Design: Metadata Cache Logging

Dana Robinson

HDF5

THG 2014-02-24

Document Version 4

As an aid for debugging, the existing ad-hoc metadata cache logging functionality will be made more robust. The improvements will include changes to the log format, code changes, and new API functions to make it easier for users to control cache logging and testing. This functionality is motivated by the new single-writer/multiple-reader (SWMR) feature which is likely to be difficult to debug due to the asynchronous nature of the feature and the addition of metadata flush dependencies. This document describes the new functionality as well as the JSON-based log format.

This document is intended for advanced users, particularly users of the SWMR feature, and HDF5 Library developers. The logging feature will appear in the future HDF5 1.10.0 release, but could also be added to the 1.8 branch, if desired.

March 16, 2014

The HDF Group

Copyright 2014 by The HDF Group.

All rights reserved.

For more information about The HDF Group, see www.hdfgroup.org.

Contents

1. Introduction

The metadata cache is a central feature of the HDF5 Library through which all *file metadata* read and write operations take place. The metadata stored in this cache is for internal use only and is not exposed to the user. It is used by the HDF5 Library to locate and characterize HDF5 objects and data. Each open file has its own metadata cache, and caches are not shared among processes. File metadata should not be confused with *user metadata* which is stored by the user as attributes attached to HDF5 objects primarily via H5A* API calls.

HDF5 1.10 will support the single-writer/multiple-reader (SWMR) data access pattern. SWMR will allow multiple reader processes to access an HDF5 file that is being written to by a single writer process with no inter-process communication (IPC) required. Support for this feature requires the writer process to order metadata writes to storage so that reader processes will not encounter an invalid HDF5 file. This could happen, for example, if the writer wrote a piece of metadata to storage that targets a piece of metadata that only existed in the writer's cache. In other words, the metadata had not been propagated to storage yet. When the reader attempted to load the targeted metadata, it would find garbage, causing an error.

Due to the lack of communication between the processes, the SWMR data access pattern is inherently asynchronous and bugs are expected to be difficult to troubleshoot due to the lack of deterministic reproducibility. Since SWMR bugs will most likely involve the metadata cache at some level, logging of cache operations would be very useful in debugging the feature especially when the error conditions are uncommon or only occur on particular hardware.

In addition to its use as a diagnostic aid for the SWMR feature, this feature would also be useful for troubleshooting general metadata cache problems or performance issues.

2. Use Cases

The primary use case for this feature is diagnosing metadata cache bugs reported by SWMR users. The most important of these is expected to be broken flush dependencies. A secondary use case is tracking cache activity over time; this may be useful for diagnosing metadata cache bugs and performance issues.

2.1. Detecting Broken Flush Dependencies

The logging functionality could be used to detect broken flush dependencies. A Python program could be used to inspect the output of each flush to ensure that no parents were flushed before their children.

2.2. Monitoring Cache Activity

The logging functionality could also be used to monitor cache usage parameters. This would be especially useful when making use of the cache/object flush control routines.

3. Enabling and Controlling the Feature

The feature will be turned off by default. It will be enabled by using the new `H5Pset_mdc_logging` function to modify the file access property list used to open or create a file. A Boolean flag parameter of this function will determine if logging begins at file open/create. Additionally, two other new functions – `H5Fstart/stop_mdc_logging` – will enable logging to be switched on and off as needed. Each call to the start function will begin by dumping the current cache contents and status. Functions have also been added that allow querying the logging properties from the file access property list and logging status via the HDF5 file identifier.

It is assumed that the logging framework overhead will be minimal when logging is switched off, and thus the feature does not warrant a compile-time build option.

4. Existing Cache Log Functionality

The current (HDF5 1.8.x/1.10.x) HDF5 Library has some existing cache logging functionality; however, it was added ad-hoc, is not documented, is a compile-time feature, and is controlled via an awkward interface.

The compile-time nature of the feature is a problem since the log should reflect the library build of interest, and a re-compile can change this. The compile-time nature is also inconvenient for users: those who use pre-compiled binaries may be unfamiliar with building the library, or those who need the library deployed to a location over which they have little control.

