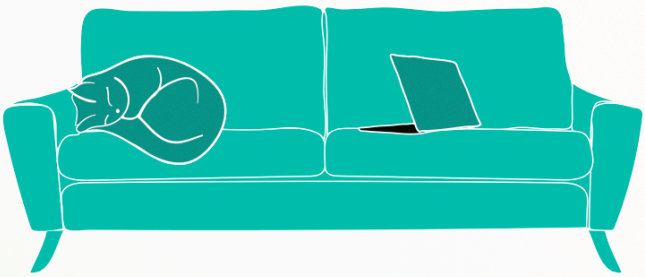
iRODS®

VIRTUAL

USER GROUP MEETING

2021



USER GROUP MEETING
2021 PROCEEDINGS

PUBLISHED BY THE iRODS CONSORTIUM

iRODS
User Group Meeting 2021
Proceedings

13TH ANNUAL CONFERENCE SUMMARY

The Virtual iRODS User Group Meeting of 2021 gathered together iRODS users, Consortium members, and staff to discuss iRODS-enabled applications and discoveries, technologies developed around iRODS, and future development and sustainability of iRODS and the iRODS Consortium.

The virtual four-day event was held from June 8th to 11th, hosted by the Wellcome Sanger Institute and the iRODS Consortium, with 274 people attending from 21 countries. Attendees and presenters represented 133 academic, governmental, and commercial institutions.

TALKS AND PAPERS

LIGHTNING TALKS

Log centralisation with rsyslog and the elasticstack and how Sanger use it to identify issues before they are reported

Brett Hartley – Wellcome Sanger Institute

ii: command line utilities for iRODS

Sietse Snel – Utrecht University

iRODS Parallel Transfer Between Python Client and S3 Storage

Justin James, Daniel Moore – iRODS Consortium

NFSRODS deployment and performance tuning at Sanger

John Constable – Wellcome Sanger Institute

Golang iRODS Web Frontend

Peter Verraedt – KU Leuven

Just Stand It Up

Kory Draughn – iRODS Consortium

Towards a Cloud Native iRODS

Jason Coposky – iRODS Consortium

CLOSING REMARKS

Peter Clapham – Wellcome Sanger Institute

iRODS Logical Locking

Alan King
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
alanking@renci.org

Terrell Russell
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
unc@terrellrussell.com

Kory Draughn
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
korydraughn@renci.org

Jason Coposky
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
jasonc@renci.org

ABSTRACT

iRODS 4.2.9 introduces Logical Locking by providing additional replica status values within the catalog. Previously, replicas in iRODS could only be marked 'Good' or 'Stale'. This did not capture the states of when data was in flight, or incomplete. This paper will explain the new Intermediate and Write-Locked states for iRODS replicas and how they are used to provide protection from uncoordinated writes into the system.

Keywords

iRODS, data management, logical locking, concurrency

INTRODUCTION

The iRODS Technical Overview[1] states:

> iRODS provides a logical representation of files stored in physical storage locations. We call this logical view a virtual file system and the capabilities it provides, Data Virtualization.

Interactions with data in an iRODS system are meant to mirror how one might interact with data in a traditional hierarchical file system. The purpose of the virtual file system is to present data across a number of geographically distributed servers under a unified namespace - in other words, a distributed system. A distributed system implies operational concurrency and operational concurrency implies race conditions.

The following document will describe how iRODS interacts with data, how the iRODS Consortium seeks to address concurrency in manipulation of data within an iRODS system, and planned efforts for protecting data access and manipulation as well as policy execution.

BACKGROUND

History of data representation and movement

The logical representation of data presented by the virtual file system is known as a Data Object, which the iRODS Beginner Training[2] defines as:

iRODS UGM 2021 June 8-11, 2021, Virtual

Data Object: a logical representation of data that maps to one or more physical instances (**Replicas**) of the data at rest in Storage Resources

A Replica could have one of two statuses:

- **Stale**: The replica is no longer known to be Good. Usually indicates that a more recently written-to replica exists. This does not necessarily mean that the data is incorrect but merely that it may not reflect the "truth" for this Data Object.
- **Good**: The replica is Good. Usually the latest replica to be written. When a replica is Good, all bytes and system metadata (size and checksum, if present) are understood to have been recorded correctly.

iRODS has historically presented an interface which interacts with data at the logical level - that is, with Data Objects. However, over time, it became clear that interacting directly with replicas is very important for iRODS users and administrators. Other operations were added to allow for replica manipulation which were very difficult to integrate with existing APIs.

As such, operations which deal directly with replicas have completely separate implementations for moving data from the logical level operations. Furthermore, operations dealing with Data Objects still need access to replica information (after all, the operations interact with the storage eventually, at which point it is interacting at the replica level).

On top of all of this, because replica status can only be Good or Stale, any replica which exists in the catalog but has not been finalized is being represented incorrectly because the data is still in flight. Good and Stale refer to data which is at rest. There is no status to represent in-flight data.

Unifying data movement with replicas as First Class Citizens

These differences in implementation have consistency and performance implications for moving data. In reality, all of these data moving operations should be (and are) identical:

open replica –> move data to replica –> close replica –> finalize system metadata in catalog

Every data moving operation in iRODS should implement the above steps. These should use the same set of low-level interfaces so that data movement is made consistent across operations. To make this possible, replicas must be treated as the operative entity in data movement, not Data Objects (which are one-to-many mappings to replicas). This requires that replicas be treated as First Class Citizens within the iRODS server where, historically, they have not - at least not in all instances.

Efforts are being made to change this in the server, but it has not yet been fully realized. We will take a slight detour to describe in detail one of the most important operations within an iRODS system: Replication. First it will be defined at an abstract level as it is intended to function, and then the implementation and interface in iRODS will be described. This highlights the importance of a unified mode of data movement within the system even while the requirements for the operation differ so greatly from others such as put, copy, and rename.

Rules For Replication

Replication refers to copying a physical replica of an existing Data Object from one storage resource to another. Replication requires 3 inputs:

1. **Logical path**: Path to an existing Data Object with at least one replica which is at rest and can be read.

case	source	destination	result	reason
0	-	-	unchanged	No source replica exists
1	-	&	unchanged	No source replica exists
2	-	X	unchanged	No source replica exists
3	&	-	&	Replication allowed
4	&	&	unchanged	Destination replica must be Stale
5	&	X	&	Replication allowed
6	X	-	X	Replication allowed
7	X	&	unchanged	Destination replica must be Stale
8	X	X	unchanged	Source replica must be Good

Table 1. This table shows the nine cases of initial state of the replicas on the source and destination resources, the resulting replica state on the destination resource, and the reason for the result.

2. **Source resource**: The resource from which data will be copied/read.

3. **Destination resource**: The resource to which data will be copied/written.

The source resource must already have an existing, at-rest replica which can be read in order for replication to be *possible* in any case.

If the destination resource has no replica, replication to the destination resource is *allowed* in **all cases**. If this is true, a **new replica** will appear on the destination resource as a result of the replication. If this is not true, the destination resource has an existing replica and replication may or may not occur.

If the destination resource has an existing replica, replication would be performing a **replica update**. The following requirements must be true in order for updating the replica to be *allowed*:

1. The destination resource must not be the source resource

2. The destination replica must be Stale

3. The source replica must be Good

Table 1 shows all possible cases for replication between a single source and destination when the source and destination replicas do or do not exist. Some notes about the table:

- '&' represents a Good replica
- 'X' represents a Stale replica
- '-' indicates a replica which does not exist.
- All replicas are assumed to be at rest

REPLICATION MECHANISM

The iRODS server provides the `rsDataObjRepl` API in order to replicate Data Objects. Here is the signature:

```
int rsDataObjRepl(rsComm_t*, dataObjInp_t*, transferStat_t**);
```

As described in the Rules for Replication subsection above, replication requires a logical path, a source resource, and a destination resource. The following describes how these requirements can be satisfied.

Logical Path

As described in the general Rules for Replication subsection, the logical path must refer to an existing iRODS Data Object of which the to-be-determined-or-specified source replica must exist. The authenticated user must have at least write permission on the Data Object in order to read from the source replica and write to the destination replica. The other qualifications regarding source and destination replica statuses have already been described in Rules for Replication.

Source Resource

In order to specify a source resource, any of the following may be provided via `condInput`:

1. `RESC_NAME_KW` - Must be a root resource. If this is not true, the following error is returned: `DIRECT_CHILD_ACCESS`.
2. `RESC_HIER_STR_KW` - Must be a full resource hierarchy.
3. `REPL_NUM_KW` - Must be an integer representing the existing replica.

If none of the above is provided, any resource is eligible for use as the source resource.

Resource hierarchy resolution is then performed on a `OPEN` operation by the API to determine the full resource hierarchy. The following must be true of the resolved resource hierarchy (depending on which of the source resource inputs was provided):

1. The resolved resource hierarchy has a root which matches `RESC_NAME_KW`
2. No resource hierarchy resolution is performed.
3. The resolved resource hierarchy has an existing replica with a replica number which matches `REPL_NUM_KW`

If the resolved resource hierarchy does not meet the requirements of the provided inputs or the requirements of the source replica as defined by the Rules of Replication, the following error is returned: `SYS_REPLICA_INACCESSIBLE`.

Destination Resource

In order to specify a destination resource, any of the following may be provided via `condInput`:

1. `DEST_RESC_NAME_KW` - Must be a root resource. If this is not true, the error `DIRECT_CHILD_ACCESS` is returned.
2. `DEST_RESC_HIER_STR_KW` - Must be a full resource hierarchy.

If neither of the above is provided, the system will look for a `DEF_RESC_NAME_KW` (the default resource). If present, `DEST_RESC_NAME_KW` is set to this value. If not, any resource is eligible for use as the destination resource.

Resource hierarchy resolution is then performed on a `CREATE` operation by the API to determine the full resource hierarchy. The following must be true of the resolved resource hierarchy (depending on which of the destination resource inputs was provided):

1. The resolved resource hierarchy has a root which matches `DEST_RESC_NAME_KW`
2. No resource hierarchy resolution is performed.

If the resolved resource hierarchy does not meet the requirements of the provided inputs, the following error is returned: `SYS_REPLICA_INACCESSIBLE`.

If the destination resource has a replica which does not meet the requirements for a destination replica as defined in the Rules of Replication, the following error is returned: `SYS_NOT_ALLOWED`.

Updating all replicas

When the `ALL_KW` is provided, the API will gather a list of all existing replicas and update each one, serially at this time, according to all of the rules described above. Specifying a destination resource along with the `ALL_KW` is not allowed.

PROBLEMS WITH UNCOORDINATED, CONCURRENT OPERATIONS

Once data movement is unified across the iRODS server, the conversation about concurrency as it relates to system correctness can happen. We have identified three ways in which the state of the system can be made incorrect:

1. Uncoordinated, concurrent writing to a single replica can lead to **Data Corruption**.
2. Uncoordinated, concurrent writing to multiple replicas of the same Data Object can lead to **Truth Corruption**.
3. Uncoordinated, concurrent operation execution can lead to **Policy Violations**.

The iRODS Consortium is actively working to address these three situations in the server. The first two have been partially addressed in released server software and the third is to be addressed in a future release.

Data Corruption and Intermediate Replicas

The first problem listed above is that in-flight replicas can be opened and modified concurrently by multiple agents in an uncoordinated fashion. Also, as a result of iRODS historically creating replicas in the Good status, the catalog does not reflect the current, true state of the data from the moment it is registered in the catalog.

To address this, a third replica status has been introduced:

Intermediate: The replica is actively being written to. The state of the replica cannot be determined because the client writing to it still has the replica opened. Replicas which are marked Intermediate cannot be opened for read or write, nor can they be unlinked or renamed.

Replicas are created in the Intermediate status for data moving operations because the replica is considered in-flight until the time that it is finalized in the catalog. In this way, the replica status is accurately represented in the catalog and the system is able to respond appropriately to concurrent access attempts to this replica.

Truth Corruption and Logical Locking

However, it is unclear which replica for a given Data Object represents the Truth when multiple replicas are in flight at the same time. Protecting individual replicas is not enough to verify the correctness of the data at the logical level.

To address this, a new mechanism called iRODS Logical Locking (ILL) has been introduced which disallows opening any replica for a given Data Object when any one of the replicas are opened for write. This has been implemented using another new (fourth) replica status:

Write-Locked: One of this replica's sibling replicas is actively being written to but it is, itself, at rest. Replicas which are Write-Locked cannot be opened for read or write, nor can they be unlinked or renamed.

By only allowing coordinated modifications to a Data Object, assertions about the correctness of the logical Data Object can be made based upon the correctness of a replica because multiple replicas cannot simultaneously claim to be correct (while potentially holding different data).

Policy Violations and Operation Locking

We have not yet addressed the third problem of uncoordinated, concurrent policy invocations in iRODS. This is addressed in the Future Work section later in this paper.

SOLUTION - DESIGN

iRODS Logical Locking operates on the basic principles of a traditional read-write lock in computing[3]

1. The Data Object is locked when any of its replicas are opened.
2. The Data Object is unlocked when the opened replica is closed/finalized.

Here, we describe this process in more detail:

The Data Object is locked when any of its replicas are opened

1. Client requests to open a replica of a given Data Object.
2. iRODS checks the status of the replica to make sure it is not already locked (equivalent to trying to "acquire" the lock). "Locked" in this case means:
   (a) the replica is Read-Locked (this or a sibling replica is opened for read by an agent): the operation will fail if it is not an open-for-read.
   (b) the replica is Write-Locked (a sibling replica is Intermediate): the operation will fail.
   (c) the replica is Intermediate: the operation will fail for an uncoordinated open-for-write or any open-for-read. See the Implementation section below for details about concurrent write coordination.
   (d) If the open intends to create a new replica, the operation will fail if any replica is found not to be at rest (that is, Intermediate, Read-Locked, or Write-Locked).
3. Depending on the operation, the Data Object is then locked for this open. "Locked" in this case means:
   (a) **open-for-read**: the status of all replicas is set to read-lock
   (b) **open-for-write**: the status of the target replica is set to Intermediate and the statuses of the sibling replicas are set to Write-Locked
   (c) **open-for-create**: the statuses of any existing sibling replicas are set to Write-Locked
4. The target replica is then physically opened and the open is complete. If a new replica is supposed to have been created, the entry is created in the catalog in the Intermediate status.

As we will see later, there is still a race condition here: the TOCTOU[4] condition in the iRODS catalog itself. We have designed a solution for this which will be described later, but we mention it here to acknowledge its existence.

The Data Object is unlocked when the opened replica is closed/finalized

1. The client requests to close and finalize a replica of a given Data Object.
2. The Data Object is unlocked atomically along with updating the other system metadata. The final states of the replicas depends mainly on the operation requesting the close. The general cases for unlocking are shown here, but some operations (e.g. replication, phymv, etc.) may behave differently as they have different requirements:
   (a) **open-for-read**: if no other agents have any of the replicas opened, the replica statuses of each replica is restored to what it was before the open
   (b) **open-for-write**: there are two cases:
      i. success: the target replica is marked Good and all sibling replicas are marked Stale
      ii. failure: the target replica is marked Stale and all sibling replicas have their statuses restored to what they were before the open

SOLUTION - IMPLEMENTATION

Code Facilities

Two mechanisms were developed to implement Logical Locking:

1. `data_object_finalize` API plugin (finalize) - atomically applies updates to (multiple) rows in `R_DATA_MAIN`
2. `replica_state_table` (RST) - An in-memory, per-agent JSON structure which describes a snapshot of rows in `R_DATA_MAIN` for a particular data_id representing a Data Object

A library was developed under the `irods::logical_locking` namespace in the server on top of these utilities which makes surgical edits to the RST and updates the catalog via the finalize API to atomically update - and therefore unlock - Data Objects. As described in the design, Data Objects are locked on open and unlocked on close/finalize using this `logical_locking` library.

Refactoring iRODS Internals

Much refactoring was necessary in the iRODS server for a correct first attempt at an ILL implementation and to fix critical existing issues, all while maintaining the existing client-facing interfaces. In order to ensure that ILL is enforced across the API surface, the requirements described in "Unifying data movement with replicas as First Class Citizens" needed to be put in place. This accomplishes both the aforementioned consistency and correctness of data movement throughout the system, and a single entry point for using ILL.

A major shift was made internally in the iRODS server away from relying on `rsDataObjClose` to handle the logic of finalizing replicas as has been done historically. Each operation has its own requirements for how the replicas in a Data Object should appear for success or failure cases and in many cases they differ from those of other operations. The finalize API plugin is now used in conjunction with the `replica_close` API plugin to determine and control how replicas should be updated after the completion of an operation. `rsDataObjClose` can still be used to close and finalize a Data Object, but the resultant replica statuses are handled generically.

Coordinated, concurrent write support

To support concurrent writes to the same replica which enables parallel data transfer using multiple iRODS agents, another mechanism was developed called the Replica Access Registry (RAR). The RAR is held in shared memory so that agents spawned from a particular iRODS server can coordinate concurrent writes to particular replicas. When an iRODS agent opens a replica for write (or create) an entry is created in the RAR which associates an agent PID with a replica of a Data Object and is accessed via a universally unique identifier called a Replica Access Token (RAT). This token is part of the L1 descriptor held in the agent process which initiated the replica open. This token allows other iRODS agents connected to the same server to open the same Intermediate replica.

Using these tokens, clients can now implement parallel data transfer to a single replica over the main iRODS port. This has been demonstrated both in the C++ client API with the Parallel Transfer Engine (PTE) and in the Python iRODS Client (PRC)[6] version 1.0.0.

KNOWN LIMITATIONS

As with most features, ILL does not protect against rogue administrators.

Performance

Now that the server is enforcing coordination of writes across the iRODS Zone, this necessarily means more trips to the iRODS Catalog to query for information and modify rows to update replica statuses. These additional round trips to the database will introduce some marginal overhead, but for most data-write operations, they are still dwarfed by the time for actual data movement.

The Database Race

ILL is currently still prone to database race conditions in the iRODS Catalog due to a classic TOCTOU[4] problem which allows multiple winners for the race. The scenario arises when two agents arrive to lock a Data Object at the same time. The lock is first checked and then set if it passes. All at-rest, unopened Data Objects are eligible to be locked. The two agents will both find that the Data Object is unlocked and then go to the database to set the status. The database will not stop this and both agents will think that they have exclusive access to their respectively opened replicas of the Data Object.

We believe this can be solved by causing the check and set of the replica statuses to be a single database transaction. The system will lean upon the locking mechanism of the database and the lock cannot be ”acquired” by more than one agent and so there can be only one winner.

FUTURE WORK

Policy Violation (Operation Locking)

The third problem listed before is violations in configured policy due to uncoordinated, concurrent policy invocations which incur data-modifying operations. We think that this will be solved by operation-aware logical locking, or **Operation Locking**. Where ILL is currently gated on opening and closing replicas of a Data Object, Operation Locking would be gated on the start and completion of a high-level operation.

As described before, open and close are considered ”low-level” interfaces where things like put, replication, and physical move (phymv) are ”high-level” and are built upon the low-level interfaces. Operation Locking would move the locking mechanism up from open into the higher level interfaces so that locking can be controlled across multiple data-modifying operations resulting from policy invocation. In this way, policy invocation will be made more correct and even enable coordinated, concurrent data movement within the policy-triggered operation(s). This will require extending ILL to make iRODS agents more aware of which lock(s) they have acquired.

An example of when this would be useful is when a client puts data into a resource hierarchy which includes a replication resource with multiple child resources. Without Operation Locking, the time between the multiple replication operations is subject to a race condition where other clients could be writing to the same Data Object.

Read Locks

The goal behind ILL is to implement a readers-writer lock[3] on Data Object resources via the replica status in the iRODS Catalog. A read lock will allow for multiple, **uncoordinated** agents to read from any replica in a Data Object at the same time. When a Data Object is Read-Locked, all data-modifying operations should be blocked (open for write, unlink, rename, etc.) Unlike write locks, which require coordination to lock and unlock properly, read locks will allow for uncoordinated reads. This means that ILL will need to be extended to track a list of agent PIDs and

hostnames which are holding the lock. This may imply a refactor of the current lock implementation so that the solution can remain generic and possibly bring it more in line semantically with traditional locks/mutexes.

SUMMARY

After more than two years of planning, development, and testing, iRODS 4.2.9 introduces Logical Locking which protects the iRODS system from uncoordinated, concurrent writes to replicas of a particular Data Object. This is implemented by adding two additional possible replica statuses, Intermediate and Write-Locked.

REFERENCES

[1] iRODS Technical Overview (2016). `https://irods.org/uploads/2016/06/technical-overview-2016-web.pdf`

[2] iRODS Beginner Training (2019). `https://github.com/irods/irods_training/blob/5418c42d7d6c33d92b14acc02f9bbba5c414b779/beginner/irods_beginner_training_2019.pdf`

[3] Lamport, Leslie; *Concurrent Reading and Writing*; Communications of the ACM, November 1977, Vol. 20 No. 11, Pages 806-811 `http://lamport.azurewebsites.net/pubs/rd-wr.pdf`

[4] Time of Check Time of Use `https://cwe.mitre.org/data/definitions/367.html`

[5] Parallel Transfer Engine `need_url`

[6] Python iRODS Client `https://github.com/irods/python-irodsclient`

The Research Data Management System at the University of Groningen (RUG RDMS): architecture, solution engines and challenges

A. Tsyganov, S. Stoica, M. Babai, V. Soancatl-Aguilar, J. Mc Farland, G. Strikwerda, M. Klein, V. Boxelaar, A. Pothaar, C. Marocico, J. van den Buijs
University of Groningen,
Groningen, Netherlands
rdms-support@rug.nl

ABSTRACT

RUG Research Data Management System (RUG RDMS) is powered by iRODS. It is developed at the University of Groningen to store and share data, and allow collaborative research projects. The system is developed based on open-source solutions, providing access to the stored data by means of command line, WebDAV, and a web-based graphical user interface.

The system provides a number of key functionalities and technical solutions such as metadata template management, data policies, data provenance, and activity auditing. It uses some of the existing iRODS functionalities and tools like the iRODS audit plugin to track the data operations or the iRODS rule engine plugin for Python to implement a set of custom developed rules. It allows users to configure and tune metadata templates for different research areas. Furthermore, iRODS advanced rule and composable resource functionalities have been used to automate metadata extraction on demand and allow long term storage on magnetic tapes. In addition, a set of custom system policies that is used for data handling has been developed. This provides a flexible environment on top of the current iRODS rules. Data replication on the storage level guarantees full data recovery.

The system is accessible via the local HPC facilities and provides a landing zone to support data to compute. The architectural design of the system allows both vertical and horizontal scalability with the potential of unifying different iRODS installations and facilities.

Keywords

RUG RDMS, metadata templates, data policies, iRODS Python rule engine plugin, iRODS auditing module.

INTRODUCTION

The general lack of centralized data storage and management within research institutions increases the risk that valuable data disappears or is more difficult to find. Moreover, datasets are increasing in volume, complexity, heterogeneity, and speed of generation [1]. This is why the demand for high quality and intuitive scientific data management tools is high and most of the funding agencies nowadays require a data management plan in an early stage of a research project. As a research output, data are often compared to a public good that should be made available to the community for re-use if possible. There is a growing demand for research data integrity, quality, public availability, and reusability, the FAIR principles [2]. In the context of research data management, the documentation of research data is an essential duty. In order to ensure the interpretability, understandability, shareability, and long-lasting usability of the data, any data management system should support standardized, and interoperable metadata. Effective data management can actually contribute to the increased pace of the research process, contribute to the

soundness of research results, and meet funding agency requirements by making research data easy to manage and share over the long term. A good example of such a system is YoDa, created at Utrecht University, Nederlands [3][4].

The RUG Research Data Management System (RUG RDMS) system has been developed to support the scientific research community at the University of Groningen [5] and University Medical Center Groningen (UMCG) [6] in the process of handling its research data. It provides an inhouse solution for the data storage, data access management, in addition to data management policies.

ARCHITECTURE

The main idea behind the architecture of the RUG RDMS is to split the system into a number of services that do not depend on the implementation. Figure 1 illustrates a number of logical components and the decisions that were made in the design of the RUG RDMS to make it a scalable and robust system. The system has been designed to store vast amounts of data, while keeping everything on-premises. The backend is based on iRODS [7][8], an open-source data management software that supports collaborative research effectively. iRODS is data-grid middleware that virtualizes access to data regardless of which physical device the data is stored on. It accomplishes this by mapping physical files and directories to logical data objects and collections. Users can make use of various interfaces to search for and retrieve data. In addition to facilitating data discovery and retrieval, iRODS has a robust security system to implement fine grained access control and facilitate robust activity auditing. The defining concept of the RUG RDMS is using iRODS' powerful metadata storage and discovery functionality to document the data stored in it.

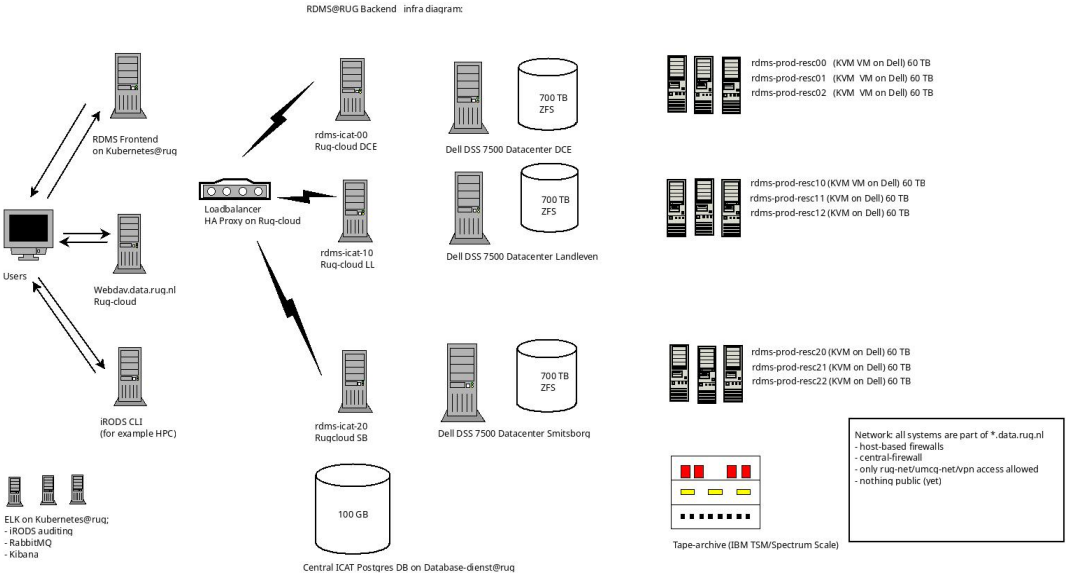


Figure 1. Users interface with the RUG RDMS in one of three ways: the web front-end, a WebDAV connection, or the iRODS CLI. Three iRODS catalog providers access a single PostgreSQL database and sit behind an HAProxy. The nine current resource servers (iRODS consumers) have multiple volumes "mounted" from a local storage server. The RUG RDMS storage is composed of 3 big storage servers located in three different data centers. These servers utilize a ZFS filesystem pool with built-in support for data-deduplication, compression, 'self'-healing, and software RAID among other features.

The RUG RDMS is configured such that it should survive broken data disks (spare and hot swappable), broken volumes, broken servers, one downed data-centre, small scale network outages, etc. Maintenance can also be performed easily, since resource virtual machines are small and can be moved or migrated if needed. There are several iRODS zones with the possibility to add new zones when necessary. Currently, next to the iRODS zones dedicated for the development, the production iRODS zones include one for the majority of the university projects and one for

the collaboration with the UMCG. In addition to the online storage resources, there is a nearline, long-term archive resource utilizing a tape storage system.

APPLICATION DESIGN AND WEB INTERFACE

Illustrated in Figure 2, the whole system is divided into a number of components intended to be independent services with their own setup and configuration:

- Proxy: the main and only entry point for the Web Interface. The proxy service is implemented using Nginx [9], the most popular proxy server.
- Web Server: The RUG RDMS uses Django as a base framework to deliver web content in combination with RUG css templates and Bootstrap 4 libraries. Django [10] is a high-level Python Web framework that encourages rapid and clean development and powers many of the Web's sites.
- Backend: iRODS with the rule engine plugin for Python and custom iRODS rule bases dedicated to the RUG RDMS functionality. Such rules include blocking of select operations on (meta)data, automatic metadata extraction, notification messages, etc.
- REST-like services within a private network: all other services that RUG RDMS uses in the intermediate level are based on REST APIs [11] or have a direct connection to the mid-tier database. This design choice makes internal communication between different modules of the system independent from the particular programming code of a service. For instance, communication with the iRODS backend is done via the Java Jargon API using Jersey REST API [12] and the Apache Tomcat Web server [13]. However, in the near future, there is a plan to switch to Python iRODS client implementation. This transition is especially easy because the implementation of the REST calls from the other parts of the system to iRODS will be the same.
- Davrods [14] and NFSRODS [15]: DavRods and NFSRODS are two implementations of a standard file system protocol to mount iRODS data as a filesystem directly on the client side.
- iCommands [16] access for the users: collectively, the iCommands are a default iRODS tool that is used to manage data on the server and on the client side.

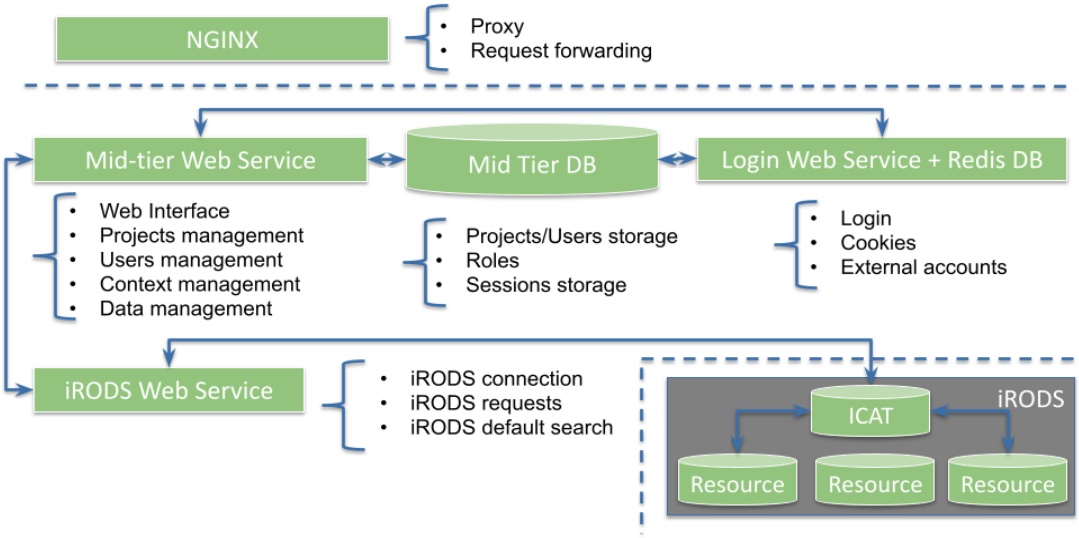


Figure 2. RUG RDMS Logical Architecture

Since the beginning of the project, there have been several versions of the web interface. Our current solution has many similarities to typical cloud storage interfaces' look and feel as shown in Figure 3.

