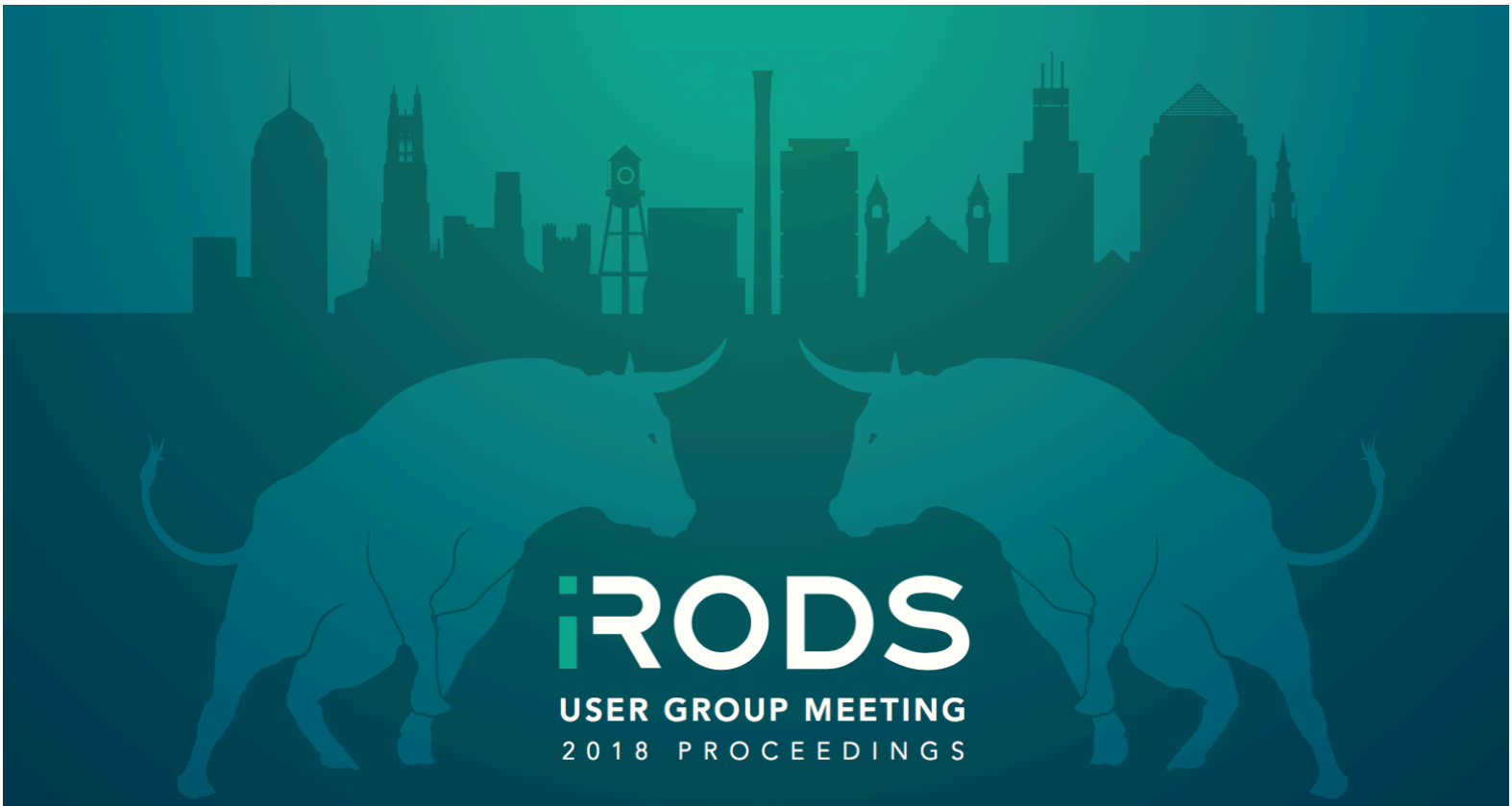

iRODS

USER GROUP MEETING

2018 PROCEEDINGS

PUBLISHED BY THE iRODS CONSORTIUM

iRODS
User Group Meeting 2018
Proceedings

10TH ANNUAL CONFERENCE SUMMARY

The iRODS User Group Meeting of 2018 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 three-day event was held from June 5th to 7th in Durham, North Carolina, hosted by the iRODS Consortium, with over 110 people attending. Attendees and presenters represented over 50 academic, government, and commercial institutions.

