CYVERSE™

Transforming Science Through Data-driven Discovery

More than just Load Balancing iRODS Using HAProxy

Tony Edgin

iRODS UGM 2019

The University of Arizona

TACC Texas Advanced Computing Center

CSH Cold Spring Harbor Laboratory

UNCW

NSF

Purpose

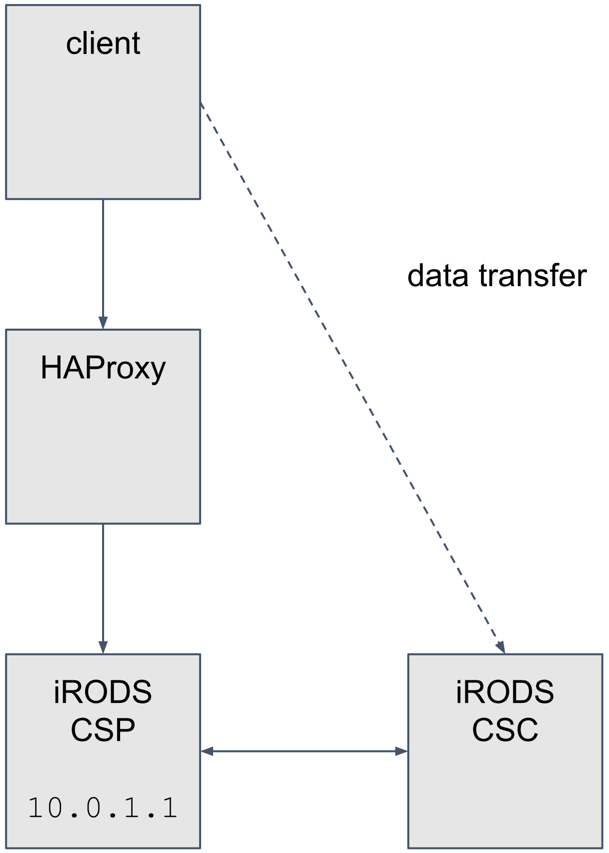
Previous work shows how to proxy iRODS for high availability.

- High Availability iRODS System (HAIRS) by Yutaka Kawai and Adil Hasan
- Configuring iRODS for High Availability by Justin James (uses HAProxy)

Show how to proxy iRODS for other use cases through concrete examples

Basic Setup in Examples

- Like prior work
  - Clients connect to HAProxy
  - HAProxy connects to Catalog Service Provider or CSP (IES for us older timers)
- Unlike prior work
  - Only one CSP
  - Catalog Service Consumer or CSC (resource server for us older timers) connects directly to CSP
- Software Versions
  - iRODS 4.1.10
  - HAProxy 1.8.4



About the Examples

- All examples are HAProxy configuration file fragments, with the first being complete.
- For HAProxy configuration file syntax, see https://cbonte.github.io/haproxy-dconv/1.8/configuration.html.
- Jinja templating syntax used
  - Jinja is a Python template engine (See http://jinja.pocoo.org/)
  - Used to encapsulate complexity
  - Not used to configure HAProxy!

Proxying Examples

1. Handling reconnections
2. Logging applications and who’s using them
3. Rejecting connections from port scanners
4. Rejecting connections based on client user concurrency limit
5. Throttling usage based on IP address concurrency limit
6. Protecting latency QOS for select IP addresses

1. Handling reconnections

For instance, `iput -T` requests reconnect from server

Some iCommands ignore reconnection host and reconnect to proxy, see https://github.com/irods/irods/issues/4339

This can be a feature! Allows backend to count reconnects for maxconn.

```
prompt> cat /etc/haproxy/haproxy.cfg

# reconnection example (complete)

global
  chroot  /var/lib/haproxy
  user    haproxy
  daemon

defaults
  mode     tcp

frontend irods_main
  bind              :1247
  default_backend   irods

frontend irods_reconn
  bind              :20000-20199
  default_backend   irods

backend irods
  server  csp 10.0.1.1 maxconn 200
```