1. The first logical type of the data is in the user's data area. It is a collection in which the user is the sole owner and manager of the data.
2. The second logical type is in the team drive. A team drive is a shared area that can be created and managed by data managers. Each scientific group can have their own data managers. Within RUG RDMS, data managers are users with the iRODS groupadmin role. Users with this role are allowed to create and manage iRODS groups. The idea behind the team drives is to provide users with a collaborative working area with flexible permission management.
3. The third logical data type resides in the project area. It contains all the facilities to fully support the data lifecycle management. Project collections can be created and managed by another special group of users: project managers. When a new project is created, three dedicated user groups are immediately attached to it. These groups have read, write and ownership of the data, respectively. Every user who is a member of a project will be at least a member of the read user group. Users that are considered owners of the data will be added to all three groups. A user that has write access to a project will be included in the write and read groups. This approach allows for fine grained permissions of the inner project data and archived data.

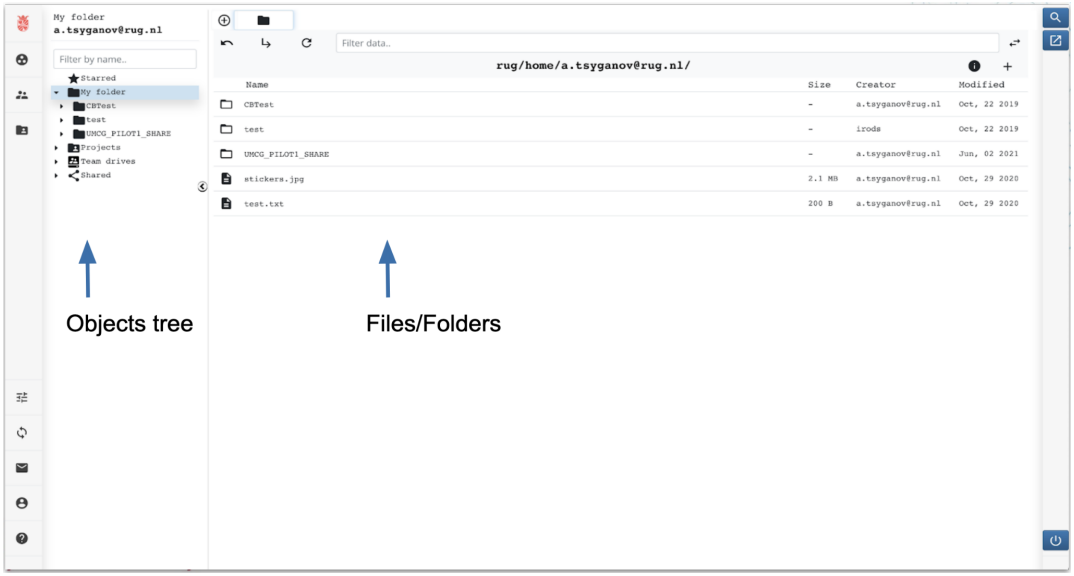


Figure 3. Main user landing page at RUG RDMS with its inbuilt data browser. The left panel shows the directory (collection) browser. The right panel shows the contents of a selected collection: data objects and sub-collections.

The metadata and metadata handling constitute another important aspect of the interface and its data management. RUG RDMS supports standard iRODS metadata operations enriched with custom rules. One such custom rule is automatic metadata extraction from standard data formats such as JPEG, TIFF, FITS, PDF, etc. This rule is triggered manually from the web interface or from the command line. It extracts all available metadata from a physical file having the supported format. With the evolution of the system the list of supported formats will only grow and the expansion of metadata templates will be used to choose what metadata is extracted and how it is stored. The interface workflow for running this metadata extraction is presented in Figure 4.

The automatic metadata extraction runs as a delayed rule on the iRODS backend that is submitted as a background job. While the results of the extraction may sometimes not be directly visible on the web interface, the user can follow the status of their submitted jobs on a dedicated page. Before such a background process starts, the process is first registered in the RUG RDMS mid-tier database. By using the mid-tier REST database service, the custom Python rule reports to the RUG RDMS about the process status.

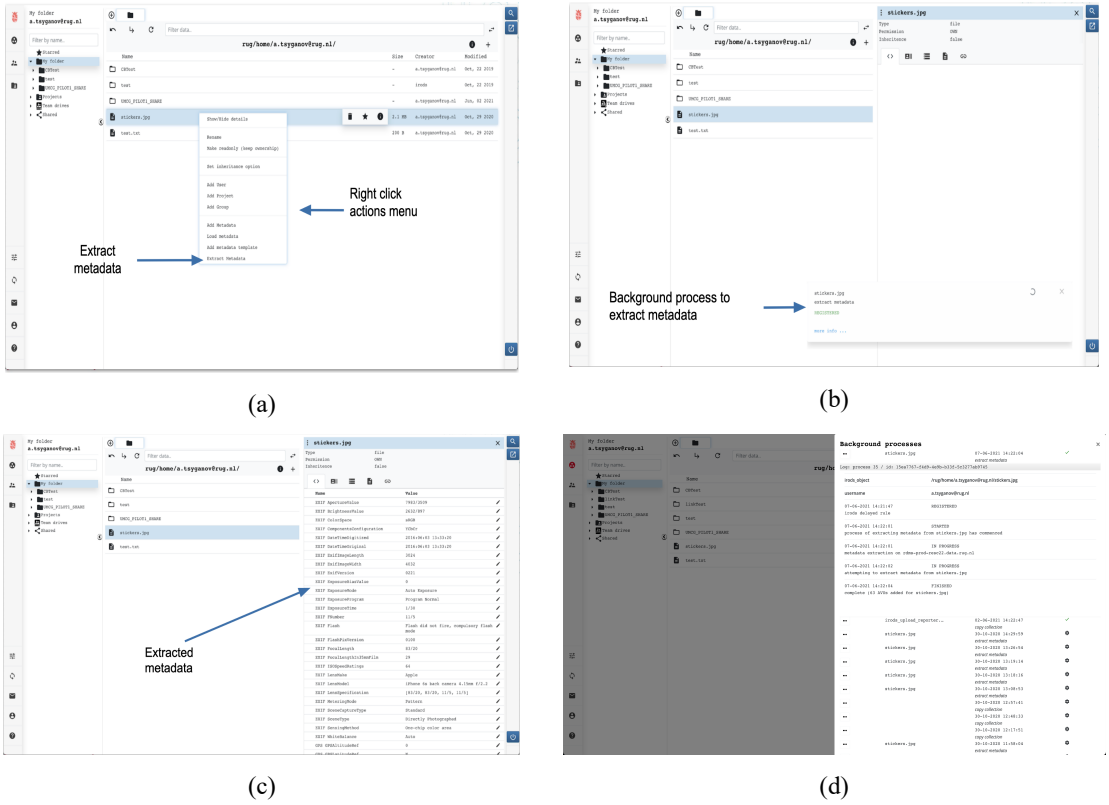


Figure 4. RUG RDMS web interface automatic metadata extraction. The four panels show (a) the right click action menu, (b) a screen with the background extraction process running, (c) the final result of the extracted metadata, and (d) extra information about the background extraction process.

CUSTOM POLICY ENGINE

Implementation of user workflows is one of the major requirements for the RUG RDMS. During the requirements engineering phase, it became clear that different research groups apply their own field-specific data processing, storage formats, and operational algorithms. A custom policy engine has been developed for this purpose. It utilizes the iRODS rule engine plugin for Python in combination with the mid-tier RUG RDMS architecture and iRODS metadata on particular collections and data objects. One needs to add special metadata to an object to initiate a custom policy.

iRODS metadata consists of three values: key, value, unit. Key – to identify metadata; value – to store actual metadata value; unit – an extra field that can be used for different purposes. Below there is an example of the metadata policy attached to the project collection «/testZone/home/Projects/project0_5n1»:

```
Name : sysmdt_rdms_policy_2c7197f0a89e1c842180756537534a81a069be79e8ec6fa1473af21c
Value: {
       "policy_name"      : "project_user_participation_enddate",
       "policy_creator"   : "atsG",
       "input_parameters" :
                    {
                        "user_name"   : "atsG",
                        "end_date"    : "10/06/2021 11:52",
                        "date_format" : "%d/%m/%Y %H:%M"
                    }
       }
Unit : POLICY|PROJECT|TORUN|atsG
```

To make a key-value pair unique, the object name is transformed into a non-reversible hash value and this is added to the key name. In the example above "/testZone/home/Projects/project0_5n1" equals to hash:

```
2c7197f0a89e1c842180756537534a81a069be79e8ec6fa1473af21c
```

The unit field is used to make this policy correctly searchable and executable on the iRODS server side. It also stores the type of the policy, status and the name of the user who initiated the policy.

The value field contains a JSON string with the information about the policy. This string is used to run Python code constructed from the "policy_name" and "input_parameters". Here is a code example to run such a JSON string:

```
from .policy_project_user_participation_enddate import
       run_policy_project_user_participation_enddate

C_AVAILABLE_POLICIES = {
"project_user_participation_enddate" : run_policy_project_user_participation_enddate
}

# method to run policy that was fetched from the metadata of the object
def run_policy(self):
   l_function_name, l_parameters = self.parce_policy()# parse JSON
   if self.policyIsValid():
       if self.c_namespace.C_AVAILABLE_POLICIES.has_key(l_function_name):
           # execute code
           self.c_namespace.C_AVAILABLE_POLICIES[l_function_name](self, l_parameters)
```

After the policy metadata is set it is picked up by the server cron process illustrated in the diagram in Figure 5.

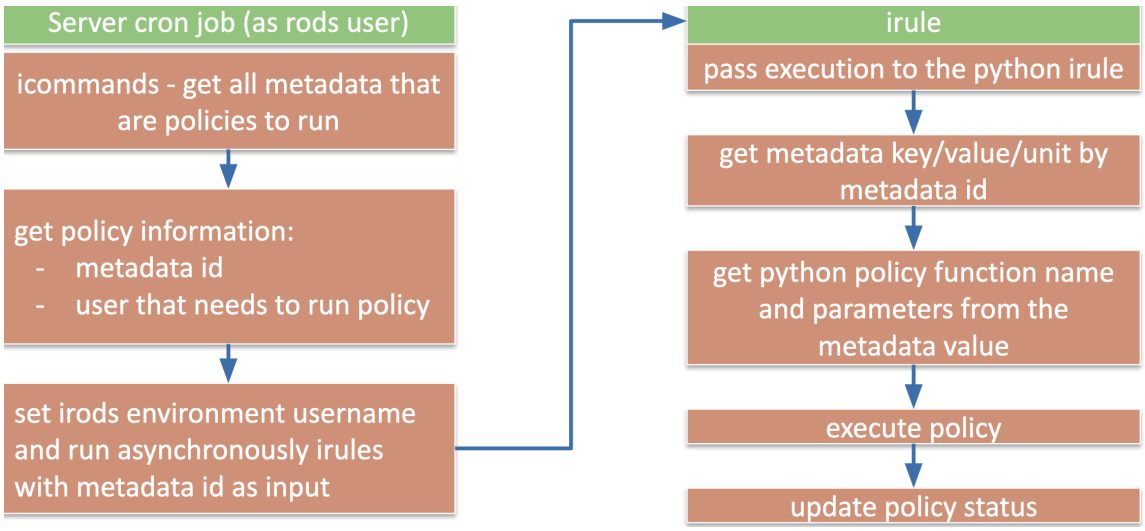


Figure 5. Shows a schematic view of the policy engine's workflow.

The Python code shown above could in principle allow users to inject malicious code into the JSON string. This will trigger server-side processes to operate on the data residing in the RUG RDMS. To avoid such an unauthorised operation, a policy validation mechanism is implemented in the mid-tier layer of the RUG RDMS. When a policy metadata is added to an object, the system generates an unique non-reversible hash value associated with the newly introduced policy. These values are stored in the mid-tier database and are used as guards to validate the user-requested policies. When a policy tag is added by a user, its hash value is generated and compared to the ones stored in the database. The process is triggered only when there is a match between the stored and computed hash values.

METADATA TEMPLATES

In recent years, researchers' capability of gathering data have increased a great deal. Descriptive metadata is used by many fields and stakeholders, and has evolved from different communities with diverse discipline-specific objectives and backgrounds. Metadata can be seen as a formally structured and documented collection of information about data that reveals minimally what is in the data, where the data originated from, who produced them, when they were produced and modified, why they were produced, and how the data can be obtained. Metadata characterizing a dataset generally contains all the attributes that are relevant to the specific research domain, but might as well contain general measurement attributes such as number of instances, dataset dimensionality, or circumstantial information. Besides describing the data being stored, metadata can also be used as triggers to configure and customize automatic workflows.

In the process of setting up RUG RDMS we have received several requests for metadata templates in different research domains, ranging from social sciences to archeology or microbiology. There is no one-size-fits-all solution, as each research domain has specific metadata attributes, and even within the same research domain there are large variations. By allowing users to define their own metadata schemas, we leave the definition of the metadata, which requires domain specific knowledge, to the domain experts. Metadata schemas define the general format of the metadata, that is, which elements are allowed, the number and order of their occurrences, of which data type they should be, which elements are required, etc. Because the technical skill of the user is also extremely diverse, RUG RDMS supports the use of JSON Schema templates together with a metadata template builder (Figure 6).

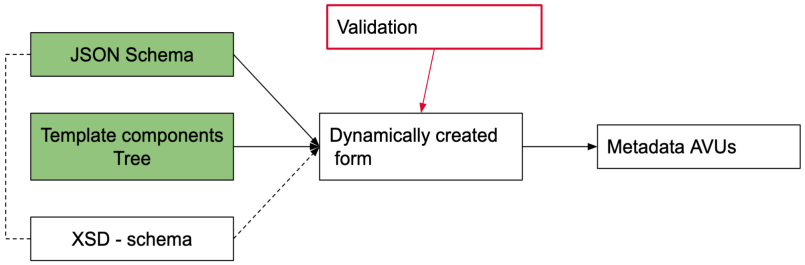


Figure 6. Working principle of the metadata templates in the RUG RDMS system.

This allows users to define their own tree-like types and type restrictions. These types of metadata attributes can be freely combined to build domain, and sometimes even user specific metadata templates. The definition of the metadata template tree of components is stored in the PostgreSQL mid-tier database. Based on the template definition an input form is dynamically built. The form handles the validation of the input values based on the template definition. When data has been validated, the form entries are serialized and stored as key-value-unit sets in iRODS. The same form is used to update the values of the metadata template fields.

THE AUDITING MODULE

The auditing module is using the iRODS audit plugin to store data operations in an elasticsearch database. This module contains functions to query operations on collections and data objects such as uploading, removing, moving, and downloading data. Figure 7 displays the auditing pipeline for the RUG RDMS system. The iRODS audit rule engine plugin can emit a single AMQP message for every policy enforcement point (PEP) encountered by the iRODS server which gets further pushed to the RabbitMQ service. AMQP stands for Advanced Message Queuing Protocol and it is an open standard application layer protocol for message-oriented middleware. The AMQP message emitted by the iRODS plugin has the information related to that particular operation, including username, filepath, filesize, etc. RabbitMQ is a message-queueing software also known as a message broker or queue manager. It is basically software where queues are defined, to which applications connect in order to transfer a message or messages. It also serves as a temporary location for messages by storing them while the destination application is busy or not connected. The messages in RabbitMQ are pulled by the ELK stack and can be displayed to the user. ELK is the acronym for three open source projects: Elasticsearch, Logstash, and Kibana. Elasticsearch is a search and analytics engine. Logstash is a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "stash" like Elasticsearch. Logstash can dynamically unify data from disparate sources and normalize the data into several destinations. Any type of event can be enriched and transformed with a broad array of input, filter, and output plugins, with many native codecs further simplifying the ingestion process. Kibana lets users visualize the data in Elasticsearch with charts and graphs.

Figure 7. Auditing pipeline for the RUG RDMS system.

The data stored in Elasticsearch contains information on the PEPs being triggered in the iRODS backend. There are custom queries to answer questions such as: who, when, and what operation was performed and on which data objects. To prevent the Elasticsearch database from rapidly growing out of control, configuring a regular expression for filtering becomes necessary so only the output of desired PEPs is retained. The exact regular expression used is:

```
"(^audit_pep_api_((?!gen_query|auth_response|auth_request).)*_(pre|post)$|audit_pep_api_auth_.*_except)"
```

This states that the system is interested in logging activity from the api plugin (while specifically excluding authentication PEPs and general queries), but specifically including any authentication errors that may occur. Additionally, empty fields are filtered using a Ruby script in the Logstash configuration. We have observed that using these filters, around 95% of the messages are excluded from being stored in Elasticsearch. The filters do not affect the data operation queries.

In order to comply with the General Data Protection Regulation (GDPR) [17] and funding agencies requirements, the system is required in some cases to store the data including its audit trail for up to ten years. Despite the fact that the amount of data being stored in the ELK stack has been considerably reduced, the amount of data from the audit trail is still large. As a consequence, Index-Lifecycle-Management (ILM) policies for managing the indices have been implemented. In a cluster under this architecture, different types of nodes can be configured to balance performance and capacity.

The exact Elasticsearch architecture is still a work in progress. We are considering a hot-warm-cold architecture, where the exact number of shards is yet to be decided. The four stages in the index lifecycle are:

- Hot—the index is actively being updated and queried.
- Warm—the index is no longer being updated, but is still being queried.
- Cold—the index is no longer being updated and is seldom queried. The information still needs to be searchable, but it's okay if those queries are slower.
- Delete—the index is no longer needed and can safely be deleted.

Additionally, we have considered using Snapshot-Lifecycle-Management policies to back up iRODS indexes periodically in a snapshot repository.

CONCLUSION

The RUG RDMS is a system for the research data storage and management. It is designed and built to meet the requirements of our local researchers and research groups. Using iRODS as a storage backend it delivers flexible and robust infrastructure. There are many challenges ahead. At the moment there are approximately seven pilot use cases employing RUG RDMS. The range of the projects is wide, from collaboration with library and publishing departments to tape storage of a large amount of data for astronomy. The RUG RDMS development team is looking forward to helping researchers at the university and working hard to deliver the system to the scientific community as an open-source project.

ACKNOWLEDGMENTS

The authors would like to thank researchers from various departments for their involvement and valuable input during the requirement engineering and test phase of the system. Furthermore, we want to thank the project's steering board, our colleagues from the Center for Information Technology, Research Data Office, University of Groningen Library and CMS team, acknowledge everyone who is and was involved in the project and had helped us to design and test the system.

REFERENCES

[1] Agarwal, R. and Dhar, V.: Big data, data science, and analytics: The opportunity and challenge for IS research. Information Systems Research. Vol. 25, issue 3, 443--448 (2014)

[2] Wilkinson, Mark D et al.: The FAIR Guiding Principles for scientific data management and stewardship. Scientific data vol. 3 160018. (2016)

[3] Ton Smeele and Lazlo Westerhof: Using iRODS to manage, share and publish research data: Yoda, iRODS UGM 2018 proceedings (2018)

[4] YoDa – a research data management service, **https://www.uu.nl/en/research/yoda**

[5] The University of Groningen, **https://www.rug.nl/**

[6] The University Medical Center Groningen (UMCG), **https://www.umcg.nl/**

[7] iRODS, **https://irods.org/**

[8] iRODS documentation. **https://docs.irods.org/**

[9] NGINX, **https://www.nginx.com/**

[10] Django, **https://www.djangoproject.com/**

[11] Cesare Pautasso, Olaf Zimmermann, Frank Leymann: RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision, 17th International World Wide Web Conference (2008)

[12] Eclipse Jersey, **https://eclipse-ee4j.github.io/jersey/**

[13] Apache Tomcat, **http://tomcat.apache.org/**

[14] Davrods - An Apache WebDAV interface to iRODS, **https://github.com/UtrechtUniversity/davrods**

[15] NFSRODS, **https://github.com/irods/irods_client_nfsrods**

[16] iCommands, **https://docs.irods.org/master/icommands/user/**

[17] General Data Protection Regulation, European Commision. April 27, 2016. url: **https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679**

Automating Data Management Flows with iRODS and Globus

Vas Vasiliadis
University of Chicago
vas@uchicago.edu

ABSTRACT

Major research instruments operating at ever higher resolutions are generating orders of magnitude more data in relatively short timeframes. As a result, the research enterprise is increasingly challenged by what should be mundane tasks: describing data for downstream discovery and making the data accessible (often with appropriate access controls) to the broader research community. The ad hoc methods currently employed place undue burden on scientists and system administrators alike, and it is clear that a more robust, scalable approach is required.

The Globus platform-as-a-service (PaaS) and, specifically, the Globus Flows service is increasingly used to easily build and execute automated data flows in this context. We will describe how Globus platform services may be used in conjunction with iRODS's robust storage capabilities to facilitate automated flows that: (a) stage data to intermediate storage, (b) extract and ingest metadata into an index for downstream discovery, and (c) manage access permissions to allow secure sharing of the data with collaborators. We will use a Jupyter notebook to demonstrate how Globus services are combined in this scenario, providing attendees with actionable code that may be easily repurposed for their needs. We will also illustrate how such an automated flow can feed into downstream data portals, science gateways, and data commons, enabling search and discovery of data by the broader community.

iRODS UGM 2021, June 8-11, 2021, Virtual

iRODS Client: iRODS Globus Connector

Justin James
Renaissance Computing Institute (RENCI)
University of North Carolina at Chapel Hill
jjames@renci.org

ABSTRACT

The iRODS Globus Connector has recently been released and provides connectivity between Globus endpoints and iRODS storage. This talk will explore the work required to port the GridFTP Data Storage Interface (DSI) into the new Globus Connect ecosystem.

iRODS UGM 2021, June 8-11, 2021, Virtual

iRODS and NetCDF Updates

Daniel Moore
Renaissance Computing Institute (RENCI)
University of North Carolina at Chapel Hill
dmoore@renci.org

ABSTRACT

This talk will cover recent changes to the iRODS NetCDF API plugins, iCommands, and microservices. This is ongoing work and we are interested to hear about current and potential use cases with this technology.

iRODS UGM 2021, June 8-11, 2021, Virtual

Grassroots Enhancements for iRODS

Simon Tyrrell
The Earlham Institute
Norwich Research Park,
Norwich, NR4 7UZ, UK
simon.tyrrell@earlham.ac.uk

Xingdong Bian
The Earlham Institute
Norwich Research Park,
Norwich, NR4 7UZ, UK
xingdong.bian
@earlham.ac.uk

Robert P. Davey
The Earlham Institute
Norwich Research Park,
Norwich, NR4 7UZ, UK
robert.davey@earlham.ac.uk

ABSTRACT

This paper explains tools we have created to enhance the interoperability and usability of iRODS storage. We have added functionality to the eirods-dav Apache module to expose iRODS collections, data objects and metadata as Frictionless Data Packages. Along with this we have created a tool to enhance the functionality of iRODS client commands within a BASH terminal by allowing for TAB key filename completion for iRODS collections and data objects.

Keywords

iRODS, FAIR Data, Frictionless Data, Bash

INTRODUCTION

The Grassroots Infrastructure [1] project aims to create an easily-deployable suite of computing middleware tools to help users and developers gain access to scientific data infrastructure that can easily be interconnected. With the data-generative approaches that are increasingly common in modern life science research, it is vital that the data and metadata produced by these efforts can be shared and reused. The Grassroots Infrastructure project wraps up industry-standard software tools along with our own custom open-source software tools to give a consistent API that can be federated with others in terms of both data and services. This means institutions and groups can deploy a simple lightweight software suite, locally or as a virtual machine, to expose institutional data, connect up any existing data services, and federate their instance of Grassroots with other remote instances.

One of the major aims of the Grassroots Infrastructure is to allow users to share their wheat data, although it is by no means organism-specific, as easily and seamlessly as possible based upon FAIR data principles [2]. For data storage, we use the iRODS [3] data grid system that gives users access to potentially differing file systems and data resources through a single data abstraction layer. Users are able to carry out typical filesystem actions as normal, such as creating files and directories and maintaining permissions, along with extra features such as distributed storage viewable across different institutions and the ability to add extensive metadata to files and directories.

Using iRODS as our data storage infrastructure, we looked to see what we could do to enhance its functionality and usability for both internal users accessing the system using the standard iCommands and external users by adding interoperability APIs. We now describe some functionality that we have added in both of these areas.

FRICTIONLESS DATA

The Grassroots Infrastructure is an integral part of the Designing Future Wheat (DFW) programme [4] and one of the tasks it is used for is to host a portal to share the datasets produced within the programme. These datasets can vary greatly in size and their data is heterogeneous so one of our goals is to standardise access to these datasets. As part of the goal to make the data FAIR as much as possible, we looked at how we could increase the interoperability

of these datasets. Given their heterogeneous nature, we needed a solution that was both flexible and extensible, along with having good support and tooling for any APIs that are provided. The solution that we found was Frictionless Data [5] which are an open set of data standards and provides software to work with data. Frictionless Data Packages are a container used to describe and package collections of data into JSON objects. These packages can store any types of data and metadata and be used with an existing set of Frictionless schemas as well as being extensible by using any custom schemas as well.

Basic Frictionless Data functionality

The Frictionless Data support can be configured to only be active at particular points in the Apache directory hierarchy using the `DavRodsFrictionlessData` configuration directive. For example, to have the data packages appear in all of the top-level child directories of */data*, but not in subsequent child folders of these, the configuration would be:

```
# Generate Data Packages for all child directories directly below /data
<LocationMatch "/data/[^\/]+/">
   DavRodsFrictionlessData true
</LocationMatch>

# Since Data Packages are generated for the child directories configured in
# the line above, exclude all directories further down
<LocationMatch "/data/[^\/]+/[^\/]+/">
    DavRodsFrictionlessData false
</LocationMatch>
```

To give a useful visual indicator to denote the Frictionless Data Packages files within the generated listings pages, a custom image can be specified using the `DavRodsFDDataPackageImage` configuration directive which specifies the path to the image file to use for these packages. For example,

```
DavRodsFDDataPackageImage /eirods_dav_files/images/archive
```

would use the image at */eirods_dav_files/images/archive* to represent the Data Packages in the generated web page listings.

Metadata configuration

Each of the datasets come with a standard set of metadata based upon the Minimum Information About a Plant Phenotyping Experiment (MIAPPE) [6] which is an open set of standards to allow for easier integration of data from plant phenotyping experiments. Each project has metadata that contains the list of authors, the name of the project that the dataset comes from, a title for the project, a description of the project, the licence and a unique persistent identifier for the dataset. These metadata values all exist within the Frictionless Data Package standard thus making the task of connecting the iRODS and Frictionless technologies together straightforward. The default mappings of iRODS metadata keys to their Frictionless counterparts along with the relevant Apache module configuration directives are shown in table 1. All of these values are completely configurable using the Apache module configuration directives. For example, to use the value associated with the metadata key *project_title* from the iMeta catalog for the title field of the Frictionless Data Package, you would need the following configuration setting:

```
DavRodsFDResourceDescriptionKey project_title
```

The metadata values stored in iRODS can be combined where necessary to produce the value required for the Frictionless Data Packages. The configuration directives can be set by a comma-separated string containing the iRODS

Data Package field	Default iRODS metadata key	Apache Configuration Directive
license_name	license	DavRodsFDResourceLicenseNameKey
license_url	license_url	DavRodsFDResourceLicenseUrlKey
description	description	DavRodsFDResourceDescriptionKey
name	name	DavRodsFDResourceNameKey
authors	authors	DavRodsFDResourceAuthorsKey
title	title	DavRodsFDResourceTitleKey
id	id	DavRodsFDResourceIdKey

Table 1. Default mapping of metadata keys between iRODS and mod_eirods_dav Frictionless Data Packages and the Apache module configuration directives.

metadata keys to use. For instance, if the value that you wish to use for the description is given by concatenating the *short_info* and *detailed_info* metadata values, then the configuration would be:

```
DavRodsFDResourceDescriptionKey short_info,detailed_info
```

As well as specifying the keys, white space, periods / full stops and newlines can be specified in the configuration value too. With the example above, if you would like to have a more complex value generated using the *short_info* metadata value followed by a period / full stop and two blank lines (denoted by the \n escape code), then the *detailed_info*, a space and the *footnote* metadata value, the configuration would be

```
DavRodsFDResourceDescriptionKey short_info,.,\n,\n,detailed_info, ,footnote
```

Frictionless Data tabular support

One of the schemas available as part of the Frictionless Data standards is for Tabular Data Resources which is designed to represent tables of data. Eirods_dav has the ability to automatically take tabular data objects such as csv or tsv files and store them as Tabular Data Resources. This is done by querying the iMeta catalog for the given data object. The first required key is *column_headings* which has a comma-separated list of the column headings for the tabular file. For each of these headings an additional key-value pair specifies the type of data in the given column of the file. The keys for these are the column name with a *_type* suffix. The core values that these types can take are defined as part of the Frictionless Data Table Schema [7] which in turn are based open those from JSON Schema [8]. These include types such as strings for text, number for floating point values, integer for whole numbers, *etc.*

For example, consider a file *data.csv* which has three columns containing a string, an integer and a floating point number respectively, shown below

```
var1,var2,var3
A,1,2.1
B,3,4.5
```

Using the Tabular Data Resource type definitions, the types of these columns are string, integer and number respectively. Eirods_dav begins by looking for the *column_headings* key in the iMeta catalog for *data.csv* which in this case

would be

```
column_headings: var1,var2,var3
```

This value is then tokenized by splitting this string into separate values using the comma delimiters. In our example this would be*var1*, *var2* and *var3*. Eirods_dav then appends *_type* to each of these to produce the names of the iMeta keys to use to query for the datatype of each column. In this case, these would be *var1_type*, *var2_type* and *var3_type*. This would make the required metadata key-value pairs

```
var1_type: string
var2_type: integer
var3_type: number
```

Frictionless Data Package storage

By default, the Frictionless Data Packages are virtual and generated on the fly for each incoming request. Although this may be fine for smaller datasets, as they grow in size the time that it takes to generate these packages increases and thus may have a perceivable impact upon serving these requests. To address this problem, there is the ability to write the generated Frictionless Data Package back to its parent collection, in effect, caching it. This is done by setting the configuration directive `DavRodsFDSaveDataPackages` to `true` and is equivalent to running *iput* for the generated file. If the content of the dataset changes, this generated file would become stale and need to be regenerated. Eirods_dav checks for the existence of this file upon each request so by simply deleting this Frictionless Data Package file, eirods_dav will automatically regenerate it upon the next relevant incoming request.

