RFC: Multi-Thread HDF5

John Mainzer
Gerd Heber, Chris Hogan, Elena Pourmal, Dana Robinson

At present, the HDF5 library is not thread safe. To allow its use by multi-threaded applications, in the "thread safe" build, the library is equipped with a global lock that allows only one thread into the library at a time – effectively making the entire HDF5 library a giant critical region.

This lack of thread safety has been a known issue for longer than the HDF group has existed as an independent entity. To date, we have made no significant effort to address the problem, due both to the perceived difficulty of the problem, and to the lack of resources.

Recently, it has become evident that well-chosen partial solutions may have significant immediate utility. Further, there has been interest in the implementation of thread safety for a small subset of the HDF5 API – most particularly data set reads.

This RFC is an attempt to define a strategy for retrofitting thread safety[1] on the HDF5 library, that provides immediately useful partial solutions, provides support for limited, multi-threaded dataset reads, and does not impose significant extra costs on other HDF5 development projects or maintenance while in progress.

1 Introduction

Multi-thread programming, like MPI programming, is difficult. The developer must reason about the multiple threads of execution, and ensure that interactions between them do not corrupt data structures or generate undesired results.

This is hard enough with new code. Retrofitting thread safety on existing programs is more difficult, as it requires near perfect knowledge of the architecture and code base. Lack of developer level documentation makes this even more difficult.

The HDF5 library is an example of this last category – it is a large, complex library with little developer level documentation. The one saving grace is that it is broken into numerous packages, some of which are "leaf" packages – by which we mean that they make no calls into other packages. This in turn means that they can viewed as independent entities for purposes of retrofitting thread safety.

[1] In this document, we interchangeably use "thread safe", 'thread safety", "multi-threaded" terms with the meaning of "concurrent threads are allowed in the HDF5 library without corrupting data in memory and in storage".

This observation suggests a strategy of retrofitting thread safety package by package, starting with "leaf" packages and then working upwards. While this is not as easy as it seems – for example, cycles exist between internal packages – it is a plausible approach which should minimize both overall cost and interference with other HDF5 development projects. However, it has the twin major deficits of being expensive, and not providing any tangible benefit until thread safety can be pushed up to the API level. For these reasons, it has not been attempted.

1.1 Recent Developments

Recently, there have been two developments that may permit a project to retrofit thread safety on the HDF5 library to return tangible benefits sooner rather than later.

1.1.1 VOL Layer

The first of these is the Virtual Object Layer (VOL) layer. Conceptually, it can be thought of as an abstraction layer driven across the HDF5 library just below the API calls. It allows development of "VOL Connectors", that use the HDF5 API and some utilities provided by the library to implement data storage in an arbitrary format and on arbitrary device(s) while supporting the HDF5 API and data model. This gives HDF5 applications access to these storage systems without code changes. Examples include the DAOS VOL connector (supporting HDF5 on DAOS) and the REST VOL connector (supporting HDF5 on HSDS). Such VOL connectors are said to be terminal, as they handle the actual data I/O.

The VOL layer also makes possible "Pass-through" VOL connectors, which act on API call streams on the way to the terminal VOL connectors (or VOLs). Such VOLs already exist for purposes of logging, caching, and adding support for asynchronous operations on the API call level.

As a point of terminology, that portion of the HDF5 library that manages standard HDF5 files is frequently referred to as the native VOL. At present, it is coupled with the VOL layer and the utilities provided both to it and other VOLs.

Finally, hybrid VOLs that are both terminal and pass though also exist – although to date they have only been used as scaffolding for development / proof of concept work.

Unfortunately, the VOL layer is below the global lock, and thus calls to VOL connectors are serialized, even if the VOL connector proper is thread safe. However, the global lock could be moved down to the native VOL connector if a relatively small number of packages were made thread safe – specifically:

- H5VL – VOL Layer
- H5E – Error handling
- H5CX – Context
- H5I – Index
- H5P – Property Lists

With the exception of H5VL, all of these packages are high on the list for retrofitting thread safety in the bottom up approach that we have considered and rejected. By adding H5VL to the list, and retrofitting thread safety on all of them, we could enable multi-thread operation for all thread safe VOL connectors.

1.1.2 Sub-Filing

For the past year, we have been actively engaged in developing sub-filing facilities for the HDF5 library. After reviewing the last attempt, we elected to implement sub-filing at the Virtual File Driver (VFD) layer to make sub-filing as configurable and flexible as possible, and thus avoid the rigidity that limited the value of the first attempt.

