RFC: Core VFD Backing Store Paged Writes

Dana Robinson

The core virtual file driver (VFD) allows manipulating HDF5 files in memory instead of in physical storage. Files can either be created in memory or existing files in physical storage can be copied into memory. Optionally, the in-memory changes can be propagated back to physical storage when the file is closed. When this is done, the entire file is written out to disk, even if only a few bytes were changed.

This document describes updates to the core VFD that track modifications and only write the changed bytes on file close. As an optimization that reduces small I/O operations, the modifications can optionally be aggregated into pages.

This feature will be introduced in HDF5 1.8.13, to be released in May 2014.

Introduction

The core virtual file driver (VFD) allows HDF5 files to be created or opened in memory instead of in physical storage (files are copied into memory on open). All subsequent file manipulations occur in memory, allowing very fast HDF5 file operations but with the disadvantage of potentially requiring significant memory resources when working with large files. On close, the changes can optionally be propagated to physical storage. Thconfigur via the following API call:

```
      herr_t H5Pset_fapl_core( hid_t fapl_id, size_t increment, hbool_t backing_sto
re )
```

The `backing_store` parameter sets whether changes are propagated to physical storage on close. If this is set to `0` (`FALSE`) then all changes will be lost when the file is closed. If this is set to `1` (`TRUE`), then the changes are written to storage on file close or flush.

The current implementation of the library writes the entire file out on close if even a single byte has changed. Naturally, this can be inefficient, especially when very large files are written out after minimal changes have been made.

Tracking Writes for Improved Performance

The changes to the core VFD fairly straightforward. As write calls pass through the core VFD, a list of `(start address, end address)` pairs representing the writes updated, serving as a map of modified regions in the file. This data structure merge overlapping or abutting regions as they inserted into the list. When the file is closed, the list traversed and the modified regions of memory

propagated to physical storage

Note that these marked regions at the granularity of the write calls that the library makes. i.e., an entire metadata object or dataset chunk be marked dirty if even a single byte is changed, since the library a single write call when these are evicted from their respective caches. The core VFD make no effort to determine the particular bytes that were modified with respect to the original data.

Figure : Effect of the paging feature. When the paging feature has been enabled, the in-memory "file" is conceptually divided into multiple pages (dashed lines). Dirtying any part of a page marks the entire page as dirty.

Using the New Feature

The write tracking feature will be off by default, even when the `backing_store` flag is set to TRUE. The feature will be controlled via the new `H5Pget/set_core_write_tracking()` HDF5 API calls (tentative RM calls appear in the appendices of this document).

```
herr_t   H5Pset_core_write_tracking(hid_t   fapl_id,   hbool_t
is_enabled, size_t page_size)
```

```
herr_t   H5Pget_core_write_tracking(hid_t   fapl_id,   hbool_t
*is_enabled, size_t *page_size)
```

Setting the page size to any nonzero value turns write tracking on at that page size. Setting a page size of 1 byte disables paging.

Performance

The performance benefits of the feature will depend heavily on the data access patterns of the

application and will have to be evaluated on a case-by-case basis. In cases where the majority of the data would be written out (e.g., creating and writing data to a new file), the new feature will likely not impart a significant performance benefit. In cases where a small amount of data will be added or changed (e.g., opening an existing file and modifying a small amount of existing data), the performance benefits could be significant.

When performance tuning, the following parameters are likely to have significant effects on I/O throughput:

- backing store `H5Pset_core_`, described in this document).
- Dataset layout and chunk size (`H5Pset_layout` and `H5Pset_chunk`).
- Metadata aggregation size (`H5Pset_meta_block_size`).
- Using the latest file format (`H5Pset_libver_bounds`).
- Data layout considerations (arrangement of groups, datasets, types, etc.).

In general, anything that promotes aggregation of changes made to the file will enhance the performance of this feature. Unfortunately, empirical testing will typically be required to determine the "sweet spot" between reducing the number of seeks and minimizing the amount of data written out.

Testing

The feature will be tested via

Acknowledgements

This work is being supported by a customer of The HDF Group.

