JOVIAL: Notebook-based Astronomical Data Analysis in the Cloud

Mauricio Araya[a,*], Maximiliano Osorio[a], Matías Díaz[a], Carlos Ponce[a], Martín Villanueva[a], Camilo Valenzuela[a], Mauricio Solar[a]

[a] *Universidad Técnica Federico Santa María, Avenida España 1680, Valparaíso, Chile*

Abstract

Performing astronomical data analysis using only personal computers is becoming impractical for the very large data sets produced nowadays. As analysis is not a task that can be automatized to its full extent, the idea of moving processing where the data is located means also moving the whole scientific process towards the archives and data centers. Using Jupyter Notebooks as a remote service is a recent trend in data analysis that aims to deal with this problem, but harnessing the infrastructure to serve the astronomer without increasing the complexity of the service is a challenge. In this paper we present the architecture and features of JOVIAL, a Cloud service where astronomers can safely use Jupyter notebooks over a personal space designed for high-performance processing under the high-availability principle. We show that features existing only in specific packages can be adapted to run in the notebooks, and that algorithms can be adapted to run across the data center without necessarily redesigning them.

Keywords: Jupyter Notebooks, Cloud Computing, Docker, Kubernetes, Dask, High-Performance Computing

1. Introduction

The fast-paced technology improvements on astronomical telescopes, instruments and archives, plus the accumulative nature of astronomical data, contribute to generate an overwhelming data space waiting to be analyzed and exploited. The data deluge problem in astronomy is not a menacing challenge in the horizon, but a very real challenge for current astronomical data analysis. This is the case for the ALMA Data, where some single data products can reach hundreds of GigaBytes, being expensive to transport, store, process, and load them into memory (Testi et al., 2010). The next generation of projects such as Large Synoptic Survey Telescope - LSST (Ivezic et al., 2008), Extreme Large Telescope - ELT (Gilmozzi and Spyromilio, 2007), and the Square Kilometer Array - SKA (Dewdney et al., 2009) will increase the data generation rate from two to three orders of magnitude, so scientists and engineers are preparing themselves to cope with this new reality.

The somewhat mundane problem of personal computers not being able to store and process the astronomical files produced nowadays, have been forcing a paradigm-shift in astronomical data analysis. It is not enough to develop new algorithms and allocate computing infrastructure; modern software and services need to be fostered to equip astronomers with the tools to deal with the data deluge problem.

It is clear that processing tasks need to be moved towards the large archives where the data resides, minimizing the data transfer to the end user and taking advantage of the high-performance computing infrastructure available at the data centers (Djorgovski et al., 2003; Moolekamp and Mamajek, 2015). However, the efficient use of the high-performance computing and storage infrastructure requires a good understanding of technical details that are not in the domain of expertise of astronomers. Moreover, new challenges arise when these services are deployed under the high-availability principle in order to compete with the convenience of local processing.

This paper presents our approach to provide a Cloud service for astronomical data analysis called JOVIAL, which provides Jupyter notebooks to astronomers in a transparent and high-available fashion. Section 2 discuss why Jupyter notebooks are a suitable tool for astronomical data analysis, while Section 3 presents the architecture that we used for offering them in the Cloud. Then, Sections 4 and 5 presents our current efforts to provide a consistent set of libraries for astronomical data analysis and how the computing tasks can be distributed along the available infrastructure. We conclude in Section 6 with our next steps towards improving the services provided by JOVIAL.

2. Notebook-based Astronomical Data Analysis

Modern data analysis strongly relies on computational methods, models and algorithms. These are usually grouped in software packages that provide from very general to very specific functionality that support the analysis process. Despite the large variety of them, one can identify the key elements that these set of tools must cover.

Interactivity. Most of the visualizations and plots that are made during the analysis process are not published nor shared, and only the key results are published. Any discovery process include some trial and error actions, fine tuning and data exploration tasks. Interactive tools help to speedup this discovery process, and that is the main reason why astronomers usually use specialized interactive desktop applications.

*Corresponding author e-mail: mauricio.araya@usm.cl

Documentation. All the findings of an analysis process need to be documented somewhere: the parameters used in the applications, the data transformations, the formulas and insightful comments need to be registered in some document. Usually, interactive desktop applications provide information in the form of action history and logs, but these are in a suitable format for being reported. The documentation process need to be carried out under a different tool such a document preparation software.

Automation. When a task needs to be executed repeatedly for a collection of data or for different parameters (i.e., batch processing), the automation is carried out using high-level programs called scripts. The documentation of these procedures are usually included in the same scripts as comments. Depending on the tools used for analysis, the execution of these scripts can be supported by the same analysis tools, or they need to be loaded into a different system. A typical example of the latter is when a time-consuming pipeline need to be executed in a computer cluster.

Visualization. Astronomy is a science that has always relied on visual representations and inspection as the first tools for discovering patterns in a natural way. Therefore, the analysis of astronomical data often relies on powerful visualization tools to help the astronomer to understand the underlying processes that generate certain data. These visualization tools becomes more sophisticated as the data increases in complexity, for example for inspecting high-resolution spectroscopic data cubes or for overlaying data from different instruments and telescopes of the same source.

2.1. Interactive Notebooks

Interactive notebooks are files containing both code and documentation that can be interactively edited, displayed and executed (Shen, 2014). The name comes from the analogy to paper notebooks, where ideas can be drafted in a dynamical way. Notebooks were first adopted by mathematical software packages like Maple, SageMath and Mathematica, and they have been used to assist researchers in the scientific workflow for all sciences since then (Kluyver et al., 2016). Most notably, major scientific breakthroughs like the first gravitational wave detection (Abbott et al., 2016), have been accompanied by notebooks to share the exact data analysis process that was followed.