During the design work, it became apparent that to attain the desired performance and flexibility, we would have to make the VFD layer and selected VFDs thread safe. Fortunately, we were able to bypass this in the initial implementation. However, it will be necessary to provide the configurability required for acceptable performance on a wide variety of machines and applications.

Note that the VFD layer's (H5FD package) only significant dependencies on other packages are H5E, H5CX, and H5P. (The in-progress work on selection I/O[2] will add a dependency on H5S (selections)).

1.2 Adding Calls for Thread Safe Data Set Reads to the Mix

In recent years, we have received expressions of interest in adding multi-threaded support for limited sections of the HDF5 API – say multi-thread support for reading contiguous data sets of scalar type without support for variable length data, object and region references, and type conversion.

As shall be seen, if we have a multi-threaded VOL layer and VFD layer, this should be possible with:

- No API changes,
- Pathway for a least limited expansion beyond the initial multi-thread support, and
- Threads not employing the multi-thread enabled API calls interact with the HDF5 library as usual.

1.3 Outline of the Remainder of this RFC

So far in this RFC, we have outlined the current state of play with regards to retrofitting thread safety onto the HDF5 library, along with some recent developments that may allow us to make some progress on the issue.

In the next section of this RFC, we outline a strategy that adapts to recent developments to:

- make lasting progress on retrofitting multithread support,
- provides thread safety in targeted areas that will offer an immediate return,
- avoid new technical debt and/or significant drag on other, unrelated development efforts, and
- allow graceful management of breaks in development.

To the extent that we can tell without more extensive study of the code in question, we think that the proposed strategy is workable – at least to the point of addressing the use cases outlined above. However, we must not forget that retrofitting thread safety on existing complex and poorly

[2] "Selection I/O" feature allows HDF5 VFD layer to see full I/O request issued by application.

documented code is challenging. Thus, allowing for breaks in development (or even abandonment of the project) is a major requirement. As shall be seen, the package by package approach should allow this.

The next section provides a high-level overview of the strategy and the work to be done. More detailed analysis of the work to be done is on hold pending strong interest in this approach.

2 Conceptual Overview

In a nutshell, the strategy we propose is the package by package approach discussed in the introduction, but with initial packages chosen to enable multi-thread VOL connectors. This allows us to implement multi-threading for limited API calls (for example, reads of contiguous data sets of scalar type, without type conversion, variable length data, or references, etc. TBD) via a hybrid VOL that routes around non-thread safe sections of the HDF5 library. For flexibility, the hybrid VOL would use the VFD layer for I/O – which must be retrofitted for thread safety as well. API calls for which multi-threading has not been implemented are routed to the native VOL as usual.

Below we list the minimum set of modules that will need modification:

1. Make H5VL (VOL Layer), H5CX (context), H5P (property lists), H5E (error reporting), and H5I (identifier) packages thread safe. This will allow us to move the global lock down to the native VOL[3].
2. Make the VFD layer (H5FD) thread safe, along with a minimal set of VFDs. Note that the VFD layer's thread safe dependencies (H5E, H5P, and H5CX) will have already been made thread safe at this point. (Note that selection I/O will add H5S to this list.)
3. Construct a Hybrid VOL (known as the Bypass VOL), that routes around the non-thread safe portions of the HDF5 library to provide multi-thread capability for limited cases of the targeted API call(s), and routes all other API calls to the native VOL.

On completion of this list, we will have retrofitted thread safety on the listed modules, enabled multi-threaded VOL connectors, provided the thread safe VFD layer needed by sub-filing, and added support for multi-threaded execution for limited cases of the targeted API call(s)[4].

From this point, there are at least two paths towards a fully thread safe version of the HDF5 library. Both will be long hard slogs, but even if we make no further progress, this initial step is of significant value.

All the above beg the questions of

- How will we manage package conversions to thread safety to as to minimize overhead imposed on other packages, avoid back sliding, and provide a path forward towards thread safety for the entire HDF5 library?

[3] This is a bit of an over simplification. Depending on complexity of I/O calls, we may need to make some other packages, for example, the H5S (selections) thread safe. However, these package(s) can be guarded by the global lock until they are made thread safe.
[4] I.e. reads of contiguous data sets of scalar type, and without type conversions, variable length data or references.

