OVID: A Python CARTA back-end

Phani Velicheti[1] and Jan-Willem Steeb[2]

[1]*University of Arizona. Tucson, Arizona*
[2]*National Radio Astronomy Observatory. Charlottesville, Virginia*
Email: phaniv@email.arizona.edu, jsteeb@nrao.edu

CARTA is the Cube Analysis and Rendering Tool for Astronomy, a visualization tool used for images and visibility data sets in the form of FITS, HDF5, MIRIAD and CASA files. In order to view very large (TB size) images created by large and newer telescopes in a performant manner, CARTA uses web browsers and various serialization and parallelization techniques in its client-server architecture. This project aims to replace CARTA's C++ back-end with a Python back-end with comparable performance, increases code readability, makes future development easier for scientific programming audiences and supports next generation CASA prototypes with newer file formats while supporting existing file formats. The source code can be found at OVID.

1. INTRODUCTION

CARTA is the Cube Analysis and Rendering Tool for Astronomy, a new image visualization and analysis tool designed for the ALMA, VLA, SKA pathfinders, and the ngVLA Comrie et al. 2021. CARTA's client-server architecture is defined by the interface control document (ICD) and has three components:

- carta-back-end,
- carta-front-end,
- carta-controller.

The carta-back-end does all the data processing and computation for images. It is intended to be run on large clusters in order to ensure performance and reduce load on the user via GPUs and enterprise grade storage solutions.

The carta-controller provides a dashboard to manage front-end and back-end sessions. The carta-front-end is responsible for receiving the data produced by the back-end and does minimal processing to render the data in an efficient manner to be viewed in modern web browsers as a GUI application.

Protocol buffers are language-neutral, enabling us to develop a python back-end without affecting the front-end. Carta-protobuf contains protocol buffer message definitions to be used in serializing structured data for the front-end and back-end interfaces.

The current carta back-end is entirely written in C++ and supports legacy formats such as FITS, HDF5, CASA and MIRIAD.

The back-end aims to achieve the following:

1. Replace the current C++ CARTA back-end with a pure python back-end using the next generation CASA prototype.
2. Support newer file formats (img.zarr)
3. Be comparably performant to the CASA C++ back-end.
4. Legacy format support for FITS, HDF5, CASA and MIRIAD.
5. Increase readability of code and make future development easier for scientific programming audiences.

While supporting existing file formats,the back-end should achieve comparable performance to the C++ back-end, by utilizing the next generation CASA prototype infrastructure which consists of a framework with support for a variety of compression, storage serialization, just-in-time compilation, file format, and cluster based cloud computing features.

2. CASA NEXT GENERATION INFRASTRUCTURE PROTOTYPE

Figure 1 is the Prototype Design Software framework for Next Generation CASA.This prototype uses the following libraries to process, store and analyse data:

- Numba is used for just-in-time compilation of python code . It translates python code into machine code through an LLVM compiler. This enables us to gain a significant performance boost while running computationally heavy parts of code such as for loops that cannot be vectorized.
- Xarray forms the basis for new file formats such as img.zarr due to its support for N-dimensional labelled datasets. The data is chunked on disk through its compatibility with Zarr.
- Zarr enables the storage of N dimensional NumPy `ndarrays` which can be chunked along any dimension and compressed through a Numcodecs supported codec. Dask's lazy execution functionality enables us to load metadata of the Xarray dataset for which it is used. These `ndarrays` can be stored locally or on cloud computing platforms in order to support larger than memory datasets.

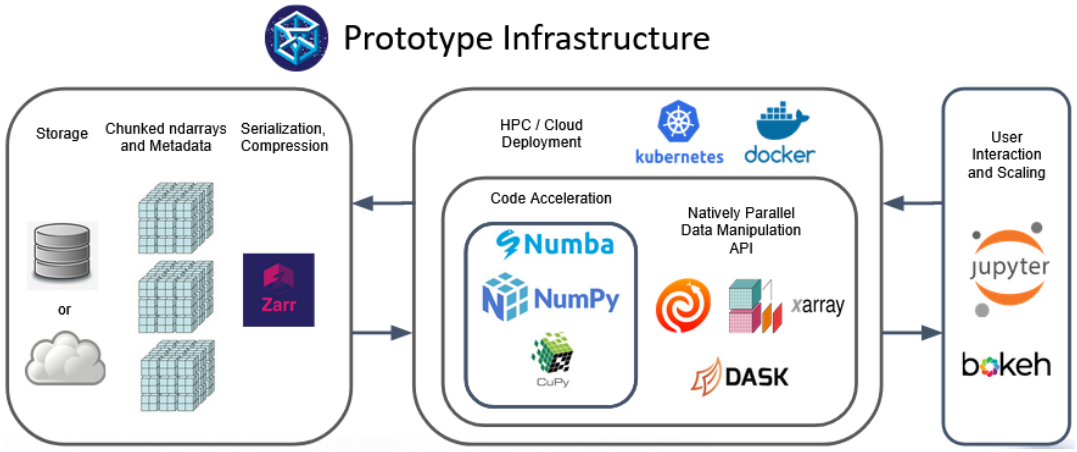

FIGURE 1. Prototype Design Software framework for Next Generation CASA Source:CARTA-Team 2021