BASH

One of the features that is available in many command-line environments is the ability to generate matching paths given partial file and directory names. For users on Linux, the common command-line terminal is Bash. An extremely useful feature of Bash is the ability to complete any text that has been typed in the shell up to that point by pressing the TAB key. As standard, iRODS does not come with this feature for the components within its iCommands package. As the number of nested levels within the iRODS zones increases, the time that it takes to type these commands and the likelihood of mistyping a path becomes greater, and the usefulness of auto-completion is lost

To address this issue, we created iRODS Bash Completer[9] to fill this gap by implementing tab-completion for the various iRODS client iCommands. The Bash shell can take advantage of the iRODS Bash Completer to auto-complete paths for the data hosted in iRODS in the standard way as it does for other mounted file systems. It does not require any escalated user privileges to use and has no dependencies other than the libraries within the iRODS development package. When the TAB key is pressed, it parses all of the available iRODS zones that the current user has access to and produces a list of data objects and collections that match the partial path that the user has currently typed in. If no text has yet been entered, it can be used to traverse through the entire hierarchy of the iRODS-hosted data. The installation process requires a small number of steps beginning with building the irods_bash_completer program using the *makefile* within the repository. Once this is built it can be copied anywhere that the user wishes. The iRODS Bash Completer can be configured for each of the iCommands, or indeed any third party application, by editing the supplied *bash/icommmands.d* file to specify where the *irods_bash_completer* has been installed. This is done be setting setting the first part of the `opts` variable within this file to the path where the iRODS Bash Completer has been installed. For example, to specify that it has been installed to */home/billy/bin/irods_bash_completer* the initial

part of the *bash/icommmands.d* would be

```
_irods_completer()
{
    local cur prev opts
    COMPREPLY=()
    cur="${COMP_WORDS[COMP_CWORD]}"
    prev="${COMP_WORDS[COMP_CWORD-1]}"
    opts=‘/home/billy/bin/irods_bash_completer ${cur}‘
```

Any other iRODS-aware commands can have TAB completion enabled by adding an entry to the end of this file. For example, this functionality can be enabled for a third party program called *irods_examine*, say, by adding the following snippet

```
complete -F _irods_completer irods_examine
```

to the end of *bash/icommmands.d*.

Once you have completed the configuration, copying *bash/icommmands.d* into the */etc/bash_completion.d/* directory will complete the installation and enable the TAB completion functionality.

ACKNOWLEDGMENTS

The Grassroots project is strategically funded through the BBSRC Designing Future Wheat programme grant, BB/P016855/1, and aims to develop a lightweight reusable set of open source software tools to allow researchers to share and federate life science datasets.

AVAILABILITY

The source code is for iRODS Bash Completer is available at `https://github.com/TGAC/irods_bash_completer` under the Apache License Version 2.0. The source code for mod_eirods_dav is available at `https://github.com/billyfish/eirods-dav` under the GNU Lesser General Public License version 3.

REFERENCES

[1] Grassroots Infrastructure, `https://grassroots.tools`, Visited last on 06.06.2021

[2] FAIR Principles `https://www.go-fair.org/fair-principles/`, Visited last on 06.06.2021

[3] Hao Xu, Terrell Russell, Jason Coposky, Arcot Rajasekar, Reagan Moore, Antoine de Torcy, Michael Wan, Wayne Shroeder, Sheau-Yen Chen: iRODS Primer 2: Integrated Rule-Oriented Data System. Synthesis Lectures on Information Concepts, Retrieval, and Services, Morgan & Claypool (2017)

[4] Designing Future Wheat, `https://designingfuturewheat.org.uk/`, Visited last on 06.06.2021

[5] Frictionless Data, `https://frictionlessdata.io/`, Visited last on 06.06.2021

[6] MIAPPE, `https://www.miappe.org/`, Visited last on 06.06.2021

[7] Frictionless Data Table Schema Types, `https://specs.frictionlessdata.io/table-schema/#types-and-formats`, Visited last on 06.06.2021

[8] JSON Schema, `https://datatracker.ietf.org/doc/html/draft-zyp-json-schema-03#section-5.1`, Visited last on 06.06.2021

[9] Tyrrell, S.: iRODS Bash Completer, `https://github.com/TGAC/irods_bash_completer`, Visited last on 06.06.2021

A Prolegomenon for Improving Observability in iRODS

Arcot Rajasekar
School of Information and Library Science
University of North Carolina at Chapel Hill
rajasekar@unc.edu

ABSTRACT

Observability is an emerging practice for measuring and interpreting the pulse of complex and distributed software systems. Software developers and IT teams need to understand when and why there is an abnormal behavior, how to mitigate that within a short time. With the acceleration of complexity, scale, and dynamic architectures coupled with automatic software patching, malicious intrusions and system breakdowns, high throughput system need more than to react to events but proactively predict and mitigate anomalous behavior before it happened. Challenges due to multiple combinations of things going wrong, and sympathetic reinforcements of faults can make it hard to track how errors are manifesting and how a system is behaving. Observability couples the ability to capture runtime telemetry with a visual and reasoning system that can detect and pinpoint abnormal behavior deep in the system before it can affect the performance of the whole system. Introduced first in control theory, observability is a measure of how well internal states of a system can be inferred by knowledge of its external outputs. The iRODS software system is not only a very complex and dynamic system it is also being increasingly deployed with other complex software on distributed infrastructures that rely on its high throughput and reliability. An introduction to the concept of observability in iRODS would be very helpful for making sure that a deployed iRODS installation is operating in an optimal manner. Moreover, with observability built into iRODS, one can find operational anomalies in systems that are relying on iRODS and help mitigate them. The iRODS system already has a very strong measurement capability through its logging. Enhancing its capability with tracing, session replay, learning and analysis systems that can provide actionable insight would take iRODS to the next level of performance and resilience. In this paper, we look at various ways one can enhance iRODS to become a highly 'observable' system.

Keywords

iRODS, observability, metrics, logs, traces

INTRODUCTION

Observability[1, 2, 3] is the ability to understand the inner workings of a system by observing the external behavior. Originally described in control theory[4, 5], it is being increasingly being adopted by software engineers and system administrators to monitor system malfunctions and if possibly proactively preempt any such malfunctions. As the complexity of systems increase due to using multiple chained services, parallel and distributed processing, and multi-cloud environments there is a concomitant increase in pressure to maintain operational viability and graceful degradation of services. Hence reliability engineering and DevOps[6] are increasingly looking at observability as a means to provide better and more reliable and resilient services. With distributed data, computing and higher reliance on networking problems can crop up at multiple levels and become apparent only at a later stage as critical systems start to fail and there is a chain reaction effect leading to eventual shut down of the system. Even small failures can be a problem which can crop up because of a sequence of events in a distributed system and hence need to be captured and solved and made resilient in the future.

iRODS UGM 2021 June 8-11, 2021, Virtual

In computing systems, reliant on distributed hardware, multiple networking interconnects (from LANs, WANs and clusters) and distributed software services, developed by different vendors, reliability of the whole system to operate at an optimal level is of high concern. Observability provides a proactive way to monitor a large system and develop strategies in case of failures. In software and hardware systems, observability is achieved by measuring system critical components through various sensors and telemetry systems for hardware and logs and metrics and traces for software systems. Observability is implemented using a combination of instrumentation methods including tools, such as Grafana [7], OpenTelemetry [8], Splunk [9], SigNoz [10], etc. Observability empowers cross-functional teams (IT Admins, system developers, application engineers, managers) to identify problems before they even manifest or become unmanageable. Observability helps increase performance, availability, resiliency and user satisfaction by enabling one to realize what is slow, what is broken, and to quickly figure out what needs to be done to improve performance.

With the increased complexity of software system, it is highly becoming crucial to apply observability principles in order to run a viable enterprise. An innocuous update in an obscure software package can have a cascading effect. Finding problems and correcting them can be a nightmare and going beyond that, predicting failure and degradation of services can be rewarding but highly challenging. Moreover, with privacy and security considerations becoming increasingly important, observability can be key factor in finding anomalies and intrusions in a highly distributed networked system.

The integrated Rule Oriented System (iRODS) [11, 12, 13, 14, 15] is an open source software used for large-scale data management software used by research, commercial, and governmental organizations worldwide. It virtualizes storage system and presents a unified view of all digital artifacts stored under its management. The fundamental difference that iRODS brings to distributed data management is its provision of a rule-based operational characteristic Users, administrators and system developers can write complex rules to govern the management of their data throughout its data life cycle. These rules can apply from a single file-level to the whole system level. The application of the rule is governed by conditions and can be applied on an event, periodically or at any arbitrary time by the user or administrator. These rules govern all aspects of the data management from file ingestion to storage, from access to deletion from the system. Indeed, one can override all default and built-in operations implemented in the iRODS system to completely perform an entirely different set of operations that is suitable for the enterprise. Hence the iRODS system provides a far more challenging environment for performing observability compared to a static system whose actions are mostly defined as per the original developers' intent.

iRODS implements a complex and dynamic system that spans multiple services that run on distributed hardware across the wide area network. It is also being increasingly deployed with other complex software on distributed infrastructures that rely on its high throughput and reliability. An introduction to the concept of observability in iRODS would be very helpful for making sure that a deployed iRODS installation is operating in an optimal manner. Moreover, with observability built into iRODS, one can find operational anomalies in systems that are relying on iRODS and help mitigate them. The iRODS system already has a very strong measurement capability through its logging. Enhancing its capability with metrics, tracing, session replay, learning and analysis systems that can provide actionable insight would take iRODS to the next level of performance and resilience. In this paper, we look at various ways one can enhance iRODS to become a highly 'observable' system.

MONITORING FOR OBSERVABILITY

Monitoring observability relies on three main concepts: metrics, logs, and traces. These three are considered the pillars of observability. One needs to add monitoring and telemetry to instrument both the software and the hardware to gather useful information that can be used to find if the system is running in a sub-optimal manner or to predict potential problems down the road. When a fault is detected, analyzing these three measures will be helpful in pinpointing areas of concern.

Metrics are the operational characteristics that are collected to define what is the optimal performance for a system. These are derived over time based on system performance. Types of metrics include response time for a service (micro-

service), uptime and downtime of systems, number of requests serviced during a period of time (e.g. concurrency), load of the system including cpu, memory and network utilization, etc. In many cases, service level agreements (SLA) or service level objectives (SLO) provide the metrics that need to be met for providing required performance. Metrics provide the baseline to which one can compare operations at the current level to find any anomalies. Moreover, metrics provide a means to detect degradation of service. More metrics is defined and continuously monitored, the better will be the reaction time to solve problems or preempt them before they become critical. Service level indicators (SLI) provide the measurement that detect the service behavior at the current time. Comparing SLI against SLO or SLA provide a measure of the performance. SLI can be used not just to measure if the system is meeting current requirements but it can also be used to find slack in the system that can be utilized to provide more service without upgrading the system. As the SLI shows a degradation against an SLO or SLA, then solutions to improve service – such as bringing an additional server online or performing better load balancing – would be needed. Similarly, Key Performance Indicators (KPI) are also necessary metrics that capture data about crucial performance measurements. KPIs provide the necessary baseline data that can be used for strategic and operational improvement and to provide an analytical basis for decision making. KPIs helps by setting targets that capture the most important critical metrics for the success of a system and help focus attention on what matters most. KPIs and SLIs together help to assess progress towards optimal performance results. They are critical for tracking efficiency, effectiveness, quality, timeliness, governance, compliance, system performance and resource utilization.

Logs are basically outputs written by software procedures to record their operations. Hence in a well-written procedure or micro-service, print statements to a log device (either a log file, or a message to a queuing system or to a remote recorder) capture the current status and actions of the procedure. Since they are written by the system developer, these can be highly structured and provide enormous details about the status of the system as well as about the operation being performed. For example, key log outputs can be written when a process enters a micro-service and when it exits a micro-service. In such a case, some entry data can be part of the log, and because there can be more than one exit point for the micro-service, details of such exit, normal or abnormal, can be captured. In particular, these two entry-exit logs will have fine-grain timestamps and provide a measure of how long the process took to perform this micro-service. Log analysis can help to uncover abnormal, unpredictable and emergent behaviors by procedures. Logs are easy to generate provided the system developer has instrumented the code appropriately. In many systems, there may be multiple levels of logging and the administrator can choose the level that would be comfortable for the system. Since a logging operation take a finite amount of time, intrusive logging by themselves can slow down the function of a procedure. Hence one needs to be aware of how much of logging is desired to monitor the system against the degradation of service that is introduced by the logging process. Distributed logs (logs gathered from distributed systems) are normally gathered at a central location and analyzed. Such analysis provides a means of finding the reliance of one system over other and can be used to drill down when an operation takes more time than normal.

Tracing is the concept of capturing the journey of a user request from start to finish and to find what are all the services and components that are involved in answering that request. Particularly in a distributed system, tracing provides a means to find all the nodes of the system through which the service request 'touches' on its way to completion. Hence trace data is more complicated than logging but rely on logging to find traces of all actions being performed by the system. Traces provides a snapshot into system behavior such as which component took the longest or shortest period of time, or whether specific underlying functions resulted in errors. Traces provide context to a user request and can be used to find what is really happening when a user request is entered into the system (by user request, we mean the high level request that a user makes – e.g. for Google it can be the search for a keyword, or a request for a routing path to the Map application, or in the case of a bank, it can be a request to find the account balance or perform a money transfer or pay a bill). In particular, tracing helps to analyze individual user calls and understand the sequence of (often distributed) steps that are taken to return a result. Tracing also helps to analyze how the calls behave for different users, and also when the results are successful or failure.

Journeys provide the sum-total of all activities that is performed during a user session. A journey can have multiple sub-journeys. Each journey can be made of several paths which can be parallel in a distributed system. A journey

captures timing, possibly call and return expressions, status code and anything else that an Observer deems to be necessary. Journey can be abstracted into templates and help find bottlenecks and errors so they can be fixed and optimized. By analyzing trace data and journeys, one measures not only the overall system health, but also by looking at similar traces can pinpoint breakdowns or slowdowns, find bottlenecks, identify issues that can lead to performance degradation and also provide a means to prioritize important code snippets for optimization and improvements. For example, tracing can help find common bottlenecks (e.g. a database call) that is used by different traces and hitting similar slowdown.

To perform better observability all three of these measures – metrics, logs, and traces – are needed. In many cases data from multiple of these measures will be used in a dashboard to provide a clear picture of what is happening in a system. The power of observability comes from a unified approach – individually these measures give only a partial picture – in taking advantage of these three types of measurements to find the health of a given software system. Observability dashboards gain insight by coalescing data from all three measures and using them to find anomalies and problems that need to be solved.

Visualization and analytics play a key role in observability. Visual analytics, with human-oriented graphs, are used by observability dashboards to provide the status of a system in real-time. Visual cues, color coding and other tricks are normally used to pinpoint abnormalities. Artificial intelligence, machine learning and predictive and post-mortem analyses are key tools in making observability a success. Continuous analytics, adaptive learning, deep analytics are increasing being used to find anomalies before they become life-threatening problems to the systems health.

CURRENT STATUS OF OBSERVABILITY IN iRODS

The iRODS system is a complex software that spans multiple hardware as well as software services. The rule-based system – using micro-services – makes it harder to define common patterns of actions as each installation can have its own idiosyncratic rule applications.

Even though observability is not currently being performed in any iRODS system, there is a rich support for eventually providing a good measure of observability capabilities. The iRODS system is a mixed bag when considered from observability point of view, it has some aspects that are vital to perform observability, but also lacks other crucial measures and tools to become successful in observability. There are three types of measurements one can identify in iRODS that can be helpful for observability.

Server Logs

Server logs are collected in each distributed storage server and the iCAT-enabled server in the form of a server log file. These contain mainly print statements from various functions and procedures in the iRODS system, crucially from the micro-services. In iRODS there are levels of server log monitoring and the site administrators can throttle the levels as needed to capture coarse or fine-grained information.

Audit Trails

The audit trails provide a different type of logging, but mainly tied to the action performed on the data objects stored in the iRODS system. For example, audit trails captured the information about when someone created a file, who wrote or modified the file and who accessed the file. Timestamps are also recorded. Audit trails collect user information on triggered actions. One can use them to recreate traces from a data object viewpoint about what happened to it during its lifetime.

Status Metadata

This information comes from the iCAT metadata catalog. iCAT stores persistent information in its database which can be used to analyze different types of metrics. For example, the size of the file and the time it took to access it can be used together to find whether the system is operating in an optimal fashion.

At the current level, iRODS is not doing any observability. One can view the current support in iRODS is more towards system monitoring activities with no observability functions. Server Logs, Audit Trails and Status Metadata in iRODS provide a strong and stable foundation for implementing observability. Use of policies, rules and microservices provide one more level for gaining information to perform observability.

IMPROVING OBSERVABILITY IN iRODS

A concerted effort is needed to upgrade iRODS functionality to perform true observability. At the current stage iRODS is lacking critical features and functionalities that hinder it from being a viable observability system that can optimize performance, predict problems, and isolate malforming processes. In its current state, iRODS provides limited log and trace measurements. Missing functionalities include metrics and journeys, apart from implementation additions for more useful logs and traces. Missing tools include visualizations, dashboards, and analytics. We believe that metrics and improved logs, traces and journeys can be implemented in iRODS without any change to its software stack, thanks to the rule-based micro-services paradigm of its underlying architecture. Additional modules for visualization and run-time analytics are needed to be developed (or adapted from open source implementations) to provide administrators insights into its day-to-day operational characteristics.

Logs and Traces

One big advantage to implementing iRODS comes from its current provision for server logs, audit trails and status metadata. One can use these three measurements that are already implemented in iRODS to perform log-based operations and tracing operations. But there are some crucial issues that need to be solved before we can do that. An important bottleneck is the distributed nature of the server logs. Server logs are stored at individual storage servers and are not synchronized enough to be a useful form of log and trace measurements. Hence, centralizing these with time synchronizations would be the first step that one needs to do to take advantage of these outputs from iRODS procedures and function calls. As mentioned earlier, audit trails are data-centric rather than session centric. Hence their value is doubtful when using them to perform traces or to use log activities to find sub-optimal operations. They may provide a valuable 'journey-type' information about how multiple data objects are being used – but those journeys may not be key to observability. Analyzing them may still be useful and can provide insights into pre-staging, load balancing and replication.

Status metadata can be used as metrics information. But the type of status that is logged in the iCAT may prove to be useless from the observability point of view. Even though iCAT captures a rich amount of metrics in these metadata, most of these are data object related and/or user and resource related. They may not provide insights into operational data necessary to define metrics measurements. Some possibilities still exists: location of objects, length of objects, limits on parallel transfer operations and such can be useful in defining some lower and upper bounds of operational metrics.

New Metrics Measures for iRODS

Operational system level metrics are necessary to be developed for iRODS in order to provide the baseline for application performance monitoring (APM). APMs provide critical checks about whether the system satisfies SLA contracts, meets SLO performance standards, identify bugs and potential issues, and provide flawless user experiences via close monitoring of IT resources and process timelines. APMs are also needed to increase mean time between problems and reduce mean time between resolution of such problems. Continuous monitoring and comparing against standard metrics is crucial towards proactive remediation. APM with metrics can be used to provide just in time alerts and perform simple analysis that can help to identify possible problems in the future.

There are several metrics measurements that can be implemented to help monitor the health of any iRODS implementation. Many of these metrics can be standardized across implementations, but others need to be defined per installation. Moreover, these measures themselves can change over time as system gets loaded and/or modified and upgraded. We provide one such set of metrics that we feel is important as a starting point for iRODS. This list is by no means an exhaustive list and more will be added as the observability capabilities are improved in the future.

Many of these can be considered as SLAs or SLOs that provide KPIs. Continuous monitoring will provide the needed SLIs to take advantage of these metrics.

- CPU/Memory Usage (max, min and average)
- Network Traffic (max, min, optimal connections and loads)
- Database Load (concurrent usage, speed for critical queries)
- Error Types/Rates (current errors that are not showstoppers)
- Request rates (connections per unit time, max, min, average)
- Response times (mean, max, min)
- Bandwidth/Throughput (volume and number of files)
- Concurrent Connections (external load on the system)
- Number of instances/threads (internal load on the system)
- Microservice/function usage/time (per micro-service metrics)
- Uptime, Restarts & Availability (system SLOs)
- User Experience KPIs: SLAs, SLOs for critical user requests
- Software Module KPIs: SLOs are each major and some minor software modules.

The above metrics are needed to measure against run-time performance of the iRODS system. Some of these measures are formulaic as they depend upon the data object size or system configuration size (e.g. how many cores). But it is a good idea to lay down some metrics that can help one to see where one is meeting important KPIs. As mentioned earlier, KPIs help in setting targets that capture the most important critical metrics for the success of a system and help focus attention on what matters most. KPIs and SLIs together help to assess progress towards optimal performance results and user experience. They are critical for tracking efficiency, effectiveness, timeliness, system performance and resource utilization.

Journeys for iRODS

Journeys are important to find critical modules, performance of modules as well as to provide insight into how a user request is addressed. Journeys use traces but build a profile of a session that is more informative than a trace. In the case of iRODS to define a journey one needs to first address the complexity of the distributed tracing (DT) that needs to be analyzed in order to define the actions performed by a user request. Chaining of services (including microservices) and peer-to-peer connections across distributed systems makes it hard to trace the activities of a session but is critical for performance monitoring. DT helps identify bottlenecks across dynamic and heterogeneous infrastructures on which an iRODS system is executed.

In order to define journeys, the current server logs need to be enhanced with more operational reporting. One may reduce the burden in developing this capability by only increasing the reporting of crucial modules with more useful data that can help when creating the journey profiles. As mentioned earlier, a critical problem is the distributed nature of the server logs in their current implementation. One may need to consider methods to centralize them in real-time so that one can take advantage of them to analyze real-time abnormalities but also to develop prototypes for journeys that can be compared for anomalies.

To take advantage of the concept of journeys in iRODS the following types of analysis need to be implemented:

- Distributed Transaction Monitoring and Analysis
- Create User or Application Profiles
- Define Patterns and Templates of Journeys and Sub-journeys
- Latency optimization
- Failure Models
- Alternate Pathways
- Service Dependency Analysis
- Critical Path Analysis
- Root Cause Analysis

Providing these analyses will help in building journeys and in timing their performance and finding deviance from optimal levels.

Observability Analytics for iRODS

At the high level, there are four types of analytics that are relevant for observability that can be utilized to gain insights from traces, logs, metrics and journeys.

- Predictive analytics can provide insights into what is likely to happen based on historical data. Using data mining, statistical modeling and machine learning predictive analytics help in defining probabilistic future outcomes. Analyzing past patterns and trends one can gain insights to induct what might happen going forward and, in doing so, inform about setting realistic goals, effective planning for failure as well as improvements, managing performance expectations and avoiding risks. Predictive analytics play a key role in observability as one can look at various patterns in journeys and traces to figure possible failures down the road. In iRODS such trends can be used to identify network degradation, resource over utilization, and take actions to overcome them.
- Diagnostic analytics (also called post-mortem analysis) provide a means to figure what adverse event happened and help to find out why it happened. Forensic analysis of traces, journeys and logs help in figuring out the root cause of a failure after it has happened. It can also provide insight into how one can mitigate such failures in future through corrective actions. Diagnostic analytics use data aggregation and data discovery methods to find historical patterns. Visual analytics also play a key role in these types of analytics. Summary statistics, clustering, pattern tracking, and regression analysis are some of the ways one can find patterns and failure mode backtracking in the data and measure performance.
- Descriptive analytics provides insights and reports on the inner working of the system. They can be used to aggregate performance metrics as well as report on events. They can be used to find the most used micro-service and such independent of traces and journeys. An important use of descriptive analytics is to check on the value of the metrics defined for the system and fine tuning some of them for optimal performance. User profiles, resource profiles, bandwidth utilizations are some of the key factors that can be aggregated and used towards improving performance.
- Prescriptive analytics helps you how to avoid certain adverse event from happening. It can tell what, when and why something can happen. It is in a sense like game playing. One can identify different scenarios and simulate actions and see how the system behaves under such circumstances. For example, one can use prescriptive analytics to find when a particular storage system can break and under what loads can we expect worse performance beyond what is promised under an SLA. Prescriptive analysis can be used in conjunction

with the other three types of analytics to provide valuable insights to make evidence-based reasoning about future events. For example, by tweaking a journey, one can see under what scenarios the journey will adversely affect the outcome of a request. Predicting fluctuations, perturbations, assessing upgrades are some of the uses in iRODS for such analyses.

Observability uses multiple types of algorithmic solutions to realize the above three types of analytics. Here are some examples of such types of analytics that will help perform observability in iRODS.

- Statistical analytics is used to analyze metrics data for informative nuggets. Max, min, median, mean, standard deviation, etc. can provide valuable insights. Statistics can be used to define norms, SLAs and expected outcomes and latencies. Hence is a way to define important KPIs.
- Graph Analytics use traces and journeys to find patterns. Pattern analysis. Critical nodes and Most used nodes. They can be used to perform predictive as well as prescriptive analysis. Graph analysis is also a very useful way of finding outliers and anomalies in data. In the iRODS context, couple of uses can be seen in pre-staging and delayed processing.
- Visual analytics is another form of graph analytics and is very useful when performing temporal and spatial data analysis. They are also used to fusing multiple layers of data together and provide insights that can stand out in a visual setting. One can find trends, collinearities, correspondences, correlations and clusters, and anomalies using visual analytics.
- Text Analytics: These can be used effectively when analyzing log files. They will also be effective in creating journeys and analyzing them. One can use contextual data of journey to define dynamic slicing and figure out repeatable experiences with possible minor deviations.
- Data Mining, Machine Learning and Deep Learning will help us to learn good and bad patterns. For example, one can teach a neural network about multiple journeys, both successful and failed journeys. Then when in real-time a journey is progressing, the neural network model can be used to predict success and failures based on the pre-path of the journey and take corrective actions to mitigate the risk of failure.

CONCLUSION

This paper provided a glimpse of how one can perform observability in a complex system such as the iRODS. Observability is becoming increasingly important because of complexities of the applications as well as need for high availability and throughput by the user community. Observability can be used as a means to monitor the system continuously and correct them on the fly. Observability can also provide insight to developers and administrators on how system performance and user experience can be improved. Current status of iRODS provides some rich measures that can be effectively used to perform observability. Server logs, audit trails and iCAT metadata can be the basis for assembling an observable system metrics and traces. But there is clear need for improvements. Metrics can be improved, and server logs can be centralized and improved for utility as a measure for observability. The concept of journeys is new for iRODS as its current metadata caters mainly to data-centric life cycle. Plugging in concepts for analytics can make iRODS more effective for future users. iRODS is used as an enterprise system and hence it needs observability tools to make it a world-class enterprise system.

REFERENCES

[1] Burns, J. The complete guide to observability. Lightstep Research White paper. https://go.lightstep.com/rs/260-KGM-472/images/observability-guide.pdf

[2] Trivedi, Tapan. (2018). Controllability and Observability. 10.13140/RG.2.2.33290.62407.

[3] Vidal, R., Chiuso, A. Soatto, S., & Sastry, S. (2003, April). Observability of linear hybrid systems. In International Workshop on Hybrid Systems: Computation and Control (pp. 526-539). Springer, Berlin, Heidelberg.

[4] Kalman R. E. On the General Theory of Control Systems, Proc. 1st Int. Cong. of IFAC, Moscow 1960 1481, Butterworth, London 1961.

[5] Kalman R. E. Mathematical Description of Linear Dynamical Systems. SIAM J. Contr. 1963 1 152.

[6] Jabbari, R., Ali, N., Petersen, K., Tanveer, B. (May 2016). What is DevOps?: A Systematic Mapping Study on Definitions and Practices. Proceedings of the 2016 Scientific Workshop.

[7] Grafana: The open Observability platform. https://grafana.com

[8] OpenTelemetry. https://opentelemetry.io

[9] Splunk: Observability for DevOps. https://www.splunk.com/en_us/observability.html

[10] SigNoz: Open Source APM. https://signoz.io

[11] iRODS: Open Source Data Management Software. https://irods.org

[12] Xu, H., R. Moore, A. Rajasekar. A Method for the Systematic Generation of Audit Logs in a Digital Preservation Environment and Its Experimental Implementation In a Production Ready System, iPRES conference, November 2015.

[13] Moore, R. W., A. Rajasekar, M. Wan. iRODS Policy Sets as Standards for Preservation, US-DPIF'10 Proceedings of the 2010 Roadmap for Digital preservation Interoperability Framework Workshop, 2010.

[14] Moore, R., A. Rajasekar, M. Wan. Policy-based Distributed Data Management Systems, OR'09, Atlanta, Georgia, May 2009.

[15] Rajasekar, A., M. Wan, R. Moore, W. Schroeder, Federation of Rule-Based Data Grids, Workshop on Digital Archive Preservation and Sustainability, September 2008, Baltimore, Maryland.

Issues in Data Sharing for Environmental Health Sciences

Mike Conway
NIEHS/NIH
mike.conway@nih.gov

Deep Patel
NIEHS/NIH
deep.patel@nih.gov

ABSTRACT

NIEHS is, like many other research enterprises, entering a new era of opportunity and challenge as methods of data-driven discovery emerge. The challenges of managing FAIR (Findable, Accessible, Interoperable and Reusable) (Hagstrom, 2014) data have multiplied as the variety, velocity and lifecycle requirements of environmental health data increase. FAIR data sharing relies on careful curation and management of upstream metadata quality, placement in appropriate data sharing environments and an expansive view of the 'data repository' as a constellation of general and specific data repositories that still exhibits the qualities of FAIR. The evolution of FAIR presents new opportunities and requires new ways of looking at the role of iRODS and policy-based data management.

Keywords

FAIR; data sharing; repositories; iRODS

INTRODUCTION

The National Environmental Health Association (NEHA) defines environmental health science as "the science and practice of preventing human injury and illness and promoting well-being" [1]. The National Institute of Environmental Health Sciences has the mission to "discover how the environment affects people in order to promote healthier lives." The importance of data and metadata management in service of this mission cannot be overstated. The NIEHS experience is not unique. Data management challenges, and the conceptual framework of policy-based data management to address these challenges have evolved over the years from data preservation in a single repository, through federated data sharing environments, to a new reality that is a network of heterogeneous generalist and specific repositories. The new distributed world supports FAIR data discovery and sharing, access controls, and distributed analysis.

