Distributing the iRODS Catalog: A Way Forward

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

Michael Stealey
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
stealey@renci.org

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

Ben Keller
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
kellerb@renci.org

Claris Castillo
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
claris@renci.org

Ray Idaszak
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
rayi@renci.org

Alex Feltus
Clemson University
Clemson, SC
ffeltus@clemson.edu

ABSTRACT

In the last few years, researchers in academia and in both governmental and corporate sectors have become more interested in spanning greater physical distances within a single logical namespace for their files (iRODS Zone). However, connecting to a distant iRODS Catalog Provider presents a significant, if not unbearable, hurdle. This paper explores a solution to this new use case with a clustered database technology providing the iRODS metadata catalog (iCAT).

Keywords

iRODS, database, SQL, MariaDB, Galera, cluster, metadata

INTRODUCTION

An iRODS[1][2][3] use case was presented in which geographically disparate participants wanted to belong to the same iRODS Zone for ease of search and discovery, but they also wanted all iRODS servers to be provider nodes serving their own catalog. There was a desire to be able to decentralize the traditionally singular iCAT catalog database in a way that all participants could make use of whichever iCAT provider was closest to them rather than having to federate with separate iRODS Zones in distant physical locations. A multi-master solution would provide local authentication and improved metadata read performance while continuing to provide locality of reference for data at rest.

The initial requirements for such a system:

- Every iRODS provider node would contain the iCAT catalog and local storage space that could be uniquely assigned to that node as a resource.
- Files would be transferred to the iRODS provider node deemed closest to the point of file origination with respect to network latency and disk I/O metrics.
- All nodes must pass some sort of quality assurance test beyond the standard iRODS test suite which only exercises a single node via the default `demoResc` resource.

iRODS UGM 2017 June 13-15, 2017, Utrecht, Netherlands

PROOF OF CONCEPT

The proof of concept solution being presented here uses MariaDB[4] configured as a Galera cluster to decentralize the iCAT catalog database across all participating iRODS provider nodes.

WAN replication will use ample latency values commensurate for an international WAN deployment.

A proof of concept testbed comprised of three iRODS provider nodes was deployed to form a single zone named `tempZone` within a MariaDB Galera cluster. Each node within the testbed is a single CentOS 7 VM, and can be configured to use differing latency values via NetEm[5] to simulate the kind of network traffic that would be experienced in a WAN configuration. Each VM is configured with a user account named **galera** which has rights to run Docker.