- How does the Bypass VOL work?
- How can we proceed toward thread safety for the entire HDF5 library?

We address these questions at a high level in the following sections.

2.1 Package by Package Conversion to Thread Safety

Recall that the package by package strategy for retrofitting thread safety starts by retrofitting "leaf" packages for thread safety, and then works its way in. Ideally this would allow us to address thread safety for one package at a time, as either the package would call no other packages, or all packages that it calls would already be thread safe. Further, once a package was converted to thread safety, neglecting maintenance, it would be finished business.

The hidden assumption here is that package dependencies are tree structured – which sadly is not the case. For example, cache client calls into the metadata cache can trigger actions in other cache clients, which can trigger re-entrant calls into the metadata cache, and so on. Taming this hairball will likely require significant re-architecting of metadata management to break the multi-thread conversion problem into manageable chunks. Indeed, if the effort to retrofit thread safety onto the HDF5 library founders, it may well be on this rock.

Fortunately, all of the packages to be addressed in the initial effort (H5VL, H5E, H5CX, H5P, H5I, and H5FD) are either "leaf" packages, or are well mannered internal packages that don't display the sort of pathological behavior discussed above. Thus, for this portion of the project at least, retrofitting thread safety can proceed as follows:

1. Pick a package[5] that is either a leaf package, or that only makes calls into packages that have already been retrofitted for thread safety.
2. Analyze the package, and modify as necessary to ensure thread safety[6] while not making any functional changes[7]. Further, these modifications must be performed with an eye to avoiding lock ordering issues as more packages are retrofitted for thread safety[8]. Document the changes required to support thread safety. The modified package must pass existing regression tests.
3. Write the necessary regression tests to verify thread safety (to the extent that this is possible). At least initially, these tests will likely be directed at verifying thread safety of code accessing internal data structures, and of calls into the package. Where thread safety touches the HDF5 API, we will require regression tests at that level as well. These tests will be compute intensive, and thus may only exist in token form in the daily regression tests. The full versions will have to be run whenever the package is modified.

[5] Observe that the VOL Layer calls arbitrary VOL connectors, that may or may not be thread safe. VOL Connectors that are not thread safe must be protected with a global lock – effectively converting them into giant critical regions.
[6] The details of this modification are TBD, and will be package dependent.
[7] While this should be possible in the initial effort, it will not be possible in general. In this case, we will re-architect as necessary.
[8] Which may require re-architecting.

4. Repeat until done.

Once a package is retrofitted for thread safety, in can be merged into the HDF5 develop branch. Un-related projects that touch the package must ensure that they do not break thread safety. However, with appropriate documentation and regression tests, this burden should be minor in most cases.

Since the thread safe version of the package can be used with either single thread or multi-thread builds of the HDF5 library (possibly with some conditional compilation to avoid unnecessary overhead in the single thread case), there is no code duplication. Further, the thread safety regression tests should prevent (or at least minimize) back sliding.

Unless re-architecting is required to support thread safety in packages that are addressed at later date, retrofitting thread safety on the target package should be finished business at this point.

While the package by package approach has the advantage of allowing us to address the thread safety question in systematic, piece by piece fashion without increasing technical debt or imposing significant extra burdens on un-related concurrent HDF5 development projects, in its most basic form, it doesn't bear fruit until we push thread safety up to the API level. However, the VOL layer exists, and thus we can make it thread safe early in the project – thus enabling multi-thread processing in VOL connectors.

One possible application of this facility is the proposed Bypass VOL connector – whose architecture is discussed below.

2.2 The Bypass VOL Connector

The Bypass VOL Connector must examine each API call as it is received.

If the Bypass VOL connector doesn't support multi-threaded execution of the API call in question, it grabs a write lock on the Bypass VOL Connector, and routes the API call to the native VOL. The write lock is dropped when the API call returns.

If multi-threaded operation is supported for this API call, it grabs a read lock on the Bypass VOL to prevent any non-multi-thread enabled API calls from executing during multi-thread operations.

If there have been any writes since the last multi-thread operation completed, it sends a flush command to the native VOL – note that this command will hit the Native VOL's global lock, and may take a while.

