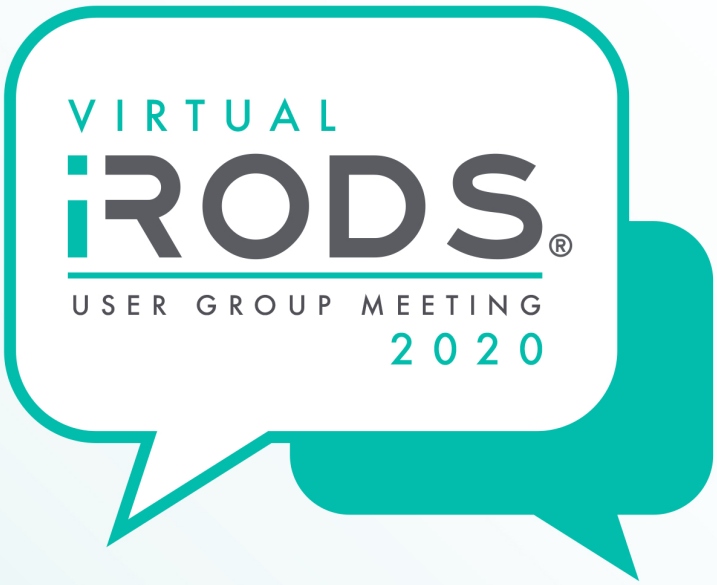

USER GROUP MEETING
2020 PROCEEDINGS

PUBLISHED BY THE iRODS CONSORTIUM

iRODS
User Group Meeting 2020
Proceedings

12TH ANNUAL CONFERENCE SUMMARY

The Virtual iRODS User Group Meeting of 2020 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 9th to 12th, hosted by the University of Arizona and the iRODS Consortium, with over 220 people attending from 19 countries. Attendees and presenters represented over 100 academic, governmental, and commercial institutions.

TALKS AND PAPERS

iRODS UGM 2020 Keynote

A Conversation With Your Data Platform

Nirav Merchant – CyVerse / University of Arizona

iRODS Consortium Update

Jason Coposky – iRODS Consortium

iRODS Technology Update

Terrell Russell. Kory Draughn, Alan King, Daniel Moore, Jaspreet Gill – iRODS Consortium

Yoda and the iRODS Python Rule Engine Plugin …… 11

Lazlo Westerhof, Chris Smeele – Utrecht University

Using JSON Schemas as metadata templates in iRODS …… 13

Venustiano Soancatl Aguilar – University of Groningen

S3:TNG – iRODS S3 Resource Plugin with Direct Streaming …… 15

Justin James, Kory Draughn, Jason Coposky, Terrell Russell – iRODS Consortium

Parallel data migration between GPFS filesystems via the iRODS rule engine …… 25

Ilari Korhonen – KTH Royal Institute of Technology

Policy-Encapsulated Objects 27

Arcot Rajasekar – University of North Carolina at Chapel Hill

Integration of iRODS with IBM Spectrum Archive Enterprise Edition – A flexible tiered storage archiving solution 29

Nils Haustein – IBM European Storage Competence Center

Mauro Tridici – Euro-Mediterranean Center on Climate Change (CMCC)

SmartFarm Data Management 37

Kieran Murphy – Agriculture Victoria

Data management in autonomous driving projects 39

Marcin Stolarek, Radosław Rowicki, Kacper Abramczyk, Mateusz Rejkowicz – Aptiv

CyVerse Discovery Environment: Extensible Data Science workbench and data-centric collaboration platform powered by iRODS 41

Sarah Roberts, Sriram Srinivasan, Nirav Merchant, Tina Lee – CyVerse / University of Arizona

iRODS and Federated Identity authentication: current limitations and perspective 43

Claudio Cacciari, Stefan Wolfsheimer, Hylke Koers, Arthur Newton, Tasneem Rahaman-Khan, Matthew Saum, Gerben Venekamp – SURF

The Past, Present and Future of iRODS at the Texas Advanced Computing Center 45

Chris Jordan – The University of Texas at Austin

iRODS_CSharp 47

Reink Fidder, Jelle Teeuwissen – Utrecht University

Best Student Technology Award Winner

Using iRODS to build a research data management service in Flanders 49

Ingrid Barcena Roig – KU Leuven

Application of iRODS to NIEHS Data Management 51

Mike Conway, Deep Patel – NIEHS / NIH

iRODS Client: NFSRODS 1.0 53

Kory Draughn, Terrell Russell, Alek Mieczkowski, Jason Coposky – iRODS Consortium

Mike Conway – NIEHS / NIH

iRODS Rule Engine Plugin: Hard Links 4.2.8.0 61

Kory Draughn, Terrell Russell – iRODS Consortium

Creating an iRODS zone with Terraform 63

Brett Hartley – Wellcome Sanger Institute

Building a national Research Data Management (RDM) infrastructure with iRODS in the Netherlands 65

Saskia van Eeuwijk, Hylke Koers – SURF

iRODS at Bristol Myers Squibb: Status and Prospects. Leveraging iRODS for scientific applications in Amazon AWS Cloud 67

Mohammad Shaikh, Oleg Moiseyenko – Bristol Myers Squibb

Keeping Pace with Science: the CyVerse Data Store in 2020 and the Future 69

Tony Edgin, Edwin Skidmore – CyVerse / University of Arizona

iRODS Logical Quotas Policy Plugin …… 71

Jonathon Anderson – University of Colorado Research Computing

Kory Draughn, Terrell Russell – iRODS Consortium

iRODS Policy Composition: Principles and Practice …… 73

Jason Coposky, Terrell Russell – iRODS Consortium

iRODS Client: AWS Lambda Function for S3 1.0 …… 75

Terrell Russell – iRODS Consortium

LIGHTNING TALKS

– iRODS / Globus Partnership Announcement

– Vas Vasiliadis – Globus

– Jason Coposky – iRODS Consortium

– Development Plan for iRODS Kubernetes Storage Driver

– Illyoung Choi – CyVerse / University of Arizona

– A Demo of irods/irods_demo

– Alan King – iRODS Consortium

– Upgrading iRODS from 4.1.12 to 4.2.7: Re-live the thrills and spills of an iRODS Administrator

– John Constable – Wellcome Sanger Institute

– Ansible Modules for iRODS using python-irodsclient

– John Xu – CyVerse / University of Arizona

– More Transport, Please!

– Kory Draughn – iRODS Consortium

– irods-fish

– Tony Edgin – CyVerse / University of Arizona

– Using iRODS as an entry point to VITAM for long-term preservation

– Samuel Viscapi – CINES

– CyVerse Continuous Analysis: Even a cave man can do it!

– Calvin McLean – CyVerse / University of Arizona

– The delay server rewrite: A tour of query_processor

– Alan King – iRODS Consortium

CLOSING REMARKS

Call to Action

– Nirav Merchant – CyVerse / University of Arizona

Yoda and the iRODS Python Rule Engine Plugin

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

Chris Smeele
Utrecht University
Utrecht, Netherlands
c.j.smeele@uu.nl

ABSTRACT

At the UGM 2018, we presented Yoda, a system for reliable, long-term storing and archiving large amounts of research data during all stages of a study. It facilitates researchers to describe, deposit, and publish research data in compliance with the FAIR principles.

Yoda deploys iRODS as its core component, customized with more than 10,000 lines of iRODS rules. With the release of the iRODS Python rule engine plugin, we sought to make use of the benefits it provides in areas of reusability, ease of development, and availability of existing libraries.

To accomplish this we have rewritten most of our rules and developed several generic wrappers and reusable utilities to make this easier. This is the story of our approach to developing Python rules and the challenges we faced along the way.

iRODS UGM 2020, June 9-12, 2020, Virtual

Using JSON Schemas as metadata templates in iRODS

Venustiano Soancatl Aguilar
University of Groningen
Groningen, Netherlands
v.soancatl.aguilar@rug.nl

ABSTRACT

In this talk, we discuss the potential of JSON schemas as metadata templates. One of the main advantages of JSON schemas is that they can be represented as strings. This feature is very convenient as strings can be stored in iRODS as AVUs, in an elasticsearch database or in any other external database. Additionally, JSON schemas are supported by programming languages such as python and java. This support makes it relatively straightforward to validate both, the schemas and the JSON metadata against the schemas. Assuming that JSON schema templates are stored somewhere else and can be accessed from iRODS, we have implemented irules to associate metadata templates with iRODS objects, ingest metadata validated against templates, display template AVUs and inherited AVUs via the command line interface. Finally, we discuss future plans regarding managing templates in our iRODS system.

iRODS UGM 2020, June 9-12, 2020, Virtual

S3:TNG - iRODS S3 Resource Plugin with Direct Streaming

Justin James
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
jjames@renci.org

Kory Draughn
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
korydraughn@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 S3 storage resource plugin has become very important to the iRODS ecosystem. Many production systems are now spanning local disk, local or remote object stores, and tape. Last year's release of the cacheless S3 plugin enjoyed immediate uptake.

This year's update shares the design and engineering underway for the iRODS S3 plugin to provide direct streaming into and out of S3-compatible storage. This rewrite uses the new iRODS IOStreams library[1] and in-memory buffering to make efficient multi-part transfers.

Keywords

iRODS, S3, AWS, streaming, multipart, data management

INTRODUCTION

iRODS has provided an interface to S3-compatible[2] storage since iRODS 2.2[3] through the compound resource (with child resources designated in the roles of cache and archive). Last year, we introduced the work to provide a 'cacheless' connection to S3-compatible storage that did not require a compound resource as parent[4]. This was better for configuration (fewer moving parts), performance (no additional replica needed), as well as for cost (no additional replica needed).

This year's progress revisits some remaining assumptions and addresses performance through direct streaming.

PRIOR LIMITATIONS AND MOTIVATION

While the existing cacheless plugin did not require a compound resource with an archive and cache resource, it still used cache files at the OS level. Because the S3 multipart protocol is different from the parallel transport mechanism in iRODS, the S3 plugin still collected the entire file 'locally' from one protocol before sending it on its way using the other protocol. This year's work addresses this performance bottleneck, but not completely. Some scenarios still require a local cache file.

Performance was also limited by having to read an entire object from S3, write to the local disk, and flush the object back to S3. This required multiple trips to the local disk, both for read and write.

iRODS UGM 2020 June 9-12, 2020, Virtual

In some cluster cases, communicating with the S3 endpoint is faster than the communicating with the local disk which means the performance is further limited by the performance of the local disk.

One last limitation of the cacheless S3 plugin is that it does not support the `dstream` interface directly.

Figure 1. Ongoing evolution of the iRODS S3 Plugin

This paper shares how the iRODS S3 Resource Plugin is migrating to a streaming plugin that streams connections directly from iRODS to the S3 backend and vice versa with as few interactions with the local disk as possible.

IMPLEMENTATION

In the new streaming S3 plugin, all reads and writes are handled by an `s3_transport` class which extends `irods::experimental::io::transport`.

As the primary goal is to remove the use of any local cache file, here we will discuss some different usage scenarios.

For normal gets and puts, no cache file is used. When the `RESOURCE_OP_READ` operation is called, a read is called to the `dstream` object. When a `RESOURCE_OP_WRITE` operation is called, this data is streamed directly to S3 via the `dstream` object. During this write, if a parallel transfer is performed in iRODS, a multipart upload is started to S3 and each transfer thread streams data directly to S3 for its part. If a single buffer write is being performed, then multipart is not used and data is streamed sequentially to S3.

