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

Illyoung Choi
CyVerse
iychoi@email.arizona.edu

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

Edwin Skidmore
CyVerse
edwin@cyverse.org

ABSTRACT

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

Keywords

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

INTRODUCTION

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

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

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

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

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

GO-IRODSCLIENT

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

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

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

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

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

IRODS FUSE LITE

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

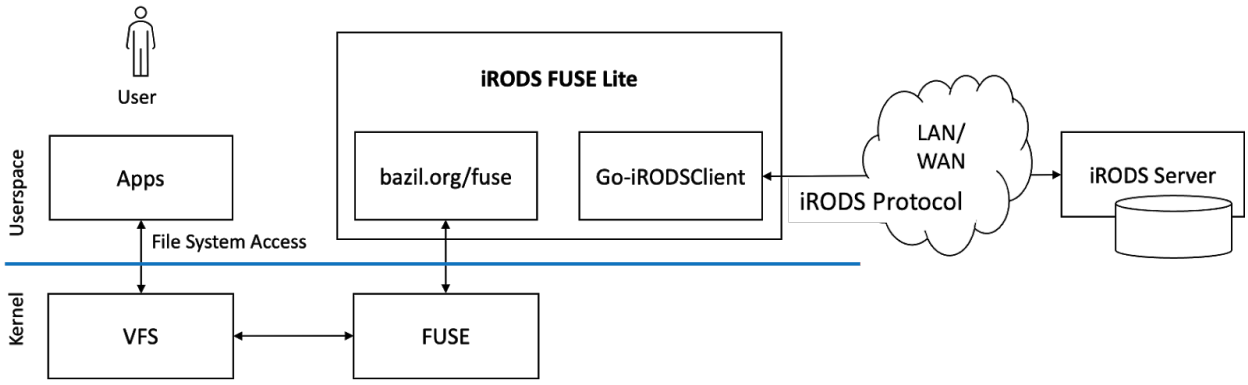

Figure 1. iRODS FUSE Lite implements the FUSE interface to enable iRODS access via the file system interface.

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

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

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

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

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

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

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

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

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

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

IRODS CSI DRIVER

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

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

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

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

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

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

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

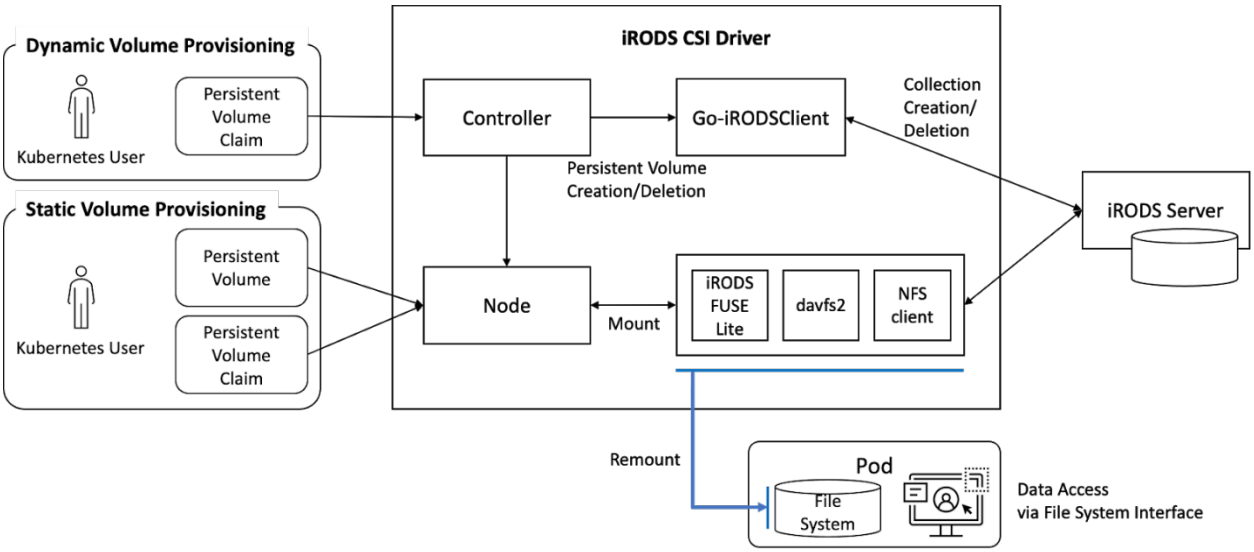

Figure 2. iRODS CSI driver provides Kubernetes users with iRODS data access by processing volume provisioning requests.

