HDF5 Single-Writer/Multiple-Reader User's Guide

HDF5

Version 3

This document describes the prototype of HDF5's single-writer/multiple-reader feature and describes how to use the feature. The feature allows concurrent access of HDF5 files by writer and reader processes.

The HDF Group

http://www.HDFGroup.org

Contents

1. Introduction

The current implementation of the HDF5 Library, 1.8.x, does not support a concurrent read from an HDF5 file while it is being written to without explicit coordination between the processes. This limitation makes inspecting data while it is being collected impossible at the file level.

The primary reason for this lack of concurrent read/write access is the complexity of the HDF5 file format combined with the presence of a caching layer in the library. The internal structure of an HDF5 file includes internal file addresses. It is possible that a file metadata object can be flushed to the disk before other objects it references are flushed. This would create a transiently invalid file. A reader that opens this partially written file could attempt to resolve the invalid file address and read garbage instead of the expected file object. Other considerations, such as free space recycling in the file, are also problematic under concurrent access.

In order to address this issue, HDF5 developers have been exploring a mechanism to allow for concurrent access by a single writer process with any number of reader processes. This data access pattern is known as single-writer/multiple-reader or SWMR for short.

Some of the HDF5 Library's internal structures were enhanced to be SWMR-safe and tests were added to validate the SWMR design assumptions.

While the general SWMR feature is far from completion, the prototype described in this document supports a particular SWMR access pattern that might be of interest to many application developers: a writer process can append data to the datasets in an HDF5 file while several reader processes can concurrently read the new data from the file. No communications between the processes and no file locking are required. The processes can run on the same or on different platforms.

This document was written for application developers who might be interested in trying SWMR and providing feedback to the HDF5 development team.

1.1. POSIX-compliant Requirement

There is one requirement that must be satisfied before the SWMR feature can be used. In order to access an HDF5 file with a SWMR-enabled application, the HDF5 file must be located on a POSIX-compliant file system.

2. The SWMR Approach

To solve the difficulties involved with allowing concurrent access to write and read processes, HDF5 developers used an approach that is lock-free and that does not require any communication between processes.

This approach is based on the assumption that the HDF5 file resides on a file system compliant with POSIX I/O semantics. There are two major features of POSIX I/O semantics that are critical for SWMR use:

- POSIX I/O writes must be performed in a sequentially consistent manner.
- Writes to the file must appear as atomic operations to any readers that access the file during the write operation.

These semantics apply to any processes that access the file from any location.

The HDF5 SWMR prototype distribution includes simple test programs that attempt to expose deficiencies in a file system that prevent proper SWMR operation. Those tests are described in the "Testing SWMR" section on page 8. Testing of this prototype showed that assumption 2 above was not valid on several file systems including Lustre and Linux ext3/4. The prototype passed testing on Linux with GPFS, FreeBSD with UFS2, and OS X with HFS+.

This approach does not have a significant impact on I/O throughput, though file size may increase slightly. An increase in file size is a consequence of the free space manager, which recycles file objects, being turned off, as well as some copy-on-write operations in the metadata cache.

For more information on SWMR-safe design, see the "HDF5 SWMR Feature Design and Semantics" document.

3. SWMR Scope and Limitations

The current SWMR implementation scope is as follows:

1. The writer process is only allowed to modify raw data of existing datasets by:
   a. Appending data along any unlimited dimension.
   b. Modify existing data.

The following operations are not allowed (the corresponding HDF5 calls will fail):

2. The writer is not allowed to add any new objects to the file such as groups, datasets, links, committed datatypes, and attributes.
3. The writer is not allowed to delete HDF5 objects such as groups, datasets, links, committed datatypes, and attributes.
4. The writer is not allowed to modify or append to any data items containing variable-size datatypes (including string and region references datatypes).
5. File space recycling is not allowed. As a result, the sizes of a file modified by a SWMR writer may be larger than if it was modified by a non-SWMR writer.

4. The SWMR-enabled HDF5 Library

The purpose of this chapter is to describe what must be installed and configured so that the SWMR feature can be used.

4.1. Getting the Source

As of the time of this writing the SWMR functionality has not been released and is in a prototype stage. The HDF5 Library source with the SWMR functionality can be obtained directly from the HDF5 SVN repository at http://svn.hdfgroup.uiuc.edu/hdf5/branches/revise_chunks/. Miscellaneous SWMR documentation and the latest source tar ball can be found at the FTP server ftp://ftp.hdfgroup.uiuc.edu/pub/outgoing/SWMR/.