IRODS AND EVOLVING DATA MANAGEMENT CHALLENGES

iRODS in the digital preservation era

If we trace the evolution of Policy-Based Data Management pioneered by the DICE Center, we see roots in the archival and data preservation community. iRODS evolved from the original Storage Resource Broker (SRB) and was redesigned as an open source data management platform with funding by the National Archives of the United States. In the original conceptual framework, the focus was on the application of management policies over the data life cycle in the form of a data grid, where a rule engine could enforce management policies in order to establish a

trustworthy preservation environment. The data grid created a logical global namespace over a distributed storage environment. This logical view and the policy management focus made iRODS a compelling platform for data preservation, creating layers of abstraction and infrastructure independence from the underlying storage technology. From these capabilities Dr. Reagan Moore was able to form a "Theory of Digital Preservation". This theory says that there are [2]:

- A minimal set of preservation processes (microservices),
- A minimal set of preservation metadata that can describe the preservation environment and effect of the preservation processes,
- A way of assessing the preservation metadata to validate that a repository is consistent (assessment criteria).

The theory posits that given these capabilities, an assertion can be made that a preservation environment is trustworthy. This model is a very concise framework for understanding the purpose of iRODS, even as the context has evolved. This model is still compelling in the world of FAIR data, where trustworthiness is a core requirement.

iRODS in federated research environments

The next era of iRODS, which we can all the "federation era", was ushered in with the launch of the National Science Foundation's DataNet initiative. The DataNet Federation Consortium was one of several DataNet initiatives funded by the National Science Foundation. DFC focused on the iRODS platform as the foundation for data federation. The DataNet Federation Consortium (DFC) retained the archival/repository view of the data grid and added multi-disciplinary sharing of scientific research data via federation mechanisms as a transformative element. DataNet identified functional areas that should be addressed in cyberinfrastructure for multi-disciplinary data-driven science. These areas were [3]:

1. data deposition, acquisition and ingestion,
2. metadata/ontology management,
3. data security,
4. data integration and interoperability,
5. data analysis and visualization.

DFC built on the notion of preservation capabilities and management policies from the data repository era and extended these concepts in a federation as means to automate many of the necessary tasks to serve the DataNet functional areas. DFC demonstrated that capabilities like indexing, publication, and data processing could be included in the scope of policy based data management [4].

In the DFC federation, multiple zones under different administrative control could be linked together, data could be shared between collaborators, and management policies could be enforced at each storage location. The policy managed framework in the context of a federated environment was extended further as a means to automate scientific workflows, including the segmentation of workflow operations such that these operations could be distributed among different parts of the federation. Moore and Rajasekar did acknowledge that more loosely coupled workflows would also be required, and interestingly proposed the capability to characterize the operations of a workflow, including the data management policies that were activated via a vocabulary as an aid to reproducibility. The authors made a prescient observation that the federation's policy capabilities, including the possibility to stage a task at an appropriate storage resource, could work in tandem with loosely coupled workflows, saying "this style of integration of processing pipelines with collection-based data management is expected to become the basis for data-driven research".

iRODS in disaggregated environments

We observe that iRODS is moving into a third era, and recent experiences at NIEHS reinforces this notion. We view this through the lens of our own data management challenges in Environmental Health Science. As we moved from the data preservation era to the federated data collaboration era, we saw that the "Theory of Digital Preservation" and the policy-based data management approach remained at the heart of the matter and this continues to be the case. The policy-based data management conceptual frameworks still apply and can help keep iRODS relevant.

There are several forces that characterize what we will call the "disaggregated environments" era. These are:

- The breaking of tightly coupled federation, replaced with multi-platform standards,
- The emphasis on highly structured data models,
- The focus on data associated with publications, with attendant focus on reproducibility and provenance,
- Cloud architectures and distributed data pipelines,
- The rise of containers and standard workflow languages,
- The requirement to manage sensitive data and honor data usage agreements.

The breaking of tightly coupled federation

As outlined above, iRODS has its roots in digital preservation and archiving. A data grid was a zone of control that provided a global logical namespace over multiple storage resources. In the federation era the focus shifted to federated zones representing multiple institutions, where iRODS still was the ultimate technology stack and policy and automation were largely under iRODS control. Federation remains a powerful mechanism for data sharing across collaborating institutions, however the data management challenges now extend far beyond the federation boundary.

In the NIH Strategic Plan for Data Science [5], much attention is paid to what is called the "biomedical research data-resource ecosystem". The strategic plan notes that there is a proliferation of repositories, including institutional repositories external to NIH, along with data repositories run by the various branches of NIH. In addition, there are multiple specialized repositories where subsets of research data are deposited. For example, a study may produce gene expression data, which is to be deposited into GEO, the Gene Expression Omnibus [6], while also producing sequencing data to be submitted into the Sequence Read Archive [7]. Meanwhile, data sets associated with a publication may end up deposited in a general repository, resulting in a patchwork of repositories associated with a study, publication, or research question. A great illustration of the problem is the existence of a guide to the diverse ecosystem of generalist and specific repositories relevant to biomedical research at NIH produced by the National Library of Medicine [8], currently populated with 66 entries for domain-specific repositories and a half-dozen generalist repositories. It is evident that federation models explored in the DFC era will not apply in such a fragmented ecosystem.

In response to this, iRODS should refine the policy-based data management philosophy, elevating policy management in two areas. First, the expansion of support for policy management of metadata is required, making metadata management a first-class concern. In this new era, especially as there is a greater focus on data harmonization and the employment of standard vocabularies and common data elements, policy-based management approaches can strengthen iRODS ability to serve as the canonical data and metadata preservation and management environment. Management policies can help maintain and validate metadata standards and automate many of the ingest and transformation tasks that underlie distribution of data to a network of loosely-coupled repositories. Second, the focus on policies as a means to automate tasks like indexing and publishing make iRODS an attractive solution for creating data coordination and data submission nodes that manage distribution while maintaining metadata that can link distributed data together.

The policy approach is well suited to the requirements of a canonical repository of data and associated metadata with a focus on metadata acquisition and quality control. In the NIEHS case this emphasizes the use of standard controlled vocabularies for curation and validation. Upstream data collection from sample and project submission along with the ingestion of data from laboratory information management (LIMS) systems and analysis workflows provides much-needed data curation. Policy-based metadata management enables researchers to generate valid submissions to various general and specific repositories, and NIEHS is currently providing tools for publishing to GEO (Gene Expression Omnibus). If iRODS can play a role in mediating data submissions, it can then retain accession information and pointers to various repositories where data is deposited, enabling FAIR data discovery across disaggregated storage locations. The COPO project [9] describes a data submission brokering service with a similar design goal. COPO provides general metadata curation tools, applying community developed vocabularies and ontologies for curation and validation and then uses these metadata to broker data submissions to target repositories. Instrumenting these publishing and distribution tasks allows foldback of additional metadata curation as well as allows establishment of links between source datasets and publications and analysis data sets in external repositories. This can aid in later cross-repository searching.

The emphasis on highly structured data models

The NIH has placed an emphasis on Common Data Elements (CDEs) as an important enabler for FAIR data sharing in a highly distributed data ecosystem. Rubenstein and McInnes [10] observed that:

> *One of the main obstacles to advancing biomedical research is the inability to exchange and share data and knowledge. This is the result of: data collected using different terminologies, databases being established with lack of interoperability and with no linkage between them, negative results and lessons learned not being shared, and resources (including funding and patient population) being used in duplicated efforts with no coordination and collaboration.*

To this end, NIEHS has convened an Environmental Language Health Collaborative to bring together interested parties to "advance community development and application of a harmonized language for describing Environmental Health Science (EHS) research" [11]. The development of CDEs and community developed ontologies and controlled vocabularies improves interoperability.

While the curation and validation of metadata using vocabularies and ontologies is critical, the data model of the repository itself and the relations between objects is equally important. For example, the Gen3 Data Commons explicitly build a commons architecture on highly structured data model, represented as a graph, and with services for indexing, submitting data, and searching the graph representation for data objects of interest [12, p. 3]. One of the base assumptions in Gen3 is that there are multiple commons connected by a core set of API and services. This is described as a "narrow middle" architecture [13]. These services include indexing and search capabilities and these capabilities are explicitly built around a structured data model. In the view of Grossman, this deep metadata model is one of the differentiating factors between a "data commons" and a "data lake", where a data lake is characterized as "when data are stored simply with digital IDs and metadata (shallow indexing), but without a data model" [14]. This data lake versus data commons distinction is important to weigh as metadata capabilities, such as metadata templates are being considered by the iRODS Consortium [15]. While the initial discussions are on flat schema for metadata, it is a matter to consider whether iRODS would need to support graph-like structures, or whether the platform would rather delegate this level of sophistication to an external service.

The focus on data associated with publications, with attendant focus on reproducibility and provenance.

Research data at NIEHS is utilized in many ways, and diverse research outputs are produced and published, ranging from data sets to reports to papers submitted to peer-reviewed journals. In all disciplines, there is a definite movement towards making research data a first-class concern on par with a published paper. When a conclusion or figure is presented in a scientific publication, the ability to cite the original data and to show how a conclusion or figure can be reproduced is now a requirement. Force11 has published a Joint Declaration of Data Citation Principals that captures this new reality [16], observing that "data must be accorded due importance in the practice of scholarship and in the enduring scholarly record. In other words, data should be considered legitimate, citable products of research. Data citation, like the citation of other evidence and sources, is good research practice and is part of the scholarly ecosystem supporting data reuse." Viewed through the policy lens, these Force11 data citation principles fit well in the data lifecycle framework described by Moore and Rajasekar, where they observed that each stage of the research data lifecycle was governed by an explicit set of policies that represented a "community consensus on data sharing" [17]. The notion that iRODS can provide automation and policy control for management of research data sets at the publication side of the data lifecycle can in turn expand the sphere of control of management policies into the distribution and publication functions, expanding the scope of metadata and allowing better determination of the authenticity and reproducibility of research results, all the way back to the original data sets.

There are several challenges in the reproducibility aspects of FAIR that NIEHS is experiencing. Our data generation strategy is currently very fragmented. With the gradual formalization of data management at the Lab Information Management System (LIMS) level there are opportunities to extract metadata related to the various assays and procedures that were carried out. The Office of Data Science at NIEHS is championing efforts to formalize the vocabularies used to describe samples and assays and data handling in order to improve the upstream metadata quality. Data analysis (mapping to and updating the "Data Processing Pipeline" stage in the data lifecycle) is an area of difficulty, as the workflows and procedures used to analyze data are often detached from the control of any machine-actionable policies. There are three strategies we are focused on:

- Championing the use of notebooks as a way of capturing analysis,
- Championing the use of standard workflow languages and containerization to capture pipelines in detail,
- Considering aids and tooling for data submission for publication and to repositories as a way to capture links between publications and the source data.

Each of these strategies do not automatically establish the "R" in fair, however, they do present opportunities to improve the daily experience of analysts and researchers, improve the ability to share and record analysis steps for day-to-day work, and finally to instrument the analysis process for improved metadata and provenance capture. For example, Samuel et al. found that employing Jupyter notebooks enhanced the provenance capture for machine learning based analysis [18]. As we will note in our discussion of the rise of containers and standard workflow languages, the enhancements to portability, the ability to share or utilize best-practices pipelines, as well as the provenance and reproducibility benefits are significant. We saw in the COPO model how the publishing phase presents opportunities for management policies to gather provenance information. Expanding the ability of management policies to "see" activities and to gather metadata about processes and provenance can lead to better reproducibility of research results.

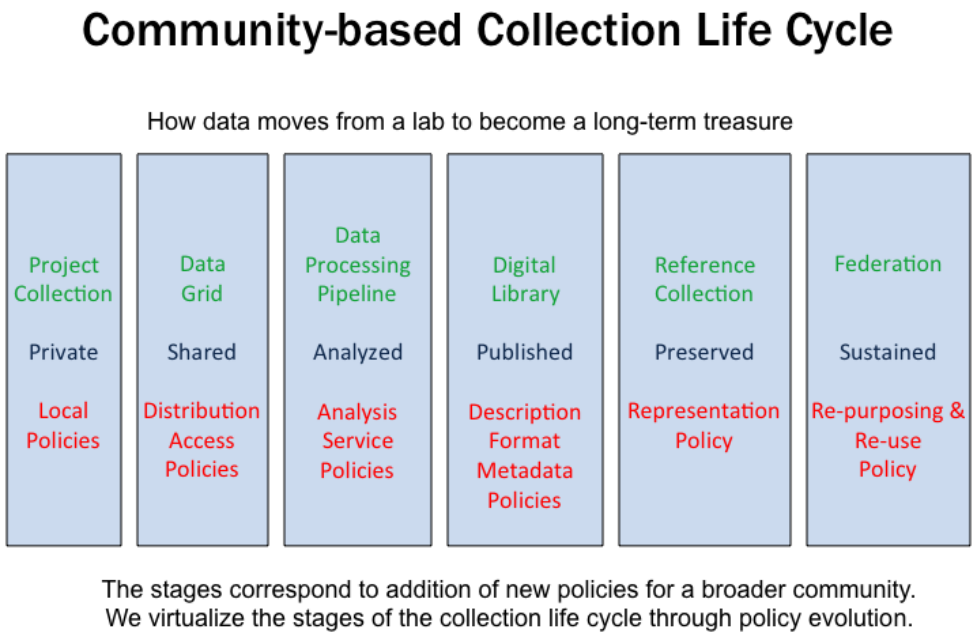


Figure 1. The research data lifecycle and relevant policies, from [17]

Cloud architectures and distributed data pipelines

One of the biggest disrupters to data management has been the rise of the cloud. While there are compelling capabilities in iRODS to connect to and manage cloud storage, there are also new challenges and opportunities in extending iRODS as a part of the broader architecture beyond storage. The notion of federation is present in discussions about cloud-based analysis but federation in these disaggregated environments is quite different, more along the lines of the "narrow middle" architecture with a sparse set of services that link otherwise independent environments. NIH, in their strategic planning, has identified several focus areas, including establishing researcher identity, managing data usage agreements, running computational tasks across multiple endpoints while enforcing fine grained authentication/authorization across multiple cloud providers as areas of concern. Each of these are natural areas where policy-based data management concepts can play a significant role.

It is the ability to rapidly scale to run complex analysis, taking advantage of the compute power and available tools, that is driving the migration of biomedical science to the cloud. The National Institutes of Health (NIH) acknowledged this migration in its original data commons pilot. The original pilot identified many challenges, including the portability of analysis to multiple cloud architectures, and the importance of minimizing ingress and egress charges, running an analysis task as close to the data as possible [19]. The Global Alliance for Genomics and Health (GA4GH) has worked to standardize workflows and pipelines, and to achieve the goals of portability and efficiency through the use of standard workflow languages, along with cloud-neutral data access and task execution standards [20]. Using standards such as the Data Repository Service (DRS) and Workflow Execution Service (WES), researchers have been able to demonstrate qualities of a federated data ecosystem [21].

The iRODS Consortium has considered "data-to-compute" and "compute-to-data" as general platform capabilities. Platforms such as CyVerse Discovery Environment have made great strides in providing "bring-your-own-compute" to their researcher community [22]. Strategies for iRODS to enable of these narrow-middle services and support emerging standards and conventions to support distributed analysis are a rich area for exploration. There has been prior work using iRODS to support distributed analysis on cloud platforms under the umbrella of the original NIH Data Commons Pilot as well as under the sciDAS program [23] which demonstrated the ability to distribute tasks based on a calculation of performance and cost factors. This work parallels many of qualities of the current GA4GH federated analysis projects, including the Task Execution Service (TES) [24]. These examples show that there may be multiple ways that iRODS could serve as a platform for distributed data pipelines.

The rise of containers and standard workflow languages

In large part, the ability to run pipelines and analysis in distributed environments relies on the use of containers such as Docker or Singularity. The CyVerse Discovery Environment is a great example within the iRODS Community, where researchers can create and containerize tools, annotate them, and add them into the research environment to run and share. Repositories such as BioContainers [25] and Dockstore [26] allow researchers to share containerized tools and workflows written in standard workflow languages such as CWL [27] and NextFlow [28]. The portability and reproducibility of containers is well understood. Formalizations for running containerized tasks on data at remote storage locations (as CyVerse does to some extent) using standard TES-compliant interfaces may be an interesting and beneficial capability.

The requirement to manage sensitive data and honor data usage agreements

If iRODS and policy-based data management have any intersection with capabilities that are desired in the proposed NIH Data Commons architecture, they would intersect with the need for fine-grained and dynamic decisions on data access. Efforts are underway to develop ontologies that can describe data usage , as well as standard systems, built on existing standards such as OpenID Connect and OAuth, for identifying researchers. There are efforts at NIH and at GA4GH to create something akin to a data passport. This system was described by Cabili et al [29] as a "Library Card":

> *A Library Card would encode identity and other attributes of a researcher as a set of standardized claims. For example, institutional affiliation may be recorded in a claim, along with a level of assurance regarding the claim: (a) self-assertion (b) institutional email (c) proof of support of the claim by an authorized third party within the institution. These claims, and others like them, are recorded by the Library Card issuer and provided over secure protocols to relying parties who will in turn make access decisions based upon these claims*

iRODS policies and the ability to make decisions based on role, status, or dynamic factors such as acknowledgment of terms or data usage agreements would be a natural area to investigate, especially were iRODS is mediating access to research data sets. NIEHS did demonstrate a Data Repository Service (DRS) implementation that can run on native iRODS [30, p.]. This DRS implementation does have, in later planned iterations, the ability to provide GA4GH Data Passports [31] that can provide a framework for exploring how iRODS policies can interact with GA4GH Data Passports.

CONCLUSIONS

This paper observes that a third "era" of policy-based data management has arrived, driven by the disaggregated nature of repositories, the rise of the cloud, and the need to create federations across disparate systems for data-driven science. The capabilities demonstrated and developed in prior eras of iRODS of have not disappeared. However, the challenge and opportunity presents itself to build upon the success of the policy-based data management model and the core capabilities in the iRODS platform and learn how they apply and can advance the platform in this new federation era.

REFERENCES

[1] Global Alliance for Genomics and Health, "Definitions of Environmental Health | National Environmental Health Association: NEHA." https://www.neha.org/about-neha/definitions-environmental-health (accessed Jul. 12, 2021).

[2] R. Moore, "Towards a theory of digital preservation," Int. J. Digit. Curation, vol. 3, no. 1, pp. 63–75, 2008.

[3] J. W. Lee, J. Zhang, A. S. Zimmerman, and A. Lucia, "DataNet: An emerging cyberinfrastructure for sharing, reusing and preserving digital data for scientific discovery and learning," AIChE J., vol. 55, no. 11, pp. 2757–2764, Nov. 2009, doi: 10.1002/aic.12085.

[4] R. W. Moore and A. Rajasekar, "Reproducible Research within the DataNet Federation Consortium," p. 8.

[5] "NIH Strategic Plan for Data Science."

[6] R. Edgar, M. Domrachev, and A. E. Lash, "Gene Expression Omnibus: NCBI gene expression and hybridization array data repository," Nucleic Acids Res., vol. 30, no. 1, pp. 207–210, Jan. 2002, doi: 10.1093/nar/30.1.207.

[7] R. Leinonen, H. Sugawara, M. Shumway, and on behalf of the International Nucleotide Sequence Database Collaboration, "The Sequence Read Archive," Nucleic Acids Res., vol. 39, no. suppl_1, pp. D19–D21, Jan. 2011, doi: 10.1093/nar/gkq1019.

[8] "Data Sharing Resources." https://www.nlm.nih.gov/NIHbmic/nih_data_sharing_repositories.html (accessed Jul. 13, 2021).

[9] A. Etuk et al., "COPO: a metadata platform for brokering FAIR data in the life sciences," bioRxiv, 2019, doi: 10.1101/782771.

[10] Y. R. Rubinstein and P. McInnes, "NIH/NCATS/GRDR® Common Data Elements: A leading force for standardized data collection," Contemp. Clin. Trials, vol. 42, pp. 78–80, May 2015, doi: 10.1016/j.cct.2015.03.003.

[11] "Environmental Health Language Collaborative - Harmonizing Data. Connecting Knowledge. Improving Health.," National Institute of Environmental Health Sciences. https://www.niehs.nih.gov/research/programs/ehlc/index.cfm (accessed Jul. 13, 2021).

[12] "Gen3 - Set up Gen3." http://gen3.org/resources/operator/ (accessed Jul. 13, 2021).

[13] R. L. Grossman, "Progress Towards Cancer Data Ecosystems," Cancer J. Sudbury Mass, vol. 24, no. 3, pp. 122–126, 2018, doi: 10.1097/PPO.0000000000000318.

[14] R. L. Grossman, "Data Lakes, Clouds, and Commons: A Review of Platforms for Analyzing and Sharing Genomic Data," Trends Genet., vol. 35, no. 3, pp. 223–234, Mar. 2019, doi: 10.1016/j.tig.2018.12.006.

[15] The iRODS Consortium, TRiRODS: Metadata Templates in iRODS. Accessed: Jul. 13, 2021. [Online Video]. Available: https://www.youtube.com/watch?v=_b4AvhhG7mc&ab_channel=TheiRODSConsortium

[16] M. Crosas, “Joint Declaration of Data Citation Principles - FINAL,” FORCE11, Oct. 30, 2013. https://www.force11.org/datacitationprinciples (accessed Jul. 14, 2021).

[17] R. W. Moore, A. Rajasekar, M. Conway, W. Schroeder, and M. Wan, “White Paper: National Data Infrastructure for Earth System Science,” White Pap. Submitt. EarthCube Proj. Httpearthcube Ning Comgrouptechnology-Resolut. Last Access 24 May 2013, 2011, Accessed: May 29, 2015. [Online]. Available: http://semanticommunity.info/@api/deki/files/13791/=003_Moore.pdf

[18] S. Samuel, F. Löffler, and B. König-Ries, “Machine Learning Pipelines: Provenance, Reproducibility and FAIR Data Principles,” ArXiv200612117 Cs Stat, Jun. 2020, Accessed: Jul. 14, 2021. [Online]. Available: http://arxiv.org/abs/2006.12117

[19] National Institutes of Health, “NIH Data Commons Pilot Phase.” Jun. 16, 2017. [Online]. Available: https://commonfund.nih.gov/sites/default/files/rm-17-026_commonspilotphase.pdf

[20] “GA4GH Cloud Workstream.” https://www.ga4gh.org/work_stream/cloud/#proposed-solution (accessed Jul. 14, 2021).

[21] “Realising the Genomics Ecosystem: ELIXIR Plays Key Role in GA4GH 2020 Connection Demos,” ELIXIR, Sep. 30, 2020. https://elixir-europe.org/news/realising-genomics-ecosystem-elixir-plays-key-role-ga4gh-2020-connection-demos (accessed Jul. 15, 2021).

[22] U. K. Devisetty, K. Kennedy, P. Sarando, N. Merchant, and E. Lyons, “Bringing your tools to CyVerse Discovery Environment using Docker,” F1000Research, vol. 5, p. 1442, Jun. 2016, doi: 10.12688/f1000research.8935.1.

[23] F. Jiang, C. Castillo, and S. Ahalt, “A Cloud-Agnostic Framework for Geo-Distributed Data-Intensive Applications,” p. 10.

[24] “GA4GH TES API: Bringing compatibility to task execution across HPC Systems, the Cloud and Beyond.” https://www.ga4gh.org/news/ga4gh-tes-api-bringing-compatibility-to-task-execution-across-hpc-systems-the-cloud-and-beyond/ (accessed Jul. 15, 2021).

[25] F. da Veiga Leprevost et al., “BioContainers: an open-source and community-driven framework for software standardization,” Bioinformatics, vol. 33, no. 16, pp. 2580–2582, Aug. 2017, doi: 10.1093/bioinformatics/btx192.

[26] B. D. O’Connor et al., “The Dockstore: enabling modular, community-focused sharing of Docker-based genomics tools and workflows,” F1000Research, vol. 6, p. 52, Jan. 2017, doi: 10.12688/f1000research.10137.1.

[27] P. Amstutz et al., “Common Workflow Language, v1.0,” Jul. 2016, doi: 10.6084/m9.figshare.3115156.v2.

[28] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, “Nextflow enables reproducible computational workflows,” Nat. Biotechnol., vol. 35, no. 4, pp. 316–319, Apr. 2017, doi: 10.1038/nbt.3820.

[29] M. N. Cabili et al., “Simplifying research access to genomics and health data with Library Cards,” Sci. Data, vol. 5, no. 1, p. 180039, Mar. 2018, doi: 10.1038/sdata.2018.39.

[30] “michael-conway/irods-ga4gh-dos: GA4GH Data Object Service for iRODS,” GitHub. https://github.com/michael-conway/irods-ga4gh-dos (accessed Jul. 21, 2021).

[31] “linking DRS with Passport Visa · Issue #339 · ga4gh/data-repository-service-schemas,” GitHub. https://github.com/ga4gh/data-repository-service-schemas/issues/339 (accessed Jul. 21, 2021).

Go-iRODSClient, iRODS FUSE Lite, and iRODS CSI Driver: Accessing iRODS in Kubernetes

Illyoung Choi
CyVerse
iychoi@email.arizona.edu

John H. Hartman
Dept. of Computer Science
University of Arizona
jhh@cs.arizona.edu

Edwin Skidmore
CyVerse
edwin@cyverse.org

ABSTRACT

As developers increasingly adopt the cloud-native paradigm for application development, Kubernetes has become the dominant platform for orchestrating their cloud-native services. To facilitate iRODS access in Kubernetes, we developed an iRODS CSI (Container Storage Interface) driver. The driver provides on-demand data access to the iRODS server using multiple connectivity modes and exposes a file system interface to Kubernetes Pods, thereby allowing cloud-native services to access iRODS without staging data within the containers. In this paper, we introduce the design and functionality of the iRODS CSI driver. We also introduce two sub-projects: Go-iRODSClient and iRODS FUSE Lite. Go-iRODSClient is an iRODS client library written in Go. iRODS FUSE Lite is a complementary FUSE-based iRODS storage backend to the iRODS CSI driver that is implemented using Go-iRODSClient to provide improved performance and enhanced capabilities compared to iRODS FUSE. We expect Go-iRODSClient, iRODS FUSE Lite, and the iRODS CSI driver to be indispensable tools for integrating iRODS into cloud-native applications.

Keywords

iRODS, client, Kubernetes, Container Storage Interface, FUSE, data management

INTRODUCTION

The cloud-native paradigm [1] is becoming increasingly popular for application development as the demand for building and running scalable applications in dynamic environments such as cloud increases. The paradigm has led to Kubernetes [2] becoming the dominant platform for orchestration of cloud-native services. CSI (Container Storage Interface) [3] is a standard interface for exposing storage systems in Kubernetes. The CSI mounts data stored in remote storage on the Linux directory hierarchy within Kubernetes Pods. We developed an iRODS CSI (Container Storage Interface) driver to facilitate iRODS access in Kubernetes. The driver manages iRODS file system mounts and provides a file system interface to Kubernetes Pods, thereby allowing cloud-native services to access iRODS via the file system interface without staging data in the containers. The iRODS CSI driver supports three iRODS file system clients: iRODS FUSE Lite, davfs2 [4], and the NFS client. Kubernetes users who create volumes (Kubernetes Persistent Volumes) choose the client that best meets their needs for use with the iRODS CSI driver. We compare the pros and cons of the clients in this paper.

We also present iRODS FUSE Lite, a new iRODS file system client. iRODS FUSE Lite is a reimplementation of the existing iRODS FUSE client [5], that was written in C++ using the iRODS C API library. iRODS FUSE has several unfixed bugs (e.g., files not replicated) and the code has not been maintained for a long time. Implementing new features and fixing the bugs requires significant effort as the client is written in C/C++. In addition, iRODS FUSE had portability challenges as it relies on the iRODS C API library. The library does not officially support some newer releases of Linux distros (e.g., Ubuntu 20.04). We re-implemented iRODS FUSE Lite in Go to address these issues and to improve portability.

iRODS UGM 2021, June 8-11, 2021, Virtual

We also present Go-iRODSClient, a library written in Go that implements the iRODS protocol. The library is used in iRODS FUSE Lite and the iRODS CSI driver for iRODS access. There is an existing Go library (GoRODS [6]), but we did not use it because it has a dependency on the iRODS C API library, which causes a portability issue. We implemented it in pure Go to improve portability.

This paper introduces the design and functionality of the iRODS CSI driver, iRODS FUSE Lite, and Go-iRODSClient. This paper also provides performance benchmark results. In short, Go-iRODSClient's data transfer bandwidth is similar to Python-iRODSClient [7], which is an iRODS API library for Python. iRODS FUSE Lite shows significant improvement in data transfer bandwidth compared to the existing iRODS FUSE client -- a 25% improvement for data upload and a 157% improvement for data download bandwidth. iRODS FUSE Lite has the highest data upload bandwidth but the lowest data download bandwidth among iRODS file system clients. We plan to improve the data download bandwidth of iRODS FUSE Lite to make it more useful.

GO-IRODSCLIENT

Go-iRODSClient is an iRODS Client library written in pure Go and provides APIs for iRODS access. Python-iRODSClient [7] served as a reference implementation for the Go-iRODSClient. Leveraging Go's portability, the library can be used in any operating system that Go supports without dependency and compatibility issues.

Initially, we attempted reusing Python-iRODSClient as the basis for the iRODS CSI driver. However, this approach caused a portability issue, causing unnecessary driver container image bloating for installation of Python packages. After replacing the functionality with Go-iRODSClient, the size of the driver container image was reduced by approximately 64%, from 259MB to 93MB. The reduced image size enables fast image download and requires less disk storage space.

The current version of Go-iRODSClient (v0.4.5) provides the following functionality:

- **Authentication**: The client authenticates iRODS users via password or PAM.
- **Communication**: The client can communicate to the iRODS server via TCP/IP or SSL. The client also manages a connection pool for concurrent accesses.
- **Collection Management**: The client can create, move, delete, and search Collections.
- **Data Object Management**: The client can create, move, delete, replicate, search, read, and write Data Objects.
- **Metadata Management**: The client can add, list, and delete iRODS Metadata for Resources, Data Objects, and Collections.
- **User Management**: The client can add, list, and delete iRODS users or groups.
- **Caching**: The client can cache frequently accessed information to improve access performance, such as a list of entries in a Collection, Data Object/Collection status information, access control lists (ACLs), and user/group information.

Go-iRODSClient is currently under active development as we continue to implement additional APIs. Go-iRODSClient is currently used in several CyVerse projects, such as iRODS FUSE Lite, the iRODS CSI driver, and CyVerse DataWatch [8]. CyVerse plans to use the library more widely to develop new services and tools.

IRODS FUSE LITE

iRODS FUSE Lite is an iRODS file system client that enables Linux users to mount iRODS data on the Linux directory hierarchy. Users can access iRODS data via file system accesses on the mount point. It is a reimplementation of the existing iRODS FUSE client that was written in C++ using the iRODS C API library. iRODS FUSE Lite was implemented in Go to address the stability and portability issues.

