A Scalable Cloud-Based Analysis Platform for Survey Astronomy

Steven Stetzler
DiRAC, Department of Astronomy
University of Washington
Seattle, WA, U.S.A.
stevengs@uw.edu

Colin Slater
DiRAC, Department of Astronomy
University of Washington
Seattle, WA, U.S.A.
ctslater@uw.edu

Petar Zecevic
Faculty of Electrical Engineering and Computing
University of Zagreb
Zagreb, Croatia
petar.zecevic@fer.hr

Mario Juric
DiRAC, Department of Astronomy
University of Washington
Seattle, WA, U.S.A.
mjuric@astro.washington.edu

***Abstract*—Research in astronomy is undergoing a major paradigm shift, transformed by the advent of large, automated, sky-surveys into a data-rich field where multi-TB to PB-sized spatio-temporal datasets are commonplace. For example the Legacy Survey of Space and Time is about to begin delivering observations of $>10^{10}$ objects, including a database with $>4 \times 10^{13}$ rows of time series data. This volume presents a challenge: how should a domain scientist with little experience in data management or distributed computing access data and perform analyses at PB-scale?**

We present a possible solution to this problem built on (adapted) industry standard tools and made accessible through web gateways. We have i) developed Astronomy eXtensions for Spark, AXS, a series of astronomy-specific modifications to Apache Spark allowing astronomers to tap into its computational scalability, ii) deployed datasets in AXS-queriable format in Amazon S3, leveraging its I/O scalability, iii) developed a deployment of Spark on Kubernetes with auto-scaling configurations requiring no end-user interaction, and iv) provided a Jupyter notebook, web-accessible, front-end via JupyterHub including a rich library of pre-installed common astronomical software (accessible at http://hub.dirac.institute).

We use this system to enable the analysis of data from the Zwicky Transient Facility, presently the closest precursor survey to the LSST, and discuss initial results. To our knowledge, this is the first application of cloud-based scalable analytics to astronomical datasets approaching LSST-scale. The code is available at https://github.com/astronomy-commons.

I. Introduction

Today's astronomy is undergoing a major change. Historically a data-starved science, it is being rapidly transformed by the advent of large, automated, digital sky surveys into a field where terabyte and petabyte datasets are routinely collected and made available to researchers across the globe.

Two years ago, the Zwicky Transient Facility (ZTF; [1]) began a three-year mission to monitor the Northern sky. With a large camera mounted on the Samuel Oschin 48-inch Schmidt telescope at Palomar Observatory, ZTF is able to monitor the entire visible sky almost twice a night. Generating about 30 GB of nightly imaging, ZTF detects approximately 1,000,000 variable, transient, or moving sources (or alerts) every night, and makes them available to the astronomical community. Towards the middle of 2022, a new survey, the Legacy Survey of Space and Time (LSST; [2]), will start operations on the NSF Vera C. Rubin Observatory. The LSST has a mirror almost seven times larger than that of the ZTF, which will enable it to search for fainter and more distant sources. Situated in northern Chile, the LSST will survey the southern sky taking $\sim 1,000$ images per night with a 3.2 billion-pixel camera that is capable of imaging an area 40x the size of the full moon in a single exposure. The stream of imaging data ($\sim$6PB/yr) collected by the LSST will yield repeated measurements ($\sim$100/yr) of over 37 billion objects, for a total of over 30 trillion measurements by the end of the next decade. These are just two examples, with many others at similar scale either in progress (Kepler, Pan-STARRS, DES, GAIA, ATLAS, ASAS-SN; [3]–[7]) or planned (WFIRST, Euclid; [8], [9]). They are being complemented by numerous smaller projects ($\lesssim$ \$1M scale), contributing billions of more specialized measurements.

