Your Data, Your Way

The iPlant Foundation API Data Services

Rion Dooley, Matthew Vaughn, Steven Terry, Dan Stanzione
Texas Advanced Computing Center
University of Texas at Austin
Austin, USA
[dooley, vaughn, sterry1, dan]@tacc.utexas.edu

Edwin Skidmore, Nirav Merchant
iPlant Collaborative
University of Arizona
Tucson, USA
[edwin, nirav]@iplantcollaborative.org

***Abstract*— This paper introduces the iPlant Foundation API Data Services, a cloud-based, hosted solution for distributed data and metadata management in the iPlant Data Store. The iPlant Data Store is a virtual storage solution for over 7000 users providing seamless access to over 6PB of distributed storage within the iPlant cyberinfrastructure using command line utilities, FUSE mounts, desktop GUI tools, and web interfaces. The Foundation API Data Services expand the standard CRUD operations by providing a collection of services that allow users to move data into and out of the iPlant Data Store using multiple transfer protocols, transform data between formats, perform advanced metadata management using both structured and unstructured data, and define their own private storage grid by registering their own resources. Concepts of unified authentication, sharing, provenance, and monitoring are baked into the services so users can focus on innovating their domain science rather than reinventing computer science. This paper briefly describe the iPlant Cyberinfrastructure, details the architecture of the iPlant Foundation API Data Services, reviews the first year of production usage, and concludes with future plans.**

Keywords—cloud, biology, cyberinfrastructure, iplant, data

I. INTRODUCTION

The iPlant Collaborative (iPlant) is a United States National Science Foundation (NSF) funded project that has created an innovative, comprehensive, and foundational cyberinfrastructure (CI) in support of plant biology research [1]. iPlant is developing cyberinfrastructure that uniquely enables scientists throughout the diverse fields that comprise plant biology to address Grand Challenges in new ways, to stimulate and facilitate cross-disciplinary research, to promote biology and computer science research interactions, and to train the next generation of scientists on the use of cyberinfrastructure in research and education. The iPlant cyberinfrastructure design is based on an unprecedented period of research community input, and leverages developments in high-performance computing, data storage, and cyberinfrastructure for the physical sciences. iPlant is an open-source project with application programming interfaces that allow the community to extend the infrastructure to meet its needs.

In this paper we present one way in which iPlant is handling data across its cyberinfrastructure. Specifically, we present the iPlant Foundation API Data Services, a cloud-based, hosted solution for distributed data and metadata management in the iPlant Data Store. The iPlant Data Store is a virtual storage solution for over 7000 users providing seamless access to distributed data within the iPlant cyberinfrastructure using command line utilities, FUSE mounts [2], desktop GUI tools, and web interfaces.

The Foundation API Data Services expand the existing basic CRUD operations by providing a collection of services that allow users to move data inside and outside of the iPlant Data Store using multiple transfer protocols, transform data between formats, perform advanced metadata management using both structured and unstructured data, and define their own private storage grid by registering their own resources.

As with the entire Foundation API, concepts of unified authentication, sharing, provenance, and monitoring are baked into the services so users can focus on innovating their domain science rather than reinventing computer science. The remainder of this paper is as follows. Section 2 describes the iPlant Cyberinfrastructure. Section 3 details architecture of the iPlant Foundation API Data Services. Section 4 highlights the first year of production usage, and Section 5 concludes with future plans.

II. THE IPLANT DATA STORE

The iPlant Data Store went into production in 2010 as a centralized facility to address the existing needs of the community to share and store scientifically relevant data sets and metadata. Underlying the Data Store is a federated network of Integrated Rule-Oriented Data System (iRODS) [3] servers running at University of Arizona and mirrored at the Texas Advanced Computing Center (TACC). The Data Store represents over 6 PB of storage capacity accessible over a 40 GB/s network. iRODS was chosen because it has a proven history of successful deployments, supports data federation as well as multi-client and multi-platform access, and has an active development team willing to address bugs and partner on new features. Other attractive features leveraged by the Data Store are the ability to search and store metadata, user and group-based access control lists (ACL), and the ability to integrate tightly into iPlant's existing infrastructure.

