A Web Portal Framework for Remote Execution of High Performance Applications in Astronomy

Otavio Migliavacca Madalosso[a], Andrea Schwertner Charão[a], Haroldo Fraga de Campos Velho[b] 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

In recent years, the volume of astronomical data is increasing due to advances in observational astronomy and simulations. To cope with this growth, there is a need for high performance computing resources and efficient tools for analysis and exploitation of the data. Moreover, while researchers develop new algorithms and applications in astronomy, there is also a need for efficient ways to make them available to the scientific community. Thus, it is necessary to create the means to spread information to a wide audience quickly and efficiently. In this scenario, Web portals can provide a simple interface for different users to have access to new applications running on a high performance computing infrastructure, with no need to perform advanced installations and settings. In this work, we present a Web portal framework for remote execution of high performance applications in astronomy. In this framework, a Web server deals with user interactions and dispatches tasks to an execution server. The framework allows for an administrator to manage applications that will be available to users, and also deals with the registration of users interested on running these applications. We developed this framework using the Python programming language along with Django Web development framework and Celery distributed task queue. The Web portal framework has passed tests using a parallel Friends-of-Friends application for classification of astronomical objects. For application execution, the portal deals with operations such as registration and activation of user accounts, dispatching requests for the application running and obtaining input and output files.

Keywords: astronomy, remote execution, web portal, high performance computing.

1 Introduction

Algorithms with high computational cost are easily found in areas such as geosciences (meteorology, oceanography, geophysics), engineering, biology and astronomy. These algorithms have the characteristic of requiring a high level of processing. Consequently, the time required for processing tend to be long and vary depending on the computational platform where they are executed.

Often, new versions of the algorithms used by the community are not made available for community use, not for license issues, but simply due to the absence of a practical method to make it available to the public.

This scenario motivates the developing of a web portal framework enabling users to run high performance algorithms provided by researchers, on a remote computational infrastructure. The users are also able to upload their own data to be processed by the algorithms and get the results back when the jobs are completed. The framework also enables and administrator or researcher to manage the available algorithms and associated configurations and permissions.

Such kind of web portals already exist [8], but they are usually developed as an ad-hoc facility for users of a given research project. Other web portal solutions are too generic and full of features, so they may be difficult for research teams to deploy. Our web portal framework aims to be a generic yet simple solution.

An application of such web portal framework, for astronomy researchers, is a *Friends-of-Friends* [5] algorithm implementation with $N*\log(N)$ computational complexity. This implementation was developed in a research project from the National Institute of Space Research (INPE), funded by the National Institute of Science and Technology in Astronomy (INCT–Astronomia[1]). For a dense distribution of astronomical objects within a given volume, the *Friends-of-Friends* algorithm degrades its complexity to $N*N$ [4]. These characteristics enforce our motivation to provide access to such implementation through a web portal.

[1] `http://www.astro.iag.usp.br/~incta/`

2 Background and Related Work

2.1 Web Technologies

Currently, there are many alternatives for developing Web applications. Languages as Java, Python and Javascript are mainstream, but there is also Web development frameworks that aim to accelerate productivity as well as enforce programming best practices. For this work, we chose Django Web development framework.

Django is a framework for developing Web applications in Python. It encourages agile development with high level constructs and pragmatic design. Since 2005, Django is an open source project with BSD license.

One of its main strengths is to facilitate creating dynamic Web applications which require a database in the back-end. It provides some ready-to-use Web components, for example Web forms for managing database tables. It is also extensible and based on a Model-View-Template pattern for organizing code development.

2.2 E-science Web Portals

There are some projects sharing some characteristics with our work. E-science portals focus on users from the scientific community that require high computing power with access to specific database. Some examples of *e-Science* web portals are:

- National e-Science Centre - NeSC [7] - NeSC was a pioneering e-Science infrastructure in United Kingdom (UK), maintained by the University of Edinburgh and the University of Glasgow. From 2001 to 2011, it sustained development of e-Science in UK, focusing on grid computing applications in diverse research areas as bioinformatics, astronomy and medical sciences.

