Leiden University
Leiden Observatory and
Leiden Institute of Advanced Computer Science

Orchestration of Distributed LOFAR Workflows

Alexandar P. Mechev

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Astronomy and Software Engineering of the Leiden University,
November 2019

Abstract

LOFAR, the LOw Frequency ARray, is an European low-frequency aperture synthesis telescope, with a dense set of core stations near Dwingeloo, the Netherlands.

Dedication

Dedication here.

‘Quote text here.’
Guy Quoted

Contents

List of Tables

List of Figures

"I don't want to go into the speculation that structures such as Stonehengde were computers, but debugging them must have been fun " -Keith Shortridge

1 Introduction

1.1 Introduction

For almost a century, computational machines have been used as a tool to improve scientific research. In part driven by national security concerns during the First and Second World Wars, as well as the Cold War, increasingly complex computers have been designed. Early computers were entirely designed for application-specific tasks, with a large drive behind them being hydrodynamics simulations for the first Hydrogen bomb. In 1945, the popular Von-Neumann architecture was developed with the goal of making Monte Carlo simulations easier to develop and to facilitate general purpose computing. This architecture was a significant improvement over previous computers where changing the program required physically flipping switches and changing cables on the computer itself. One of the first computers built according to the von-Neumann architecture was the MANIAC computer commissioned by Los Alamos National Laboratory, seen on the left in Figure 1.1.

With the advancement of an architecture that treats code and data identically, it was possible to create more complex programs including compilers: programs that could create machine code from human-readable code. As the '50s and '60s passed, general purpose computers were increasingly used in science. From weather dynamics to fluid dynamics, from chaos theory to game theory, these computers were being adopted by a wide range of scientific fields. Astronomy was likewise also a driving force for computational innovation. In 1953, for example, the first high-level programming language for IBM computers was developed by John Backus, a programmer frustrated with the difficulty of accurately calculating the moon's position using only machine code. John Backus' 'Speedcode' was a direct predecessor of Fortran, a language developed at IBM in the '50s and still used by the scientific community today. Another important discovery on our road was the Fast Fourier Transform

(FFT), discovered by two researchers from Princeton and IBM. The FFT has been described as 'the most important numerical algorithm of our lifetime' and the author's personal favourite 'an algorithm the whole family can use'[35]. As we will soon see, Radio Astronomers quickly became part of this family.

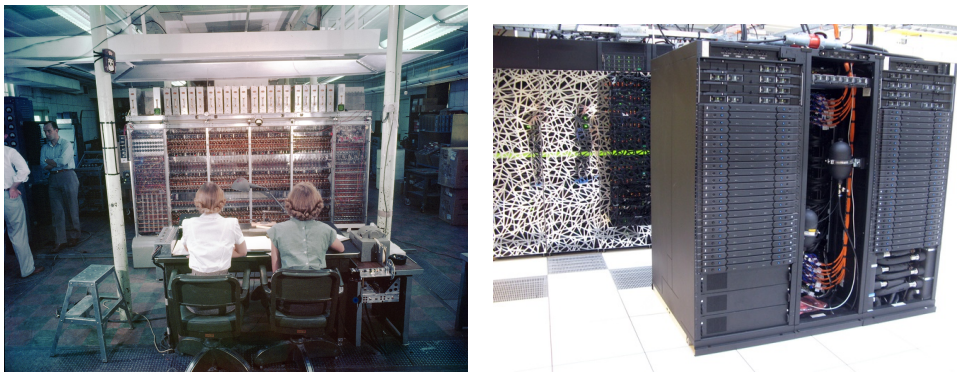


Figure 1.1: Two supercomputers sixty years apart. On the left is the MANIAC computer from 1952 at Los Alamos, while on the right is the Cartesius cluster at SURFsara, Amsterdam in 2018.

As computers became more available, they became increasingly adopted by universities and research institutes. In the '70s, computers began to talk to each other over a network connection. This capability not only made scientific collaboration easier, but made it possible to distribute computation across multiple sites. Moreover, the development of the integrated circuit and subsequent drop in price/performance of computers made it financially feasible for scientific institutes to purchase multiple computers dedicated to scientific research. As hardware, networks, and software matured, clusters of computers became more widely used[122]. In part because of their cost effectiveness, and potential for parallelization, computer clusters became more widely used as the '80s wound down. By then, general purpose computing was widely adopted by the astronomical community. In the '80s several astronomical software suites have been developed, with software such as AIPS and IRAF and standards such as FITS used to this day.

The '90s continued the distributed computing trend with the appearance of commodity compute clusters, with virtual 'supercomputers' being created from Commercial Off-The-Shelf (COTS) hardware, and networking. These clusters became quickly adopted by scientists to perform simulations and data processing. Around the same time, the idea of 'grids' was created. The concept was that of a country, continent or even world-wide network of hardware that can transparently handle distributed tasks, and provide researchers with a vast pool of resources. Grid processing

was pioneered by CERN with the goal of meeting the computational and storage requirements of their High Energy Physics modelling and data reduction. While this infrastructure was originally built for HEP experiments, it is also useful for other scientific projects, particularly low-frequency Radio Astronomy.

1.1.1 Astronomy and Computing

Since the early days of computing, the field of astronomy has embraced digitization of data acquisition and processing. Being able to store astronomical data digitally makes it possible to transfer, copy, backup and process them easily. With the rapid development of CCDs, optical astronomy entered the digital age, however with the wide availability of Analog-Digital Converters (ADCs), radio astronomy has been digital since the 1970s. By the end of the '70s, the Very Large Array (VLA) in New Mexico and the Westerbork Synthesis Radio Telescope (WSRT) had consistently been using processing pipelines, running on Digital Equipment Corporation's line of PDP, and later VAX, minicomputers. Notably their imaging algorithms were taking advantage of the FFT developed a decade earlier[25].

With the complete digitization of astronomical observations, over the past decade, all of astronomy has entered the big data regime. As of 2019, there are multiple planned and ongoing large-scale sky surveys across the electromagnetic spectrum, each expecting to produce multiple tens of petabytes. This breadth of data is poised to expand the frontiers of astronomy and astrophysics and allow us to study and understand various phenomena in more detail.

The longest wavelength of the spectrum accessible to Earth observatories lies in the Megahertz range, starting at 10 MHz, up to 300 MHz. This regime corresponding to wavelengths of 30 meters up to 1 meter. In astronomy, this range is termed the low-frequency or meter-wave regime. These wavelengths help uncover physical phenomena invisible to telescopes in the X-ray, Visible, Infrared or Microwave. In particular, long-wavelengths can be used to study supermassive black holes, galaxy formation and evolution, magneto-hydrodynamics, solar physics, radio spectroscopy, and many more science cases. Additionally, data in this domain can complement other telescopes in multi-wavelength studies.

Photons provide us with rather few independent properties. Light provides us with the wavelength, the direction, the intensity, and the polarization of a distant source, as well as the change of those properties with time. Astronomers need to measure properties accurately and use the data to better model distant sources and

validate or reject astronomical theories. The accuracy of these models, or the rejection power of our observations, depend critically on how accurately we can measure the four properties listed above.

Astronomical observations in the long-wavelength regime have always been at the mercy of the diffraction limit, an effect that relates the wavelength of light, the diameter of the aperture, and angular resolution obtained with that aperture. The angular resolution of a telescope determines how accurately the direction of an incoming photon is determined. Unfortunately, the diffraction limit dictates that the angular resolution of a telescope with a fixed aperture decreases inversely proportional to the wavelength observed. For example, if you take a telescope at 100MHz and one at 10GHz, the 100MHz telescope would need to have 100 times the radius of its higher frequency counterpart in order to reach the same angular resolution. In other words, for a low frequency telescope (at 100 MHz) to match the 100-m Effelsberg telescope (at 10GHz), it would need a dish with a diameter of 10 kilometers. Constructing, and operating a telescope of that size is currently outside our engineering capabilities, and thus low-frequency astronomers have developed a method to synthesize a telescope aperture of arbitrary size, termed 'Aperture Synthesis'.

1.1.2 Aperture Synthesis

Aperture synthesis is the practice of combining the signal of multiple antennas to produce data with the angular resolution of a much larger antenna, as seen in Figure 1.2. More specifically, the maximum angular resolution achievable is related to the distance between your furthest two antennas. This technique has been used in a wide wavelength range, from the near- and mid-infrared (VLTI), sub-millimeter (ALMA) and radio wavelengths (VLA, GMRT, LWA).

Aperture synthesis can be used to increase the angular resolution of an array of radio telescopes, however the resulting data requires significant post-processing. A single telscope operates in the 'image' domain, meaning the data collected by it is directly related to an area in the sky. Conversely, an array of telescopes records waveforms between pairs of antennas. This data needs to be transformed, in order to obtain a map of the sources on the sky. The equation relating the data recorded by these arrays and the 'true' sky distribution is the Radio Interferometry Measurement Equation, RIME. This equation describes propagation effects from the source B to two antenna, p and q, with n effects towards antenna p and m effects towards antenna q. Each effect is described by a 2x2 'Jones' matrix describing the transformation of the original signal. This formulation is shown in Equation 1.1. When expressed in terms

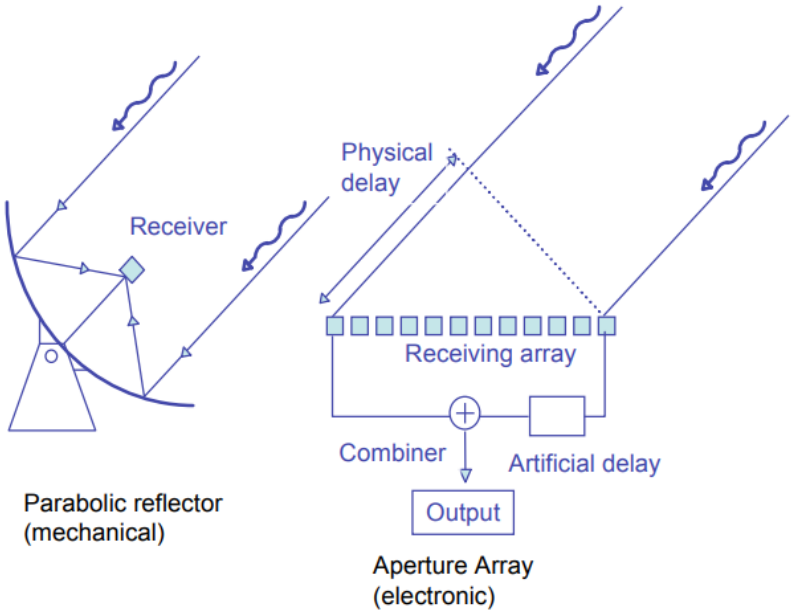


Figure 1.2: We can simulate a single dish with an array of antennas. These antennae are pointed in different directions by introducing a corresponding hardware delay in each antenna feed. While this process can enable us to synthesize an arbitrarily large telescope, it produces artefacts in the final image that need extensive processing to remove.

of the directions in the sky (l,m), and a continuous sky model, B, the measurement equation becomes Equation 1.2.

Equation 1.2 neatly separates the direction independent terms ($\boldsymbol{G}_p$ and $\boldsymbol{G}_q^H$) seen by antenna p and q, and the direction dependent effects that correspond to directions l and m inside the integral. This formalism neatly separates the direction independent and dependent effects. A comparison between equations 1.2 and 1.3 shows the similarity between the RIME and the Fourier Transform. Specifically, $f(x)$ represents the sky brightness B, ξ represents our directions l and m and the transformed function $f(\xi)$ is the visibilities (V_{pq}) measured by the telescope. This formalism separates the direction dependent and independent effects, and further shows how we can use Fourier transforms to obtain a model of the radio sources. As Fourier Transforms are computationally expensive, an efficient solution is to use the Fast Fourier Transform algorithm mentioned above.

$$\boldsymbol{V}_{pq} = \boldsymbol{J}_{pn}(...(\boldsymbol{J}_{p2}(\boldsymbol{J}_{p1}B\boldsymbol{J}_{q1}^{H})\boldsymbol{J}_{q2}^{H})...)\boldsymbol{J}_{qm}^{H} \tag{1.1}$$

$$\boldsymbol{V}_{pq} = \boldsymbol{G}_{p}\left(\iint_{lm}\frac{1}{n}\boldsymbol{E}_{p}B\boldsymbol{E}_{q}^{H}e^{-2\pi i(u_{pq}l+v_{pq}m+w_{pq}(n-1))}dldm\right)\boldsymbol{G}_{q}^{H} \tag{1.2}$$

$$\hat{f}(\xi) = \int f(x)e^{-2\pi ix\xi}dx \tag{1.3}$$

In this work, we will discuss the technical challenges of creating radio images of astronomical sources and our solutions to these challenges. In the following section, we aim to introduce LOFAR, the European Low-Frequency Array, the data sizes and processing challenges that come with LOFAR data as well as our solutions to these challenges. We will conclude with the scientific results this work has led to, as well as suggestions for future large-scale astronomical projects.

1.2 LOFAR

LOFAR is a large low-frequency radio telescope centered near Dwingeloo, Drenthe, in the Netherlands. LOFAR has thousands of Dutch antennas grouped in Core (near Dwingeloo) and Remote stations. LOFAR also has International stations across Europe, spanning from Ireland to Estonia, Sweden to Italy. These international stations make it possible to create images of radio sources with a similar angular resolution to leading higher frequency telescopes. Much like these telescopes, LOFAR was also designed to support a variety of science cases, from large scale broadband studies to spectroscopy and transient detection.

LOFAR stores its broadband data at one of several Long-Term Archive locations. These locations store the data on tape, due to its large size and infrequent access. Typical broadband observations are up to 16TB in size, which can drop down to 10TB with compression. While individual researchers use this data to study their object of interest, the majority of the broadband data will be imaged to produce the LOFAR Two-Meter Sky Survey (LoTSS).

1.2.1 LoTSS

The LOFAR Two-Meter Sky Survey, LoTSS, is an ambitious project to map the Northern Radio sky at low frequencies namely 120-168 MHz. Expected to produce more than 3000 8-hour observations, LoTSS will create radio maps with sensitivity below 100 μJy/beam. This survey will help study supermassive black holes and their impact on galaxy formation in the early Universe. Additionally, understanding the formation and evolution of galactic clusters and the interaction of galaxies within these clusters will be made possible with this low-frequency data. Furthermore, the survey will enable us to study star formation in nearby and distant galaxies and galactic sources such as supernova remnants. Finally, LoTSS will help study and discover patterns in the large-scale structure of the Universe. The observations produced by LoTSS will total more than 30 petabytes and require extensive processing before the survey is completed.

Processing Requirements

With its 3000+ observations, the LoTSS project requires a large amount of processing, bandwidth and storage infrastructure in order to complete its scientific goals within the survey timespan. The total size of raw data is more than 30 petabytes, while the total size of the finished products will be on the order of 10s of terabytes. Furthermore, moving all the raw data to processing facilities is limited by the bandwidth of the connection between the archive site and the processing facility. Finally, each data set requires 500 core-hours for the DI pipeline and roughly 3000 core-hours for the DD pipeline. In total this means that the LoTSS project will take more than 10 million core-hours to produce scientific results, assuming no re-processing of data.

In addition to the raw hardware requirements, a large project as such needs to be able to track the status and location of data products, automate processing and make results easily available. As LOFAR data is stored at multiple locations, it is also important that the framework tasked with processing LoTSS data is portable and can run independently of the infrastructure details.

1.2.2 SURFsara

One of the archive locations storing LOFAR data is SURFsara at the Amsterdam Science Park. Aside from a large storage archive, SURFsara also supports several clus-

ters, including the Gina cluster, part of the Dutch Grid infrastructure. Grid computing is a non-interactive application-oriented computational paradigm for distributed computing where a 'grid' consists of a large pool of nodes where users can submit batch jobs. A grid can consist of one cluster or groups of clusters at one or multiple geographical locations, connected with high-speed links and a common job management interface. Using this interface, users can scale out their projects, given that their processing is massively parallel. Computational resources on such a platform are granted based on a scientific proposal and are used freely across the Grid, while jobs are scheduled based on the job requirements and the current resource availability of the grid nodes. This processing paradigm is perfect for large grid-search simulations, but also for the first steps of LOFAR processing. Furthermore, the high-speed connection to the LOFAR archive and available storage makes SURFsara a logical location to orchestrate large scale LOFAR projects and distribute processed data.

1.2.3 LoTSS Processing

The observations produced by LoTSS will total more than 30 petabytes and require extensive processing before the survey is completed. Each observation is stored as a set of 244 individual files spanning frequency space from 120MHz to 168MHz. Each of these files, named a Subband, is a CASA Measurement set and is identified by its three-digit Subband number, starting from 000. Each Subband, thus, contains a sub-sample of the data in frequency space, stored at a resolution of 1 second and 12.2 kHz per sample. While this high-resolution data is useful for some science cases, our processing algorithms scale with data size, and thus it is necessary to average our data in order to complete the LoTSS processing within the project's time-frame.

In order to create an image from an archived data set, the data needs to be staged, retrieved and processed. Staging the data refers to sending a request to the archive site to move the data from tape to disk. Once all your data is on disk ('staged'), it is ready to be transferred from the storage to the processing cluster. On this cluster, a science-ready image is produced by processing the raw data through two pipelines. The first pipeline, Direction Independent Calibration pipeline removes artifacts created by 'direction independent' effects, i.e. effects that are constant across the field of interest. This pipeline is followed by the Direction Dependent Calibration pipeline which removes effects that change within the field of view.

The Direction Independent Calibration pipeline (DI pipeline) consists of two main stages. The first stage is calibration on the calibrator, which uses a short observation of a bright calibration source to determine systematic effects that are independent

of the direction of the pointing. The solutions obtained from this step can be applied to the scientific target, improving the data quality. The second step of the DI pipeline is the calibration of the target field against a sky model produced by a previous survey. This calibration determines the gain parameters of all antennas, however it does not correct for effects that vary across the field of view.

In order to create a high fidelity radio image, we need to correct for effects that not only change in time but also across the field of view. These effects, such as the ionosphere or the beam response can be modelled and removed, and their removal is the responsibility of the Direction Dependent pipeline (DD pipeline). Upon successful completion, the DD pipeline produces a radio image that can be used for further scientific studies.

There are a few software packages used to process LOFAR data, typically used in a series of steps creating a processing pipeline. The LOFAR processing pipeline steps use the software, each step encoding the processing parameters in a parameter-set (parset). A pipeline is defined by the list of steps and the parameters of each step concatenated together into a parset file. The LoTSS DI pipeline, `prefactor` contains a set of scripts used to remove direction independent effects from LOFAR data. Many of the `prefactor` steps can be executed on the data in parallel: each Subband can be processed independently. Because of the large amount of data, the best architecture for these steps is a cluster of isolated machines with dedicated disks and a high-speed connection to the data. Later we will show the benefit of automating these steps on the Dutch Grid infrastructure.

1.2.4 The life of a data set

A broadband LOFAR data set has to be processed by the Direction Independent and Direction Dependent pipelines. In this section, we will show the progress of one observation[1] from raw data to a final scientific image. Because of the large size of the data involved, we only use half of the bandwidth, from 132.2MHz to 156.1MHz.

Figure 1.3 shows an image of the data downloaded from the LOFAR Long Term Archive. There have been no corrections done to this data, resulting in a large amplitude offset (see scale bar below figure) and strong artifacts around bright sources. In order to decrease these artifacts and calibrate the brightness of the sources, we use calibration data from a known bright radio source and apply these solutions to our data. The resulting data produces Figure 1.4. We remove Radio Frequency

[1] The target is P18Hetdex03, observed on 2014-05-28 with phase centre 11h55m41.282, +049d44m52.908

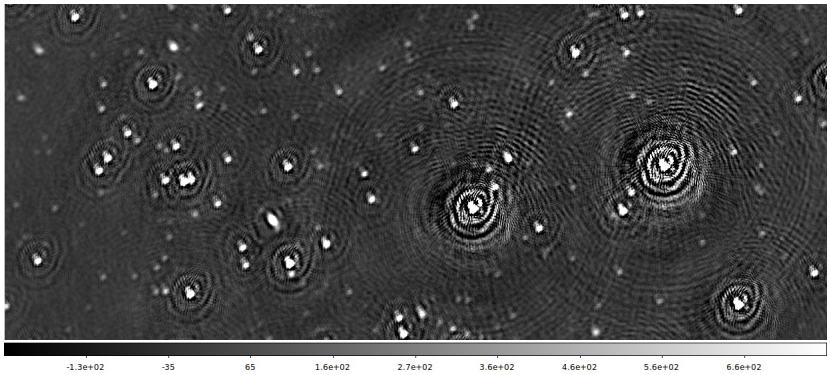


Figure 1.3: Raw data for LOFAR observation L229587. We only image half of the bandwidth, from Subband 061 to Subband 183.

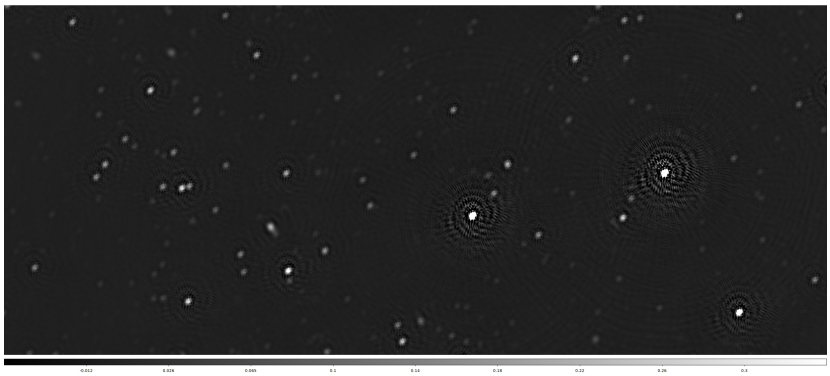


Figure 1.4: Preprocessed data for the same observation as figure 1.3. Note the corrected scale bar, obtained by applying the calibrator solutions.

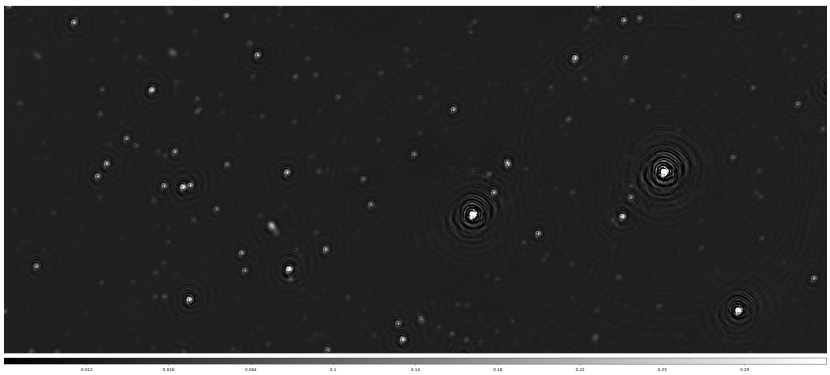


Figure 1.5: Data from L229587 after calibration against a global skymodel. This calibration removes the direction independent effects

Figure 1.6: Full direction dependent calibrated image of the L229587 data, done at a high resolution.

Figure 1.7: Full direction dependent calibrated image of the L229587 data, done at a low resolution in order to better retrieve large scale emission.

Interference (caused by manmade sources) from our data and correct for bright off-axis radio sources. Finally, we use a model of the radio sky obtained by a previous survey to calibrate all our direction independent gain parameters. The resulting data produces Figure 1.5.

After the Direction Independent calibration, we perform correction for Direction Dependent effects. This correction is done by the ddf-pipeline scripts using the DDFacet and killMS software packages. To produce the images, we use the 'tier1-jul2018' parameters with bootstrapping turned off. This processing produces two images, a high resolution and a low resolution image of our data. These are shown in Figures 1.6 and 1.7 respectively.

1.3 Problem Statement and Research Questions

Radio Astronomy data sets are too large to process in bulk on individual workstations and often strain the resources of small clusters at universities and other institutions. This limitation in resources requires high throughput processing capability, and automation in order to serve processed data in bulk to astronomers. The LOFAR radio telescope acquires data at a rate of roughly a terabyte per hour. This data is stored in a Long Term Archive as it can serve multiple science cases. Our goal is to create the tools for scientists to be able to efficiently process this data with their scripts and software. As such, these tools need to be fast, easy to use, general, and scalable.

> ***Problem Statement:*** *How can we efficiently process broadband LOFAR data in a generic way?*

1.3.1 Research Question 1

To efficiently process LOFAR data, we need to take advantage of the data level parallelism of the early processing steps, each of which can be parallelized by a factor of 244. Because of the large sizes of LOFAR observations, transfer time can be comparable to the processing time. To minimize transfer time, we need to study how to deploy processing pipelines at the LTA storage sites. In this research question, we ask how to best build a framework for massively distributed shared platform for LOFAR, and how to deploy LOFAR processing in massively parallel manner.

> ***Research Question 1:*** *How can we use a distributed shared infrastructure for efficient LOFAR data processing?*

1.3.2 Research Question 2

Once we have determined the utility of distributed processing for the LOFAR case, we ask how to automate complex LOFAR workflows. The LOFAR radio telescope serves multiple science cases, each of which is served by a multi-step pipeline with a wide set of parameters. Running an entire pipeline on a single computational node is inefficient, thus a workflow orchestration software is needed to parallelize the relevant steps. In this research question, we ask how to build software to efficiently integrate scientific pipelines with a massively parallel distributed processing platform.

> ***Research Question 2:*** *How can we build software to easily accelerate complex pipelines for Radio Astronomy?*

1.3.3 Research Question 3

Once complex pipelines can be executed on a distributed environment, researchers may ask whether the software running on hundreds of systems concurrently is running optimally. Manually monitoring automated runs is not possible, hence software is needed to collect this performance data per pipeline step. Furthermore, some of the processing parameters for LOFAR pipelines result in large data sets. In order to serve LOFAR processing to the scientific community, we need to understand how our resource usage scales with each of the processing parameters. We ask whether it is possible to integrate monitoring tools to our processing framework in a way that we can transparently collect performance data along scientific processing.

> ***Research Question 3:*** *Can we automatically collect performance information during massively distributed processing and predict run times for future data sets?*

1.4 Administrata

1.4.1 Motivation and Objectives

The goal of this work is to create a system to quickly, easily and reliably process data from the LOFAR radio telescope. Our software needs to be able to process tens of terabytes of data per day automatically, and be easy to integrate with different infrastructure providers.

1.4.2 Software Contributions

For this work, we built several software packages to define, launch, and orchestrate jobs on a high throughput cluster. The software packages built are GRID_LRT[2], GRID_PiCaS_Launcher[3] and AGLOW[4]. They are available on GitHub and their documentation is hosted on ReadTheDocs.

[2] https://github.com/apmechev/GRID_LRT/
[3] https://github.com/apmechev/GRID_PiCaS_Launcher
[4] https://github.com/apmechev/AGLOW

1.4.3 Statement of Originality

I hereby certify that the contents of this thesis is my own original work, consisting of six manuscripts submitted to peer reviewed journals and conferences. This thesis and the works therein have not been submitted to any other degree program. Finally, I certify that the intellectual content in this work and all software referenced therein are of my own work unless explicitly cited otherwise, and that all assistance in compiling this work has been properly acknowledged.

1.4.4 Publications

Chapter 2 describes our first attempts to do large-scale distributed LOFAR processing on a shared infrastructure. We detail our successes with the LOFAR Radio Recombination Lines and Pre-Processing Pipelines, our software set-up as well as the limitations and future uses of this platform.

Chapter 3 is our implementation for portable LOFAR processing on a massive scale. We show early results encapsulating LOFAR processing pipelines, and discuss future uses on other clusters. It is based on: A. Mechev, J. B. R. Oonk, et al. "An Automated Scalable Framework for Distributing Radio Astronomy Processing Across Clusters and Clouds". In: *Proceedings of the International Symposium on Grids and Clouds (ISGC) 2017, held 5-10 March, 2017 at Academia Sinica, Taipei, Taiwan (ISGC2017). Online at https://pos.sissa.it/cgi-bin/reader/conf.cgi?confid=293, id.2.* Mar. 2017, p. 2. arXiv: 1712.00312 [astro-ph.IM].

Chapter 4 discusses our work collecting detailed performance statistics from automated LOFAR processing. It is based on: A.P. Mechev, A. Plaat, et al. "Pipeline Collector: Gathering performance data for distributed astronomical pipelines". In: *Astronomy and Computing* 24 (2018), pp. 117–128. ISSN: 2213-1337. DOI: https://doi.org/10.1016/j.ascom.2018.06.005. URL: http://www.sciencedirect.com/science/article/pii/S2213133718300490.

Chapter 5 describes the capabilities of the initial workflow manager for automatic processing of LOFAR SKSP data. It shows several different scientific workflows and is based on: A. Mechev, R. Oonk, et al. "Fast and Reproducible LOFAR Workflows with AGLOW". in: *2018 IEEE 14th International Conference on e-Science (e-Science)*. Oct. 2018, pp. 136–144. DOI: 10.1109/eScience.2018.00029.

Chapter 6 describes a parametric model of resource usage for the LOFAR `prefactor` pipeline. We discuss difficulties that may arise from scaling LOFAR data,

as well the utility of our modelling method for SKA-size data. This chapter is based on A.P. Mechev, T.W. Shimwell, et al. "Scalability model for the LOFAR direction independent pipeline". In: *Astronomy and Computing* 28 (2019), p. 100293. ISSN: 2213-1337. DOI: https://doi.org/10.1016/j.ascom.2019.100293. URL: http://www.sciencedirect.com/science/article/pii/S2213133719300290.

Finally, Chapter 7 demonstrates the flexibility of our AGLOW software by implementing a full Continuous Integration pipeline for LOFAR software containers and LOFAR scientific pipelines. This pipeline can verify software pipelines against a test data set, and can be used to track the data quality of data products as the processing software evolves. This work is submitted to the IEEE e-Science Conference 2019.

1

2 Radio astronomical reduction on distributed and shared processing infrastructures: a platform for LOFAR

The contents of this chapter are based on a manuscript to be submitted to Astronomy and Computing

2.1 Introduction

In recent years, the need for multi-user oriented surveys supporting a large astronomical community with diverse scientific topics has increased, as it is often believed that this maximizes the scientific return value for a given amount of observation time. However, to encompass a large range in scientific topics within a single survey, it is often necessary to take the data at a common resolution (in terms of space, frequency and time) that supersedes the needs of the individual use cases. Such surveys therefore also require an increase in data transport, storage capabilities and (post-)processing power.

Here we will discuss the need for and implementation of a large-scale compute platform to process data sets obtained with the Low Frequency Array (LOFAR). LOFAR is a modern radio telescope observing the radio sky at low frequencies from 10 to 270 MHz [128]. Its flexible observing setup means that it supports a large range in observing modes and settings that are utilized by a diverse user community with a broad range in scientific interests and goals. Some of the main science goals for LOFAR are established through the large key science projects [KSP; 128], but many other independent projects are also performed through open skies time observing.

We present the LOFAR distributed, shared processing (LOFAR-DSP) platform. This platform represents a first step towards facilitating radio astronomical

processing on high-throughput compute infrastructures within the Netherlands and across Europe. The implementation of LOFAR-DSP has initially focused on processing interferometric imaging data for the LOFAR Surveys Key Science Project (SKSP). However, being a generic high-throughput data processing platform, it is also relevant to a broader scientific audience and next generation Big Data experiments such as the square kilometre array (SKA).

For the SKSP the typical ∼8-14TB size of a single 8 hr data set (depending on the level of compression) and the ∼50 PB size for the full SKSP survey imply that the traditional way of working, where radio astronomers use their own interactive recipes and facilities[1] to process the data, is neither efficient nor feasible. In this light, the value of well defined and robust pipelines that run on a scalable platform delivering tractable, science-ready data products can not be overstated. This is even more so with future radio telescopes, such as the SKA, through which radio astronomy will enter the Exabyte-scale era.

As radio astronomy software development remains on a fast moving track [30], we have developed LOFAR-DSP as a light-weight solution that allows for the necessary rapid adaptations in processing pipelines. The technical setup chosen borrows many elements that have been developed for distributed Grid computing and the reasons for doing this are explained in more detail in Sections 2.2 and 2.3.

LOFAR-DSP was developed primarily by and for the SKSP and the radio recombination line (RRL) processing teams. However, the platform supports a larger variety of complex LOFAR processing pipelines that require large-scale, distributed, high-throughput compute infrastructures [e.g. 8, 36, 86, 99, 108, 140]. For the SKSP case we will show how this platform, together with the Grid_LRT processing framework [78] and the AGLOW LOFAR workflow manager [71], has enabled the petabyte-scale processing of LOFAR data. Since 2017 more than 8 petabytes of SKSP data have been processed using LOFAR-DSP. For the recent SKSP LOFAR Two Metre Sky Survey (LoTSS) data release I [106] our solution directly contributed to 70 percent of the 26 accompanying papers.

The paper is structured as follows. In Sect. 2.2 we discuss the nature of radio astronomical observations and their intrinsic parallelism in the context of interferometric LOFAR SKSP data. In Sect 2.3 we briefly discuss the different compute facilities before introducing the Grid infrastructure. The implementation of the LOFAR-DSP platform is described in Sect. 2.4 and we show two workflow examples that are built on top in Sect. 2.5. We end with a discussion of the current platform

[1] These facilities typically range from a laptop computer to small clusters comprised of a handful of large desktop machines.

and possible future improvements in Sect. 2.6 and then present our conclusions in Sect. ??.

2.2 LOFAR observations, archive and retrieval

The raw and/or reduced data for approved LOFAR projects is ingested by the LOFAR radio observatory in to the LOFAR long term archive (LTA). The LTA is a federated and distributed data archive that is hosted by three data centers: SURFsara in the Netherlands [116], Forschungszentrum Jülich in Germany [39] and the Poznań Super computing and Networking Center in Poland [93].

The goal of the LOFAR SKSP team is to perform a tiered survey of the entire northern hemisphere aimed at imaging the low-frequency sky at unprecedented spatial resolution, depth and frequency coverage. This survey serves the scientific goals of about 200 researchers across Europe and its details are described in [95]. **(NB: Is this the one you want, Raymond?)**

The first tier of the survey, carried out with the LOFAR high band antenna array (HBA), is described in [106, 108]. The SKSP HBA data is observed with 1 sec and 3 kHz resolution. This is subsequently pre-processed by the radio observatory flagging and averaging pipeline [10] that performs a first round of radio frequency interference removal and then averages the data in frequency to 12.2 kHz. Given the broad range in SKSP science goals, requiring different processing strategies, the LOFAR radio observatory does not process the data beyond this initial pre-processing stage and ingests it in to the LTA.

2.2.1 Archived data sizes and transport

The archived LOFAR measurements are managed using dCache[2] as a front-end data storage manager and protected by a combination of access control lists (ACL) and Grid native X.509 based certificates. The measurements themselves are stored on tape backends and need to be moved to temporary disk storage prior to retrieval. This data staging is handled via a request through the LTA archive interface[3]. The staging service interacts with dCache that enables seamless integration between disk and tape storage. After staging, the user can download the data via either the LOFAR

[2] https://www.dcache.org/

[3] The web interface is hosted at https://lta.lofar.eu In addition a python based application programming interface (API) also exists.

download server or through a variety of Grid data transfer tools. The former provides URLs that can be resolved though HTTP for data retrieval. The latter use SURLs (Storage URLs) and TURLs (Transfer URLs) that are resolved via gridftp and SRM for data retrieval[4].

The size of an archived LOFAR SKSP data set is in excess of 8 TB for a single 8 hr observation of a 5 degree by 5 degree area of the sky[5]. These data sizes are typically too large to allow for efficient transport from the LTA sites to the compute facilities at the home institutes of LOFAR users, unless dedicated (and often costly) network connections are considered. Furthermore significant storage space would be required at these institutes, especially when considering that the processing pipelines inflate the data by factors 2–3 during processing.

This data size and transport problem implies that a different solution has to be found in order to further process and reduce these large data sets before the data is retrieved by the user. With this goal in mind we created in 2016 the LOFAR e-infra group[6] to develop a bulk processing solution enabling LOFAR processing at the LTA sites themselves. This is possible because these sites also provide access to HTC and HPC compute facilities for research that have fast connections to the data storage systems holding the LOFAR data, thus eliminating the data transfer and storage issues.

2.2.2 Radio astronomical data & parallelization

Interferometric radio astronomical observations measure visibilities. These visibilities are a Fourier transform of the sky and are represented by complex numbers consisting of measured phases and amplitudes as a function of frequency and time. The visibility measurements being independent of each other allows for a natural parallelization across time and frequency. We use this intrinsic parallelization of the data to very effectively spread (significant parts of) the processing of each large data set over many independent jobs. This setup has been demonstrated in the framework presented in [78].

The embarrassingly parallel nature of the data processing, coupled with the large data sizes and I/O intensive nature of the processing means that HTC clusters are most suitable to carry out radio astronomical processing for LOFAR data. Furthermore considering the fact that the archived data is stored in dCache managed

[4] https://www.dcache.org/manuals/Book-3.2/config/ch13s07-fhs.shtml
[5] For related working groups such as the RRL processing group a single data set can be as large as 100 TB
[6] https://www.universiteitleiden.nl/en/research/research-facilities/science/lofar-e-infrastructure-group

Grid storage naturally leads us to consider a Grid computing solution for the LOFAR Surveys processing.

The Grid computing solution invoked here is based on elements that were originally designed for the Worldwide LHC Computing Grid (WLCG) project by the WLCG collaboration and the various Grid initiatives. The Grid elements that are part of LOFAR-DSP are described in more detail in Sect. 2.4. The Grid connects a network of heterogeneous and distributed compute clusters to meet the massive compute requirements of I/O intensive data processing projects, such as the WLCG collaboration. Each compute node within and across Grid sites can be seen as an isolated island, i.e. it has contact to the outside world via internet and the workload manager, but it is not connected to the other compute nodes and cannot be logged into. This has advantages in that the use of a local file system and local scratch space is very efficient for I/O intensive jobs. However, this also has disadvantages in that the orchestration and movement of data, software and processing scripts becomes more elaborate, as compared to having access to the compute nodes over a shared file system.

Radio astronomical workflows differ from traditional high energy physics (HEP) jobs in that, in many cases, they are not completely parallel. Although, initially the SKSP workflows start out as highly parallel there are typically several instances within these workflows that require all data to be brought together in order to derive better solutions and hence deeper imaging. Examples of such workflows are provided in [30, 78, 106, 132]. In these, more complex workflows it is also an advantage to have the intermediate data products, that need to be combined, collected at a site with very fast data transfer connections between the (temporary) storage back-end and the compute cluster.

