Facilities for Remote Execution of High Performance Applications in Astronomy

Ana Luisa Solórzano[a], Vinicius Monego[a],
Haroldo Fraga de Campos Velho[b], Andrea Schwertner Charão[a],
Alice Kozakevicius[a] and Renata Sampaio da Rocha Ruiz[b]
[a]Universidade Federal de Santa Maria, RS, Brazil
[b]National Institute for Space Research, São José dos Campos, SP, Brazil

Abstract

The field of Astronomy deals with a large amount of observational and simulated data. There are several algorithms and software to process this data, revealing information about star dynamics, the center of galaxies, or even the structure of the Universe. Virtual observatory (VO) is a strategy to take advantage of distributed computer systems integrated by the Internet, sharing hardware and software resources for data storage, processing and analysis. This work is a VO effort for remote execution of high performance applications in Astronomy. The VO tool was developed using the Python programming language with the Django web framework and the Celery distributed task queue. Currently, the platform offers new parallel versions of the Friends-of-Friends algorithm for astronomical objects classification, and algorithms for image restoration using wavelet filtering. The web platform is hosted in an in-house cluster with different parallel architectures to be explored. Users can set their execution parameters or use the default ones. In addition, they can choose input files from the portal's database or provide their own. At the end of each execution, the user can download the output file and a log file.

Keywords: astronomy, virtual observatory, remote execution, high performance computing.

1 Introduction

Due to scientific advances in the Astronomy area with more powerful telescopes and detectors, for example, the amount of data generated in simulations is increasing. Allied to this, advances in the computing area, especially in the processing and networks fields, allowed the generation of more powerful computers to process, store and analyze a huge amount of data, even if stored in different computational environments needing faster communication networks. However, many of these algorithms are not available to the

community, especially because they are implemented by private initiatives dealing with license issues or because they do not practice the reproducibility of their work.

Seen this, virtual observatories (VO) initiatives are great alternatives to keep applications documented and available for execution in the same environment easily accessible via web-based tools. A VO is capable of generating significant benefits for the astronomy research community and any other interested group as students and educators, at the same time taking advantage of distributed computer systems integrated by the Internet.

This work is based on a previous web portal framework implementation for remote execution of a specific algorithm in Astronomy [7]. The main objective is to provide a VO that allows registered users to execute Astronomy applications using different data and obtaining the result of their executions after completing. It is developed as free and open source software that anyone can self host and add their own data and applications as services.

The project includes two high performance applications: the *Friends-of-Friends* algorithm implemented in serial and parallel versions and algorithms for image restoration using wavelet filtering. The platform is hosted in a cluster maintained by LAC/INPE containing different parallel architectures including CPUs, GPUs and FPGAs. For the executions, users can set their parameters and upload their input files using the VO interface, generating at the end output files about the executions.

2 Virtual Observatories

Virtual observatories are an international community-based initiative to provide information on astronomical data archives in easily accessible worldwide environments. There are some VO projects available, that share similarities and differences with ours. Some of them are:

- The European Virtual Observatory [1] (EURO-VO): It aims at deploying an operational VO in Europe and provides a set of scientific applications that can be used as a service. However, registration is not public and it couldn't be tested.
- The Chilean Virtual Observatory[2] (ChiVO): It was developed in Chile due to the large amount of data coming from the astronomical observatories in the country that needed to be stored, processed and

[1] https://aladin.u-strasbg.fr/aladin.gml
[2] https://chivo.cl/

analyzed using tools and algorithms. ChiVO tends to be useful to astronomers and students, specially from schools of rural areas. Some of the Official Software available are a Python library to stack images of astronomical objects, a Python package to provide an easy interface and I/O operations for using astronomical libraries over Jupyter notebooks, a virtual service to generate synthetic spectroscopic data that can be used to assess advanced computing algorithms for astronomy and a package with algorithms for Astronomers, including some produced by ChiVO and their High Performance Computing versions. Like EURO-VO, registration is not public and the service is not useful outside of Chile.

- Virtual Astronomical Observatory[3] (VAO): The U.S. National Science Foundation and NASA funded this international effort to bring a large-scale electronic integration of astronomy data, tools, and services to the global community. VAO seeks to put efficient astronomical tools in the hands of U.S. astronomers, students, educators, and public outreach leaders. Some of the available tools run online and others have to be downloaded and installed locally. Also, they make available a database to use in the executions, in which the user can see the progress of their task and download the files used.

