Astronomy and Computing 32 (2020) 100389

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

Full length article

Development and application of an HDF5 schema for SKA-scale image cube visualization

A. Comrie [a,*], A. Pińska [a], R. Simmonds [a], A.R. Taylor [a,b]

[a] Inter-University Institute for Data Intensive Astronomy, University of Cape Town, Cape Town, Western Cape, 7700, South Africa
[b] University of the Western Cape, Cape Town, Western Cape, 7535, South Africa

ARTICLE INFO

Article history:
Received 1 September 2019
Received in revised form 25 April 2020
Accepted 1 May 2020
Available online 15 May 2020

Keywords:
HDF5
File formats
Data visualization
Radio astronomy

ABSTRACT

In this paper, we describe an HDF5 schema created to support the efficient visualization of the large image cubes that will be produced by SKA Phase 1 and precursor radio telescopes. We demonstrate how the "HDF5-IDIA" schema's features can improve the performance of visualization software, using both low-level metrics and real-world tests of the schema's implementation in CARTA, an image viewer that is being developed to replace the existing CyberSKA and CASA viewers.

1. Introduction

Data produced by radio telescopes such as MeerKAT (Jonas et al., 2018), as well as other Square Kilometer Array (SKA) pathfinders, is now being used to create multi-dimensional image cubes which are large enough to pose multiple challenges to visualization software, requiring an increase in storage space, memory and computation resources. In our ongoing effort to facilitate efficient access to these images, we realized that the way in which the data was represented on storage systems created a considerable limitation to the performance that we could achieve. In particular, we wanted the ability to store additional pre-calculated representations and access them through a well-defined hierarchy.

Most astronomical image files are currently packaged using the FITS standard[1] (Wells and Greisen, 1979). However, the FITS standard is primarily used for image transport and archiving, and 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 large radio astronomy image cubes for the purpose of visualization and visual analytics.

We initially attempted to utilize existing HDF5 schemas developed for image cubes, but found that they did not meet our needs. The LOFAR HDF5 schema (Anderson et al., 2010; Alexov et al., 2012) did not meet performance requirements: each 2D image plane is stored in a separate group, therefore a separate group and dataset must be opened for each pixel when data is read along the third axis of the image cube.

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 precalculated and cached datasets. We have therefore created a new schema tailored to our application, with a hierarchy similar to that of HDFITS, but extensions have been added to support a number of features required for efficient visualization of large datasets.

The rest of the paper is laid out as follows: Section 2 details the requirements we have for the new schema. Section 3 considers the types of workloads that datasets using this schema will commonly be used for, in the context of image cube visualization. Section 4 describes the optional datasets that are defined in the schema in order to accelerate these workloads, as well as an outline of the schema hierarchy and naming conventions. Section 5 details integration of support for the schema into CARTA: The Cube Analysis and Rendering Tool for Astronomy (Comrie et al., 2018), as of Version 1.2 of the software package. Section 6 shows performance metrics of the schema with low-level benchmarks, and compares its performance to that of FITS within CARTA.

2. Requirements

Our application is the use of client–server visualization tools to view large image cubes remotely, with the image cube remaining

* Corresponding author.
E-mail address: angus@idia.ac.za (A. Comrie).

[1] Available at https://fits.gsfc.nasa.gov/fits_home.html.

https://doi.org/10.1016/j.ascom.2020.100389
2213-1337/

on the server and data that is currently being examined streamed to a browser-based viewer on an end user's computer. We are currently working with data from the MeerKAT Large Survey Projects (LSPs) and the Atacama Large Millimeter/submillimeter Array (ALMA) (Wootten and Thompson, 2009), with the aim of developing a tool that will scale to support data produced by the SKA. This scale of data needs to be maintained on compute clusters with high-capacity, fast storage systems: individual image cubes are often too large to be downloaded to a workstation for local use, either because of the time it would take to download them, or because they are larger than the storage systems on most end users' computers. A remote, distributed approach mitigates these challenges (Hassan and Fluke, 2011).

At millimeter and sub-millimeter wavelengths, broadband molecular line spectral image cubes of increasing spectral and spatial resolution are being generated by antenna arrays. ALMA images can contain up to 7680 channels for a single polarization, or 3840 channels for dual polarization.[2] SKA pathfinder projects are creating wide-field, wide-band spectral cubes for deep searches to detect extragalactic atomic hydrogen, and for spectro-polarimetric imaging to probe the Faraday signatures of cosmic magnetic fields.

The large instantaneous field of view and arcsecond-scale resolution of MeerKAT (Jonas et al., 2018) requires frames of 8 k × 8 k pixels to image a single pointing. With up to 32768 spectral channels in each polarization, data cubes of tens of terabytes per pointing will be typical. Multi-field mosaicking, as will be carried out for the MeerKAT MIGHTEE deep imaging survey (Jarvis et al., 2016), and images from very large field-of-view SKA pathfinders such as ASKAP (Johnston et al., 2007), require 64 k × 64 k pixel image frames.

Terapixel image cubes are now also being generated by wide-field VLBI experiments that image large fields of view at milli-arcsecond resolution (Deane, 2016). SKA-1 mid will have a similar field of view to MeerKAT but 10 times the angular resolution, requiring image frames on the order of 100 k × 100 k for full imaging of the primary beam of a single pointing. Image cubes of hundreds of terabytes will be typical.

