The NOAO Data Lab virtual storage system

Matthew J. Graham[a, b], Michael J. Fitzpatrick[a], Patrick Norris[a], Kenneth J. Mighell[a], Knut Olsen[a], Elizabeth B. Stobie[a], Stephen T. Ridgway[a], Adam S. Bolton[a], Abhijit Saha[a], and Lijuan (Wendy) Huang[a]

[a]National Optical Astronomy Observatory, 950 N. Cherry Avenue, Tucson, AZ 85719, USA
[b]California Institute of Technology, 1200 E California Blvd, Pasadena, CA 91107, USA

ABSTRACT

Collaborative research/computing environments are essential for working with the next generations of large astronomical data sets. A key component of them is a distributed storage system to enable data hosting, sharing, and publication. VOSpace[1] is a lightweight interface providing network access to arbitrary backend storage solutions and endorsed by the International Virtual Observatory Alliance (IVOA). Although similar APIs exist, such as Amazon S3, WebDav, and Dropbox, VOSpace is designed to be protocol agnostic, focusing on data control operations, and supports asynchronous and third-party data transfers, thereby minimizing unnecessary data transfers. It also allows arbitrary computations to be triggered as a result of a transfer operation: for example, a file can be automatically ingested into a database when put into an active directory or a data reduction task, such as Sextractor, can be run on it. In this paper, we shall describe the VOSpace implementations that we have developed for the NOAO Data Lab. These offer both dedicated remote storage, accessible as a local file system via FUSE, and a local VOSpace service to easily enable data synchronization.

Keywords: Science exploration framework, Virtual storage, data services, remote execution

1. INTRODUCTION

Driven by new facilities and instruments with ever larger collecting areas, higher sensitivities and faster sampling rates, astronomy is transitioning from dealing with giga- and terascale data sets to petascale and beyond. These present new technological challenges, both for data analysis and data management. Unfortunately, Moore's law only helps with computing power and storage needs. The limitations of latency, schedulability, and ultimately physics (the speed of light and the nonlinear Shannon limit), mean that it does not apply to networking. Existing work practices, such as downloading a whole data set to local disk to analyze it will soon be a thing of the past. *"Code to data"* is the mantra of the new data analysis paradigm/movement and it is being enabled by an emerging landscape of science exploration frameworks (SEFs).

There is nothing inherently novel in (using) a remote computing infrastructure – the Cloud is an example of a recent trend – but SEFs focus on colocation, user experience (particularly interactivity), and integration. Placing computation as close to the data as possible minimizes unnecessary network traffic and potential bottlenecks in the data flow at transition points. The storage may be treated as virtual (and managed through web interfaces) but unlike generic cloud systems, its geographic location is never in doubt. Minimized latencies also make interactive tasks, such as exploratory visualization, tractable with local tools driving remote processing seamlessly in real time.

Collaborative analysis is also a strong feature of working with such large data sets. The expertise required for a correct and rigorous statistical evaluation employing machine learning to do the heavy lifting and appropriate domain-specific interpretation of the visualized results rarely resides in a single location but is normally distributed. Any supporting framework must therefore enable easy sharing of data products and associated metadata, such as annotations, within user-defined groups. Simultaneous write access to group workspaces (directories) is also a necessary requirement for collaborative editing.

Further author information: (Send correspondence to M.J.G.)
M.J.G.: E-mail: graham@noao.edu

Software and Cyberinfrastructure for Astronomy IV, edited by Gianluca Chiozzi, Juan C. Guzman, Proc. of SPIE Vol. 9913, 99131I · © 2016 SPIE
CCC code: 0277-786X/16/$18 · doi: 10.1117/12.2233306

From these considerations, it can be seen that not only is the data storage system at the heart of any such infrastructure but that it also needs to do more than just store bytes. In this paper, we describe the virtual storage system implemented within the NOAO Data Lab,[2] a science exploration framework built around the US community ground-based nighttime astronomy data archives. It is based on the International Virtual Observatory Alliance[*] (IVOA) VOSpace standard.[1]