The lack of a documented, easily-consumed format is a problem since investigative tools will have a difficult time working with the generated log files.

The lack of testing is also clearly an issue if metadata cache logging is to be a robust, supported feature of the HDF5 Library.

The existing logging feature is enabled via the `H5Pset_mdc_config()` function. This function takes a large struct of cache configuration values as a parameter and acts on the file access property list. Flags for opening and closing the log file (`open_trace_file` and `close_trace_file`) as well as the log file name (`trace_file_name`) can be passed via this function. The problem with this scheme is that it exposes the user to a large number of unfamiliar cache parameters in the struct that must be set. It also requires an awkward file reopen to set the values. The new special-purpose functions avoid all of these issues and make enabling/disabling the feature much more natural.

Since this functionality was not really a part of the external-facing HDF5 API, it has been removed. At this point, the `H5AC_cache_config_t` struct has not been modified. Instead, the `open_trace_file`, `close_trace_file`, and `trace_file_name` members are simply ignored.

5. New HDF5 API Functions

5.1. H5Pset_mdc_log_options

```
herr_t H5Pset_mdc_log_options(hid_t fapl_id, hbool_t is_enabled, char *location,
hbool_t start_on_access)
```

`hid_t fapl_id`	IN: file access property list identifier
`hbool_t is_enabled`	IN: whether logging is enabled
`char *location`	IN: location of log in ASCII or UTF-8 (file path/name) (On Windows, this must be ASCII)
`hbool_t start_on_access`	IN: whether the logging will begin as soon as the file is opened or created

This function will set the logging parameters in a file access property list.

The location parameter will be a simple file path/name but may be expanded to include URLs in the future. There will be no default file name. The location parameter must specify a file name and not a directory. The default location for the log will be the current working directory.

NOTE: The log file is currently manipulated using the C standard library's buffered I/O calls (fprintf, for example) regardless of the virtual file driver (VFD) used. Log events are flushed immediately after the write call. On Windows, the location parameter must be an ASCII string since the Windows standard C library's I/O functions cannot handle UTF-8 file names.

The `start_log_on_access` flag will determine whether or not logging will begin on file open/create. This, combined with the begin/end functions, would allow users to selectively log troublesome areas of their code, potentially drastically decreasing running time and keeping log files smaller and more manageable.

There is currently no plan to add a file or source identifier to the log messages, so it normally will not be possible to send log messages from more than one cache to the same log location.

An option for the future would be to add a bitwise flag parameter that would be used to determine which types of messages are of interest (for example, flush dependencies). If this proved to be of use, it could be added while the SWMR feature is being developed (before the official HDF5 1.10 release).

Another option for the future would be to add a parameter that would control how often cache statistics were emitted.

5.2. H5Pget_mdc_log_options

```
herr_t H5Pget_mdc_log_options(hid_t fapl_id, hbool_t *is_enabled, char *location,
size_t *location_size, hbool_t *start_on_access)
```

`hid_t fapl_id`	IN: file access property list identifier
`hbool_t *is_enabled`	OUT: whether logging is enabled
`char *location`	OUT: location of log in ASCII or UTF-8 (just a file path/name for now)
`size_t *location_size`	OUT: size in bytes of the location string
`hbool_t *start_on_access`	OUT: whether the logging begins as soon as the file is opened or created

This function gets the current status of the logging (enabled/disabled), whether the logging begins at file open/create, and the location (file/path name) of the log file.

The `location` string must be allocated by the caller. A suitable size for the string can be determined by calling the function with a NULL location pointer, which will cause the function to emit the size via the `location_size` parameter.

5.3. H5Fstart_mdc_logging

```
herr_t H5Fstart_mdc_logging(hid_t file_id)
```

`hid_t file_id`	IN: HDF5 file identifier on which to start logging metadata operations

This function opens the log file and starts logging metadata cache operations for a particular file. Calling this function when logging has already been enabled will be considered an error.

5.4. H5Fstop_mdc_logging

```
herr_t H5Fstop_mdc_logging(hid_t file_id)
```

`hid_t file_id`	IN: HDF5 file identifier on which to stop logging metadata operations

This function only suspends the logging operations. The log file will remain open and will not be closed until the HDF5 file is closed.

