Migrating Complex Workflows to the Exascale: Challenges for Radio Astronomy

Pascal Jahan Elahi [1], **Matthew Austin** [2], **Eric Bastholm** [2], **Paulus Lahur** [2], **Wasim Raja** [2], **Max Voroknov** [2], **and Matthew Whiting** [2]

[1]Pawsey Supercomputing Research Centre, Kensington, WA, Australia
[2]Space and Astronomy, CSIRO

***Abstract*—Real-time processing of radio astronomy data presents a unique challenge for HPC centers. The science data processing contains memory-bound codes, CPU-bound codes, portions of the pipeline that consist of large number of embarrassingly parallel jobs, combined with large number of moderate- to large-scale MPI jobs, and IO that consists of both parallel IO writing large files to small jobs writing a large number of small files, all combined in a workflow with a complex job dependency graph and real-world time constraints from radio telescope observations. We present the migration of the Australian Square Kilometre Array Pathfinder Telescope's science processing pipeline from one of Pawsey's older Cray XC system to our HPE-Cray EX system, Setonix. We also discuss the migration from bare-metal deployment of the complex software stack to a containerized, more modular deployment of the workflow. We detail the challenges faced and how the migration unearthed issues in the original deployment of the EX system. The lessons learned in the migration of such a complex software stack and workflow is valuable for other centers.**

I. Introduction

High Performance Supercomputing (HPC) centers have historically dealt with the challenge of efficiently scaling monolithic workflows, e.g. a single simulation using as much of the system as possible, and running as many of them as possible. Now HPC centers are asked to run more diverse workflows, some of which consist of many different subtasks with a complex inter-dependency.

The Pawsey Supercomputing Research Centre has been at the forefront of non-traditional HPC workflows due to our relation with radio astronomy telescopes. Pawsey provides critical infrastructure and assistance to the operations of the Australian Square Kilometre Array Pathfinder (ASKAP) Telescope [1].

Science data processing for radio astronomy telescope presents an unique challenge:

- The software stack is diverse: memory-bound codes; CPU-bound codes; serial codes used in an embarrassingly parallel fashion; and codes that use the Message Passing Interface (MPI, [2]) library to run on many nodes.
- IO patterns that range from parallel IO writing large TB-scale files to writing larger number of small files.
- A workflow where files are written as intermediate data products for a number of other components, some with inefficient access patterns.
- Complex job dependency graph with thousands of jobs, able to stress standard HPC job schedulers
- Real-world time constraints from radio telescope observations.

We present the migration of the ASKAP Science Data Processing (SDP) pipeline to Exascale systems like Pawsey's HPE-Cray EX system Setonix, discuss challenges faced and lessons learned. The first section describes outlines the ASKAP telescope's overall workflow, the science processing and the migration of this processing to an EX system. We then discuss the challenges we faced migrating and scaling the workflow and the solutions solutions. We end with a summary of lessons learned followed by what the future may hold.

*correspondence: pascal.elahi@pawsey.org.au

II. Science Data Processing Workflow of Radio Astronomy Data

A. ASKAP Telescope

The ASKAP telescope consists of 36 dish antennas working together as one giant interferometer. These dishes are 12 meters in diameter and provide a wide field of view of the sky, roughly 30x larger than conventional radio telescope receiver. The dishes are distributed in a radio-quiet zone, with the longest distance between two antennas being 6 km. The radio-quiet zone is located in "Inyarrimanha Ilgari Bundara" or Murchison Radio-astronomy Observatory (MRO), WA. The traditional owners of the land are the Wajarri Yamaji people.

Interferometers require significant computational resources to be operational and ASKAP is no exception. For ASKAP, these resources consist of an onsite computational infrastructure composed of a large number of FPGA's, which does the initial processing of the radio signals by correlating them, and Pawsey supercomputing infrastructure which ingests this partially processed data for further processing to produce science-ready data products used by the radio astronomy community. An outline of the telescopes operational pipeline is presented in Fig. 1 along with a sample radio astronomy image.