Contents

Listing of Presentations ……. 1

ARTICLES

Using iRODS to manage, share and publish research data: Yoda ……..…… 5

Ton Smeele, Lazlo Westerhof – Utrecht University ITS/RDM

iRODS Capability: Automated Ingest …… 13

Hao Xu, Alan King, Terrell Russell, Jason Coposky, Antoine de Torcy – RENCI, UNC-Chapel Hill

Listing of Presentations

The following presentations were delivered at the meeting:

iRODS Consortium Update
Jason Coposky – iRODS Consortium

iRODS Technology Update
Terrell Russell – iRODS Consortium

Towards a Parallel and Restartable Data Transfer Mechanism in iRODS
Zoey Greer and Hao Xu – iRODS Consortium

Virtues of Combining Locally Managed Storage with iRODS
Mark Pastor – Quantum

iRODS Deployment Seven Years On
John Constable – Wellcome Sanger Institute

Using iRODS to Manage, Share and Publish Research Data: Yoda
Ton Smeele and Lazlo Westerhof – Utrecht University

Data Archiving in iRODS
Matthew Saum – SURFsara

Deployment of a National Research Data Grid Powered by iRODS
Ilari Korhonen – KTH Royal Institute of Technology

The NIEHS Data Commons
Mike Conway – NIEHS

WDC, HGST, and iRODS
Linda Zhou – Western Digital Corporation

High-ish Availability Genomics
John Constable – Wellcome Sanger Institute

OpenID Connect Authentication in iRODS
Kyle Ferriter – RENCI at UNC-Chapel Hill

iRODS for Clinical and Instrument Data Lifecycle Management and Archiving
Masilamani Subramanyam – Genentech / Roche

iRODS Capability: Automated Ingest
Terrell Russell and Hao Xu – iRODS Consortium

iRODS Capability: Storage Tiering
Jason Coposky – iRODS Consortium

iRODS in the Cloud: SciDAS and NIH Helium Commons
Claris Castillo – RENCI at UNC-Chapel Hill

iRODS Usage at CC-IN2P3/CNRS
Yonny Cardenas – CC-IN2P3

The Brain Image Library
Derek Simmel – Pittsburgh Supercomputing Center, Carnegie Mellon University

FAIR Data Management and Disqoverability
Maarten Coonen – Maastricht University

Metalnx 2.0 – The Future of Metalnx
Stephen Worth – Worthwize Consulting
Terrell Russell – iRODS Consortium

Implementing a Storage Abstraction Service with iRODS
Jordan de La Houssaye – Bibliothéque nationale de France (BnF)

Discovery Environment and UNC's Implementation of a Community Edition
Don Sizemore – Odum Institute at UNC-Chapel Hill

A Pilot of iRODS for Managing Next Generation Sequencing Data
Todd Moughamer – Syngenta

Integrating Scale-Out NAS into an iRODS Data Fabric
Paul Evans – Daystrom

HydroShare: How iRODS Manages Data for a Hydrology Community of 1000's of Users
Ray Idaszak – RENCI at UNC-Chapel Hill

Lightning Talk: Scientific Animal Image Analysis (SANIMAL)
Tony Edgin – CyVerse at University of Arizona

ARTICLES

Using iRODS to manage, share and publish research data: Yoda

Ton Smeele
Utrecht University ITS/RDM
Heidelberglaan 8, Utrecht,
The Netherlands
a.p.m.smeele@uu.nl

Lazlo Westerhof
Utrecht University ITS/RDM
Heidelberglaan 8, Utrecht,
The Netherlands
l.r.westerhof@uu.nl

ABSTRACT

Researchers face challenges when they want to manage, share and publish their research data.

In 2014, Utrecht University commenced development of Yoda, a research data management system to meet these challenges. More recently features have been added to facilitate researchers to describe, deposit and publish research data in compliance with the FAIR principles.

Yoda deploys iRODS as its core component, customized with approximately 10,000 lines of rules and extended with a graphical user interface. It accommodates multiple metadata schemas to support varying requirements across fields of science.

We will discuss Yoda design principles related to the new features and their iRODS realization.

Keywords

Research data management, iRODS, FAIR, metadata.

INTRODUCTION

As the amount, complexity and use of digital research data grows, so does the need to manage data properly. The requirements for research data management may vary during the research data life cycle [1]. For instance, during creation, processing and analysis phases sensitive data typically is shared only between project members while the data sharing and reuse phases imply the need for publication of metadata to the research community at large. Other requirements such as compliance with law and institute policies have impact on the entire data life cycle.

Many research data management tools are focused only on the last phases of the data life cycle. Our approach is to support data management *during* research so that researchers can benefit from services immediately and data management integrates smoothly with existing research processes. In this paper we discuss key requirements and principles related to the design of Yoda and their realization using iRODS [2].

CHALLENGES

Need to safely collaborate on research data

An internal survey held in 2014 at Utrecht University amongst 3200 researchers revealed that approximately two third of the researchers work with research data which is at least partly sensitive. Half of the researchers that already work with sensitive data expect this fraction to increase over time.

European, national and institutional funders stimulate international and cross discipline research cooperation. Therefore while data may be sensitive, often there is a need to collaborate on data between research consortium members that

iRODS UGM 2018 June 5-7, 2018, Durham, North Carolina, USA

originate from different institutes and countries. An example is the Consortium on Individual Development [3], a longitudinal research project in the Netherlands that seeks to understand why most children develop well yet not all children manage to do so.

