



ASTERICS - H2020 - 653477

Technology Benchmark Report D-INT Work Package

ASTERICS GA DELIVERABLE: D3.19

Document identifier:	ASTERICS-D3.19.doc
Date:	1 May 2019
Work package:	WP3 OBELICS
Lead partner:	ASTRON
Document status:	Final
Dissemination level:	Public
Document link:	https://www.asterics2020.eu/documents/ASTERICS-D3.19.pdf

Abstract

An overview of the work carried out in the D-INT task in the OBELICS work package of the ASTERICS project in the last two years of its work. In particular, we focus here on benchmarks of various technologies that have been created or tested for data systems integration.

I. COPYRIGHT NOTICE

Copyright © Members of the ASTERICS Collaboration, 2015. See www.asterics2020.eu for details of the ASTERICS project and the collaboration. ASTERICS (Astronomy ESFRI & Research Infrastructure Cluster) is a project funded by the European Commission as a Research and Innovation Actions (RIA) within the H2020 Framework Programme. ASTERICS began in May 2015 and will run for 4 years.

This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, and USA. The work must be attributed by attaching the following reference to the copied elements: "Copyright © Members of the ASTERICS Collaboration, 2015. See www.asterics2020.eu for details of the ASTERICS project and the collaboration". Using this document in a way and/or for purposes not foreseen in the license, requires the prior written permission of the copyright holders. The information contained in this document represents the views of the copyright holders as of the date such views are published.

II. DELIVERY SLIP

	Name	Partner/WP	Date
From	Tammo Jan Dijkema	ASTRON/WP3	03 March 2019
Author(s)	Marcel Loose Tammo Jan Dijkema Thomas Vuillaume Bojan Nikolic Antonio Falabella Tarek Hassan Léa Jouvin Javier Rico Kay Graf David Carreto-Fidalgo José Luis Contreras	ASTRON Astron LAPP Univ. Cambridge INFN IFAE IFAE IFAE FAU UCM UCM	
Reviewed by	Rob van der Meer		26 May 2019
Approved by	AMST		28 May 2019

III. DOCUMENT LOG

Issue	Date	Comment	Author/Partner
1	01-03-2019	First draft	T.J. Dijkema (ASTRON)
2	30-04-2019	Contributions integrated	T.J. Dijkema / Rob v.d. Meer (ASTRON)
3	21-05-2019	Corrections and Introduction	T.J. Dijkema (ASTRON)

IV. APPLICATION AREA

This document is a formal deliverable for the GA of the project, applicable to all members of the ASTERICS project, beneficiaries and third parties, as well as its collaborating projects.

V. TERMINOLOGY

ASTERICS	Astronomy ESFRI & Research Infrastructure Cluster
CTA	Cherenkov Telescope Array
E-ELT	European Extremely Large Telescope
ESFRI	European Strategy Forum on Research Infrastructures
KM3NeT	Cubic Kilometre Neutrino Telescope
LOFAR	The Low Frequency Array

LSST	The Large Synoptic Survey Telescope
OBELICS	Observatory E-environments Linked by common ChallengeS
SKA	The Square Kilometre Array

A complete project glossary is provided at the following page:

<http://www.asterics2020.eu/glossary/>

VI. PROJECT SUMMARY

ASTERICS (Astronomy ESFRI & Research Infrastructure Cluster) aims to address the cross-cutting synergies and common challenges shared by the various Astronomy ESFRI facilities (SKA, CTA, KM3Net & E-ELT). It brings together for the first time, the astronomy, astrophysics and particle astrophysics communities, in addition to other related research infrastructures. The major objectives of ASTERICS are to support and accelerate the implementation of the ESFRI telescopes, to enhance their performance beyond the current state-of-the-art, and to see them interoperate as an integrated, multi-wavelength and multi-messenger facility. An important focal point is the management, processing and scientific exploitation of the huge datasets the ESFRI facilities will generate. ASTERICS will seek solutions to these problems outside of the traditional channels by directly engaging and collaborating with industry and specialised SMEs. The various ESFRI pathfinders and precursors will present the perfect proving ground for new methodologies and prototype systems. In addition, ASTERICS will enable astronomers from across the member states to have broad access to the reduced data products of the ESFRI telescopes via a seamless interface to the Virtual Observatory framework. This will massively increase the scientific impact of the telescopes, and greatly encourage use (and re-use) of the data in new and novel ways, typically not foreseen in the original proposals. By demonstrating cross-facility synchronicity, and by harmonising various policy aspects, ASTERICS will realise a distributed and interoperable approach that ushers in a new multi-messenger era for astronomy. Through an active dissemination programme, including direct engagement with all relevant stakeholders, and via the development of citizen scientist mass participation experiments, ASTERICS has the ambition to be a flagship for the scientific, industrial and societal impact ESFRI projects can deliver.

VII. EXECUTIVE SUMMARY

This document will focus on the work carried out in the D-INT task in the OBELICS work package of the ASTERICS project in the last two years of its work. In particular, it focuses on benchmarks of various technologies that have been created or tested for data systems integration.