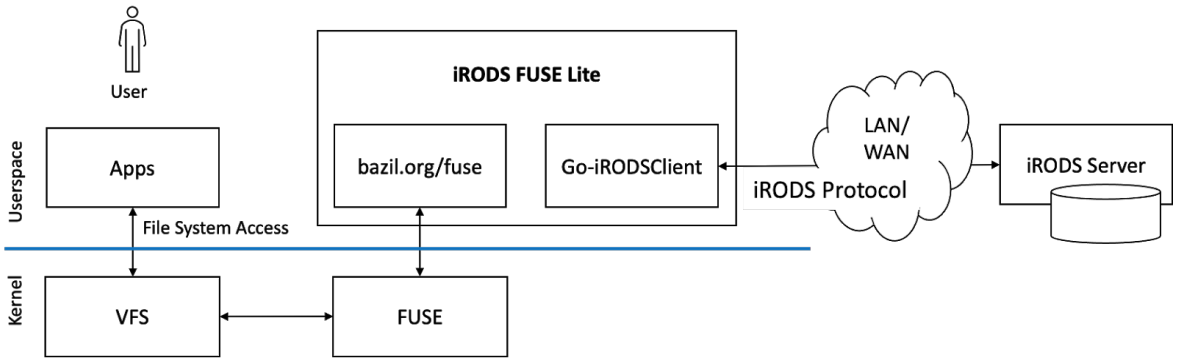


Figure 1. iRODS FUSE Lite implements the FUSE interface to enable iRODS access via the file system interface.

iRODS FUSE Lite implements the FUSE interface [9] to enable iRODS access via the file system interface (Figure 1). It uses the "bazil.org/fuse" library [10] that provides the FUSE interface in Go. iRODS FUSE Lite can run on any FUSE-compatible systems such as Linux and MacOS.

iRODS FUSE Lite relies on Go-iRODSClient for iRODS access. iRODS FUSE Lite uses Go-iRODSClient for authentication, data access, parallel connection management, and metadata caching. iRODS FUSE Lite also implements data access optimizations, such as asynchronous data writing.

The current version of iRODS FUSE Lite (v0.2.3) supports the following features:

- **Data Access**: The client provides listing, creation, update, and deletion of iRODS Data Objects and Collections.
- **Authentication**: The client authenticates iRODS users via password. In addition, the client supports proxy authentication that allows an admin to login on behalf of another user.
- **Permission Mapping**: The client maps iRODS Permissions to Linux permissions.
- **Data Caching**: The client caches recently accessed content of iRODS Data Objects. The data caching is done by Linux page caching.
- **Asynchronous Data Writing**: The client buffers written data into local disks and transfers to iRODS in the background asynchronously to improve data write performance. The client flushes buffered data before closing the file handle to ensure close-to-open data consistency.
- **Path Mapping**: The client can map iRODS Data Objects or Collections to files or directories on the Linux directory hierarchy. The mappings are customizable.

Unlike traditional Unix permissions, iRODS has a multi-owner model with a linear permission system, which means permissions for a Data Object or a Collection can be set to OWN, READ, WRITE, or NONE per iRODS user. An iRODS user with OWN permission can read, write and delete files and change permissions for others. WRITE permission enables file read and write. READ permission enables reading the content in a file.

The current implementation of iRODS FUSE Lite maps iRODS permissions to the traditional Unix permissions in a simple way. If the current user logged-in has OWN permission for an iRODS Data Object (or Collection), iRODS FUSE Lite assigns read, write, and execute permissions (rwx------). If the current user logged-in has WRITE permission, read and write permissions (rw-------) are assigned. For READ permission, only read permission (r--------) is assigned. iRODS FUSE Lite uses the simple permission mapping because iRODS users may not exist in the local system so it cannot map the iRODS users to local Linux users. The same approach is also used in NFSRODS [11].

iRODS FUSE Lite has two new features that are absent in the existing iRODS FUSE client: proxy authentication and custom path mapping. Proxy authentication allows an iRODS admin to login and access iRODS data on behalf

of another iRODS user. The iRODS accesses are emulated for the proxied user according to the user's access rights. Doing so, the cloud service can provide a web interface for accessing a user's iRODS data or broker iRODS access to other apps without having direct access to the user's password.

Custom path mapping enables users of iRODS FUSE Lite to customize path mapping between the source iRODS data and the destination mount location. By default, iRODS FUSE Lite mounts an iRODS Collection on the Linux directory hierarchy. Data Objects and Collections contained in the Collection are also mapped to files and directories under the mount point, respectively. However, the mapping is also customizable for various purposes, such as to mount multiple Data Objects and Collections spread over different paths, or to exclude Data Objects containing sensitive data from mounting. We present a detailed use case of the custom path mapping in the Integration Example section.

iRODS FUSE Lite is also under active development. We plan to add more data access optimizations, such as prefetching and caching. We also continue to improve client stability.

IRODS CSI DRIVER

Data written by apps running in Kubernetes are volatile. Data written in a Kubernetes Pod is lost after the termination of the Pod. To store data persistently, Kubernetes provides Persistent Volumes [12]. Persistent Volumes provide an abstraction of storage and Container Storage Interface (CSI) mediates Persistent Volumes and storage. Drivers of the CSI implement storage specifics. We developed an iRODS CSI driver to enable iRODS to function as persistent storage in Kubernetes.

CSI relies on Linux file system mount for data access. By doing so, it does not require modifications to apps using a local file system for input and output to access Persistent Volumes. The iRODS CSI driver uses iRODS file system clients (such as iRODS FUSE Lite) to mount iRODS data. The iRODS CSI driver supports three iRODS file system clients to mount iRODS data: iRODS FUSE Lite, davfs2, and the NFS client. One of the clients is selected when creating a Persistent Volume.

First, iRODS FUSE Lite implements the iRODS protocol and communicates with iRODS servers directly. This is advantageous because iRODS FUSE Lite can be used for any iRODS setup. Unlike iRODS FUSE Lite, davfs2 and the NFS client require additional services, such as DavRODS [13] and NFSRODS [11]), for protocol conversion. In addition, iRODS FUSE Lite provides iRODS-specific features, such as proxy authentication.

Second, davfs2 implements the WebDAV protocol [14], which is a data access protocol over HTTP. Because it is HTTP-based, web-cache services (e.g., Varnish Cache [15]) can be used to improve read performance for frequently requested data. However, because iRODS does not provide native support for the WebDAV protocol, an additional service (DavRODS [13]) is required for protocol conversion between the WebDAV and iRODS protocols.

Lastly, the NFS client is a well-known file system client used by many network storage systems. It communicates with storage servers via the NFS protocol [16]. The NFS client is available in most computing environments because of its popularity. However, because the NFS protocol is not natively supported by iRODS, an additional service (NFSRODS [11]) is required for protocol conversion. In addition, the NFS client is not suitable for providing iRODS access in untrusted networks as it only supports host authentication (AUTH_SYS).

The iRODS CSI driver (v0.3.1) has two component modules: controller and node (Figure 2). The controller module runs on several Kubernetes cluster nodes in a replication mode for fault-tolerance and processes dynamic volume provisioning requests. The node module runs on every Kubernetes cluster node and is responsible for processing static volume provisioning requests.

The static volume provisioning mode is useful for accessing existing data stored in iRODS. In the static volume provisioning mode, Kubernetes users create Persistent Volumes and Persistent Volume Claims. A Persistent Volume specifies the iRODS data to mount, login credentials, and mount options for iRODS clients. The node module in the iRODS CSI driver receives the creation request of the Persistent Volume and mounts iRODS data on the Linux directory hierarchy. Then, Persistent Volume Claims are used to request access to the Persistent Volume.

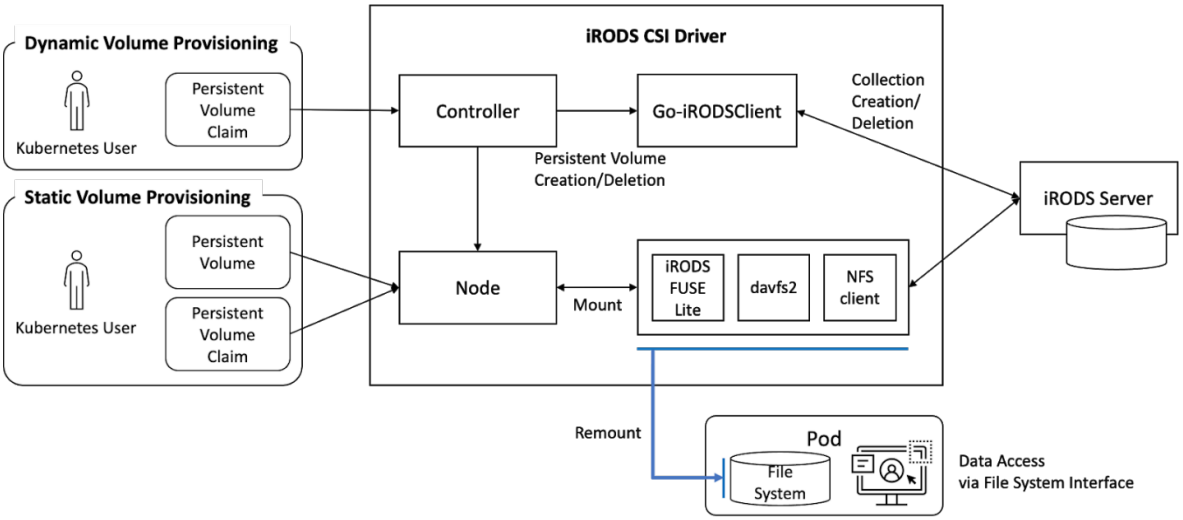


Figure 2. iRODS CSI driver provides Kubernetes users with iRODS data access by processing volume provisioning requests.

The dynamic volume provisioning mode is useful for users who do not require access to existing iRODS data but require storage for archives. In the dynamic volume provisioning mode Persistent Volumes are automatically created by the controller module upon the creation of Persistent Volume Claims. The controller module creates Persistent Volumes dynamically using pre-configured iRODS access information (e.g., iRODS data to be mounted and login credentials). Instead of mounting an existing iRODS data, the iRODS CSI driver creates a new empty iRODS Collection per Persistent Volume. The created Persistent Volumes are then passed to the node module. When a Persistent Volume Claim is reclaimed, the Persistent Volume associated with it is deleted by the controller module.

We are integrating the iRODS CSI driver to CyVerse Discovery Environment (DE), a scientific research cloud platform. A detailed description of the integration is given in the next section.

INTEGRATION EXAMPLE

CyVerse Discovery Environment (DE) [17]–[19] is a cloud computing service that allows scientists to perform research apps. The service uses Kubernetes for computing resource orchestration and iRODS for data storage. CyVerse Data Store [20] is the iRODS-based storage service that stores large community datasets and user data.

The current implementation uses a data staging approach for exposing iRODS user data into containerized apps. To run CyVerse DE apps, scientists select input data (iRODS Data Objects or Collections) and an output directory (iRODS Collection) in CyVerse Data Store. Before executing the app, CyVerse DE downloads the input data to a local disk in the Pod in which the app will run. The app reads the local copies, performs analysis, then produces output on the local disk. CyVerse DE then uploads the output to CyVerse Data Store after the app terminates.

The staging approach has a few drawbacks. First, it produces local copies of input data. This makes it challenging to process large input data that exceeds the local disk capacity. As the service runs multiple apps simultaneously, the local disk capacity is also shared among the apps currently running. Each running app gets only a small portion of the local disk capacity. Second, apps must wait for completion of staging input and output data. This prevents computation and data transfer from overlapping; it may cause longer execution times of the apps [21]. Lastly, staging requires unused parts of input files to be transferred and stored. Apps may require only some parts of input data for analysis, (i.e., reading some records in a database file). In this case, downloading the unused parts of input files wastes network bandwidth, disk space, and time. The iRODS CSI driver addresses these drawbacks by transferring data on-demand when it is accessed, instead of staging it.

For the integration, we first configured the iRODS CSI driver to use proxy authentication. Using proxy authentication, CyVerse DE does not have direct access to iRODS users' passwords for user operations on data in CyVerse Data Store. We configured the iRODS access information (e.g., iRODS host address and admin credentials) for proxy authentication using Kubernetes Secrets [22]. A Secret is a storage for small sensitive data, such as passwords or tokens. Data stored in the Secret are encrypted, making it a good medium for passing sensitive configuration information to the iRODS CSI driver securely. We also configured the iRODS CSI driver to use iRODS FUSE Lite for iRODS access.

We configured the iRODS CSI driver using the static volume provisioning mode. To run an app in CyVerse DE, a scientist selects existing input data and an output directory. CyVerse DE then creates a new Persistent Volume and Persistent Volume Claim with the data paths and iRODS username. The iRODS CSI driver processes the volume creation request and mounts the iRODS data in the Pod before running the app.

CyVerse DE mounts input and output data using a single Persistent Volume. In many cases the input and output data for an app are in different paths. With default path mapping in iRODS FUSE Lite, this requires multiple file system mounts; so multiple Persistent Volume and Persistent Volume Claims are created. Creating hundreds of processes of iRODS FUSE Lite for the mounts on a Kubernetes node will degrade data access performance as they can create thousands of iRODS connections, overloading the system. The custom path mapping enables CyVerse DE to mount the input and output data for an app using a single process of iRODS FUSE Lite.

Figure 3 describes how the path mapping works. CyVerse DE generates path mappings with two subdirectories, "input" and "output". All the input data are mapped under the "input" subdirectory and output data are mapped to the "output" subdirectory. CyVerse DE passes the mappings to the iRODS CSI driver via a Persistent Volume creation request and iRODS FUSE Lite uses the mappings to construct a directory hierarchy.

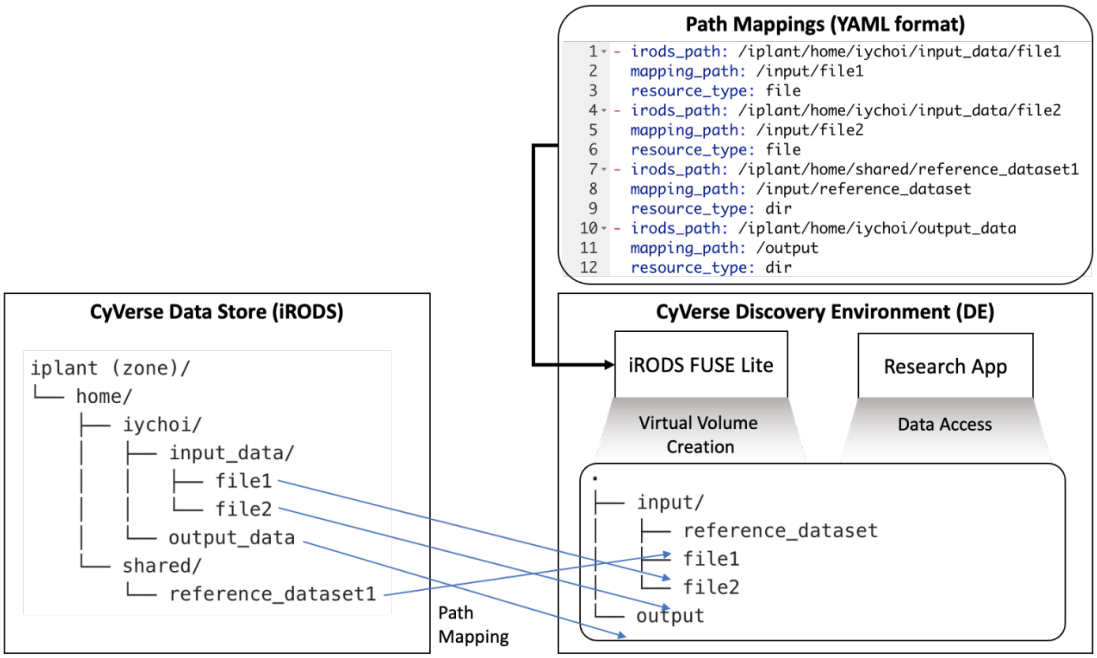


Figure 3. iRODS FUSE Lite and the iRODS CSI driver allow users to manipulate path mappings between iRODS data and local files. The mappings are defined using YAML and passed to iRODS FUSE Lite via the standard input (stdin).

PERFORMANCE EVALUATION

We evaluated data access performance of Go-iRODSClient and iRODS FUSE Lite by comparing them with existing iRODS libraries and file system clients. The performance of the iRODS CSI driver is not evaluated independently because it relies on the iRODS file system clients used in iRODS FUSE Lite evaluation.

The evaluation is performed in a test cluster that runs OpenStack. We used an instance with 2 VCPUs and 8GB RAM for running the iRODS clients and benchmarks. We chose the small instance type similar to end-users' computing environments (e.g., Laptop). We also set up a separate iRODS server on another instance with 4 VCPUs and 8GB RAM. We chose instance types with more VCPUs for the iRODS server for concurrent processing. To evaluate the NFS client and davfs2 we also set up two instances, each with 2 VCPUs and 8GB RAM for NFSRODS and DavRODS, respectively. These servers are only responsible for protocol conversion; large computing resources are not required for our benchmarks.

To evaluate the performance of Go-iRODSClient, we compared it with existing iRODS API libraries, Python-iRODSClient [7] and Jargon [23] that are well-maintained and widely used in the iRODS community. We wrote simple data transfer code using the libraries to measure the elapsed times for transferring 1GB.

To evaluate the performance of iRODS FUSE Lite, we compared it with existing file system clients that provide iRODS access, davfs2 and the NFS client. We used the Linux "cp" command to transfer 1GB data and measured the elapsed time.

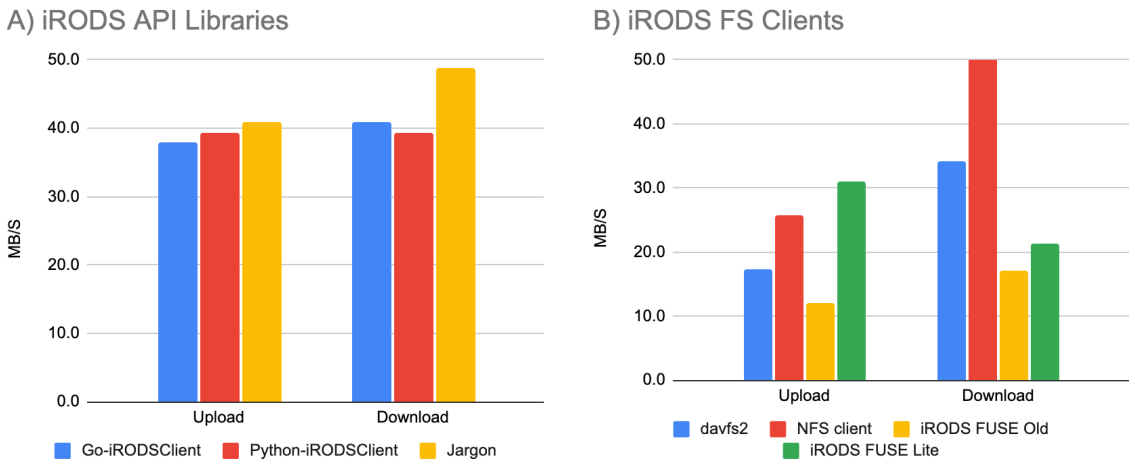


Figure 4. Data transfer performance of iRODS client libraries (A) and file system clients (B). We implemented simple data transfer benchmarks using the libraries for the evaluation. We used "cp" to test data transfer performance of the file system clients.

Go-iRODSClient

Figure 4 (A) shows the data transfer performance of the iRODS API libraries. Go-iRODSClient showed similar performance as Python-iRODSClient. The performance difference was within 4%. This is because Go-iRODSClient was developed with reference to the source code of Python-iRODSClient.

Jargon showed the highest data transfer bandwidth. Jargon showed 19% better performance for download and 8% better performance for upload compared to Go-iRODSClient. We think this is because Jargon continuously transfers the data as the buffered data is consumed. In comparison, data transfer in Go-iRODSClient and Python-

iRODSClient is not pipelined because they transfer data when it is requested on-demand. We plan to implement the transfer optimization in Go-iRODSClient.

We did not use the parallel data transfer APIs supported by Jargon. While the parallel data transfer APIs improve data transfer performance dramatically, it requires disks for storing data received. Thus, the APIs are only useful when uploading and downloading data from and to disks. Also, Go-iRODSClient and Python-iRODSClient do not implement the APIs yet. So, we excluded these APIs in the evaluation. As Python-iRODSClient is going to release new parallel data transfer APIs, we plan to implement the APIs in Go-iRODSClient in the future.

iRODS FUSE Lite

Figure 4 (B) shows the data transfer performance of the iRODS file system clients. iRODS FUSE Lite outperformed the existing iRODS FUSE client (iRODS FUSE Old) for both upload and download bandwidths. iRODS FUSE Lite showed 25% improved data bandwidth for download and 157% improved performance for upload. The existing iRODS FUSE client (iRODS FUSE Old) and iRODS FUSE Lite implement data reads (download) using the same block-based transfer (block size is 64KB). However, we think the new implementation of cache management and connection management is more efficient, improving the bandwidth. iRODS FUSE Lite implements data writes (upload) differently from the existing iRODS FUSE client (iRODS FUSE Old). The existing iRODS FUSE client buffers written data in RAM and transfers them to iRODS asynchronously. As this relies on RAM, however, it cannot buffer large data; only few kilobytes of data are buffered in RAM. This can break transfer pipelining. iRODS FUSE Lite implements disk buffers to buffer even larger data, possibly hundreds of MB. Then transfers buffered data asynchronously. This improved data write (upload) performance significantly than using the existing RAM buffers.

The NFS client showed the highest data read (download) bandwidth. The NFS client showed 185% better performance than iRODS FUSE Lite. The NFS client communicates with iRODS via NFSRODS that is implemented using Jargon for data access. Therefore, it showed similar data read performance as Jargon shown in Figure 4 (A).

Davfs2 showed 60% higher data read (download) bandwidth compared to iRODS FUSE Lite. We think davfs2 implements more efficient data buffering than iRODS FUSE Lite. We will continue to optimize the data transfer and buffering code.

Unlike the download performance, the NFS client and davfs2 did not show good performance for data writes (upload). The NFS client and davfs2 showed 17% and 44% worse performance for data writes than iRODS FUSE Lite. Both the NFS client and davfs2 require additional services, NFSRODS and DavRODS, for protocol conversion, causing overheads. Further investigation is required to measure these overheads.

To sum up, Go-iRODSClient and iRODS FUSE Lite showed acceptable data access performance compared to existing iRODS libraries and file system clients. We plan to continuously introduce new features and improve the data transfer performance.

CONCLUSION

We have described the design and implementation of three projects. Go-iRODSClient provides iRODS access in the Go programming language. iRODS FUSE Lite allows users to access iRODS via the file system interface. The iRODS CSI driver is implemented using the two projects to provide iRODS access in Kubernetes. The iRODS CSI driver enables Kubernetes users to create Persistent Volumes for iRODS and provide iRODS access in Kubernetes Pods. The projects provide data access performance comparable to existing iRODS libraries and file system clients. However, the value of our work lies in that they enable more convenient iRODS access in the Kubernetes environment. We are currently integrating our work into several CyVerse projects and evaluating its usefulness. Future developments will focus on performance optimizations in Go-iRODSClient and iRODS FUSE Lite. We plan to implement parallel data transfer APIs to improve data access performance. We also plan to perform real world

large-scale benchmarks in a production environment to measure the performance improvements of our work. We expect our work to facilitate Kubernetes adoption in iRODS communities.

ACKNOWLEDGMENTS

We thank Peter Verraedt at KU Leuven for the contribution to Go-iRODSClient implementing admin APIs, metadata modification APIs, PAM authentication, and various bug fixes. We also thank Sarah Roberts in CyVerse for helping the iRODS CSI driver integration with CyVerse DE. We also thank Tony Edgin and Jeremy Frady in CyVerse for the test system setup.

This material is based upon work supported by the National Science Foundation under Award Numbers DBI-0735191, DBI-1265383, and DBI-1743442.

CODE AVAILABILITY

Source codes of the projects are publicly available via Github under BSD license.

Go-iRODSClient: **https://github.com/cyverse/go-irodsclient**
iRODS FUSE Lite: **https://github.com/cyverse/irodsfs**
iRODS CSI Driver: **https://github.com/cyverse/irods-csi-driver**

REFERENCES

[1] Cloud Native Definition. **https://github.com/cncf/toc/blob/main/DEFINITION.md**
[2] Kubernetes. **https://kubernetes.io**
[3] Kubernetes CSI Developer Documentation. **https://kubernetes-csi.github.io/docs/**
[4] Davfs2. **http://savannah.nongnu.org/projects/davfs2**
[5] iRODS FUSE. **https://github.com/irods/irods_client_fuse**
[6] GoRODS. **https://github.com/jjacquay712/GoRODS**
[7] Python-iRODSClient. **https://github.com/irods/python-irodsclient**
[8] CyVerse DataWatch. **https://gitlab.com/cyverse/datawatch**
[9] FUSE. **https://www.kernel.org/doc/html/latest/filesystems/fuse.html**
[10] bazil.org/fuse. **https://github.com/bazil/fuse**
[11] NFSRODS. **https://github.com/irods/irods_client_nfsrods**
[12] Persistent Volumes. **https://kubernetes.io/docs/concepts/storage/persistent-volumes**
[13] Davrods. **https://github.com/UtrechtUniversity/davrods**
[14] WebDAV Protocol (rfc4918). **https://datatracker.ietf.org/doc/html/rfc4918**
[15] Varnish HTTP Cache. **http://varnish-cache.org**
[16] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., Lyon, B.: Design and implementation of the Sun network filesystem. In Proceedings of the Summer USENIX conference (pp. 119-130). (1985)
[17] CyVerse Discovery Environment. **https://de.cyverse.org/de**
[18] Merchant, N., Lyons, E., Goff, S., Vaughn, M., Ware, D., Micklos, D., Antin, P.: The iPlant collaborative: cyberinfrastructure for enabling data to discovery for the life sciences. PLoS biology, 14(1), e1002342. (2016)
[19] Devisetty, U., Swetnam, T., McEwen, I., Wregglesworth, J., Merchant, N.: Get a Grip on Your Data Science Tools with CyVerse VICE (Visual Interactive Computing Environment). In Plant and Animal Genome XXVII Conference (January 12-16, 2019). PAG. (2019)
[20] Data Store. **https://cyverse.org/data-store**
[21] Choi, I., Nelson, J., Peterson, L., Hartman, J.: SDM: A scientific dataset delivery platform. In 2019 15th International Conference on eScience (eScience) (pp. 378-387). IEEE. (2019)
[22] Secrets. **https://kubernetes.io/docs/concepts/configuration/secret/**
[23] Jargon. **https://github.com/DICE-UNC/jargon**

Deep Dive into Ceph and how to use it for iRODS

Danny Abukalam
SoftIron
danny@abukalam.com

ABSTRACT

Ceph is the leading open source software defined storage solution. As an extremely portable and flexible solution it's an ideal approach to deploying and provisioning a one-size fits all storage solution within your data center.

In this talk I will provide a brief overview of Ceph, how it compares to other SDS solutions, new features and developments in recent releases, and different approaches for deployment.

I will also cover how it integrates with iRODS.

Archiving off-line and beyond, using the BagIt format and the bdbag library

Claudio Cacciari
SURF
Netherlands
claudio.cacciari@surf.nl

Arthur Newton
SURF
Netherlands
arthur.newton@surf.nl

ABSTRACT

SURF, the cooperative association of Dutch educational and research institutions, offers a wide range of services for different phases in the life cycle of research data, including the long term preservation resource, based on a tape library. That storage resource is offered as a standalone service, but also through iRODS and it is optimized for files with a size of the order of magnitude of one gigabyte. A common need of the researchers is to use the long term preservation resource within iRODS as part of a seamless multi-tier data flow, however moving many small files to that resource would deteriorate the performance of the tape library, therefore they have to package them together in bigger files, using tar, zip or other formats. This can be done using the icommand ibun, manually or automated via iRODS rules.

That solution is still lacking in many aspects. For example, the operation is synchronous, so in case of many users or many files, the system could be overloaded, potentially exposing the service to denial of service attacks. The metadata are not part of the package, both the system level ones, like checksums, and the user defined, therefore, if restored at a later point in time, the collection would miss important information. Naturally all these features can be implemented with iRODS rules, but it would still be a custom implementation, the more it grows in complexity the harder it is to maintain. For these reasons we decided to explore an alternative path which is the main topic of this presentation: offloading the package management (creation, checksum computation, compression) to an external library compliant with the standard BagIt format, using the rules to implement only the data staging and the asynchronous mechanism.

We argue that our approach can be extended beyond the archiving flow. It can be viewed as a way to export and import data and metadata of collections in a consistent way between iRODS and other services, among which the tape library is just one, but others can be included such as publishing tools, data repositories, data clouds, etc. Moreover, the library we use, bdbag, would support further extensions. For example, it allows to create packages containing only the metadata and pointers to the real data, which could be considered virtual packages that can be exported without moving the real data, but rather delegating the real data transfer to the user client. Or, also, it supports the definition and validation of profiles on top of the basic BagIt structure, which formalize complex package structures making them shareable and discoverable.

iRODS UGM 2021, June 8-11, 2021, Virtual

A transnational data system for HPC/Cloud-Computing Workflows based on iRODS/EUDAT

Martin Golasowski
IT4Innovations, VŠB – Technical University of Ostrava
Ostrava, Czechia
martin.golasowski@vsb.cz

Mohamad Hayek
Leibniz Supercomputing Centre (LRZ)
Garching b. Munich, Germany
hayek@lrz.de

Rubén J. García-Hernández
Leibniz Supercomputing Centre (LRZ)
Garching b. Munich, Germany
garcia@lrz.de

ABSTRACT

In this contribution, we present a transnational iRODS federation as a backend for distributed computational/Big Data workflows from science and industry. This system, the "LEXIS Distributed Data Infrastructure (DDI)", has been built in the project "Large-Scale EXecution for Industry and Society" (LEXIS, H 2020 GA 825532). It makes use of EUDAT B2SAFE, B2HANDLE, and B2STAGE on top of iRODS, to support a variety of use cases, starting with the three LEXIS Pilots: Simulations in Aeronautics Engineering, Earthquake/Tsunami Analysis, and Weather and Climate Prediction. Our presentation covers different aspects of setting up this system, from system to high-level concepts. We layout our experience in setting up a version of HAIRS (High-Availability iRODS System, cf. contributions of Kawai et al. to this meeting series), where we adapted the concept, and of installing EUDAT modules on top of it to provide the functionalities required within LEXIS. Afterward, we lay out our approach to integrating the iRODS system with the LEXIS platform, based on providing REST APIs to control and address the data system. These REST APIs are mostly custom LEXIS developments, based on conventional programming interfaces (e.g. python client) available for iRODS. Finally, we give a short status and outlook on the application of the system, and further aspects of our project interesting for the iRODS community (e.g., authentication via OpenID Connect with adaptation to Keycloak, collection structure of the iRODS system and backend systems, as also described in LEXIS publications).