Users access the Data Store in multiple ways. The iPlant Discovery Environment (DE) [4], and the Davis web application [5] are the primary web interfaces. iDrop, a Java desktop application, has been very popular both from within Atmosphere virtual machines (VM) and from users' local desktops[6][7]. The iPlant Foundation API provides a RESTful web service interface, and client libraries exist in in multiple languages for further programmatic access. The FUSE interface provides a convenient, mounted file system view of iRODS, but does not support any metadata functionality. The FUSE interface is a very common access mechanism for Atmosphere users, who primarily need to access their data from command line tools as if it were a local folder. More advanced users or ones who need to move terabytes of data use i-commands due to their high performance parallel transfer capabilities[8].

All users have an initial allocation of 100GB. With a simple request, they can increase their allocation up 1 TB. Users requiring allocations greater than 1 TB can submit a justification letter, which will be reviewed by an allocations committee. The mirroring features of the Data Store mean that data is always located close to the computational resources. This means greater throughput on computational jobs, and more responsive data access within the CI.

The first two years of production usage brought conceptual, technical, and user-related challenges. One ongoing conceptual problem was identifying ways to deal with structured data. The number of data formats available today within the Plant Biology community is large and growing. Despite multiple efforts within the community, the lowest common denominator to represent data remains a file. As a result, finding general ways to store structured data in a form that can be easily imported, exported, and searched is not currently possible. Instead, the Data Store provides raw storage, and higher level services like the Foundation API and DE to provide a set of tools that operate identically on roughly a dozen well-known data types. While not optimal, this approach casts the biggest net and, hopefully, in time will incentivize the community to come to consensus on a small number of common formats.

Another conceptual challenge in building the Data Store was developing intuitive, general purpose interfaces to search through different kinds of metadata that work consistently across the CI. As mentioned above, there are multiple ways to access the Data Store. Each tool brings its own concept of interaction with metadata. The FUSE client has no concept of metadata, while the iDrop interface views metadata queries as SQL-type operations. The i-commands support both key-value lookups as well as structured queries. It may be that metadata is too contextual to make the kinds of generalizations needed for a generic interface. Clearly this is an area of ongoing research effort.

One technical challenge in using iRODS was authentication and identity management. iRODS comes with its own internal user management facility; however, iPlant already had a sophisticated web-based single sign on and identity management infrastructure. At the core of the identity management infrastructure were the Central Authentication Service (CAS) [10] and Shibboleth [11], both backed by LDAP [12], providing the web-based single sign-on mechanisms. The DE, Foundation API, user management portal, Atmosphere, ticketing system, documentation site, and forums already fully integrated with this identity management infrastructure. CAS and Shibboleth proved to be sufficient at the web layer but challenging at the web service and resource layers. Without a browser, the delegated authentication process does not work effectively. As a result, it was not possible to support the same browser-based login mechanisms for iRODS. Furthermore, earlier version of iRODS did not support external authentication, so all user accounts had to be synched by a background process out of step with the user creation and password reset mechanisms. Recently, a new version of CAS has been released which provides an OAuth2 [13] interface. This, combined with the 3.2 release of iRODS with PAM [14] support, will go a long way toward improving identity management across all of iPlant.

Another technical challenge was the need for provenance throughout the CI. In the Data Store, provenance is addressed through the use of universally unique identifiers (UUID) for every file, folder, and piece of metadata. Every action taken by a user is associated with one or more UUID and logged by a centralized tracking service. Time will tell whether this approach will scale sufficiently to handle the increased utilization of resources and the inclusion of larger HPC systems. Given the current growth projections over the coming year, the development team is confident that this solution will be adequate in the near term.