The integration of data systems plays part on various levels in processing of astronomical data. The first contribution on **APERTIF** is on integrating an entire system, including the Westerbork telescopes themselves. The data in this case is the metadata about the state of the system, and the commands to control it. Tying together all these systems is an art in itself, which is of course very much agnostic of what the astronomical data will be used for - say radio astronomy or gamma ray astronomy. This makes that the control software is a piece that could profit from further collaboration between the ASTERICS partners.

Once data has been stored safely on disk, always further processing is needed to get the science out of it. These scientific pipelines very often require human feedback to tweak the pipeline while looking at the results. A modern technology that enables this is the Jupyter Notebook interface, enabling user interaction through a web interface, while the processes may run remotely on a supercomputer. **GammaBoard** builds on top of Jupyter notebooks to provide an easy interface for CTA data analysis, especially the GammaLearn project. Improvements to the underlying **GammaPy** have also been made, in particular to the data format. This enables a pipeline like the multi-instrument analysis of the Crab Nebula.

While designing pipelines, typically various version of the pipeline are run on the same data. Within these pipelines, some of the steps of the pipeline will perform exactly the same computation as the previous version. If these computations are very expensive, it is useful to eliminate this re-computation. This is what **Recipe** and **SWIFT/T CASA** aim to do. To be sure which version of the pipeline ran on data, or more broader to guarantee the reproducibility of data, **containerisation** can be used to fix the environment (all software used in a pipeline) for a certain run of the pipeline.

To improve the computation of expensive steps themselves, we can use GPUs to increase performance of the computer. For minimisation, a typical step in astronomical pipelines, **PyTORCH** enables use of the GPU with minimal change to existing code. This was benchmarked, and shows good results. Another performance boost can be to use a **parallel filesystem on low-power architectures**. A prototype of this has been built, showing that these optimizations do not exclude each other.

The contributions in this report cover various parts in the data processing pipeline. Because most of the contributions focus on integrating systems, they are not domain specific. That means they can be used across the disciplines, stressing the synergy between radio astronomy

and particle physics. Collaboration on the contributions has happened over the Obelics progress meetings, where prototypes of the contributions of this report have been presented.

The software presented in this report shows what can be achieved by using common standards, and using existing or emerging technologies.

Table of Contents

I.	COPYRIGHT NOTICE	1
II.	DELIVERY SLIP	1
III.	DOCUMENT LOG.....	2
IV.	APPLICATON AREA.....	2
V.	TERMINOLOGY.....	2
VI.	PROJECT SUMMARY	3
VII.	EXECUTIVE SUMMARY.....	4
	Table of Contents	6
1.	APERTIF monitoring and control software (ASTRON)	8
	Introduction.....	8
	Design considerations	8
2.	Gammaboard (LAPP/CNRS)	10
	Introduction.....	10
3.	Contribution to common DL3 format and Gammapy (IFAE and UCM)	12
	Introduction.....	12
	Improving the Gammapy package.....	12
4.	Towards open and reproducible multi-instrument analysis in gamma-ray astronomy (IFAE)	14
5.	Recipe (University of Cambridge)	16
	Introduction.....	16
	Benchmark, comparison with existing technologies.....	16
6.	SWIFT/T CASA (University of Cambridge).....	17
	Introduction.....	17
	Benchmark, comparison with existing technologies.....	18
7.	Containerisation for Data and Software Preservation (FAU)	19
	Introduction.....	19
	Benchmark, comparison with existing technologies.....	19
8.	PyTORCH Acceleration of function minimisation in astronomy (University of Cambridge)	21

Introduction..... 21

Benchmark, comparison with existing technologies 21

9. Benchmark of parallel distributed file systems on low-power architectures (INFN) 23

Introduction..... 23

Benchmark, comparison with existing technologies 23

10. Conclusion 24

11. Next steps..... 24

1. APERTIF monitoring and control software (ASTRON)

Introduction

The APERTIF project has improved the Westerbork Synthesis Radio Telescope (WSRT) with new phased-array feeds that enable it to observe 40 nearby pointings simultaneously. The extra flexibility brought to the system by having many pointings per dish, brings in the need for improved monitoring and control software.

The APERTIF system contains a lot of custom-made hardware that requires dedicated drivers. Due to its increased complexity, the new system requires a much more complex monitoring and control system (Figure 1), which needed to be built almost completely from scratch.

In building the control software, we have tried to use many off-the-shelf solutions, and have focused on interoperability between these components. The use of messaging middleware to facilitate the communication between custom high-performance components and off-the-shelf software will be essential for even larger projects like the SKA.

Design considerations

The software consists of a number of layers. The bottom layer contains the hardware drivers. All communication with the hardware is synchronous. The next layer contains the controllers, which are responsible for handling commands they receive and for serializing access to the drivers.

All communication is done using messaging middleware, based on the AMQP standard. This improves reliability and robustness in the communication between the controllers and services running on the different nodes. Messages are routed based on subject (Figure 2).

Every controller or service has a number of interfaces and must be able to handle a number of message types. The service interface provides synchronous RPC-like communication, and the functional- and management interfaces provide asynchronous command-response communication. Events and notifications will be sent to any service that subscribes to these messages.