2.3 LOFAR processing on distributed compute systems

The intrinsic data parallelism, I/O intensive nature of the workflow tasks and large data sizes involved for LOFAR all imply that we need a solution suitable for high throughput compute that is connected over a fast network with the data stored in the LTA. As discussed in Sect. 2.2.2, Grid computing offers this solution. In addition to the data size and transport issues the LOFAR workflow tasks executed on the data are also demanding in terms of memory and scratch space. For example, the SKSP prefactor pipeline discussed in [78] requires a minimum of 8 GB RAM memory and 50 GB of scratch space for the most demanding tasks. Similarly, the SKSP DDF

pipeline discussed in [117] and [106] requires minimally 256 GB RAM and 3 TB of scratch space to complete.

Having been optimized for less demanding HEP processing tasks not all Grid clusters are capable of handling the requirements for SKSP processing tasks. Here we will focus on the Grid infrastructure in the Netherlands that is able to satisfy these requirements. Other Grid clusters and facilities will be discussed in Sect. 2.6.

2.3.1 SURFsara & the Dutch Grid infrastructure

The Dutch National Grid Initiative (NGI) is a node of the European GRID initiative (EGI). It is hosted by SURFsara and Nikhef in Amsterdam. The Grid infrastructure provided at SURFsara is well suited for LOFAR data processing[7]. The Grid compute resources at SURFsara are provisioned on a per core basis to Grid jobs. This provisioning guarantees minimally 8 GB RAM memory and 80 GB scratch space per requested core. Compute nodes with up to 40 cores are available. In total the cluster provides 58 TB RAM memory, 2.3 PB scratch storage and 7400 cores.

The Grid cluster has a fast connection to the more than 60 dCache disk pool nodes that are also configures as doors and that serve the dCache managed data at SURFsara[8]. Each Grid compute node has a 2×25 Gbit s^{-1} network connection and the total network bandwidth between the dCache managed Grid storage at SURFsara and the Grid cluster is 1.2 Tbit s^{-1}.

SURFsara also provides a dedicated user interface (UI) for Grid projects. This UI is aimed at easing the interaction between the Grid architecture and its users by having a rich set of Grid software and tooling pre-installed. The UI setup is identical to that of the worker nodes on the Grid cluster in terms of the supported software configuration. This enables the users to test and debug their processing pipelines on the UI before porting them to the Grid.

LOFAR has a dedicated UI called `loui`, and LOFAR Grid jobs are submitted from `loui` to the SURFsara Grid cluster using the gLite middleware software. From `loui` it is also possible to submit Grid jobs to other Grid clusters and we will discuss this in Sect 2.6. Here we consider `loui` itself to be part of the SURFsara infrastructure. The LOFAR-DSP platform at SURFsara is deployed on `loui` (see Sect. ?!).

[7] http://doc.grid.surfsara.nl/en/latest/Pages/Service/system_specs.html
[8] dCache currently manages over 50 petabytes of data at SURFsara.

2.4 LOFAR distributed shared processing platform

The processing framework (Grid_LRT) and workflow manager (AGLOW) for our LOFAR Grid processing solution have previously been presented in Mechev, Oonk, et al. [78] and Mechev, Oonk, et al. [71]. Here we present the underlying platform that we have named LOFAR-DSP. This platform re-uses and combines individual building blocks that were designed for high throughput GRID processing.

Large experiments, such as LOFAR, that generate big petabyte-sized datasets typically have lifetimes in excess of 10 years. However, compute technologies and their underlying software and hardware have a typical lifetime of maximally 5 years. This mismatch implies that portability becomes a very important aspect in the design of a long-term processing solution.

The overall aim of our LOFAR Grid_LRT processing framework is to have as few dependencies as possible and thereby enable portability across both Grid clusters and other compute facilities. The LOFAR-DSP platform provides the interface between this dedicated processing framework and the generic compute infrastructure. The platform itself is also portable, as we will discuss in Sect. ?!.

LOFAR-DSP consists of 4 elements, the: (i) LOFAR software, (ii) workload manager, (iii) PiCas client, and (iv) Grid tools. These four elements are shown graphically in Figure 2.1. Here we will first discuss the requirements imposed upon the platform by the Grid_LRT framework. Following this we will describe each of four main elements that together comprise the platform, their connection and their interfaces.

2.4.1 LOFAR Grid_LRT requirements

The LOFAR Grid processing framework, as presented in Mechev, Oonk, et al. [78], has a few basic requirements. These are, (i) access to the LOFAR dCache managed Grid storage, (ii) a VOMS client, (iii) access to the LOFAR software, (iv) Python 2.7 or higher, and (v) outbound internet connectivity to connect the local job to the main job management database. This job management database is hosted on a CouchDB instance at SURFsara, accessed through a HTTP connection, and considered to be part of the infrastructure.

The concept of Pilot jobs, see Sect. ?!, within the setup of the framework means that job submission is part of the platform rather than the framework. Job

scheduling is considered to be part of the infrastructure. Job orchestration and management are handled by the Grid_LRT framework and the AGLOW workflow manager. The LOFAR-DSP platform fulfils the requirements of the Grid_LRT framework by providing the software tools necessary to access to the software, data, job scheduler and the job management database.

2.4.2 Grid tools

The LOFAR-DSP platform defines the interface between the data storage system and the processing framework. Archived LOFAR data is stored at the LTA sites (Sect. 2.1?!). Data staging from tape to disk storage is done with the LOFAR staging service. Once staged there are two ways to access and transfer the LOFAR data, (i) the LOFAR download server as provided by ASTRON and (ii) the Grid tools. For efficiency reasons we have chosen the latter option[9] for the Grid_LRT framework.

These Grid native data transfers tools are generic and are therefore considered to be part of the platform rather than the framework. The data transport tools selected as part of the LOFAR-DSP platform are `globus-url-copy`[10], `uberftp`[11] and `GFAL2`[12]. The authorisation necessary to access LOFAR data requires a valid X.509 certificate and membership of the LOFAR virtual organisation (VO) in order to create an associated X.509 proxy. The proxy is created using the client `voms-client3` and the tools mentioned above will then use this proxy to authenticate with the Grid data storage system (i.e., dCache).

2.4.3 Workload management

The LOFAR-DSP platform defines the interface between the workload management system (or job scheduler) and the processing framework. At SURFsara access to the Grid cluster is provisioned via a variety of middleware software. This middleware software does not directly schedule the jobs on a local Grid cluster. Instead, the middleware provides and translates the Grid job to a format that can be understood by the local job scheduler (e.g. torque, pbs, Slurm, etc). The LOFAR-DSP platform uses the gLite[13] middleware and hence contains this software to interact with the Grid

[9] The limited connectivity of the download server means that the Grid data transfer tools provide data transfers that are on average more than order of magnitude faster when the network allows for it.

[10] https://www.globus.org/

[11] https://github.com/JasonAlt/UberFTP

[12] https://dmc.web.cern.ch/projects/gfal-2/home

[13] http://repository.egi.eu/

workload management system. In order to submit jobs to a Grid cluster via gLite we need a valid X.509 proxy and the LOFAR VO needs to be enabled on that cluster. The current gLite software is nearing its end of life and in Sect. ?! we will discuss alternative software.

The local job scheduler typically varies between different clusters and for non-Grid clusters there is no middleware layer to translate jobs to the required local format. Therefore this part of the LOFAR-DSP platform is less generic and can only be applied to Grid-published clusters. Fortunately, our use of Pilot jobs means that it is straightforward to change this part of the platform and accommodate different workload management systems and job schedulers. In Sect. ?! we provide an example where we run a modified version of the LOFAR-DSP platform on a cloud-based compute cluster with Slurm as the local job scheduler.

2.4.4 LOFAR Software

The LOFAR-DSP platform defines the interface between the LOFAR software distribution and the processing framework. The Grid resources, both locally and globally, are inherently heterogeneous and not accessible through a shared filesystem. In order to have a consistent LOFAR software stack available on all worker nodes we need a uniform way to distribute and compile the software.

To compile the LOFAR software we initially used the Softdrive[14] virtual drive solution offered by SURFsara. Softdrive offers a software environment that is identical to that of the SURFsara Grid cluster. Software compiled within this environment is therefore less likely to encounter errors upon execution locally at SURFsara.

The compiled software on Softdrive is then distributed across the Grid worker nodes via the softdrive.nl directory as part of the CERN VM-Filesystem[15] (CVMFS). CVMFS is optimised to deliver software in a fast, scalable and reliable way. It is implemented as a POSIX read-only file system in user space. Files and directories in CVMFS are hosted on standard web servers and mounted in the universal namespace /cvmfs. For LOFAR-DSP we host our software in /cvmfs/softdrive.nl. The softdrive.nl directory is linked to the Softdrive virtual drive and maintained by SURFsara.

CVMFS can be mounted on most computers and clusters. However, the software compiled within the Softdrive environment typically only applies to systems

[14] http://doc.grid.surfsara.nl/en/latest/Pages/Advanced/grid_software.html
[15] https://cernvm.cern.ch/portal/filesystem

with a matching operating system and similar hardware. To deploy the LOFAR software across different infrastructures the Softdrive solution for compilation is therefore insufficient. Software containerization, as provided by for example Singularity and Docker, enables software to be abstracted from the environment in which they run. This allows us to port the LOFAR software across different compute systems. Since late 2017 we have containerized the LOFAR software using Singularity and provided images[16] that we distribute via the softdrive.nl directory in CVMFS. We chose Singularity as, opposite to Docker, it enables us to execute the software image in user space.

Containerized software and it associated images provide an excellent first step in abstracting the LOFAR software from the local operating system and software environment. However, it does not fully abstract the LOFAR software from the underlying hardware and directly associated software. Important here are for example CPU instruction sets. If a software image is compiled on a system that has a CPU instruction set which is not compatible with that of the compute system where the image is executed then the software will very likely fail[17]. To eliminate this problem for the LOFAR software we have therefore setup a KVM-based virtual machine (VM) that emulates the lowest common denominator in the accessible Grid hardware for LOFAR Grid jobs. This VM is used for LOFAR software compilation and hosted on the HPC Cloud system at SURFsara.

We have tested the performance of common LOFAR processing tasks for both natively compiled and CVMFS-hosted software in [73]. In that work, we show no significant difference in performance between native compilation and software distributed by CVMFS. Similarly no significant differences in performance are found between natively compiled software and Singularity-based software images for LOFAR software.

Singularity images can also be compiled, hosted and versioned on SingularityHub. The downside of remote hosting of software images is the transfer time for those images, which can become comparable to the processing time for short jobs. We visualise the pros and cons of different distribution methods in Table 2.1.

[16] Our full LOFAR Singularity images have sizes of 5–10 GB. Removing unnecessary source ode and invoking squash-fs compression we can reduce these images to sizes of 1–2 GB.

[17] Typically an error of the 'illegal instruction' is cast by the system.

	image on SingularityHub	image on softdrive	image on Grid Storage
Versioning	automatic	manual	manual
Authentication	none	none	grid proxy
Download time	∼minutes	instant	∼seconds
Requirements	singularity	singularity and cvmfs	singularity and gridtools
Compilation difficulty	easy	easy	easy
Deployment time	instant	∼minutes	instant

Table 2.1: Pros and cons of the different distribution methods for the LOFAR software. Deployment time refers to the time taken for the compiled software to be accessible at the processing nodes.

2.4.5 Job management: PiCas & CouchDB

The LOFAR-DSP platform defines the interface between the job management database and the processing framework. The Grid_LRT framework makes use of the PiCas pilot job workflow[18] and the LOFAR-DSP platform therefore includes the PiCas client.

The PiCas pilot job workflow was created by SURFsara as a light-weight Pilot job framework[19] that is easily adaptable and extendable. The central server for the PiCas framework is based on a web accessible CouchDB[20] database. For LOFAR-DSP this central job database is hosted by SURFsara and considered to be part of the infrastructure.

Prior to pilot job submission, the central job database has to be populated. This is done by a set of dedicated CouchDB scripts that generate so-called job Tokens containing the required job input and tasks for a pilot job. Pilot jobs submitted to a compute cluster are like regular jobs, but instead of executing the task directly they contact a central server once they are running on a worker node. Then, and only then, will they be assigned a task, get their data and start executing.

The central server handles the request from pilot jobs and keeps a log of what tasks are being handled, are finished, and can still be handed out. This enables a powerful way of conducting central administration and management of jobs across a set of distributed compute resources. Only one central database is required to serve job input to pilot jobs running on any of the Grid worker nodes. Similarly, the same database is used to serve job input to pilot jobs running on any other cluster. Examples

[18] http://doc.grid.surfsara.nl/en/latest/Pages/Practices/picas/picas_overview.html
[19] http://doc.grid.surfsara.nl/en/latest/Pages/Practices/pilot_jobs.html
[20]

of LOFAR-DSP pilots jobs on cloud-based compute clusters and HPC systems are provided in Sect. ?!

At SURFsara, LOFAR-DSP based pilot jobs are submitted via gLite to the Grid cluster. Once the job lands on a worker node it contacts the PiCas server for job input. During execution the status of a job is tracked via the PiCas client and the associated job Token is updated in real-time within CouchDB database. The web frontend interface of PiCas enables quick sorting of all job Tokens into user defined views that provide real-time monitoring of all jobs within the database.

2.4.6 SPUI: continuous Pilot job processing

!! TBD - Final subsection in section 4: First explain that the above LOFAR-DSP platform is basically been integrated as part of LOUI (refer to loui in start of this section). For just LOFAR-DSP we could also generate a dedicated portable VM/-container... then continue with as loui is for generic lofar users and for testing we use spui for ...

The PiCaS job token database can be updated at any time. Every time a job token is created, a pilot job also needs to be submitted to a worker node so that it 'locks' the token and processes the data. Without a pilot job, the PiCaS tokens will stay in the 'todo' state, waiting to be launched.

To automate processing, LOFAR-DSP provides a dedicated UI, the Surveys Project UI, `spui`. The goal of `spui` is to use a robot proxy in combination with a cron job to schedule the submission of new pilot jobs at regular intervals (e.g. daily for LGPPP, see Sect. 2.5.1). If these pilot jobs do not find a job token in the 'todo' state, they simply exit. In addition these jobs can also reset error Tokens (check scrub count) and reset any failed Tokens, in the case of errors caused by heavy infrastructure load or network interruptions.

Users can perform multiple runs of processing pipelines in a single PiCaS database. These runs may be for testing purposes, different scientific projects, or even multiple runs for a single direction in the case of deep field studies. To discriminate between the purpose of these runs, each job token includes a text field used to combine 'connected' jobs. Alternatively one could also consider multiple PiCaS databases, with each database specializing for a specific pipeline or project.

2.4.7 LOFAR DSP Workflow and scripting rules

Consider moving this to Sect. 5 and shortening it (TBD)

The LOFAR DSP workflow is designed around two main scripts, the so-called master.sh script and the job.py script. The job.py script provided the actual data processing flow taking the input data set through a number of user defined processing steps to a self-contained output product. This user script is generic (independent of the underlying infrastructure) and only relies on the presence of the processing software and the input data set. The only rule that the job.py script has to adhere to is that the input and output data set naming has to follow the definitions set in the master script. To allow for a single and easy output naming convention we require that job.py tars all required output products into a single file with the extension .fa.tgz. The job parameters, as defined in the job Token, are passed to the script as arguments by master.sh and thus are known upon execution. A skeleton version of the job.py script is provided in appendix A.

The master.sh script provides all the necessary interaction between the DSP platform and the job.py processing script. This script thereby provides a natural abstraction for the user defined workflow from the underlying platform and infrastructure. The script sets the environment (e.g. initializes and links the processing softwarem and enviroment paths), reads the job parameters from the job Token, loads the input data, executes (incl. passing of the job parameters) the job.py script and saves the output data from job.py to the storage. A skeleton version of the master.sh script is provided in appendix B.

2.4.8 Job readiness and submission

Consider moving this to Sect. 5 and shortening it (TBD)

LOFAR-DSP uses the glite workload manager system (WMS) to schedule its jobs on the GINA and NIKHEF clusters. Glite jobs are defined in a Job-Definition Language (JDL) file, which describes the entry point of the job, the processing queue, the processing requirements, and the list of files uploaded to the worker node.

While there are several job queues offered by SURFsara, we submit on the medium queue, where jobs have a maximum run time of 36 hours. Once the pilot jobs have been submitted to glite, the CREAM job manager resolves the jobs in the queue and sends them to Compute Elements (CE's) that are registered to the queue and have available resources.

Glite and CREAM do not have access to the job outputs while the job is running, thus job status needs to be logged at a remote location. In our case we use a CouchDB instance following the PiCaS framework. Once che job is completed, the user can retrieve logs using the `glite-wms-job-output` command.

Before we launch a processing job, it is necessary to verify that the required files are available. As the LOFAR LTA stores all data on tape, before each job, we send a staging request, requesting data be moved to disk and made accessible. We can send this request through an XMLRPC interface provided by ASTRON, or through a gfal commands, assuming an active X.509 certificate. We use the above two interfaces to check when all the required files are available before launching processing jobs.

2.5 LOFAR DSP examples

Here we describe two cases making use of the LOFAR-DSP platform in detail; (i) the LOFAR custom script pipeline (LGCSP) with as a specific example the LOFAR RRL spectroscopy pipeline (tied-array and interferometric data), and (ii) the LOFAR GRID pre-processing pipeline (LGPPP). The first case highlights the data flow and the detailed user interaction necessary for a typical LOFAR reduction flow on a shared system. The second case highlights our first steps to abstract the user from the details underlying the specific data storage and compute facility thus allowing the user to focus only on his or hers specific data reduction parameters. This we hope will create an easier accessible and more user friendly environment for LOFAR data reduction. The two cases presented in more detail below are examples of an end-to-end reduction procedure, although the output products can be accessed for subsequent reduction steps. A more advanced implementation of the LOFAR prefactor calibration and imaging pipeline on the same platform is discussed in [78].

2.5.1 Lofar Grid Pre Processing Pipeline, LGPPP

While LOFAR-DSP supports complex processing pipelines, some LOFAR users prefer interactively processing their data. This use case is common for new science cases, or when developing and debugging new processing techniques.

In these cases, users need to transfer LOFAR data to their personal machines or clusters at small institutions. To minimize transfer time, we serve pre-processed data to these users by automating the first few processing steps. Typically, the first few steps of broadband LOFAR processing are common to many processing techniques and they reduce data sizes by factors of 4x up to 64x.

The LOFAR Grid Pre-Processing Pipeline, LGPPP, aims to automate this processing and serve pre-processed data to users. This pipeline consists of a separate PiCaS database, where LOFAR users can create pre-processing job tokens. Each of

these tokens contains a data set to process and its processing parameters. Once the tokens have been created, the data is moved online by automated scripts running on `spui` and the pilot jobs are launched. The processed data is distributed on a WebDAV client connected to a dCache directory holding the results.

2.5.2 Direction Independent Spectroscopic Calibration,DISC

Options: ... (i) dedicated user (spec-interferom, spec-tab, KE pipeline) ... (iv) aglow is now the workhorse for prefactor ... (v) ddf (containerized, 6-7 days runtime, 32 cores, 3 TB scratch, atleast 256 GB ram i.e. need a dedicated node, with special memory settings)

2.6 Discussion

The LOFAR radio telescope produces terabytes of data daily. These data rates are too high to transfer and process at scientific institutions. To enable efficient processing of this data, we need to pre-process the incoming LOFAR data and serve smaller, averaged data sets to the wider community. We present the LOFAR-DSP platform, which we show is a major building stone to enabling massively distributed LOFAR processing.

2.6.1 The need for further automation and optimization

The complex LOFAR processing pipelines can fail at times when the data has errors or is incomplete. In an automated environment, these failures need to be reported and addressed. The minimal check that needs to be done is to verify that the jobs completed, and that the resulting data is uploaded to the dCache storage. These checks can easily be done on `loui`. Furthermore, it is a good idea to determine data quality criteria and verify that the processed data meets these criteria. This can be either done on the user interface node or as an automated processing job launched on the cluster. These steps require further work and will likely be implemented and detailed in future work.

2.6.2 Grid computing for Radio Astronomy

LOFAR processing jobs fall into two different job types. Parallel, distributed jobs require many small nodes, and can be spread across one or multiple clusters. Whole

bandwidth jobs require the data to be on the same node, and thus require more than 256GB of RAM and several TB of local disk space. While both of these job types can be served by a Grid environment, we focus on the former as it uniquely requires a large scale distributed environment such as the grid. Moreover, the large imaging pipelines run for more than the 36 hours allotted to jobs in the medium queue.

One of the biggest hurdles to distributed jobs on the grid are the requirements of an authenticated user account and active proxy on the processing nodes. The main requirement for this authentication is to provide access to LOFAR SKSP observations and directories storing processing results. While archived LOFAR data can be accessed with the use of ASTRON login credentials and an http download server, uploading processed data cannot currently be done without a grid certificate. In future work, we'll discuss the possibility of using dCache macaroons as a method to authenticate both the download and upload of data.

2.6.3 The other LTA sites

LOFAR data is stored at two additional locations beside SURFsara in Amsterdam. The majority of data outside Amsterdam is at the Forschungszentrum Jülich in Germany, and the rest is at PSNC in Poznań. The processing cluster at Jülich has networking restrictions which prevent it from using CouchDB on the worker nodes. Additionally, it neither has CVMFS nor Singularity in terms of software distribution methods. In order to process LOFAR data on this cluster, we manually set up a copy of the softdrive installation and use scripts to interface the worker nodes with the CouchDB database through a shared file system.

The nodes of the grid cluster at PSNC can be registered with the workload manager, and this jobs can be sent to them from `loui`. The integration of this site with our workload manager makes it easy to launch processing jobs at SURFsara and Poznań from a single location. Doing so minimizes the workflow orchestration overhead for LOFAR jobs.

2.6.4 Non-Grid clusters & Cloud systems

The LOFAR-DSP platform is designed to be lightweight and portable. These design considerations make it possible to launch jobs on scientific clusters and cloud providers. We have tested jobs running on multiple clusters at scientific institutions. We successfully processed LOFAR data on the herts cluster at Hertfordshire and on

the lofar cluster at Leiden University. Additionally, we tested launching jobs on the Digital Ocean cloud platform, Linode and Google Cloud compute.

Some setup is required for job processing on these sites. Minimally, they need either access to softdrive or an installation of Singularity to be able to run the most recent version of the LOFAR software. Furthermore, they need an active grid proxy, or a dCache macaroon to be interface with the LOFAR Archive and intermediate data at SURFsara. Finally, they need an automated way to start job processing, either as a part of the deployed image, through a REST API or by registering to the workload management system. Orchestration of jobs on these sites can either be done through their respective APIs or through IaaS software such as Kubernetes.

2.7 Conclusions and Future perspectives

In this paper, we present the requirement for a distributed shared platform to serve large scale LOFAR processing. We describe the underlying technology and our implementation of the platform. We note the strengths of our solution as well as suggestions for future improvements, which will make the LOFAR-DSP platform easier to use and scale across multiple sites.

The contributions of this work are the following:

- Platform for Distributed processing of LOFAR data on a Shared e-infrastructure.
- Tools to manage job progress on an external database.
- Two examples of LOFAR data reduction pipelines.
- Discussion on LOFAR-DPS software life-cycle and integration with future tools.

The platform described by this work has processed over 1000 data sets for the LOFAR Two-Meter Sky Survey (LoTSS)[108], and has contributed to the data reduction for more than 30 scientific publications. The scientific achievements thanks to this work is clear evidence of the need for distributed processing of LOFAR data. We discuss the future evolution of this platform including its potential for large processing for data by the Square Kilometer Array.

Aside from the rapid development of LOFAR pipelines, we need to consider the software life-cycle of the underlying distributed infrastructure and software tools.

2.7.1 Future of Grid

The LOFAR-DSP platform uses glite-wms and glite-ce for job management and execution. The above software packagess are at their end of life, meaning they may not be supported on the grid for longer. Pilot jobs are portable by design and independent of the workload manager, however they need to be launched on a distributed infrastructure by a workload manager. Suitable replacements for the WMS are HTCondor and Dirac. Both of these solutions enable launching pilot jobs on multiple clusters, and as such are a good fit inside the LOFAR-DSP platform.

Another tool nearing its end of life is the `globus-url-copy` utility used to interface with the GridFTP. Alternatives to this tool are a dCache native API or dCache macaroons. As noted before these solutions have the added benefit of granting temporary access to private data and dCache directories. We plan on discussing their integration into the LOFAR-DSP platform in the future.

2.7.2 Improving SKSP processing

Due to the large data rates produced by the LOFAR telescope, the only viable processing and re-processing location is at the Long-Term Archive locations. Doing full processing at the radio observatory is technically difficult and considering that the data processing pipelines are constantly improving, doing so may lead to the need for future re-processing at the LTA sites anyways.

Having a stable, scalable platform to automatically process LOFAR data also means that we can implement Continuous Integration pipelines. These pipelines can verify and validate code committed in LOFAR pipelines to ensure that the fast development pace doesn't break the LOFAR pipeline.

With the upgrade of the LOFAR correlator, we expect data acquisition rates to double, increasing the processing challenges. The increased data rates can only be handled by a distributed processing platform, as they will be too much for one single site. Future radio telescopes such as the Square Kilometer Array will have data rates hundreds of times that of LOFAR **(REF?!?)**, requiring a massively scalable distributed processing platform. With a enough clusters registered to the LOFAR-DSP platform, it will be able to tackle complex workflows at SKA data rates.

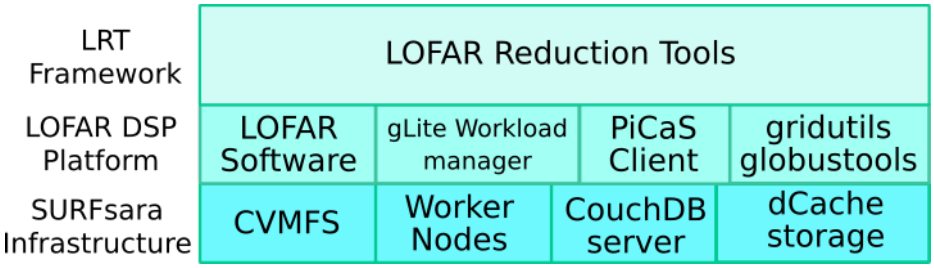


Figure 2.1: Note: We miss here the Airflow server and the Time series (TSDB) data base for profiling. For CVMFS we also need to add Softdrive (see AJDI) image. Finally for the dCache storage we should also mention the staging service (API) by ASTRON and how it is used in connection with the LTA meta data server (AstroWise) to make the LOFAR data 'FAIR'. Other missing components (macaroons as a future option)?

Acknowledgments

This work was carried out on the Dutch national e-infrastructure with the support of the SURF Cooperative through grants e-infra 160022 & 160152. The authors would like to thank the staff at SURFsara for all assistance received during this project. The authors are also grateful for the support received from Forschungszentrum Juelich and the Poznan Super computing and Networking Center (PSNC). JBRO acknowledges financial support from NWO Top LOFAR-CRRL project, project No. 614.001.351.

This work has made use of data from the distributed LOFAR long-term archive (LTA). The LTA is hosted at SURF/SURFsara (The Netherlands), Forschungszentrum Juelich (Germany) and the Poznan Super Computing Center (Poland).

LOFAR (van Haarlem et al. 2013) is the Low Frequency Array designed and constructed by ASTRON. It has observing, data processing, and data storage facilities in several countries, which are owned by various parties (each with their own funding sources) and are collectively operated by the ILT foundation under a joint scientific policy. The ILT resources have benefited from the following recent major funding sources: CNRS-INSU, Observatoire de Paris and Université d'Orléans, France; BMBF, MIWF-NRW, MPG, Germany; Department of Business, Enterprise and Innovation (DBEI), Ireland; NWO, The Netherlands; The Science and Technology Facilities Council (STFC), UK.

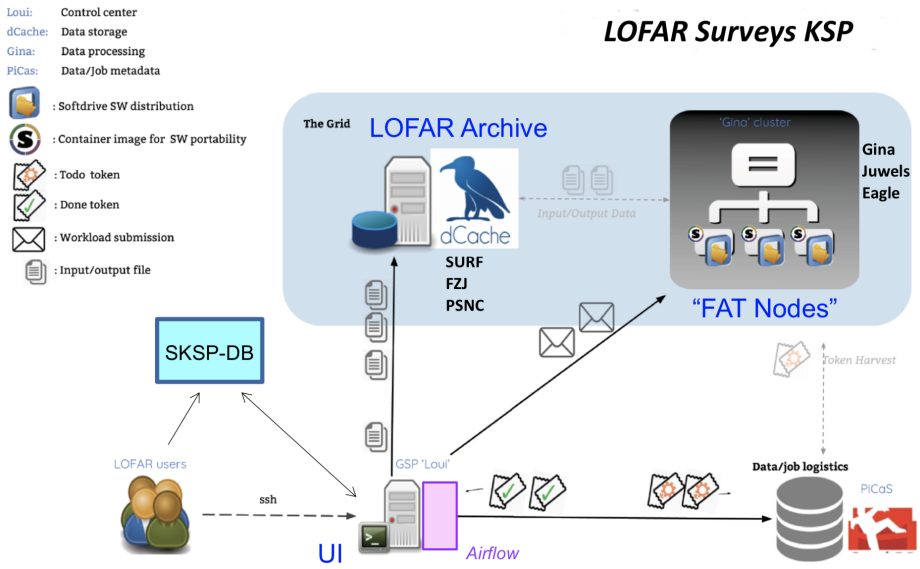


Figure 2.2: Note: are there missing components? E.g. Softdrive, CVMFS, SingularityHub, Github, CI, TSDB, LOFAR dCache distrib-webdav (open via webdav without a certificate, alternative secure option is via dCache macaroon (key-value pair with caveats) (**NB: What does this mean?!**).

3 An Automated Scalable Framework for Distributing Radio Astronomy Processing Across Clusters and Clouds

The contents of this chapter are based on a manuscript Published at the International Symposium for Grids and Clouds 2017.

Original Abstract:

The Low Frequency Array (LOFAR) radio telescope stationed near Exloo, the Netherlands is an international aperture synthesis radio telescope used to study the universe at low frequencies. Aperture synthesis requires large amounts of computation between data acquisition and science ready images. The LOFAR Two Meter Sky Survey (LoTSS) will require to process 50 PB of data within five years. The data rates demanded by this project require processing at locations with high-speed access to the data. The current software packages are not suited for all cluster architectures, and cannot launch and monitor processing at multiple locations.

To complete the LoTSS project, the processing software needs to be made portable and moved to clusters capable of handling the data rates above. This work presents a framework that makes the LOFAR software portable, and is used to scale out LOFAR data reduction. The hight throughput achieved will make imaging 3000 observations possible within five years.

3.1 Introduction

The LOFAR radio telescope is the world's largest aperture synthesis array with more than 20,000 antennas and baselines of 60 m to 1000 km[129]. With its unprece-

dented sensitivity and angular resolution at ultra-low frequencies, LOFAR's goals are far reaching: from studying pulsars and supernova remnants to the evolution of distant galaxies. Additionally, LOFAR is a pathfinder for the larger Square Kilometer Array (SKA) radio telescope. The SKA-Low telescope is expected to increase data size[45] to 400TB per day creating more than 120PB per year[11].

The LOFAR Two Meter Sky Survey (LoTSS) Survey[109] will observe 3000 different fields that collectively will map the entire northern radio sky. The size of each of these data sets is 16TB, making for a total of 48 Petabytes. To complete the LoTSS survey in the project's anticipated 5 year duration, 1PB of data need to be processed each month.

To mitigate delays caused by data transfer, the reduction must be done at a location with a high bandwidth connection to the raw data. Software packages for the initial processing of LOFAR data already exist[133], however they were not designed to work on all cluster architectures. To complete the LoTSS project in time, a framework is needed to automatically process multiple data sets at a cluster with a fast connection to the data.

Building on previous work (Oonk+, in prep), we created a framework to launch and monitor processing for multiple data sets. We present this framework, built to process data sets across multiple machines. The framework is named the LOFAR Reduction Tools (LRT) and it provides:

- Automation, enabling processing of multiple concurrent jobs
- Portability, enabling processing at different locations
- Scalability, enabling adding worker machines as required by the workload
- Generalization, enabling integration of software from other scientific domains

The LOFAR data reduction software known as 'pre-FACTOR'[30] was integrated in this framework. This software has been in use since November 2016 and at the time of writing (Feb 2017) had processed more than 100 data sets. This equals one quarter of all the LoTSS data gathered from September 2014 to March 2017. By deploying the LRT framework on a cluster with a high-bandwidth connection to the data, the entirety of the LoTSS data can be reduced within the five year time span of the project.

The paper is structured as follows: Section 3.3 outlines the LOFAR data reduction process and computational requirements. Section 3.4 describes the design of the LRT framework and its capabilities, Section 3.4.2 describes the modification of the existing LOFAR software, and the performance and results are in Section 3.4.4. Finally, conclusions and future work are in Section 3.5.

3.2 Related Work

Scientific processing is increasingly moving to grid and cloud based distributed computing. With increasing data sizes, researchers have begun focusing on scalable ways to parallelize their workflows. From genetic sequencing [104] and bioinformatics[94] to neuroscience[131] and ecology [79], ever growing data sets have driven the development of distributed workflow systems in science[31][146].

The framework presented in this publication is built on previous work distributing LOFAR pre-processing on a computing cluster (Oonk+ in prep) using a PiCaS server to track progress[91]. The required infrastructure, a PiCaS[91] server and a CernVM Filesystem client[1] have already been deployed on the target cluster and tested prior to this work. Additionally, continual support for these packages was provided by the SURFsara science support group.

PiCaS[15] and CouchDB[5] have been used in other distributed computing projects to launch and monitor jobs and exchange metadata. Job monitoring using PiCaS is also used in projects such as Sim-City[14] and Finite Element modelling for sea dyke design[61]. In these works, pilot jobs were automatically launched and tracked remotely. The PiCaS framework enabled a high degree of automation, and has helped process large amounts of data.

CouchDB is also successfully used by the LHCb team to monitor the nightly build process of their software[26] and by Sante et al.[102] to launch asynchronous jobs to visualize and analyze gene sequencing data. As CouchDB documents can hold arbitrary information and attachments, the use of the CouchDB platform favors projects requiring the storing of metadata for many concurrent jobs.

CernVM-FS[1] has been used by projects to package and publish software. Software such as the ATLAS [96] and the NOνA [29] are compiled on a central server and published to worker nodes. The LOFAR software has been similarly packaged (Oonk+ in prep). This makes deployment of processing scripts possible without compilation on the worker machines.

3.3 LOFAR Data Processing

Creating images from data collected by an aperture synthesis array[19] requires several steps of calibration and imaging. In order to place this work in the proper radio astronomy context, a brief introduction to LOFAR data processing follows. Section

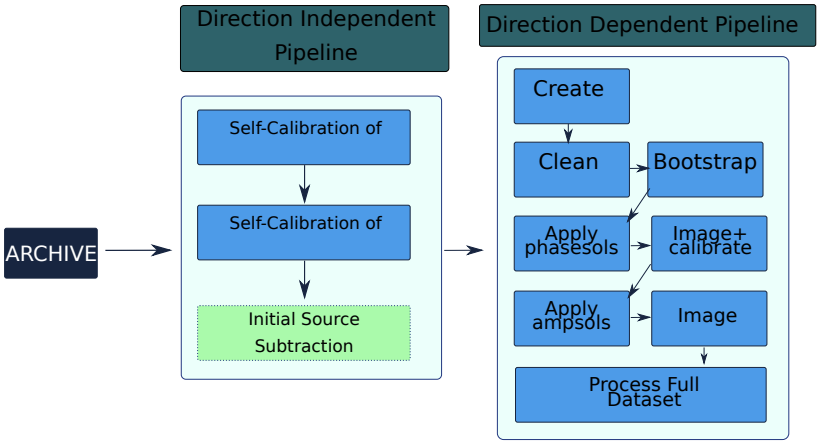


Figure 3.1: A schematic of the data flow through the Direction Independent Pipeline (pre-FACTOR[30]) and one of the Direction Dependent Pipelines (DDFacet). The Initial Source Subtraction is only necessary for some DD pipelines and is not currently implemented.

3.3.1 gives an overview of LOFAR processing from an archived observation to a final image. A schematic of this is in Fig.3.1. Section 3.3.2 details the processing steps currently implemented as well as their computational challenges. Section 3.3.3 contains an overview of the benefits of integrating the processing software with the LRT framework. Finally, section 3.3.4 gives description of the processing by focusing on the data flow. A visualization of the processing detailing the flow of data is presented in Fig. 3.2.

3.3.1 Producing Images From LOFAR Data

The raw data is stored in the LOFAR Long Term Archive. Typically, an 8 hour observation results in a 16 TB data set split into 244 65GB files. Throughout the LoTSS data processing, this data is reduced to a 500GB set of calibrated data. The calibrated data is then imaged producing a final set of a few 1.2GB images of 25k*25k pixels. The calibrated set is archived as it can be re-imaged in the future.

Data reduction for the LoTSS survey is split into two pipelines: Direction Independent (DI) calibration and Direction Dependent (DD) calibration (Fig3.1). The Direction Independent pipeline[30][133] produces images that are limited in resolution and contain instrumental effects[109]. To achieve high fidelity continuum images, the ionospheric and beam errors must be corrected[120]. As those effects vary across the field of view, a Direction Dependent calibration step must follow. Without

these corrections the image quality and resolution is severely limited[133]. We are currently working on a GRID implementation of the Direction Dependent pipeline and will report on this in a future paper.

3.3.2 Direction Independent Processing

The LOFAR telescope consists of many antennae, each with its own electronic gain. A gain calibration needs to be performed by observing a bright calibrator source before or after the science target[120]. Using this observation, the antenna gains for the telescope are calculated. This is performed by the Calibration pipeline of the pre-FACTOR software[30]. The results from this step are applied to the science target and target observation is averaged and processed. This consists of removing Radio Frequency Interference and subtraction of bright off-axis sources, and finally calibration against a sky model derived from surveys conducted with previous telescopes[120][133]. These steps are performed by the Target pipeline of the pre-FACTOR software[30]. The result is a calibrated data set which is up to 64 times smaller than the uncalibrated archived data.

A 16TB data set cannot fit into a machine's memory, which is typically less than 128GB. Because of this, the Direction Independent processing is be split by dividing the original full-bandwidth observation into independent chunks of narrower bandwidth. A typical observation spanning 48 MHz is split into 244 files called Subbands each of which spans 0.1953 MHz. These Subbands are processed simultaneously, as each undergoes the same processing steps. This is a form of data-level parallelism.

The Direction Independent calibration pipeline (Fig.3.2) consists of an existing set of scripts which use the LOFAR software suite[92] to process the initial data sets. These scripts also handle the post-processing and application of the calibration results[133]. These scripts are contained in the package *pre-FACTOR*[30]. The order of the scripts and their parameters are contained in a parameter-set file (henceforth 'parset'). The parset defines a sequence of procedures, each launching one or more executables. The execution of the procedures in a parset defines a step in the Direction Independent Pipeline.