Need powerful tool yet easy to use

Some disciplines such as Bioinformatics tend to prefer to use the command line e.g. Linux bash to access their data. They may have to manage files with sizes in the order of many gigabytes for which fast and resumable data transfer is a key requirement to minimize the impact of poor network connections. Large data files may also imply a need to have multiple physical storage units act as one logical unit. The iRODS iCommands [4] are well suited to their needs.

Yet most disciplines demand a graphical user interface with intuitive access, informally summarized during a Yoda requirements for I-Lab [5] workshop as "We need Dropbox [6] on steroids". This workshop also revealed the wish to be able to revert to a previous revision of a data file e.g. in case of accidental deletion.

Need to comply with data policies

For reasons of reproducibility, Utrecht University's research data policy [7] requires researchers to retain data underpinning research output for a period of at least 10 years after conclusion of the research. This is in line with national [8] and European [9] guidelines. Patent, privacy or otherwise sensitive data necessitates appropriate measures to protect against unauthorized access.

All the above policies require research data to be protected against loss as a result of hazards or data corruption. Replication across multiple geographically distributed datacenters is adopted as a strategy against data loss by the EUDAT project's B2SAFE services [10].

Need support for data beyond the research project

Research data often has value beyond the scope of a research project. A UKRDS survey [11] reveals that duration of the data life cycle tends to vary per discipline where less than 20 years is common for business & management studies while e.g. geography data does not appear to expire. Therefore stewardship of data is unable to depend on the existence of and funding by the research project. This underpins the importance of well described data packages so that entitlements and reuseability of the data can be assessed at any time. The FAIR principles [12] provide researchers with guidance on the required level of data description.

The availability of reusable data can benefit new research and is promoted via initiatives such as the European Open Science Cloud [13].

Flexible cost coverage for data storage and data management is needed to support changes in stewardship and to support reuse based billing.

SOLUTION

Communities and data compartments

Yoda stores data in compartments and access to the compartment is granted only to users that are a member, a viewer or a data manager. Members have edit rights on data while viewers have read-only access to the data. Multiple data compartments are jointly refered to as a community[1]. Each community can have one or more data managers.

Data compartments are implemented as a set of iRODS groups so that authorizations can be role based. Note that in iRODS access rights granted on an object are equivalent to the sum of access granted to each group membership (in addition to access granted to the user specifically) so this conveniently fits with our approach.

[1] in the Yoda portal a community is listed as a "category"

Given a compartment named X belonging to community C then (full) members are listed in the iRODS group `research-X` and viewers are listed in the iRODS group `read-X`. Datamanagers are listed in a group `datamanager-C`.

Data compartments consist of a research "workspace" and an accompanying "vault". The vault is a space where data is deposited that must remain immutable forever for reasons of demonstrable data integrity. While data objects in the workspace are editable for full members and readable for viewers and data managers, the data in the vault is read-only for all roles. Researchers may opt to deposit data in the vault. Subsequently the data manager approves the deposit and the Yoda system takes care of the actual deposit action. Hence even the data manager cannot directly write to the vault.

Storage resource strategy

Each iRODS storage resource is tagged with a tier identifier that indicates its relative cost factor. Yoda administrators can edit this tag. Hence a community can be invoiced for the total size of data stored on Yoda based on the tier rate times the sum of data object sizes for each storage tier. This information is collected by a cron job and annotated to the iRODS group object. The Yoda statistics module shows users their current and rolling 12-months use per tier consolidated on a group and community level.

We use composable resources in a hierarchy. All user accessible resources are underneath resource `irodsResc`. The data stored here is replicated asynchronously to a top level resource `irodsRescRepl` located in another datacenter to protect it against loss. This is complemented by a revision strategy to maintain previous versions of a data object.

The `irodsResc` resource distributes data objects randomly among its children, the actual leaf storage resources. Each leaf has a pass-thru parent resource to allow an administrator to switch the resource on and off for reads and/or writes. This facilitates easy maintenance on resources. Resources can be decommissioned by replicating their data to another resource and then trimming the original resource.

Ease of use: graphical user interface

One of the requirements of Yoda is a graphical user interface with intuitive access. We meet this requirement through a combination of a network disk access to transfer and process data and a web portal to authorize access and to manage the data. The network disk interface Davrods [14] is written in C and supports WebDAV protocol class 2. The portal is written in PHP, JavaScript and HTML. Using the iRODS PHP library [15], the graphical user interface of Yoda is connected to the back office. It is strictly presentation layer only, all business logic is present in the back office. The business logic is implemented using iRODS rules and microservices. This facilitates that Yoda functions and policies are applied regardless of user interface.

Collaboration: revisions and folder locking

To satisfy the wish to be able to revert to a previous revision of a file and collaborate safely as a group in the research workspace, we have implemented folder locking and revisions.