- New Zealand e-Science - NeSI [8] - NeSI is a research infrastructure service from New Zealand. Access to this service is tied to existing projects from New Zealand. New projects and applications may be registered by filling a request form.

3 Portal Framework Design

In order to achieve our goals, we designed our Web portal framework to fulfill some requirements. First of all, our design should be as generic as

possible, without becoming large and difficult to use.

The applications should run on high performance servers, but the Web interface should be isolated from time-consuming executions which could affect the responsiveness to user interactions. To meet this requirement, our software architecture comprises two types of software servers: a **front-end server**, which runs a Web server, and one or more execution servers, called **workers**. In the front-end server, we use Django to implement user interaction and orchestrate the interaction with the workers, which in turn use the Celery to manage the execution of tasks. In this process, there may be transfers of input and/or output files, as required by the applications running on the execution servers. This execution flow can be depicted in Figure 1.

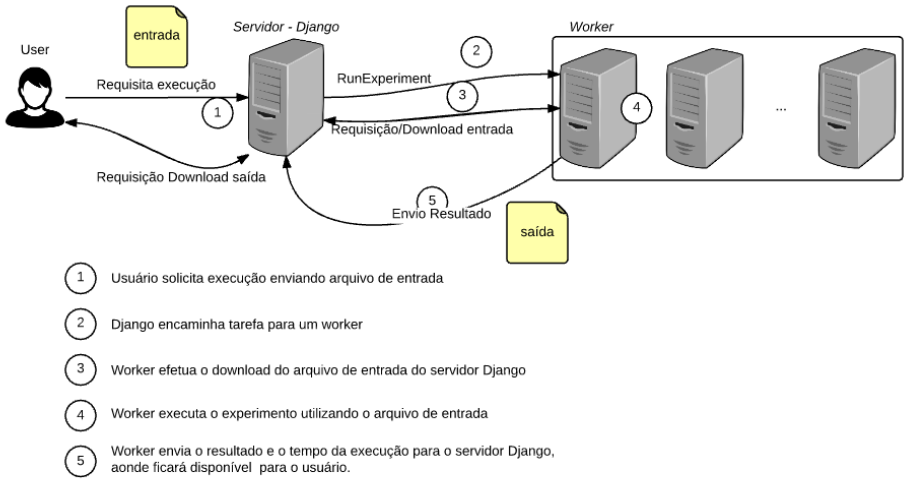

Figure 1: Execution flow.

Another requirement is access control, so different user profiles could have distinct permissions. To achieve this, we divided the system's features into 3 distinct groups according to the user profile: Anonymous, Registered and Administrator.

An Anonymous user has permission to access information about the system and to contact the system administrator. This user may also apply for registration and, if granted, log in as a registered user.

A Registered user is allowed to select applications and start high performance execution jobs. To do so, the user uploads a file that will be used as input to the selected algorithm. Such user is also allowed to monitor the

status of her/his jobs and download the output files of each experiment. A Registered user has also permission to cancel his/her jobs.

The Administrator has the same permissions than a registered user and has privileged access to the Django admin panel. This allows the Administrator to register new applications in the system and edit any information that the system stores in its database.

4 Implementation

This section discusses some issues and solutions adopted in the implementation of our Web portal framework.

4.1 Remote job execution

Implementing remote job execution required us to analyze different tools and techniques. It is not feasible that the same process that deals with all requests made by users need also to handle the jobs themselves, as this would cause a very slow progress in the system.

To work around this problem, were found two techniques: create of a new process that would manage the remote job execution, or use an external application to manage job queues and distribute them to other processes and/or machines (workers). In order to avoid creating a new system to manage job executions, we decided to search for applications compatible with the technologies used in the project that could meet our requirements.

We chose an application called Celery [2], which generates and manages task execution queues by exchanging messages. The machine that keeps the portal also maintains a process for implementing the broker Redis[9], which coordinates sending and receiving messages between the process that creates new jobs and the workers available to receive tasks.