In order to serve a large number of users concurrently and with minimal startup delay, the server-based application needs to load image cubes quickly and to be able to start manipulating them without a large amount of initial processing. Results of commonly used compute or I/O-intensive tasks therefore need to be precalculated and stored in a hierarchical structure for easy access.

Compute clusters commonly use hard drive-based distributed file systems for primary storage, and thus heavily favor large sequential reads over small random reads that commonly occur during exploration of a large image cube. Efforts must therefore be made to reduce the required number of random reads by rearranging or duplicating data for optimal use during commonly used workloads, some of which are discussed below. For simplicity, we assume that image cubes consist of two spatial axes (X and Y, with width W and height H and an image area of $W \times H$), and one spectral axis (Z, with depth D, also referred to as the number of channels), and require 4 bytes per pixel. Cubes of this configuration generally have data stored with the X coordinate contiguous, followed by the Y- and Z-axes. The schema is designed to support more axes, such as the Stokes axis in full-polarization cubes.

[2] As of time of writing, as specified at https://almascience.nrao.edu/about-alma/alma-basics.

3. Common workloads

Some commonly used workloads are described below, along with their associated read access patterns. The access pattern calculations are based on a contiguous data layout, and would be different if a tiled data layout, such as the HDF5 file format's chunking approach (Folk et al., 2011) or the CASA file format (McMullin et al., 2007), were used.

3.1. Rendering a 2D slice

Rendering a 2D slice of an image cube requires reading a single plane of the cube from disk, before mapping the data to a color, based on a defined transfer function. Simple color mapping transfer functions clamp the data to a chosen value range, apply a transformation to scale the data to a value between zero and one, and use the scaled value as a lookup index in a color map. Alternatively, the slice data can be rendered in the form of contour lines generated for a given set of levels, usually after it has been processed with an appropriate method in order to reduce noise.

Choosing appropriate minimum and maximum bounds for the transfer function, as well as appropriate contour levels, often requires interrogation of the data distribution, either for the individual slice or the entire cube. Histograms of the distribution are often sufficient for this purpose.

Most commonly, a slice of the cube aligned to the XY-plane at a given Z-axis coordinate is chosen. Other projections, such as a combination of a single spatial axis with the spectral axis (position–velocity images) are also widely used. Rendering an XY-slice requires fewer I/O operations than other alignments, as the slice can be acquired using a single sequential read of $W \times H$ pixels. Rendering XZ- or YZ-slices is considerably more I/O-intensive, as the slice must be acquired using a large number of smaller non-sequential reads. Reading an XZ-slice requires D reads of size W pixels, while reading a YZ-slice requires $D \times H$ reads of a single pixel, or D reads of size $W \times H$ pixels (discarding all but H pixels of each read). Utilizing a tiled data storage pattern can significantly reduce the asymmetry of these different workloads, but will result in reduced performance in the most common (XY-slice) workload.

The dataset is also sometimes averaged along a particular axis, in order to produce an image with a higher signal-to-noise ratio. For example, a user might view an average image to identify faint signals more effectively, but still utilize the full cube for further analysis, such as calculating a spectral profile.

3.2. Single-pixel profiles

Users will commonly want to display the X-, Y- and Z-profiles for a chosen pixel. While an X-profile for a fixed Y and Z coordinate can be obtained through a single contiguous read of W pixels, reading profiles for the other axes is much slower. Reading a Y-profile requires either H reads of a single pixel, or a single read of $W \times H$ pixels (discarding all but H pixels for each read). Reading a Z-profile requires D reads of a single pixel.

3.3. Region profiles and statistics

In addition to the profiles described in Section 3.2, users will often want to select a 3D region of the cube, and reduce it along two axes to a single profile along the remaining axis. One of the most common workloads of this nature is to generate a spectral profile of an $N \times M \times D$ region (where N and M are the spatial dimensions of the region and D is the cube depth), reducing it along the spatial axes to produce a profile along the Z-axis with a

length of D. We assume a constant D, as the user typically wishes to examine the full spectrum. Reducing an $N \times M \times D$ region to a single Z-profile of length D requires $M \times D$ reads of N pixels, or D reads of $W \times M$ pixels (discarding all but $N \times M$ pixels for each read). The spatial area $N \times M$ of these regions is generally much smaller than the image area $W \times H$, so the overhead of reading additional pixels and discarding them can be prohibitively slow.

3.4. Zooming and panning

If an image has a higher resolution than that of the user's viewport (often only a portion of the screen itself), it does not need to be transferred to the client device in its entirety. Instead, the image may be cropped to match the viewport resolution when the user requests a 100% zoom, or it may be downsampled prior to sending, if the user requests a wider field of view. As the user zooms or pans around the image, the server delivers the required image data to the user.

If the user is viewing the entire image, even at a reduced resolution, the entire channel must be loaded from disk before it can be downsampled. Some remote viewers use a tile-based approach, limiting the downsampling to block averaging, with power-of-two block sizes, and progressively stream the image in fixed tile sizes. This limits the number of times that new data must be sent to the client, and allows the client to reuse cached portions of the image while awaiting higher-resolution data.

