Technology Update

Terrell Russell, Ph.D.
@terrellrussell
Chief Technologist, iRODS Consortium

June 8-11, 2021
iRODS User Group Meeting 2021
Virtual Event

Philosophical Drivers

- Plugin Architecture
  - core is generic - protocol, api, bookkeeping
  - plugins are specific
  - policy composition

- Modern core libraries
  - standardized interfaces
  - refactor iRODS internals
    - ease of (re)use
    - fewer bugs

- Replicas as first class entities
  - logical locking

- Consolidation of data movement
  - dstreams all on 1247, no more high ports

- Configuration, Not Code

In The Last Year

iRODS Release	Issues Closed
4.2.9	314

```
~/irods $ $ git shortlog --summary --numbered 4.2.8..4.2.9
   129  Kory Draughn
   125  Alan King
    35  Markus Kitsinger (SwooshyCueb)
    22  Terrell Russell
     9  d-w-moore
     8  Justin James
     5  Jason Coposky
     2  Ilari Korhonen
     1  Martin Pollard
     1  Matthew Vernon
     1  Nick Hastings
```

In The Last Year

Plugins

- Python Rule Engine Plugin
- Storage Tiering Rule Engine Plugin
- Auditing (AMQP) Rule Engine Plugin
- S3 Resource Plugin
- Kerberos Authentication Plugin
- Curl Microservice Plugin
- Indexing Rule Engine Plugin
- Logical Quotas Rule Engine Plugin
- Metadata Guard Rule Engine Plugin
- **Policy Composition Framework**
- **Policy Composition Event Handlers**

Clients

- **Python iRODS Client**
- **Metalnx**
- **NFSRODS**
- Automated Ingest Framework
- AWS Lambda for S3
- **C++ REST API**
- **Zone Management Tool (ZMT)**
- **iRODS Globus Connector**
- **iCommands**

Working Groups

iRODS

Technology Working Group

- Goal: To keep everyone up to date, provide a forum for roadmap discussion and collaboration opportunities

Metadata Templates Working Group

- Goal: To define a standardized process for the application and management of metadata templates by the iRODS Server
  - NIEHS / Data Commons
  - Utrecht / Yoda
  - Maastricht / DataHub+
  - Arizona / CyVerse

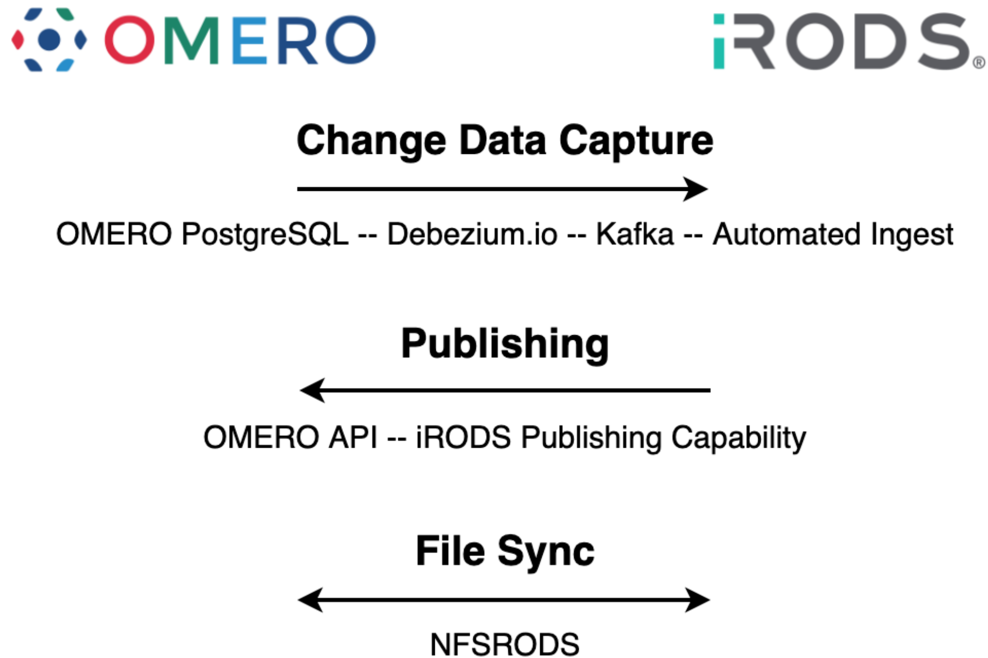
Authentication Working Group

- Goal: To provide a more flexible authentication mechanism to the iRODS Server
  - SURF
  - NIEHS
  - Sanger
  - CyVerse
  - Utrecht

Working Groups

Imaging Working Group (Announcing Now!)