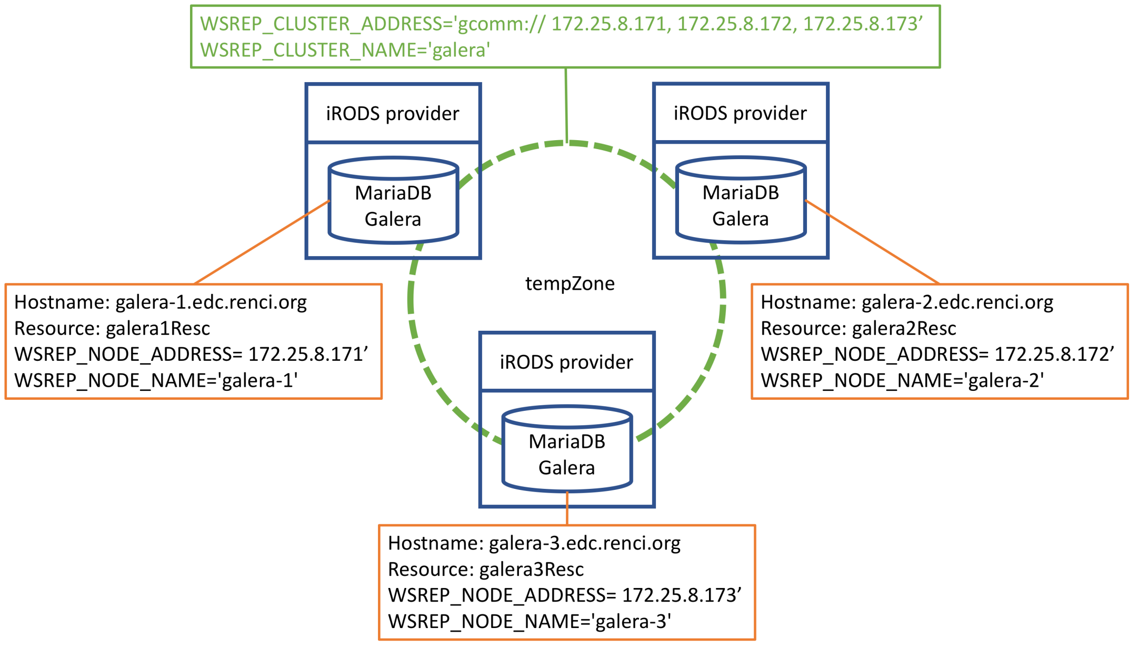

WSREP_CLUSTER_ADDRESS='gcomm:// 172.25.8.171, 172.25.8.172, 172.25.8.173'
WSREP_CLUSTER_NAME='galera'
iRODS provider
MariaDB Galera
iRODS provider
MariaDB Galera
tempZone
Hostname: galera-1.edc.renci.org
Resource: galera1Resc
WSREP_NODE_ADDRESS= 172.25.8.171'
WSREP_NODE_NAME='galera-1'
Hostname: galera-2.edc.renci.org
Resource: galera2Resc
WSREP_NODE_ADDRESS= 172.25.8.172'
WSREP_NODE_NAME='galera-2'
iRODS provider
MariaDB Galera
Hostname: galera-3.edc.renci.org
Resource: galera3Resc
WSREP_NODE_ADDRESS= 172.25.8.173'
WSREP_NODE_NAME='galera-3'

Figure 1. Baseline Docker deployment of three iRODS Catalog Providers via MariaDB Galera Cluster.

The iRODS provider node[6][7] is using MariaDB Galera cluster in Docker[8] and is based on a centos:7 image[9].

Design

The proof of concept has been designed to run in Docker and does not prohibit storing the files for iRODS and MariaDB to the host. Generally an iRODS system deployment would use service level accounts named **irods** and **mysql** to retain/own the iRODS service files and the MariaDB files respectively. The Docker image used herein has defaulted these system users to be:

```
irods: UID=996, GID=996
mysql: UID=997, GID=997
```

If the user chooses to deploy the **irods-provider-galera** image using local volume mounts, then these **UID** and **GID** values would be found on the local system for the **user:group** in charge of the shared volumes. The **UID** and **GID** for both the **irods** and **mysql** user can be set at runtime to be any valid combination within the host system the container is being run from.

Setup

A set of directories in `/var` are set aside to mount to the **irods-provider-galera** container to hold the iRODS and MariaDB files. The following `setup-galera` script outlines the creation and initial permissions for these directories:

```
#!/usr/bin/env bash
sudo rm -rf /var/galera
sudo mkdir -p /var/galera/init
sudo mkdir -p /var/galera/vault
sudo mkdir -p /var/galera/var_mysql
sudo mkdir -p /var/galera/var_irods
sudo mkdir -p /var/galera/etc_irods
sudo chown -R galera:galera /var/galera
exit 0
```

The user named **galera** will be in charge of running **irods-provider-galera**, and has the following attributes:

```
$ id galera
uid=2112(galera) gid=2112(galera) groups=2112(galera),992(docker)
```

When instantiating the iRODS container we set the environment variables of the **irods** and **mysql** users to use UIDs and GIDs that are locally available:

```
-e UID_MYSQL=2000 \ # UID that is unassigned on the localhost
-e GID_MYSQL=2112 \ # GID assigned to user galera on the localhost
-e UID_IRODS=2112 \ # UID assigned to user galera on the localhost
-e GID_IRODS=2112 \ # GID assigned to user galera on the localhost
```

View of `/var/galera` before **irods-provider-galera** is run:

```
$ ls -alh /var/galera/
total 4.0K
drwxr-xr-x   7 galera galera   77 Jun 11 11:34 .
drwxr-xr-x. 21 root   root   4.0K Jun 11 11:34 ..
drwxr-xr-x   2 galera galera    6 Jun 11 11:34 etc_irods
drwxr-xr-x   2 galera galera    6 Jun 11 11:34 init
drwxr-xr-x   2 galera galera    6 Jun 11 11:34 var_irods
drwxr-xr-x   2 galera galera    6 Jun 11 11:34 var_mysql
drwxr-xr-x   2 galera galera    6 Jun 11 11:34 vault
```

The next three sections will step through testing the clustered iCAT, first with Docker deploying all three nodes on a single machine, then Docker deploying an iRODS catalog provider on three VMs, and finally on three VMs with an additional latency introduced to simulate longer distances.

