CASA Next Generation Infrastructure Development Plan

PREPARED BY	ORGANIZATION	DATE
	NRAO	

Change Record

VERSION	DATE	REASON
0.1		Draft Release

Table of Contents

Links

- Github: https://github.com/casangi/cngi_prototype
- API: https://cngi-prototype.readthedocs.io/en/latest/
- Install: https://pypi.org/project/cngi-prototype/
- Examples: https://github.com/casangi/examples

Slides

- CUC and Astropy Briefs: https://safe.nrao.edu/wiki/pub/Software/CASA/CASAUsersCommittee2019/2019%20-%20CNGI.pdf
- Internal Review: https://docs.google.com/presentation/d/1SCGjE-GtTcSFDoHCihCIMDERROaP00lasZXUlZ1tboA/edit?usp=sharing

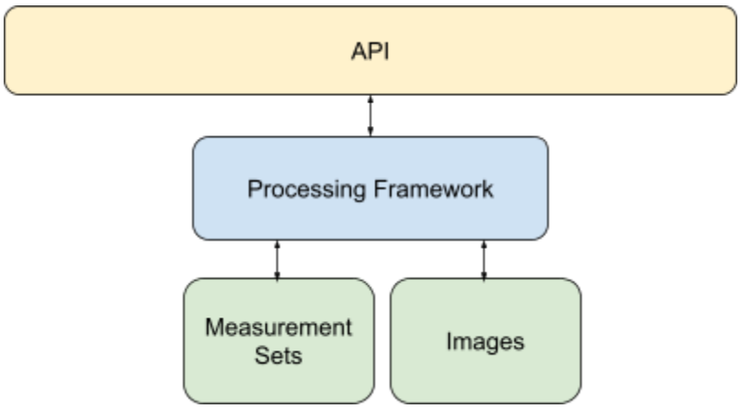
Overview

CASA software development faces several competing challenges that are expected to only worsen with time. As both a scientific research and development platform as well as a necessary component in multi-million dollar telescope operations, CASA must overcome the contradictory goals of being a flexible experimentation platform with mature, documented, tested, and stable performance. Furthermore, it must accomplish this with a relatively modest development staff that is diverse, remote, and matrixed across many competing projects.

Historically, much of CASA traces its roots to algorithms developed and refined over many decades and implemented in a custom code base utilized and maintained by a global, yet small developer base. In years past, it was necessary to develop a custom infrastructure to support the unique needs of radio astronomy science. Yet as data sizes, performance demands, and conflicting needs of multiple telescopes continue to rise, the overhead of maintaining this infrastructure grows exponentially.

Scope

The new CNGI library will focus solely on MeasurementSet and Image access rather than general purpose table manipulation. It is anticipated that foundational packages (casa, scimath, tables, measures, lattices, meas, and python) will be largely (if not entirely) replaced by an off-the-shelf framework. Remaining packages with more specialized functionality (fits, derivedmscal, msfits, coordinates) will be substantially if not entirely replaced with alternative community supplied functionality such as astropy. Remaining custom functionality will be limited to wrapping CASA specific actions under MeasurementSet and Image functions. The resulting new casa infrastructure interface will have fewer packages and far less custom code. The specific final API definition will be determined during prototyping. **The new API is not required to be backward compatible with the old.**



The CNGI package will give developers and researchers a revolutionary new capability to access and manipulate radio astronomy MeasurementSet and Image data. The API will provide a simple representation of the data and methods to apply transformations and mathematics that are inherently parallel, scalable and divorced from hardware memory limitations. File formats will conform to industry standards and be easily transportable to other packages. An included processing framework will execute CNGI functions on a diverse set of hardware platforms making optimal use of specified resources.

The CNGI package will be the cornerstone for future CASA evolution to meet the demands of next generation instruments, heuristic development, and data visualization.

Project Plan and Schedule

CNGI will be developed in phases, beginning with six months of study and review followed by another six months of prototyping and experimentation culminating in an initial prototype release.

Software Trade Study - 3 months

The first trade study will explore modern developments in the field of software engineering with respect to high-performance scalable computing and data analysis. Many methods of "parallelizing" the execution of code as well as the reading and writing of data now exist with a wide range of complexity, control, and performance. In most cases, the choice of programming language itself plays a critical role in what methods of parallelization are available and their level of complexity.