In some circumstances covered later, a local cache file will still be used.

The iRODS S3 plugin is no longer using the `S3FS`[5] libraries. Both the cacheless and cache versions use `libs3`[6] directly which allows for more fine-grained control of the underlying transfer mechanisms.

The `s3_transport` code in this plugin is a proving ground for two new libraries that have been recently added to the iRODS core. The first is a space-limited circular buffer[7] with notifications for threads waiting to read and write. The other is the use of `dstream` within the data movement layer in iRODS.

PARALLEL PUT

When an iRODS object is opened in write only mode with the truncate flag set, a full file upload (PUT) is being performed. This is the 'usual' case for when files are put into iRODS.

Each thread creates the `dstream` and `s3_transport` objects when it receives the first call to the `RESOURCE_OP_WRITE` operation (Figure 2). The very first `s3_transport` object that is opened calls the S3 function `CreateMultipartUpload`. The `RESOURCE_OP_WRITE` operation simply forwards to `dstream.write()` which calls `s3_transport.send()`. On the `s3_transport.send()`, the data is written to a per-thread, in-memory circular buffer. The `s3_transport` object creates a thread to read the data from the buffer and stream it to S3.

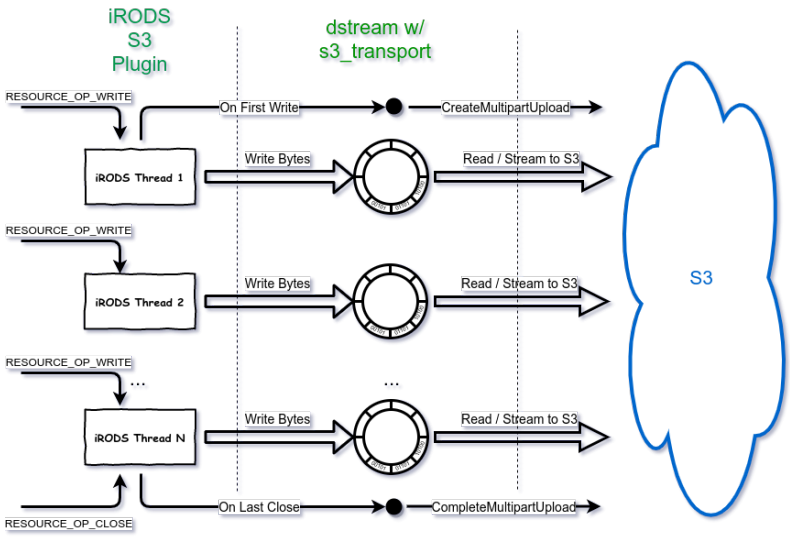

Figure 2. Streaming Parallel PUT

Circular Buffer

The blocking circular buffer is designed to improve performance and limit the amount of memory that is used when uploaded very large files. If the circular buffer is full, the `s3_transport.send()` waits until it can complete the write operation. The thread which reads from the circular buffer and streams to S3 will wait if the buffer is empty.

Each parallel transfer process/thread has its own circular buffer.

The size of the circular buffer is set in the resource context's `S3_CIRCULAR_BUFFER_SIZE` parameter. This size is in entries, not bytes. Each entry is equal to the size of the buffer sent to the plugin. The maximum number of bytes that may be used per iRODS Agent is `numberThreads * bufferSize * numberEntries`.

PARALLEL GET

When an object is opened in read-only mode (GET), the requested bytes are simply read from the S3 object. This is the 'usual' case for when files are retrieved from iRODS.

S3 allows random access reads for objects, so a call to `RESOURCE_OP_READ` translates directly to an S3 request to `GetObject` (Figure 3). Each thread creates the `dstream` and `s3_transport` objects when it receives the first call to the `RESOURCE_OP_READ` operation. The `RESOURCE_OP_READ` operation simply forwards to `dstream.read()` which calls `s3_transport.receive()`. The read operations are all synchronous.

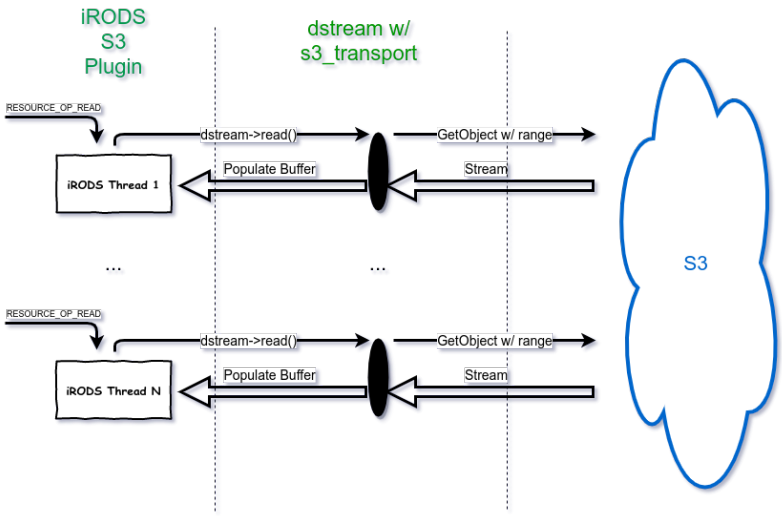

Figure 3. Streaming Parallel GET

CACHE FILE

In some cases, a local cache file will still be necessary. These include the following:

- The iRODS data object is opened in both read and write mode.

- The iRODS data object is opened in write-only mode but the object exists in S3 and is not being truncated.

When the `s3_transport` object is created, it detects any need for a cache file and the S3 object is downloaded to cache (if it exists and not truncated) (Figure 4). All iRODS reads and writes are performed directly on the cache file (Figure 5). When the last `close()` is performed on the iRODS data object, the cache file is flushed to S3 (Figure 6).

If the cache file is large enough, multiple upload threads are used and a multipart S3 upload is performed.

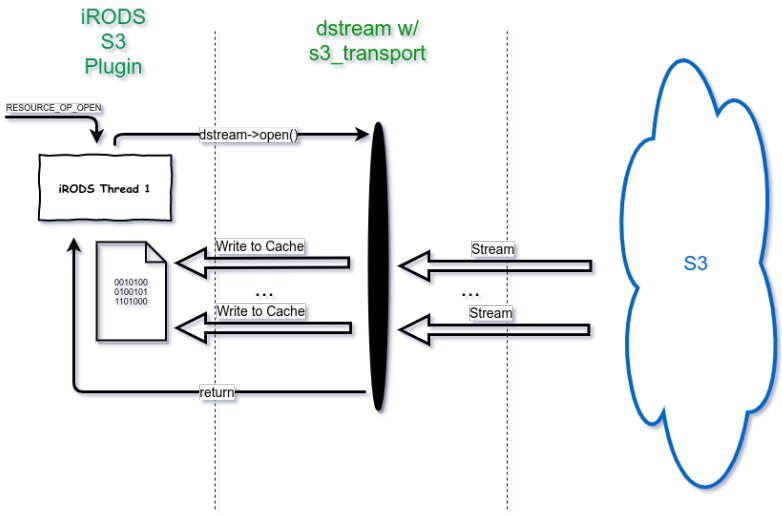

Figure 4. Streaming Cache Open

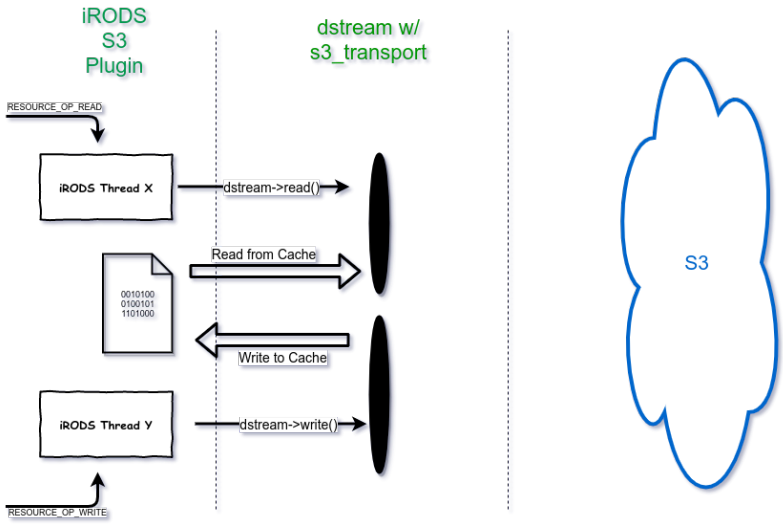

Figure 5. Streaming Cache Read / Write

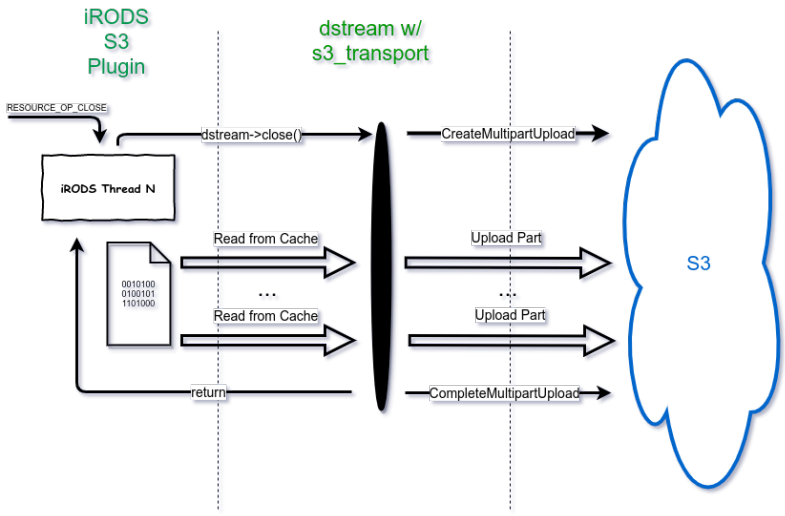

Figure 6. Streaming Cache Close

PERFORMANCE

We ran a battery of tests that compared the performance of uploads and downloads among the following three implementations:

- S3 Plugin w/ Streaming (2020)

- S3 Plugin w/ S3FS (Cacheless) (2019)

- Amazon AWS CLI Tool

Since iRODS generally uses 16 threads to transfer large files, the maximum number of threads for the S3 API was increased to 16 threads.

The tests were run against a local MinIO[8] server backed by an SSD drive. This was to simulate a case where the network throughput and storage latency is not a bottleneck so that we could compare the performance improvement by not using a cache file. The chunk size was set to 64MB.

Each upload and download was performed 6 times and the median time value was used to measure the performance.