The system is implemented with off-the-shelf technologies where possible. The main programming language is Python. C++ is used where performance is critical.

The software is GPL licensed and can be downloaded from our Subversion server: <https://svn.astron.nl/apertif/software>.

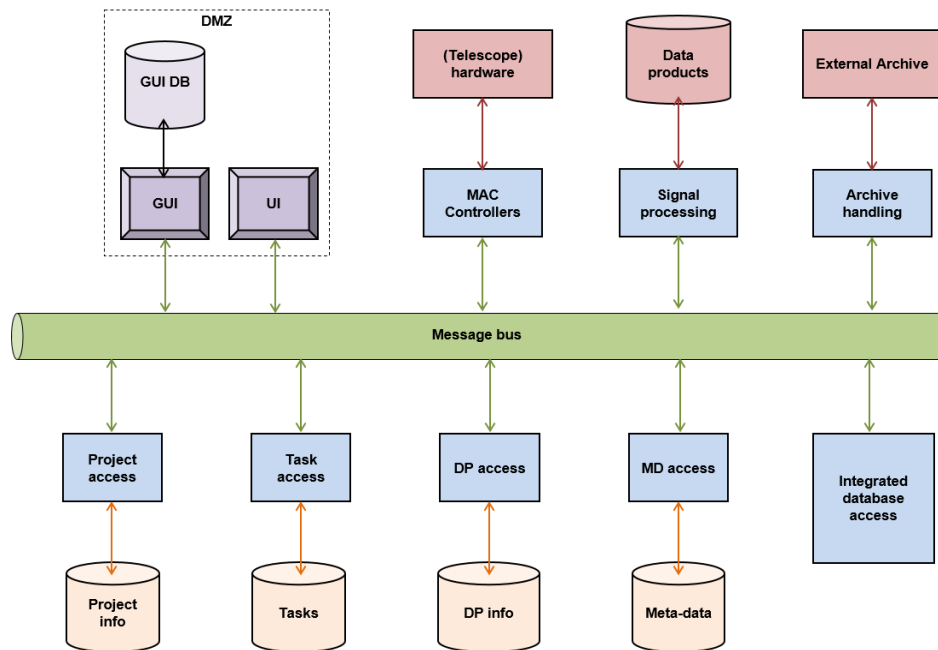


Figure 1. Overview of the APERTIF monitoring and control setup

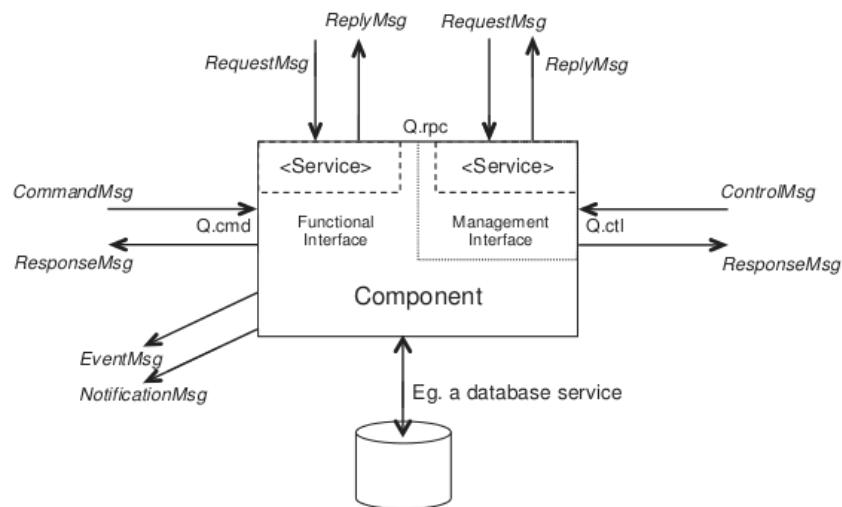


Figure 2. Overview of a component in the APERTIF setup, with messaging interfaces.

References

1. ASTRON-RP-1439 APERTIF Software Architecture v2.5
2. ASTRON-RP-1441 APERTIF Global Design System Services and MAC v1.3
3. ASTRON-SP-077 APERTIF Software Architectural Rules v0.4
4. ASTRON-RP-1510 APERTIF Controller Framework v1.3
5. ASTRON-RP-1517 APERTIF High-Level Design UniBoard Driver v1.0

2. Gammaboard (LAPP/CNRS)

Introduction

GammaBoard is a dashboard based on Jupyter notebooks technology, developed to display specific metrics assessing the reconstructions performances of Imaging Atmospheric Cherenkov Telescopes (IACTs).

Deep learning research is very experimental and is a lot about trials and errors, and bookkeeping of the different experiments realised. GammaBoard eases the bookkeeping and allows quick comparison of the reconstruction performances of your machine learning experiments.

It is a working prototype used in CTA, especially for the [GammaLearn](#) project.