The notebook concept consist then in combining *interactivity*, *documentation*, *automation* and *visualization* in a single file, and therefore, using a single comprehensive tool for data analysis. The interactivity allows to understand a problem by performing small, incremental improvements that lead to a complete analysis. The documentation allows sharing, from early to publication stages, a valuable computational narrative that it is difficult to present otherwise. The automation allows to generate self-documented, exportable and reproducible pipelines. Finally, inline visualization allows inspecting data and presenting results interactively and opportunely, meaning that the images and figures are presented alongside the documentation and the code.

2.2. Jupyter Notebooks

Jupyter is an open-source web application for authoring interactive notebooks that has gained an important popularity in recent years. For example, in January of 2018 the number of Jupyter notebook files in Github was over 1.6 millions(Parente, 2018). The Jupyter project is the evolution of the IPython interactive shell, which since 2001 has provided tools to extend Pythons interactive capabilities (Pérez and Granger, 2007). Currently, Jupyter support several *kernels* types, which are the interpreter processes that actually execute the inline code inside the notebooks. This feature allows Jupyter to support several programming languages like Python (IPython), R (IRkernel), Julia (IJulia), JVM based languages (BeakerX), C++ (cling) and many more using the same notebook format specification (Kluyver et al., 2016)[1]. Besides authoring notebooks, Jupyter web interface includes simple process and file managers, an spawning terminals system and extensions that allow to graphically manage packages and parallelism. The JupyterLab project (JupyterLab-Team, 2018) extends these in-browser capabilities to produce a window-manager-like interface, including text editors, documentation viewers, resource monitors, etc.

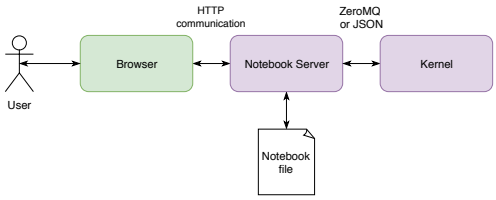

Figure 1: Jupyter Notebook Architecture.

The architecture of Jupyter notebooks shown in Figure 1, detaches the presentation logic from the business logic, being the first one handled by the browser through Javascript code, and the second one by a service written in Python. This allows Jupyter to run in a web-based client-server model. At the server, the architecture also detaches the processes in charge of interpreting the notebooks files from those in charge of interpreting the code snippets in the notebook (i.e., the kernel). This allows kernels to halt or be restarted without compromising the service itself, and also it ease the support of kernels outside Python. The communication between these processes is done with an efficient JSON messaging using the ZeroMQ library.

This architecture makes it possible to use Jupyter as a remote notebook authoring application that executes arbitrary code within the host server. This opens up several opportunities and challenges that we address in Sections 3, 4 and 5.

2.3. Notebooks and Astronomical Libraries

There are several software packages for performing astronomical data analysis; from simple file viewers to complete pipeline toolkits. Each package have its own scope, semantics and installation dependencies. Simple operations may overlap

[1] `https://github.com/jupyter/jupyter/wiki/Jupyter-kernels`

between packages, but there are others that are provided only by one specific package. Currently, an astronomer must install all the software tools that he might need, and make all the data and metadata conversions needed to inter-operate between these tools. Some efforts of standardization and inter-application communication have been developed by the virtual observatory community, yet not all packages support protocols like SAMP nor publish data as VoTables (Taylor et al., 2012).

While Jupyter supports languages that traditionally have been used for astronomical data analysis like R, Matlab and IDL, most of the available notebooks for astronomy use the IPython kernel. This is for two major reasons: before all these languages were supported by Jupyter, IPython notebooks were already very popular (Shen, 2014), and Python itself is a major trend in astronomy (Greenfield, 2011).

There are many Python libraries focused in generic scientific data analysis, such as `NumPy`, `SciPy`, `Pandas`, or `Matplotlib`. These are actually *notebook friendly*, which means that their outputs are compatible or have an special extension to be better displayed into notebooks. There are also several astronomical data analysis tools written in Python, but not all of them are designed for execution within interactive notebooks. The `astropy` project (Robitaille et al., 2013), and the important number of affiliated packages built over it, have used notebook-friendly packages and developed the necessary features for notebook compatibility. On the other hand, projects like CASA (McMullin et al., 2007), which actually uses the IPython console, does not offer a compatible interface to notebooks. This is because its graphic user interfaces are designed as stand-alone applications in QT.

2.4. Jupyter as a Cloud Service

Using Jupyter as a remote service is a recent trend within astronomical data centers. Several institutions are deploying their own Jupyter-based architecture to support astronomers in their work. For instance, SciServer Compute (Medvedev et al., 2016) uses Jupyter notebooks attached to large relational databases and storage space to bring advanced data analysis capabilities closer to the data. Another example is the NOAO Data Lab (Fitzpatrick et al., 2014), which provides a collaborative computing environment where users can easily access and visualize large datasets, and conduct experiments on subsets of the data using Jupyter notebooks.

There are several projects in astronomy that are moving to cloud services, for example common virtual observatory storage spaces for astronomical data (Bertocco et al., 2018), or running pipelines for the massive data that SKA will produce (Sabater et al., 2017). While the two aforementioned services do not rely on notebooks, they share similar challenges to the ones addressed in this article. Moreover, they explore other problems that this work does not discuss, such as supporting service interoperability (i.e., IVOA-compliance) and using cloud infrastructure (i.e., Amazon Web Services) respectively.

Offering notebooks as a web-based service is an important step towards moving interactive analysis from the user' computers to where the data resides (i.e., data centers). However, several gaps must be filled in order to produce a quality service that astronomers can trust and rely on. A Cloud service is not only a web-application, but a service that hides the complexities of the deployment, maintenance and high-availability configurations of dynamic resources over one or several data-centers.