A third technical challenge to which an acceptable solution has yet to be found is that of exposing external data sources as local collections from within the Data Store. Often times, users need to access an external dataset from within the CI. In this situation it is helpful to be able to pre-define dynamic collections that, when read, will fetch the latest copy of a dataset. Examples of such dynamic collections could be the result of a database queries against the National Center for Biotechnology Information database, a downsampling of the most recent version of a derived dataset, or a web service call. In the context of databases, this would be termed a View. The justification for needing user-defined views of data is obvious, however several obstacles remain. First, performance needs to be carefully thought out. Fetching large amounts of data too often can degrade system performance both within the Data Store and on the target system. Similarly, fetching data that takes long periods of time to generate can degrade system performance just as easily. As a result, caching strategies must be derived for both scenarios. Second, the syntax for defining data views must be planned out in advance. Simply allowing users to provide arbitrary executable scripts to obtain their data exposes the underlying data and system to significant risk and must be avoided. Lastly, when an individual data object itself is dynamic, provenance is not guaranteed and reproducibility is not possible. This has implications beyond the technology. By enabling this feature, the project's service guarantees fundamentally change. That is a decision that must be approved by the organization as a whole and accepted by the user community. Whether they are willing to do so remains to be seen.

The last type of challenges experienced during the first two years of the Data Store's production operation were user-related. In comparison, these we're less challenging from a technical standpoint, but unsolvable from a practical perspective and as such will require sustained attention over the life of the project. The first user-related challenge was that of educating users on the limitations of the web-based clients for large data transfers. Given the ease of use of the DE and Davis web application, users often overlooked the fact that the data was flowing across the web via HTTP and attempted to upload files ranging from several gigabytes to over a terabyte in size. This rarely happened to the same user more than once, and never took more than a gentle reminder and note with information on the i-commands and Foundation API Data Services.

The second user-related challenge was helping users understand the network limitations between client and server. In the first year of the Data Store's operation, iPlant was holding regular workshops to train users on the proper use the CI. During the workshops, users would attempt to transfer large files over the wireless network. To further exasperate the situation, they would all attempt to do so at the same time. A short time later, tickets would begin trickling in as the workshop participants reported low bandwidth and an unresponsive DE. This had less to do with the DE more to do with the wireless router and low bandwidth in the conference room. Today it is standard practice to ask users to benchmark their own network when they observe slow transfer rates. While there could be several causes, more times than not, the performance degradation is due to a network condition close to the client.

Despite the challenges of the first two years, the iPlant Data Store has been a valuable and productive core piece of the iPlant CI. The next section describes the next-level abstraction to the Data Store, the Foundation API Data Services. The Data Services build upon the Data Store and provide a set of value-adding web services that developers can use to build the next generation of science gateway tools and applications.

III. The iPlant Foundation API Data Service

The iPlant Foundation API (fAPI) is a set of RESTful web services that collectively provide multi-tenant Software-as-a-Service (SaaS) infrastructure for the plant biology community. It is a biology-focused implementation of the AGAVE API, which provides an overlapping set of services for the general science community. The fAPI provides identity, data and metadata management, application registration and execution, resource discovery and registration, monitoring, analytics, and several services designed to make collaboration easy both in high and low trust situations.

The Foundation API Data Services are a subset of the fAPI dealing specifically with metadata, data management, and data transformation. While the previously discussed Data Store provides core data and metadata functionality, the Foundation API Data Services expand that functionality by providing interfaces that allow users to schedule data transfers using multiple transfer protocols, transform data between formats, perform advanced metadata management using both structured and unstructured data, and extend the Data Store by registering additional storage resources to form private storage grids. Four services make up the Foundation Data Services: IO, Data, Systems, and Meta. Each is described in turn below.

A. The Foundation API IO Service

The IO service supports basic CRUD file operations as well as partial data queries in the form of range requests [14], multi-protocol data staging, fine-grained ACL, and file preprocessing. Given the RESTful nature of Foundation, the IO service supports both synchronous and asynchronous data movement operations. This allows users to interactively upload and download files in part or in whole, as well as "fire and forget" their transfers when data sizes are too large or offline movement is desired. In the case of asynchronous requests the notification support built into IO allows users to receive notifications when their transfers are done. The IO service currently supports email notifications and webhooks. A webhook is simply a URL provided by the users to which the service will send an HTTP for [15]. To provide a higher degree of flexibility, the IO service supports a handful of macros such as INPUT_NAME, OUTPUT_URL, TRANSFER_ID, and TOTAL_RETRIES such that users can customize the data sent their URL.