Folder locking

To prevent modification of a folder in the research workspace, a researcher can use the lock function. This will catch attempts to write data to or to remove data from that folder until the folder is unlocked again. The same mechanism is used to prevent data modification while it is being approved and copied to the vault. We have opted not to use iRODS Access Control Lists (ACL) for this purpose. ACLs can be modified by the object creator which is not desirable for system-initiated lock actions.

Folder locks are implemented through metadata on the folder, its children and its parents. The lock consists of an iRODS Attribute-Value-Unit (AVU) triplet with `org_lock` as attribute name and the name of the collection that the lock applies to as its value. Whenever a folder is locked an AVU is set on every child recursively. In addition it is set on all parent folders up to and including `/{rodsZone}/home/research-{groupName}`.

Revisions

For each new file and upon a file modification Yoda creates a revision. This is a timestamped backup of the new or modified file in the revision store. The revision store is a system collection.

Not all revisions are kept, only a predefined number of revisions is kept per time bucket. A time bucket is a time offset from now into the past. Each revision strategy has a predefined set of time buckets and minimum number of revisions stored in those buckets.

Metadata handling

Metadata is registered in an XML [16] file together with the data in a data package. This XML file complies with a metadata schema chosen by the community. Researchers can register the metadata through the web portal or upload a metadata XML file directly through WebDAV. The second scenario is suitable for experiment software that can produce a metadata file automatically.

Metadata schemas

Each community in Yoda can configure its own metadata schema. Metadata can be entered and edited through a web form. A Yoda metadata form is constructed from three files (see Figure 1): an Extensible Markup Language (XML) file that contains the actual metadata, an XML Schema Definition (XSD) [17] file and a form elements XML file.

The XSD file lists requirements and restrictions that apply to the metadata. It describes properties such as input length and datatype of each metadata field and selectable options of List-Of-Values fields. While Yoda uses the XSD to validate the XML structure, it does not use the XSD to specify mandatory metadata fields required before a data folder can be submitted for inclusion into the vault.

The form elements XML file contains all presentation information relevant to the web portal metadata form and specifies which metadata fields are required when submitting a data folder into the vault. This file has been designed for the Yoda implementation and is not based on any standard apart from XML.

The Yoda metadata XML file holds the actual metadata as entered by the researcher in the research workspace.

In the research workspace, the XML metadata file is validated against the metadata schema XSD before it is loaded into the metadata form. If the metadata XML does not comply with the metadata schema, it will not be loaded into the metadata form. When a data folder is deposited to the vault, it is validated against the metadata schema XSD and against the form elements XML to check if all required metadata fields have been completed.

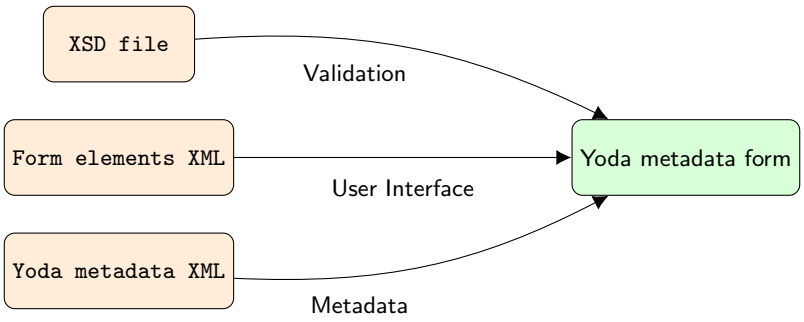

Figure 1. Construction of Yoda metadata form

Metadata indexing

The content of the XML metadata file is indexed in the iCAT database so that users can find data by its metadata. The submitted metadata forms are transformed with an Extensible Stylesheet Language Transformations (XSLT) [18] stylesheet into an XML containing AVUs. Subsequently this XML is loaded into the iCAT database using a microservice. Metadata is added to the data folder to make it searchable in the web portal.

Metadata mapping

When Yoda publishes a data package, the metadata stored in XML is transformed into several other formats using XSLT stylesheets. The first transformation creates a landing page for the publication. This page is published in a public directory. A second transformation generates a DataCite Metadata Schema 4.0 [19] compliant XML used for the DataCite Metadata Store. The last transformation creates an XML file used by the public OAI-PMH [20] service of Yoda which maps the Yoda metadata to Dublin Core Schema [21] and DataCite Metadata Schema 4.0.

Workflows

The two main workflows in Yoda are depositing a data package (see Figure 2) and publishing a data package (see Figure 3). We faced several challenges upon workflow implementation. First of all integrity: data packages in the vault must be protected from undesired changes. In case of desired changes we need traceability: it should be traceable who was responsible for each change. Furthermore we want interactive and scalable workflows. Users should not have to wait for tasks to finish and long running tasks should not block other processes.

To provide security and traceability for data packages in the vault, tasks in the vault require exclusive privileges. These privileges are granted to actors based on their role. To achieve interactive and scalable workflows, tasks with long or uncertain execution time must be executed asynchronously (e.g. file copy actions).

Workflow progress is registered as a state with the data package. The state of a data package is stored as metadata on the data package collection.

