DMR++ vs. Kerchunk Feature Study

Aleksandar Jelenak
Hyo-Kyung (Joe) Lee
The HDF Group

February 2023

Table of Contents

1 Introduction

Native cloud computing applications often rely on the Hypertext Transfer Protocol (HTTP) for data access. This protocol has a significant network latency, much larger compared to the conventional or high-performance file systems, so it is imperative for cloud applications to utilize any available optimization to achieve faster data access and reduce overall application runtime while retaining agile scalability.

The HDF5 library currently does not offer good options for HDF5 files hosted by cloud object storage systems. Since many of the NASA EOSDIS datasets are in this format, there is great need in alternative cloud-native methods for HDF5 data access.

One possible approach involves a *manifest file*, also called *sidecar file*, listing one HDF5 file's entire content. Having entire file content readily available is very useful for any cloud-optimized data access. This study examines two manifest file formats: DMR++ and Kerchunk. Both formats share the same fundamentals: a text format capable of representing HDF5 groups, datasets, their attributes including values, and dataset storage settings. The actual data in a file's HDF5 datasets are represented as byte ranges (file offset and size). When HDF5 files are rarely or not at all modified, their manifest files provide readily available source of information required for data access.

1.1 About DMR++

DMR++ is the XML-based format for describing the content of HDF5 and netCDF-4 files used by the Hyrax server. It was created by combining Hyrax server's Dataset Metadata Representation (DMR) document with the HDF5 dataset chunk file location and its storage settings (the "++" part). Access to HDF5 data described with DMR++ is currently only possible via the Hyrax server and its web services.

Hyrax server provides all the required software to generate DMR++ files. One DMR++ file describes one HDF5 file. Data aggregation of collections of DMR++ files employs the same techniques as for HDF5 files.

1.2 About Kerchunk

Kerchunk is a Python package, part of the *fsspec* project that provides a unified file system abstraction for many backend storage systems. Kerchunk's output is a representation of the content from one or more files as a virtual file system, called *ReferenceFileSystem*, which is then accessed with the *fsspec* API as just another storage backend. For this study, we will focus on the *ReferenceFileSystem* as applied to single files since this is the most relevant aspect of the Kerchunk's features for data access.

Kerchunk originated from a project by The HDF Group and funded by the US Geological Survey, to enable access to HDF5 and netCDF-4 data from the software stack based on the Python packages *zarr* and *xarray*. That project formulated an approach of translating HDF5 file content and storage settings to Zarr format, which formed the basis for Kerchunk's *ReferenceFileSystem*. It also established that access performance between an HDF5 file and a Zarr store with the same data and storage settings is statistically equal, and determined the same approach was likely applicable to many other formats for multidimensional data arrays.

Kerchunk is a very recent and still actively developed software. The activities are split between adding support for new data formats and tooling to efficiently generate aggregated Kerchunk (*ReferenceFileSystem*) documents from multi-file collections.

2 DMR++ and Kerchunk Feature Comparison

The main features of both DMR++ and Kerchunk are summarized in the table below.

Feature	**DMR++**	**Kerchunk**
Document format	XML	JSON
Data model	DAP	Zarr v2 (support for v3 is pending).
Programming language	C++	Python
Supported data formats	HDF5 (HDF-EOS5, netCDF-4).	HDF5 (HDF-EOS5, netCDF-4), TIFF, FITS, GRIB2, netCDF3.
HDF5 content support	Compatible with netCDF-4.	Compatible with netCDF-4.
Inline HDF5 compact layout dataset data	Yes	Yes
Inline HDF5 contiguous layout dataset data	No	Yes, with a configurable threshold.
Inline HDF5 variable-length data	Yes	Yes, with several inline storage options.
HDF5 filter support	Compression: deflate; Other: shuffle, fletcher32	Any filter supported by the *numcodecs* Python package. Includes: deflate, shuffle, fletcher32.
Subchunking	No	Only for netCDF-3 files.
Architecture	Static files, primarily intended for backend Hyrax server use.	Static files accessed directly via native object store web services.
Primary access software stack	Hyrax server	*xarray*, *zarr*, *fsspec* Python packages.
Supported storage locations	Local or network file system, AWS S3.	Any supported by the *fsspec* Python package. File system and AWS S3 are supported.
Data aggregation format	NcML	Kerchunk JSON
CF convention support	Yes. An option when generating DMR++ files.	*xarray* Python package.

