iRODS S3 Resource Plugin: Cacheless and Detached Mode

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

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

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

ABSTRACT

The iRODS S3 Resource Plugin has been updated with a new set of operating modes. It can now be configured to operate without the need for a local replica to be housed in a cache resource under a compound resource. Additionally, in detached mode, iRODS will not need to redirect to a particular iRODS server before connecting to the S3 object store. This functionality is made available alongside iRODS 4.2.6.

Keywords

iRODS, S3, AWS, cacheless, data management

INTRODUCTION

iRODS has been capable of surfacing S3-compatible object stores for more than a decade. Amazon launched their Simple Storage Service (S3) in March of 2006[1]. The original iRODS file driver code for S3 was written in 2009 and included in iRODS 2.2[2]. The iRODS S3 Resource Plugin was pulled from the main code and ported to the plugin architecture in 2013 with E-iRODS 3.0[3].

Since its inception, the iRODS S3 interface depended on being surfaced through a compound resource that held both cache and archive resources. The cache resource would hold 'local' replicas of data and be available for POSIX operations. The archive resource would hold non-POSIX interface replicas (object stores, tape formats, etc.) and usually require access with greater latency.

```
$ ilsresc
s3compound:compound
|-- s3archive:s3
‘-- s3cache:unixfilesystem
```

The S3 plugin itself only implemented a few operations:

```
irods::RESOURCE_OP_UNLINK
irods::RESOURCE_OP_STAT
irods::RESOURCE_OP_RENAME
irods::RESOURCE_OP_STAGETOCACHE
irods::RESOURCE_OP_SYNCTOARCH
```

All of the other POSIX-style operations were handled by the cache resource.

iRODS UGM 2019 June 25-28, 2019, Utrecht, Netherlands

This compound configuration was flexible but required a cache management policy to be devised, implemented, and executed unique to each deployment. This burden has proven more than most deployments wish to maintain. There has been a stated desire for an iRODS Consortium-backed cache management policy or set of rules to help with the 'normal' cases. Since every deployment has different opinions and requirements about what proper cache management looks like, a more elegant solution has presented itself... not having a cache to manage in the first place.

Network speeds have increased. Affordable access and availability to the cloud has increased. Both the academic and commercial comfort level with data in the cloud has increased. It was time for the S3 resource to manage its own temporary cache and not require additional complexity to perform well.

This initial release of the iRODS S3 Resource Plugin with Cacheless and Detached Mode satisfies this design goal and provides a simpler way to attach an S3-compatible storage system to iRODS.

MODES

The new plugin now supports three operating modes. This mode is set using the `HOST_MODE` parameter in the resource context string. If the `HOST_MODE` is not set, the default is `archive_attached`, which operates the same way as the legacy S3 plugin.

	Archive	**Cacheless**
Attached	archive_attached (default)	cacheless_attached
Detached	N/A	cacheless_detached

Table 1. Three modes for the iRODS S3 Resource Plugin >= 2.6.1

Note that "archive_detached" is not a valid entry and will be ignored.

The columns of Table 1 represent the behavior of the plugin. 'Archive' mode means the S3 resource acts in the archive role behind or under a compound resource. It requires a cache resource to provide POSIX semantics and it must be attached to a specific iRODS server configured with the S3 credentials. 'Cacheless' mode means the S3 resource can be standalone and may be detached from any specific iRODS server (as per 'attached' or 'detached'). In this role, the S3 plugin provides POSIX semantics with no cache resource and requires no explicit cache management policy.

The rows represent the configuration determining which iRODS server will answer a particular S3 request. 'Attached' means that only the server that is defined as the host in the resource configuration will serve the request. 'Detached' means all iRODS servers may serve a request for a data object. This is appropriate if all servers have connectivity to the S3 backend and are designed to provide a horizontal scale-out solution.

ARCHITECTURE

To implement the cacheless S3 resource, we implemented the missing operations that the legacy 'archive-only' S3 resource had not implemented.