Download

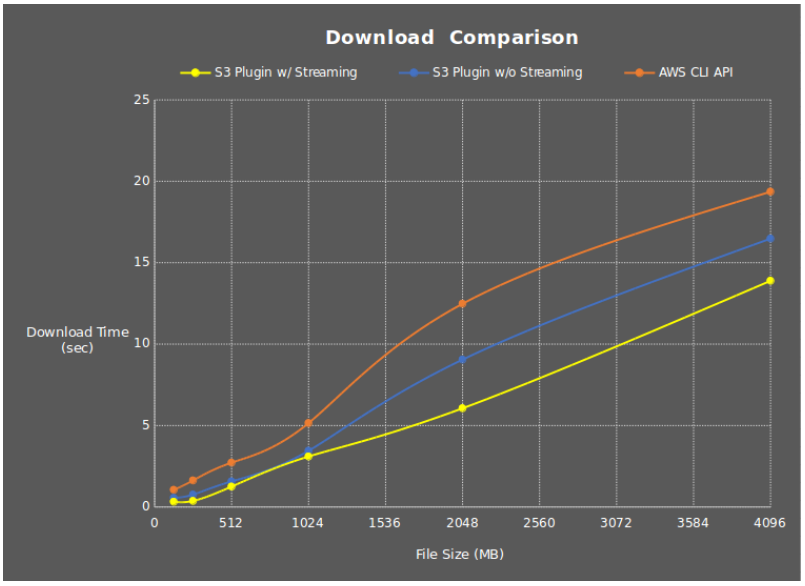

Figure 7. Download Performance Comparison

	128MB	256MB	512MB	1024MB	2048MB	4096MB
AWS S3 CLI	1.05	1.63	2.72	5.15	12.48	19.36
S3 Plugin w/ S3FS (2019)	0.56	0.76	1.57	3.45	9.05	16.48
S3 Plugin w/ Streaming (2020)	0.32	0.37	1.25	3.10	6.06	13.89

Table 1. Download times in seconds (median, n=6)

As seen in Figure 7 and Table 1, all three implementations were competitive with files under 1GB, but then the Streaming S3 Plugin (2020) began to outdistance the other two. With file downloads between 1GB and 4GB, the Streaming S3 Plugin (2020) consistently outperformed the S3FS implementation (2019) by about three seconds. The AWS CLI Tool was an additional couple seconds slower than S3FS.

It is also clear from this graph that download performance is relatively linear as the downloaded file grows in size.

Upload

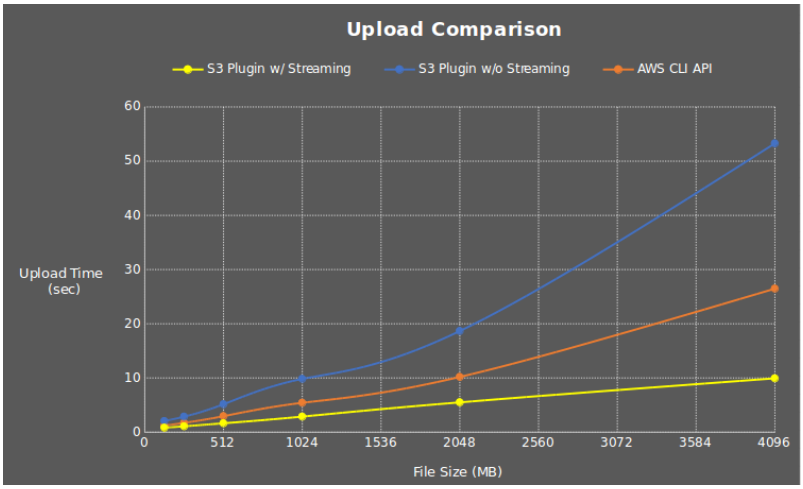

Figure 8. Upload Performance Comparison

	128MB	256MB	512MB	1024MB	2048MB	4096MB
S3 Plugin w/ S3FS (2019)	2.06	2.88	5.16	9.83	18.66	53.28
AWS S3 CLI	1.21	1.73	2.96	5.43	10.19	26.48
S3 Plugin w/ Streaming (2020)	0.83	1.09	1.65	2.88	5.51	9.93

Table 2. Upload times in seconds (median, n=6)

The Streaming S3 Plugin (2020) came out ahead for the upload comparison as well (Figure 8 and Table 2). However, the AWS CLI Tool came in second, with a trailing third for the S3FS implementation (2019) as the file sizes increased. For 4GB files, the Streaming S3 Plugin was 2.5x faster than the AWS CLI Tool.

The Streaming S3 Plugin, most notably, maintains a linear performance profile for upload due to the circular buffers providing consistent memory usage throughout and helping to mitigate the differences in storage medium performance by moving any bottleneck to the network.

These same tests were performed against an Amazon S3 backend, including live public network latencies. The results are not included here, but the relative performance among the three options remained.

FUTURE WORK

Development is almost complete but more testing and real-world usage will certainly bring new insights and optimizations.

With the current default iRODS settings, files between 32 MB and 80 MB are failing due to the S3 limitation on each multipart upload part (except the last) being at least 5 MB in size. Increasing the `transfer_buffer_size_for_parallel_transfer_in_megabytes` configuration setting is a manual workaround for this. This will be fixed and no configuration change will be necessary.

Additional work will also be required to implement the `RESOURCE_OP_READDIR` operation.

SUMMARY

The iRODS S3 Plugin has made a lot of progress in the last few years. It has moved from an archive class resource under a compound resource, to providing cacheless operations, to now providing direct streaming access to S3-compatible backends.

This progress has also increased its performance significantly. This year's streaming plugin is now almost 30% faster when downloading files than the AWS CLI Tool and 2.5x faster when uploading.

REFERENCES

[1] Draughn, Kory: iRODS IOStreams Library (2019). `https://github.com/irods/irods/issues/4268`

[2] Amazon S3 (2006) `https://en.wikipedia.org/wiki/Amazon_S3`

[3] Wan, Mike: Initial S3 File Driver commit (2009).
`https://github.com/irods/irods-legacy/commit/2d204c14687340828483abecf8f73a8ea4dea944`

[4] James, Justin; Russell, Terrell; Coposky, Jason; iRODS S3 Resource Plugin: Cacheless and Detached Mode (2019)
`https://irods.org/uploads/2019/James-iRODS-S3_Resource_Plugin_Cacheless_and_Detached-paper.pdf`

[5] s3fs-fuse: FUSE-based file system backed by Amazon S3 `https://github.com/s3fs-fuse/s3fs-fuse`

[6] libs3 `https://github.com/bji/libs3`

[7] Circular Buffer `https://en.wikipedia.org/wiki/Circular_buffer#References`

[8] MinIO: High Performance Object Storage `https://min.io/`

Parallel data migration between GPFS filesystems via the iRODS rule engine

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

ABSTRACT

At the PDC Center for High Performance Computing at KTH, in collaboration with our colleagues at the supercomputing center of Linköping University, we operate the iRODS-based section of the national research data storage for Sweden. We have a heterogeneous, asymmetric data grid based on iRODS with several underlying storage solutions and technologies. At PDC we have the performance tier of the system running on top of GPFS filesystems. Our GPFS cluster alongside the filesystems it is hosting, are due for an upgrade - since we would like to deploy the newest generation of GPFS (5.0.x) for its space efficiency and several other enhancements. For this we prepared by initially building the cluster with two physical filesystems to accommodate the envisioned upgrade and (online) data migrations. After the cluster has been upgraded to the latest software we will upgrade (reformat) the on-disk GPFS filesystems one at a time, migrating the data online between the filesystems via iRODS. All data will remain accessible at all times and users uninterrupted, not realizing their data objects have been migrated underneath. At this moment the first step of this operation has been done, i.e. the other physical filesystem has been drained of iRODS resources and those have been migrated to its pair. This was done via the asynchronous and parallel rule execution of iRODS 4.2.x with a set of custom rules developed with the iRODS Consortium. This enabled us to gain more parallelism of the system, not only with the parallel read/write performance of GPFS with iRODS parallel streams but also the checksumming of the migrated data objects in parallel, jobs being launched from the iRODS delay execution queue. The successes and challenges of this process are to be presented.

iRODS UGM 2020, June 9-12, 2020, Virtual

Policy-Encapsulated Objects

Arcot Rajasekar
University of North Carolina at Chapel Hill
Chapel Hill, NC, USA
rajasekar@unc.edu

ABSTRACT

With increasing movement of data objects across distributed and remote storage data objects lose their policies that were applied and used where they were generated, created or administered. When a file is moved from a local storage to a remote storage, all such metadata including metadata about ownership, creation, modification audit trail and security and permission information are not transferred. Moreover, all links to the original is lost and there is no lineage captured or maintained and as the data object gets copied and moved across multiple storages and modified at various stages the information about all these actions are not captured and is lost forever. Even policy-based data management systems, such as iRODS, that instrument policy enforcement points within the data management infrastructure and apply policies as computer actionable rules but do not control once the object is copied out of its domain. We propose the concept of a policy encapsulated object (PEO) that encodes policies that govern the life-cycle of a data object as part of the data payload and serve as a gatekeeper for the data. Also, to make the system self-contained we propose inclusion of an execution infrastructure (similar to the iRODS rule engine) which will run on top of any operating system and capture all lineage and administrative policies.

By including policies to verify the trustworthiness of the execution infrastructure within the PEO, a trusted environment can be implemented. Each PEO can verify that it is in a trusted environment while controlling manipulation of the associated data set. By providing mechanisms for a data object to be aware of its environment, PEOs enable controlled operations including redaction, integrity checking, derived product generation, data caching, and access control. A PEO can characterize all provenance information needed to instantiate a derived data product, including governing policies, and the required trusted environment. There is a strong link between trusted environments and containers used for reproducible computing. We discuss various issues related to policy encapsulated objects.

iRODS UGM 2020, June 9-12, 2020, Virtual

Nils Haustein
IBM European Storage Competence Center
Am Weiher 24, 65451 Kelsterbach
nils_haustein@de.ibm.com

Mauro Tridici
Euro-Mediteranean Center on Climate Change
via Augusto Imperatore 16, 73100 Lecce
mauro.tridici@cmcc.it

Integration of iRODS with IBM Spectrum Archive™ Enterprise Edition

A flexible tiered storage archiving solution

iRODS UGM 2020, June 9-12, 2020, Tucson, Arizona, USA

INTRODUCTION

A tiered storage system provides lower total cost of ownership for large volumes of data by storing data on the most appropriate storage tier (flash, disk and tape). Independent studies have shown that total cost of ownership of tape solution provides an expected TCO that is more than 80% lower than that of the all-disk solution [1].

While tape storage is suitable for storing large volumes of data over long periods of time at lower cost, access time to data on tape is significantly higher than to data on disk. Providing data from tiered storage file systems with tape in multi-user environment bears several challenges. These challenges and solutions are further elaborated in this blog article [2].

In summary, tiered storage file systems with tape storage are a blessing and a curse. The blessing is that the user can see all files regardless if these are stored on disk or tape. Cursing starts when the when the user opens a file that is stored on tape because the recall takes one or more minutes. Unfortunately, the user is not aware that the file is on tape because standard file systems do not indicate whether the file is on disk or on tape. It gets even worse if the many users simultaneously open several files that are on tapes. This causes even longer waiting times because transparent recalls are not tape optimized.

To address these challenges, the user must be able to determine the location of files and request files from tapes to be recalls. These recall requests coming from multiple users can be queued and recalled periodically in a tape optimized manner whereby the files are sorted by the tape-ID and the location on tape. The combination of iRODS with IBM Spectrum Archive Enterprise Edition can accommodate this.

In this paper Mauro Tridici from the Euro-Mediterranean Center on Climate Change (CMCC) and Nils Haustein from the IBM European Storage Competence Center give a brief introduction to iRODS and explain examples for integrating iRODS with IBM Spectrum Archive and its advantages. For more information refer to our whitepaper [3].