The main products of these surveys are *astronomical catalogs*. At the highest level[1], astronomical catalogs are large ($\gtrsim 10^{10}$ rows) collections of *objects* – such as galaxies, stars, planets, asteroids, and more – identified in acquired imaging. They are accompanied by 1-3 orders of magnitude larger tables of individual *observations* ($\gtrsim 10^{13}$ rows) that store *measurements* of received flux, position, shape, and other properties of the object. Both tables are information rich and fairly wide: for example, the object table in Pan-STARRS contains 129 columns while the detection table is 58 columns wide.

However, this 10-100x increase in survey data output has

Presented at Gateways 2020, Online, USA, October 12–23, 2020. https://osf.io/meetings/gateways2020/

[1]This is a rather simplified description; a typical catalog will have dozens of additional tables recording numerous calibration and other metadata.

not been followed by commensurate improvements in tools and platforms available to astronomers to manage and analyze those datasets. Most survey-based studies today are performed by navigating to archive websites, entering (very selective) filtering criteria to download "small" ($\sim$10s of millions of rows; $\sim$10GB) subsets of catalog products. Those subsets are then stored locally and analyzed using custom routines written in high-level languages (e.g., Python or IDL), with the algorithms generally assuming in-memory operation. With the increase in data volumes and subsets of interest growing towards the $\sim$100GB-1TB range, this mode of analysis is becoming infeasible.

Beyond the shear input size, the *nature* of astronomical investigations itself is changing. In the first two decades of survey-driven science, archived data has often been seen as a shortcut to observations: instead of requesting telescope time, one could "observe" a region of sky with a Structured Query Language (SQL) query. Thus, the typically requested and analyzed data subsets were relatively small. (Previous studies show frequencies of Sloan Digital Sky Survey queries follow a f^{-1} power law [10], [11].) The next decade, however, is expected to be weighted towards studies examining the *whole dataset*: performing large-scale classification (including using machine learning techniques), clustering analyses, searching for exceptional outliers, or measuring faint statistical signals [12]. The change is driven by the needs of the big scientific questions of the day. For some — such as the nature of dark energy — we are reaching the limits of measurement precision or the numbers of objects that can be observed: utilization of all available data and improved statistical treatments are the only way to an answer.

For all these reasons, the traditional "subset-download-analyze" paradigm in place today is expected to break down in the early 2020's. In this new environment, how does a domain expert access the data of interest and write/run analyses? How do they write a code simultaneously fitting a billion+ stars for distances and the interstellar dust distribution? What software should an astronomer use to orchestrate running a light-curve classifier using state-of-the-art machine learning methods that robustly handles a trillion data points?

II. A Platform for User-Friendly Scalable Analysis of Large Astronomical Datasets

We address these challenges by building a general platform for scalable computing that is co-located with data that we have uploaded to and maintain in the cloud.

A. System Architecture

The system can be broken down into three distinct components, outlined in Fig. 1:

1) An interface for computing. We use the Jupyter ecosystem, a JupyterHub deployment based on the `zero-to-jupyterhub` project that spawns Jupyter notebook [13], [14] servers for authenticated users. A Jupyter notebook server provides a web-interface to interactively run code on a remote machine alongside a set of pre-installed software libraries.
2) A scalable analytics engine. We use Apache Spark [15], an industry standard tool for distributed computing.
3) A scalable storage solution. We use Amazon Web Service's (AWS) Simple Storage Solution (S3). Amazon S3 is a managed object store that can store arbitrarily large data volumes and scale to an arbitrarily large number of requests for this data.

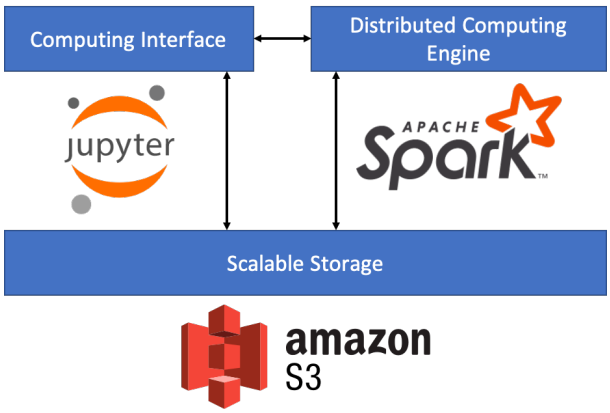

Fig. 1: Underlying the analysis platform are three components: 1) An interface to computation (Jupyter), 2) an engine for distributed computing (Apache Spark), and 3) a scalable storage solution for very large data (Amazon S3).