WORD of CAUTION: Please note that HDF5 files created by the SWMR HDF5 library may not be read by the tools based on the HDF5 1.8 libraries due to the new features introduced in the HDF5 development branch (see the link above). An application built with the HDF5 Library with SWMR capability can always read and modify files created by all previous versions of HDF5.

4.2. Building the Library

No special configuration options are required to build the HDF5 library with the SWMR functionality. The usual `configure`, `make`, `make check`, and `make install` process should be followed for building the library. Please see the `release_docs/INSTALL` file in the source distribution for more information about building and installing the HDF5 Library.

The HDF5 SWMR prototype is currently only supported on Unix-like systems. The prototype has not yet been tested on Windows systems.

4.3. Building Applications

Writer and reader applications have to set a single flag when opening a file for SWMR access to assure that the HDF5 Library uses internal structures in a SWMR-safe manner. Applications that will write to an HDF5 file set the `H5F_ACC_SWMR_WRITE` flag. Applications that will read from an HDF5 file set the `H5F_ACC_SWMR_READ` flag. See the "Setting up SWMR Access to an HDF5 File" section on page 9 for more information.

4.4. Testing SWMR

In the prototype, SWMR functionality is tested as a part of the `make check` build step as described below:

- POSIX atomicity tests `test/atomic_writer.c` and `test/atomic_reader.c` check to confirm if a file system supports POSIX atomicity. Information on how to run the POSIX tests can be found in the `test/AtomicWriterReader.txt` file.
- The metadata cache test exercises the flush ordering functionality required for SWMR. See the `test/swmr*.c` files.
- Basic SWMR operation between processes is tested via a shell script named `testswmr.sh`. This script invokes a writer and some readers which then operate on a central file using the SWMR access pattern.
- The programs `test/use_append_chunk` and `test/use_append_mchunk` append data to datasets with unlimited dimensions and exercise the main functionality of the current SWMR prototype. The programs would be useful to anyone who would like to start using the SWMR feature in their HDF5 applications and who needs to know how to create writer and reader applications. The programs are described in detail in the "SWMR Examples" chapter beginning on page 12.
- A test script `test_usecases.sh` is installed in the same directory as the programs and is used to run the programs during `make check`.

Output from the tests is displayed on `stdout` and `stderr` like all other HDF5 test results. No special log files are created.

5. The Programming Model

Our main consideration in this stage of the prototype is appending data to chunked datasets with unlimited dimensions. We anticipate this to be the most common scenario where SWMR semantics are needed. Any number of dimensions (up to the limit of the library) is supported, and any of those dimensions can be unlimited in size. No new file objects can be created under SWMR operation. Variable-length and reference datatypes are not supported.

The current SWMR prototype does not support data updates of the fixed-size chunked datasets due to a chunk indexing method that has not been fully converted and tested with SWMR-safe semantics. The work-around is to declare one dimension to be unlimited using `H5S_UNLIMITED`.

Note that applications that want to take advantage of the SWMR feature must follow the SWMR programming model to assure correct results. The model is described below.

5.1. Setting up SWMR Access to an HDF5 File

Go through the sequence of steps below when to use the SWMR data access pattern:

1. Set up the file
   - Create/open the HDF5 file as in non-SWMR access.
   - Create any file objects such as datasets and groups that are required for data storage.
   - Close the file.

2. Start a writer process
   - Open the file by calling `H5Fopen` using the `H5F_ACC_SWMR_WRITE` flag and begin writing to the dataset.

     Here is a sample: `fid = H5Fopen(FILE, H5F_ACC_RDWR | H5F_ACC_SWMR_WRITE, H5P_DEFAULT);`

   - Use `H5Dflush` periodically to flush the data for a particular dataset to the file. New data will appear as file objects are evicted/flushed from the cache (see page 10).

3. Start at least one reader process
   - Open the file by calling `H5Fopen` using the `H5F_ACC_SWMR_READ` flag and begin reading data from the dataset.