3.3.3 Implementing the DI Calibration on the GRID

The LoTSS survey is mainly conducted by a team at Leiden University. The bandwidth between a University cluster and the LOFAR data archive is too low to down-

load the data, 10 MB/s in the case of Leiden University. At Leiden University, downloading of one data set would take 10 times longer than the processing. The processing was moved to the SURFsara grid location at the Amsterdam Science Park[1] as there were previous successes in processing LOFAR data at SURFsara (Oonk+ in prep). In order to take advantage of the computational resources at the SURFsara Gina cluster[40], the LOFAR pre-FACTOR pipeline was modified as part of the development LRT framework. The two steps of the DI reduction, the Calibrator and Target, were each split in two parts. The first part of the Calibrator and Target processing is parallelized by running one file per node, and the second runs combine these results. This takes advantage of the data level parallelism of LOFAR processing.

Additionally, splitting the computation makes it more robust. In the case that the download or processing of one job fails, it can be restarted without disrupting parallel jobs. When a step has finished processing, the next step can be launched automatically enabling the massive processing of LOFAR Surveys data.

The pre-FACTOR software was designed to be run on single machines or clusters with a shared file system. Because the worker machines at the SURFsara cluster have isolated storage, scripts are included in the LOFAR Reduction Tools to load the relevant data on the worker node before processing. After a job is finished, the scripts save intermediate results to a storage location external to the cluster.

3.3.4 Data Flow and Processing Steps

A researcher interested in processing LOFAR data needs to download it from the LOFAR Long Term Archive. Leiden University leads the LoTSS survey and has a computer cluster dedicated for LOFAR processing. The connection between the Leiden location and the LOFAR data archive is typically 10 MB/s. At this rate, downloading a single 16TB LoTSS data set completes in two weeks. At this rate, transferring 3000 data sets would take over a century.

Unlike at Leiden University, the SURFsara clusters have a gigabit connection to the data archive, accelerating the data retrieval to 1.5 days. Additionally, having two orders of magnitude more processing nodes than at Leiden, the reduction can be further parallelized. This has accelerated the processing of one (downloaded) data set from more than two days to less than half a day.

Using intermediate storage to hold the results from each step, the pre-FACTOR DI pipeline (Fig.3.1) was split into four steps as in Fig. 3.2. The Calib

[1] http://docs.surfsaralabs.nl/projects/grid/en/latest/Pages/Service/system_specs.html

1 and Target 1 steps download the raw data at one piece per worker machine and store the processing results (Calibration Tables and Processed data respectively) in storage. The second Calibration and Target steps combine these results and process them producing the calibration solutions and data sets respectively.

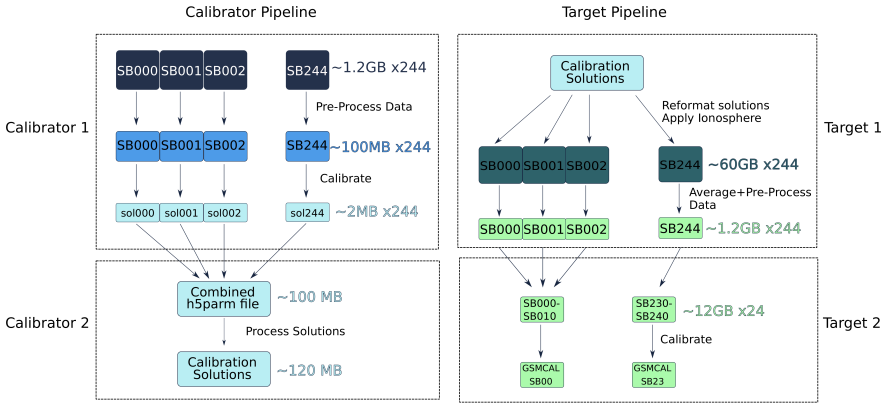


Figure 3.2: Data flow and parallelization of the Direction Independent Processing. The Calibrator 1 and Target 1 steps run concurrently as independent jobs. Calibrator 2 and Target 2 combine these results. Note that the Target 1 step requires the solutions produced by the Calibrator 2 step. This places a strict ordering on the processing steps.

3.4 Framework Design

The LRT framework (Fig. 3.3) was developed to automate the LOFAR Direction Independent calibration by processing the data at the Gina cluster SURFsara[40]. The goal of the framework was to adapt the pre-FACTOR package to take advantage of the large computational resources and high bandwidth at this site. Thanks to the data-level parallelism, using the computational resources at the Gina cluster accelerates the data reduction.

3.4.1 Framework Elements

The LRT framework consists of a set of modules responsible for different parts of the data reduction. The `srmlist` module handles the links to the data. If the data is on tape, it sends a command to stage it to disk. The `sandbox` module creates an archive of processing scripts and uploads it to storage. The `Token` module is responsible for managing metadata, which defines a processing job. Appendix 3.A contains information on the functionality of each module and their use.

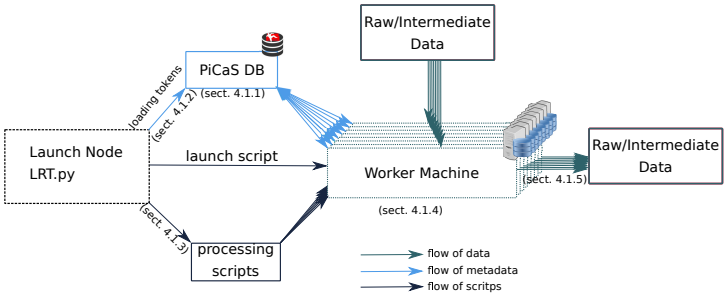


Figure 3.3: Overview of the design of the LRT framework

Storing Job Metadata

The LRT framework is effective for pipelines that execute the same processing steps on a large data set split across many machines. Each part of the data set is processed on a single machine, and the metadata of this job is stored in a remote database which can be read from and written to by the worker machine. By using a concurrent document oriented database such as CouchDB[5], each document can store the metadata regarding a single processing job. This is not possible with relational databases such as MySQL. These documents are called Tokens, as defined by the PiCaS framework[91].

The first implementation of PiCaS and CouchDB for LOFAR data reduction was carried by J. B. R. Oonk and N. Danezi in the context of the LOFAR spectroscopy project and custom user processing (Oonk et al. in prep). This first implementation focused on processing individual data sets and required a high-level of user interaction. Here we extend this implementation to automatically handle and connect multiple runs (calibrator and target) and their products.

By logging the name of the pipeline step in the job description, each job is aware of which pipeline step it is processing and executes the appropriate scripts. Important to note is that the CouchDB documents can store text and integer values as well as file attachments. The LOFAR implementation uses attachments to store diagnostic files produced by the worker machines, lists of links to the data and parset files that define the pre-FACTOR workflow.

Creating Job Tokens

The first step to automating batch processing is to create the job tokens which hold the processing metadata and load them into the database. In order to track multiple

concurrent reductions, these tokens are combined in sets. Once this set is given a name, a batch of tokens can be created for each pipeline step and uploaded into the PiCaS server. These tokens are set in the 'todo' state, indicating the processing has not yet started. The specific implementation of the 'pre-FACTOR' software is discussed in Appendix 3.A.

Packing Scripts

While the PiCaS database can hold metadata which differs across jobs, pipelines or observations; the processing scripts need to be stored in a location where the worker machines have access to them. These scripts are archived and uploaded to storage and their location is added to the job token. After a worker machine locks a job token, it downloads and extracts the processing scripts, reads the metadata from the token and begins the processing (Fig.3.4).

The location of the scripts is stored in the job token. This allows different steps of the pipeline to use different sets of scripts. The benefits from this design is that as long as the worker node has access to the URI (Universal Resource Identifier) of the scripts, it can process the data, making data reduction portable over a variety of distributed computing environments, including the GRID. This capability will be used in the future to move processing to the location of the archive (Section 3.5.2). Implementation of this process is summarized in Appendix 3.A.

Processing

Processing is launched with a launch script executed on a worker node. This script is responsible for locking a job token, taking it from the 'todo' state to the 'locked' state. After the token is locked, the launch script downloads the script archive and launches the processing. For the LOFAR pre-FACTOR software, the scripts include the latest version of the pre-FACTOR repository. Additionally, there are helper scripts that set up the processing environment, download the data, post-process the output and upload the data to intermediate storage. The metadata stored in the PiCaS job token is fed into the setup and processing scripts.

Since the processing scripts and metadata are stored remotely, the same small launch script can load many different reduction pipelines simply by changing the group of tokens it should lock. This design choice makes it easy to create other script archives for the Direction Dependent pipeline or other LOFAR processing workflows.

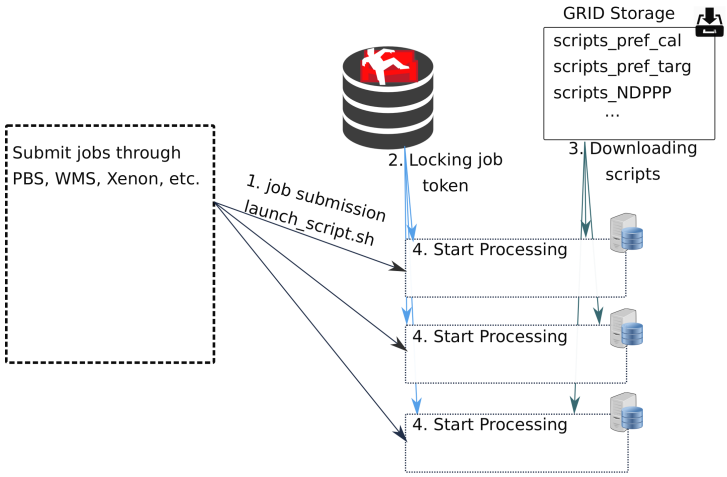


Figure 3.4: Starting processing on worker machines. Currently processing is done on SURFsara Gina nodes, however the framework has been tested at the Leiden University cluster.

Intermediate Data Storage

Splitting the processing into multiple steps requires intermediate data to be stored at a location accessible to the worker machines. As the current processing is done at the Gina cluster, the intermediate results are stored in several dedicated storage pools hosted by SURFsara. The LRT processing scripts check for the availability of an initial data set or intermediate data product on launch and download it. This avoids unnecessary repetition of reduction steps and allows to restarting a failed job.

Since the data location is static, the paths of the data are hard coded into the scripts. The framework design, however, also allows a job to log the location of its output data into the CouchDB database. Jobs in subsequent steps can read the location of their input data from the tokens of the previous job. This will be implemented when the LOFAR data processing becomes distributed across multiple locations.

3.4.2 LOFAR Surveys Use Case

The LOFAR Two Meters Sky Survey requires processing of more than 8 PB of data each year in order to keep up with the data produced by the telescope. As there are more than 3000 observations planned, processing them manually is untenable. Additionally, the large raw data sizes require the data be reduced in parallel before

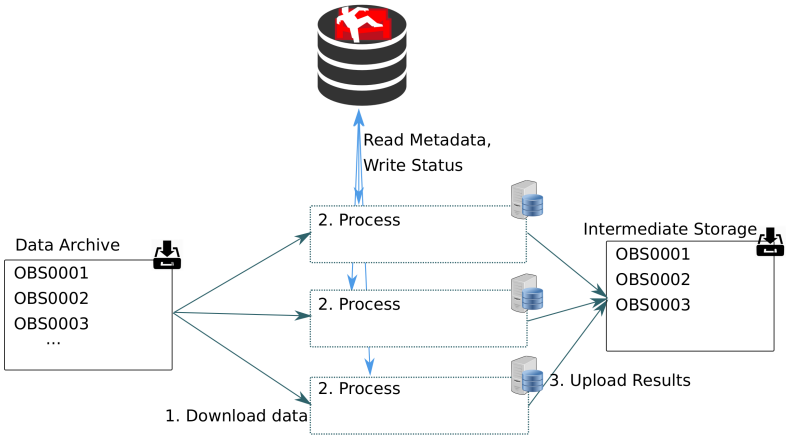


Figure 3.5: Processing of LOFAR data from the Long Term Archive with results stored at an intermediate storage location.

the Direction Dependent calibration step since the data will not fit in the memory of a single machine.

Re-purposing the LOFAR pre-FACTOR software to take advantage of the GRID computing resources by leveraging the automation provided by the LRT framework has allowed the processing of more than a hundred data sets in the time span of four months. Additionally, this was done in the time frame that an astronomer would normally produce less than 10 Direction Independent calibrated observations, hence a providing a necessary speed up by an order of magnitude.

3.4.3 Framework Capabilities

The LRT implementation is designed to be as platform independent as possible. It allows for easy extensions enabling LOFAR reduction schemes other than the LoTSS reduction. Two examples of this are the updated LOFAR GRID spectroscopy and LOFAR GRID pre-processing pipelines (Oonk+ in prep). The GRID pre-processing pipeline runs flagging of bad data and averaging, however performs no calibration. The spectroscopy pipeline uses an independent set of scripts to perform calibration, bandwidth correction and imaging.

Thanks to the abstraction of the metadata and scripts storage, processing is also possible at other locations. The pre-FACTOR scripts require an installation of the LOFAR software stack[64]. These requirements are met by mounting a CernVM-FS[1][114] installation of the LOFAR stack. The CERN-VM FileSystem

service provides a portable pre-compiled copy of the LOFAR software. With the CVMFS prerequisite satisfied and an active grid proxy, any computer can download data and process a job in any data reduction step.

3.4.4 Initial Results

Automatically launching jobs has made it possible to process more than 100 data sets between November 2016 and February 2017. Without a framework to automate and distribute the processing and a cluster at a Grid location, these data sets would need to be downloaded to an institute's cluster. Such standalone runs of the 'pre-FACTOR' scripts typically process one observation in two weeks taking into account the data transfer time. At the 10MB/s connection (The sustained speed at Leiden University), the downloading would take over 5 years alone. Porting the LOFAR LoTSS data reduction to a Grid location using the LRT framework has resulted in a 15x increase in data throughput. Suggestions on further increasing the throughput are presented in Section 3.5.2.

3.5 Conclusion and Future Work

The goal of the LRT Framework is to create an automated software which can be used to port LOFAR processing to a massively distributed compute environment. The Direction Independent calibration of the LOFAR Two Meter Sky Survey was used as a demonstration of the capabilities of the LRT software.

Combining the 'pre-FACTOR' scripts with the LRT tools resulted in a 15x increase in throughput compared to previous data reduction strategies. Thanks to the automation provided, it is possible to process hundreds of observations, necessary for large astronomical survey projects such as the 3000+ observations of the LoTSS.

The scalability of this framework allows to launch multiple data reductions concurrently and easily monitor their progress. The portability of the LRT framework makes it easy to move processing to archive locations, further increasing the throughput. Finally, as the framework is general, other LOFAR projects can increase throughput and automation by integrating their software.

3.5.1 Throughput Improvement

Using the LRT framework, more than 100 data sets have passed through the Direction Independent calibration. This is more than a 15 fold increase compared to the throughput at a institution with a limited bandwidth to the data archive. From these data sets, 30 images have been produced since November 2016. The Direction Dependent pipeline, responsible for the imaging of the DI calibrated data, is not yet automated using the LRT framework. The topic of this automation will be handled in future work. Future improvements (Section 3.5.2) are expected to increase the throughput to one image per day.

Currently, the data reduction is launched manually. There are upcoming plans to create a trigger launched by the Observatory at the end of a successful observation. Using this trigger, the processing can be integrated with the data acquisition and launched automatically at the end of the observation. Doing so enables producing an image less than a week after the observation has completed without requiring human interaction.

3.5.2 Future Work

Most of the LoTSS data is not stored at the SURFsara Grid location. Because of the high data sizes, the 1 Gbps transfer between these remote sites and SURFsara is insufficient to process the 3000+ data sets. The proposed solution is to launch the initial reduction steps (Calibrator 1 and Target 1 in Fig.3.2) on compute clusters at the archive locations. Running the Target 1 step at these sites will reduce the data size from 16TB to 0.5TB. The resulting intermediate data can easily be transferred over a 1 Gbps connection in 1 hour compared with the 35 hours for the original data.

While the LRT framework successfully automated the Direction Independent calibration pipeline, it still needs to implement the Direction Dependent processing scripts. The DI data can be easily split into pieces and processed concurrently, however current Direction Dependent pipelines cannot split the data easily. This means they will only benefit from the automation aspect of the framework. Because of this limitation of the algorithms, (Direction Dependent) processing of each data set needs to be done on a single machine. This part of the processing currently takes more than four days per data set per node. To process all 3000 observations within five years, the DD reduction will need at least 8 dedicated machines continuously processing data. Nevertheless, the input data is only 200-500GB making it easier to store and transfer to institutes not part of the European Grid Infrastructure.

The Xenon framework[67] allows launching jobs at multiple clusters from a single location. Thanks to the portability of the LRT software, it will be possible to integrate the LOFAR reduction with Xenon. This will automate the launching of Direction Dependent processing at multiple institutions. Automatically launching these jobs will make efficient use of the computer resources at SURFsara and other sites. Using this strategy, the LoTSS project data can be processed within the anticipated time span.

Finally, as the reduction is automated, it can be started right after the telescope finishes the observation. Launching jobs immediately after an observation will minimize the time staging the data from tape to disk. Currently, the staging process can take up to a week. Triggering the processing immediately after observation, an image will be produced less than a week after the data is acquired.

Minimizing the latency between observation and science quality images will benefit the LOFAR community immensely by allowing radio astronomers to focus on their specific science case. An all-sky survey at the 150 MHz range will create a multitude of targets for follow-up with optical telescopes and result in many discoveries in the field of Radio Astronomy. A full list of science results expected from the LoTSS project can be found in[109]. Efficient high-throughput processing of LOFAR data will empower the above science cases opening the way to exciting new discoveries.

3.A Execution of LOFAR Reduction Tools

The LRT framework handles staging of LOFAR data, packaging and uploading worker node scripts (named sandboxes), creating PiCaS job tokens and launching pilot jobs on the SURFsara Gina cluster. The framework is modular, allowing an user to execute any of the previous steps manually, or alternatively launch an automated reduction. It can be downloaded from the Github page (github.com/apmechev/GRID_LRT) and installed with
`python setup.py build && python setup.py install` . Documentation on the usage of the tools can be found at the Github page.

4 *Pipeline Collector*: Gathering performance data for distributed astronomical pipelines

The contents of this chapter are based on a manuscript Published in Astronomy and Computing in 2018

Original Abstract: Modern astronomical data processing requires complex software pipelines to process ever growing data sets. For radio astronomy, these pipelines have become so large that they need to be distributed across a computational cluster. This makes it difficult to monitor the performance of each pipeline step. To gain insight into the performance of each step, a performance monitoring utility needs to be integrated with the pipeline execution. In this work we have developed such a utility and integrated it with the calibration pipeline of the Low Frequency Array, LOFAR, a leading radio telescope. We tested the tool by running the pipeline on several different compute platforms and collected the performance data. Based on this data, we make well informed recommendations on future hardware and software upgrades. The aim of these upgrades is to accelerate the slowest processing steps for this LOFAR pipeline. The *pipeline_collector* suite is open source and will be incorporated in future LOFAR pipelines to create a performance database for all LOFAR processing.

4.1 Introduction

Astronomical data often requires significant processing before it is considered ready for scientific analysis. This processing is done increasingly by complex and au-

tonomous software pipelines, often consisting of numerous processing steps, which are run without user interaction. It is necessary to collect performance statistics for each pipeline step. Doing so will enable scientists to discover and address software and hardware inefficiencies and produce scientific data at a higher rate. To identify these inefficiencies, we have extended the performance monitoring package *tcollector*[1][7]. The resulting suite, *pipeline_collector*, makes it possible to use *tcollector* to record data for complex pipelines. We have used a leading radio telescope as the test case for the *pipeline_collector* suite. The discoveries made with our software will help remove bottlenecks and suggest hardware requirements for current and future processing clusters. We summarize our findings in Table 4.1 in Section 4.3.

Over the past two decades, processing data in radio astronomy has increasingly moved from personal machines to large compute clusters. Over this time, radio telescopes have undergone upgrades in the form of wide band receivers and upgraded correlators [18, 43]. In addition, several aperture synthesis arrays such as the Low Frequency Array [LOFAR, 130], Murchison Widefield Array [MWA 65, 125] and MeerKAT [50] have begun observing the radio sky, leading to an increase of data rates by up to 3 orders of magnitude [28, 143].

As the data acquisition rate has increased, data size has entered the Petabyte regime, and processing requirements increased to millions of CPU-hours. In order for processing to match the acquisition rate, the data is increasingly processed at large clusters with high-bandwidth connections to the data. An important case where data processing is done at a high throughput cluster is the LOFAR radio telescope.

The LOFAR telescope is a European low frequency aperture synthesis radio telescope centered in the Netherlands with stations stretching across Europe. This aperture synthesis telescope requires significant data processing before producing scientific images [87, 112, 134, 141]. In this work, we will use our performance monitoring utility, *pipeline_collector*[2], to study the first half of the LOFAR processing, the Direction Independent (hereafter DI) pipeline.

One major project for the LOFAR telescope is the Surveys Key Science Project (SKSP) [109]. This project consists of more than 3000 observations of 8 hours each, 600 of which have been observed. These observations need to be processed by a DI pipeline, the results of which are calibrated by a Direction Dependent (DD) pipeline. The DI pipeline is implemented in the software package *prefactor*[3]. The *prefactor* pipeline is itself split into four stages and implemented at the SURFsara

[1] https://github.com/OpenTSDB/tcollector
[2] https://gitlab.com/apmechev/pipeline_collector.git
[3] available at `https://github.com/lofar-astron/prefactor`

Grid location at the Amsterdam e-Science centre [75, 116]. The automation and simple parallelization has decreased the run time per data set from several days to six hours, making it comparable to the observation rate. To better understand and optimize the performance of the *prefactor* pipeline, we require detailed performance information for all steps of the processing software. We have developed a utility to gather this information for data processing pipelines running on distributed compute systems.

In this work, we will use the *pipeline_collector* utility to study the LOFAR *prefactor* pipeline and suggest optimization based on our results. To test the software on a diverse set of hardware, we will set up the monitoring package on four different computers and collect data on the pipeline's performance. Using this data, we discuss several aspects of the LOFAR software which we present in Table 4.1. Finally we discuss the broader context of these optimizations in relation to the LOFAR SKSP project and touch on the integration of *pipeline_collector* with the second half of the data processing pipeline, the DD calibration and imaging.

Result #	Description
R1	Native compilation of the software performs comparably to pre-compiled binaries on two test machines.
R2	The processing steps do not appear to accelerate significantly on a faster processor or with larger cache size.
R3	Both calibration steps (*calib_cal* and *gsmcal_solve*) show linear correlation between speedup and memory bandwidth.
R4	Disk read/write speed does not affect the completion time of the slowest steps.
R5	Both calibration steps do not use large amounts of RAM despite processing data on the order of Gigabytes.
R6	The *calib_cal* step can suffer up to 20% of Level 1 Instruction Cache misses, while gsmcal only has 5% of these misses.
R7	Both calibration steps are impacted by Level 2 Cache eviction at comparable rates.
R8	The *calib_cal* step stalls on resources 70% of cycles while the gsmcal step only 30% of them.
R9	The *calib_cal* uses the CPU at full efficiency for only 10 % of the CPU cycles.

Table 4.1: A table of all the results presented in Section 4.3.

4.1.1 Related Work

Scientific fields that need to process large data sets employ some type of data processing pipelines. Such pipelines include e.g. solar imaging [23], neuroscience imaging

[115] and infrared astronomy [89]. While these pipelines often log the start and finishing times of each step (using tools such as pegasus-kickstart [138]), they do not collect detailed time series performance data throughout the run.

At a typical compute cluster the performance of every node in a distributed systems is monitored using utilities, such as Ganglia [69]. These tools only monitor the global system performance. If one is interested in specific processes, then the Linux procfs [16] is used. The procfs system can be used to analyze the performance of individual pipeline steps. Likewise, the Performance API [PAPI, 81] is a tool which collects detailed low level information on the CPU usage per process. Collecting detailed statistics at the process level is required to understand and optimize the performance of the LOFAR pipeline and we will integrate PAPI into *pipeline_collector* in the future. Finally, DTrace[42] is a Sun Microsystems tool which makes it possible to write profiling scripts that access data from the kernel and can be used to monitor process or system performance at run time with minimal overhead. As DTrace was not installed on either of the processing clusters, we have not used it to monitor the pipeline's performance.

The Linux procfs system and PAPI record data which is already made available by the Linux kernel. This option incurs insignificant overhead as it uses data the kernel and processor already log. Likewise PAPI reads performance counters that the CPU automatically increments during processing. These profiling utilities can run concurrently with the scientific payload without using more than 1-2% of system resources. Their low overhead is why we choose to use them to collect performance data.

Other tools for performance analysis such as Valgrind [82] collect very detailed performance information. This comes at the expense of execution time: running with Valgrind, the processing time slows by up to two orders of magnitude. As such, we do not use Valgrind along the LOFAR software.

4.2 Measuring LOFAR Pipeline performance with pipeline_collector

We developed the package *pipeline_collector* as an extension of the performance collection package *tcollector*. *pipeline_collector* makes it possible to collect performance data for complex multi-step pipelines. Additionally, it makes it easy to record performance data from other utilities. A performance monitoring utility that we plan

to integrate in the future are the PAPI tools described in section 4.1.1. The resulting performance data was recorded in a database and analyzed. For our tests, we used the LOFAR *prefactor* pipeline, however with minor modifications, any multi-step pipeline can be profiled.

tcollector is a software package that automatically launches 'collector' scripts. These scripts are sample the specific system resource and send the data to the main tcollector process. This process then sends the data to the dedicated time series database. We created custom scripts to monitor processes launched by the *prefactor* pipeline (4.A.1).

In this work, we use a sample LOFAR SKSP data set as a test case. A particular focus was to understand the effect of hardware on the bottlenecks of the LOFAR data reduction. To gain insight into the effect of hardware on *prefactor* performance, the data was processed on four different hardware configurations (Table 4.2). As typical upgrade cycle for cluster hardware is five years, our results will be used to select optimal hardware for future clusters tasked with LOFAR processing.

4.2.1 *Prefactor* Pipeline

The LOFAR *prefactor* pipeline [134] is a software pipeline that performs direction independent calibration using the LOFAR software. The LOFAR software stack is a software package containing commonly used processing software used by LOFAR pipelines [63, 85]. These tools are built and maintained by ASTRON[4].

The *prefactor* pipeline performs a sequence of four stages, namely the calibrator and target calibration. The first half of *prefactor* processes data from a calibration source and the second half processes a science target. Altogether, this processing takes six hours on a high-throughput cluster. The final result is a data-set ready for creating images of the sky at radio wavelengths. Figure 4.1 shows a graphical view of the prefactor pipeline's Calibrator and Target stages.

The Calibrator stage consists of the *ndppp_prep_cal* and the *calib_cal* step. The former flags radio interference and averages the data, and the latter performs gain calibration on a bright calibration source. It is followed by the *fitclock* step which fits a clock-TEC model to the calibration solutions [134].

The Target stage consists of a *ndppp_prep_targ* step, *predict_ateam*, *gsmcal_solve* and *gsmcal_apply* steps. The first two of these steps flag and average the target data and calculate contamination by bright off-axis radio sources. The *gsmcal_solve*

[4] ASTRON: Netherlands Institute for Radio Astronomy, urlhttps://www.astron.nl/

step determines phase solutions for each antenna using a model of the target and the results of the *ndppp_prep_targ* step. Finally, the *gsmcal_apply* step applies these solutions to the target data. Figure 4.2 shows the percentage of time spent by these steps for the four *prefactor* stages.

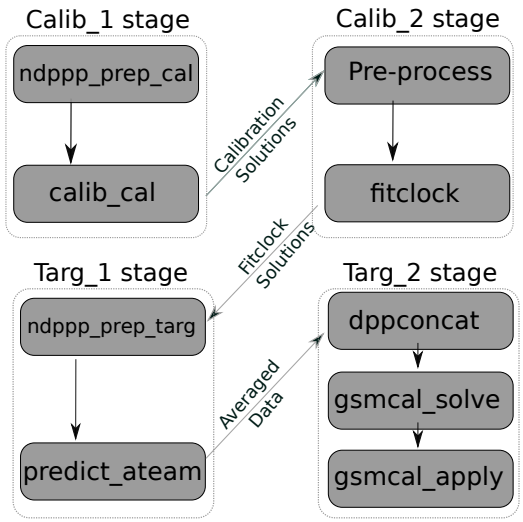


Figure 4.1: The four processing stages that make up the prefactor pipeline. The Calibrator stages (top) process a known bright calibrator to obtain the gain for the LOFAR antennas. The Target stages (bottom) process the scientific observation to remove Direction Independent effects. The *pref_cal1* and *pref_targ1* stages are massively parallelized across nodes without the need of an interconnect. The *pref_cal2* step runs only on one node, while *pref_targ2* is parallelized on 25 nodes. As the LOFAR software does not use MPI, we can run each processing job in isolation.

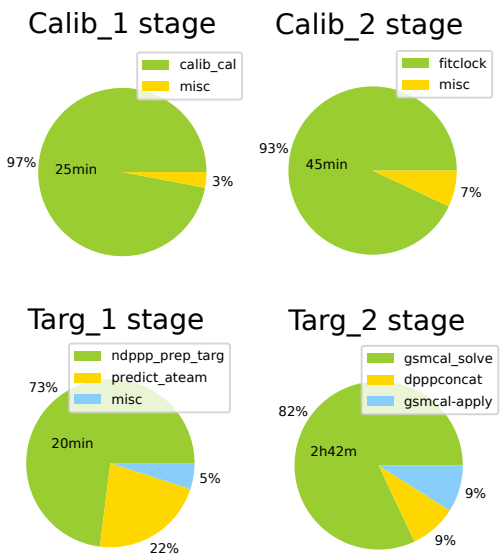


Figure 4.2: Portion of processing time taken by each step for the four prefactor stages, as reported by the Prefactor software. For each stage, the majority of processing time is spent during one or two steps. This is due to the fact that each prefactor stage also has intermediate book-keeping steps explicitly included in the pipeline. For each pipeline stage, the mean processing time for the longest running steps at SURFsara is also indicated. It should be noted that while faster, pref_cal1 runs 10 times as many jobs as pref_targ2.

4.2.2 Performance suite

Cluster performance is frequently monitored using utilities such as Ganglia [69, discussed in Section 4.1.1]. These tools cannot access individual processes and thus cannot collect data on a per-process basis. To collect such data, each process launched by the active pipeline step needs to be profiled individually. Our utility is designed to gather such performance data.

Our monitoring package, *pipeline_collector* adds custom performance collectors (4.A.1) to the performance collection framework *tcollector*. We use these collectors to monitor individual pipeline steps as defined by the user[5]. The tools attach to processes launched by the pipeline and record performance data at a one second interval. This sampling frequency is at high enough resolution to detect trends and anomalies in hardware utilization, and still result in a reasonable database size. The performance data is uploaded to a remote time series database, OpenTSDB [110]. Details on the data collection can be found in 4.A.

[5]https://gitlab.com/apmechev/pipeline_collector.git

Location	CPU Speed (MHz)	CPU Model	Micro-architecture	Cache Size	RAM Speed[7]	Disk Speed[8]
Leiden	2200	E5-4620	Sandy Bridge	16 MB	1.4 GB/s	99.7 MB/s
SURFsara	2500	E5-2680	Sandy Bridge	30 MB	2.5 GB/s	65.4 MB/s
Hatfield	2900	E5-2660	Sandy Bridge	20 MB	2.4 GB/s	155 MB/s
Laptop	3300	E3-1505M	Skylake	8 MB	4.7 GB/s	822 MB/s

Table 4.2: CPU, Cache, RAM and Storage specifications of the four test machines. The tested machines span a factor of 1.5x in CPU speed, 4x in cache and RAM Speed and 10x in Disk speed.

Performance API

The time-series database is also used to collect low-level CPU information for each process. This information is collected by the PAPI interface (discussed in Section 4.1.1). This was done through the papiex utility[6] [2]. This utility records the CPU's internal performance counters. A CPU's performance counters record information on how efficiently the software uses the CPU's resources. The results from this test are detailed in Section 4.4.

4.2.3 Test Hardware

In order to study the effect of different hardware configurations on the performance of LOFAR processing, the *prefactor* pipeline was run on four different sets of hardware. The four machines tasked with processing LOFAR data comprised nodes at three computational clusters and a personal computer. The tests were run while the systems were idle to make sure there is no interference of other software with the LOFAR processing. Table 4.2 details the specifics of the four test machines.

4.3 LOFAR Prefactor Test Case

With the test set described in Section 4.2, we aim to understand processing bottlenecks in the *prefactor* pipeline and make informed decisions on future hardware and software upgrades. To do so, we processed a sample observation at institutes that typically process LOFAR data.

From the data collected by processing the sample observation, we determined the slowest pipeline steps. These steps were the *calib_cal* and *gsmcal_solve*, seen in Figure 4.2. The *calib_cal* step is implemented by the software `bbs-reducer` [63,

[6] Available at https://bitbucket.org/minimalmetrics/papiex-oss

66] and the *gsmcal_solve* step is implemented by `NDPPP` [63, 85]. Both `bbs-reducer` and `NDPPP` are part of the LOFAR software suite.

We collected performance statistics using the *pipeline_collector* suite as discussed in Section 4.2. The run time of the slowest *prefactor* steps on the four machines is shown in figure 4.3. The results discovered using *pipeline_collector* are listed in Table 4.1 and discussed in Section 4.3.2. Using the PAPI interface (discussed in Section 4.2.2) CPU performance data was collected. The results from this test are detailed in Section 4.4.

We will present a number of insights into the performance of the LOFAR software collected by the profiling suite. The results are presented in Table 4.1 and are grouped in three main areas. The effect of compilation on the run time was result **R1**. The set of results **R2**, **R3**, **R4** and **R5** were obtained using the *pipeline_collector* package. Results **R6**, **R7**, **R8** and **R9** were collected with the PAPI package, which will be integrated into *pipeline_collector* in the future.

4.3.1 Pre-compiled vs native compilation

The performance trade-off between pre-compiled and native compilation was studied first. The majority of the processing for the LOFAR SKSP Project [109] is done at the SURFsara GINA cluster in Amsterdam. This location is part of the European Grid Initiative (EGI)[116]. At this location, software is deployed by compiling on a virtual machine and mounting it on all worker nodes through the CernVM FileSystem (CVMFS) service [13]. The CVMFS server allows any client to mount a fully compiled LOFAR installation, making it easy to distribute and version control the software within and outside of SURFsara. An alternative is to locally compile the LOFAR packages on each cluster. The performance of the natively compiled[9] vs CVMFS installations was compared on the laptop test machine using *pipeline_collector*. In order to validate this result, the two compilations of the same software were also tested at the Data Science Lab at the Leiden Institute of Advanced Computer Science (LIACS)[10].

An interesting discovery is that the LOFAR software did not process data faster when compiled natively. This is despite the fact that the local install was com-

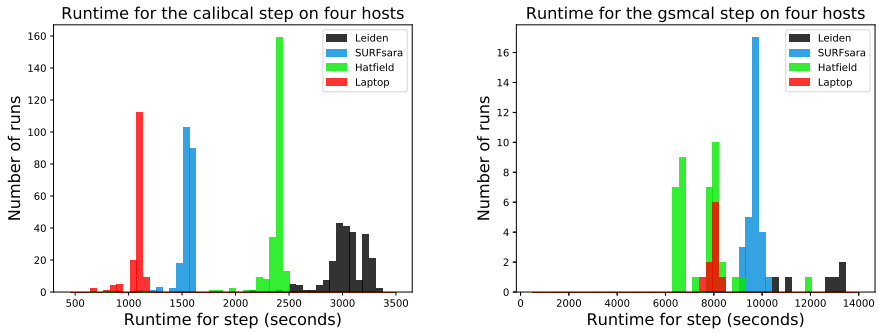
[7] benchmarked using *dd*

[8] sequential disk read, benchmarked using `fio` - flexible I/O tester:

```
fio -randrepeat=1 -ioengine=libaio -direct=1 -gtod_reduce=1 -name=test -filename=test
-bs=4k -iodepth=256 -size=4G -readwrite=read -ramp_time=4
```

[9] The software was compiled using `-march=native` and `-O3` compilation flags. On the laptop, gcc resolves `-march=native` as `broadwell`. The CVMFS installation resolves `-march=native` as `core-avx-i`.

[10] https://www.universiteitleiden.nl/en/science/computer-science/about-us/our-facilities



(a) calib_cal (b) *gsmcal_solve*

Figure 4.3: Job completion times for *calib_cal* and *gsmcal_solve* steps tested on four hardware setups. The *calib_cal* step ran 244 times. The *gsmcal_solve* ran 24 times as the data is concatenated from 244x1 to 24x10 sets. The step with the longest latency is *gsmcal_solve* while *calib_cal* consumes a comparable number of core-hours over 244 jobs.

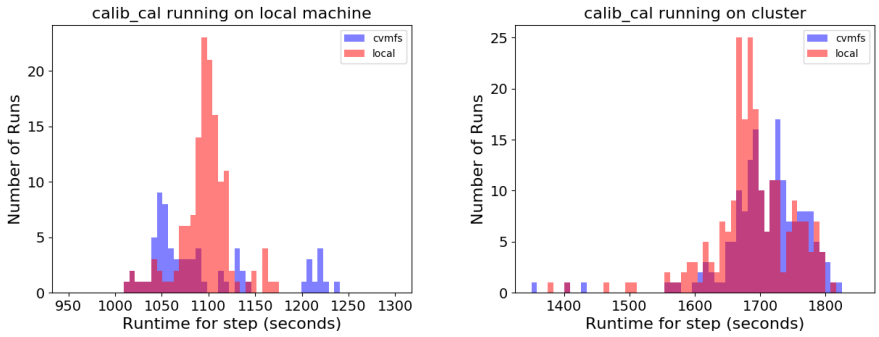
piled with advanced processor instructions available on the host machine. Figure 4.4 shows a histogram of its processing time with the two different compilation options for the *calib_cal* software running on the sample data set. The same test was done for the software performing the gain calibration (*gsmcal_solve*), seen in Figure 4.5. The result of this experiment is shown in Figures 4.4a and 4.5a. The software compiled at SURFsara showed a minor improvement for the *calib_cal* step on the laptop machine, however this improvement is not seen on the computational cluster node.

Overall, the software for both steps show no significant improvement when compiled natively. This is result **R1** in Table 4.1. The second run at LIACS also confirms this result for both steps (Figures 4.4b and 4.5b). This result suggests that the slowest *prefactor* steps are not optimized for modern processors.