The entire process of ingesting raw telescope data and producing science results is managed by several tools:

a) `askap-cpmanager:`

Communicates with TOS (Telescope Operating System) to start observation raw data ingest. Coordinates/orchestrates CP services / applications - consumes events and produces new events in response.

b) `askap-tosconnector:`

Listens for scheduling block state change events in the TOS ICE environment, and re-emits these events using `clink` (Python library and CLI tool which wrapes a few tools: `cloudevent`, `amqp`, `rabbitmq`.). Generates JSON metadata for scheduling blocks from TOS ICE interface.

c) `askap-datamanager:`

Manages two copy queues (high and standard priority) to move scheduling block data from ingest-related filesystem to processing-related filesystems when enough space at the destination is available. The high priority queue is used for calibration and time sensitive observations, and requires less of a free-space buffer to be available.

d) `askap-processingmanager:`

Launches initial calibration processing. Manages calibration observations which have been processed and the queue of science observations which require processing. Launches science processing (ASKAP SDP pipeline) when the required matching calibration observations are available.

e) `askap-arwen:`

Generates diagnostic plots from the unprocessed ingested data, once it has been copied to the processing related filesystems and archives the plots. Plots are used to ensure telescope is operating as expected, and to look for signs of RFI/ducting/receiver issues.

In this paper we focus on ASKAP SDP, which is the primary portion of the HPC workload.

B. ASKAP SDP

The ASKAP SDP pipeline was previously deployed on a dedicated Cray XC system, Galaxy. The roughly 0.7 PFLOP provided by Galaxy meant that the ASKAP Telescope could not run in a real-time processing mode, the telescope always being able to produce data at faster rates than Galaxy could analyze. It did, however, provide an almost fully isolated system (in both compute and parallel filesystems) from all other users. This isolation meant identifying and debugging issues, particularly those related to filesystem performance was significantly simpler than it would be with shared resources. Nevertheless, the migration of the SDP to our new HPE Cray EX system, Setonix, was necessary to move towards real-time processing, even if it might introduce complexity in an already complex pipeline. The time-scale in which this migration needed to happen was compressed as Galaxy was nearing it's end-of-life.

The SDP software stack is dominated by `askapsoft` [3], a `C++` package. It also makes extensive use of Python packages. The workflow itself is complex: the processing path taken depends on the science goal and the input data quality. Some