5.5. H5Fget_mdc_logging_status

```
herr_t H5Fstop_mdc_logging(hid_t file_id, hbool_t *is_enabled, hbool_t
*is_currently_logging)
```

`hid_t file_id`	IN: HDF5 file identifier
`hbool_t *is_enabled`	OUT: whether logging is enabled
`hbool_t *is_currently_logging`	OUT: whether events are currently being logged

This function gets metadata cache status information. Logging status can be enabled (TRUE) or disabled (FALSE), and if enabled, the current logging status can be ongoing (TRUE) or paused (FALSE).

6. Log Messages

6.1. Log Format

The log is emitted using JSON notation (a schema can be found in the appendices of this document). The entire log is a valid JSON object consisting of the file name and an array of JSON-formatted log messages.

```
{
	"create_time":			<POSIX/Unix timestamp (int)>,
	"messages":
	[
		<log message 1 (as described below) (object)>,
		<log message 2 (object)>,
		…
		<log message n (object)>
	],
	"close_time":			<POSIX/Unix timestamp (int)>,

}
```

JSON was selected due to its ability to handle rich data and ubiquity, especially with dynamic analysis languages (for example, Python) and display libraries. Simple event-based log formats might be easier for humans to read, but would be less able to present rich data for more in-depth analysis.

Other log formats and/or libraries were considered, but none met our needs for a simple, yet expressive format combined with a well-supported, platform-independent, appropriately licensed library with a C API. Two libraries deserve mention, however:

- SLOG (http://www.mcs.anl.gov/research/projects/perfvis/software/log_format/) is a part of MPE and might be interesting for viewing process activity as a function of time. It is not clear if the library is suitable for this purpose (SWMR does not pass messages, for example), and the problem of time skew between separate machines might be troublesome.
- Pantheios (http://www.pantheios.org/) is a platform-independent logging library that might be considered in the future; however, it was decided to not add this dependency into the library code.

NOTE: This log format may evolve as the HDF5 1.10 release moves forward.

6.2. Log Messages

Each JSON message consists of a timestamp, a string describing the action being recorded, and any auxiliary data required such as offsets in the file or state transitions. The return values from internal cache API functions are included to help with debugging. Times in the log file are always recorded in POSIX time (in other words, number of seconds since epoch).

6.2.1. Eviction Pass

This message is emitted when the cache runs the eviction algorithm.

```
{
       "timestamp":         <POSIX/Unix timestamp (int)>,
       "action":            "evict",
       "returned":          <int>
}
```

6.2.2. Expunge Entry

This message is emitted when an entry is expunged (removed and not written, even if dirty) from the cache.

```
{
       "timestamp":         <POSIX/Unix timestamp (int)>,
       "action":            "expunge",
       "address":           <int>,
       "type_id":           <int>,
       "returned":          <int>
}
```

6.2.3. Flush Pass

This message is emitted when the cache runs the eviction algorithm.

```
{
       "timestamp":         <POSIX/Unix timestamp (int)>,
       "action":            "flush",
       "returned":          <int>
}
```

6.2.4. Insert Entry

This message is emitted when an entry is inserted into the cache.

```
{
       "timestamp":         <POSIX/Unix timestamp (int)>,
       "action":            "insert",
       "address":           <int>,
       "flags":             <int>,
       "type_id":           <int>,
       "size":              <int>,
       "returned":          <int>
}
```

6.2.5. Mark Dirty Entry

This message is emitted when a cache entry is marked dirty.

```
{
       "timestamp":         <POSIX/Unix timestamp (int)>,
       "action":            "dirty",
       "address":           <int>,
       "returned":          <int>
}
```

6.2.6. Move Entry

This message is emitted when a cache entry is moved in the file, changing its address.

```
{
       "timestamp":         <POSIX/Unix timestamp (int)>,
       "action":            "move",
       "old_address":       <int>,
       "new_address":       <int>,
       "returned":          <int>
}
```

6.2.7. Pin Entry

This message is emitted when a cache entry is pinned.

```
{
       "timestamp":         <POSIX/Unix timestamp (int)>,
       "action":            "pin",
       "address":           <int>,
       "returned":          <int>
}
```