     Here is a sample: `fid = H5Fopen(FILE, H5F_ACC_RDONLY | H5F_ACC_SWMR_READ, H5P_DEFAULT);`

   - Poll the dataset. SWMR presumes no communication between processes. Newly appended data is checked for via polling the dataset. Polling is done by calling `H5Drefresh` on the dataset, checking the number of stored elements in each dimension of interest (via `H5S* extent` calls), and comparing this with the previous number of elements (see page 10). When the size has been seen to change, the new data is read from the file using the usual `H5Dread` calls.

4. Stop the writer process
    a. The writer process closes the file.

5. Stop Reader(s)
    a. Each reader process closes the file.

Note that after step 1, the file can be opened by the writer and reader(s) in any order. In other words, steps 2 and 3 can occur in any order. At this time, there is no enforcement of any SWMR policies at the library level, though this is a planned addition for a future release. It is recommended for this prototype that readers always start after a writer process has been started.

5.2. H5Dflush and H5Drefresh

Two new APIs were introduced in the SWMR branch to allow applications to flush and refresh cached metadata in order for readers to see new data in the file. This section provides a short description of each function.

5.2.1. H5Dflush

Signature: `herr_t H5Dflush(hid_t dset_id)`

Purpose: Flushes all data and metadata associated with a dataset.

Description: The function flushes the specified dataset's metadata from the metadata cache to the file and all raw data buffers associated with the dataset to the file. If the dataset is chunked, raw data chunks are written to the file when this call is issued.

Parameters:

`hid_t` `dset_id`	IN: Dataset identifier

Returns: Non-negative on success; negative on failure.

5.2.2. H5Drefresh

Signature: `herr_t H5Drefresh(hid_t dset_id)`

Purpose: Refreshes metadata items associated with a dataset in a metadata cache.

Description: The function refreshes all metadata items associated with the dataset in the metadata cache.

Parameters:

`hid_t` `dset_id`	IN: Dataset identifier

Returns: Non-negative on success; negative on failure.

6. SWMR Examples

The HDF5 SWMR distribution contains two example programs for the SWMR access feature: `test/use_append_chunk` and `test/use_append_mchunk`. Each example consists of a writer and a reader(s) that use the SWMR programming model. The writer adds data along an unlimited dimension, and the reader(s) reads newly appended data. These programs are described below.

The HDF5 SWMR prototype distribution also has a new tool called `h5watch` and located in the `hl/tools/h5watch` directory. The tool is an example of a polling application and is described in "The h5watch Tool" section on page 17.

6.1. Appending a Single Chunk to a Dataset

6.1.1. Description of use_append_chunk

The program `use_append_chunk` appends a single chunk of raw data to a dataset along an unlimited dimension within a pre-created file and reads the new data back.

The program first creates one 3-dimensional dataset using chunked storage. Each chunk is a `(1,chunksize,chunksize)` square. The dataset dimensions are `(unlimited,chunksize,chunksize)`. The datatype is a 2 byte integer. No data is written to a dataset; in other words, the size of the first current dimension is 0.

The writer first appends planes, each of `(1,chunksize,chunksize)`, to the dataset. Next, it fills each plane with a plane number `n`, and then it writes new plane at the n^{th} plane. The writer increases the plane number and repeats until the first dimension becomes `chunksize` long. The end dataset is a $\texttt{chunksize}^3$ cube as shown in the figure below.

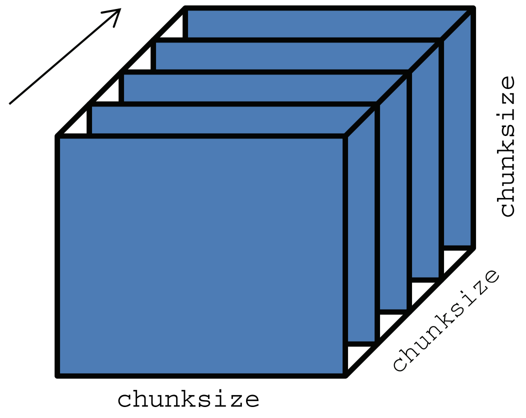

Figure 1. Writer fills cube with planes along the slowest changing dimension

The reader is a separate process running concurrently with the writer. It reads planes from the dataset as the dataset is growing. When it detects that the size of the unlimited dimension increases, the reader will read in the new planes, one by one, and verify the correctness of the data. The `n`th plane should contain all "`n`" (all values = n). When the unlimited dimension grows to the `chunksize`, which is the expected end of data, the reader exits.