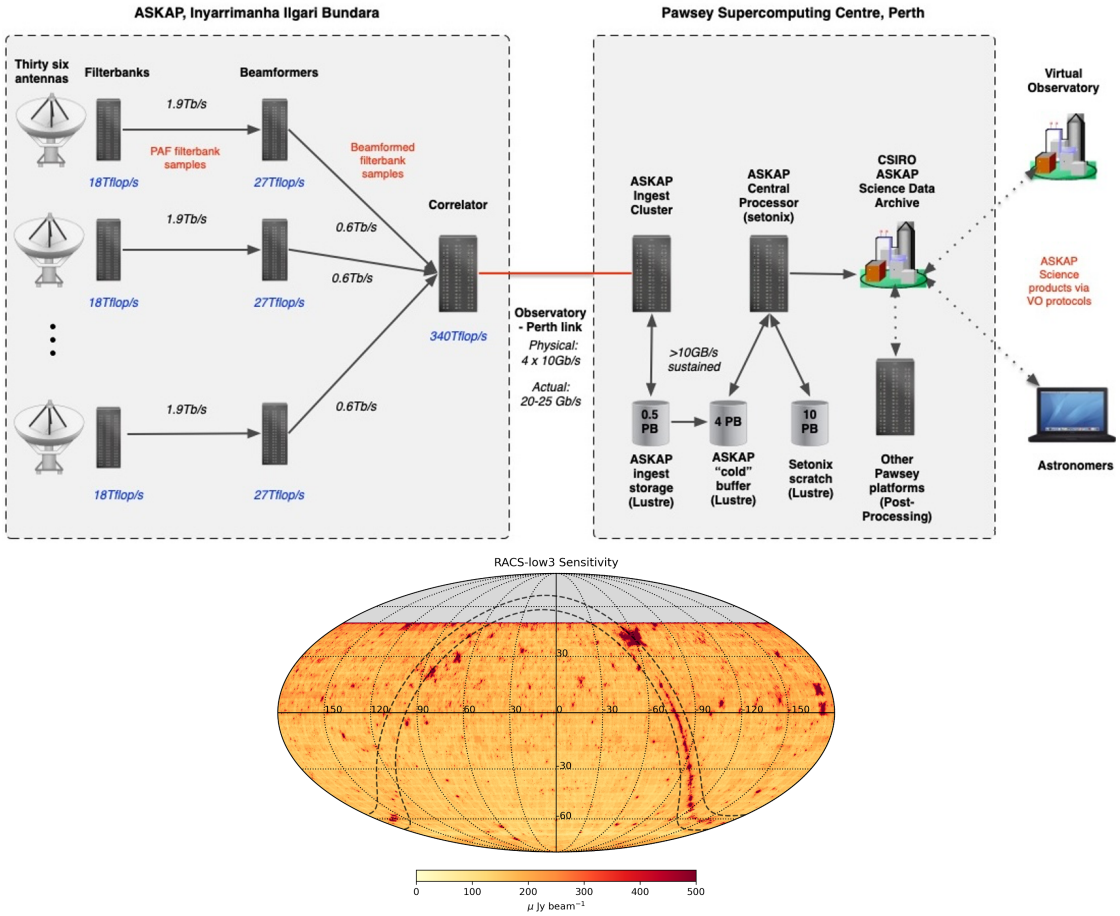

Fig. 1. (Top) Outline of ASKAP telescope operational pipeline, showing ingest from rado telescopes to correlator, transmission to Pawsey infrastructure and processing on said infrastructure. (Bottom) Example of radio emission of the sky with emission from our Galaxy highlighted, from the Rapid ASKAP Continuum Survey.

processing will run several 100 node MPI jobs, each of which will read and write TB of data. These jobs also can have large computational and memory imbalances between different MPI processes, meaning that these jobs are most efficiently run as heterogeneous jobs by the HPC job scheduler (like , [4]). Other processing will consist of large number of small shared-memory jobs. The number of these jobs can quite quickly stress `slurm`.

An example of the imaging pipeline is presented in Fig 2. This particular processing will generate several groups of tasks, one for each beam, 36 of which are produced by the ASKAP telescope. The memory footprint and computational time to solution per beam is not uniform as the data quality can vary between beams. Additionally, the processing involves a large number of IO operations. All tasks require reading of reference data, and often separate tasks are stitched together not by making use of memory but with files. There is a final larger processing job combining the data from all the beams, followed by analysis to produce diagnostic information and a radio source catalog.

This imaging stage is also not the only processing that can occur and is also by no means the most complex. However, this example highlights the base-level complexity of processing radio astronomy data.

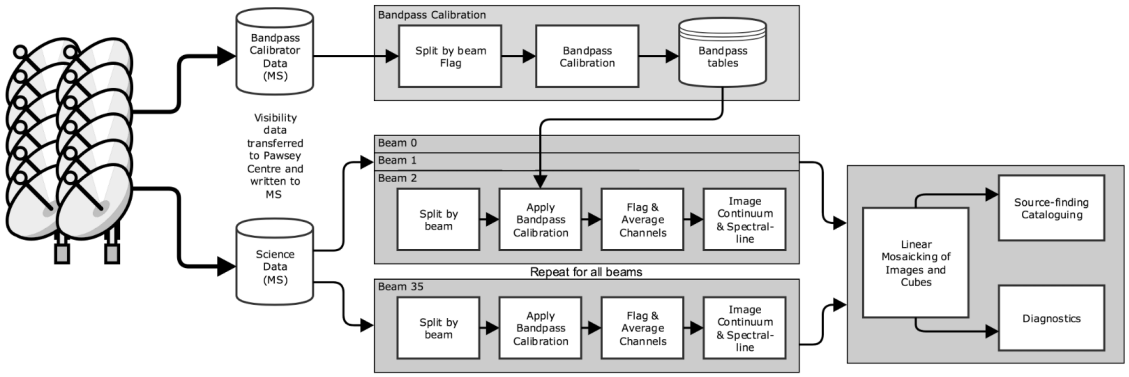

Fig. 2. ASKAP imaging pipeline. Of note is the number of `slurm` processes and the job dependency structure shown in this figure as implied by the number of `slurm` jobs, here highlighted as rectangular boxes. This process is one of the simpler ones in terms of variety of tasks and complexity of job dependency.