- Goal: To provide a standardized suite of imaging policies and practices for integration with existing tools and pipelines
  - Open Microscopy Environment (and OMERO)
  - Neuroscience Microscopy Core at UNC School of Medicine
  - New York University
  - Santa Clara University
  - UC San Diego
  - CyVerse
  - NIEHS



Last Year and Next Year

- Not in This Talk / Separate Talks
  - Alan King
    - Logical Locking
  - Markus Kitsinger
    - iCommands Userspace Packaging
  - Daniel Moore
    - NetCDF Updates
    - Python iRODS Client 1.0
  - Kory Draughn
    - NFSRODS 2.0
  - Jason Coposky
    - Policy Composition
    - C++ REST API
  - Bo Zhou
    - Zone Management Tool (ZMT)
    - Metalnx 2.4.0 and GalleryView
  - Justin James
    - iRODS Globus Connector

- Included in This Talk
  - Kory Draughn
    - Core Libraries
  - Justin James
    - S3 Optimization

Internal Refactoring - New C++ iRODS Libraries

iRODS

6 in 2019

- **filesystem**
  - server, plugins, icommands
- **iostreams**
  - server, indexing, S3 resource, icommands
- **thread_pool**
  - delay execution server, S3 resource
- **connection_pool**
  - delay execution server
- **query**
  - server, indexing, publishing, storage tiering
- **query_processor**
  - delay execution server, storage tiering

9 in 2020

- **key_value_proxy**
  - Provides a map-like interface over an existing keyValuePair_t.
- **lifetime_manager**
  - Guarantees that heap-allocated iRODS C structs are free'd at scope exit.
- **user group administration**
  - Simplifies management of iRODS users and groups.
- **shared_memory_object**
  - Simplifies access and management of shared memory.
- **with_durability**
  - A convenient retry mechanism for functions and function-like objects.
- **query_builder**
  - Enables query objects to be constructed lazily.
- **client_api_whitelist** *(server-side only)*
  - An interface for managing and querying the client API whitelist.
- **scoped_privileged_client** *(server-side only)*
  - Elevates the client's privileges for the duration of a scoped block.
- **scoped_client_identity** *(server-side only)*
  - Changes the client's identity for the duration of a scoped block.

2021 - 11 New iRODS C++ Libraries

iRODS

- **Replica State Table** *(server-side only)*
  - Keeps track of all information for each replica of a given data object. The primary building block used to implement Logical Locking.
- **Logical Locking** *(server-side only)*
  - Works with the Replica State Table library to manage the replica status of a group of replicas. This library is the coordination mechanism for multiple readers and writers to the same data object.
- **DNS Caching** *(server-side only)*
  - Helps to improve network performance within a zone by caching DNS query results for a certain amount of time.
- **Hostname Caching** *(server-side only)*
  - Helps to improve performance by caching the results obtained from the hosts_config.json file. This ultimately avoids disk I/O.
- **Catalog Utilities** *(server-side only)*
  - Provides convenience functions for database operations and redirection.
- **Metadata**
  - A high level interface around the existing metadata operations that simplifies manipulation of AVUs.
- **Replica**
  - Provides operations that make it easy to manipulate and manage replicas. Created as a result of the Filesystem library being updated to handle only the logical space.
- **Parallel Transfer Engine**
  - A low-level tool that enables data to be transferred in parallel over multiple iRODS connections.
- **Replica Access Table** *(server-side only)*
  - Controls write access to intermediate replicas through the use of tokens.
- **Client Connection**
  - Provides a very simple way to connect to iRODS without having to know the C APIs.
- **Resource Administration**
  - A high level interface around the existing resource administration operations that simplifies manipulation of resources.