This implementation allows the portal server to create tasks that will be performed by the workers. The workers are independent processes that should be started on server machines that will run the high-performance applications and exchange messages with the process that requested the execution.

4.2 Monitoring executions

Each job execution request generates an entry on a table which registers all requests of a given user. Such table is presented in a section of the Web portal (Figure 2). This table present an interface for users to monitor

and manage its execution requests (select, view, remove). The interface also allows the user to upload input data to the server and download output data after finishing executions. 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: Monitoring job executions

4.3 File System Schema

Jobs started through the Web portal usually require input data. They also generate output data which have to be stored for the user to download. For a given application registered in the Web portal, there may be multiple experiments which requires different input and output files. To cope with file management, we create a file system schema presented in Figure 3. In this schema, there is a root folder which will store multiple sub-folders for each registered user. Users may request execution of multiple experiments with varying applications and input data, so each experiment generates a sub-folder for a given user. The leaf folders store all files for a given experiment.

5 Results

Our Web portal framework is able to fulfill the requirements we presented in Section 3. The framework allows for researchers and developers to easily showcase their applications and make them available to collaborators, as well as share a computing infrastructure under strict permissions.

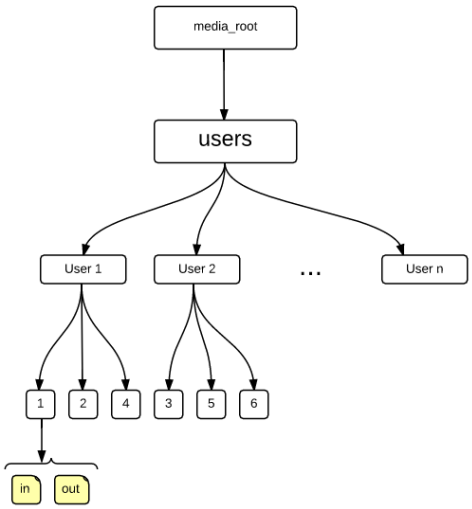

Figure 3: File system schema

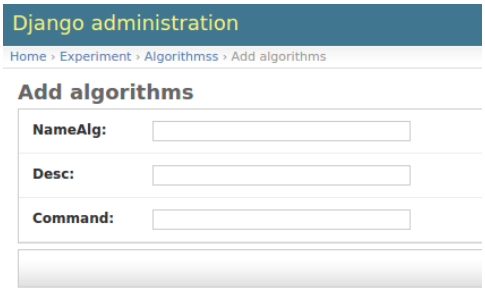

Figure 4: Registering a new application

To register a new application, the Web portal administrator have to fill the form illustrated in Figure 4. This form resembles a standard form in Django, which we have customized to meet our requirements.

As a test case of our framework, we built a Web portal for showcasing a parallel Friends-of-Friends application for classification of astronomical objects. We describe this test in the next paragraphs.

5.1 Friends-of-Friends Algorithm

The Friends-of-Friends algorithm (FoF) [5] is used to manipulate and analyze large amounts of data produced by simulations or observations in astronomy, for example analysis of distribution of dark matter in large scale,

the formation of halos of dark matter, formation and evolution of galaxies and clusters. These simulations or observations have a key role in the study of these subjects [1, 3].

In previous works, we have presented multiple approaches for implementing the FoF algorithm [6]. Depending on implementation strategies, the computational complexity can be reduced from $O(N^2)$ to $O(N \log N)$. Also, parallel programming can be used to reduce processing times.

The algorithm works using a data entry consisting of positions of N celestial bodies that must be grouped together if there is gravitational interaction between them. The interaction will only occur if the objects are at a distance within a given radius. When two bodies are positioned at a distance less than the radius informed, they belong to the same group. Any other body that is at a distance less than or equal to the defined distance, also belong to the group. The expected result of the algorithm is to identify groups of objects that interact.