C. Migration Process

The team at the Pawsey Supercomputing Research Centre and the ASKAP Operations team took an approach similar to AGILE for the migration. With the number, complexity and inter-dependency of components, migration of small modules of the pipeline was not possible. However, through weekly meetings, sprints, early access to the CPU architecture used on Setonix, access to a HPE-Cray EX Test and Development System (TDS), and constant communication between the two teams, we successfully migrated the pipeline.

The first stage not only involved rebuilding codes with newer compilers and libraries. We implemented algorithmic changes to be able to better utilize nodes with more cores (e.g. Nyquist gridding + other memory reductions), not just running the same code. We also moved from bare-metal builds to a more reproducible and modular deployment using containers. Moreover, the need to use a shared filesystem with limits on the number of files per project meant prior approach could encounter these file limits and containers offered a simple solution.

III. Challenges & Solutions

A variety of challenges were faced during the migration processes, some of which were significant blockers. The constant testing on the TDS allowed some issues to be identified and addressed early but some only occurred upon partial migration to Setonix, when jobs could be run at production scale.

A the major issues encountered were:

- Issues with MPI.
- Issues with developing and running containers.
- Issues with the `Lustre` file system.
- Issues with job orchestration using `slurm`.

A. MPI

A number of MPI bugs that were missed in standard acceptance testing were found during migration due to the more complex communication patterns used in `askapsoft`. A key blocker was discovered on Phase-1 Setonix running an older `libfabric` library, which underpins the MPI on HPE-Cray EX systems. Codes that had large time between associated MPI send and receives would just hang.

Only through development of related unit tests and communication with HPE-Cray was a workaround identified. The tests is now part of our acceptance and regression tests and is vital in checking for regressions of this bug.

B. Containers on EX systems

As part of the migration process, the ASKAP team moved from bare-metal builds `askapsoft`

and all its dependencies to containers. The motivation was to alleviated some of the dependency challenges and also to improve reproducibility. This process started on the XC system and then was moved to the EX system. An outline of the current development and deployment workflow is shown in Fig. 3

The new container-based deployment immediately encountered several issues running with the Singularity container engine [5] on the EX system. The first was not specific to ASKAP but impacted all containerized workflows relying on MPI: how to inject host libraries into the container to enable inter-node communication. The integration of basic MPI required some development with little to no guidance from the vendor.

Once this issue had been addressed, another issue was encountered: incompatible libraries within the container and the host arising from the need to inject host libraries. `askapsoft` relies on a number of lower level libraries to run some services, most notably `curl`. The standard procedure had been to install all possible packages in an `Ubuntu` image using the package manager tool `apt`. This limits the versions of a package that are available and also the build options and this limitation was giving rise to the conflicting packages, specifically when host libraries need to run the MPI where injected into the container.

The solution was to make use of the `spack` package manager [6] to build most packages from source. `spack` is able to determine the dependencies of a large list of packages and build them from source or use external pre-build packages. Again, the conflicting libraries discovered here would have impacted other workflows but was discovered first during the migration.

A final issue was encountered when trying to use parallel IO. Again, this required exploring what libraries on the host needed to be injected into the container and what to build in the base container with no guidance from the vendor. We found it was necessary to build the `Lustre` package [7] and an `Lustre`-aware MPI inside the container, something that was not necessary on the earlier XC system. This solution is now present for all containers running on Pawsey.

C. IO

Running at scale on Setonix stressed the `Lustre` filesystems. The load placed on the filesystem was higher with the pipeline running on our EX system than it was running on the older XC system simple because more compute resources were available. Additionally, the filesystems on Setonix are shared between ASKAP and all other Pawsey users, unlike the filesystem available to the older XC system. Thus, when ASKAP generated very high loads, it impacted all users.

Some of the IO issues stem from the technical debt in `askapsoft`, such as the data format and the habit of using files to stitch together various executables in the pipeline. The exact IO footprint is variable as different science processing produce different IO footprints. Some produce significant number of large files and an even larger number of small files containing metadata. Others produce just an enormous number of small files. All can in principle generate tens of thousands of open/close operations per second reading reference data.