IRODS

iRODS software is a data management layer - maintained by the iRODS consortium - that sits above the storage that contain data, and below domain-specific applications [4]. The data virtualization capabilities of iRODS make it a one-stop shop for all data regardless of the heterogeneity of storage devices. Whether data is stored on a local hard drive, on remote file systems or object storage, iRODS' virtualization layer presents data resources in the classic files and folders format, within a single namespace.

iRODS is open-source, data management middleware that enables users to:

- Access, manage, and share data across any type or number of storage systems through iRODS APIs (iCommands, REST, WebDAV, Python, C++, Java)
- Automate workflows through powerful rules and microservices
- Search and find data through descriptive metadata and query tools

iRODS rules are executed based on conditions or, in iRODS terminology, Policy Enforcement Points (PEPs). iRODS can be integrated with different kind of storage system providing storage space for the archived data. In the next section we describe a solution that integrates iRODS with a tiered storage file system based on IBM Spectrum Scale and IBM Spectrum Archive.

TIERED STORAGE FILE SYSTEM

IBM Spectrum Scale™ [5] is a software-defined scalable parallel file system providing tiered storage capabilities. IBM Spectrum Archive Enterprise Edition [6] provides and manages the tape tier within an IBM Spectrum Scale file system. The IBM Spectrum Scale file system can be accessed via standardized protocols such as POSIX, NFS, SMB, HDFS and Object.

As shown in Figure 1, the combination of IBM Spectrum Scale with IBM Spectrum Archive provides a tiered storage file system with different storage media including Flash and SSD, disk and tape. While Flash and disk storage are managed by IBM Spectrum Scale directly, the tape storage is managed by IBM Spectrum Archive. IBM Spectrum Scale integrates a policy engine that allows to place the files on a storage tier upon file create and migrate the files to other storage tiers over the data lifecycle. Policies are defined and tested once and can then be configured to run automatically in the background. For example, a policy can places all new files on the disk storage tier of the IBM Spectrum Scale file system and if files have not been accessed for 30 days then migrate these files to tape storage.

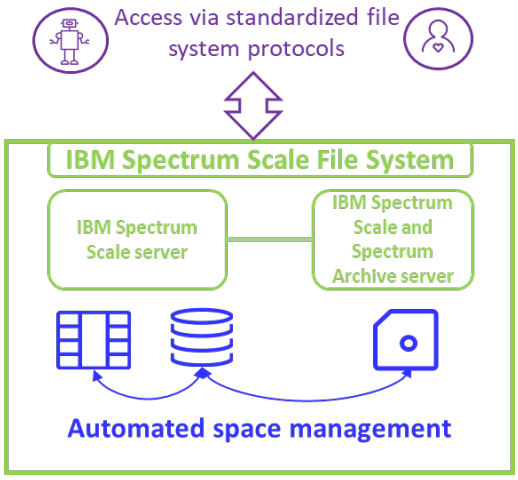

Figure 1: Combination of IBM Spectrum Scale with IBM Spectrum Archive tape tier

SOLUTION INTEGRATING IRODS WITH IBM SPECTRUM ARCHIVE

This solution integration iRODS and IBM Spectrum Archive is shown in Figure 2 and is comprised of three servers that are interconnected. One server represents the IBM Spectrum Scale cluster containing a tiered storage file system which is placed on disk and tape. The tape tier is managed by IBM Spectrum Archive. This file system is exported via NFS to the iRODS server.

The iRODS server hosts the iRODS Metadata Catalog (iCAT) database. The iCAT is a relational database that holds all the information about data, users, and zone that the iRODS servers need to facilitate the management and sharing of data.

The iRODS client can host an application that interacts with the iRODS server through the available API. In this example the iRODS client command line (iCommand) is used to archive, describe, search and retrieve data.

iRODS server, client and the NFS mounted tiered storage file system represent an iRODS zone.

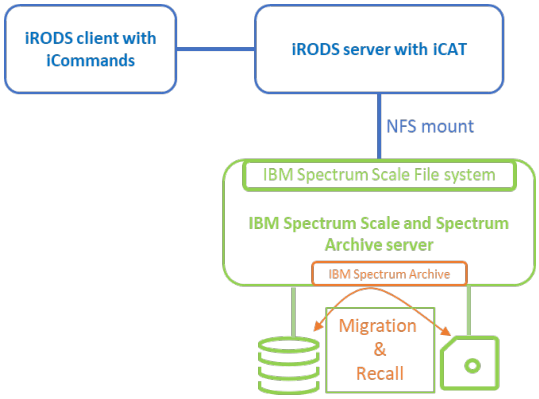

Figure 2: Solution architecture of iRODS with IBM Spectrum Archive

This solution can be configured to provide value adding functions, including:

- Prevent transparent recalls of files that are on tape and add files that are requested by the iRODS user to a queue and recall them in a tape optimized manner
- Determine file migration state from an iRODS user perspective
- Set storage quota for users that apply to all files regardless if stored on disk or on tape in the underlying
- Extracting and ingesting file metadata to the iRODS metadata catalog (iCAT) automatically after a file has been stored

Subsequently we briefly explain these functions. The actual implementation can be found at the GitHub repository [7].

Prevent transparent recall

To prevent transparent recalls, we can leverage a new iRODS rule along with a new custom microservice. The iRODS rule, runs on the iRODS server, intercepts an open request for a file using the system defined PEP rule `acPreprocForDataObjOpen` and invokes the new custom microservice along with the path and file name of the file to be opened. The new microservice determines if the file is migrated. If the file is not migrated, then the microservice returns "1" to the rule. Otherwise, if the file is migrated, then the microservice returns "0" to the rule and adds the path and filename queue. The queue can be a file list that resides on the IBM Spectrum Archive server. If the microservice returned "0" then the rule fails the file open request and informs the user that the file is still on tape.

Here is an example of a file open request for a migrated file:

```
$ iget -f file1
file /archive/home/mia/col1/file1 is still on tape, but queued to be staged.
```

To recall the files that have been added to the queue, a recall-program must be implemented that recalls these files using the tape optimized recall functions. This recall-program can be scheduled to run periodically on the IBM Spectrum Archive server, if the queue for the files to be recalled is a file list that is accessible by the Spectrum Archive server.

The time interval of the recall-program execution defines the maximum time the user must wait before he can access a file that was migrated to tape. To provide the user the capability to display the file status, we created another example which is explained next.

Display file status

To display the migration state of a file stored in an iRODS zone we created a new command for the iRODS user: `ifilestate`. This new command invokes a new iRODS rule that invokes a new microservice that checks the state of a file using the UNIX command: stat. Depending on the result of this check done by the new microservice the rule program returns the appropriate message to the user. Find below an example output of the new command:

```
$ ifilestate /archive/home/mia/col1/file1
Level 0: file /archive/home/mia/col1/file1 is MIGRATED

$ ifilestate /archive/home/mia/col1/file0
Level 0: file /archive/home/mia/col1/file0 is NOT migrated
```

Set quota for the entire file space

To set and enable quota for a given user using a given iRODS storage resource we did the following:

Enable quota by editing the file /etc/core.re and adding the following line:

```
acRescQuotaPolicy {msiSetRescQuotaPolicy("on"); }
```

Set quota limit of 2 GB for user1 on the iRODS storage resource that represents the tiered storage file system provided by IBM Spectrum Scale. In this example we have one iRODS storage resource in the zone that is named “buffer”. Because we only have one storage resource the total quota limit is identical to the quota limit of the storage resource buffer:

```
$ iadmin suq user1 buffer 2147483648
$ iadmin suq user1 total 2147483648
```

To calculate the current storage consumption on a periodic basis we created a delayed iRODS rule that invokes the integrated microservice msiQuota and loaded this into the rule engine using the following command:

```
$ irule -F /etc/irods/quota.r -r irods_rule_engine_plugin-irods_rule_language-instance
```

Now if the user tries to store more than 2 GB on the storage resource he gets a quota exceeded error:

```
$ iput bigfile2
/archive/home/user1/col1/bigfile2, status = -110000 status = -110000
SYS_RESC_QUOTA_EXCEEDED
```

Extracting and ingesting metadata

The last project we implemented (essentially based on Daniel Moore NetCDF header extraction microservice [10]) extracts metadata from ingested files and add this into the iRODS catalog to make it available for subsequent searches.

For the implementation we again used a custom iRODS rules and microservice. We created a new iRODS rule that is invoked after a file has been stored in the iRODS zone, for example by using the iput command. This rule implements the integrated iRODS PEP acPostProcForPut and invokes a new microservice. The new microservice harvests the information from the file and return this to the iRODS rule which adds it to the file metadata.

To make it simpler in this paper, imagine the microservice determines the type of the file using the UNIX command: file and returns this as string to the iRODS rule. The iRODS rule adds the value of the file type string to the attribute Filetype to the file metadata. After ingesting files to iRODS using the iput command, the file will automatically obtain the file type as metadata as shown below:

```
$ iput document.pdf file1

$ imeta ls -d file1
AVUs defined for dataObj file1:
attribute: Filetype
value:  PDF document
units:
```

It is also possible to search in iRODS for all files based on their type using the command: imeta:

```
$ imeta qu -d Filetype like %PDF%
collection: /archive/home/mia/col1
dataObj: file1
collection: /archive/home/mia/col1
dataObj: file2
```

As shown above the search found two files.

This is a simple example. There are many iRODS projects that leverage this mechanism to extract file header information from JPEG-files [8] or NETCDF-files [9] and many other file types.

REFERENCES

[1] Disk and Tape TCO study by ESG:
https://www.lto.org/wp-content/uploads/2018/08/ESG-Economic-Validation-Summary.pdf

[2] Blog article challenges and solutions with tiered storage file systems
https://community.ibm.com/community/user/imwuc/blogs/nils-haustein1/2020/02/21/irods-with-ibm-spectrum-archive

[3] IBM Whitepaper "Integration of iRODS with IBM Spectrum Archive Enterprise Edition"
http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102815

[4] iRODS: https://irods.org/

[5] Spectrum Scale: https://en.wikipedia.org/wiki/IBM_Spectrum_Scale

[6] Spectrum Archive: https://developer.ibm.com/storage/products/ibm-spectrum-archive/

[7] Github repository for this project: https://github.com/nhaustein/irods-tieredStorage-tape

[8] JPEG file: https://en.wikipedia.org/wiki/JPEG

[9] NETCDF data format: https://en.wikipedia.org/wiki/NetCDF

[10] Project for extracting NETCDF metadata by Daniel Moore: https://github.com/d-w-moore/extract_netcdf_header_msvc

DISCLAIMER

The Foundation Euro-Mediterranean Centre on Climate Change has its registered office and administration in Lecce and other units in Bologna, Venice, Capua, Sassari, Viterbo and Milan. The CMCC Foundation doesn't pursue profitable ends and aims to realize and manage the Centre, its promotion, and research coordination and different scientific and applied activities in the field of climate change study.

The information contained in this documentation is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information provided, it is provided "as is" without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this documentation or any other documentation. Nothing contained in this documentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM (or its suppliers or licensors), or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

The following terms are registered trademarks of International Business Machines Corporation in the United States and/or other countries: IBM Spectrum Scale, IBM Spectrum Archive

LINUX is a registered trademark of Linus Torvalds.