Deliverable: Software Trade Study Report containing downselection of technology choices for further study.

This step is now complete, the report can be found here: Software Trade Study Report v0.2.pdf

Framework Trade Study - 3 months

After the completion of the first trade study, the desired programming language(s) and method of parallelization is defined. The second trade study focuses on what off-the-shelf frameworks are available to satisfy the selected paradigm. Limited experimentation is conducted in each candidate option using a more detailed understanding of what specific functionality from the old casacore API is necessary to replicate and/or re-implement in the new framework. The selected framework should satisfy the motivations stated herein but also be versatile enough to handle the nuances and any unforeseen complexities that arise in actual implementation.

Deliverable: Framework Trade Study Report containing final selection of technology to pursue in an initial prototyping effort.

This step is now complete, the report can be found here: Framework Trade Study Report.pdf

Prototype Implementation - 6 months

Using the selected technology from the trade studies, a prototype next generation infrastructure will be implemented. Given that the intention is to move to a largely off-the-shelf software stack, the amount of custom code written should be drastically lower than comparable CASA development projects of the past. The prototype may not be feature complete, however it must adequately cover major functional areas so as to allow for useful scientific evaluation and utility. The prototype will also include necessary conversion routines to migrate old MeasurementSet and Image data formats to the new implementation as well as extensive documentation of how the new library is intended to work. The prototype is intended to serve as the initial implementation release of the new infrastructure.

Deliverable: Initial 0.1 release of CNGI infrastructure with documented API

This step is currently underway, progress is captured in github, readthedocs, and the remainder of this document.

During the prototyping effort two reviews will be held. An **internal CASA review** of prototype progress and API functionality will be conducted to ascertain the scope and impact to a next generation CASA (ngCASA) effort to re-write the science algorithms of CASA. The results of this internal study will feed a **Conceptual Design Review** with various NRAO technical and scientific staff to examine both CNGI and the broader ngCASA impacts.

Full Implementation - 6 months

Complete the full implementation of remaining development items to make CNGI feature complete for the initial major release. Ensure API is well documented and complete. Develop a test plan against documented API functionality and install automated test execution against that plan.

Deliverable: CNGI Release 1.0

Resources and Cost

The CNGI initiative will require a dedicated team of four individuals. This team must be primarily focused on this project alone with minimal matrixed duties to other projects. The team will be staffed by currently funded and existing positions and require no additional FTE's. Two positions will be filled by current vacancies open on the CASA development team while the other two will be filled by reassignment of existing development team members. The final team makeup will be as follows:

- **Technical Lead** - responsible for more detailed planning and tracking beyond this document, to be filled with current vacancy
- **Casacore SW Engineer**
- **HPC SW Engineer**
- **Infrastructure Engineer**

It is expected that the selected framework for the re-implementation of casacore will be a free and open source off-the-shelf software product. As such, there should be no licensing or equipment costs. Furthermore, the software stack should execute on existing NRAO hardware systems and not require the purchase of new or additional processing capacity.

Total cost of the initial CNGI development is estimated to be 4 FTE x 1.5 years = **6 labor years**.

Requirements

As this project is attempting to develop new software infrastructure that may be in future use for quite some time, it is tempting to demand clear, specific requirements on what it shall do. However, the functional science requirements of CASA are not currently defined in a single specification, nor are the derived infrastructure requirements or performance requirements of CASA for each instrument. Furthermore, the way in which CASA has been specified and developed over the years is steeped in implementation specifics, making it difficult to separate what is actually needed for science in absolute terms from what is needed because of the way in which it was already built.

It is beyond the scope of the Next Generation Infrastructure project to reverse engineer a complete set of functional and performance requirements for the current generation of CASA/casacore or future iterations of CASA needed by next generation instruments. Rather, this project is studying *implementation* choices that exist independent of requirements. This will present unavoidable challenges with stakeholders and users as the legacy implementation they are used to will necessarily be changing.