4. Schema

4.1. Optional datasets

To accelerate the workloads described in Section 3, several different types of datasets are required, each of which is discussed below. These datasets are defined in the schema hierarchy described in Section 4.2, but each dataset is optional, and applications attempting to read files adhering to the schema should not require any of the optional datasets to be included in a file.

4.1.1. Aggregate datasets

Aggregate datasets store the data for the entire cube, reduced along a particular axis using a combining function such as the mean. The Z-axis is usually chosen, but the schema is general enough to support averages along any axis. Calculating aggregate datasets on the fly is I/O- and computationally expensive, requiring the entire $W \times H \times D$ cube to be read from disk, while reading a precalculated image reduced along the Z-axis requires only a read of a single $W \times H$ slice. Datasets reduced along the same axis are collected within an appropriately named group. Dataset names indicate the combining function used. This approach is similar to that of the GIPSY data model (Van der Hulst et al., 1992) and the Starlink Extensible N-Dimensional Data Format.[3]

4.1.2. Histograms

As mentioned in Section 3.1, histograms 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. 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 on a typical desktop PC, while calculating the histogram for an entire cube can take far longer. Using the "square root" guideline (where the number of bins is equal to the square root of the number of pixels), a histogram with 4096 bins would require an additional 16 KB of storage space. Stored histograms can be used to calculate approximate percentile values. Approximate percentiles are sufficient for our purposes, provided that the number of histogram bins is large enough. Storing histograms for each channel and for the entire cube allows users to switch seamlessly between a color mapping configuration which is adjusted for each channel and one which remains constant for all channels. While users may need histograms with a different range or bin count, these pre-calculated histograms will commonly offer sufficient flexibility for users.

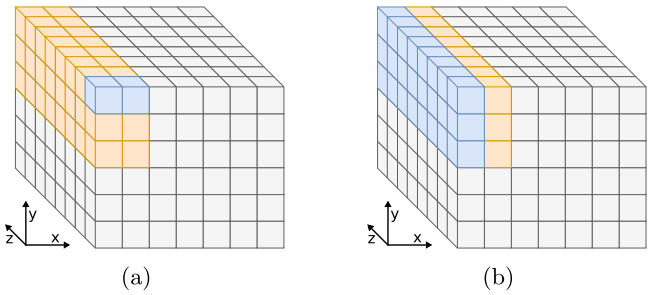

Fig. 1. Example of accessing a $2 \times 3 \times 8$ region of an image cube with dimensions $7 \times 6 \times 8$, with the contiguous coordinate chosen as (a) X and (b) Z.

4.1.3. Permuted datasets

Storing an additional copy of the data with permuted axes (e.g. $XYZ \rightarrow ZYX$, changing the contiguous axis from X to Z) allows for enormous performance improvements when image slices are read along non-contiguous axes. In the example shown in Fig. 1, a spectral profile is calculated for a $2 \times 3 \times 8$ region of an image cube with dimensions $7 \times 6 \times 8$. In the standard approach, when the X-coordinate is contiguous, each read operation consists of a $2 \times 4 = 8$ 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 YZ read operation now consists of a $3 \times 8 \times 4 = 96$ byte read, followed by a seek to the next column, yielding a total of 2 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 dramatically increases disk throughput.

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 indicates the permuted layout. Software making use of the permuted datasets can determine which copy of the dataset to read from, depending on the access pattern.

4.1.4. Mipmapped datasets and tiling

Mipmapped images (Williams, 1983) are commonly used in real-time computer graphics to store multiple copies of an image, with the dimensions of each subsequent copy of the image decreasing by a factor of two in each dimension. This increases the total storage size by one third, but allows an appropriate precalculated "mip" to be chosen for display, rather than downsampled from the original full-resolution image. It is also useful for texture streaming (Van Waveren, 2006), a technique used to load images progressively. In the context of our schema, we store copies of the dataset downsampled into different resolutions across a particular image plane (e.g. XY). While regular mipmaps are downsampled until the image is a single pixel in size, we

[3] Available at http://starlink.eao.hawaii.edu/docs/sun33.htx/sun33.html.