Actors

Yoda workflows involve three actors:

- **Researcher** Researchers initiate the workflows by submitting data folders for the vault and for publication.
- **Data manager** Data managers react on the actions initiated by the researchers, submitted data folders have to be approved by a data manager.
- **System** The system will copy and publish the packages after they have been approved by a data manager. Since tasks in the vault require exclusive privileges this is done by the system.

Data deposit workflow

Researchers can deposit their data folder from the research workspace into the vault. To start the data deposit workflow a researcher submits a data folder for inclusion into the vault. The data manager will check if the submitted data folder with metadata complies with institutional, faculty and other policies. If the data folder complies with policies the data manager accepts the data folder for the vault. If the data folder does not comply with policies the data manager rejects the data folder. The researcher can now change the data folder before resubmitting it to the vault. After a data folder is accepted for the vault the system will deposit a copy of the data folder in the vault. A data package is created and the data folder content is included in an `original` sub folder. The license chosen by the researcher is automatically added to the data package in the vault.

FAIR data publication workflow

Once secured in the vault, a data package can be published. The researcher initiates a data publication via the Yoda web portal by clicking the `submit for publication` button. The data manager accepts or rejects the request

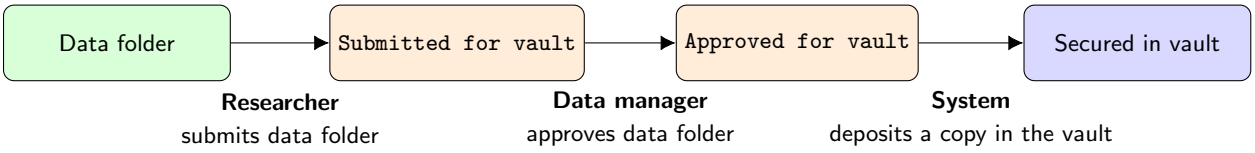

Figure 2. The workflow to deposit a data package into the vault

depending on whether the data package complies with publication policies. If the publication is rejected the researcher can make the necessary changes before submitting the data package again. After the data manager has accepted the publication request, the system will publish the data package. The system mints and registers a persistent Digital Object Identifier [22], submits the data package metadata to DataCite, publishes a landing page for the data package and makes the metadata available through OAI-PMH. These actions make the data package findable, accessible and reusable in line with the FAIR principles. Third parties can harvest the metadata of the published package through OAI-PMH. If the data package is classified as "open", the contents of the data package will be made publicly accessible as well.

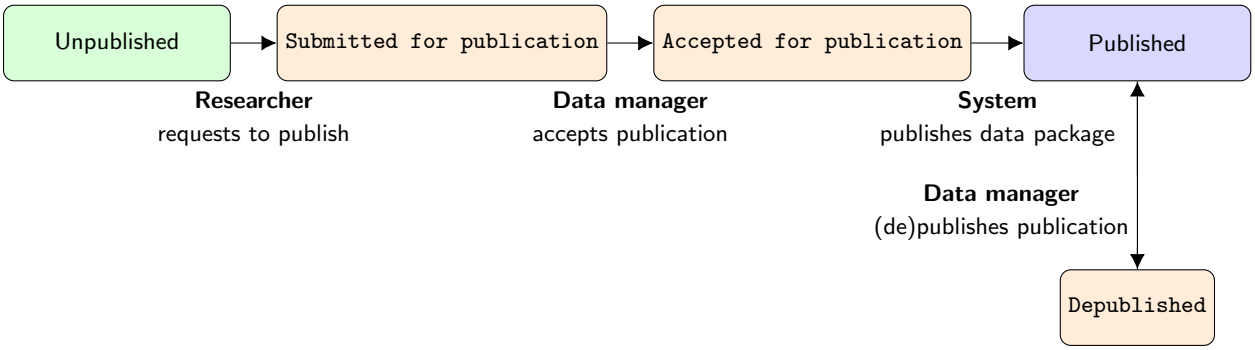

Figure 3. The workflow to publish a data package

FUTURE WORK

Going forward we intend to extend Yoda with functions to manage collections of data packages (e.g. manage membership and import, export and distribute data). Also we aim to support other data types such as software and experiment configurations.

CONCLUSION

The current functions of Yoda facilitate institutional researchers to manage and share sensitive data in line with policies during and after research. Yoda allows researchers to comply with grant requirements on data management and open data.

ACKNOWLEDGEMENTS

The development of Yoda would not have been possible without the input, support and executive sponsorship of researchers and IT management. In particular we would like to thank Chantal Kemner (Dynamics of Youth research) and her team. Chantal has provided the requirements for the initial Yoda functions that help to manage sensitive lab data and she trusted us to deliver on our promises. Also we thank Vincent Buskens (Institutions for Open Societies research) and his team. Vincent helped us to shape the data sharing and data publication functions. Finally we thank Carolien Besselink (IT Services) for her sustained executive sponsorship. Carolien put research data management on the IT Services agenda and facilitated the development of Yoda.

