An HDF5 Schema for SKA Scale Image Cube Visualization

A. Comrie,[1,2] A. Pińska,[1,2] R. Simmonds,[1,2] and A. R. Taylor[1,2,3]

[1] *Inter-University Institute for Data Intensive Radio Astronomy, South Africa;* `angus@idia.ac.za`

[2] *University of Cape Town, Cape Town, South Africa*

[3] *University of the Western Cape, Cape Town, South Africa*

Abstract. We describe work that has been performed to create an HDF5 schema to support the efficient visualization of image cubes that will result from SKA Phase 1 and precursor observations. The schema has been developed in parallel to a prototype client-server visualization system, intended to serve as a testbed for ideas that will be implemented in systems developed to replace the existing CyberSKA and CASA viewers.

1. Introduction

Most astronomy image files are currently packaged using the FITS standard (Wells & Greisen (1979)); however, the FITS standard is not well-suited for storing or defining additional derived data products in a hierarchical structure. The HDF5 technology suite (Folk et al. (2011)) provides a data model, file format, API, library, and tools to enable the creation of structured schemas for different applications. We will show how these can be beneficial in packaging of radio astronomy (RA) data. In particular, our interest is in supporting fast interactive visualization of data that will be produced by the SKA telescope and its precursors.

Existing HDF5 schemas developed for RA data did not meet our requirements. The LOFAR HDF5 schema (Anderson et al. (2010)) did not meet performance requirements, because of its approach of storing each 2D image plane in a separate group. The HDFITS schema (Price et al. (2015)) serves as a starting point for an HDF5 schema that maintains round-trip compatibility with the FITS format, but lacks the additional structures required for pre-calculated and cached datasets. We have therefore created a new schema tailored to our application, although it may be advantageous for other processing and analysis applications as well. The schema is similar to that of HDFITS, but extensions have been added to support a number of features required for efficient visualization of large data sets.

2. Requirements

For our application, we use client-server visualization tools to view large RA image cubes remotely. We are currently working with data from MeerKAT Large Science

Projects, but aim to have a tool that will scale to the the data produced by the SKA. This scale of data needs to be maintained on compute clusters with large, fast storage systems: it is too large to download to workstations.

The server-based tool needs to be able to load image cubes quickly and to be able to start manipulating them without a large amount of initial processing. Reproducible results of commonly used compute or I/O intensive tasks therefore need to be pre-calculated and stored in a hierarchical structure for easy access. To support these aims, a number of different types of datasets are required:

- **Average datasets** store the average of the dataset along a particular axis (normally Z). These generally have a higher signal-to-noise ratio, and are useful during data visualization. Calculating them on the fly is computationally expensive. The name of the axis along which the average is taken should be indicated by the dataset name.

- **Permuted datasets** (e.g. $XYZ \rightarrow ZYX$) will allow for enormous speedups when reading image slices along non-principal axes. The schema defines how optional permuted datasets are stored in a standardized manner, so that software supporting the schema can check for these datasets when performing I/O-intensive dataset slices, such as reading a Z-profile at a given (X,Y) pixel value. The name of the permuted dataset should indicate the permuted layout.

- **Mip-mapped datasets** store a copy of the dataset, down-sampled across a particular image plane (e.g XY). As the visualization of large data generally requires down-sampling of generated images to match the user's viewport, this allows for the visualization of large data sets without loading entire image planes and performing down-sampling for each generated image. In addition, this will enable an efficient delivery of images to the client using tiling techniques commonly used in geographic information system (GIS) applications.

- **Histograms** defined along a particular image plane (e.g. XY or YZ) are I/O intensive to calculate, but relatively small and simple to store. For example, calculating the histogram for a 4096 × 4096 image slice takes approximately 80 ms of calculation time, while calculating the histogram for an entire cube can take far longer. Using the "square root" guideline, a histogram with $N = 4096$ would take an additional 16 KB of storage space. Stored histograms can be used to calculate approximate percentile values, which are commonly used in image visualization to restrict color mapping to a range of the data values, thus preventing outliers from skewing the color-mapped image. Approximate percentiles are sufficient for our purposes, provided that the number of histogram bins is large enough.

3. Schema

Initial tests of the schema are based on files converted from existing FITS files produced by scientists working on the MeerKAT data pipeline. When a FITS file is converted, a top-level group called `0`, which corresponds to the first Header Data Unit (HDU) in the original file, is created. Additional HDUs are stored in sequentially numbered groups, with the name of the HDU stored as the `NAME` attribute of each top-level group. The FITS data is saved as the `DATA` dataset of each top-level group.

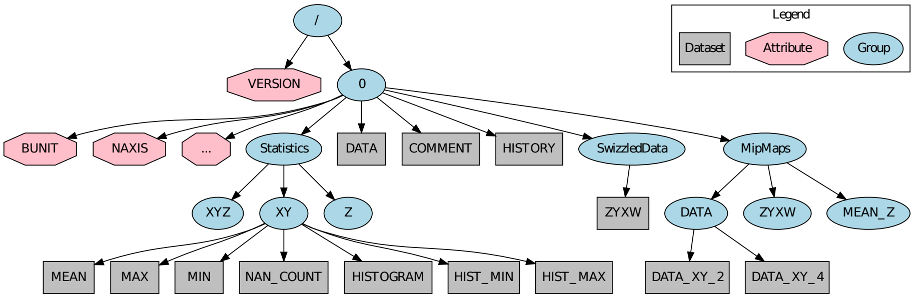

Figure 1. Outline of our HDF5 schema, indicating the structure of basic attributes, datasets and additional precomputed data.