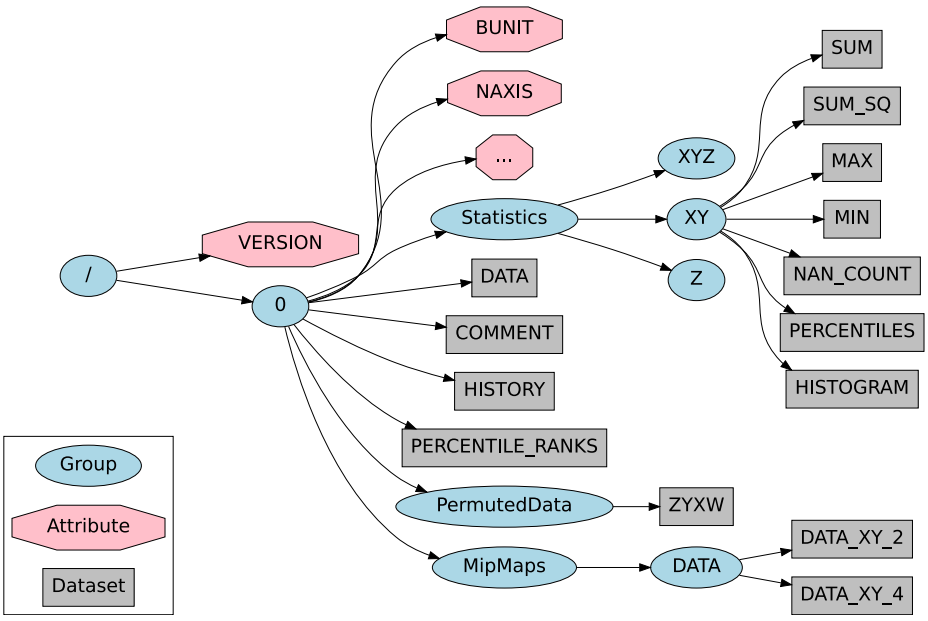

Fig. 2. Outline of our HDF5 schema, indicating the structure of basic attributes, datasets and additional precomputed data.

perform downsampling until the image has at most 256 pixels along either axis. We found that images below 256 × 256 have little practical value for our application.

Precalculation and storage of this data reduces both processing overhead and memory usage by eliminating the need to load and downsample entire image slices on demand during visualization of large datasets. In addition, it enables an efficient delivery of images to the client using tiling techniques commonly used in geographic information system (GIS) applications, which are under development in CARTA.

To optimize reads of *XY* tiles from the main dataset and the mipmapped datasets, at the cost of some reduction in performance of reads of *XY* slices with arbitrary dimensions, we are transitioning these datasets from contiguous storage to a chunked storage layout (The HDF Group, 2019). We use 2D chunks and a tile size of 256 × 256. Because we use the rotated dataset for reads of small *XY* regions over a large *Z* range, 3D chunks would provide no additional benefit and only decrease the performance of the most common use case of reading tiles from a single channel at a time.

4.2. Schema hierarchy and naming

Initial tests of the schema are based on files converted[4] 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.

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 schema for storing these additional features in the HDF5 file in Fig. 2. 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 downsampling 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.

5. Integration into CARTA

We have integrated support for HDF5 files which use our schema into the CARTA software package, a viewer designed to provide performant access to very large astronomical images, and developed as an eventual replacement viewer for both the CASA astronomical software package (McMullin et al., 2007) and the CyberSKA portal (Kiddle et al., 2011).

CARTA has a client–server model: remotely stored images are viewed through a web interface. Portions of image data are read by the server, downsampled to match the client display resolution, compressed, and sent to the client as they are requested. The server also performs certain I/O- and CPU-intensive calculations, while rendering is done on the client machine where it may take advantage of local graphics hardware.

Several features of the schema are used by the CARTA server both to speed up reads from HDF5 files and to avoid certain expensive calculations. Basic support for reading files using the schema was introduced in Version 1.0, with additional features added in subsequent versions to improve performance and take advantage of the optional datasets described in Section 4.1.

- **Histograms** are stored for each channel and for each Stokes cube. In addition to displaying the histograms to the user, CARTA uses them to approximate percentile clipping values. When the cube is animated by stepping through the image for each channel, calculating the histogram for each channel on demand adds a delay before each animation frame can

[4] Converter available online at https://github.com/idia-astro/cpp_hdf5converter.

be rendered for the first time. Calculating a whole-cube histogram in order to use the same color map bounds for all channels adds a long delay before the animation is played for the first time, as the entire cube has to be read from disk. These delays are eliminated when stored values are used.

- **Aggregate datasets** are also stored per channel and per Stokes cube. When statistics are requested for the whole image, these values can be read, and not calculated as they would be for any other image region. In earlier versions, we stored the mean, minimum, maximum and count of NaN values; however, in the latest version we have replaced the mean with the sum and sum of squares. The mean, root mean squared and standard deviation can be derived inexpensively from these basic statistics.
- **Permuted datasets** speed up the calculation of spectral profile data for regions by reducing the required number of reads. A region may be a single *XY* coordinate, or a polygonal or elliptical area selected within the image. For a point region a single contiguous row of data is read from the *ZYXW* dataset. Larger regions are read one contiguous *ZY* slice at a time, up to a region size cutoff where $Y \times Z < X$, at which point using the original dataset requires fewer reads. This speed increase allows the spectral profile for a region to be refreshed more quickly as the user moves the region within the image.

Integration of these features has led to significant performance improvements for visualizing large images, as shown in Section 6.2.

Our latest development is focused on the utilization of tiled and mipmapped datasets when loading HDF5 files using this schema. As of CARTA 1.2, the server must load an entire channel into memory in order to downsample the image before sending it to the client. With the addition of mipmapped datasets and a 2D tiling model described in Section 4.1.4, the server will load only the tiles required to render the image on the user's viewport, and only at the required resolution.

This significantly reduces the memory requirements of the server, making them mostly independent of image size, and permitting the server to open images with an image size exceeding the available physical memory. It also reduces the total size of data which is read from disk and eliminates on-demand downsampling calculations. This in turn improves the speed with which the image is initially loaded and re-rendered after every channel change, which is particularly important during animations.