2021 - 4 New API Plugins

iRODS

- **Atomic Apply ACL Operations**
  - Allows multiple ACLs to be manipulated within a single transaction.
  - Includes a wrapper microservice.
- **Data Object Finalize**
  - Atomically commits changes to multiple replicas to the catalog.
- **Replica Open**
  - A convenience function that bundles rxDataObjOpen and rx_get_file_descriptor_info into one call.
  - Avoids additional network traffic.
- **Replica Close**
  - Provides controls around closing a specific replica.
  - This API plugin is meant for experts.
- **Touch**
  - The iRODS equivalent of UNIX touch, but as an API plugin.
  - Includes a wrapper microservice and new icommand (i.e. itouch).

2021 - 1 Refactored API

iRODS

- **Checksum**
  - The checksum API now has two modes: Verification and Lookup/Update

Verification mode performs the following operations:

1. Reports replicas with mismatched size information (physical vs catalog).
2. Reports replicas that are missing checksums.
3. Reports replicas with mismatched checksums (computed vs catalog).
4. Reports if the replicas do not share the same checksum.

Step 3 can be time consuming depending on the size of the replica. The server can be instructed to skip detection of mismatched checksums by passing the NO_COMPUTE_KW keyword.

The verification operations work for one or more replicas. However, when a specific replica is targeted, step 4 is not performed.

Lookup/Update mode performs the following operations:

1. Returns the existing checksum or computes and updates checksums.
2. Reports if the replicas do not share the same checksum assuming no errors were encountered during the previous step.

The following keyword(s) are now NOPs (in regards to rxDataObjChksum):

- VERIFY_VAULT_SIZE_EQUALS_DATABASE_SIZE_KW

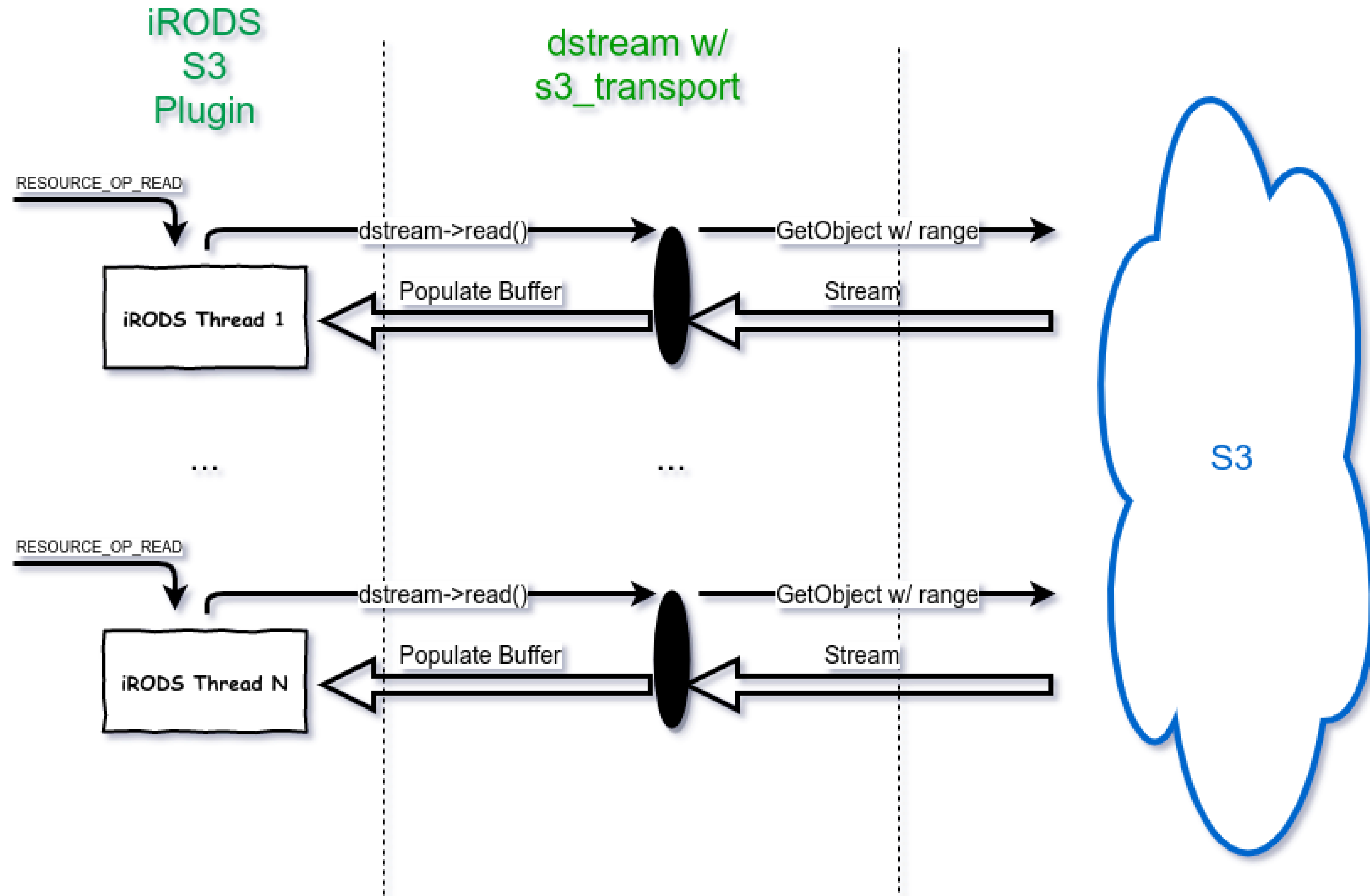
Operations that target multiple replicas will only affect replicas that are marked good. This means intermediate, locked, and stale replicas will be ignored.