In this case, the migration offered the opportunity to test new technologies and even flesh out prototypes that could reduce the IO load of the pipeline. For instance, a possible solution to the some of the IO load is being explored by move to a new data format using the `adios2` library [8]. This on-going work is discussed in other conference proceedings.

Another issue encountered arose from the large number of IO operations per seconds per second produced by certain science process workflows. The larger number of writes that could occur also presented a challenge, but not necessarily in the obvious sense. The IO patterns exposed several `Lustre` bugs, some of which are still in current releases.

We are continuing to refine how to best work around the bugs in `Lustre`. Some involve changes to the `Lustre` configuration, as advised by HPE. A more long-term solution is a work-in-progress is focused on adding the use of compressed filesytems using `squashfs` while running the ASKAP containers. The idea is to move all IO of small files

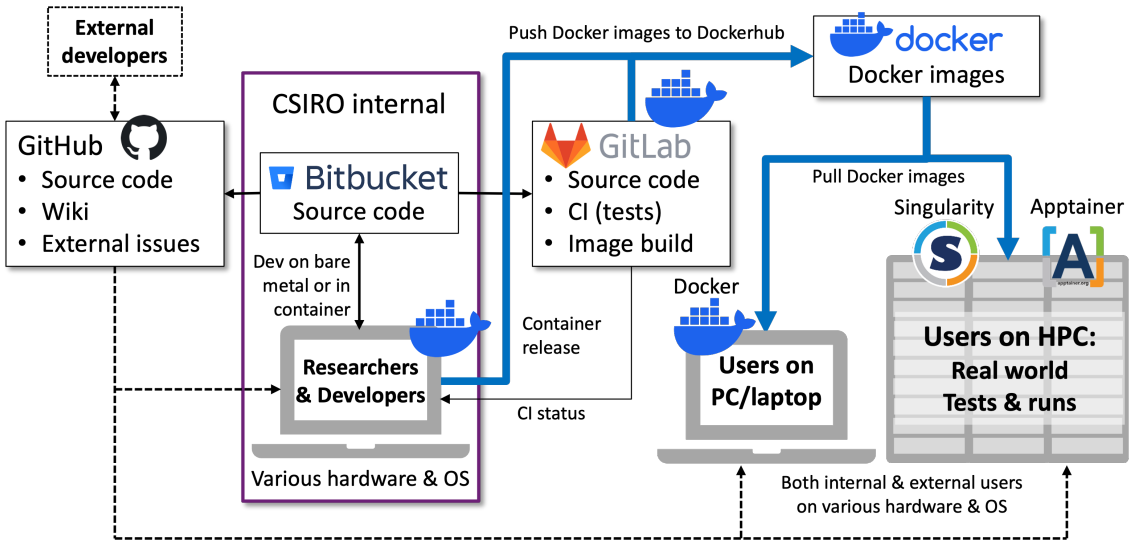

Fig. 3. The current ASKAP CD/CI using containers.

and the reference data sets to `squashfs` files that are mounted at runtime, thereby heavily reducing the load on the `Lustre` metadata servers.

The job orchestration relied on global file locks to indicate tasks within the pipeline had completed. The use of file locks triggered yet more `Lustre` bugs. Turning of global file locks off and allowing only local locks is the current workaround to the `Lustre` bug. However, this is less than ideal as all workflows on Setonix are impacted. Most workflows do not rely on global locks but those that do will be negatively impacted. The orchestration was also restructured so as to not make use of file locks.

It is important to reiterate that none of these issues were present on smaller systems or when running the pipeline at smaller scales. It was only obvious when running at production scale.

D. Workflow orchestration

The primary pipeline workflow orchestration tool is a collection of complex bash scripts that make use of the in-built functionality of `slurm` for generating and tracking job dependencies. Restarts for processing are handled outside of `slurm` and rely on files on the filesystem to indicate that some portion of the processing has completed. The design suffers from one surprising major flaw: the reliance on `slurm`.

`slurm` was never designed to be a workflow orchestration tool but rather a resource allocation tool. This is evident in the stress that can be placed on the `slurm` daemon if a center allows users to submit thousands to hundreds of thousands of jobs, with jobs having some dependencies on other jobs.