This algorithm may require a large volume of data that make up the input file. During the development of the algorithm, we used an input data file comprising 317,000 bodies. As this file needs to be sent from the user to the system, it must be given a file size limit and a validity period for which this file is still available in the system after use.

5.2 Web Portal for Friends-of-Friends Remote Execution

Our Web portal installation for FoF remote execution uses two machines at the Laboratory of Computer Systems of Universidade Federal de Santa Maria. One of them runs the Django system and the other runs a single worker which execute job requests dispatched by the user.

We performed tests consisting of multiple remote executions of the FoF algorithm, using different input data sets. All executions were dispatched through the Web portal. We then compare the results with those obtained from local runs of the FoF algorithm. All output data files resulted identical, as expected.

We also tested the registration of a new user, using an e-mail address to validate the account. After registering and confirming the e-mail, the user have access to a restricted area on the Web portal. He or she may then visualize execution requests or submit a new remote execution request for the FoF algorithm. Also, he or she may visualize a table presenting information on all of his/her requests, as well as its associated input and output data files (Figure 5).

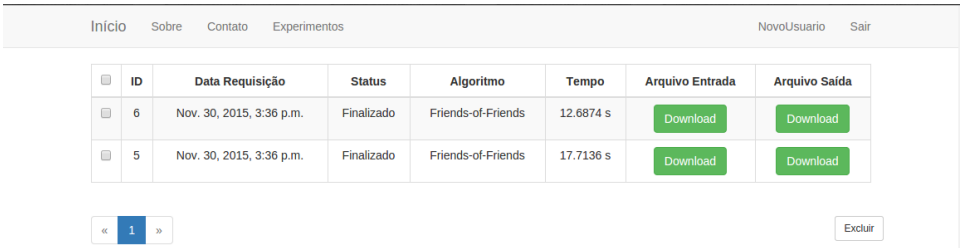

Figure 5: FoF remote executions

6 Conclusions

This work presented a Web portal framework aimed to support remote execution of high-performance applications and, in particular, the Friends-of-Friends algorithm developed in a previous work.

Our solution is built upon Django Web development framework and uses some extensions for remote job execution. Using distinct servers for user interaction and high-performance processing, the Web portal is free of contention for computing resources. Using access control, different user profiles may have distinct permissions on system features. The framework enables administrators to register new high-performance applications and registered users to manage their remote execution jobs for previously registered applications.

To facilitate reuse in other cases, we created a repository with the source code of the project, which can be accessed in `https://github.com/Madalosso/TG`. The remote execution servers will migrate to the heterogeneous computing cluster at LAC-INPE, where the Friends-of-Friends classifier will be available for the astronomy community.

Acknowledgments. The authors gratefully acknowledge financial support from the National Institute of Science and Technology for Astrophysics (INCT-A), who granted the Scientific Initiation scholarship to the first author, and the CNPq, Brazilian agency for research support.

References

[1] E. Bertschinger. Simulations of structure formation in the universe. *Annu. Rev. Astron. Astrophys 36*, 1998.

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

[3] G. Efstathiou, M. Davis, S. D. M. White, and C. S. Frenk. Numerical techniques for large cosmological n-body simulations. *Astrophysical Journal Supplement Series (ISSN 0067-0049), vol. 57*, 1985.

[4] B. Howe and M. Balazinska. *New requirements for Scalable Data Processing, Chapter 8 (in: "Data-Intensive Computing: Architectures, Algorithms, and Applications". Editors: Ian Gorton and Deborah K. Gracio).* Cambridge University Press, 2012.

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

[6] O. M. Madalosso, A. S. Charo, and H. F. de Campos Velho. Implementao do algoritmo friends of friends de complexidade n*logn(n) para classificao de objetos astronmicos. In *Anais da XV Escola Regional de Alto Desempenho do RS*, 2015.

[7] NeSC. National e-science centre, Novembro 2015. `http://www.nesc.ac.uk/`, acessado em Novembro de 2015.

[8] NeSI. New zealand escience infrastructure, Novembro 2015. `https://www.nesi.org.nz/`, acessado em Novembro de 2015.

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