2. THE NOAO DATALAB

NOAO is currently supporting two wide-field instruments in its portfolio: the Dark Energy Camera (DECam) is deployed on the Blanco 4m at CTIO, Chile and surveying most of the southern sky through a combination of the Dark Energy Survey (DES) and community-led observations. The Dark Energy Spectroscopic Instrument (DESI) will be deployed on the Mayall 3.6m at Kitt Peak, Arizona in 2018 and obtain spectra for tens of millions of galaxies and quasars, primarily in the northern sky. The image set for DECam already totals 375 TB of 32-bit floating point data and together with the DESI imaging targeting surveys (DECaLS, DECaLS+, and MzLS) will easily approach 1 PB of imaging data within the next couple of years. These are coupled with very large catalogs produced from the surveys - the DES catalog alone is expected to total 45 TB.

The NOAO Data Lab is designed to enable efficient exploration and analysis of these data sets (it will be released to the public in mid-2017). In particular, it aims to support four approaches to science with such large amounts of data:

- Catalog Science, for which discoveries are made purely from catalogs;
- Data Exploration, for which access to catalogs and pixels is needed to provide the freedom to explore data sets;
- Collaborative Research, for which the work required to make a discovery depends on the coordinated efforts of a team of people;
- User-defined Custom Workflows, which will involve potentially complex analyses of catalog objects and/or pixels, requiring interfaces to user scripts and software.

Architecturally, the Data Lab follows a modular design (see Fig. 1) with a set of data access services acting as the primary interface to external requests (either from clients or software). These implement the appropriate IVOA standard - Simple Image Access[3] (SIA) for pixel access, Simple Cone Search[4] (SCS)/Table Access Protocol[5] (TAP) for catalog access, etc. - and so can work with existing community VO tools, such as TOPCAT[†] and Aladin[‡], out of the box. These interfaces can be unfamiliar, however, to end user astronomers (they emphasize machine interoperability rather than user friendliness) and so the Data Lab also provides a set of more intuitive service APIs that act as middleware to both the data access services and directly to backend infrastructure. These include a query manager service, a storage manager service (see below), a job manager service, and an authentication service.

Each registered user of the Data Lab, who can be individuals or groups, will get a storage allocation (nominally 1 TB), a database allocation, and a computational resource allocation. Such an allocation pack of Data Lab resources may also be associated with an award of observing time on one of the NOAO facilities. The Data Lab will also provide a command line client and client side libraries to interact with these allocations via the service APIs.

[*]http://www.ivoa.net
[†]http://www.star.bris.ac.uk/ mbt/topcat/
[‡]http://aladin.u-strasbg.fr/

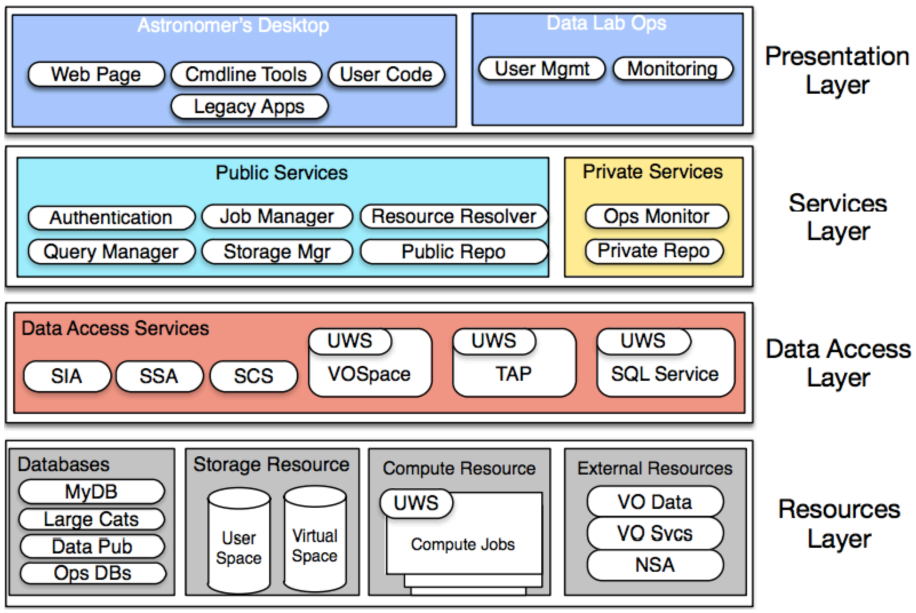


Figure 1. The modular and layered architecture of the NOAO Data Lab.