Each of these components are largely disconnected from one another and can be mixed and matched with other drop-in solutions.[2] At a low-level, each of these components are comprised of simple processes communicating with each other through an API over a network. This means that each solution for (1), (2), and (3) is largely agnostic to the choice of running on a bare-metal machine, inside a virtual machine (VM), inside a Linux container, or using a managed Cloud service as long as each component is properly networked.

In our particular system, we use Kubernetes to manage scheduling of the JupyterHub/Jupyter notebook and Spark processes inside of Docker containers. Kubernetes orchestrates scheduling of Docker containers with specific resource requests (among other scheduling constraints) across a set of networked computers (a cluster of bare-metal or virtual machines). We use the managed AWS Elastic Kubernetes Service (EKS) which provisions for us a fully-managed Kubernetes master node. Independent of EKS, we have a set of managed virtual machines provisioned through AWS Elastic Compute Cloud (EC2) on which the Kubernets master can schedule containers. The Kubernetes Cluster Autoscaler[3], an optional add-on to Kubernetes, can automatically scale the cluster of VMs up when there are Docker containers that

[2]Dask is a competing drop-in for Apache Spark that scales Python code natively. A Lustre file system could be a drop-in for Amazon S3. Amazon EFS, a managed and scalable network filesystem, is also an option.

[3]https://github.com/kubernetes/autoscaler

cannot be scheduled without more compute resources and can scale the cluster down when there are VMs that are under-utilized. Both JupyterHub[4] and Apache Spark[5] have built-in support for scheduling Jupyter notebook servers and Spark executor processes as containers running on Kubernetes. In addition, Kubernetes has built-in support for provisioning of cloud resources such as virtual machines, load balancers and storage devices in a cloud-agnostic way; the system we've built could easily be transferred to a different cloud provider or to any on-premises computing center that is running a supported version of Kubernetes.

Figure 2 shows the state of the Kubernetes cluster during normal usage of the platform along with the pathway of API interactions that occur as a user interacts with the system. A user gains access to the system through a JupyterHub, which is a log-in portal and proxy to one or more managed Jupyter notebook servers. The spawned notebook server can be run on any of the virtual machines rented from the cloud provider as long as these machines are networked to the JupyterHub server and the Kubernetes cluster. A proxy forwards external authenticated requests from the internet to a user's notebook server. Users can use the Apache Spark software, which is pre-installed on their server, to create a Spark cluster using the Spark on Kubernetes API.

B. Computing Interface: JupyterHub and Jupyter Notebooks

The Jupyter notebook server launches with a set of pre-installed software packages. Notably, the server has Apache Spark installed with Python bindings along with the Astronomy eXtensions for Spark (AXS; see Section II-C) that is configured to both access data stored in the cloud and have permissions to create workloads on the Kubernetes cluster. When creating a Spark cluster to query and analyze data, the user has the choice to use the resources of their notebook server (a local Spark cluster) or use the underlying Kubernetes cluster to use additional, potentially larger, compute resources (a distributed Spark cluster). In either case, computation is performed close to the data since datasets are stored on S3 (see Section II-D) in the same region as the notebook servers. When the user wishes to scale their analyses to more computers, they only need to switch their Spark cluster to use the Kubernetes API and include a few extra configurations. When requests are made for more resources, the Kubernetes cluster has the capability to automatically rent more virtual machines from the cloud provider to accommodate the increased workload.