4.3.2 Prefactor Run time and Hardware Parameters

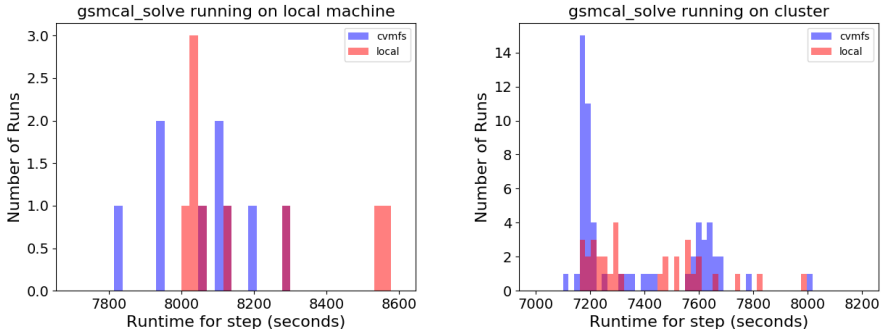
Next, we studied the dependence of run time on different hardware parameters. With software that collects per-step performance statistics for the LOFAR pipeline, the dependence of the pipeline processing on hardware performance can be easily profiled and studied. Using *pipeline_collector* we determined the pipeline's slowest steps with respect to different hardware parameters.

The system parameters studied here are the CPU speed, memory throughput, cache size and disk speed. Modern computers can have a complex memory hierarchy as demonstrated in Figure 4.6 [53]. This is due to the cost trade-off between



(a) Two compilation options on a Laptop (b) Two compilation options on cluster node

Figure 4.4: Difference in processing time for *calib_cal* when compiled remotely and natively. *calib_cal* was run 244 times with the native software and 40 times with the CVMFS compilation. Two tests were done, one on the personal laptop (4.4a) and one on a cluster node at the LIACS Data Science Lab (4.4b). The test on a cluster node shows no significant difference in run time between compilation options. The laptop test suggests that the remotely compiled software may run 5% faster than the local compilation.



(a) Two compilation options on a Laptop (b) Two compilation options on cluster node

Figure 4.5: Difference in processing time for *gsmcal_solve* when compiled remotely and natively. *gsmcal_solve* was run 50 times with the native software and 120 times with the CVMFS compilation. Two tests were done, one on the personal laptop (4.5a) and one on a cluster node at the LIACS Data Science Lab (4.5b). Just like with the *calib_cal* step, the *gsmcal_solve* step also doesn't accelerate significantly when natively compiled.

memory size and memory speed. Because of this trade-off, the full data set is stored on disk, while the working set is placed in RAM. This is the data that the processor needs to access at the current time [32]. The most frequently accessed parts of the data are stored in the CPU cache, which evicts the oldest data when full [44].

The CPU processing speed is faster than the RAM latency, so a hierarchy of caches exist. Caches store small subsets of the working set and have a fast connection to the processor. The fastest data link is between the CPU and the L1 Cache, with the link to RAM being slower and the disk read speed slower still. The limited memory capacity of the different levels of the memory hierarchy as well as the throughput between them will lead to performance bottlenecks. These bottlenecks will lead to the processor waiting on memory. Such stalls lead to longer processing times.

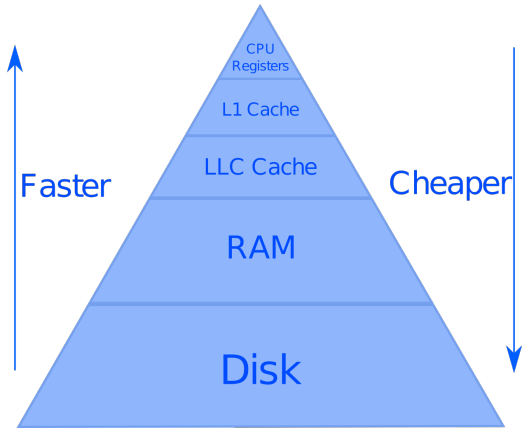
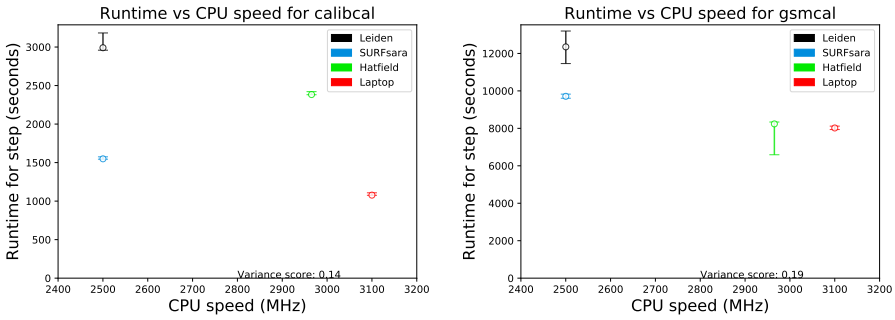


Figure 4.6: A model of the memory hierarchy, as described in [41].

CPU

The CPU speed is usually the primary factor determining how fast computations can be made. In general, a faster CPU will result in faster data processing.

However, Fig. 4.7a shows that the run time of the calibration of the calibrator does not strongly depend on the CPU frequency. While the test nodes at SURFsara and Leiden run at the same CPU frequency, running on a cluster node at SURFsara takes half the time as on a node at Leiden. Even more surprisingly, the *gsmcal_solve* step does not benefit significantly from a faster CPU, despite being the most computationally heavy *prefactor* step (**R2**). This step does the gain calibration on the target field using the StEFCal algorithm [100]. Figure 4.7b shows only a slight



(a) calib_cal (b) *gsmcal_solve*

Figure 4.7: Performance of the bottleneck steps compared with the CPU speeds of the four test machines. The values are the mean of 244 runs (Standard prefactor run) and the error bars show the 1-sigma of the distribution of the run time.

improvement over faster CPU clock speeds for both steps. The correlation between completion time and CPU speed is similar for both steps.

4

Cache

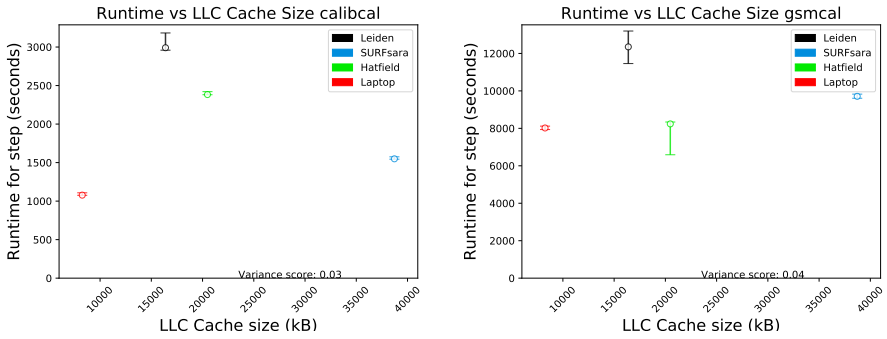
The CPU has a hierarchy of caches consisting of Level 1, Level 2 Cache and LLC Cache. For the four processors tested, the Level 1 and 2 caches were all the same size, thus the only difference is the Last Level Cache (LLC or just Cache in Figure 4.6). This cache stores data needed by the CPU, so the larger it is, the less the processor needs to wait for RAM to return data.

In general, numerical codes benefit from larger cache sizes [41, 111]. Interestingly, figure 4.8b suggests that the *gsmcal_solve* step does not exclusively depend on larger cache **R3** (Table 4.1). On the machines with a larger cache, the *gsmcal_solve* step completed processing as quickly as on the machines with smaller cache, even down to 8MB.

RAM Bandwidth

If the entire data set does not fit into cache, the software needs to transfer data from RAM to the CPU. In these cases, *prefactor* benefits from a fast bandwidth between the cache and RAM. For this study, the RAM throughput was benchmarked[11]. This

[11] Using the command `$> dd if=/dev/zero of=/dev/shm/test bs=1M count=2048`



(a) calib_cal (b) *gsmcal_solve*

Figure 4.8: Performance of the two bottleneck steps with respect to Last Level Cache size. The *gsmcal_solve* step shows no trend between cache size and completion time. The *calib_cal* step runs the fastest on the machine with the smallest cache.

command copies dummy data into system memory. As this utility exists on all Unix systems, this is a standardized benchmark of the RAM performance.

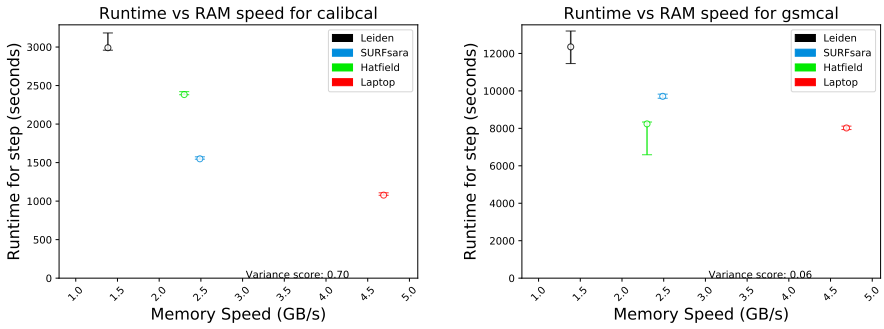
Figure 4.9a showed that higher bandwidth is correlated with a faster completion time for the *calib_cal* and *gsmcal_solve* steps (**R4**). The result is to be expected as the working set of these steps is 200MB and 1.0GB respectively, and cannot fit into cache readily, however it is loaded into RAM within the first 5 seconds of the run (Figure 4.10), and is streamed from memory throughout the run.

4

Disk Read speeds

The slowest link in the memory hierarchy is the disk read speed. For the *calib_cal* step, the entire data is loaded into memory during the first few seconds of the run, after which the disk only becomes important when the results need to be written out. The *gsmcal_solve* step streams data from the disk to memory throughout the entire run. The plot of disk read speeds (Fig. 4.11b) also shows that a faster disk does not speed up the slowest step **R5**. To verify that disk throughput was not the limiting factor, the entire data set (25 GB) was moved to main memory (using /dev/shm). The resulting run time for both bottleneck steps did not change.

The calibration steps both stored less than 200MB of data in memory throughout their run. Figure 4.10 shows the time-series of the total memory used by these steps. The *calib_cal* step uses only 200MB of memory and *gsmcal_solve* only 35MB. While the *gsmcal_solve* step works on a 1GB data set, it streams the data in memory and thus does not require 1GB of RAM. Alternatively, the *calib_cal* step



(a) *calib_cal* (b) *gsmcal_solve*

Figure 4.9: Performance of the two bottleneck steps and RAM bandwidth in GB/s. Both the *calib_cal* and *gsmcal_solve* steps show a trend of faster processing times on machines with higher RAM bandwidth. Both steps show a trend of decreasing processing time with increasing RAM throughput.

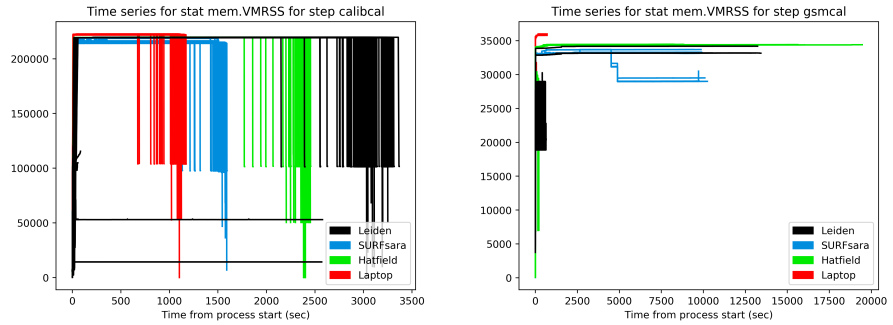
loads the entire (200MB) data set into memory for the entire duration of the run. The RAM usage time-series in Figure 4.10 show that the RAM is filled for the first 5 seconds of the run, further confirming that the processing is effectively independent from disk speed.

4.4 CPU Utilization Tests with PAPI

To gain more fine grained data on the CPU utilization, the *calib_cal* and *gsmcal_solve* steps were tested with the PAPI package. We ran this package as a test, to determine whether collecting PAPI data is helpful in understanding pipeline performance. PAPI can record data such as cache performance, branch prediction rate, fraction of memory/branch instructions and others. This data is complementary to the procfs information, which is collected by the Linux kernel. As the collected data was useful in understanding the *prefactor* pipeline, we will include PAPI in the *pipeline_collector* suite in the future. In the following sections we will discuss the results obtained for the *calib_cal* and *gsmcal_solve* steps.

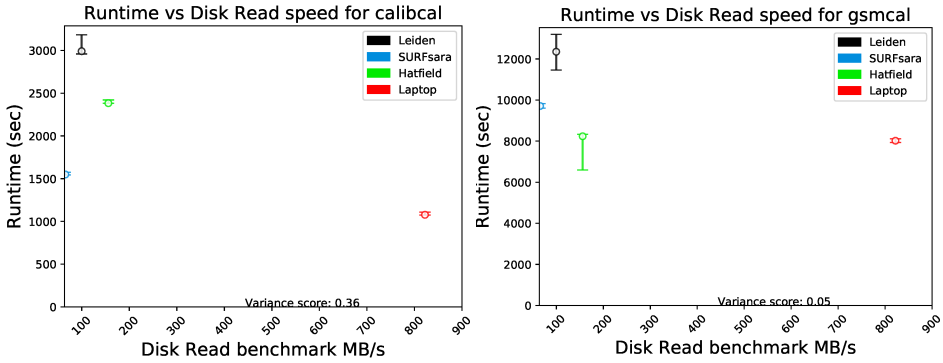
4.4.1 Level 1 Data Misses

The Level 1 Cache is split into cache for instructions and data. For all our test hardware the L1 Data cache is 32 Kb, and has a direct link to the processor's computational units [49]. The processor collects information logging how many times data



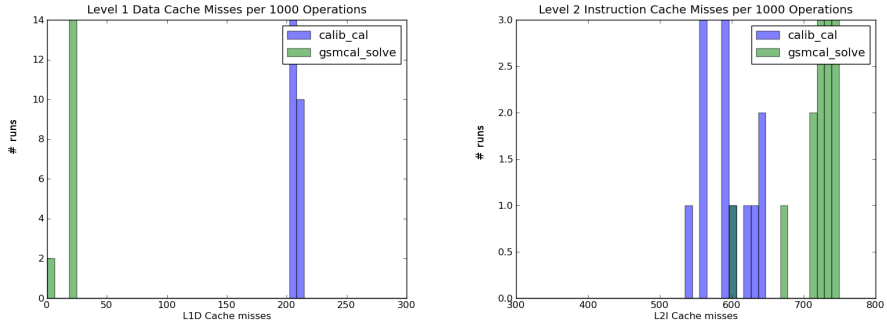
(a) *calib_cal* (b) *gsmcal_solve*

Figure 4.10: Time series of the Virtual Memory Resident Set Size. This is the amount of data stored in RAM (in Kb) during the *calib_cal* and *gsmcal_solve* steps. Both steps show the same amount of memory use on all test machines. Additionally, after a brief loading of data, the memory usage remains constant until processing is finished.



(a) calib_cal (b) *gsmcal_solve*

Figure 4.11: Performance of the two bottleneck steps and Disk bandwidth in MB/s. There is no correlation between the Disk read speed and the Run time of the steps.



(a) Level 1 Data cache misses

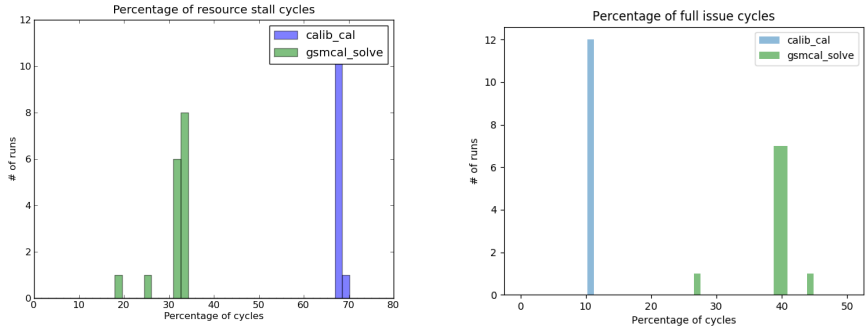
(b) Level 2 Instruction cache misses

Figure 4.12: Cache miss rates for *calib_cal* and *gsmcal_solve*, executed on the SURFsara gina cluster. The cache is split into instruction and data caches. The figures above show the difference in number of cache misses for both instruction cache and data cache for the slowest *prefactor* steps. *calib_cal* suffers significantly more Data cache misses than *gsmcal_solve* while the two steps undergo similar instruction Cache misses.

requested by the CPU is not located into the L1 Data cache. This counter is called the Level 1 Data Cache Miss rate. To resolve this type of cache miss, the data needs to be fetched from L2 Cache. When this happens, the processor has to wait for the requested data. **R7**: The recorded L1 data misses in Figure 4.12a, show that the software performing the *calib_cal* step misses 20% of its L1 data cache requests, while the software implementing the *gsmcal_solve* step misses less than 5% of L1 Cache requests. These cache misses often happens in multi-threaded applications where there are instructions shared by multiple threads on the same cache line [103].

4.4.2 Level 2 Instruction Misses

Unlike the Level 1 cache, Level 2 cache stores data and instructions in the same location. When the cache is full, it evicts the last used element in order to make space for newly requested data. PAPI also counts these eviction events. Figure 4.12b shows that for both steps, between 50 and 70% of L2 requests for an instruction do not match the contents of L2 Cache. This is significantly more than the applications benchmarked in [59, Table 2]. Because both steps process data of considerable size, the large amount of data required can evict instructions from the L2 cache (insight number **R7** in table 4.1).



(a) Resource Stall Cycles (b) Full Issue cycles

Figure 4.13: Resource stall cycles and Full Instruction Issue cycles. The two steps were executed on the SURFsara gina cluster.

4.4.3 Resource Stalls

Modern processors have multiple computational pipelines on chip, in order to process data in parallel [47]. There are times when the processor's internal pipeline needs to wait for other instructions to finish. When this happens, it flags that it has 'stalled on a resource'. These resource stall cycles are also recorded by PAPI and represented as a percentage of total cycles. From figure 4.13a, it can be seen that *calib_cal* stalls on 70% of the processor cycles, while *gsmcal_solve* only on 33% of cycles (**R8**).

The Full Issue Cycles counter indicates the percentage of processor cycles, in which the theoretical maximum number of instructions are executed. During these cycles, the software uses the CPU optimally. The full issue cycles counter (Fig. 4.13b) also shows the difference in efficiency between the *calib_cal* and *gsmcal_solve* step (**R9**), with the former only working at peak efficiency for 10% of the processor cycles.

The plots in Figures 4.13a and 4.13b indicate that the *calib_cal* step does not use the internal CPU pipelines efficiently leading to waiting on resources and sub-optimal use of the CPU's Computational Units.

4.5 Discussions and Recommendations

With an increase of data acquisition rates and data complexity in radio astronomy, it is becoming important to thoroughly understand and optimize the performance of processing pipelines. Using *pipeline_collector*, data can be collected for each pipeline step without altering the processing software. We store this data in a time-series database.

The collected data can be studied to help researchers understand the pipeline performance for different processing parameters, data sets, and on different hardware. The *pipeline_collector* suite is easy to deploy for mature pipelines and has minimal impact on pipeline performance. Typical CPU usage is <0.2% with a memory footprint of $\sim$ 1-10 MB.

Creating a performance model with the collected data will allow us to to optimize future clusters for LOFAR data processing. Doing so is necessary given the current data throughput, number of observations and time-line of the SKSP project. Similar issues will be encountered with upcoming radio telescopes [17].

To showcase the power of the pipeline_collector suite, the LOFAR *prefactor* pipeline was run through a single data set on three clusters and a personal machine. A number of insights were made using the high resolution timing data collected from this package (such as in Figure 4.10) and are listed in Table 4.1. In the future, we'll apply the *pipeline_collector* software to the more complex LOFAR DD pipeline, *ddf-pipeline*[12].

The slowest processing steps for the *prefactor* pipeline were identified as the *calib_cal* and *gsmcal_solve* steps. While the data can fit into the RAM for all of the processing machines, it is much larger than the processor's internal cache (Figure 4.6). The discoveries made concerned the memory hierarchy in Figure 4.6. Results labeled **R2**, **R8** and **R9** related to the CPU performance; **R2**, **R6** and **R7** related to the Cache performance; **R3** and **R5** related to the Memory usage and **R4** discussed the Disk speed.

Faster processors did not accelerate the *gsmcal_solve* step significantly, as this step streams data between the RAM and CPU. As the CPU speed increases, streaming applications become bottlenecked by the throughput of data into the CPU from RAM. As the *gsmcal_solve* algorithm iteratively calibrates chunks of the data, these chunks need to be loaded from disk once, however they are moved from RAM to CPU multiple times during calibration.

Similarly, the *calib_cal* step is more dependent on memory throughput than on CPU speed as this step moves data to and from memory frequently. This step also does minimization looping over the data set. As the data set does not fit in the cache, parts of it need to be constantly moving from memory and back. Figure 4.8a shows that the machine with the smallest LLC cache runs the *calib_cal* step the fastest. This is likely a combination of the benefit of faster RAM and poor cache optimization for

[12]https://github.com/mhardcastle/ddf-pipeline

this software. The same effect is much less pronounced in Figure 4.8b, suggesting that software optimization at least plays a part in the outliers for the laptop machine.

4.5.1 Recommendations

Based on these results, the top hardware recommendation is that *prefactor*'s slowest steps can be accelerated by running on machines with faster memory or upgrading the memory of the current machines. The two slowest *prefactor* steps showed improvements on machines with faster RAM.

One software recommendation is to improve the efficiency of the *calib_cal* step through refactoring or by replacing the software package used. Unfortunately, the software used for the *gsmcal_solve* step cannot be used for the *calib_cal* step as it is not yet able to correct for Faraday Rotation [100], making it impossible to currently use the software used by the *gsmcal_solve* step. Faraday Rotation has recently been implemented in a development version of the *prefactor* pipeline and is currently undergoing testing. This version of the pipeline will be implemented by September 2018.

Additionally, the large number of data cache misses recorded for the *calib_cal* step suggests that its source code is not optimized for multi-threaded processing. Data cache misses are often encountered when multiple threads have instructions on the same cache line[13], forcing the memory controller to move this cache line between cores [59]. This can also explain the large number of stalled cycles (Fig. 4.13a) and low number of full issue cycles (Fig. 4.13b) for the *calib_cal* step. It is recommended to further study the inefficiencies of *calib_cal* or to replace it with a newer software. If the software processing for this step is updated, analyzing the cache and CPU performance of the new software will be necessary to determine whether it efficiently uses the available computational resources.

Finally, we discovered that compiling the software on a virtual machine did not lead to a processing slowdown. This means that the current slowest *prefactor* steps are not optimized to use advanced processor instructions. Nevertheless, the resulting cross-compatibility is an encouraging result as it will allow to easily distribute pre-compiled versions of the software without increasing the processing time. We recommend continuing CVMFS deployment of LOFAR software.

[13] A cache line is a row of cache memory which is loaded into CPU as a single unit [27]

4.6 Conclusions

In this paper, we present a novel system for automated collection of performance data for complex software pipelines. We use this suite to study the LOFAR *prefactor* pipeline. The results are discussed aiming to understand the effect of different hardware parameters on the data processing. To do so, we run the pipeline on four different machines.

The software automatically collects performance data at the operating system level without impacting processing time. Data for each pipeline step is extracted using the OpenTSDB API, plotted and analyzed. Additionally, the *pipeline_collector* suite is easy to extend with new collectors that record more detailed time-series data for each pipeline step. The performance data is stored in the time series database OpenTSDB.

Here, we used this data to find 9 insights into the LOFAR prefactor pipeline listed in Table 4.1. The implementation details are described in 4.A.

The *prefactor* pipeline is used to do the initial processing for over 3000 observations that are part of the LOFAR SKSP Tier 1 survey. However, this pipeline is also used for lots of other LOFAR data sets outside the SKSP project. We have shown that increasing the RAM throughput is the easiest way to speedup *prefactor* processing. Running the *calib_cal* step on hardware with RAM faster than 4 GB/s will save up to 700k CPU hours for the 3000+ unprocessed data sets. This throughput increase will also speedup the *gsmcal_solve* step by 30% saving an additional 400k CPU hours. This is a significant fraction of the estimated 2,400k CPU hours required to process this data with the *prefactor* pipeline.

As shown in this work, we can correlate the performance of the LOFAR software with different hardware specifications. Additionally, the data sets can vary in size and job overheads on the compute cluster can depend on the processing parameters. All of these parameters affect the processing latency for the calibration and imaging pipelines. As such, a thorough parametric model is required to further optimize the end-to-end LOFAR processing pipeline and predict processing times on future clusters.

The design of this utility makes it easy to apply to future LOFAR pipelines. It is important to note that *pipeline_collector* is general enough that it can be used by other scientific pipelines, with no modification of the pipeline. Integrating *pipeline_collector* with a different pipeline requires only minor work. In future work, we will integrate *pipeline_collector* with *ddf-pipeline*. We will use this data to create a

performance model of the full LOFAR imaging pipeline [112, 134, 141]. including the DI step, implemented by *prefactor*, and the DD step, implemented by *ddf-pipeline*. This model will make it possible to first understand and then optimize the LOFAR pipeline and suggest for hardware and software improvements.

4.A Performance Collection Implementation Details

In this work, we have developed the *pipeline_collector* suite, aimed at collecting detailed time-series information from distributed scientific pipelines.

The *tcollector* package is a python software suite that can collect system performance data at predetermined intervals. The package is designed to monitor the performance statistics for web-servers and cluster nodes. The tcollector software records time series of the different performance metrics and sends them to a Time Series Database through HTTP. The Time Series Database, OpenTSDB stores the data in an HBase [6] instance at the performance collection server. Users interested in plotting time series can plot real time or historical data through an HTTP interface with OpenTSDB. With a central performance collection server, data from multiple processing sites can be collected and analyzed.

Tcollector formats the time series information in four fields. First is the name of the metric which is measured. Second is the UNIX timestamp. Third is the time series recorded as an integer or a float. Finally, a set of tags (key-value pairs) can be added to the data point. These four fields are discussed below and can be seen on the right side of Figure 4.14.

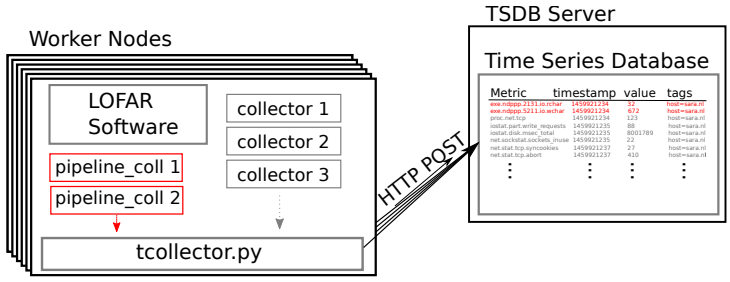


Figure 4.14: Communication between worker nodes and the TSDB server, including the pipeline_collector modules (in red). The *pipeline_collector* suite collects information on the running LOFAR pipelines, while the rest of the tcollector package collects system performance data. The existing tcollector package and its collectors are shown in gray. The collectors in gray only record metrics from the global system.

4.A.1 The pipeline_collector suite

The tcollector package cannot collect data on individual processes, nor can it associate these processes with specific steps of a data processing pipeline. We've supplemented the software with the *pipeline_collector* suite[14] using an executable that monitors a pipeline's running processes[15]. When an executable that is part of the LOFAR pipeline launches, a dedicated collector begins reporting information on the individual process. Every time a new processing step starts, the *prefactor* pipeline records the step name in a log file. *pipeline_collector* determines the current running step using this log. Running the LOFAR processing concurrently with the tcollector package gives us per-step performance data without changing or slowing down the LOFAR *prefactor* pipeline. Furthermore, *pipeline_collector* can be integrated with any processing pipeline as long as each pipeline step's name is recorded at its launch.

The *pipeline_collector* suite sends data to the time series database in the same format as the rest of the collectors included in the tcollector package.

4.A.2 Setting up for future pipelines

The setup options for *pipeline_collector* are stored in a configuration file in the root directory of the package. This file holds the sample interval, executables to monitor and the location where *pipeline_collector* can read the current pipeline step

The *pipeline_collector* suite reads the current pipeline step from a file, the location of which is specified in the configuration. This file needs to be updated each time the pipeline begins a new step. For LOFAR we have a script running with the pipeline, and determining the current step using the pipeline logs. As each pipeline has a unique sequence of steps, the current step needs to be recorded in a file in order for *pipeline_collector* to report it to the time series database. The location of the file recording the current pipeline step is read from the configuration file.

Next, the names of the specific processes need to be included in the configuration file. In the case of LOFAR, we select the `NDPPP`, `bbs-reducer` and `losoto` processes. The *pipeline_collector* searches the running processes for the current user for these process names and launches a collector for each new process launched by the current step.

[14] Located at https://gitlab.com/apmechev/procfs_tcollector.git
[15] Located at https://github.com/apmechev/procfsamp

Acknowledgments

APM would like to acknowledge the support from the NWO/DOME/IBM programme "Big Bang Big Data: Innovating ICT as a Driver For Astronomy", project #628.002.001.

HJR gratefully acknowledge support from the European Research Council under the European Unions Seventh Framework Programme (FP/2007-2013)/ERC Advanced Grant NEWCLUSTERS- 321271.

JBRO acknowledges financial support from NWO Top LOFAR-CRRL project, project No. 614.001.351.

This work was carried out on the Dutch national e-infrastructure with the support of SURF Cooperative through grant e-infra 160022 & 160152.

5 Fast and Reproducible LOFAR Workflows with AGLOW

The contents of this chapter are based on a manuscript Published at the IEEE e-Science conference 2018

Original Abstract:

The LOFAR radio telescope creates petabytes of data per year. This data is important for many scientific projects. The data needs to be efficiently processed within the time span of these projects in order to maximize the scientific impact. We present a workflow orchestration system that integrates LOFAR processing with a distributed computing platform. The system is named Automated Grid-enabled LOFAR Workflows (AGLOW). AGLOW makes it fast and easy to develop, test and deploy complex LOFAR workflows, and to accelerate them on a distributed cluster architecture. AGLOW reduces the setup time of complex workflows: typically, from months to days. We lay out two case studies that process the data from the LOFAR Surveys Key Science Project. We have implemented these into the AGLOW environment. We also describe the capabilities of AGLOW, paving the way for use by other LOFAR science cases. In the future, AGLOW will automatically produce multiple science products from a single data set, serving several of the LOFAR Key Science Projects.

5.1 Introduction

Data sets in radio astronomy have increased 1000-fold over the past decade[97]. It is no longer feasible to move, store and process these data sizes at university clusters, nor to process these data manually. LOFAR, the Low-Frequency Array[130]

is a modern and powerful radio telescope that creates more than 5 Petabytes of data per year. At present, the majority of LOFAR time is allocated to several Key Science Projects (KSPs)[109]. These projects need to process hundreds or thousands of observations. Typical observations produce approximately 14 TB of archived data. Obtaining high fidelity images from this data requires complex processing steps. To manage and automate the data processing, workflow management software is needed. This software needs to accelerate LOFAR processing on a High Throughput Computing (HTC) cluster while ensuring it is easy to prototype, test, and integrate future algorithms and pipelines.

To automate LOFAR data processing, we have worked with the LOFAR Surveys KSP (SKSP). Together, we designed a software suite that integrates LOFAR software[63] with the Dutch grid infrastructure[124]. This software, based on Apache Airflow[1], makes it easy to add future science cases, extend and modify pipelines, include data quality checks, and rapidly prototype complex pipelines. For the SKSP use cases, AGLOW achieves a significant reduction in development time: from months to days, allowing researchers to concentrate on data analysis rather than management of processing. Additionally, and perhaps more importantly, the software versions and repositories used are defined within the workflow. This makes reproducibility an integral part of the AGLOW software. Finally, the software is built to leverage an HTC cluster by seamlessly submitting the processing jobs through the cluster's job submission system[58]. The work presented here builds on our previous work parallelizing single LOFAR jobs[76] on a distributed environment. The majority of processing was done at SURFsara at the Amsterdam Science Park[116], which is one of the sites used by the LOFAR Long Term Archive (LTA)[2]. Ongoing efforts include scheduling and processing data at clusters in Poznań in Poland and Jülich in Germany.

Contributions: The main features of the AGLOW software are the following:

- Integration of the Grid middleware with Apache Airflow, allowing us to dynamically define, create, submit and monitor jobs on the Dutch national e-infrastructure.
- Integration of the LOFAR (LTA) utilities in Airflow, facilitating pipeline developers to automate staging (moving from tape to disk) and retrieval of LOFAR data.

[1] https://airflow.apache.org/
[2] https://lta.lofar.eu

- Integration of the SURFsara storage with Airflow, making LOFAR pipelines aware of the storage layer available at the Dutch national e-infrastructure.
- Ease of creating simple software blocks, with which users can integrate and test their pipelines.
- Storing all software versions and script repositories as part of the workflow to make LOFAR processing reproducible and portable.

Outline: The organization of this manuscript is as follows: We provide background on data processing in radio astronomy and why LOFAR science requires complex workflows and cover workflow management algorithms and capabilities (section 7.2). We discuss related work in workflow management (section 7.3). In section 5.4, we introduce our software and two use cases. Both of our use cases require acceleration at an HTC cluster and automation by a workflow orchestration software. We follow these examples with details on the integration between LOFAR software, LOFAR data and the resources at SURFsara in Amsterdam in section 5.4.2. Finally, we discuss our results (sect. 7.5) and look ahead to the demands of future LOFAR projects and upcoming telescopes in section 7.6.

5.2 Background

This work lies at the intersection of Radio Astronomy and Computer Science. The goal of the study is to leverage the flexibility of an industry standard workflow management software and use CERN's Worldwide Computing Grid[3] at SURFsara [105] to accelerate reproducible processing of LOFAR data.

A single LOFAR surveys observation is recorded in distinct frequency chunks (henceforth called 'Subbands'), each of which is uploaded to the LTA as a separate file. Some of the processing steps require the entire frequency information, while other steps can run independently and operate on a single Subband. The latter steps can be easily accelerated on an HTC cluster by taking advantage of the data level parallelism.

Multiple scientific projects may desire to run different processing steps on a single LOFAR observation. To minimize time spent on retrieving data from the LTA and eliminate re-processing of data, pipelines for multiple science cases need to be integrated together. This integration should be done by a software that encodes

[3] `http://wlcg.web.cern.ch/`

the dependencies between different steps and automatically executes processing steps once their dependencies have been met. Software packages that solve these challenges are called 'workflow management software' (see, e.g., [3, 24, 62].)

5.3 Related Work

A workflow is described by a set of tasks. The dependencies between these tasks are encoded in a Directed Acyclic Graph (DAG) [139]. This data structure imposes a strict dependency hierarchy between the tasks [90]. This means that there exists a well-defined execution order and a well-defined list of dependencies for each task. The execution order is typically determined by algorithms such as Kahn's algorithm [52] or a depth-first search [147].

Workflow management software is used in various fields from research to industry. In biology, gene sequencing and analysis pipelines require automation of multiple processing steps. In gene sequencing, Toil[4] has been successfully used to automate RNA sequence analysis[137]. Additionally, many software teams in biotech develop their own in-house workflow management software [34].

Currently, we can parallelize a single processing step of the pipeline using the Grid LOFAR Tools (GRID_LRT[5]) [76]. The LOFAR Surveys science cases incorporate multiple steps with inter-linked dependencies. Resolving these dependencies can be done efficiently by a comprehensive workflow orchestration software. The purpose of such software is to resolve dependencies between the multiple tasks in a workflow, execute these tasks, and track the status, logs, output, and run time of each task.

In astronomy, workflow systems have been developed that are telescope specific, such as ESOReflex[37] by the European Southern Observatory. Other projects, such as astrogrid[6] and 'Workflow 4Ever'[7], have either been completed or are no longer supported. The astrogrid project, for example, was a collaboration to create standards, infrastructure, and software for distributed astronomical processing. Its operation phase spanned 2008-2010. Workflow4Ever, likewise, has been out of support since 2013. To ensure continuing support for the LOFAR workflows, we have decided to use a leading enterprise workflow management software, Airflow.

[4] `https://toil.readthedocs.io`
[5] `https://github.com/apmechev/GRID_LRT`
[6] http://www.astrogrid.org
[7] http://wf4ever.github.io/ro/

Airflow is an open source Python software package developed by Airbnb[8] to manage complex workflows. It encodes workflows in Python files and makes it easy to re-use, re-arrange, schedule and execute blocks in a user-defined workflow. Airflow is capable of scheduling and executing workflows by resolving the dependencies between tasks and scheduling these tasks for execution. The software uses a metadata database[9] to retain metadata such as task state, execution date, and output. While Airflow allows building workflows easily from Python and bash functions, it can easily be extended to support custom processing scenarios. Additionally, Airflow conforms to the Common Workflow Language (CWL) [4] standard using the *cwl-airflow* package [56], meaning it can execute CWL workflows as well. Finally, Airflow is part of the Apache incubator and upon certification will receive continual support by the Apache software foundation[10].

5.4 AGLOW

Complex astronomical pipelines are time consuming to develop and operate. Furthermore, they may evolve rapidly to incorporate new processing techniques or requirements. Migrating these pipelines to a distributed, high throughput environment is often justified, or even required, in order to meet the timelines set by scientific projects. The time saved by running on a cluster must be balanced by the flexibility and development time required to implement or update the scientific pipelines. To address these concerns, we have developed a software package, Automated Grid-enabled LOFAR Workflows (AGLOW)[11]. AGLOW is based on Airflow and the LOFAR software and addresses issues with automation and acceleration of LOFAR processing.

With AGLOW, we can translate LOFAR pipelines into DAGs. We provide the tools that enable users to easily implement LOFAR science pipelines and execute them on a distributed architecture. Using these tools, the data processing required by various LOFAR science cases is automated and accelerated.

5.4.1 AGLOW: Case Study

For our case-study, we have chosen two ways to process LOFAR Surveys data: coverage and depth. The Surveys Key Science Project (SKSP)[109] is an ambitious project

[8] `https://www.airbnb.com/`
[9] In our case implemented by Postgresql
[10] https://www.apache.org/
[11] `https://github.com/apmechev/AGLOW`

to map the northern sky at low frequencies using the Dutch LOFAR stations. These maps will help understand the formation and evolution of massive black holes, galaxies, clusters of galaxies and large-scale structure of the Universe.

The LOFAR surveys observations consist of several tiers with the widest Tier (Tier 1) covering the whole sky visible from the Northern Hemisphere with 3168 observations of 8 hours each [109]. The other tiers (Tier 2, Tier 3) consist of much longer observations of smaller sections of the sky and can collect hundreds of hours of data for a single direction. The deepest single field being analyzed, in collaboration with the EoR group, is the North Celestial Pole (NCP) field which has ∼1700 hrs of observations to date. Processing this data will create an image with an unprecedented resolution and sensitivity. Here we have implemented processing pipelines for both the Tier 1 data and Tier 2 data into AGLOW.