These include:

```
irods::RESOURCE_OP_OPEN
irods::RESOURCE_OP_READ
irods::RESOURCE_OP_WRITE
irods::RESOURCE_OP_CLOSE
irods::RESOURCE_OP_LSEEK
```

S3FS

The prototype implementation started with S3FS[4], an open source C++ FUSE-based filesystem presentation of S3.

The first step was to validate that S3FS was stable. An S3FS mount point was created and then an iRODS resource was configured with its vault assigned a directory within the mount point. A suite of iRODS tests was run and all the tests passed, except for a few bundle test cases. This was convincing enough that S3FS was a good starting point for the cacheless plugin.

The next step was to translate the FUSE operations to iRODS resource plugin operations. The iRODS resource plugin operations follow POSIX semantics instead of FUSE semantics. In order to implement this, additional state information about every open file must be stored and maintained.

The cacheless S3 plugin keeps a map of open file descriptors and file offsets to objects in S3. Any time the `RESOURCE_OP_OPEN` operation is called, a unique file descriptor is generated and a new entry in this map is created with an offset of `0`. This file descriptor is the same file descriptor that is known to the iRODS core via the file first class object.

As reads, writes, and seeks are performed, the offset in the map is updated accordingly to match the POSIX behavior. If a file is opened multiple times (for example in a parallel read or write) each opened file is given a unique file descriptor and offset. When the `RESOURCE_OP_CLOSE` operation is called the appropriate entry is deleted from the map.

Mitigating Global Variables

S3FS uses many global variables for S3 configuration, internal data structures, flow control, etc.

These can conflict when using parallel uploads and downloads or when writing to more than one resource within the same agent (e.g. when two S3 resources are children of a replication resource in a resource hierarchy).

Care was taken to mitigate this by 1) using ”thread_local” variables so that each thread manages its own values and 2) storing configuration information in the `irods::plugin_property_map` making the iRODS catalog the source of persistent truth.

Two Protocols

The cacheless S3 plugin must provide a live translation of the conversation between two different protocols, single buffer or parallel iRODS on one side and multipart S3 on the other. For downloads from S3, this is straightforward since multipart S3 data is translated and sent to the iRODS client. For uploads, the plugin handles this translation by writing chunks of data to a temporary on-disk scratch space and then reading them back out before cleaning the scratch space. Overall, the plugin provides a management-free, cacheless experience to the iRODS administrator.

PERFORMANCE

Included in this section are some basic assessments of the plugin's performance and the steps taken to improve that performance. Both download and upload performance are compared against the AWS S3 CLI.

Download

Problem

When iRODS performs large file / parallel downloads, the plugin receives requests for bytes in a seemingly random order. If these individual random download requests are performed separately and in an uncoordinated manner, the S3 multipart download performance is not optimized.

Additionally, S3FS core code does read-ahead and will retrieve more than is requested which helps download performance but was still significantly slower than using the AWS CLI.

Goals

The problems above stood in the way of two goals for download performance. First, for this cacheless S3 plugin to be a viable alternative for users, download performance must be reasonably close to the performance of using the AWS CLI. Second, this plugin must be able to quickly service small requests within large files. If a file is 100GB in size, but a request is only for 1K of its data, this plugin should definitely not download the entire file.

Solution

The following approach solved both problems.

First, only read the requested bytes when requested, do not read ahead. Designate that if a second subsequent simultaneous read is requested this means the client is interested in performing a parallel download. If this occurs, initiate a full multipart S3 download of the requested file. Then, all open threads wait for a notification that a multipart chunk has been downloaded. On each multipart completion, the threads check if their requested bytes have been downloaded. If so, return these bytes. If not, wait for the next notification.

Results

Downloads times with the cacheless S3 plugin are very close to download times using the AWS CLI using the same `S3_MPU_CHUNK` size and `S3_MPU_THREAD` count. If a user only requests a small part of a large file, this is returned quickly. No full file downloads are performed unless requested.