SmartFarm Data Management

Kieran Murphy
Agriculture Victoria
Victoria, Australia
kieran.murphy@ecodev.vic.gov.au

ABSTRACT

Agriculture Victoria's research group is geographically disperse, with research data from research 'SmartFarms' requiring many manual steps. Data management challenges increase with large datasets generated with new sensing technologies. This requires the development of standardised, automated, on line, authenticated and verifiable standard processes for uploading data for storage and analytics on computing facilities.

Working with iRODS, Agriculture Victoria are piloting new data management workflows of 'SmartFarm' data, and this talk will discuss lessons from small, medium and high data Agriculture SmartFarm use cases using edge computing and collaborative data infrastructure and the flow on development of capability for AVR researchers.

iRODS UGM 2020, June 9-12, 2020, Virtual

Data management in autonomous driving projects

Marcin Stolarek
Aptiv
Kraków, Poland
marcin.stolarek@aptiv.com

Radosław Rowicki
Aptiv
Kraków, Poland
radoslaw.rowicki@aptiv.com

Kacper Abramczyk
Aptiv
Kraków, Poland
kacper.abramczyk@aptiv.com

Mateusz Rejkowicz
Aptiv
Kraków, Poland
mateusz.rejkowicz@aptiv.com

ABSTRACT

Aptiv deployed iRODS in production around 1.5 year ago, together with the start of the development phase of one of the big projects on autonomous driving. The tool was selected after a few POC installations. The major advantage of iRODS we recognized at the time was a number of side projects and plugins available.

In such industrial projects, it's quite common to have multiple partners working on different parts of a workflow. Tracking data status - migrating them between partners and within engineering groups responsible for data collection, manual and automatical analysis is a fairly complicated task.

From a technical perspective, our deployment is based on two DNS round-robin groups of iRODS resource servers, both groups are using Lustre filesystem as a storage backend. During testing, we reached ~90Gbps, which was our estimated data collection rate. Besides HPC filesystems resource daemons are also configured to use AWS S3 buckets connected over AWS direct connect(30Gbps).

I'll explain our configuration with a DNS round-robin trick. Share our current struggles related to automatic registration of files from Lustre filesystem, audit rule engine. Difficulties we had in early stages(adoption) and current issues.

iRODS UGM 2020, June 9-12, 2020, Virtual

CyVerse Discovery Environment: Extensible Data Science workbench and data-centric collaboration platform powered by iRODS

Sarah Roberts
CyVerse / University of Arizona
Tucson, Arizona, USA
sarahr@cyverse.org

Sriram Srinivasan
CyVerse / University of Arizona
Tucson, Arizona, USA
sriram@cyverse.org

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

Tina Lee
CyVerse / University of Arizona
Tucson, Arizona, USA
tinal@cyverse.org

ABSTRACT

The Discovery Environment, a web-based Data Science workbench that supports data management, analysis and collaboration tasks for diverse communities of users from astronomers to zoologists, is actively utilized by thousands of scientists world-wide. In this presentation we highlight how we have leveraged iRODS alongside other frameworks like Kubernetes, NodeJS, React, and Asynchronous Tasks to meet researchers' growing demands for reproducible, extensible, collaborative and scalable analysis environments. We also provide an overview of the Terrain API which provides developers with programmatic access to extend and adopt the Discovery Environment's underlying cyberinfrastructure. Finally, we touch upon our Visual and Interactive Computing Environment (VICE), our newest service that allows researchers to use Jupyter Notebooks, RStudio, Rshiny and other custom web-based, interactive data analysis and visualization tools. VICE provides secure out-of-the-box, single sign-on access to all container (Docker)-based applications and can manage CPU- and GPU-based analysis with configurable resource allocation per task.

iRODS UGM 2020, June 9-12, 2020, Virtual

iRODS and Federated Identity authentication: current limitations and perspective

Claudio Cacciari
claudio.cacciari@surfsara.nl

Stefan Wolfsheimer
stefan.wolfsheimer@surfsara.nl

Hylke Koers
hylke.koers@surfsara.nl

Arthur Newton
arthur.newton@surfsara.nl

Tasneem Rahaman-Khan
tasneem.rahaman-khan@surfsara.nl

Matthew Saum
matthews@surfsara.nl

Gerben Venekamp
gerben.venekamp@surfsara.nl

SURF Netherlands

ABSTRACT

iRODS does not support natively authentication protocols for federated identity management, such as SAML or OpenID Connect (OIDC). Additional security measures, like two factor authentication (2FA), are neither supported. There are some third-party plugins or modules that support a limited sub-set of those features, but a comprehensive and flexible solution is missing. In this presentation we would like to outline use cases and explain the limits of the current implementation. Consider a web application against which a user authenticates using OIDC. The application is connected to iRODS to upload data on behalf of the user. We want the interaction between iRODS and the web application to be transparent for the user. The existing plugin (auth_plugin_openid) is not suitable because it requires an explicit authentication from the user. We could make the web application pass the OAuth2 access token to iRODS and validate it through a Pluggable Authentication Module (PAM) extension acting as an OIDC client. Since the token expires after a while, it would need to be refreshed on the iRODS side using an refresh token. The current implementation does not support this workflow, especially dealing with two tokens.

A comprehensive solution would be able to overcome those and other limitations. At the same time, it would simplify the life of the users and of the administrators. For example, when an iRODS instance supports multiple authentication protocols and the client is a single entry point shared among multiple users, like a WebDAV endpoint based on Davrods, the administrator is forced to expose a different endpoint for each authentication protocol because the protocol is defined client-side. Enabling the server to support a fall-through mechanism, would allow the client to just pass the credentials without the need to pick one of the protocols in advance.

SURF has started to develop a proof of concept that aims to achieve that solution extending the current iRODS PAM support so that it can deal with an arbitrary exchange of tokens and challenges and delegating the implementation of the specific federated identity protocols to dedicated PAM modules. In parallel the iRODS consortium promoted the design of a more general implementation through the discussion in the Authentication Working Group. The group has adopted the idea of supporting a flexible conversation between client and server, but rather than implementing it on the PAM side, it decided to extend the iRODS API to support different authentication methods through plugins.

This presentation describes the main scenarios related to the support of federated identity management in iRODS and the possible solutions.

iRODS UGM 2020, June 9-12, 2020, Virtual

The Past, Present and Future of iRODS at the Texas Advanced Computing Center

Chris Jordan
The University of Texas at Austin
Austin, Texas, USA
ctjordan@tacc.utexas.edu

ABSTRACT

The Texas Advanced Computing Center has operated iRODS services for over 10 years, both for shared support of general purpose research data management, and as a dedicated service supporting specialized cyberinfrastructure projects. We will provide a brief history of iRODS at TACC, and give an overview of the current uses of iRODS and iRODS-based cyberinfrastructure at TACC. Projects utilizing iRODS at TACC have data collections ranging from a few terabytes to a few petabytes, and span the gamut from CT scanning through genome sequencing and archival of digital artworks; we will briefly discuss how TACC utilizes iRODS to support this wide variety of use cases, and how we plan to deploy iRODS in the future to support the continued growth of research data in both size and complexity.

iRODS UGM 2020, June 9-12, 2020, Virtual

iRODS_CSharp

Reink Fidder
Utrecht University
Utrecht, Netherlands
rienkfidder@gmail.com

Jelle Teeuwissen
Utrecht University
Utrecht, Netherlands
j.teeuwissen@students.uu.nl

Best Student Technology Award Winner

ABSTRACT

We are two computer science students at the Utrecht University. We are currently in our second year of our bachelor's degree.

We are both partaking in the honours programme, which is also the reason we created this client library.

As part of our honours requirements, we worked as part of the Care2Report (C2R) research team (https://sites.google.com/view/care2report). This research is aimed at creating a program which can transcribe and summarize medical consultation, so that doctors don't have to spend a lot of time writing consultation reports and can spend more time actually consulting.

Utrecht University uses a system called YODA for cloud storage, which is a portal that uses iRODS as a backend. For our assignment, we needed to create a way to upload logs from the C2R system to YODA and since the program is written mainly in C#, we decided to create a client that could be used to establish a connection to the YODA backend and transfer files with.

Since many researchers at Utrecht University use C# for their programming, we figured this would be a problem that would be encountered more often, so we thought it was a good idea to create a solution that wasn't just a way to solve our problem, but could also be used by others. And so, we started building a general iRODS C# client library.

The finished product is a client library which performs all the basic tasks that an iRODS client library should be able to perform, such as collection operations (create/remove/rename), data object operations (create/download/upload), metadata operations and a variety of queries.

The repository can be found at https://github.com/UtrechtUniversity/irods-Csharp

iRODS UGM 2020, June 9-12, 2020, Virtual

As for the impact on individuals, society, science and systems & technology; all areas are affected in roughly the same way. Anybody who wishes to get access to iRODS from their C# code will no longer need to create a way to use some other client library, but can use the native C# client library. This decreases the amount of work needed and increases the performance and ease of use.

In conclusion, the client library we have created can be viewed simply as a tool to make iRODS more accessible. Even though the main motivation for creating it was our own project, we hope there will be others that can use the functionalities we have created, or perhaps even improve on it.

Using iRODS to build a research data management service in Flanders

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

ABSTRACT

This presentation will discuss how iRODS is being used by the Flemish Supercomputing Centre (VSC) to implement a new research data management service highly coupled with the VSC High Performance Computing infrastructure. The current status of the project as well as the future plans will be presented.

The Tier-1 supercomputing infrastructure in Flanders has until 2018 mainly been targeted at users with serious calculation issues (typical HPC/HTC workloads). Although this platform in its current form is already very successful, the current focus on compute no longer meets all the needs of many researchers. More and more users have computational work that makes intensive use of large data sets. Migrating this data to and from the compute infrastructure whenever it is to be used for a calculation is very inefficient because of the scale.

Therefore, VSC decided on 2018 to start a new service focused on research data management. The new Tier-1 Data service aims to provide a service to allow users to store research data during the active phase of the research data life cycle (that is, data that is being collected and analysed) and has not yet being published. This service is restricted to data of research projects that are using the VSC Tier-1 Compute infrastructure.

This Tier-1 Data service is based on iRODS and has as primary goal to offer the users a platform to easy manage research data and help them to apply the FAIR principles to their research data from the very beginning of their projects. This should make it easier to transfer their research data at the end of the project to institutional or domain specific repositories for publication and preservation and when applicable ensure they are made publicly available (open access). This platform should also help the researchers to run their scientific workflows more efficiently by providing tools to automate data collection, data quality control and stage data from and to the Tier-1 Compute system.

The platform has recently started its pilot phase. During this phase a reduced number of research groups will be invited to build their research workflows using the new data service. The pilot projects selected are from different scientific domains (Climate Change studies, Humanities and Arts, Biological research, Life science, Plasma Astrophysics, …), have a strong collaborative nature between research groups of several Flemish universities and the usage of the new data service should facilitate the way they create, manage, share and reuse research data.

iRODS UGM 2020, June 9-12, 2020, Virtual

Application of iRODS to NIEHS Data Management

Mike Conway
NIEHS / NIH
Durham, NC, USA
mike.conway@nih.gov

Deep Patel
NIEHS / NIH
Durham, NC, USA
deep.patel@nih.gov

ABSTRACT

This will be a survey of current NIEHS data management strategy, in two parts. First will be an overview of data management challenges at NIEHS and the context in which we are employing iRODS, including developments in data governance policy, data sharing policy, knowledge management, LIMS (Laboratory Information Management Systems), standard workflow languages and pipelines, and cloud migration.