Keywords

iRODS, EUDAT, High Performance Computing, Workflows

INTRODUCTION

The LEXIS project takes on a challenge of orchestrating complex HPC, Cloud and Big Data workflows on the resources of competitive European Supercomputing centers (LRZ/DE, IT4I/CZ, ICHECH/IL). It is a European Horizon 2020 project which runs between 2019-2021. The challenge has three main parts. The first one is the orchestration of workflows running on geographically distributed, federated Cloud and HPC resources. The second challenge is to provide a unified way for storing complex data and transparently move it between the compute resources, which led us to design the iRODS-based LEXIS Distributed Data Infrastructure (DDI). The last challenge is to provide a robust and secure Authentication and Authorization Infrastructure (AAI) for the whole platform.

The orchestration system in the LEXIS Platform is based on the advanced distributed orchestration framework YORC [1] which uses an extension of the TOSCA [2] standard to describe the workflows. The orchestration system then uses the LEXIS DDI to manage the data used by the individual workflow tasks [3] [4].

iRODS UGM 2021, June 8-11, 2021, Virtual

This paper focuses on integration of iRODS [5] in the LEXIS platform, where it is used as core of the DDI. The platform is implemented by a set of services which communicate by REST-based APIs with zero-trust architecture based on JWT [6] tokens and the OpenID Connect [7] authentication protocol. In this work, we also briefly describe the REST APIs we built on top of the iRODS to provide data management and staging capabilities for the platform.

LEXIS DISTRIBUTED DATA INFRASTRUCTURE

The LEXIS DDI must be able to handle large data sets (TBs) following the FAIR[1] principles, must move the data seamlessly between various locations and expose REST based APIs for the management. A schema showing the distributed infrastructure context and with various compute and storage resources connected to the LEXIS DDI is given in Figure 1

The iRODS solution has been selected as a base of the LEXIS DDI since it provides sufficient abstraction of different storage resources available at each center. Individual zones run by the centers are federated which allows seamless transfer of the data between locations while maintaining strict access policy. Data annotation and publication capabilities of the LEXIS DDI are implemented using the iRODS metadata available for each dataset and collection. The EUDAT [8] modules B2HANDLE [9] and B2SAFE [10] are used to obtain unique identifiers (PIDs) for the datasets stored in the iRODS. These add-ons leverage the internal iRODS rule engine and Python scripts.

The LEXIS DDI is formed by federated iRODS zones, which host data in a strictly defined structure, a set of REST APIs and various support services for token exchange, monitoring, auditing and publishing data.

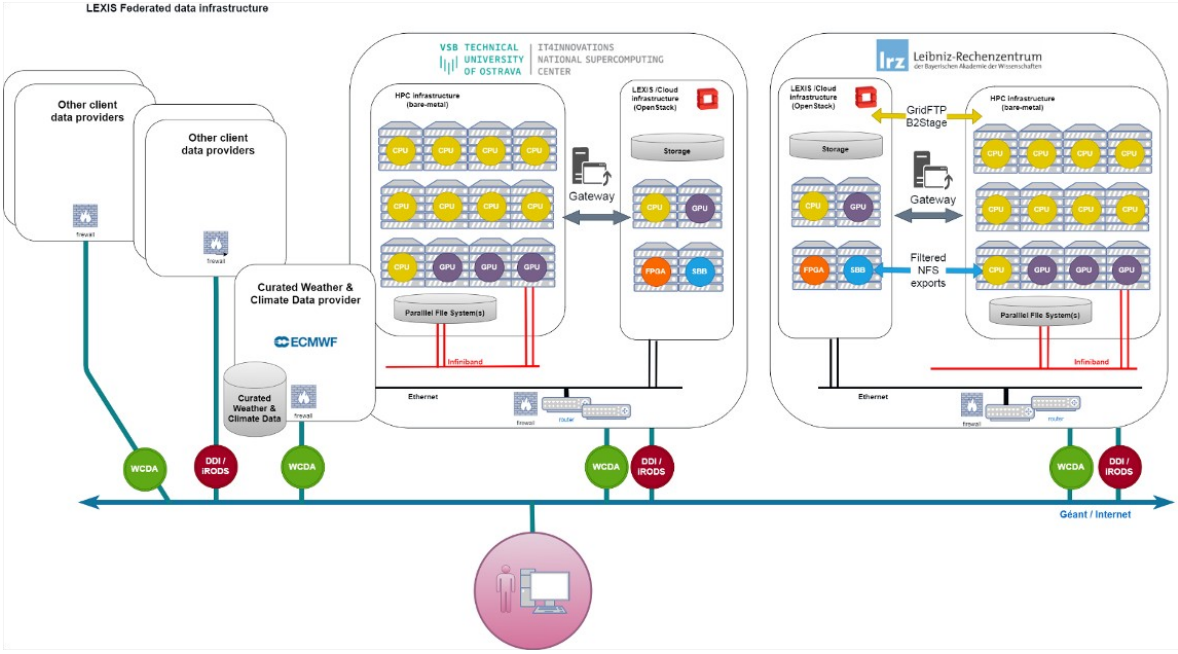


Figure 1 LEXIS Distributed Data Infrastructure connecting several compute and data providers

[1] FAIR data are data which meet principles of findability, accessibility, interoperability, and reusability cf. e.g. www.force11.org/group/fairgroup/fairprinciples

IRODS OPENID INTEGRATION

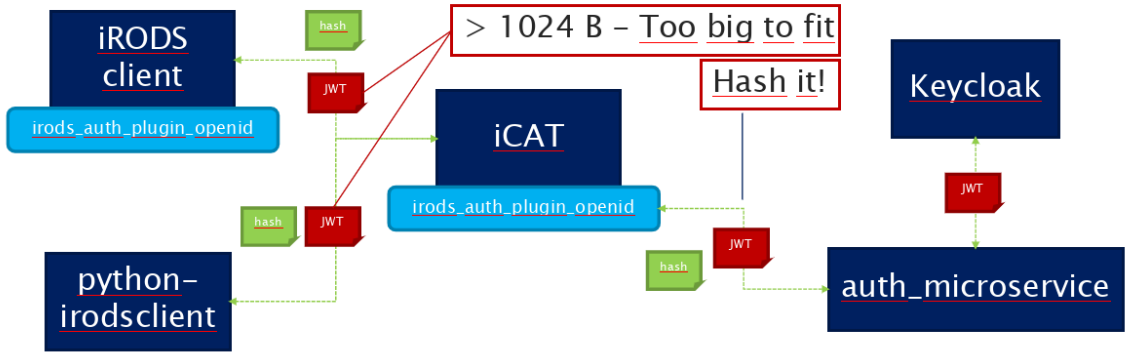


Figure 2 iRODS - OpenID authentication flow in LEXIS. The three JWT transfers (marked red) are substituted by transfers of the token hash (green)

The core iRODS software provides basic authentication mechanisms such as username / password combinations, and more advanced authentication is handled via plugins. An OpenID authentication plugin was developed by Kyle Ferriter in 2018 [11], and the iRODS python client was forked to support OpenID access, although the changes were not ported upstream, and it did not see widespread use. The current iRODS plan, to the authors' knowledge, is the integration of OpenID functionality directly in iRODS in an upcoming version.

We used these plugins as a starting point to integrate the DDI with LEXIS AAI, but some changes were needed. The most significant one relates to the fact that Keycloak [12] (the Identity Managment solution which LEXIS uses) creates quite large JWT [6] tokens, larger than the space allocated for them in the iRODS plugin, which uses the 1024 bytes long username field to transfer tokens. Our solution checks if the token is too large, and hashes it before sending it to iRODS (Figure 2). On the iRODS plugin side, the changes include an additional check in the database to see if a token exists with the proper hash. This necessitates that the webportals using iRODS as a back-end pre-authorize the token by calling the iRODS broker (the plugin component taking care of interfacing with the Keycloak server.

Further changes included bugfixes and merging of upstream changes to ensure compatibility with the latest iRODS versions. We have published the changes, although once a new iRODS version including native OpenID Connect support is published, they will be rendered obsolete.

SETTING UP IRODS IN HIGH AVAILABILITY

Since LEXIS interconnects many different components, any failure in one of the components can have huge consequences on the entire system. To avoid a single point of failure scenario, we aimed to setup iRODS in high availability mode in all centers involved in LEXIS.

A version of HAIRS [13] (High-Availability iRODS System, cf. contributions of Kawai et al. to this meeting series) was deployed. The setup consists of two iCAT servers based on the iCAT database. In front of the two servers is an instance of HA Proxy. All three instances refer to themselves with the FQDN of the iRODS server. A request to iRODS will be routed through the HA Proxy to one of the iCAT servers.

The weak remaining then is the iCAT database. If the database is down, the two iCAT servers cannot respond appropriately to requests. To secure the database setup, we opted for a deployment based on repmgr [14] and pgpool [15]. The setup consists of two PostgreSQL instances. On both instances repmgr is deployed to manage the replication between the two instances. At any point in time, only one instance is the primary instance and read and

write access is allowed. The second instance is in secondary mode. All updates to the primary database are propagated to the secondary via repmgr. In front of the PostgreSQL instances is an instance of pgpool. The main role of pgpool is to manage and trigger the failover once the primary database is down. In that scenario, pgpool declares the secondary database as primary and allows read and write access to it. The LEXIS high availability setup is depicted in Figure 3.

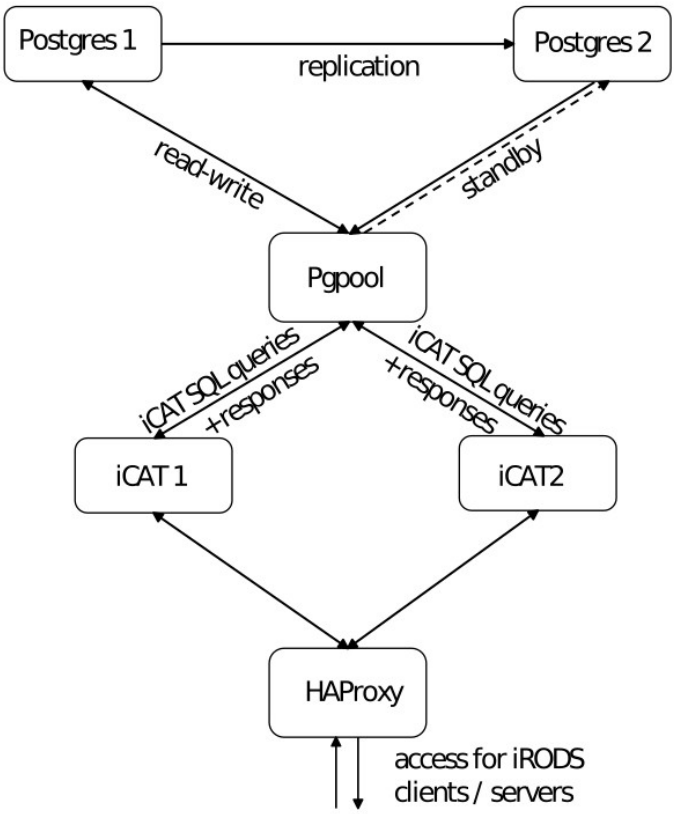


Figure 3 LEXIS redundant iRODS setup per zone

INTEGRATION WITH EUDAT

The EUDAT [8] Collaborative Data Infrastructure (CDI) offers different data management solutions for European research. Since LEXIS and EUDAT aim at achieving the FAIR data principles, we decided to deploy some of their components such as B2HANDLE [9], B2SAFE [10] which relies on iRODS, and B2STAGE [16] in LEXIS. B2HANDLE guarantees long term references to data by assigning persistent identifiers (PID). B2HANDLE is based on the HANDLE [17] system.

Another important EUDAT component is B2SAFE. B2SAFE is employed in LEXIS to manage the replication and long-term preservation of data between the different iRODS zones. This helps in optimizing the access for users on the different computing systems available in LEXIS.

To move data from a non iRODS environment to iRODS, B2STAGE was deployed. B2STAGE employs a GridFTP server on top of iRODS and allows user to move data in or out of iRODS via basic GridFTP commands,

CUSTOM APIS AND THE USE OF THE IRODS PYTHON CLIENT

The LEXIS DDI provides custom APIs to communicate with the webportal of the LEXIS platform and the LEXIS orchestrator. This helps in automating the execution of workflows. The APIs are divided into three categories: LEXIS iRODS API, Staging API, and encryption/compression API.

The LEXIS iRODS API manages the lifecycle of users and projects. The API provides endpoints to create and delete users across the federated iRODS zones, create projects and collections across the federated iRODS zones, set user's ACLs based on roles in LEXIS, and to obtain tokens that are used to connect to iRODS.

The Staging API provides the possibility to move data between the DDI and our cloud computing and HPC systems. The API is based on Django [18] and uses LEXIS Authentication and Authorization Infrastructure (AAI) to authenticate users' requests. Since data transfer is a time-consuming task, a queuing system using Celery [19] and RabbitMQ [20] is deployed. The user receives a request ID when initiating a request. The ID can be used to track the status of the data transfer, which proceeds asynchronously.

To improve data transfer rate and to secure sensitive data in iRODS, the encryption/compression API is used. The API uses a similar queuing system and performs compression on datasets with large numbers of small files for transfers. The encryption works asynchronously: Data are encrypted when moving to iRODS and then decrypted when staged to the target. The API uses aes-256-ctr encryption mode and runs on a dedicated LEXIS Burst Buffer/Data Node with 64 CPU cores, and large NVMe disk.

CONCLUSION

The distributed data infrastructure showcased in this paper provides a fault-tolerant back-end for the data tasks in the LEXIS European Cloud-HPC Workflow Platform (H2020), including efficient data transfer across sites across Europe. We leverage iRODS capabilities to federate geographically distributed data sources, and EUDAT services (especially B2SAFE) to ensure DATA FAIR principles.

We are currently at the stage of benchmarking and optimizing the performance of the system. We tested different iRODS setups, and finally selected the HAIRS deployment with redundant PostgreSQL setup due to its high-availability properties. We have also made extensive use of the iRODS Python client as part of the interfaces to other LEXIS components. For further information, the reader is invited to have a look at our book publication [21].

Setting up iRODS-OpenID Connect authentication in order to provide a single-login interface across all the LEXIS components was a challenging endeavor, due to some iRODS assumptions (maximum token size) which were not satisfied by our OpenID solution (Keycloak), and which required modifications to the iRODS authentication plugin. We therefore look forward to the finalization of the iRODS work on native implementation of OpenID Connect authentication.

ACKNOWLEDGMENTS

This work and all contributing authors are funded/co-funded by the EU's Horizon 2020 research and innovation programme (2014-2020) under grant agreement no. 825532 (Project LEXIS – "Large-scale EXecution for Industry and Society"). The authors would like to sincerely thank the colleagues from the iRODS consortium and EUDAT for their help and collaboration.

REFERENCES

[1] Atos BDS R&D, "Yorc 4.0.2," 2020. [Online]. Available: http://yorc.readthedocs.io/. [Accessed 29 Jul. 2020].

[2] OASIS TC, "Topology and Orchestration Specification for Cloud Applications Version 1.0," OASIS Standard, 25 Nov. 2013. [Online]. Available:

http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html. [Accessed 27 Apr. 2020].

[3] V. Svatoň, J. Martinovič, J. Křenek, T. Esch and P. Tomančák, "HPC-as-a-Service via HEAppE Platform," in *Proceedings of the 13th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS-2019). Advances in Intellegent Systems and Computing, 993*, Sydney, Australia, 2019.

[4] LEXIS Project, "Deliverable 4.2: Design and Implementation of the HPC-Federated Orchestration System - Intermediate," [Online]. Available: https://lexis-project.eu/web/wp-content/uploads/2020/08/LEXIS_Deliverable_D4.2.pdf. [Accessed 2 Apr. 2021].

[5] H. Xu, T. Russell, J. Coposky, A. Rajasekar, R. Moore, A. de Torey, M. Wan, W. Shroeder und S.-Y. Chen, iRODS Primer 2: Integrated Rule-Oriented Data System, Williston, VT: Morgan & Claypool Publishers, 2017.

[6] M. Jones, B. Campbell and C. Mortimore, "RFC 7523 - JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants," Internet Engineering Task Force (IETF), May 2015. [Online]. Available: https://tools.ietf.org/html/rfc7523. [Accessed 1 Apr. 2020].

[7] N. Sakimura, J. Bradley, M. B. Jones, B. de Medeiros and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1," The OpenID Foundation, 8 Nov 2014. [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html. [Accessed 27 04 2020].

[8] EUDAT Collaborative Data Infrastructure, "EUDAT," EUDAT Ltd, 2020. [Online]. Available: https://www.eudat.eu. [Accessed 27 Apr. 2020].

[9] EUDAT Collaborative Data Infrastructure, "B2HANDLE - EUDAT," EUDAT Ltd, 2020. [Online]. Available: https://www.eudat.eu/services/b2handle. [Accessed 13 Apr. 2020].

[10] EUDAT Collaborative Data Infrastructure, "B2SAFE - EUDAT," EUDAT Ltd, 2020. [Online]. Available: https://www.eudat.eu/services/b2safe. [Accessed 13 Apr. 2020].

[11] K. Ferriter, „ OpenID Connect Authentification in iRODS,“ in *iRODS User Group Meeting*, Durham, 2018.

[12] JBoss (Red Hat Inc.), Keycloak Community, "Keycloak," [Online]. Available: https://www.keycloak.org/. [Accessed 10 Apr. 2020].

[13] Y. Kawai und A. Hasan, „High-Availability iRODS System (HAIRS),“ in *Proceedings of the iRODS User Group Meeting 2010: Policy-Based Data Management, Sharing and Preservation*, Chapel Hill, NC, 2010.

[14] 2ndQuadrant Ltd., "repmgr - Replication Manager for PostgreSQL clusters," 2020. [Online]. Available: https://repmgr.org/. [Accessed 13 Apr. 2020].

[15] SRA OSS, Inc. et al., "pgpool Wiki," 2020. [Online]. Available: https://www.pgpool.net/mediawiki/index.php/Main_Page. [Accessed 13 04 2020].

[16] EUDAT Collaborative Data Infrastructure, "B2STAGE - EUDAT," EUDAT Ltd, 2020. [Online]. Available: https://www.eudat.eu/services/b2stage. [Accessed 13 Apr. 2020].

[17] B. Boesch, S. X. Sun and L. Lannom, "RFC 3650 - Handle System Overview," Internet Engineering Task Force (IETF), Nov. 2003. [Online]. Available: https://tools.ietf.org/html/rfc3650. [Accessed 27 Apr. 2020].

[18] Django Software Foundation, "Django," 2020. [Online]. Available: https://www.djangoproject.com. [Accessed 2020 Apr. 1].

[19] Celery, "Celery: Distributed Task Queue," 2020. [Online]. Available: http://www.celeryproject.org/. [Accessed 1 Apr. 2020].

[20] Pivotal Software, "Messaging that just works — RabbitMQ," 2020. [Online]. Available: https://www.rabbitmq.com/. [Accessed 1 Apr. 2020].

[21] J. M. O. Terzo, HPC, Big Data, AI Convergence Towards Exascale. Challenge and Vision, Taylor & Francis Ltd (Verlag), 2021.

Hierarchical indexes of large file systems and iRODS

Peter Braam
ThinkParQ
peter.braam@gmail.com

ABSTRACT

Understanding the content of high-performance file systems encounters at least two major challenges. First, the number of items in such file systems now regularly exceeds a billion items and can be much larger. Scanning such file systems is very time-consuming. Secondly, the rate at which content can be added to such parallel file systems can easily exceed the rate at which change events can be digested.

During the rise of campaign storage and independently also seen in the APFS file system, a few hierarchical content indices were promoted. Such systems can provide a logarithmic search for items with particular properties and may be a useful intermediary between iRODS and extreme file systems.

In this lecture, I will give an overview of this approach, what has been implemented, and some of the open questions in this area.

Herons, Yaks and Technical Debt - 2020 at Sanger

John Constable
Wellcome Sanger Institute
United Kingdom
jc18@sanger.ac.uk

ABSTRACT

In 2020 the Informatics Systems Group at Sanger had a challenge - get the iRODS estate up to a supported, patched, latest reasonable version, both application and Operating System.

It also had another challenge - move 8PB off of ageing hardware to new equipment, preferably with no user disruption. Of course, we had to add a few more PB of capacity, too.

While that was going on, we also planned to deploy Proof Of Concepts on the Indexing plugin, NFSRODS and the MetalNX GUI. Replacing the HA system seemed a good idea too. Oh yes, and we should probably catch up on our backlog.

Did we mention this all had to happen while supporting the science of sequencing this COVID thing (you may not have heard of it)?

Come along to hear all the war stories, and be prepared to count Yaks and Unicorns.

iRODS Policy Composition

Jason Coposky
Renaissance Computing Institute (RENCI)
UNC-Chapel Hill
jasonc@renci.org

ABSTRACT

This talk will demonstrate the updates made to policy composition from last year's introduction. The code is now real and does what was promised.

iRODS UGM 2021, June 8-11, 2021, Virtual

iRODS Client Library: Python iRODS Client 1.0

Daniel Moore
Renaissance Computing Institute (RENCI)
UNC-Chapel Hill
dmoore@renci.org

ABSTRACT

This talk will cover the last year's updates to the Python iRODS Client. These include additional coverage of the iRODS API, better authentication options and connection handling, atomic metadata manipulation, and parallel transfer.

A Year of iRODS: Lessons Learned

Ingrid Barcena Roig
KU Leuven
Leuven, Belgium
ingrid.barcenaroig@kuleuven.be

ABSTRACT

A year ago the VSC Tier-1 Data service was launched. The primary goal of this service is to offer the Flemish researchers a platform to manage research data that is being processed using the VSC High Performance Computing infrastructure. iRODS was selected as basis for this service to facilitate the way the researchers create, manage, share and reuse research data.

The first year the platform was in a pilot phase with a reduced number of research groups from different domains (Climate Change studies, Humanities and Arts, Life science, Technology, …).

This talk will present the lessons we have learned in this first year designing, implementing and managing an iRODS based platform. The current status of the project, including some of the use cases that are already using the platform and the future plans will also be presented.

iRODS UGM 2021, June 8-11, 2021, Virtual

iRODS Policy: Read-only local analysis staging policy for BRAIN-I

Terrell Russell
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
unc@terrellrussell.com

Michelle Itano
Neuroscience Microscopy Core (NMC)
UNC Chapel Hill
itano@unc.edu

Jason Stein
Stein Lab, Department of Genetics
UNC Chapel Hill
jason_stein@med.unc.edu

Oleh Krupa
Stein Lab, Department of Genetics
UNC Chapel Hill
ok37@email.unc.edu

ABSTRACT

The BRAIN-I project, a collaboration to image and study mouse brains between the Renaissance Computing Institute (RENCI) and the UNC Neuroscience Microscopy Core (NMC) at the UNC Neuroscience Center at the UNC-Chapel Hill School of Medicine, has been working on policy to allow researchers to do local analysis of data that is already under management by iRODS. This paper will explain the design decisions, the policy that was developed and deployed, and how it works alongside the rest of the BRAIN-I configuration.

Keywords

iRODS, data management, policy, brain images, neuroscience

INTRODUCTION

This project is a collaboration between multiple groups on the same campus. This includes team members from the Renaissance Computing Institute (RENCI), Steve Cox, Director of Software Architecture, and Terrell Russell, Chief Technologist for the iRODS Consortium. The UNC Neuroscience Microscopy Core (NMC) in the School of Medicine is represented by Michelle Itano, the Core's Director. The Stein Lab in UNC-Chapel Hill Department of Genetics is represented by Jason Stein, Lab Director, and Oleh Krupa, Ph.D.

The Stein Lab studies how variations in the genome affect the structure and development of the brain, and in doing so, create risk for neuropsychiatric illnesses. One of the lab's research projects involves using high-powered optical microscopes to create extremely detailed images of mouse brains.

This paper represents the work done to formalize and codify the process of image data ingest and staging with an eye for local analysis by lab technicians and managers.

ARCHITECTURE

This policy project began with iRODS already in place and NFSRODS[1] providing access to image analysis toolsets.

iRODS UGM 2021 June 8-11, 2021, Virtual

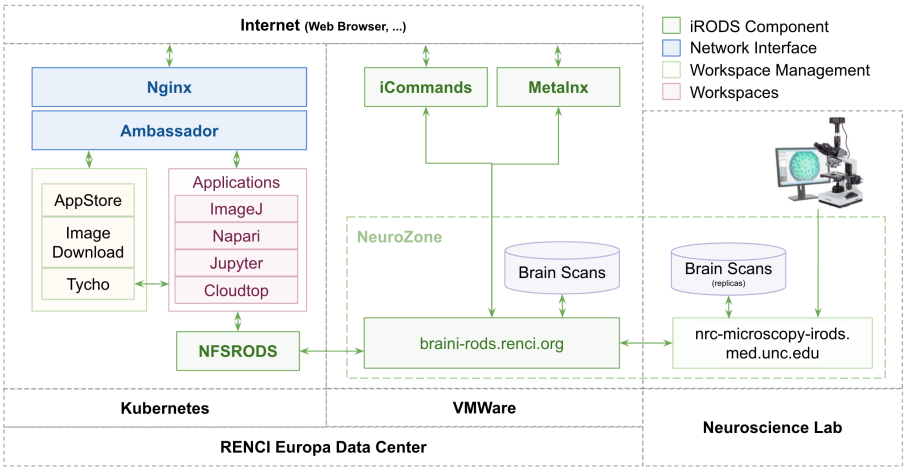


Figure 1. BRAIN-I Architecture

Figure 1 shows the connections between the RENCI Europa Data Center and the Neuroscience Lab in the School of Medicine. The iRODS Catalog Service Provider (`braini-rods.renci.org`) runs in a long-running virtual machine provisioned by VMWare connected to long-term storage provisioned and backed up by RENCI. Various iRODS clients connect to this provider, including the iCommands[2], Metalnx[3], and NFSRODS. NFSRODS runs in the Kubernetes[4] cluster and provides NFSv4.1 access to Data Objects within iRODS to various other applications made available by Kubernetes. An iRODS Catalog Service Consumer is running in the Neuroscience Lab and provides local storage closer to the microscope(s), used for ingest. The policy described in this paper is concerned with the relatively hands-free management of the ingested data and then the ongoing analysis of that data.

USE CASES

There are four main use cases defined and agreed to by the stakeholders of this project. These were developed over 2-3 months in late 2020 through brainstorming sessions, collaborative editing, and reflective meetings and cover the desired and expected workflows of lab members at the time.

The stakeholders involved included scientists doing the capture, lab technicians, the lab director, the microscopy core director, the RENCI infrastructure team, the iRODS development team, and the iRODS administrator of this Zone.

The four main use cases and their envisioned solutions are provided and discussed in the next four subsections, numbered NMC1 through NMC4.

NMC1 - Import of data to iRODS

A lab member saves image data into a well-known location on a local disk attached to the iRODS server (Maybe a folder named 'iRODS import'). (Maybe this can be automated later, but for now... just 'saved/moved' is good enough). The lab member expects the data to appear in iRODS, perhaps with some particular permissions, perhaps with some particular metadata (provided or extracted or associated). The lab expects the data to be managed/moved off the local disk, so it doesn't fill up. Ideally, once in iRODS, there would be a backup (replica) made to a second more secure location. Once copied into another iRODS location with more storage, the import files are removed from the local disk.

SOLUTION - Storage Tiering[5] (and possibly Automated Ingest[6])

NMC2 - Import of data to iRODS and local analysis

A lab member saves image data into a well-known location on a local disk attached to the iRODS server (Maybe a folder named 'iRODS import'). (Maybe this can be automated later, but for now... just 'saved/moved' is good enough). The lab member expects the data to appear in iRODS, perhaps with some particular permissions, perhaps with some particular metadata (provided or extracted or associated). The lab expects the data to be managed/moved off the local disk, so it doesn't fill up. Ideally, once in iRODS, there would be a backup (repica) made to a second more secure location. Once copied into another iRODS location with more storage, the import files are moved to an 'analysis' folder on the local iRODS server.

SOLUTION - Storage Tiering (and labels for 'local analysis') local analysis could be read-only in the vault, perhaps with permissions changed(?!), or another local copy if read/write was important. And any products of local analysis would get routed back through the front door of iRODS. Also possible... NFSRODS, but concerns about network latency are real.

NMC3 - Local iRODS analysis

Analysis of files on the local (NMC) iRODS server from the 'analysis' folder using the local hardware accessible. Could be a custom python script, Matlab code, or through iRODS on ImageJ or Napari. Ideally, would utilize as much computational power as is available, without making it impossible for another user to run a job. Not sure if this can be set to somehow utilize up to 100% or 90% power, but then if another job is started to drop down to maybe 75% or something like that? Or maybe only do that if the job is set to last more than 6 hours, or some relatively arbitrary 'long time'. Leaving 25% for other jobs to still run during that time period? Otherwise for 'shorter' times jobs would be run sequentially with full computational power? Would ideally save analyzed files into the 'iRODS import' folder where again it would be copied and moved to a larger external iRODS server, and removed from the local drive to free up space. Files remaining in the 'analysis' folder would still be accessible to run code on locally.

SOLUTION - Handled by the solution to NMC2

NMC4 - Grant external user iRODS access to published data

Published data would be put on an iRODS protected folder and available post publication for download. Ideally, this would be a very quick process, not requiring a person to verify the request validity. But would also be good to mark how many times files had been downloaded, track who downloaded (maybe just by email address and verifying that address was real?), and if possible allow users to only download specific file sets and not the entire file set. Would want to also ensure that downloading this data wouldn't take up all the bandwidth for the iRODS server.

SOLUTION - Storage Tiering and labeling of particular Collections or Data Objects - policy can fire and run the 'publish()' function, whatever that is defined to be.

DESIGN GOALS

After agreeing on the interesting use cases, the commonalities were identified and turned into design goals that would satisfy the use cases. These were presented as three 'reasons' to move data around the iRODS Zone.

First, the initial ingest of microscope data would be handled by Automatic Storage Tiering, providing a hands-free migration from the local storage in the lab to the long-term storage in the RENCI data center.

The second design goal was manual targeting of interesting data to NMC for local analysis. This could be performed by anyone with appropriate permission in the system and would have a relatively quick effect of staging the data for the scientists to run their local software tools.

The third design goal was manual targeting, using the same mechanism, to a future location as 'published' or to prepare for being published by an as-yet-determined pipeline. This last design goal was deemed to be less important

than the first two, and would later become 'future work'.