From the user point of view, notebooks running in the Cloud offer several benefits such as world-wide access, non-blocking computations, installation-free experience, augmented storage and computing resources, task distribution, and collaboration opportunities. However, each one of those imposes new challenges, such as difficulties to work off-line, synchronization problems, custom installation of libraries, personal backups, managing distributed resources, and privacy issues.

In the following section, we describe our approach to deal with some of these challenges.

3. JOVIAL: Notebooks in the Cloud

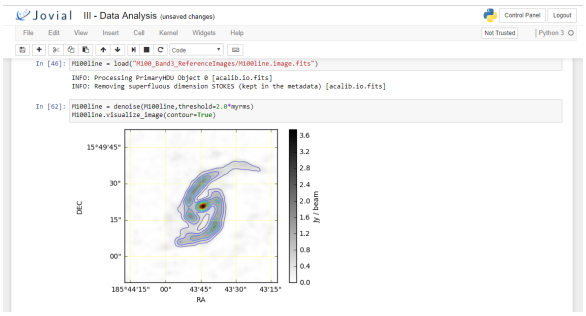

Figure 2: Jupyter Notebook Running on JOVIAL.

JOVIAL[2] is a Cloud service developed by the Chilean Virtual Observatory (Solar et al., 2015), where astronomers can author their own Jupyter notebooks over a personal space designed for high-performance processing (Araya et al., 2018b). The main goals of JOVIAL is to bring data analysis closer to the data, take advantage of high-performance infrastructure, and protect the users' data and results. An screenshot of JOVIAL running is shown in Figure 2.

The architecture of JOVIAL (Díaz et al., 2018) is composed by several software, systems and hardware components that we have incrementally included to provide a better service. In this section we describe and explain how we use these components, in order to be able to replicate our configurations. In the last part of this section we present the complete architecture.

3.1. Multi-User Jupyter Service

The original Jupyter notebook server allows only one user per instance, meaning that each interface need to be bound to a different port. JupyterHub is a multiple-user version of Jupyter, where each user owns its own server but through the same instance. The process of starting a Jupyter notebook server for

[2] Jupyter OVerride for Astronomical Libraries.

a user is known as *"spawn"*, and where it spawns is determined by JupyterHub modules called spawners (JupyterHub-Team, 2018).

The concept that each user spawns a server might seem unnecessary at first glance, but remember that notebooks are just one part of the Jupyter interface. Users need to manage their files, install custom packages, manage the running kernels, etc. All these operations must run with an user ID for accountability and access control.

3.2. Docker Containers

The main problem with JupyterHub is that by default the user of the account is a generic system user, and the notebook spawns in the host machine. This combined with the feature of running arbitrary code in the notebooks represents a risk for the data center and for the other users.

To solve this problem we decided to spawn the services in isolated environments, where users are restricted to use only the resources that are assigned to them. Also, this isolation provides a security layer, since users are not even aware of the existence of other users. Isolation of the notebooks can be achieved through virtualization, for which we found two approaches: the traditional hypervisor-based virtualization and the relatively new container-based virtualization. The first one provides virtualization at the hardware level creating and running virtual machines. Vmware (Walters, 1999), Xen (Barham et al., 2003), and KVM (Kivity et al., 2007) are examples of Hypervisor-based virtualization. The second one is a lightweight virtualization approach that uses the host kernel to run multiple virtual environments. This provides isolated environments with only the necessary resources to run the applications (Bui, 2015).

We have chosen container-based virtualization because studies show that containers have a performance close to a native application (Xavier et al., 2014) and they are fairly secure (Bui, 2015). We use the Docker system (Merkel, 2014), which is an open-source container technology with the ability to build, ship, and run distributed applications. In our context, containers recreate a system environment inside the host, where we can spawn a notebook server safely, including the personal user files through Docker volumes. Please note that the user does not have administrator access neither at the host nor at the container, but it can install packages and perform configurations for its own account. Similar projects to JOVIAL, like SciServer Compute (Medvedev et al., 2016), are also using Docker containers for the same purpose.

To control the resources used by the users, we use a Linux kernel feature called control groups (cgroups). A cgroup limits an application to an specific set of resources. Cgroups allows Docker to enforce limits and constraints, so it is possible to limit the memory, CPU, disk, and others to an specific container and user (Higgins et al., 2015).

3.3. Orchestration Over Multiple Hosts

The notebook servers can run on a single host or multiple hosts. By keeping them in a single host, we are taking the risk of centralizing the infrastructure, so the failure of this single host will compromise the whole platform. Also, the multiple-hosts approach allows us also to use better our data-center resources. But containers deployed within multiple hosts needs to be orchestrated in accordance with the high-availability principle, meaning that they should be robust and self-healed under standard hardware or network malfunctioning.

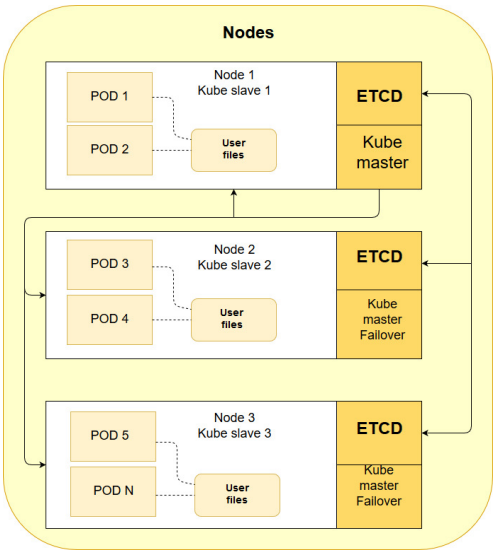

Figure 3: Kubernetes Deploy, each host has a database named etcd. One node is the master if one node fails the others keep Kubernetes working, if the master fails, the other ones vote for a new master and keep working.