The second part will be a review of technology developments, including collaborative development of Metadata Templates, work on web interfaces, standard pluggable search integration, indexing, and developments in the GA4GH Cloud Work Stream.

It is anticipated that several releases of various code libraries will also be announced.

iRODS UGM 2020, June 9-12, 2020, Virtual

iRODS Client: NFSRODS 1.0

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

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

Alek Mieczkowski
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
info@irods.org

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

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

ABSTRACT

An update from last year's preview, this v1.0 release of NFSRODS[1] now provides multi-user support for NFSv4 ACLs by handling calls from `nfs4_setfacl` and `nfs4_getfacl`. It also supports sssd for easier AD/LDAP integration and secure connections to iRODS via SSL. NFSRODS v1.0 can provide a direct NFSv4.1[2] mount point to iRODS[3] users in enterprise environments.

Keywords

iRODS, client, NFS, NFSv4, data management

INTRODUCTION

Since 2019's initial implementation[4], the iRODS Consortium has worked to complete the implementation of an NFSv4.1 server that provides a complete lossless bidirectional permission mapping of multiple owners of collections and data objects in iRODS to NFSv4 ACLs.

Along the way, as interest and usage increased in the community, additional features were requested and implemented. Easy Active Directory (AD) and LDAP support is included via support for sssd.

NFSRODS v1.0 is nearly feature complete. With this release, the roadmap for NFSRODS includes adding support for parallel file transfer and hard links.

ARCHITECTURE

The core of NFSRODS has two components. The NFS server side of NFSRODS is provided by NFS4J[5] and is largely lifted directly from the open source project. NFS4J has a plugin architecture for its VirtualFileSystem and allows for other technologies to provide the filesystem interface. The iRODS Jargon client library[6] is used to implement an iRODS VirtualFileSystem. Jargon implements the iRODS protocol and communicates as an iRODS client.

The security model of NFSRODS deployment makes a few assumptions about its environment. First, the usernames and UIDs must be consistent from the mountpoint, to the NFSRODS server, to within the iRODS catalog. The NFS connection between the mountpoint and NFSRODS communicates which user is requesting access by unix UID. If `alice` (UID 509) makes a request to `ls` within the mountpoint, the NFSRODS server sees a request from UID 509 only. The NFSRODS server must be able to map the incoming UID to an iRODS username. This is done by the OS and uses the standard `/etc/passwd` file. Therefore, these must be kept in sync across the different machines in the

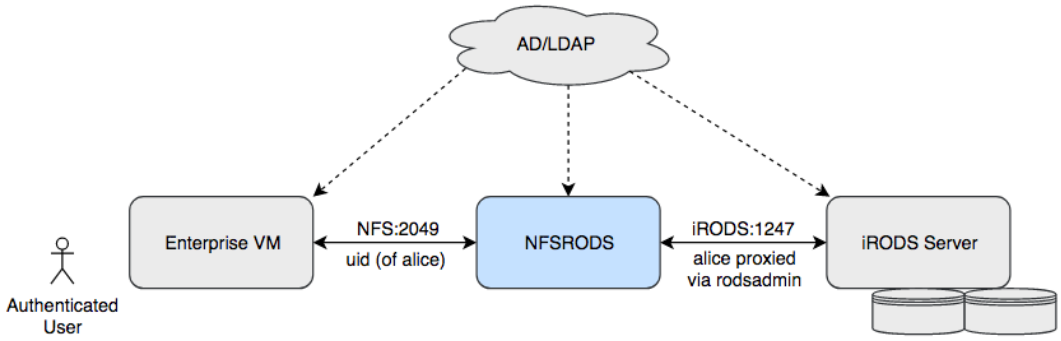

Figure 1. NFSRODS assumes an authenticated user without sudo access within the Enterprise VM.

system. It is assumed that this will be handled via external systems, most usually LDAP. The NFSRODS server maps the incoming request to an iRODS request which uses the matching username. It is assumed that the mechanism keeping the UIDs and usernames consistent is also keeping the list of users within the iRODS catalog consistent.

With this model, it is very important to note that any user with `sudo` rights on the Enterprise VM can become any other user, and therefore gain access to iRODS as that other user. It is recommended that there be no `sudo` rights available on the Enterprise VM where the mountpoint is accessible to the end user.

PERMISSIONS

In iRODS, multiple users and groups can be given different permissions on a collection or data object. Unix does not provide this capability and therefore, iRODS permissions cannot be mapped into traditional Unix permissions[7] without losing information. To get around this, NFSRODS uses NFSv4 ACLs.

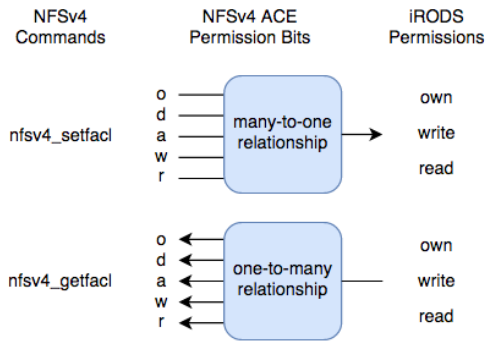

Figure 2. Bidirectional mapping of NFSv4 Commands and iRODS Permissions.

NFSv4 ACLs provide more than enough control for reflecting iRODS permissions in Unix. To manage permissions through NFSRODS, you'll need to install the package that contains `nfs4_getfacl` and `nfs4_setfacl`. On Ubuntu 16.04, that package would be `nfs4-acl-tools`. With these commands, you can view and modify all permissions in iRODS.

The order of Access Control Entries (ACEs) within an ACL does not matter in NFSRODS. When NFSRODS has

to decide whether a user is allowed to execute an operation, it takes the highest level of permission for that user (including groups the user is a member of).

Using nfs4_setfacl

When using `nfs4_setfacl`, it is important to remember the following:

- Domain names within the user and group name field are ignored.
- Special ACE user/group names (e.g. OWNER, GROUP, EVERYONE, etc.) are not supported.
- Unsupported permission bits are ignored.
- The highest permission level provided is what NFSRODS will set as the permission.

Below is the permissions translation table used by NFSRODS when `nfs4_setfacl` is invoked. The list is in descending order of iRODS permissions.

NFSv4 ACE Permission Bit	NFSv4 ACE Permission Bit Name	iRODS Permission
o	ACE4_WRITE_OWNER	own
a	ACE4_APPEND_DATA	write
w	ACE4_WRITE_DATA	write
r	ACE4_READ_DATA	read

Table 1. NFSRODS v1.0 Mapping of NFSv4 ACE Permission Bits to iRODS Permissions

A simple example is as follows:

```
$ nfs4_setfacl -a A::john@:ro foo.txt
```

NFSRODS will see that the `ACE4_READ_DATA` and `ACE4_WRITE_OWNER` bits are set. It then maps these to appropriate iRODS permissions and takes the max of those. NFSRODS will then set `john`'s permission on `foo.txt` to `own`.

Using nfs4_getfacl

Using this command is much simpler. When invoked, it returns the list of iRODS permissions on an object as an ACL. The mapping used for translation is shown below.

iRODS Permission	NFSv4 ACE Permission Bits
own	rwado
write	rwa
read	r

Table 2. NFSRODS v1.0 Mapping of iRODS Permissions to NFSv4 ACE Permission Bits

nfs4_setfacl Whitelist

NFSRODS offers a whitelist for granting `nfs4_setfacl` permission to particular users.

If a user is in the whitelist or in a group in the whitelist, they can run `nfs4_setfacl` on the specified logical path or any collection or data object below it, regardless of their iRODS permissions on that collection or data object.

A rodsadmin can add a user to the whitelist by adding a specific iRODS AVU (metadata) on the user.

```
$ imeta add -u <username> irods::nfsrods::grant_nfs4_setfacl <logical_path_prefix>
```

The following example demonstrates adding `alice#tempZone` to the whitelist with a prefix of `/tempZone/project_a/lab/notes`:

```
$ imeta add -u alice irods::nfsrods::grant_nfs4_setfacl /tempZone/project_a/lab/notes
$ imeta ls -u alice
AVUs defined for user alice#tempZone:
attribute: irods::nfsrods::grant_nfs4_setfacl
value: /tempZone/project_a/lab/notes
units:
```

A user can set permissions via `nfs4_setfacl` on a collection or data object if any of the following are true:

1. The user is an iRODS administrator (i.e. rodsadmin).
2. The user has own permission on the collection or data object.
3. The user is a member of a group that has own permission on the collection or data object.
4. The user is in the whitelist with a prefix that covers the collection or data object.
5. The user is a member of a group in the whitelist with a prefix that covers the collection or data object.

USAGE

Deployment of NFSRODS v1.0 requires some preparation and then three steps.

The preparation includes making sure that the necessary user UIDs and usernames are available for the different components (Enterprise VM, NFSRODS server, and within the iRODS Catalog). The three steps include configuration, the `docker run` command, and setting up the mountpoint.

Configuration

Configuration for NFSRODS includes three configuration files, two of which do not need changes from the distributed examples. The `exports` and `log4j.properties` files can be used as is.

The `server.json` file needs to be updated to point to the correct iRODS server:

```
{
    // This section defines options for the NFSRODS NFS server.
    "nfs_server": {
        // The port number within the container to listen for NFS requests.
        "port": 2049,

        // The path within iRODS that will represent the root collection.
        // We recommend setting this to the zone. Using the zone as the root
        // collection allows all clients to access shared collections and data
        // objects outside of their home collection.
        "irods_mount_point": "/tempZone",
```

```
        // The refresh time for cached user information.
        "user_information_refresh_time_in_milliseconds": 3600000,

        // The refresh time for cached stat information.
        "file_information_refresh_time_in_milliseconds": 1000,

        // The refresh time for cached user access information.
        "user_access_refresh_time_in_milliseconds": 1000,

        // 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
    },

    // This section defines the location of the iRODS server being presented
    // by NFSRODS. The NFSRODS server can only be configured to present a single zone.
    "irods_client": {
        "host": "hostname",
        "port": 1247,
        "zone": "tempZone",

        // Defines the target resource for new data objects.
        "default_resource": "demoResc",

        // Enables/disables SSL/TLS between NFSRODS and the iRODS server.
        //
        // The following options are available:
        // - CS_NEG_REQUIRE: Only use SSL/TLS.
        // - CS_NEG_DONT_CARE: Use SSL/TLS if the iRODS server is not set to CS_NEG_REFUSE.
        // - CS_NEG_REFUSE: Do NOT use SSL/TLS.
        "ssl_negotiation_policy": "CS_NEG_REFUSE",

        // The total amount of time before an idle connection times out.
        // Defaults to 600 seconds.
        "connection_timeout_in_seconds": 600,

        // An administrative iRODS account is required to carry out each request.
        // The account specified here is used as a proxy to connect to the iRODS
        // server for some administrative actions. iRODS will still apply policies
        // based on the requesting user's account, not the proxy admin account.
        "proxy_admin_account": {
            "username": "rods",
            "password": "rods"
        }
    }
}
```

The `nfs_server` section of the configuration file defines the settings for the NFSv4 side of NFSRODS. This includes the

`port` number to expose as NFS (default `2049`), the `irods_mount_point` to define how deep within iRODS the mount-point will expose the virtual filesystem, and some cache settings (`user_information_refresh_time_in_milliseconds`, `file_information_refresh_time_in_milliseconds`, and `user_access_refresh_time_in_milliseconds`) for how long the NFSRODS server will keep a local copy of information found from the underlying Unix system or the iRODS catalog.