Proposed Solution

To satisfy the design goals, the following configuration and algorithm was proposed.

There would be three different iRODS storage resources, `nmc-ingest` and `nmc-analysis` located on the lab workstation, and `renciResc`, theh primary long-term storage at RENCI.

For the first use case (NMC1), the iRODS automated Ingest tool would be configured to register new files 'in-place' into `nmc-ingest`. The iRODS Storage Tiering framework would be configured to automatically migrate data (replicate and trim) from `nmc-ingest` to `renciResc` after a file had been at rest for 12 hours.

For the local analysis use cases (NMC2 and NMC3), there would be two dynamic policy enforcement points (PEPs) that would fire whenever a Collection or Data Object was tagged with the iRODS AVU of `nmc` and `analysis` (no Unit). After metadata was added, the policy would replicate the Collection or Data Object to `nmc-analysis` and set physical permissions to read-only for other unix users on the NMC workstation to be able to perform analysis. After metadata was removed, the replica(s) on the NMC workstation would be trimmed. The original replica(s) would still remain in the long-term storage at RENCI.

The publishing use case (NMC4) would also be handled by a PEP firing after metadata was added to a Collection or Data Object and replicate, then checksum, and then generate a DOI in a public-accessible area of the server.

Takeaways

The main takeaways from this proposed solution were four-fold:

1) There would always be a replica in primary storage at RENCI after the initial ingest and migration progress. This migration could be from nearly 'real-time' to '1 hour' to '1 day' or '1 week' and could be changed at a later date. Additionally, the 'minimum restage tier' could be set to `renciResc`, so no data ever restages back to the NMC workstation upon retrieval or use.

2) Manual tagging and replication gives control to the lab participants to manage their limited disk space on the lab workstation.

3) Manual tagging will be easily extended to provide for a future publication workflow.

4) Existing and future Cloud Apps will pull from the always-available and in-the-same-datacenter `renciResc`, without triggering any replication or data movement other than to the tool itself.

IMPLEMENTED POLICY

As part of the BRAIN-I project, this iRODS policy set defines the policies for data analysis, replication, and retention in the NMC.

`https://github.com/irods/irods_policy_examples/tree/main/nmc_analysis`[7]

There are two parts of the policy managing the data flow within the iRODS Zone:

Automatic

The iRODS Storage Tiering Framework (Figure 2) is handling newly ingested data and moving it into the long-term storage housed at RENCI. RENCI is providing storage and visualization tooling that prioritizes that local, long-term storage. No new policy was written for BRAIN-I for this deployment. The Tiering Framework handled the use case requirements out-of-the-box and only required configuration.

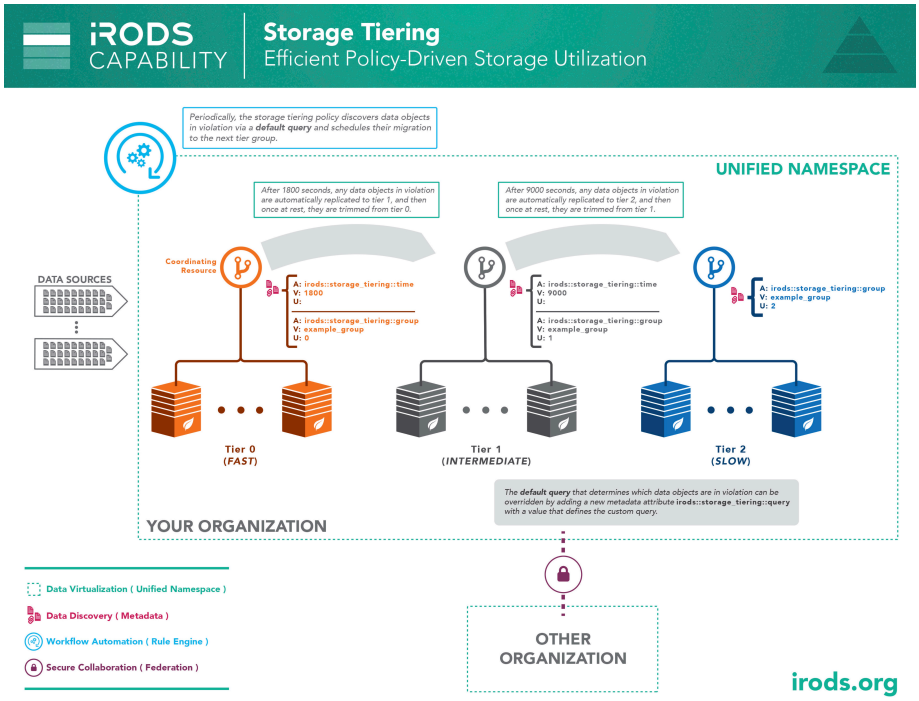


Figure 2. iRODS Storage Tiering Framework

Manual

When NMC staff want to run local analysis on data already in the iRODS namespace, they can 'tag' the data of interest, and this policy will manage the replication to their local machine, set permissions, and prevent removal of that data from the system until it has been 'untagged'. Once 'untagged', the data will be trimmed from the researchers' local storage and remain housed only in long-term storage at RENCI.

This custom policy was designed, written, deployed, and tested for BRAIN-I specifically. However, this mark-and-sweep approach is generic enough it could be extended to meet similar workflow requirements in other domains.

The sweeper does three things every time it fires (configured to be every 30 seconds). It checks which collections and data objects need to be enqueued for replication, and it checks which data objects need to be trimmed.

```
callback.nmc_replicate_dataobjs_under_tagged_collections();
callback.nmc_replicate_tagged_dataobjs();
callback.nmc_trim_untagged_dataobjs_on_target_resource();
```

The other PEPs only run pre-checks to see whether the items being operated on are 'tagged' and therefore protected from being manipulated because they are 'in analysis mode'.

TESTING

To prove that the policy was behaving as expected, a series of BATS[8] tests were written to assert that the system was in a good state before and after each required operation. These scenarios include the simple tagging and untagging

of items in the iRODS Zone, but also their recursive counterparts. When an item is tagged for analysis, neither it nor its descendents can be removed manually, and this must be enforced by policy.

```
$ git clone https://github.com/bats-core/bats-core
$ time bash bats-core/bin/bats test_nmc_analysis.bats
 ✓ tag a collection
 ✓ tag a data object
 ✓ untag a collection
 ✓ untag a data object
 ✓ overwrite a tagged data object
 ✓ overwrite a data object under a tagged collection
 ✓ trim a tagged data object - DISALLOWED
 ✓ trim a data object under a tagged collection - DISALLOWED
 ✓ remove a tagged data object - DISALLOWED
 ✓ remove a tagged collection - DISALLOWED
 ✓ remove a data object under a tagged collection - DISALLOWED
 ✓ remove a collection under a tagged collection - DISALLOWED
 ✓ remove a collection containing a tagged data object - DISALLOWED
 ✓ remove a collection containing a tagged collection - DISALLOWED
 ✓ untag an enqueued data object - DISALLOWED
 ✓ untag a collection with an enqueued descendent data object - DISALLOWED

16 tests, 0 failures

real    2m4.745s
user    0m8.606s
sys     0m2.172s
```

STATUS

At the time of this paper, the BRAIN-I project had no items in active analysis, but had nearly 1 million data objects under management representing nearly 13TB of storage.

```
$ iquest "%s" "select count(DATA_ID) where RESC_NAME = 'nmc-ingest'"
0

$ iquest "%s" "select count(DATA_ID) where RESC_NAME = 'nmc-analysis'"
0

$ iquest "%s" "select count(DATA_ID) where RESC_NAME = 'renciResc'"
921670

$ iquest "%s" "select sum(DATA_SIZE) where RESC_NAME = 'renciResc'" \
  | awk '{print $1/1024^3 " GB "}'
12743.7 GB
```

LIMITATIONS AND FUTURE WORK

The original proposed solution included ingest, tiering, local analysis, and publication, but these were not all implemented in the initial rollout of this policy.

The Automated Ingest could be deployed with a Landing Zone pattern to catch and process the data coming from the microscopes. This was not implemented due to the microscope not being directly attached to the network. Lab personnel currently need to manually transfer the data to the NMC workstation before running `iput`.

Publication can easily be added once more analysis is done and useful data products are ready to be shared with others.

Additional microscopes could be added to this configuration without much trouble. The additional concerns would appear as namespacing problems and should be easily mitigated with good metadata and naming practices.

As the NMC becomes accustomed to having their images available in a platform like BRAIN-I, we expect additional labs to become interested in having something similar or being added to the BRAIN-I infrastructure directly. This should also scale as needed, since additional storage capacity could be added at RENCI and/or at other labs or data centers. The unified namespace of iRODS already accounts for growth and expansion of this kind.

SUMMARY

This paper covers the design process and results of providing a read-only local analysis staging policy for the BRAIN-I project at UNC-Chapel Hill. Live code is available and can serve as a prototype of policy for similar use cases.

iRODS proves flexible and powerful enough to provide transparency and control for a relatively small lab while demonstrating that the platform is capable of driving this type of policy-based approach for any organization.

REFERENCES

[1] Draughn, Kory; Russell, Terrell. iRODS Client: NFSRODS 2.0 (2021). Draughn `https://irods.org/uploads/2021/Draughn-iRODS-NFSRODS_2.0-paper.pdf`

[2] iRODS Client: iCommands `https://github.com/irods/irods_client_icommands`

[3] Zhou, Bo; Draughn, Kory; Coposky, Jason; Russell, Terrell; Conway, Mike; iRODS Client: Metalnx 2.4.0 with GalleryView (2021) `https://irods.org/uploads/2021/Zhou-iRODS-Metalnx_2.4.0_with_GalleryView-slides.pdf`

[4] Kubernetes `https://kubernetes.io/`

[5] iRODS Unified Storage Tiering Rule Engine Plugin. `https://github.com/irods/irods_capability_storage_tiering`

[6] Xu, Hao; King, Alan; Russell, Terrell; Coposky, Jason; de Torcy, Antoine; iRODS Capability: Automated Ingest (2018) `https://irods.org/uploads/2018/Xu-RENCI-Automated_Ingest-paper.pdf`

[7] Russell, Terrell: UNC Neuroscience Microscopy Core (NMC) iRODS Policy (2021). `https://github.com/irods/irods_policy_examples/tree/main/nmc_analysis`

[8] Bats-core: Bash Automated Testing System (2018). `https://github.com/bats-core/bats-core`

Panel - Storage Chargeback: Policy and Pricing

Nirav Merchant
CyVerse / University of Arizona
nirav@email.arizona.edu

Peter Clapham
Wellcome Sanger Institute
United Kingdom
pc7@sanger.ac.uk

Jason Coposky
Renaissance Computing Institute (RENCI)
University of North Carolina at Chapel Hill
jasonc@renci.org

ABSTRACT

Storage offerings from third parties (including cloud providers) have made significant inroads in the last few years. Many clients and customers may now request to bring their own storage into an existing managed software stack or environment. This panel will discuss the opportunities, the costs, and the complexities involved in servicing these types of requests.

iRODS UGM 2021, June 8-11, 2021, Virtual

XtreemStore - Scalable Object Store Software for Archive Medium Tape

Christian Wolf
GRAU DATA
christian.wolf@graudata.com

ABSTRACT

This talk will cover the technology (object storage with S3 interface to tape) and existing use cases for XtreemStore as well as recent work to certify against iRODS 4.2.7 and the iRODS S3 storage resource plugin.

iRODS UGM 2021, June 8-11, 2021, Virtual

Refactoring Kanki - Towards a Modern Native iRODS Client Implementation

Ilari Korhonen
KTH Royal Institute of Technology
ilarik@kth.se

ABSTRACT

The latest developments in the Kanki project are presented, i.e the recent work which has been carried out in collaboration with the iRODS consortium for both the modernizing of the Kanki CMake build environment and the refactoring of the code base for the next generation of iRODS client APIs. Many of the old fully C-compatible constructs have now been replaced and/or wrapped with object-oriented C++17 interfaces while enabling RAII. The simultaneous iRODS connections are now being pooled into connection pools, from which worker threads (instantiated from thread pools) can reserve their exclusive comms channel to an iRODS agent and perform client-side RPCs asynchronously. This enables Kanki to become a fully parallel-tasking high-performance iRODS client unlocking more and more of the new parallelism found in later iRODS versions.

iRODS UGM 2021, June 8-11, 2021, Virtual

Retrospective: Migrating Yoda from the PHP iRODS client to the Python iRODS client

Lazlo Westerhof
Utrecht University
Netherlands
l.r.westerhof@uu.nl

ABSTRACT

At the UGM 2018, we presented Yoda, a system for reliable, long-term storing and archiving large amounts of FAIR research data during all stages of a study. Yoda deploys iRODS as its core component, customized with more than 10,000 lines of iRODS rules. With the iRODS Python rule engine plugin, we rewrote most of our rules to Python and developed an API for the Yoda web portal. Meanwhile the Yoda web portal was still communicating with iRODS trough the PHP client. This is the story of migrating the Yoda web portal from the PHP to the Python iRODS client.

iRODS UGM 2021, June 8-11, 2021, Virtual

Leveraging iRODS for Scientific Applications in AWS Cloud

Radha Konduri
Bristol Myers Squibb
radha.konduri@bms.com

Dmitry Khavich
Bristol Myers Squibb
Dmitry.Khavich@bms.com

ABSTRACT

Advanced digital microscopes have revolutionized biology and pharmaceutical research, generating massive volumes of files like images, including 3D and time series data. But scientists can't study what they can't measure. The need to quantify biological characteristics, such as cell count, or to calculate metrics such as drug occupancy, has driven a need to seamlessly ingest, annotate, share, analyze and archive image datasets. Our projects seek to make image datasets easily discoverable, accessible, transformable, and analyzable, and to ensure that numeric datasets derived from image datasets are easily transferable to downstream data analysis platforms.

Objectives: On-demand instrument data search and retrieval; Phenotypic feature measurement, analysis and zero manual intervention required to move images or metadata; Instrument files not locked in siloed analysis platforms; Image formats are globally readable; Image feature measures can be integrated and compared; Files origin / provenance available; Ability to annotate images and features; File data set transparency or visibility / no duplicated effort; Image data sets actionable.

BMS took the approach to implement the integrated Rule Oriented Data System (iRODS) to ingest, validate, and assign metadata to instrument file datasets, to provide provenance, and to make file datasets discoverable and actionable. Enable storage of numeric datasets as attributes of imaging datasets and make available for downstream processing.

Projects – Immuno-Oncology Cellular Therapy (IOCT), Discovery Imaging Platform (DIP)

iRODS UGM 2021, June 8-11, 2021, Virtual

iCommands Userspace Packaging

Markus Kitsinger
Renaissance Computing Institute (RENCI)
University of North Carolina at Chapel Hill
kitsinger@renci.org

ABSTRACT

Since iRODS 4.2.0, the iCommands have been harder to deploy in HPC environments due to the packaged nature of the releases. Non-package-install builds were possible, but not very portable for administrators to provide binaries for their users. This talk will cover the work done to provide extractable userspace builds of the iCommands.

iRODS UGM 2021, June 8-11, 2021, Virtual

Towards a scaled system for ingest, analysis, manipulation, and deployment of multiple HEVC streams with HPC and iRODS

David Wade
Integral Engineering
david.wade@cox.net

ABSTRACT

High End Video Codec (HEVC) streams, at 4K, 8K, and 16K pixels per frame (perhaps arising from autonomous mobile agents, at the 96 frames per second necessary for potential 3D Virtual viewing) provoke real concerns of scalability, both for performance and throughput, if they are to be ingested, analyzed, manipulated, and deployed at scale for multiple dozens, hundreds, or thousands of streams simultaneously. If analysis across multiple frames in a stream, across multiple streams, and/or conditioning (as in on-the-fly editing of frames in and out from multiple streams) are to become desirable, the existing networks, processors, memory, and data management schemes are likely to be insufficient.

From COTS High Performance Computing design techniques, software and hardware for ingest and analysis, and employing iRODS as a data system control, we propose a design scalable to the potential demands of intelligent agents which must observe and report across such a large domain of HEVC inputs.

iRODS UGM 2021, June 8-11, 2021, Virtual

iRODS Client: C++ REST API

Jason Coposky
Renaissance Computing Institute
(RENCI)
UNC Chapel Hill
jasonc@renci.org

Terrell Russell
Renaissance Computing Institute
(RENCI)
UNC Chapel Hill
unc@terrellrussell.com

ABSTRACT

The iRODS C++ REST API has been discussed for years, but is now ready to show to others. This paper will explore the different aspects of what is possible with the REST API today and invite discussion about what else it may need.

Keywords

iRODS, data management, administration, REST, API

INTRODUCTION

iRODS itself is a protocol and an API, by default running on port 1247/1248, with a server implementation written in C++. Since iRODS was designed, the world has largely standardized on HTTP REST as a means to quickly set up and maintain connections between different services on the Internet. This iRODS REST API is written in C++, designed to be run alongside an iRODS Server, and provides a wrapper around the iRODS protocol, giving new accessibility and development speed to REST-familiar developers of web services and other applications.

It runs under the same linux service account as the iRODS Server, accesses `/etc/irods/server_config.json`, and uses the active authenticated `rodsadmin` iRODS account.

This initial REST API implementation requires to be run alongside an iRODS Server v4.2.0 or greater. It provides endpoints for `/access`, `/admin`, `/auth`, `/configuration`, `/list`, `/query`, `/stream`, and `/zone_report`. Additional functionality may be added if community usage generates new use cases.

MOTIVATION

The iRODS community is diverse and demands relatively unique applications depending on the science or industry domain (microscopy, genomics, agriculture, archives, etc.) or the specific roles of its users (instrument service account, metadata curator, policy administrator, etc.). These different use cases sometimes render general search and browse applications (like Metalnx [1]) too powerful and/or too confusing. Many organizations now have the capacity to develop their own specific, customized applications built with modern web-development frameworks, but these frameworks all assume available HTTP server endpoints with which to interact.

iRODS needs to provide these type of endpoints through an easy to use, easy to deploy, fast, and lightweight REST API.

An initial iRODS Consortium application to consume these endpoints is the new Zone Management Tool [2].

iRODS UGM 2021 June 8-11, 2021, Virtual

IMPLEMENTATION

This API is based on the iRODS C++ API and acts as a proxied mid-tier application layer between web applications and an iRODS server. It utilizes standard JSON Web Tokens (JWT)[3] for authentication and authorization. The REST API provides a single executable per endpoint which is designed to be compatible with containerized deployments.

This API is designed to follow the Hypermedia as the Engine of Application State (HATEOAS)[4] model where the client does not need to keep state and the API itself will share URLs in its responses dictating the next relevant links.

CONFIGURATION

The REST API provides an executable for each individual API endpoint. These endpoints may be grouped behind a reverse proxy in order to provide a single port for access.

Two configuration template files are placed in `/etc/irods` in a packaged deployment and need to be copied and edited before activation. The `/etc/irods/irods_client_rest_cpp.json.template` is the REST API template and `/etc/irods/irods_client_rest_cpp_reverse_proxy.conf.template` is an nginx[5] template.

/etc/irods/irods_client_rest_cpp.json

The REST API service relies on a configuration file in `/etc/irods` that dictates which port is used for each endpoint, as well as how many threads it can run concurrently and its network timeout in seconds.

```
{
    "irods_rest_cpp_access_server" : {
        "port" : 8080,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_admin_server" : {
        "port" : 8087,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_auth_server" : {
        "port" : 8081,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_get_configuration_server" : {
        "port" : 8088,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10,
        "api_key" : "default_api_key"
    },
    "irods_rest_cpp_put_configuration_server" : {
        "port" : 8089,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_list_server" : {
```

```
        "port" : 8082,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_query_server" : {
        "port" : 8083,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_stream_get_server" : {
        "port" : 8084,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_stream_put_server" : {
        "port" : 8085,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_zone_report_server" : {
        "port" : 8086,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    }
}
```

/etc/nginx/sites-available/irods_client_rest_cpp_reverse_proxy.conf

The nginx reverse proxy relies on a configuration file (sometimes deployed to `/etc/nginx/sites-available`) which defines how to route incoming paths to the separate REST API ports.

```
server {
    listen 80;

    add_header 'Access-Control-Allow-Origin' '*' always;
    add_header 'Access-Control-Allow-Headers' '*' always;
    add_header 'Access-Control-Allow-Methods' 'AUTHORIZATION,ACCEPT,GET,POST,OPTIONS,PUT,DELETE' always;

    location /irods-rest/1.0.0/access {
        if ($request_method = 'OPTIONS') {
            return 204;
        }
        proxy_pass http://localhost:8080;
    }

    location /irods-rest/1.0.0/admin {
        if ($request_method = 'OPTIONS') {
            return 204;
        }
        proxy_pass http://localhost:8087;
    }
```

```
location /irods-rest/1.0.0/auth {
    if ($request_method = 'OPTIONS') {
        return 204;
    }
    proxy_pass http://localhost:8081;
}

location /irods-rest/1.0.0/configuration {
    if ($request_method = 'OPTIONS') {
        return 204;
    }

    if ($request_method = GET ) {
        proxy_pass http://localhost:8088;
    }

    if ($request_method = PUT ) {
        proxy_pass http://localhost:8089;
    }
}

location /irods-rest/1.0.0/list {
    if ($request_method = 'OPTIONS') {
        return 204;
    }
    proxy_pass http://localhost:8082;
}

location /irods-rest/1.0.0/query {
    if ($request_method = 'OPTIONS') {
        return 204;
    }
    proxy_pass http://localhost:8083;
}

location /irods-rest/1.0.0/stream {
    if ($request_method = 'OPTIONS') {
        return 204;
    }

    if ($request_method = GET ) {
        proxy_pass http://localhost:8084;
    }

    if ($request_method = PUT ) {
        proxy_pass http://localhost:8085;
    }
}

location /irods-rest/1.0.0/zone_report {
```

```
        if ($request_method = 'OPTIONS') {
            return 204;
        }
        proxy_pass http://localhost:8086;
    }
}
```

AUTHENTICATION

This REST API relies on the use of JSON Web Tokens (JWT) to communicate identity, authentication information, authorization information, and in the future, role-based information.

A JWT is generated by invoking the `/auth` endpoint. This JWT must be reused by the client as an Authentication header for use with subsequent requests to other endpoints.

ENDPOINTS

The following endpoints describe the entirety of the REST API at this time. Additional endpoints or changes to these initial endpoints could change the way an application may have to interact with the API. This initial look at the REST API could change before version 1.0.0.

/access

This endpoint provides a service for the generation of an iRODS ticket to a given logical path (either a collection or a data object). A ticket can grant read or write permission to an otherwise anonymous user.

`POST` Parameters:

- path: The url-encoded logical path to a collection or data object for which access is desired

Example Usage:

```
curl -X POST -H "Authorization: ${TOKEN}" \
"http://localhost/irods-rest/1.0.0/access?path=%2FtempZone%2Fhome%2Frods%2Ffile0"
```

Returns:

An iRODS ticket token within the X-API-KEY header, and a URL for streaming the object.

```
{
  "headers": [
    "X-API-KEY: CS11B8C4KZX2BIl"
  ],
  "url": "/irods-rest/1.0.0/stream?path=%2FtempZone%2Fhome%2Frods%2Ffile0&offset=0&limit=33064"
}
```

/admin

The administration interface to the iRODS Catalog which allows the creation, removal and modification of users, groups, resources, and other entities within the zone.

`POST` Parameters:

- action: dictates the action to be taken (add, modify, or remove)
- target: the subject of the action: user, zone, resource, childtoresc, childfromresc, token, group, rebalance, unusedAVUs, specificQuery
- arg2: generic argument, could be user name, resource name, depending on the value of action and target
- arg3: generic argument, see above
- arg4: generic argument, see above
- arg5: generic argument, see above
- arg6: generic argument, see above
- arg7: generic argument, see above

Example Usage:

```
curl -X POST -H "Authorization: ${TOKEN}" \
"http://localhost/irods-rest/1.0.0/admin?action=add&target=resource&arg2=ufs0&
arg3=unixfilesystem&arg4=/tmp/irods/ufs0&arg5=&arg6=tempZone&arg7="
```

Returns:

"Success" or an iRODS exception

/auth

This endpoint provides an authentication service for the iRODS zone.

Currently only native iRODS authentication is supported, as `Basic` or `Native`.

`POST` Parameters:

- None

Example Usage:

```
export BASE64USERPASS=$(echo -n "rods:secret" | base64 -)
export TOKEN=$(curl -X POST -H "Authorization: Basic ${BASE64USERPASS}" \
"http://localhost:80/irods-rest/1.0.0/auth")
```

Returns:

An encrypted JWT which contains everything necessary to interact with the other endpoints. This token is expected in the Authorization header for the other endpoints.

/configuration

This endpoint will return a JSON structure holding the configuration for an iRODS server. This endpoint takes a known API key for authorization which is configured in `/etc/irods/irods_client_rest_cpp.json`.

`GET` Parameters:

Example Curl Command:

```
curl -X GET -H "X-API-KEY: ${API_KEY}" \
"http://localhost/irods-rest/1.0.0/configuration"
```

Returns:

A JSON array of objects whose key is the file name and whose contents is the configuration file.

```
{
    "host_access_control_config.json": {
        <SNIP>
    },
    "hosts_config.json": {
        <SNIP>
    },
    "irods_client_rest_cpp.json": {
        <SNIP>
    },
    "server_config.json": {
        <SNIP>
    }
}
```

This endpoint will write the url-encoded JSON to the specified files in `/etc/irods`.

`PUT` Parameters:

cfg - a url encoded JSON string of the format

```
[
    {
        "file_name": "test_rest_cfg_put_1.json",
        "contents" : {
            "key0" : "value0",
            "key1" : "value1"
        }
    },
    {
        "file_name": "test_rest_cfg_put_2.json",
        "contents" : {
            "key2" : "value2",
            "key3" : "value3"
```

```
        }
    }
]
```

Example Usage:

```
export CONTENTS="%5B%7B%22file_name%22%3A%22test_rest_cfg_put_1.json%22%2C
%20%22contents%22%3A%7B%22key0%22%3A%22value0%22%2C%22key1%22%20%3A%20
%22value1%22%7D%7D%2C%7B%22file_name%22%3A%22test_rest_cfg_put_2.json%22%2C
%22contents%22%3A%7B%22key2%22%20%3A%20%22value2%22%2C%22key3%22%20%3A
%20%22value3%22%7D%7D%5D"
curl -X PUT -H "Authorization: ${TOKEN}" \
"http://localhost/irods-rest/1.0.0/configuration?cfg=${CONTENTS}"
```

Returns:

None

/list

This endpoint provides a recursive listing of a collection, or stat, metadata, and access control information for a given data object.

Method : GET

Parameters: path : The url encoded logical path which is to be listed stat : Boolean flag to indicate stat information is desired permissions : Boolean flag to indicate access control information is desired metadata : Boolean flag to indicate metadata is desired offset : number of records to skip for pagination limit : number of records desired per page

Example Curl Command:

```
curl -X GET -H "Authorization: ${TOKEN}" \
"http://localhost/irods-rest/1.0.0/list?path=%2FtempZone%2Fhome%2Frods&stat=0&
permissions=0&metadata=0&offset=0&limit=100"
```

Returns:

A JSON structured response within the body containing the listing, or an iRODS exception

```
{
  "_embedded": [
    {
      "logical_path": "/tempZone/home/rods/subcoll",
      "type": "collection"
    },
    {
      "logical_path": "/tempZone/home/rods/subcoll/file0",
      "type": "data_object"
```

```
    },
    {
      "logical_path": "/tempZone/home/rods/subcoll/file1",
      "type": "data_object"
    },
    {
      "logical_path": "/tempZone/home/rods/subcoll/file2",
      "type": "data_object"
    },
    {
      "logical_path": "/tempZone/home/rods/file0",
      "type": "data_object"
    }
  ],
  "_links": {
    "first": "/irods-rest/1.0.0/list?path=%2FtempZone%2Fhome%2Frods&stat=0&
    permissions=0&metadata=0&offset=0&limit=100",
    "last": "/irods-rest/1.0.0/list?path=%2FtempZone%2Fhome%2Frods&stat=0&
    permissions=0&metadata=0&offset=UNSUPPORTED&limit=100",
    "next": "/irods-rest/1.0.0/list?path=%2FtempZone%2Fhome%2Frods&stat=0&
    permissions=0&metadata=0&offset=100&limit=100",
    "prev": "/irods-rest/1.0.0/list?path=%2FtempZone%2Fhome%2Frods&stat=0&
    permissions=0&metadata=0&offset=0&limit=100",
    "self": "/irods-rest/1.0.0/list?path=%2FtempZone%2Fhome%2Frods&stat=0&
    permissions=0&metadata=0&offset=0&limit=100"
  }
}
```


/query

This endpoint provides access to the iRODS General Query language, which is a generic query service for the iRODS catalog.

`GET` Parameters:

- query_string: A url-encoded general query
- query_limit: Number of desired rows
- row_offset: Number of rows to skip for paging
- query_type: Either 'general' or 'specific'

Example Usage:

```
curl -X GET -H "Authorization: ${TOKEN}" \
"http://localhost/irods-rest/1.0.0/query?query_limit=100&
row_offset=0&query_type=general&query_string=SELECT%20COLL_NAME%2C
%20DATA_NAME%20WHERE%20COLL_NAME%20LIKE%20%27%2FtempZone%2Fhome%2Frods%25%27"
```

Returns:

A JSON structure containing the query results


```
{
  "_embedded": [
    [
      "/tempZone/home/rods",
      "file0"
    ],
    [
      "/tempZone/home/rods/subcoll",
      "file0"
    ],
    [
      "/tempZone/home/rods/subcoll",
      "file1"
    ],
    [
      "/tempZone/home/rods/subcoll",
      "file2"
    ]
  ],
  "_links": {
    "first": "/irods-rest/1.0.0query?query_string=SELECT%20COLL_NAME%2C
    %20DATA_NAME%20WHERE%20COLL_NAME%20LIKE%20%27%2FtempZone%2Fhome%2Frods%25%27&
    query_limit=100&row_offset=0&query_type=general",
    "last": "/irods-rest/1.0.0query?query_string=SELECT%20COLL_NAME%2C
    %20DATA_NAME%20WHERE%20COLL_NAME%20LIKE%20%27%2FtempZone%2Fhome%2Frods%25%27&
    query_limit=100&row_offset=0&query_type=general",
    "next": "/irods-rest/1.0.0query?query_string=SELECT%20COLL_NAME%2C
    %20DATA_NAME%20WHERE%20COLL_NAME%20LIKE%20%27%2FtempZone%2Fhome%2Frods%25%27&
    query_limit=100&row_offset=0&query_type=general",
    "prev": "/irods-rest/1.0.0query?query_string=SELECT%20COLL_NAME%2C
    %20DATA_NAME%20WHERE%20COLL_NAME%20LIKE%20%27%2FtempZone%2Fhome%2Frods%25%27&
    query_limit=100&row_offset=0&query_type=general",
    "self": "/irods-rest/1.0.0query?query_string=SELECT%20COLL_NAME%2C
    %20DATA_NAME%20WHERE%20COLL_NAME%20LIKE%20%27%2FtempZone%2Fhome%2Frods%25%27&
    query_limit=100&row_offset=0&query_type=general"
  },
  "count": "4",
  "total": "4"
}
```


/stream

Stream data into and out of an iRODS data object

`PUT` and `GET` Parameters:

- path: The url-encoded logical path to a data object
- offset: The offset in bytes into the data object

- limit: The maximum number of bytes to read

Example `PUT` Usage:

```
curl -X PUT -H "Authorization: ${TOKEN}" -d"This is some data" \
"http://localhost/irods-rest/1.0.0/stream?
path=%2FtempZone%2Fhome%2Frods%2FfileX&offset=0&limit=1000"
```

Returns:

Nothing, or iRODS Exception

Example `GET` Usage:

```
curl -X GET -H "Authorization: ${TOKEN}" \
"http://localhost/irods-rest/1.0.0/stream?
path=%2FtempZone%2Fhome%2Frods%2FfileX&offset=0&limit=1000"
```

Returns:

The data requested in the body of the response

/zone_report

Requests a JSON formatted iRODS Zone report, containing all configuration information for every server in the zone.

