Evaluating Cloud-Optimized HDF5 for NASA's ICESat-2 Mission

Luis A. Lopez[1], Andrew P. Barrett[1], Amy Steiker[1], Aleksandar Jelenak[2], Lisa Kaser[1], Jeffrey E. Lee[3]

[1]CIRES, National Snow and Ice Data Center, University of Colorado, Boulder., Boulder, CO, USA
[2]The HDF Group, Champaign, IL, USA
[3]NASA Goddard Space Flight Center, NASA / KBR, Greenbelt, MD, USA

Abstract

The Hierarchical Data Format (HDF) is a common archival format for n-dimensional scientific data; it has been utilized to store valuable information from astrophysics to earth sciences and everything in between. As flexible and powerful as HDF can be, it comes with big tradeoffs when it's accessed from remote storage systems, mainly because the file format and the client I/O libraries were designed for local and supercomputing workflows. As scientific data and workflows migrate to the cloud , efficient access to data stored in HDF format is a key factor that will accelerate or slow down "science in the cloud" across all disciplines. We present an implementation of recently available features in the HDF5 stack that results in performant access to HDF from remote cloud storage. This performance is on par with modern cloud-native formats like Zarr but with the advantage of not having to reformat data or generate metadata sidecar files (DMR++, Kerchunk). Our benchmarks also show potential cost-savings for data producers if their data are processed using cloud-optimized strategies.

1 Problem

Scientific data from NASA and other agencies are increasingly being distributed from the commercial cloud. Cloud storage enables large-scale workflows and should reduce local storage costs. It also allows the use of scalable on-demand cloud computing resources by individual scientists and the broader scientific community. However, the majority of this scientific data is stored in a format that was not designed for the cloud: The Hierarchical Data format or HDF.

The most recent version of the Hierarchical Data Format is HDF5, a common archival format for n-dimensional scientific data; it has been utilized to store valuable information from astrophysics to earth sciences and everything in between. As flexible and powerful as HDF5 can be, it comes with big tradeoffs when it's accessed from remote storage systems.

HDF5 is a complex file format; we can think of it as a file system using a tree-like structure with multiple data types and native data structures. Because of this complexity, the most reliable way of accessing data stored in this format is using the HDF5 C API. Regardless of access pattern, nearly all tools ultimately rely on the HDF5-C library and this brings a couple issues that affect the efficiency of accessing this format over the network:

1.0.1 Metadata fragmentation

By default, file-level metadata associated with a dataset is stored in chunks of 4kb. This produces a lot of fragmentation across the file especially for data with many variables and nested groups.

1.0.2 Global API Lock

Because of the historical complexity of operations with the HDF5 format, there has been a necessity to make the library thread-safe and similarly to what happens in the Python language, the simplest mechanism to implement this is to have a global API lock. This global lock is not as big of an issue when we read data from local disk but it becomes a major bottleneck when we read data over the network because each read is sequential and latency in the cloud is exponentially bigger than local access.

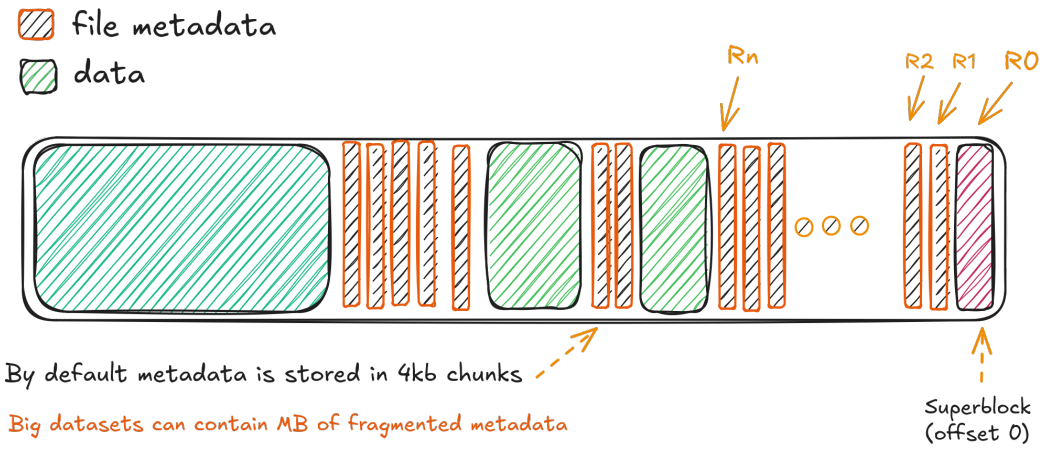

Figure 1: shows how reads (Rn) are done in order to access file metadata, In the first read, R0, the HDF5 library verifies the file signature from the superblock, subsequent reads, R1, R2,...Rn, read file metadata, 4kb at the time.