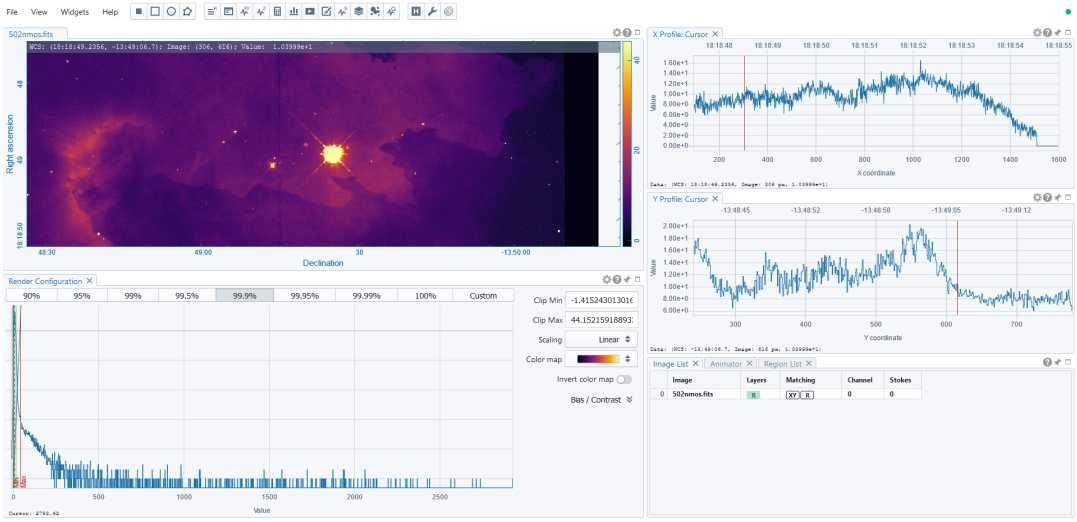

FIGURE 2. CARTA front-end GUI default view Comrie et al. 2021

- Dask enables parallel computing compatible with Zarr. Its specification to encode a Directed Acyclic Graph (DAG) consists of python data structures that are abstracted in the form of Dask collections. These DAG task graphs are then executed by a dynamic task scheduler in parallel.

3. CARTA FRONT-END GUI

As seen in figure 2, CARTA front-end default view has three main components:

- Spatial profile panels
- Region configuration and statistics panel
- Region view

The spectral profile panels give the X,Y and Z profiles of the location of the cursor on the image.

The statistics panel shows a per-channel histogram with bin count determined by the geometric mean of the image and user-defined boundary values.

The region view contains the render associated with the raster tile data streamed by the back-end and is governed by the "tiled web map" (Maso, Pomakis, and Julia 2010) convention associated with geographic information system (GIS) software.

The front-end can request specific tiles of an image to be delivered.

4. THE PYTHON BACK-END

The UML diagram figure 3 shows the structure of the python backend.It is made up of the following:

- Google protocol buffers are a language-neutral, platform-neutral, extensible mechanism for serializing structured data which allow us to communicate with the existing protobuf interface and keep the front-end. In order to communicate with the front-end, Tornado and Google protocol buffers are used.
- The server (handled by Server) is written in Tornado and provides a single threaded event-loop for asynchronous and non-blocking application

FIGURE 3. UML diagram for python back-end

code. It calls Session for each EventType message (ICD message) to handle server side configuration. The server class can be easily refactored to accommodate other python server libraries.

- FileInfo handles file headers, metadata and basic loading. It is written in Astropy and standard python libraries.
- Frames are associated with the opened image and index the image to provide raster tile data and statistics to the front-end.Some of the features handled by Frame are viewing profiles based on the position of the cursor,passing on information about raster tiles to other classes and handling image view and stokes changes.

5. RASTER TILING

CARTA follows the "tiled web map" (Maso, Pomakis, and Julia 2010) convention to render images. The user defined settings in the render configuration widget determines how a raster image is rendered.As shown in figure 4, CARTA splits the image into downsampled 256 x 256 wide squares and renders the image when the back-end sends specific tile data. Contour rendering works in a similar manner. The CARTA Interface Control Document has various ICD messages that seek to optimise raster tile data delivery. They give us the following features:

- If tile data for a certain tile is no longer needed as a consequence of user zoom/pan and has not been sent by the front-end, it is removed from the list of required tiles.
- CARTA supports various levels and algorithms dealing with the compression of raster tile data.
- The raster tiles are reused while zooming and panning the image.

This approach is efficient as it reuses tile data which affects performance while viewing large images.

6. DEVELOPMENT PROCESS

Test driven development is done with the CARTA Interface Control Document.Feature testing is done to a standalone front-end application via a browser.

7. PROGRESS

The directory and file browsing interface,file metadata and header loading, region statistics and exception handling have been done.