Feature	DMR++	Kerchunk
Templating object location at runtime	Yes	Yes, in Kerchunk JSON v1 only.

Several entries in the table above warrant further clarification:

- *Data inlining* means including actual data in DMR++ XML or Kerchunk JSON document. This is a requirement for HDF5 datasets with compact layout because their file location (offset and size) is not currently available from the HDF5 library API. The maximum size of HDF5 compact datasets is 64 kilobytes and they are overall seldom present in the EOSDIS HDF5 files.
  The other need for data inlining is for HDF5 datasets with variable-length datatype. The current storage implementation of variable-length data by the HDF5 library does not allow for block-like (HTTP range GET) access. The EOSDIS granules do not contain large amounts of such data so inlining is not going to yield unwieldy DMR++ or Kerchunk documents.
  Kerchunk does support a few other storage options for variable-length HDF5 data, but we aren't sure whether they work as intended and did not explore further as this issue is not that important for this study.
- *Subchunking* means partial access to one chunk's data, i.e. "chunk within chunk". Only Kerchunk supports this and only for the old netCDF3 file format which does not have true chunking. This file format only permits extending arrays along the first dimension, making subchunk selections straightforward to apply.
- DMR++ document represent one file so data aggregation relies on NcML configuration documents, which is the standard Hyrax server feature. NcML does have some runtime aggregation update capabilities, although only if the files are in on-prem (local or network) file systems. Kerchunk provides a programmable method for data aggregation and the generated JSON document is static. Adding new files to the aggregation requires either regenerating the entire JSON document or manual JSON editing.

3 DMR++ and Kerchunk Software Comparison

We evaluated several software tools and libraries for ability to generate DMR++ or Kerchunk documents, interoperability in using these document formats, and current ability to access HDF5 data via these two formats. The following granules have been selected for this study as reasonable representatives of different HDF5 storage features present in the EOSDIS data:

- 3A-MO.GPM.GMI.GRID2014R1.20140601-S000000-E235959.06.V03A.h5
- GLDAS_NOAH025_3H.A20210101.0000.021.nc4.h5
- OMI-Aura_L2-OMNO2_2016m0215t0210-o61626_v003-2016m0215t200753.he5
- AMSR_2_L3_DailySnow_P00_20160831.he5
- VNP09A1.A2015257.h29v11.001.2016221164845.h5
- ATL08_20181014084920_02400109_003_01.h5
- SMAP_L3_SM_P_20150406_R14010_001.h5
- 20020602090000-JPL-L4_GHRSST-SSTfnd-MUR-GLOB-v02.0-fv04.1.h5

- 20220524080000-STAR-L2P_GHRSST-SSTsubskin-ABI_G17-ACSPO_V2.71-v02.0-fv01.0.nc.h5
- 20220607120000-STAR-L3S_GHRSST-SSTsubskin-LEO_PM_D-ACSPO_V2.80-v02.0-fv01.0.nc.h5
- SWOT_L2_HR_PIXC_007_483_235R_20220821T102608_20220821T102618_Dx0000_01.nc.h5

The software used in the study are:

- netCDF C library with NCZarr and OPeNDAP support
- Hyrax server (as Docker container)
- *zarr*, *xarray*, *xarray-datatree*, *pydap*, *kerchunk* Python packages

The details of the technical part of this study are in a special GitHub repository and its wiki.

4 Findings and Recommendations

We compared manifest file formats DMR++ XML and Kerchunk JSON for their capability to represent EOSDIS granules in HDF5-based file formats (HDF5, HDF-EOS5, netCDF-4). These manifest files are the foundation of their respective software stacks for access to HDF5 data without the HDF5 library.

DMR++ is based on the Data Access Protocol (DAP) while Kerchunk is based on the Zarr format. Both share the same approach for storing HDF5 dataset chunk location as a three-tuple of HDF5 file URI, chunk file offset, and chunk size (length). Thus, the differences are really about all the other content in these two formats.

