Enhancing HDF5 with Multi-Threading, Sparse Data Storage, and Encryption

Preliminary Results and Demonstrations

Elena Pourmal elena.pourmal@lifeboat.llc
John Mainzer john.mainzer@lifeboat.llc

Lifeboat

Outline

- Intro to Lifeboat, LLC
- Toward multi-threaded HDF5
- Sparse data in HDF5
- Integrity of data in HDF5 (HDF5 encryption)

Lifeboat, LLC

U.S. DEPARTMENT OF ENERGY AWARDEE™

We don't make HDF5 - we make HDF5 *better*

- Goal: Sustain and enhance open source HDF5
  - Founded in August 2021; located in Champaign, IL and Laramie, WY
  - www.lifeboat.llc info@lifeboat.llc
- Funded by DOE SBIR/STTR Program
  - Phase I: "Toward Robust HDF5"
  - Phase II: "*Toward multi-threaded concurrency in HDF5*" (started in April 2023)
  - Phase II: "*Supporting sparse data in HDF5*" (started in April 2024)
  - Completed: Phase I: *"Protecting the confidentiality and integrity of data stored in HDF5" (HDF5 Encryption)*

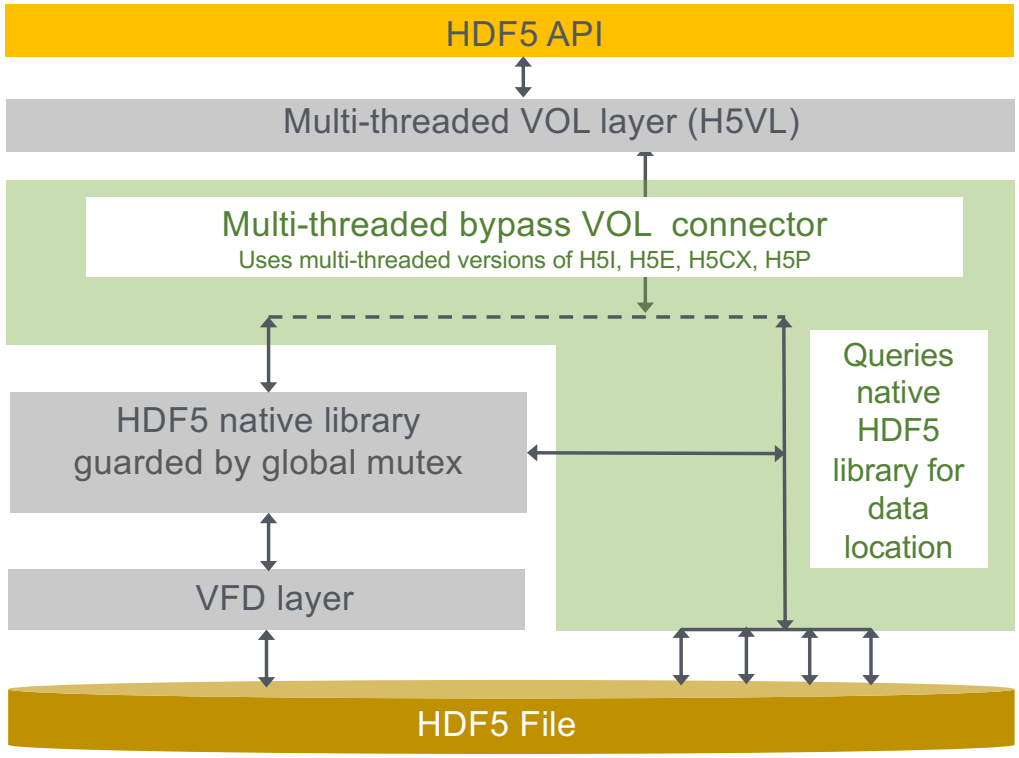
Toward multi-threaded concurrency in HDF5

- The goal is to allow multiple threads to enter the library without corrupting data structures in memory and data in the HDF5 file
  - HDF5 was designed as a single thread library
    - Thread safety supported via a global mutex – only one thread active in the library at a time.
    - This constraint is imposed on VOL connectors, even if they can support multi-thread operation.
- Approaches to multi-threaded HDF5
  - Retrofitting multi-thread support to an existing large, and largely un-documented code base is a daunting task.
  - Redesign and rewrite in contemporary language(s) that support multi-threading, e.g., C++ 11 or Rust.
  - Leverage existing VOL architecture.