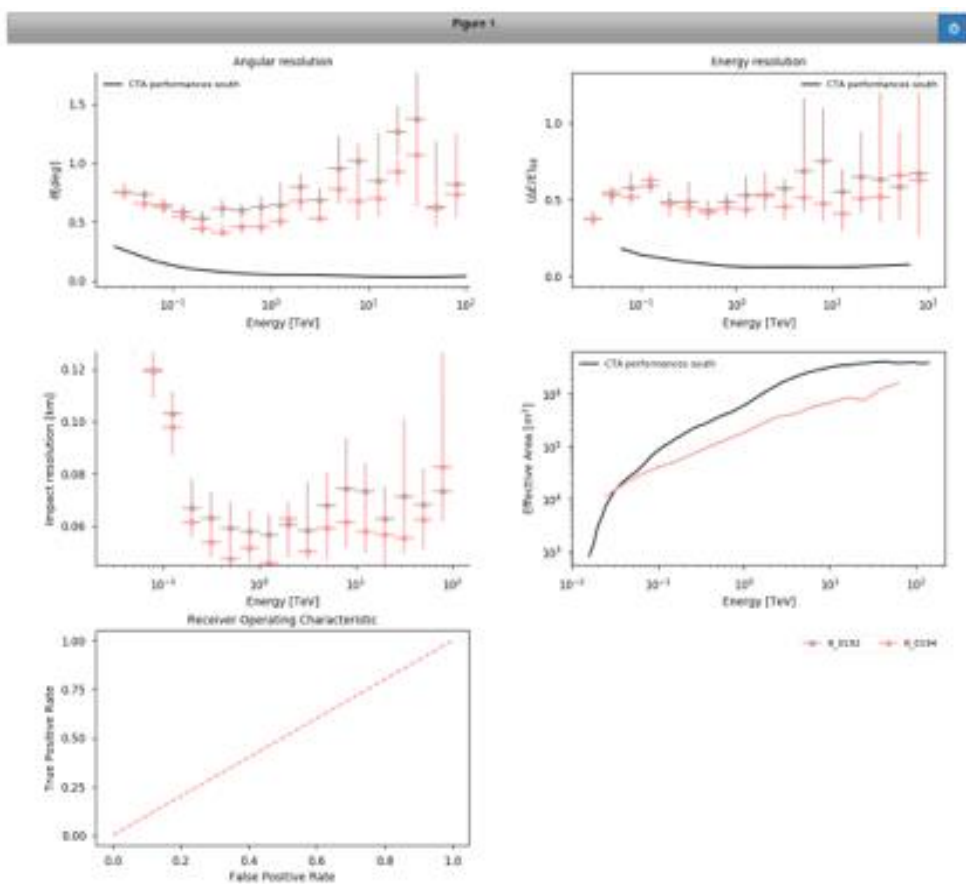
GammaBoard, written by Thomas Vuillaume, Mikael Jacquemont, is composed of:

- Plots (metrics) such as angular resolution and energy resolution.
- The list of experiments in the user folder.

When an experiment is selected in the list, the data is automatically loaded, the metrics computed and displayed. A list of information provided during the training phase is also displayed.

Since GammaBoard is built on top of Jupyter, it is possible to interactively further explore the data with the command interface (Figure 3).

As many experiment results as needed can be overlaid.



Command interface



Figure 3. Illustration of GammaBoard. The plots on top show aggregated results of the data. The command-interface makes it possible to interactively explore the data, while the bottom half shows detailed statistics.

3. Contribution to common DL3 format and Gammapy (IFAE and UCM)

Introduction

The data level 3 (DL3, [1]) format is an open high-level data format currently developed for the Cherenkov Telescope Array (CTA, [2]) observatory. The data files are stored using the FITS file format and are made up of a binary table containing all event-wise parameters required by the scientific analysis of the data (e.g. event energy, time of arrival, instrument response, etc.). A full description of the format can be found in <https://github.com/open-gamma-ray-astro/gamma-astro-data-formats> [3].

Gammapy [3] is an open-source python package for gamma-ray astronomy built on Numpy and Astropy. It is able to analyze the DL3 data format to create sky images, spectra and lightcurves of gamma-ray sources. It is currently used for the simulation and analysis of observations of the Cherenkov Telescope Array (CTA) and has been proposed as a prototype for the CTA science tools. Apart from its usage for CTA, it has also proven to be able to analyze real data from current gamma-ray astronomy instruments and combine data from different observatories by means of the DL3 data format [4].

Improving the Gammapy package

Our contributions to Gammapy led to new features in the analysis of DL3 data. We focused on the handling of the observation data in the early stages of an analysis when the user selects the relevant events from event lists stored on disk. An analysis with Gammapy is usually carried out in a Jupyter Notebook, and hence the selection process should maintain the dynamical nature of the notebook.