The exact processing from this point depends on the nature of the multi-thread support, but for purposes of this discussion let us presume that we wish to support multi-threaded reads of contiguous data sets of a scalar type, without type conversion, variable length data, or references. Given this presumption, processing proceeds as follows:

1. Query the Native VOL Connector to obtain the base address of the contiguous data set, its dimensions, and data type. This query will hit the Native VOL Connector's global lock, but since the necessary data should be cached, it should return quickly. It may also be necessary to obtain a pointer to the instance of H5FD_t used by the top level VFD.
2. Given the above data, construct VFD read call(s) to obtain the required data from file, and load it into the buffer supplied by the caller. Until H5S (selections) is made thread safe, these calls must be either the regular POSIX like calls, or vector I/O calls. Otherwise the VFD layer

would have to call the non-thread safe H5S package to walk the selection, and thus hit the Native VOL's global lock repeatedly.

Note that this operation duplicates a small subset of the functionality of the H5D (dataset) package – and thus it can be viewed as duplicate code if you squint just right.

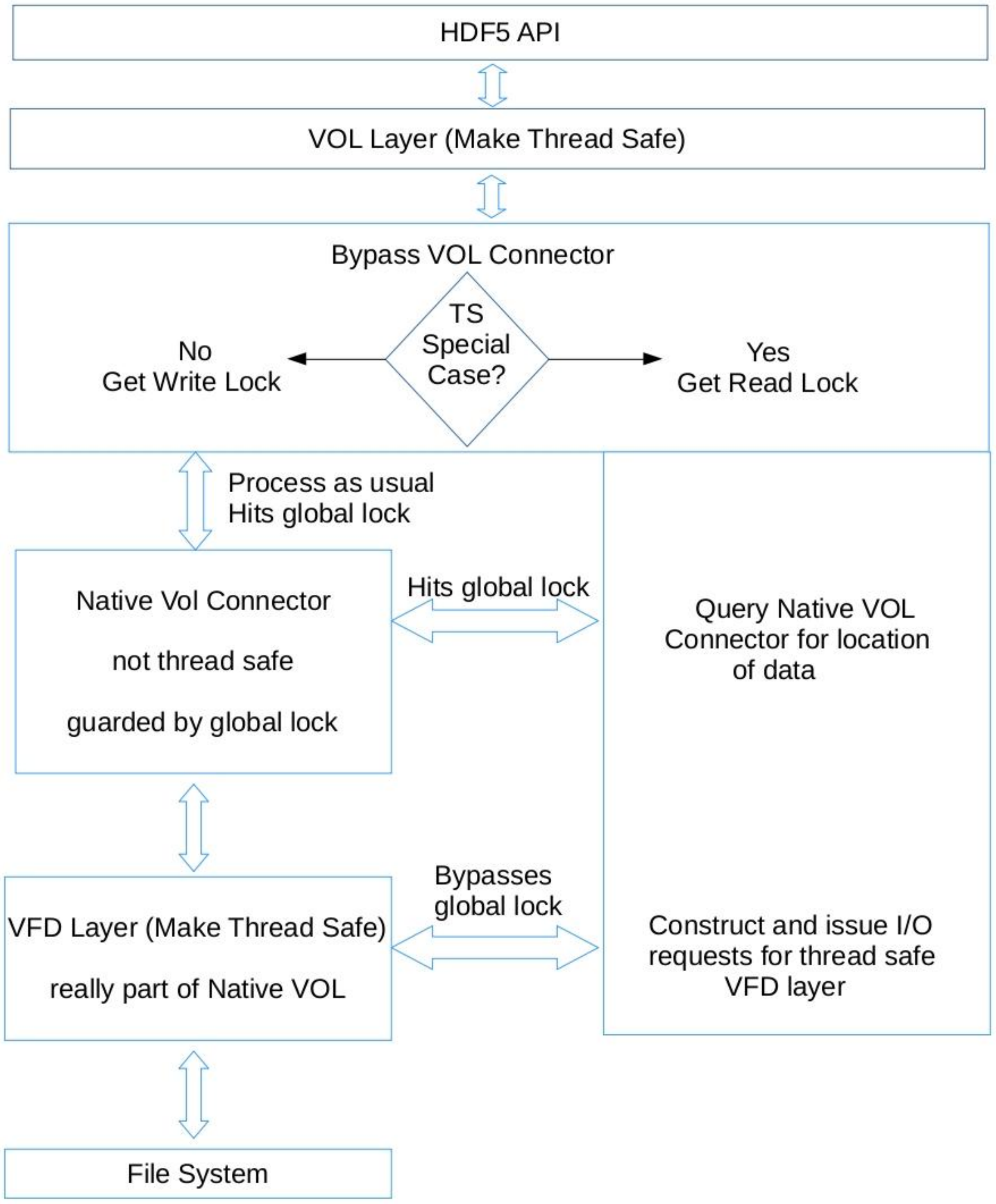
3. Make the required H5FD read call(s). Since the VFD layer and the relevant VFDs should be thread safe at this point, these calls can bypass the Native VOL's global lock – allowing an arbitrary number of reads to proceed concurrently.