LAN - LOCAL MACHINE

The first test runs on a single VM and was created to demonstrate the basic principles of using the MariaDB Galera cluster as the iRODS catalog for multiple provider nodes. The test script does the following:

- Creates a local Docker network named `galeranet` so that known IP addresses can be assigned to each node.
- Stands up the initial bootstrap node using mostly defaults as set by the Docker image.
- Stands up two additional nodes in series that join the cluster named `galera` as they discover others on the local `galeranet` network.

As each node completes its stand up routine, it reports the number of nodes participating as the `wsrep_cluster_size`, lists the databases and the grants for user `'irods'@'localhost'`, and finally prints out all tables within the iCAT database.

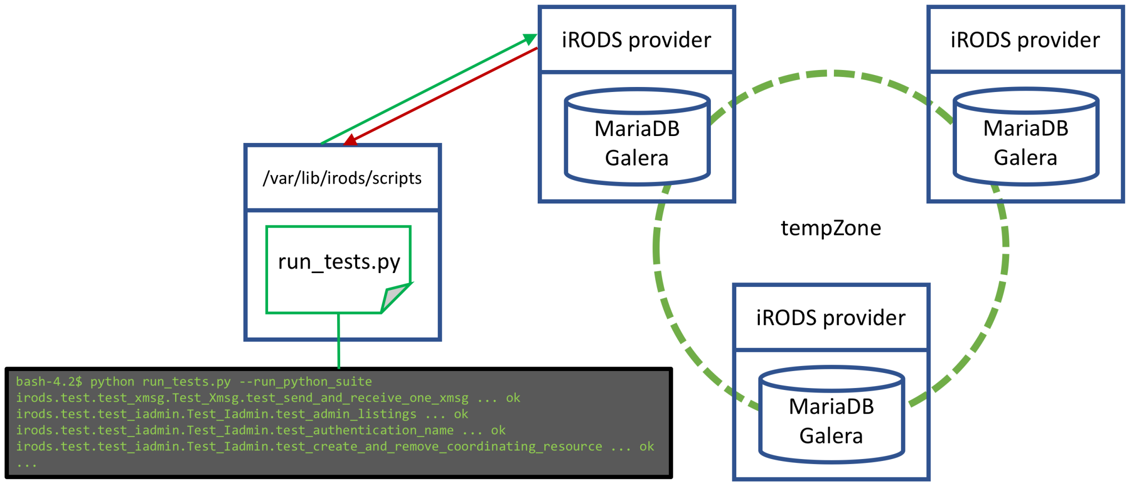
Since this initial test was performed on a single VM using a Docker network, it was not subjected to any external testing and only subject to simple iCommands to validate synchronization between nodes and partitioning between named node resource definitions.

The expected output was observed and showed that the three nodes were up and connected within the Docker network:

```
$ docker ps
CONTAINER ID   IMAGE                                  STATUS          NAMES
b51fcb1501ec   mjstealey/irods-provider-galera:4.2.1  Up 44 minutes   irods-galera-node-3
654b094fc554   mjstealey/irods-provider-galera:4.2.1  Up 45 minutes   irods-galera-node-2
cc51c3413306   mjstealey/irods-provider-galera:4.2.1  Up 45 minutes   irods-galera-node-1
```