Operations that target a specific replica are allowed to operate on stale replicas.

S3 Streaming Plugin - Review

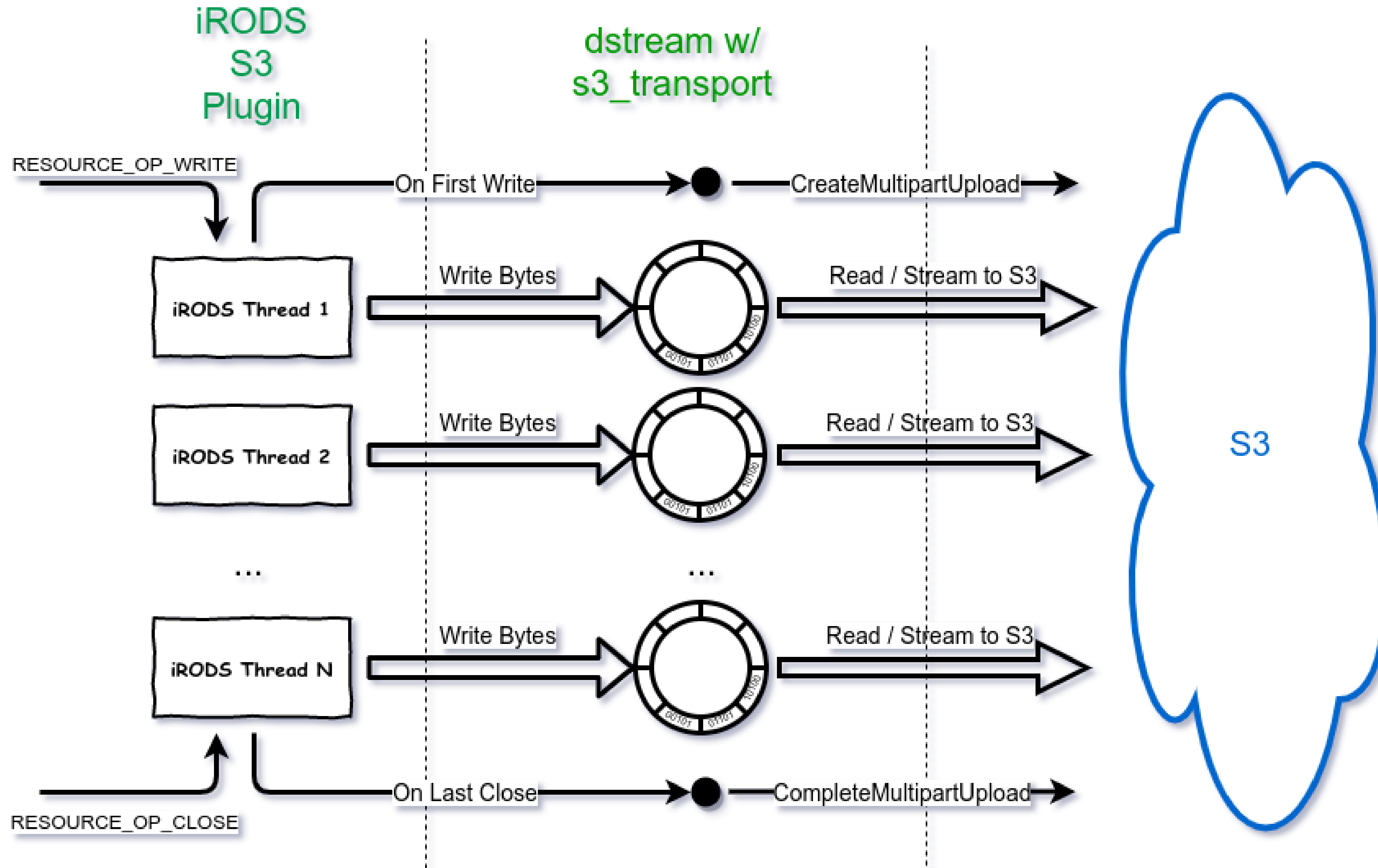
- The existing "cacheless" plugin always uses a cache file on the server.
- The new plugin will (in most common cases) stream data directly from iRODS to S3 without using an intermediate cache file.
- All reads and writes are handled by an **s3_transport** class which extends **irods::experimental::io::transport**.
  - If a single buffer write is being performed then multipart is not used and data is streamed sequentially to S3.
  - If parallel transfer on write is performed in iRODS, a multipart upload is started and each transfer thread streams data directly to S3 for its part.