The future objectives for this project are:

- Full interoperability with C++ back-end (tested by CARTA ICD test).
- img.zarr (next generation CASA prototype format) integration.

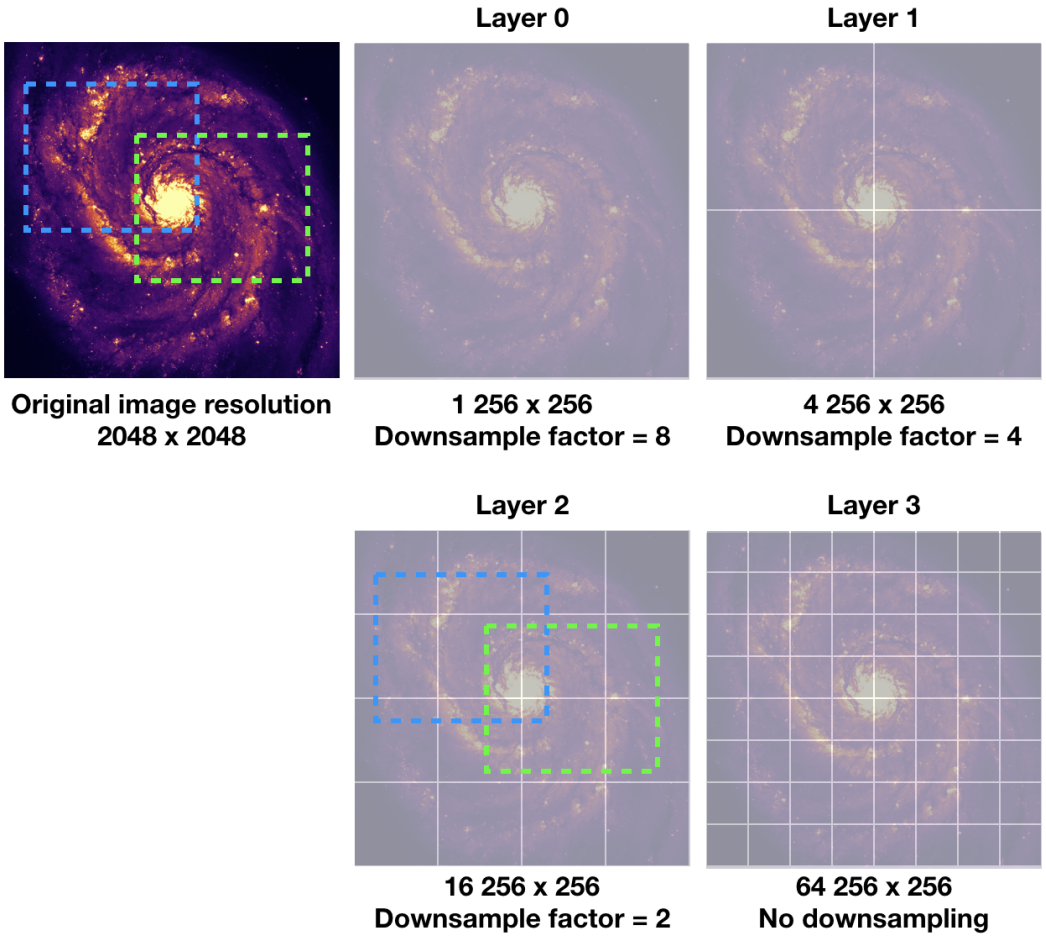

FIGURE 4. Raster Tile view CASA-Team 2021

- Integration with next generation CASA prototype image analysis.
- Parallelization using Dask.

8. CONCLUSION

We propose that moving to a python back-end for CARTA can enable better scalability, be comparably performant, reduce custom code dependence, bring in community development and support legacy formats while transitioning towards newer file formats.With further development, this project can make that transition possible.

ACKNOWLEDGEMENTS

This research was undertaken as part of the Summer Student Research Assistantship Program at the National Radio Astronomy Observatory and supervised by Jan-Willem Steeb.

REFERENCES

[1] CARTA-Team. *CASA Next Generation Infrastructure 0.1b documentation*. NRAO, 2021. URL: `https://cngi-prototype.readthedocs.io/en/latest/`.

[2] CASA-Team. *Image cube visualization CARTA 2.0 documentation*. NRAO, 2021. URL: `https://carta.readthedocs.io/en/latest/_static/carta_fn_tiledRendering.png` (visited on 08/31/2021).

[3] Angus Comrie et al. *CARTA: The Cube Analysis and Rendering Tool for Astronomy*. Version 2.0.0. June 2021. DOI: `10.5281/zenodo.4905459`. URL: `https://doi.org/10.5281/zenodo.4905459`.

[4] Joan Maso, Keith Pomakis, and Nuria Julia. *Category: OpenGIS Implementation Standard OpenGIS Web Map Tile Service Implementation Standard*. 2010. URL: `http://portal.opengeospatial.org/files/?artifact_id=35326` (visited on 09/01/2021).