In order to achieve this, we combined our Docker containers with Kubernetes. Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications called POD (Burns et al., 2016). Kubernetes decides where to run the POD based on the current load of the nodes and permissions, providing on-demand growth and load balancing between the hosts. For networking, Kubernetes uses a service called Flanneld that gives each container an IP address in a logical private network (different from the one of the nodes) that connects the PODs across the nodes (Marmol et al., 2015).

If one node fails, Kubernetes will migrate all the PODs to other available nodes (Bernstein, 2014). So Kubernetes is not only a Load Balancer but it ensures high availability of the infrastructure. According to (Verma et al., 2015), Kubernetes supports high-availability applications with run-time features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures.

For offering both a secure and a high-available service, not only the Jupyter servers are spawned in PODs managed by Kubernetes, but also the JupyterHub server itself. We do this by spawning multiple hubs in the same Kubernetes cluster, with support for namespaces. You can control the number of resources that each namespace can use, effectively limiting the resources of a single JupyterHub (and its users). We also control several security parameters (such as userid/groupid, SELinux, etc.) via flexible POD security policies.

Additionally, the database of Kubernetes (ETCD) is dis-

tributed and replicated through the nodes so that the infrastructure becomes resistant to the failure of multiple nodes as shown in Figure 3.

In summary, we used the standard features of Kubernetes to use Docker within JupyterHub, so no new modules were needed support this setup. This was the main reason to use Docker rather than alternative solutions such as Shifter or Singularity.

3.4. Distributed Filesystem

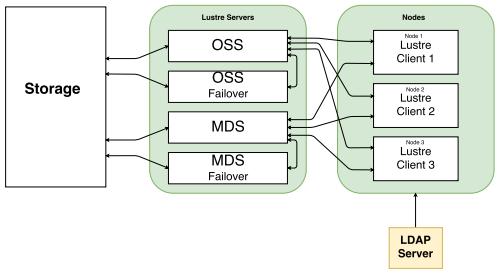

Figure 4: Lustre infrastructure. The MGS is not shown since only manages the hosts. The failovers keeps the infrastructure working if the master server fails.

Another problem that arises from using multiple hosts is to provide distributed storage services. Moreover, current astronomical data analysis aims to process very large files that need to be accessed fast and often. Therefore, we need a filesystem that work through a high-speed network, with low latency and that can manage large file sizes.

Lustre is a distributed filesystem optimized to operate with low latency at a TeraByte/PetaByte scale (Braam and Zahir, 2002). The main drawback of Lustre is that additional infrastructure is needed for its correct operation: not only an InfiniBand network is recommended, but also dedicated servers are required for attaining good performance. The Lustre infrastructure consist in three kind of servers: management servers (MGS), metadata servers (MDS) and object storage servers (OSS). When a client asks for a storage block, the petition is managed by the MGS, which looks up in the MDS for the block location and the user information (i.e., identification, privileges, etc). Finally the MGS establish a connection between the client and the OSS so the block can be mounted directly from the OSS.

In our setup at the Chilean Virtual Observatory, we use 5 servers for offering a high-available Lustre installation (i.e., 2 OSS, 2 MDS, and 1 MGS) as shown in Figure 4. While OSSs and MDSs are originally responsible for the half of the Lustre storage space, the other of-a-kind server is a failover for that same storage space. If the MGS fails, the blocks are still exported to clients with active mounts, but no new mounts can be made. A dedicated InfiniBand network is used to connect Lustre severs and clients, and our two disk controllers are connected through optical fiber to both OSS servers.

For JOVIAL we use Lustre to mount the users' files on the same nodes where Docker containers are hosted. Then, the mounted files can be passed to the containers as Docker volumes, so the whole Lustre architecture is completely hidden from the user. This simple solution is aligned with the cloud computing paradigm of hiding deployment details from the end user. However, it is worthy to mention that there is a potential performance degradation cost due to the additional layer of Docker volumes. The proper empirical evaluation of this potential problem escapes the scope of this paper, but no significant degradation was observed so far.

3.5. JOVIAL Architecture

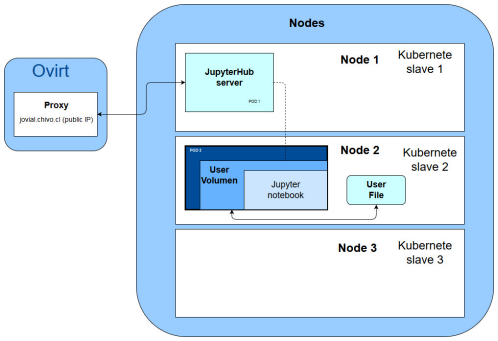

Figure 5: JupyterHub-Kubernetes system. The JupyterHub server and the Jupyter Notebooks Servers run in Docker containers orchestrated by Kubernetes. User accounts are mounted through Lustre at the nodes and then are passed to the containers using Docker volumes. The KubeSpawner is a module responsible for running the actual Jupyter Notebook server in an isolated environment. All the nodes are in a private network and the JupyterHub servers can be accessed from the internet through a proxy.

We use 4 computing nodes (servers) that work as hosts of the JupyterHub-Kubernetes system as shown in Figure 5, which are also Lustre clients since they need to mount the users' files. Users are managed globally using LDAP, a software protocol that makes entries in which management data of the user is stored. The Jupyter spawner that allows us to spawn notebook servers in Kubernetes clusters is called KubeSpawner[3]. This module uses the global LDAP `id` and `username` to recreate the same user credentials in the container, mounting the user home directory as a Docker volume. The recreated user owns the files mounted in the container since it has the same `id` and `username` as the original one. Finally, KubeSpawner runs the Jupyter Notebook server in an isolated but functional environment. We keep all the servers in a private network and the public access to the JupyterHub server is given by a proxy running on a virtual machine over oVirt.