3. VIRTUAL COLLABORATIVE STORAGE

The science use cases for the Data Lab define two main motivations for providing a large virtual storage capability: (i) analysis services need to save intermediate results or stage data for download at a later time; and (ii) to support sharing of any resultant data with other users in a collaboration. To address these, the Data Lab will provide a large storage area as a basic feature of the system. As mentioned above, registered users will receive a storage allocation and this can take one of three forms, depending on the nature of the user as well as the number of users being supported and the resources available:

- Personal space - a quota allocated to individual users for anything they like
- Transient space - a working partition with a limited lifespan for staging result data
- Collaborative space - a long-term redundant storage for survey groups/supported projects

It should be noted that the transient space potentially overlaps in scope with the user's database allocation which may be used to stage intermediate query results. Although a common interface to manage such temporary spaces could be defined (and possible extended to include in-memory staged data as well), it seems more expedient to make the distinction based on the nature of the backend store - file system vs. database - rather than temporary vs. permanent space. Collaborative spaces are also not just available for groups with ongoing survey programs on NOAO facilities but also for supported archival projects. One could therefore envisage a new category in the regular NOAO Call For Proposals identifying projects for which no new photons will be collected but rather just existing measured ones reused.

4. VOSPACE

VOSpace[1] is the IVOA interface to distributed storage. It is designed to be a lightweight protocol, abstracting any details of the backend storage implementation and focusing on data control operations. It is transport protocol agnostic, delegating such details to client-server negotiation, although it does support HTTP for certain expedient operations. It uses the IVOA Universal Worker Service (UWS)[6] pattern to manage asynchronous operations. There are a number of similar APIs for managing networked file stores, such as Amazon S3, Dropbox, and WebDav, that could have been used instead. However, VOSpace was chosen as the Data Lab storage

API because it was designed to specifically address astronomy use cases, supporting third-party data transfers, arbitrary metadata tagging, on-the-fly format conversions, and triggered computations.

In VOSpace, users interact with logical representations of the (stored) data objects called *nodes* which carry all the associated metadata. There are several types, corresponding to directories, links, files of a known format (e.g., FITS), and blobs. The physical bytes reside in whatever backend storage implementation is being used by a particular VOSpace instance and are only handled when a data transfer is actually performed. Table 1 gives a summary of the various operations that the RESTful API supports. The negotiated operations involve the /transfers endpoint. The client provides a list of protocols (in preferential order) that they are willing to use for a data transfer; the server then responds with the subset that it is willing to support and the operational details for each. The client then tries the first and works down the list if it fails. The actual process of the transfer is managed as an asynchronous operation using the UWS endpoints to check for status, get results, etc.

Table 1. The VOSpace API and how it maps to the storage manager service API.

Method	Description	VOSpace action	Storage manager method
Service metadata			
getProtocols	Get supported protocols	GET /protocols	-
getViews	Get available views	GET /views	-
getProperties	Get system properties	GET /properties	-
Node manipulation			
createNode	Create a node	PUT /nodes	-
moveNode	Move/rename a node	POST /transfers	mv
copyNode	Copy a node	POST /transfers	cp
deleteNode	Delete a node	DELETE /nodes	rm/rmDir
Transferring data			
pushToVoSpace	Client-initiated import	POST /transfers	put, cp
pullToVoSpace	Server-initiated import	POST /transfers	cp
pullFromVoSpace	Client-initiated export	POST /transfers	get, cp
pushFromVoSpace	Server-initiated export	POST /transfers	cp
Service metadata			
getNode	Get a node representation	GET /nodes	ls
setNode	Set a node representation	PUT /nodes	-
findNodes	Find a node matching search criteria	PUT /searches	-

Third-party transfers between two VOSpace instances are orchestrated by the user using either a client-initiated import and server-initiated export or a client-initiated export and server-initiated input; for example, a user negotiates a *pushToVoSpace* with service B and uses the results as input to a *pushFromVoSpace* with service A. It is thus only the data control that goes through the client with the actual bytes transferred directly from one service to the other.