The *Observation* class is the main container class in Gammapy holding all the information necessary for an analysis, which is the event list, observation time and instrument response. Gammapy applies a lazy loading scheme via the *DataStore* class, in which the data is only loaded into memory when needed for computation and only in small chunks, that is one *Observation* object at a time, which usually corresponds to about 20-30 minutes of observation time. This enables the analysis of huge amounts of data without running into memory shortage issues. To allow for a continuous event selection in time or phase, we implemented observation filters that are applied on-the-fly when loading the events and GTIs into memory. For this, an instance of an *ObservationFilter* class is attached to an *Observation* object, which is automatically applied when the relevant data is accessed (see Figure 4 and Figure 5). This approach is fully compatible with the lazy loading scheme of Gammapy and permits a fast and dynamical selection process throughout an analysis, since the filters can be

updated at any stage (before carrying out computations on the events). We also added convenient methods to the *Observation* and *Observations* class (the latter being a collection of *Observation* objects) that allow a direct and user-friendly update of the observation filters. The implementation of the observation filters was guided by a PIG (Proposal for Improvement of Gammapy) and was part of a new observation handling scheme in Gammapy (the open pull request is found here: <https://github.com/gammapy/gammapy/pull/1877>).

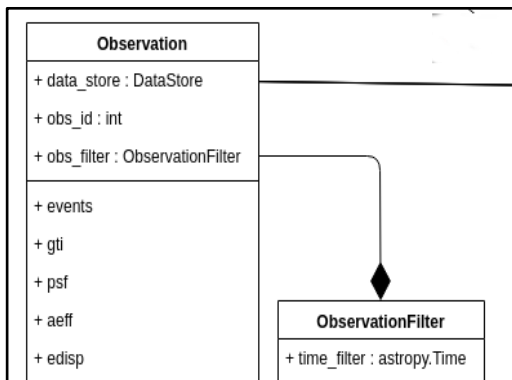


Figure 4. Sketch of the new class diagram (UML) in the Gammapy package, zoomed-in to the observation handling

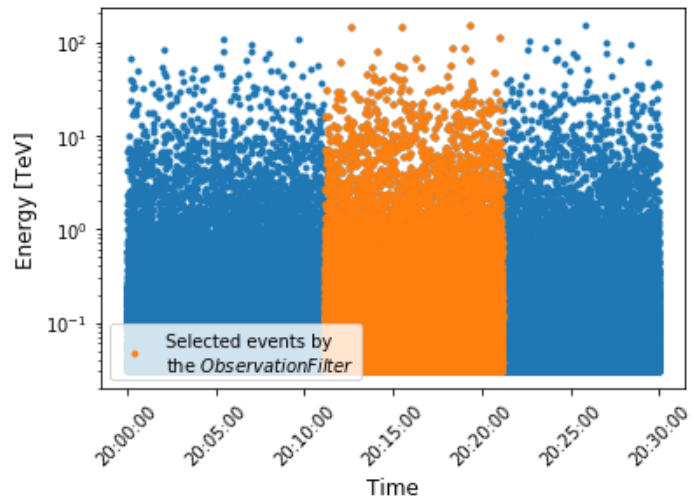


Figure 5. Result of a simple event selection by time with the new observation filters.

The development of a 3D or cube analysis to simultaneously fit a spectral and morphological model to the data is a challenge in Cherenkov gamma-ray astronomy. The strong variation of the instrument response functions and of the residual cosmic rays background with the observation conditions makes it very difficult to build a coherent background model across the whole energy range. Nevertheless, with improving sensitivities and angular resolution of current Cherenkov telescopes and next generation instruments such as the Cherenkov Telescope Array, the complex morphology of the regions with diffuse emission or multiple sources requires the development of this technique. We are also involved in the development of this analysis in Gammapy. The conversion of current Cherenkov telescope data as we do in MAGIC (report in D3.18) is also a major step in order to test this new kind of analysis on data we already know.

4. Towards open and reproducible multi-instrument analysis in gamma-ray astronomy (IFAE)

We contributed to the first joint likelihood analysis of the Crab Nebulae (Figure 6). For this, we used the data from H.E.S.S., MAGIC, VERITAS, FACT and Fermi. Our analysis focuses on the MAGIC side (with the previous DL3 converter for MAGIC data reported in D3.18) and the VERITAS side. The likelihood fit and the combination of the data was obtained using the software Gammapy presented above. The results presented in this work are obtained using open-access online assets that allow for a long-term reproducibility of the results. This work is a major achievement in order to promote the use of open-source tools and data in gamma-ray astronomy. Having all the current Cherenkov data in the DL3 format for data archival will allow us to perform this kind of analysis for multiple sources including CTA data and later on using the 3D analysis that requires a certain type of instrument response function that are not yet available for MAGIC or VERITAS.

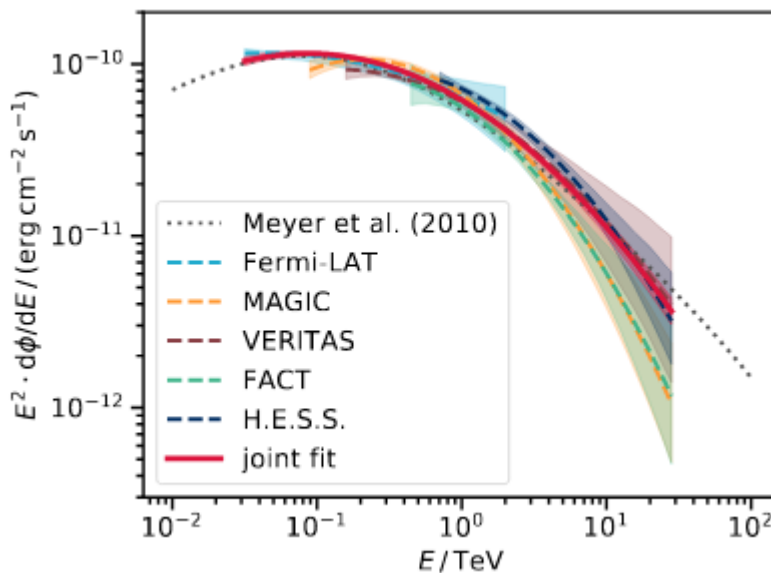


Figure 6. The first joint likelihood analysis of the Crab Nebulae using the data from H.E.S.S., MAGIC, VERITAS, FACT and Fermi.