When a user enters the portal through the proxy and logs in into his account, JupyterHub checks if the container with the users' notebook server exists. If not, it asks Kubernetes to create it, spawn the files of the user into the container and contact the server. An overview of the components that interact in this process is shown in Figure 6.

[3] `https://github.com/jupyterhub/kubespawner`

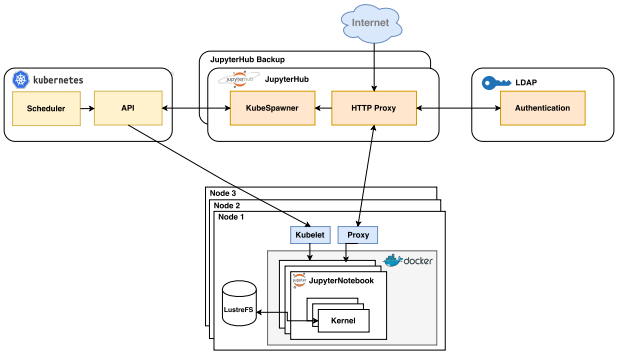

Figure 6: JupyterHub is the responsible for the authentication and authorization using LDAP. Also, JupyterHub requests the creation of the notebook to Kubernetes. Kubernetes schedules a container on a node and the node starts the container using Docker. Lustre provide the user files to the container through a Docker volume. Moreover, there are multiple JupyterHub services running in case of a failure.

4. Extending Jupyter's Astronomical Toolkit

As presented in Section 2, some software packages are notebook friendly, but most of the astronomical data analysis software needs to be adapted for running into notebooks. Porting legacy software to make it work over new systems is convenient for many reasons: usually is easier than re-implementing, reduces the risk of producing inconsistent software, and most important, it gives the possibility to make this software interact with new technologies such as Jupyter.

The first barrier is binary and language compatibility. For example, the CUPID module of the Starlink suite (Berry et al., 2007), is written in C and uses data structures strongly rooted in Starlink libraries. As CUPID does not use all the libraries, we developed wrappers using Cython, that uses only a few of the Starlink dynamic libraries. In this way it is possible to offer Python bindings, allowing the use of CUPID algorithms directly from the notebooks (Villanueva et al., 2018). This package is called `pyCupid`.

A second barrier are versions and compatibility. For example, even though CASA has Python bindings, and uses libraries that currently are notebook friendly (e.g., MatplotLib), the versions shipped by CASA are not. We have followed the `patchelf` strategy of CASANOVA, re-linking the binaries from the (old) self-provided packages to the versions installed in JOVIAL. This allows compatibility with notebooks of both the CASA core and the CASA tasks (McMullin et al., 2007). This package is called `Caspyter`.

The third barrier is package flooding. While having a large number of software packages for astronomy data analysis gives more options and functionality, it is not easy to handle the API details of each package. For this issue, we are developing a package that provides a unified object-oriented API for common tasks. This is implemented through suitable wrappers for data input and output operations provided by notebook-friendly packages. For example, the method `visualize(data)` will do its best to display the data. If it is an spectra, then will use `matplotlib`; if it is an image `AplPy` will be used; if it is a cube

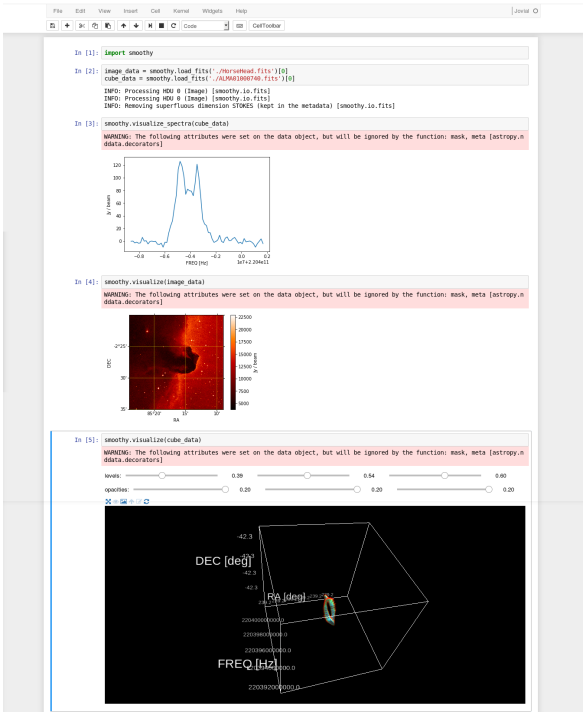

Figure 7: JOVIAL running `Smoothy`. The example shows an image (optical) and a spectral cube (millimeter) both displayed using the function `visualize`. The spectra of the cube is also shown.

`PyVolume` will be used for a volumetric visualization. This package is called `smoothy`, and Figure 7 shows and example of its execution.

5. Task Distribution: Proof of Concept

While Section 3 presents an architecture to scale in terms of users and notebooks, one of the main advantages of bringing code to data is task distribution across the data center infrastructure.

We developed a proof of concept of a distributed pipeline for notebooks that finds regions of interest using a fast algorithm called RoISE (Araya et al., 2016) that is implemented in the `ACAlib` python package (Araya et al., 2018c). The main objective of this pipeline is to show that a large number of data products can be processed despite the different resolutions, signal-to-noise ratios, densities, morphologies, imaging parameters, among others (Araya et al., 2018a).