C. Distributed Computing: Apache Spark and the Astronomy eXtensions for Spark

A key backend element of the described platform is the Astronomy eXtensions for Spark (AXS; [16]). Building on capabilities present in Spark, AXS aims to enable querying and analyzing almost arbitrarily large astronomical catalogs using

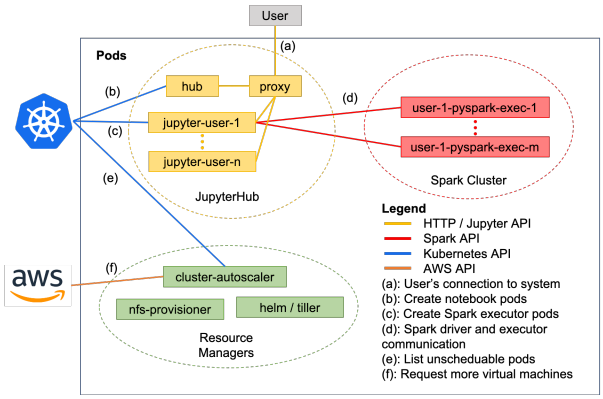

Fig. 2: A diagram of the essential components of the Kubernetes cluster when the science platform is in use. Each box represents a single Kubernetes Pod scheduled on the cluster. The colored paths and letter markers indicate the pattern of API interactions that occur when users interact with the system. (a) shows a user connecting to the JupyterHub from the internet. The JupyterHub creates a notebook server (`jupyter-user-1`) for the user (b). The user creates a Spark cluster using their notebook server as the location for the Spark driver process (c). Scheduled Spark executor Pods connect back to the Spark driver process running in the notebook server (d). In the background, the Kubernetes cluster autoscaler keeps track of the scheduling status of all Pods (e). At any point in (a)-(d), if a Pod cannot be scheduled due to a lack of cluster resources, the cluster autoscaler will request more machines from AWS to meet that need (f).

familiar Python/AstroPy concepts, DataFrame APIs, and SQL statements. We achieve this by i) adding support to Spark for efficient on-line positional cross-matching and ii) supplying a Python library supporting commonly-used operations for astronomical data analysis. To support scalable cross-matching, we developed a variant of the ZONES algorithm [17] capable of operating in distributed, shared-nothing architecture. We couple this to a data partitioning scheme that enables fast catalog cross-matching and handles the data skew often present in deep all-sky datasets. Using AXS we were able to perform an on-the-fly cross-match of Gaia DR2 (1.8 billion rows) and AllWise (900 million rows) datasets in $\sim$30 seconds. The cross-match and other often-used functionalities are exposed to the end users through an easy-to-use Python API, making this performant algorithm accessible to domain scientists.

D. Scalable Storage: Datasets Stored in S3

We store data on Amazon S3 in Apache Parquet format, a compressed columnar data storage format optimized for very fast reads of large tables. The columnar nature and partitioning of the files means that one can obtain a subset of one or more columns of a table without scanning through all of the files. There are monetary costs associated with both storing and accessing the data in S3, however costs are minimized by restricting data access to the same AWS region where the data

[4]through the development of the `KubeSpawner` (https://jupyterhub-kubespawner.readthedocs.io/en/latest/spawner.html) and Zero-to-JupyterHub project (https://zero-to-jupyterhub.readthedocs.io/)

[5]https://spark.apache.org/docs/latest/running-on-kubernetes.html

are stored. Amazon S3 provides scalable, simultaneous access to the data through simple GET/PUT API calls. Very high throughput can be achieved at the terabit-per-second level by optimizing storage access patterns and scaling requests across very many machines.[6]

III. Case Study: A Gateway for ZTF Dataset Analysis

We test the ability of this platform to enable large-scale analysis by having a group of domain scientists use it to search for Boyajian star [18] analogs in the ZTF dataset, a ~ 4 TB catalog describing light curves of ~ 3 billion objects. We note that an upcoming paper will fully describe the results of this project; here we limit ourselves to aspects necessary for the validation of the analysis system.