The scientific importance of these two examples, as well as the large processing requirements, make them ideal candidates for acceleration and automation with AGLOW.

Surveys Project: All Sky Survey

The main driver for the development of AGLOW and its constituent packages has been the LOFAR SKSP Project. A typical 8-hour observation produces 14 TB of data. This data is eventually reduced to several hundred gigabytes. Data needs to be processed by two pipelines: first by the Direction Independent (DI) pipeline, and then by the Direction Dependent (DD) pipeline.

We have split the DI pipeline into four stages, and the DD pipeline into two subsequent stages. Splitting up the pipelines in stages allows speedup through parallelization for the stages that can benefit from data-level parallelism. Additionally, this setup allows fault tolerance and easy re-processing.Importantly, with AGLOW, adding new functionality to the pipeline is easy and can be done at any time without disrupting current processing. Current SKSP processing is easily started by launching a new DAG-run in Airflow. We schedule these runs daily at midnight, however they can also be started manually bu users.

Deeper Surveys Fields

To create deep images of a single field, minor modifications were made to the processing pipeline described in the previous section. Scripts were included to re-align the data in the frequency axis, and the DD processing steps include an extra final

combination step that stacks multiple observations. Being able to rapidly test alternative processing strategies is crucial to creating a deep image of the NCP field. With the success of this project, future deep LOFAR observations will be processed with these pipelines.

5.4.2 AGLOW: Implementation

AGLOW combines the LOFAR software, the Grid LOFAR Tools (GRID_LRT), and Airflow to allow automation and makes large-scale LOFAR processing easily reproducible. The components of the AGLOW software are shown in Fig. 5.1. In this section, we will discuss these components and their functions.

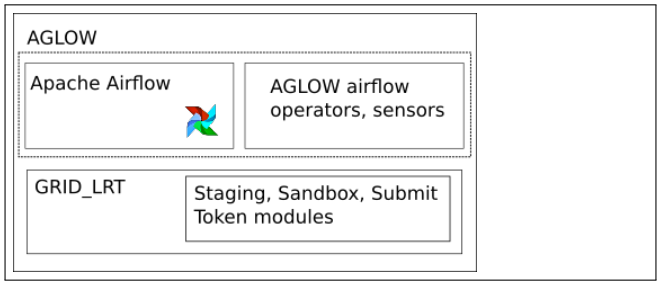


Figure 5.1: Design of the AGLOW software, incorporating Airflow, the GRID_LRT package[76] and custom operators designed to integrate LOFAR software, Grid middleware and dCache storage. GRID_LRT is a software package developed to parallelize single LOFAR jobs at SURFsara. It contains several modules to help set up, and launch jobs on an HTC cluster at SURFsara. Airflow is a stand-alone package by Airbnb, which is extended with several classes that couple Airflow with the Grid infrastructure. These classes are collectively named the AGLOW operators/sensors.

GRID LOFAR Tools and LOFAR software

We have previously developed tools to create LOFAR jobs and launch them on a distributed infrastructure[76]. These tools have matured to a point where it is easy to both plug and play existing scripts and extend the framework to add more complex pipelines. These steps make it possible for a user to batch execute bash or Python scripts on their LOFAR data in parallel. After the scripts are executed, the results are uploaded to shared dCache storage[38] at SURFsara[116].

More complex steps use additional Github repositories, such as the prefactor[12] for direction independent calibration or DDF-pipeline[13] for direction-

[12]https://github.com/lofar-astron/prefactor

dependent calibration and imaging. The sequence of steps is encoded in parameter-set files (parsets), which can be modified and dropped into AGLOW depending on the processing requirements. Users have full control over which git repository, branch and commit number to use for each processing step.

With AGLOW, we can easily include the DDF-pipeline and prefactor repositories, as well as any other scripts. Since these scripts are tracked by git [126], a full commit and branch history of the scripts is available. We use this history to make processing reproducible, by using the same git-commit for all LOFAR data sets.

In addition to these script repositories, we have integrated the most common software packages used to process LOFAR data with AGLOW. These are the Default Pre-Processing Pipeline (DPPP)[63], the LOFAR Solutions Tool (LoSoTo), WSclean [84], AOflagger[83], CASA[70], pyBDSF[80], DDFacet[118] and KillMS[113, 119].

Extending Airflow

Two types of modifications were made to Airflow to allow processing on a Grid environment. First, functions were added to check the number of files located in intermediate grid storage. We use this to decide whether to stage files, or whether enough files have been successfully processed by a previous task.

Second, more complex tasks were implemented as Airflow operators or sensors (Figure 5.2). These tasks include creating job description files, setting up job scripts, launching jobs using the gLite workload management system and monitoring the status of these jobs. Future additions will include operators that evaluate the current cluster workload and make decisions on location to launch the data processing. With the AGLOW package, such tasks are easy to implement without modifying or interrupting processing. This leads to an easily reproducible, intelligent scientific processing that is also efficiently executed and requires minimal interaction. The operators and sensors added to Airflow are shown in Fig. 5.2. The most commonly used ones are described below:

Staging AGLOW stages data through the ASTRON staging API, which is made available through the GRID_LRT software [14].This API loads the user's LTA credentials from a file. Using these credentials, AGLOW stages the required files at the

[13]https://github.com/mhardcastle/ddf-pipeline. DDF-pipeline is a leading example of a Direction Dependent calibration pipeline used for LOFAR data. It uses DDFacet [118], KillMS [113] and to create high quality images.

[14]`https://grid-lrt.readthedocs.io/en/latest/staging.html#grid-lrt-staging-stage-all-lta`

respective LTA site. The AGLOW staging operator waits until all the files have been staged before returning a successful exit status.

Sandbox Scripts to retrieve, process and upload the data are stored in a sandbox, as described in [76]. The LRT_Sandbox operator that builds this sandbox directly uses the GRID_LRT Sandbox module to build and upload the sandbox according to the user-provided configuration file.

Token Creation Jobs parallelized with GRID_LRT store processing parameters inside CouchDB documents called PiCaS Tokens. Each token corresponds to one grid processing job. These tokens are created by the LRT_Token operator in AGLOW. This operator uses the GRID_LRT Token module to define tokens and create them from a list of links to LOFAR data.

Job Submission The job submission at SURFsara is done through the GRID_LRT submit module. This module is a python interface to the glite job submission system. It is wrapped in the glite-submit operator in AGLOW, which returns the ID number of the job. AGLOW includes a gliteSensor, which uses this ID to wait until all the processing jobs have finished.

Using AGLOW to accelerate the execution of a pipeline requires deciding how to split the processing to benefit from parallelization. Once the steps to be parallelized are selected, users can add git repositories of scripts to the configuration file. Next, each step is added to a Python script called the DAG file. This file is placed in the Airflow's dags folder, which adds it to AGLOW. An example DAG file can be found at `https://aglow.readthedocs.io/en/latest/dags.html`. Users who want to implement a new workflow design the DAG files and add them to Airflow's dags folder. Workflows are migrated to a new server, by transferring the DAG and configuration files to the new AGLOW instance.

5.4.3 AGLOW: Jobs

Once LOFAR observations are downloaded from the LTA, they are typically processed with several packages before producing a science ready data set. We have integrated these packages with Airflow to make it easy to create complex LOFAR workflows.

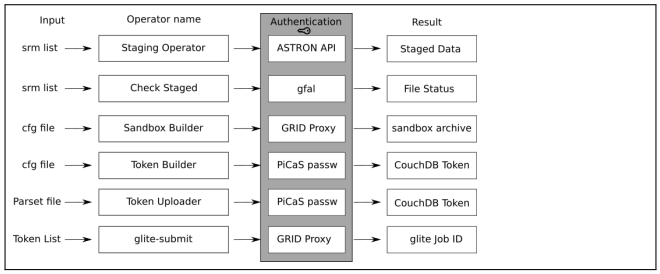


Figure 5.2: Airflow Operators for Staging LOFAR data, creating job descriptions and submitting jobs to the Dutch grid. On the left is the input given to each operator. 'SRM lists' are lists of links to data at the LOFAR LTA or located on the SURFsara dCache storage. Parsets are files specific to 'prefactor' and 'DDF-pipeline' and define the processing for each pipeline step. Finally, the 'Sandbox' and 'Token' operators read their parameters from a configuration file. The use of a scripts sandbox and job description tokens is detailed in our previous work[76]. Documentation of the operators is available at `https://aglow.readthedocs.io/en/latest/operators.html`

Each of the processing steps above requires extra set-up to process on the Dutch Grid infrastructure. The job scripts setup, job description, and job submission are done by the GRID_LRT package[76]. With AGLOW, we automate this setup, enabling users to focus on developing more comprehensive data processing pipelines. Below we outline several possible steps a user can use in their pipeline.

DPPP Parset

The DPPP software is used extensively in LOFAR data processing. It has many capabilities such as flagging bad data, averaging data in time and frequency, and calibrating the data with a sky-model.

The input parameters of this software are stored in a text file called a parset. The input data and the DPPP parset are sufficient to define a DPPP execution step. As noted in section 7.2, LOFAR data is split in frequency into Subbands. Much of the DPPP processing, such as averaging and flagging, can be done independently for each Subband, thus they can be processed on independent machines. This parallelization makes these steps a perfect candidate for an HTC cluster. For a data set that is split into 244 Subbands, 244 jobs are launched concurrently.

In Airflow, the DPPP parset task is encoded in a DAG (Fig. 5.3). The DPPP DAG is a linear workflow that consists of the 'sandbox' setup, creation of the job-description documents, uploading of the DPPP parset and job launching and monitoring.

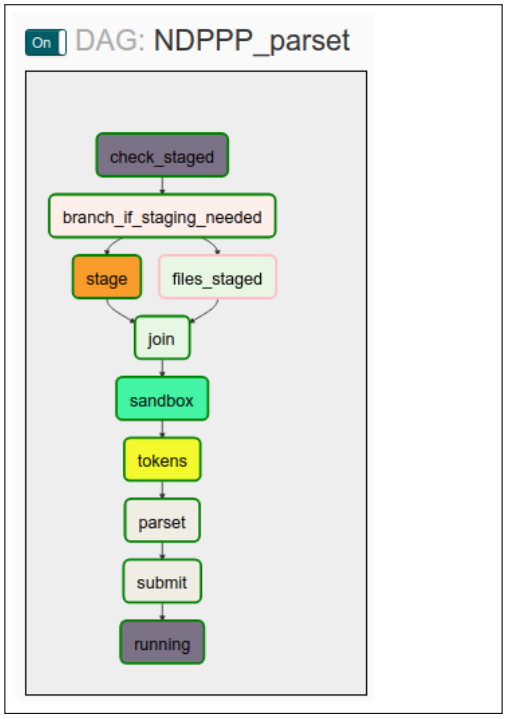


Figure 5.3: Render of the DPPP parset DAG in the Airflow User Interface. This view shows the setup and submission steps. Even this simple DAG can include branching options such as the `branch_if_staging_needed` task which checks if the data is not staged and stages it. All of the operators in this figure are part of the AGLOW software. Their inputs and outputs are shown in Fig. 5.2. Using configuration files, the NDPPP DAG can be used by different users for different science cases making it portable and maintainable. These features make reproducible science with LOFAR data easy.

WSclean Job

The WSclean[84] package is used to create an image from a LOFAR data set. This software has a very wide range of parameters options, however, it cannot take a parset file as an input. Instead, the parameters are specified in the command line. In the AGLOW implementation, we parse all the command-line parameters from a text file, referred to as the 'wsclean parset'. This file is added to the jobs in the same way as the DPPP parset, i.e. using the *Token Uploader* Operator. The DAG for the wsclean software uses the same blocks as the DPPP DAG, with different configuration and parset files. The reuse of Airflow operators makes maintainability of all tasks easier.

5.4.4 Shell/Python Script

Users that require the run of multiple software packages on a single data set can craft a custom shell or Python script that executes these steps using the LOFAR tools during a single distributed job. This option increases flexibility and minimizes the overhead associated with scheduling and running multiple jobs in sequence. At the workflow orchestration level, we use the same Airflow operators as the above tasks. The script is uploaded to the job description database using the *Token Uploader* Operator. It is executed once the jobs are launched.

Currently only the LOFAR Spectroscopy project uses custom shell scripts to process LOFAR data. A recent study of carbon recombination lines used a custom bash script to calibrate and image LOFAR data on the SURFsara `GINA`[15] cluster [98].

5.4.5 Prefactor parset

The input to the prefactor pipeline software is a parset file which describes a linear workflow. The description of this workflow consists of a list of processing steps and their associated parameters. The 'prefactor' package uses the LOFAR software to do the direction-independent calibration of the archived LOFAR datasets. Prefactor steps are executed by the generic pipeline framework[63]. While this framework can run a sequential pipeline, it is not capable of conditional branching nor parallelization on all cluster architectures. The original goal of the GRID_LRT software was to tackle the parallelization challenge while AGLOW solves the additional challenge of pipeline management.

We have already processed more than 50 datasets through the 'prefactor' DAG using AGLOW. The full 'prefactor' pipeline is shown in figure 5.4. This DAG shows the four processing steps as well as additional Python operators that manage the staging and result verification.

5.4.6 DDF-pipeline

The final AGLOW DAG is the implementation of the DDF-pipeline repository which is a pipeline that is extensively used by the LOFAR surveys KSP and is described in detail in [109]. This pipeline operates on the products of the prefactor pipeline and consists of a series of calibration and imaging loops with the objective

[15] The `GINA` cluster is an HTC cluster located at SURFsara integrated with the Dutch Grid initiative. It supports massively parallel processing which is required to efficiently process LOFAR data with *prefactor*.

of creating a final science quality image. For each of these loops the majority of the processing time is spent in DDFacet[118] and KillMS[113, 119] steps that perform the direction-dependent imaging and calibration respectively.

In total, DDF-pipeline takes ∼4 days of processing to complete. As DDF-pipeline creates large intermediate files we have so far not divided the pipeline into too many steps to avoid filling the storage on the GINA cluster. However, we have split the pipeline into two steps and there is further potential for parallelization that will be implemented in the future.

5.4.7 Linking Multiple Jobs

Pre-processing of LOFAR SKSP data can be done by a single DPPP task, with 244 jobs running in parallel. More complex LOFAR pipelines will include multiple processing tasks as well as tasks responsible for job setup. Therefore, it is important to facilitate running multi-step pipelines with AGLOW.

Creating workflows by defining dependencies between tasks is a core Airflow capability. We use this functionality to link multiple steps of a LOFAR pipeline together. In the SKSP pipeline, we take advantage of the data level parallelism for the initial processing steps for the calibrator and target. The other two steps are run as a single grid job. Switching the parallelization for each step is done by changing the number of datasets per node parameter in the configuration file for each step.

5.5 Results and Discussions

The implementation of AGLOW makes it possible to efficiently process LOFAR data with minimal user interaction. The scheduling algorithm automatically launches pipelines, meaning that there is little time spent between runs. Additionally, controlling/fixing the version of the scripts is done by specifying the commit of each script repository. This makes data processing easily reproducible. Once the dependencies of multiple science pipelines have been encoded in a DAG, Airflow efficiently executes this DAG, running tasks in parallel where possible.

An important feature of AGLOW is the loose coupling between pipeline logic, software versions, pipeline parameters, and datasets. The goal of this decoupling is to give users complete control over all the processing variables. With AGLOW, one can develop the pipeline logic independently of the LOFAR software

versions and conversely update the LOFAR software and script repositories independently from the pipeline logic. Finally, the Airflow operators are themselves decoupled from the scientific pipelines. As these operators are reused, this decoupling makes them easy to maintain and extend.

The first LOFAR processing pipeline integrated with AGLOW was a single linear workflow, with only one submission to the compute cluster. This workflow is used to reduce the data size making data retrieval to research institutes less time consuming. We offer this workflow as a service to LOFAR users who do not have a high-bandwidth connection to the LOFAR Archive.

A more complex pipeline was implemented: the LOFAR direction independent calibration pipeline ('prefactor'). The scientific importance and complexity of this pipeline make it a good case study for the capabilities of the AGLOW software. We show that AGLOW's design allows integration of more complex data processing workflows with the Dutch Grid resources. These workflows can be either used by PIs of LOFAR projects or offered as a processing service to the wider astronomical community. Since March 2018, more than 300 'pre-factor' datasets have been processed with the SKSP workflow. With AGLOW, we have been able to process data with an average cadence of 50 observations per month.

In large part thanks to their flexibility, automation, and Grid integration, AGLOW and GRID_LRT have become a standard part of the Direction Independent processing for the LOFAR SKSP project.

5.6 Conclusions

In this work, we have detailed a comprehensive workflow management software for processing radio astronomy data on a distributed infrastructure. We leverage an industry standard workflow management software, Airflow. Using its capabilities, we make it possible to build, test, automate and deploy LOFAR pipelines on short timescales, generally from months to days. With the flexibility of Airflow's Python and Bash operators, users can design their own workflows, as well as co-ordinate more complex science cases. In this way, AGLOW facilitates reproducible processing of scientific data. In the future, AGLOW will support additional LOFAR science cases including Long Baselines and Spectroscopy. In this article, we have described our implementation of the data processing pipelines used by the LOFAR Surveys Key Science Project.

Future work includes further de-coupling of the Grid-setup and pipeline logic. We will do this by creating 'sub-dags' (details in 5..1) for each type of LOFAR jobs. Using these sub-dags will reduce the complexity of scientific workflows while also making the code even more reusable and thus easier to maintain and upgrade. Efforts to integrate processing at the other two LTA sites, (Jülich and Poznań) have already started with 'prefactor' runs being performed on Jülich using a modified version of the SKSP workflow. The software also currently works on the Eagle cluster at Poznań. Combining the Jülich and SURFsara workflows will be done in the future so that AGLOW can track and start processing at multiple clusters.

Finally, AGLOW can be used as a 'LOFAR As A Service' model. In this model, users only provide an observation ID and processing parameters and receive the final results upon job completion. This model will build upon previous success offering LOFAR processing to users without login to the `GINA` cluster [88]. This previous work was already useful for studying radio absorption in Cassiopeia A [9] and a 'data-to-images' service will be valuable to the whole LOFAR community.

Our experience with automating LOFAR scientific workflows on a distributed architecture will be valuable when setting up data processing for future Radio Telescopes such as the Square Kilometer Array [33] .

APPENDIX

The LOFAR SKSP workflow is shown in Figure 5.4. This figure shows how reuse of the staging, setup operators, and glite-wms sensors makes maintainability easy and allows rapid prototyping of complex pipelines.

This workflow additionally takes advantage of Airflow's *PythonOperator* to check if the LOFAR data is on disk at the archive and whether all final products were uploaded by each step. AGLOW also allows for staging the calibrator and target files concurrently. When the data is staged, Airflow continues with the processing of that data.

5..1 Sub-DAG

Airflow allows developers to include entire DAGs as a single task in their workflow. Airflow can trigger a DAG execution based on parameters provided by the parent DAG. This feature makes it possible to concatenate short, commonly used tasks into DAGs and call them in a parent workflow. Using sub-DAGS makes the code more

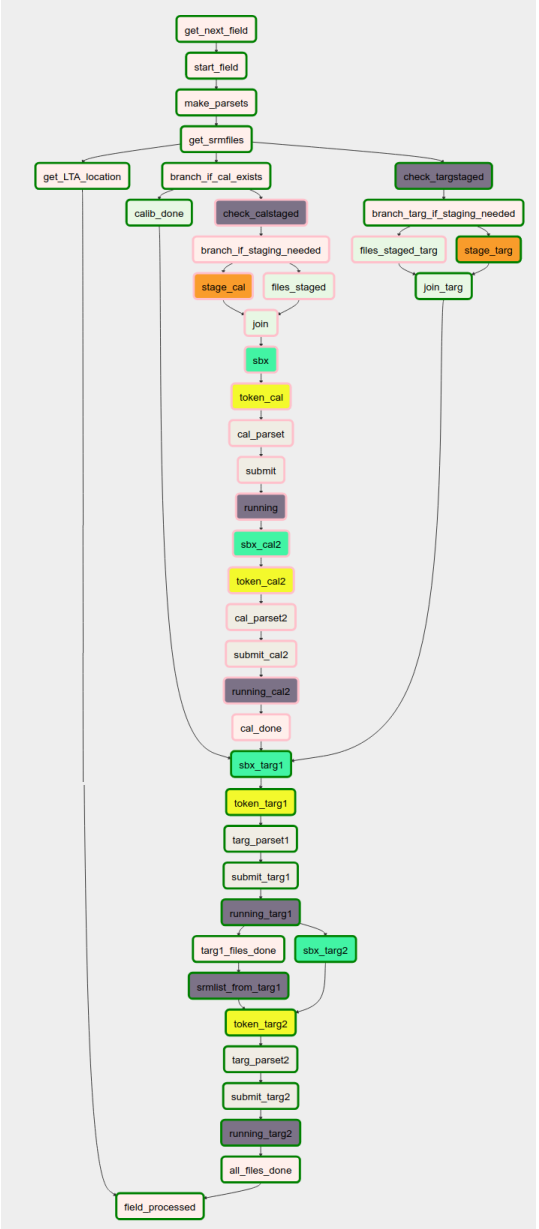


Figure 5.4: Workflow for the prefactor pipeline. Here we show the reuse of AGLOW operators for the four prefactor steps. In addition to the LOFAR processing, we also have conditional operators to skip processing of the calibrator if it has been previously processed. This is done by the 'branch_if_cal_exists' task. We also have a final step that checks if all the results have been uploaded, done by the 'all_files_done' task. Likewise, quality checks can be added in this workflow wherever needed.

maintainable and easy to use, while it makes workflows simpler. For LOFAR, SubDAGs are used to automate job submission, making the resulting scientific workflows simpler.

Acknowledgment

APM would like to acknowledge the support from the NWO/DOME/IBM programme "Big Bang Big Data: Innovating ICT as a Driver For Astronomy", project #628.002.001.

The processing and storage functionality that has made this project possible was enabled by SURF Cooperative through grant e-infra 160022 & 160152. The LOFAR software and dedicated reduction packages on `https://github.com/apmechev/GRID_LRT` were deployed on the e-infrastructure by the LOFAR e-infra group, consisting of J.B.R. Oonk (SURFsara & Leiden Observatory), A.P. Mechev (Leiden Observatory).

This paper is based on data obtained with the International LOFAR Telescope (ILT) under project codes LC2_038 and LC3_008. LOFAR (van Haarlem et al. 2013) is the Low-Frequency Array designed and constructed by ASTRON. It has observing, data processing, and data storage facilities in several countries, which are owned by various parties (each with their own funding sources), and which are collectively operated by the ILT foundation under a joint scientific policy. The ILT resources have benefited from the following recent major funding sources: CNRS-INSU, Observatoire de Paris and Université d'Orléans, France; BMBF, MIWF-NRW, MPG, Germany; Science Foundation Ireland (SFI), Department of Business, Enterprise and Innovation (DBEI), Ireland; NWO, The Netherlands; The Science and Technology Facilities Council (STFC), UK

6 Scalability Model for the LOFAR Direction Independent Pipeline

The contents of this chapter are based on a manuscript Published in Astronomy and Computing 2019

Abstract

Original Abstract:

LOFAR is a leading aperture synthesis telescope operated by the Netherlands with stations across Europe. The LOFAR Two-meter Sky Survey (LoTSS) will produce more than 3000 14 TB data sets mapping the entire northern sky at low frequencies. All of the data produced by LoTSS needs to be processed by the LOFAR Direction Independent (DI) pipeline, `prefactor`. Understanding the performance of this pipeline is important when trying to optimize the throughput for its large projects, such as LoTSS and other deep surveys. Making a model of its completion time will enable us to predict the time taken to process large data sets, optimize our parameter choices, help schedule other LOFAR processing services, and predict processing time for future large radio telescopes. We tested the `prefactor` pipeline by scaling several parameters, notably number of CPU's, data size and size of calibration sky model. We present these results as a comprehensive model which will be used to predict processing time for a wide range of processing parameters. We also discover that smaller calibration models lead to significantly faster calibration times, while the calibration results do not significantly degrade in quality. Finally, we validate the model and compare predictions with production runs

from the past six months, quantifying the performance penalties incurred by processing on a shared cluster. We conclude by noting the utility of the results and model for the LoTSS Survey, LOFAR as a whole and for other telescopes.

6.1 Introduction

Astronomy has entered the big data era with many projects creating petabytes of data per year. This data is often processed by complex multi-step pipelines consisting of various algorithms. Understanding the scalability of astronomical algorithms theoretically, in a controlled environment, and in production is important for making predictions for the data reduction of future projects and upcoming telescopes.

The Low Frequency Array (LOFAR) [130] is a leading European low-frequency radio telescope. The majority of LOFAR's stations are in the Netherlands, however the telescope can use stations across Europe to create ultra-high resolution radio maps. LOFAR data needs to undergo several computationally intensive processing steps before obtaining a final scientific image.

To create a broadband image, LOFAR data is first processed by a Direction Independent (DI) Calibration pipeline followed by Direction Dependent (DD) Calibration pipeline [e.g. 112, 121, 134, 141]. The goal of DI calibration is to remove effects that are constant across the target field such as radio frequency interference, contamination by bright off-axis sources and antenna gains. After this step, DD Calibration focuses on removing effects which vary across the field, such as ionospheric and beam effects. The result of these two pipelines is a science-ready image.

Our implementation of the DI LOFAR processing, `prefactor`, can be parallelized on a high throughput cluster [75]. The Direction Dependent processing, implemented in `ddf-pipeline`[1], is subsequently performed on a single HPC node.

The LOFAR Surveys Key Science Project (SKSP) [107, 109] is a long running project consisting of several low frequency surveys of the northern sky. The broadest tier of the survey, LoTSS, will use more than 3000 8-hour observations to create maps with a noise levels below 100 μJy. We have already processed more than 500 of these observations using the `prefactor` DI pipeline [46, 134].

[1] Available at https://github.com/mhardcastle/ddf-pipeline/releases

While the current LoTSS imaging algorithms can process data averaged by up to a factor of 64 in frequency and time, it is important to understand how LOFAR processing scales with processing parameters, such as averaging parameters. Since LOFAR data is used by multiple scientific teams, not every team can produce scientific results from data averaged by such a high factor. Users from those teams need to be able to predict the time and computational resources required to process their data, taking into account the increasing LOFAR observation rates, data sizes and scientific requirements.

We study the scalability of processing LOFAR data, by setting up processing of a sample SKSP data set on an isolated node on the `GINA` cluster at SURFsara, part of the Dutch national e-infrastructure [123]. We test the software performance as a function of several parameters, including averaging parameters, number of CPUs and calibration model size. Additionally, we test the performance of the underlying infrastructure, i.e. queuing and download time, for the same parameters. Finally, we compare those isolated tests with our production runs of the `prefactor` pipeline to measure the overhead incurred by running on a shared system.

We discover that the computationally intensive LOFAR processing steps scale linearly with data size, and calibration model size. Additionally, we find that the time taken by these steps is inversely proportional to the number of CPUs used. We discover that the time to download and extract data on the `GINA` cluster is linear with size up to 32GB, but becomes slower beyond this data size. We also find that the queuing time on the `GINA` cluster grows exponentially for jobs requesting more than 8 CPUs. We validate these isolated tests with production runs of LOFAR data from the past six months. We combine all these tests into a single model and show its prediction power by testing the processing time for different combinations of parameters. Finally, we discuss the utility of our method, the results in this work and applications to the SKSP projects, the broader impact of our results to LOFAR processing and the applications for other large astronomical surveys. The major contributions of this work can be summarized as:

- A model of processing time for the LOFAR Direction Independent Calibration Pipeline.
- A model of queuing time and file transfer time which is used by current or future jobs processed on the `GINA` cluster.
- Quantification of overheads incurred when processing in production.
- Validation of our methods with discussion of future applications.

We introduce LOFAR processing and other related work in Section 6.2 and describe our software setup and data processing methods in Section 6.3. We present our results and performance model in Section 6.4 and discussions and conclusions in Section 6.5.

6.2 Related Work

In previous work, we have parallelized the Direction Independent LOFAR pipeline on a High Throughput infrastructure [75]. While this parallelization has helped accelerate data processing for the SKSP project, creating a performance model of our software is required if we are to predict the resources taken by future jobs. This model will be particularly useful in understanding how processing parameters will affect run time.

Performance modelling on a distributed system is an important field of study related to grid computing. A good model of the performance of tasks in distributed workflows can help more efficiently schedule these jobs on a grid environment [101]. The performance modeling systems require knowledge of the source code and an analytical model of the slowest parts of the code [144]. Many systems exist to model the performance of distributed jobs [12, 57, 142, 144], with some employing Black Box testing [54, 145] or tests on scientific benchmark cases [21]. Such performance analysis does not require intimate knowledge of the software and can be applied on data obtained from processing on a grid infrastructure.

Empirical modelling is useful in finding performance bugs in parallel code [20] and modelling the performance of big data architectures [22]. The insights from these models are used to optimize the architecture of the software system or determine bottlenecks in processing. Here, we use empirical modelling to determine how the LOFAR `prefactor` performance scales with different parameters.

6.3 Processing Setup

Using the LOFAR software installation described in [75], we processed a typical LOFAR SKSP observation[2], while changing the averaging rate in time and frequency. Changing these averaging parameters will change the final data size (with the data

[2]LOFAR Observation ID L658492, co-ordinates [17h42m21.785, +037d41m46.805] observed by the LOFAR High Band Array for 8 hours between 2018-06-20 and 2018-06-21.

sizes studied shown in Table 6.1). We test the processing time for different averaging parameters by running 15 runs per parameter step.

The data used by the LOFAR surveys is archived at a time resolution of 1 second intervals and frequency resolution of 16 channels per subband (equivalent to 12kHz channel width). While some of the processing steps such as flagging of Radio Frequency Interference and removal of bright off-axis sources produce better results when performed on the high-resolution data, later steps can be performed on averaged data with little impact on the final product quality. To speed up processing, the raw data is averaged in time and frequency, decreasing the input data size to later tasks. The main aims of the LoTSS survey can be accomplished if the final data products from the `prefactor` pipeline are averaged to a time resolution of 8 seconds per sample and frequency of 2 channels per subband. These averaging parameters correspond to a reduction in data size by a factor of 64. Nevertheless, other science cases require less averaging of the data. Our aim is to understand how the processing of this larger data will scale. To measure the scalability of processing, we measure the performance of the `prefactor` pipeline for data sizes between the raw data of 64GB/subband and the averaged data of 1GB/subband. The tested data sizes and parameters are shown in table 6.1, and discussed in Section 6.4.1.

We performed the scalability tests on a dedicated node of the SURFsara `GINA` cluster, f18-01. The node is a typical hardware node used by our production LoTSS processing, however it is dedicated for the tests in order to ensure there is no contamination by other software. The node is described in Section 6.3.3.

We processed the sample data set with the LOFAR `prefactor` pipeline. The `prefactor` version used was the same as we use for the LOFAR SKSP broadband surveys [46]. This software consists of several steps executed in sequence, shown graphically in Figure 6.1. The important `prefactor` steps are as follows. The `predict_ateam` and `ateamcliptar` steps predict the contamination by bright off-axis sources and remove these effects respectively. The `dpppconcat` step is responsible for concatenating 10 Subbands into a single file which is in turn calibrated. The step `gsmcal_solve` is responsible for calibration of the data against a model of the radio sky. The solutions produced by `gsmcal_solve` are used by `gsmcal_apply` and applied to the scientific observation.

6.3.1 Processing Metrics

The goal for our scalability model is to understand the effect of several parameters on the job completion time of LOFAR software. We do this by testing the processing

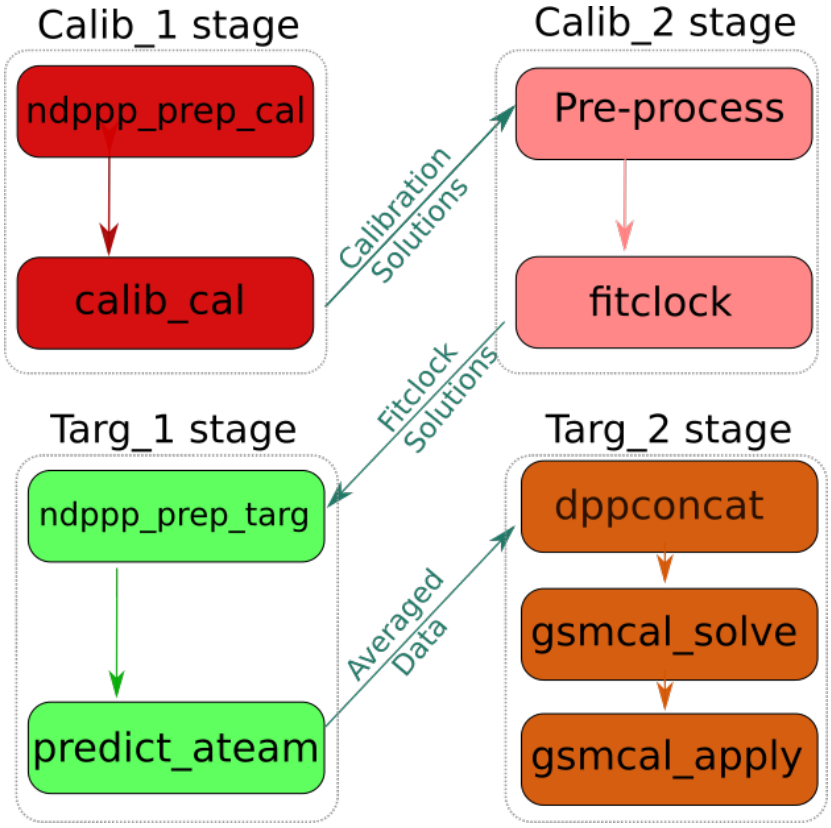


Figure 6.1: The major steps of the `prefactor` DI pipeline.

time for various values of data size, number of CPUs used and sky model size.

Averaging ratio	Time averaging parameter (sec)	Channels per Subband	Averaged Size (Gb)
64x	8	2	1.235
32x	4	2	2.459
16x	2	2	4.906
8x	1	2	9.802
4x	1	4	18.00
2x	1	8	36.72
1x	1	16	66.88

Table 6.1: Averaging parameters and final data sizes tested for the sample LOFAR SKSP observation. The raw data is 64 GB per subband. The LOFAR SKSP data processing uses averaging parameters of 8 seconds and 2 channels per subband. This reduces the raw data by a factor of 64. We highlight the data size used in the LOFAR SKSP survey.

The slowest step of the `prefactor` pipeline is the `gsmcal_solve` step, which performs the gain calibration against a model of the radio sky. This step operates on a concatenated data set that consists of 10 Subbands. We obtain the calibration model through the TGSS sky model creator[3]. By default this service creates a text file describing the sky-model from the TGSS survey [48] as a combination of gaussians and point sources. By default, it sets a threshold of sources brighter than 0.3 Jy. Lowering this threshold creates longer sky-model files with more faint sources, while increasing it will return only the few brightest sources. Since sky model calibration requires converting the sky-model into modelled visibilities [e.g. 55, 60, 127], a longer sky model will increase the time taken to gain calibrate a data set. We created 7 sky models with a flux cutoff ranging between 0.05 Jy and 1.5 Jy. The number of sources in the resulting models are listed in Table 6.2. For production[4], we used the minimum sensitivity parameters for model 3.

It is important to note that the complexity and accuracy of the sky model depend on the direction of observation and the ionospheric conditions in which the observation was performed. As such, our test is only a heuristic for predicting the run-time based on the calibration model length. Additionally, it is notable that the number of sources is exponentially dependent on the minimum sky model sensitivity (seen in Figure 6.2, more in [48, 141]). According to this relationship, even a modest decrease in sensitivity cutoff can significantly decrease the size of the model.

[3] Accessible at the TGSS ADR portal.

[4] The query used to obtain model 3 is at the following link `http://bit.ly/tgss_model`

Sky model #	min sensitivity	# sources
model 1	0.05 Jy	809
model 2	0.1 Jy	503
model 3	0.3 Jy	180
model 4	0.5 Jy	96
model 5	0.8 Jy	49
model 6	1.0 Jy	34
model 7	1.5 Jy	16

Table 6.2: List of test sky models. Model 3 is created with the parameters used in our production processing of LOFAR data. All models include objects within 5 degrees from the centre of the pointing.

Finally, the number of CPU cores (henceforth just 'CPUs') used by each step is a parameter that can be optimized for the entire pipeline. While increasing the number of CPUs can make some steps run faster, requesting jobs that reserve a large number of CPUs will take longer to launch on shared infrastructure. In order to understand the interplay between these effects, we study the queuing time and processing time as a function of the number of CPUs. For the parameter steps we choose to test 1, 2, 3, 4, 8 and 16 CPUs.

6.3.2 Infrastructure Performance

In production, we run hundreds of LoTSS jobs on a cluster supporting several different use cases. The requested resources on this cluster are allocated by a job queue, in our case implemented by the glite workload management system [68]. As queuing jobs can take a significant amount of time, we test the queuing time as a function of the number of requested CPUs. In order to do that, we create test jobs that log the launch time and submit them, requesting 1, 2, 3, 4, 8 and 16 CPUs. We run 10 to 15 tests for each parameter step to ensure that we capture system variability at different times of day during the week and the weekend.

Besides queuing, time is also spent during downloading and unpacking data, as well as packing and uploading the results. Despite using no compression to pack the data, untarring and tarring large files still takes time depending on the system workload. We measure the time taken to transfer and unpack data of different sizes. The data sizes we chose were 0.5GB, 1GB, 2GB, 4GB, 8GB, 16GB, 32GB and 64GB. As our largest data sets are 64GB and our smallest results are ∼0.2GB, these values span a realistic range expected for LOFAR data processing. We test this by uploading mock data to the dCache storage pool at SURFsara and launching a small 1 CPU job

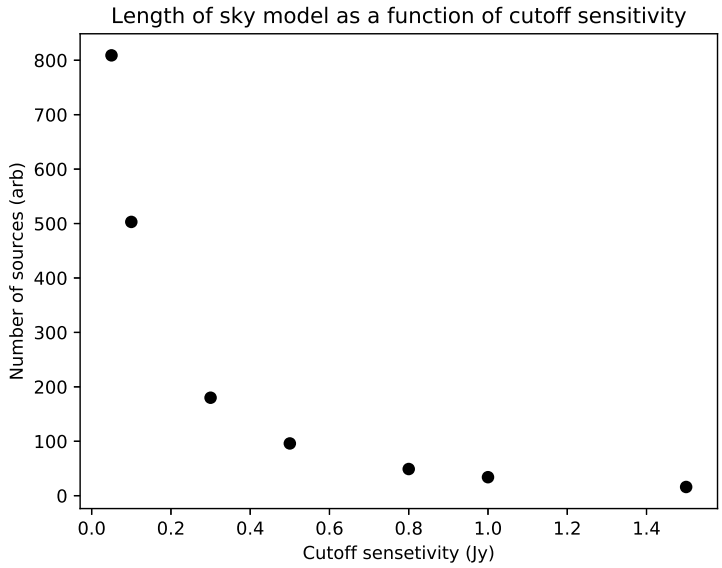


Figure 6.2: The size of the sky model (measured in number of sources) increases exponentially as we decrease the flux cutoff of the model (i.e. increase the sensitivity).

on the cluster, which downloads and untars the data and logs the start time of each step. We present the results of this test in the next section.

