RFC: SWMR Timeouts

Mike McGreevy

This RFC proposes a method that can be used to control the growth of an HDF5 file under single-writer/multiple-reader access.

Introduction

When performing single-writer/multiple-reader (SWMR) access to an HDF5 file, several conditions must be maintained so that the writer process never prematurely overwrites or removes data that a reader has loaded or is currently loading into its cache. Currently, the method employed simply never recycles space in the HDF5 file, thus a reader accessing an old location will still find a valid (though stale) HDF5 data structure and will be able to continue without error.

This poses the problem of the HDF5 file's size enlarging without bound. Without the ability to recycle the space used by stale data structures in the HDF5 file, there is no telling how large the file might get under extreme circumstances.

This RFC proposes adding a timeout value to the HDF5 file, accessible to any reader or writer process, which enforces a refresh policy (for any readers) and a recycle policy (for the writer) to allow the writer to safely recycle space in the file while in a single-writer/multiple-reader access environment.

Approach

To allow the writer to recycle space in the file under SWMR access without causing problems for any reader process, a real-world clock duration, , is shared between the writer and all readers. This clock duration imposes a time limit on how long the readers can maintain access to a loaded cache entry and how long until the writer can safely recycle space in the file.

On the reader side, the cache will be modified to timestamp any entry loaded into the cache. Every time a piece of metadata is accessed, its timestamp is compared to the timeout value, , to determine if the entry is still valid. If has elapsed since the entry was loaded into the cache, then the entry is considered to have timed out and is potentially stale. The entry will then need to be re-loaded from the file before being returned from the cache.

On the writer side, any data structure releasing space in the file whose space is sent into the free-space manager will have that space time-stamped and queued. The space will not actually be freed until 2 + has passed since it was added into the free-space queue. As long as readers are ensured not to have any data structure in their cache that is older than , the writer can safely free space in the file that is older than 2 + without worrying about causing problems for any readers. Note that in this case,

represents an extra bit of margin to avoid any potential corner case problems when data is read or recycled just before or after .

Use Cases

Any instance in which the single-writer/multiple-reader scenario is employed will benefit from this addition so as to keep the HDF5 file size from growing uncontrollably.

Implementation Details

Some design decisions are being proposed based off of certain access situations, so those situations and proposed solutions are outlined here.

Superblock Extension to store Timeout value

The timeout value, , will be stored in the file in a superblock extension. The writer process will store the value when it opens the file and any reader can read the value and synchronize with the writer process by using the same value as its timeout. The timeout value can be modified by an application by setting a file access property when accessing the file from the writer process. The default value will be zero, to indicate that SWMR access is not being performed and the library will behave normally (i.e., not SWMR-safely), for backward compatibility with previous releases of the library.

Retirement of SWMR_READ And SWMR_WRITE files access flags

Because the timeout value is stored in the file, it can act as an indicator as to the intent of the access, and whether or not SWMR-safe mechanics should be employed. If a writer accesses the file and sets up a timeout value, then any subsequent reader accessing the file will see the timeout in the file and know that it is in a single-writer/multiple-reader scenario, and behave accordingly. Additionally, if another writer attempts to access the file, it will see that a timeout value has already been set, and disallow access to the file. When a writer completes its process, it will remove the timeout value from the superblock, indicating that future readers need not worry about being SWMR safe and that another process can write to the file.

We can thus retire the SWMR-specific access flags (SWMR_READ and SWMR_WRITE), as HDF5 readers will automatically detect when a file access should be accessed SWMR-safely and a writer will need to specify a timeout value in order to enable SWMR access or fail if it already finds one in the file.

New Tool to remove SWMR Timeout Value from HDF5 file

A problem arises when a writing process is killed prematurely in that a timeout value will be left in the file's superblock extension, so subsequent writers trying to access the file will always fail citing that another writer has the file open. To fix this, a tool will be provided to remove the timeout value from any HDF5 file, disregarding its current state. This will either be a new tool designed specifically for this task (h5fix?), or lumped in as part of 'h5recover', which currently only recovers files after a crash when journaling was enabled, in an attempt to make h5recover responsible for fixing all potential file

problems that might arise when using hdf5. This has yet to be determined.

Object-wide timestamps

For file objects composed of multiple metadata cache entries, it's desirable to keep all the metadata related to the object in sync, thus timestamps will be applied on an object-by-object basis. All metadata related to a single object header (or hid_t value) will receive the same timestamp. When the object is determined to have timed out, all metadata related to the object will be evicted, to ensure the entire object is reflective of its state on disk, and not just the most recently accessed metadata. The eviction of an entire object can be achieved by taking advantage of the metadata tagging and single object flush and refresh code, as described in the following similarly named RFC:

http://www.hdfgroup.uiuc.edu/RFC/HDF5/flush_refresh_objects/RFC_flushevict_objects_v1.docx

API reattempts

In the case where an API call that loads metadata into the cache takes longer than and thus potentially reads in already-timed-out metadata (and may fail if the space has been recycled by the writer and contains garbage), a mechanism will be put in place to allow an API call to reattempt a read. In the SWMR read scenario, all API calls will be evaluated after their execution (resulting in either a success or failure) to determine if it took too long to be considered a valid read.

If the API call has taken too long, it will re-attempt the call number of times, where is a predefined maximum number of reattempts. If after attempts the API has not completed within the time limit, it will fail citing that the operation took too long.

If the API returns in a period of time within the bounds of the timeout value, , either after the first attempt or after a reattempt, then it will report success or failure and provide its return value as usual.

Object removal queue in writer

There is one case that a reader would not currently be able to handle without failure in any capacity, and that is when an object is simply deleted by the writer. Refreshing a piece of metadata when the metadata is gone (rather than simply moved elsewhere) will fail. To avoid this situation, the writer process will queue objects deleted from the file and disallow the recycling of that space until file close. Note that when a writer accesses a file without the intent to behave SWMR-safely, it will recycle all space (including that of deleted objects) normally.

Unresolved Issues

One issue remains unresolved, and that's the case when a reader accesses a file before a writer. In the case where a writer accesses the file first, it will set the timeout value so subsequent readers will behave correctly, but if there's already a reader accessing a file when a writer starts up, it will have already read a timeout value of zero, and will not be behaving in a SWMR-safe manner.

A potential solution to this problem is to have all reader processes periodically refresh a file's superblock and check to see if the SWMR timeout has been modified, but we wouldn't want this to be frequent enough to cause any potential performance hit, while it couldn't be so infrequent so as to be

rendered useless, so the effectiveness of this solution is questionable.

Aside from telling users not to do this, no really great solution has been identified.

Recommendation

The recommendation is to implement SMWR timeouts in HDF5 to prevent uncapped file size growth by writer processes within single-writer/multiple-reader access environments.

Revision History

November 16, 2010	Version 1 passed to Quincey for comment and copyediting.
November 22, 2010	Version 2 circulated for comment within The HDF Group.