The main method for Boyajian-analog searches relies on querying and filtering large volumes of ZTF light curves using AXS and Apache Spark in search of dimming events in the light curves. These dimming events are a period of time in the light curve when the brightness of the star dims significantly. After filtering of the data, the group created a set of User-Defined Functions (UDFs) for model fitting that wraps the optimization library from the `scipy` [19] package. These UDFs are applied to the filtered lightcurves to parallelize least-squared fitting routines of various models to the dipping events. Figure 3 shows an outline of this science process using AXS.

The use of Apache Spark speeds up queries, filtering, and fitting of the data tremendously when deployed in a distributed environment. The group members used a Jupyter notebook on our platform to allocate a Spark cluster consisting of 96 EC2 instances. Each instance had access to 8 threads running on an AMD EPYC processor with 32 GiB of RAM, a cluster with 768 threads and 3,072 GiB of RAM. We used the Spark cluster to complete a complex filtering task on the full 4 TB ZTF data volume in $\sim$ three hours. The underlying system was able to scale to full capacity within minutes, and scale down once the demanding query was completed just as fast, providing extreme levels of parallelism at minimal cost. The total cost over the time of the query was $\sim$ \$100.

This same complex query was previously performed on a large shared-memory machine at the University of Washington with a single AMD EPYC processor with 96 threads, 1,024 GiB of RAM, and with the dataset stored on directly connected SSDs. This query previously took a full two days to execute on this hardware in comparison to the $\sim$ three hours on the cloud based science platform. Performing an analysis of this scale would not be feasible if performed on a user's laptop using data queried over the internet from the ZTF archive.

The group was able to benefit from the extreme parallelism afforded by Spark without investing significant time writing Spark-specific code. The majority of coding time was spent developing science-motivated code/logic to detect, describe, and model dipping events within familiar Python UDFs and using familiar Python libraries. In alternative systems that provide similar levels of parallelism, such as HPC systems based on batch scheduling, a user would typically have to spend significant time altering their science code to conform with underlying software that enables their code to scale. For example, they may spend significant time re-writing their code in a way that can be submitted to a batch scheduler like PBS/Slurm.

[6]This is detailed in the S3 documentation: https://docs.aws.amazon.com/AmazonS3/latest/dev/optimizing-performance.html

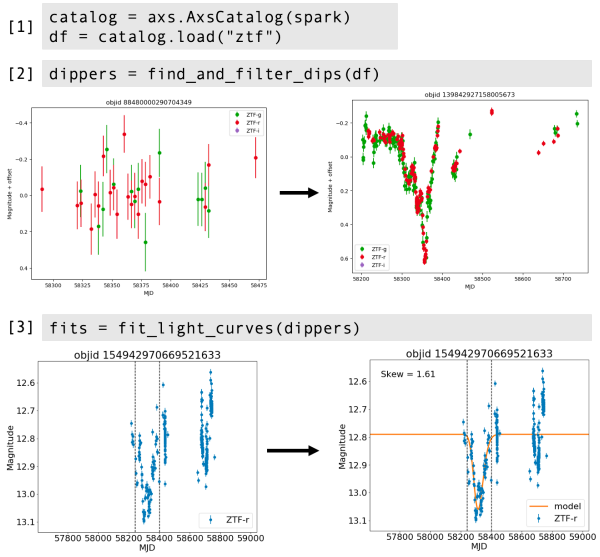

Fig. 3: An example analysis (boiled down to two lines) that finds light curves in the ZTF dataset with a dimming event. (1) shows how the ZTF dataset is loaded as a Spark DataFrame (`df`), (2) shows the product of filtering light curves for dimming events, and (3) shows the result of fitting a model to the remaining light curves. This process exemplifies that analyses can often be represented as a filtering and transformation of a larger dataset, a process that Spark can easily execute in parallel.