As we need to process thousands of FITS data cubes, we evaluated tools to run hundreds of cpu-intensive and memory-intensive tasks in parallel. Between alternatives such as MPI, IParallel, Apache Spark or Hadoop MapReduce, we selected a Python library called Dask (Rocklin, 2015) mainly because required only a few modification to the original code.

Dask provides two main components: a set of dynamic task scheduling systems that is internally optimized for interactive and non-interactive computational workloads, and Big Data

collections like parallel arrays, dataframes and lists that extend common interfaces like `NumPy`, `Pandas`, or Python iterators to larger-than-memory or distributed environments. These parallel collections run on top of the dynamic task schedulers.

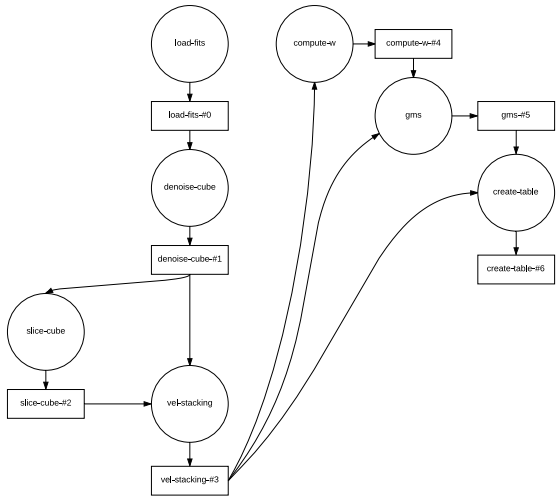

Figure 8: Generated computational graph for the RoISE algorithm

In particular, the Dask *delayed* interface is particularly useful for parallelizing custom code. It is useful whenever the problem does not directly translate to a high-level parallel object like Dask arrays or dataframes, but could still benefit from parallelism. It works by delaying the function evaluations and putting them into a Dask graph. We decided to use the *delayed* object since the use of high-level objects like arrays or dataframes would require redesigning the whole algorithms. The *delayed* object wraps the function calls into a lazy evaluated task graph so it can be later executed by one of the workers. Figure 8 shows the version of the RoISE algorithm using the Dask *delayed* object; this pipeline represents all the operations that need to be executed to analyze one FITS cube. To process many of them, we used the Dask distributed scheduler, which is backed by a single central scheduler process and many workers processes (i.e., python interpreters) spread across multiple machines.

We consider the dataset of all the FITS files produced by the ALMA pipeline, which are synthesized images and cubes from interferometric millimeter/sub-millimeter data. From these files we selected only the data cubes at the highest calibration level (i.e, primary-beam-corrected data with more that one spectral channel), which represents a total of 1.87 Terabytes stored in 4474 files. As users of JOVIAL, we have direct (read-only) access to the Chilean Virtual Observatory archive which is mounted within the computing nodes. As we needed to process hundreds of them in parallel, we moved the FITS files to Lustre storage in batches, allowing low latency and concurrent access. We were able to run our pipeline with four Dask workers, each of them running on a different server of 24 cores and 126 gigabytes of RAM, using a total of 96 cores and 504 gigabytes of RAM, processing a total of 2666 ALMA data products in a few hours. Please note that these workers are launched and managed by a notebook running in a Docker container through a Dask scheduler, but the workers and the actual tasks are running natively in four of the computing nodes of the ChiVO data center.

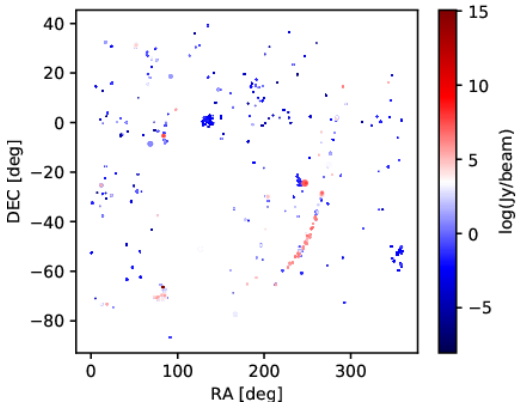

Figure 9: Graphical summary of the intensities of the RoIs in sky coordinates. The size of each point also represents the area covered by the region.

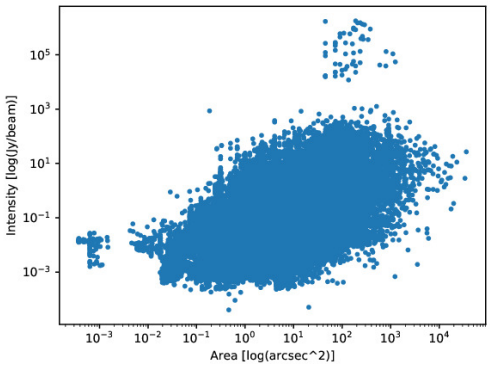

Figure 10: Area vs Intensity in logarithmic scale for all RoIs. Three clusters are observed, where the ones with large area show low intensity and those with high intensity have a relatively small area.

We found 47765 RoIs in all these files, for which we computed their centroid, semi-major/semi-minor axes, area, eccentricity, solidity, intensity and frequency range. As an example, we show in Figures 9 and 10 the summary of the results for intensity and area. For more details on the obtained results please refer to Araya et al. (2018a).

This proof of concept shows that high-performance capabilities can be exploited from Jupyter notebooks running in the Cloud with only minor configurations and at a user level. However, for a full Cloud user experience the data transfer from the archive to the Lustre storage, and the specific configurations of the Dask scheduler and hosts, should be hidden or at least anonymized from the user perspective.