Approach to multi-threaded HDF5

- Push the global mutex down to allow multiple threads of execution into multi-threaded VOL connector:
  - Retrofit multi-thread support onto “service” HDF5 packages – H5E (error reporting), H5I (index), H5P (property lists), H5CX (context), and H5VL (VOL).
- Status:
  - Implemented multi-threaded VOL connector to read contiguous or chunked data
    - Optional: VOL connector can use internal thread pool to parallelize I/O request.
  - Completed conversion to multi-threaded H5E, H5I, H5CX, and H5VL.
  - Designed changes to multi-threaded H5S.
  - We are currently testing multi-threaded H5P implementation and usage of the connector with multi-threaded HDF5 application

Bypass VOL architecture



Bypass VOL Concept

- Query HDF5 library for the location of raw data
- Execute raw data I/O in parallel in multiple threads

Current functionality

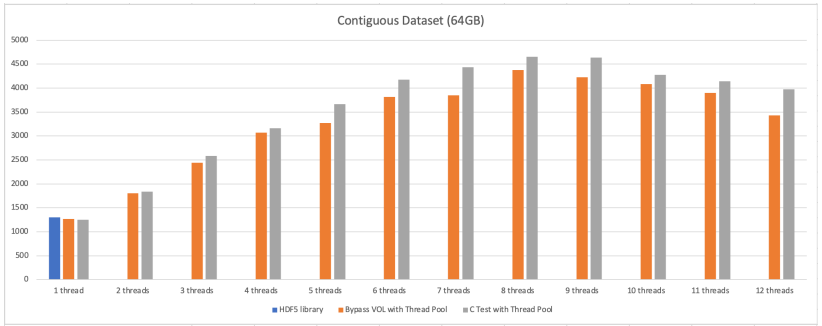
- Supports reads for a dataset with contiguous or chunked storage and numeric datatype; no data filtering
- Uses internal thread pool (optional)

Future enhancements

- Make H5S, H5T and H5FD layers multi-thread
- Redirect I/O to the VFD layer

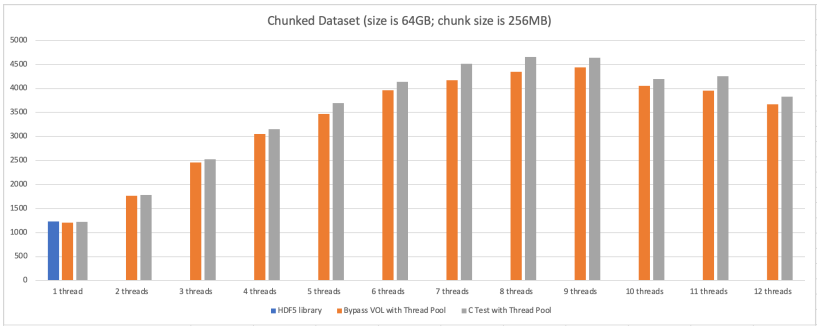
Bypass VOL connector with internal thread pool
Single threaded HDF5 application

Contiguous Dataset Reads



Dataset Size = 64GB on Linux	1 thread	2 threads	3 threads	4 threads	5 threads	6 threads	7 threads	8 threads	9 threads	10 threads	11 threads	12 threads
HDF5 library	1303											
Bypass VOL w	1268	1807	2442	3074	3276	3813	3852	4382	4226	4087	3902	3434
C Test with TI	1251	1838	2587	3160	3666	4178	4439	4652	4640	4281	4143	3980

Chunked Dataset Reads

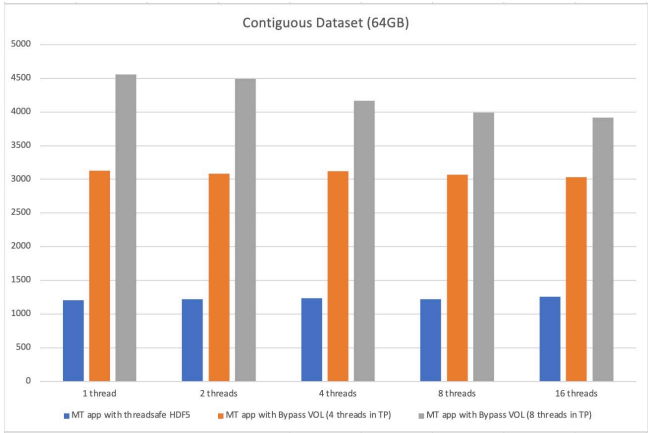


Dataset Size = 64GB on Linux	1 thread	2 threads	3 threads	4 threads	5 threads	6 threads	7 threads	8 threads	9 threads	10 threads	11 threads	12 threads
HDF5 library	1227											
Bypass VOL w	1206	1769	2457	3055	3472	3959	4173	4349	4442	4058	3955	3674
C Test with TI	1225	1784	2530	3151	3692	4141	4518	4653	4643	4195	4258	3833