6.2.8. Create Flush Dependency

This message is emitted when a flush dependency is being created between two pieces of metadata in the cache.

```
{
       "timestamp":         <POSIX/Unix timestamp (int)>,
       "action":            "create_fd",
       "parent_addr":       <int>,
       "child_addr":        <int>,
       "returned":          <int>
}
```

6.2.9. Protect Entry

This message is emitted when a cache entry is protected.

```
{
       "timestamp":         <POSIX/Unix timestamp (int)>,
       "action":            "protect",
       "address":           <int>,
       "readwrite":         <string "READ" | "WRITE" | "UNKNOWN">,
       "size":              <int>,
       "returned":          <int>
}
```

6.2.10. Resize Entry

This message is emitted when a cache entry is resized.

```
{
       "timestamp":         <POSIX/Unix timestamp (int)>,
       "action":            "resize",
       "address":           <int>,
       "new_size":          <int>,
       "returned":          <int>
}
```

6.2.11. Unpin Entry

This message is emitted when a cache entry is unpinned.

```
{
       "timestamp":        <POSIX/Unix timestamp (int)>,
       "action":           "unpin",
       "address":          <int>,
       "returned":         <int>
}
```

6.2.12. Destroy Flush Dependency

This message is emitted when a flush dependency between two pieces of metadata in the cache is being destroyed.

```
{
       "timestamp":        <POSIX/Unix timestamp (int)>,
       "action":           "destroy_fd",
       "parent_addr":      <int>,
       "child_addr":<int>,
       "returned":         <int>
}
```

6.2.13. Unprotect Entry

This message is emitted when an entry in the cache is unprotected.

```
{
       "timestamp":        <POSIX/Unix timestamp (int)>,
       "action":           "unprotect",
       "address":          <int>,
       "type_id":          <int>,
       "flags":            <int>,
       "returned":         <int>
}
```

7. Testing the Feature

A new test program (cache_logging(.c)) will be added to the test/ directory. This will be a fairly simple test program that will ensure that the setup and control functions work and ensure that the cache generates appropriate log messages.

8. Glossary, Terminology

Cache Entry

A cache entry is an item that is stored in the metadata cache. An HDF5 object will often be represented by multiple cache entries. For example, each node in a B-tree index is represented as a separate cache entry.

File Metadata

File metadata is metadata that describes the internal structure of the file. File metadata is created by the HDF5 Library and is largely invisible to users.

HDF5 Object

A "thing" stored in HDF5 storage. Objects include datasets, groups, and committed datatypes. Note that attributes are not considered HDF5 objects in their own right, but instead are considered a part of the object to which they are attached.

User Metadata

User metadata refers to attributes created by the user that are attached to datasets, groups, or committed datatypes.

9. Appendix: Reference Manual Entries

The *HDF5 Reference Manual* entries for the logging APIs are included in this chapter.

9.1. H5Pset_mdc_log_options

Name: H5Pset_mdc_log_options

Signature:

```
herr_t H5Pset_mdc_log_options(hid_t fapl_id, hbool_t is_enabled,
        char *location, hbool_t start_on_access)
```

Purpose:

Sets metadata cache logging options.

Description:

The metadata cache is a central part of the HDF5 library through which all *file metadata* reads and writes take place. File metadata is normally invisible to the user and is used by the library for purposes such as locating and indexing data. File metadata should not be confused with *user metadata*, which consists of attributes created by users and attached to HDF5 objects such as datasets via the H5A API calls.

Due to the complexity of the cache, a trace/logging feature has been created that can be used by HDF5 developers for debugging and performance analysis. The functions that control this functionality will normally be of use to a very limited number of developers outside of The HDF Group. They have been documented here to help users create logs that can be sent with bug reports.

Control of the log functionality is straightforward. Logging is enabled via the `H5Pset_mdc_log_options()` function which will modify the file access property list used to open or create a file. This function has a flag that determines whether logging begins at file open or starts in a paused state. Log messages can then by controlled via the `H5Fstart/stop_logging()` functions. `H5Pget_mdc_log_options()` can be used to examine a file access property list, and `H5Fget_mdc_logging_status()` will return the current state of the logging flags.

