Single-Writer/Multiple-Reader (SWMR) User Guide (v2)

Stage I Prototype
22 May 2013

NOTE: This document only pertains to the stage I prototype. SWMR support is an experimental feature and a work in progress. Anything outlined in this document is subject to change as work progresses.

Introduction

This document describes the SWMR semantics, programming model, and limitations as implemented in the first prototype of SWMR-safe functionality.

In the current implementation of the HDF5 library, it is not possible to concurrently read from an HDF5 file that is also being written without explicit coordination between the processes. This unfortunately makes inspecting data while it is being collected impossible at the file level. In order to address this issue, we propose a mechanism to allow for concurrent access by a single writer process with any number of reader processes, a data access pattern known as single-writer/multiple-reader (SWMR - pronounced "swimmer").

For those who are curious, 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 and it is possible that a file metadata object can be flushed to disk before the other objects it references are flushed, creating 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 for obvious reasons.

Our proposed solution for supporting the SWMR pattern will be lock-free, will not require any communication between processes, and will not have a significant impact on I/O throughput, though file size may increase slightly[1]. Readers and writers will only have to set a single flag when opening a file for SWMR access,

[1] 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.

though we may add optional functions that allow users to tweak any user-visible SWMR settings from the defaults.

Building and Testing

<Describe where to get the code - tarball? svn tag?>

The distribution includes a simple test program that attempts to expose deficiencies in the I/O stack and file system that prevent proper SWMR operation. This test is non-deterministic and must be run in a loop in order to have a reasonable probability of exposing a potential issue. Instructions for running this test can be found in <LOCATION>.

The HDF5 library with SWMR capability is built by default in the prototype release. No special configuration options are required. The usual configure, make, make check, etc. process should be followed for building the library.

In the prototype, SWMR functionality is tested as a part of 'make check' as follows:

- The metadata cache test exercises the flush ordering functionality required for SWMR.

- Basic SWMR operation between processes is tested via a shell script that invokes a writer and some readers, which then operate on a central file using the SWMR access pattern. These are only acceptance tests and are a weak, though useful, test of SWMR. Unit testing of SWMR will come in later releases of the prototype.

Output from the tests is dumped to stdout and stderr, like all other HDF5 test results. No special log files are created.

Programming Model

To begin with, the current implementation of SWMR is very limited and only allows appending data to pre-existing datasets. No new file objects can be created under SWMR operation and variable-length data types are not supported. Updating dataset elements is supported, in theory, but with no notification mechanism, there's no way to tell which data have been modified.

Our main consideration in this stage of the prototype is appending data to chunked datasets, which is anticipated to be the most common scenario where SWMR semantics are needed. Any number of dimensions (up to the limits of the library) is supported and any of those dimensions can be unlimited in size.

There is one particular case that is not supported, due to a chunk indexing method that has not been fully converted to SWMR-safe semantics and tested in the current prototype: *Chunked datasets that use the latest version of the file format (set via the <CALL> call) AND that have no unlimited dimensions are not supported.* The work-around for this is to simply declare one dimension to be unlimited (H5S_UNLIMITED).

The basic SWMR setup is as follows:

1. **WRITER:** Create/open the HDF5 file and create any file objects (datasets, groups, etc.) that are required for data storage. Close the file.

2. **WRITER:** Open the file by calling H5Fopen with the SWMR_WRITE flag and begin writing to the dataset. New data will appear in the file as file objects are evicted/flushed from the cache. This can be more explicitly controlled by calling H5Fflush on the dataset ID.

3. **READER(S):** Open the file and begin reading data from the dataset. Since there is no push notification of changes (SWMR presumes no communication between processes), newly appended data is checked for via polling the dataset. Polling is done by opening 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. For each poll, the dataset must be closed and reopened so that the reader gets its information from the disk and not the reader's cache. When the size has been seen to change, the new data is read from the file using the usual H5Dread calls.

Note that, after step 1, the file can be opened by the writer and reader(s) in any order (i.e., 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 will be added in a future release.

The readers and writers can be created on the same or different machines, although it should be kept in mind that many network file systems (e.g., NFS) do not support the atomicity and ordering constraints required by SWMR (described in more detail below).

Examples of how to write readers and writers can be found in a few places in the prototype code:

- **test/swmr_*.c/h** - The SWMR test code. The swmr_reader/writer code is very straightforward and makes a good framework for writing your own

readers and writers[2]. The testswmr.sh script in the same directory shows how they are invoked.

- **hl/tools/h5watch/*** - A new tool that functions like the Unix tail command for an HDF5 dataset being written to using the SWMR feature. A good example of polling.

- **hl/src/H5LD*.c/h** - Several new functions used in h5watch that simplify monitoring a dataset for new data.

Limitations

Alpha software

At this time, SWMR is an experimental, unsupported feature and is not ready for production use.

Operating systems

SWMR has been tested and found to work on the following operating systems:

- Linux with GPFS

- FreeBSD with UFS2

- OS-X with HFS+

Windows has not been tested as our test code uses shell scripts and fork, which do not exist on Windows. Cygwin may work, but this has also not been tested.

We have a small test program that you can run to see if your system is likely to work under SWMR semantics.

Atomicity and ordering

SWMR requires the entire I/O chain to be atomic with respect to the POSIX write call and to preserve write ordering (often referred to as "POSIX ordering and atomicity"). Many file systems currently do not meet this requirement. In particular, the ext3 and ext4 file systems do not meet this requirement, as they are only atomic with respect to disk pages. NFS is also not compliant and cannot be used for SWMR.

---

[2] In the writer code, we cork and tweak the metadata cache for testing purposes. You don't need to do this. You can also ignore the complexity of the data we write (# of datasets, etc.) since that's a historical artifact.