6. Conclusions

We have introduced JOVIAL, a multi-user data analysis service for astronomers in the Cloud that offers safe and secure Jupyter notebooks. The service is deployed under the high-availability and user-isolation principles, while maintaining all the flexibility and customization opportunities of Jupyter. We integrated several technologies such as JupyterHub, Docker, Kubernetes and Lustre to build the service, and the main configurations are available at its development web page (ChiVO-Team, 2018).

We have equipped JOVIAL with popular Python libraries for astronomical data analysis, and developed a few others to fill functionality gaps with respect to popular desktop-based software. We showed that legacy or outdated astronomical data analysis software can be adapted to run in notebooks, and that the complexity of having too many libraries can be mitigated by integrated APIs. In the case of irreconcilable versions or incompatible binaries, a client-server architecture (e.g., API REST) could be a last resource strategy. Also, there are specific applications that are very challenging to make them notebook friendly, such as those that use their own display engine or a very old one. In such cases, even the wrapping strategy might fail, and a more updated application with similar features should be considered.

At last, we presented our proof of concept for distributing tasks across the data center, showing that modular code can be easily adapted to run tasks in parallel, for example, by using the delayed API of Dask.

The main argument to build this service was to move the analysis near the data. The benefits of this are shown in Section 5, where a large dataset of FITS were analyzed from a robust Jupyter notebooks service, without transferring them through internet and using the high-performance resources of the data center.

6.1. Future Work

In the short term, we have several undergoing projects to improve the user experience, the completeness and the efficiency of JOVIAL:

- We are deploying an automatic user creation system for authorized domains, so astronomers can register to JOVIAL using their institutional e-mail.
- We are developing a bi-directional SAMP interface for Jupyter notebooks, so results can be transparently sent to (or received from) local desktop applications.
- Even though we have successfully integrated CASA to Jupyter notebooks, some GUIs are not suitable for the notebook concept. We are exploring IPython widgets to create panels that can replace this functionality.
- The only algorithm that was parallelized so far is RoISE, but there are several other algorithms developed by the Chilean Virtual Observatory that can benefit of Dask's delayed API. We plan to include parallel versions of them in the `ACALib` package, and include popular algorithms from other packages to be used within JOVIAL.
- At last, we plan to continue wrapping libraries into `Smoothy` in order to produce simple yet powerful notebooks under a homogeneous API.

In the long term, we need to explore the possibility of providing collaborative notebooks (or at least shared notebooks) to promote the use of JOVIAL between groups of researchers, similar to what Google Docs is doing. In a similar direction, we need also to provide a proper notebook publishing mechanism, in order to encourage reproducible research through notebooks, and increase the visibility of the service.

7. Acknowledgements

This work has been partially funded by FONDEF IT 15I10041, CONICYT PIA/Basal FB0821, CONICYT PIA/Basal FB0008 and PIIC USM.

References

Abbott, B.P., Abbott, R., Abbott, T., Abernathy, M., Acernese, F., Ackley, K., Adams, C., Adams, T., Addesso, P., Adhikari, R., et al., 2016. Observation of gravitational waves from a binary black hole merger. Physical review letters 116, 061102.

Araya, M., Caceres, R., Gutierrez, L., Mendoza, M., Ponce, C., Valenzuela, C., 2018a. Towards large-scale RoI indexing for content-aware data discovery, in: XXVII Astronomical Data Analysis Software & Systems, ADASS. pp. 1–4. [To appear].

Araya, M., Candia, G., Gregorio, R., Mendoza, M., Solar, M., 2016. Indexing data cubes for content-based searches in radio astronomy. Astronomy and Computing 14, 23 – 34.

Araya, M., Hochfarber, T., Valenzuela, C., Farias, H., Solar, M., 2018b. JOVIAL: Jupyter override for astronomical libraries, in: XXVI Astronomical Data Analysis Software & Systems, ADASS. pp. 1–4. [To appear].

Araya, M., Solar, M., Mardones, D., Arevalo, L., Mendoza, M., Valenzuela, C., Hochfarber, T., Villanueva, M., Jara, M., Simonsen, A., 2018c. The chivo library: advanced computational methods for astronomy, in: XXV Astronomical Data Analysis Software & Systems, ADASS. pp. 1–4. [To appear].

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A., 2003. Xen and the art of virtualization, in: ACM SIGOPS operating systems review, ACM. pp. 164–177.

Bernstein, D., 2014. Containers and cloud: From lxc to docker to kubernetes. IEEE Cloud Computing 1, 81–84. doi:10.1109/MCC.2014.51.

Berry, D., Reinhold, K., Jenness, T., Economou, F., 2007. CUPID: a clump identification and analysis package, in: Astronomical Data Analysis Software and Systems XVI, p. 425.

Bertocco, S., Dowler, P., Gaudet, S., Major, B., Pasian, F., Taffoni, G., 2018. Cloud access to interoperable IVOA-compliant VOSpace storage. Astronomy and Computing 24, 36–44. doi:10.1016/j.ascom.2018.05.003, arXiv:1806.04986.

Braam, P.J., Zahir, R., 2002. Lustre: A scalable, high performance file system. Cluster File Systems, Inc .

Bui, T., 2015. Analysis of docker security. CoRR abs/1501.02967. URL: http://arxiv.org/abs/1501.02967, arXiv:1501.02967.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., Wilkes, J., 2016. Borg, omega, and kubernetes. Communications of the ACM 59, 50–57.

ChiVO-Team, 2018. A JupyterHub server over high-availability technologies. URL: https://github.com/ChileanVirtualObservatory/jovial.chivo.cl. (accessed February 26, 2018).

Dewdney, P.E., Hall, P.J., Schilizzi, R.T., Lazio, T.J.L., 2009. The square kilometre array. Proceedings of the IEEE 97, 1482–1496.