References

- [1] Contreras, J. L. et al. 2015, in Proceedings, 34th International Cosmic Ray Conference (ICRC): The Hague, The Netherlands, July 30 - August 6, 2015, Vol. 34, 960
- [2] The Cherenkov Telescope Array Consortium, 2017, "Science with the Cherenkov Telescope Array", eprint arXiv:1709.07997
- [3] Dell C. et al. <https://github.com/open-gamma-ray-astro/gamma-astro-data-formats>.

- [4] Deil, C. et al. 2017, “Gammapy - A prototype for the CTA science tools”, in Proceedings, 35th International Cosmic Ray Conference (ICRC): Busan, South Korea, Vol. 34, 766
- [5] Nigro, C. et al. 2019, “Towards open and reproducible multi-instrument analysis in gamma-ray astronomy”, eprint arXiv:1903.06621

5. Recipe (University of Cambridge)

Introduction

Recipe implements a technique to automatically minimise the re-computation when a data analysis program is iteratively changed, or added to, as is often the case in exploratory data analysis in astronomy. A typical example is flagging and calibration of demanding or unusual observations where visual inspection suggests improvement to the processing strategy. The technique is based on memorization and referentially transparent tasks. We describe the implementation of this technique for the CASA radio astronomy data reduction package. We also propose a technique for optimising efficiency of storage of memorized intermediate data products using copy-on-write and block level de-duplication and measure their practical efficiency. We find the minimal re-computation technique improves the efficiency of data analysis while reducing the possibility for user error and improving the reproducibility of the final result. It also aids exploratory data analysis on batch-schedule cluster computer systems.

Benchmark, comparison with existing technologies

Recipe is in production use in several institutions and is especially useful to students experimenting with data reduction pipelines.

There is virtually no speed hit for including recipe in your code on a first run but when re-running an old code the speedup is approaching 100%.

It is important to note that to really offer the improvements in performance a modern Copy-on-Write file system should be used.

References

Minimal Re-computation for Exploratory Data Analysis in Astronomy, Nikolic, Small and Kettenis, proceedings of ADASS 2017, arXiv:1711.06124

Minimal Re-computation for Exploratory Data Analysis in Astronomy, Nikolic, Small and Kettenis, Astronomy and Computing 2018A&C....25..133N, arXiv:1809.01945

6. SWIFT/T CASA (University of Cambridge)

Introduction

Accelerating the time-to-solution for analysis of radio interferometry data is desirable for a number of reasons, e.g.:

- More efficient use of scientist/analyst time when the analysis is not fully automated
- More efficient use of valuable computing resources, e.g., very fast storage
- More timely response to time-critical phenomena, whether they are of astronomical origin (i.e., transient sources) or in the telescope (e.g., a fault developing which is subtly corrupting the data)

The speed of interferometric data processing can be limited by a variety of computing resources, including at least:

- Storage input/output throughput
- Working memories capacities, latencies and throughputs
- CPU instruction throughput and latencies

The balance of how these factors determine the speed of processing depends on the configuration of the telescope, type of observation and processing required. The balance is predictable in principle, but not easily so, and this balance has very substantially changed in the past with the evolution of computer hardware architectures.

For this reason, the way to accelerate the time-to-solution for a general-purpose radio astronomy problem is not obvious. My subjective experience is that improvements with respect to a particular use case tend to have mixed, but on average a positive, success in general.

In this memo, I investigate distributing CASA (Common Astronomy Software Applications, the tool set regularly used in radio astronomy) processing by combining it with the SWIFT/T system described by Wozniak et al. (2013), with the ultimate objective accelerating time to solution.

SWIFT/T consists of a programming language (SWIFT) and a dataflow engine (Turbine). The word dataflow is used here in the technical sense of Johnston et al. (2004). The most relevant feature of the SWIFT language is the use of strictly single assignment data structures (an idea

originating, I believe, in the I-Structures of the Id programming language, see Arvind et al. 1989) which makes it easy for SWIFT to infer the dataflow semantics of the program.

Benchmark, comparison with existing technologies

Simple initial tests of scalability were performed using a simple example application. Only 23 input datasets were used, limiting available parallelism. Scalability was investigated by varying the number of SWIFT/T workers on both a single node and a cluster

On the single node tests, the program `/usr/bin/time` was used to record the wall clock time. Other parameters, including the total number of pages read and written to disk were recorded as well. On the Darwin cluster the total MPI execution time reported by the Turbine PBS script was used as the wall-clock run time. The results of simple initial scalability tests are shown in Figure 7.