REFERENCES

[1] S. Higgins, "The DCC curation lifecycle model," *International Journal of Digital Curation*, vol. 3, no. 1, 2008.

[2] A. Rajasekar, R. Moore, C.-y. Hou, C. A. Lee, R. Marciano, A. de Torcy, M. Wan, W. Schroeder, S.-Y. Chen, L. Gilbert, *et al.*, "irods primer: integrated rule-oriented data system," *Synthesis Lectures on Information Concepts, Retrieval, and Services*, vol. 2, no. 1, pp. 1–143, 2010.

[3] C. Kemner, "Consortium on individual development." `http://www.individualdevelopment.nl`, 2014. Visted on 2018-05-31.

[4] "iRODS documentation." `httpss://docs.irods.org`. Visted on 2018-05-31.

[5] "Institutions for Open Societies." https://www.uu.nl/en/research/institutions-for-open-societies. Visited on 2018-05-31.

[6] "Dropbox." `http://www.dropbox.com`. Visted on 2018-05-31.

[7] E. Stiekema, "University policy framework for research data Utrecht University," 2017.

[8] Association of Universities in the Netherlands (VSNU), "The Netherlands Code of Conduct for Academic Practice." `http://www.vsnu.nl/files/documenten/Domeinen/Onderzoek/The_Netherlands_Code%20of_Conduct_for_Academic_Practice_2004_(version2014).pdf`, 2014. Visited on 2018-05-31.

[9] "Guidelines on FAIR Data Management in Horizon 2020." `http://ec.europa.eu/research/participants/data/ref/h2020/grants_manual/hi/oa_pilot/h2020-hi-oa-data-mgt_en.pdf`, 2018. Visited on 2018-05-31.

[10] EUDAT, "B2safe." https://eudat.eu/services/userdoc/b2safe. Visited on 2018-05-31.

[11] N. Beagrie, R. Beagrie, and I. Rowlands, "Research data preservation and access: The views of researchers," *Ariadne*, no. 60, 2009.

[12] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, *et al.*, "The fair guiding principles for scientific data management and stewardship," *Scientific data*, vol. 3, 2016.

[13] "European Open Science Cloud." `http://ec.europa.eu/research/openscience/index.cfm?pg=open-science-cloud`, 2018. Visited on 2018-05-31.

[14] C. Smeele and T. Smeele, "Davrods, an Apache WebDAV Interface to iRODS," in *iRODS User Group Meeting 2016 Proceedings*, pp. 41–47, iRODS Consortium, Dec. 2016.

[15] DICE-UNC, "Prods php irods client library." https://github.com/DICE-UNC/irods-php. Visited on 2018-05-31.

[16] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible markup language (xml).," *World Wide Web Journal*, vol. 2, no. 4, pp. 27–66, 1997.

[17] P. V. Biron, A. Malhotra, W. W. W. Consortium, *et al.*, "Xml schema part 2: Datatypes," 2004.

[18] World Wide Web Consortium (W3C), "XSL Transformations (XSLT) Version 2.0." `https://www.w3.org/TR/xslt20/`, 2007.

[19] DataCite Metadata Working Group, "DataCite Metadata Schema for the Publication and Citation of Research Data. Version 4.0." `http://doi.org/10.5438/0013`, 2016.

[20] Open Archives Initiative, "The Open Archives Initiative Protocol for Metadata Harvesting. Version 2.0." `http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm`, 2015.

[21] Dublin Core Metadata Initiative, "The Dublin Core Metadata Element Set." `http://www.ietf.org/rfc/rfc5013.txt`, 2007.

[22] I. ISO, "26324: Information and documentation-digital object identifer system," 2009.

iRODS Capability: Automated Ingest

Hao Xu
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
xuh@cs.unc.edu

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

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

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

Antoine de Torcy
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
adetorcy@renci.org

ABSTRACT

The iRODS Automated Ingest Framework is a new iRODS client that has been designed to scale up to match the demands of data coming off instruments, satellites, or parallel filesystems and provide a front door to the policy-based data management platform of iRODS.

Initial testing shows promising flexibility and a roughly linear performance curve.

Keywords

iRODS, capability, ingest, Celery, Redis, data management

INTRODUCTION

The iRODS Automated Ingest Framework[1] has been designed to solve two major use cases: registering large amounts of existing data into an iRODS namespace without moving the source data (filesystem scanning) and ingesting new or updated data from a known location in a filesystem (a landing zone) into place within an iRODS Vault.

Based on the Python iRODS Client[3], Celery[4], and Redis[5], the goal of this framework is to scale up to match the demands of data coming off instruments, satellites, or parallel filesystems and provide a front door to the policy-based data management platform of iRODS[2].

For testing, this framework has been deployed manually. For enterprise customers, this framework is prototyped as Docker containers to be deployed and run on a Kubernetes cluster via Helm charts.