Notes:

Logging is disabled by default.

When enabled and currently logging, the overhead of the logging feature will almost certainly be significant.

The log file is currently manipulated using the C standard library's buffered I/O calls (for example, fprintf) regardless of the virtual file driver (VFD) used. Log events are flushed immediately after the write call.

On Windows, the location parameter must be an ASCII string since the Windows standard C library's I/O functions cannot handle UTF-8 file names.

The log file will be created when the HDF5 file is opened or created, regardless of the value of the `start_on_access` parameter. The log file will stay open as long as the HDF5 file is open.

Parameters:

`hid_t fapl_id`	IN: file access property list identifier
`hbool_t is_enabled`	IN: whether logging is enabled
`char *location`	IN: location of log in ASCII or UTF-8 (file path/name) (On Windows, this must be ASCII)
`hbool_t start_on_access`	IN: whether the logging will begin as soon as the file is opened or created

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

9.2. H5Pget_mdc_log_options

Name: H5Pget_mdc_log_options

Signature:

```
herr_t H5Pget_mdc_log_options(hid_t fapl_id, hbool_t *is_enabled,
        char *location, size_t *location_size, hbool_t *start_on_access)
```

Purpose:

Gets metadata cache logging options.

Description:

The metadata cache is a central part of the HDF5 library through which all *file metadata* reads and writes take place. File metadata is normally invisible to the user and is used by the library for purposes such as locating and indexing data. File metadata should not be confused with *user metadata*, which consists of attributes created by users and attached to HDF5 objects such as datasets via the H5A API calls.

Due to the complexity of the cache, a trace/logging feature has been created that can be used by HDF5 developers for debugging and performance analysis. The functions that control this functionality will normally be of use to a very limited number of developers outside of The HDF Group. They have been documented here to help users create logs that can be sent with bug reports.

Control of the log functionality is straightforward. Logging is enabled via the `H5Pset_mdc_log_options()` function which will modify the file access property list used to open or create a file. This function has a flag that determines whether logging begins at file open or starts in a paused state. Log messages can then by controlled via the `H5Fstart/stop_logging()` functions. `H5Pget_mdc_log_options()` can be used to examine a file access property list, and `H5Fget_mdc_logging_status()` will return the current state of the logging flags.

Notes:

The `location_size` string must be allocated by the caller. The appropriate size can be determined by calling the function with `location_size` set to NULL which will return the buffer size in bytes via the `location_size` pointer.

Parameters:

`hid_t fapl_id`	IN: file access property list identifier
`hbool_t *is_enabled`	OUT: whether logging is enabled
`char *location`	OUT: location of log in ASCII or UTF-8 (just a file path/name for now)
`size_t *location_size`	OUT: size in bytes of the location string
`hbool_t *start_on_access`	OUT: whether the logging begins as soon as the file is opened or created

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

9.3. H5Fstart_mdc_logging

Name: H5Fstart_mdc_logging

Signature:

```
herr_t H5Fstart_mdc_logging(hid_t file_id)
```

Purpose:

Starts logging metadata cache events if logging was previously enabled.

Description:

The metadata cache is a central part of the HDF5 library through which all *file metadata* reads and writes take place. File metadata is normally invisible to the user and is used by the library for purposes such as locating and indexing data. File metadata should not be confused with *user metadata*, which consists of attributes created by users and attached to HDF5 objects such as datasets via the H5A API calls.

Due to the complexity of the cache, a trace/logging feature has been created that can be used by HDF5 developers for debugging and performance analysis. The functions that control this functionality will normally be of use to a very limited number of developers outside of The HDF

Group. They have been documented here to help users create logs that can be sent with bug reports.

Control of the log functionality is straightforward. Logging is enabled via the `H5Pset_mdc_log_options()` function which will modify the file access property list used to open or create a file. This function has a flag that determines whether logging begins at file open or starts in a paused state. Log messages can then by controlled via the `H5Fstart/stop_logging()` functions. `H5Pget_mdc_log_options()` can be used to examine a file access property list, and `H5Fget_mdc_logging_status()` will return the current state of the logging flags.