VOSpace provides a standard set of metadata for each node in the form of *properties*. These map to standard POSIX file system metadata, such as owner, size, creation date, and access permissions. Almost all service-provided properties are immutable - the obvious exception are access permissions. Users can, however, define and attach an arbitrary set of properties to any node. These are limited to keyword-value type relationships and multiple value properties need to be managed by the user, e.g., separating different values with a comma. The VOSpace API does support a *findNodes* operation so that nodes with particular property values can be sought (this is again an asynchronous operation managed with the UWS pattern) but multiple-valued parameters need to be handled explicitly by the user.

Format conversions are a common operation with both tabular, e.g., binary FITS to CSV, and image data, e.g., FITS to JPEG. In VOSpace, such a conversion is considered to be an alternate *view* of the data, akin to database views. By default, stored data is tagged as having the default view which means that it is treated as a blob and no alternate views (format conversion) are supported. However, if the format was explicitly stated when it was stored, either through the view of the transfer task or the type of node used to represent the data, then a data transfer will allow alternate views to be specified as an additional argument. It should be noted that even when data is tagged with just the default view, the input byte signature (checksum) need not necessarily match the output byte signature as the underlying storage implementation may not preserve the original structure, e.g., if an archive format, such as a tarball, is unpacked.

Workflows often feature common operations, particularly as part of any preprocessing or data munging steps before the main analysis: for example, ingesting a tabular file into a database, running a source finder over a downloaded image to generate a source catalog, or retrieving a set of images from a remote server. VOSpace provides a mechanism for the automatic triggering of any computational task (called a *capability* in this context) when a file is placed in a specific location. Specific capabilities can be enabled on directories which then become active working directories for those tasks. A capability may well be an asynchronous task that needs to be managed but this is outside the scope of the VOSpace specification, although the UWS pattern does provide a means to do so. Capabilities will also commonly generate additional files on the backend and these may need to be registered with the VOSpace somehow but this is again outside the scope of the current specification and an implementation detail.

5. THE NOAO DATALAB VOSPACE INSTANCE

The NOAO Data Lab VOSpace instance is a complete implementation of the IVOA VOSpace 2.0 specification,[1] i.e., it implements all the optional methods in the protocol as well as the mandatory ones. The service is written in Java 7.0 using the Apache Wink[§] framework, which implements the JAX-RS (Java API for RESTful web services) API. It uses the VTD-XML[¶] library for fast XML processing and the CDS UWS[‖] library for managing all asynchronous tasks. The service is built using Ant and deployed as a webapp using the Apache Tomcat framework and integrated with the Data Lab authorization service for authentication and authorization.

The service currently employs a standard flat file system as the storage backend and MySQL[**] as a metadata store containing all node and transfer instances, both in the form of extracted parameters, e.g., identifier, view, and creation date, and the respective full XML documents. It supports HTTP as the data transfer protocol and makes use of one-time URLs with a 1 hour persistence for all transfer endpoints. Extensions to support other storage backends, metadata stores, and transfer protocols are straightforward with a technology-relevant implementation of the appropriate interface class, e.g., StoreManager, MetaStore, or ProtocolHandler, all that is required.

Capabilities are implemented as either a specific Java class (part of the VOSpace installation) or as a task definition that is passed to an instance of a capability service. This is currently a simple Python Flask[††]-based service that interfaces to the Data Lab Job Manager. A task definition is essentially a parameterized command line argument, for example, saying to run Sextractor[7] with a set of parameter values or a Python script with a set of specified arguments. A capability is enabled on a directory by putting a capability configuration file (<*capname*>_cap.conf) into that directory; removing the configuration file will disable that capability for the directory. Multiple capabilities may be active on a single directory at a given time and all will be invoked for any file placed in that directory. In the case of a task definition capability, invocation means that a request is sent to the appropriate capability service with the name of the file to run on (HTTP POST with parameter name=<file>).