In lieu of a formal requirements specification, a top-down functional decomposition from the current task-level definition of CASA and the CASAdocs data reduction documentation can be used to derive infrastructure API functions and their implicit requirements. Additionally, a bottom-up synthesis of API functions and their implicit requirements can be generated from a survey and collection of basic mathematics and data manipulation needed by radio interferometry. These two methods should meet in the middle to provide a reasonable and robust definition of CNGI functionality.

Ground Rules

The following high-level ground rules also form a starting point of this project:

- Cover the same functionality provided by casacore
- Callable from a Python 3.x environment
- Support older legacy data
- Industry standard usage with expected multi-decade longevity
- Meet the needs of ngVLA

The following points are worthwhile to note as not being required:

- Does NOT need to be backwards compatible with casacore – but that would certainly be nice
- Does NOT need to be C++
- Does NOT need to be a single solution for both MS and Image data processing

Requirements Decomposition

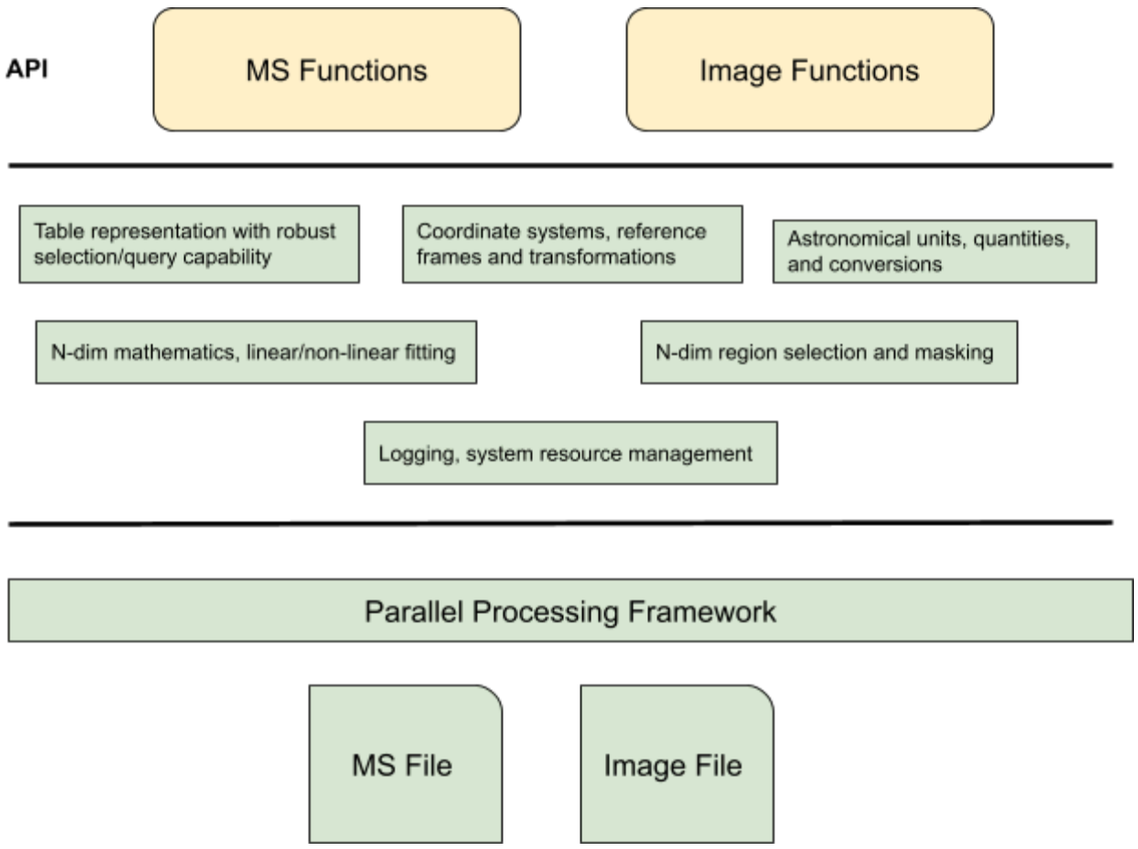
The aforementioned top-down requirements decomposition from CASAdocs documentation can be found in a standalone spreadsheet here:
https://docs.google.com/spreadsheets/d/1ihhoOFa-aRpCUnHe1eh8oL6fSTwtn6BFyz52yB5ko7A/edit?usp=sharing

Requirements Synthesis

The aforementioned bottom-up requirements synthesis from data survey/collection can be found in a standalone spreadsheet here (second sheet):
https://docs.google.com/spreadsheets/d/1ihhoOFa-aRpCUnHe1eh8oL6fSTwtn6BFyz52yB5ko7A/edit?usp=sharing

Functional Hierarchy

The following functional hierarchy serves as a starting point for future design and implementation decisions. The boxes in green ▭ are anticipated to be entirely off-the-shelf entities from existing third party software packages, but may require a conversion from current custom implementations.