It is important to note that the features of both DMR++ and Kerchunk are guided by how HDF5 data they represent are interpreted by their software stacks: Hyrax server and its clients for DMR++, and *xarray* & *zarr* Python packages for Kerchunk. Rather than going over their similarities, we will focus on important differences relevant for EOSDIS HDF5 data.

Kerchunk JSON reflects accurately HDF5 file hierarchy while DMR++ may not if the group flattening configuration option is set. This DMR++ functionality enables better support for Hyrax clients who expect certain aspects of the CF metadata conventions compliance. Kerchunk currently does not have this feature. In this context, it is important to note that the Kerchunk Python package provides programmable support for generating file aggregations from single-file Kerchunk JSON documents and that this functionality seems to be most interesting to its users. All modifications for improving data content compliance are applied at this level in Kerchunk.

The most consequential deficiency in Kerchunk JSON is the lack of support for HDF5 attribute datatypes. This comes from the Zarr format which only can store attributes as simple JSON values: string, number, boolean, array, or *null*. This prevents successful round-trip conversion of file content information between DMR++ and Kerchunk documents. This also makes compliance checking for certain conventions not possible because precise attribute datatype information is missing. In this regard, DMR++ has a clear advantage as it preserves attribute datatypes.

There is one extraneous attribute added to DMR++ named *build_dmrpp_metadata* that does not exist in source HDF5 file. It needs to be removed when converting to Kerchunk JSON.

Kerchunk inserts *_ARRAY_DIMENSIONS* Zarr attribute to store netCDF dimension/HDF5 dimension scale information. This attribute also needs to be excluded when converting to DMR++ and use the appropriate DMR++ feature for the attribute's information.

Chunk file locations is another critical type of information not possible to extract from Kerchunk JSON using the *zarr* Python package. This package *de facto* acts as the data I/O API for Kerchunk information. It is possible to get chunk file locations by directly parsing Kerchunk JSON. This approach is, however, dependent on the Kerchunk document format which already has two versions (V0 and V1).

Final issue related to HDF5 file content representation: both DMR++ and Kerchunk cannot currently store chunk filter mask, a bitmap value indicating which filters have actually been applied by the HDF5 library to each particular chunk. This is an obscure HDF5 storage feature and we have no indication any of the existing EOSDIS HDF5 files contain chunks with differing filter masks. Currently the DMR++ and Kerchunk software does not use chunk filter mask information. We think some basic level of checking should be implemented to ensure the correctness of DMR++ and Kerchunk file content information.

The sizes of generated DMR++ XML and Kerchunk JSON documents for our sample granules indicate that Kerchunk JSON is likely going to produce smaller files. The reasons are that JSON is a less verbose format than XML and Kerchunk JSON V1 format supports templated variables to avoid repeating strings like HDF5 file location for every chunk.

Producing DMR++ documents from our sample granules did not have any errors while we found several granules for which generating Kerchunk failed or omitted certain HDF5 content. This is expected given that DMR++ software is actively developed specifically for the EOSDIS HDF5 data. Kerchunk relies on the Zarr format and its community doesn't actively use much of the EOSDIS HDF5 data yet.

Below are conclusions and recommendations based on our study:

- DMR++ is more suitable for EOSDIS HDF5 files at present.
- DMR++ XML can be used to generate Kerchunk JSON without any loss of information.
- Lack of attribute datatypes in Kerchunk prevents it being used as the reference baseline for HDF5-based file content.
- DMR++ and Kerchunk software should consider the values of chunk filter masks during processing as an additional data integrity check.
- The real expertise in translating EOSDIS HDF5 file content to any manifest format, be it DMR++ or Kerchunk, is in the Hyrax server software (handlers, modules, etc.), and is the result of long-term development. Supporting any specific manifest file format is just a matter of exporting HDF5 file content information into a specific text-based format.
- Having Kerchunk available, statically or on-demand, would enable support of the open-source Python software stack based on the *xarray* and *zarr* packages with a very active and engaged developer and user community.