Notes:

Logging can only be started or stopped if metadata cache logging was enabled via `H5Pset_mdc_log_options()`.

When enabled and currently logging, the overhead of the logging feature will almost certainly be significant.

The log file is opened when the HDF5 file is opened or created and not when this function is called for the first time.

This function opens the log file and starts logging metadata cache operations for a particular file. Calling this function when logging has already been enabled will be considered an error.

Parameters:

`hid_t file_id`	IN: HDF5 file identifier on which to start logging metadata operations

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

9.4. H5Fstop_mdc_logging

Name: H5Fstop_mdc_logging

Signature:

```
herr_t H5Fstop_mdc_logging(hid_t file_id)
```

Purpose:

Stops logging metadata cache events if logging was previously enabled and is currently ongoing.

Description:

The metadata cache is a central part of the HDF5 library through which all *file metadata* reads and writes take place. File metadata is normally invisible to the user and is used by the library for purposes such as locating and indexing data. File metadata should not be confused with *user* *metadata*, which consists of attributes created by users and attached to HDF5 objects such as datasets via the H5A API calls.

Due to the complexity of the cache, a trace/logging feature has been created that can be used by HDF5 developers for debugging and performance analysis. The functions that control this functionality will normally be of use to a very limited number of developers outside of The HDF Group. They have been documented here to help users create logs that can be sent with bug reports.

Control of the log functionality is straightforward. Logging is enabled via the `H5Pset_mdc_log_options()` function which will modify the file access property list used to open or create a file. This function has a flag that determines whether logging begins at file open or starts in a paused state. Log messages can then by controlled via the `H5Fstart/stop_logging()` functions. `H5Pget_mdc_log_options()` can be used to examine a file access property list, and `H5Fget_mdc_logging_status()` will return the current state of the logging flags.

Notes:

Logging can only be started or stopped if metadata cache logging was enabled via `H5Pset_mdc_log_options()`.

This function only suspends the logging operations. The log file will remain open and will not be closed until the HDF5 file is closed.

Parameters:

hid_t file_id	IN: HDF5 file identifier on which to stop logging metadata operations

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

9.5. H5Fget_mdc_logging_status

Name: H5Fget_mdc_logging_status

Signature:

```
herr_t H5Fget_mdc_logging_status(hid_t file_id, hbool_t *is_enabled,
       hbool_t *is_currently_logging)
```

Purpose:

Gets the current metadata cache logging status.

Description:

The metadata cache is a central part of the HDF5 library, through which all file metadata reads and writes take place. File metadata is normally invisible to the user and is used by the library for purposes such as locating and indexing data. File metadata should not be confused with user metadata, which consists of attributes created by users and attached to HDF5 objects such as datasets via the H5A API calls.

Due to the complexity of the cache, a trace/logging feature has been created that can be used by HDF5 developers for debugging and performance analysis. The functions that control this functionality will normally be of use to a very limited number of developers outside of The HDF Group. They have been documented here to help users create logs that can be sent with bug reports.

Control of the log functionality is straightforward. Logging is enabled via the `H5Pset_mdc_log_options()` function which will modify the file access property list used to open or create a file. This function has a flag that determines whether logging begins at file open or starts in a paused state. Log messages can then by controlled via the `H5Fstart/stop_logging()` functions. `H5Pget_mdc_log_options()` can be used to examine a file access property list, and `H5Fget_mdc_logging_status()` will return the current state of the logging flags.

Notes:

Unlike `H5Fstart/stop_mdc_logging()`, this function can be called on any open file identifier.

Parameters:

hid_t `file_id`	IN: identifier of an open HDF5 file
hbool_t `*is_enabled`	OUT: whether logging is enabled
hbool_t `*is_currently_logging`	OUT: whether events are currently being logged

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

10. Revision History

February 24, 2014:	Version 1 circulated for comment within The HDF Group SWMR team.
March 15, 2014:	Version 2 includes many updates, circulated for comment with THG SWMR team.
March 16, 2014:	Version 3 updates the document in light of recent changes. Circulated within the SWMR team.
March 17, 2014:	Version 4: editing and formatting changes.