Software Versions

For the current test, we use the LOFAR software stack, version 2.20.2 [63]. This software was compiled on a virtual machine and distributed using the CERN CVMFS virtually mounted file system [1]. We use this software version and distribution method as it is the same software version and distribution used to process the data for the LOTSS Data Release 1.

6.3.3 Test Hardware

We test the LOFAR software on a reserved node on the SURFsara `GINA` cluster. The node, f18-01 has 348 GB of RAM, 3TB of scratch space[5]. The CPU is an Intel Xeon(R) Gold 6148 CPU with 40 cores clocked at 2.40GHz. As this hardware node was reserved, there was no other scientific processing aside from our tests, meaning

[5] More detailed specifications are at the `GINA` specification page linked here

there was no resource contention aside for that inherent in the LOFAR software. In the results section, we compare these isolated runs with processing results over the past two years.

6.4 Results

Using a test data set, we tested the LOFAR `prefactor` target pipeline on the SURFsara `GINA` cluster. First we will present the tests done in an isolated environment and then compare them to the run time in production on a shared infrastructure. We will integrate all the results in a complete model which can be used to predict processing time for a variety of parameters. Finally, we will make some predictions on the run time of our processing based on the model and validate these predictions.

Since we are processing a sample data set in the context of the LOFAR Surveys project, we will compare these tests with the production runs of our pipeline. In production, we run the `gsmcal_solve` step with a data size of 1GB, a sky model with 180 sources and 8 CPUs.

6.4.1 Isolated Environment tests

We first tested the LOFAR software in isolation in order to determine the scalability of processing time in terms of data size. We run the entire `prefactor` target pipeline which which removes Direction Independent Calibration errors from a LOFAR science target. In the following sections, we present the models obtained from these tests.

Input Data Size

LOFAR data can be averaged to different sizes based on the scientific requirements. Smaller data sets are processed faster, so it is important to understand the effect of data size on processing time as measured by wall-clock time. We show the processing time for our test data set, averaged to different sizes for several `prefactor` steps in Figures 6.3a- 6.3d and 6.4. We run this test using 8 CPUs. The figures also show linear fits for consecutive pairs of parameter steps, in gray dashed lines, used to help guide the selection of parametric model.

All of the steps show a linear behavior with respect to input data size, while the `gsmcal_solve step` is best fit by two linear relationships, for data smaller and

$$T_{predict\_ateam} = 5.19 \times 10^{-8}\mathcal{S} + 4.20 \times 10^{1} \tag{6.1a}$$

$$T_{ateamcliptar} = 4.57 \times 10^{-9}\mathcal{S} - 8.42 \times 10^{0} \tag{6.1b}$$

$$T_{dpppconcat} = 3.51 \times 10^{-8}\mathcal{S} + 4.20 \times 10^{1} \tag{6.1c}$$

$$T_{gsmcal\_solve} = \begin{cases} 7.38 \times 10^{-7}\mathcal{S} - 8.20 \times 10^{1} & |\mathcal{S} \leq 1.6 \times 10^{10} \\ 1.04 \times 10^{-6}\mathcal{S} - 4.04 \times 10^{3} & |\mathcal{S} > 1.6 \times 10^{10} \end{cases} \tag{6.1d}$$

$$T_{gsmcal\_apply} = 2.07 \times 10^{-8}\mathcal{S} - 1.38 \times 10^{1} \tag{6.1e}$$

Equation 6.1: Equations describing the processing time of five `prefactor` steps as a function of the input data size ($\mathcal{S}$) in bytes.

larger than 16 GB. The linear fit to the run times are shown in Equations 6.1a-6.1e. The equations show the processing time as a function of the data size ($\mathcal{S}$), with the slope in the units of seconds/byte. The fits are also shown in Figures 6.3a to 6.4 as a black dashed line.

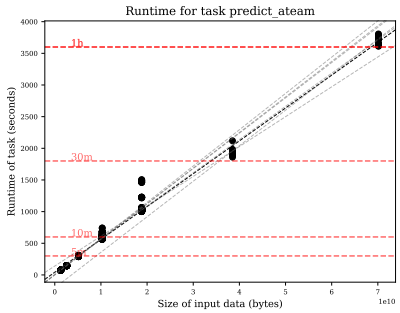
Calibration Model Size

To test the effect of the calibration model size on run time, we test our calibration with several different lengths of the sky model file. We create these models by changing the minimum sensitivity using values ranging from 0.05 Jy to 1.5 Jy. The most sensitive model (0.05 Jy) had 809 sources while the 1.5 Jy model had only 16 sources.

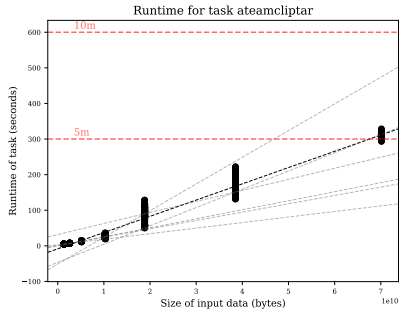
Figure 6.5 shows that the calibration time is directly proportional to the length of the sky model. Figure 6.6 shows the run time as a function of the processing parameter: the cutoff sensitivity. As the relationship between the number of sources and cutoff sensitivity is a power law, here we see the same relationship holding for processing time.

We model the processing time as a function of the cutoff frequency using a power law, and fit the data to the function $y = \alpha \cdot \mathcal{F}^{-k}$. Our fit found the best model to be shown in Equation 6.2, where $\mathcal{F}$ value is the cutoff flux in Jansky and T is the run time in seconds.

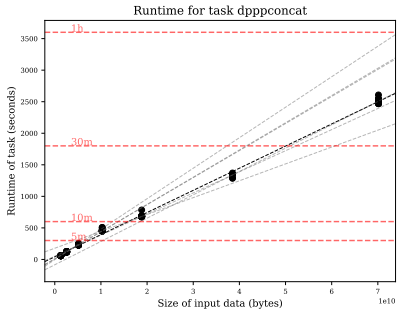
We show four images made from data sets in Figure 6.7. The top left image is calibrated with a 0.05Jy and the other three show the difference between the top left image and the images created from the 0.3Jy, 0.8 Jy and 1.5 Jy data. The statistics for the four images, taken from the regions in green on Figure 6.7) are shown in Table 6.3. We discuss the implication and caveats of these results in Section 6.5.



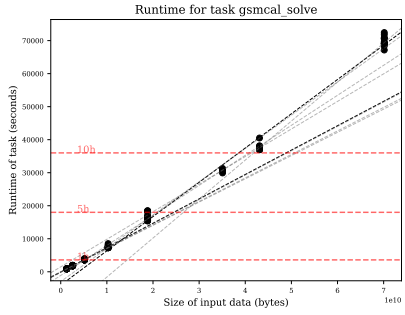
(a) Tests of the `predict_ateam` step for input data size ranging from 1GB to 64 GB. This step calculates the contamination from bright off-axis sources. Dashed lines are shown connecting each pair of points, to highlight the trend. We can see that the data can be described well by a linear model. We show the model in Equation 6.1a in black.



(b) Tests of the `ateamcliptar` step for input data size ranging from 1GB to 64 GB. This step removes the contamination from bright off-axis sources. We can see that the data fits a linear model. We show the model in Equation 6.1b in black.



(c) Tests of the `dpppconcat` step for input data size ranging from 1GB to 64 GB. This step concatenates 10 files into a single measurement set. We can see that the data fits a linear model. We show the model in Equation 6.1c in black.



(d) Tests of the `gsmcal_solve` step for input data size ranging from 1GB to 64 GB. This step performs gain calibration of the concatenated data set against a sky model. It is the slowest and most computationally expensive **`prefactor`** step. We fit two linear models, for data below 16GB and above 16GB. We can see the two models, shown in (Equation 6.1d) as two black dashed lines, intersecting at 1GB.

Figure 6.3: Plots of the run time as a function of input data size

[6] Using the command wsclean -absmem 50 -niter 3 -size 4096 4096 -scale 20asec

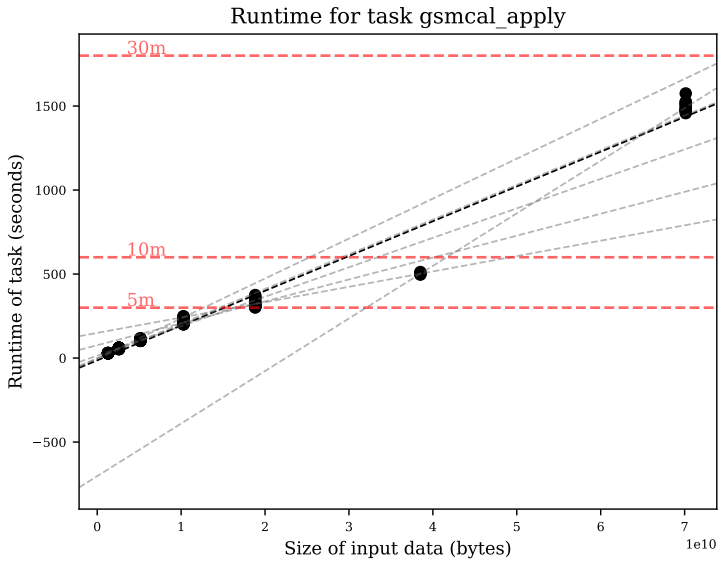


Figure 6.4: Tests of the `gsmcal_apply` step for input data size ranging from 1GB to 64 GB. This step applies the calibration solutions to the data. We can see that the data fits a linear model, described in Equation6.1e, as the black dashed line.

$$T = 1185 \cdot \mathcal{F}^{-0.854} \tag{6.2}$$

Equation 6.2: Processing time for the `gsmcal_solve` step as a function of the flux cutoff of the calibration model ($\mathcal{F}$) in Jansky

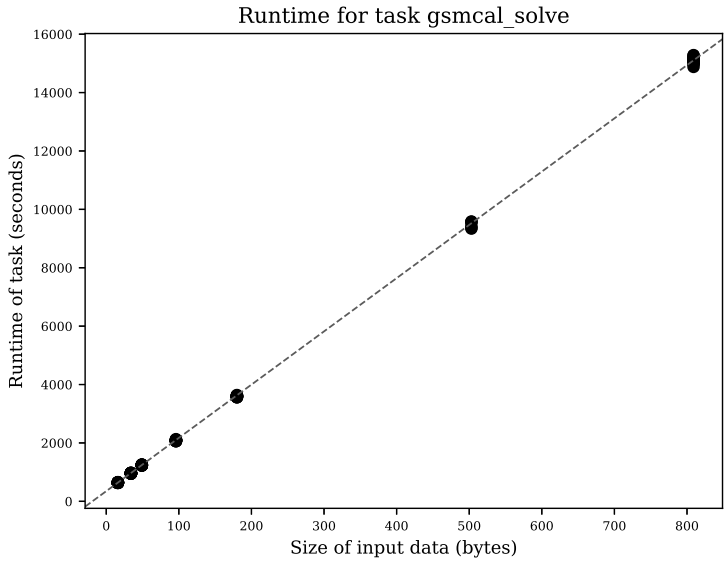


Figure 6.5: The processing time of the `gsmcal_solve` step is linear with the size of the sky model as measured by the number of sources.

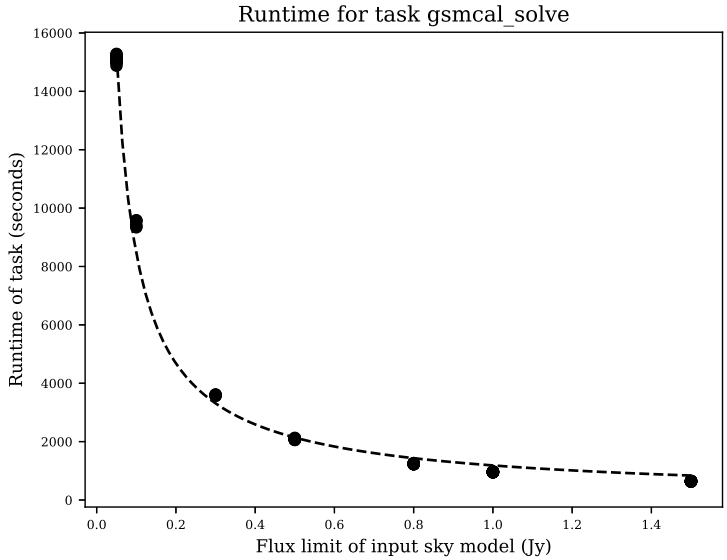


Figure 6.6: The run time of the `gsmcal_solve` step as a function of the cutoff sensitivity is not linear. As shown in Figure 6.2, the number of sources increases exponentially as the minimum sensitivity decreases. The dashed line shows the model fitted in Equation 6.2.

Calibration Model Flux Cutoff	# of sources	RMS Noise (Jy)
0.05Jy	809	0.00402834
0.3 Jy	180	0.00402311
0.8 Jy	49	0.00404181
1.5 Jy	16	0.00410204

Table 6.3: Statistics for an empty region for the four images shown in Figure 6.7. The 0.3Jy model, here shown shaded in gray, is the one used in production.

$$T = 503.37 + \frac{3062.6}{\mathcal{N}} \tag{6.3}$$

Equation 6.3: Processing time for the `gsmcal_solve` step as a function of ($\mathcal{N}$), the Number of CPUs used by the process.

Number of CPUs

One further parameter that can be optimized is the number of CPUs requested when the job is launched. We investigated the processing speedup as a function of the number of CPUs for the `prefactor` target pipeline. From the steps tested, only the `gsmcal_solve` step shows a significant speedup as the number of CPUs is increased. The run time of this step is an inverse power law with respect to the number of CPUs as seen in Figure 6.8. Unlike the solving step, the step applying the calibration solutions (`gsmcal_apply`) is constant in time with respect to the number of CPUs as seen in Figure 6.9. The difference in performance is most likely because the `gsmcal_apply` code uses a parallel for loop to calculate antenna gains while `gsmcal_apply` does not.

We fit a model with processing time inversely proportional to the number of CPUs used. We show the resulting model in Equation 6.3, with the parameter ($\mathcal{N}$) being the number of CPUs used.

6.4.2 Queuing Tests

Aside from performance of the LOFAR software, we measured the queuing time at the `GINA` cluster, as a function of the number of CPUs requested. This data was obtained between 16 Nov 2018 and 10 Dec 2018 for 1, 2, 3 ,4, 8, and 16 CPUs per job. A histogram of the queuing time for these jobs is shown in Figure 6.10. Statistics for these runs are in Table 6.4. We use the 75th percentile of the queuing time for

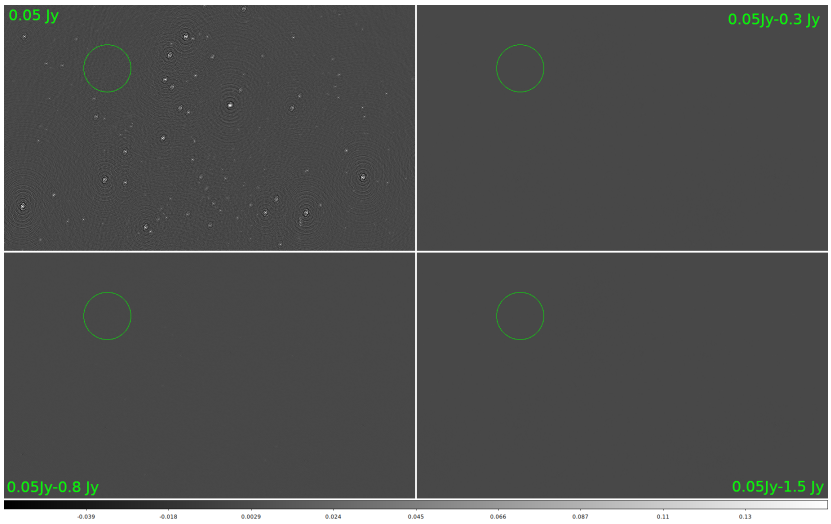


Figure 6.7: Four images made using the `wsclean` software [84] from the data set[6]. The four images were calibrated with sky models of various flux cutoffs ranging from 0.05Jy (top left) to 1.5Jy (bottom right). Flux statistics for the green regions in the four images are listed in Table 6.3. The top right, and bottom two quadrants show the pixel difference between the 0.05Jy image and the 0.3Jy, 0.8Jy and 1.5Jy images respectively. The four images are all on the same scaleThe green region shows the same area in all four quadrants.

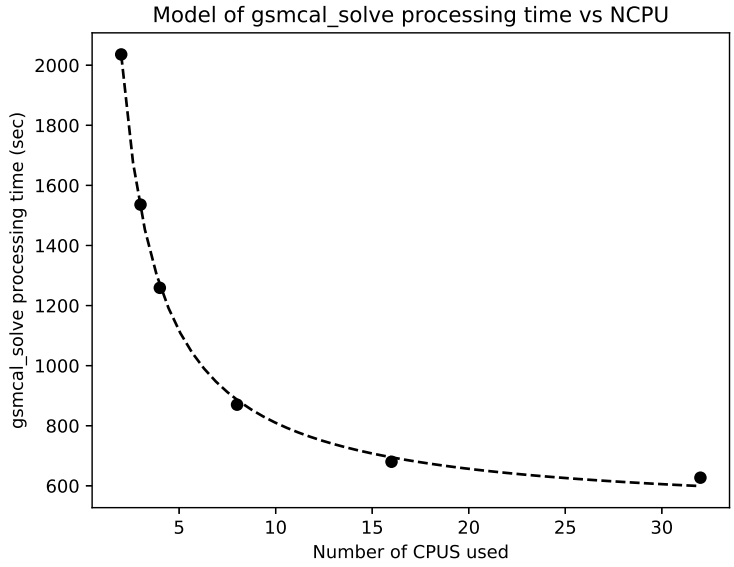


Figure 6.8: The processing time of the `gsmcal_solve` step decreases exponentially with the number of CPUs requested. The model in Equation 6.3 is shown in a dashed line. As this is a 1/x model, it shows diminishing returns past 16 CPUs.

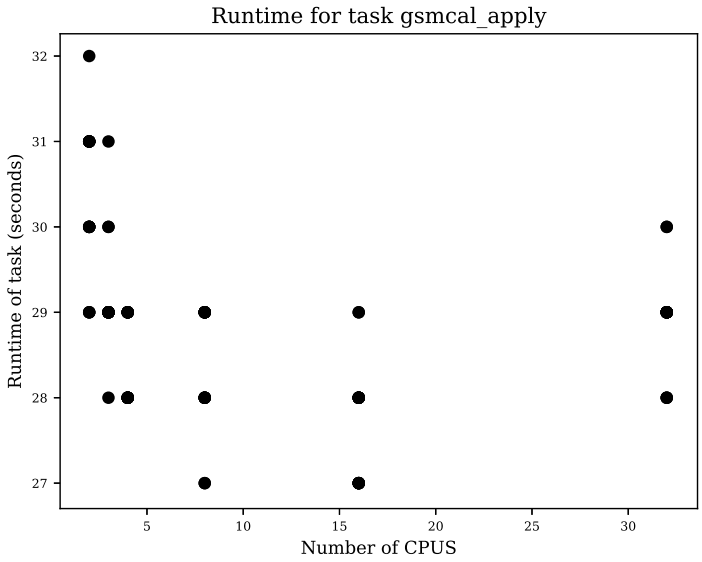


Figure 6.9: The step that applies the calibration solutions, `gsmcal_apply`, does not show a speedup when run on multiple cores, as all runs take roughly 30 seconds to complete.

$$T = \begin{cases} 49.3 \cdot \mathcal{N} + 120 & |\mathcal{N} \leq 4 \\ 726 \cdot \mathcal{N} - 3071 & |\mathcal{N} > 4 \end{cases} \tag{6.4}$$

Equation 6.4: The model for the Queuing time as described by two linear models.

each parameter step to fit a model. This scenario will include 75% of runs and is a good trade-off between ignoring and including outliers.

We fit two linear models for this queuing time. One model for 1-4 CPUs and one for 4-16 CPUs. The model, as a function of the number of CPUs $\mathcal{N}$ is in equation 6.4. The two models are plotted against the 75th percentile of the queuing times (last column in Table 6.4) in Figure 6.11.

6.4.3 Transfer and Unpacking Time

We tested the downloading and unpacking time for data sizes ranging from 512MB to 64GB. We discovered that the unpacking of files below 64GB scaled linearly with file size, however unpacking individual data sets larger than 16GB becomes considerably slower than downloading it.

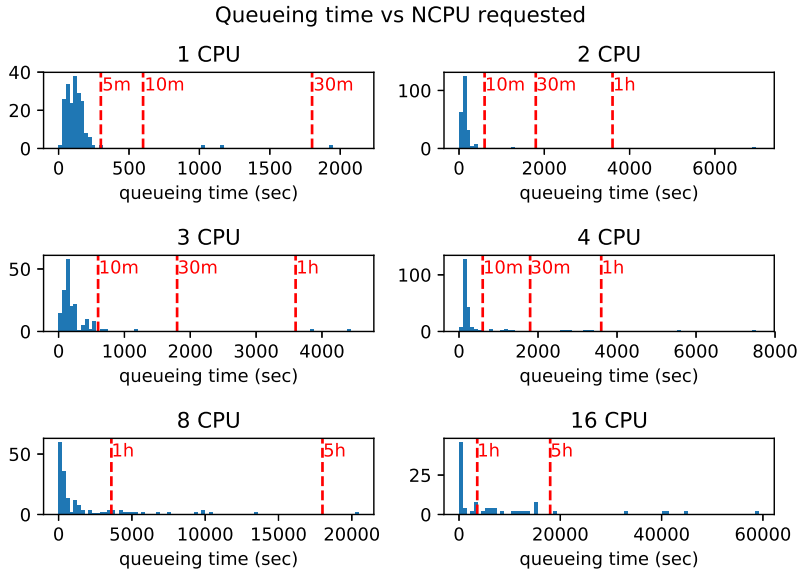


Figure 6.10: Test randomly submitting jobs to the GINA with different number of requested CPUs. The long tail for 8 and 16 CPU jobs shows that some jobs can take several hours to launch.

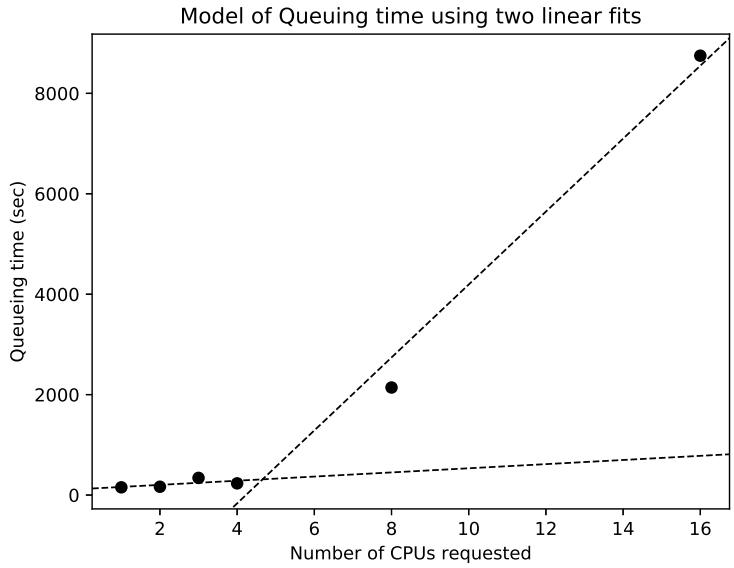


Figure 6.11: The queuing model built from two linear fits to the queuing times. We use the 75th percentile of the queuing data as a upper bound of job queuing.

NCPU requested	Mean time (sec)	Median time (sec)	75th percentile (sec)
1 CPU	150.5	116.2	154.1
2 CPU	201.1	125.8	165.8
3 CPU	296.2	152.0	243.0
4 CPU	498.9	167.7	233.7
8 CPU	1944.2	428.4	2142.4
16 CPU	7079.0	696.4	8750.6

Table 6.4: Statistics for queuing time for different values of CPUs requested. Queuing time for jobs requesting less than 8 CPUs is typically less than five minutes. It can drastically increase for larger jobs.

$$T = 5.918 \times 10^{20} \cdot \mathcal{S}^{2.336} \tag{6.5}$$

Equation 6.5: Model of the downloading and extracting time as a function of the data size ($\mathcal{S}$) in bytes.

Figure 6.12 shows the histogram of the download tests, and Figure 6.13 displays the tests as a function of data size. Both figures show that extracting of the 32 and 64GB data sets has more slow outliers than the downloading of this data.

We fit a power law model to the time taken to transfer and unpack the data. In this case, we also consider the 75th percentile of these times in order to capture the majority of runs and ignore outliers. The plot of the data and our model can be seen in Figure 6.13 and the model is in Equation 6.5, as a function of the input data size, $\mathcal{S}$ in gigabytes.

6.4.4 Comparison with production runs

Over the past two years, the LOFAR software has been running in production and collecting data on run time for each processing step. We have saved detailed logs for these runs starting in July 2018. We can compare this to the isolated model in order to determine the overhead incurred by processing LOFAR data on shared nodes.

Using the logs recorded by our processing launcher [7], we made plots showing the processing time for the downloading and extracting, and for the slowest steps, `ndppp_prepcal` and `gsmcal_solve`. The results are shown in Figures 6.15 and 6.16. We include predicted extract times from Section 6.4.3 as vertical dashed lines for both plots. The agreement between our model and production runs are an encouraging result for future software performance modelling.

[7] GRID_PiCaS_Launcher, `https://github.com/apmechev/GRID_picastools`

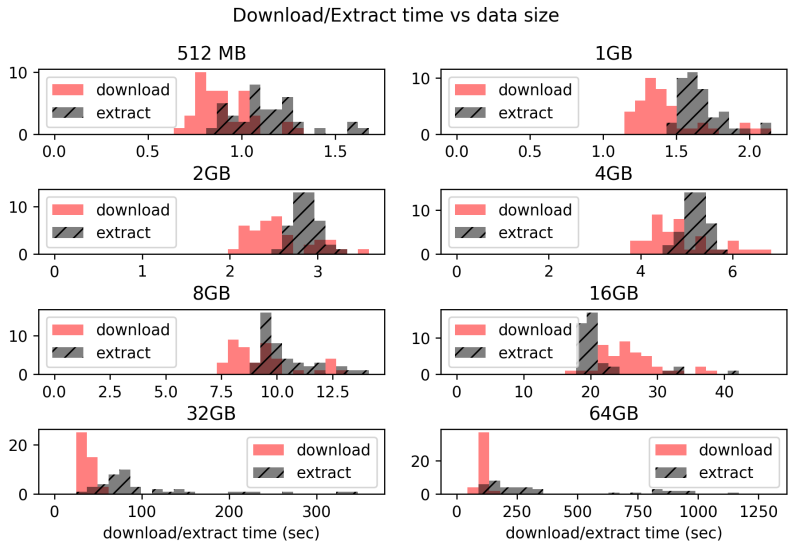


Figure 6.12: A histogram of the download and extracting times of multiple data sizes on the GINA worker nodes. Download and extract times are comparable for data up to 8GB, however above that, the extracting time dominates.

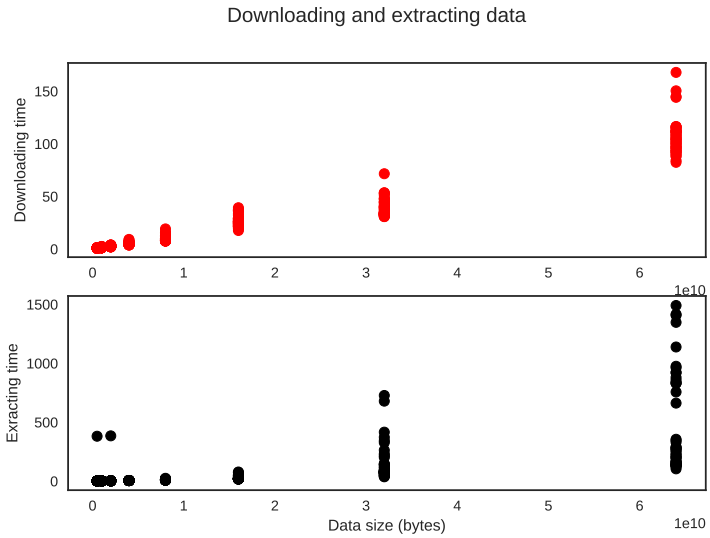


Figure 6.13: A scatter plot of the download and extracting times of multiple data sizes on the GINA worker nodes. The difference between download and extract time for the 32 and 64 GB data sets can be seen.

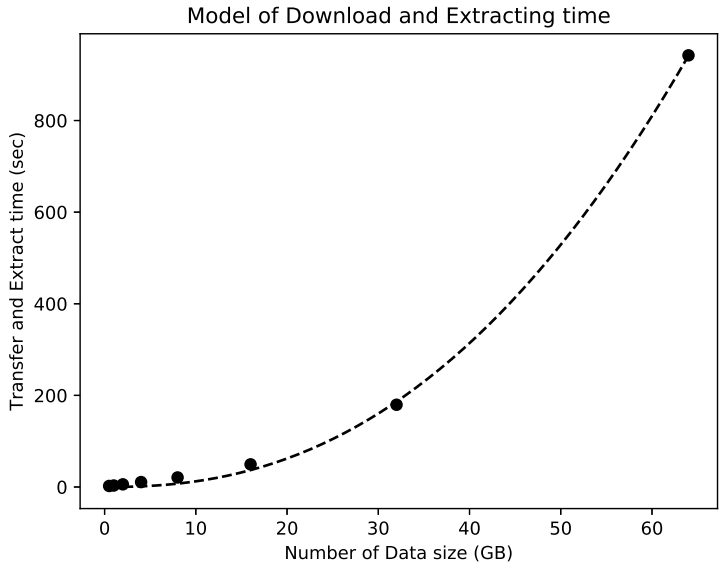


Figure 6.14: Fit of an exponential model to the Download and Extraction time for different data sizes. For the transfer overhead, we took the 75th percentile from the data shown in Figure 6.12. The model in Equation 6.5 is shown in a dashed line.

Finally, we present Figure 6.17 which shows a comparison of `gsmcal_solve` run times and our model's prediction for a 1GB data set. Figure 6.18 plots the processing time vs data size for these production runs and includes the model from Equation 6.1d. The significant overhead incurred on a shared infrastructure can be noted.

6.4.5 Complete Scalability Model

To incorporate all our data into a complete model, we consider the slowdown of each parameter as a multiplier to the time taken to process our base run. We incorporate the models for each parameter above for the model of the run time. We add the transfer and queuing time to the processing time to obtain a final function of all our parameters. We can use this function to predict the processing time for an arbitrary data set.

The final performance model for the slowest steps, `gsmcal_solve`, `dppconcat` and `predict_ateam` are in Equation 6.6.

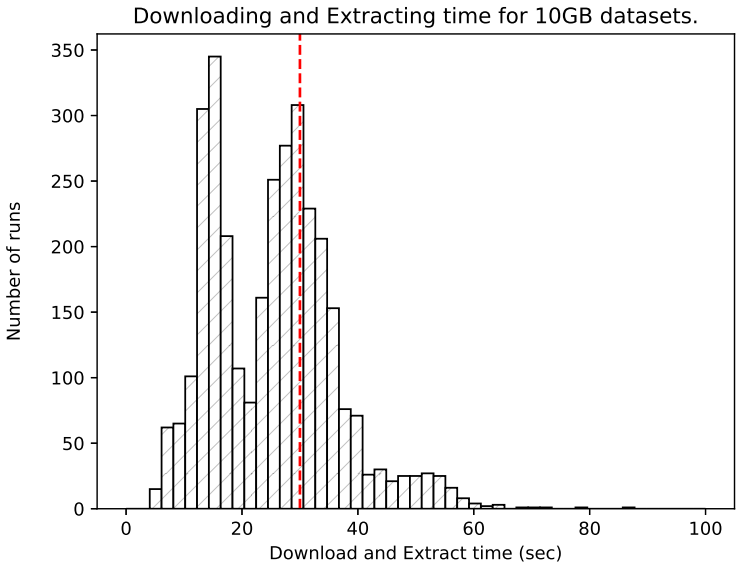


Figure 6.15: Downloading and extracting time for 10 1GB data sets performed in our production environment. Data from this test ranges from July 2018 to January 2019. The dashed red line shows the prediction obtained from section 6.4.3. We see a bimodal distribution corresponding to 10 GB data (right peak) and data averaged further to 5GB (left peak).

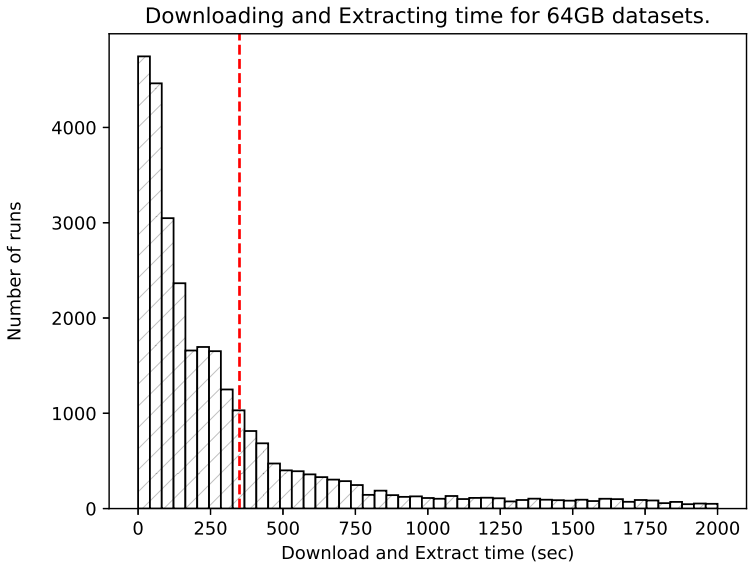


Figure 6.16: Downloading and extracting time for a 64GB data set performed in our production environment. Data from this test ranges from 07/2018-01/2019. The dashed red line shows the prediction obtained from Figure 6.3d in Section 6.4.3.

6.5 Discussions and Conclusions

The goal of this work is to understand the performance of the LOFAR Direction Independent Pipeline as processing parameters are changed. We modify several parameters and compare the wall clock time taken to process the data. Finally, we study data from queuing jobs and downloading data, in order to fully model infrastructure overheads. We compare our model with past runs of the software and discuss the results and implications. We discuss the utility of this model for current and upcoming LOFAR projects. Finally, we note the effectiveness of this modelling technique for understanding the performance of large-scale processing of astronomical data.

6.5.1 Software Performance

We performed several tests to determine the scalability of LOFAR processing with respect to several parameters. We outline our findings below as well as their implications for processing in context of the LOFAR surveys and other LOFAR projects.

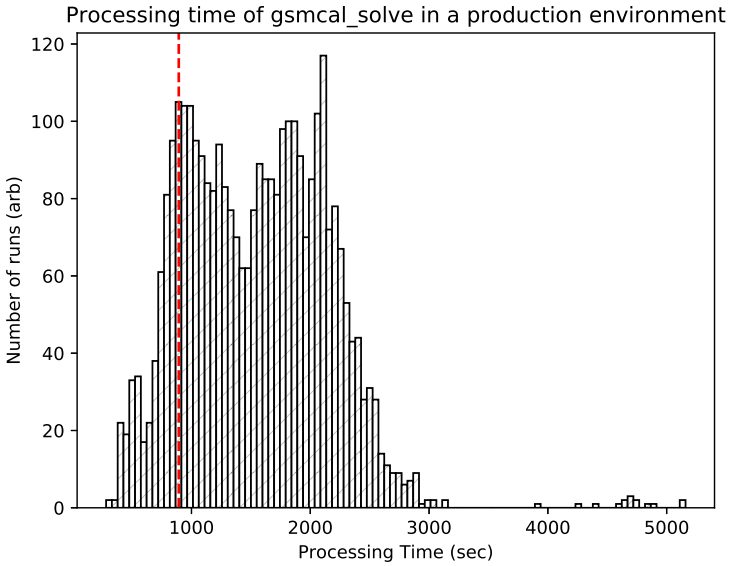


Figure 6.17: Processing time for the `gsmcal_solve` step in a production environment. Data from this test ranges from 07/2018-01/2019. The dashed red line shows the prediction for a 1GB run, obtained from section 6.4.3. We see two distributions, which correspond to data averaged to 1GB and 512 MB. It should be noted that the left peak corresponds to 512MB data, as seen in Figure 6.18.

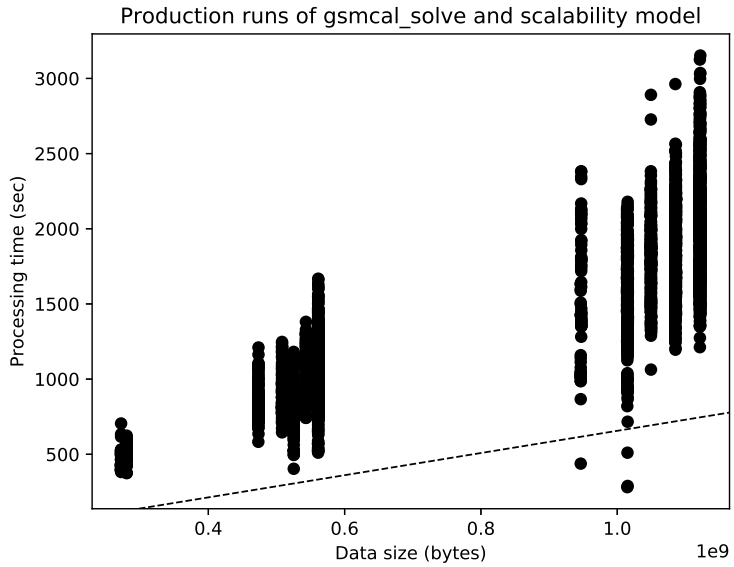


Figure 6.18: The scalability model for processing data through the `gsmcal_solve` step, shown in a dashed line. The scatter plot shows the performance for production runs of this step between July 2018 and January 2019. The two large clusters are for data products that are 1.0 and 2.0 GB respectively.

$$t_{infrastructure} = \begin{cases} 49.3 \cdot \mathcal{N} + 120 & |\mathcal{N} \leq 4 \\ 726 \cdot \mathcal{N} - 3071 & |\mathcal{N} > 4 \end{cases} + 2 \cdot 0.056 \cdot \mathcal{S}^{2.336} \tag{6.6a}$$

$$t_{gsmcal\_solve} = t_{infrastructure} + [3566 \cdot \frac{1}{3.012}\mathcal{F}^{-0.854} \cdot (0.1412 + \frac{0.8589}{\mathcal{N}})] \cdot \begin{cases} 7.38 \cdot 10^{-7}\mathcal{S} | \mathcal{S} \leq 16 \\ 1.04 \cdot 10^{-6}\mathcal{S} | \mathcal{S} > 16 \end{cases} \tag{6.6b}$$

$$t_{dpppconcat} = t_{infrastructure} + 3.51 \times 10^{-8}\mathcal{S} + 4.20 \times 10^{1} \tag{6.6c}$$

$$t_{predict\_ateam} = t_{infrastructure} + 5.19 \times 10^{-8}\mathcal{S} + 4.20 \times 10^{1} \tag{6.6d}$$

Equation 6.6: Model of the total time of the most computationally expensive steps for the parameters $\mathcal{N}$, Number of CPUs; $\mathcal{S}$, Size of data in bytes and $\mathcal{F}$, cutoff calibration model flux in Jansky. These models include processing times, as well as infrastructure overheads. As the model for the queuing, downloading and uploading time does not change for different processing steps, we decide to keep it separate for clarity. The complete scalability models for the rest of the steps can be derived similarly, however are omitted here as they consist of the minority of processing time for LOFAR DI processing.