6.1.2. Running use_append_chunk

The simplest way to run the program is without specifying any arguments:

```
$ use_append_chunk
```

The program first creates a skeleton dataset (0,256,256) of 2 byte integers. Next, it forks off a process, which becomes the reader process to read planes from the dataset, while the original process continues as the writer process to append planes onto the dataset.

Other possible options:

1. ***-z*** specifies a different chunk size. The default chunk size value is 256.

   ```
   $ use_append_chunk -z 1024
   ```

   The command above uses (1,1024,1024) chunks to produce a 1024^3 cube, about 2 GB in size.

2. ***-f*** *filename* specifies different file name

   ```
   $ use_append_chunk -f /gpfs/tmp/append_data.h5
   ```

The data file is /gpfs/tmp/append_data.h5. This allows two independent processes in separate compute nodes to access the HDF5 file on the shared /gpfs file system.

3. ***-l** option* launches only the reader or writer process.

   ```
   $ use_append_chunk -f /gpfs/tmp/append_data.h5 -l w  # in node X
   $ use_append_chunk -f /gpfs/tmp/append_data.h5 -l r  # in node Y
   ```

   Valid values for *option* are w for writer and r for reader.

   In node X, launch the writer process to create the data file and append to it. In node Y, launch the read process to read the data file. Note that one needs to time the read process to start AFTER the write process has created the skeleton data file. Otherwise, the reader will encounter errors such as data file not found.

4. ***-n** option* specifies the number of planes to write or read. The default value is the same as the chunk size as specified by option **-z**.

   ```
   $ use_append_chunk -n 1000
   ```

   The command above writes 1000 planes in one HDF5 I/O call.

5. ***-s** option* enables the SWMR file access mode. The default value is yes.

   ```
   $ use_append_chunk -s 0
   ```

   The command above opens the HDF5 data file without the SWMR access mode (0 means off). This likely will result in error. This option is provided for users to see the effect of the needed SWMR access mode for concurrent access.

6.2. Appending a Hyperslab of Multiple Chunks

6.2.1. Description of use_append_mchunk

The program `use_append_mchunk` appends a hyperslab that spans several chunks of a dataset with unlimited dimensions within a pre-created file and reads the new data back.

The program first creates one 3-dimensional dataset using chunked storage. Each chunk is a `(1,chunksize,chunksize)` square. The dataset dimensions are `(unlimited,2*chunksize,2*chunksize)`. The datatype is a 2 byte integer. Therefore, each plane consists of 4 chunks. No data is written to a dataset; in other words, the size of the first current dimension is 0.

The writer first appends planes, each of `(1,2*chunksize,2*chunksize)` to the dataset. Next, it fills each plane with a plane number `n` and then writes the new plane at the `n`th plane. The writer increases the plane number and repeats until the first dimension becomes `2*chunksize` long. The end dataset is a $(2*chunksize)^3$ cube as shown in the figure below.

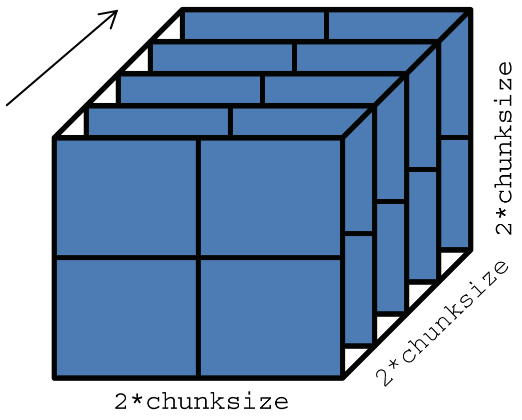

Figure 2. Writer fills cube with planes along the slowest changing dimension
Each plane consists of four chunks.

The reader is a separate process running concurrently with the writer. It reads planes from the dataset as the dataset is growing. When it detects that the size of the unlimited dimension increases, the reader will read in the new planes, one by one, and verify the correctness of the data. The `n`th plane should contain all "`n`" (all values = n). When the unlimited dimension grows to `2*chunksize`, which is the expected end of data, the reader exits.

6.2.2. Running use_append_mchunk

The simplest way to run the program is without specifying any arguments:

```
$ use_append_mchunk
```

The program creates a skeleton dataset (0,512,512) of 2 byte integers. Next, it forks off a process, which becomes the reader process to read planes from the dataset, while the original process continues as the writer process to append planes onto the dataset.