The `irods_client` section of the configuration file defines the settings for the iRODS client side of NFSRODS (`host`, `port`, and `zone`). The `default_resource` setting will define where any newly created files within the mount-point are physically created within iRODS. Also found here are the `ssl_negotiation_policy` and the `connection_timeout_in_seconds` setting for idle connections to refresh.

NFSRODS occasionally needs to take action within iRODS that it would not be able to take without a higher privilege level. In these cases, NFSRODS uses the proxy mechanism of iRODS to request actions on behalf of the requesting user. The `proxy_admin_account` is used to configure a rodsadmin username and password.

Docker

Starting NFSRODS requires a single `docker run` command of the form:

```
$ docker run -d --name nfsrods \
             -p <public_port>:2049 \
             -v </full/path/to/nfsrods_config>:/nfsrods_config:ro \
             -v </full/path/to/etc/passwd/formatted/file>:/etc/passwd:ro \
             nfsrods
```

The options launch the image known as `nfsrods`, put the container into daemon mode, and define the name of the running container (`nfsrods`), the port mapping from the outside world into the container, the volume mount to the configuration files, and the volume mount of the host system's `/etc/passwd`-formatted file.

It is important to note that the volume-mounted `/etc/passwd`-formatted file is expected to contain all of the users planning to use NFSRODS. The users defined in this file MUST be defined in iRODS as well. Their usernames must match the names defined in this file exactly as this is how NFSRODS matches users to the correct account in iRODS.

Restarting the NFSRODS server will not affect existing mountpoints other than the requirement to re-fetch any lost cache information.

SSL

If you want to connect NFSRODS to an iRODS Zone that is using SSL, a certificate file can be mounted for use within the container:

```
             -v </full/path/to/certificate.crt>:/nfsrods_ssl.crt:ro
```

The container will load any cert it finds at `/nfsrods_ssl.crt` within the container into the OpenJDK keystore.

sssd Integration

As an alternative to an `/etc/passwd`-formatted file, the default NFSRODS container also supports `libnss-sss`. It can be used by configuring sssd on the container host and binding the sssd socket into the container.

```
$ docker run -d --name nfsrods \
             -p <public_port>:2049 \
             -v </full/path/to/nfsrods_config>:/nfsrods_config:ro \
             -v /var/lib/sss:/var/lib/sss \
             nfsrods
```

Using sssd, NFSRODS can use any sssd domain for ID mapping, including AD or LDAP. If sssd and `/etc/passwd` are used together, passwd will be consulted first.

Mountpoint

Once the NFSRODS server is running, the standard `mount` command can be used to mount the remote filesystem and provide a location for regular users to get access to the iRODS namespace:

```
$ sudo mkdir <mount_point>
$ sudo mount -o sec=sys,port=<public_port> <hostname>:/ <mount_point>
```

Note the `hostname` is the hostname where NFSRODS is running and the `:/` after the hostname express to the mount command to mount the entire namespace provided by NFSRODS.

If you do not receive any errors after mounting, then a unix user with a properly mapped UID and username should be able to access the mount point like so:

```
$ cd <mount_point>/path/to/collection_or_data_object
```

FUTURE WORK

NFSRODS v1.0 represents a nearly feature-complete release and has been deployed into production in multiple enterprise environments.

The only major features remaining to be added are hard link support and parallel file transfers in and out of iRODS. Parallel transfer may be possible with the upcoming release of iRODS 4.2.9 where multiple streams can operate on port 1247, reading or writing to a single data object.

NFSRODS v1.0 incorporates the NFStest[8] suite but should be paired with a performance testing model to characterize its overhead. Anecdotal feedback suggests best and usable performance when the NFSRODS server is co-hosted with the iRODS catalog provider. This makes sense as it reduces additional network hops.

SUMMARY

The demand for a virtual filesystem with included policy and well-understood semantics is very strong. iRODS provides that abstraction and capability. However, it takes a lot of engineering effort to teach existing tools and workflows to speak the iRODS protocol. It is more likely that tools can read and write into a mountpoint provided by a compatibility layer between POSIX and iRODS.

NFSRODS v1.0 provides this compatibility layer and has been deployed into production in multiple enterprise environments. Existing tools can read and write into the iRODS namespace without any changes to their own code, and iRODS organizational policy is enforced on the server.

REFERENCES

[1] iRODS Client NFSRODS. https://github.com/irods/irods_client_nfsrods

[2] Haynes, T., Noveck, D.: Network File System (NFS) Version 4 Protocol (2015) https://tools.ietf.org/html/rfc7530

[3] 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)

[4] Draughn, K., Russell, T., et al: NFSRODS: Presenting iRODS as NFSv4.1. 6pp. 2019 iRODS User Group Meeting. (2019) https://irods.org/uploads/2019/Draughn-iRODS-NFSRODS-paper.pdf

[5] NFS4J. https://github.com/dCache/nfs4j

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

[7] Traditional Unix permissions. https://en.wikipedia.org/wiki/File_system_permissions#Traditional_Unix_permissions

[8] NFStest http://wiki.linux-nfs.org/wiki/index.php/NFStest

iRODS Rule Engine Plugin: Hard Links 4.2.8.0

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

This new C++ rule engine plugin provides an iRODS system the ability to convey hard links to its users. An iRODS system stores a hard link when replicas of two different iRODS data objects with different logical paths share a common physical path on the same host. When this occurs, metadata is added to both logical data objects for bookkeeping. This talk will explain the original use cases for hard links in iRODS and introduce Conway Diagrams to help visualize the various corner cases.

iRODS UGM 2020, June 9-12, 2020, Virtual

Creating an iRODS zone with Terraform

Brett Hartley
Wellcome Sanger Institute
Hinxton, Cambridgeshire, UK
bh9@sanger.ac.uk

ABSTRACT

A year ago, Sanger had 2 types of zones: production and development zones.

The development zones originally were for testing and development of both server and client components. Over time, these zones became more and more necessary for client side testing. Server side testing became increasingly limited. For the most part this was fine, because we didn't really need to do server side testing of potentially breaking changes, because we hadn't upgraded in a while (most zones were 4.1.12 at the time)

The decision was made that we should upgrade both the iRODS version and the operating system version to 4.2.7 and Ubuntu 18.04. This meant upgrading iRODS on over 100 machines, with minimal disruption to the services and the >9PB they serve. Part of any good upgrade process is testing on a suitable test infrastructure. Our objective was to produce an Infrastructure as Code template to create an iRODS zone, so that zones could be created whenever needed, in our OpenStack environment, freeing up resources when they were no longer required.

The end product has been used extensively in our upgrade testing, and has proven to be a useful tool for other miscellaneous testing. Being able to stand up a new zone in minutes, rather than days has also added 2 more types of zone: testing zones, which we spin up to test specific parts of iRODS, e.g. to produce simple reproducers for otherwise hard to find bugs, and demonstration zones, which have features that we are looking to add to development and production zones in the future.

iRODS UGM 2020, June 9-12, 2020, Virtual

Building a national Research Data Management (RDM) infrastructure with iRODS in the Netherlands

Saskia van Eeuwijk
SURF
Netherlands
saskia.vaneeuwijk@surfsara.nl

Hylke Koers
SURF
Netherlands
hylke.koers@surfsara.nl

ABSTRACT

In the Netherlands a lot of universities are looking at iRODS to support their researchers, as they recognize the powerful potential of the tool in two areas: support for secure cooperation, and support over the entire research data life cycle. Unfortunately, support teams in universities are hesitant to introduce the tool for two reasons:

- iRODS in itself is more suitable for IT-power users
- The support needed of iRODS within the university asks specific knowledge.

SURF, a national organization providing IT support and infrastructure for universities, stepped in and is now working closely together with six universities towards a national RDM infrastructure based on iRODS.

SURF offers a hosted environment for iRODS for all participating universities, thus creating possibilities for the researchers without the need for universities to invest upfront. Also, SURF unburdens the universities by offering a hosted, supported environment. YODA, open source software created by the University of Utrecht (UU) on top of iRODS, is being used to also attract users that have high demands in user friendliness, thanks to a web interface designed to guide the researchers in many steps of the data life cycle, from the ingestion of the data to their publication. SURF offers together with UU the support for the combined environments. The service is in pre-production state at the moment. Already, the participating universities join in the development of YODA and iRODS.

In the next two years, we hope to prove to the participating universities specifically, but also to the other universities in the Netherlands, that iRODS and YODA are useful RDM tools for a lot of researchers. Early 2022 we plan to expand the service and the cooperation. We hope by that time we can truly state that iRODS an YODA are an important part of the RDM infrastructure in the Netherlands.

In our presentation we want to focus on describing a case study for the use of iRODS, not for a specific research group, but for an entire nation to enhance the support of their researchers by working together on this iRODS based infrastructure.

iRODS UGM 2020, June 9-12, 2020, Virtual

iRODS at Bristol Myers Squibb: Status and Prospects. Leveraging iRODS for scientific applications in Amazon AWS Cloud

Mohammad Shaikh
Bristol Myers Squibb
New Jersey, USA
Mohammad.Shaikh@bms.com

Oleg Moiseyenko
Bristol Myers Squibb
New Jersey, USA
oleg.moiseyenko@bms.com

ABSTRACT

The iRODS practice at Bristol Myers Squibb is growing as we continue use it as the primary system of record across several different scientific projects at multiple cloud environments. This presentation shares the latest updates on how Bristol Myers Squibb is leveraging iRODS to manage and enrich various datasets in Amazon AWS Cloud. We will cover typical data flows, architectural patterns, as well as interesting approaches for how we manage AWS Lambda functions to update an iRODS Catalog with events that occur in one or more S3 buckets.

iRODS UGM 2020, June 9-12, 2020, Virtual

Keeping Pace with Science: the CyVerse Data Store in 2020 and the Future

Tony Edgin
CyVerse / University of Arizona
Tucson, Arizona, USA
tedgin@cyverse.org

Edwin Skidmore
CyVerse / University of Arizona
Tucson, Arizona, USA
edwin@cyverse.org

ABSTRACT

This talk will describe the current features of the CyVerse Data Store and plans for its evolution. Since its inception in 2010, the Data Store has leveraged the power and versatility of iRODS by continually extending the functionality of CyVerse's cyber-infrastructure. These features include project-specific storage, offsite replication, third-party service and application integrations, several data access methods, event stream publishing for indexing, and optimizations for accessing large sets of small files. Current efforts to enhance the Data Store include project-specific THREDDS Data Servers, S3 integration to allow bidirectional data flow between third-party storage and compute, and integration with CyVerse's Continuous Analysis platform, an event-driven container-native execution platform.

iRODS UGM 2020, June 9-12, 2020, Virtual

iRODS Logical Quotas Policy Plugin

Jonathon Anderson
University of Colorado Research Computing
Boulder, Colorado, USA
jonathon.anderson@colorado.edu

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

University of Colorado Research Computing uses iRODS to provision space in its PetaLibrary/archive research data storage service. This storage is implemented as top-level collections and is sold at a $/TB/year rate. In our experience on other platforms, implementing storage allocations with user and/or group quotas leads to confusion, particularly when individual users have access to multiple discrete storage allocations, as ownership metadata falls out-of-sync from the logical spacial hierarchy of the file system. To provide more logical quotas atop the iRODS collection hierarchy, the iRODS logical quotas policy plugin tracks the logical size of a collection--calculated as the total size of all data objects nested within it--as collection-level metadata that is consulted before and updated after i/o. This allows us to place a logical size limit on a collection, more closely matching our end-users expectations of how storage allocations should behave. This talk covers our deployment experience and details about the plugin implementation.

iRODS UGM 2020, June 9-12, 2020, Virtual

iRODS Policy Composition: Principles and Practice

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

Historically a single static policy enforcement point, such as acPostProcForPut, was the sole location for all policy implementation. With the addition of a continuation code to the rule engine plugin framework, we may now configure multiple policies to be invoked for any given policy enforcement point. This subsequently allows for a separation of concerns and clean policy implementation. The policy developers now have the ability to separate the "when" (the policy enforcement points) from the "what" (the policy itself). How the policy is then invoked becomes a matter of configuration rather than implementation.