A. Additional Scaling Tests

Additional scaling tests verify the performance of our system. Figure 4 shows the runtime of a simple Spark query, a sum of the values of a single column of the ZTF dataset, as a function of the number of cores allocated to the Spark cluster. We observe runtime scaling of $\sim 1/\#\text{Cores}$, which is expected given this simple query is embarrassingly parallel in the map-reduce framework underlying Spark.

IV. Discussion

The scale of future datasets and the demand for large-scale and complex analyses poses significant challenges to the "subset-download-analyze" paradigm common today. Rather than downloading (now large) subsets, there are strong arguments to "bring the code to the data" and remotely perform next-to-the-data analysis via *science platforms* such as the one described in this paper (also see [20]). But this would place

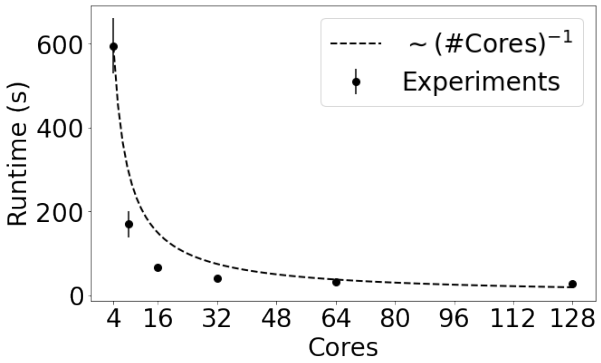

Fig. 4: The runtime scaling of a simple Spark query. Black dots show average runtimes with 1σ errorbars. Overplotted is the expected runtime scaling of $\sim 1/\#\text{Cores}$.

new demands on astronomical archives: the kinds of analyses and the size of the community to be supported would require petascale-level end-user computing resources to be deployed at archive sites.

An alternative may be to consider migrating away from the traditional on-premises solutions and towards the management of large datasets using cloud-hosted industry-standard distributed frameworks accessed through web-based remote interfaces and APIs. This shift could be dovetailed with a de-facto physical co-location of datasets on (public or private) cloud resources. This would have the benefit of making joint whole-dataset analyses possible.

Such *cloud-native* approaches offer tremendous benefits to both archives and researchers: elasticity and cost effectiveness, scalability of processing, shareability of input datasets and results, as well as increased reproducibility. In this work we've built, deployed, and demonstrated the usefulness and feasibility of one of these systems.

Acknowledgment

The authors acknowledge the support from the University of Washington College of Arts and Sciences, Department of Astronomy, and the DiRAC Institute. The DiRAC Institute is supported through generous gifts from the Charles and Lisa Simonyi Fund for Arts and Sciences and the Washington Research Foundation. M. Jurić wishes to acknowledge the support of the Washington Research Foundation Data Science Term Chair fund, and the University of Washington Provost's Initiative in Data-Intensive Discovery.

Based on observations obtained with the Samuel Oschin Telescope 48-inch and the 60-inch Telescope at the Palomar Observatory as part of the Zwicky Transient Facility project. Major funding has been provided by the U.S National Science Foundation under Grant No. AST-1440341 and by the ZTF partner institutions: the California Institute of Technology, the Oskar Klein Centre, the Weizmann Institute of Science, the University of Maryland, the University of Washington, Deutsches Elektronen-Synchrotron, the University of Wisconsin-Milwaukee, and the TANGO Program of the University System of Taiwan.

S. Stetzler acknowledges the support of the Department of Energy Computational Science Graduate Fellowship, supported through grant number DE-SC0019323.

References

[1] E. C. Bellm *et al.*, "The Zwicky Transient Facility: System Overview, Performance, and First Results," *Publications of the Astronomical Society of Pacific*, vol. 131, no. 1, p. 018002, Jan. 2019.

[2] Ž. Ivezić *et al.*, "LSST: From Science Drivers to Reference Design and Anticipated Data Products," *Astrophysical Journal*, vol. 873, no. 2, p. 111, Mar 2019.