2. Logging Applications and Users, Part 1

```
# logging example, based on reconnection example

frontend irods_main
  bind              :1247
  default_backend   irods
  tcp-request       inspect-delay 5s
  tcp-request       content capture {{ msgTypeCapture }} len 16
  acl               is-conn capture.req.hdr(0) -m str RODS_CONNECT
  tcp-request       content capture {{ captureMsg('option') }} len 250 if is-conn
  tcp-request       content capture {{ captureMsg('clientUser') }} len 250 if is-conn
  tcp-request       content capture {{ captureMsg('clientRcatZone') }} len 250 if is-conn
  log-format        %ci\ app=%[capture.req.hdr(1)]\ client={{ clientFmt }}
```

First tcp-request capture stored in capture.req.hdr(0), second in capture.req.hdr(1), etc.

Jinja templating

Creates a boolean expression named 'is-conn'. This one is true if the first captured value is 'RODS_CONNECT'

Interlude - iRODS Connection Initiation, Part 1

See https://wiki.mcs.anl.gov/CDIGS/images/b/bd/IrodsProtPaper.doc for details

Client initiates connection to server by sending

[####][Header][Startup Packet]

[####]	- Four byte integer (binary) holding length of [Header]
[Header]	- MsgHeader_PI XML element
[Startup Packet]	- StartupPack_PI XML element

Interlude - iRODS Connection Initiation, Part 2

Partial DTD for MsgHeader_PI

```
<!ELEMENT MsgHeader_PI (type, ...)>
<!ELEMENT type (“RODS_CONNECT”| ...)>
...
```

Partial DTD for StartupPack_PI

```
<!ELEMENT StartupPack_PI
  (reconnFlag, proxyUser, proxyRcatZone, clientUser, clientUserRcatZone, option, ...)
 >
<!ELEMENT reconnFlag (“0”|”200”)>   <!-- 200 to enable reconnection -->
<!ELEMENT proxyUser (#CDATA)>       <!-- proxy user’s name -->
<!ELEMENT proxyRcatZone (#CDATA)>   <!-- proxy user’s zone -->
<!ELEMENT clientUser (#CDATA)>      <!-- client user’s name -->
<!ELEMENT clientRcatZone (#CDATA)>  <!-- client user’s zone -->
<!ELEMENT option (#CDATA)>          <!-- often holds name of client app -->
...
```

2. Logging Applications and Users, Part 2

```
frontend irods_main
  bind                :1247
  default_backend     irods
  tcp-request         inspect-delay 5s
  tcp-request         content capture {{ msgTypeCapture }} len 16
  acl                 is-conn capture.req.hdr(0) -m str RODS_CONNECT
  tcp-request         content capture {{ captureMsg('option') }} len 250 if is-conn
  tcp-request         content capture {{ captureMsg('clientUser') }} len 250 if is-conn
  tcp-request         content capture {{ captureMsg('clientRcatZone') }} len 250 if is-conn
  log-format          %ci\ app=%[capture.req.hdr(1)]\ client={{ clientFmt }}

{% set msgTypeCapture  = 'req.payload_lv(0,4),' ~ stripBeforeType ~ ',' ~ stripAfterType %}
{% set stripBeforeType = 'regsub(^\s*<MsgHeader_PI\s*>[\s\S]*<type\s*>,)' %}
{% set stripAfterType  = 'regsub(</type\s*>[\s\S]*</MsgHeader_PI\s*>\s*$,)' %}
```

Sample TCP packet payload using first 4 bytes to determine how much to read

remove the bytes before and after type field value

2. Logging Applications and Users, Part 3

```
frontend irods_main
  bind                 :1247
  default_backend      irods
  tcp-request          inspect-delay 5s
  tcp-request          content capture {{ msgTypeCapture }} len 16
  acl                  is-conn capture.req.hdr(0) -m str RODS CONNECT
  tcp-request          content capture {{ captureMsg('option') }} len 250 if is-conn
  tcp-request          content capture {{ captureMsg('clientUser') }} len 250 if is-conn
  tcp-request          content capture {{ captureMsg('clientRcatZone') }} len 250 if is-conn
  log-format           %ci\ app=%[capture.req.hdr(1)]\ client={{ clientFmt }}

{% macro captureMsg(field) -%}
  req.payload(4,0),{{ stripBefore(field) }},{{ stripAfter(field) }}  {%- endmacro %}
{% macro stripBefore(field) -%}
  regsub([\s\S]*<StartupPack_PI\s*>[\s\S]*<{{ field }}\s*>,)          {%- endmacro %}
{% macro stripAfter(field) -%} regsub(</{{ field }}\s*>[\s\S]*,)      {%- endmacro %}
```