The VOSpace configuration file defines which service capabilities are supported by a particular VOSpace instance. For implemented capabilities, the class name of the implementation is given as well as any necessary parameters, such as port number. If no class name is given then it is assumed that the capability will be a defined task. When the VOSpace service is started, it creates a mapping between the specified capabilities and their respective class instances. Each implemented capability returns an identifier for the capability it encodes

[§]https://wink.apache.org
[¶]http://vtd-xml.sourceforge.net/
[‖]http://cdsportal.u-strasbg.fr/uwstuto/
[**]https://www.mysql.com
[††]http://flask.pocoo.org

ivo://datalab.noao.edu/vospace/capabilities#<capName> which is used in the mapping. Defined tasks map to the capability service.

To ensure that any changes to the code base have minimal impact on the expected functionality of the service, there is a comprehensive test framework written in Python. This is run after each fresh check -in of code to the Data Lab GitLab repository (an internally hosted web-based Git repository manager), which is typically after a new feature has been added or an identified issue has been addressed – these are tracked with the issue tracking facility within GitLab. The test framework shares common code with Python client software for VOSpace that is part of several tools, such as the datalab command line client and the storage manager interface.

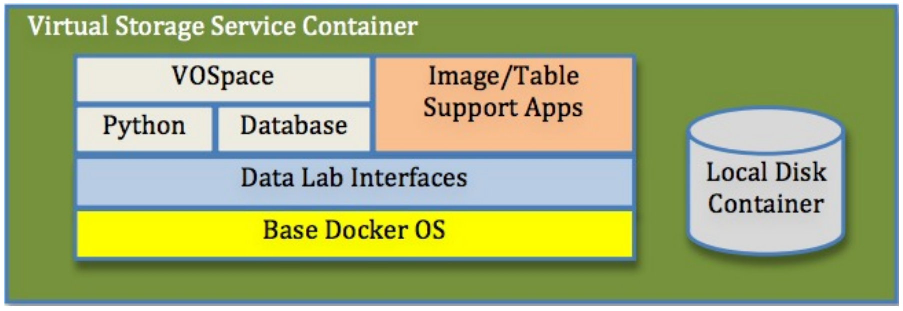


Figure 2. This shows the architecture of the virtual storage service packaged as a Docker container for easy distribution.

In addition to the main Java implementation, there is also a Python one. This is again a complete implementation of the VOSpace 2.0 specification but now written in Python 2.7. It is built around the CherryPy[‡‡] web framework and makes use of an in-house UWS library for management of asynchronous services. MySQL is still used as the metastore but the code is easily extensible to support other databases. The Python implementation is ideally suited for distribution since the language easily supports an embedded web server and database (e.g., SQLite), making the entire service self-contained. Data Lab makes use of the Docker[*] container system to build self-contained application containers that will execute on the compute servers via the Job Manager service. This container mechanism allows VOSpace and its supporting software to be packaged (see Fig. 2) in a way that isolates it from the underlying system, which reduces the installation process for the user to just that of enabling Docker on their machine (which is trivial for both Linux and Mac systems), optionally modifying a local configuration file and then simply executing the container to run the service.

6. STORAGE MANAGER SERVICE

VOSpace is a transactional REST API involving the orchestrated exchange of XML documents which can be quite overwhelming for a scientist end user. There are also a number of Data Lab operations involving the VOSpace service but that are beyond the scope of its functionality, e.g., interfacing with a user's private database space or coordinating the transfer process of data to/from another VOSpace. The storage manager service provides the simpler-to-use higher level interface to perform such tasks. The basic mapping between the VOSpace API and the storage manager API is shown in Table 1.

The storage manager service is implemented as a basic Flask-based app in Python 2.7. It verifies all HTTP requests with the Data Lab authentication service (via a security token in the message header) so that only allowed users have access to data control operations. VOSpace operations are designed to be fairly atomic but the corresponding service manager methods are more polymorphic; for example, the cp method acts as both a client to the VOSpace copyNode method and the data transfer methods, depending on what the relative locations of the cp method's source and target arguments are. The use of such a service API as the main interface point to Data Lab storage is a more flexible approach since it is easily extensible to incorporate additional features addressing new identified requirements.