Revision History

September 18, 2013:	Version 1 circulated for comment within The HDF Group.
November 7, 2013	Version 2 includes updates concerning recent work on the feature. Circulated for comment within The HDF Group.
March 28, 2014	Version 3 updated to reflect how the feature will be integrated into HDF5 1.8.13 (and the 1.10 trunk). Circulated within The HDF Group.
March 31, 2014	Version 4 updated to reflect Elena's changes. Circulated on the forum.

Glossary, Terminology

virtual file layer (VFD)	The virtual file layer is an abstraction layer in the HDF5 library that maps I/O operations such as "read" to concrete I/O calls like the POSIX read() call or the Win32 ReadFile() call.
virtual file driver (VFD)	Implements a particular mapping of abstract to concrete I/O calls.

References

1. The HDF Group. “Reference Manual: H5Pset_fapl_core,” http://www.hdfgroup.org/HDF5/doc/RM/RM_H5P.html#Property-SetFaplCore (retrieved September 18, 2013 - refers to HDF5 v1.8.11).
2. The HDF Group. “HDF5 Virtual File Layer,” http://www.hdfgroup.org/HDF5/doc/TechNotes/VFL.html (November 18, 1999 - This document is slightly out of date).

1)

Appendix: Virtual File Drivers

The HDF5 library uses a layered architecture, the lowest of which is the Virtual File Layer (VFL). The VFL handles low-level file I/O via Virtual File Drivers (VFDs). Each VFD implements a different I/O scheme: e.g., MPI-I/O, POSIX I/O, in-memory I/O, etc. This VFL/VFD scheme allows abstract HDF5 file manipulations to be separated from storage I/O operations. A fairly in-depth, though slightly out-of-date, description of how a VFD is implemented can be found in the references.

2) Appendix: H5Pset_core_write_tracking RM entry

Name: H5Pset_core_write_tracking

Signature:

```
herr_t H5Pset_core_write_tracking(hid_t fapl_id, hbool_t is_enabled, size_t page_size)
```

Purpose:

Sets information about the write tracking feature used by the core VFD.

Description:

When a file is created or opened for writing using the core VFD with the backing store option turned on, the VFD can be configured to track changes to the file and only write out the modified bytes. To avoid a large number of small writes, the changes can be aggregated into pages of a user-specified size.

Setting the `page_size` parameter to zero will turn off tracking and cause the entire file to be written out to storage when closed.

Setting the `page_size` parameter to 1 will enable tracking but with no paging.

Note:

Write tracking is turned off by default.

This function is only for use with the core VFD and must be used after the call to `H5Pset_fapl_core`. It is an error to use this function with any other VFD.

This function only applies to the backing store write operation, which typically occurs when the file is flushed or closed. It has no relationship to the increment parameter passed to `H5Pset_fapl_core`.

For optimum performance, the `page_size` parameter should be a power of two.

Parameters:

hid_t fapl_id	IN: File access property list identifier
hbool_t is_enabled	IN: Whether the feature is enabled
size_t page_size	IN: Size, in bytes, of write aggregation pages

Returns:

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

3) Appendix: H5Pget_core_write_tracking Reference Manual Entry

Name: H5Pget_core_write_tracking

Signature:

```
herr_t H5Pget_core_write_tracking(hid_t fapl_id, hbool_t *is_enabled, size_t *page_size)
```

Purpose:

Gets information about the write tracking feature used by the core VFD.

Description:

When a file is created or opened for writing using the core VFD with the backing store option turned on, the VFD can be configured to track changes to the file and only write out the modified bytes. To avoid a large number of small writes, the changes can be aggregated into pages of a user-specified size.

Note:

This function is only for use with the core VFD and must be used after the call to `H5Pset_fapl_core`. It is an error to use this function with any other VFD.

This function only applies to the backing store write operation, which typically occurs when the file is flushed or closed. It has no relationship to the increment parameter passed to `H5Pset_fapl_core`.

For optimum performance, the `page_size` parameter should be a power of two.

Parameters:

`hid_t fapl_id`	IN: File access property list identifier
`hbool_t *is_enabled`	OUT: Whether the feature is enabled
`size_t *page_size`	OUT: Size, in bytes, of write aggregation pages

Returns:

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

4)