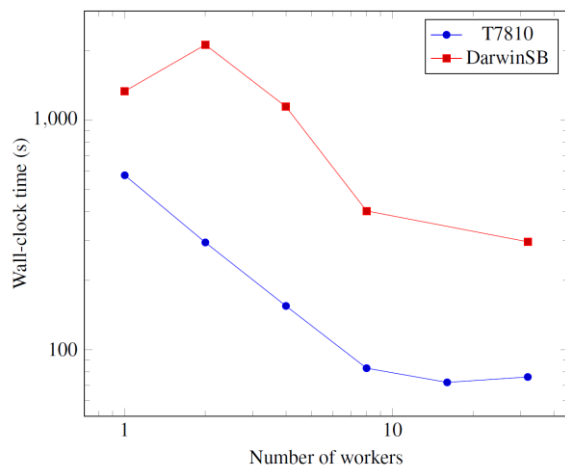


Figure 7. Result of single node tests, where in each test the number of workers is doubled.

In the single node tests, the `/usr/bin/time` program reported total file system input was about 33 GigaBytes (GB) read and 15GB written. For the fastest execution this corresponds to an average read rate of 460 MB/s and simultaneous average write rate of 200 MB/s. Although the P3700 is capable in ideal conditions of even higher read speeds (up to 2,800 MB/s is claimed in the sales brochure), these very high achieved I/O figures suggest that the plateau in scalability at around 8 workers is due to throughput limitations of the I/O subsystem.

Run times on the Darwin cluster were, in most cases, substantially longer than on the single node. Running CASA interactively showed similar performance. I have not determined the reason for this; it seems the most likely factor is the performance of the file systems used for the type of I/O access that CASA does.

7. Containerisation for Data and Software Preservation (FAU)

Introduction

Scientific research bases on interoperability and reproducibility of results. Methods, procedures, software and data have to be preserved in a way that they remain re-usable over and beyond the full lifespan of research infrastructures (RIs). This includes the ability to run analyses and data processing steps in the quickly evolving software environments, and on the heterogeneous computing systems and architectures found in the RIs. FAU as partner of the KM3NeT collaboration has helped to define and develop workflows based on state of the art containerisation tools and techniques to achieve these goals while minimising the compatibility requirements.

The web-based open source Git-repository manager [GitLab](#) provides all the ingredients for future-proof software development including essential features like version management, issue tracking, discussion platforms, and continuous integration and deployment. The latter is based on Docker, which performs operating-system-level virtualisation (containerization). Each software and analysis project is developed, compiled, and tested in an isolated and independent container and later also deployed to a target production system as a single bundled image including all requirements.

These so called Docker images serve as a starting point for Singularity images, which is another containerisation solution specifically designed to run on HPC clusters and thus is well suited for fully reproducible data processing or analysis chains running on large and sometimes heterogeneous computer clusters and grids. Singularity images have proven to be a convenient solution with minimal setup, maintenance and compatibility requirements. Docker and singularity registries are used to deploy the containers to the data processing sites and in the end, only the necessary data is added to the containers to have a fully preservable data/software bundle that is stored in the long-term storage systems of the collaboration.

Benchmark, comparison with existing technologies

Two dedicated server instances were set up, one for the GitLab software development and one for continuous integration and deployment based on containers and the container registries.

The system was tested and benchmarked to proof that it can handle the development tasks of the KM3NeT collaboration. Also, employing the containers for data processing and analysis on PCs, servers and HPC clusters has been benchmarked and no significant drop of performance has been found.

There have been different approaches on data and software preservation, e.g. self-describing formats for the data, or packaging of software using the software packages repositories of large Linux distributions. While continuous integration and continuous deployment are no innovative solutions in themselves, the extension to data and software provenance in an integrated end-to-end software development and deployment system to be used at each step of the full data processing pipeline, and at the same time producing long-term preservation for those tasks, is – to our knowledge – something new in the community.

References

Tamás Gál, “Data and Software Preservation through Containerisation in KM3NeT”, The New Era of Multi-Messenger Astrophysics, Groningen, 2019

8. PyTORCH Acceleration of function minimisation in astronomy (University of Cambridge)

Introduction

Minimisation (or, equivalently, maximisation) of non-linear functions is a widespread tool in science, engineering and information technology. Common examples of its use are maximum likelihood and maximum a-posteriori estimates of model parameters given a set of measurements; and, optimisation of system designs by minimising a cost function.

Training of neural networks can also be expressed as a minimisation problem. Although the algorithms used for this minimisation are sometimes specific (to take into account that only a part of the available training set is considered at a time). Rapid adoption of neural networks in information technology systems has led to significant investment into software to support their training, including PyTorch. The new software packages developed in this area emphasise both efficiency and ease of use.

Benchmark, comparison with existing technologies

Relative performance of the NumPy and PyTorch variants were measured on a dedicated Dell PowerEdge server with dual socket Intel Xeon E5-2630 CPUs and dual NVIDIA Tesla K20c GPUs.