3 Web Platform

3.1 Implementation

The platform is built upon the tools: (i) Django Web Framework, because it offers access to several extensions which facilitated the project development, such as object-relational mapping and a Model-View-Template (MVT), (ii) Celery, an asynchronous task manager based in message exchanges between the application that will launch the tasks and the workers (processes that execute the tasks), making possible to expand the number of workers in distinct machines and (iii) Redis[11], which coordinates sending and receiving messages between the machines that will act like workers and the machine that will keep the application available for the users.

To host the VO in the cluster of the authors' institution, there was a strong dependence on the IT management sector to authorize the installation of tools and enabling access to network services necessary for the deployment of the platform. The development and tests presented in this work

[3] http://www.virtualobservatory.org/

were tested and executed locally, meanwhile we are preparing the execution environment on the cluster for deployment.

The project is divided into classes that inherit the Django model base. The classes of the available algorithms were implemented separately, since each one uses different attributes including their description, execution command and parameters. There is an execution class that keeps all the information about one execution: the user ID, the date requisition, the execution status, the algorithm used, the input, output and log files, and the execution time. The last one is the portal class, that keeps the user's information, needed for authentication, and other user preferences related to the VO interface.

There are three distinctive users types:

- Anonymous: can visualize information about the platform and the registered algorithms, register as a new user and contact the system administrator;
- Registered: can execute experiments with all available algorithms, visualize the executions of the experiments monitoring the status of their job and download its respective input, output and log data or cancel the execution;
- Administrator: can register new algorithms, edit the users' account and any other operation related to the database.

3.2 Functionalities

Compared to the original implementation, which was unmaintained since 2015, the software stack was upgraded to the latest versions as of the date of publication: Django 2.1, Celery 4.2, Bootstrap 4.3. The most noticeable architectural change is the division of specific functionalities into Django *apps*, a change referred here as *modularization*, that was important to ease the addition of new applications.

During development we had to modify some Django *models* in order to facilitate the storage in the database and to add more information. In the templates we modified some navigation options to improve the user experience.

Besides the platform functionalities we also had to fill some web pages with textual information about the VO, the project's idea, the usage instructions and the available algorithms. The source code of the algorithms

also had to be adapted to run easily by command line, and the outputs had to be translated and standardized for better understanding.

To obtain and validate the forms' information, we used the *Django Crispy Forms* application. When a user incorrectly fills a form, Crispy Forms tries to render a new one by making changes to the fields filled incorrectly, making it clear where the errors are. For user registration, we use the *Django-Registration-Redux*, which has specific templates and forms for registration. We implemented the user account activation via email using this extension.

To guarantee the performance of the remote executions, avoiding that the same process dealt with user requests and the execution of tasks, the project was divided into two types of software servers:

- Front-end server: runs a web server developed in Django, that implements the user interactions with the workers. The machine that hosts the portal maintains a process for implementing the broker Redis.
- Workers: are independent processes responsible for managing the execution of algorithms. They should be started on the server machine that will run the applications and exchange messages with the process that requested the execution. We use the Python package Celery [2], which is an application to manage job queues by exchanging messages to other processes and/or machines.

4 Currently implemented applications

Modularization allowed each application to have its own distinctive Django *app*, making it easier to add new applications. The currently available algorithms hosted in the platform are:

4.1 Friends-of-Friends

Friends-of-Friends (FoF) [6] is a percolation algorithm used to identify structures in the Universe based on physical proximity. This is one of the most used percolation methods, due to its simplicity (it has only one free parameter) and reproducibility (to a given linking length it produces one catalog of unique groups) [9].

As input, FoF receives data from N-body cosmological simulations and classifies them considering a percolation radius, defined by the user. Each

line of the input file defines a particle, that may be a star, a galaxy, a cluster or a cluster of galaxies.

The algorithm considers a sphere of radius R around each particle of the total input set. If there are other particles within this sphere, they will be considered belonging to the same group and will be called friends. The procedure continues considering a sphere around each friend and using the rule of "any friend of my friend is my friend", until no new friends can be added to the group. The expected result of the algorithm is to identify groups of objects that interact gravitationally.

To improve FoF's performance taking advantage of modern CPU designs, there are several implementations of the algorithm using parallel-based approaches. Other works presented some FoF versions using parallelization tools, such as MPI, OpenMP and OpenACC [10, 1, 12]. Table 1 presents the four FoF implementations available in the VO.