ARCHITECTURE

Overview

The motivation for this tool was to provide a parallel and distributed means of getting data into the iRODS catalog. Celery and Redis provide the coordination mechanism for concurrent 'stat' gathering from the source location as well as concurrent iRODS connections for an initial scan. They also provide a very fast insulating layer for a 'delta sync' when a data source is scanned again and many of the source files have not changed. In this case, the Redis cache reports that nothing has changed for a particular file, and the client determines that no connection to iRODS is necessary. Delta scans are much faster than initial scans for this reason.

iRODS UGM 2018 June 5-7, 2018, Durham, North Carolina, USA

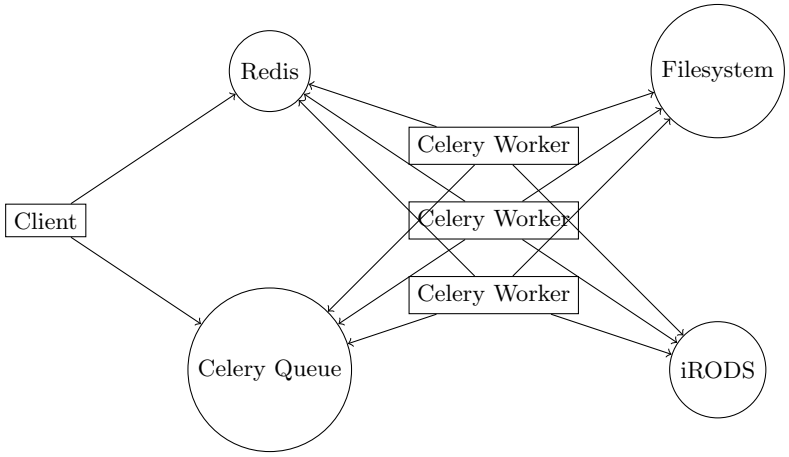

Figure 1. The number of Celery Workers can scale up to meet the required performance demand.

Client

The automated ingest client takes an initial path to scan, submits the job to the celery queue, and sets the following necessary Redis metadata. If the job is to be run a single time, the job name is added to the `singlepass` list. If the job is to be run continuously (scanning the same source again and again), then the job name is added to the `periodic` list. These two lists keep track of jobs that are running. If the job is in `singlepass`, a `restart` task is called one time synchronously to start the job. If the job is in `periodic`, a `restart` task is added to the `restart` queue which will start the work again once the current pass is complete.

The client can also stop a job and all tasks under that job.

Celery Queue

The `restart` task resets the `count` and `dequeue` lists, and the `tasks`, `retries`, and `failures` counters. The `restart` task calls a `sync_path` task that recursively and asynchronously walks the requested filesystem location (the source).

If the requested path is a directory, the `sync_path` task calls a `sync_dir` task that asynchronously creates and populates the metadata for that directory in the Redis cache, creates the collection in iRODS, and then lists and calls `sync_path` asynchronously on the immediate children of the directory. The `sync_dir` task compares the last sync time with the mtime and ctime of the directory, and if the directory has changed, the collection in iRODS is synchronized.

If the requested path is a file, the `sync_path` task calls a `sync_file` task that asynchronously creates and populates the metadata for that file in the Redis cache, and then puts or registers the file into iRODS. If the Redis entry already exists, the `sync_file` task compares the last sync time with the mtime and ctime of the file, and if the file has changed, the file contents and system metadata in iRODS are synchronized.

Before a task is added to the queue, the `count` counter is incremented and the task id is added to the task list. Each task has a retry handler, a failure handler, and an after return handler. The retry handler and failure handler increments the `retry` and `failure` counter respectively. The after return handler decrements the `count` counter and removes the task id from the `tasks` list. When the `tasks` counter is zero, it calls the cleanup function.

Redis

A Redis database can be used by Celery as a broker. Another Redis database is used to store metadata about jobs, including the `singlepass` and `periodic` lists, the `count` and `dequeue` lists, and the `tasks`, `retries`, and `failures`

counters. The `singlepass` list contains all the names of single pass jobs, the `periodic` list contains all the names of periodic jobs. These two lists are used to calculate running tasks for stopping a job. The `count` list contains all tasks created by the job and the `dequeue` list contains all tasks that are finished (with either success or failure). These two lists are used to calculate the list of running tasks. A running task can be stopped by name. The `tasks` counter is used to keep track of the remaining tasks. When the `tasks` counter reaches zero, the cleanup function is triggered. The `retries` and `failures` counters keep track of retried and failed tasks.