   Alternatively, if H5FD has not been retrofitted for thread safety, the Bypass VOL can simply open the HDF5 file and read it directly with the usual POSIX calls.

4. On return, drop the read lock and exit.

Figure 1 below gives a simplified block diagram of the Bypass VOL.

Figure 1: Bypass VOL Block Diagram


HDF5 API
VOL Layer (Make Thread Safe)
Bypass VOL Connector
TS Special Case?
No Get Write Lock
Yes Get Read Lock
Process as usual Hits global lock
Native Vol Connector
not thread safe
guarded by global lock
Hits global lock
Query Native VOL Connector for location of data
VFD Layer (Make Thread Safe)
really part of Native VOL
Bypasses global lock
Construct and issue I/O requests for thread safe VFD layer
File System

With the exception of the flush call and the use of the R/W lock, the above functionality has already been implemented in a hybrid VOL used for the initial development and performance testing of the sub-filing VFD.

Observe that this approach segregates all code specific to the multi-thread enabled HDF5 API calls in the Bypass VOL Connector – away from any possible interaction with HDF5 library development proper.

2.3 Paths Forward

While there are numberof logical possibilities, for purposes of this RFC, we will restrict ourselves to considering only two:

- Proceed with the package by package retrofit of thread safety on the Native VOL until it is fully thread safe.
- Expand the Bypass VOL to a thread safe re-implementation of the Native VOL

Both options have their plusses and minuses. In a nut shell, it comes down to what we want to optimize for – minimum total effort or early delivery of multi-thread execution of a sub-set of the HDF5 API. Obviously, hybrid approaches are also possible.

2.3.1 Proceed with Package by Package Conversion to Thread Safety

The primary advantage of this approach is minimization of total effort. While there will doubtless be some re-working as we untangle hair balls such at the metadata cache / cache client reentrancy issue discussed above, in the main this approach should allow us to approach conversion of the HDF5 library to thread safety in a systematic fashion with little wasted effort. Given the perceived magnitude of the task, this is no small thing.

Perhaps equally important, the package by package approach allows us to put down the process of retrofitting thread safety on the HDF5 library at any package boundary at minimal cost.

On the other hand, we have used up most of our bag of tricks for delivering early results. Beyond targeting desired API calls and pushing thread safety towards them first, there is little we can do to offer additional multi-thread support until we push up to the entry points of the Native VOL.

2.3.2 Expand the Bypass VOL to a Thread Safe Re-Implementation of the Native VOL

Unless we are prepared to commit significant resources up front, the situation with the Bypass VOL is just the opposite. We can expand multi-thread support to additional API calls, or relax restrictions on API calls already supported. But in general, each time we do so, we will have to revisit existing code and duplicate more functionality from the Native VOL. Doing this piecemeal is unlikely to be cheap.

On the other hand, if we are prepared to commit the necessary resources, we should be able to do a thread safe re-write of the Native VOL more cheaply than the above package by package approach. However, this requires a large, up front commitment of resources, will not bear fruit for some time, and is probably best done in the context of a complete redesign and re-implementation effort. In any case, we are no longer talking about retrofitting thread safety on the HDF5 library.

3 Recommendation

Decide whether there is sufficient interest in the proposed approach to retrofitting thread safety on the HDF5 library. If so, start working out the details, and start thinking on which if any of the paths forward is acceptable.

Acknowledgements

The initial versions of this RFC were paid for with THG sustaining engineering funds.

Revision History

May 28, 2021:	Version 1 circulated for comment within The HDF Group.
May 31, 2021	Addressed comments from the HDF Group.
	Version 2 circulated for comment.
June 6, 2021	Version 3: Reworked for general distribution.
August 19, 2021	Version 4: Minor edits before publication.