Table 1: Friends-of-Friends algorithms available in the VO

Complexity	Serial	Parallel	Runs in	Tools
$O(NlogN)$	X		CPU	
$O(N^2)$	X		CPU	
$O(N^2)$		X	CPU	OpenMP
$O(N^2)$		X	CPU + GPU	OpenACC

4.2 Image Restoration

Visual data acquired by astronomical instruments more than often are affected by some form of corruption. Technical difficulties like vibrations and electromagnetic interference can profoundly degrade the result [8]. Acquired images in their raw form may not be useful at all for researchers, requiring some form of post-processing.

Digital image restoration is about finding the best estimation of the original image knowing only the degraded result. There are many forms of degradation. In this work, however, we focus on the Additive Gaussian White Noise (AWGN) form of degradation.

Several techniques exist to reduce noise from images, of which only wavelet filtering is currently implemented. In a nutshell, the wavelet transform decomposes the signal into low and high frequency sub-bands. Since noise adds a lot of amplitude variation into the signal, noise propagates to the high frequency sub-bands, and those are the bands to be filtered by applying a *threshold* operation on them.

Wavelet filtering is a vast area by itself. Several wavelet families and threshold estimators exist. We focus on the Daubechies family and the VisuShrink [5] and BayesShrink [3] threshold estimators.

VisuShrink is the simplest estimator. It uses a Universal Threshold calculated from the standard deviation of the high frequency diagonal sub-band and applies it to every sub-band with a high frequency decomposition in some dimension. BayesShrink is adaptive, meaning each of these sub-bands have their own threshold value and generally yields better results.

In addition, the algorithm of *cycle spinning* [4] was included. For different choices of the number of shifts and the shift amount itself, the filtering process is applied on each shifted signal and the result is unshifted by the corresponding shift quantity. The final result is the signal obtained by the average of all unshifted results. This procedure is useful to reduce artifacts, for instance, caused by the Gibbs phenomenon generated by the periodized wavelet decomposition, at the expense of complexity and consequentially higher execution time. With the maximum number of shifts assumed for computing the average, cycle spinning has $O(Nlog(N))$ complexity, while regular wavelet filtering is $O(N)$.

The image restoration algorithms included in the VO are from the *scikit-image* [13] project, version 0.14. Being developed in Python, it integrates seamlessly with Django.

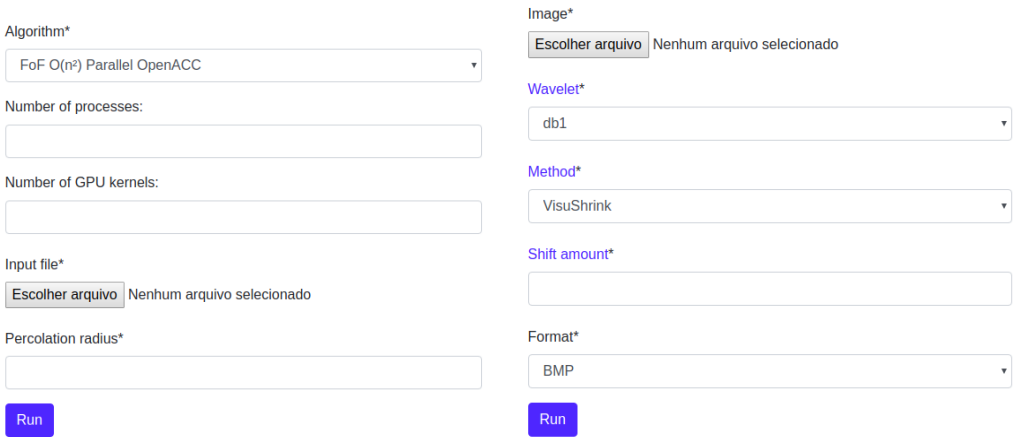
5 Remote execution

As soon as a registered user logs in for the first time, they will be presented with an "empty" home page saying "You have no experiments". This is an empty experiment table, or execution panel, that will begin to pile up as the user run experiments.

Registered users can perform experiments using the navigation bar, where the algorithms are. Since many distinct FoF and image restoration algorithms exist, it is a drop-down menu where the user chooses a specific algorithm before hitting the experiment configuration page.