B. The Foundation API Data Service

Data: The Data service acts as a Rosetta Stone to transform data from one format to another using a series of predefined filters. The file formats the Data service supports as well as the file formats to which a particular format can be translated are discoverable through the service. Thus, it is possible to conceptually view the data types as a graph and chain together multiple transforms to obtain the desired output format. While it is not always possible to achieve lossless transforms, in many cases this is not necessarily a requirement, or even a desired feature. An example would be requesting an MP3 copy of a WAV file or a PNG image in JPEG format. As with the IO service, the Data service supports both synchronous and asynchronous transforms, notifications, and partial data queries.

Both the IO and Data services leverage an elastic master-worker design pattern. Two instances of the service interfaces sit behind a load balancer and handle all the interactive requests. Asynchronous requests are passed on to the transform and transfer queues for processing. The queues are implemented using the Quartz Enterprise Job Scheduler. Multiple transfer and transform worker instances run in the cloud on individual VM. Each worker watches the relevant queue and claims a task when one comes available. As load increases, more workers can be instantiated as new VM are spun up. When load decreases, the workers and VM can be torn down as needed. If a worker crashes or becomes unresponsive, another worker will claim the missing worker's task and attempt to rerun the task.

Fig. 1 illustrates the flow of data through the IO service into the Data Store. In this situation a user requests data from an external source be staged into the Data Store and transformed into a specific format before being saved. The transfer queue handles the physical movement of the data from one location to

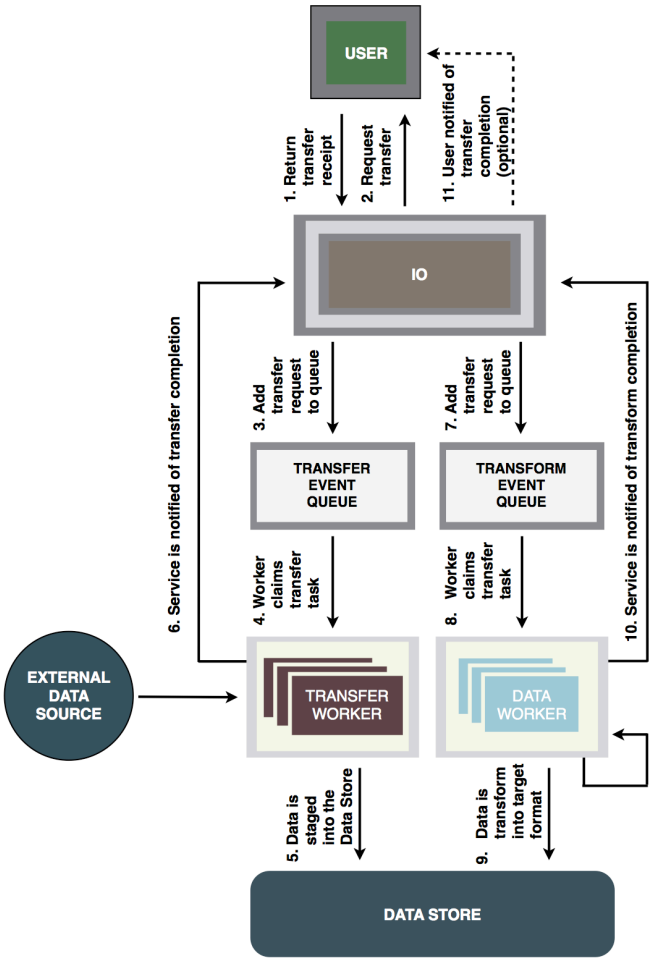

Figure 1. The flow of data into the iPlant Data Store via the Foundation API Data Services.