64GB dataset, 256 MB chunks
Bypass VOL with the thread pool of 1 to 12 threads

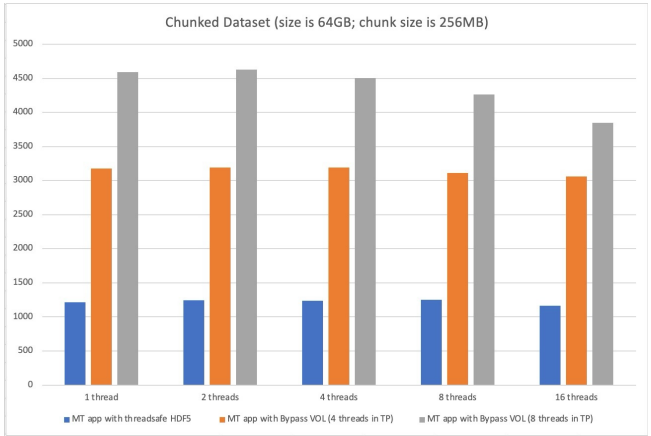
Bypass VOL connector with internal thread pool
Multi-threaded HDF5 application (1,2,4,8,and 16 threads)

Contiguous Dataset Reads



Contiguous Dataset Size = 64GB on Linux					
	1 thread	2 threads	4 threads	8 threads	16 threads
MT app with threadsafe HDF5	1208	1223	1232	1221	1255
MT app with Bypass VOL (4 threads in TP	3129	3085	3118	3067	3031
MT app with Bypass VOL (8 threads in TP	4560	4491	4165	3990	3913

Chunked Dataset Reads



Chunked Dataset Size = 64GB on Linux					
	1 thread	2 threads	4 threads	8 threads	16 threads
MT app with threadsafe HDF5	1215	1242	1237	1253	1166
MT app with Bypass VOL (4 threads in TP	3179	3191	3191	3114	3062
MT app with Bypass VOL (8 threads in TP	4591	4631	4504	4263	3846

- MT application with thread safe HDF5
- MT application with Bypass VOL with 4 threads in the pool
- MT application with Bypass VOL with 8 threads in the pool

Sparse and Variable-Length Data Storage in HDF5

New storage paradigm for sparse and variable-length data

New Storage Paradigm: Structured Chunk

Chunked dataset

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	66	69	72	75	78	81	0	0
0	0	96	99	102	105	108	111	0	0
0	0	126	129	132	135	138	141	0	0
0	0	0	0	0	0	0	0	0	2
100	0	-100	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	3	0

0 may represent a value that is not-defined

Chunked storage: all elements in the chunk are stored

```
0 0 0 0 0 0 0 0 0 0 0 0 66 69 72 0 0 96 99 96 102
```

Structured chunk storage for sparse data:

Locations and values of defined elements
are stored in different sections of the chunk

Section 0	Encoded selection
Section 1	66 69 72 96 99 96 102

Structured chunk storage for VL data:

Length of the values and values
are stored in different sections of the chunk

Section 0	len1, len2, len3,….
Section 1	string1, string2, string3,..

Structured chunk storage for sparse VL data:

Locations, length of each element and values
are stored in different sections of the chunk

Section 0	Encoded selection
Section 1	len1, len2, len3,….
Section 2	string1, string2, string3,..

Sparse Storage Implementation Status

- Design documents are in Lifeboat GitHub (see References slide)
  - Programming model and APIs
  - File Format extensions
  - Shared chunk cache
    - Better performing chunk cache including multi-threaded implementation
  - Improved I/O pipeline in HDF5 library
- Current status
  - Designed and implemented data structure in shared chunk cache
  - Implemented new public APIs to write sparse data and internal APIs to allow data flow through new cache
  - Started implementation of new filter APIs
  - Version 0 release in Summer 2025