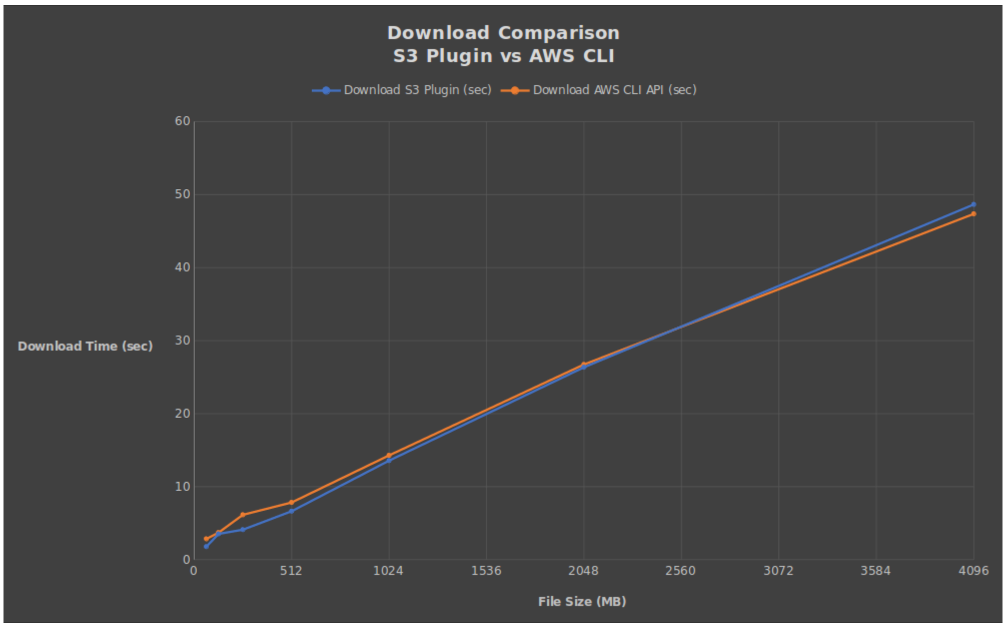

Figure 1. Download performance is nearly identical to AWS CLI.

	64MB	128MB	256MB	512MB	1024MB	2048MB	4096MB
Cacheless S3 Plugin	1.77	3.52	4.10	6.63	13.56	26.35	48.65
AWS S3 CLI	2.84	3.70	6.14	7.83	14.28	26.73	47.36

Table 2. Download times in seconds (average over 5x runs)

Upload

Problem

When uploading files through iRODS to S3, the full file must be uploaded. For large files, iRODS sends multiple data buffers in parallel. A naive approach would be to write these buffers directly to S3 but this requires rewriting parts of the file multiple times as S3 is an object store and can only write the entire object at a time.

Solution

While the S3 object is opened for writes, the data buffers are written to a local scratch file on the iRODS server. When the last `close()` is performed, flush the scratch file to S3 and remove it.

Results

File uploads are roughly 50% slower using the S3 plugin than using the AWS CLI but significantly better than the naive approach. The delay of writing and then reading the entire scratch file explains the increased linear difference in upload speed.