Trade Study 1

Refer to the stand alone report here: Software Trade Study Report

Trade Study 2

Refer to the stand alone report here: Framework Trade Study Report

Design and Prototyping

Per the outcome of the second trade study report (here), CNGI will use the Dask processing infrastructure and Dask compatible data structures. The API will follow a **functional paradigm** in which the MS and Data routines take in a dask-compatible data object and return a new dask-compatible data object as output.

Initial Design Ideas

The second trade study report suggested an initial design based on the Dask Dataframe model for in-memory data representation and the Apache Parquet format for on-disk storage. However, a critical design limitation in which Dask cannot utilize the Pandas multi-index concept, combined with the design of the Apache Parquet storage format, creates edge cases with exceptionally poor performance on very large data sizes due to constant data shuffling operations. The Dask documentation warns of these scenarios (as discussed here) and the dangers of data shuffling, but does not highlight just how poor or how unavoidable these scenarios would be. Operations on MS data will necessarily need to support different columns as axes of parallelization (i.e. map operations across time, antenna/baseline, channel, polarization etc). Since only one column may be used as an index (no multi-index support), only that column will avoid data shuffles.

This limitation is compounded by the fact that Apache Parquet has limited support for indexing multiple columns within the file format itself. Otherwise one could build multi-indexing into the storage structure, and then simply open it multiple times with multiple single-column indices set. However, Parquet only allows for sorting of column values and quick retrieval of start/end values per row-group. This means that an index can only be created for columns that can be sorted such that the start of each subsequent row group is always greater than or equal to the end of the previous row-group value. This works for a single index column, and indeed might work for a couple index columns if they are suitably related, but it is not possible to sort time, antenna/baseline, channel and polarization simultaneously in this way.

The only other avenue towards using Dask Dataframes and Apache Parquet storage as originally intended, would be to partition the data by all desired axes of parallelization. This essentially spreads the indexing into the partitioning structure of the parquet directory tree. However, for even modestly sized datasets, this can result in many thousands of partitions on disk, and experimentation quickly reveals that the overhead involved in reading and maintaining such a large number of partitions is worse than the data shuffling.

Consequently, CNGI has opted against using Dask Dataframes as the in-memory data model and Apache Parquet as the on-disk storage.