The complexity of real-world workflows can be in an example from a rather standard DAG from the Spectra and Polarisation in Cutouts of Extragalactic sources from RACS (SPICE-RACS) survey in Fig. 4. This processing would be just for a single input data set and would be need to be run many times, ideally concurrently for independent data sets.

The current default processing pipeline struggles with managing these tasks, to say nothing of logging, tracking and visualizing the progress of a given bit of processing. That is not to say that it is not functional. The orchestration uses Grafana to monitor CPU and IO load on the HPC partition and filesystems involved in the processing.

IV. Lessons Learned

Complex workflows, such as present in radio astronomy, are composed of many different soft-

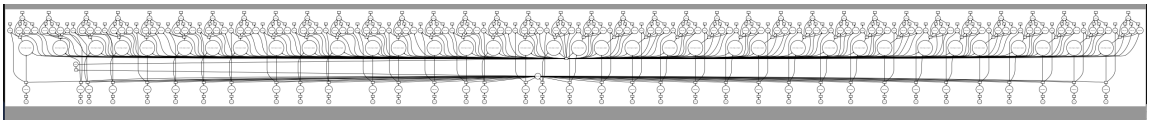

Fig. 4. Job dependencies in a portion of the SPICE-RACS pipeline. This workflow would be run many times on each individual observation from the radio telescope.

ware packages working together. Migration to new CPU architectures and simply compiling the CPU optimized software stack can be a challenge, even with tools such as `spack`. The real challenge is running workflows at scale on ever larger systems and debugging the workflow when issues are encountered.

The standard approach of profiling a single monolithic code to identify computational bottlenecks, though useful, is not a solution. The diversity of bottlenecks and issues that can arise, from job scheduling, IO, along with sub-optimal usage of available compute resources, simply requires expertise in a variety of areas.

We find it necessary to adopt an AGILE approach with both the HPC center staff with a variety of expertise, domain scientists, developers of the workflow, and developers of the tools used working hand-in-hand to ensure all components are migrated to new architecture and scale to exascale and beyond. We recommend understanding the current design of the workflow at a high-level.

- What tool(s) are used to run the workflow? Workflows now can consist of lots of steps with a complex inter-dependency. This might require a mix of tools as no single one might be fit-for-purpose.
- What tools are used across the workflow? Workflows might consist of a variety of technologies, from containers to bare-metal builds. Some tools might be rapidly developing while others provide slowly evolving API and functionality.
- What is the file footprint? Workflows will produce many different sizes of files, some of which are temporary and others need to be archived. Workflows might also try accessing lots of input files, possibly in less than ideal fashion.
- How are failures identified? With a full workflow, identifying underlying issues is far more involved.

Updating the workflow will consist of:

- Investigating IO operations across the entirety of the workflow. Portions of a workflow are well suited to parallel filesystems like `Lustre`, while others are the optimal worst case for such file systems. Improving and enabling exascale scaling will require
  - Code development of some specific tools. Example is incorporation of `adios2` into `askapsoft`.
  - Workflow development to use a variety of technologies. Example is the testing of integrating `Squashfs` into the python heavy diagnostic workflow of ASKAP. Other possibilities are the use of libraries like `capio`.
- Developing a detailed understanding of the variety of tools and their possible bottlenecks. Some packages will be memory bound while others will be compute bound, requiring varying resource requests to optimally distribute the workflow on compute resources. Improving and enabling exascale scaling will require
  - Code development of specific computationally heavy tools. Example is exploration of offloading specific portions of the ASKAP processing to GPUs, like the gridding and degridding of data.
  - Workflow development to accommodate heterogeneous architecture and resource footprint. Example is
- Investigation of workflow orchestration tool itself and it's ability to scale. Often older tools would rely on `slurm` or `pbworks`, neither of which are designed purely as an orchestra-

tion tool. We recommend investigating tools like `Prefect`, `Nextflow` and prototyping a pipeline.

- Investigation of a variety of deployment technologies. Often bare-metal builds are run on HPC systems. Building some packages can be a challenge and does not lend itself to reproducible environments. Containers offer a means of solving both issues. However, it also has its draw-backs since the rigidity does not lend itself to a fast development process. It may be necessary to use both approaches and explore other options as well.