[3] N. Kaiser *et al.*, "The Pan-STARRS wide-field optical/NIR imaging survey," in *Ground-based and Airborne Telescopes III*, ser. Proceedings of the SPIE, vol. 7733, Jul. 2010, p. 77330E.

[4] Dark Energy Survey Collaboration *et al.*, "The Dark Energy Survey: more than dark energy - an overview," *Monthly Notices of the Royal Astronomical Society*, vol. 460, pp. 1270–1299, Aug. 2016.

[5] Gaia Collaboration *et al.*, "The gaia mission," *Astronomy and Astrophysics*, vol. 595, p. A1, 2016. [Online]. Available: https://doi.org/10.1051/0004-6361/201629272

[6] J. L. Tonry *et al.*, "ATLAS: A High-cadence All-sky Survey System," *Publications of the Astronomical Society of Pacific*, vol. 130, no. 988, p. 064505, Jun 2018.

[7] B. Shappee *et al.*, "All Sky Automated Survey for SuperNovae (ASAS-SN or "Assassin")," in *American Astronomical Society Meeting Abstracts #223*, ser. American Astronomical Society Meeting Abstracts, vol. 223, Jan. 2014, p. 236.03.

[8] D. Spergel *et al.*, "Wide-Field InfrarRed Survey Telescope-Astrophysics Focused Telescope Assets WFIRST-AFTA 2015 Report," *arXiv e-prints*, p. arXiv:1503.03757, Mar 2015.

[9] R. Scaramella *et al.*, "Euclid space mission: a cosmological challenge for the next 15 years," in *Statistical Challenges in 21st Century Cosmology*, ser. IAU Symposium, A. Heavens, J.-L. Starck, and A. Krone-Martins, Eds., vol. 306, May 2014, pp. 375–378.

[10] W. O'Mullane *et al.*, "Batch is back: CasJobs, serving multi-TB data on the Web," *eprint arXiv:cs/0502072*, Feb. 2005.

[11] M. J. Kurtz and J. Bollen, "Usage Bibliometrics," *Annual Review of Information Science and Technology*, vol. 44, pp. 3–64, Jan. 2010.

[12] LSST Science Collaboration *et al.*, "LSST Science Book, Version 2.0," *ArXiv e-prints*, Dec. 2009.

[13] B. Ragan-Kelley *et al.*, "Collaborative cloud-enabled tools allow rapid, reproducible biological insights," *ISME Journal*, vol. 7, no. 3, pp. 461–464, 3 2013.

[14] T. Kluyver *et al.*, "Jupyter notebooks — a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Scmidt, Eds. IOS Press, 2016, pp. 87–90. [Online]. Available: https://eprints.soton.ac.uk/403913/

[15] M. Zaharia *et al.*, "Spark: Cluster computing with working sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. USA: USENIX Association, 2010, p. 10.

[16] P. Zečević *et al.*, "AXS: A Framework for Fast Astronomical Data Processing Based on Apache Spark," *Astronomical Journal*, vol. 158, no. 1, p. 37, Jul 2019.

[17] J. Gray, M. A. Nieto-Santisteban, and A. S. Szalay, "The Zones Algorithm for Finding Points-Near-a-Point or Cross-Matching Spatial Datasets," *eprint arXiv:cs/0701171*, Jan. 2007.

[18] T. S. Boyajian *et al.*, "Planet Hunters IX. KIC 8462852 – where's the flux?†," *Monthly Notices of the Royal Astronomical Society*, vol. 457, no. 4, pp. 3988–4004, 01 2016. [Online]. Available: https://doi.org/10.1093/mnras/stw218

[19] P. Virtanen *et al.*, "Scipy 1.0: fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, Mar 2020. [Online]. Available: https://doi.org/10.1038/s41592-019-0686-2

[20] M. Jurić, G. P. Dubois-Felsmann, and D. Ciardi, "Lsst science platform vision document," 2017. [Online]. Available: http://ls.st/lse-319