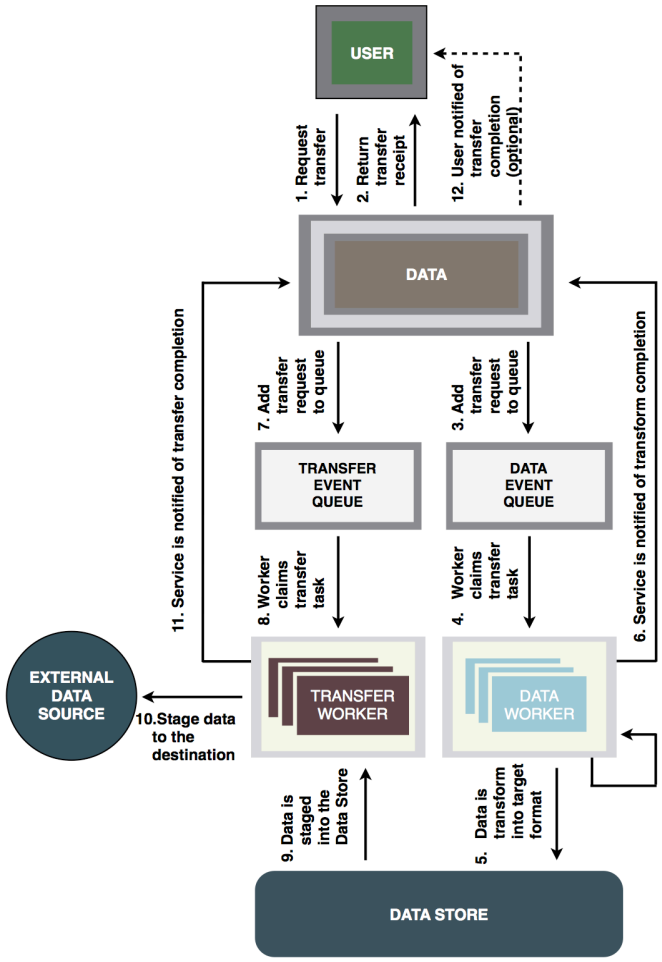

Figure 2. Flow of data from the iPlant Data Store through the Data Service and to the user.

another. Once the raw data is present in the Data Store, it is then iteratively operated on by a predefined set of filters until it reaches its final state. At that point the resulting file is copied to the target destination and the user is notified that their import is complete.

Fig. 2 illustrates the flow of data from the Data Store, through the Data Service, and to an external home. In this situation the data is already stored within the iPlant Data Store and needs to be converted into another form prior to being sent to the target destination.

C. *The Foundation API Systems Service*

The Systems service allows users to discover systems available for use. There is a well-defined taxonomy of supported system types ranging from HPC to Amazon S3. In the context of this paper, storage resources are the primary focus. Some systems, such as the iPlant Data Store will be publicly provided to all users. Others systems are defined by individual users and shared among a small subset of users. In the latter case, these resources are registered by the user with the Systems service and shared in the same way files and jobs are shared. System registration is little more than specifying basic descriptive information such as name, hostname, default path, etc. and attributing one or more connection profiles with it. Connection profiles can be short-term delegated X.509 credentials, OAuth tokens, or temporary logon credentials. We currently discourage, but do not forbid, the use of usernames and passwords in connection profiles. Once registered, storage systems can be used in the same way as any other data system through IO and Data. In this way, it is possible to leverage the fAPI as a hosted data service for local enterprises.

D. *The Foundation API Meta Service*

The Meta service provides metadata support for all of the Foundation services. It exists as an unstructured data store backed by a column-oriented NoSQL database[16]. Because metadata is, by definition, data about data, it has no inherent meaning. Thus, the Meta service allows users to register schemas that describe their data. Schemas are json or xml descriptions of data. These descriptions may be a series of tuples, or highly structured object definitions. The decision on how to describe the data is left up to the user. In order to bridge the gap between different schemas, users can register schema mappings. These mappings allow the Meta service to identify opportunities to do apples-to-apples comparisons between

metadata associated with different schemas. This is helpful when doing global searches, constructing user-derived responses, and when using metadata to reconcile different raw data sets.