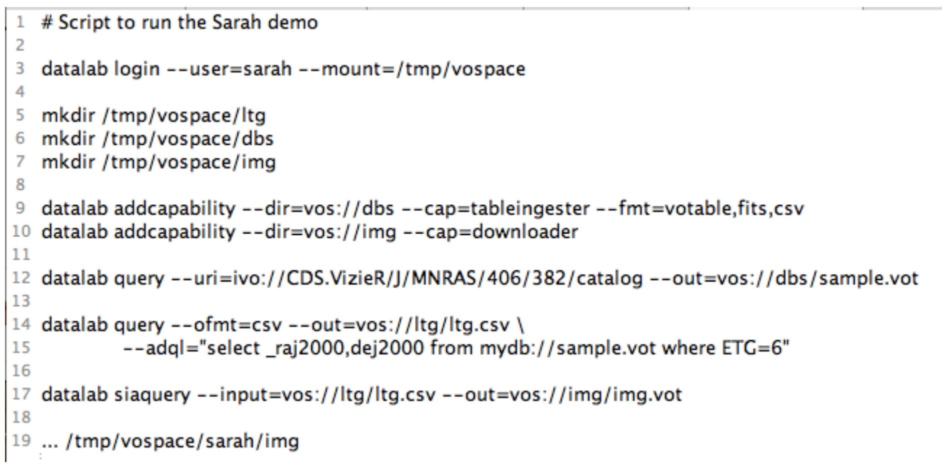
[‡‡]http://www.cherrypy.org
[*]http://www.docker.org

7. LOCALIZED WORKSPACES

Ideally, a user would like to interact with their remote storage as though it were a part of the local file system. This is a common operating mode for cloud storage file systems, such as Dropbox[*] and Google Drive[†]: the client works with files and directories in a local cache which is kept synchronized with the cloud store via a proprietary tool and protocol as long as the client is on-line. The Data Lab supports two different mechanisms to provide this same type of functionality. The containerized Python VOSpace implementation can be run as a local storage system with a localized version of the storage manager service enabling easy access. A synchronization feature can be added to this to update the user's remote virtual storage at user-specified instants with the local content. This places control of the state of the two spaces squarely with the user.

A more transparent mechanism is provided by enabling access to the remote VOSpace via FUSE[‡] (file system in Userspace). A FUSE file system is an operating system mechanism that allows a non-privileged user to mount a data source as a standard Unix file system. In this case, the FUSE client will access the virtual storage using the standard VOSpace protocols but to the user and any applications they run, it will appear as though it is a normal Unix file system, accessible via the usual means. The FUSE client is also responsible for authenticating itself to the service using the Data Lab authentication service. The Data Lab employs a modified version of the CADC FUSE client[§] to support this mode of access to its VOSpace installation. It should be noted that this makes use of a local file system cache for better performance.

8. PROOF OF CONCEPT: THE SARAH DEMO



```
1  # Script to run the Sarah demo
2
3  datalab login --user=sarah --mount=/tmp/vospace
4
5  mkdir /tmp/vospace/ltg
6  mkdir /tmp/vospace/dbs
7  mkdir /tmp/vospace/img
8
9  datalab addcapability --dir=vos://dbs --cap=tableingester --fmt=votable,fits,csv
10 datalab addcapability --dir=vos://img --cap=downloader
11
12 datalab query --uri=ivo://CDS.VizieR/J/MNRAS/406/382/catalog --out=vos://dbs/sample.vot
13
14 datalab query --ofmt=csv --out=vos://ltg/ltg.csv \
15           --adql="select _raj2000,dej2000 from mydb://sample.vot where ETG=6"
16
17 datalab siaquery --input=vos://ltg/ltg.csv --out=vos://img/img.vot
18
19 ... /tmp/vospace/sarah/img
```

Figure 3. This script for the Sarah demo.

To demonstrate the capabilities of the Data Lab's virtual storage system and the benefits of a colocalized approach to data and computing, a toy scenario was constructed (see Fig. 3). Sarah, an astronomer, is interested in late-type galaxies (LTGs) and is wondering whether DECam-reduced images in Y1 Dark Energy Survey (DES) release have sufficient resolution and depth to be of scientific use to her. She decides to perform an exploratory analysis on a sample of known LTGs in the SDSS Stripe 82 field. This will inform her whether it is worth carrying out a more detailed study using the entire DES release.