Díaz, M., Araya, M., Jauregui, C., Valenzuela, C., Pizarro, L., Osorio, M., Solar, M., 2018. Docker-based implementation for an astronomical data analysis cloud service, in: XXVII Astronomical Data Analysis Software & Systems, ADASS. pp. 1–4. [To appear].

Djorgovski, S., Brunner, R., Mahabal, A., Williams, R., Granat, R., Stolorz, P., 2003. Challenges for cluster analysis in a virtual observatory, in: Statistical Challenges in Astronomy. Springer, pp. 127–141.

Fitzpatrick, M.J., Olsen, K., Economou, F., Stobie, E.B., Beers, T., Dickinson, M., Norris, P., Saha, A., Seaman, R., Silva, D.R., et al., 2014. The noao data laboratory: a conceptual overview, in: SPIE Astronomical Telescopes+ Instrumentation, International Society for Optics and Photonics. pp. 91491T–91491T.

Gilmozzi, R., Spyromilio, J., 2007. The european extremely large telescope (e-elt). The Messenger 127, 3.

Greenfield, P., 2011. What python can do for astronomy, in: Astronomical Data Analysis Software and Systems XX, p. 425.

Higgins, J., Holmes, V., Venters, C., 2015. Orchestrating docker containers in the hpc environment, in: International Conference on High Performance Computing, Springer. pp. 506–513.

Ivezic, Z., Tyson, J., Abel, B., Acosta, E., Allsman, R., AlSayyad, Y., Anderson, S., Andrew, J., Angel, R., Angeli, G., et al., 2008. Lsst: from science drivers to reference design and anticipated data products. arXiv preprint arXiv:0805.2366 .

JupyterHub-Team, 2018. Multi-user server for Jupyter notebooks. URL: `https://github.com/jupyterhub/jupyterhub`. (accessed February 26, 2018).

JupyterLab-Team, 2018. JupyterLab computational environment. URL: `https://github.com/jupyterlab/jupyterlab`. (accessed February 26, 2018).

Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A., 2007. kvm: the linux virtual machine monitor, in: Proceedings of the Linux symposium, pp. 225–230.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B.E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J.B., Grout, J., Corlay, S., et al., 2016. Jupyter notebooks-a publishing format for reproducible computational workflows., in: ELPUB, pp. 87–90.

Marmol, V., Jnagal, R., Hockin, T., 2015. Networking in containers and container clusters. Proceedings of netdev 0.1, February .

McMullin, J., Waters, B., Schiebel, D., Young, W., Golap, K., 2007. Casa architecture and applications, in: Astronomical data analysis software and systems XVI, p. 127.

Medvedev, D., Lemson, G., Rippin, M., 2016. Sciserver compute: Bringing analysis close to the data, in: Proceedings of the 28th International Conference on Scientific and Statistical Database Management, ACM. p. 27.

Merkel, D., 2014. Docker: lightweight linux containers for consistent development and deployment. Linux Journal 2014, 2.

Moolekamp, F., Mamajek, E., 2015. Toyz: A framework for scientific analysis of large datasets and astronomical images. Astronomy and Computing 13, 50–57.

Parente, P., 2018. Estimate of Public Jupyter Notebooks on GitHub. URL: `http://nbviewer.jupyter.org/github/parente/nbestimate/blob/master/estimate.ipynb`. (accessed February 26, 2018).

Pérez, F., Granger, B.E., 2007. Ipython: a system for interactive scientific computing. Computing in Science & Engineering 9.

Robitaille, T.P., Tollerud, E.J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A.M., Kerzendorf, W.E., et al., 2013. Astropy: A community python package for astronomy. Astronomy & Astrophysics 558, A33.

Rocklin, M., 2015. Dask: parallel computation with blocked algorithms and task scheduling, in: Proceedings of the 14th Python in Science Conference, pp. 130–136.

Sabater, J., Sánchez-Expósito, S., Best, P., Garrido, J., Verdes-Montenegro, L., Lezzi, D., 2017. Calibration of LOFAR data on the cloud. Astronomy and Computing 19, 75–89. doi:`10.1016/j.ascom.2017.04.001`, `arXiv:1704.05064`.

Shen, H., 2014. Interactive notebooks: Sharing the code. Nature 515, 151.

Solar, M., Araya, M., Arévalo, L., Parada, V., Contreras, R., Mardones, D., 2015. Chilean virtual observatory, in: 41st Latin American Computing Conference (CLEI), IEEE. pp. 1–7.

Taylor, M., Boch, T., Fay, J., Fitzpatrick, M., Paioro, L., 2012. Samp: Application messaging for desktop and web applications. Astronomical Data Analysis Software and Systems XXI 461, 279.

Testi, L., Schilke, P., Brogan, C., 2010. Report on the Workshop Data Needs for ALMA From Data Cubes to Science: Ancillary Data and Advanced Tools for ALMA. The Messenger 139, 53–55.

Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., Wilkes, J., 2015. Large-scale cluster management at google with borg, in: Proceedings of the Tenth European Conference on Computer Systems, ACM, New York, NY, USA. pp. 18:1–18:17. URL: `http://doi.acm.org/10.1145/2741948.2741964`, doi:`10.1145/2741948.2741964`.

Villanueva, M., Valenzuela, C., Sanchez, M., Araya, M., 2018. Wrapping and deploying legacy astronomical code into python environments: An applied case study, in: XXVII Astronomical Data Analysis Software & Systems, ADASS. pp. 1–4. [To appear].

Walters, B., 1999. Vmware virtual platform. Linux journal 1999, 6.

Xavier, M.G., Neves, M.V., De Rose, C.A.F., 2014. A performance comparison of container-based virtualization systems for mapreduce clusters, in: 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), IEEE. pp. 299–306.