Making H5VL Multi-Thread Safe:
A Sketch Design

John Mainzer
Lifeboat, LLC

12/03/22

Introduction

Conceptually, H5VL is fundamentally a straight forward module. Its purpose is to route HDF5 API calls to the appropriate VOL connector for execution. Thus, the majority of its multi-thread issues are inherited either from its dependencies or from the underlying VOL connectors.

It will be impossible to fully address the multi-thread issues of the VOL Layer (H5VL) until the multi-thread issues of the supporting packages are well understood. Even then, much effort will be required to move the global mutex down into the VOL Layer (and possibly into the VOL connectors) to allow the desired multi-thread operation in VOL connectors that support it.

Thus, for now, this document is little more than a list of know issues, combined with some musings on how the mutex issue might be addressed.

A Quick Overview of H5VL

In essence, H5VL is a switch board relaying HDF5 API calls to the appropriate VOL connector, or in some cases, relaying the API calls through the specified stack of VOL connectors. While it is similar to the VFD layer (H5FD) in certain ways, it is vastly more complex due to much richer HDF5 API. Further, for objects that are implemented different ways between different VOL connectors, it must perform the necessary wrapping and unwrapping so that the different variations may be stored using the same utilities (e.g., H5I).

At present, the HDF5 global mutex is acquired on entry to any HDF5 API call. As our objective is to relax this requirement for VOL connectors (or portions of VOL connectors) that are multi-thread safe, H5VL will have to be modified to support this. The details of a clean, maintainable solution for this problem are unknown pending multi-thread revisions for the minimal set of packages required for initial functionality.

While the management of the HDF5 global mutex must be put to one side for now, a brief discussion of the multi-thread issues surrounding the VOL layer will help direct the multi-thread modification of the packages that H5VL depends on.

Multi-Thread Issues in H5VL

From my initial review, it seems that three type of multi-thread issues in H5VL: use of other, potentially non-multi-thread safe packages, issues internal to H5VL proper, and the need to acquire the global mutex whenever directing control flow into a VOL connector that is not multi-thread safe.

Use of other HDF5 packages by H5VL

H5VL makes use of the many packages, which are best discussed in the following groups:

- H5MM
- H5FL
- H5E
- H5I
- H5P
- H5CX

As usual, issues with H5MM and H5FL may be avoided by using C memory allocation functions directly, and by either not maintaining free lists, or doing so internally.

Once they are made multi-thread safe, calls to H5E, H5I, and H5P are non-issues, as long as H5VL can refrain from holding locks during these calls. There are calls to H5I_iterate() in H5VLint.c, which should be replaced with more thread friendly calls once these are added to H5I.

Some of the not obviously multi-thread safe H5CX API calls (e.g., H5CX_retrieve_state / restore_state()) are used. More study of H5CX is needed to see the multi-thread implications of this.

In addition, there are several utility packages that not on our list for immediate conversion to multi-thread safety.

- H5PL
- H5ES
- H5S
- H5SL

The first three of these (H5PL, H5ES, and H5S) are all on our list for conversion as time and resources permit, but are not seen as critical for development of initial multi-thread VOL connector support. Calls into these packages will have to be protected by the global mutex to exclude the possibility of multi-thread processing in these packages pending conversion.

In contrast, H5SL (skip lists) may never be converted to multi-thread safety due to the performance issues that have dogged it. Instead, it will almost certainly be replaced with some other thread safe data structure. In H5VL, it is used in the registration / deregistration of dynamic operations. Unless this proves to be a common and costly operation, this will probably be protected by the HDF5 global mutex and left for another day.

Finally, there are the many HDF5 native VOL calls that appear in the H5VLnative*.c files.

- H5A
- H5AC
- H5C
- H5D
- H5F
- H5G
- H5HG
- H5M
- H5O
- H5PB
- H5T

These will have to be protected by the HDF5 library global mutex to preclude multi-thread execution. While this is conceptually straight forward, the number and complexity of these calls present a challenge – whose solution is still very much undecided.

Multi-thread thread issues in H5I proper

A review of the H5VL public, private, and developer APIs reveals the following multi-thread safety issues in the current H5I implementation.

Global Variables:

H5VL uses global variables for convenience to store the ID assigned to the native and passthrough VOLs (H5VL_NATIVE_ID_g and H5VL_PASSTHRU_g). These variables are used to note whether the target VOL connectors have been registered, and if so, to store their IDs.

This isn't a major issue, but to avoid the possibility of duplicate registrations, some sort of mutual exclusion is needed.

There are also the H5VL_native_cls_g and H5VL_pass_through_g global tables, but as they are constant, they are not a multi-thread issue,

VOL Connector Registration and Deregistration:

While it isn't common, VOL connector registration / deregistration must be protected to avoid duplicate registrations. As indicated above, at least initially, the relevant code in H5PL will be operated behind the HDF5 global mutex. Care must be taken to either verify that this will be sufficient to avoid duplicate registrations, or to otherwise avoid the problem.