1.0.3 Background and data selection

As a result of community feedback and "hack weeks" organized by NSIDC and UW eScience Institute in 2023, NSIDC started the Cloud Optimized Format Investigation (COFI) project to improve access to HDF5 from the ICESat-2 mission. A spaceborne lidar that retrieves surface topography of the Earth's ice sheets, land and (oceans Neumann et al., 2019). Because of its complexity, large size and importance for cryospheric studies we targeted the ATL03 dataset. ATL03 core data are geolocated photon heights from the ICESat-2 ATLAS instrument. Each file contains 1003 geophysical variables in 6 data groups. Although our research was focused on this dataset, most of our findings are applicable to any dataset stored in HDF5 and NetCDF4.

2 Methodology

We tested access times to original and cloud-optimized small (1 GB), medium (2 GB) and large (7 GB) HDF5 ATL03 files [list files tested] stored in AWS S3 buckets in region us-west-2, the region hosting NASA's Earthdata Cloud archives. Files were accessed using Python tools commonly used by Earth scientists: h5py and Xarray. h5py is a Python wrapper around the HDF5 C API. xarray[1] is a widely used Python package for working with n-dimensional data. We also tested access times using h5coro, a python package optimized for reading HDF5 files from S3 buckets and kerchunk, a tool that creates an efficient lookup table for file chunks to allow performant partial reads of files.

HDF5 ATL03 files were originally cloud optimized by "repacking" them, using a relatively new feature in the HDF5 C API called "paged aggregation". Page aggregation does 2 things: it collects file-level metadata from datasets and stores it on dedicated metadata blocks in the file; and it forces the library to write data using fixed-size blocks. Aggregation allows client libraries to read file metadata with only a few requests and uses the page size used in the aggregation as the minimal request size, overriding the 1 request per chunk behavior.

[1] `h5py` is a dependency of Xarray

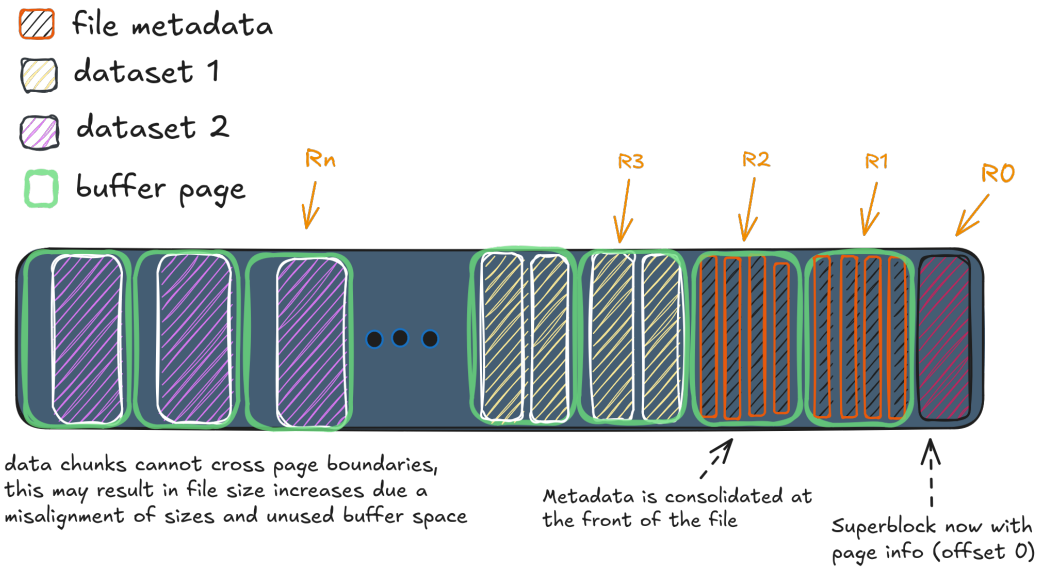

Figure 2: shows how file-level metadata and data gets internally packed once we use paged aggregation on a file.

3 Results

4 Recommendations

We have split our recommendations for the ATL03 product into 3 main categories, creating the files, accessing the files, and future tool development.