Sample entire TCP packet payload skipping first 4 bytes

remove the bytes before and after the value of the given field

2. Logging Applications and Users, Part 4

```
frontend irods_main
  bind                :1247
  default_backend     irods
  tcp-request         inspect-delay 5s
  tcp-request         content capture {{ msgTypeCapture }} len 16
  acl                 is-conn capture.req.hdr(0) -m str RODS_CONNECT
  tcp-request         content capture {{ captureMsg('option') }} len 250 if is-conn
  tcp-request         content capture {{ captureMsg('clientUser') }} len 250 if is-conn
  tcp-request         content capture {{ captureMsg('clientRcatZone') }} len 250 if is-conn
  log-format          %ci\ app=%[capture.req.hdr(1)]\ client={{ clientFmt }}

{% set clientFmt = '%[capture.req.hdr(2)]\#%[capture.req.hdr(3)]' %}
```

3. Rejecting Port Scanner Connections

```
# port scanner example, based on logging example

frontend irods_main
  bind                  :1247
  default_backend  irods
  tcp-request      inspect-delay 5s
  tcp-request      content capture {{ msgTypeCapture }} len 16
  acl              is-conn capture.req.hdr(0) -m str RODS_CONNECT
  tcp-request      content reject unless is-conn
```

4. Rejecting Connections Based on Client User Concurrency Limit (HAProxy 1.9+)

```
# rejection example 1.9, based on port scanner example

frontend irods_main
  bind               :1247
  default_backend    irods
  tcp-request        content capture {{ msgTypeCapture }} len 16
  acl                is-conn capture.req.hdr(0) -m str RODS_CONNECT
  tcp-request        content reject unless is-conn
  stick-table        type string len 512 size 100k store conn_cur
  tcp-request        content capture {{ captureMsg('clientUser') }} len 250
  tcp-request        content capture {{ captureMsg('clientRcatZone') }} len 250
  tcp-request        content track-sc1 capture.req.hdr(0),concat(\#,capture.req.hdr(1),)
  acl                too-many-conn sc1_conn_cur gt 1
  tcp-request        content reject if too-many-conn
```

Allocate table to track number of open connections with certain ‘name#zone’