S3 Streaming Plugin - Review (Read)



S3 Streaming Plugin - Review (Write)

iRODS



S3 Streaming Plugin - Changes in the last year

iRODS

We have had many partners test the streaming S3 plugin over the last year. Throughout this process bugs and limitations were identified and resolved. The following is a summary of some of the issues.

- During this process we discovered some bugs in both our fork of the libs3 code and in the base libs3 code. In addition the libs3 code had quite a few bug fixes and enhancements that were made since our initial fork.
  - We rebased our changes on the latest libs3 code.
  - Fixed bugs we found in the libs3 code.
  - Fixed some bugs we found in our previous updates to the libs3 code - specifically the use of global variables that could cause issues when an agent is both reading and writing simultaneously.

- Retries after retryable failures were not handled properly in the streaming S3 code.
  - Since data was read and removed from the circular buffer, it was no longer available for retries. Rather than popping data we are now doing a peek and only removing data when a file or part upload is successful.
  - Exponential backoff on retries.

S3 Streaming Plugin - Changes in the last year

iRODS

- Design of the circular buffer has changed.
  - Rather than holding chunks, it holds bytes directly. However, the CIRCULAR_BUFFER_SIZE parameter is multiples of the S3_MINIMUM_PART_SIZE.
  - Timeouts were added to the circular buffer in cases a reader or writer fails. This was noticed when replicating from one S3 resource to another.

- Configuration options that were not yet implemented (such as S3_ENABLE_MPU and S3_MPU_THREADS) have been implemented.

- Better partition of bytes to parts for multipart uploads:
  - Required because with data remaining on the buffer, each thread may have more bytes than the buffer size.
  - Each thread breaks its bytes up into parts that are between the minimum part size and twice the minimum part size.

S3 Streaming Plugin - Rules for Cache File Use / Policy For Transfers

iRODS

- Rules for when a cache file is used has been refined.
  - All objects opened in read-only mode (including `iget`) will be cacheless as S3 allows random access reads on S3 objects.
  - All `iput` and `irepl` will stream without a cache except in the following two cases:
    - iRODS is performing a parallel transfer but each part size < S3_MPU_CHUNK size.
    - iRODS is performing a parallel transfer but multipart uploads are disabled.
  - For non-icommand clients, to avoid using cache the oprType must be set to PUT_OPR, REPLICATE_DEST, or COPY_DEST which informs the plugin that the client will follow iput-like behavior:
    - The file will be split between N processes/threads so that each has the same number of bytes with the "extra" N-1 bytes handled by the last process or thread.
- If the oprType is set as above and the contract is not met, transfers will not work!!!
- In all cases, if the client is not using the server's policy to set the number of transfer threads, the NUM_THREAD_KW needs to be set by all client processes.