LAN - VIRTUAL MACHINES

The second test runs on three VMs with Docker running a clustered iRODS catalog provider on each and uses the built-in iRODS test suite located at `/var/lib/irods/scripts/run_tests.py`. The limitation of the test suite is that it was not necessarily designed to run against a clustered database, so the default notion of `demoResc` is problematic when the goal is to test across a distributed iCAT cluster. Because of this limitation, we issued `python run_tests.py --run_python_suite` on each VM/node within the cluster, one at a time.



```
bash-4.2$ python run_tests.py --run_python_suite
irods.test.test_xmsg.Test_Xmsg.test_send_and_receive_one_xmsg ... ok
irods.test.test_iadmin.Test_Iadmin.test_admin_listings ... ok
irods.test.test_iadmin.Test_Iadmin.test_authentication_name ... ok
irods.test.test_iadmin.Test_Iadmin.test_create_and_remove_coordinating_resource ... ok
...
```

Figure 2. Simple test run on one of the three nodes in a clustered catalog provider.

While the test suite runs on one node, the other nodes were manually monitored via various iCommands and SQL queries validating that the test files and corresponding database entries were being created as expected and visible from the other nodes.

The `run_tests.py` script is the default script used for iRODS testing and is part of a normal iRODS installation. As an initial check, it was chosen to run with the `--run_python_suite` option which takes roughly four hours to complete all its nearly 1500 tests.

Since iRODS is being run in Docker, it is necessary to invoke the test suite from within the Docker container. This can be accomplished by connecting to the container as the **irods** user, changing to the `scripts` directory, and issuing the test run call.

```
[galera@galera-1 ~]$ docker exec -ti -u irods irods-galera-1 /bin/bash
bash-4.2$ cd ~
bash-4.2$ pwd
/var/lib/irods
bash-4.2$ cd scripts/
bash-4.2$ python run_tests.py --run_python_suite
irods.test.test_xmsg.Test_Xmsg.test_send_and_receive_one_xmsg ... ok
irods.test.test_iadmin.Test_Iadmin.test_addchildtoresc_forbidden_characters_3449 ... ok
irods.test.test_iadmin.Test_Iadmin.test_admin_listings ... ok
irods.test.test_iadmin.Test_Iadmin.test_authentication_name ... ok
irods.test.test_iadmin.Test_Iadmin.test_create_and_remove_coordinating_resource ... ok
...
```

The running tests can be observed from the other nodes via iCommands or mysql queries (in this case, `ils ../` from `irods-galera-2`:

```
[galera@galera-2 ~]$ docker exec -ti -u irods irods-galera-2 ils ../
/tempZone/home:
  C- /tempZone/home/alice
  C- /tempZone/home/bobby
  C- /tempZone/home/issue_3104_user
  C- /tempZone/home/otherrods
  C- /tempZone/home/public
  C- /tempZone/home/rods
```

Over nearly twelve hours, the three test runs completed, in turn, from each node in the cluster and the outputs were consistent with running the test suite in a single server configuration.

```
$ python run_tests.py --run_python_suite
irods.test.test_xmsg.Test_Xmsg.test_send_and_receive_one_xmsg ... ok
irods.test.test_iadmin.Test_Iadmin.test_addchildtoresc_forbidden_characters_3449 ... ok
irods.test.test_iadmin.Test_Iadmin.test_admin_listings ... ok
irods.test.test_iadmin.Test_Iadmin.test_authentication_name ... ok
irods.test.test_iadmin.Test_Iadmin.test_create_and_remove_coordinating_resource ... ok
...
irods.test.test_irmdir.Test_Irmdir.test_irmdir_of_collection_containing_dataobj ... ok
irods.test.test_irmdir.Test_Irmdir.test_irmdir_of_dataobj ... ok
irods.test.test_irmdir.Test_Irmdir.test_irmdir_of_empty_collection ... ok
irods.test.test_irmdir.Test_Irmdir.test_irmdir_of_nonexistent_collection ... ok
irods.test.test_iquest.Test_Iquest.test_iquest_MAX_SQL_ROWS_results__3262 ... ok

----------------------------------------------------------------------
Ran 1468 tests in 13106.840s

OK (skipped=89)
<__main__.RegisteredTestResult run=1468 errors=0 failures=0>
```

WAN - VIRTUAL MACHINES

The third test was designed to run a parallel test script that puts and gets large files into iRODS in a continuous loop until broken by the user. This script uses GNU parallel[10].