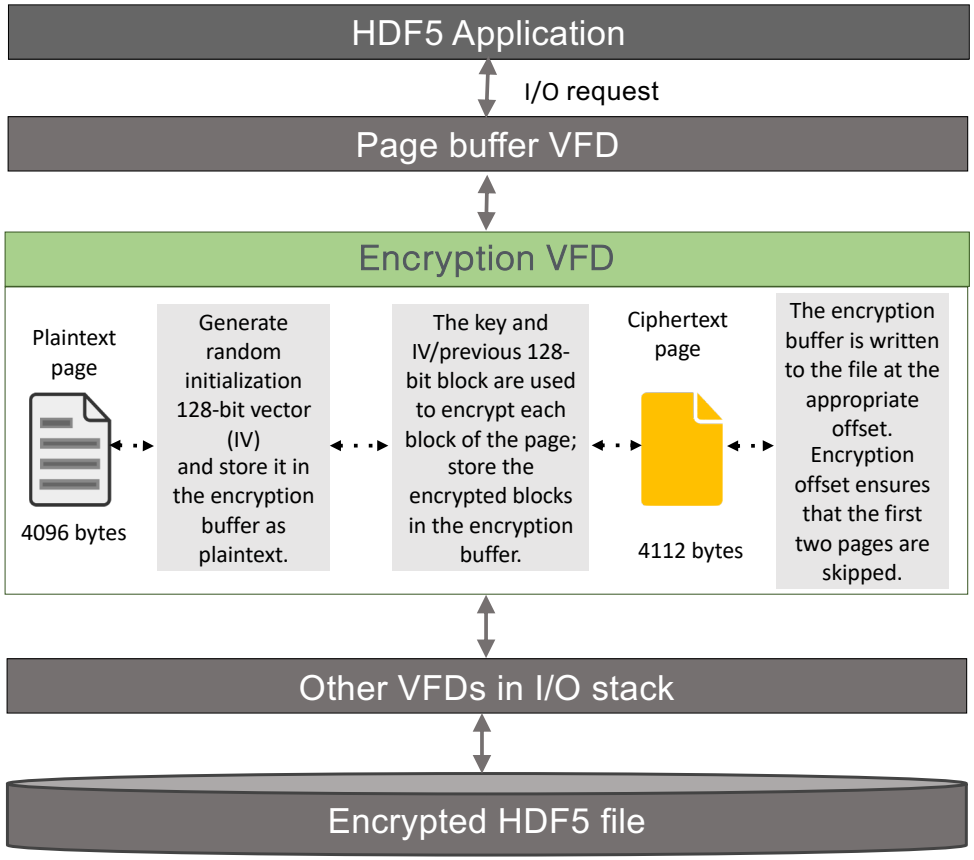
Integrity of Data in HDF5

HDF5 Encryption

Why introduce encryption to HDF5?

- Data integrity **is not guaranteed** by the HDF5 library
- HDF5 encryption:
  - Preserves data integrity
    - Encryption protects data from being stolen, *changed* or *compromised*
  - Assures data confidentiality and integrity during *data transfer, storage* and *computation*
- Design highlights:
  - Implemented as VFD
  - Supports all HDF5 features including partial I/O, compression, parallel etc., and any desired encryption methods
- Current prototype:
  - Supports any encryption method that allows computation of the maximum ciphertext page size from the plaintext page size
  - Built into the HDF5 release 1.14.3
  - Supports `AES` and/or `Twofish` symmetrical-key encryption from the GNU `gcrypt` library

HDF5 Encryption Implementation



- Prototype implementation
  - Page buffer VFD divides file address space into plaintext pages, translates incoming I/O request to or from page I/O, passes data up the stack or to encryption VFD
  - Encryption VFD handles offsets of the ciphertext pages, decrypts/encrypts pages, passes encrypted/decrypted pages to appropriate VFDs
  - See new files in the `src` sub-directory: `H5FDcrypt.c(h)`, `H5FDpb.c(h)`
- Future work
  - Support for parallel applications
  - Implement as pluggable and configurable VFDs
  - Use threads to speedup encryption

Lifeboat

Programming Model

- On creation:
  - Set up configuration of encryption and page buffer VFDs
  - Modify file access property with the calls to `H5Pset_fapl_crypt` and `H5Pset_fapl_pb` to use encryption and page buffer VFDs
  - Proceed with file creation and other operations as for plaintext HDF5 file
- On open:
  - Provide a key
  - Proceed as with operations for plaintext HDF5 file

Demo

in Working/HDF-Encryption/test-git-ssh-20240930/demo

See README in

https://github.com/LifeboatLLC/HDF5-Encryption

References

Documentation and code on GitHub

- Multithreaded project
  https://github.com/LifeboatLLC/MT-HDF5.git
  https://github.com/LifeboatLLC/hdf5_lifeboat.git

- Sparse project
  https://github.com/LifeboatLLC/SparseHDF5.git

- Encryption project
  https://github.com/LifeboatLLC/HDF5-Encryption.git

Acknowledgement

This work is supported by the U.S. Department of Energy, Office of Science under award number:

- DE-DE-SC0022506 for Phase II SBIR project "Toward multi-threaded concurrency in HDF5"
- DE-SC0023583 for Phase II SBIR project "Supporting Sparse Data in HDF5"
- DE-SC0025856 for Phase I SBIR project "Toward Robust HDF5"
- DE-SC0024823 for Phase I SBIR project "Protecting the confidentiality and integrity of data stored in HDF5"
- The HDF Group developers Jordan Henderson, Neil Fortner, Vailin Choi, and Matt Larson
- Lifeboat engineers Ray Lu, Kyle Lofthus, Dr. Clay Carper

Thank you!

Questions?