Once loaded, tiles may be stored for reuse in an least-recently-used (LRU) cache with a fixed maximum capacity tuned according to the memory available and the number of concurrent clients, thereby allowing for better scaling when serving a large number of clients or serving large images to multiple clients.

6. Performance

6.1. Standalone performance tests

We compared the execution time of common imaging workloads described in Section 3.1, 3.2 and 3.3 when data was read from the original dataset and from a permuted copy. Measurements were performed on three sets of synthetic images created using the Astropy package (Robitaille et al., 2013) and filled with Gaussian noise, with increasing square image area and increasing numbers of channels.[5] We performed measurements on a subset

[5] Script is available at https://github.com/idia-astro/image-generator.

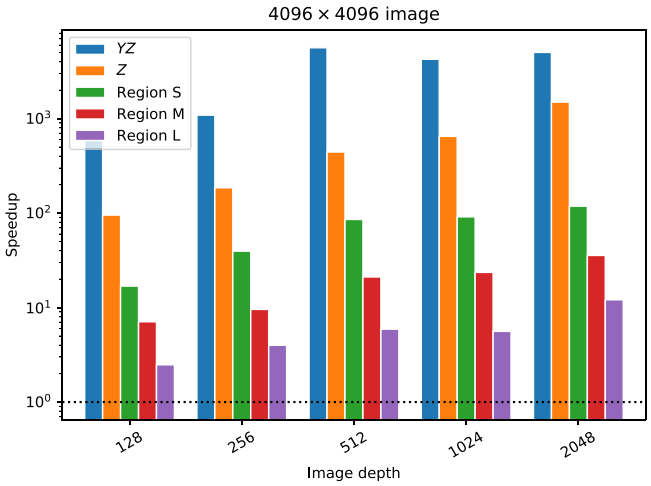

Fig. 3. Performance speedup on SSD versus image depth for a number of workloads described in Section 3.

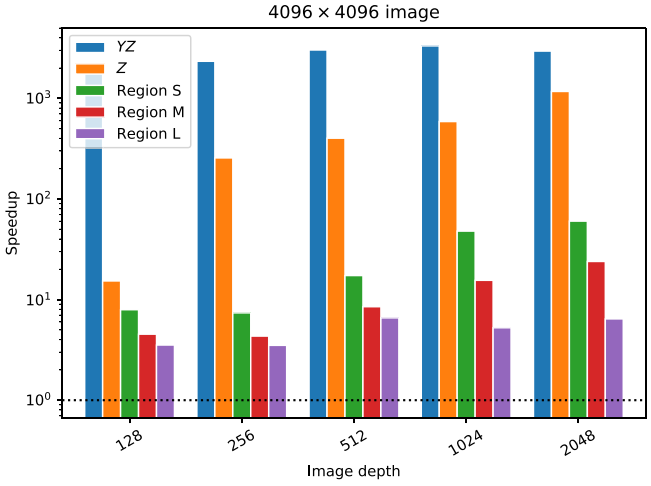

Fig. 4. Performance speedup on HDD versus image depth for a number of workloads described in Section 3.

of the 4096 square pixel images using both a PCIe 3.0 NVMe SSD (as shown in Fig. 3) and a 5400 rpm SATA HDD (Fig. 4), clearing the system buffer cache before each benchmark run to ensure that results were not skewed by operating system-controlled caching. Benchmarks were performed on an otherwise unloaded machine, and each benchmark was run ten times, with the average result reported.

We tested five workloads: reading a single-pixel *Z* profile, reading a *YZ*-slice, and reading small, medium and large square regions, defined as 0.01%, 0.1%, and 1% of the image area, respectively. The coordinates of these selections within the image were randomized for every run. The speedup of each workload is defined as t_o/t_p, where t_o and t_p are the times taken to read the data from the original dataset and the permuted dataset, respectively.

Significant speedups are seen in all tested workloads when a permuted dataset is used, with the reading of *YZ*-slices and *Z*-profiles being most affected. The region workload speedup reduces as the region 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.