The parallel put/get test was first run on the testbed without any additional latency introduced to the nodes and allowed to run for six hours without error.

Then, varying amounts of latency were added to each node using NetEm to simulate the distances involved with real world WAN deployments of iRODS. The parallel test script was again allowed to run against the nodes in this configuration for multiple hours.

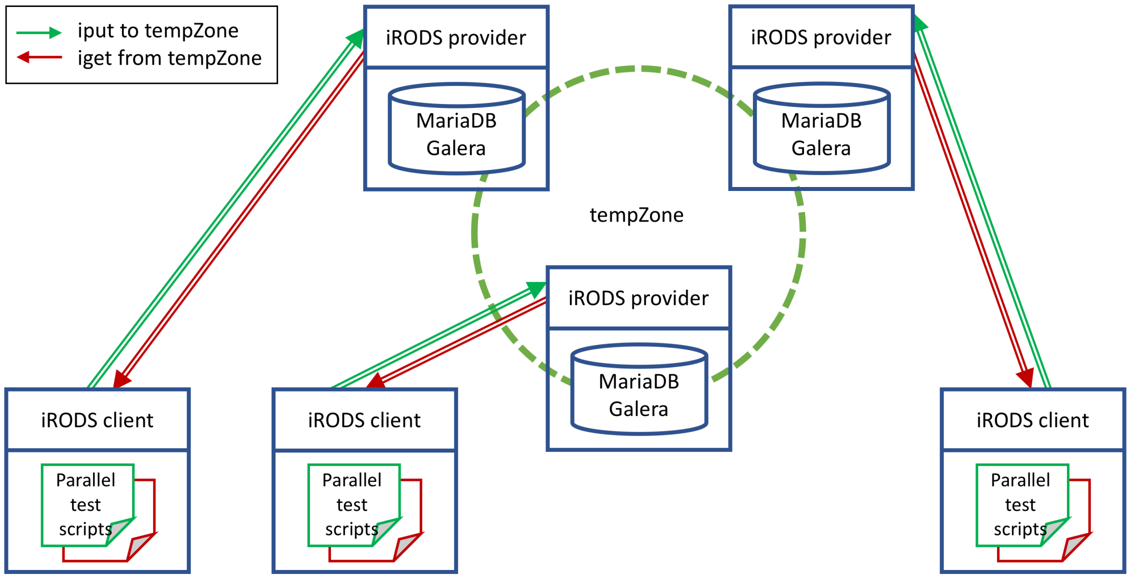


Figure 3. Concurrent parallel test scripts running on a three-node clustered catalog provider.

The following real-world latency approximations were used to add delay to each node, respectively:

Low National (Chicago/RENCI):	20ms
Medium National (Coast to Coast - SanFrancisco/RENCI):	60ms
International (RENCI/Netherlands):	117ms

The approximations were implemented via NetEm as follows:

```
galera-1: sudo tc qdisc add dev eth0 root netem delay 20.0ms
galera-2: sudo tc qdisc add dev eth0 root netem delay 60.0ms
galera-3: sudo tc qdisc add dev eth0 root netem delay 120.0ms
```

Since NetEm affects both incoming and outgoing traffic, the effect was cumulative across nodes which can be observed in the RTT between nodes.

	galera-1	**galera-2**	**galera-3**
galera-1	n/a	80.5ms	140.5ms
galera-2	80.5ms	n/a	180.4ms
galera-3	140.5ms	180.5ms	n/a

Table 1. Round Trip Time (RTT) within the three-node testbed

After the initial parallel test on a single node, the `parallel_get_put.sh` script was launched simultaneously on all three nodes. The scripts first generate 256 40 MB files and then start sending them via `iput` in 30 parallel threads to the local iRODS server instance onto its local storage resource. Once all 256 files have been transferred, they are retrieved using `iget` in 30 parallel threads. This process is embedded in a loop and continues until an error is encountered or is manually stopped.