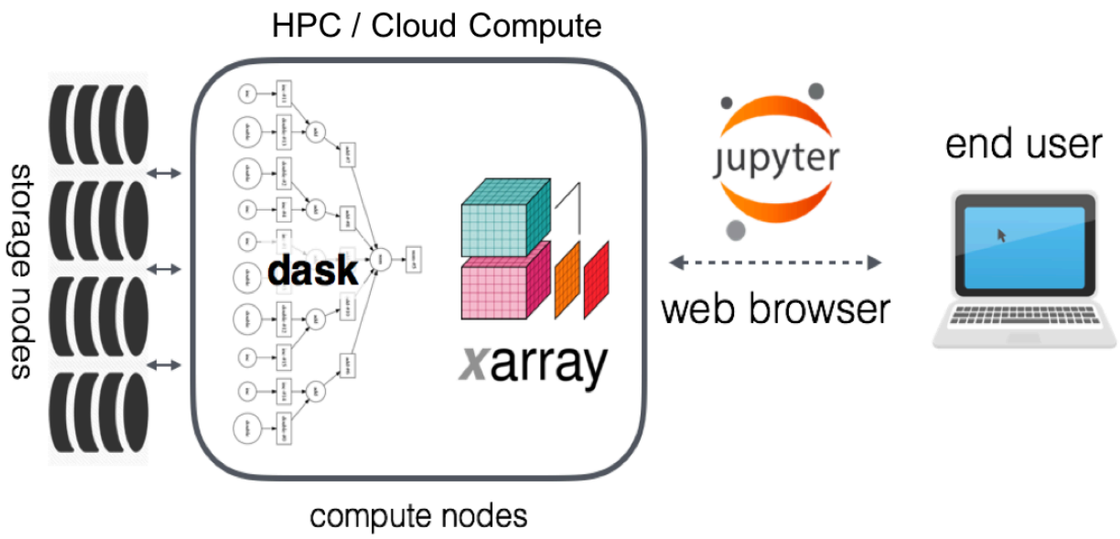
Revised Design Decisions

CNGI will use the **xarray Dataset** structure as the in-memory data model and the **Zarr** format for on-disk storage. Xarray structures are built from Dask and thus completely in keeping with decisions to this point. The xarray Dataset is very well suited for image data and the intuitive choice for that type of structure. The MS data structure is much more complex and requires much more adaptation during the conversion process. Consequently the learning curve will be a bit steeper, however the benefits are substantial and worth the investment. The subsequent sections will detail these decisions.

http://xarray.pydata.org/en/stable/data-structures.html

https://zarr.readthedocs.io/en/stable/tutorial.html#persistent-arrays
https://examples.dask.org/xarray.html#Custom-workflows-and-automatic-parallelization

Combining these design decisions with the demonstrated power and utility of Jupyter notebooks (via Google colab) first utilized during the trade studies, we arrive at the following high level architecture:



This diagram is from the Pangeo project in the geosciences, which has pioneered this same design on similar data sizes (https://pangeo.io/architecture.html)

Prototype Detailed Design

As CNGI prototype development proceeds, design and architecture details will be decided and filled in. The desire is to keep these details in a location and format that meets the following needs:

1. Publicly accessible by users and contributors to code base, to better understand how things work
2. Coupled to the code implementation, to prevent drift and staleness
3. Coupled to usage examples, guides, and user level documentation, to better illustrate the impacts of design decisions on end users

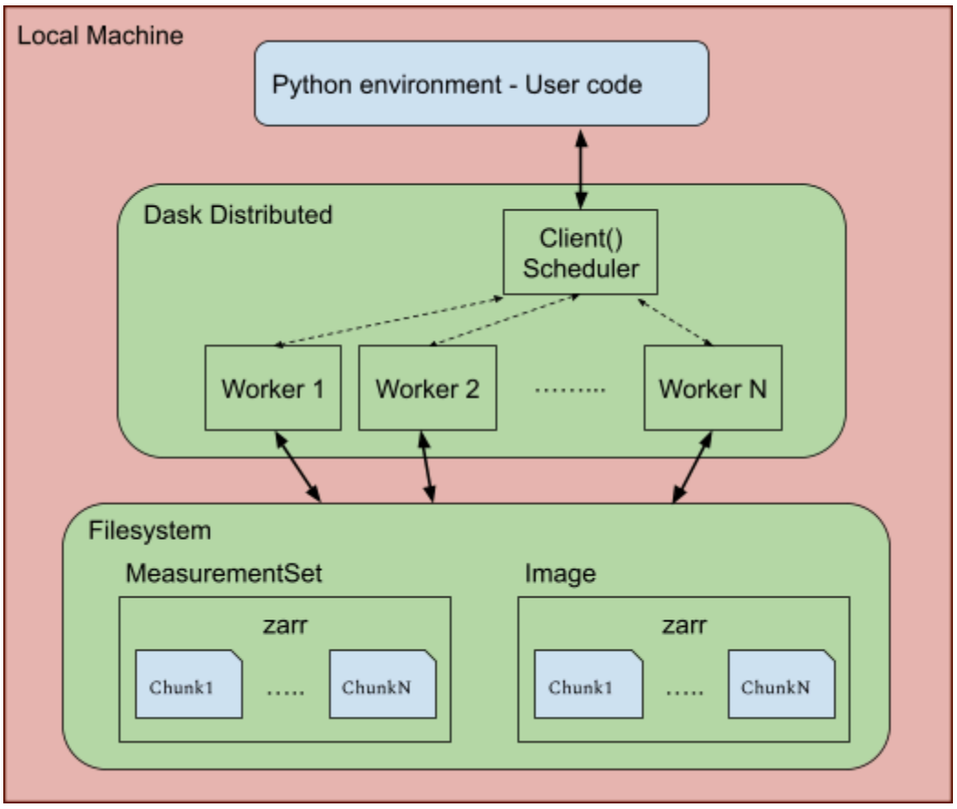
To meet these needs, CNGI will utilize readthedocs.org coupled to a public github repository. Further design documentation will not take place in this document, but rather in the github docs folder and displayed on the readthedocs website. Here we will link to several important locations in the prototype documentation specifically dealing with detailed design.