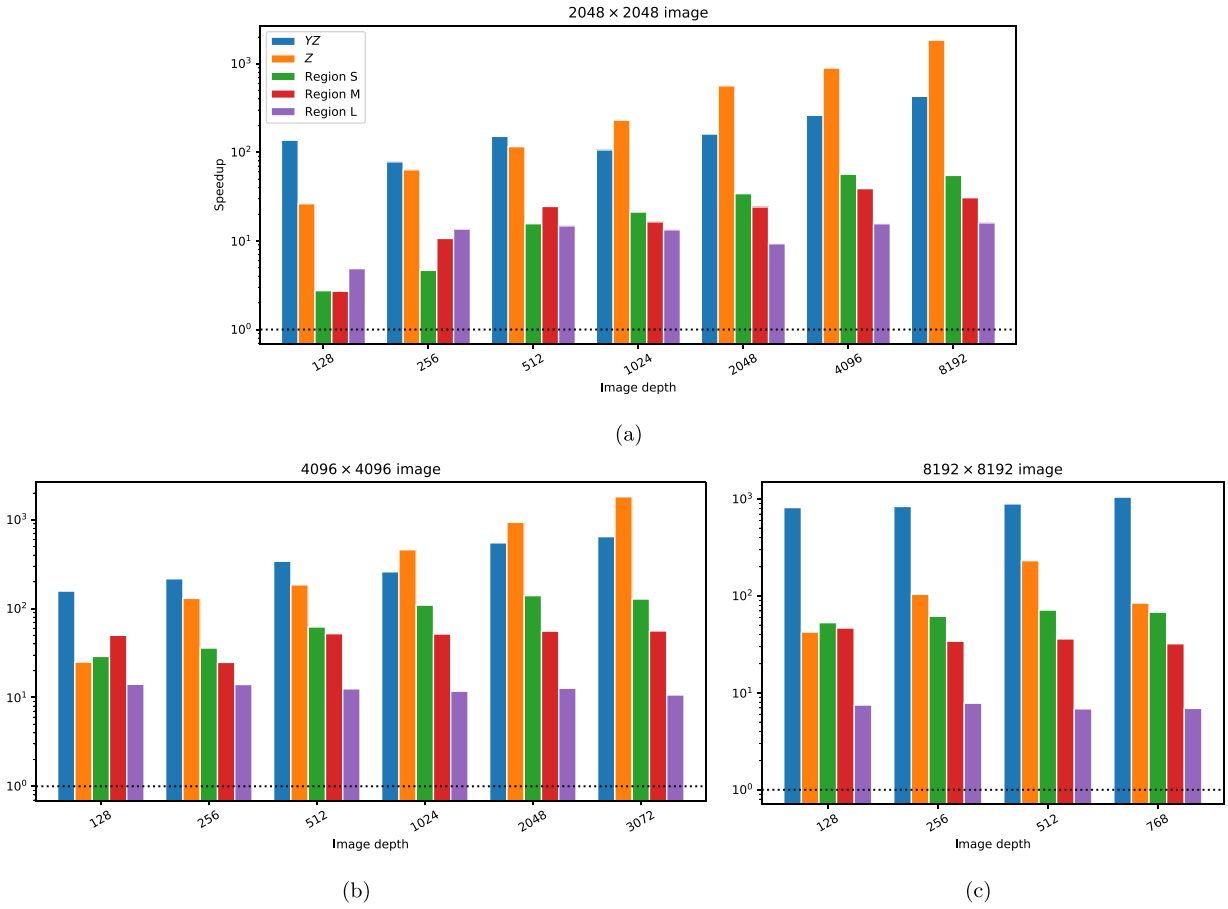

Fig. 5. Performance speedup versus image depth for a number of workloads described in Section 3 and a number of typical spatial dimensions, running on a cluster with a BeeGFS hard drive-based file system.

We also performed measurements on the full range of images on a cluster with a HDD-based BeeGFS file system (Heichler, 2014), as shown in Fig. 5, to show how the speedups scale at larger image sizes. There is more variance in the results for this test, which we believe is due to the effects of caching performed by BeeGFS and I/O operations performed concurrently by other users on the system.

We also performed a few basic benchmarks to demonstrate the performance difference between 2D slice reads along different axes from the unpermuted dataset, as described in Section 4.1.3. When we compared random *XY*, *XZ* and *YZ* slice reads from a 1 k × 1 k × 1 k cube, we observed average read times of 39 ms, 504 ms and 5425 ms (respectively) on SSD, and 26 ms, 235 ms and 9070 ms (respectively) on a BeeGFS HD-based file system. As expected, *YZ* read performance was particularly poor.

We then compared the load time of a full image channel and the duration of various downsampling tasks to the load time of a precalculated downsampled tile, to illustrate the utility of the mipmapped image datasets described in Section 4.1.4. In benchmarks performed on an 8 k × 8 k image, reading a full *XY* slice required on average 984 ms on SSD and 938 ms on BeeGFS, and downsampling a 1 k × 1 k, 2 k × 2 k or 4 k × 4 k region to a 256 × 256 tile required 95 ms, 182 ms and 388 ms (respectively). In contrast, reading a single 256 × 256 tile from disk required on average only 30 ms on SSD and 22 ms on BeeGFS. The advantage of reading pre-calculated down-sampled data (rather than reading full-resolution images and down-sampling on demand) is therefore significant.

6.2. Real-world performance in CARTA

To test the effect of our HDF5 schema integration on user experience in the CARTA viewer, we performed several real-world performance tests, comparing the speed of various tasks performed on a series of synthetic FITS images (as described in Section 6.1) and the same images converted to HDF5. These tests were performed with a CARTA server process running on 8 cores of an Intel Xeon E5-2697A v4 CPU, reading from the BeeGFS file system.

Fig. 6 shows time taken to load the image, measured from the moment that the request is sent to the server to the moment when the first image is delivered to the client. The load times show a large amount of variation, due to the effect of caching on the BeeGFS file system. Despite this variation, there are several trends in the load times that can be explained by the different approaches used to load FITS and HDF5 images.