The three nodes ran without error for over two hours until manually interrupted and forcibly quit. During this period each node completed a number of put/get loops based on the distance from its peers:

galera-1	48 loops
galera-2	12 loops
galera-3	3 loops

There was a single multi-master deadlock error during this testing where the database complained of a timeout during the concurrent writes. This was due to the relatively large number of operations that occur within the database transaction during a single iRODS connection. We have since heard anecdotal evidence of this same case from others; concurrent load tripping a cluster timeout when the latency is high between cluster nodes. The clustered nodes have a default timeout setting that interprets a delay as their peer not responding. We expect that adjusting the default timeout can reduce the rate of incidence.

In addition to adjusting the timeout, there are two main ways that iRODS can improve to avoid deadlock errors of this type. The first would be to add a simple retry upon network timeout. Such a retry would push back the threshold where the concurrent writes trigger a timeout, but ultimately, would be a temporary fix and would not solve the problem for a growing system. A more robust path would be to reduce the number of operations that occur within the database transactions themselves. If the database is doing less work within each transaction, it is less likely that the system will hit a timeout and interpret it as a deadlock.

FUTURE WORK

This exercise has demonstrated that a distributed cluster for the iCAT is feasible, but that there is much more work to do. The most straightforward efforts will be to spend more time testing on real networks over real distances, rather than simulated. A second focus should be to test other database technologies besides MariaDB's Galera. We are interested in working with CockroachDB as the next viable candidate. The third type of future work should be testing targeted edge cases that will push on the overlap and potential friction between the database itself and how iRODS aims to present a consistent surface to its clients. And lastly, we need to investigate the upgrade or migration path for how an existing iRODS Zone moves from a singular iCAT to a clustered iCAT.

CONCLUSION

iRODS works when using a clustered SQL database and can satisfactorily address the use case defined by global organizations that want to leverage a unified namespace. Like with all new use cases, some defaults settings should be adjusted to accommodate a different set of assumptions. The iRODS architecture proved flexible enough to easily adapt to this new use case of a high latency, WAN, multi-master deployment, however we need more testing and real world use cases to drive the next set of improvements.

ACKNOWLEDGEMENTS

The authors would like to thank the ongoing support of the Renaissance Computing Institute (RENCI) at UNC-Chapel Hill for the networking and virtual machine infrastructure to conduct these tests. In addition, the expertise and use case support from the SciDAS[11] project and BioTeam[12] proved invaluable.

REFERENCES

[1] iRODS website. `https://irods.org`

[2] iRODS. `https://github.com/irods/irods`

[3] iRODS Primer 2: Integrated Rule-Oriented Data System (2017). Hao Xu, Terrell Russell, Jason Coposky, Arcot Rajasekar, Reagan Moore, Antoine de Torcy, Michael Wan, Wayne Schroeder, and Sheau-Yen Chen. Synthesis Lectures on Information Concepts, Retrieval, and Services, March 2017, Vol. 9(3):1-131. `https://doi.org/10.2200/S00760ED1V01Y201702ICR057`

[4] MariaDB website. `https://mariadb.org/`

[5] Stephen Hemminger (2005). NetEm - Emulating Real Networks in the Lab. . Linux Conf, Canberra, Australia. April 2005. `https://linux.org.au/conf/2005/Papers/Stephen%20Hemminger/index.html`

[6] Michael Stealey (2017). iRODS Provider, Galera. `https://github.com/mjstealey/irods-provider-galera`

[7] Michael Stealey (2017). Documentation for iRODS Provider, Galera. `https://mjstealey.github.io/irods-provider-galera/`

[8] Michael Stealey (2017). MariaDB Galera Docker Container. `https://github.com/mjstealey/mariadb-galera`

[9] Docker Hub - CentOS. `https://hub.docker.com/_/centos/`

[10] O. Tange (2011). GNU Parallel - The Command-Line Power Tool. login: The USENIX Magazine, February 2011:42-47.

[11] Frank Feltus, Claris Castillo, Ray Idaszak, Melissa Smith, Stephen Ficklin (2017). National Cyberinfrastructure for Scientific Data Analysis at Scale (SciDAS). NSF Award 1659300. `https://www.nsf.gov/awardsearch/showAward?AWD_ID=1659300`

[12] BioTeam website. `https://bioteam.net/`