S3 Streaming Plugin - Rationale for the Parallel Transfer Contract

iRODS

- Constraints in how multipart uploads work:
  - Part numbers have to be sequential. (They can arrive in any order but part numbers can't be skipped.)
  - All part sizes (except last) must be greater than the minimum part size (5 MB).
  - So retries can be done, all part sizes must be less than the circular buffer size (default 10 MB).

- Limited knowledge of each S3 transfer thread. It knows:
  - Beginning offset
  - The file size
  - The number of threads.

- Each thread must determine its starting part number and how many bytes it will be sending. This is impossible without the contract mentioned in the previous slide or initial coordination between threads.

Example: 24 MB file with a 10 MB per-thread buffer and 3 transfer threads.

Scenario 1: Thread 0 sends 11 MB. Thread 1 sends 5 MB. Thread 2 sends 8 MB.
Scenario 2: Thread 0 sends 8 MB. Thread 1 sends 8 MB. Thread 2 sends 8 MB.

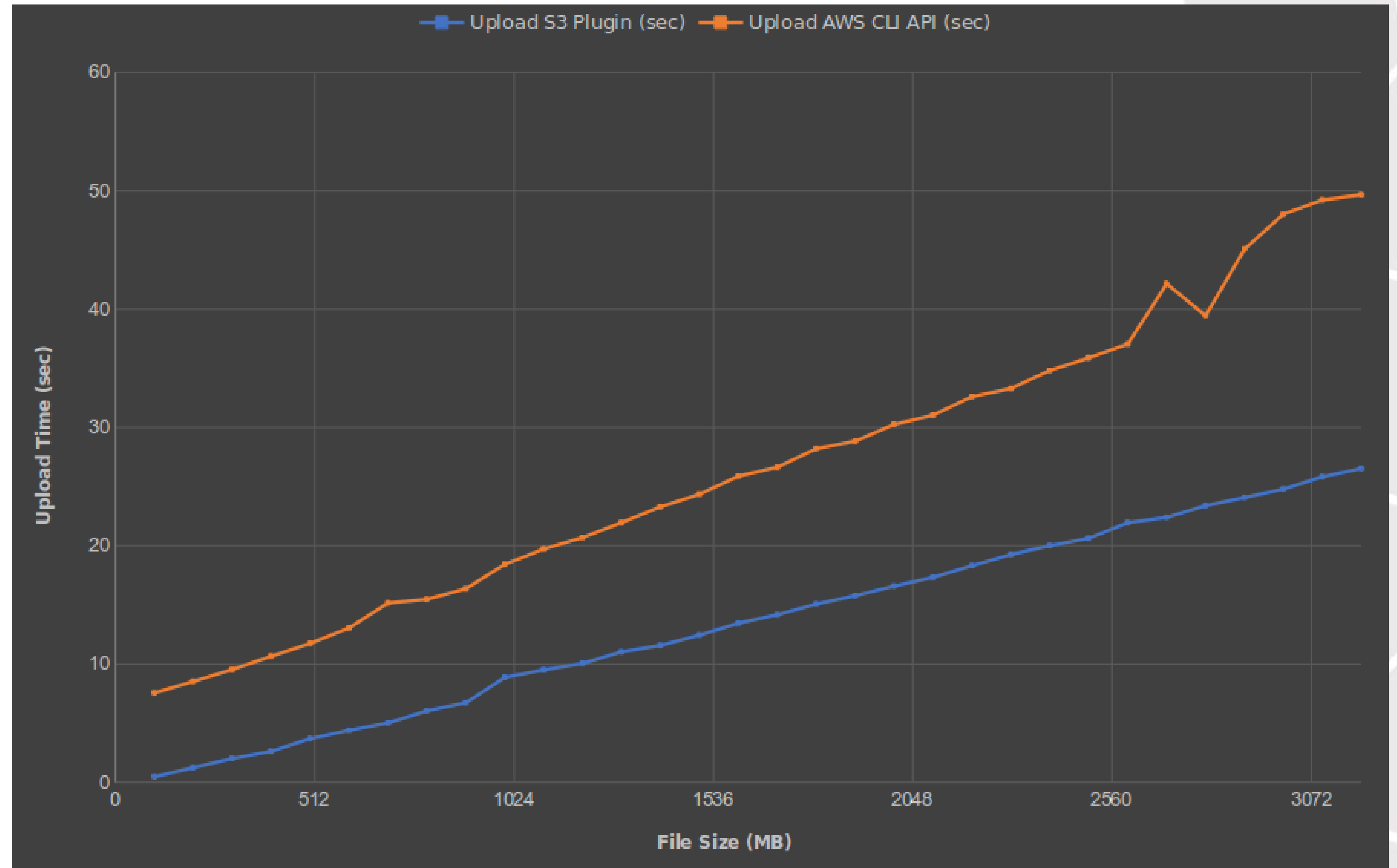
In both scenarios, thread 2 gets an offset of 16 MB, knows there are 3 threads, and the file size (24 MB).