Data Size

We tested broadband LOFAR data ranging in size by a factor of 64, and discover that all our processing steps scale linearly in time with respect of the input data size. We learn that for input data above 16GB, the slope of our scaling relation is higher than for the smaller data sets. The linear scaling of our processing suggests that projects interested in processing massive LOFAR data sets can scale well in terms of processing time.

As the calibration step concatenates 10 input Subbands, data larger than 16GB shows a higher slope (Figure 6.3d), meaning they take longer to process than smaller data. This can be attributed to the large memory requirement for data larger than 160GB which are is likely due to the additional memory requirements of the minimization algorithm. Splitting the performance model in two also helps make a more accurate processing time prediction as fitting a single linear model would have a

large negative y-intercept, predicting negative processing times for data smaller than 2GB.

Our analysis shows the following results. Overall, the slowest step was the `gsmcal_solve` step, and its run time scales more strongly with data size than the other steps (equation 6.1d has the steepest slope). This suggests that as data sizes increase, `gsmcal_solve` will increasingly dominate the processing time over the other steps (As seen in Figures 6.3a and 6.3d). This effect is especially prominent for data larger than 16GB (160GB when 10 Subbands are concatenated). As such, it is recommended to avoid calibration of data larger than 160GB. This limitation suggests that science requiring operations on non-averaged data sets, such as long-baseline imaging, will require significant computational requirements for high-fidelity images.

Calibration Model Size

We discover that the calibration time scales linearly in as a function of the length of the calibration model, however as a power law with respect to the model's cutoff sensitivity. This is because of the (expected) power law relation between the number of sources and cutoff sensitivity, seen in Figure 6.2. We can use this discovery to accelerate the processing time by increasing the flux cutoff to the LOFAR direction independent calibration to 0.5 Jy. Doing so will execute the calibration step in 60% of the time, saving 83 CPU-h per run. Over the remaining 2000 `prefactor` runs left in the LOTSS project, this change in sensitivity will save more than 167k CPU-hours.

Figures 6.7 show a data set calibrated with sky models with cutoff sensitivities listed in Table 6.3, and figures 6.19 and 6.21 show the calibration solutions obtained by calibrating with sky models of cutoff ranging from 0.05 Jy to 1.5 Jy. These results suggest that performing gain calibration with less complex, and thus smaller, calibration models will not degrade image and solution quality while taking less than 20% of processing time. Table 6.3 also confirms this result.

While this result is encouraging, there are caveats suggesting future study is required. The results we present are for a single observation, and has not been processed through the Direction Dependent Calibration pipeline. This pipeline produces final images used for scientific research. Future work will need to confirm that smaller calibration models used in the `prefactor` pipeline do not degrade the quality of these final images. Nevertheless if this result holds, the calibration model threshold can be decreased for a wide range of LOFAR projects, significantly saving processing time and computing resources.

Comparison with production runs

When comparing our model's prediction with real processing runs over the past six months, we note that there are considerable overheads when running on a shared infrastructure vs. when processing data on an isolated (Figure 6.18). The overhead in processing is roughly a factor of two-three from our model. This discrepancy suggests that a model for `gsmcal_solve` needs to be built using data when running on a shared environment, to better predict processing time.

6.5.2 Infrastructure Performance

We tested downloading and extracting LOFAR data of various sizes. Both downloading and extracting are shown to be linear in time with respect to the data size for data up to 32 GB. Beyond those sizes, there is more scatter in data extraction due to high file-system load. This is because load on the worker node's file-system can be unpredictable and can affect the data extraction times negatively. Nevertheless, when comparing our extraction tests and processing for the past 6 months, the predictions by our models (Figure 6.12) correspond to the production runs (Figures 6.15 and 6.16).

Part of the LOFAR SKSP processing is done on shared infrastructure, which requires requesting processing time ahead of time for each grant period. Being able to predict the amount of resources required to process data each grant period is required to make a reliable estimate on what resources to request. These results can be used by other projects sharing the SURFsara `GINA` cluster to predict their processing time before submitting jobs.

6.6 Applications and Conclusions

Our performance model shows that it is possible to predict the processing time and computational resources used by a complex astronomical pipeline. Our results suggest that LOFAR LoTSS processing can be further optimized without sacrificing quality of the final product. Additionally, our results are transferable to other scientific pipelines that process LOFAR data with different parameters. Any pipeline that performs gain calibration or application of calibration solutions will benefit from these results.

In order to provide LOFAR processing as a service to scientific users, we need to estimate the processing time for each request. We need this estimation in order to determine whether the user has sufficient resources left in their resource pool. Knowing the performance of the software pipelines as a function of the input parameters will help predict the run time for each request and the resources consumed. Knowing this will make it possible to notify users how long the request will take and how much of their quota will be depleted. It is important to note that while this model is specific to LOFAR processing, the method we detail can be used by other scientific teams in order to predict the computational requirements for their pipelines. Doing this is necessary if large scale scientific processing is to be offered as a service to the wider community.

Finally, a performance model of the LOFAR software will help make predictions on the time and resources needed to process data for other telescopes such as the Square Kilometer Array (SKA). Once operational, the SKA is expected to produce Exabytes of data per year. Processing this data efficiently requires understanding the scalability of the software performance to facilitate scheduling and resource allotment. Overall, we show that our method helps guide algorithm development in radio astronomy, can be used to predict resource usage by complex pipelines and will be promising in optimizing data processing by future telescopes.

6.A Calibration Solutions for the sky model tests

The output of the calibration step is a data set corrected for direction independent effects, as well as a set of calibration solutions. Figures 6.19 and 6.21 show the calibration solutions for core stations obtained when calibrating with sky models with minimum flux cutoffs of 0.05, 0.3, 0.8 and 1.5Jy. Much like in Figure 6.7, we can see that there is no significant difference between the calibration solutions for these stations. As a note, the naming scheme for LOFAR stations is CS/RS for core/remote stations, three digits for station number, HBA0/HBA1 for High Band antennas and LBA0/LBA1 for low-band antennas. The 0,1 suffixes correspond to sub-arrays in the core stations which can be correlated separately. Additionally, the CS/RS is replaced with the 2-letter country code for international stations[136].

We compare the phase solutions for stations CS032HBA0 and CS003HBA0 and the reference station for two different calibrations in Figure 6.20. We note that this difference is within 0.3 rad for the entire observation. Combined with the results in the other two plots, our results suggest that the calibration solutions do not degrade when calibration is done with a smaller sky model.

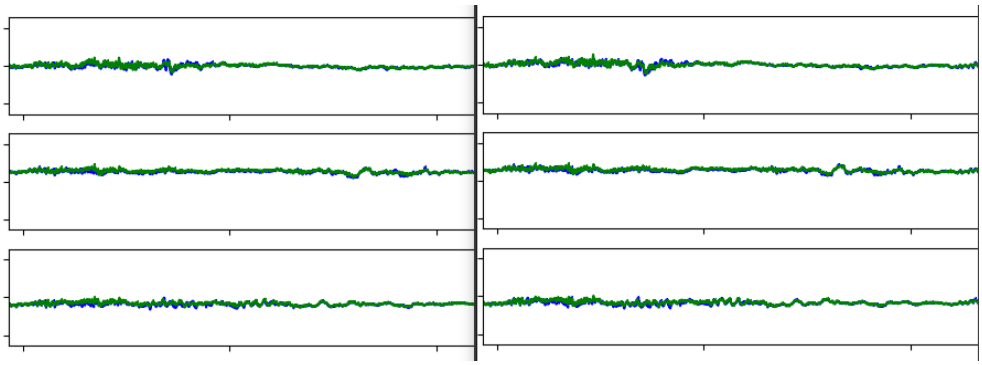


Figure 6.19: The calibration (phase) solutions for the test data set obtained when calibrating with sky models of 0.05 Jy cutoff (left) and 0.3Jy cutoff (right). The data shows the phase solutions for baselines including stations CS003HBA0, CS003HBA1 and CS004HBA0, with respect to the reference station, CS001HBA0. The right solutions were obtained using the production calibration model. We do not see any improvement in results in the left figure, which took twice as long to obtain.

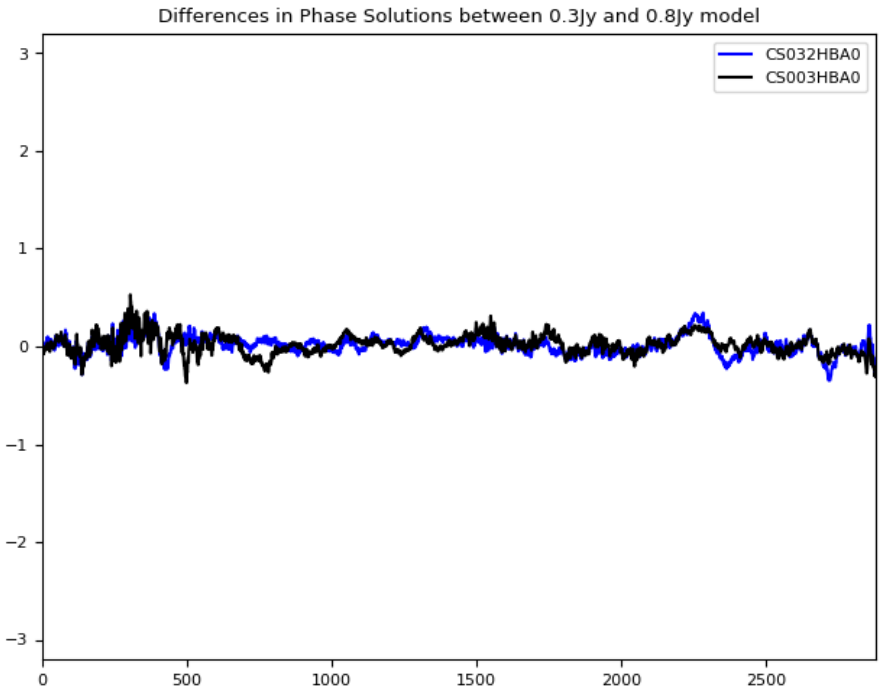


Figure 6.20: Difference of phase solutions between calibrations with the 0.3Jy and 0.8Jy sky models. The solutions for both stations are around zero phase for the duration of the observation.

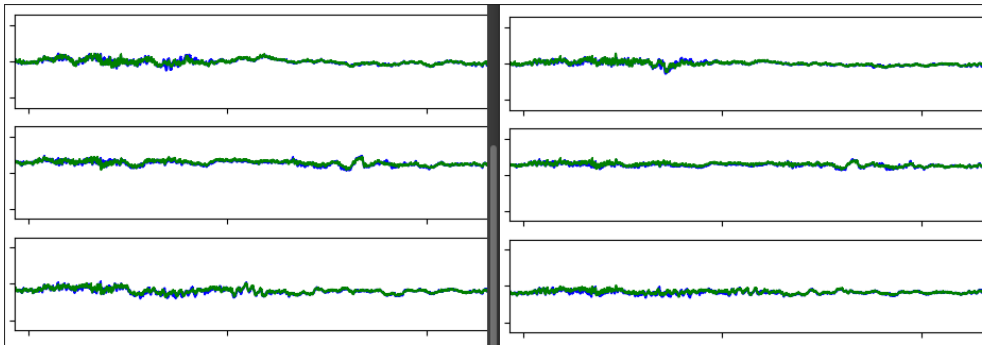


Figure 6.21: The calibration (phase) solutions for the test data set obtained when calibrating with sky models of 0.8 Jy cutoff (left) and 1.5Jy cutoff (right). The data shows the phase solutions for baselines including stations CS003HBA0, CS003HBA1 and CS004HBA0, with respect to the reference station, CS001HBA0. We can see that the calibration solutions shown here are not significantly different than those shown in 6.19, despite taking a fraction of the processing time.

6.B Parametric model parameters and fit accuracy

In this section, we note the uncertainties to the models fit in Equations 6.1-6.5.

6.B.1 Fits quality of run time vs input size model

The models of the processing time vs input size were fit as a linear regression. In this work we present such models for the `gsmcal_solve`, `gsmcal_apply`, `dpppconcat`, `predict_ateam` and `ateamcliptar`, the five slowest steps. The resulting models, calculated by the `scipy linregress`[51] routine, are shown in Equation 6.1. We present the R^2 values, P values and standard error below, in Table 6.5.

`prefactor` step	R^2	P value	Standard Error
predict_ateam	0.996	0	1.92×10^{-10}
ateamcliptar	0.979	0	3.94×10^{-11}
dpppconcat	0.999	1.2×10^{-128}	1.78×10^{-10}
gsmcal_solve <=16GB	0.995	3.12×10^{-75}	6.80×10^{-9}
gsmcal_solve >16GB	0.951	7.07×10^{-40}	1.58×10^{-8}
gsmcal_apply	0.989	5.6×10^{-82}	3.12×10^{-10}

Table 6.5: Fit parameters for the models in Equation 6.1.

$$\begin{bmatrix} 6.83 \times 10^{2} & -1.94 \times 10^{-1} \\ -1.94 \times 10^{-1} & 5.81 \times 10^{-5} \end{bmatrix} \tag{B.7}$$

Equation 6.7: The covariance matrix of the parameters in model in Equation 6.2.

$$\begin{bmatrix} 171.94 & -504.11 \\ -504.11 & 2322.95 \end{bmatrix} \tag{B.8}$$

Equation 6.8: The covariance matrix for the parameters for the model predicting run time vs Number of CPUs used, shown in Equation 6.3.

6.B.2 Fit of run time vs calibration model flux cutoff

The run time vs Flux cutoff model shown in Equation 6.2 is defined by the equation $y = a \cdot x^{-k}$ and two parameters, a and k. The covariance matrix for these two parameters is shown in Equation 6.7. The standard deviation for the fit of the parameters a and k is 26.134 and 7.624×10^{-3} respectively.

6.B.3 Fit of the NCPU model

The covariance matrix for the fit parameters of equation 6.3, a and k in $y = a + \frac{k}{\mathcal{N}}$ are shown in Equation 6.8. The standard deviation of the fits for a and k are 13.11 and 48.20 respectively.

6.B.4 Fit for the queuing time model

The statistics of the model fit parameters for the queuing time model (Equation 6.4) are in Table 6.6. The queuing model is fit to the 75^{th} percentile of the queuing times for each parameter step. Since this results in a single number for each step, the model's P values are larger than the models from the other sections.

Value of $\mathcal{N}$	R^2	P value	Standard Error
$\mathcal{N} \leq 4$	0.382	0.381	37.086
$\mathcal{N} > 4$	0.986	0.075	86.293

Table 6.6: Goodness of fit parameters for the model in Equation 6.4. Since the model is split into two parts, we treat each section as a single linear model.

$$\begin{bmatrix} 2.53 \times 10^{-4} & 1.08 \times 10^{-3} \\ 1.08 \times 10^{-3} & 4.60 \times 10^{-3} \end{bmatrix} \tag{B.9}$$

Equation 6.9: The covariance matrix for the parameters for the model for Download and Extract time, shown in Equation 6.5.

6.B.5 Fit of the download and extract model

Equation 6.9 shows the covariance matrix for the two parameters a and k, $y = a \times 10^k$ with the best fit values shown in Equation 6.5. The standard deviations of the fits for a and k are 0.016 and 0.068 respectively.

7 Continuous Integration of LOFAR Science with AGLOW

7.1 Introduction

The Low Frequency Array (LOFAR) radio telescope is a European low frequency aperture synthesis telescope operated by the Netherlands[130], that observes astronomical objects in the 20MHz-240MHz frequency range. It is designed as a flexible telescope able to serve multiple science cases, such as large surveys, transient studies, galactic and extra-galactic sources, radio spectroscopy, as well as long-baseline interferometry.

LOFAR data is stored as a Measurement Set **[Casacore_MS]** and archived at one of the three Long Term Archive locations[1]. The Measurement Set standard and the high time and frequency resolution of the archived data allows a single observation to serve multiple science cases. Additionally, the standard data structure is supported by several software packages which can manipulate the data in the Measurement Sets according to the scientific goal and include their own data products to the Measurement Set.

The scientific data products for each project are produced by a series of transformations of the input data, done by a set of scripts termed a scientific pipeline. These pipelines are typically written as Python scripts which call various software packages with parameters specific to the pipeline and science case.

Both the scientific pipelines and the underlying processing software are developed at a high pace. Often new updates in the software introduce bugs in the scientific pipelines, or upgrades to the scientific scripts introduce errors in the sceintific image. In order to catch these errors early and to ensure that the scientific pipelines

[1] https://lta.lofar.eu/

do not degrade image quality, we have built a suite to run tests for LOFAR's scientific pipelines. We run our integration tests in the same environment as our scientific processing to ensure that deployed software will not affect the processing of LOFAR surveys. Upon success, we deploy the working scientific software as a complete software image.

Contributions: The main features of our work are the following:

- Ability to automatically execute tests in a production environment.
- Support for multiple, concurrent scientific pipelines and easy integration for future scientific tests.
- Automatic time stamps of data products allowing comparison of data quality over time.
- Automatic versioning of the nightly releases with a possibility of automating deployment and versioning.
- Storing all software versions and script repositories as part of the workflow to make LOFAR processing reproducible and portable.

Outline: The organization of this manuscript is as follows: We provide details about two LOFAR pipelines and their scientific value in Section 7.2. We note previous work in Continuous Integration for scientific processing in Section 7.3. We detail the CI workflow in section 7.4. We discuss our results and show quality metrics for several regions in section 7.5. Finally, we conclude and make remarks on future development in section 7.6.

7.2 Background

One of the major Key Science Projects for the LOFAR telescope is the Surveys Key Science Project (SKSP)[**LOFAR_SKSP**]. The goal of the SKSP project is to create multiple large-scale maps of the northern sky at low frequencies and unmatched resolution and sensitivity. To reach this unmatched sensitivity and to serve multiple science cases, each observation needs to be integrated for several hours. Observations are stored at a high time and frequency resolution to be able to serve multiple scientific surveys. One of the surveys projects is the LOFAR Two-Meter Sky Survey, LoTSS [109]. This survey is expected to produce more than 3000 8 hour observations, each of which is up to 16TB in size. The size and number of the LoTSS data

sets pose a significant infrastructure and processing challenges, the solutions to which are described in our previous work [76].

Besides decreasing the processing time for a single data set, we also built a framework to co-ordinate the processing of multiple LOFAR observations with various pipelines on multiple clusters that we named AGLOW[**AGLOW_mechev**]. Using AGLOW, we deploy the `prefactor`[2] [**prefactor3_gasperin**] Direction Independent (DI) calibration pipeline on clusters in Amsterdam and Germany, located at the LOFAR Archive sites. `Prefactor` aims to remove direction independent effects from the broadband data, a necessary step in obtaining a scientific image. It does so by removing radio interference, flagging bad data and antennas, removing contamination from bright off-axis sources and calibrating the data against a model of the radio sky [135, 141]. These steps remove image artifacts caused by direction independent effects and the resulting data is passed to a Direction Dependent (DD) Calibration pipeline.

We use a fixed version of `prefactor` [**prefactor_LOTSS_DR1**] to process the bulk of the LoTSS data and create DI calibrated images. This version runs on the AGLOW system, on the `GINA` high throughput cluster[3] located at SURFsara, the Dutch national High Performance Computing Centre[4]. Since this release of `prefactor`, in early 2017, the repository has undergone significant development and it is important to verify the quality of the data as software development is ongoing.

The LOFAR Surveys project plans to deploy its software in the form of pre-built images using Singularity[**Singularity**]. Singularity allows to create software images that can be used by an unprivileged user to execute the contained software. Singularity is being tested at multiple university clusters, and is already installed and tested at the `GINA` cluster. To safely deploy software images to multiple scientific teams, it is important to verify whether these images are compatible with scientific pipelines.

7.3 Related Work

Continuous Integration (CI) [**ci1**] and Continuous Delivery (CD)[**cd1**] are concepts that have used by software developers over the past decade and a half. Continuous

[2] `https://github.com/lofar-astron/prefactor/`
[3] Specifications of the GINA cluster can be found at http://docs.surfsaralabs.nl/.
[4] https://userinfo.surfsara.nl/

Integration allows for large team to collaborate on software without worrying about introducing bugs in their development process. This method relies on unit tests and integration tests being shipped with the software, and includes a system that runs such tests every time code is updated. Continuous Delivery builds on this process to automatically validate output products in a production environment, and release a working software package.

Large scientific teams have began introducing CI workflows in their development cycle with many previous successes [**MOOSE_CI**, **genetic_CI**, **CI_proteomics**, **Subaru_CI**] in various fields from genetics to telescope control. These works show the power of CI systems to enable rapid software iteration in an academic environment.

Commercial integration services such as TravisCI[**travisCI**], CircleCI[**circleCI**] and GitLabCI[**gitlabCI**] are widely used in industry and combined with GitHub or GitLab for source control. Additionally, automation suites such as Jenkins[**jenkins**] are typically used to automate CI pipelines on dedicated infrastructure. While Jenkins is a fully featured automation suite, it needs to be installed and managed by a user with elevated privileges.

Apache Airflow[5] is an Apache workflow orchestration software built in Python. It allows building and scheduling generic workflows, and in previous work we implemented the full LOFAR `prefactor` pipeline running on a distributed infrastructure using Airflow (which we named AGLOW) [**AGLOW_mechev**]. We deploy this software in user-space allowing us to easily build and manage LOFAR workflows without requiring elevated privileges. AGLOW has been proven useful in running LOFAR processing on a shared cluster.

Singularity is a container run time which allows deploying containers and images without having elevated privileges. Once the Singularity software is installed by the admin, users can bring their own images which hold various scientific software packages. We have chosen to use Singularity for software containerization and distribution since its portability is a good fit for deploying LOFAR software to multiple scientific groups outside our dedicated Surveys processing.

7.4 Continuous Integration with AGLOW

We use our previous success automating complex LOFAR pipelines with AGLOW to build a Continuous Integration pipeline and deploy it on the shared infrastruc-

[5] http://airflow.apache.org/

ture available at SURFsara. We will deploy the CI pipeline on the same cluster as we use for our production runs in order to ensure compatibility of deployed software with our infrastructure. The source code of our CI workflows are located in the AGLOW software repository located at `https://github.com/apmechev/AGLOW/tree/master/AGLOW/airflow/dags`.

7.4.1 Distributed CI Workflow

Figure 7.1 shows a run of the CI workflow in progress. The workflow checks the latest commit of both the repository and software image and compares it with the date of the last successful run of both. If testing is required, the workflow sends a request to bring the test calibrator and target data to disk. Once the calibrator data is available, the calibrator part of the `prefactor` pipeline is tested. Figure 7.2 shows a run of the 'calibrator' step of `prefactor`. The Airflow tasks shown here can easily be modified based on the scientific pipeline. When the processing completes, the sub-DAG in 7.2 checks if the required files have been created successfully.

Once the calibration is completed, the same steps are repeated with the `prefactor` scripts for the science target. Upon successful completion, the move_results SubDag uploads the software image to our distribution repository. We currently distribute software on a WebDAV[**webdav**] server hosted by SURFsara.

7.4.2 Nightly Builds

Because running the `prefactor` pipeline on our test data set (described in the Appendix) takes several hours and several hundred CPU-hours, we only test the software on a nightly basis. We run this nightly build process at midnight every day. Our software checks if there have been new commits in the pipeline repositories or whether a new software image version has been released. If either the software or pipeline repository have been updated, we run the full CI suite on our test data set. If the tests succeed, we upload the new software images, scientific images and an archive of the pipeline scripts to a deployment area identified by the date of the run.

7.4.3 Container Testing

We build the LOFAR software on Singularity-Hub by integrating a GitHub repository containing the build scripts[6] with the Singularity-Hub builders, and storing the

[6] located at https://github.com/tikk3r/lofar-grid-hpccloud

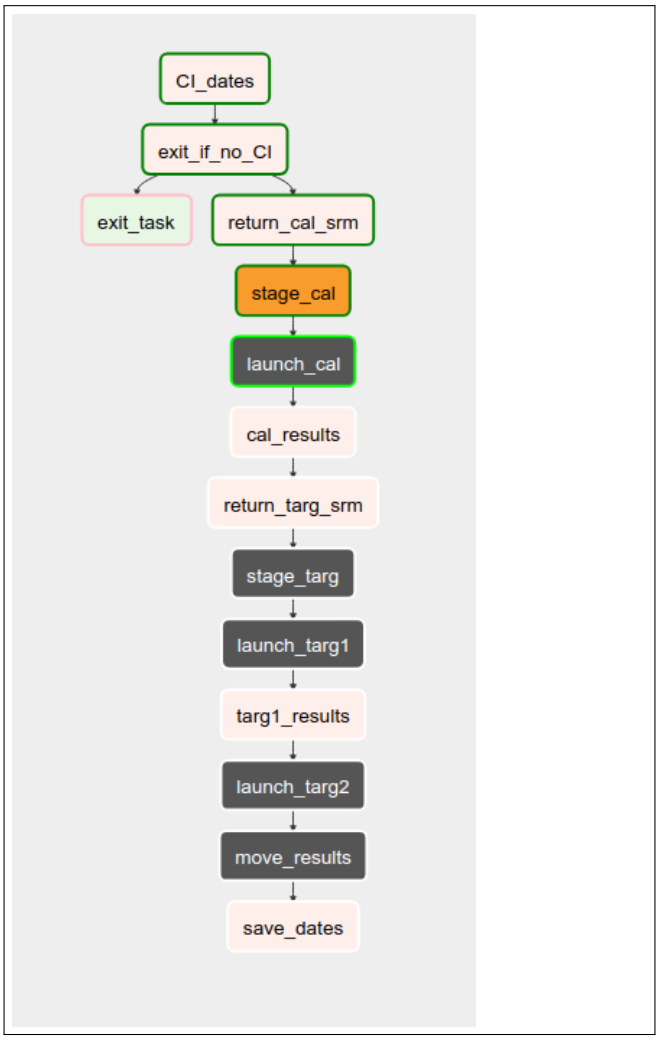


Figure 7.1: The workflow responsible for testing the `prefactor` pipeline as well as the current LOFAR software image. The dark gray tasks are a SubDags, consisting of a short workflow to build launch and monitor the jobs for each `prefactor` step.

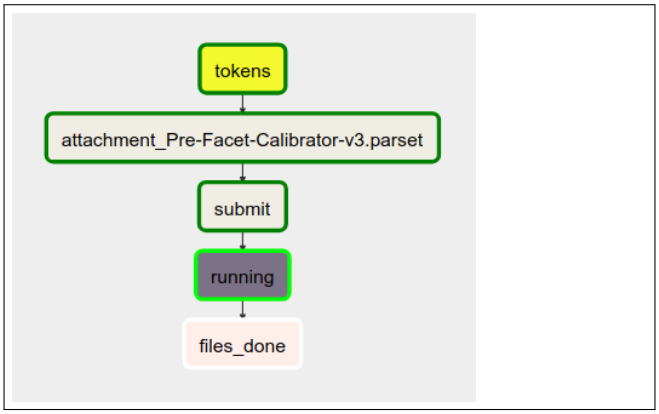


Figure 7.2: Each task that requires processing on the SURFsara infrastructure is represented as a SubDag workflow shown here. This SubDag, launch_cal, has tasks to create the pilot job tokens, attach the required parameter set (parset) file to each job token, submit the jobs to the `GINA` cluster, monitor those jobs and verify that the result files have been created.

completed image directly in Singularity-Hub[7]. Singularity-Hub allows multiple versions of an image to be hosted, and differentiates releases by their MD5 hashes. We use the 'frozen' version of a container as a stable release and the most recent version as a test release.

In addition to the `prefactor` pipeline, we also run the unit tests of the numpy[**numpy:2011**], scipy[51] and astropy[**astropy:2018**] scientific libraries. We import and run the tests module for each of these libraries. While we continue despite failed tests, we still log the test results to help debugging library errors.

7.4.4 Repository testing

To test scientific scripts, we use the latest commit of the GitHub repository in combination with a parameter set file, named the parset. This parset describes the processing steps as well as the processing parameters. We use the same parameters as in the LoTSS processing, to ensure that the image quality obtained during our CI tests will match the quality of our production run. Once the data is processed, we store the intermediate products with a time-stamp to allow future comparison between data processed with different versions of the software and scripts.

[7]Images are hosted at http://singularity-hub.org/collections/1999

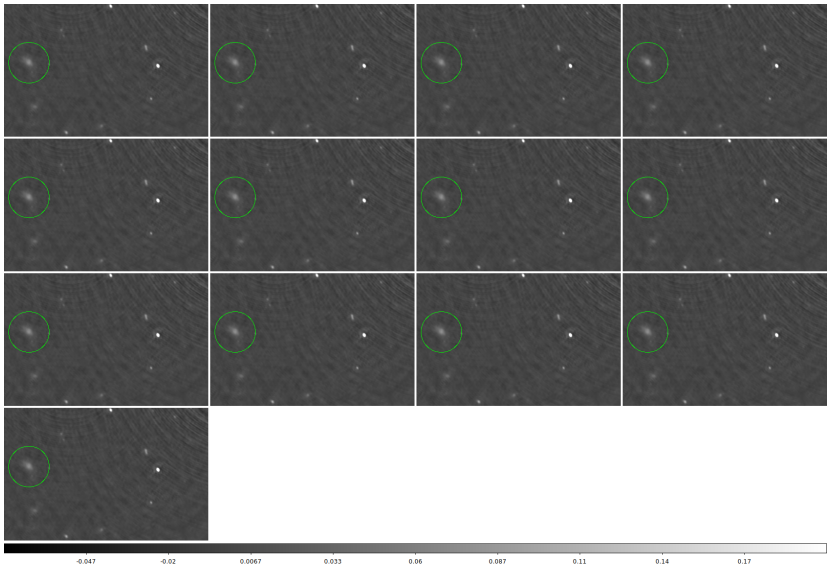


Figure 7.3: Images created by the CI runs from 2019-03-29 to 2019-04-23. We sample the diffuse region selected to get statistics in Figure 7.7. The images contains some calibration artifacts that will be removed by the Direction Dependent Calibration that follows the `prefactor` pipeline. This figure shows that there were no significant changes in image quality during the 1 month period in the study.

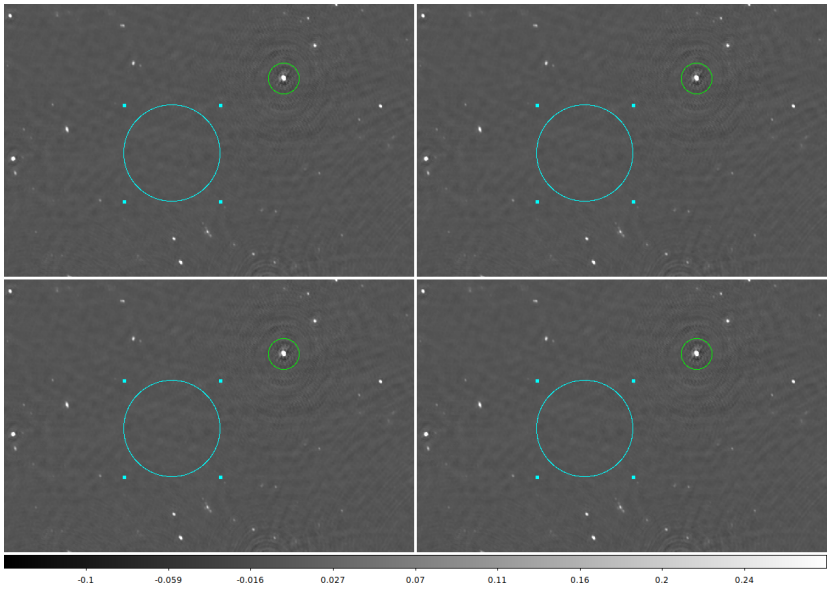


Figure 7.4: Images created by the CI runs from 2019-03-29 (left column) and 2019-04-23 (right column). The green region shows a bright source of ∼6 Jansky, which we use to validate the extracted flux for point sources. The cyan region is an empty part of the sky that we use to obtain noise measurements of the image. We show pixel statistics

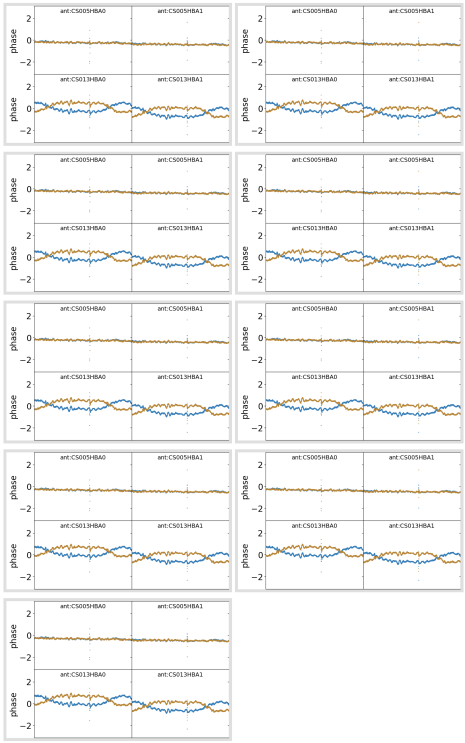


Figure 7.5: Phase calibration solutions for four core stations for the duration of the test observation as produced by our CI runs from 2019-03-29 to 2019-04-23. The plots show the phase offset between the core station and the reference station for both polarizations throughout the duration of the observation. The calibration solutions shown here have not changed significantly during our CI tests.

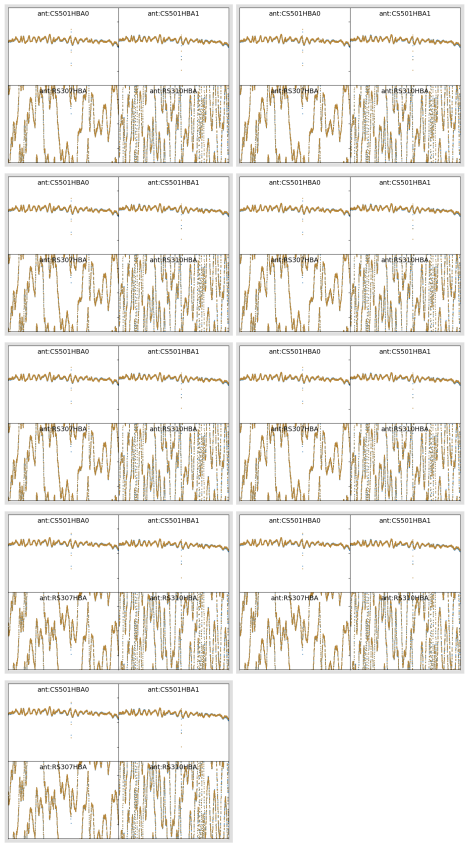


Figure 7.6: Phase calibration solutions for four remote stations for the duration of the test observation as produced by our CI runs from 2019-03-29 to 2019-04-23. Remote stations typically have phase wrapping, which can be seen in the bottom two plots in each inset. The largest difference between the solutions from the last three days (19, 21 and 23 of April) and the previous solutions is a global phase offset in the data.

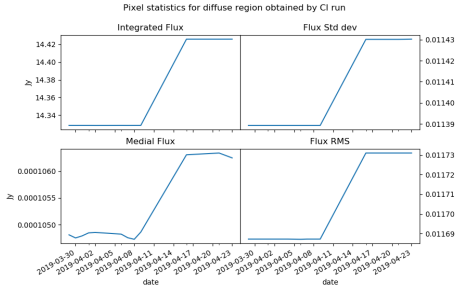


Figure 7.7: Pixel statistics for the region circled in Figure 7.3 obtained from calibrated images produced by the AGLOW CI runs from 2019-03-29 to 2019-04-23. The sharp difference from April 11 onwards is due to a change in time and frequency averaging parameters.

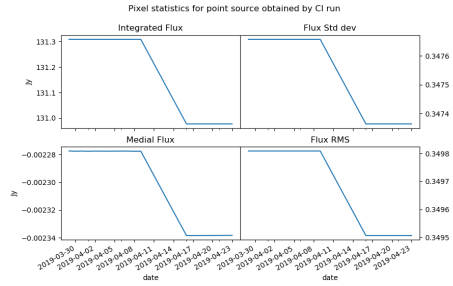


Figure 7.8: Pixel statistics for the green circled in Figure 7.4 obtained from calibrated images produced by the AGLOW CI runs from 2019-03-29 to 2019-04-23. This data is used to determine the ability of `prefactor` to retrive the flux of point sources.

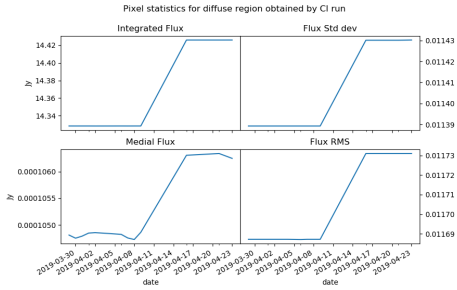


Figure 7.9: Pixel statistics for the cyan region circled in Figure 7.4 obtained from calibrated images produced by the AGLOW CI runs from 2019-03-29 to 2019-04-23. We use the statistics from this region to determine the sensitivity of the image, i.e. determining the faintest detectable sources.

7.5 Results and Discussions

7.5.1 Data Quality

Astronomers use multiple methods when comparing data quality of final images. While the results from the `prefactor` pipeline need further processing to obtain a final scientific image, the intermediate results produced by the DI pipeline can still be compared across different software versions. This comparison will give early bounds on the final image quality and is an important part to ensuring we reach a sufficient image quality.

The primary way to compare data processed by `prefactor` is by eye, however quantitative metrics such as the background noise and integrated flux around sources are also used to compare results. In addition to images, the resulting data includes calibration solutions. The statistics, such as the the mean phase, phase variance and quartile range of these solutions are useful to scientists to determine calibration accuracy.