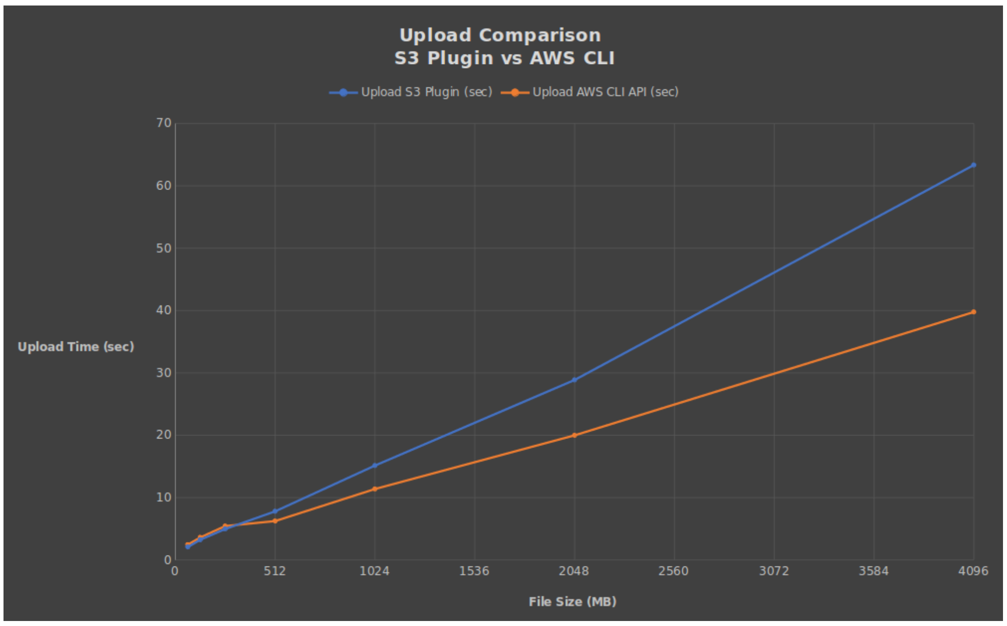

Figure 2. Upload performance loses ground with larger files... will investigate

	64MB	128MB	256MB	512MB	1024MB	2048MB	4096MB
Cacheless S3 Plugin	2.08	3.24	4.97	7.80	15.14	28.87	63.34
AWS S3 CLI	2.46	3.62	5.44	6.24	11.38	19.99	39.77

Table 3. Upload times in seconds (average over 5x runs)

USAGE

Using the cacheless S3 plugin is seamless. There is nothing to configure other than the mode itself in the context string.

The additional key/value pair in the context string could be `HOST_MODE=cacheless_attached`

Configuration

The following single `iadmin mkresc` command creates a new s3 resource of the name `news3resc` in `cacheless_attached` mode. The `bucket_name` specifies the S3 bucket. The S3 `prefix/to/iRODS/Vault` is optional. The backslashes at the end of each line are presentational only to ensure readability in this PDF.

```
iadmin mkresc news3resc s3 \
$(hostname):/bucket_name/prefix/to/iRODS/Vault \ "S3_DEFAULT_HOSTNAME=s3.amazonaws.com;\
S3_AUTH_FILE=/var/lib/irods/news3resc.keypair;\
S3_REGIONNAME=us-east-1;S3_RETRY_COUNT=1;\
S3_WAIT_TIME_SEC=3;S3_PROTO=HTTP;\
ARCHIVE_NAMING_POLICY=consistent;\
HOST_MODE=cacheless_attached"
```

Demonstration

With the cacheless S3 plugin properly configured, confirming correct behavior is straightforward. There is no second replica and the file sent is the same as the file returned:

```
$ ipwd
/tempZone/home/rods

$ iput -R news3resc foo

$ ils -L foo
  rods              0 news3resc          234 2019-06-25.14:40 & foo
        generic    /bucket_name/prefix/to/iRODS/Vault/home/rods/foo

$ iget foo bar

$ diff foo bar

$ echo $?
0
```

FUTURE WORK

The cacheless S3 plugin has passed all CI tests and has been released to the community. There are a few open issues, but no known configuration bugs at this point.

There are plans to focus on and improve the upload performance to try and match the AWS CLI performance as well as additionally implement the `RESOURCE_OP_READDIR` operation to facilitate recursive registrations.

The `S3_DEFAULT_HOST` will be improved to handle a comma separated list.

One of the most interesting requested enhancements is for the S3 plugin to retrieve its S3 credentials from the catalog or some other service like Hashicorp's Vault[5].

The iRODS Consortium has plans to take what has been learned with the S3 plugin and implement cacheless versions of the other iRODS archive resource plugins (WOS, etc.).

SUMMARY

The work to get the cacheless S3 plugin ready for release was more than expected. It presented some tricky scenarios and global variables which needed to be refactored. But it works and the performance is comparable to the AWS S3 CLI for downloads. Uploads still need some work.

The iRODS Consortium plans to continue this line of work with other plugins and remove configuration and maintenance complexity for iRODS administrators.

REFERENCES

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

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

[3] E-iRODS 3.0 (2013) `https://github.com/irods/irods/tree/f67864a8d89251fc9288f6dc43f6c21191f81af1`

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

[5] Vault by Hashicorp `https://www.vaultproject.io`