What is thread 2's starting part number? In scenario 1 it would be part 4 because (thread 1 would require two parts) and in scenario 2 it would be part 3. **How many bytes is it sending?**

S3 Streaming Plugin - Upload Performance to Local MinIO Server

iRODS

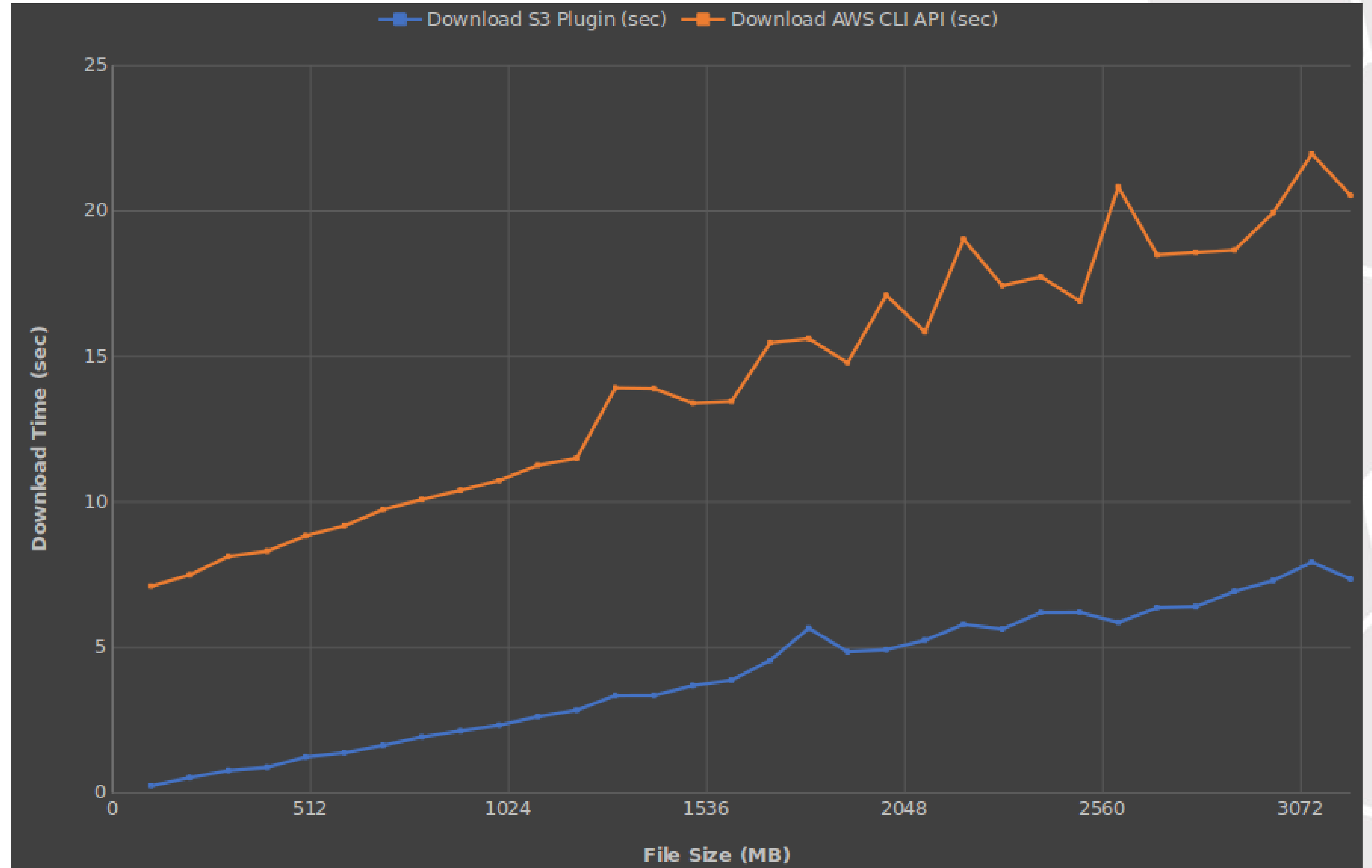
- 10 transfer threads (each)
- Uploads every 100 MB between 100 MB and 3200 MB
- Median of 5 uploads for each size