4.1 Recommended cloud optimizations

Based on our testing we recommend the following cloud optimizations for creating HDF5 files for the ATL03 product: Create HDF5 files using paged aggregation by setting HDF5 library parameters:

1. File page strategy: H5F_FSPACE_STRATEGY_PAGE
2. File page size: 8000000 If repacking an existing file, h5repack contains the code to alter tese variables inside the file

```
h5repack -S PAGE -G 8000000 input.h5 output.h5
```

3. Avoid using unlimited dimensions when creating variables because the HDF5 API cannot support it inside bffered pages and representation of these variables is not supported by Kerchunk.

4.1.1 Reasoning

Based on the variable size of ATL03 it becomes really difficult to allocate a fixed metadata page, big files contain north of 30MB of metadata, but the median sized file is below 8MB. If we had adopted user block we would have caused an increase in the file size and storage cost of approximate 30% (reference to our tests). Another consequence of using a dedicated fixed page for file-level metadata is that metadata overflow will generate the same impact in access times, the library will fetch the metadata in one go but the rest will be using the predefined block size of 4kb.

Paged aggregation is thus the simplest way of cloud optimizing an HDF5 file as the metadata will keep filling dedicated pages until all the file-level metadata is stored at the front of the file. Chunk sizes cannot be larger than the page size and when chunk sizes are smaller we need to take into account how these chunks will fit on a

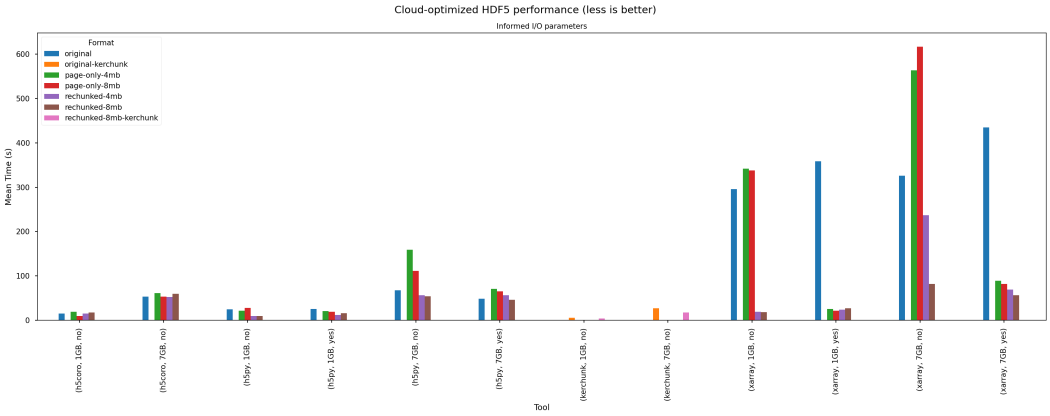

Figure 3: Benchmarks show that cloud optimizing ATL03 files improved access times at least an order of magnitude when used with aligned I/O patterns, this is telling the library about the cloud optimization and page size.

page, in an ideal scenario all the space will be filled but that is not the case and we will end up with unused space See 2.

4.2 Recommended access patterns

Placeholder

4.3 Recommended tooling development

Placeholder

4.4 Mission implementation

ATL03 is a complex science data product containing both segmented (20 meters along-track) and large, variable-rate photon datasets. ATL03 is created using pipeline-style processing where the science data and NetCDF-style metadata are written by independent software packages. The following steps were employed to create cloud-optimized Release 007 ATL03 products, while minimizing increases in file size:

1. Set the “file space strategy” to H5F_FSPACE_STRATEGY_PAGE and enabled “free space tracking” within the HDF5 file creation property list.
2. Set the “file space page size” to 8MiB.
3. Changed all “COMPACT” dataset storage types to “CONTIGUOUS”.
4. Increased the “chunk size” of the photon-rate datasets (from 10,000 to 100,000 elements), while making sure no “chunk sizes” exceeded the 8MiB “file space page size”.
5. Introduced a new production step that executes the “h5repack” utility (with no options) to create a “defragmented” final product.

5 Discussion

1. Chunking shapes and sizes
2. Paged aggregation vs User block
3. Side effects on different access patterns, e.g. Kerchunk

References

Neumann, T. A., Martino, A. J., Markus, T., Bae, S., Bock, M. R., Brenner, A. C., et al. (2019). The ice, cloud, and land elevation satellite – 2 mission: A global

geolocated photon product derived from the advanced topographic laser altimeter system. *Remote Sensing of Environment*, *233*, 111325. https://doi.org/https://doi.org/10.1016/j.rse.2019.111325