FITS header entries for each HDU are stored as attributes of the relevant top-level group. This allows files in the schema to be translated back into FITS format if needed. We translate the `COMMENT` and `HISTORY` attributes to datasets rather than multidimensional attributes.

We show an outline of our proposed schema for storing these additional features in the HDF5 file in Figure 1. The name of each permuted dataset indicates the permuted order of axes. In the case of the example shown, a 4D cube (XYZ with the W-coordinate being Stokes parameters) has an additional dataset stored with the Z- and X-coordinates permuted. Statistics and mipmapped datasets are named as shown, with the mip factor indicated by the suffix of the dataset name.

A file will generally contain only a selection of the above additional features, depending on the application. We can strip features out by copying datasets selectively when offering downloads to clients. For example, permuted datasets and mipmaps are stored purely for performance reasons, and can be removed when we offer a download to clients, to minimize file size.

4. Performance

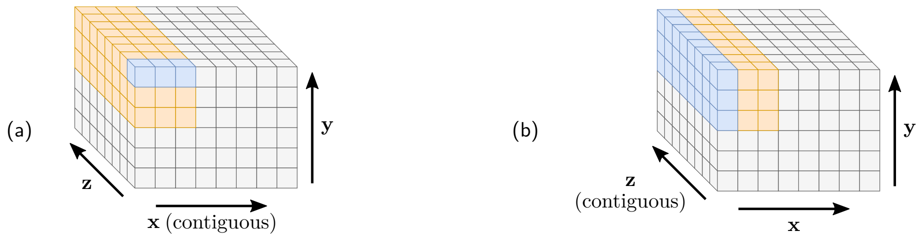

Figure 2. Example of accessing a $3\times3\times8$ region of an image cube with dimensions $8 \times 6 \times 8$, with the the contiguous coordinate chosen as (a) X and (b) Z.

In the example shown in Figure 2, a spectral profile is calculated from a $3 \times 3 \times 8$ region of an image cube with dimensions $8 \times 6 \times 8$. In the standard approach, when the X-coordinate is contiguous, each read operation consists of a $3 \times 4 = 12$ byte read, followed by a seek to the next row, yielding a total of $3 \times 8 = 24$ read operations. When we use the permuted dataset, with the Z-coordinate contiguous, each read operation

Table 1. Performance comparison of workloads on a 5850 × 1074 × 376 image.

Workload	Original [ms]	Permuted [ms]	Speedup
YZ-slice	5033 ± 7	2.08 ± 0.05	2420 ± 50
Z-profile	52.9 ± 0.1	0.182 ± 0.002	291 ± 4
32 × 32 × 376 region	340.1 ± 1.0	5.27 ± 0.04	64.5 ± 0.5
64 × 64 × 376 region	604.4 ± 1.7	13.1 ± 0.1	46.0 ± 0.5
128 × 128 × 376 region	876.8 ± 2.9	54.4 ± 0.7	16.1 ± 0.2

now consists of a $3 \times 8 \times 4 = 96$ byte read, followed by a seek to the next column, yielding a total of 3 read operations. For small read sizes, disk throughput is bounded by the total number of I/O operations per second. Therefore, reducing the number of read operations will dramatically increase disk throughput.

Several features inherent to HDF5 provide performance advantages. The Parallel HDF5 library provides support for multiple processes to write to a single HDF5 file simultaneously using MPI. This can speed up image cube generation. In addition, HDF5 allows us to alter dataset chunk size without changing the schema or code required to read the data, as this is abstracted by the HDF5 interface. This means that we can performance-tune different files to suit different common access patterns while maintaining schema compatibility.

Table 1 compares the execution time of common imaging workloads when data is read from the original dataset and from a permuted copy. All measurements were performed using an NVMe SSD, and the system buffer cache was cleared between each benchmark run to ensure that results were not skewed by operating system-controlled caching. Significant speedups are seen in all tested workloads when a permuted dataset is used, with the readout of YZ plane-aligned subsets being most affected. The speedup reduces as the size in the X and Y dimensions increases, which indicates that for regions above a threshold the original dataset should be utilized for maximum efficiency.

5. Summary

In this paper we have presented a new HDF5 schema for astronomical image data. We have explained our motivation for creating this schema to support our requirements for the visualization of large data from radio astronomy. We have provided an overview of the schema and the types of data access patterns that it supports. Initial tests of reading from a permuted dataset defined in the schema show significant benefits for commonly performed workloads.

References

Anderson, K., Alexov, A., Baehren, L., Griessmeier, J.-M., Wise, M., & Renting, A. 2010, PoS, ISKAF2010, 062. [ASP Conf. Ser.442,53(2011)], `1012.2266`

Folk, M., Heber, G., Koziol, Q., Pourmal, E., & Robinson, D. 2011, in Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases (ACM), 36

Price, D., Barsdell, B., & Greenhill, L. 2015, Astronomy and Computing, 12, 212

Wells, D. C., & Greisen, E. W. 1979, in Image Processing in Astronomy, 445