S3 Streaming Plugin - Download Performance to Local MinIO Server

iRODS

- 10 transfer threads (each)
- Uploads every 100 MB between 100 MB and 3200 MB
- Median of 5 downloads for each size



Philosophy to Policy

iRODS

With the new libraries and first class replicas, we can rewrite 90% of the internals, and then fix the things that depend on them later, with little expectation of regression, because the interfaces remain the same.

Internally

- We will have a new API... but not really
- Instead, we stepped back and built good tools
  - Allows us to refactor and go faster without breaking the 4.x API
  - This has turned out to be more powerful than originally expected

Externally

- It's a good story, the ability to compose policy into capabilities
- Can build smaller pieces of functionality which can be composed to help solve larger problems
- We don't have to worry about side effects

Continuation within the Rule Engine Plugin Framework allows administrators to break apart monolithic policy implementations into reusable components.

Active Development Work

- Core
  - iRODS 4.2.10
  - iRODS 4.3.0
  - Cloud Native iRODS (5.0.0?)

iRODS Internships - Summer 2021

iRODS Server Async Facility (4.2.x)
The iRODS Server Process Model consists of a main long-running server, a child long-running Agent Factory, and many short-lived Agent processes to serve client requests. There are a number of 'background tasks' that would be nice to have running as well that could do clean up, bookkeeping, etc. This project would be to design and implement an asynchronous facility for the iRODS Server.

Automated Ingest Refactor (Python client)
The iRODS Automated Ingest tool can currently scan in parallel a local filesystem and an S3 bucket for new and updated files. We are interested in adding the ability to scan an iRODS path as well. This will allow the scanner to see when files are removed (the negative space). The current logic needs to be refactored to provide these different targets as separate scanning strategies. Separating these strategies would also allow us to scan a queue or log (Kafka, RabbitMQ, etc.).

iRODS delayServer w/ Implicit remote() (4.3.0)
Currently, jobs executed by the delay server are tied to the machine on which the delay server is running (which is the catalog provider in 4.2.x). The delay server should be able to execute delayed rules on any machine in a zone. This is currently possible by embedding a remote() block inside of a delay() block. The server should grow a configuration option to hold a list of eligible rule execution servers which would be chosen randomly to execute rules on the delay queue. If no such list is provided, the server would behave as it does now, defaulting to a list of one server.

Big Picture

iRODS

Core

- 4.3.0 - Harden and Polish

Clients

- GUIs (Metalnx, ZMT, et al.)
- Onboarding and Syncing (Automated Ingest)
- File System Integration (NFSRODS / SMBRODS)
- iRODS Console (alongside existing iCommands)
- C++ REST API

Continue building out policy components (Capabilities)

We want installation and management of iRODS to become about policy design, composition, and configuration.

Please share your:

- use cases
- pain points
- hopes and dreams

Open Source Community Engagement

iRODS

Get Involved

- Working Groups
- GitHub Issues
- Pull Requests
- Chat List
- Consortium Membership

Tell Others

- Publish, Cite, Advocate, Refer