`POST` Parameters:

- None

Example Usage:

```
curl -X POST -H "Authorization: ${TOKEN}" \
"http://localhost/irods-rest/1.0.0/zone_report"
```

Returns:

JSON formatted Zone Report

```
{
  "schema_version": "file:///var/lib/irods/configuration_schemas/v3/zone_bundle.json",
  "zones": [
    {
    <snip>
    }]
}
```

FUTURE WORK

This initial look at the new iRODS C++ REST API covers the majority of our use cases and design goals. We expect there to be some changes as we become more familiar with JWT usage and deployment patterns. Additional functionality is expected to grow around configuration management, as iRODS has never allowed outside processes to manipulate its configuration before. Handling zone-wide configuration in a single location has never been possible, so care must be taken to make sure there are sufficient protections in place (dry-run, rollback, etc.).

SUMMARY

The iRODS C++ REST API can now provide new access and increased speed of client development to modern web development frameworks. The Zone Management Tool (ZMT) is being built alongside this new API and the needs of each are driving the development process.

The functionality included in this first release is sufficient for full-featured applications to be rapidly built and deployed for the public, or for specific internal positions in large organizations.

REFERENCES

[1] Zhou, Bo; Draughn, Kory; Coposky, Jason; Russell, Terrell; Conway, Mike; iRODS Client: Metalnx 2.4.0 with GalleryView (2021)
https://irods.org/uploads/2021/Zhou-iRODS-Metalnx_2.4.0_with_GalleryView-slides.pdf

[2] Zhou, Bo; Russell, Terrell; iRODS Client: Zone Management Tool (ZMT) (2021).
https://irods.org/uploads/2021/Zhou-iRODS-Zone_Management_Tool_ZMT-paper.pdf

[3] Jones, M.; Bradley, J.; Sakimura, N.; JSON Web Token (JWT): RFC 7519 (2015).
https://datatracker.ietf.org/doc/html/rfc7519

[4] Hypermedia as the Engine of Application State (HATEOAS). https://en.wikipedia.org/wiki/HATEOAS

[5] nginx: HTTP and reverse proxy server. http://nginx.org/en/

iRODS Client: NFSRODS 2.0

Kory Draughn
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
korydraughn@renci.org

Terrell Russell
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
unc@terrellrussell.com

ABSTRACT

NFSRODS has been updated to version 2.0 and now provides significant performance improvements and caching capabilities. This talk will cover what has changed and future work.

Keywords

iRODS, client, NFS, NFSv4, data management

INTRODUCTION

Last year's NFSRODS 1.0 release[1] marked the point where iRODS filesystem integration became easy for system administrators to deploy and support. The functionality was in place, but there had been no usage. After a few deployments, the feedback began to come in and with large numbers of users and files, NFSRODS was not capable of keeping up. This work describes the efforts in moving from NFSRODS 1.0 to NFSRODS 2.0[2].

CHANGES SINCE 1.0

Many of the changes in NFSRODS 2.0 are incremental and represent standard software development and refinement. These include updates of its software dependencies, cleanup of warnings, and better integration with a Docker-based deployment environment.

Since iRODS 4.2.9 incorporated the functionality of the Update Collection MTime Rule Engine Plugin[3], that dependency on the server has been relaxed.

More substantial changes are represented by the work done to increase performance for the list operation.

NEW CONFIGURATION

NFSRODS 2.0 introduces six additional configuration options for deployment. These represent four cache timeout settings, a boolean to determine whether overwriting existing data objects is allowed through the NFS interface, and a workaround for an Oracle-backed iRODS deployment.

```
// The refresh time for cached iRODS object type information.
"object_type_refresh_time_in_milliseconds": 300000,

// The refresh time for cached iRODS user permissions information.
"user_permissions_refresh_time_in_milliseconds": 300000,

// The refresh time for cached iRODS user type information.
```

```
"user_type_refresh_time_in_milliseconds": 300000,

// The refresh time for cached GenQuery results used to produce the
// output of a list operation.
"list_operation_query_results_refresh_time_in_milliseconds": 30000,

// Specifies whether the force flag should be applied when overwriting
// an existing file. If this option is false, an error will be reported
// back to the client.
"allow_overwrite_of_existing_files": true,

// Allows NFSRODS to make adjustments for running against an Oracle-backed
// iRODS deployment.
"using_oracle_database": false
```

LIST OPERATION

The list operation of NFSRODS 1.0 had performance issues. It could not handle large collections. This was manifested by the return of truncated and inconsistent results. iRODS servers that were backed by Oracle databases were not returning the correct results. Redundant information was being queried from the catalog, and NFSRODS was naively executing identical queries upon every operation without caching any results.

To fix these list operation shortcomings, NFSRODS 2.0 now caches many, if not all, results retrieved from the Jargon (Java)[4] layer which interacts with iRODS. The Oracle support has been corrected and the default log level has been lowered to reduce time spent writing to disk.

Additionally, best practice has been established to turn off color listings in the calling terminal to reduce the number of stat operations and therefore reduce the network traffic with the connected iRODS server.

PERFORMANCE

To test the new speed of NFSRODS 2.0, the following test harness was configured:

- iRODS provider (backed by PostgreSQL) running on Ubuntu 16.04 (32 cores)
- NFSRODS container running on the same machine hosting the provider
- Set NFSRODS log level to INFO
- Mount command: `sudo mount -o port=2050 localhost:/ /mnt/nfsrods`
- Timing command: `time /bin/ls <collection> | wc -l`

Collection Size	1.0	2.0 (no cache)	2.0 (cached)
500	1m7.211s	0m0.857s	0m0.046s
1,000	3m49.246s	0m0.708s	0m0.041s
3,000	31m45.147s	0m2.345s	0m0.045s
6,000	86m56.771s (results truncated)	0m5.274s	0m0.052s
10,000	87m18.675s (results truncated)	0m9.302s	0m0.058s

Table 1. Performance Comparison between NFSRODS 1.0 and 2.0 (n=5)

Table 1 illustrates both the troubles that NFSRODS 1.0 had with large collections as well as the increased performance of NFSRODS 2.0. Each cell represents the mean of 5 runs, except the two largest collection sizes (6,000 and 10,000) for NFSRODS 1.0 where the results were truncated. An initial listing of a collection with 3,000 data objects was reduced by a factor of over 800x from over 31 minutes to just over 2 seconds. Once that listing information had been seen by NFSRODS 2.0 and cached, a second listing returned the results from local memory in 1/20th of a second, another 50x speedup.

These improvements make NFSRODS appear much more transparent to the user and ready for production deployments.

FUTURE WORK

While this optimization work makes NFSRODS usable, there is still work to be done. Large file transfers are still serial and can appear painfully slow to users who are used to the multi-threaded, network saturating performance of the iCommands. Parallel I/O support is now the highest priority feature to be added next.

Support for iRODS metadata, perhaps via extended file attributes will be investigated since iRODS metadata is currently not visible through the NFS network interface. Additionally, test coverage and error messages need to be improved.

SUMMARY

NFSRODS 2.0 provides a significant increase in performance for collection listings. This deployable production release will encourage continued community feedback about the best way to present iRODS to existing tools and applications.

REFERENCES

[1] Draughn, K., Russell, T., Mieczkowski, A., Coposky, J., Conway, M.: iRODS Client: NFSRODS 1.0. 8pp. 2020 iRODS User Group Meeting. (2020) `https://irods.org/uploads/2020/Draughn-iRODS-NFSRODS_v1.0.0-paper.pdf`

[2] iRODS Client - NFSRODS. `https://github.com/irods/irods_client_nfsrods`

[3] iRODS Rule Engine Plugin - Update Collection MTime. `https://github.com/irods/irods_rule_engine_plugin_update_collection_mtime`

[4] Jargon - iRODS Java client library. `https://github.com/DICE-UNC/jargon`

iRODS Client: Zone Management Tool (ZMT)

Bo Zhou
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
zbo@renci.org

Jason Coposky
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
jasonc@renci.org

Terrell Russell
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
unc@terrellrussell.com

ABSTRACT

The iRODS Zone Management Tool (ZMT) is a new client that uses the iRODS C++ REST API. It has a design goal of handling the administrative side of running an iRODS Zone (managing users/groups/resources, etc.). This paper will introduce the ZMT, current status, and future work.

Keywords

iRODS, data management, administration, REST, reactjs

INTRODUCTION

The iRODS ecosystem has historically been driven by user demand and much community development. Over the years, this has led to a number of different software clients issuing requests to the iRODS server. Some of those clients have been primarily for `rodsuser`-level operations, and others may have included some `rodsadmin`-level operations.

This new client will be `rodsadmin`-only, and is designed to provide a baseline and future home for a single-pane-of-glass to administer an entire iRODS Zone. ZMT aims to include managing users and groups and resources, but also server configuration itself, including policy editing, deployment, and rollback.

This paper provides a first look at the work so far.

DESIGN GOALS

The iRODS community needs a a user-friendly, easy to deploy, fast, and light-weight web-based graphical user interface (GUI) to manage an iRODS Zone. The iRODS Zone Management Tool[1] meets those design goals.

ZMT is based on the React.JS[2] single page application (SPA) framework and speaks only HTTP/HTTPS to the configured iRODS C++ REST API[3] endpoint. It exposes existing configuration of all iRODS Servers in a Zone and an administration endpoint to manipulate system elements similar to `iadmin`.

It is envisioned that ZMT will assume the administration duties of Metalnx[4] which should free that codebase of some of its historical complexity. As a `rodsadmin`-only tool, ZMT will continue to focus on administrator concerns.

CONFIGURATION

ZMT configuration is handled by a single environment file that is loaded at application startup. This `.env` file provides read-only variables that can control different aspects of a ZMT deployment.

iRODS UGM 2021 June 8-11, 2021, Virtual

The minimum required configuration involves a single value representing the iRODS C++ REST API endpoint for the iRODS Zone that ZMT is being used to manage:

```
REACT_APP_REST_API_URL=protocol://host:port
```

Additional configuration options are available to control which port ZMT itself runs on and how the application looks. A `sample.env` file is included in the repository for reference:

```
LISTEN_PORT=3000
REACT_APP_REST_API_URL=protocol://host:port
REACT_APP_APPBAR_LOGO=iRODS-logo.jpg
REACT_APP_LOGIN_LOGO=iRODS-logo-1.png
REACT_APP_BRANDING_NAME=Zone Management Tool
REACT_APP_PRIMARY_COLOR=#04bdaf
REACT_APP_SECONDARY_COLOR=#ffffff
```

ENDPOINTS

The initial release of the ZMT includes four endpoints, represented in the left sidebar as sections of the web application. Each section provides visibility over and affords management of a particular 'noun' in the iRODS Zone's namespace. These include `/servers`, `/resources`, `/users`, and `/groups`. The `/` (or `/home`) endpoint serves as an overview or dashboard and will be populated in future work.

/servers

The `/servers` section provides a table view (Figure 1) of basic server information in the local iRODS Zone. Displayed columns include, for each server, the role (Catalog Service Provider or Catalog Service Consumer), the hostname, the number of attached storage resources, and the operating system and version of that machine.

The data in this table is gathered from the results of querying the iRODS REST C++ REST `/zone_report` endpoint (similar to the output from `izonereport`. The table provides paging for a large number of servers and each column is sortable in two directions.

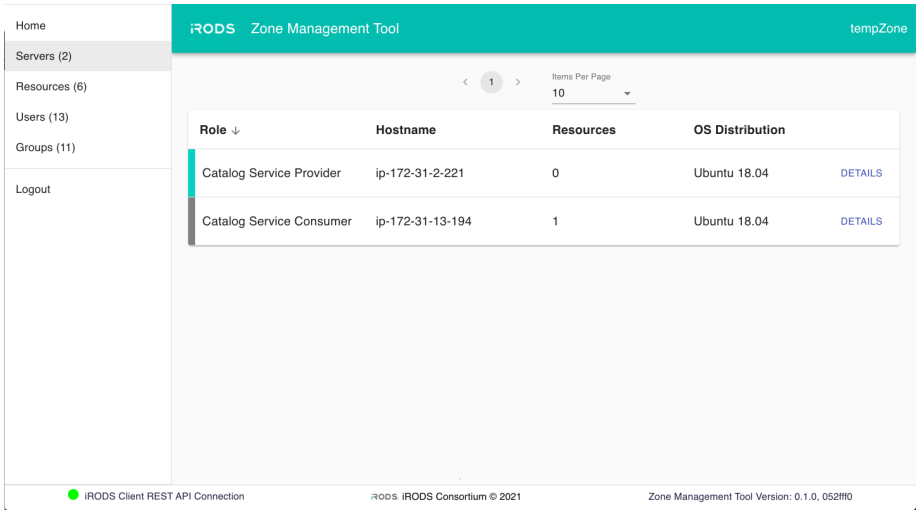


Figure 1. ZMT /servers - Table View

More details about each server are available in the Details View (Figure 2). This information includes the contents of each server's `server_config.json` file. Future work will allow for manipulation of the server information, but it is not editable via ZMT at this time.

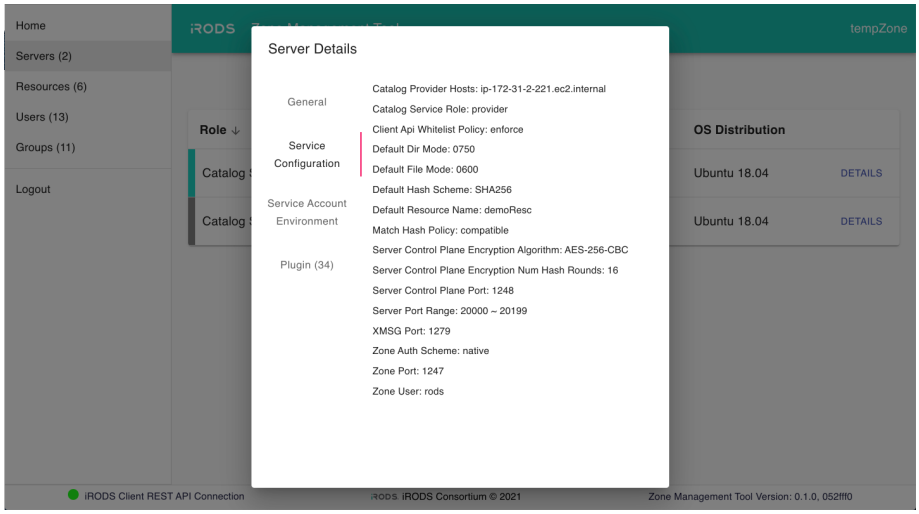


Figure 2. ZMT /servers - Details View

/resources

The `/resources` section provides two different views.

Similar to `/servers`, the table view (Figure 3) is a sortable set of columns including the name, type, associated hostname, and vault path of each resource. The listing can be live filtered by name and provides functionality for creating, renaming, removing, and otherwise updating resources (including context strings).

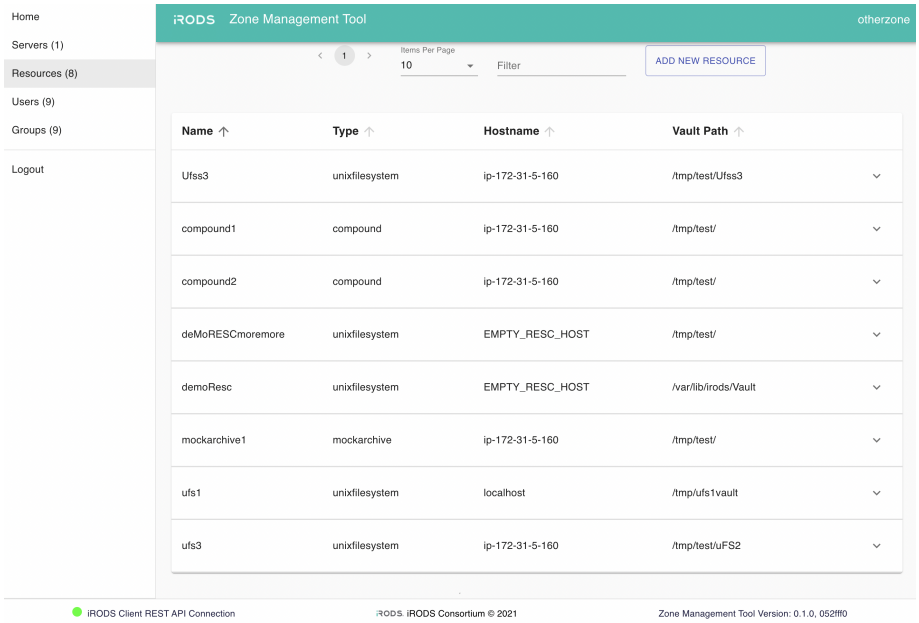


Figure 3. ZMT /resources - Table View

The tree view (Figure 4) shows the resource hierarchies of the connected Zone with the parent-child relationships between the composable resources[5].

The tree provides drag-and-drop editing capability of these parent-child relationships. A series of edits can be staged and reviewed before being undone or saved to the server. This allows an administrator to minimize the window of time when the resource hierarchies are being updated to their new target state.

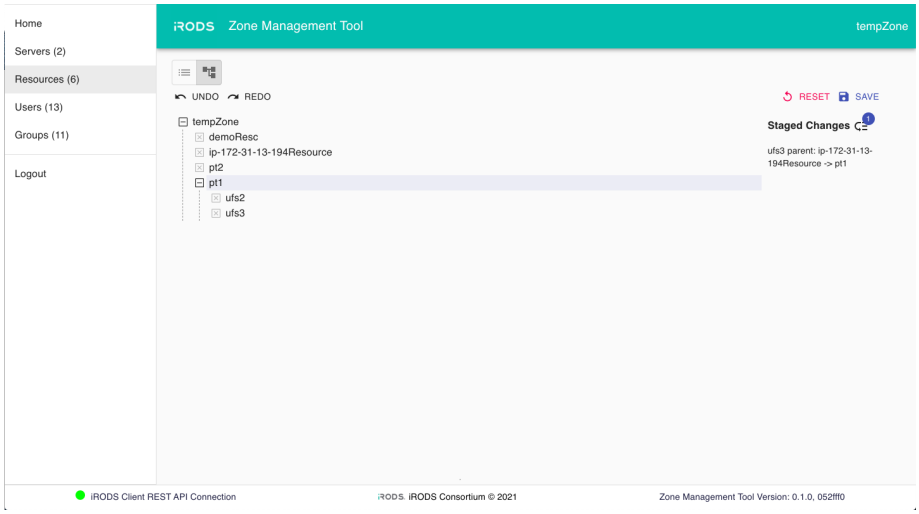


Figure 4. ZMT /resources - Tree View

/users

The `/users` section lists the users defined in the local Zone's catalog (Figure 5). This includes both local and remote (federated) users. The table can be paged and is filterable. New users can be created and users can be removed.

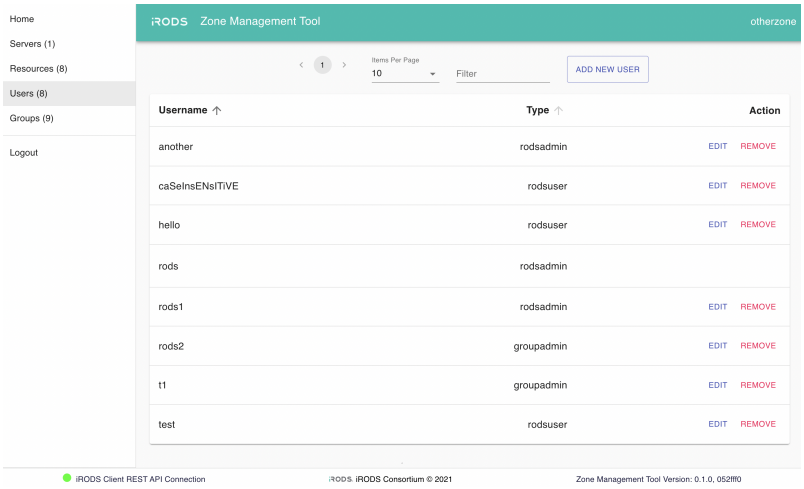


Figure 5. ZMT /users - Table View

When editing a particular user (Figure 6), the existing groups are listed and whether this user is a member of each group. A user can be added or removed from groups in this view.

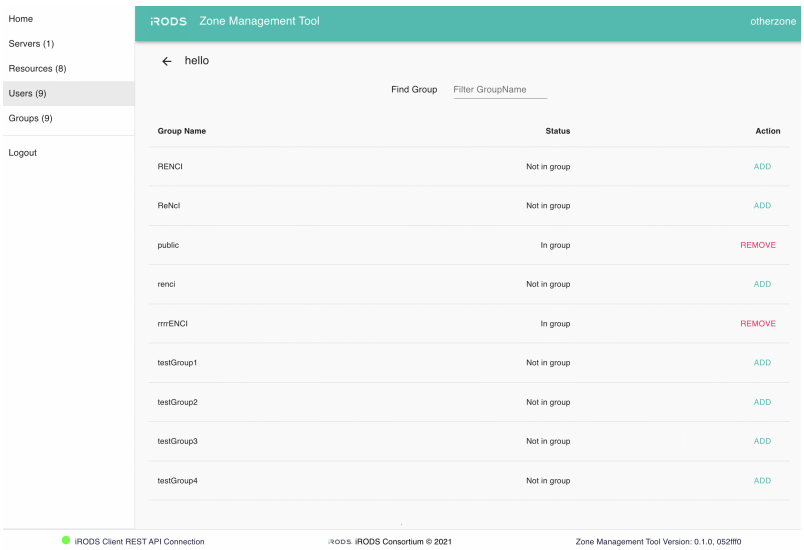


Figure 6. ZMT /users - Editing Group Membership

/groups

The `/groups` section functions similarly to `/users`. The table view (Figure 7) lists group name and the number of users in each group and allows for editing a group's membership. The table can be paged and is filterable. New groups can be created and groups can be removed.

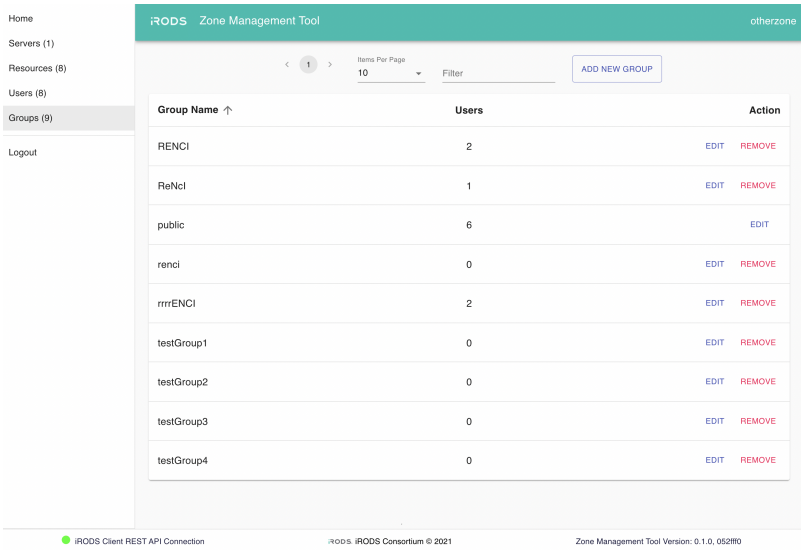


Figure 7. ZMT /groups - Table View

When editing a particular group (Figure 8), the existing users are listed and whether they currently belong to this group. Users can be added or removed from the group in this view.

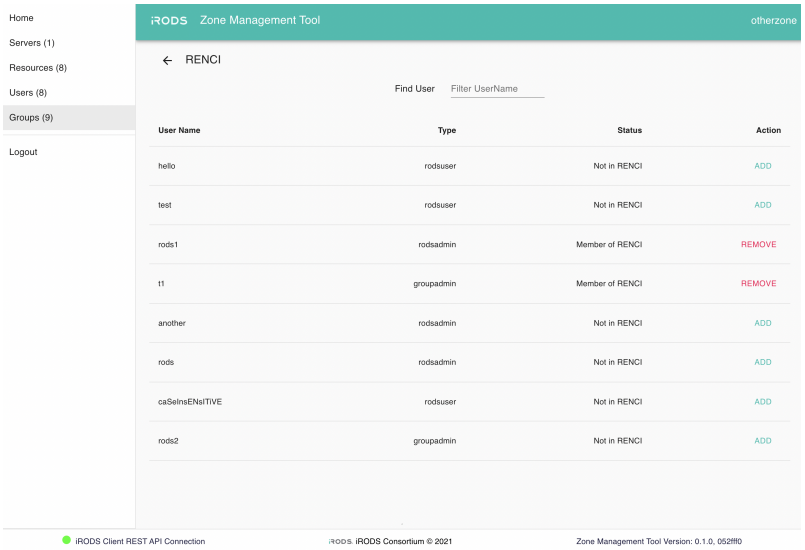


Figure 8. ZMT /groups - Editing Group Membership

iRODS REST API Endpoint Connection

The footer of the ZMT shows the iRODS REST API endpoint connection indicator. The ZMT uses four REST API endpoints to provide the different views (`/admin`, `/auth`, `/query`, and `/zone_report`). The overlay provides status information about whether those REST API endpoints are alive and responding.

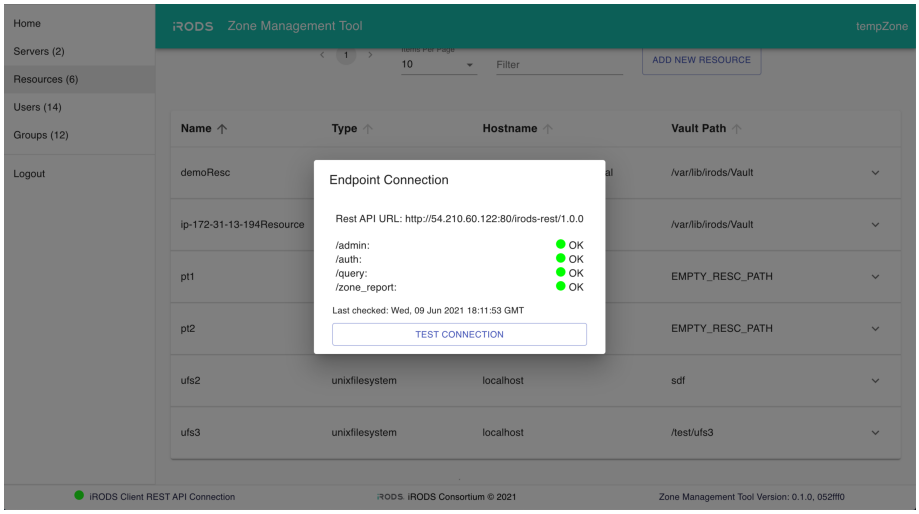


Figure 9. endpoint-connection

FUTURE WORK

As this is a first look at a new React-based iRODS administrative GUI, there are many features on the roadmap. Future capabilities will include management functions for the delay queue, tickets, and remote zones. There will be more monitoring provided for the different servers and their storage. These may include both up/down status indicators as well as health checks for some (un-)common cases that cause trouble for administrators.

In the farther future, we hope that ZMT will grow the ability to manage the server configuration directly, including policy sets and higher order capabilities.

SUMMARY

This paper provides a vision and describes a pre-release of the new iRODS Zone Management Tool (ZMT). Basic administration of resources, users, and groups is covered while connected to the iRODS C++ REST API. As this interface gains additional functionality, it will allow other applications to simplify and remove some of their `rodsadmin` complexity.

REFERENCES

[1] Zone Management Tool (ZMT). `https://github.com/irods/irods_client_zone_management_tool`

[2] React - A JavaScript library for building user interfaces. `https://reactjs.org/`

[3] Coposky, Jason; Russell, Terrell; iRODS Client: C++ REST API (2021)
`https://irods.org/uploads/2021/Coposky-iRODS-C_Plus_Plus_REST_API-paper.pdf`

[4] Zhou, Bo; Draughn, Kory; Coposky, Jason; Russell, Terrell; Conway, Mike; iRODS Client: Metalnx 2.4.0 with GalleryView (2021)
`https://irods.org/uploads/2021/Zhou-iRODS-Metalnx_2.4.0_with_GalleryView-slides.pdf`

[5] Russell, Terrell; Coposky, Jason; Johnson, Harry; Idaszak, Ray; Schmitt, Charles; E-iRODS Composable Resources (2013). iRODS User Group Meeting 2013.
`https://irods.org/uploads/2013/02/eirods-composable-resources.pdf`

iRODS Client: MetaInx 2.4.0 with GalleryView

Bo Zhou
Renaissance Computing Institute (RENCI)
University of North Carolina at Chapel Hill
zbo@renci.org

Kory Draughn
Renaissance Computing Institute (RENCI)
University of North Carolina at Chapel Hill
korydraughn@renci.org

Jason Coposky
Renaissance Computing Institute (RENCI)
University of North Carolina at Chapel Hill
jasonc@renci.org

Terrell Russell
Renaissance Computing Institute (RENCI)
University of North Carolina at Chapel Hill
unc@terrellrussell.com

Mike Conway
NIEHS / NIH
mike.conway@nih.gov

ABSTRACT

Metalnx 2.4.0 includes a new view when browsing iRODS Collections. This view displays thumbnails for images within a particular Collection. This talk will describe the process of adding this view to the client as well as explaining the server-side rule that provides the thumbnail information.