Event Handlers

The automated ingest Celery workers can be deployed with a variety of options to describe their behavior around gathering, preparing, and sending information to iRODS. These options are described in event handler files and handed to the workers. This allows for custom behavior to be written for particular deployments of the Ingest Framework. The event handler methods made available to the workers include:

method	**effect**	**default**
pre_data_obj_create	user-defined Python	none
post_data_obj_create	user-defined Python	none
pre_data_obj_modify	user-defined Python	none
post_data_obj_modify	user-defined Python	none
pre_coll_create	user-defined Python	none
post_coll_create	user-defined Python	none
pre_coll_modify	user-defined Python	none
post_coll_modify	user-defined Python	none
as_user	takes action as this iRODS user	authenticated user
target_path	set mount path on the iRODS server which can be different from client mount path	client mount path
to_resource	defines target resource request of operation	as provided by client environment
operation	defines the mode of operation	Operation.REGISTER_SYNC
max_retries	defines max number of retries on failure	0
timeout	defines seconds until job times out	3600
delay	defines seconds between retries	0

Table 1. Available event handler methods

Where user-defined Python can be written, the event handler is just providing hooks for data preparation or bookkeeping. The other methods are either Celery-specific or iRODS-specific configuration options for the operation to be performed.

Operations

The event handler method `operation` defines the mode for any iRODS connection. The following six operations are available and determine whether data is transferred and how the iRODS catalog is updated:

operation	new files	updated files
Operation.REGISTER_SYNC (default)	registers in catalog	updates size in catalog
Operation.REGISTER_AS_REPLICA_SYNC	registers first or additional replica	updates size in catalog
Operation.PUT	copies file to target vault, and registers in catalog	no action
Operation.PUT_SYNC	copies file to target vault, and registers in catalog	copies entire file again, and updates catalog
Operation.PUT_APPEND	copies file to target vault, and registers in catalog	copies only appended part of file, and updates catalog
Operation.NO_OP	no action	no action

Table 2. Available event handler operation modes

SOURCE AS S3

The Automated Ingest Framework was originally designed for ingestion from mounted filesystems. However, during the framework's development, an additional use case of ingesting from an existing S3 bucket was presented. Once a suitable python library was identified (`minio`), ingesting data via the S3 protocol worked as expected.

FLEXIBILITY

With these different configuration options, the functionality required for a particular use case can be realized.

The two main use cases that are solved by this framework are the filesystem scanner (source files stay in place after scanning) and the landing zone (source files are moved aside in some manner after being scanned). The good way to think about the difference between these two ways of setting up the Automated Ingest Framework is to consider where the source of truth will be once the scanning has been performed.

With the filesystem scanner use case, the truth remains in the original source location since that data could continue to move as new science is performed and other systems are writing into that source location. This is most likely to be useful when putting jobs into the `periodic` list.

With the landing zone use case, the truth now lies in the iRODS Catalog. iRODS has a copy of the data under management and it is now owned and operated within a Vault that iRODS controls. This is most likely to be useful when putting jobs into the `singlepass` list.

Within these two different use cases, there is the opportunity for registering new physical replicas of already registered data, syncing updated data, or even only updating a delta if the source material has been appended.

These different settings, in addition to the pre- and post- methods for data object and collection creation and modification, provide a full programmatic surface for writing data preparation policy and harvesting of metadata from external sources on the way to having data ingested into iRODS.

EARLY PERFORMANCE

Initial Ingest

This data is preliminary, and we expect the rates to increase once we have more experience with enterprise configurations and topologies.

However, we see that over 4M files can be ingested by 32 workers in an hour and a half with this initial codebase (Table 3).

What is also notable is that the performance is relatively linear until the catalog provider itself is overwhelmed by the number of incoming concurrent connections (and their subsequent concurrent connections to the underlying iCAT database).

files	workers	minutes	files/worker/minute
1585094	32	36	1375.95
4000000	32	90	1388.89
13564110	32	299	1417.65

Table 3. Three early samples of rate of ingest.

Delta Sync

When scanning the same data source again, the Redis cache handles most of the load and since it is an in-memory data structure store, it never needs to touch the disk or the network. It can handle the lookups very quickly and prevent the system from needing to go to the iRODS catalog except where a source file has changed.

For the most common of loads (a few files per thousand have changed), the delta sync (Table 4) runs nearly 10x as fast as the initial ingest (Table 3).

files	workers	minutes	files/worker/minute
247641	32	0.633	12219.13
2337705	32	6.166	11847.76

Table 4. Syncing data already in Redis is much faster.

SUMMARY

The iRODS Automated Ingest Framework is a new client that is being tested as it is being built. It has been designed to solve a myriad of interesting data ingesting scenarios and scale out to keep up with incoming data rates from batteries of sensors, microscopes, sequencers, and satellites.

Early indications show that the performance is roughly linear as the number of workers scales up.

We are actively looking for additional use cases and look forward to increased community feedback.

REFERENCES

[1] iRODS Capability Automated Ingest. `https://github.com/irods/irods_capability_automated_ingest`

[2] Xu, H., Russell, T., Coposky, J., et al: iRODS Primer 2: Integrated Rule-Oriented Data System. In: Synthesis Lectures on Information Concepts, Retrieval, and Services. 131pp. Morgan Claypool. (2017)

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

[4] Celery: Distributed Task Queue. `http://www.celeryproject.org/`

[5] Redis. `https://redis.io/`