Given this new approach, multiple policies can be configured together, or composed, without the need to touch the code. For example, the Storage Tiering capability is effectively a collection of several basic policies: Replication, Verification, Retention, and the Violating Object Discovery. All of these policies are configured via metadata annotating root resources, and taken as a whole provide a flexible system for automated data movement.

iRODS UGM 2020, June 9-12, 2020, Virtual

iRODS Client: AWS Lambda Function for S3 1.0

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

ABSTRACT

Under development for less than six months, this new AWS Lambda[1] function updates an iRODS Catalog with events occurring in one or more S3 buckets. Files created, renamed, or deleted in S3 appear quickly in iRODS.

The following AWS configurations are supported with the 1.0 release:

- S3 -> Lambda -> iRODS

- S3 -> SNS -> Lambda -> iRODS

- S3 -> SQS -> Lambda -> iRODS

Keywords

iRODS, S3, AWS, lambda, data management

INTRODUCTION

As the iRODS Server has continued to improve its reliability, stability, and speed, additional functionality is being demanded of the clients that connect to the iRODS Server. One of the additional bits of functionality requested over the past few years has been a better ”out-of-the-box” onboarding, or ingest, experience for existing data.

In 2018, the Automated Ingest Capability[2], a Python-based client provided and supported by the iRODS Consortium, was introduced. It provided a parallel, distributed solution for quickly scanning and re-scanning an existing filesystem (and later, S3[3]) and registering or PUT-ting files into iRODS. It works very well, but it requires a bit of configuration and computing overhead, and cannot see the ”negative space”, or when a file is removed from the filesystem.

This Lambda function represents an alternative approach. It reduces the complexity involved in keeping up with a changing filesystem and improves the coverage of removed files.

DESIGN GOALS

The design goals of this new approach to getting files registered into iRODS were three-fold. First, it needed to play nicely with the universe of tools that already know how to write to S3 directly. Second, it must allow those updates within the S3 namespace to smoothly flow into the iRODS Catalog. And third, it would trigger iRODS automated data management due to crossing the policy boundary (the iRODS API).

A tool that could provide all three of these goals would solve a number of use cases the Consortium is seeing in the community. Everyone who begins to consider iRODS as a solution already has a lot of data in existing systems. Many of these existing systems are now in the ”cloud”, most commonly in Amazon S3 buckets.

iRODS UGM 2020 June 9-12, 2020, Virtual

Considerations

In looking at other clients and existing work, we realized that Amazon's Lambda service could provide the "place" to run the new client code. Lambda can run Python code, and iRODS already provides a well-tested and robust Python client library (python-irodsclient[4]).

Success with this approach would be defined as near-real-time, asynchronous updates to the iRODS Catalog. We were interested not only in visibility of create and rename operations, but also delete operations.

IMPLEMENTATION

As a single Python file, the implementation is straightforward and self-contained. The `lambda_handler` function captures variables from its configuration and environment, parses the event coming from S3, and then selects whether to perform a registration into the iRODS Catalog or to perform a delete from the iRODS Catalog based on the parsed S3 event.

The S3 Events that trigger an iRODS registration are:

- `ObjectCreated:Put`
- `ObjectCreated:Copy`
- `ObjectCreated:CompleteMultipartUpload`

The S3 Events that trigger an iRODS deletion are:

- `ObjectRemoved:Delete`
- `ObjectRemoved:DeleteMarkerCreated`

Rename operations sent to S3 are decoupled into independent `ObjectRemoved` and `ObjectCreated` events. This is discussed later in the Limitations section.

At this time, each firing of the Lambda function reacts to a single S3 event.

CONFIGURATION OPTIONS

Inputs to the Lambda function come from three places.

First, the S3 Event payload itself provides the event type, bucket name, key name, and for `ObjectCreated` events, the file size. This is not configuration, but supplies important context for the functioning of the Lambda.

Second, the Python runtime environment provides configuration information to the Lambda including `IRODS_COLLECTION_PREFIX`, `IRODS_ENVIRONMENT_SSM_PARAMETER_NAME`, and optionally, `IRODS_MULTIBUCKET_SUFFIX`. These must be defined by the owner of the Lambda.

And third, the iRODS connection information required by the Lambda is stored in the `AWS Systems Manager > Parameter Store`
as a JSON object of the type `SecureString` under the name that matches the environment variable `IRODS_ENVIRONMENT_SSM_PARAMETER_NAME`. This information must be defined and protected separately because it contains the password of the configured iRODS user.

The `SecureString` is expected to have the form:

```
{
    "irods_default_resource": "s3Resc",
    "irods_host": "irods.example.org",
    "irods_password": "rods",
    "irods_port": 1247,
    "irods_user_name": "rods",
    "irods_zone_name": "tempZone"
}
```

iRODS is assumed to have its associated S3 Storage Resource(s) configured with `HOST_MODE=cacheless_attached`. There should be no compound resources involved in the relevant resource hierarchy.

The Lambda function must be configured to trigger on all `ObjectCreated` and `ObjectRemoved` events for a connected S3 bucket. Defining and maintaining the AWS Policies can vary widely in practice and is beyond the scope of this document.

A well-configured Lambda function will update the iRODS Catalog in near-real-time as events flow from the S3 bucket(s).

The following AWS configurations are supported at this time:

Figure 1. S3 to Lambda to iRODS - Direct Connection

Figure 1 shows the simplest and most straightforward configuration. Events from S3 flow directly to the Lambda function and triggered immediately.

Figure 2. S3 to Simple Notification Service (SNS) to Lambda to iRODS

Figure 2 shows how the Lambda can be triggered as one of many services being notified by S3 Events coming out of a bucket as the Simple Notification Service (SNS) can be configured to send its Events to multiple endpoints. This is useful for adding Lambda to an existing ecosystem of cloud microservices and APIs.

Figure 3. S3 to Simple Queue Service (SQS) to Lambda to iRODS

Figure 3 shows how the Lambda can be triggered in a more robust store-and-forward configuration. The Simple Queue Service (SQS) provides retry functionality for failed operations as well as provides the ability to operate on multiple events at once to save operational costs.

SSL Support

SSL to iRODS is supported by placing a certificate in a relative path within the Lambda package.

If the Lambda needs to be configured to connect with an SSL-enabled iRODS Zone, the following additional keys need to be included in the environment in the Parameter Store:

```
{
    "irods_client_server_negotiation": "request_server_negotiation",
    "irods_client_server_policy": "CS_NEG_REQUIRE",
    "irods_encryption_algorithm": "AES-256-CBC",
    "irods_encryption_key_size": 32,
    "irods_encryption_num_hash_rounds": 16,
    "irods_encryption_salt_size": 8,
    "irods_ssl_verify_server": "cert",
    "irods_ssl_ca_certificate_file": "irods.crt"
}
```

The `irods_ssl_ca_certificate_file` is a relative path to a certificate file (or certificate chain file) within the Lambda package.

Multi-Bucket Support

This Lambda function can also be configured to receive events from multiple sources at the same time.

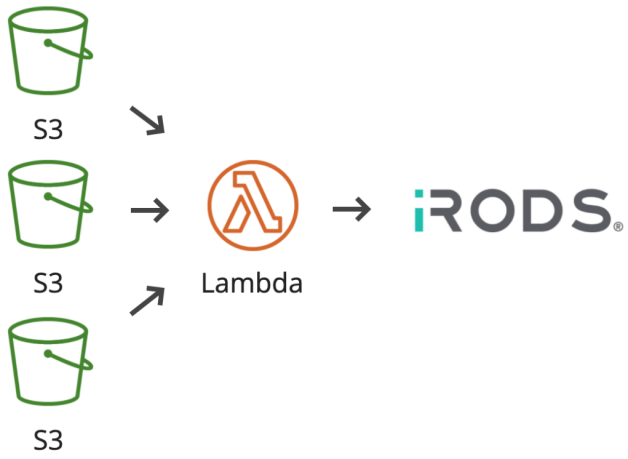

Figure 4. Multi-bucket S3 to Lambda to iRODS

Figure 4 shows how the Lambda can be triggered by an event in multiple different S3 buckets. This is made possible by having the target iRODS Resource be determined at runtime. This configuration reduces the number of Lambda functions that need to be deployed and maintained within an organization.

If the `irods_default_resource` is NOT defined in the environment in the Parameter Store, then the Lambda function will derive the name of a target iRODS Resource.

By default, the Lambda function will append `_s3` to the incoming bucket name found in the S3 event.

For example, if the incoming event comes from bucket `example_bucket`, then the iRODS resource that would be targeted would be `example_bucket_s3`.

However, if `IRODS_MULTIBUCKET_SUFFIX` is defined as `-S3Resc`, the the iRODS resource that would be targeted would be `example_bucket-S3Resc`.

LIMITATIONS AND FUTURE WORK

This first release of the Lambda function meets all the design goals defined earlier. However, some limitations have been documented.

The first is that S3 is decoupled from the Lambda itself. A rename operation sent to S3 is decomposed into a create event and a delete event. Because of this decoupling and without the ability to confirm that two events are related to one another, iRODS must treat this as a new data object. This means any metadata AVUs associated with the now-deleted data object is lost.

It is possible this could be remedied with a comparison of full checksums of the object to be deleted and any new incoming objects within a certain window of time, but this would be slow, and therefore, more expensive to operate.

The second limitation is that SQS configuration is limited to `batch_size = 1`. Operating on more than one message

at a time would increase performance (event throughput) and reduce the cost of running this Lambda at AWS. However, it is unclear how the Lambda could signal partial success at this time. What happens to the failed events?

It is possible this could be remedied with an atomic database operations API at the iRODS Server level. If any events fail to be processed, the entire batch could be safely returned to the SQS queue with no side effects on the iRODS Catalog.

SUMMARY

The AWS Lambda function for S3, a new iRODS client written in Python, has been developed relatively rapidly and is already in multiple production deployments. It handles both creates and deletes within an S3 bucket and updates the associated iRODS Catalog in near-real-time.

I would like to thank Bristol Myers Squibb for providing the pre-release testing environment and conformance test scenarios that drove much of this initial work.

REFERENCES

[1] Russell, Terrell: iRODS Client AWS Lambda S3 (2020). `https://github.com/irods/irods_client_aws_lambda_s3`

[2] 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`

[3] Amazon S3 (2006) `https://en.wikipedia.org/wiki/Amazon_S3`

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