Paradise Lost - Moving away from HDF5

Gerd Heber, The HDF Group

The HDF Group

Causa

Should you use HDF5?

Ref. 2

january 30, 2016
tags:

This is a follow-up on my post *Moving away from HDF5* (see also Konrad Hinsen's post, and discussions on Twitter and Hacker News). Here are some further thoughts, in no particular order.

First, others have pointed out alternative implementations of the HDF5 specification (complete or not), notably in Julia and Java. I haven't tried them so I don't know how good they are. I don't know of any alternative implementation in Python. It would be interesting to

n't depend on libhdf5.

Moving away from HDF5

Ref. 1

january 06, 2016
tags: python

Update [2016-01-30]: I wrote a follow-up here

In the research lab where I work, we've been developing a data processing pipeline for several years. This includes not only a program but also a new file format based on **HDF5** for a specific type of data. While the choice of HDF5 was looking compelling on paper, we found many issues with it. **Recently, despite the high costs, we decided to abandon this format in our software**.

In this post, I'll describe what is HDF5 and what are the issues that made us move away from it.

ght tool for us, others reported
es HDF5 with lots of tiny
so, we have large volumes of
simulations. These are quite

Cyrille Rossant, PhD CV
Neuroscience researcher and software engineer at IBL and UCL
- IBL: International Brain Laboratory
- UCL: University College London

firstname.lastname@gmail.com

#neuroscience #python #dataviz #maths #gpu #opendata

The HDF Group

First Things First

- Read it!
- I believe Mr. Rossant made an important contribution
- I (we) have the benefit of hindsight
- The first article is tainted by (understandable!) frustration
- The second article is more sober
- For the next ~15 min, I will focus on the good bits (= which make this an important contribution)
- The not-so-good bits are easily spotted by knowledgeable readers & many were addressed in the comments already
- We should follow up w/ a blog post of our own (volunteers?)

Reactions

- 43 comments on the first post
- Commenters
  - Pointed out independent (from the HDF5 library) implementations
  - Refuted some of the poor performance claims (mistakes in benchmark)
  - Referred to HDF5 success stories
- 18 comments on the second post
- Commenters
  - Called out alternatives (DBMS, Exdir, Zarr, ROOT, ASDF, ...)

Issues (from Ref. 1)

- High risks of data corruption
- Bugs and crashes in the HDF5 library and in the wrappers
- Poor performance in some situations
- Limited support for parallel access
- Impossibility to explore datasets with standard Unix/Windows tools
- Hard dependence on a single implementation of the library
- High complexity of the specification and the implementation
- *Opacity of the development and slow reactivity of the development team*

Conclusion (From Ref. 1)

We've learned our lesson. **Designing, maintaining, and promoting a file format within a community is hard**. It cannot be reasonably done by a small group of people who also need to write software, develop algorithms, and do research.

...

We've now rewritten our software to make it modular and completely agnostic to file formats. **We've moved from writing a monolithic application to writing a library**. We're encouraging our users to adapt these components to whatever file format they're already using. The APIs we provide make this straightforward.

There is always a tension, in that many of our users are biologists without a computer science background [to simplify, they're using Windows, Word, and MATLAB instead of Unix, vim/emacs, and Python] and they expect an integrated single-click graphical program. The solution we've found is to develop the library *first*, and *then* write separately an integrated solution based on this library.

Sharing vs. Access (Ref. 2)

Another thing is that we must make a distinction between creating, analyzing, and sharing a dataset. With our file format we tried to do all of these things with the same structure. As far as I understand it, this is what HDF5 encourages you to do. But these different use cases pose different constraints on how you store your data.

When creating a dataset, you want a fast write access. For analysis, you want a fast read access. For sharing, you want as few files as possible (ideally, one), with a clean internal structure. Of course this is overly simplistic.

It's hard to have a one-size-fits-all format. In our experience, HDF5 seemed to be a good option for sharing large datasets, but not that good for our peculiar read/write access patterns.

Four Legs Good, Two Legs Bad? (Ref. 2)

What we ended up doing at some point is using HDF5 only for sharing data. When importing the data into our software, we copied it into an internal format based on flat binary arrays. With this change, our software was much faster, at the expense of disk space and a longer initial loading time.

Effectively, we used a different format for sharing and for analyzing our data. If you need a file format, think hard about your requirements. Which is the most important to you: sharing, reading, writing?

ASDF “Straw Man” (Ref. 2)

[...] Because of the complexity, there is effectively only one implementation. The drawback of having only one implementation is that it may deviate from the published specification (who would know since there is no independent verification?). [...]

A related issue is that for some time the HDF format was not considered archival as it kept changing, and for a time it was considered more of a software API than a specific representation on disk. HDF5 has been relatively stable, though given the lack of multiple implementations and self documenting nature makes it less appropriate as an archival format. Will the future library be able to read much older files?

Decisions, Decisions, Decisions… (Ref. 2)

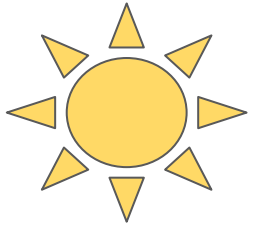
Hopefully you should now be in a position to decide whether HDF5 is the right tool for you, or if you need to explore other options. The main question you should ask is: do you absolutely need a portable container format containing many numerical arrays? If the answer is yes, you might have no other choice than HDF5, and you should be aware of its drawbacks. Do prototypes and benchmarks to avoid bad surprises in production.

Do We Need / Who Needs File Formats?

The more important question is: do you *really* need a file format in the first place? If you're targeting advanced users who are familiar with Python, it might be sufficient to provide a sensible API and let them deal with file format issues. Savvy users tend to prefer keeping control of their data.

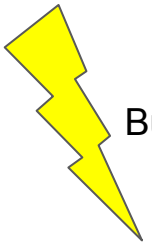
On the other hand, if your users aren't programmers and expect an easy-to-use integrated solution, you may have no other choice than deciding the file format and structure of the data generated by your software. This was our case. I tried to push hard our users (who are biologists) to learn Python and regain control of their workflows and data formats, but I failed. This is sad, as I think that in 2016 *any* researcher needs to know a programming language, Unix, bash, a version control system, etc. Still, many researchers continue to be allergic to command-line interfaces and programming languages, and you might have to comply with their requests. Maybe the customer is always right.

HDF5 “Project Management Triangle”



Sponsors

Cloud



Bugs

Data Model

Everything is an implementation detail
Don’t get much work done

GH (Me)

Being near the center is no guarantee for being useful

HSDS

(Fourth dimension?)

Use Cases

Tailored access
DIY library

H5Coro

DAOS VOL

No pesky format details
Data’s not going anywhere

File Format

Library

Is HDF5 a File Format or an API?

The HDF Group