The load time for a FITS file scales with the image area *A*, but remains largely independent of the number of channels *D*. This is to be expected, as the server must load the image slice for a single channel (an $O(A)$ I/O operation), and then calculate the histogram for the loaded data (an $O(A)$ CPU-intensive operation). Load time for HDF5 files scales with both *A* and *D*. This is because at present we preload histograms and other statistics for all channels from an HDF5 file when the image is initially loaded, requiring an $O(\sqrt{A} \times D)$ I/O operation (as described in Section 4.1.2, the number of histogram bins scales with $\sqrt{A}$), but no further histogram calculation. In order to reduce this dependency on the number

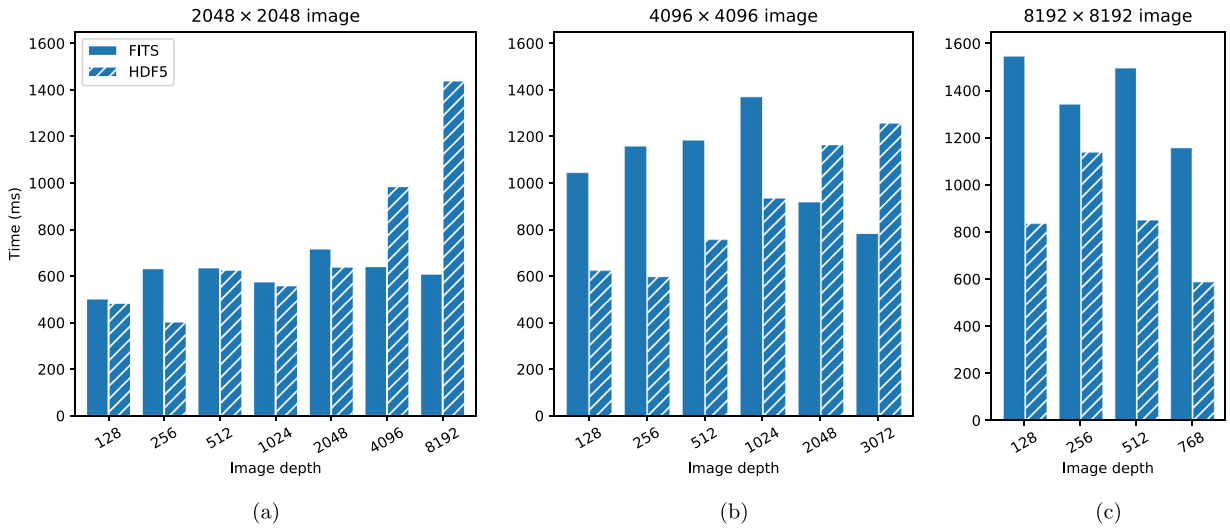

Fig. 6. Time taken to load an image in CARTA versus image depth, for three typical spatial dimensions. In order to display the image, the first channel is read from disk. In the case of HDF5 files, per-channel histograms described in Section 4.1.2 are also read from disk.

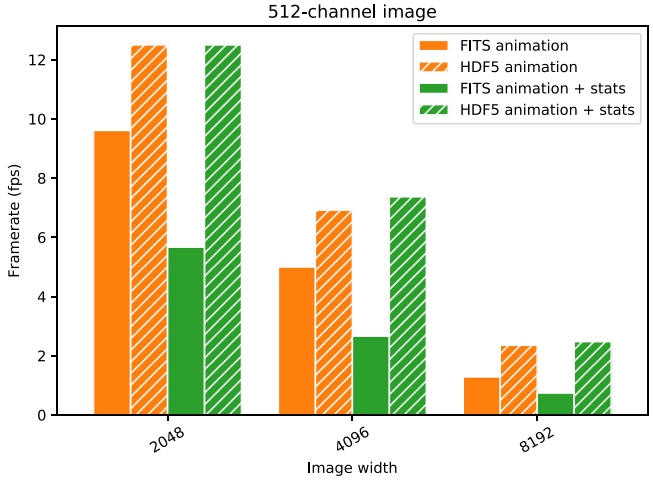

Fig. 7. Channel animation frame rate in CARTA versus image width for FITS and HDF5 files, with and without additional per-channel statistics calculations. Square images with equal heights and widths were used.

of channels, we could instead load histograms on demand, as the user switches channels. However, as shown in Fig. 6(c), when the image area becomes sufficiently large, the time taken to calculate the channel histogram for a FITS image can exceed the time taken to load all of the channel histograms from file. Furthermore, as the image area increases, this preload time becomes an increasingly small fraction of the time taken to load the image data, and thus the total loading time.

Fig. 7 shows the animation playback frame rate when consecutive channels are streamed to the client, with the default widget selection and with the addition of a statistics widget which requires full-channel statistics to be displayed for each channel. In all cases, the frame rate is higher when an HDF5 file is used, as a result of the preloaded channel histogram and (if the statistics widget is open) other channel statistics. There may be a negative impact on the animation frame rate should we load channel statistics on demand. However, the cost of loading statistics for a single channel is low when compared to the cost of calculating them, and it is possible to mitigate it by preloading several channels at once while an animation is playing.

Further improvements to HDF5 image load times and animation frame rates are expected once support for the mipmapped datasets described in Section 4.1.4 is added to CARTA, as it will drastically reduce the size of data that must be read from file when the user loads a new image or switches channels. For example, when an 8192 × 8192 image is viewed on a typically sized 2048 × 2048 client screen viewport, only 16 MB of data would need to be read from an HDF5 image's mipmapped dataset, compared to 256 MB of data for an identically sized FITS image.