We show the images obtained from our CI data in Figures 7.3 and 7.4. The former figure shows 13 CI runs highlighting a region of diffuse emission, while the latter shows a bright point source and an empty region for calculating noise statistics. Using statistics from all three regions, we can track the ability for the pipeline to image diffuse sources, compact sources and faint sources. All three types of sources are important for different radio astronomy science cases.

Additionally, Figures 7.5 and 7.6 show the inspection plots produced by the sky model calibration from our CI runs between March and April 2019. These plots are automatically produced by `prefactor` and show the accuracy of the calibration solutions obtained by the pipeline. Typically the calibration quality is assessed by eye by an experienced astronomer using these plots.

Finally, we present the statistics for three regions of interest. Statistics about the diffuse source show how accurately the software extracts fainter, diffuse sources. The pixel statistics for this source are shown in 7.7. We have also chosen a bright point source (the quasar 4C 49.22) in our field to show the ability for `prefactor` to retrieve bright point source fluxes.

7.5.2 Discussion

During the first month of `prefactor` Continuous Integration tests, we produced 13 data sets each processed with a different version of the `prefactor` pipeline. We automatically create images of these data sets and compare different statistical metrics for each image. From those results, we can detect the effects changing processing parameters such as the time and frequency resolution of the data.

This data is crucial for determining whether changes in software or processing scripts and parameters will affect image quality, noise levels and source extraction rates. Running our tests nightly makes it possible to create a time series of data quality metrics and notify astronomers when these metrics change or degrade significantly.

Additionally, the AGLOW software allows for multiple scientific pipelines to be tested concurrently. Each pipeline will produce its own set of images and statistics, which are used by astronomers to verify their scripts. As all LOFAR scientific pipelines are hosted on GitHub, they are easy to integrate with the CI suite.

7.6 Conclusions

We have implemented a Continuous Integration suite that is able to test LOFAR pipelines and software, create radio images, upload and version software and report image quality and statistics automatically. We can easily detect changes in image quality caused by software, algorithm or parameter changes in the `prefactor` scientific pipeline, and act quickly on these changes.

Additionally, the flexibility of AGLOW allows to test commits of the pipelines in the past, comparing the data products from historical versions of the software to the current version. Moreover, we can see that most changes in the pipeline do not have a noticeable effect on the data quality, meaning that the scientific quality of the final images is expected to be stable across most commits. Having a time series of image statistics makes it easier to detect, understand and fix code that leads to significant changes in image quality.

Our successes testing the `prefactor` pipeline was the first case of Continuous Integration used to verify scientific data quality for the LOFAR surveys. The extensibility of the AGLOW system makes it easy to add further pipelines or quality checks to ensure the high development pace does not result in degradation of the scientific products.

APPENDIX

We test our processing pipelines on a standard LOFAR surveys observation designated L229587 in the direction of the HetDex field[**hetdex**] in the direction (11h55m41.282, +049d44m52.908). The data is stored at the SURFsara LTA site at the following URI:

```
srm://srm.grid.sara.nl:8443/pnfs/grid.sara.nl/data/lofar
    /ops/projects/lc2_038/229587/
```

To minimize processing time and resources, we only use a fraction of the entire frequency bandwidth corresponding to Subbands 100-110. This corresponds to a frequency between 139.844 and 141.602 MHz. The data was observed on 28-May-2014 between 15:00 and 23:00. The initial data was 150GB and the final averaged data set was 2.3 GB.

We make images with the following software and parameters:

```
wsclean -size 2560 1080 -maxuv-l 7000 -baseline-
    averaging 5.34930608721 -local-rms-method rms-with-
    min -mgain 0.8 -auto-mask 3.3 -pol I -weighting-
    rank-filter 3 -auto-threshold 0.5 -j 5 -local-rms-
    window 50 -mem 20 -weight briggs 0.0 -scale 0.00208
    -niter 5000 -no-update-model-required
```

Acknowledgement

APM would like to acknowledge the support from the NWO/DOME/IBM programme “Big Bang Big Data: Innovating ICT as a Driver For Astronomy”, project #628.002.001.

The processing and storage functionality that has made this project possible was enabled by SURF Cooperative through grant e-infra 160022 & 160152. The LOFAR software and dedicated reduction packages on `https://github.com/apmechev/GRID_LRT` were deployed on the e-infrastructure by the LOFAR e-infra group, consisting of J.B.R. Oonk (SURFsara & Leiden Observatory), A.P. Mechev (Leiden Observatory).

This paper is based on data obtained with the International LOFAR Telescope (ILT) under project codes LC2_038 and LC3_008. LOFAR (van Haarlem

et al. 2013) is the Low-Frequency Array designed and constructed by ASTRON. It has observing, data processing, and data storage facilities in several countries, which are owned by various parties (each with their own funding sources), and which are collectively operated by the ILT foundation under a joint scientific policy. The ILT resources have benefited from the following recent major funding sources: CNRS-INSU, Observatoire de Paris and Université d'Orléans, France; BMBF, MIWF-NRW, MPG, Germany; Science Foundation Ireland (SFI), Department of Business, Enterprise and Innovation (DBEI), Ireland; NWO, The Netherlands; The Science and Technology Facilities Council (STFC), UK

8 Conclusion

As astronomical observatories collect ever growing data-sets, the processing challenges for these data will continue increasing. Large scale surveys expected to produce petabytes of data can no longer be processed on single machine or small dedicated clusters at scientific institutions. Large scale distributed processing is needed to serve the scientific requirements of these survey projects.

CERN's World-Wide computing grid provides sufficent resources for such projects, however due to its focus on distributed Monte-Carlo simulations, it also presents some design constraints. Namely, porting complex workflows to a grid-like environment requires a framework to distribute and monitor jobs. Additionally, a workflow orchestration software is needed to schedule and automate processing.

8.1 Summary of Thesis Achievements

This work focuses on the software built to accelerate, parallelize, and automate LOFAR processing, as well as the insights obtained into large scale processing of LOFAR data. To date, we have helped process an unprecedented 8 petabytes of data for the LOFAR Two-Meter Sky Survey (LoTSS), data which has led to more than 30 publications. We describe a generic platform for scaling astronomical processing across multiple clusters, focused on the application of bulk LOFAR processing.

We have built software that can encapsulate LOFAR processing steps and distribute them acros a heterogenous infrastructure. Our tools have been used by several scientists, implementing multiple complex pipelines, processing a total of 8 petabytes of data.

We implemented a complex monitoring suite along our processing to track the performance of individual pipeline steps.

8.2 Answers to Research Questions

Research Question 1: *How can we use a distributed shared infrastructure for efficient LOFAR data processing?*

In Chapters 2 and 3, we detail our success with massively distributed processing of LOFAR data. We describe the underlying platform, inherited from the High Energy Physics community and the modifications to these tools that were required to host complex processing software. We detail these modifications and discuss the increas in throughput that distributed processing leads to. Finally, we make estimations on the processing time saved by parallelizing LOFAR data processing. The work described in these chapters is essential to producing scientific data sets at a high cadence, particularly considering the high data rates produced by LOFAR.

Research Question 2: *How can we build software to easily accelerate complex pipelines for Radio Astronomy?*

Chapters 5 and 7 detail the advances in parallelizing complex scientific pipelines on a distributed shared infrastructures. We integrate a mature workflow orchestration package with distributed LOFAR processing. We discuss the need for this orchestration, as well as the abilities to support additional complex pipelines. As an example application, we build a Continuous Integration pipeline tasked with verifying and validating the initial steps of LOFAR processing.

Research Question 3: *Can we automatically collect performance information during massively distributed processing and predict run times for future data sets?*

Chapters 4 and 6 describe a performance monitoring suite for LOFAR data and our scalability model for LOFAR processing. When running massively distributed processing, scientists are unable to monitor the performance of the underlying software. Collecting these statistics is necessary for understanding processing inefficiencies and suggest ways to accelerate data processing. Performance data can also be used to understand the effect of processing parameters on the resource usage of complex pipelines. We study this in detail, building a model that can be used to understand the scalability of multiple processing steps. This model shows the limitations on scientific parameters imposed by limited processing resources as well as suggestions on decreasing processing time without sacrificing scientific data quality.

8.3 Future Work

This work focuses on the first stages of LOFAR data processing, because of the large gains possible by parallelization. We take in mind the complexity of our processing workflows, the wide range of scientific pipelines and the heterogenous nature of the underlying infrastructure. Because of these factors, the software we've built can be used by a wide range of astronomical pipelines. Moreover, we can incorporate processing hosted at scientific institutions and cloud providers, to scale scientific processing horizontally. One application for large scale distributed processing is the Square kilometer Array.

The Square Kilometer Array, (SKA) is a planned aperture synthesis radio telescope expected to have a total collecting area of one square kilometer.

Acknowledgements

I would like to express (whatever feelings I have) to:

- My supervisor
- My second supervisor
- Other researchers
- My family and friends

Glossary

LOFAR The LOw Frequency ARray: A large, low-frequency aperture synthesis radio telescope.. 8

LoTSS The LOFAR Two-Meter Sky Survey is a whole-sky study of the low-frequency radio sky. 9

Bibliography

[1] C Aguado Sanchez et al. "CVMFS-a file system for the CernVM virtual appliance". In: *Proceedings of XII Advanced Computing and Analysis Techniques in Physics Research*. Vol. 1. 2008, p. 52.

[2] Dong H Ahn. *Measuring FLOPS using hardware performance counter technologies on LC systems*. Tech. rep. Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2008.

[3] Ilkay Altintas et al. "Kepler: an extensible system for design and execution of scientific workflows". In: *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*. IEEE. 2004, pp. 423–424.

[4] Peter Amstutz et al. "Common Workflow Language, v1. 0". In: (2016).

[5] Chirs ANDERSON. "Apache couchdb: The definitive guide". In: *http://couchdb. Apache. org/index. htm Acessado em* 5.06 (2009), p. 2009.

[6] Team Apache HBase. "Apache hbase reference guide". In: *Apache, version* 2.0 (2015).

[7] Software Foundation Apache. *TCollector: OpenTSDB documentation*. Available at `http://opentsdb.net/docs/build/html/user_guide/utilities/tcollector.html`. 2017.

[8] M. Arias et al. "Low-frequency radio absorption in Cassiopeia A". In: *A&A* 612, A110 (Apr. 2018), A110. DOI: `10.1051/0004-6361/201732411`. arXiv: `1801.04887 [astro-ph.HE]`.

[9] M Arias et al. "Low-frequency radio absorption in Cassiopeia A". In: *Astronomy & Astrophysics* 612 (2018), A110.

[10] ASTRON. *Grid at SURFsara*. `https://old.astron.nl/radio-observatory/lofar-system-capabilities/major-observing-modes/interferometric-mode/processing--0`. 2019.

[11] Domingos Barbosa et al. “Power Monitoring and Control for Large Scale projects: SKA, a case study”. In: *SPIE Astronomical Telescopes+ Instrumentation*. International Society for Optics and Photonics. 2016, pp. 99100L–99100L.

[12] Bradley J Barnes et al. “A regression-based approach to scalability prediction”. In: *Proceedings of the 22nd annual international conference on Supercomputing*. ACM. 2008, pp. 368–377.

[13] Jakob Blomer et al. “Distributing LHC application software and conditions databases using the CernVM file system”. In: *Journal of Physics: Conference Series*. Vol. 331. 4. IOP Publishing. 2011, p. 042003.

[14] Joris Borgdorff, Harsha Krishna, and Michael H Lees. “SIM-CITY: An e-Science framework for Urban Assisted Decision Support”. In: *Procedia Computer Science* 51 (2015), pp. 2327–2336.

[15] Jan Bot. *PiCaS: Python client using CouchDB as a token pool server*. `https://github.com/jjbot/picasclient`. 2017.

[16] Terrehon Bowden. *THE /proc FILESYSTEM v1.3*. `https://www.kernel.org/doc/Documentation/filesystems/proc.txt`. 2009.

[17] P Chris Broekema, Rob V van Nieuwpoort, and Henri E Bal. “The Square Kilometre Array science data processor. Preliminary compute platform design”. In: *Journal of Instrumentation* 10.07 (2015), p. C07004.

[18] P. Chris Broekema et al. “Cobalt: A GPU-based correlator and beamformer for LOFAR”. In: *Astronomy and Computing* 23 (2018), pp. 180–192. ISSN: 2213-1337. DOI: `https://doi.org/10.1016/j.ascom.2018.04.006`. URL: `http://www.sciencedirect.com/science/article/pii/S2213133717301439`.

[19] WN Brouw. “Aperture synthesis”. In: *Image Processing Techniques in Astronomy*. Springer, 1975, pp. 301–307.

[20] Alexandru Calotoiu et al. “Using automated performance modeling to find scalability bugs in complex codes”. In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM. 2013, p. 45.

[21] Laura Carrington, Allan Snavely, and Nicole Wolter. “A performance prediction framework for scientific applications”. In: *Future Generation Computer Systems* 22.3 (2006), pp. 336–346.

[22] Aniello Castiglione et al. "Exploiting mean field analysis to model performances of big data architectures". In: *Future Generation Computer Systems* 37 (2014). Special Section: Innovative Methods and Algorithms for Advanced Data-Intensive Computing Special Section: Semantics, Intelligent processing and services for big data Special Section: Advances in Data-Intensive Modelling and Simulation Special Section: Hybrid Intelligence for Growing Internet and its Applications, pp. 203–211. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2013.07.016. URL: http://www.sciencedirect.com/science/article/pii/S0167739X13001611.

[23] R Centeno et al. "The Helioseismic and Magnetic Imager (HMI) vector magnetic field pipeline: optimization of the spectral line inversion code". In: *Solar Physics* 289.9 (2014), pp. 3531–3547.

[24] David Churches et al. "Programming scientific and distributed workflow with Triana services". In: *Concurrency and Computation: Practice and Experience* 18.10 (2006), pp. 1021–1037.

[25] BG Clark. "An efficient implementation of the algorithm'CLEAN'". In: *Astronomy and Astrophysics* 89 (1980), p. 377.

[26] Marco Clemencic and B Couturier. "A New Nightly Build System for LHCb". In: *Journal of Physics: Conference Series*. Vol. 513. 5. IOP Publishing. 2014, p. 052007.

[27] A Patterson David and L Hennessy John. "Computer organization and design: the hardware/software interface". In: *San mateo, CA: M organ Kaufmann Publishers* 1 (2005), p. 998.

[28] David B Davidson. "Potential technological spin-offs from MeerKAT and the South African Square Kilometre Array bid". In: *South African Journal of Science* 108.1-2 (2012), pp. 01–03.

[29] GS Davies et al. "Software Management for the NOνAExperiment". In: *Journal of Physics: Conference Series*. Vol. 664. 6. IOP Publishing. 2015, p. 062011.

[30] de Gasperin, F. et al. "Systematic effects in LOFAR data: A unified calibration strategy". In: *A&A* 622 (2019), A5. DOI: 10.1051/0004-6361/201833867. URL: https://doi.org/10.1051/0004-6361/201833867.

[31] Ewa Deelman et al. "Pegasus: A framework for mapping complex scientific workflows onto distributed systems". In: *Scientific Programming* 13.3 (2005), pp. 219–237.

[32] Peter J Denning. "The working set model for program behavior". In: *Communications of the ACM* 11.5 (1968), pp. 323–333.

[33] Peter E Dewdney et al. "The square kilometre array". In: *Proceedings of the IEEE* 97.8 (2009), pp. 1482–1496.

[34] Paolo Di Tommaso et al. “Nextflow enables reproducible computational workflows”. In: *Nature biotechnology* 35.4 (2017), pp. 316–319.

[35] Jack Dongarra and Francis Sullivan. “Guest Editors’ Introduction: The Top 10 Algorithms”. In: *Computing in Science & Engineering* 2.1 (2000), pp. 22–23. DOI: 10.1109/MCISE.2000.814652. eprint: https://aip.scitation.org/doi/pdf/10.1109/MCISE.2000.814652. URL: https://aip.scitation.org/doi/abs/10.1109/MCISE.2000.814652.

[36] K. L. Emig et al. “The first detection of radio recombination lines at cosmological distances”. In: *A&A* 622, A7 (Feb. 2019), A7. DOI: 10.1051/0004-6361/201834052. arXiv: 1811.08104 [astro-ph.GA].

[37] W. Freudling et al. “Automated data reduction workflows for astronomy. The ESO Reflex environment”. In: *Astronomy & Astrophysics* 559, A96 (Nov. 2013), A96. DOI: 10.1051/0004-6361/201322494. arXiv: 1311.5411 [astro-ph.IM].

[38] Patrick Fuhrmann and Volker Gülzow. “dCache, storage system for the future”. In: *European Conference on Parallel Processing*. Springer. 2006, pp. 1106–1113.

[39] FZJülich. *Grid at Forschungszentrum Jülich*. https://www.fz-juelich.de/ias/jsc/EN/Expertise/Services/Certificates/Grid/grid_node.html. 2019.

[40] *Gina specifications - GRID Documentation v1.0*. Available at http://docs.surfsaralabs.nl/projects/grid/en/latest/Pages/Service/system_specifications/gina_specs.html. 2017.

[41] Kazushige Goto and Robert A Geijn. “Anatomy of high-performance matrix multiplication”. In: *ACM Transactions on Mathematical Software (TOMS)* 34.3 (2008), p. 12.

[42] Brendan Gregg and Jim Mauro. *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD*. Prentice Hall Professional, 2011.

[43] Y Gupta et al. “The upgraded GMRT: opening new windows on the radio Universe”. In: *Current Science* 113.4 (2017), p. 707.

[44] Kim Hazelwood and James E Smith. “Exploring code cache eviction granularities in dynamic optimization systems”. In: *Code Generation and Optimization, 2004. CGO 2004. International Symposium on*. IEEE. 2004, pp. 89–99.

[45] Hanno Holties, Adriaan Renting, and Yan Grange. “The LOFAR long-term archive: e-infrastructure on petabyte scale”. In: *SPIE Astronomical Telescopes+ Instrumentation*. International Society for Optics and Photonics. 2012, pp. 845117–845117.

[46] Andreas Horneffer et al. *apmechev/prefactor: LOTSS Data Release 1*. Nov. 2018. DOI: 10.5281/zenodo.1487962. URL: https://doi.org/10.5281/zenodo.1487962.

[47] Shiliang Hu et al. "An approach for implementing efficient superscalar CISC processors". In: *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*. IEEE. 2006, pp. 41–52.

[48] HT Intema et al. "The GMRT 150 MHz all-sky radio survey-First alternative data release TGSS ADR1". In: *Astronomy & Astrophysics* 598 (2017), A78.

[49] Tarush Jain and Tanmay Agrawal. "The haswell microarchitecture-4th generation processor". In: *International Journal of Computer Science and Information Technologies* 4.3 (2013), pp. 477–480.

[50] Justin L Jonas. "MeerKAT-The South African array with composite dishes and wideband single pixel feeds". In: *Proceedings of the IEEE* 97.8 (2009), pp. 1522–1530.

[51] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online; accessed July 12, 2019]. 2001–. URL: `http://www.scipy.org/`.

[52] A. B. Kahn. "Topological Sorting of Large Networks". In: *Commun. ACM* 5.11 (Nov. 1962), pp. 558–562. ISSN: 0001-0782. DOI: `10.1145/368996.369025`. URL: `http://doi.acm.org/10.1145/368996.369025`.

[53] Randy H. Katz and David A. Patterson. *Memory Hierarchy, CMPUT429/CMPE382 Winter 2001. University of Calgary*. Available at `https://webdocs.cs.ualberta.ca/~amaral/courses/429/webslides/Topic4-MemoryHierarchy/sld003.htm`. 2001.

[54] S. Kavulya et al. "An Analysis of Traces from a Production MapReduce Cluster". In: *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. May 2010, pp. 94–103. DOI: `10.1109/CCGRID.2010.112`.

[55] S Kazemi et al. "Radio interferometric calibration using the SAGE algorithm". In: *Monthly Notices of the Royal Astronomical Society* 414.2 (2011), pp. 1656–1666.

[56] Michael Kotliar, Andrey Kartashov, and Artem Barski. "CWL-Airflow: a lightweight pipeline manager supporting Common Workflow Language". In: *bioRxiv* (2018), p. 249243.

[57] Michael Kuperberg, Klaus Krogmann, and Ralf Reussner. "Performance prediction for black-box components using reengineered parametric behaviour models". In: *International Symposium on Component-Based Software Engineering*. Springer. 2008, pp. 48–63.

[58] Erwin Laure et al. *Middleware for the next generation Grid infrastructure*. Tech. rep. CERN, 2004.

[59] Alvin R Lebeck et al. “A large, fast instruction window for tolerating cache misses”. In: *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*. IEEE. 2002, pp. 59–70.

[60] Ronny Levanda and Amir Leshem. “Synthetic aperture radio telescopes”. In: *Signal Processing Magazine, IEEE* 27 (Feb. 2010), pp. 14–29. DOI: 10.1109/MSP.2009.934719.

[61] Y Li. “Reliability of long heterogeneous slopes in 3d: Model performance and conditional simulation”. In: (2017).

[62] Ji Liu et al. “A survey of data-intensive scientific workflow management”. In: *Journal of Grid Computing* 13.4 (2015), pp. 457–493.

[63] *LOFAR Imaging Cookbook*. Available at http://www.astron.nl/sites/astron.nl/files/cms/lofar_imaging_cookbook_v19.pdf.

[64] *Lofar Software Stack*. http://www.lofar.org/wiki/doku.php?id=public:software_stack_installation. 2017.

[65] Colin J Lonsdale et al. “The murchison widefield array: Design overview”. In: *Proceedings of the IEEE* 97.8 (2009), pp. 1497–1506.

[66] GM Loose. “LOFAR self-calibration using a blackboard software architecture”. In: *Astronomical Data Analysis Software and Systems XVII*. Vol. 394. 2008, p. 91.

[67] Jason Maassen et al. *Xenon: Xenon 1.1.0*. Dec. 2015. DOI: 10.5281/zenodo.35415. URL: https://doi.org/10.5281/zenodo.35415.

[68] Cecchi Marco et al. “The glite workload management system”. In: *Journal of Physics: Conference Series*. Vol. 219. IOP Publishing. 2010, p. 062039.

[69] Matthew L Massie, Brent N Chun, and David E Culler. “The ganglia distributed monitoring system: design, implementation, and experience”. In: *Parallel Computing* 30.7 (2004), pp. 817–840.

[70] JP McMullin et al. “CASA architecture and applications”. In: *Astronomical data analysis software and systems XVI*. Vol. 376. 2007, p. 127.

[71] A. P. Mechev et al. “Fast and Reproducible LOFAR Workflows with AGLOW”. In: *arXiv e-prints*, arXiv:1808.10735 (Aug. 2018), arXiv:1808.10735. arXiv: 1808.10735 [astro-ph.IM].

[72] A.P. Mechev et al. “Pipeline Collector: Gathering performance data for distributed astronomical pipelines”. In: *Astronomy and Computing* 24 (2018), pp. 117–128. ISSN: 2213-1337. DOI: https://doi.org/10.1016/j.ascom.2018.06.005. URL: http://www.sciencedirect.com/science/article/pii/S2213133718300490.

[73] A.P. Mechev et al. “Pipeline Collector: Gathering performance data for distributed astronomical pipelines”. In: *Astronomy and Computing* 24 (2018), pp. 117–128. ISSN: 2213-1337. DOI: https://doi.org/10.1016/j.ascom.2018.06.005. URL: http://www.sciencedirect.com/science/article/%20pii/S2213133718300490.

[74] A.P. Mechev et al. “Scalability model for the LOFAR direction independent pipeline”. In: *Astronomy and Computing* 28 (2019), p. 100293. ISSN: 2213-1337. DOI: https://doi.org/10.1016/j.ascom.2019.100293. URL: http://www.sciencedirect.com/science/article/pii/S2213133719300290.

[75] A. Mechev et al. “An Automated Scalable Framework for Distributing Radio Astronomy Processing Across Clusters and Clouds”. In: *Proceedings of the International Symposium on Grids and Clouds (ISGC) 2017, held 5-10 March, 2017 at Academia Sinica, Taipei, Taiwan (ISGC2017). Online at https://pos.sissa.it/cgi-bin/reader/conf.cgi?confid=293, id.2*. Mar. 2017, p. 2. arXiv: 1712.00312 [astro-ph.IM].

[76] A. Mechev et al. “An Automated Scalable Framework for Distributing Radio Astronomy Processing Across Clusters and Clouds”. In: *Proceedings of the International Symposium on Grids and Clouds (ISGC) 2017, held 5-10 March, 2017 at Academia Sinica, Taipei, Taiwan (ISGC2017). Online at https://pos.sissa.it/cgi-bin/reader/conf.cgi?confid=293, id.2*. Mar. 2017, p. 2. arXiv: 1712.00312 [astro-ph.IM].

[77] A. Mechev et al. “Fast and Reproducible LOFAR Workflows with AGLOW”. In: *2018 IEEE 14th International Conference on e-Science (e-Science)*. Oct. 2018, pp. 136–144. DOI: 10.1109/eScience.2018.00029.

[78] AP Mechev et al. “An Automated Scalable Framework for Distributing Radio Astronomy Processing Across Clusters and Clouds”. In: *arXiv preprint arXiv:1712.00312* (2017).

[79] William K Michener and Matthew B Jones. “Ecoinformatics: supporting ecology as a data-intensive science”. In: *Trends in ecology & evolution* 27.2 (2012), pp. 85–93.

[80] Niruj Mohan and David Rafferty. “PyBDSF: Python Blob Detection and Source Finder”. In: *Astrophysics Source Code Library* (2015).

[81] Philip J Mucci et al. “PAPI: A portable interface to hardware performance counters”. In: *Proceedings of the department of defense HPCMP users group conference*. Vol. 710. 1999.

[82] Nicholas Nethercote and Julian Seward. “Valgrind: a framework for heavyweight dynamic binary instrumentation”. In: *ACM Sigplan notices*. Vol. 42. 6. ACM. 2007, pp. 89–100.

[83] A. R. Offringa, J. J. van de Gronde, and J. B. T. M. Roerdink. "A morphological algorithm for improving radio-frequency interference detection". In: *Astronomy & astrophysics* 539, A95 (Mar. 2012), A95. DOI: 10.1051/0004-6361/201118497. arXiv: 1201.3364 [astro-ph.IM].

[84] A. R. Offringa et al. "WSCLEAN: an implementation of a fast, generic wide-field imager for radio astronomy". In: *Monthly Notices of the Royal Astronomical Society* 444 (Oct. 2014), pp. 606–619. DOI: 10.1093/mnras/stu1368. arXiv: 1407.1943 [astro-ph.IM].

[85] AR Offringa et al. "The LOFAR radio environment". In: *Astronomy & astrophysics* 549 (2013), A11.

[86] J. B. R. Oonk et al. "Carbon and hydrogen radio recombination lines from the cold clouds towards Cassiopeia A". In: *MNRAS* 465.1 (Feb. 2017), pp. 1066–1088. DOI: 10.1093/mnras/stw2818. arXiv: 1609.06857 [astro-ph.GA].

[87] J. B. R. Oonk et al. "Discovery of carbon radio recombination lines in absorption towards Cygnus A". In: *MNRAS* 437 (Feb. 2014), pp. 3506–3515. DOI: 10.1093/mnras/stt2158. arXiv: 1401.2876.

[88] J.B.R. Oonk et al. "Radio astronomy on a distributed shared computing platform: The LOFAR case". In: (2018).

[89] S. Ott. "The Herschel Data Processing System — HIPE and Pipelines — Up and Running Since the Start of the Mission". In: *Astronomical Data Analysis Software and Systems XIX*. Vol. 434. Dec. 2010, p. 139.

[90] David J Pearce and Paul HJ Kelly. "A dynamic topological sort algorithm for directed acyclic graphs". In: *Journal of Experimental Algorithmics (JEA)* 11 (2007), pp. 1–7.

[91] *PiCaS Overview - Grid Documentation v1.0*. http://doc.grid.surfsara.nl/en/latest/Pages/Practices/picas/picas_overview.html. 2017.

[92] RF Pizzo et al. "The Lofar Imaging Cookbook v2.0". In: *internal ASTRON report* (2010).

[93] PSNC. *PSNC Online*. http://www.man.poznan.pl/online/en/. 2019.

[94] Anjani Ragothaman et al. "Developing ethread pipeline using saga-pilot abstraction for large-scale structural bioinformatics". In: *BioMed research international* 2014 (2014).

[95] H. J. A. Rottgering et al. "LOFAR - Opening up a new window on the Universe". In: *arXiv e-prints*, astro-ph/0610596 (Oct. 2006), astro–ph/0610596. arXiv: astro-ph/0610596 [astro-ph].

[96] Grigory Rybkin. "ATLAS software packaging". In: *Journal of Physics: Conference Series*. Vol. 396. 5. IOP Publishing. 2012, p. 052063.

[97] J Sabater et al. "Calibration of radio-astronomical data on the cloud. LOFAR, the pathway to SKA." In: *Highlights of Spanish Astrophysics VIII*. 2015, pp. 840–843.

[98] P. Salas et al. "LOFAR observations of decameter carbon radio recombination lines towards Cassiopeia A". In: *Monthly Notices of the Royal Astronomical Society* 467 (May 2017), pp. 2274–2287. DOI: 10.1093/mnras/stx239. arXiv: 1701.08802.

[99] P. Salas et al. "Mapping low-frequency carbon radio recombination lines towards Cassiopeia A at 340, 148, 54, and 43 MHz". In: *MNRAS* 475.2 (Apr. 2018), pp. 2496–2511. DOI: 10.1093/mnras/stx3340. arXiv: 1801.05298 [astro-ph.GA].

[100] Stefano Salvini and Stefan J Wijnholds. "StEFCal-An Alternating Direction Implicit method for fast full polarization array calibration". In: *General Assembly and Scientific Symposium (URSI GASS), 2014 XXXIth URSI*. IEEE. 2014, pp. 1–4.

[101] H.A. Sanjay and Sathish Vadhiyar. "Performance modeling of parallel applications for grid scheduling". In: *Journal of Parallel and Distributed Computing* 68.8 (2008), pp. 1135–1145. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2008.02.006. URL: http://www.sciencedirect.com/science/article/pii/S0743731508000464.

[102] Tom SANTE. "Development of (graphical) web applications for the processing and interpretation of arrayCGH data". In: (2010).

[103] Subhradyuti Sarkar and Dean Tullsen. "Compiler techniques for reducing data cache miss rate on a multithreaded architecture". In: *High Performance Embedded Architectures and Compilers* (2008), pp. 353–368.

[104] Shayan Shams et al. "A Scalable Pipeline For Transcriptome Profiling Tasks With On-demand Computing Clouds". In: *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*. IEEE. 2016, pp. 443–452.

[105] Jamie Shiers. "The worldwide LHC computing grid (worldwide LCG)". In: *Computer physics communications* 177.1-2 (2007), pp. 219–223.

[106] Shimwell, T. W. et al. "The LOFAR Two-metre Sky Survey - II. First data release". In: *A&A* 622 (2019), A1. DOI: 10.1051/0004-6361/201833559. URL: https://doi.org/10.1051/%200004-6361/201833559.

[107] T. W. Shimwell et al. "The LOFAR Two-metre Sky Survey - II. First data release". In: *arXiv e-prints*, arXiv:1811.07926 (Nov. 2018), arXiv:1811.07926. arXiv: 1811.07926 [astro-ph.GA].

[108] T. W. Shimwell et al. "The LOFAR Two-metre Sky Survey. I. Survey description and preliminary data release". In: *A&A* 598, A104 (Feb. 2017), A104. DOI: 10.1051/0004-6361/201629313. arXiv: 1611.02700 [astro-ph.IM].

[109] TW Shimwell et al. "The LOFAR Two-metre Sky Survey. I. Survey description and preliminary data relase". In: *Astronomy & Astrophysics* (2016).

[110] B Sigoure. *OpenTSDB scalable time series database (TSDB)*. 2012.

[111] Kevin Skadron et al. "Branch prediction, instruction-window size, and cache size: Performance tradeoffs and simulation techniques". In: *IEEE Transactions on Computers* 48.11 (1999), pp. 1260–1281.

[112] OM Smirnov and Cyril Tasse. "Radio interferometric gain calibration as a complex optimization problem". In: *Monthly Notices of the Royal Astronomical Society* 449.3 (2015), pp. 2668–2684.

[113] OM Smirnov and Cyril Tasse. "Radio interferometric gain calibration as a complex optimization problem". In: *Monthly Notices of the Royal Astronomical Society* 449.3 (2015), pp. 2668–2684.

[114] *Softdrive on the GRID*. Available at http://docs.surfsaralabs.nl/projects/grid/en/latest/Pages/Advanced/grid_software.html#softdrive.

[115] Stephen Strother et al. "Optimizing the fMRI data-processing pipeline using prediction and reproducibility performance metrics: I. A preliminary group analysis". In: *Neuroimage* 23 (2004), S196–S207.

[116] SURF. *Pre-Processing Pipeline*. https://www.surf.nl/en/services-and-products/grid/index.html. 2018.

[117] C. Tasse et al. "Faceting for direction-dependent spectral deconvolution". In: *A&A* 611, A87 (Apr. 2018), A87. DOI: 10.1051/0004-6361/201731474. arXiv: 1712.02078 [astro-ph.IM].

[118] C Tasse et al. "Facetting for direction-dependent spectral deconvolution". In: *arXiv preprint arXiv:1712.02078* (2017).

[119] Cyril Tasse. "Nonlinear Kalman filters for calibration in radio interferometry". In: *Astronomy & Astrophysics* 566 (2014), A127.

[120] Cyril Tasse et al. "LOFAR calibration and wide-field imaging". In: *Comptes Rendus Physique* 13.1 (2012), pp. 28–32.

[121] C Tasse et al. "Faceting for direction-dependent spectral deconvolution". In: *Astronomy & Astrophysics* 611 (2018), A87.

[122] Gerard Tel. *Introduction to Distributed Algorithms*. 2nd ed. Cambridge University Press, 2000. DOI: 10.1017/CBO9781139168724.

[123] Jeff Templon and Jan Bot. "The Dutch National e-Infrastructure". To appear in Proceedings of Science edition of the International Symposium on Grids and Clouds (ISGC) 2016 13-18 March 2016, Academia Sinica, Taipei, Taiwan. Oct. 2016. URL: https://doi.org/10.5281/zenodo.163537.

[124] Jeff Templon and Jan Bot. "The Dutch National e-Infrastructure". In: *International Symposium on Grids and Clouds (ISGC)*. Vol. 13. 18. 2016.

[125] S J Tingay et al. "The Murchison widefield array: The square kilometre array precursor at low radio frequencies". In: *Publications of the Astronomical Society of Australia* 30 (2013).

[126] Linus Torvalds and Junio Hamano. "Git: Fast version control system". In: *URL http://git-scm. com* (2010).

[127] Ger van Diepen and Tammo Jan Dijkema. *DPPP: Default Pre-Processing Pipeline*. Astrophysics Source Code Library. Apr. 2018. ascl: 1804.003.

[128] M. P. van Haarlem et al. "LOFAR: The LOw-Frequency ARray". In: *A&A* 556, A2 (Aug. 2013), A2. DOI: 10.1051/0004-6361/201220873. arXiv: 1305.3550 [astro-ph.IM].

[129] MP Van Haarlem et al. "LOFAR: The low-frequency array". In: *Astronomy & Astrophysics* 556 (2013), A2.

[130] MP Van Haarlem et al. "LOFAR: The low-frequency array". In: *Astronomy & astrophysics* 556 (2013), A2.

[131] JD Van Horn et al. *Grid-Based Computing and the Future of Neuroscience Computation, Methods in Mind*. 2005.

[132] R. J. van Weeren et al. "LOFAR Facet Calibration". In: *ApJS* 223.1, 2 (Mar. 2016), p. 2. DOI: 10.3847/0067-0049/223/1/2. arXiv: 1601.05422 [astro-ph.IM].

[133] RJ Van Weeren et al. "LOFAR facet calibration". In: *The Astrophysical Journal Supplement Series* 223.1 (2016), p. 2.

[134] RJ Van Weeren et al. "LOFAR facet calibration". In: *The Astrophysical Journal Supplement Series* 223.1 (2016), p. 2.

[135] RJ Van Weeren et al. "LOFAR facet calibration". In: *The Astrophysical Journal Supplement Series* 223.1 (2016), p. 2.

[136] I Virtanen et al. *Station Data Cookbook v1.2*. 2018. URL: http://lofar.ie/wp-content/uploads/2018/03/station_data_cookbook_v1.2.pdf.

[137] John Vivian et al. "Toil enables reproducible, open source, big biomedical data analyses". In: *Nature biotechnology* 35.4 (2017), p. 314.

[138] Jens-S Vöckler et al. “Kickstarting remote applications”. In: *2nd International Workshop on Grid Computing Environments*. 2006, pp. 1–8.

[139] Lin Wang. “Directed acyclic graph”. In: *Encyclopedia of Systems Biology*. Springer, 2013, pp. 574–574.

[140] A. Wilber et al. “LOFAR discovery of an ultra-steep radio halo and giant head-tail radio galaxy in Abell 1132”. In: *MNRAS* 473.3 (Jan. 2018), pp. 3536–3546. DOI: 10.1093/mnras/stx2568. arXiv: 1708.08928 [astro-ph.GA].

[141] WL Williams et al. “LOFAR 150-MHz observations of the Boötes field: catalogue and source counts”. In: *Monthly Notices of the Royal Astronomical Society* 460.3 (2016), pp. 2385–2412.

[142] Carl Witt et al. “Predictive Performance Modeling for Distributed Computing using Black-Box Monitoring and Machine Learning”. In: *CoRR* abs/1805.11877 (2018).

[143] Chen Wu et al. “Optimising NGAS for the MWA Archive”. In: *Experimental Astronomy* 36.3 (2013), pp. 679–694.

[144] Zhichen Xu, Xiaodong Zhang, and Lin Sun. “Semi-empirical Multiprocessor Performance Predictions”. In: *Journal of Parallel and Distributed Computing* 39.1 (1996), pp. 14–28. ISSN: 0743-7315. DOI: https://doi.org/10.1006/jpdc.1996.0151. URL: http://www.sciencedirect.com/science/article/pii/S0743731596901513.

[145] Leo T Yang, Xiaosong Ma, and Frank Mueller. “Cross-platform performance prediction of parallel applications using partial execution”. In: *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society. 2005, p. 40.

[146] Yong Zhao et al. “Swift: Fast, reliable, loosely coupled parallel computation”. In: *Services, 2007 IEEE Congress on*. IEEE. 2007, pp. 199–206.

[147] Jianjun Zhou and Martin Müller. “Depth-first discovery algorithm for incremental topological sorting of directed acyclic graphs”. In: *Information Processing Letters* 88.4 (2003), pp. 195–200.