One of the primary challenges providing a web service API to the Data Store is that the data is frequently changing independently of the Data Services. This can create situations that compromise data integrity. One example being when a user overwrites an HDF file from the command line with binary data. The file name and size may be unchanged, but the contents and all related metadata are no longer consistent with their states prior to the overwrite. This can cause significant problems and introduce unforeseen points of failure when automating processes. It also impacts the design of a provenance solution. The point made in the section on the Data Store with the challenge of providing data views applies here as well. If data will change without the service knowing about it, then provenance must either be abandoned or implemented at a much lower level in such a way that no change will be missed. In this case, provenance was maintained by pushing it down into the Data Store where raw actions could be detected regardless of access mechanism.

A second problem caused by the Data Store being decoupled from the Data Services is that race conditions can occur without the Data Services being aware. It is not uncommon for users to work simultaneously in the browser and on the command line at the same time. In this situation, two different clients can potentially modify a single piece of data simultaneously. When this happens, the process that finishes first loses because the second process will overwrite its data.

The third problem caused by the Data Store being decoupled from the Data Services is compromised access control states. This is one of the most difficult problems to address. This situation can occur when access permissions change during the execution of a file operation. An example would be user A revoking the read permissions of user B on a file while user B was copying that file. Another example would be user A revoking write permissions to a folder after user B submits a job that will archive its data to that folder.

Lastly, the fourth problem caused by the Data Store being decoupled from the Data Services is the opportunity for flight conditions to occur when a file is physically transferring to a server, but appears to already be present in the Data Store. This is an artifact of the underlying iRODS system. Because the Data Store is configured as mirrored iRODS instances, a file will appear in the catalog as soon as it is physically present on a server. However when that file is requested, it may still be in process of mirroring between servers, thus it may not physically be present on the server closest to the user requesting the file. This could result in corrupted data being returned to the user.

In production operation, any of these situations would cause the Data Services to appear unstable, unreliable, or unusable. Picking and choosing which problem to address in the initial release and which to address in later versions was not an option. All of these problems had to addressed in order to provide a reliable service to users. This was accomplished this through a hybrid approach of creative service side compromises and a minor re-architecting of the underlying Data Store.

The problem of data integrity was addressed at several levels. First, data was categorized according to its potential affect on provenance. Data that must be preserved for reproducibility, archiving, or community use is replicated, restricted to read-only access, and the checksum is stored. Whenever it is requested via the Data Services, the checksum of the physical file is checked against its stored value. If the two differ, administrators are notified, the backup is verified and restored, and the operation continues. This is far too expensive to do for every file and, as such, is reserved for data requiring high assurance. For less critical data, simple date, size, and checksum support is available upon request from the IO service. In the remaining cases, it is now a matter of policy to inform users of what is colloquially known as the Stan Lee principle. Users are taught, "With great power comes great responsibility," and encouraged to act appropriately [17].

When considering solutions to race conditions, one must consider if race conditions should be eliminated completely, or just avoided in certain situation. There are legitimate situations where it is desirable to write to a single file from multiple locations simultaneously. One example being a parallel application writing output data to a single file. To this end, write-locking is enforced at the file system level, while overwriting race conditions are addressed through policy and user alerts.

Solving the problem of compromised access control states required a change in the original Data Services architecture. In the original IO service implementation, all permissions were managed at the service layer and propagated to the underlying Data Store, however it quickly became apparent that people would modify permissions from the command line just as frequently as from the service layer, thus permission management was moved out of the fAPI Data Services and into the underlying iRODS permission model. This was not the most performant approach, and it introduced its own complexities, but it allowed for consistency across access mechanisms and provided a better user experience.

Flight conditions cannot currently be detected by iRODS. Thus, they are avoided by changing the architecture of the Data Store while iPlant engages the iRODS development team to support this feature. Previously, the Data Store mirrored the data from the primary server in Arizona to the secondary server at TACC. When the file system was queried, all traffic went to the primary server, while data requests were served from the instance geographically closest to the user. This configuration was modified such that all data is now served from the primary server. When the primary fails, it is taken completely offline and the secondary server at TACC becomes the primary serving all requests. This change guarantees that a valid copy of the data is always served and only the most accurate information is returned when querying the Data Store.