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

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

INTEGRATION EXAMPLE

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

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

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

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

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

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

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

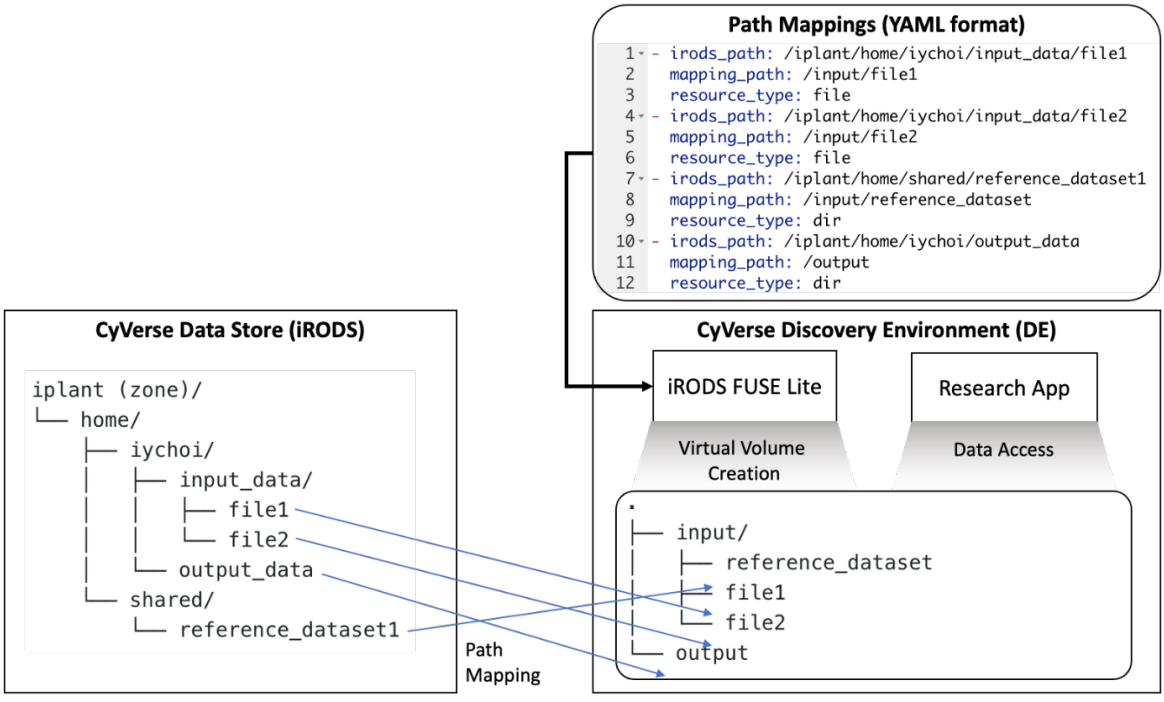

Figure 3. iRODS FUSE Lite and the iRODS CSI driver allow users to manipulate path mappings between iRODS data and local files. The mappings are defined using YAML and passed to iRODS FUSE Lite via the standard input (stdin).

PERFORMANCE EVALUATION

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

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

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

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

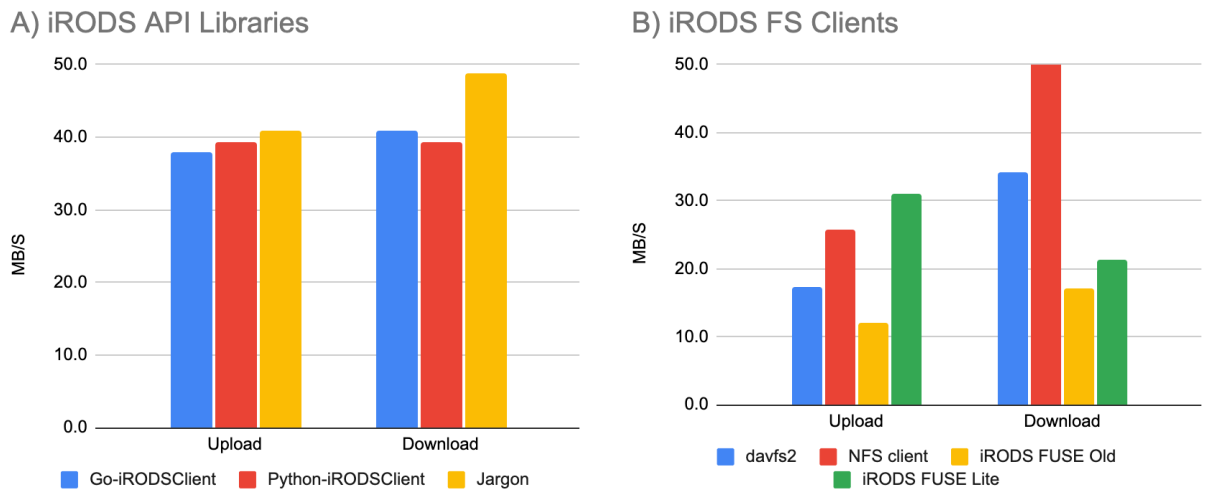

Figure 4. Data transfer performance of iRODS client libraries (A) and file system clients (B). We implemented simple data transfer benchmarks using the libraries for the evaluation. We used "cp" to test data transfer performance of the file system clients.

Go-iRODSClient

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

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

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

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

iRODS FUSE Lite

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

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

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

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

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

CONCLUSION

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

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

ACKNOWLEDGMENTS

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

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

CODE AVAILABILITY

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

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

REFERENCES

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