- **Architecture -** https://cngi-prototype.readthedocs.io/en/latest/development.html#Architecture
- **Visibility Data Processing -** Note that due to significant differences in the representation of data after conversion, CNGI will refer to MeasurementSet data as Visibility data - https://cngi-prototype.readthedocs.io/en/latest/visibilities.html
- **Image Data Processing** - https://cngi-prototype.readthedocs.io/en/latest/images.html
- **Development Process -** https://cngi-prototype.readthedocs.io/en/latest/development.html

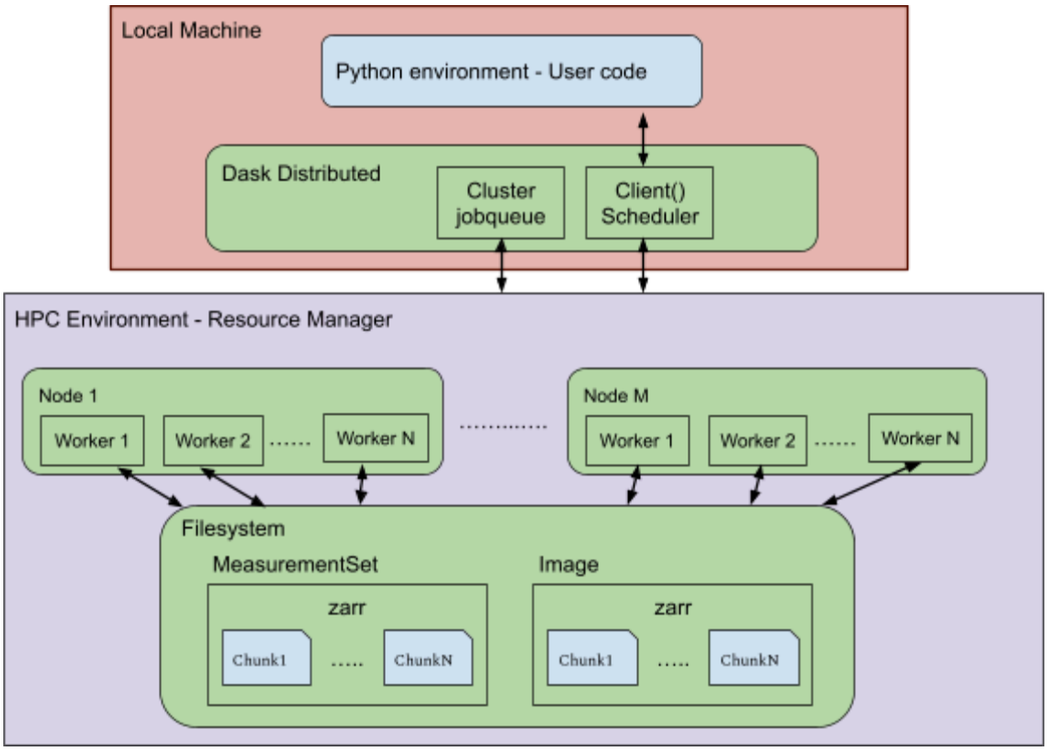
Processing

TBD

Run CASA on a single machine using Dask LocalCluster mode as described here:
https://docs.dask.org/en/latest/setup/single-distributed.html



Run CASA on a larger cluster that is managed by some resource scheduler (i.e. Torque) using Dask as described here: https://docs.dask.org/en/latest/setup/hpc.html



Alternatively, dask-gateway provides a secure, multi-tenant server for managing Dask clusters so we should aim to get that set up in-house.