Despite the challenges, the Data Services remain a heavily leveraged tool by applications built on top of iPlant. Their core features extend and add value to the underlying Data Store and enable the construction of modern applications without thinking about the underlying architecture, network, security,

and storage protocols traditionally associated with leveraging distributed data resources. The remainder of this paper describes the adoption of the Data Store and Data Services and concludes with a roadmap to future development.

IV. Usage And Adoption

At the time of this writing, the iPlant Data Store is growing by nearly 15TB monthly. This figure has increased over time and is outpacing the initial growth projections. As discussed in the Data Store section, usage is predictably distributed across client tools and sorted proportional to file size. Large files, usually exceeding 200MB, are usually transferred using command line tools shipped by iPlant. These tools wrap the native i-commands and provide iPlant-specific capabilities such as metadata registration and permission management in addition to the high speed parallel file transfers and familiar scp-like syntax. Small and medium sized files are usually transferred using a web interface or other client software. In these instances, the network and transport protocol are sufficient to move the data in a reasonable amount of time. It has been our observation that users prefer the convenience of the drag-and-drop interfaces over the command line tools when 1) the transfer is possible via the GUI tool and 2) the performance difference is not significant.

Usage of the Foundation Data Services is largely driven by the latter use case. Over the past year, the fAPI has grown to roughly 50K requests a month. The fAPI Data Services comprised nearly 1/3 of that usage and have moved tens of terabytes of data. The Meta service is still in beta testing at the time of this writing. Initial analysis indicate that the service will launch with more than a quarter billion pieces of information and potentially grow at a rate 10x faster than the raw data. This will be an area of intense observation and research going forward.

V. Future Work

Lessons learned from the first year of operation along with extensive user engagement and support have helped set the roadmap for the coming year. The first priority is solidifying the metadata service and ensuring the schema and map support is sufficient for the user community. Through engagement of a select group of early adopters the project will continue to ensure that the service meets the demands of real world use cases.

Security has also been a challenge in supporting private storage grids as a hosted service. The fAPI utilizes an OAuth2 inspired token service to enable passwordless access to the API. This has been a good solution for usage entirely within the iPlant ecosystem, however for user-defined resources, this had some drawbacks. Several users with existing infrastructures and identity management solutions preferred to authenticate against their own internal mechanisms rather than expose their user credentials to outside sources. This is a well-defined credential delegation scenario already provided by InCommon[18], the MyProxy For OAuth project[19], and the CILogon project [20]. The CAS + OAuth2 solution described in the Data Store section will facilitate this functionality by allowing for both client and user authentication.

Another area of research is the exploration of ways to make partial data queries more intuitive and effective. Currently the fAPI supports byte range queries on raw and derived data. The development team will look into supporting context-aware queries such that ranges could be specified in terms of things other than bytes. For example, when requesting a FAST-A file, the range query could specify the first 4 sequences rather than the first N bytes.

Lastly, planning has recently started for the design of a data mining and machine learning service across all of iPlant in an effort to provide better recommendations, monitoring, and predictive resource scaling. This will be an ongoing effort and an area of long-term research.

VI. Conclusion

This paper described a cloud-based data storage solution called the iPlant Data Store and a hosted, SaaS solution for programmatic access to the Data Store called the iPlant Foundation API Data Services. We discussed in detail several of the challenges faced in providing this service as well as our current solutions in addressing them. We concluded with a discussion of future plans. For more information on the iPlant Foundation API Data Services, please visit our website at http://foundation.iplantcollaborative.org, browse our forums at http://foundation.iplantcollaborative.org/forums, or consult our documentation at http://foundation.iplantcollaborative.org/docs.

Acknowledgment

The iPlant Collaborative is funded by a grant from the National Science Foundation Plant Cyberinfrastructure Program (#DBI-0735191). This work was also partially supported by a grant from the National Science Foundation Cybersecurity Program (#1127210).