First (line 3), Sarah logins into the Data Lab and mounts her virtual storage on her local machine via the FUSE layer functionality. She (lines 5-7) creates working directories in the space: one as a workspace (*/ltg*), one to hold tables (*/dbs*) and one to hold images (*/img*). She enables (lines 9-10) capabilities on two of the directories: one (*tableingester*) to automatically ingest any tabular file placed in one into her personal Data Lab

[*]https://www.dropbox.com
[†]https://www.google.com/drive
[‡]http://fuse.sourceforge.net
[§]https://github.com/canfar/vos

database (referred to as MyDB); and one (*downloader*) to automatically retrieve any images referred to in any VOTable placed in the directory (via any accessURL columns). Her Data Lab environment is now ready for her to start her exploratory work.

Sarah downloads (line 12) a paper she remembers has a list of LTGs in it from Vizier[¶] and saves it to the */dbs* directory. It is parsed and a database table created in her MyDB with its contents. She then extracts (line 14) the positions of the LTGs in this table with a simple query and saves them in a CSV file in the workspace directory (*/ltg*). Next (line 17), she uses the CSV file as input to the Data Lab image access (SIA) service and saves the results in the */img* directory. The output from the service is a VOTable with an accessURL column linking to the Data Lab image cutout service. As the table is placed in the */img* directory with an active downloader capability, thumbnail cutouts are automatically retrieved for each row. Finally (line 19), Sarah can examine the DES image thumbnails of the LTGs with her favorite legacy code.

The total amount of data that Sarah works with is around 200 GB of images. Traditionally, she would have downloaded this all to her local filesystem and then run local code on it to produce the thumbnails before examining them. The data load for this simple exploratory analysis is at the limit of what is tractable with current networks and would take about 15 hours to download alone. Using the Data Lab, she can be inspecting the final results in about 15 minutes and the only data that will be transferred will be the small cutout images. All the heavy data transfers and processing has happened remotely but to Sarah, it appears as though it is all local.

9. CONCLUSIONS

NOAO is experiencing an explosion in image, catalog and (soon) spectral data that is beyond the scope of most researchers to download and analyze locally. The Data Lab is a science exploration framework that aims to provide the community with the wherewithal to efficiently access and interact with this treasure trove. At its core is a virtual storage system that fosters collaborative work through both the lightweight but powerful VOSpace interface and a higher-level service coordinating interface. The remote storage can be used as though it were local. Third-party applications can also be triggered to act on files placed in active directories. These features can help automate common preprocessing steps and workflows and make the Data Lab an essential part of the future community astronomer's toolkit.

Acknowledgments

NOAO is operated by the Association of Universities for Research in Astronomy (AURA). under cooperative agreement with the National Science Foundation.

REFERENCES

[1] Graham, M., Morris, D., Rixon, G., Dowler, P., Schaaff, A., and Tody, D., "VOSpace specification Version 2.0." IVOA Recommendation 29 March 2013 (Mar. 2013).

[2] Fitzpatrick, M., Graham, M., Olsen, K., Mighell, K., Norris, P., Ridgway, S., Stobie, E., and Bolton, A., "The NOAO data lab: science-driven development," in [*Software and Cyberinfrastructure for Astronomy IV*], Chiozzi, G. and Guzman, J. C., eds., *Proc. SPIE* **9913**, 128 (2016).

[3] Harrison, P., Tody, D., and Plante, R., "Simple Image Access Specification Version 1.0." IVOA Recommendation 11 November 2009 (Nov. 2009).

[4] Plante, R., Williams, R., Hanisch, R., and Szalay, A., "Simple Cone Search Version 1.03." IVOA Recommendation 22 February 2008 (Feb. 2008).

[5] Dowler, P., Rixon, G., and Tody, D., "Table Access Protocol Version 1.0." IVOA Recommendation 27 March 2010 (Mar. 2010).

[6] Harrison, P. and Rixon, G., "Universal Worker Service Pattern Version 1.0." IVOA Recommendation 10 October 2010 (Oct. 2010).

[7] Bertin, E. and Arnouts, S., "SExtractor: Source Extractor." Astrophysics Source Code Library (Oct. 2010).

[¶]http://vizier.u-strasbg.fr