7. 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. Tests of reading from a permuted dataset defined in the schema show significant benefits for commonly performed workloads, on HDD- and SSD-based file systems. Speedups in the order of 10^3 have been measured for some of the workloads tested. These performance improvements allow for better scaling, both in terms of the number of potential users that a remote image viewing server could support, and the size of the images served in a performant manner. Many of the speedup factors measured in Section 6 increase with channel count or spatial resolution, indicating that the schema is likely to be well-suited to SKA-scale image cubes. The schema is supported by the CARTA software package, with ongoing development adding support for more features defined by the schema. Subsequent developments will focus on features of the schema that will improve image cube loading times, and reduce the memory required to load large image cubes.

CRediT authorship contribution statement

A. Comrie: Conceptualization, Methodology, Software, Writing - original draft. **A. Pińska:** Software, Writing - review & editing,

Visualization, Formal analysis. **R. Simmonds:** Writing - review & editing. **A.R. Taylor:** Supervision, Writing - review & editing, Resources.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Alexov, A., Schellart, P., ter Veen, S., Van den Akker, M., Bahren, L., Grießmeier, J.-M., Hessels, J., Mol, J., Renting, G., Swinbank, J., et al., 2012. Astron. Data Anal. Softw. Syst. XXI 461, 283.

Anderson, K., Alexov, A., Baehren, L., Griessmeier, J.-M., Wise, M., Renting, A., 2010. PoS ISKAF2010, 062, doi:10.22323/1.112.0062. arXiv:1012.2266 [ASP Conf. Ser.442, 53(2011)].

Comrie, A., Wang, K.-S., Ford, P., Moraghan, A., Hsu, S.-C., Pińska, A., Chiang, C.-C., Jan, H., Simmonds, R., 2018. CARTA: The cube analysis and rendering tool for astronomy. doi:10.5281/zenodo.3377984.

Deane, R., 2016. MeerKAT Science: On the Pathway to the SKA. p. 17.

Folk, M., Heber, G., Koziol, Q., Pourmal, E., Robinson, D., 2011. Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases. ACM, pp. 36–47.

Hassan, A., Fluke, C.J., 2011. Publ. Astron. Soc. Aust. 28 (2), 150–170, doi:10.1071/AS10031. arXiv:1102.5123.

Heichler, J., 2014. An Introduction to BeeGFS. Technical Report.

Van der Hulst, J., Terlouw, J., Begeman, K., Zwitser, W., Roelfsema, P., 1992. Astronomical Data Analysis Software and Systems I, Vol. 25. p. 131.

Jarvis, M., Taylor, R., Agudo, I., Allison, J.R., Deane, R.P., Frank, B., Gupta, N., Heywood, I., Maddox, N., McAlpine, K., Santos, M., Scaife, A.M.M., Vaccari, M., Zwart, J.T.L., Adams, E., Bacon, D.J., Baker, A.J., Bassett, B.A., Best, P.N., Beswick, R., Blyth, S., Brown, M.L., Bruggen, M., Cluver, M., Colafrancesco, S., Cotter, G., Cress, C., Davé, R., Ferrari, C., Hardcastle, M.J., Hale, C.L., Harrison, I., Hatfield, P.W., Klockner, H.R., Kolwa, S., Malefahlo, E., Marubini, T., Mauch, T., Moodley, K., Morganti, R., Norris, R.P., Peters, J.A., Prand oni, I., Prescott, M., Oliver, S., Oozeer, N., Rottgering, H.J.A., Seymour, N., Simpson, C., Smirnov, O., Smith, D.J.B., 2016. MeerKAT Science: On the Pathway to the SKA. p. 6, arXiv:1709.01901.

Johnston, S., Bailes, M., Bartel, N., Baugh, C., Bietenholz, M., Blake, C., Braun, R., Brown, J., Chatterjee, S., Darling, J., et al., 2007. Publ. Astron. Soc. Aust. 24 (4), 174–188.

Jonas, J., et al., 2018. MeerKAT Science: On the Pathway to the SKA, Vol. 277. SISSA Medialab, p. 001.

Kiddle, C., Taylor, A.R., Cordes, J., Eymere, O., Kaspi, V., Pigat, D., Rosolowsky, E., Stairs, I., Willis, A.G., 2011. Proceedings of the 2011 ACM Workshop on Gateway Computing Environments. GCE '11, ACM, New York, NY, USA, pp. 65–72, doi:10.1145/2110486.2110496.

McMullin, J.P., Waters, B., Schiebel, D., Young, W., Golap, K., 2007. Astronomical Data Analysis Software and Systems XVI, Vol. 376. p. 127.

Price, D., Barsdell, B., Greenhill, L., 2015. Astron. Comput. 12, 212–220.

Robitaille, T.P., Tollerud, E.J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A.M., Kerzendorf, W.E., et al., 2013. Astron. Astrophys. 558, A33.

The HDF Group, 2019. Chunking in HDF5. https://portal.hdfgroup.org/display/HDF5/Chunking+in+HDF5. (Accessed April 24, 2020).

Van Waveren, J., 2006. Real-time texture streaming & decompression.

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

Williams, L., 1983. ACM SIGGRAPH Computer Graphics, vol. 17, (3), ACM, pp. 1–11.

Wootten, A., Thompson, A., 2009. Proc. IEEE 97 (8), 1463–1471.