References

[1] Stanzione, Dan, "The iPlant Collaborative: Cyberinfrastructure to Feed the World," IEEE Computer, November 2011. doi: 10.1109/MC.2011.297.

[2] Filesystem in Userspace. http://fuse.sourceforge.net.

[3] Rajasekar A., R. Moore, C-Y. Hou, Christopher L., R. Marciano, A. de Torcy, M. Wan, W. Schroeder, S-Y. Chen, L. Gilbert, P. Tooby, and B. Zhu, "iRODS Primer: integrated Rule-Oriented Data Systems", ISBN: 9781608453337, Morgan-Claypool Publishers, San Rafael, CA., January 2010.

[4] Andrew Lenards, Nirav Merchant, and Dan Stanzione. 2011. Building an environment to facilitate discoveries for plant sciences. In Proceedings of the 2011 ACM workshop on Gateway computing environments (GCE '11). ACM, New York, NY, USA, 51-58. DOI=10.1145/2110486.2110494 http://doi.acm.org/10.1145/2110486.2110494.

[5] Australian Research Collaboration Service; Davis: A Generic Interface for SRB and iRODS, www.dhpc.adelaide.edu.au/reports/197/dhpc-197.pdf.

[6] Conway, M. iRODS iDrop. https://code.renci.org/gf/project/iRODSidrop/, 2012.

[7] "iPlant Atmosphere: A Gateway to Cloud Infrastructure for the Plant Sciences." Skidmore E, Kim SJ, Kuchimanchi, S Singaram S, Merchant N, Stanzione D. Proceedings from Gateway Computing Environments 2011 at Supercomputing11 (2011).

[8] M. Szeredi. File System in User Space. http://fuse.sourceforge.net, 2006.

[9] Rajasekar A. iRODS i-commands. https://www.iRODS.org/index.php/icommands, 2012.

[10] H. Naito and S. Kajita. CAS as an institutional-wide authentication and authorization infrastructure, http://www.jasig.org/sites/isapps/jasig/2005SummerBaltimore/. 2005 Summer uPortal Conference, Baltimore, USA.

[11] Shibboleth Architecture Technical Overview, http://shibboleth.internet2.edu/docs/draft-mace-shibboleth-tech-overview-latest.pdf.

[12] Vassiliki Koutsonikola, Athena Vakali, "LDAP: Framework, Practices, and Trends," IEEE Internet Computing, vol. 8, no. 5, pp. 66-72, Sept.-Oct. 2004, doi:10.1109/MIC.2004.44

[13] The OAuth 2.0 Authorization Framework. http://tools.ietf.org/html/draft-ietf-oauth-v2-31, 2012.

[14] Schemers, R. Unified Login With Pluggable Authentication Modules (PAM). http://www.kernel.org/pub/linux/libs/pam/pre/doc/rfc86.0.txt.gz, Oct. 1995.

[15] Fitz, Timothy. "What Webhooks Are And Why You Should Care." http://timothyfitz.wordpress.com/2009/02/09/what-webhooks-are-and-why-you-should-care/. Feb. 9, 2009.

[16] Rick Cattell. 2011. Scalable SQL and NoSQL data stores. SIGMOD Rec. 39, 4 (May 2011), 12-27. DOI=10.1145/1978915.1978919 http://doi.acm.org/10.1145/1978915.1978919

[17] "Spider-Man." Quotes.net. STANDS4 LLC, 2012. 11 September. 2012. http://www.quotes.net/quote/8628.

[18] InCommon Federation. A Roadmap for Using NSF Cyberinfrastructure with InCommon. {Accessed 10 Sept 2012}; Available from http://www.incommonfederation.org/cyberroadmap.html.

[19] Jim Basney, Rion Dooley, Jeff Gaynor, Suresh Marru, and Marlon Pierce. 2011. Distributed web security for science gateways. In Proceedings of the 2011 ACM workshop on Gateway computing environments (GCE '11). ACM, New York, NY, USA, 13-20. DOI=10.1145/2110486.2110489 http://doi.acm.org/10.1145/2110486.2110489.

[20] CILogon Service. http://www.cilogon.org/service