Other possible options:

1. ***-z*** specifies different chunk size. The default chunk size value is 256.

```
$ use_append_mchunk -z 512
```

The command above uses (1,512,512) chunks to produce a 1024^3 cube, about 2 GB size.

2. ***-f** filename* specifies different file name

   ```
   $ use_append_mchunk -f /gpfs/tmp/append_data.h5
   ```

   The data file is /gpfs/tmp/append_data.h5. This allows two independent processes in separate compute nodes to access the HDF5 file on the shared /gpfs file system.

3. ***-l** option* launches only the reader or writer process.

   ```
   $ use_append_mchunk -f /gpfs/tmp/append_data.h5 -l w  # in node X
   $ use_append_mchunk -f /gpfs/tmp/append_data.h5 -l r  # in node Y
   ```

   Valid values for *option* are w for writer and r for reader.

   In node X, launch the writer process to create the data file and append to it. In node Y, launch the read process to read the data file. Note that one needs to time the read process to start AFTER the write process has created the skeleton data file. Otherwise, the reader will encounter errors such as data file not found.

4. ***-n** option* specifies the number of planes to write or read. The default value is same as the chunk size as specified by option -z.

   ```
   $ use_append_mchunk -n 1000
   ```

   The command above writes 1000 planes in one HDF5 I/O call.

5. -s *option*: use SWMR file access mode or not. The default value is yes.

   ```
    $ use_append_mchunk -s 0
   ```

   The command above opens the HDF5 data file without the SWMR access mode (0 means off). This likely will result in error. This option is provided for users to see the effect of the needed SWMR access mode for concurrent access.

6.3. Appending n-1 Dimensional Planes

Both `use_append_chunk` and `use_append_mchunk` can be used to append n-1 dimensional planes.

6.3.1. Description

The programs `use_append_chunk` and `use_append_mchunk` append n-1 dimensional planes or regions to a chunked dataset where the data does not fill the chunk. This means the chunks have multiple planes, and when a plane is written, only part of each chunk is written. This use case is achieved by extending the previous examples by defining the chunks to have the slowest changing dimension size greater than 1. The `-y` option is implemented for both `use_append_chunk` and `use_append_mchunk`.

6.3.2. Running the Program

The simplest way to run the program `use_append_mchunk` is:

```
$ use_append_mchunk -y 5
```

The program creates a skeleton dataset (0,512,512) of 2 byte integers with chunk sizes (5,512,512). Next, it forks off a process, which becomes the reader process to read one plane at a time, while the original process continues as the writer process to append one plane at a time.

Other possible options will work as in the previous examples.

6.4. The h5watch Tool

6.4.1. Description

The h5watch tool allows a user to monitor the growth of a dataset written by an HDF5 application. It prints out the new elements whenever the application extends the size and adds data to that dataset.

h5watch polls the specified dataset periodically for changes in its dimension sizes. If at least one of the dataset's dimension sizes has increased, h5watch prints out the new appended data. For compound datasets, the tool has an option to print new data for the specified fields in the compound datatype. h5watch can be used only on chunked datasets with unlimited dimensions. The application that writes the file needs to ensure that changes to the dataset are flushed to the file with `H5Dflush`.

6.4.2. Running h5watch

To run the tool, use the following command:

```
$ h5watch Path_to_dataset
```

`Path_to_dataset` consists of two parts: the path to the HDF5 file and the path to the dataset within the file. For example, to monitor dataset A in the file /home/user/file.h5, the argument `path_to_dataset` is `/home/user/file.h5/A`.

The following are other useful options for h5watch:

1. **`--help`** prints help message

```
$ h5watch --help
```

2. **`–polling=N`** sets the polling interval to N (in seconds) when the dataset will be checked for changes in dimension sizes. The default interval for polling is 1.

```
$ h5watch –polling=5 ./myfile.h5/mygroup/mydataset
```

With the command-line above, the tool will read data every 5 seconds from the dataset /mygroup/mydataset in the file ./myfile.h5.

For more options, check the tool's help message.

7. Revision History

June 29, 2013:	Version 1. Circulated for comment within The HDF Group. It contains combined materials from UG distributed with the source code and MS version of UG.
June 30, 2013	Version 2. Expanded the first three sections, added the h5watch section, and sent the document to The HDF Group and the customer.
September 10, 2013	Version 3. Formatting and editing.