The migration and improvement on new EX systems does not end with the direct collaboration. We have found it very valuable to take a proactive approach to community training, identifying the domain scientists who could help improve maintance of the new workflow with said training. Without this additional, more passive interaction, the workflow again might accumulate significant technical debt and encounter the same issues when the next migration occurs.

We also recommend revising the acceptance tests run on HPC systems to better mimic these complex workflows. A number of blockers we faced in our migration could have been identified earlier with a more exhaustive testing regime. Case in point, a significant fraction of the tests we now run on Setonix are inspired or directly taken from the issues we faced migrating the ASKAP pipeline.

V. Future of Exascale Radio Astronomy, the SKA and Exascale Workflows

The future of radio astronomy is the Square Kilometre Array (SKA), an international collaboration involving 15 countries, will ultimately boast a square kilometer of collecting area, making it the most sensitive radio telescope globally. Its construction is underway in Australia (low-frequency) and South Africa (mid-frequency), with the SKA Observatory (SKAO) headquartered in the United Kingdom. Once completed, the SKA will operate over a wide range of frequencies, providing unprecedented sensitivity and enabling rapid sky surveys and will be an order of magnitude larger in scope and complexity than current telescopes. The workflow is still in its design phase. The migration of the ASKAP pipeline demonstrates that the incorporation of a variety of technologies and approaches is the only path forward for running this complex instrument.

This project also shows that similar approaches will be needed for other science domains with complex workflows. The issues we encountered are often generic enough that these issues will be seen again. If the same approach is taken for updating these workflows, and HPC centers take a similar holistic approach, we will be ready for exascale workflows, not just exascale codes.

Acknowledgements

We would like to acknowledge the Whadjuk people of the Noongar nation as the traditional custodians of this country, where the Pawsey Supercomputing Research Centre is located and where we live and work. We also thank the Wajarri Yamaji people as the traditional custodians of the country on which sits the ASKAP radio telescope. We pay our respects to elders past, present, and emerging of both people. This work was supported by resources provided by the Pawsey Supercomputing Research Centre with funding from the Australian Government and the Government of Western Australia.

References

[1] J. C. Guzman and *et al.*, "Are we There Yet? Experiences from Developing and Commissioning the High Performance Computing (HPC) System for the ASKAP Telescope," in *Astronomical Data Analysis Software and Systems XXVII* (P. Ballester, J. Ibsen, M. Solar, and K. Shortridge, eds.), vol. 522 of *Astronomical Society of the Pacific Conference Series*, p. 31, Apr. 2020.

[2] "Message passing interface forum, mpi: A message-passing interface standard." https://hpc.nmsu.edu/discovery/mpi/introduction/, 1994. Accessed: 2022-12-12.

[3] J. Guzman and *et al.*, "ASKAPsoft: ASKAP science data processor software." Astrophysics Source Code Library, record ascl:1912.003, Dec. 2019.

[4] M. Jette, A. Yoo, and M. Grondona, "Slurm: Simple linux utility for resource management," 07 2003.

[5] B. M. Kurtzer GM, Sochat V, "Singularity: Scientific containers for mobility of compute," 2017.

[6] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and S. Futral, "The Spack Package Manager: Bringing Order to HPC Software Chaos," Supercomputing 2015 (SC15), (Austin, Texas, USA), November 15-20 2015. LLNL-CONF-669890.

[7] "Lustre : A scalable , high-performance file system cluster," 2003.

[8] W. F. Godoy, N. Podhorszki, R. Wang, C. Atkins, G. Eisenhauer, J. Gu, P. Davis, J. Choi, K. Germaschewski, K. Huck, A. Huebl, M. Kim, J. Kress, T. Kurc, Q. Liu, J. Logan, K. Mehta, G. Ostrouchov, M. Parashar, F. Poeschel, D. Pugmire, E. Suchyta, K. Takahashi, N. Thompson, S. Tsutsumi, L. Wan, M. Wolf, K. Wu, and S. Klasky, "Adios 2: The adaptable input output system. a framework for high-performance data management," *SoftwareX*, vol. 12, p. 100561, 2020.