Measurement was made as function of:

1. Number of model parameters, represented by the maximum order of Zernike polynomials used, n . The actual number of polynomials up to order n is $n(n+3)/2+1$, so for example if Zernike polynomials of the order $n = 8$ are used, there are 45 parameters to be optimised.
2. Computational cost of the model calculation, represented by the number of pixels N in each dimension of the grid.

Simulated measurements (including simulated noise) were used as an input into the phase retrieval and it was found that the different implementations converged to the same result up to the tolerances specified to the BFGS algorithm¹.

Measured time-to-solution as a function of grid size is shown in Figure 8.

¹ Broyden-Fletcher-Goldfarb-Shanno, see *Fletcher, Roger (1987), Practical Methods of Optimization (2nd ed.)*, New York: [John Wiley & Sons](#), ISBN 978-0-471-91547-8

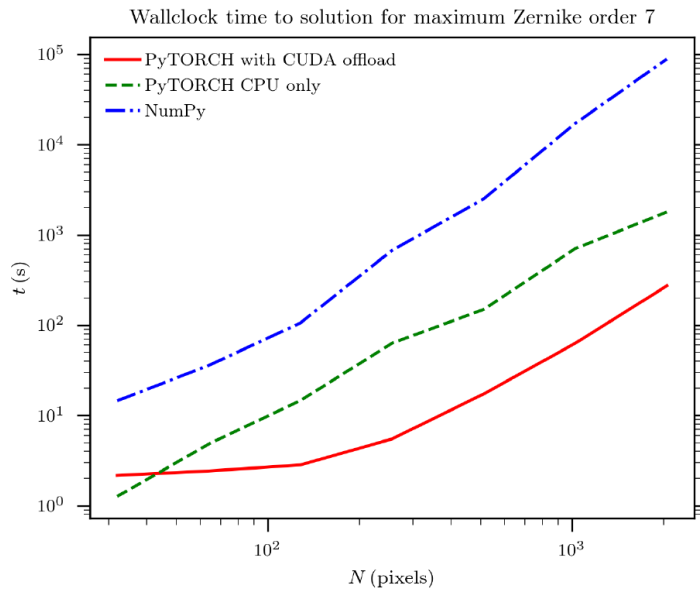


Figure 8. Performance of NumPy, PyTorch with and without CUDA offload to the GPU.

It can be seen that the PyTorch implementation is far faster in all configurations, and that the GPU-offloaded execution is faster than the CPU-only execution above grid size $N \geq 64$ (note that the grid size is typically a power of two). For intermediate and large grids ($N > 256$) the PyTorch implementation running on CPUs is approximately an order of magnitude faster than the NumPy implementation, while the GPU offloaded execution is an order of magnitude faster still. Measured time-to-solution as a function of the number of model parameters is shown in Figure 9.

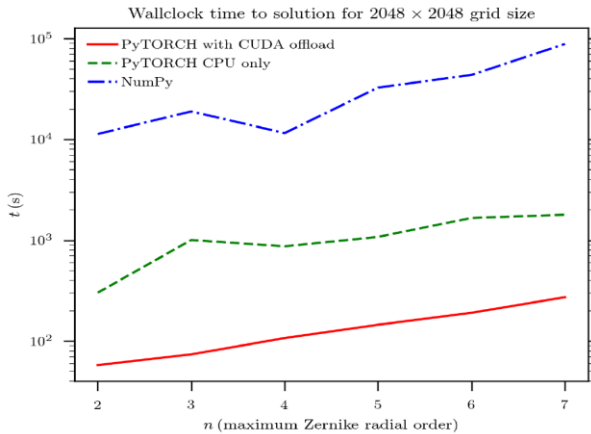


Figure 9. Wallclock time to solution as a function of the number of free parameters.

The large improvement in speed by PyTorch is seen to be apparently independent of the model parameters, at least in this case.

9. Benchmark of parallel distributed file systems on low-power architectures (INFN)

Introduction

We aimed at benchmarking the BeeGFS parallel cluster filesystem on low power SoCs such as ARM but also x86. We chose BeeGFS because it is open source, actively developed and also because it is easy to deploy. The SoC architectures lend themselves very well for IO-bound application, and allow to limit the costs in terms of hardware and energy consumption.

Benchmark, comparison with existing technologies

We tested BeeGFS on a small but complete setup (four machines and 8TB of disk space), with filesystem operation such as concurrent copy, and data transfers and also with parallel application such as mpi. Our test proved such setup is easy to obtain and it can be extended without particular concerns. Our test bench showed that a low power storage solution is possible without evident I/O limitations.

10. Conclusion

Several systems have been developed and then tested or benchmarked versus other existing systems to show optimal behaviour in required circumstances. Even when developing with the best intentions, one should not assume new ideas immediately lead to better performance. This should be tested and demonstrated.

The presented results show that good progress has been made over the last two years of the ASTERICS project.

11. Next steps

All these results and underlying software (and hardware) developments are part of (soon to be) running systems that will be further developed for years to come, as long as scientist will analyse the data form the astronomy and astroparticle physics RIs. Many of the developments are well documented in Git systems and it is the interest of the scientists to keep the new and older versions accessible.