issues for the splitter, as there is no requirement that two different VFDs will allocate the same space given the same inputs. While this is not an issue in the prototype, as we will be using the sec2 VFD on both sides, for a production version we will need lists of compatible VFDs.

2.8) H5FDtruncate: herr_t (***truncate**)(H5FD_t *file, hid_t dxpl_id, hbool_t closing);

The **H5FDtruncate** function causes the remote **H5FD_SEC2 VFD** to truncate the file to a size defined by the **H5FDset_eoa** or **H5FDalloc**, in concert with the data transfer properties defined by **"dxpl_id"**. If the file is larger than this size, any data past the new EOF is lost. If the file is shorter, then it is extended, and the extended part padded as null bytes ('\0'). The file offset is not changed.

2.9) H5FDclose: herr_t (***close**)(H5FD_t *file);

The **H5FDclose** function causes the remote **H5FD_SEC2 VFD** to close the file on the remote host. After which, the **mirror_S /mirror_R** network connection is closed, and the **mirror_R** process terminates.

3) API Additions

herr_t (H5Pset_fapl_mirror)(* char ipa, int sp)

Users may optionally override the default IP address and server port.

Future development using fapl parameters will allow for adjusting the remote **H5FD_SEC2 VFD** properties, **scheduler** properties, or **mirror_R** properties.

4) Implementation Details

The **H5FD_MIRROR VFD** will be prototyped on Linux laptops utilizing Sockets/TCP.

4.1) Justification for Sockets/TCP over Sockets/RDMA

Infiniband, iWarp and ROCE RDMA were closely looked at. In the long term these methods offer the highest technical performance possible, potentially by a factor of 10x; at the same time being they are “zero-copy, they require very limited CPU resources. Due to the relative simplicity in this prototype software implementation and project time constraints, Sockets/TCP was chosen. Later on, should Infiniband, iWarp or ROCE RDMA be required, it should not involve difficult software changes, because many of the RDMA APIs verbs use the same names and parameters.

4.2) Sockets/TCP activity for mirror_S and mirror_R

The **H5FD_SPLITTER VFD** directs **H5FD** function calls to a local hosts **H5FD_SEC2 VFD** and **the H5FD_MIRROR VFD**. After **mirror_R** is assigned a dedicated network port by **scheduler**, **mirror_S** establishes a connection and transmits the write-only **H5FD_SPLITTER VFD** functions to **mirror_R,** beginning with **H5FDopen**. Each dedicated **mirror_R** process will perform a single HDF5 file open, and all subsequent **H5FD** functions associated with this **H5FDopen,** are performed by the remote **H5FD_SEC2 VFD**.

Two types of Socket/TCP interactions are planned for the prototype. The first is to schedule an exclusive network connection between **mirror_S**, and resulting in potentially multiple **mirror_R** processes created though **schedule** upon each **H5FDopen.**

The second type of connection is a dedicated network channel to pass all **H5FD** functions and related data. Most **H5FD** functions require only one data transmission from **mirror_S** to a **mirror_R** to execute one **H5FD** function by **mirror_R** and **H5FD_SEC2 VFD**. The exception being the **H5FDwrite** function because of the write data buffer. This is provided by a second data transmission.

4.3) The chronology of Sockets events

1) The **scheduler** process is executing on the remote host, monitoring a static network port that is dedicated to receiving **mirror_S** service requests***.***
2) Upon successful synchronization, **scheduler** will assign a network port in which further **H5FD** functions associated with the **H5FDopen** will be handled.
3) This port number is sent to **mirror_S**, and to the **mirror_R** process which **scheduler** starts.
4) **mirror_S** and **mirror_R** then establish an exclusive network connection.
5) When **mirror_S** transmits an **H5FDopen** function to **mirror_R,** this causes the remote **H5FD_SEC2 VFD** to initialize the needed remote H5FD data structures. These data structures are only maintained on the remote host by **mirror_R.**
6) Subsequent **H5FD** functions are received by **mirror_R** and are executed by the remote **H5FD_SEC2 VFD.**
7) When **mirror_S** transmits a **H5FDclose** function, the remote **H5FD_SEC2 VFD** performs the file closure, after which **mirror_R** closes the network socket and exits the **mirror_R** process.

5) Outstanding issues to be considered further

1) How will write buffer sizes impact network performance?
2) What network errors might occur and what is the impact operationally?
3) What are timeout margins?
4) What might be needed for error management?
5) What is the need for local/remote file comparisons during **H5FD** writes or closure?
   - simplest: byte count comparison of remote and local file sizes, which should be identical.
   - advanced: file checksum comparison during close.
6) Is the use of error detection/error correction plugin filters advised if network errors abound?
7) Propose methods if the remote file is corrupted, such as requiring the local HDF5 file to be sent using system level OSI network layer 7 methods (e.g., bbcp) at a later time.
8) Need a mechanism for recycling ports after file close.

6) Testing

6.1) Test #1 will be performed on a single laptop loaded with Centos operating systems. A local loop back IP address will be used. Test software will be written independent of **H5FD_SPLITTER VFD**, capable of performing basic write-only **H5FD** functions to verify general software code reliability.

6.2) Test #2 will be performed on two laptops networked with a crossover cable, each loaded with Centos operating systems, and HDF5 libraries with the **H5FD** changes. The “local host” will perform the standard HDF5 regression test cases. The objective of the test is to evaluate functional file drivers simultaneously, by creating identical HDF5 files across the network.

6.3) Test #3 will use the same hardware configuration of **test #2**, but will include the **H5FD_SPLITTER VFD** regression test suite. The objective of the test is to evaluate the integrated stacked **VFDs** simultaneously creating identical HDF5 files across the network.

7) Recommendation

Upon successful testing, it is recommended further software upgrades could be made to the networking design optimizing for equipment (e.g., ESnet, SLAC and NERSC), software (e.g., psana) and operational facility protocols/permissions. This objective would be to evaluate network high performance and needed operational upgrades at the desired user locations.

8) Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under Contract Number DE-AC02-05CH11231.

9) Revision History

Oct. 5, 2018: Version 2 circulated for comment.