Associate counter sc1 with with name#zone of current request

Number of open connections with same name#zone as current request

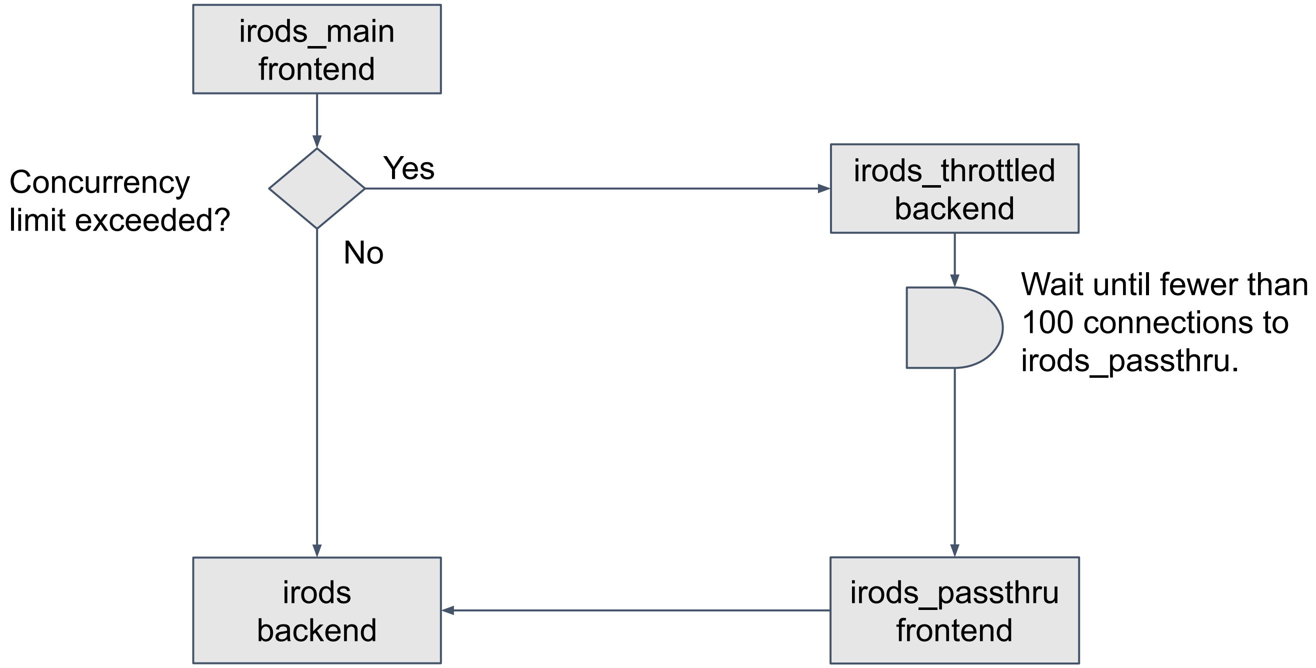
4. Rejecting Connections Based on Client User Concurrency Limit (HAProxy 1.8)

```
# rejection example 1.8, based on rejection example 1.9

frontend irods_main
  bind                :1247
  default_backend     irods
  tcp-request         content capture {{ msgTypeCapture }} len 16
  acl                 is-conn capture.req.hdr(0) -m str RODS_CONNECT
  tcp-request         content reject unless is-conn
  stick-table         type string len 512 size 100k store conn_cur
  # concat doesn't exist, assume clientUser always comes before clientRcatZone
  tcp-request         content track-sc1 {{ uzCapture }}
  acl                 too-many-conn sc1_conn_cur gt 1
  tcp-request         content reject if too-many-conn

{% set uzCapture = stripBefore('clientUser')
                  ~ ',regsub(</clientUser\s*>[\s\S]*<clientRcatZone\s*>,\#),'
                  ~ stripAfter('clientRcatZone') %}
```

5. Throttling Usage Based on IP Address Concurrency Limit, Part 1


irods_main frontend
Concurrency limit exceeded?
Yes
No
irods_throttled backend
Wait until fewer than 100 connections to irods_passthru.
irods backend
irods_passthru frontend

5. Throttling Usage Based on IP Address Concurrency Limit, Part 2

```
# throttling example, based on rejection example

global
  chroot     /var/lib/haproxy
  user       haproxy
  unix-bind  prefix /var/lib/haproxy/ mode 770 user haproxy

frontend irods_main
  bind                :1247
  default_backend     irods
  stick-table         type ip size 100k store conn_cur
  tcp-request         connection track-sc0 src
  acl                 too-many-conn sc0_conn_cur gt 1
  use_backend         irods_throttled if too-many-conn

backend irods_throttled
  server  throttled unix@haproxy_irods.sock send-proxy maxconn 100

frontend irods_passthru
  bind                unix@haproxy_irods.sock accept-proxy
  default_backend     irods
```

Limit to 50% of the 200 available connections

Routing extra connections through UNIX socket

6. Protecting Latency QOS for Select IP Addresses

```
# latency QOS example, based on address throttling example

frontend irods_main
  bind                :1247
  default_backend     irods_throttled
  acl                 vip-src src -f /etc/haproxy/fastpass.lst
  use_backend         irods if vip-src

backend irods_throttled
  server   throttled unix@haproxy_irods.sock send-proxy maxconn 160
```

Inverse of previous example, default backend is now throttled

Increase to 80%, reserving 40 for select IP addresses

```
prompt> cat /etc/haproxy/fastpass.lst
10.4.0.0
10.4.0.32/27
```

Questions?