Pyfive: A pure-python HDF5 reader

Bryan Lawrence [1], Ezequiel Cimadevilla [2], Wout De Nolf [6], David Hassell [1], Jonathan Helmus[3], Brian Maranville [4], Kai Mühlbauer [5], and Valeriu Predoi [1]

1 NCAS-CMS, Meteorology Department, University of Reading, Reading, United Kingdom. **2** Instituto de Física de Cantabria (IFCA), CSIC-Universidad de Cantabria, Santander, Spain. **3** TBD **4** NIST Center for Neutron Research **5** Institute of Geosciences, Meteorology Section, University of Bonn, Germany. **1** European Synchrotron Radiation Facility (ESRF), Grenoble, France.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

DRAFT

Summary

Pyfive (https://pyfive.readthedocs.io/en/latest/) is an open-source thread-safe pure Python package for reading data stored in HDF5. While it is not a complete implementation of all the specifications and capabilities of HDF5, it includes all the core functionality necessary to read gridded datasets, whether stored contiguously or with chunks, and to carry out the necessary decompression for the standard options (INCLUDE OPTIONS). It is designed to address the current challenges of the standard HDF5 library in accessing data remotely, where data is transferred over the Internet from a storage server to the computation platform. Furthermore, it aims to demonstrate the untapped capabilities of the HDF5 format in the context of remote data access, countering the criticism that it is unsuitable for cloud storage. All data access is fully lazy, the data is only read from storage when the numpy data arrays are manipulated. Originally developed some years ago, the package has recently been upgraded to support lazy access, and to add missing features necessary for handling all the environmental data known to the authors. It is now a realistic option for production data access in environmental science and more widely. The API is based on that of h5py (which is a python shimmy over the HDF5 c-library, and hence is not thread-safe), with some API extensions to help optimise remote access.

Statement of need

HDF5 is probably the most important data format in physical science, used across the piste. It is particularly important in environmental science, particularly given the fact that NetCDF4 is HDF5 under the hood. From satellite missions, to climate models and radar systems, the default binary format has been HDF5 for decades. While newer formats are starting to get mindshare, there are petabytes, if not exabytes of existing HDF5, and there are still many good use-cases for creating new data in HDF5. However, despite the history, there are few libraries for reading HDF5 file data that do not depend on the official HDF5 library maintained by the HDFGroup, and in particular, apart from pyfive, in Python there are none that cover the needs of environmental science. While the HDF5 c-library is reliable and performant, and battle-tested over decades, there are some caveats to depending upon it: Firstly, it is not thread-safe, secondly, the code is large and complex, and should anything happen to the financial stability of The HDF5group, it is not obvious the C-code could be maintained. Finally, the code complexity also meant that it was not suitable for developing bespoke code for data recovery in the case of partially corrupt data. From a long-term curation perspective both of these last two constraints are a concern.

In addition, the paradigm of remote data access has gained traction in recent years due to the advantages it offers, primarily by eliminating the need for file downloads, a task that often consumes significant time for users engaged in data analysis workflows. This trend has accelerated with the increasing adoption of cloud platforms for storing environmental and climate data, which provide more scalable storage capabilities than those available to many research centers that produce the original datasets. The combination of remote data access and cloud storage has opened a new paradigm for data access; however, the technological stack must be carefully analyzed and evaluated to fully assess and exploit the performance offered by these platforms.

In this context, HDF5 has faced challenges in providing users with the performance and capabilities required for accessing data remotely in the cloud, showing relatively slow performance when accessed from cloud storage in a remote data access setting. However, the specific aspects of the HDF5 library responsible for this performance limitation have not been thoroughly investigated. Instead, the perceived inadequacy of HDF5 has often been superficially justified based on empirical observations of performance when accessing test files. Pyfive leverages the HDF5 format to provide efficient access to datasets stored in the cloud and accessed remotely, thereby contributing the currently missing knowledge needed to adequately assess HDF5 as a suitable storage format for cloud environments.

The original implementation of pyfive (by JH), which included all the low-level functionality to deal with the internals of an HDF5 file was developed with POSIX access in mind. The recent upgrades were developed with the use-case of performant remote access to curated data as the primary motivation, but with additional motivations of having a lightweight HDF5 reader capable of deploying in resource or operating-system constrained environments (such as mobile), and one that could be maintained long-term as a reference reader for curation purposes. The lightweight deployment consequences of a pure-python HDF5 reader need no further introduction, but as additional motivation we now expand on the issues around remote access and curation.