Each experiment chosen from the navigation bar has it's own input form for configuration. Figure 1 shows the Django forms where the user can input their data and set execution parameters for both implemented *apps*.

As soon as the user hits the "Run" button, they are redirected to their home page and the execution panel is updated. The user will be notified by email when the task is completed and can then refresh the page to enable the download button.



(a) Friends-of-Friends (b) Image restoration

Figure 1: Django forms for both *apps*

Each job execution request generates an entry on a table which registers all requests of a given user. This table is presented in a section of the web portal (Figure 2). It has an interface for users to monitor and manage their execution requests (select, view, remove). This interface also allows the user to download output data after the execution is finished. The table also presents the following information to the user:

- Execution time: elapsed time of a job execution (for finished jobs).
- Status: tells the user whether a request is already finished or still waiting a worker to perform the task.

Figure 2: Experiments table

6 Conclusions

Virtual Observatories are an efficient framework to share knowledge and data associated with Astronomy, allowing a secure spread of information. This work presented a VO effort for remote execution of high performance applications, which contribute for data storage, access and analysis.

The full source code along with instructions to locally run it can be found in the Git repository at `https://gitlab.com/monego/webfriends-imagens`. Thanks to modularization, the code should be easy to fork and to adapt for custom applications.

As future works, we intend to finish hosting the platform in the cluster and to study other applications to be inserted in the platform. Another improvement we want to achieve is the implementation of the possibility of scheduling executions, facilitating the user's planning. More ideas for further improvements can be found in the issue page of the repository.

Acknowledgments. The authors gratefully acknowledge financial support from the CNPq, Brazilian agency for research support, in partnership with Universidade Federal de Santa Maria (UFSM) and the National Institute for Space Research (INPE), that granted the Scientific Initiation scholarship to the first and second authors.

References

[1] L. Berwian, E. T. Zancanaro, D. J. Cardoso, A. S. Charão, R. S. da Rocha Ruiz, and H. F. de Campos Velho. Comparação de estratégias de paralelização de um algoritmo friends-of-friends com openmp. In *Anais da XVII Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul*, pages 279 – 252, 2017.

[2] Celery. Celery: Distributed task queue, 2015. `http://celery.readthedocs.org/en/latest/`, acessado em Outubro de 2015.

[3] S. G. Chang, B. Yu, and M. Vetterli. Adaptive wavelet thresholding for image denoising and compression. *IEEE transactions on image processing*, 9(9):1532–1546, 2000.

[4] R. R. Coifman and D. L. Donoho. Translation-invariant de-noising. In *Wavelets and statistics*, pages 125–150. Springer, 1995.

[5] D. L. Donoho and J. M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *biometrika*, 81(3):425–455, 1994.

[6] J. P. Huchra and M. J. Geller. Groups of galaxies I. nearby groups. *The astrophysical Journal*, 257:423–437, 1982.

[7] O. M. Madalosso, A. S. Charão, H. F. de Campos Velho, and R. S. da Rocha Ruiz. A web portal framework for remote execution of high performance applications in astronomy. In *Proceedings of the 4th Conference of Computational Interdisciplinary Science*. Pan-American Association of Computational Interdisciplinary Sciences, 2016a.

[8] R. Molina, J. Núñez de Murga, F. J. Cortijo, and J. Mateos. Image restoration in astronomy: a bayesian perspective. *IEEE Signal Processing Magazine, 2001, vol. 18, núm. 2, p. 11-29.*, 2001.

[9] R. S. d. R. Ruiz. *Turbulência em cosmologia: análise de dados simulados e observacionais usando computação de alto desempenho*. PhD thesis, Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2011-05-26 2011.

[10] R. S. R. Ruiz, H. F. Campos Velho, A. Caretta, C., S. Charão A., and P. Souto R. Grid Environment for Turbulent Dynamics in Cosmology. *Journal of Computacional Interdisciplinary Sciences*, 2:87, 2011.

[11] S. Sanfilippo. Redis, Novembro 2015. `http://redis.io/`, acessado em Novembro de 2015.

[12] A. Solórzano, A. S. Charão, R. S. da Rocha Ruiz, and H. F. de Campos Velho. Exploração de computação híbrida com openacc em um algoritmo friends-of-friends para classificação de objetos astronômicos. In *Anais da XVIII Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul*, pages 133 – 136, 2018.

[13] S. Van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.