Taking remote access first, one of the reasons for the rapid adoption of pure-python tools like xarray with zarr has been the ability for thread-safe parallelism using dask. Any python solution based on the HDF5 c-library could not meet this requirement, which led to the development of kerchunk mediated direct access to chunked HDF5 data (https://fsspec.github.io/kerchunk/). However, in practice using kerchunk requires the data provider to generate kerchunk indices to support remote users, and it leads to issues of synchronicity between indices and changing datasets. pyfive was developed in such a way to have all the benefits of using kerchunk, but without the need for provider support. Because pyfive can access and cache (in the client) the b-tree (index) on a variable-by-variable basis, most of the benefits of kerchunk are gained without any of the constraints. The one advantage left to kerchunk is that the kerchunk index is always a contiguous object accessible with one get transaction, this is not necessarily the case with the b-tree, unless the source data has been repacked to ensure contiguous metadata using a tool like h5repack. However, in practice, for many use cases, b-tree extraction with pyfive will be comparable in performance to obtaining a kerchunk index, and completely opaque to the user.

The issues of the dependency on a complex code maintained by one private company in the context of maintaining data access (over decades, and potentially centuries), can only be mitigated by ensuring that the data format is well documented, that data writers use only the documented features, and that public code exists which can be relatively easily maintained. The HDF5group have provided good documentation for the core features of HDF5 which include all those of interest to the weather and climate community who motivated this reboot of pyfive, and while there is a community of developers beyond the HDF5 group (including some at the publicly funded Unidata institution), recent events suggest that given most of those developers and their existing funding are US based, some spreading of risk would be desirable. To that end, a pure-python code, which is relatively small and maintained by an international constituency, alongside the existing c-code, provides some assurance that the community can maintain HDF5 access for the foreseeable future. A pure python code also makes it easier to develop scripts which can work around data and metadata damage should they occur.

Examples

A notable feature of the recent pyfive upgrade is that it was carried out with thread-safety and remote access using fsspec (filesystem-spec.readthedocs.io) in mind. We provide two examples of using pyfive to access remote data, one in S3, and one behind a modern http web server:

For accessing the data on S3 storage, we will have to set up an s3fs virtual file system, then pass it to Pyfive:

```python
import pyfive
import s3fs


# storage options for an anon S3 bucket
storage_options = {
    'anon': True,
    'client_kwargs': {'endpoint_url': "https://s3server.ac.uk"}
}
fs = s3fs.S3FileSystem(**storage_options)
file_uri = "s3-bucket/myfile.nc"
with fs.open(file_uri, 'rb') as s3_file:
    nc = pyfive.File(s3_file)
    dataset = nc[var]
```

for an HTTPS data server, the usage is similar:

```python
import fsspec
import pyfive


fs = fsspec.filesystem('http')
with fs.open("https://site.com/myfile.nc", 'rb') as http_file:
    nc = pyfive.File(http_file)
    dataset = nc[var]
```

Mathematics

Single dollars ($) are required for inline mathematics e.g. $f(x) = e^{\pi/x}$

Double dollars make self-standing equations:

$$\Theta(x) = \left\{\begin{array}{l} 0\textrm{ if } x < 0\cr 1\textrm{ else} \end{array}\right.$$

You can also use plain $\LaTeX$for equations

$$\hat f(\omega) = \int_{-\infty}^{\infty} f(x) e^{i\omega x} dx \tag{1}$$

and refer to Equation 1 from text.

DRAFT

Citations

Citations to entries in paper.bib should be in rMarkdown format.

If you want to cite a software repository URL (e.g. something on GitHub without a preferred citation) then you can do it with the example BibTeX entry below for Smith et al. (2020).

For a quick reference, the following citation commands can be used: - `@author:2001` -> “Author et al. (2001)” - `[@author:2001]` -> “(Author et al., 2001)” - `[@author1:2001; @author2:2001]` -> “(Author1 et al., 2001; Author2 et al., 2002)”

Figures

Figures can be included like this: Caption for example figure. and referenced from text using section .

Figure sizes can be customized by adding an optional second parameter: Caption for example figure.

Acknowledgements

We acknowledge contributions from Brigitta Sipocz, Syrtis Major, and Semyeong Oh, and support from Kathryn Johnston during the genesis of this project.

References

Smith, A. M., Thaney, K., & Hahnel, M. (2020). Fidgit: An ungodly union of GitHub and figshare. In *GitHub repository*. GitHub. https://github.com/arfon/fidgit

DRAFT